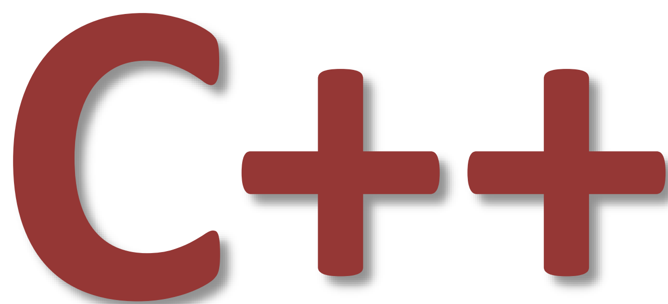


Supplementary
with
Threads Programming

2019 January

Ribomation datakonsult AB
www.ribomation.se

Kopieringsförbud
Detta kursmaterial är skyddat enligt lagen om upphovsrätt och får därmed ej helt eller delvis mångfaldigas utan skriftligt godkännande av Ribomation datakonsult AB.

Den som bryter mot lagen om upphovsrätt riskerar att åtalas av allmän åklagare och kan dömas till böter eller fängelse i upp till två år, samt bli skyldig att erlägga ersättning till upphovsman/rättighetsinnehavare.

© Ribomation datakonsult AB, 2009-2019.

Table of Contents

1 COURSE INTRODUCTION 1
1.1 Myths & Facts about C++ 4
1.2 Evolution of C++ 8

2 PART-1: MODERN C++ 11
2.1 Initialization 11
2.2 Automatic Type Inference 16
2.3 Misc. Syntax Improvements 22
2.4 Lambda Expressions 28
2.5 Move Semantics 33

3 PART-2: C++ REFRESHER 39
3.1 Constructors 39
3.2 Members 50
3.3 Inheritance 56
3.4 Const-ness 65
3.5 Namespaces 72
3.6 Usage of new & delete 76
3.7 Templates 83
3.8 Operator Overloading 95

4 PART-3: C++ LIBRARY 105
4.1 Text Strings 105
4.2 Container Types 113
4.3 Iterators & Intervals 127
4.4 Algorithms 135

5 PART-4: CLASSIC THREADING 142
5.1 Introduction to Threads 142
5.2 POSIX Threads C API & C++ Threads 151
5.3 Critical Sections & Mutexes 158
5.4 Race Conditions & Condition Variables 169
5.5 Deadlocks 182
5.6 Message Passing 186
5.7 Rendezvous 195

6 PART-5: MORE ABOUT MODERN C++ 202
6.1 Helper Types 202
6.2 String Views 210
6.3 Smart Pointers 213
6.4 File Systems 221
6.5 Date & Clocks 225

7 PART-6: MODERN THREADING 231
7.1 C++11 Threads 231
7.2 C++11 Synchronization 238
7.3 C++11 Promises & Asynchronous Tasks 244

8 PART-7: TECHNIQUES 252
8.1 Template Meta-Programming 252
8.2 Common C++ Idioms 262
8.3 Translating C++ into C 272

9 SUMMARY 285
9.1 Upcoming Features of C++ 285
9.2 Course End 288

C++ Supplementary with Threads Programming

2019 January

jens.riboe@ribomation.se

1

Course Objectives

- Improve your "++" skills
- Convince you to use Modern C++
- Ensure you know how to utilize the library
- Show you have to design & write multi-threaded applications in Classic and Modern C++

2

Course Contents

- Intro
  - Myths & Facts
  - Evolution of C++
- PART-1: Modern C++ (a)
  - Initialization
  - Auto
  - Misc. improvements
  - Lambdas
  - Move semantics
- PART-2: C++ Refresher
  - Constructors
  - Members
  - Inheritance
  - Const-ness
  - Namespaces
  - Usage of new & delete
  - Templates
  - Operator overloading
- PART-3: C++ Library
  - Text strings
  - Containers
  - Iterators
  - Algorithms
- PART-4: Classic Threads
  - What is a thread
  - POSIX threads
  - Critical sections
  - Race conditions
  - Deadlocks
  - Message passing
  - Rendezvous
- PART-5: Modern C++ (b)
  - Helper Types
  - String views
  - Smart pointers
  - File systems
  - Date & clocks
- PART-6: Modern Threads
  - C++11 threads
  - Synchronization
  - Tasks
- PART-7: Techniques
  - Meta-programming
  - Common idioms
  - Translating C++ into C
  - Upcoming features

3

Course web site

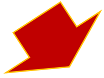
Bookmark this URL

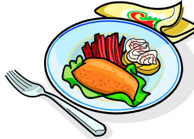
https://gitlab.com/ribomation-courses/cxx/cxx-supplementary-with-threads

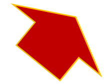
```
$ mkdir -p ~/cxx-course/my-solutions
$ cd ~/cxx-course
$ git clone https://gitlab.com/ribomation-courses/cxx/cxx-supplementary-with-threads.git gitlab
. . .
$ cd ~/cxx-course/gitlab
$ git pull
```

4

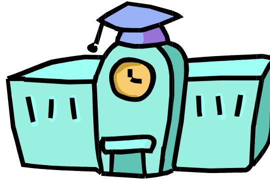
Schedule





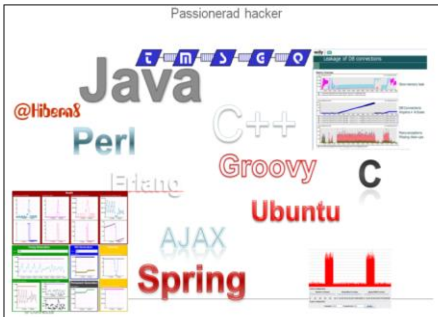


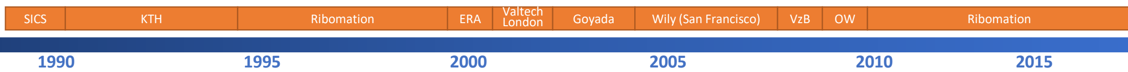
09:00 | 10:15 / 15m | 12:15 / 1h | 14:15 / 15m | 16:30



5

About Jens Riboe





6

Myths & Facts about C++

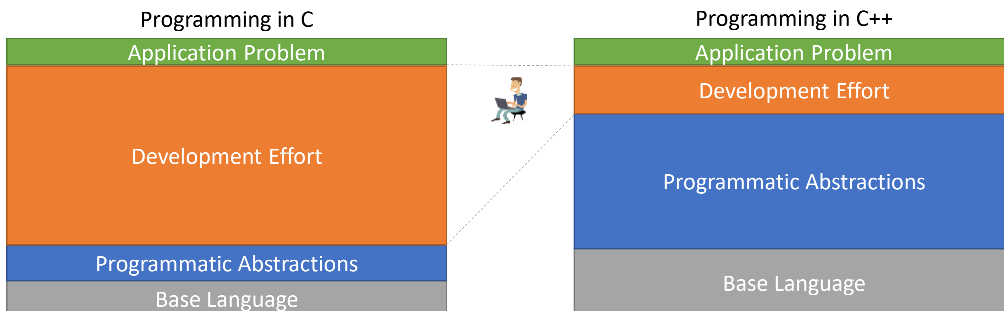
Some developers just change the file extension from *.c to *.cxx and the compiler from gcc to g++, but carries on writing C style spaghetti code, in the false belief it is needed for efficiency.

This course will defeat that misapprehension

7

Myth: C++ is more complex then C

- Yes
  - It takes longer time to master the language and its library.
- However
  - You still have an application problem to solve, which means there is a much longer programmatic path to travel if you have to write & debug all code yourself



8

Myth: C++ generates more code

- No
  - C++ is just a thin compiler layer around C.
  - Slogan: "Don't pay for stuff you don't use"
- In this course we will see
  - What kind of equivalent C code the compiler generates
  - How to turn off certain features

9

Myth: C++ requires more RAM

- No
  - You can choose how to manage your memory requirement
  - To use the heap or not use the heap; that's the question
  - Easy to write re-usable components that optimizes the memory requirement
- In this course we will
  - Look into user-defined memory allocators using areas outside the system heap

10

Myth: C++ hides functionality from the programmer

- No
  - The language and its library is based on solid principles and "battle tested" by many
  - Compared to relaying on home cooked CPP macros it's so much better
- In this course we will
  - See how C++ can be used in an efficient way, using inline functions and templates

11

Myth: Templates are more expensive

- No
  - The generated code is typically more optimized
- Although
  - Extensive use of templates increases the compilation time, but it also helps reducing the execution time
  - Depending on how it's applied, it will increase the code size
    - When code size is a critical concern, it's best to pre-instantiate templates and let the linker relate to the instantiated template functions and classes

12

Fact: There are no run-time overhead in C++

- For conventional languages there are a lot of "magic" happen in run-time
- That's not the case for C++: ***All the magic happens in compile-time***

The fundamental design strategy of C++ is:
You do not pay for features you do not use.

13

Fact: C++ reduces the *time-to-market*

- Because C++ is a strongly typed language, you are forced to express your abstractions in a rigid way and when you slip, the compiler will catch it

14

Evolution of C++

- ✓ Language Inventor
- ✓ Classic C++
- ✓ Modern C++
- ✓ Online Reference

15

Bjarne Stroupstrup (1950 -)

- Inventor and lead developer of C++
  - MSc from Aarhus University in Denmark
  - Relocated to AT&T Bell Labs
- Achievements
  - Started with a C preprocessor
    - "C with Classes", 1979
  - Renamed to C++ in 1983
  - First official publication
    - Software Practice & Experience, 1983
  - First official book
    - The C++ Programming Language, 1987
- Language design philosophy
  - "As close to C as possible, but no closer"

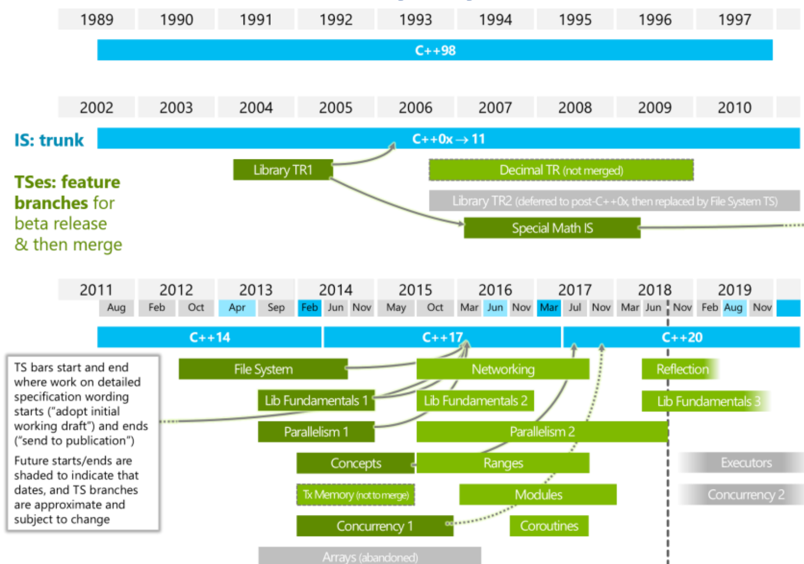
16

Language Evolution of "Modern C++"

- C++11 (2011)
  - Major improvement in both language and library
  - Lambdas, auto (type inference), static initializer lists, move semantics, variadic templates, for-each loop, regex, threads, date & time, random number generators, smart pointers, type traits
- C++14
  - Generic lambdas, more duck typing support, other improvements
- C++17
  - Multi assignments using auto and string views
  - File system functions in the std::lib
  - optional<T>, variant<T>, any<T>
  - Compile-time if statements
- C++20
  - Parallel STL algorithms
  - Ranges

17

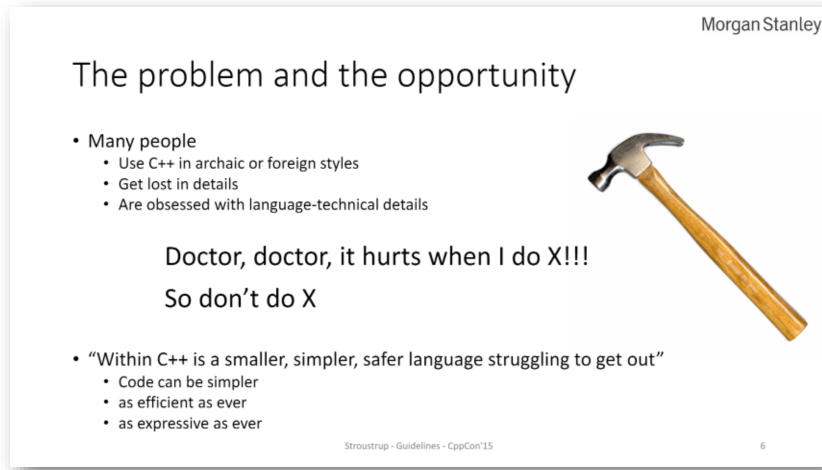
New C++ standard every 3 years



https://isocpp.org/std/status

18

What is Modern C++?



I ❤ C++

Slide from the key note speech by Stroustrup at cppcon October 2015

YouTube: https://youtu.be/1OEu9C51K2A
GitHub: https://github.com/isocpp/CppCoreGuidelines/blob/master/talks/Stroustrup%20-%20CppCon%202015%20keynote.pdf

19

Classic C/C++ vs. Modern C++

```
char* s = strdup("hi there");
char* p=s;
while (*p++ = toupper(*p++)) ;
printf("s: %s (%d)\n", s, strlen(s));

Account* a = new Account(100, 2.5);
a->balance *= 1 - a->rate/100;
cout << "a: "<< a->balance << ", "
     << a->rate << "\n";

if (a->balance < 0) return;

free(s);
delete a;
```

Oooops!!

```
string s = "hi there"s;
transform(s.begin(),s.end(),s.begin(),[](auto ch){
  return ::toupper(ch);
});
cout << "s: " << s << " (" << s.size() << ")\n";

auto a = make_unique<Account>(100, 2.5);
a->balance *= 1 - a->rate/100;
cout << "a: " << *a << endl;

if (a->balance < 0 ) return;
```

The key design aspect of Modern C++ is to primarily use local objects, which have a well-defined life-cycle.

When an object goes out of scope, it disposes any internally dynamically allocated memory blocks.

20

PART-1
Modern C++

```
$ g++ -std=c++17 -Wall -Wextra app.cxx -o app
```

21

Initialization

22

Use {...} for object initialization

- Creating an object
  - `Person p{"Nisse", 42};`
    - `Person p("Nisse", 42);        // classic style still works`
- Invoking the default constructor
  - `Person p{};`
    - `Person p;                     // classic style still works`
    - `Person p();                   // Error or function declaration`
- Non-class types have a pseudo default constructor
  - `int num{};                      // initialized to 0`
    - `int num = 0;                  // classic style still works`
  - `struct stat  info{};            // padded with '\0' bytes`
    - `memset(&info, 0, sizeof(struct stat));   //not needed`

23

Use {...} for container initialization

- Inflating a sequence container
  - `vector<string>  words{"C++", "is", "cool", "!!"};`
  - `vector<string>  words = {"C++", "is", "cool", "!!"};`
- Inflating an associative container
  - ```
    map<string, unsigned> freqs = {
         {"c++", 17U} , {"is", 42U}, {"cool", 3U}
    };
    ```
- Must be a homogenous list of values
  - ```
    vector<int> v = {6, 7, 8.3, 9};
    //compilation error: double->int not allowed
    ```

24

Usage in libraries

```
json j2 = {
  {"pi", 3.141},
  {"happy", true},
  {"name", "Niels"},
  {"nothing", nullptr},
  {"answer", {
    {"everything", 42}
  }},
  {"list", {1, 0, 2}},
  {"object", {
    {"currency", "USD"},
    {"value", 42.99}
  }}
};
```

How to create a JSON object, using the **nlohmann::json** library

```
{
  "pi": 3.141,
  "happy": true,
  "name": "Niels",
  "nothing": null,
  "answer": {
    "everything": 42
  },
  "list": [1, 0, 2],
  "object": {
    "currency": "USD",
    "value": 42.99
  }
}
```

https://github.com/nlohmann/json

25

Initialization statement within an if statement

```
{
  int pos = haystack.find(needle);
  if (pos != string::npos) {
    cout << "yup, got it\n";
  }
  //...
  int pos = whatever.find(phrase); //compilation error
  if (pos != string::npos) {. . .}
}
```

Instead of writing like this

Write it like this

```
{
  if (int pos = haystack.find(needle); pos != string::npos) {
    cout << "yup, got it\n";
  }
  //...
  if (int pos = whatever.find(phrase); pos != string::npos) {. . .}
}
```

26

Initialization statement within a switch statement

```
switch (Record r = load(filename); r.type()) {
  case RecordType::check: ...break;
  case RecordType::save: ...break;
  case RecordType::invest: ...break;
}
//... r is not accessible here ...
```

27

CHAPTER SUMMARY

Initialization

- Always use the new {...} syntax for initialization
  - Unless you need automatic type conversion
- Usage of initializer_list<T> populates an object with compile-time data

28

EXERCISE

Investigate

- Write a small program where you are using the modern initialization syntax
  - Try different types
- Write a small program where you are using initialization within an if statement

29

Intentional Blank

30

Automatic Type Inference

31

Automatic type inference

- The compiler figures out the correct type (~ duck typing)
- The keyword `auto` has gotten a complete new meaning

auto.cpp

```
 1 #include <iostream>
 2 #include <string>
 3 #include <map>
 4 using namespace std;
 5
 6 int main(int numArgs, char* args[]) {
 7     // auto => char*
 8     auto pgmName = args[0];
 9     cout << "Program Name = " << pgmName << endl;
10
11     map<string, int>    words;
12     words["one"] = 1; words["two"] = 2; words["three"] = 3;
13     // auto => map<string, int>::iterator
14     for (auto iter = words.begin(); iter != words.end(); ++iter) {
15         cout << iter->first << " = " << iter->second << endl;
16     }
17
18     return 0;
19 }
```

jens@vbox4: ~/Documents/c++11

```
jens@vbox4:~/Documents/c++11$ g++ -std=c++0x -Wall auto.cpp -o auto
jens@vbox4:~/Documents/c++11$ ./auto
Program Name = ./auto
one = 1
three = 3
two = 2
jens@vbox4:~/Documents/c++11$
```

It's important to understand that C++ still is a strongly typed language. Once the compiler deduced the type, the variable keeps that type.

32

Modern style of variable declaration

- Use auto and place a typed expressions to the right
- Simple variables
  - auto a = 42; //int
  - auto b = 42UL; //unsigned long
  - auto c = 42.L //long double
  - auto d = uint64_t{0xCAFE0042}; //unsigned 64bit word
- Strings
  - auto s = "tjabba"; //const char[7] ~→ const char*
  - auto t = "tjabba"s; //std::string
- Pointers
  - auto p = &number; //int*
  - auto q = new float[3]; //float*
- Objects
  - auto o = Person{"Anna", 42};

33

Multi-assignment (a.k.a. structured binding)

```
tuple<string, unsigned, float>
mkPerson() {
    return make_tuple("Justin Time"s, 42U, 63.7F);
};

void multi_assign_from_tuple() {
    auto[name, age, weight] = mkPerson();
    cout << "name  : " << name << endl;
    cout << "age   : " << age << endl;
    cout << "weight: " << weight << endl;
}
```

```
void multi_assign_from_map_iteration() {
    map<string, string> words = {
            {"one",   "ett"},
            {"two",   "två"},
            {"three", "tre"},
            {"four",  "fyra"},
            {"five",  "fem"},
    };
    for (auto & [en, sw] : words)
        cout << en << " -> " << sw << endl;
}
```

multi-assign

```
/home/jens/Courses/cxx/cxx-embedde
name   : Justin Time
age    : 42
weight: 63.7
----
five -> fem
four -> fyra
one -> ett
three -> tre
two -> två

Process finished with exit code 0
```

34

Functions with auto as return type

```
auto sum(unsigned n) {
    return n * (n + 1) / 2;
}

int main(int argc, char** argv) {
    auto n = argc == 1 ? 10U : stoi(argv[1]);
    cout << "SUM(1.." << n << ") = " << sum(n) << endl;
    return 0;
}
```

```
/mnt/c/Users/jensr/Dropbox/Riboma
SUM(1..10) = 55

Process finished with exit code 0
```

The compiler must be able to deduce the return type from a return expression. For more complex functions with many return statements, it might not work.

```
struct Person {
    string   name;
    unsigned age;
};

ostream& operator<<(ostream& os, const Person& p) {
    return os << "Person{" << p.name << ", " << p.age << "}";
}

auto mk() {
    return Person{"Nisse", 42};
}

int main() {
    auto p = mk();
    cout << "p = " << p << endl;
    return 0;
}
```

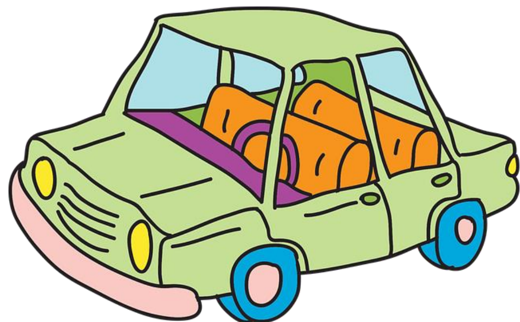
```
/mnt/c/Users/jensr/Dropbox/Ribomat
p = Person{Nisse, 42}

Process finished with exit code 0
```

35

Historical background for keyword auto

- Used to declare a vehicle

```
auto myCar;
```



36

Historical background for keyword `auto`

- Especially for a vehicle parked outdoors

```
extern auto myCar,
```

37

Historical background for keyword `auto`

- Unless, it was in the automobile repair shop

```
static auto myCar;
```

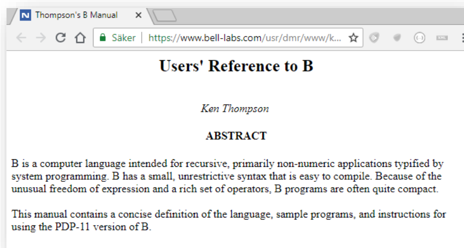
38

Historical background for keyword `auto`

- Jokes, aside

  Language B did not have any types, because a machine word was the default type.
  Keyword auto was used to declare a local variable.
  When Ken Thompson and Dennis Ritchie created C as an extension of B, they had to support keyword auto, to allow linking legacy B code with C.

https://www.bell-labs.com/usr/dmr/www/kbman.html



Thompson's B Manual
https://www.bell-labs.com/usr/dmr/www/k...

Users' Reference to B

Ken Thompson

ABSTRACT

B is a computer language intended for recursive, primarily non-numeric applications typified by system programming. B has a small, unrestrictive syntax that is easy to compile. Because of the unusual freedom of expression and a rich set of operators, B programs are often quite compact.

This manual contains a concise definition of the language, sample programs, and instructions for using the PDP-11 version of B.

```
/* The following function will print a non-negative number, n, to
  the base b, where 2<=b<=10,  This routine uses the fact that
  in the ANSCII character set, the digits 0 to 9 have sequential
  code values.  */

printn(n,b) {
        extrn putchar;
        auto a;

        if(a=n/b) /* assignment, not test for equality */
                printn(a, b); /* recursive */
        putchar(n%b + '0');
}
```


CHAPTER SUMMARY

Automatic Type Inference

- Usage of auto simplifies the code
  - auto x = create(42);
  - auto y = load("things.db");
  - auto z = populate(x, y);
- More reading
  - https://herbsutter.com/2013/08/12/gotw-94-solution-aaa-style-almost-always-auto/

EXERCISE

Fibonacci

$$F(n) = \begin{cases} 0 & \text{om } n = 0; \\ 1 & \text{om } n = 1; \\ F(n-1) + F(n-2) & \text{om } n > 1. \end{cases}$$

- Write a Fibonacci function with return type auto
- Write a function that takes one argument an returns a small struct of the input argument and the computed Fibonacci value
- Write function that takes an argument (n) and populates a std::map with argument (1..n) and Fibonacci number using the previous function and returns the map
- Declare an auto variable that receives the map
- Print out the map

41

Intentional Blank

42

Misc. Syntax Improvements

43

Possible to put the return type last

- A return type of auto can be combined with a concrete type

```
auto funcNamn(params) -> returnType
```

Changing the classic way of declaring functions, can improve the readability of a source file

```
auto split(string s, string delim) -> vector<string> {
    ...
}

auto index(string fileName)
      -> unordered_map<string_view, unordered_set<unsigned>>
{
    ...
}
```

44

For-each

- Given a bunch of elements traverse them all in sequential order

```
vector<int> numbers = {1,2,3,4,5,};
for (int k : numbers) cout << k << endl;

vector<string> words = {"C++", "is", "cool", "!!!"};
for (string w : words) cout << w << endl;

string sentence = "C++ is indeed cool.";
for (char ch : sentence) cout << ch << endl;
```

45

It's even better when combined with auto

```
void for_each_over_int_vector() {
    vector<int> numbers = {1, 2, 3, 5, 8, 13, 21};
    for (auto   n : numbers) cout << n << " ";
    cout << endl;
}

void for_each_over_string_set() {
    set<string> words = {"to", "language", "use", "C++ is a", "cool"};
    for (auto& w : words) cout << w << " ";
    cout << endl;
}

void for_each_over_bounded_array() {
    double    rates[] = {0.015, 0.025, 0.035, 0.075, 0.125, 0.085};
    for (auto r : rates) cout <<fixed<< setprecision(2) << r * 100 << "% ";
    cout << endl;
}
```

```
for-each
/home/jens/Courses/cxx/cxx-embedded/git
1 2 3 5 8 13 21
C++ is a cool language to use
1.50% 2.50% 3.50% 7.50% 12.50% 8.50%

Process finished with exit code 0
```

46

Strongly typed enums

- Classic enums
  - Alias for int → Cannot reuse same symbol in different enum definitions
- Enum classes
  - Properly defined type
  - Might have a storage size specifier

```
enum class Oper {PLUS, MINUS, MULT, DIV};
enum class Html {DIV, SPAN, PRE};
```

```
enum class OP {add, sub, mul, div};
OP    oper = OP::div;

enum class HTML {h1, h2, p, div, br}; //no symbol clash
HTML  html = HTML::div;

oper = html; //ERROR
```

```
#include <cstdint>
enum class Color : std::int8_t {R=0xf0, W=0xff, B=0x0f};
```

47

Integral literals

- Digit grouping makes it easier to read

```
int n = 1'000'000;
```

- Binary literals can be an alternative to hex and octal literals

```
unsigned short w = 0b1100'0000'0110'0001;  //0xC061
```

48

Null pointer literal

- Used to denote a null pointer value
  - As opposed to value 0
  - Keyword: nullptr
  - Type: nullptr_t

```
void  somefunction(int x);   //[1]
void  somefunction(char* p); //[2]
//...
somefunction(NULL);     //invokes [1]
somefunction(nullptr); //invokes [2]
```


Implicit invocation of a constructor when returning

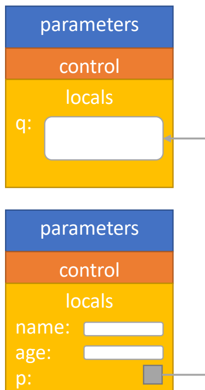
```
Person mk() {
  auto name = …;
  auto age = …;
  return {name, age};
}
```

This assumes a constructor similar to:
`Person::Person(string name, int age)`

Return value optimization (RVO)

```
int main() {
  Person q = mk();
  //...
}
```

```
Person mk() {
  auto name = …;
  auto age = …;
  Person p{name, age};
  return p;
}
```



Using RVO is an implementation technique recommended by the standard.

It means that a returned object is initialized in the stack frame of the caller.

51

CHAPTER SUMMARY

Minor Language Improvements

- Always declare the compiler generated members and do one of:
  - Mark it as default
  - Mark it as delete
  - Implement it
- The support for compile-time computation are increasing for each C++ version

52

EXERCISE

The answer to everything

- Print out the value of 42 using
  - Unsigned
  - Hexadecimal
  - Octal
  - Binary

53

Intentional Blank

54

Lambda Expressions

55

Lambda expressions (a.k.a. closures)

- Anonymous function expressions that can be
  - bound to a variable
  - passed to a function
  - returned from a function
- Introduced in the language
  - C++: with C++11 (2011)
  - Part of many languages
    - Lambda in Java
    - Closures in Groovy
    - Anonymous functions in JavaScript
    - Anonymous subs in Perl
    - and many more

Expression	Meaning
$\lambda x.1$	The constant value 1
$\lambda x.x$	The identity
$\lambda x.x+1$	$f(x) = x + 1$
$\lambda n.\lambda x.x+n$	$f(n) = g$ where $g(x) = x + n$

56

Some simple examples

- Syntax

```
[](params){ . . . return expr;}
```

```
auto  f = [](int n){return n*n;};
int sq = f(5);  //sq == 25
```

```
vector<int>  v = {1,2,3,4,5};
transform(v.begin(), v.end(), v.begin(), [](int n){
    return n * n;
});
//v == 1,4,9,16,25
```

57

Lambda syntax

- Syntax variants

```
[] (param-list) -> returnType { function-body }
```

  - No need to specify return-type, unless the compiler complains

```
[] (param-list) { function-body }
```

  - No need to specify parameter parenthesis, if there are no parameters

```
[] { function-body }
```

- Type inference
  - Use `auto` to declare a lambda variable
  - Possible to declare parameter types as `auto`

```
auto f = [](auto a, auto b) { return 2*a*b; };
int  n = f(3,7);    //n == 42
```

58

Captures of surrounding local variables

- Syntax
  `[capture-list] (params) {body}`
- Capture variables by-value (read-only)
  `[=] (params) {body}`
- Capture variables by-reference
  `[&] (params) {body}`

```
int  a = 10, b = 2;
auto f = [=](auto x) { return a*x + b; };
int  n = f(4);    //n == 42
```

- Individual variables capture (*r by-ref , v by-value*)
  `[&r, v] (params) {body}`
- Capture the owning object (lambda inside member function)
  `[this] (params) {body}`

59

Capture of free variables in action

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> v   = {1, 2, 3, 4, 5};
    int         cnt = 0;
    transform(v.begin(), v.end(), v.begin(), [&](auto n) {
        ++cnt;
        return n * n;
    });
    cout << "cnt=" << cnt << ", v=";
    for (auto n : v) cout << n << " ";
    cout << "\n";
    return 0;
}
```

```
transform(v.begin(), v.end(), v.begin(), [=](auto n) {
    ++cnt;
    return n * n;
});
```

```
[ 50%] Building CXX object CMakeFiles/simple-lambdas.dir/simple-lambdas.cxx.o
/cygdrive/d/CloudStorage/Dropbox (Personligt)/Ribomation/Ribomation -
Training/C++/c-plus-plus__cxx-basics-4-java/src/explorations/simple-lambdas.cxx: In lambda
function:
/cygdrive/d/CloudStorage/Dropbox (Personligt)/Ribomation/Ribomation -
 Training/C++/c-plus-plus__cxx-basics-4-java/src/explorations/simple-lambdas.cxx:10:11:
 error: increment of read-only variable 'cnt'
       ++cnt;
        ^
```

simple-lambdas

```
"D:\CloudStorage\Dropbox (
cnt=5, v=1 4 9 16 25
```

60

Functions taking lambda expressions

- Include
  - <functional>
- Parameter

`function<ReturnType (ParameterType, …)> func`

61

Functions taking lambda expressions in action

```
void repeat(unsigned n, function<void(unsigned)> expr) {
    for (auto k = 1U; k <= n; ++k) expr(k);
}

void map(vector<unsigned>& v, function<unsigned(unsigned)> f) {
    for (int k = 0; k < v.size(); ++k) v[k] = f(v[k]);
}

int main() {
    vector<unsigned> nums;

    repeat(10, [&](unsigned k) { nums.push_back(k); });
    cout << "nums = " << nums << endl;

    map(nums, [](unsigned k) { return k * k; });
    cout << "nums = " << nums << endl;
    return 0;
}
```

```
functions-with-lambda
"D:\CloudStorage\Dropbox (Personligt)\Ribomatic
 Training\C++\c-plus-plus__cxx-basics-4-java\s
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
nums = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Process finished with exit code 0
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;

ostream& operator<<(ostream& os, const vector<unsigned>& v) {
    bool first = true;
    os << "[";
    for (auto k : v) {
        cout << (first ? "" : ", ") << k;
        first = false;
    }
    os << "]";
    return os;
}
```

62

CHAPTER SUMMARY

Lambda Expressions

- Syntax
  - [capture](params) {... return *expr*;}
- The addition of lambda expression to C++ has made it one of the modern and hip languages

63

EXERCISE

Function reduce(arr, N, func)

- Write a function named reduce() that takes
  - numeric array
  - number of elements of the array
  - aggregating lambda that applies a binary function (e.g. +) between all elements

```
[](accumulatedValue, elementValue) { return newAccumulatedValue; }
```

- It should return the aggregated result
- Apply your function and use it to compute the
  - Sum of all values in the array
  - Product of all values in the array
  - Maximum value in the array

64

Move Semantics

65

Performance improvement when returning an object

```
struct Thing {
    static int count;
    explicit Thing(int) { ++count; }
    Thing(const Thing&) { ++count; }
    ~Thing() { --count; }
};
int Thing::count = 0;

vector<Thing> alloc(int n) {
    vector<Thing> words;
    while (--n >= 0) { words.emplace_back(n); }
    cout << "[alloc] count: " << Thing::count << endl;
    return words;
}
```

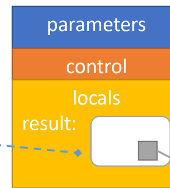
```
move-sematics
/home/jens/Courses/cxx/cxx-embedded/
[first] count: 0
[alloc] count: 1 000 000
[inner] size : 1 000 000
[inner] count: 1 000 000
[last]  count: 0

Process finished with exit code 0
```

```
int main(int, char**) {
    cout.imbue(locale(""));
    cout << "[first] count: " << Thing::count << endl;
    {
        vector<Thing> result = alloc(1'000'000);
        cout << "[inner] size : " << fixed << result.size() << endl;
        cout << "[inner] count: " << Thing::count << endl;
    }
    cout << "[last]  count: " << Thing::count << endl;
    return 0;
}
```

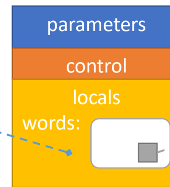
66

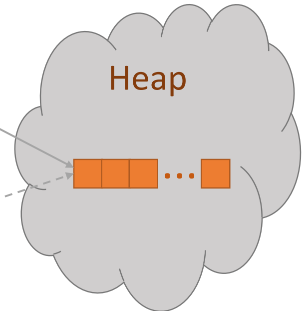
Only the pointer value is transferred at return

```
main(int, char**) {
cout.imbue(locale(""));
cout << "[first] count: " << Thing::count << 
{
    vector<Thing> result = alloc(1'000'000);
```




```
vector<Thing> alloc(int n) {
    vector<Thing> words;
    while (--n >= 0) { words.em
    cout << "[alloc] count: " <
    return words;
}
```

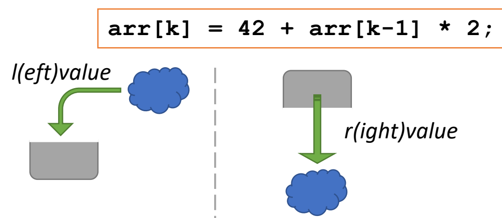




67

New form of reference: rvalue ref (Type&&)

- It is now possible to have a reference to a transient value
  - int value = 42;
  - int&& r = 10 * value;
- In classic C++
  - A returned (transient) object was copied into its destination, using the copy constructor
- In modern C++
  - The destination object is initialized with a move constructor, that takes a rvalue ref



68

Rvalue references: Type&& (modern C++)

- Important new concept in C++ and the re-design of the standard library
- Intended to eliminate unnecessary copying, a.k.a. move-semantics

```
void magic(int& v) {
    cout << "magic(int&) : " << v << endl;
    ++v;
}

void magic(int&& v) {
    cout << "magic(int&&): " << v << endl;
    ++v;
}

int main(int, char**) {
    int value = 10;

    magic(value);
    magic(value + 15);
    magic(5 + 15);
    magic(move(value));

    return 0;
}
```

```
references
/home/jens/Courses/cxx/cxx-embedded
magic(int&) : 10
magic(int&&): 26
magic(int&&): 20
magic(int&&): 11

Process finished with exit code 0
```

69

Move constructor and assignment operator

- Syntax

```
Type(Type&& that)
Type&  operator=(Type&& that)
```

- Used for automatic initialization from transient expressions

70

Most data-types in stdlibc++ provides move semantics

- Important aspect of the design of user-defined types to avoid data copying and ensure it is performant

```
vector<Account>  load(string filename) {
  vector<Account> v;
  //... load 1'000'000 account records ...
  return v;
}

int main() {
  auto accounts = load("accounts.db"s);
  //...
}
```

71

Class Blob

```
class Blob {
    unsigned megaBytes = 0;
    byte*    payload   = nullptr;

public:
    Blob(unsigned megaBytes) : megaBytes{megaBytes}, payload{new byte[megaBytes * (1024U * 1024U)]} {
        cerr << "CREATE Blob{" << megaBytes << "MB} @ " << this
             << ", &payload=" << payload << endl;
    }

    ~Blob() {
        cerr << "~Blob() @ " << this;
        if (payload != nullptr) {
            delete[] payload;
            cerr << ", " << megaBytes << "MB disposed" << ", &payload=" << payload;
        }
        cerr << endl;
    }
```

```
    Blob() = delete;
    Blob(const Blob&) = delete;
    Blob& operator =(const Blob&) = delete;

    friend ostream& operator <<(ostream& os, const Blob& blob) {
        return os << "Blob{" << blob.megaBytes << "MB}";
    }
};
```

72

Class Blob: move operations

```
Blob(Blob&& that) : megaBytes{that.megaBytes}, payload{that.payload} {
    that.megaBytes = 0;
    that.payload   = nullptr;
    cerr << "CREATE Blob{&&" << megaBytes << "MB} @ " << this
         << ", &payload=" << payload << ", moved from Blob @ " << &that << endl;
}
```

```
Blob& operator =(Blob&& that) {
    cerr << "operator =(Blob&&) @ " << this;
    if (this != &that) {
        if (payload != nullptr) {
            delete[] payload;
            cerr << ", " << megaBytes << "MB disposed" << ", &payload=" << payload;
        }
        megaBytes = that.megaBytes;
        that.megaBytes = 0;
        payload = that.payload;
        that.payload = nullptr;
        cerr << ", " << megaBytes << "MB moved from " << &that << ", &payload=" << payload;
    }
    cerr << endl;
    return *this;
}
```

73

Class Blob: app

```
Blob mkOne() {
    Blob nob{50};
    cerr << "[mkOne] nob: " << nob << endl;
    return nob;
}

int main() {
    Blob bob{125};
    cout << "bob: " << bob << endl;

    Blob rob{move(bob)};
    cout << "bob: " << bob << endl;
    cout << "rob: " << rob << endl;

    rob = mkOne();
    cout << "bob: " << bob << endl;
    cout << "rob: " << rob << endl;

    return 0;
}
```

```
move-semantics
/mnt/c/Users/jensr/Dropbox/Ribomation/Ribomation-Training-2017-Autumn/cxx/cxx-17
 /src-exploration/move-semantics/cmake-build-debug/move-semantics
CREATE Blob{125MB} @ 0x7fffd6e278b0, &payload=0x7fb33ffa0010
bob: Blob{125MB}
CREATE Blob{&&125MB} @ 0x7fffd6e278c0, &payload=0x7fb33ffa0010, moved from Blob
 @ 0x7fffd6e278b0
bob: Blob{0MB}
rob: Blob{125MB}
CREATE Blob{50MB} @ 0x7fffd6e278d0, &payload=0x7fb33cd90010
[mkOne] nob: Blob{50MB}
operator =(Blob&&) @ 0x7fffd6e278c0, 125MB disposed, &payload=0x7fb33ffa0010,
 50MB moved from 0x7fffd6e278d0, &payload=0x7fb33cd90010
~Blob() @ 0x7fffd6e278d0
bob: Blob{0MB}
rob: Blob{50MB}
~Blob() @ 0x7fffd6e278c0, 50MB disposed, &payload=0x7fb33cd90010
~Blob() @ 0x7fffd6e278b0

Process finished with exit code 0
```

74

CHAPTER SUMMARY

Move Semantics

- Lvalue reference
  - Type& r = variable;
- Rvalue reference
  - Type&& r = variable * 42;
- Move constructor
  - Type(Type&&)
- Move assignment operator
  - Type& operator=(Type&&)

75

EXERCISE

Moveable string

- Write a very simple string class (pointer to heap allocated char array), that provides move semantics but not copy semantics
- Create one such string and pass it around in and out of a few functions

76

PART-2
C++ refresher

77

Constructors

- ✓ Constructor chaining
- ✓ Member initialization
- ✓ Specially named members
- ✓ Compiler generated members
- ✓ Enforce or delete generated members
- ✓ Type conversion members

78

Constructor

- Invoked when an object is created
  - To ensure the new object is in a well-defined initial state

```
//Account.hpp
class Account {
  string    accno;
  int       balance;
 public:
  Account(string, int);
  //...
};
```

```
//Account.cpp

Account::Account(string an, int b)
{
  accno=an;  balance=b;
}

//...
```

79

Instantiation using a constructor

- Constructor parameters are passed in when an object is created

```
void run() {
  Account  acc{"5237-123456", 150}; //Modern C++ uses {...}
  useAccount(acc);
  cout << "acc = " << acc.toString() << endl;
}
```

```
void run() {
  Account  acc("5237-123456", 150); //Classic C++ uses (...)
  //...
}
```

80

Destructor

- Invoked when an object is disposed
- Same name as constructor, but with ~ and no parameters

```
//Account.hpp
class Account {
  string    accno;
  int       balance;
 public:
  Account(string, int);
  ~Account();
  //...
};
```

```
void run() {
  Account  acc("5237-123456", 150);
  useAccount(acc);
  if (...) return;
  cout << "acc = " << acc.toString() << endl;
}
```

destructor invoked here

81

Use initialization for object members

- A constructor is always invoked for a member object
- Using assignment in the constructor means two function calls, instead of one

```
class Account {
  private:
    string  accno;
    int     balance;
  public:
    Account(char* a, int b) {
      accno   = a;
      balance = b;
    }
    //...
};

Account acc{"1234-567890", 1000};
```

(1) string::string()

(2) string::operator =(char*)

82

Constructor chaining

- Invoking a constructor from another constructor
  - Similar to invoking 'this' in Java

```
class Person {
    string  name;
    int     age;
  public:
    Person(string& n, int a) : name(n), age(a) {}
    Person() : Person{"Per Silja", 42} {}
};
```

83

Specially named constructors and operators

- Default constructor `T::T()`
- Destructor `T::~T()`
- Type-conversion constructor `T::T(X)`
- Type-conversion operator `T::operator X()`
- Copy constructor `T::T(const T&)`
- Copy assignment operator `T& T::operator=(const T&)`
- Move constructor `T::T(T&&)`
- Move assignment operator `T& T::operator=(T&&)`

84

Default constructor

- Syntax

```
T::T()
```

- Compiler generated version (if no constructors)
  - Empty
- Used for default initialization of
  - Simple object declaration

```
Type t;
```

  - Array objects

```
Type  array[5];  //invoked for each object
```

  - Super class

```
class A {};
class B : public A {};
```

  - Member object (with no initialization)

```
class A {X   x;};
```

85

Destructor

- Syntax
  - T::~T()
- Used for object clean-up of dynamic members
- Compiler generated version
  - Empty
- Automatically called when
  - A local object goes out of scope by return or throw

```
void compute() {
  string s;
  if (…) return; →
  if (…) throw "Bye, bye"; →
  ...
} →
```

  - Operator delete is invoked

```
Person* ptr = new Person;
delete ptr; →
```

  - The destructor is invoked directly

```
ptr->~Person();
```

86

Copy constructor

- Syntax
  - T::T(const T&)
- Compiler generated version
  - Member wise assignments
- Used for automatic object copy
  - An object initialized with another object of the same type

```
string  s{"hello"}
string  t{s};
```

  - Call by-value

```
void print(Person p) { ... p ... }
Person  anna;
print(anna);
```

  - Return by-value

```
Person create() {Person p; ... return p;}
Person bea = create();
```

87

Remarks of the copy constructor

- You need to invoke the copy constructor for members and super classes as well

```
class Person {
    string   name;
  public:
    Person(string n) : name(n) {}
    Person(const Person& that) : name{that.name} {}
};
```

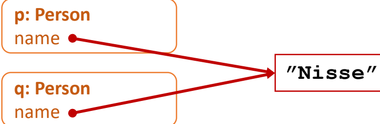
88

Pitfall

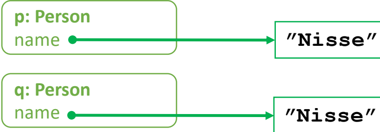
- Member pointers → implement a copy constructor

```
class Person {
  char*   name;
  public:
  Person(char* n) : name(n) {}
};
```

```
Person  p("Nisse");
Person  q = p;
```




```
class Person {
  char*   name;
  public:
  Person(char* n) : name(n) {}
  Person(const Person& p)
    : name( strdup(p.name) )
  { }
};
```



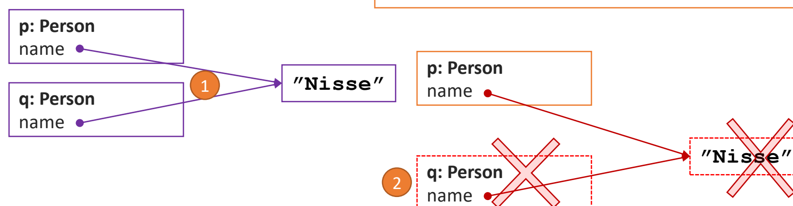
89

Pitfall

- Need a destructor? Then you need a copy constructor as well!

```
class Person {
  char*   name;
  public:
  Person(char* n) : name(n) {}
  ~Person() {delete name;}
};
```

```
void print(Person q) {
  ...q.getName()...
} (2)

Person  p("Nisse");
(1) print(p);
char* x = p.getName(); //oops
```



90

Type conversion constructor

- Syntax
  - T::T(X)
- Used to convert from some other type X into your type T
- Often invoked implicitly by the compiler

```
class Money {
  int     value = 0;
  string  currency = "SEK";
 public:
  Money(int v) : value(v) {}
  void add(int n) {value+=n;}
  //...
};
```

```
Money useMoney(Money x) {
  x.add(42);
  return x;
}
```

```
void run() {
  Money  m{100};
  m = useMoney(250);
}
```

```
void run() {
  Money  m{100};
  m = useMoney( Money(250) );
}
```

91

Prevent implicit type conversion: T::T(X)

- Keyword ‘explicit’, might catch some nasty bugs

nocopy.cpp ✖ no-coercion.cpp ✖

```
1 #include <iostream>
2 #include <string>
3 #include <cstring>
4 using namespace std;
5
6 class Wrapper {
7     char*  str;
8   public:
9     Wrapper(int sz) : str(new char[sz]) {memset(str, '.', sz); str[sz] = '\0'
10    Wrapper(const char* s) : str(strdup(s))     {}
11    ~Wrapper() {delete str;}
12    string  toString() { return string("Wrapper[") + str + "]"; }
13 };
14
15 void print(int id, Wrapper w) { cout << "["<<id<<"] " << w.toString() << endl; }
16
17 int main() {
18     print(1, "5");
19     print(2, '5'); //BUG: ASCII('5') = 53
20     print(3, 5);
21     print(4, Wrapper("5") );
22     print(5, Wrapper('5') );
23     print(6, Wrapper(5) );
24     return 0;
25 }
```

```
6 class Wrapper {
7     char*  str;
8   public:
9     explicit Wrapper(int s
10    Wrapper(const char* s)
11    ~Wrapper() {
12    string  toSt
```

```
cuments/c++11$ g++ -std=c++0x -Wall no-coercion.cpp -o no-coercion
: In function ‘int main()’:
:19:17: error: invalid conversion from ‘char’ to ‘const char*’ [-fpermissive]
:10:5: error:   initializing argument 1 of ‘Wrapper::Wrapper(const char*)’ [-fpe
:20:15: error: invalid conversion from ‘int’ to ‘const char*’ [-fpermissive]
:10:5: error:   initializing argument 1 of ‘Wrapper::Wrapper(const char*)’ [-fpe
uments/c++11$
```

jens@vbox4: ~/Documents/c++11

```
jens@vbox4:~/Documents/c++11$ g++ -std=c++0x -Wall no-coercion.cpp -o no-coercion
jens@vbox4:~/Documents/c++11$ ./no-coercion
[1] Wrapper[5]
[2] Wrapper[.....................................................]
[3] Wrapper[.....]
[4] Wrapper[5]
[5] Wrapper[.....................................................]
[6] Wrapper[.....]
jens@vbox4:~/Documents/c++11$
```

92

Type conversion *operator*

- A member operator that returns a value of the target type
  - T:: operator X()
- N.B.: It's declared without explicit return type

```
Money::operator int() {
  return value;
}

Money  m(42, "EUR");
int  n = m;
```

```
//Interpretation
int  n = m.operator int();
```

93

Compiler generated members

Member	Generated	Body
`T()`	if no other constructors	empty
`~T()`	if no destructor	empty
`T(const T&)`	if no move constructor/assignment	member-wise copying
`T& operator=(const T&)`	if no move constructor/assignment	member-wise copying
`T(T&&)`	if no destructor and no copy/move constructor/assignment	member-wise move
`T& operator=(T&&)`	if no destructor and no copy/move constructor/assignment	member-wise move

94

Enforce compiler-generated members

- Reasons, why
  - Change the visibility into non-public
  - Force generation, when it would not
  - Make a generated destructor virtual
  - Documentation purpose

```
class Whatever {
  protected:
    Whatever() = default;
  public:
    virtual ~Whatever() = default;
    Whatever(const Whatever&) = default;
    Whatever&  operator=(const Whatever&) = default;
};
```

95

Delete compiler-generated members

- Reasons, why
  - Compiler supported semantics
  - Documentation purpose

```
class Whatever {
    string payload;
  public:
    Whatever(const string& s) : payload(s) {}

    Whatever() = delete;
    Whatever(const Whatever&) = delete;
    Whatever&  operator=(const Whatever&) = delete;
};
```

96

CHAPTER SUMMARY

Constructors

- Special members
  - T::T() / T::~T()
  - T::T(X) / T::operator X()
  - T::T(const T&) / T::T(T&&)
  - T& T::operator =(const T&) / T& T::operator =(T&&)

97

EXERCISE

Number class

- Design & implement a number class that contains a number
- Provide type conversion constructor(s) and operator(s)

98

Members

99

Self reference - this

- Each object has an implicitly declared pointer that always points to itself
  - Type* const this;

```
int Account::update(int amount) {
  this->balance += amount;
  return this->balance;
}
```

redundant → not needed

```
Account& Account::update(int amount) {
  this->balance += amount;
  return *this;
}
//...
Account a;
a.update(100).update(50).update(25);
```

Returns a reference to the object itself.
Used to support cascade-calls

100

Inline member functions

- It is possible implement member function bodies direct in the class (a.k.a. Java style), which will mark them as inline

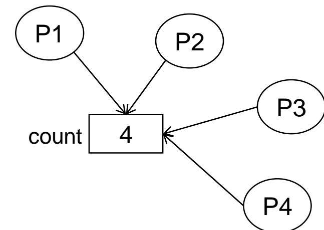
```
//filename: Account.hxx
class Account {
  string    accno;
  int       balance;

 public:
  Account(string an, int b)  {accno=an; balance=b;}
  int     update(int amount) {return balance += amount;}
  int     getBalance()       {return balance;}
  string  toString()         {return accno+":"+to_string(balance);}
};
```

101

Class variables

- Global variables within the namespace of the class

```
class Person {
  private:
   static int  instanceCount;
   ...
  public:
    Person() {instanceCount++; ...}
    ~Person() {instanceCount--; ...}
    ...
};
```



- You *must* create the allocation (*.cxx file)

```
int  Person::instanceCount = 0;
```

This is very easy to forget

102

Class functions

- Functions that operates on class variables
  - No ’this’ within the function

```
class Person {
  ...
  static void  resetCount() {instanceCount = 0;}
};
```

- Invoke using the class name

```
Person::resetCount();
```

103

Member references

- A member variable can be declared as a reference
- Must be initialized (not assigned) using the constructor initialization list

```
class Foo {
  private:
    int&  ref;
  public:
    Foo(int& n) : ref(n) {}
    void  magic() {ref *= 10;}
};

int  number = 42;
Foo  f( number );
f.magic();
//assert: number == 420
```

104

Inner types

- A public inner class is visible for the "outside world"
- A private inner class is hidden from the "outside world"

```
class List {
    struct Node { //private inner class
      int    data;
      Node*  next;
      Node(int d, Node* n) : data(d), next(n) {}
    };

    Node*  head = nullptr;
 public:
    List() {head = nullptr;}
    void  insert(int n) {head = new Node(n, head);}
    //...
};
```

An object of an inner class cannot access members of the outer class, unless it has a pointer or reference to it.

105

Friends can access private members

- For implementation reasons of related non-member functions and/or classes, they can be allowed access to private members by declaring them as a friend

```
class Whatever {
  int x, y;
 public:
  Whatever() {...}
  friend void print(Whatever&);
};

void print(Whatever& c) {
  cout << c.x << "," << c.y << end;
}
```

```
Whatever w;
print(w);
```

The is the classical joke from the Usenet group comp.lang.c++:

"Dear Bjarne.
I have a child object Daughter, and she is letting all of her friends have access to her private members.
What should I do?"

106

Friend member functions

- Possible to define a friend function inside a class
- Technically, it's still not a member, just a friend

```
class Whatever {
  int x, y;
 public:
  Whatever() {...}
  friend void print(Whatever& c) {
    cout << c.x << "," << c.y << end;
  }
};
```

```
Whatever w;
print(w);
```

107

CHAPTER SUMMARY

Members

- The pointer 'this' is available in every non-static member function
- Header-only classes has started be very common
- Remember to allocate storage to a static member variable
- Declare a function as friend if it's strongly connected to a class

108

EXERCISE

Instance count

- Write a header-only class with instance count
- Ensure it has a friend(-ly) print function, that also prints out the count
- Create a few objects inside a scoped block and print them out
- After the scoped block, print out the count and verify it's 0

109

Intentional Blank

110

Inheritance

111

Base class visibility

- class Sub : **public** Super {}
  - All inherited visible members are public in the sub class
  - This is the normal form of inheritance
- class Sub : **protected** Super {}
  - All inherited visible members are at most protected in the sub class
- class Sub : **private** Super {}
  - All inherited visible members are private in the sub class
- class Sub : Super {}
  - Same as *private* inheritance

That means;
don't forget public

112

Default visibility of `struct` is public

- Members are public, unless declared private/protected
- Subclasses inherits public, unless declared otherwise

```
struct Base {
    int num;
    Base(int n) : num(n) {}
    virtual int magic() const { return num; }
};

struct Sub : Base {
    Sub(int n) : Base(n) {}
    int magic() const override {
        return Base::magic() * 10;
    }
};

int main(int argc, char* argv[]) {
    Base b(10);
    b.num = 42;
    cout << "b.magic=" << b.magic() << endl;

    Sub s(10);
    s.num = 17;
    cout << "s.magic=" << s.magic() << endl;

    return 0;
}
```

Terminal

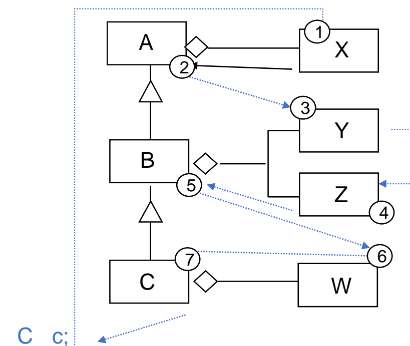
```
cmake-build-debug $ ./interfaces
b.magic=42
s.magic=170
cmake-build-debug $
```

113

Initialization order

- Constructor invocation order
  1. Super class constructor
  2. Member objects
  3. Constructor body
- Destructor invocation order is the reverse

```
class A {
  X x;
};
class B : public A {
  Y y;
  Z z;
};
class C : public B {
  W w;
};
```



114

Invoking super similar to Java

```
class LandVehicle : public Vehicle {
    using super = Vehicle;
    unsigned numWheels = 4;
public:
    LandVehicle(const string r, unsigned n) : super(r), numWheels(n) {}
    string toString() const override {
        return super::toString() + ", numWheels=" + to_string(numWheels);
    }
};

class Car : public LandVehicle {
    using super = LandVehicle;
    bool turbo = true;
public:
    Car(const string& r) : super(r, 4) {}
    string toString() const override {
        return super::toString() + ", turbo=" + (turbo ? "yes" : "no");
    }
};
```

```
class Vehicle {
    string regno;
public:
    Vehicle(const string& r) : regno(r) {}
    virtual string toString() const {
        return "regno=" + regno;
    }
};
```

```
int main() {
    Vehicle* v = new Car{"ABC123"};
    cout << "v = " << v->toString() << endl;
    return 0;
}
```

super-usage

```
"D:\CloudStorage\Dropbox (Personligt)\Ribo
v = regno=ABC123, numWheels=4, turbo=yes

Process finished with exit code 0
```

115

Down casts

- Convert a super-class pointer to a sub-class ditto
- Use the dynamic_cast operator
  - Returns a pointer of the request type
  - Or nullptr, if the target is of another type
- In case of passing a super-class reference and the target is invalid, it throws an exception (bad_cast) instead

```
void doit(Vehicle* v) {
  if (Car* c = dynamic_cast<Car*>(v); c != nullptr) {
    c.setTurbo(true);
  } else {...}
}
```

```
void doit(Vehicle& v) {
  try {
    Car&  c = dynamic_cast<Car&>(v);
    c.setTurbo(true);
  } catch (bad_cast) {...}
}
```

116

Virtual methods

- A virtual method provides dynamic method binding

```
class Vehicle {
  ...
  virtual string toString();
};

class LandVehicle : public Vehicle {
  ...
  string toString();
};

class Car : public LandVehicle {
  ...
  string toString();
};
```

```
void print(Vehicle& v) {
  cout << v.toString() endl;
}
```

```
Car  volvo;
print(volvo);          Car::toString()

MC  hd;
print(hd);             MC::toString()

Airplane boing;
print(boing);          Airplane::toString()
```

117

Virtual destructor

- Always declare a virtual destructor for root classes

Without a virtual destructor

```
class Vehicle {
  ...
  ~Vehicle();
};
class LandVehicle...;
class Car...;

Vehicle* vp = new Car;

delete vp; //Vehicle::~Vehicle()
```

With a virtual destructor

```
class Vehicle {
  ...
  virtual ~Vehicle();
};
class LandVehicle...
class Car...;

Vehicle* vp = new Car;

delete vp; //Car::~Car()
```

118

Abstract methods and classes

- An abstract method has the function body replaced with '= 0'
- A sub class must implement the abstract methods

```
class Vehicle {
  public:
    ...
    virtual void moveForward(Velocity) = 0;
};
class LandVehicle : public Vehicle {
    ...
    virtual void moveForward(Velocity) = 0;
};
class Car : public LandVehicle {
    ...
    void moveForward(Velocity v) {...v...}
};
```

Abstract method

Concrete method

119

Mark an overridden method

- A method declared as override
  - → Must have a method with the same signature in its base-class

```
struct Factory {
  virtual Product  mk(string type);
};

struct MyFactory : Factory {
  Product  mk(string type) override;
};

struct FaultyFactory : Factory {
  Product  mK(string type) override;
  Product  create(string type) override;
}
```

Will not compile

120

An overridden method can be marked as final

- When an overridden want to prevent further overrides in sub-classes, mark it as final override

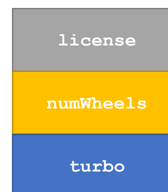
```
struct Factory {
  virtual Product  mk(string type);
};

struct MyFactory : Factory {
  Product  mk(string type) final override;
};

struct FaultyFactory : MyFactory {
  Product  mk(string type) override; 
}
```

Will not compile

121

Memory layout for inheritance

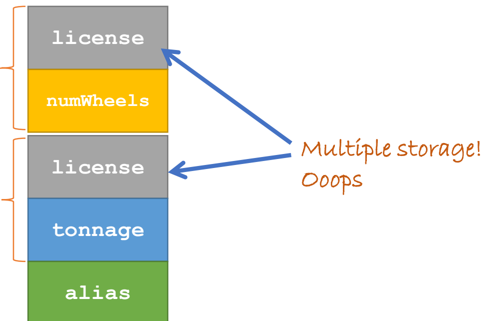
```
class Vehicle {
  string  license;
  //. . .
};

class LandVehicle : public Vehicle {
  int  numWheels;
  //. . .
};

class Car : public LandVehicle {
  bool  turbo;
  //. . .
};

Car volvo;
```



122

Memory layout for multiple inheritance

```
class Vehicle {
  string  license;
  . . .
};
class LandVehicle : public Vehicle {
  int  numWheels;
  . . .
};
class WaterVehicle : public Vehicle {
  int  tonnage;
  . . .
};
class JamesBondVehicle
  : public LandVehicle, public WaterVehicle
{
  string alias;
  . . .
};

JamesBondVehicle bmw;
```



123

"Java Interfaces" within C++

- C++ do not have interfaces.
  However, it's easy to simulate it using structs with abstract methods and struct inheritance

```
struct Stringable {
    virtual string toString() const = 0;
};

template<typename T>
struct Comparable {
    virtual int compareTo(const T& rhs) const = 0;
};
```

```
struct Person : Stringable, Comparable<Person> {
private:
    string name;
public:
    Person(const string& s) : name{s} {}

    string toString() const override {
        ostringstream buf;
        buf << "Person{" << name << "}";
        return buf.str();
    }

    int compareTo(const Person& rhs) const override {
        return name.compare(rhs.name);
    }
};
```

124

"Java Interfaces" within C++, demo

```cpp
void dump(const Stringable& obj) {
    cout << "*** " << obj.toString() << " @ " << &obj << endl;
}

int main(int argc, char* argv[]) {
    Person p1{"Sham Poo"};
    dump(p1);

    vector<Stringable*> persons = {
            new Person{"Anna Conda"}, new Person{"Per Silja"}, new Person{"Inge Vidare"},
            new Address{"17 Hacker Street", "SEA PP"}, new Address{"42 Reboot Lane", "ASM"}
    };
    for (auto p : persons) dump(*p);

    Person p2{"Sham Poo"};
    if (p1.compareTo(p2) == 0) cout << "p1 and p2 are equals\n";

    return 0;
}
```

```cpp
struct Address : Stringable {
private:
    string street, city;
public:
    Address(const string& s, const string& c) : street{s}, city{c} {}

    string toString() const override {
        ostringstream buf;
        buf << "Address{" << street << ", " << city << "}";
        return buf.str();
    }
};
```

Terminal

```
cmake-build-debug $ ./interfaces
*** Person{Sham Poo} @ 0x7fffe17983c0
*** Person{Anna Conda} @ 0x994c20
*** Person{Per Silja} @ 0x994c60
*** Person{Inge Vidare} @ 0x994ca0
*** Address{17 Hacker Street, SEA PP} @ 0x994ce0
*** Address{42 Reboot Lane, ASM} @ 0x994d50
p1 and p2 are equals
cmake-build-debug $
```

125

CHAPTER SUMMARY

Inheritance

- Default visibility for a class is private, but for a struct it's public
- Always declare a destructor as virtual, if there are sub-classes
- Do not use multiple inheritance, if the base-classes contains variables

126

EXERCISE

Demo

- Write small demo programs that applies the following topics
  - struct inheritance
  - Down-cast
  - Java-style 'super'
  - Java-style 'interface'
- Write a program that demonstrates the problem of multiple inheritance

127

Intentional Blank

128

Const-ness

129

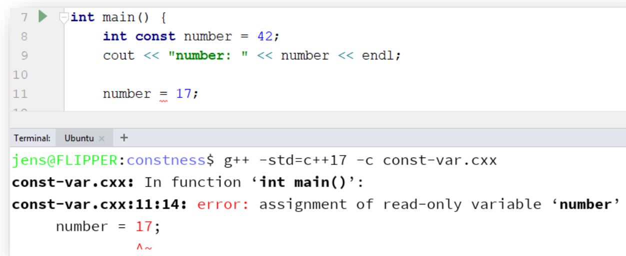
A variable can be read-only

- Declaring an initialized variable as const, mark it as read-only

```
const int  answer = 42;
const auto pi     = 3.141592654;
```

Swapping the placement of const and the type works as well.
As a matter of fact, it's now the recommended style.

```
int  const answer = 42;
auto const pi     = 3.1415926;
```



130

Reference can be const

- Possible (and common) to let a reference be const, but not what if refers to

```
{
    int         value = 42;
    const int&  ref   = value;
    cout << "value: " << value << ", ref: " << ref << endl;
    ++value;
    cout << "value: " << value << ", ref: " << ref << endl;
//  ++ref; --> error: increment of read-only reference 'ref'
}
```

```
number: 42
value: 42, ref: 42
value: 43, ref: 43

Process finished with exit code 0
```

131

A pointer can also be marked as read-only

- However, a pointer consist of two parts, the address and what's on the address. Both parts can be marked individually.

```
const int*        ptr = &number;
int*   const      ptr = &number;
const int*  const ptr = &number;
```

132

Error messages

```
int number = 42;
int other  = 17;
{
    int* ptr = &number;
    cout << "*ptr=" << *ptr << ", &ptr=" << ptr << endl;
    *ptr = *ptr * 2;
    cout << "*ptr=" << *ptr << ", &ptr=" << ptr << endl;
    ptr = &other;
    cout << "*ptr=" << *ptr << ", &ptr=" << ptr << endl;
}
{
    const int* ptr = &number;
//  *ptr = other; --> error: assignment of read-only location '* ptr'
    ptr = &other;
}
{
    int* const ptr = &number;
    *ptr = other;
//  ptr = &other; --> error: assignment of read-only variable 'ptr'
}
{
    const int* const ptr = &number;
//  *ptr = other; --> error: assignment of read-only location '*(const int*)ptr'
//  ptr = &other; --> error: assignment of read-only variable 'ptr'
}
```



```
/mnt/c/Users/jensr/Dropbox/Ribomat
*ptr=42, &ptr=0x7fffc26f7310
*ptr=84, &ptr=0x7fffc26f7310
*ptr=17, &ptr=0x7fffc26f7314

Process finished with exit code 0
```

133

Read-only member variables

- When a member variable is marked with const, it must be initialized either by
  - Variable declaration; if the value always is the same
  - Initialization list; if the value is provided by the constructor

```
class Foo {
  const double PI = 3.1415926;
  const string filename;
public:
  Foo(string name) : filename{name} {}
  //...
};
```

134

Read-only member functions

- Possible to mark a member function as const
- A const reference function argument object can only invoke member functions marked as const

```
class Foo {
  int theValue = 42;
public:
  int getValue() const {return theValue;}
};
```

135

Cannot invoke a non-const member function in a const context

```
class Account {
    int balance = 42;
public:
    Account()   = default;
    ~Account() = default;
    int  getBalance() const { return balance; }
    void update(int amount) { balance += amount; }
};

void print(const Account& a) {
    cout << "Account{balance=" << a.getBalance() << "}\n";
}

void modify(const Account& a) {
  // a.update(100); --> error: passing 'const Account' as 'this' argument discards qualifiers
}

int main() {
    Account acc{};
    print(acc);
    modify(acc);
}
```

```
Account{balance=42}

Process finished with exit code 0
```

```
const-member-func.cxx: In function 'void modify(const Account&)':
const-member-func.cxx:19:15: error: passing 'const Account' as 'this' argument discards qualifiers
   a.update(100); //--> error: passing ?const Account? as ?this? argument discards qualifiers
               ^
const-member-func.cxx:11:10: note:   in call to 'void Account::update(int)'
     void update(int amount) { balance += amount; }
```

The message from GCC, left some to desire when it comes to clarity, what it exactly mean!

136

Member functions can be overloaded based on const-ness

```
class Account {
    int balance = 10;
public:
    int getBalance()       { return balance - 5; }
    int getBalance() const { return balance + 5; }
};

void print(Account& a) {
    cout << "       Account&: " << a.getBalance() << "\n";
}
void print(const Account& a) {
    cout << "const Account&: " << a.getBalance() << "\n";
}
int main() {
    Account acc{};        print(acc);
    const Account acc2{}; print(acc2);
}
```

```
/mnt/c/Users/jensf/Dropbox/KTbolla
       Account&: 5
const Account&: 15

Process finished with exit code 0
```

137

Lifting const-ness temporarily

- Using the const-lifting operator
  - const_cast<int>(someConstVar)

```
void modify2(const Account& a) {
    const_cast<Account&>(a).update(100);
}

int main() {
    Account acc{}; print(acc);
    modify2(acc);  print(acc);
}
```

```
/mnt/c/Users/jensf/Dropbox/KTbolla
Account{balance=42}
Account{balance=142}

Process finished with exit code 0
```

138

Lifting const-ness permanently

- Declaring a variable as mutable
  - mutable int value = 42;
- That mean it will co-exist with const marked member functions

```cpp
class Account {
    mutable pthread_mutex_t   mutex{};
    int                       balance{42};
public:
    Account()  { pthread_mutex_init(&mutex, nullptr); }
    ~Account() { pthread_mutex_destroy(&mutex); }
    int getBalance() const {
        // without marking the mutex as mutable, it will not compile
        // error: invalid conversion from 'const pthread_mutex_t*' to 'pthread_mutex_t*'
        pthread_mutex_lock(&mutex);
        auto const result = balance;
        pthread_mutex_unlock(&mutex);
        return result;
    }
};

int main() {
    Account  acc{};
    cout << "acc: " << acc.getBalance() << endl;
    return 0;
}
```

```
acc: 42

Process finished with exit code 0
```

139

CHAPTER SUMMARY

Const-ness

- Variable
  - Type const var = expr;
- Pointer
  - const Type*   ptr = expr;
  -         Type* const ptr = expr;
  - const Type* const ptr = expr;
- Const member variables must be initialized
  - In its declaration , or
  - At a constructor's initialization list
- A const member function is used in a const context
- Declare a member as mutable, if it has to be modified in a const context

140

EXERCISE

Demo const-ness

- Write your own demo code to illustrate the various error messages demonstrated in this chapter

141

Intentional Blank

142

Namespaces

143

Namespaces

- Namespaces are used as a syntactic module concept,
  in a similar spirit as in Java, but besides that they are very different
- A namespace defines a named scope for identifiers (function, classes)
- Can be nested, but there is no correspondence with directories
- The 'using' keyword has some remote similarity with 'import' in Java
- Imports from a namespace
  - `using namespace XYZ;` //import all names from XYZ
  - `using XYZ::MMM;` //import just MMM from XYZ

144

The standard library uses the `std` namespace

```
#include <iostream>
#include <string>
int main()  {
  std::string  message = "Hello from C++";
  std::cout << message << std::endl;
  return 0;
}
```

Using the full names

Imports can be file global or function local.

Selective import of a name

```
#include <iostream>
#include <string>

int main()  {
  using std::string;
  string  message = "Hello from C++";
  std::cout << message << std::endl;
  return 0;
}
```

Import of all names

```
#include <iostream>
#include <string>
using namespace std;

int main()  {
  string  message = "Hello from C++";
  cout << message << endl;
  return 0;
}
```

145

Namespace definitions are additive

Stack.hxx

```
namespace MyLib {
  class Stack {...};
}
```

List.hxx

```
namespace MyLib {
  class List {...};
}
```

App.cxx

```
#include "Stack.hxx"
#include "List.hxx"
using namespace MyLib;

int main()  {
  . . .
  Stack stk;
  List  lst;
  . . .
}
```

146

Usage of a nested namespace

Thing.hxx
```
namespace se {
  namespace ribomation {
    namespace app {
      class Thing {...};
    }
  }
}
```

Thing.hxx
```
namespace se::ribomation::app {
  class Thing {...};
}
```

In C++17 it's possible to define it more compactly

App.cxx
```
using namespace se::ribomation::app;

Thing  theThing{"The Addams Family", 666};
```

147

Usage of an anonymous namespace

- Defines a set of file private items (functions, variables)
  - Similar to declaring them as `static`
- Used to
  - Avoid name clashes between compilation units
    - AKA: multiple defined symbol
  - Restricted import of other namespaces, such as std

```
namespace {
  using namespace std;
  string  toUpper(const string& s) {...}
}
```

148

functions.hxx
```cpp
#pragma once

namespace ribomation {
    unsigned long factorial(unsigned);
}
```

functions.cxx
```cpp
#include <iostream>
#include <string>
#include "functions.hxx"

namespace {
    using namespace std;

    string greeting() {
        return "Hello from the FUNCTIONS module";
    }

    unsigned long factorialUsingRecursion(unsigned n) {
        return n <= 1 ? 1 : n * factorialUsingRecursion(n - 1);
    }
}

namespace ribomation {
    unsigned long factorial(unsigned n) {
        using namespace std;
        cout << greeting() << endl;
        return factorialUsingRecursion(n);
    }
}
```

app
```cpp
#include <iostream>
#include <string>
#include "functions.hxx"

namespace {
    using namespace std;
    string greeting() {
        return "Hello from the APP module";
    }
}

int main() {
    std::cout << greeting() << std::endl;
    std::cout << ribomation::factorial(5) << std::endl;
    return 0;
}
```

namespaces
```
/home/jens/Courses/cxx/cxx-embedded/s
Hello from the APP module
Hello from the FUNCTIONS module
120

Process finished with exit code 0
```

```
jens@vbox2:~/Courses/cxx/cxx-embedded/src/explorations/namespaces$ nm --demangle --line-numbers --format=posix --radix=d  ./cmake-build-debug/namespaces | grep greet
(anonymous namespace)::greeting() t 0000000000006282 0000000000000134   /home/jens/Courses/cxx/cxx-embedded/src/explorations/namespaces/app.cpp:8
(anonymous namespace)::greeting() t 0000000000009160 0000000000000134   /home/jens/Courses/cxx/cxx-embedded/src/explorations/namespaces/functions.cxx:8
jens@vbox2:~/Courses/cxx/cxx-embedded/src/explorations/namespaces$
```

149

CHAPTER SUMMARY

Understanding Namespaces

- Importing a single member
  - using ns::member;
- Importing all members
  - using namespace ns;
- Anonymous namespaces are better than usage of static

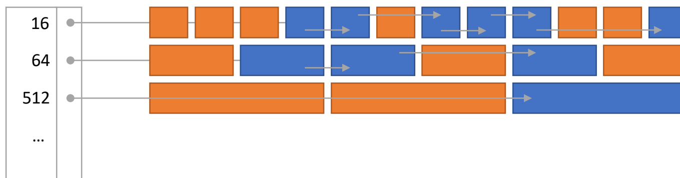
150

Usage of new & delete

151

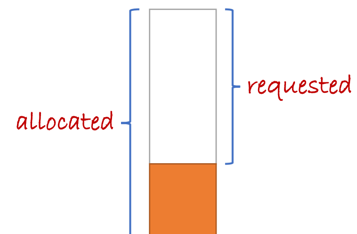
What is dynamic memory?

- Allocation of a memory block from a special memory area (HEAP)
- The HEAP management system handle blocks of standard sizes
- An allocated block must be returned to the HEAP (reclaimed)



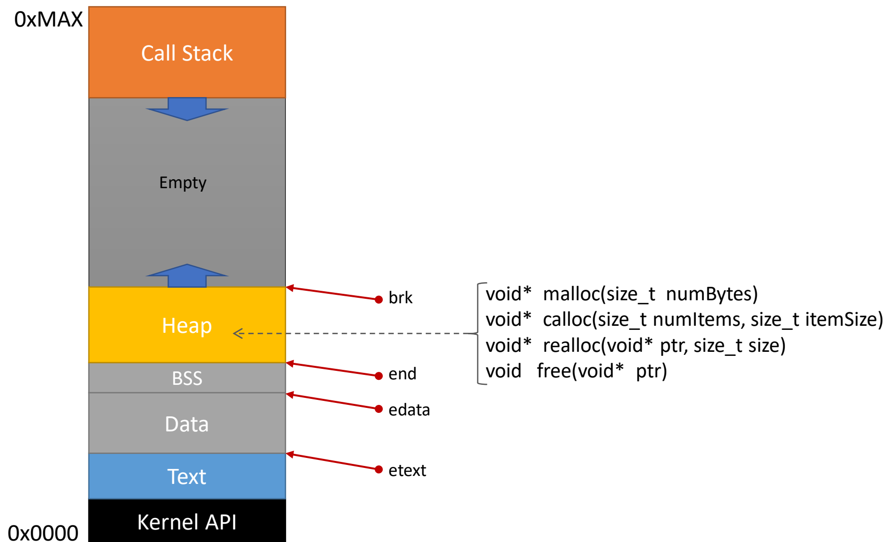
`malloc()` Find an available block; possible by combining/splitting blocks. If not found, invoke sbrk() and start over.

`free()` Insert the block into the corresponding list of available blocks.



152

Allocation functions



153

The new & delete operators

- Allocate a memory block (1) and run the constructor (2)

```
T*  ptr1 = new T;        //invoke T()
T*  ptr1 = new T{};      //invoke T()
T*  ptr2 = new T{arg1, arg2, …};
```

  - Throws `std::bad_alloc`, if no more heap storage
  - Don't write `new T()` the compiler will call function T() and then invoke new
- Run the destructor (1) and release the memory block (2)

```
delete ptr;
```

154

Sample usage of new & delete

```
int*  number = new int{42};
*number *= 10;
cout << "number=" << *number << endl;
delete number;
```

```
Person* p = new Person{"Bob", 42};
p->setName("Nisse");
cout << "p.name=" << p->getName() << endl;
delete p;
```

155

The new & delete operators for arrays

- Allocate a memory block of size `n*sizeof(T)` and invoke the default constructor `T{}` for each element
  `T*  arr = new T[n]`
- Run the destructor for each element and release the memory block
  `delete [] ptr`
- Rule of thumb:
  - *If you allocated by '[]', you should de-allocate by '[]'*

156

Sample usage of new [] & delete []

Array of objects

```
const int  N = 5;
Phone*  phones = new Phone[N];
for (int k=0; k<N; ++k) phones[k].setNumber(k+1);
//. . .
delete [] phones;
```

Array of pointers (to objects)

```
const int  N = 5;
Phone**  phones = new Phone*[N];
for (int k=0; k<N; ++k) phones[k] = new Phone(k+1);
//. . .
for (int k=0; k<N; ++k) delete phones[k];
delete [] phones;
```

157

The placement variant of operator new

- Don't allocate any memory, just invoke the constructor

```
T* ptr = new (address) T{arg,...}
T* arr = new (address) T[n]
```

- Used when you need to separate allocation from object initialization

```
SharedMemory  shm{10 * 1024};

void*  semStorage = shm.allocate( sizeof(Semaphore) );
Semaphore*  s = new (semStorage) Semaphore{1};
// ...
s->~Semaphore();  //invoke destructor explicitly
shm.dispose(semStorage);
```

158

std::operator new/delete

- The standard implementation is just a wrapper around malloc/free
  - Because new/delete are operators, they can be overridden for a class

```
void* operator new(size_t numBytes) {
  return malloc(numBytes);
}
void operator delete(void* ptr) {
  free(ptr);
}
```

```
Thing* t = new Thing{}; ... delete t;
```

```
Thing*  t = Thing::Thing( operator new(sizeof(Thing)) );
...
operator delete( Thing::~Thing(t) )
```

159

Overloading new/delete

- Possible to overload new/delete for a class (and its sub-classes)

```
class Person {
  string  name;
public:
  Person(const string& s) : name(s) {}

  void* operator new(size_t n) {
    void* blk = malloc(n);
    if (blk == NULL) throw runtime_error("heap alloc failed");
    return blk;
  }
  void  operator delete(void* ptr) { free(ptr); }
}
```

http://en.cppreference.com/w/cpp/memory/new/operator_new
http://en.cppreference.com/w/cpp/memory/new/operator_delete

160

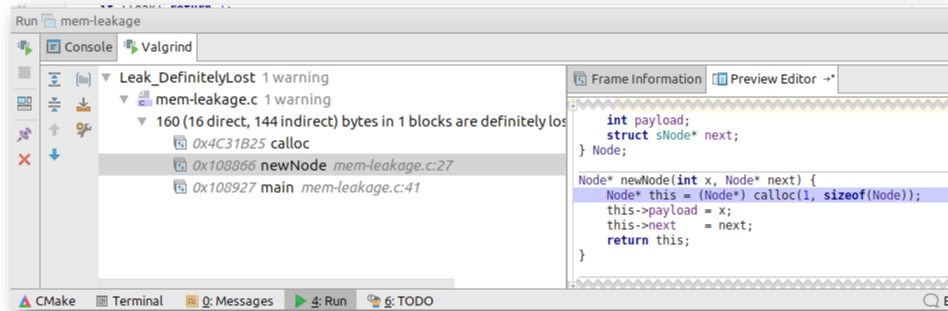
Checking for memory leakage

```
// --- main prog ---
int main(int numArgs, char* args[]) {
```

Run 'mem-leakage' Ctrl+Shift+F10
Debug 'mem-leakage'
Run 'mem-leakage' with Valgrind Memcheck

```
    while (N-- > 0) first = newNode(N + 1, first);
    for (Node* n = first; n != NULL; n = n->next)
        printf("%d ", n->payload);
    printf("\n");
```

```
$ sudo apt install valgrind
```



161

Chapter Summary

Usage of the operators new & delete

- Operator new
  - `new T{...}`
  - `new T[N]`
  - `new (addr) T{...}`
  - `new (addr) T[N]`
- Operator delete
  - `delete prt`
  - `delete [] ptr`
- Possible to override new/delete operators for a class and its sub-classes
  - `void*  T::operator new(size_t numBytes)`
  - `void   T::operator delete(void* ptr)`

162

EXERCISE

Dynamic accounts

- Define a simple account class and overload new/delete for it
  - Add print statements inside, so you know they are invoked

163

Intentional Blank

164

Templates

- ✓ Function templates
- ✓ Class templates
- ✓ Method templates
- ✓ Template parameters
- ✓ Template specialization
- ✓ Template type alias

165

In plain C, there's no function overloading

```
int abs(int x) {return x < 0 ? -x : x;}

void useit() {int result = abs(-42);}
```

```
int              abs(int x);
long             labs(long x);
long long        llabs(long long x);
float            fabsf(float x);
double           fabs(double x);
long double      fabsl(long double x);
```

Many functions with different names, all doing the same thing!

166

In C++, we do have function overloading

```
int       abs(int x)    {return x < 0 ? -x : x;}
long      abs(long x)   {return x < 0 ? -x : x;}
float     abs(float x)  {return x < 0 ? -x : x;}
double    abs(double x) {return x < 0 ? -x : x;}
//...etc...

void useit() {int result = abs(-42);}
```

Fixes the naming problem.
But, we may add up many unused functions!

167

In C++, let the compiler do the overloading job instead

```
template<typename T>
T abs(T x) {return x < 0 ? -x : x;}

void useit() {
  int         result  = abs<int>(-42);
  long double result2 = abs<long double>(-42);
}
```

A template is a code snippet, the compiler uses to generate code and substitute types for each type variable. The generated code, is what gets compiled and invoked.
Only as many type variants used, will generate functions.

168

How the compiler performs the job

```
template<typename T>
T abs(T x) {return x < 0 ? -x : x;}
```

```
abs<int>(-42)
```

```
int abs01(int x) {return ...;}
int result = abs01(-42);
```

```
abs<double>(-42)
```

```
double abs02(double x) {return ...;}
double result = abs02(-42);
```

```
abs<char>(-42)
```

```
char abs03(char x) {return ...;}
char result = abs03(-42);
```

169

Let the compiler figure out the type

```
template<typename T>
T abs(T x) {return x < 0 ? -x : x;}
```

```
int r1 = abs<int>(-42);
int r2 = abs<>(-42);
int r3 = abs(-42);
```

```
int abs01(int x) {...}
```

In most cases, the compiler performs the right "guess" of the type.
When that's not the case; help it!

170

When in doubt; let the compiler tell you about the type

```cpp
#include <cstdio>
#include <string>

template<typename T>
void printType(T) {puts(__PRETTY_FUNCTION__);}

int main(int, char**) {
    using namespace std::string_literals;
    printType(42);
    printType(3.1415926);
    printType('A');
    printType("Hello");
    printType("Hello"s);
    printType<std::string>("Hello");
    return 0;
}
```

```
print-type
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/c
void printType(T) [with T = int]
void printType(T) [with T = double]
void printType(T) [with T = char]
void printType(T) [with T = const char*]
void printType(T) [with T = std::__cxx11::basic_string<char>]
void printType(T) [with T = std::__cxx11::basic_string<char>]

Process finished with exit code 0
```

```
__PRETTY_FUNCTION__        //GCC CPP variable
```

171

You can create template specializations

```cpp
template<typename T>
T mulBy2(T n) {
    cout << __PRETTY_FUNCTION__ << ": ";
    return n * 2;
}

template<>
int mulBy2(int n) {
    cout << __PRETTY_FUNCTION__ << " SPZ: ";
    return n << 1;
}

int main(int, char**) {
    cout << "3.1415 = " << mulBy2(3.1415) << endl;
    cout << "21     = " << mulBy2(21) << endl;
    return 0;
}
```

Type specialization for int of mulBy2<T>

```
simple-math
/home/jens/Courses/cxx/cxx-embedded/src/explorati
3.1415 = T mulBy2(T) [with T = double]: 6.283
21     = T mulBy2(T) [with T = int] SPZ: 42

Process finished with exit code 0
```

172

Template parameters, can be of primitive types as well

```cpp
template<short N, long M, typename T>
T mul(T x) { return N * M * x; };

int main() {
    cout << "mul = " << mul<3, 7>(2) << endl;
    cout << "mul = " << mul<3, 7>(3.1415) << endl;
    return 0;
}
```

Non-type parameters can any of:

- *integral types*
- *pointers*

```
/home/jens/Courses/cxx/cxx-embedded/
mul = 42
mul = 65.9715

Process finished with exit code 0
```

173

Possible to use compile-time assertions

```cpp
template<int N, typename T>
T mul(T x) {
    static_assert(N >= 2);
    return x * N;
};

int main(int, char**) {
    cout << "mul = " << mul<2>(21) << endl;
    cout << "mul = " << mul<1>(42) << endl;
    return 0;
}
```

```cpp
int main(int, char**) {
    cout << "mul = " << mul<2>(21) << endl;
    //cout << "mul = " << mul<1>(42) << endl;
    return 0;
}
```

```
assertions
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/cmake-bui
mul = 42

Process finished with exit code 0
```

```
Messages Build
/home/jens/Tools/apps/CLion/ch-0/181.3007.15/bin/cmake/bin/cmake --build /home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/c
Scanning dependencies of target assertions
[ 50%] Building CXX object CMakeFiles/assertions.dir/assertions.cxx.o
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx: In instantiation of 'T mul(T) [with int N = 1; T = int]':
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx:13:34:   required from here
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx:7:5: error: static assertion failed
     static_assert(N >= 2);
     ^~~~~~~~~~~~~
compilation terminated due to -Wfatal-errors.
CMakeFiles/assertions.dir/build.make:62: recipe for target 'CMakeFiles/assertions.dir/assertions.cxx.o' failed
```

174

Can also check for type properties (type_traits)

```
#include <type_traits>

template <typename T>
T mulBy2(T x) {
    static_assert(is_integral<T>::value,
                  "Param type must be integral, i.e. short/int/long");
    return x << 1;
}

int main(int, char**) {
    cout << "mulBy2 = " << mulBy2(21) << endl;
    cout << "mulBy2 = " << mulBy2(21.0) << endl;
    return 0;
}
```

```
int main(int, char**) {
    cout << "mulBy2 = " << mulBy2(21) << endl;
    //cout << "mulBy2 = " << mulBy2(21.0) << endl;
    return 0;
}
assertions
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/cmake-build-debug/
mulBy2 = 42

Process finished with exit code 0
```

```
Messages Build
/home/jens/Tools/apps/CLion/ch-0/181.3007.15/bin/cmake/bin/cmake --build
 /home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/cmake-build-debug --target assertions -- -j 2
Scanning dependencies of target assertions
[ 50%] Building CXX object CMakeFiles/assertions.dir/assertions.cxx.o
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx: In instantiation of 'T mulBy2(T) [with
 T = double]':
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx:26:39:   required from here
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/assertions.cxx:19:5: error: static assertion failed:
 Param type must be integral, i.e. short/int/long
     static_assert(is_integral<T>::value,
     ^~~~~~~~~~~~~
compilation terminated due to -Wfatal-errors.
```

175

Type traits

- Specify type and method/operator requirements
  - Type predicates
  - Type selectors
  - Type mutators

`is_void` (C++11)	checks if a type is `void` (class template)
`is_integral` (C++11)	checks if a type is integral type (class template)
`is_floating_point` (C++11)	checks if a type is floating-point type (class template)
`is_array` (C++11)	checks if a type is an array type (class template)
`is_enum` (C++11)	checks if a type is an enumeration typ (class template)
`is_union` (C++11)	checks if a type is an union type (class template)
`is_class` (C++11)	checks if a type is a class type (but no (class template)

`remove_cv` (C++11) `remove_const` (C++11) `remove_volatile` (C++11)	removes `const` or/and (class template)
`add_cv` (C++11) `add_const` (C++11) `add_volatile` (C++11)	adds `const` or/and `vo` (class template)
References	
`remove_reference` (C++11)	removes reference fro (class template)
`add_lvalue_reference` (C++11) `add_rvalue_reference` (C++11)	adds *lvalue* or *rvalue* re (class template)
Pointers	
`remove_pointer` (C++11)	removes pointer fro (class template)
`add_pointer` (C++11)	adds pointer to the giv (class template)

`alignment_of` (C++11)	obtains the type's alignment requirements (class template)
`rank` (C++11)	obtains the number of dimensions of an array type (class template)
`extent` (C++11)	obtains the size of an array type along a specified di (class template)

More info: `http://en.cppreference.com/w/cpp/types`

176

Template Classes

177

In plain C, it's error prone to design generic data types

```
typedef struct sNode {
  void*          payload;
  struct sNode*  next;
} Node;
```

```
Node* node_new(void* payload, Node* next) {
  Node* this = (Node*)calloc(1, sizeof(Node));
  this->payload = payload;
  this->next    = next;
  return this;
}
```

```
Node* first = NULL;
first = node_new(account_new("1234-8888", 1500), first);
for (Node* n = first; n != NULL; n = n->next)
  printf("ACC: %s\n", account_toString((Account*)(n->payload)));
```

178

In C++, it's much easier to make it right

```
template<typename T>
class Node {
    T       payload;
    Node*   next;
public:
    Node(T payload, Node<T>* next = nullptr)
            : payload{payload}, next{next} {}
    T       data() const { return payload; }
    Node*   link() const { return next; }
};
```

```
simple-list
  /home/jens/Courses/cxx/cxx-embedded/src/
  acc: Account{4321-3333, 2100}
  acc: Account{1234-8888, 4200}

  Process finished with exit code 0
```

```
int main(int, char**) {
    Node<Account>* first = nullptr;
    first = new Node<Account>{Account{"1234-8888"s, 4200}, first};
    first = new Node<Account>{Account{"4321-3333"s, 2100}, first};
    for (Node<Account>* n = first; n != nullptr; n = n->link())
        cout << "acc: " << n->data() << endl;
    return 0;
}
```

```
struct Account {
    const string accno;
    const int balance;
    Account(const string& a, int b) : accno{a}, balance{b} {}
    friend ostream& operator<<(ostream& os, const Account& a){
        return os << "Account{" << a.accno << ", " << a.balance << "}";
    }
};
```

179

A template class can also have non-type parameters

```
template<typename T, unsigned N>
class Stack {
    T        stk[N];
    unsigned top = 0;
public:
    void     push(T x) { stk[top++] = x; }
    T        pop() { return stk[--top]; }
    bool     empty() const { return top == 0; }
    bool     full() const { return top >= N; }
};

int main(int, char**) {
    Stack<int, 10> s;

    for (auto k = 1; !s.full(); ++k) s.push(k);
    while (!s.empty()) cout << s.pop() << " ";
    cout << endl;

    return 0;
}
```

This data-type has a compile-time fixed size and spans only one single memory block.

```
simple-stack
  /home/jens/Courses/cxx/cxx-embedded
  10 9 8 7 6 5 4 3 2 1

  Process finished with exit code 0
```

180

Template parameters can have defaults

```
template<typename T=int, unsigned N=12>
class Stack {...};

int main(int, char**) {
    Stack<> s;

    for (auto k = 1; !s.full(); ++k) s.pu
    while (!s.empty()) cout << s.pop() <<
    cout << endl;

    return 0;
}
```

```
imple-stack
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templ
12 11 10 9 8 7 6 5 4 3 2 1

Process finished with exit code 0
```

Instantiation variants

```
Stack<float, 42>
Stack<float>
Stack<>
```

181

Type equivalence

- Two template instantiated types are equals, only if their template parameter arguments are equals

```
template<typename T>
struct Number {
    T value;
};

int main() {
    short a = 42; int b = 42;
    b = a;

    Number<short> n{42}; Number<int> m{42};
    m = n;
    return 0;
}
```

The compiler will not perform any implicit type conversion for template types, which it do perform for non-template types.

```
C++> g++ -std=c++17 -Wall -Wfatal-errors -c template-equals.cxx
template-equals.cxx: In function 'int main()':
template-equals.cxx:16:9: error: no match for 'operator=' (operan
d types are 'Number<int>' and 'Number<short int>')
     m = n;
         ^
compilation terminated due to -Wfatal-errors.
C++>
```

182

Template alias

- Classic "C-style" alias

```
typedef  queue<list<Person>>  Q;
Q  persons;
```

- C++11 alias
  - using *name* = *some-type;*
- C++11 template alias
  - template<typename T>
    using *name* = *template-expr<T>*

```
template<typename T>
using SymTab = std::map<std::string, T, std::greater<T>>;

class Expression {. . .};
SymTab<Expression>  symbols;
```

183

Template specializations

- Specialized implementation for selected types

```
template<>
class pair<short, short> {
  int storage;
 public:
  pair(short fst, short snd) : storage((fst << 16) | (snd)) {}
  short  getFirst()  {return storage >> 16;}
  short  getSecond() {return storage & ((1 << 16) - 1)}
};
```

```
pair<int,float>    p(17, 3.1415); //uses template class
pair<short,short>  q(42, 47);     //uses specialization class
```

Also, possible to do a partial specialization, where some template parameters are left for later instantiation.

```
template<typename First, typename Second>
class pair {
    First   first;
    Second  second;
  public:
    pair(First f, Second s) : first(f), second(s) {}
    First  getFirst()  {return first;}
    Second getSecond() {return second;}
};
```

184

Template methods

- A method can be a template function in its own
- The class itself might be a template or a non-template class

```
class Thing {
    //. . .
  public:
    //. . .
    template<typename T> void  push(T x) {. . .}
};
```

```
Thing  t;
t.push(17);                //push(int)
t.push("hello");           //push(char*)
t.push(new Thing{});       //push(Thing*)
```

Admittedly, this is nonsense. But it illustrates the idea.

185

Templates and class variables

- Each type instantiation requires a corresponding class variable instantiation

```
template<typename T>
class Foo {
   ...
  static  float   f;
  static  T       t;
};
```

```
Foo<int>   i;
```

```
float   Foo<int>::f = 0.0;
int     Foo<int>::t = 42;
```

```
Foo<char*>   c;
```

```
float   Foo<char*>::f = 0.0;
char*   Foo<char*>::t = "hello";
```

Recommendation: Don't!

186

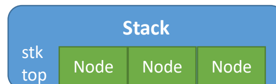
CHAPTER SUMMARY

Templates

- Templates is the most important language part of C++
- Ensure all functions of a template class are inline
- Type equality means all parameters must be the same
- Don't mix templates with class variables or friends

187

EXERCISE



Stack<T, N>

- Design a very simple template class realizing a stack (last-in-first-out)

```
template<typename T, unsigned N>
class Stack {
  T         stk[N];
  unsigned  top = 0;
 public:
  unsigned  size()
  unsigned  capacity()
  bool      empty()
  bool      full()
  void      push(T x)
  T         pop()
};
```

```
Stack<int, 10> stk;
stk.push(10); stk.push(20); stk.push(30);
while (!stk.empty()) cout << stk.pop() << endl;
```

```
Stack<string, 10> stk;
stk.push("hi"); stk.push("yo"); stk.push("howdy");
while (!stk.empty()) cout << stk.pop() << endl;
```

188

Operator Overloading

189

Why overload operators?

- Improves readability and understandability
  - Domain Specific Language (DSL)

```
Currency SEK{"SEK", 1.0};
Currency EUR{"EUR", 0.10783};
Currency GBP{"GBP", 0.0765501};
Money m1{100, SEK};
Money m2{10, EUR};
Money m3{20, GBP}
```

```
Money m;
assign(m, add(mul(m1,m2), mul(25,m3)));
```

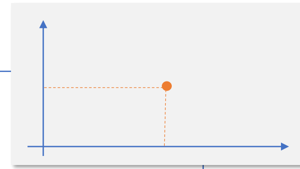
Hard to understand

```
Money m = m1*m2 + 25*m3;
```

Easy to understand

190

Sample class: Point



```
class Point {
  double  x, y;

 public:
  Point(double x=0.0, double y=0.0) : x(x),y(y) {}
  Point operator !() { return {x, -y}; }
  Point operator +=(Point z) { x += z.x; y += z.y; return *this; }
};

Point operator +(Point left, Point right) {
  return Point(left) += right;
}

ostream&  operator <<(ostream& os, const Point& p) {
  return os << "{" << p.x << ", " << p.y << "}";
}
```

```
Point z, a(1,2), b(3, 4);
z = a + !b;  //z.operator=( operator+(a, b.operator!()) );

cout << "z = " << z << endl;
//operator<<(operator<<(operator<<(cout, "z = "), z), endl);
```

191

Operator overloading syntax

- As a non-member
  - Unary operator: +a

    `T_return  operator ®(T_operand)`
  - Binary operator: a + b

    `T_return  operator ®(T_left, T_right)`
- As a member
  - Unary operator: *p

    `T_return T_operand::operator ®()`
  - Binary operator: p * q

    `T_return  T_left::operator ®(T_right)`

192

Declare as member or non-member?

- Destructive operators should be members
  - Non-destructive as non-members
- Binary operators as non-members
  - Might be a member
    - String operator +(const String&, char);
  - Might not be a member
    - String operator +(char, const String&);
- As a rule of thumb; put most operators outside the class

193

Overloadable operators

- Only C++ operators, (cannot invent new operators)

```
T operator@(T,T) //error
```

- At least one operand must be user defined

```
int operator+(int, int) //error
```

- These must be declared as members

```
=    []    ->    ()
```

- These operators cannot be overloaded

```
sizeof    ::    .    .*    ?:
```

194

Arity – number of parameters

- Cannot change the arity

```
String operator!(String&, String&); //error
```

- Some operators can be declared as both unary and binary, but they are completely different
  - Unary operator

    ```
    -x
    ```
  - Binary operator

    ```
    x - y
    ```

195

Operator precedence

- Cannot change the precedence order

```
a*b + c*d == (a*b) + (c*d)
```

- Beware of pitfalls
  - The expression az^n can be realized as

    ```
    a*z^n
    ```
  - However it is interpreted as

    ```
    (a*z)^n
    ```
  - Because the XOR (^) operator has lower priority than MULT (*)
  - The only solution is to use parenthesis

    ```
    a*(z^n)
    ```

196

Overloading of arithmetic operators

- Implement the core functionality once and then reuse

```
SimpleInt&  SimpleInt::operator +=(SimpleInt right) {
  value += right.value;
  return *this;
}

SimpleInt&  SimpleInt::operator -() {
  value *= -1;
  return *this;
}
```

```
SimpleInt  operator +(SimpleInt left, SimpleInt right) {
  return SimpleInt(left) += right;
}

SimpleInt  operator -(SimpleInt left, SimpleInt right) {
  return left + -right;
}
```

```
SimpleInt x{45}, y{3}, z;
z = x - y;
```

```
z.operator =(operator -(x, y));
```

```
z.operator =(operator +(x, y.operator -()));
```

```
z.operator =(SimpleInt(x).operator +=(y.operator -()));
```

197

Overloading of relational operators

- Implement the core functionality once and then reuse

```
bool  operator <(SimpleInt left, SimpleInt right) {
  return left.value < right.value;
}
bool  operator ==(SimpleInt left, SimpleInt right) {
  return left.value == right.value;
}

// --- reuse ---
bool  operator !=(SimpleInt left, SimpleInt right) {
    return !(left == right);
}
bool  operator <=(SimpleInt left, SimpleInt right) {
    return left < right || left == right;
}
bool  operator >(SimpleInt left, SimpleInt right) {
    return !(left <= right);
}
bool  operator >=(SimpleInt left, SimpleInt right) {
    return !(left < right);
}
```

198

No need to implement all relational operators

cppreference.com

Page Discussion

C++ / Utilities library

std::rel_ops::operator!=,>,<=,>=

Defined in header <utility>

```
template< class T >
bool operator!=( const T& lhs, const T& rhs );   (1)
template< class T >
bool operator>( const T& lhs, const T& rhs );    (2)
template< class T >
bool operator<=( const T& lhs, const T& rhs );   (3)
template< class T >
bool operator>=( const T& lhs, const T& rhs );   (4)
```

```
#include <iostream>
#include <utility>

struct Foo {
    int n;
};

bool operator==(const Foo& lhs, const Foo& rhs)
{
    return lhs.n == rhs.n;
}

bool operator<(const Foo& lhs, const Foo& rhs)
{
    return lhs.n < rhs.n;
}

int main()
{
    Foo f1 = {1};
    Foo f2 = {2};
    using namespace std::rel_ops;

    std::cout << std::boolalpha;
    std::cout << "not equal?     : " << (f1 != f2) << '\n';
    std::cout << "greater?       : " << (f1 > f2) << '\n';
    std::cout << "less equal?    : " << (f1 <= f2) << '\n';
    std::cout << "greater equal? : " << (f1 >= f2) << '\n';
}
```

Output:

```
not equal?     : true
greater?       : false
less equal?    : true
greater equal? : false
```

199

The increment/decrement operators

- Prefix and postfix operators are different
- Prefix operator is an unary operator

```
++t;  // T  operator ++(T&)
--t;  // T  operator --(T&)
```

- Postfix operator is a binary operator

```
t++;  // T  operator ++(T&, int)
t--;  // T  operator --(T&, int)
```

The compiler views it as

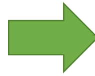
```
t ++ 0;
t -- 0;
```

200

The ++ operators

```
MyInt  MyInt::operator++() {
    this->value++;
    return *this;
}

MyInt  MyInt::operator++(int) {
    MyInt  tmp(this->value);
    ++(*this);
    return tmp;
}
```

```
MyInt  cnt(42);
++cnt;
cnt++; //→ cnt ++ 0;
```



```
//Interpretation:
cnt.operator++();
cnt.operator++(0);
```

201

Overloading of I/O operators

- Output operator

```
ostream&  operator <<(ostream& out, const T& t) {
  // write t to os: out << t
  return out;
}
```

- Input operator

```
istream&  operator >>(istream& is, T& t) {
  // read and parse value: is >> n
  // assign to t: t = T(n)
  return is;
}
```

202

Sample I/O operators

```
ostream&  operator <<(ostream& out, const Person& p) {
  out << "Person[name="<<p.name<<", age="<<p.age<<"]";
  return out;
}
```

```
istream&  operator >>(istream& in, Person& p) {
  Person tmp;
  in >> tmp.name >> tmp.age; //Expect: name SPACE age
  p = tmp;
  return in;
}
```

```
class Person {
  string  name = "Anna";
  int     age  = 42;
 public:
  //...
  friend ostream&  operator <<(ostream&, const Person&);
  friend istream&  operator >>(istream&, Person&);
};
```

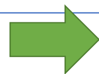
203

Index operator

- Index operator is a binary operator
  `ElemType&  T:: operator[](IndexType)`
- The index can be of any type, such as a text string

```
template<typename T>
class Vector {
    T*  arr; int size;
    void check(int i) {…}
  public:
    Vector(int sz) : arr(new T[sz]), size(sz) {}
    T&  operator [](int idx) {check(idx); return arr[idx];}
    //…
};
```

```
Vector<int>  v(10);
v[5] = 10;
```



```
//Interpretation:
v.operator[](5) = 10;
```

204

Arrow operator

- Unary member operator

```
X* T:: operator ->()
```

- Returns the address of something

```
template<typename T>
class SmartPtr {
    T*  ptr;
  public:
    SmartPtr(T* p) : ptr(p) {}
    T*  operator ->() {return ptr;}
};
```

```
SmartPtr<Person>  p( new Person("Nisse") );
char* s = p->getName();
```

```
//Interpretation:
char* s = ( p.operator->() )->getName();
```

205

Function call operator

- Member operator with variable number of parameters

```
S  T:: operator()(X, Y, Z, ...)
```

```
class LinearTransformation {
    double  a, b;
  public:
    LinearTransformation(double _a=1, double b=0)
      : a(_a), b(_b) {}
    double  operator()(double x) {
      return a * x + b;
    }
};
```

```
LinearTransformation   h(2.5, 10.0);
double  samples[N]; //...fill it...
for (int k=0; k<N; ++k) {
  samples[k] = h(samples[k]);
}
```

```
//Interpretation:
samples[k] = h.operator()( samples[k] );
```

206

CHAPTER SUMMARY

Understanding Operator Overloading

- Unary operator
  - R operator@(T)
  - R T::operator@()
- Binary operator
  - R operator@(Tlhs, Trhs)
  - R Tlhs::operator@(Trhs)
- Must be a member
  - operator=(T)
  - operator[](T)
  - operator->()
  - operator()(...)

207

EXERCISE

3D vector

- Design a copiable data-type for a 3D vector <x, y, z>
- Add a decent set of operators, one at a time, such as
  - Printing using <<
  - Reading using >>
  - Multiplication by a scalar value
  - Addition/subtraction of two vectors
  - Scalar/inner product of two vectors
- EXTRA; If you have time
  - Template based vector, so one can choose representation of the coordinate values
  - Verify that the type parameter is numeric

208

PART-3
C++ Library

209

Text Strings

- ✓ Why using std::string instead of nativ strings
- ✓ Overview of std::string and its functions/operators
- ✓ What is SSO (Short String Optimization) and how it is used
- ✓ Working with legacy strings
- ✓ Character oriented functions
- ✓ Multi-line string literals
- ✓ Using string_view

210

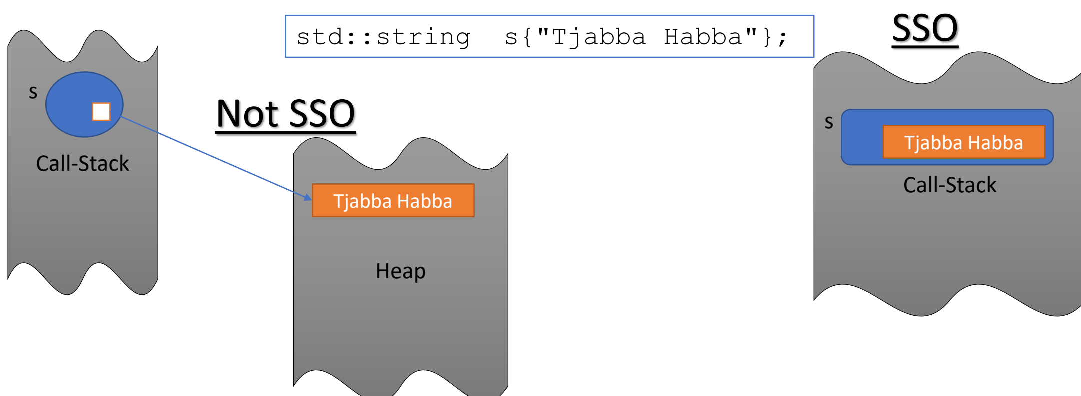
Reduce bugs

- More than 80% of bugs in C/C++ programs are related to problems with pointers such as native text strings.
  Hence, by just using std::string the code quality increases
- When using std::string there is no need to:
  - Struggle with dynamic memory allocations, i.e. new / delete or malloc() / free()
  - Design string manipulation functions
  - Be careful with text arguments in or out of functions
  - Define your own string-class
- Easy to use
  - Create a string object, modify it and pass it around between functions just as you would do with a primitive value (int, float, ...)
  - std::string should be your 1st hand choice when dealing with text

211

Short String Optimization (SSO)

- Very performant → don't bother to compete
- Short strings are embedded in the string object (stack-allocated), instead of heap-allocated



212

C++ Text String == class std::string

- Standard definition of text string objects in C++
  - http://en.cppreference.com/w/cpp/string/basic_string

Several typedefs for common character types are provided:

Defined in header <string>

Type	Definition
std::string	std::basic_string<char>
std::wstring	std::basic_string<wchar_t>
std::u16string (C++11)	std::basic_string<char16_t>
std::u32string (C++11)	std::basic_string<char32_t>
std::pmr::string (C++17)	std::pmr::basic_string<char>
std::pmr::wstring (C++17)	std::pmr::basic_string<wchar_t>
std::pmr::u16string (C++17)	std::pmr::basic_string<char16_t>
std::pmr::u32string (C++17)	std::pmr::basic_string<char32_t>

213

std::string_literals

- Modern C++ has support for user-defined literal suffixes
- C++14 added a literal suffix ("..."s) to std::string literals

```
auto msg = "I'm a const char* text string";
```

```
using namespace std::string_literals;
auto msg = "I'm a std::string text string"s;
```

```
#include <string>
#include <iostream>

int main()
{
    using namespace std::string_literals;

    std::string s1 = "abc\0\0def";
    std::string s2 = "abc\0\0def"s;
    std::cout << "s1: " << s1.size() << " \"" << s1 << "\"\n";
    std::cout << "s2: " << s2.size() << " \"" << s2 << "\"\n";
}
```

Possible output:

```
s1: 3 "abc"
s2: 8 "abc^@^@def"
```

A std::string object can contain null bytes ('\0'), which are printed as '^@'

214

Raw strings

- Useful for
  - Multi-line strings
  - Strings containing '\' and '"'
- Syntax
  - R"(. . .)"

```
char* ugly = "\"\\w+\\\\d+\"";
char* nice = R"("\w+\\d+")";
```

raw-strings.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  string html = R"(
6  <html>
7  <head></head>
8  <body>
9      <h1>Message of the Day</h1>
10     <p>
11         C++11 is indeed cool
12     </p>
13 </body>
14 </html>
15 )";
16
17 int main() {
18     cout << html << endl;
19     return 0;
20 }
```

jens@vbox4: ~/Documents/c++11

```
jens@vbox4:~/Documents/c++11$ g++  -std=c++0x -Wall raw-strings.cpp -o raw-strings
jens@vbox4:~/Documents/c++11$ ./raw-strings

<html>
<head></head>
<body>
    <h1>Message of the Day</h1>
    <p>
        C++11 is indeed cool
    </p>
</body>
</html>

jens@vbox4:~/Documents/c++11$
```

215

String API

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)
empty	Test if string is empty (public member function)

Element access:

operator[]	Get character in string (public member function)
at	Get character in string (public member function)

The header also declares some functions that extend the functionality

getline	Get line from stream (function)
operator<<	Insert string into stream (function)
operator>>	Extract string from istream (function)

operator
operator==
operator!=
operator<
operator>
operator<=
operator>=

Modifiers:

operator+=	Append to string (public member function)
append	Append to string (public member function)
push_back	Append character to string (public member function)
assign	Assign content to string (public member function)
insert	Insert into string (public member function)
erase	Erase characters from string (public member function)
replace	Replace part of string (public member function)
copy	Copy sequence of characters from string (public mem
swap	Swap contents with another string (public member f

String operations:

c_str	Get C string equivalent (public member function)
data	Get string data (public member function)
get_allocator	Get allocator (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public mem
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member
find_first_not_of	Find absence of character in string
find_last_not_of	Find absence of character in string from the end (pu
substr	Generate substring (public member function)
compare	Compare strings (public member function)

216

Type conversion functions

- From primitive value to string
  - std::string to_string(* value)
- From string to primitive value
  - int stoi(string s)
  - long stol(string s)
  - float stof(string s)
  - double stod(string s)

```
#include <iostream>
#include <string>
using namespace std;

int main(int nArgs, char* args[]) {
  int  N = (nArgs == 1 ? 10 : stoi(args[1]));
  string txt = "[0";
  for (int k=1; k<=N; ++k) {
    txt += "," + to_string(k);
  }
  txt   += "]";
  txt[1] = '#';

  cout << "txt = " << txt << endl;
  return 0;
}
```

```
C++> ./type-conversions 5
txt = [#,1,2,3,4,5]
```

217

String search

- Find start-position of substring
  - find(str, startPos=0)   //from start
  - rfind(str, startPos=0)   //from end
- Find index of character-set
  - find_first_of(charset, startPos=0)
  - find_first_not_of(charset, startPos=0)
  - find_last_of(charset, startPos=0)
  - find_last_not_of(charset, startPos=0)

```
void p(string& s, string::size_type ix) {
  if (ix == string::npos)
    cout<<"not found\n"
  else
    cout<<"found:\""<<s.substr(ix)<<"\"\n";
}

int main() {
  string  s = "this is a string";
  p(s, s.find("is"));
  p(s, s.find("is", 4));
  p(s, s.find("A"));
}
```

```
C++> ./text-search
found:"is is a string"
found:"is a string"
not found
```

218

Character oriented functions

- Include file
  - #include <cctype>
- Functions

```
int isalnum(int c) //true, if alpha-numerical
int isalpha(int c) //true, if letter
int isdigit(int c) //true, if digit

int islower(int c) //true, if lower-case
int isupper(int c) //true, if upper-case
int tolower(int c) //convert into lower-case
int toupper(int c) //convert into upper-case

int isspace(int c) //true, if white space
int isprint(int c) //true, if printable
int iscntrl(int c) //true, if CTRL char
```

219

Working with legacy C/C++ code

```
int main(int n, char* args[]) {
  int value = 10;
  for (int k=1; k<n; ++k) {
    string arg = args[k];
    if (arg == "-n") value = stoi(args[++k]);
  }
}
```

```
extern char* getStringFromC();
extern void  putStringToC(char*);

void doit() {
  string s = getStringFromC();
  string r;
  for (auto ch : s) { r += toupper(ch);}
  putStringToC(r.c_str());
}
```

Convert from C → C++

Convert from C++ → C

220

Line oriented input

- Read a sequence of characters until a line-terminator
  - `string getline(inStream, outString, delimChar = '\n')`

```
int count() {
  int numChars = 0;
  for (string line; getline(cin, line); ) {
    numChars += line.size();
  }
  return numChars;
}
```

221

Word oriented input

- Input to a string
  - string w; cin >> w; //reads the next sequence of non-blank chars

```
int count() {
  int numWords = 0;
  for(string word; cin >> word; ) {
    ++numWords;
  }
  return numWords;
}
```

222

CHAPTER SUMMARY

Text Strings

EXERCISE

Search

- Write a program that
- Takes a search phrase on the command-line
- Reads its input from cin
- Prints out all lines (numbered) containing the phrase
- Example

```
./search iostream < search.cpp
```

```
if (line.find(phrase) != string::npos) . . .
```

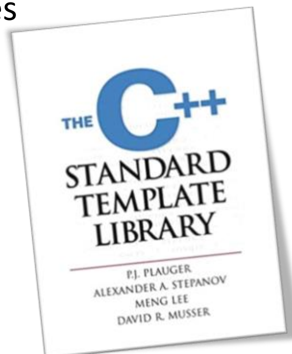
Container Types

- ✓ Sequence containers
- ✓ vector<T>
- ✓ deque<T>
- ✓ array<T,N>
- ✓ list<T>
- ✓ forward_list<T>
- ✓ Associative containers
- ✓ set<T> / unordered_set<T>
- ✓ map<K,V> / unordered_map<K,V>
- ✓ Multi key versions
- ✓ Adapters
- ✓ Allocators

225

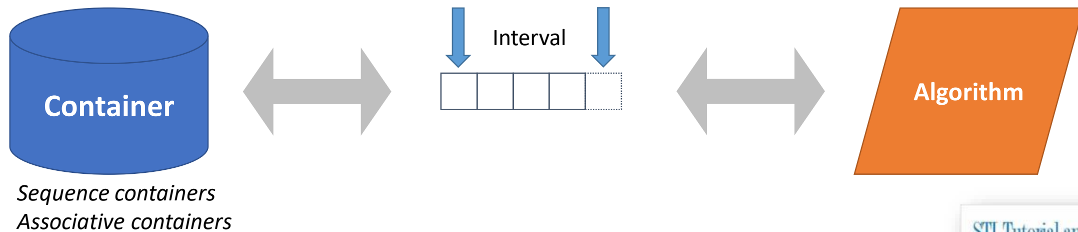
Alexander Stepanov

- The primary designer and implementer of the C++ Standard Template Library,[1] which he started to develop around 1992 while employed at HP Labs.
- Tried to convince Bjarne Stroustrup to introduce something like Ada generics in C++, this later became what we now know as templates

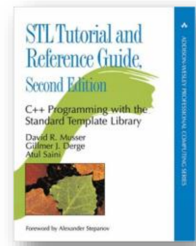


226

The STL architecture



Sequence containers
Associative containers

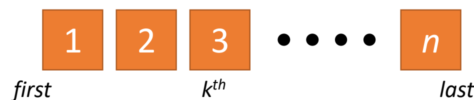


STL = Standard Template Library
https://en.wikipedia.org/wiki/Standard_Template_Library

227

Sequence containers

- Linear sequence of elements



228

Typical methods of sequence containers

- Constructors
  - () – empty container
  - (count) – insert count T() values
  - (count, value) – insert count values
  - (first, last) – take elements from range
  - ({e1,e2,...}) – take elements from args
- Adding elements
  - push_front(value) / push_back(value)
- Predicates
  - size_type size() / bool empty()
- Removing elements
  - T front() / pop_front()
  - T back() / pop_back()
  - clear()
- Inserting elements
  - insert(pos, value)
  - insert(pos, count, value)
  - insert(pos, first, last)
  - insert(pos, {value1, value2, ...})

N.B. Some containers might not support all methods

229

vector<T>

- Include file
  - #include <vector>
- Insertion and extraction
  - push_back(V) / pop_back()
  - insert(Iterator, Value) / insert(Iterator, N, Value)
- Selectors
  - front() / back()
- Iterators
  - begin() / end()

```
vector<string>  v = {
  "hi", "hey", "hello"
};
v.push_back("howdy");
v[0] = "Hi";
for (auto& s : v) cout << s << endl;
```

Do not insert elements at the front, because it has to shift all elements down one step.

230

deque<T>

- Similar as vector<T>
  - Means ”double-ended queue”
- Include file
  - #include <deque>
- Additional operations
  - push_front() / pop_front()

```
deque<int>  d;
d.push_front(10);
d.push_front(20);
d.push_front(30);
while (!d.empty()) {
  int value = d.back();
  d.pop_back();
  cout << value << endl;
}
```

231

array<T, N>

- Type safe alternative to native arrays
- Include file: <array>
- Declaration: array<Type, Size>

```
array<int, 5> a = {1,2,3,4,5};
for (auto& i : a) cout << i << endl;
a[4] = 50;
a[5] = 60; //ERROR
```

array.cpp

```
1 #include <iostream>
2 #include <array>
3 using namespace std;
4
5 int main() {
6     array<int,6> nums = {1,2,3};
7     cout << "nums.size=" << nums.size() << endl;
8     for (auto n : nums) cout << n << " "; cout<<endl;
9     nums[5] = 100;
10    for (auto n : nums) cout << n << " "; cout<<endl;
11
12    nums[10] = 1234;
13    cout << "nums.size=" << nums.size() << endl;
14
15    //int* arr = nums; //error: cannot convert ‘std::array<int, 6u>’ to ‘int*’ in initialization
16
17    nums.at(10) = 1234;
18    cerr << "We will not see this message" << endl;
19    return 0;
20 }
```

jens@vbox4: ~/Documents/c++11

```
jens@vbox4:~/Documents/c++11$ g++ -std=c++0x  array.cpp -o array
jens@vbox4:~/Documents/c++11$ ./array
nums.size=6
1 2 3 0 0 0
1 2 3 0 0 100
nums.size=6
terminate called after throwing an instance of 'std::out_of_range'
  what():  array::at
Aborted
jens@vbox4:~/Documents/c++11$
```

232

list<T>

- Double linked sequential data structure
  - Similar operations as deque
- Include file
  - #include <list>

```
list<int>  l;
l.push_front(10);
l.push_front(20);
l.push_front(30);
while (!l.empty()) {
  int value = l.back();
  l.pop_back();
  cout << value << endl;
}
```

233

forward_list<T>

- Single linked sequential data structure
- Include file
  - #include <forward_list>
- Absent methods
  - size()
  - back(), push_back(), pop_back()

```
foward_list<int>  fl;
d.push_front(10);
d.push_front(20);
d.push_front(30);
while (!d.empty()) {
  int value = d.front();
  d.pop_front();
  cout << value << endl;
}
```

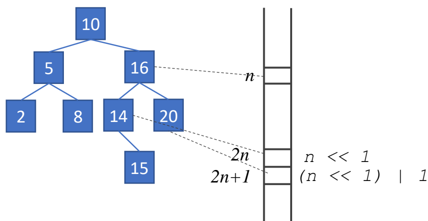
234

Associative containers

- Containers that provides access using a key
- Set
  - Collection of key elements
- Map
  - Collection of <key, value> pairs
- Variants
  - Inserted key can be unique or allow multiple occurences
    - set vs. multiset
  - Implementation can be via binary tree or hash table
    - set vs. unordered_set

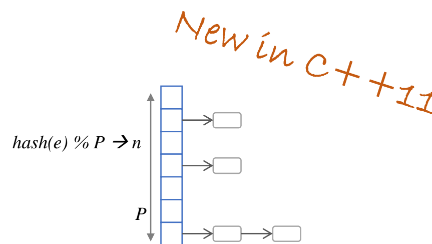
235

Binary tree vs. Hash table



Implicit balanced binary tree
Access time = O(log n)
Keeps the element in sorted order

```
set<T>
multiset<T>

map<K,V>
multimap<K,V>
```



Hash table
Access time ≈ O(1)
Elements are in "random" order

```
unordered_set<T>
unordered_multiset<T>

unordered_map<K,V>
unordered_multimap<K,V>
```

236

Typical methods of associative containers

- Constructors
  - () - empty
  - (begin,end) - populate from range
  - ({e1,e2,...}) - populate from static list
- Access
  - begin() / end()
  - operator[]
- Lookup
  - count() / find()
- Inserting elements
  - insert()
- Removing elements
  - erase() / clear()
- Capacity
  - size() / empty()

237

set<T, Comparator = less<T>>

- Keeps the elements in an implicit balanced binary tree
  - Similar to java.util.TreeSet in Java
- Include file
  - #include <set>
- Basic operations
  - insert(value)
  - insert(begin, end)
  - insert(init-list)
  - count(value) //if > 0 → value is a member
  - find(value) //returns iterator to value, or end()

238

unordered_set<T>

- A hash-table of elements
  - Similar to java.util.HashSet in Java
- Can be provided with
  - a hash function and
  - an equality predicate
- Include file
  - #include <unordered_set>
- Basic operations
  - Same as for set

compare-sets.cpp | using-set.cpp

```
1 #include <iostream>
2 #include <set>
3 #include <unordered_set>
4 using namespace std;
5
6 int main() {
7     set<int>  s = {10,7,3,5,12,7,1,8,10,3,3,3};
8     cout << "set: ";
9     for (auto& i : s) cout << i << " ";
10    cout << endl;
11
12    unordered_set<int>  ms = {10,7,3,5,12,7,1,8,10,3,3,3};
13    cout << "unordered_set: ";
14    for (auto& i : ms) cout << i << " ";
15    cout << endl;
16 }
```

```
[jens@vbox3 std-containers]$ g++ --std=c++11 compare-sets.cpp -o compare-sets
[jens@vbox3 std-containers]$ ./compare-sets
set: 1 3 5 7 8 10 12
unordered_set: 8 1 12 5 3 7 10
[jens@vbox3 std-containers]$
```

239

Comparing set<T> with multiset<T>

using-set.cpp

```
1 #include <iostream>
2 #include <set>
3 using namespace std;
4
5 int main() {
6     set<int>  s = {10,7,3,5,12,7,1,8,10,3,3,3};
7     cout << "set: ";
8     for (auto& i : s) cout << i << " ";
9     cout << endl;
10
11    multiset<int>  ms = {10,7,3,5,12,7,1,8,10,3,3,3};
12    cout << "multiset: ";
13    for (auto& i : ms) cout << i << " ";
14    cout << endl;
15 }
16
```

```
[jens@vbox3 std-containers]$ g++ --std=c++11 using-set.cpp -o using-set
[jens@vbox3 std-containers]$ ./using-set
set: 1 3 5 7 8 10 12
multiset: 1 3 3 3 3 5 7 7 8 10 10 12
[jens@vbox3 std-containers]$
```

240

Case insensitive unordered_set

```cpp
#define PAYLOAD "Hello", "HELLO", "hello", "HeLLo"

int main() {
    {
        cout << "case : ";
        std::unordered_set<std::string> words{PAYLOAD};
        for (auto& w : words) std::cout << w << " ";
        cout<< std::endl;
    }
    {
        cout << "icase: ";
        std::unordered_set<std::string, icase_hash, icase_hash> words{PAYLOAD};
        for (auto& w : words) std::cout << w << " ";
        cout<< std::endl;
    }
    return 0;
}
```

```
/mnt/c/Users/jensr/Dropbox/Ribomati
case : hello HeLLo HELLO Hello
icase: Hello

Process finished with exit code 0
```

241

Custom hasher

- Provide two functions
  - size_t hash(Type item)
  - bool equals(Type lhs, Type rhs)
- The hash() function must return the same numeric value for lhs and rhs when they are equals

```cpp
struct icase_hash {
    size_t operator()(const string& item) const {
        return std::hash<string>()(lc(item));
    }

    bool operator()(const string& lhs, const string& rhs) const {
        return lc(lhs) == lc(rhs);
    }

private:
    string lc(string s) const {
        transform(s.begin(), s.end(), s.begin(), [](auto ch) {
            return ::tolower(ch);
        });
        return s;
    }
};
```

242

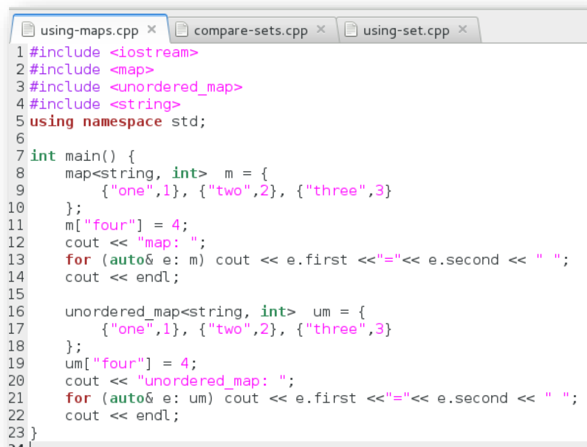
map<Key, Value, Comparator = less<Key>>

- A map is a set of pair<> elements
  - Similar to java.util.TreeMap in Java
- Include file
  - #include <map>
- Basic operations
  - Insert, count, find //similar as for set
  - operator[](key) //creates the slot if the key was absent

243

unordered_map<K,V>

- A hash-table of elements
  - Similar to java.util.HashMap
- Include file
  - #include <unordered_map>
- Basic operations
  - Same as for map




```
[jens@vbox3 std-containers]$ g++ --std=c++11 using-maps.cpp -o using-maps
[jens@vbox3 std-containers]$ ./using-maps
map: four=4 one=1 three=3 two=2
unordered_map: four=4 three=3 one=1 two=2
[jens@vbox3 std-containers]$
```

244

Container adapters

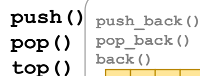
- Provides a cleaner and reduced interface for an existing container

245

stack<T, Container = deque<T>>

- LIFO container adapter, i.e. wraps another container
- Include file
  - #include <stack>
- Basic operations
  - void push(const T&)
  - T& top()
  - void pop()
  - bool empty()
  - size_type size()

```
stack<int>  s;
s.push(10);
s.push(20);
s.push(30);
while (!s.empty()) {
  int value = s.top();
  s.pop();
  cout << value << endl;
}
```

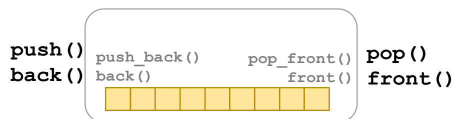


246

queue<T, Container = deque<T>>

- FIFO container adapter, i.e. wraps another container
  - Requirs push_back() and pop_front()
    - list<T>
    - deque<T>
- Include file
  - #include <queue>
- Basic operations
  - void push(const T&)
  - T& front()
  - void pop()
  - T& back()
  - bool empty()
  - size_type size()

```cpp
queue<int>  q;
q.push(10);
q.push(20);
q.push(30);
while (!q.empty()) {
  int value = q.front();
  q.pop();
  cout << value << endl;
}
```



247

priority_queue<T,
Container = vector<T>, Comparator = less<T>>

- A queue that keeps the elements sorted according to the provided comparator
- Include file
  - #include <queue>
- Basic operations
  - void push(const T&)
  - T& top()
  - void pop()

```cpp
int main(int numArgs, char* args[]) {
    priority_queue<Person> pers;

    pers.push(Person("Zeke"));
    pers.push(Person("Berit"));
    pers.push(Person("Carin"));
    pers.push(Person("Anna"));
    pers.push(Person("Bosse"));

    while (!pers.empty()) {
        cout << pers.top().toString() << endl;
        pers.pop();
    }
    return 0;
}
```

```
priority-queue
"D:\CloudStorage\Dro
Person{name=Anna}
Person{name=Berit}
Person{name=Bosse}
Person{name=Carin}
Person{name=Zeke}

Process finished wit
```

```cpp
class Person {
    string name;
public:
    Person(const string& name) : name(name) {}
    string getName() const { return name; }
    string toString() const {
        return "Person{name=" + name + "}";
    }
};
bool operator<(const Person& left, const Person& right) {
    return left.getName() > right.getName();
}
```

248

C++ Allocator

- Collection types, like std::vector<T> and std::list<T> uses a pluggable allocator for its internal elements
- Possible to implement your own allocator
  - https://www.cppmemorymastery.com/2017/03/15/how-to-use-standard-library-containers-with-custom-allocators/

```
template<typename T>
class MyAllocator {
  T*    allocate(size_t numObjects);
  void  deallocate(T* ptr, size_t numObjects) noexcept;
  //plus constructors and equality operators
};
```

249

CHAPTER SUMMARY

Container Types

- It's important you master all C++ containers
- Use vector<V> for simple sequence operations
- Use map<K,V> for simple table lookup operations
- Understand the run-time properties of
  - vector / array / deque / list / forward_list
  - set / unordered_set
  - map / unordered_map

250

EXERCISE

Word count

- Use an unordered_multiset<std::string> to keep track of the word count
- Read a bunch of words from stdin and put them into the set
- Then pour them over to a muliset<std::string>
- Finally, print out each word and its frequency, now in word sorted order

```
for(auto it = words.begin(); it != words.end(); )
{
    auto cnt = words.count(*it);
    std::cout << *it << ":\t" << cnt << '\n';
    std::advance(it, cnt); // all cnt elements have equivalent keys
}
```

251

Intentional Blank

252

Iterators and Intervals

- ✓ What is an iterator
- ✓ What is a range and how is it used
- ✓ Iterator categories
- ✓ I/O iterators
- ✓ Iterator adapter functions
- ✓ Implementing an interator

253

What is an iterator

- An iterator object maintains a position into a group of elements
  - The position can be changed and the referred element can be retrieved or updated
- A C++ iterator is modelled to mimic usage of pointers
- Modelled as a pointer abstraction

```
for(int*                    p = &arr[0]     ; p != &arr[N]   ; ++p)   *p = 0;
```

```
for(list<int>::iterator p = lst.begin(); p != lst.end(); ++p)   *p = 0;
```

```
for(auto p = lst.begin(); p != lst.end(); ++p)   *p = 0;
```

```
for(auto& p : lst)   p = 0;
```

254

What is an interval

- An interval if defined by two iterators pointing into the same group of elements and such as when the position of first iterator is monotonically incremented it would eventually refer to the position of the second iterator

```
vector<int> dst = {1,2,3,4,5};
for_each(dst.begin(), dst.end(), [](auto n){ cout << n << endl; });
```

```
void for_each(Iter first, Iter last, function<void(int)> f) {
    for (; first != last; ++first) f( *first );
}
```

255

Iterators ≈ pointers

- In most cases, iterators can be used instead of pointers

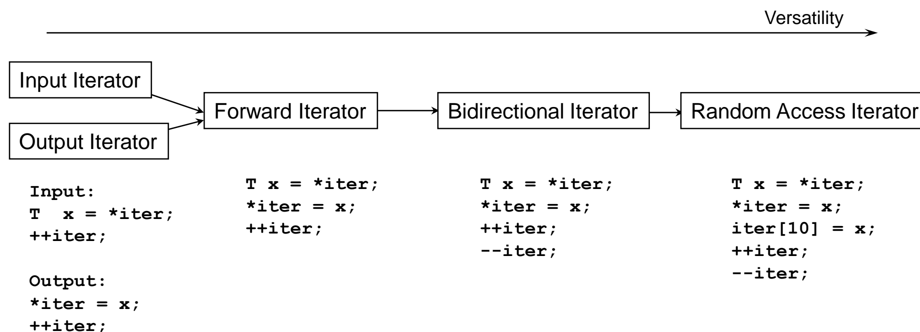
```
int arr[] = {1,2,3,4,5};
copy(&arr[0], &arr[5], destination);
```

```
int arr[] = {1,2,3,4,5};
copy(arr, arr+5, destination);
```

```
vector<int> vec = {1,2,3,4,5};
copy(vec.begin(), vec.end(), destination);
```

256

There are several iterator categories



http://en.cppreference.com/w/cpp/iterator

257

(In/Out)put Iterators

```
template<typename Iter, typename T>
Iter  find(Iter first, Iter last, const T& value) {
  for (; first != last && *first != value; ++first) ;
  return first;
}
```

```
template<typename In, typename Out>
void  copy(In first, In last, Out dst) {
  for (; first != last; ++first, ++dst)
    *dst = *first;
}
```

258

Forward and Bidirectional Iterators

```
template<typename Iter, typename T>
void  replace(Iter first, Iter last, const T& from, const T& to) {
    for (; first != last; ++first)
        if (*first == from) *first = to;
}
```

```
template<typename Iter>
void  reverse(Iter first, Iter last) {
    for (; first != last && first != --last; )
        std::iter_swap(first++, last);
}
```

```
template<typename Fwd1, typename Fwd2>
void  iter_swap(Fwd1 a, Fwd2 b) { std::swap(*a, *b); }
```

http://en.cppreference.com/w/cpp/algorithm/swap

259

Random Access Iterator

```
template<typename Iter>
void  shuffle(Iter first, Iter last) {
    auto n = last - first;
    for (auto k = n-1; k > 0; --k)
        std::swap(first[k], first[ std::rand() % (k+1) ]);
}
```

http://en.cppreference.com/w/cpp/numeric/random/rand

260

Implementing an iterator

- Minimum set of operators
  - ✓iterator& operator ++() *move to next*
  - ✓bool operator !=(const iterator& that) *comparison*
  - ✓T operator *() *fetch data*

```
class Sequence {
  int last = 10;
 public:
  Sequence(int l) : last(l) {}
  struct iterator {…};
  iterator begin() {return {1};}
  iterator end() {return {last};}
};
```

```
struct iterator {
  int current;
  iterator(int c) : current(c) {}
  bool operator !=(const iterator& that) {
    return this->current != that.current;
  }
  iterator&  operator ++() {
    ++current; return *this;
  }
  int  operator *() {return current;}
};
```

```
Sequence s(100);
for (auto it = s.begin(); it != s.end(); ++it) { cout << *it; }
```

261

I/O-iterator

- Input iterator reading from a stream
  - istream_iterator<T> iter{input_stream}
  - istream_iterator<T> eof{}
- Output iterator writing to a stream
  - ostream_iterator<T> iter{output_stream, separator}

```
int main(int numArgs, char* args[]) {
    set<string>                words;
    istream_iterator<string> in{cin};
    istream_iterator<string> eof;
    copy(in, eof, inserter(words, words.end()));

    ostream_iterator<string> out{cout, "\n"};
    copy(words.begin(), words.end(), out);
    cout << endl;
    return 0;
}
```

```
C++> ./stream-iterators.exe < ../stream-iterators.cxx | head -10
"\n"};
#include
0;
<<
<algorithm>
<iostream>
<iterator>
<set>
<string>
args[])
C++>
C++>
```

262

Back inserter function

- Adapter function that provides an iterator, which invokes push_back() on the target container

```
vector<int>   numbers;
fill_n(back_inserter(numbers), 5, 42);

for (auto n : numbers) cout << n << ' ';
//prints: 42 42 42 42 42
```

263

Iterator adapters

- Provides an insertion side-effect

Iterator adaptors

reverse_iterator	iterator adaptor for reverse-order traversal (class template)
make_reverse_iterator (C++14)	creates a std::reverse_iterator of type inferred from the argument (function template)
move_iterator (C++11)	iterator adaptor which dereferences to an rvalue reference (class template)
make_move_iterator (C++11)	creates a std::move_iterator of type inferred from the argument (function template)
back_insert_iterator	iterator adaptor for insertion at the end of a container (class template)
back_inserter	creates a std::back_insert_iterator of type inferred from the argument (function template)
front_insert_iterator	iterator adaptor for insertion at the front of a container (class template)
front_inserter	creates a std::front_insert_iterator of type inferred from the argument (function template)
insert_iterator	iterator adaptor for insertion into a container (class template)
inserter	creates a std::insert_iterator of type inferred from the argument (function template)

264

End-point adapters

- Instead of using the begin/end methods of a collection and pointer access, use the begin/end adapters instead

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(int argc, char* argv[]) {
    int  arr[] = {1, 3, 6, 8, 11, 25, 42};
    auto pos   = find_if(begin(arr), end(arr), [](auto n) { return n % 5 == 0; });

    if (pos != end(arr))
        cout << "found number divisable with 5: " << *pos << endl;

    return 0;
}
```

```
algo
/home/jens/docs/courses/adv-c++/explore/cmake-build-debug/algo
found number divisable with 5: 25

Process finished with exit code 0
```

265

CHAPTER SUMMARY

Iterators and Intervals

- An interval is defined by a pair of iterators
  - The first pointing to the current position and
    the second to the position just outside the range
- An iterator contains a notion of a current position

266

Exercise

Squared number streams

- Use stream iterators to
  1. Read a sequence of integral numbers, using an istream iterator
  2. Compute the square of each number
  3. Print them out, using an ostream iterator

267

Intentional Blank

268

Algorithms

- ✓ What is a std::* algorithm
- ✓ The STL architecture
- ✓ How to pass the business logic into algorithm functions
- ✓ Understanding *_n and *_if suffixes of functions
- ✓ Using functions with predicates
- ✓ Using back_insereter iterators
- ✓ Transforming and aggregating
- ✓ Sorting
- ✓ Populating data into containers

269

C++ algorithms

- Include
  - <algorithm>, <numeric>
- Operates on intervals
  - Almost independent of container type
- Sequence
  - for_each, find, fill, count, copy, transform, replace, generate, remove, …
- Sorting
  - sort, stable_sort, binary_search, max_element, …
- Generalized numerical
  - accumulate, inner_product, …
- Predicate
  - all_of, any_of, none_of
- URL
  - http://en.cppreference.com/w/cpp/algorithm

270

Algorithms

- C++ algorithms can take iterators or native pointers

```
int  ints[] = {1,2,2,3,3,3,4,4,4,4};
int cnt3 = count(ints, ints+10, 3);

vector<int>  vec(ints, ints+10);
int cnt4 = count(vec.begin(), vec.end(), 4);
```

271

Different ways of providing the business logic to C++ algorithms

- Ordinary *function* and pass a function pointer

```
bool odd(int n) {return n%2 == 1;}
count_if(begin, end, odd)
```

- Class/struct with *function call operator* and pass an object

```
struct Odd {
    bool operator()(int n){return n%2 == 1;}
};
count_if(begin, end, Odd{})
```

- *Lambda* expression

```
count_if(begin, end, [](int n){return n%2 == 1;})
```

272

Sample C++ algorithm invocations

for-each-demo.cpp

```cpp
//COMPILE: g++ --std=c++11 for-each-demo.cpp -o for-each-demo
//RUN: ./for-each-demo

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

#define BUSINESS  cout << n*n << " "

void print(int n) {BUSINESS;}

struct PrintFunctor {
    void operator()(int n) {BUSINESS;}
};

int main() {
    vector<int>     v{2,3,5,7,11,13,17,19};

    cout << "STL Classic: ";
    for_each(v.begin(), v.end(), print);
    cout<<endl;

    cout << "STL Functor: ";
    for_each(v.begin(), v.end(), PrintFunctor());
    cout<<endl;

    cout << "STL Lambda : ";
    for_each(v.begin(), v.end(), [](int n){BUSINESS;});
    cout<<endl;

    return 0;
}
```

jens@vbox3: ~/Documents/Advanced-C++-and-Threads

```
Yes> g++ --std=c++11 for-each-demo.cpp -o for-each-demo
Yes> ./for-each-demo
STL Classic: 4 9 25 49 121 169 289 361
STL Functor: 4 9 25 49 121 169 289 361
STL Lambda : 4 9 25 49 121 169 289 361
Yes>
```

273

Function suffixes

- Operates over an existing range
  `xyz(begin, end, …)`
- Uses a count to control the number of invocations
  `xyz_n(begin, count, …)`
- Uses a predicate to control the invocation outcome
  `xyz_if(begin, end, predicate, …)`
  - A predicate is a function that returns a bool

274

Three ways to copy elements

```
vector<int>        numbers{1,2,3,4,5,6,7,8,9,10};
ostream_iterator   out{cout, " "};
```

```
copy(numbers.begin(), numbers.end(), out);
```

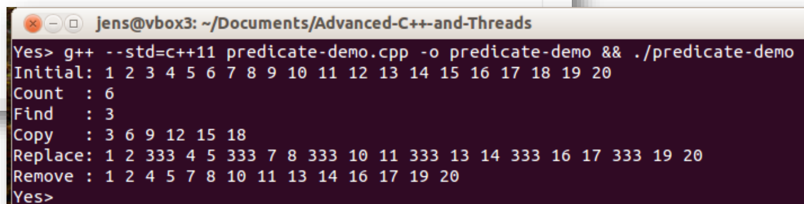
```
copy_n(numbers.begin(), number.size()/2, out);
```

```
copy_if(numbers.begin(), numbers.end(), out, [](int n){
  return n%2 == 0;
});
```

275

Using a predicate: *_if(begin, end, predicate)

```
17 int main() {
18     vector<int>  v(20);
19     iota(v.begin(), v.end(), 1);
20     dump("Initial", v);
21
22     auto divisableBy3 = [](int n){return n % 3 == 0;};
23
24     cout << "Count   : " << count_if(v.begin(), v.end(), divisableBy3) << endl;
25     cout << "Find    : " << *find_if(v.begin(), v.end(), divisableBy3) << endl;
26
27     cout << "Copy    : ";
28     copy_if(v.begin(), v.end(), ostream_iterator<int>(cout, " "), divisableBy3);
29     cout<<endl;
30
31     replace_if(v.begin(), v.end(), divisableBy3, 333);
32     dump("Replace", v);
33
34     auto last = remove_if(v.begin(), v.end(), divisableBy3);
35     v.erase(last, v.end());
36     dump("Remove ", v);
37
38     return 0;
39 }
```



276

transform

- Applies a function on each element

```
void main() {
  vector<int> v = {1,2,3,4,5};
  transform(v.begin(), v.end(), v.begin(), [](int n){return n*n;});
  //v === 1 4 9 16 25
}
```

```
int  sum(int a, int b) {return (a+b);}
void main() {
    int arr1[] = {1,2,3,4,5};
    int arr2[] = {5,4,3,2,1};
    transform(arr1, arr1+5, arr2,
              ostream_iterator<int>(cout, " "), sum);
}
//prints: 6 6 6 6 6
```

277

accumulate

- Aggregates all elements to a single value

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int>  numbers = {10,5,12,3,8,4};
    int prod = accumulate(numbers.begin(), numbers.end(), 1, [](int acc, int n) {
        return acc * n;
    });
    cout << "PROD = " << prod << endl;

    int max = accumulate(numbers.begin(), numbers.end(), 0, [](int acc, int n) {
        return n > acc ? n : acc;
    });
    cout << "MAX  = " << max << endl;

    return 0;
}
```

```
PROD = 57600
MAX  = 12
```

278

Sorting

```
int main() {
    vector<int> v = {5, 7, 3, 10, 1, 12, -5};
    print("unsorted", v);

    sort(v.begin(), v.end(), [](int left, int right){
        return left < right;
    });
    print("sorted (asc)", v);

    sort(v.begin(), v.end(), [](int left, int right){
        return left > right;
    });
    print("sorted (desc)", v);

    return 0;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
using namespace std;
void print(const char* msg, const vector<int>& v) {
    cout << left << setw(15) << msg;
    for (auto n :v) cout << n << " "; cout<<endl;
}
```

```
unsorted       5 7 3 10 1 12 -5
sorted (asc)   -5 1 3 5 7 10 12
sorted (desc)  12 10 7 5 3 1 -5
```

279

Different ways to populate a container

```
int main() {
    vector<int>  numbers(10, 42);
    print("Constructor", numbers);

    fill(numbers.begin(), numbers.end(), 21);
    print("Constant", numbers);

    iota(numbers.begin(), numbers.end(), -4);
    print("Stepper", numbers);

    int f2 = 0, f1 = 1;
    generate(numbers.begin(), numbers.end(), [&](){
        int f = f1 + f2;
        f2 = f1; f1 = f;
        return f;
    });
    print("Generator", numbers);

    int k = 10;
    transform(numbers.begin(), numbers.end(), numbers.begin(), [=](int n){
        return n*k;
    });
    print("Transformer", numbers);
    return 0;
}
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
using namespace std;

void print(const char* msg, const vector<int>& v) {
    cout<<left<<setw(12)<<msg;
    for (auto n :v) cout<<n<<" ";
    cout<<endl;
}
```

```
Constructor 42 42 42 42 42 42 42 42 42 42
Constant    21 21 21 21 21 21 21 21 21 21
Stepper     -4 -3 -2 -1 0 1 2 3 4 5
Generator   1 2 3 5 8 13 21 34 55 89
Transformer 10 20 30 50 80 130 210 340 550 890

Process finished with exit code 0
```

280

CHAPTER SUMMARY

Algorithms

- The design of the C++ algorithms is simply brilliant
- Important to master the most common algorithm functions and the most common iterator types

281

EXERCISE

Numbers

- Load a bunch of numbers from stdin into a container, using istream_iterator
- Find and compute the following, using STL algorithms
  - Smallest and largest value
  - Mean/average value (μ)
  - Standard deviation (σ)

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \qquad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

282

283

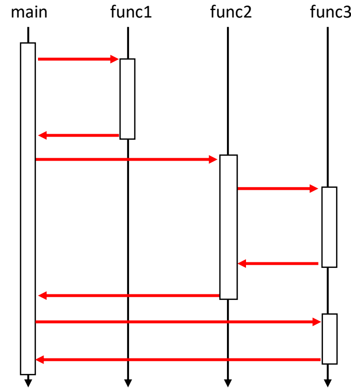
Introduction to Threads

- ✓ Concurrent vs. Parallel
- ✓ Application/Process/Threads/ Task
- ✓ Virtual address space
- ✓ Function call mechanics
- ✓ Concurrency operations

284

A single thread program

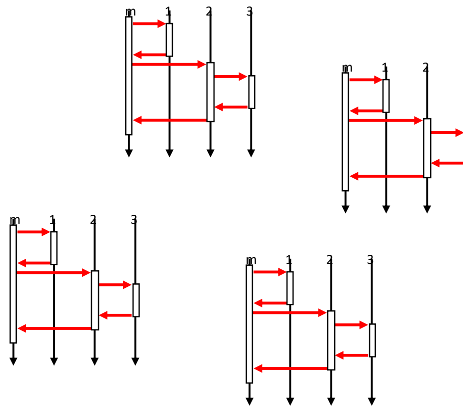
```
int main() {
  func1(17);
  func2(3);
  func3(42)
}
void func1(int n) {
  printf("Hepp");
}
void func2(int n) {
  func3(2*n);
}
void func3(int n) {
  printf("Happ");
}
```



285

A multi-threaded program

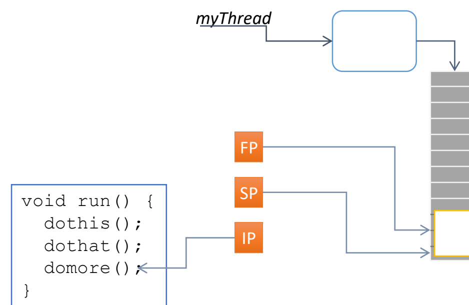
- Every thread has its own "main" subroutine
- The OS manages the execution time of each thread



286

Thread (Thread of Control)

- A context for executing
  - a sequence of instructions and
  - maintaining a stack of subroutine local variables
- Manages
  - Instruction pointer
  - Stack/Frame pointers



287

Hello thread

```
#include <iostream>
#include <pthread.h>

void*  run(void* arg) {
  int    N = 1000;
  char*  msg = (char*)arg;

  for(int i=1; i<=N; ++i)
    cout << msg << endl;
}

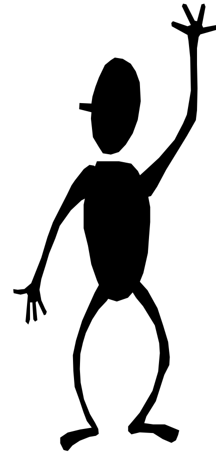
void main() {
  pthread_t  thrId1, thrId2, thrId3;
  pthread_create(&thrId1, NULL, run, (void*)"hi");
  pthread_create(&thrId2, NULL, run, (void*)"     howdy");
  pthread_create(&thrId3, NULL, run, (void*)"          hello");

  pthread_join(thrId1, NULL);
  pthread_join(thrId2, NULL);
  pthread_join(thrId3, NULL);
}
```

288

Running

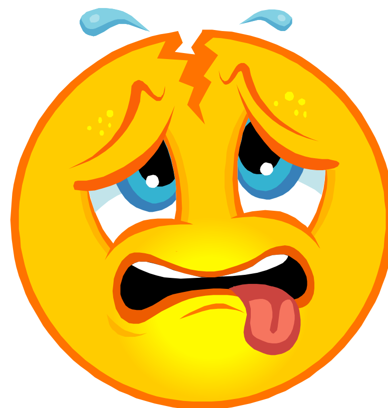
```
$ g++ helloThreads.cpp -lpthread -o helloThreads
$ ./helloThreads
hi
hi
        hello
    howdy
    howdy
    howdy
        hello
        hello
        hello
    howdy
    howdy
    howdy
hi
hi
    howdy
```



289

Concurrency Problems

- Shared Mutual States
  - Critical sections
  - Race conditions
- Resources
  - Deadlocks
  - Starvations
- Processors
  - Livelocks



290

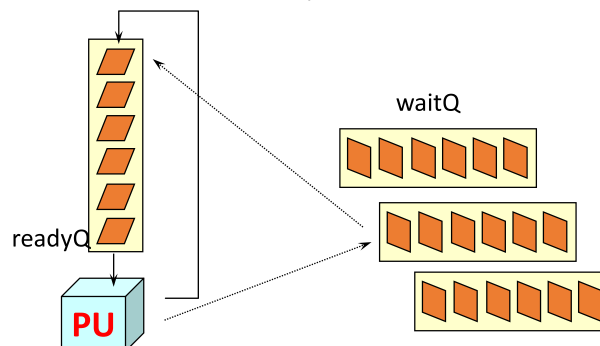
Recommended strategy

- Write programs according to well-proven design-patterns
- Avoid *shared mutual state*
- Use message passing
- Prefer implicit concurrency and synchronization

291

Run-time system

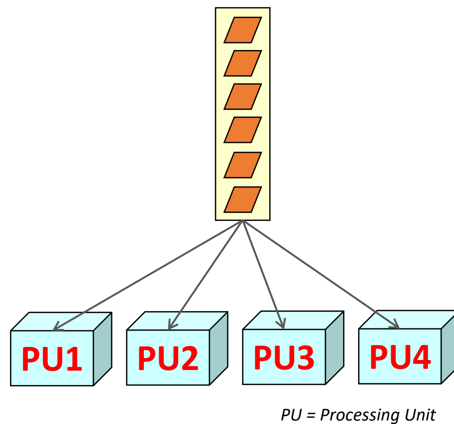
- Behind the scenes there are a set of queues
  - A thread spends its life time moving between different queues indicating different life cycle states
- ReadyQ
  - All threads waiting to execute at (one of) the CPU(s)/Core(s) (PU=Processing Unit)
- WaitQ
  - Waiting for some kind of condition (synchronization)



292

Multi core/cpu

- Can execute one thread per processing unit (core/cpu)

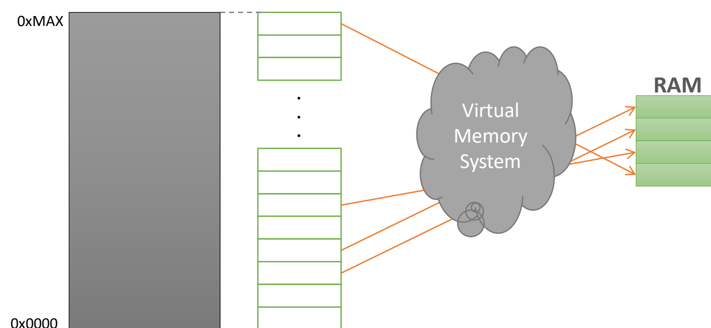


PU = Processing Unit

293

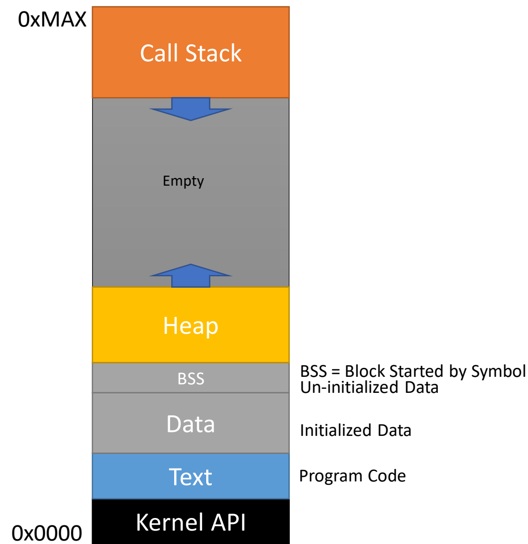
Virtual memory pages

- The VM space is divided into a set of pages
  - Typically 4K (32-bit), 8K (64-bit)
- The paging system map pages from VM into RAM



294

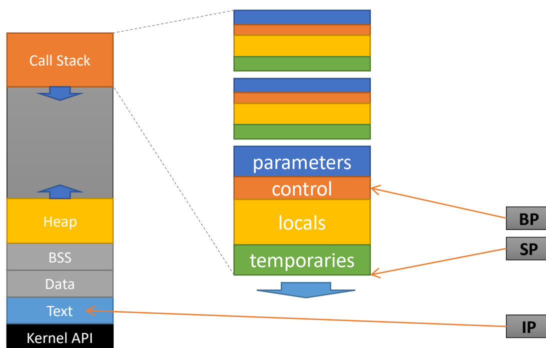
Address space layout



295

The call stack

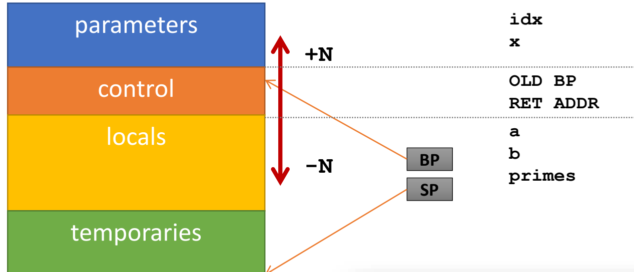
- The call stack contains the actual invocation of functions and their data



296

Addressing

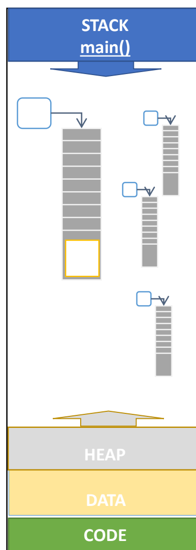
- Data are offset addressed with BP as the base




```
int compute(int x, int idx) {
    int a = 42;
    int primes[] = {2,3,5,7,11,13,17,19};
    int b = primes[idx];
    return a * x + b;
}
```

297

Where do all the thread lives?



Not used, for a multi-threaded application

A thread is basically a call-stack area. Each thread-stack is allocated as a memory segment using `mmap()`.

All dynamic object are shared → available for any thread

All global objects are shared → available for any thread

The compiled code still goes in here

298

CHAPTER SUMMARY

Threads Intro

- Threads are used for
  - Hiding latency, utilizing multi-core/CPU, simply programming
- One readyQ and several waitQ
- Scheduling is needed for time sharing
  - Cooperative or preemptive
- Suspension (wait) is caused by the thread itself

299

EXERCISE

HelloThreads

- Compile and run the HelloThreads application
  - `g++ HelloThreads.cpp -lpthread -o threads`
- Read the loop count from the command line

```
#include <iostream>
#include <string>
using namespace std;
int     N = 100;
. . .
int main(int argc, char* argv[]) {
  if (argc > 1) N = stoi(argv[1]);
  . . .
}
```

300

POSIX Threads C API and C++ Threads

- ✓ Basic usage of a POSIX threads
- ✓ Configurations
- ✓ Wrapping POSIX threads in C++
- ✓ class Thread

301

POSIX Threads in C

- A Unix/Linux standard
- Used as the basis for
  - Concurrent applications in C and C++
  - Implementation base for other programming languages such as Java and Ada
  - Real-time programming
- Contains functions for
  - Launching and terminating threads
  - Various synchronization operations

302

Rudimentary C API

- Include file
  `#include <pthread.h>`
- Thread ID
  `pthread_t`
- Thread creation
  - ```
    int  pthread_create(pthread_t* thrId,
         threadConfig,
         void* (*runFunc)(void*),
         void* runArg)
    ```
- Waiting for thread termination
  - `int  pthread_join(pthread_t thr, void** exitValue)`

303

Create a thread

```
//Compile: g++ hello.cpp -lpthread -o hello
#include <pthread.h>

void* run(void* arg) {
  for(int i=0; i<100; ++i) cout << "Hi there" << endl;
}

void main() {
    pthread_t    thrId;
    pthread_create(&thrId, NULL, run, NULL);
    pthread_join(thrId, NULL);
}
```

Must wait for thread termination, else the application will terminate.

304
```

Create a thread with a parameter

```
#include <pthread.h>

void* run(void* arg) {
  int  id = (int)arg;
  for(int i=0; i<10; ++i) cout << id << " Hi" << endl;
}

void main() {
  pthread_t    thrId[5];
  for (int i=0; i<5; ++i)
    pthread_create(&(thrId[i]), NULL, run, (void*)(i+1));

  for (int i=0; i<5; ++i)
    pthread_join(thrId[i], NULL);
}
```

305

Create a thread with several parameters

```
struct Args {
  int id; string msg;
  Args(int i, string m) {id=i; msg=m;}
};

void* run(void* arg) {
  Args* args = (Args*)arg;
  for(int i=0; i<10; ++i) cout << args->id << args->msg << endl;
  delete args;
}

void main() {
  pthread_t    thrId[5];
  for (int i=0; i<5; ++i) {
    pthread_create(&(thrId[i]),NULL,run, new Args(i+1,"Hello"));
  }
  for (int i=0; i<5; ++i) pthread_join(thrId[i], NULL);
}
```

306

Thread configuration

- Specific struct for configuration attributes

```
pthread_attr_t threadConfig;                    1
pthread_attr_init(&threadConfig);

int rc = pthread_attr_setXYZ(&threadConfig, args);        2
if (rc != 0) {
  errno = rc;
  perror("failed to set the thread stacksize");
  exit(1);
}

pthread_create(&thrId, &threadConfig, startRunFunc, this);      3

pthread_attr_destroy(&threadConfig);            4
```

307

Thread configuration attributes

- Many implementation specific configuration options

```
•*_setdetachstate (…, int          detachstate)
•*_setguardsize   (…, size_t       guardsize)
•*_setinheritsched(…, int          inheritsched)
•*_setschedparam  (…, struct sched_param* param)
•*_setschedpolicy (…, int          policy)
•*_setscope       (…, int          scope)
•*_setstack       (…, void*        stackaddr, size_t stacksize)
•*_setstackaddr   (…, void*        stackaddr)
•*_setstacksize   (…, size_t       stacksize)
```

308

Setting the stack size

```
size_t   stackSize;

void   setStackSize(size_t _stackSize) {
  if (_stackSize < PTHREAD_STACK_MIN) {
    _stackSize = PTHREAD_STACK_MIN;
  }
  stackSize = _stackSize;
}

void    start() {
  pthread_attr_t    threadConfig;
  pthread_attr_init(&threadConfig);
  if (stackSize > 0) {
    int rc = pthread_attr_setstacksize(&threadConfig, stackSize);
    if (rc != 0) {...exit(1);}
  }
  pthread_create(&thrId, &threadConfig, startRunFunc, this);
  pthread_attr_destroy(&threadConfig);
}
```

309

POSIX Threads and C++

- Using plain C, clutters up the code
- Better to use C++ to and hide the gory details
- Basic idea

```
class MyThread : public Thread {
  void run() { . . . }
};
MyThread*  t = new MyThread{};
t->start();
t->join();
delete t;
```

310

A C++ hello thread

```
class Hello : public Thread {
  int    id, n;

public:
  Hello(int _id, int _n) : id(_id), n(_n) { }

protected:
  virtual void    run() {
    for (int i=1; i<=n; ++i) cout << id << ":" << i << endl;
  }
};
```

```
#include <vector>
int main(int numArgs, char* args[]) {
  int     T = numArgs > 1 ? stoi(args[1]) : 5;
  int     N = numArgs > 2 ? stoi(args[2]) : 10;
  vector<Hello*> threads;
  for (int i=0; i<T; ++i) threads.push_back( new Hello(i+1, N) );
  for (Hello* t : threads) t->start();
  for (Hello* t : threads) t->join();
  for (Hello* t : threads) delete t;
}
```

311

class Thread

```
#include <pthread.h>
class Thread {
  pthread_t    thrId;

  static void*  runWrapper(void* arg) {
    Thread* self = (Thread*)arg;
    self->run();
  }

 public:
  void    start() {
    pthread_create(&thrId, NULL, runWrapper, this);
  }

  void    join() { pthread_join(thrId, NULL); }

 protected:
  virtual void    run() = 0;
};
```

312

CHAPTER SUMMARY

POSIX Threads C API

- POSIX Threads is the low-level C standard for acquiring and maintaining a thread from the OS
  - pthread_create
  - pthread_join
- C++ helps wrapping and hiding the verbose C layer
- C++11 provides a whole new range of thread capabilities

313

CHAPTER SUMMARY

C++ threads

- Implement class Thread
  - Add it to a namespace
- Use it to re-implement the HelloThreads demo

Thread.hpp

```
namespace cxx_threads {
  class Thread {
    . . .
  };
}
```

KEEP THIS CLASS
YOU ARE GOING TO USE IT FOR
THE REMAINDER OF THE COURSE

314

The Critical-Section problem

- ✓ Problem demonstration and analysis
- ✓ The solution using POSIX mutex locks
- ✓ Lock configuration
- ✓ class Mutex
- ✓ class Synchronized

315

Model Problem: The ATM Problem

316

Account

```
class Account {
    int balance = 0;

public:
    Account() = default;
    virtual ~Account() = default;
    Account(const Account&) = delete;
    Account& operator=(const Account&) = delete;

    virtual void update(int amount) {
        int b = balance;     //READ
        b += amount;         //MODIFY
        balance = b;         //WRITE
    }

    int getBalance() const {
        return balance;
    }

};
```

317

Updater thread

```
class Updater : public cxx_threads::Thread {
    const string   name;
    const unsigned numTransactions;
    const unsigned amount;
    Account& theAccount;

public:
    Updater(unsigned id, unsigned numTransactions, unsigned amount, Account& theAccount) :
            name{"updater-"s + to_string(id)},
            numTransactions{numTransactions},
            amount{amount},
            theAccount{theAccount} {}

    ~Updater() = default;

protected:
    void run() override {
        cout << (name + " started\n");
        for (auto k = 0U; k < numTransactions; ++k) theAccount.update(+amount);
        for (auto k = 0U; k < numTransactions; ++k) theAccount.update(-amount);
        cout << (name + " done\n");
    }
};
```

318

ATM

```
template<typename Account>
struct ATM {
    unsigned numUpdaters     = 10;
    unsigned numTransactions = 100'000;
    unsigned amount          = 100;
    Account* theAccount;
    vector<Updater*> updaters;

    Account* createAccount() {
        return new Account{};
    }

    void parseArgs(int n, char* args[]) {...}

    void setup() {
        theAccount  = createAccount();
        for (auto k = 0U; k < numUpdaters; ++k)
            updaters.push_back(new Updater{k + 1, numTransactions, amount, *theAccount});
    }

    void launch() {
        for (auto u : updaters) u->start();
        for (auto u : updaters) u->join();
    }

    void finalBalance() {
        cout << "Final balance = " << theAccount->getBalance() << endl;
    }

    void dispose() {
        for (auto u : updaters) delete u;
        delete theAccount;
    }

    void run(int n, char* args[]) {
        cout.imbue(locale("en_US.UTF8"));
        parseArgs(n, args);
        setup();
        launch();
        finalBalance();
        dispose();
    }
};
```

```
int main(int argc, char* argv[]) {
    ATM<Account>  atm;
    atm.run(argc, argv);
    return 0;
}
```

319

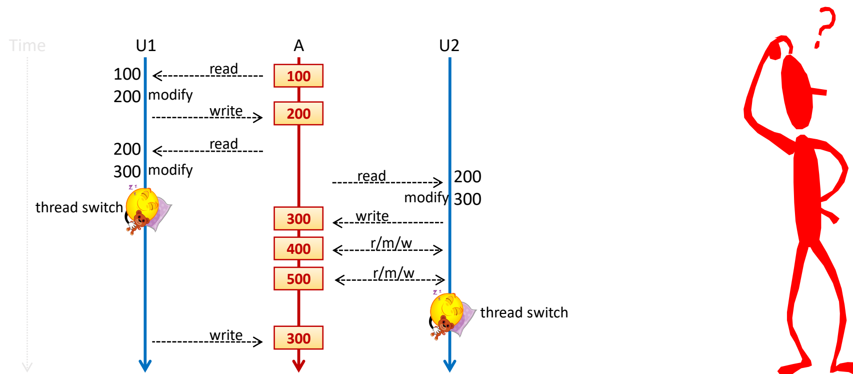
Running

```
C++> ./bank-unsafe -q
params: 10 / 100,000
Final balance = -3,113,400
C++> ./bank-unsafe -q
params: 10 / 100,000
Final balance = 12,550,500
C++> ./bank-unsafe -q
params: 10 / 100,000
Final balance = 4,403,900
C++> ./bank-unsafe -q
params: 10 / 100,000
Final balance = -3,834,300
C++>
```

320

What is wrong?

- A thread switch after reading the balance means two or more threads get the same balance
- The writing the updated balance overwrites the balance of the account



321

The critical code segment

```
class Account {
    int balance = 0;

public:
    Account() = default;
    virtual ~Account() = default;
    Account(const Account&) = delete;
    Account& operator=(const Account&) = delete;

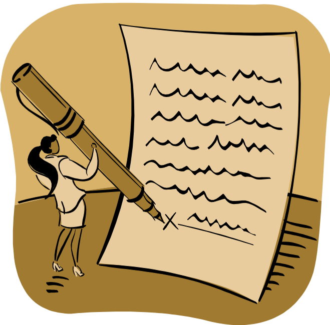
    virtual void update(int amount) {
        int b = balance;      //READ
        b += amount;          //MODIFY
        balance = b;          //WRITE
    }

    int getBalance() const {
        return balance;
    }
};
```

322

Conditions required for the CS problem

1. A non-atomic READ/MODIFY/WRITE code sequence
2. Concurrent invocations (threads > 1) of the code sequence
3. Several thread-switches happening inside the code sequence

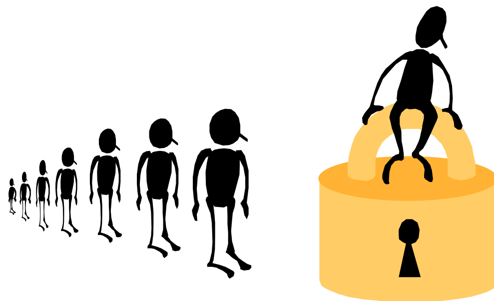


This means:
Invalidate at least of the conditions above and you don't have the CS problem.

323

Usage of a mutex lock

- A *mutex* (MUTual EXclusion) lock ensures that at most 1 thread executes within a code segment guarded by
  - `lock() ... unlock()`
- Logically, a mutex is a boolean flag plus a queue for waiting threads



324

POSIX Mutex

- Data type
  - `pthread_mutex_t  myLock;`
- Life cycle operations
  - `int pthread_mutex_init(pthread_mutex_t* mutex, attr)`
  - `int pthread_mutex_destroy(pthread_mutex_t* mutex)`
- Lock operations
  - `int pthread_mutex_lock(pthread_mutex_t* mutex)`
  - `int pthread_mutex_trylock(pthread_mutex_t* mutex)`
  - `int pthread_mutex_unlock(pthread_mutex_t* mutex)`

325

Safe ATM

```
#include <pthread.h>
#include "ATM.hxx"

class SynchronizedAccount : public Account {
    using super = Account;
    pthread_mutex_t mutex{};

public:
    SynchronizedAccount() {
        pthread_mutex_init(&mutex, nullptr);
    }

    ~SynchronizedAccount() {
        pthread_mutex_destroy(&mutex);
    }

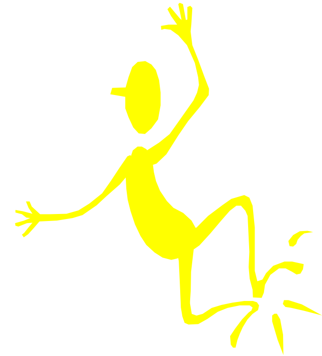
    void update(int amount) override {
        pthread_mutex_lock(&mutex);
        super::update(amount);
        pthread_mutex_unlock(&mutex);
    }
};

int main(int argc, char* argv[]) {
    ATM<SynchronizedAccount> atm;
    atm.run(argc, argv);
    return 0;
}
```

326

Running again

```
C++> ./bank-safe -q
params: 10 / 100,000
Final balance = 0
C++> ./bank-safe -q
params: 10 / 100,000
Final balance = 0
C++> ./bank-safe -q -u 50 -t 1000000
params: 50 / 1,000,000
Final balance = 0
C++> ./bank-safe -q -u 100 -t 1000000
params: 100 / 1,000,000
Final balance = 0
C++>
```

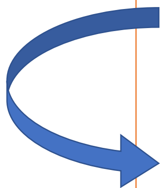


327

Self-deadlock

```
class Account {
  pthread_mutex_t  lock;
  int     balance;
 public:
  ...
  int  update(int n) {
    pthread_mutex_lock(&lock);      (1)
    if (getBalance() < 0) {...}
    ...
    pthread_mutex_unlock(&lock);
  }

  int  getBalance() {
    pthread_mutex_lock(&lock);      (2)
    ...
    pthread_mutex_unlock(&lock);
  }
};
```



328

Recursive mutex lock configuration

- If there is change that the same mutex will be locked twice or more, the lock should be initialized as "recursive"

```
recursive = true;

pthread_mutexattr_t    cfg;
pthread_mutexattr_init(&cfg);
if (recursive) {
    pthread_mutexattr_settype(&cfg, PTHREAD_MUTEX_RECURSIVE);
}
pthread_mutex_init(&lock, &cfg);
pthread_mutexattr_destroy(&cfg);
```

329

class Mutex

```
class Mutex {
  pthread_mutex_t    mutex;

 public:
  Mutex()   {pthread_mutex_init(&mutex, nullptr);}
  ~Mutex()  {pthread_mutex_destroy(&mutex);}

  void    enter() {pthread_mutex_lock(&mutex);}
  void    exit()  {pthread_mutex_unlock(&mutex);}

  pthread_mutex_t*  native() {return &mutex;}
};
```

330

class Guard

```
class Guard {
  Mutex&  region;

public:
  Guard(Mutex& m) : region{m} {
    region.enter();
  }

  ~Guard() {
    region.exit();
  }
};
```

```
Mutex  region;
//. . .
void  updateSomething() {
  Guard  g{region};
    //. . .
  return;
}
```

Constructor locks the mutex

Destructor unlocks the mutex

331

A better Account class

```
class Account {
  Mutex    lock;
  int      balance;

 public:
  Account(int b=0) : balance(b) {}

  int      getBalance() {
    Guard g{lock};
    return balance;
  }

  int     update(int amount) {
    Guard g{lock};
    balance += amount;
    return balance;
  }
};
```

332

Chapter Summary

The Critical-Section problem

- Conditions needed for the CS problem
  - Non-atomic READ/MODIFY/WRITE
  - Concurrent invocations
  - Thread switches within the code segment
- POSIX
  - mutex_t lock
- Design recommendation
  - Design shared data structures, avoid code locking

333

Exercise

Mutex & Guard

- Implement class Mutex
  - Add support for recursive locking
- Implement class Guard

SAVE IT TO YOUR LIBRARY

334

EXERCISE

Bank++

- Design & implement
  - class Account to be thread-safe
  - class Updater as a thread
- Implement a *working solution* of the Bank-problem

335

Intentional Blank

336

The Race-Condition problem

- ✓ Problem demonstration and analysis
- ✓ The solution using POSIX condition variables
- ✓ Configuration
- ✓ Wrapping it in C++
- ✓ The minimum amount of code needed, in order to safely transfer data from one thread to another

337

Model Problem: The SWIFT Problem

SWIFT is the name of the international money transfer network.
https://en.wikipedia.org/wiki/Society_for_Worldwide_Interbank_Financial_Telecommunication

338

MoneyBox

```cpp
class MoneyBox {
protected:
    int   payload = 0;
    Mutex mutex;

public:
    MoneyBox() = default;
    virtual ~MoneyBox() = default;
    MoneyBox(const MoneyBox&) = delete;
    MoneyBox& operator=(const MoneyBox&) = delete;

    virtual void send(int amount) {
        Guard g{mutex};
        payload = amount;
    }

    virtual int recv() {
        Guard g{mutex};
        return payload;
    }
};
```

339

Sending & Receiving banks

```cpp
class SendingBank : public Thread {
    const unsigned numTransactions;
    MoneyBox& out;
    const int amount;
    const int stop;

public:
    SendingBank(unsigned numTransactions, MoneyBox& out, int amount = 1, int stop = -1) :
            numTransactions{numTransactions},
            out{out},
            amount{amount},
            stop{stop} {}

    unsigned getTotal() const {
        return numTransactions*amount;
    }

protected:
    void run() override {
        for (auto k = 0U; k < numTransactions; ++k) {
            out.send(amount);
        }
        out.send(stop);
    }
};
```

```cpp
class ReceivingBank : public Thread {
    MoneyBox& in;
    const int stop;
    int       count = 0;
    int       total = 0;

public:
    ReceivingBank(MoneyBox& in, int stop = -1) :
            in{in},
            stop{stop} {}

    int getCount() const { return count; }
    int getTotal() const { return total; }

protected:
    void run() override {
        for (int amount; (amount = in.recv()) != stop;) {
            ++count;
            total += amount;
        }
    }
};
```

340

The SWIFT object

```
template<typename MoneyBox>
struct SWIFT {
    MoneyBox* theBox;
    unsigned numTransactions = 10'000;
    int      amount          = 1;
    bool     verbose         = true;

    MoneyBox* createBox() {
        return new MoneyBox{};
    }

    void parseArgs(int n, char* args[]) {...}

    void setup() { theBox = createBox(); }

    void launch() {
        SendingBank   sender{numTransactions, *theBox, amount};
        ReceivingBank receiver{*theBox};

        receiver.start();
        sender.start();
        sender.join();
        receiver.join();

        cout << "sender  : total=" << sender.getTotal() << endl;
        cout << "receiver: total=" << receiver.getTotal() << ", count=" << receiver.getCount() << endl;
    }

    void dispose() { delete theBox; }

    void run(int n, char* args[]) {
        cout.imbue(locale("en_US.UTF8"));
        parseArgs(n, args);
        setup();
        launch();
        dispose();
    }
};
```

```
int main(int argc, char* argv[]) {
    SWIFT<MoneyBox> swift;
    swift.run(argc,argv);
    return 0;
}
```

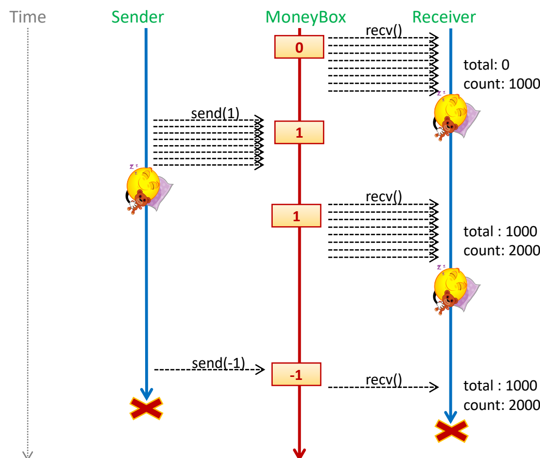
341

Running

```
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=9,377, count=9,522
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=7,850, count=8,187
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=7,857, count=8,390
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=10,811, count=11,444
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=3,674, count=17,472
C++> ./swift-unsafe -q
params: 10,000
sender  : total=10,000
receiver: total=39,255, count=40,814
C++>
```

342

What is wrong?



The producer does not wait for the consumer and vice versa.

That means; the producer must wait until the consumer has consumed and vice versa

343

Event/condition synchronization

- When a logical condition is not true, within a mutex guarded code segment, the thread must be suspend and wait for it to become true
  - When a thread is suspended it releases the mutex lock it holds as well
  - When it's resumed it reacquires the mutex
- Basic operations
  - wait()
    - Suspends the calling thread and unlocks the mutex
  - signal() / notify()
    - Resumes the first waiting thread
  - broadcast() / notifyAll()
    - Resumes all waiting threads
    - However, only one at a time enters the guarded code segment

344

POSIX Condition

- Data type
  - `pthread_cond_t  myCond;`
- Life cycle operations
  - `int pthread_cond_init(pthread_cond_t* cond, attr)`
  - `int pthread_cond_destroy(pthread_cond_t* cond)`
- Synchronization *suspend* operations
  - `int pthread_cond_wait(pthread_cond_t* cond, pthread_mutex_t* mutex)`
  - `int pthread_cond_timedwait(..., ..., struct timespec* abstime)`
- Synchronization *resume* operations
  - `int pthread_cond_signal(pthread_cond_t* cond)`
  - `int pthread_cond_broadcast(pthread_cond_t* cond)`

345

Safe SWIFT object

```
class SafeMoneyBox : public MoneyBox {
    using super = MoneyBox;
    pthread_cond_t cond{};
    bool full = false;

public:
    SafeMoneyBox() {
        pthread_cond_init(&cond, nullptr);
    }

    ~SafeMoneyBox() {
        pthread_cond_destroy(&cond);
    }

    void send(int amount) override {
        Guard g{mutex};
        while (full) { pthread_cond_wait(&cond, mutex.native()); }
        full = true;
        payload = amount;
        pthread_cond_signal(&cond);
    }

    int recv() override {
        Guard g{mutex};
        while (!full) { pthread_cond_wait(&cond, mutex.native()); }
        full = false;
        auto amount = payload;
        pthread_cond_signal(&cond);
        return amount;
    }
};
```

```
int main(int argc, char* argv[]) {
    SWIFT<SafeMoneyBox> swift;
    swift.run(argc, argv);
    return 0;
}
```

346

Running again

```
C++> ./swift-safe -q
params: 10,000
sender  : total=10,000
receiver: total=10,000, count=10,000
C++> ./swift-safe -q
params: 10,000
sender  : total=10,000
receiver: total=10,000, count=10,000
C++> ./swift-safe -q -t 100000
params: 100,000
sender  : total=100,000
receiver: total=100,000, count=100,000
C++> ./swift-safe -q -t 1000000
params: 1,000,000
sender  : total=1,000,000
receiver: total=1,000,000, count=1,000,000
C++>
```

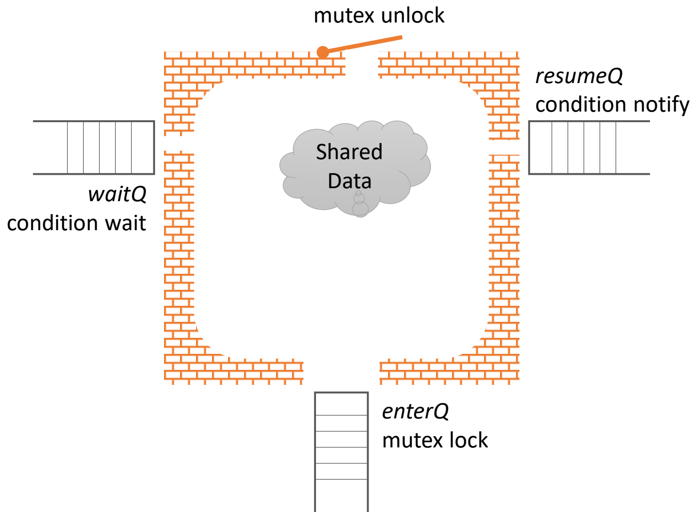
347

The concept "monitor"

- The combination of
  - One mutex
  - One (or more) condition(s)
- Is called a *monitor*, in the concurrency theory literature

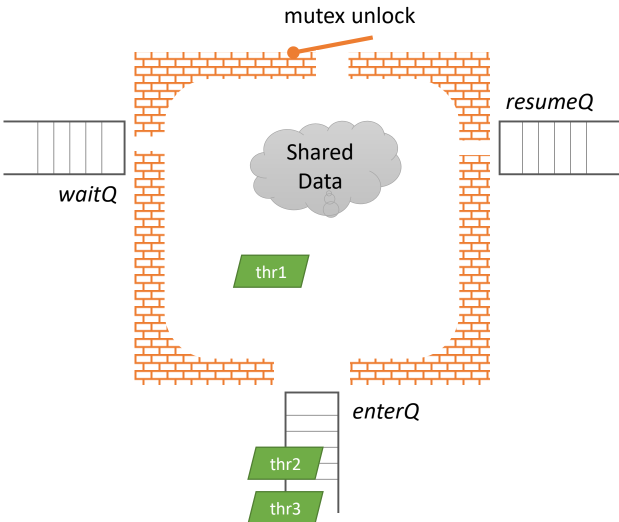
348

Monitor semantics



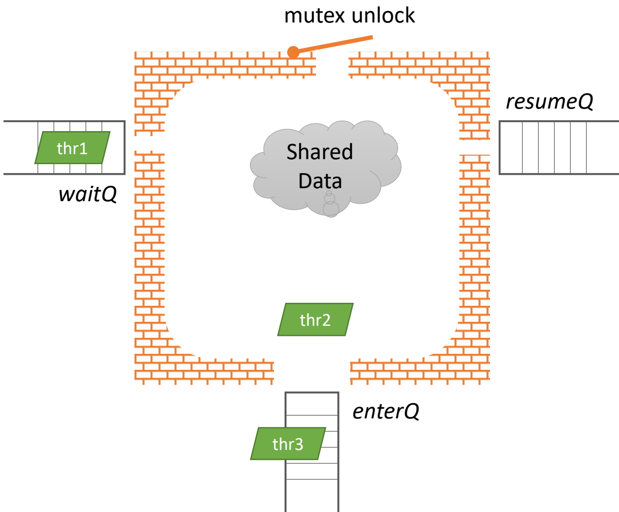
349

Mutex lock



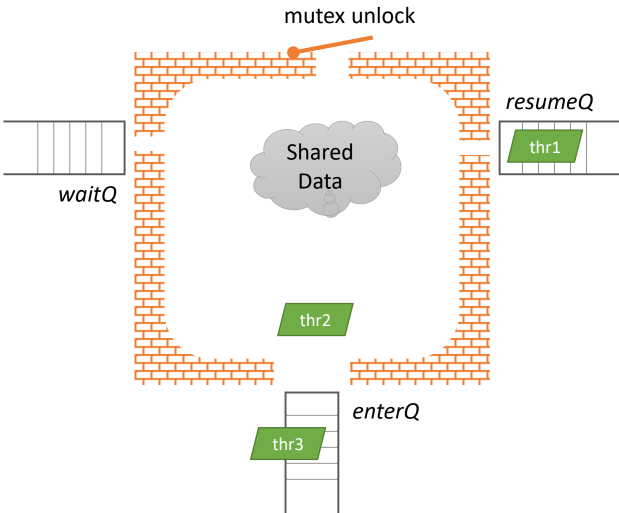
350

Condition wait



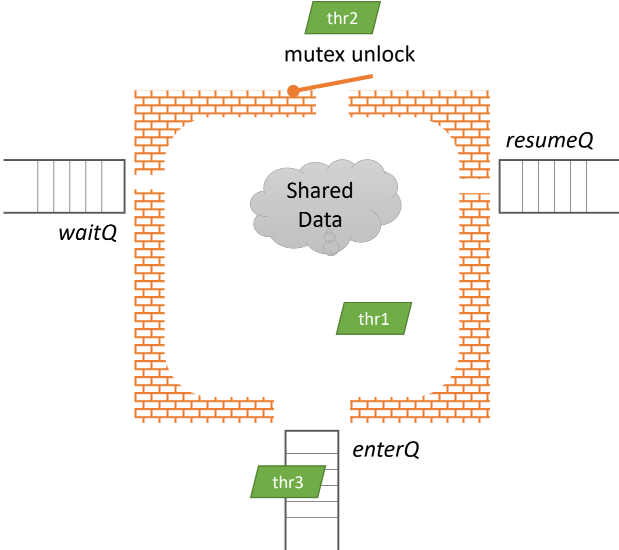
351

Condition notify



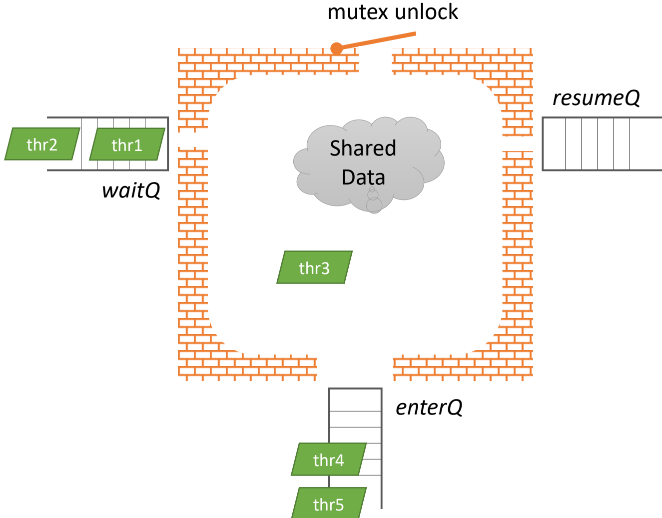
352

Mutex unlock



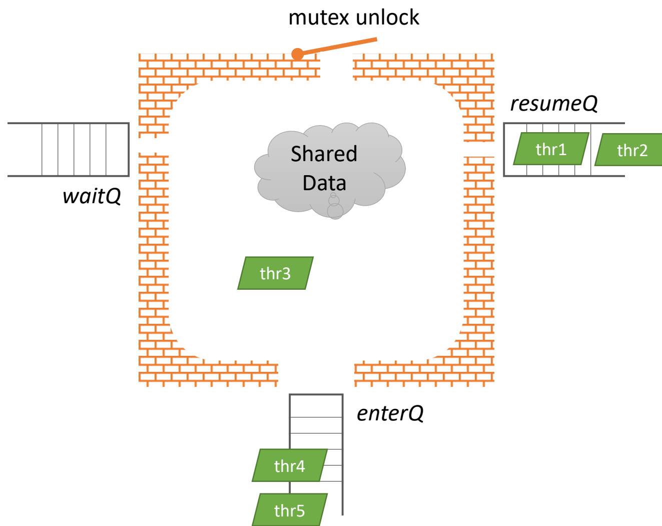
353

Condition wait (many)



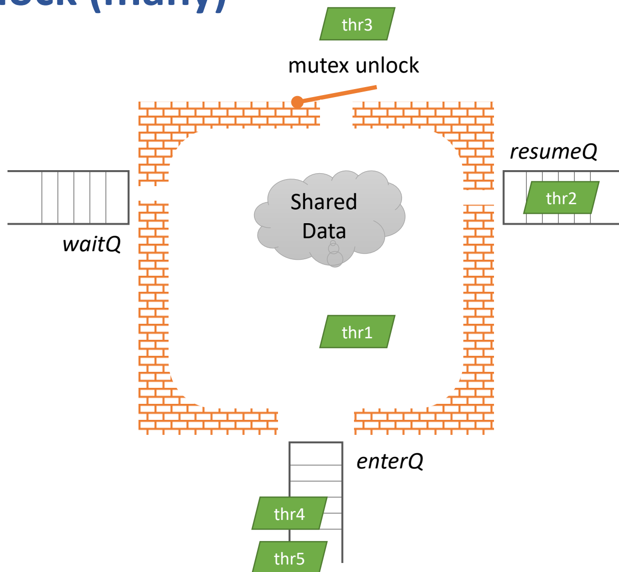
354

Condition notifyAll (many)



355

Mutex unlock (many)



356

class Condition

```
class Condition {
  Mutex&              mutex;
  pthread_cond_t      cond;

 public:
  Condition(Mutex& m) : mutex(m) {
    pthread_cond_init(&cond, NULL);
  }
  ~Condition() {
    pthread_cond_destroy(&cond);
  }

  void    wait() {
    pthread_cond_wait(&cond, mutex.getPosixMutex());
  }
  void    notify()    {pthread_cond_signal(&cond);}
  void    notifyAll() {pthread_cond_broadcast(&cond);}
};
```

357

class SharedData (Java style)

```
class SharedData {
    Mutex         lock;
    Condition     event;

  public:
    SharedData() : event(lock) {}

  protected:
    void    enter()     {lock.enter();}
    void    exit()      {lock.exit();}
    void    wait()      {event.wait();}
    void    notify()    {event.notify();}
    void    notifyAll() {event.notifyAll();}

    class Synchronized {
        SharedData*    context;
      public:
        Synchronized(SharedData* m) : context(m) {context->enter();}
        ~Synchronized() {context->exit();}
    };
};
```

358

class MailBox (Java style)

```
template<typename MsgType>
class MailBox : protected SharedData {
  MsgType      payload{};
  bool         full = full;
 public:
  MailBox() = default;

  void  put(MsgType msg) {
    Synchronized s{this};
    while (full) wait();
      payload = msg; full = true;
    notifyAll();
  }

  MsgType get() {
    Synchronized s{this};
    while (!full) wait();
      MsgType  msg = payload; full = false;
    notifyAll();
    return msg;
  }
};
```

359

CHAPTER SUMMARY

The Race-Condition problem

- Condition variables are needed for any non-trivial shared data-structure
- Monitor (mutex/cond) semantics

360

EXERCISE

Condition

SAVE IT TO YOUR LIBRARY

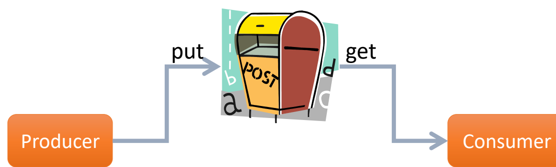
- Implement class Condition
- EXTRA
  - Add an additional method for wait() that takes a predicate lambda, that returns true when the wait should stop
  - This method will encapsulate the while loop

361

EXERCISE

Producer - Consumer

- Design and implement
  - class Mailbox as a thread-safe communication conduit
  - class Producer as a thread
    - Sends 1,2,3, …, N to the mailbox
  - class Consumer as a thread
    - Receives all numbers from the mailbox and prints them out



362

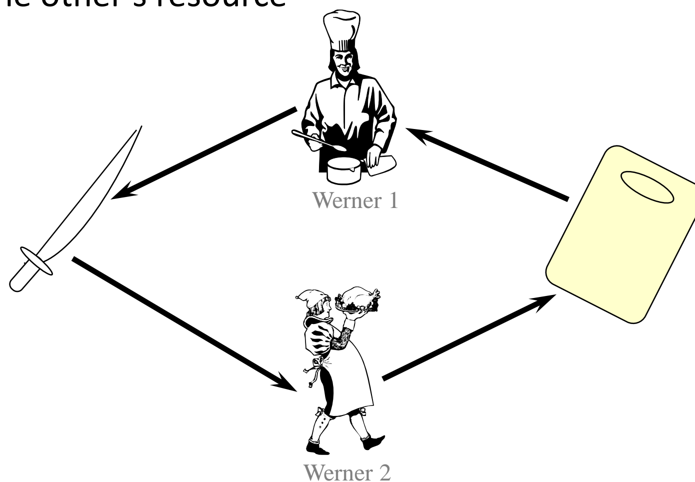
The Deadlock problem

- ✓ Deadlocks and how they appear
- ✓ How to prevent deadlocks

363

Deadlock

- Two or more threads has acquired a reasource each and at the same time waiting for the other's resource

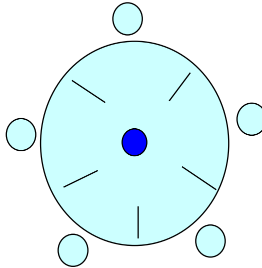


https://www.oppetarkiv.se/video/10438939/nojesmassakern

364

Dining philosofers

- Five philosofers are spending their time thinking and eating
- There is one bowl of rice at the table and five chopsticks
- A philosofer need two chopsticks to eat

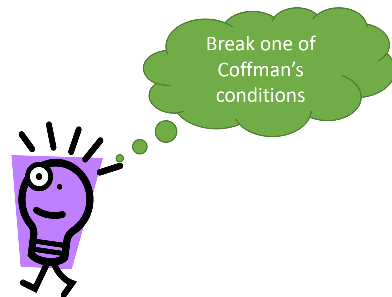


https://en.wikipedia.org/wiki/Dining_philosophers_problem

365

Coffman's conditions for deadlock

1. Exclusive ownership
2. Hold and wait
3. Non-preemptive possession
4. Circular wait



https://en.wikipedia.org/wiki/Deadlock

366

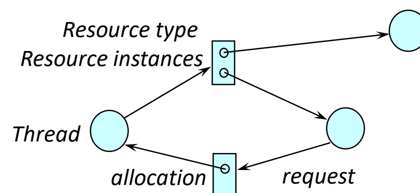
How to handle deadlocks

- Prevention
  - Invalidate one of the Coffman conditions
    - Shared ownership
    - Allocate all or nothing
    - Preempt possession
    - Well defined acquisition order that everybody follows
- Avoidance
  - Wait for a resource only a limited time
  - Before granting an acquisition, check if it might lead to deadlock
- Detection
  - Reduce a resource-allocation graph

367

Detection of deadlock

- Empty ReadyQ
- Inspection of thread activities
- Resource allocation graph reduction



368

CHAPTER SUMMARY

The Deadlock Problem

- Resources
- Starvation
- Livelock
- Deadlock
  - Coffman's 4 conditions

CHAPTER SUMMARY

Dining Philosophers

- Design and implement a deadlocking version first
- Then implement prevention – somehow…

```
class Philosofer : Thread {
  ChopStick left, right;

 protected:
  void run() {
   while (true) {
     //sleep...
     left.acquire();
     right.acquire();
       //eat...
     right.release();
     left.release();
   }
 }
};
```

```
class ChopStick {
  bool  busy;
  //… sync needed here …

 public:
  void acquire() {...}
  void release() {...}
};
```

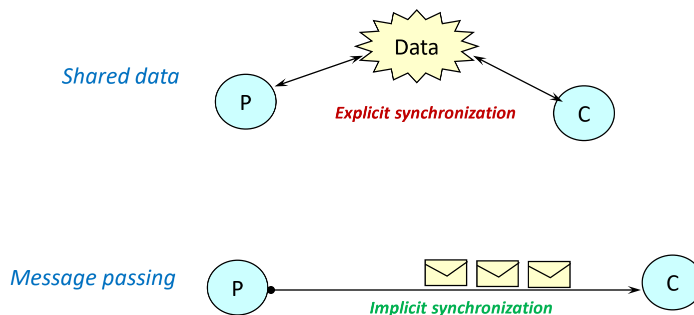
Pseudo Code

Message-Passing

- ✓ What it is and its pros & cons
- ✓ Design of a message queue in C++
- ✓ MessageQueue<T>
- ✓ MessageThread<T>
- ✓ Receivable<T>

371

Shared data *vs.* Message passing



372

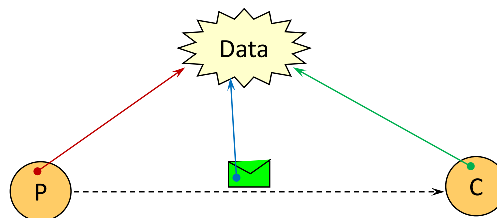
What is a 'message'

- Just a bunch of bytes
  - JBoB
- Primitive data values
  - char, int, float, double, …
- Composite data values
  - struct, array
- Pointer types
  - string, object
- Complex linked pointer types
  - List, set, map, tree, …

373

The problem of sending a pointer (to data)

Passing just the pointer (reference) to the message data re-introduces the problem of shared data

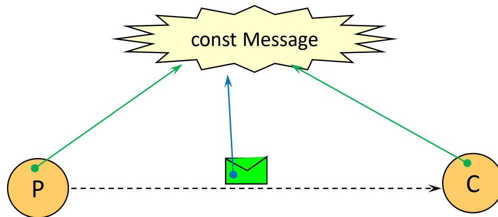


There are three ways to solve this

374

Sending immutable data

Sharing immutable data, do not cause problem



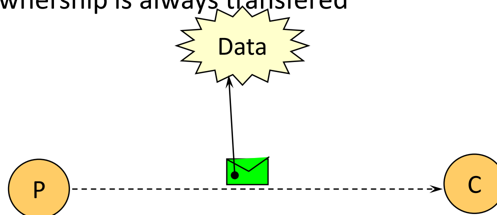

```
template<typename T>
class Message {
  T&  payload;
 public:
  Message(const T& msg)  : payload(msg)  {}
  const T&  get() const {return payload;}
};
```

375

Transfering ownership of a reference

The producer relinquish its reference to the message

The invariant condition is that at most one thread has a reference to the data at any given point in time – data ownership is always transfered



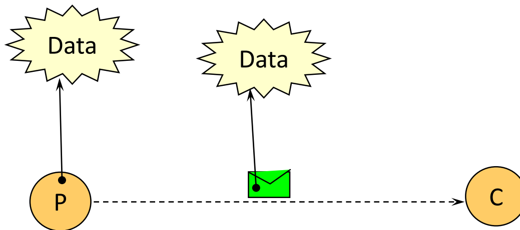

```
template<typename T>
class Message {
  std::unique_ptr<T>  payload;
 public:
  Message(T* msg)  : payload(msg)  {}
  std::unique_ptr<T>  get() const {return payload;}
};
```

376

Sending a copy of the data

Every send operation copies the data

The invariant condition is that at most one thread refers a specific data instance – data is always copied



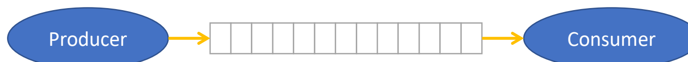

```
template<typename T>
class Message {
  T  payload;
 public:
  Message(T msg)  : payload(msg)  {}
  T  get() const {return payload;}
};
```

For example, use data types / classes on standard class form (SCF)

377

Message Queue

- A mailbox is a queue with 1 message slot
- A message queue allow threads concurrently in both ends
  - As long as there are available slots, a producer can send data
  - As long as there are messages available, a consumer can receive data



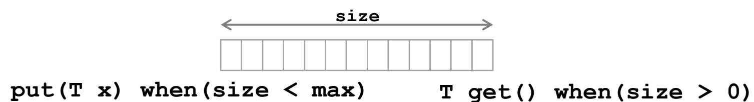
378

Queue capacity

- Unbounded (asynchronous)
  - Will never be full
  - Can lead to memory leaks (too many messages)
- Bounded (synchronous)
  - What to do when the queue is full
    - Suspend the sender
    - Drop new messages
    - Drop old messages (overwriting)

379

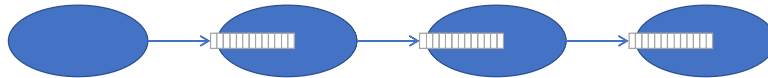
Logical implementation of a message-queue



380

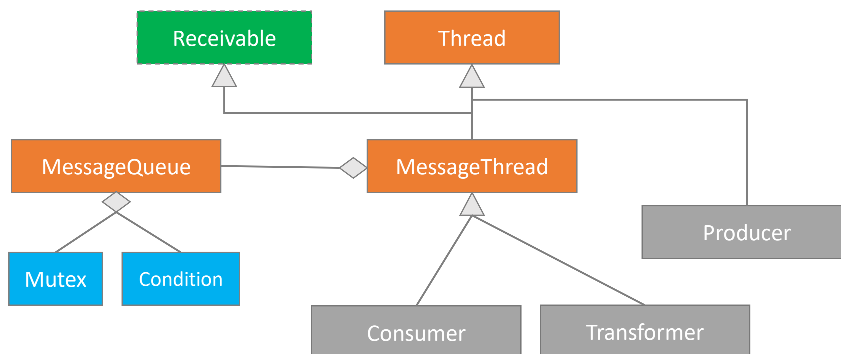
Program structure

- Instead of placing queues (explicitly) between threads it's far more manageable to move each queue into the recipient and wrap it with a set of methods



381

Class Overview



382

interface Receivable<T>

```
template<typename MsgType>
struct Receivable {
  virtual void  send(MsgType msg) = 0;
};
```

```
template<typename T>
class Producer : public Thread {
    Receivable<T>*   next;
  protected:
    void  run() {...next->send(x);...}
    ...
};
```

```
template<typename T>
class Consumer : public MessageThread<T> {
  . . .
  public:
    void  send(T msg) {. . .}
  protected:
   void  run() {...x = recv()...}
};
```

383

Class MessageThread<T>

```
template<typename MessageType>
class MessageThread
   : public Thread,
     public Receivable<MessageType>
{
  MessageQueue<MessageType>    inbox;

protected:
  MessageThread(int queueMaxSize=0) : inbox(queueMaxSize) { }

  MessageType  recv() {return inbox.get();}

  int          queueSize() const {return inbox.size();}

public:
  void    send(MessageType msg) {inbox.put(msg);}
};
```

384

CHAPTER SUMMARY

Message Passing

- Uni-directional message passing simplifies your program logic
- Hide the message sending logic
- API
  - class MessageQueue<T>
  - class MessageThread<T>
  - "interface" Receivable<T>

385

EXERCISE

MessageQueue

- Design and implement class MessageQueue
  - You need 1 Mutex and 2 Condition objects for synchronization
  - Use std::queue for the payload
    - http://en.cppreference.com/w/cpp/container/queue
- Start without templates, if you have time then add template support
- Test it with a pair of producer and consumer threads
  - Producer: send 1,2,3,…,N and finish by sending -1
  - Consumer: receive numbers and print out, end when received negative number

SAVE IT TO YOUR LIBRARY

386

EXERCISE

MessageThread<T> and Receivable<T>

- Implement
  - "interface" Receivable<T>
  - class MessageThread<T>

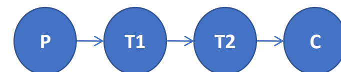
SAVE IT TO YOUR LIBRARY

387

EXERCISE

Pipeline

- Implement a chain of threads
  - class Producer : public Thread
    - Sends a sequence of integers (1,2,3,…,N) to its next
  - class Transformer : public MessageThread<int>
    - Receives a number, multiplies by 2 and sends to next in chain
  - class Consumer : public MessageThread<int>
    - Receives a number and prints it out
- EXTRA: command-line arguments
  - Number (N) of integers to send
  - Number (T) of transformers



388

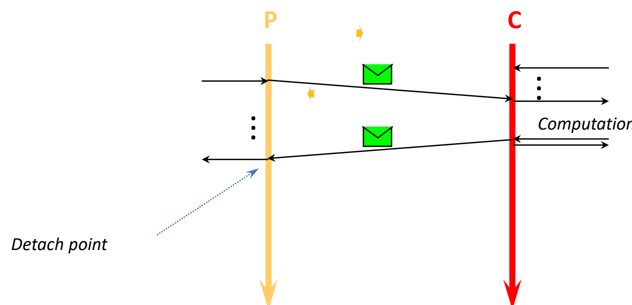
Bi-Directional Message-Passing

- ✓ Rendezvous
- ✓ Futures

389

Extended Rendezvous

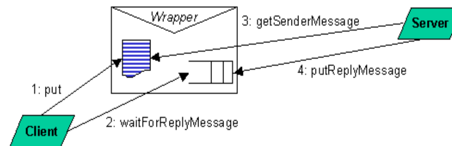
- A value is passed as argument to the consumer and a return value is passed back to the producer
- Function call semantics,
  - a.k.a. Thread Method Invocation (TMI)
  - or Remote Procedure call (RPC)



390

Implementation of TMI

- Message wrapper, inserted into an unbounded channel
  - Payload
  - Single entry channel for the return value
- Producer
  - Creates a wrapper with the message
  - Waits for the return value at the wrapper's channel
- Consumer
  - Processes the payload in the wrapper
  - Puts the return value into the wrapper's channel



391

TMI Message Wrapper

```
template<typename SendType, typename ReplyType>
class RendezvousMessage {
  SendType                    payload;
  MessageQueue<ReplyType>     replyQ;

 public:
  RendezvousMessage(SendType msg) : payload(msg) {}

  SendType  getPayload() const {return payload;}
  void      reply(ReplyType msg) {replyQ.put(msg);}
  ReplyType waitForReply() {return replyQ.get();}
};
```

392

Rendezvous Thread

```
template<typename SendType, typename ReplyType>
class RendezvousThread : public Thread {
  MessageQueue< RendezvousMessage<SendType,ReplyType>* >    msgQ;

 protected:
  RendezvousMessage<SendType,ReplyType>* recv() {
      return msgQ.get();
   }

 public:
  ReplyType   send(SendType msg) {
    RendezvousMessage<SendType,ReplyType>* m = new RendezvousMessage(msg);
    msgQ.put(m);
    ReplyType r = m->waitForReply();
    delete m;
    return r;
  }
};
```

393

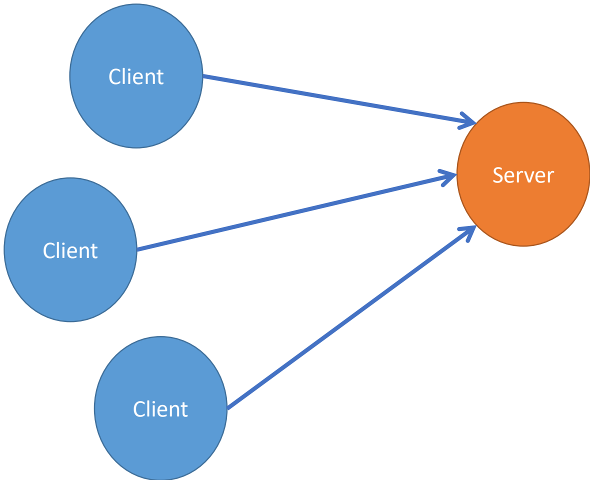
Usage of Rendezvous Thread

```
class Client : public Thread {
  RendezvousThread<int, long>* srv;
 public: virtual void run() {
     long sum = srv->send(100);
     . . .
   }
};
```

```
class Server : public RendezvousThread<int, long> {
 public: virtual void run() {
     do {
       RendezvousMessage<int,long>* m = recv();
       int  n = m->getPayload();
       long result = n * (n+1) / 2;
       m->reply(result);
     } while(true);
   }
};
```

394

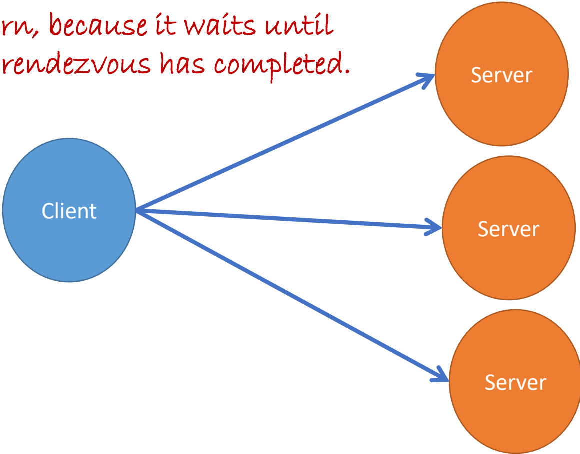
Communication pattern for rendezvous



395

Inverse communication pattern

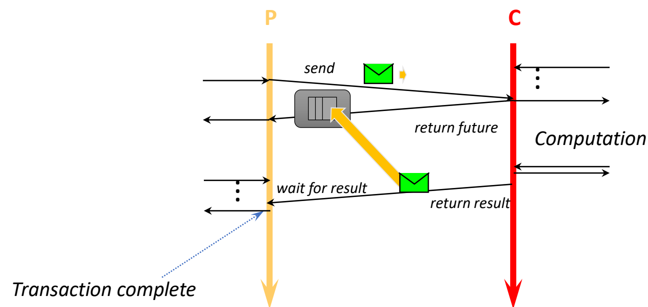
Rendezvous cannot handle this pattern, because it waits until each rendezvous has completed.



396

Future - A rendezvous with a swing

- A message wrapper object is returned from send()
  - Allows the sender to choose when to wait for the reply
- More info
  - http://en.wikipedia.org/wiki/Future_(programming)



397

Future - Message Wrapper

```
template<typename SendType, typename ReplyType>
class Future {
  SendType                    payload;
  MessageQueue<ReplyType>     replyQ;

 public:
  Future(SendType msg) : payload(msg) {}

  SendType  getPayload() const {return payload;}
  void      reply(ReplyType msg) {replyQ.put(msg);}
  ReplyType waitForReply() {return replyQ.get();}
  bool      isDone() {return !replyQ.isEmpty();}
};
```

398

CHAPTER SUMMARY

Bi-Directional Message-Passing

- Message passing is the preferred way of designing concurrent applications
- Asynchronous send
- Synchronous send
- Rendezvous
- Future
- Communication patterns
  - N→1
  - 1→N

EXERCISE

Rendezvous

- Implement
  - class RendezvousMessage
  - class RendezvousThread

SAVE IT TO YOUR LIBRARY

EXERCISE

FibonacciServer

- Design a FibonacciServer thread that computes Fibonacci numbers for clients
  - long fib(int n) {return n <=2 ? 1 : fib(n-1) + fib(n-2);}
- Design a Client thread that asks the server for the Fibonacci numbers
- EXTRA
  - Create more than one client

401

Intentional Blank

402

403

404

std::tuple<T1, T2, T3…>

- Generalization of pair, which defines a fixed-sized collection of values
- Create a tuple
  - auto t = make_tuple(3.14, "Anna Conda", 42, 'B');
- Element access: get<n>(obj)
  - double x = get<0>(t);
  - string s = get<1>(t);
  - int i = get<2>(t);
  - char c = get<3>(t);
- Multi assignment: tie(…)
  - tie(x,s,I,c) = t;

```cpp
#include <tuple>
#include <iostream>
#include <string>
#include <stdexcept>

std::tuple<double, char, std::string> get_student(int id)
{
    if (id == 0) return std::make_tuple(3.8, 'A', "Lisa Simpson");
    if (id == 1) return std::make_tuple(2.9, 'C', "Milhouse Van Houten");
    if (id == 2) return std::make_tuple(1.7, 'D', "Ralph Wiggum");
    throw std::invalid_argument("id");
}

int main()
{
    auto student0 = get_student(0);
    std::cout << "ID: 0, "
              << "GPA: " << std::get<0>(student0) << ", "
              << "grade: " << std::get<1>(student0) << ", "
              << "name: " << std::get<2>(student0) << '\n';

    double gpa1;
    char grade1;
    std::string name1;
    std::tie(gpa1, grade1, name1) = get_student(1);
    std::cout << "ID: 1, "
              << "GPA: " << gpa1 << ", "
              << "grade: " << grade1 << ", "
              << "name: " << name1 << '\n';
}
```

Output:

```
ID: 0, GPA: 3.8, grade: A, name: Lisa Simpson
ID: 1, GPA: 2.9, grade: C, name: Milhouse Van Houten
```

405

std::optional<T> [C++17]

- Representation of a value or absence of value

```cpp
#include <iostream>
#include <string>
#include <optional>
using namespace std;

optional<string> lookup(const string& haystack, const string& phrase) {
    auto pos = haystack.find(phrase);
    if (pos != string::npos) return {haystack.substr(pos, phrase.size())}; else return {};
}

void doit(const string& phrase){
    const auto sentence = "It was a dark and silent night. Suddenly there was a sound of a shot."s;
    cout << "find("<<phrase<<"): " << lookup(sentence, phrase).value_or("NOT FOUND") << endl;
}

int main(int, char**) {
    doit("shot");
    doit("what");
    return 0;
}
```

using-optional

```
/home/jens/Courses/cxx/cxx-embedded
find(shot): shot
find(what): NOT FOUND

Process finished with exit code 0
```

406

variant<T> [C++17]

- Type safe union

```
#include <iostream>
#include <string>
#include <variant>
using namespace std;
using namespace std::literals;

int main(int argc, char** argv) {
    variant<int, double, string> payload{"tjabba"s};
    cout << "string: " << get<string>(payload) << " [" << payload.index() << "]\n";

    payload = 42;
    cout << "int    : " << get<int>(payload) << " [" << payload.index() << "]\n";

    payload = 3.1415926;
    cout << "double: " << get<double>(payload) << " [" << payload.index() << "]\n";
    cout << "double: " << get<1>(payload) << " [" << payload.index() << "]\n";
    try {
        auto v = get<int>(payload);
    } catch (bad_variant_access& x) { cout << "*** " << x.what() << endl; }

    return 0;
}
```

```
using-variant
/mnt/c/Users/jensr/Dropbox/Ribomati
string: tjabba [2]
int    : 42 [0]
double: 3.14159 [1]
double: 3.14159 [1]
*** Unexpected index

Process finished with exit code 0
```

407

std::any<T>

- A generic variable holding any type of payload

```
#include <iostream>
#include <any>
using namespace std;
using namespace std::literals;

int main() {
    any  whatever = 42U;
    if (whatever.has_value() && whatever.type() == typeid(unsigned)) {
        auto v = any_cast<unsigned>(whatever);
        cout << "unsigned: " << v << endl;
    }

    whatever = 3.1415926;
    if (whatever.type() == typeid(double)) {
        auto v = any_cast<double>(whatever);
        cout << "double: " << v << endl;
    }

    whatever = "tjolla hopp"s;
    if (whatever.type() == typeid(string)) {
        auto v = any_cast<string>(whatever);
        cout << "string: " << v << endl;
    }

    return 0;
}
```

```
using-any
/mnt/c/Users/jensr/Dropbox/Ribomation/
unsigned: 42
double: 3.14159
string: tjolla hopp

Process finished with exit code 0
```

408

Using any within a map

```
#include <iostream>
#include <unordered_map>
#include <any>

using namespace std;
using namespace std::literals;

int main() {
    unordered_map<string, any> stuff;
    stuff["int"s]    = 42;
    stuff["double"s] = 42 * 1234.56789;
    stuff["string"s] = "foobar strikes again"s;

    for (auto e : stuff) {
        cout << e.first << ": ";
        if (e.second.type() == typeid(int)) {
            cout << any_cast<int>(e.second);
        } else if (e.second.type() == typeid(double)) {
            cout << any_cast<double>(e.second);
        } else if (e.second.type() == typeid(string)) {
            cout << any_cast<string>(e.second);
        }
        cout << endl;
    }

    return 0;
}
```

```
any-map
/mnt/c/Users/jensr/Dropbox/Ribomation/
string: foobar strikes again
int: 42
double: 51851.9

Process finished with exit code 0
```

409

Rational numbers

- The class template std::ratio and associated templates provide compile-time rational arithmetic support. Each instantiation of this template exactly represents any finite rational number.

```
#include <iostream>
#include <ratio>

int main()
{
    typedef std::ratio<2, 3> two_third;
    typedef std::ratio<1, 6> one_sixth;

    typedef std::ratio_add<two_third, one_sixth> sum;
    std::cout << "2/3 + 1/6 = " << sum::num << '/' << sum::den << '\n';
}
```

Output:

```
2/3 + 1/6 = 5/6
```

410

Complex numbers

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <cmath>

int main()
{
    using namespace std::literals;
    std::cout << std::fixed << std::setprecision(1);

    std::complex<double> z1 = 1i * 1i;     // imaginary unit squared
    std::cout << "i * i = " << z1 << '\n';

    std::complex<double> z2 = std::pow(1i, 2); // imaginary unit squared
    std::cout << "pow(i, 2) = " << z2 << '\n';

    double PI = std::acos(-1);
    std::complex<double> z3 = std::exp(1i * PI); // Euler's formula
    std::cout << "exp(i, pi) = " << z3 << '\n';

    std::complex<double> z4 = 1. + 2i, z5 = 1. - 2i; // conjugates
    std::cout << "(1+2i)*(1-2i) = " << z4*z5 << '\n';
}
```

Output:

```
i * i = (-1.0,0.0)
pow(i, 2) = (-1.0,0.0)
exp(i, pi) = (-1.0,0.0)
(1+2i)*(1-2i) = (5.0,0.0)
```

411

Numeric arrays

- std::valarray is the class for representing and manipulating arrays of numeric values. It supports element-wise mathematical operations and various forms of generalized subscript operators, slicing and indirect access

```
#include <iostream>
#include <complex>
#include <valarray>

int main()
{
    const double pi = std::acos(-1);
    std::valarray<std::complex<double>> v = {
        {0, 0}, {0, pi/2}, {0, pi}, {0, 3*pi/2}, {0, 2*pi}
    };
    std::valarray<std::complex<double>> v2 = std::exp(v);
    for(auto n : v2) {
        std::cout << std::fixed << n << '\n';
    }
}
```

Output:

```
(1.000000,0.000000)
(0.000000,1.000000)
(-1.000000,0.000000)
(-0.000000,-1.000000)
(1.000000,-0.000000)
```

412

Class bitset<N>

- The class template bitset represents a fixed-size sequence of N bits. Bitsets can be manipulated by standard logic operators and converted to and from strings and integers

```cpp
#include <bitset>
#include <string>
#include <iostream>
#include <climits>

int main()
{
    // empty constructor
    std::bitset<8> b1; // [0,0,0,0,0,0,0,0]

    // unsigned long long constructor
    std::bitset<8> b2(42);          // [0,0,1,0,1,0,1,0]
    std::bitset<70> bl(ULLONG_MAX); // [0,0,0,0,0,0,1,1,1,...,1,1,1] in C++11
    std::bitset<8> bs(0xfff0);      // [1,1,1,1,0,0,0,0]

    // string constructor
    std::string bit_string = "110010";
    std::bitset<8> b3(bit_string);       // [0,0,1,1,0,0,1,0]
    std::bitset<8> b4(bit_string, 2);    // [0,0,0,0,0,0,1,0]
    std::bitset<8> b5(bit_string, 2, 3); // [0,0,0,0,0,0,0,1]

    // string constructor using custom zero/one digits
    std::string alpha_bit_string = "aBaaBBaB";
    std::bitset<8> b6(alpha_bit_string, 0, alpha_bit_string.size(),
                      'a', 'B');         // [0,1,0,0,1,1,0,1]

    // char* constructor using custom digits
    std::bitset<8> b7("XXXXYYYY", 8, 'X', 'Y'); // [0,0,0,0,1,1,1,1]

    std::cout << b1 << '\n' << b2 << '\n' << bl << '\n' << bs << '\n'
              << b3 << '\n' << b4 << '\n' << b5 << '\n' << b6 << '\n'
              << b7 << '\n';
}
```

```
00000000
00101010
0000001111111111111111111111111111111111111111111111111111111111111111
11110000
00110010
00000010
00000001
01001101
00001111
```

http://en.cppreference.com/w/cpp/utility/bitset/bitset

Fixed width integer types

- Integral types with well-defined bit sizes
  - Typically used as "data channels" for memory mapped I/O
- Info
  - http://en.cppreference.com/w/cpp/types/integer
- Include
  - <cstdint>

Type	Description
uint8_t **uint16_t** **uint32_t** **uint64_t** (optional)	unsigned integer type with width of exactly 8, 16, 32 and 64 bits respectively (provided only if the implementation directly supports the type) (typedef)
int8_t **int16_t** **int32_t** **int64_t** (optional)	signed integer type with width of exactly 8, 16, 32 and 64 bits respectively with no padding bits and using 2's complement for negative values (provided only if the implementation directly supports the type) (typedef)

```
uintmax_t: maximum width unsigned integer type
intmax_t : maximum width integer type
```

std::numeric_limits

- Provides a standardized way to query various properties of arithmetic types
- Info
  - http://en.cppreference.com/w/cpp/types/numeric_limits
- Include
  - <limits>

```
template<> class numeric_limits<bool>;
template<> class numeric_limits<char>;
template<> class numeric_limits<signed char>;
template<> class numeric_limits<unsigned char>;
template<> class numeric_limits<wchar_t>;
template<> class numeric_limits<char16_t>;   // C++11 feature
template<> class numeric_limits<char32_t>;   // C++11 feature
template<> class numeric_limits<short>;
template<> class numeric_limits<unsigned short>;
template<> class numeric_limits<int>;
template<> class numeric_limits<unsigned int>;
template<> class numeric_limits<long>;
template<> class numeric_limits<unsigned long>;
template<> class numeric_limits<long long>;
template<> class numeric_limits<unsigned long long>;
template<> class numeric_limits<float>;
template<> class numeric_limits<double>;
template<> class numeric_limits<long double>;
```

Member	Description
lowest()	Lowest finite value
min()	Smallest finite value
max()	Largest finite value
is_signed	True if not unsigned
is_integer	True if not floating-point

Selected subset of all members

415

CHAPTER SUMMARY

Helper Types

- There are many handy and versatile data-types in the standard library
  - any / optional / variant
  - ratio / complex / valarray
  - bitset / numeric_limits

416

EXERCISE

Contact card

- Define a class holding name and contact information
- Let all fields except name be represented as optional fields
- Add getters for all the fields
- Overload the left shift operator for printout
  - ostream& operator <<(ostream&, const Contact&)
  - Print out fields only having a value

417

Intentional Blank

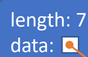
418

Using string_view Objects

419

Using string_view [C++17]

- Efficient usage of character sequences without memory allocations
  - It's a cheap operation to copy a string_view object
- A string_view object is a *read-only* view-port into a character array
  - Has a similar API as std::string, but without modifiers
- Has `"..."sv` literal suffix as well



`A horse, a horse. My kingdom for a horse!`

```
char buf[1024];
read(fd, buf, sizeof(buf));
string_view payload{21, 7};
//...
parseIt("simple string-view literal"sv);
```

420

Parsing a HTTP request 1st line, using string_view

```cpp
tuple<string_view, string_view, string_view>
HttpHandler::parseRequestLine(string_view line) {
    //GET<SP1>/file<SP2>HTTP/1.0
    const auto SPACE = " "sv;

    const auto space1 = line.find(SPACE, 0);
    if (space1 == string_view::npos) throw MalformedRequest{"HTTP Method"};

    const auto space2 = line.find(SPACE, space1 + 1);
    if (space2 == string_view::npos) throw MalformedRequest{"HTTP Path"};

    return make_tuple(
            line.substr(0, space1),
            line.substr(space1 + 1, space2 - space1 -1),
            line.substr(space2 + 1));
}
```

```cpp
Options         emptyOpts{};
vector<Route>   emptyRoutes{};
vector<Filter*> emptyFilters{};

TEST(requestParsing, parseRequestLine) {
    HttpHandler handler{emptyOpts, emptyRoutes, emptyFilters};

    auto [method, path, proto] = handler.parseRequestLine("GET /msg HTTP/1.0"sv);
    EXPECT_EQ(method, "GET"sv);
    EXPECT_EQ(path  , "/msg"sv);
    EXPECT_EQ(proto , "HTTP/1.0"sv);
}
```

```
Run: All in http-handler-test.cxx (1)
Tests passed: 1 of 1 test – 0 ms
Test Results                0 ms
  requestParsing            0 ms
    parseRequestLine        0 ms

Testing started at 15:10 ...
/mnt/c/Users/jensr/CLionProjects/res
Running main() from gtest_main.cc
Process finished with exit code 0
```

421

Handling one HTTP request

```cpp
void HttpHandler::run(int fromClientFd, int toClientFd) {
    ResponseImpl response;
    RequestImpl  request;
    const auto   payloadSize = options.max_request_size_in_bytes;
    auto         payload     = make_unique<char[]>(payloadSize);
    try {
        loadAndParseRequest(fromClientFd, request, payload.get(), payloadSize);
        auto route = findRoute(request);
        filters[0]->invoke(request, response, *route);
    } catch (RouteNotFound& err) {
        cerr << "*** Route not found " << err.what() << endl;
        response.status(404, err.what());
    } catch (MalformedRequest& err) {
        cerr << "*** Malformed REQ " << err.what() << endl;
        response.status(400, err.what());
    } //...

    sendResponse(toClientFd, response);
}
```

The one and only memory allocation. Contains the whole request, where all fragments are light-weight `string_view` objects. The char buffer is managed by a smart pointer.

```cpp
struct RequestImpl : public Request {
    string_view                                  method_;
    string_view                                  path_;
    string_view                                  protocol_;
    string_view                                  body_;
    string_view                                  query_;
    string_view                                  param_;
    unordered_map<string_view, string_view>      headers_;
    unordered_map<string, any>                   attributes_;
```

422

CHAPTER SUMMARY

Using string_view Objects

- The string_view class is an excellent companion to std::string
- Keep in mind that it's just a read-only viewport to an existing char array

423

EXERCISE

Parsing and memory efficiency

- Load the content of a file into a heap allocated text string (char*)
- Extract all words (sequence of letters) into string_view objects and put them into a std::map where the map-value is the word frequency
- Use a for-each loop to print out the words together with their frequencies

424

Smart Pointers

425

What is a smart pointer

- A local object holding an address to a target memory block (owner)
- When an owner of a resource goes out of scope (last '}', return, throw), it must dispose the resource or transfer the ownership of it to another owner
- Benefits
  - Reduces the risk for memory leakage
  - Automatic disposal of unreachable objects
- Realizations
  - Different pointer handling semantics

426

Simple (*incomplete*) implementation

```
template<typename T>
class Ptr {
    T* addr;
public:
    explicit Ptr(T* addr) : addr{addr} {}
    ~Ptr() { delete addr; }

    T&  operator *()  const {return *addr;}
    T*  operator ->() const {return addr;}

    Ptr()                = delete;
    Ptr(const Ptr<T>&) = delete;

    Ptr<T>&      operator=(const Ptr<T>&) = delete;
    const void* operator&() const           = delete;
    void*        operator new(size_t)       = delete;
};
```

```
void using_ints() {
    cout << "--- using ints ---\n";
    Ptr<int> p{new int{42}};
    cout << "*p = " << *p << endl;
    *p *= 10;
    cout << "*p = " << *p << endl;
}
```

```
void using_accounts() {
    cout << "--- using accounts ---\n";
    struct Account {
      private: int balance;
      public:  Account(int b) : balance{b} {}
      public:  int  getBalance() const {return balance;}
      public:  void updateBalance(int b) {balance += b;}
    };

    Ptr<Account> p{new Account{1500}};
    cout << "p->getBalance()   = " << p->getBalance() << endl;
    cout << "(*p).getBalance() = " << (*p).getBalance() << endl;
    p->updateBalance(-750);
    cout << "p->getBalance()   = " << p->getBalance() << endl;
}
```

```
/usr/bin/valgrind --tool=memcheck --x
--- using ints ---
*p = 42
*p = 420
--- using accounts ---
p->getBalance()   = 1500
(*p).getBalance() = 1500
p->getBalance()   = 750

Process finished with exit code 0
```

Automatic destruction of heap allocated memory block

427

No pointer to a smart pointer

```
template<typename T>
class Ptr {
    T* addr;
public:
    explicit Ptr(T* addr) : addr{addr} {}
    ~Ptr() { delete addr; }

    T&  operator *()  const {return *addr;}
    T*  operator ->() const {return addr;}

    Ptr()                = delete;
    Ptr(const Ptr<T>&) = delete;

    Ptr<T>&      operator=(const Ptr<T>&) = delete;
    const void* operator&() const           = delete;
    void*        operator new(size_t)       = delete;
};
```

```
//void* x = &p;
//error: use of deleted function 'const void* Ptr::operator&() const'

//void* y = new Ptr{new int{42}};
//error: use of deleted function 'static void* Ptr::operator new(size_t)'
```

428

To delete or not to delete, that's the question?

```
class Car {
  Ptr<Person>    owner;
 public:
  Car(Ptr<Person> p) : owner{p} {…}
  //...
};

void doSomethingWrong() {
  Ptr<Person>  bob = new Person{"Bob"};

  if (...) {
    Ptr<Car> volvo = new Car{bob};
    ...
  }

  cout << "p=" << bob->getName() << endl;
}
```

Oooops

A destructor, also run all destructors of its member objects as well. Hence, Bob got disposed!

429

Different smart pointer semantics

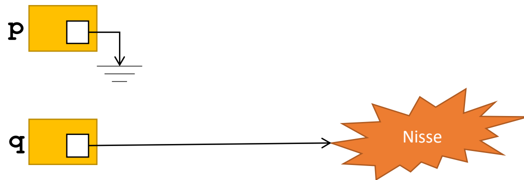
- How to handle copying and destruction
  - T::T(const T&)
  - T::~T()
  - T& T::operator =(const T&)
- Different smart pointer semantics realization
  - UniquePtr<T>
  - CopyPtr<T>
  - RefCountPtr<T>

430

UniquePtr<T>

- Manages a single reference
- Each copy/assignment transfers the ownership
- When the pointer object goes out of scope it disposes the memory block

```
UniquePtr<Person> p{ new Person{"Nisse"} };
UniquePtr<Person> q = p;
```

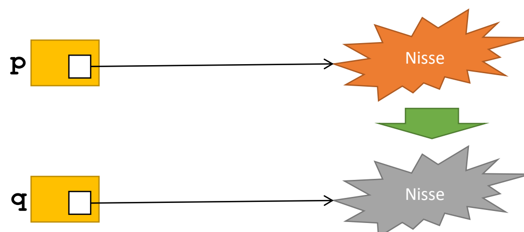


431

CopyPtr<T>

- Manages a single reference
- Each copy/assignment makes a copy of the target memory block
- When the pointer object goes out of scope it disposes the memory block

```
CopyPtr<Person> p{ new Person{"Nisse"} };
CopyPtr<Person> q = p;
```

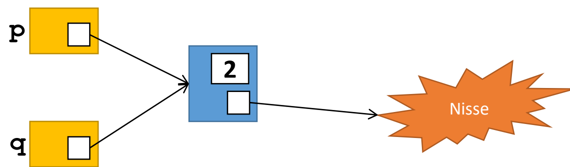


432

RefCountPtr<T>

- Manages multiple references
- Each copy/assignment increments a reference count
- When the pointer object goes out of scope it decrements the reference count
- When the reference count hits zero the memory block is disposed

```
RefCountPtr<Person> p{ new Person{"Nisse"} };
RefCountPtr<Person> q = p;
```



433

Smart pointers in Modern C++

- unique_ptr<T>
  - Exclusive ownership
- shared_ptr<T>
  - Reference counting
- weak_ptr<T>
  - Handles circular chains of shared_ptr<T>

434

Usage of unique_ptr<T>

```cpp
unique_ptr<Thing> mk() {
    unique_ptr<Thing> ptr{new Thing{42}};
    cout << "[mk]    count: " << Thing::count << endl;
    return ptr;
}

unique_ptr<Thing> use(unique_ptr<Thing> ptr) {
    cout << "[use]   count: " << Thing::count << endl;

    auto q = make_unique<Thing>(17);
    cout << "[use]   count: " << Thing::count << endl;
    ptr = move(q);

    return ptr;
}

int main(int, char**) {
    cout << "[first] count: " << Thing::count << endl;
    {
        unique_ptr<Thing> ptr = mk();
        ptr = use(move(ptr));
        cout << "[block] count: " << Thing::count << endl;
    }
    cout << "[last]  count: " << Thing::count << endl;
    return 0;
}
```

```cpp
struct Thing {
    static int count;
    const int  value;

    explicit Thing(int v) : value{v} {
        ++count;
        cout << "Thing(int=" << value << ") cnt=" << count << endl;
    }

    ~Thing() {
        --count;
        cout << "~Thing(" << value << ") cnt=" << count << endl;
    }

    Thing(const Thing&) = delete;
    Thing& operator=(const Thing&) = delete;
};
int Thing::count = 0;
```

using-unique_ptr

```
/home/jens/Courses/cxx/cxx-embedded/
[first] count: 0
Thing(int=42) cnt=1
[mk]    count: 1
[use]   count: 1
Thing(int=17) cnt=2
[use]   count: 2
~Thing(42) cnt=1
[block] count: 1
~Thing(17) cnt=0
[last]  count: 0

Process finished with exit code 0
```

435

Both smart pointers types can take an custom deleter

```cpp
int main(int, char**) {
    cout << "[main] enter\n";
    {
        struct Deleter {
            void operator()(int* addr) {
                cout << "custom deleter\n"; delete addr;
            }
        };
        unique_ptr<int, Deleter> p{new int{42}, Deleter{}};
        cout << "*p = " << *p << endl;
    }
    cout << "[main] exit\n";
    return 0;
}
```

custom-deleter

```
/home/jens/Courses/cxx/cxx-embedded/
[main] enter
*p = 42
custom deleter
[main] exit

Process finished with exit code 0
```

436

API of unique_ptr<T>

Member functions

(constructor)	constructs a new unique_ptr (public member function)
(destructor)	destructs the managed object if such is present (public member function)
operator=	assigns the unique_ptr (public member function)
Modifiers	
release	returns a pointer to the managed object and releases the ownership (public member function)
reset	replaces the managed object (public member function)
swap	swaps the managed objects (public member function)
Observers	
get	returns a pointer to the managed object (public member function)
get_deleter	returns the deleter that is used for destruction of the managed object (public member function)
operator bool	checks if there is associated managed object (public member function)
Single-object version, unique_ptr<T>	
operator* operator->	dereferences pointer to the managed object (public member function)
Array version, unique_ptr<T[]>	
operator[]	provides indexed access to the managed array (public member function)

Non-member functions

make_unique (C++14)	creates a unique pointer that manages a new object (function template)
operator== operator!= operator< operator<= operator> operator>=	compares to another unique_ptr or with nullptr (function template)
std::swap(std::unique_ptr) (C++11)	specializes the std::swap algorithm (function template)

437

API of shared_ptr<T>

Member functions

(constructor)	constructs new shared_ptr (public member function)
(destructor)	destructs the owned object if no more shared_ptrs link to it (public member function)
operator=	assigns the shared_ptr (public member function)
Modifiers	
reset	replaces the managed object (public member function)
swap	swaps the managed objects (public member function)
Observers	
get	returns a pointer to the managed object (public member function)
operator* operator->	dereferences pointer to the managed object (public member function)
use_count	returns the number of shared_ptr objects referring to the same managed object (public member function)
unique	checks whether the managed object is managed only by the current shared_ptr instance (public member function)
operator bool	checks if there is associated managed object (public member function)
owner_before	provides owner-based ordering of shared pointers (public member function)

Non-member functions

make_shared	creates a shared pointer that manages a new object (function template)
allocate_shared	creates a shared pointer that manages a new object allocated using an allocator (function template)
static_pointer_cast dynamic_pointer_cast const_pointer_cast	applies static_cast, dynamic_cast or const_cast to the type of the managed object (function template)
get_deleter	returns the deleter of specified type, if owned (function template)
operator== operator!= operator< operator<= operator> operator>=	compares with another shared_ptr or with nullptr (function template)
operator<<	outputs the value of the managed pointer to an output stream (function template)
std::swap(std::shared_ptr) (C++11)	specializes the std::swap algorithm (function template)

438

CHAPTER SUMMARY

Smart Pointers

- Smart pointers provides a limited form of garbage collection without any overhead
- Useful together with exceptions and threads
- Principles
  - UniquePtr / CopyPtr / RefcntPtr
- C++11
  - unique_ptr / --- / shared_ptr
- Can have custom deleter
  - If you use a custom allocator, e.g. to non-heap memory, then use a custom deleter as well

439

EXERCISE

Play with shared_ptr<T>

- Create an Account class
  - Make it non-copyable (no copy constructor & assignment operator)
  - Augment it with an instance count (static int managed in constructor(s) & destructor)
- Create one account object managed by a shared_ptr<T>
- Pass it around in and out of functions
  - Ensure that you at some point have at least three users (smart pointers to the account)
  - Hint: use_count()

440

File Systems

441

File systems support

- Representation of directory entry and path
- A path to an existing entity provides permissions, meta-data and size
- Possible to manipulate files and directories
  - Copy files, create/remove directories and links, resize files
- Need to link with fs lib (*GCC version 8.0+*)

```
g++ -std=c++17 -lstdc++fs app.cxx
```

442

FS API

Classes

Defined in header <filesystem>
Defined in namespace filesystem

path (C++17)	represents a path (class)
filesystem_error (C++17)	an exception thrown on file system errors (class)
directory_entry (C++17)	a directory entry (class)
directory_iterator (C++17)	an iterator to the contents of the directory (class)
recursive_directory_iterator (C++17)	an iterator to the contents of a directory and its subdirectories (class)
file_status (C++17)	represents file type and permissions (class)
space_info (C++17)	information about free and available space on the filesystem (class)
file_type (C++17)	the type of a file (enum)
perms (C++17)	identifies file system permissions (enum)
perm_options (C++17)	specifies semantics of permissions operations (enum)
copy_options (C++17)	specifies semantics of copy operations (enum)
directory_options (C++17)	options for iterating directory contents (enum)
file_time_type (C++17)	represents file time values (typedef)

Non-member functions

absolute (C++17)	composes an absolute path (function)
canonical weakly_canonical (C++17)	composes a canonical path (function)
relative proximate (C++17)	composes a relative path (function)
copy (C++17)	copies files or directories (function)
copy_file (C++17)	copies file contents (function)
copy_symlink (C++17)	copies a symbolic link (function)
create_directory (C++17) create_directories (C++17)	creates new directory (function)
create_hard_link (C++17)	creates a hard link (function)
create_symlink (C++17) create_directory_symlink (C++17)	creates a symbolic link (function)
current_path (C++17)	returns or sets the current working directory (function)
exists (C++17)	checks whether path refers to existing file system object (function)
equivalent (C++17)	checks whether two paths refer to the same file system object (function)
file_size (C++17)	returns the size of a file (function)
hard_link_count (C++17)	returns the number of hard links referring to the specific file (function)
last_write_time (C++17)	gets or sets the time of the last data modification (function)
permissions (C++17)	modifies file access permissions (function)
read_symlink (C++17)	obtains the target of a symbolic link (function)
remove (C++17) remove_all (C++17)	removes a file or empty directory removes a file or directory and all its contents, recursively (function)
rename (C++17)	moves or renames a file or directory (function)
resize_file (C++17)	changes the size of a regular file by truncation or zero-fill (function)
space (C++17)	determines available free space on the file system (function)
status (C++17) symlink_status (C++17)	determines file attributes determines file attributes, checking the symlink target (function)
temp_directory_path (C++17)	returns a directory suitable for temporary files (function)

443

std::filesystem path

```
#include <filesystem>
namespace fs = std::filesystem;

int main() {
  auto tmpFile      = fs::current_path() / "dummy.txt"s;

  ofstream f{tmpFile};

  auto lastModified = fs::last_write_time(tmpFile);
  auto ts           = system_clock::to_time_t(lastModified);
  cout << "modif: " << asctime(localtime(&ts)) << endl;

  auto fsSpace      = fs::space("/home"s);
  cout << "/home: " << fsSpace.capacity <<
       <<" avail: " << fsSpace.available << endl;

  return 0;
}
```

444

Recursive directory traversal

```cpp
#include <filesystem>
namespace fs = std::filesystem;

int main(int argc, char** argv) {
    const auto baseDir = fs::path{argc == 1 ? "." : argv[1]};
    if (!fs::is_directory(baseDir)) {
        throw invalid_argument{"not a directory: "s + baseDir.string()};
    }

    cout << "Base dir: " << baseDir << endl;
    for (auto& entry : fs::recursive_directory_iterator(baseDir)) {
        auto filename = entry.path().string();
        if (fs::is_regular_file(entry)) {
            auto lastModif = fs::last_write_time(entry);
            auto ts        = decltype(lastModif)::clock::to_time_t(lastModif);
            cout << put_time(localtime(&ts), "%F %T") << "\t"
                 << fs::file_size(entry) << " bytes\t"
                 << filename << endl;
        } else if (fs::is_directory(entry)) {
            cout << filename << "/" << endl;
        } else {
            cout << filename << " ??" << endl;
        }
    }

    return 0;
}
```

```
2018-09-17 09:29:09     804 bytes       ./cmake-build-debug/CMakeFiles/project-stats.dir/DependInfo.cmake
2018-09-17 09:29:09     233 bytes       ./cmake-build-debug/CMakeFiles/project-stats.dir/flags.make
2018-09-17 09:29:09     98 bytes        ./cmake-build-debug/CMakeFiles/project-stats.dir/link.txt
2018-09-17 09:29:09     43 bytes        ./cmake-build-debug/CMakeFiles/project-stats.dir/progress.make
2018-09-17 10:24:37     620208 bytes    ./cmake-build-debug/CMakeFiles/project-stats.dir/project-stats.cxx.o
2018-09-17 09:29:09     516 bytes       ./cmake-build-debug/CMakeFiles/TargetDirectories.txt
2018-09-17 08:59:34     1719 bytes      ./cmake-build-debug/cmake_install.cmake
2018-09-17 09:29:09     8276 bytes      ./cmake-build-debug/file_systems.cbp
2018-09-17 09:29:09     5579 bytes      ./cmake-build-debug/Makefile
2018-09-17 10:24:37     547008 bytes    ./cmake-build-debug/project-stats
2018-09-17 09:29:08     242 bytes       ./CMakeLists.txt
2018-09-17 10:41:08     1102 bytes      ./project-stats.cxx
C++>
```

445

CHAPTER SUMMARY

File Systems

- The FS addition makes it easy to work with file entities

446

EXERCISE

Dir count

- Write a program that takes a directory name on the command-line
  - Verify the name refers to a directory or set it to the current, if absent
- Open all text files and count the number of lines, words and chars
  - Define a set of extensions of text files, to know if a file is text or binary
- Print out the counts for each file plus a summary

447

Intentional Blank

448

Date / Time / Clock Handling

449

The chrono sub-system

- Include
  - <chrono>
  - using namespace std::chrono
- Primary data types
  - duration
  - time_point
  - clock
  - time unit literals

450

Duration

- Combination of:
  - Count, measured as the number of ticks, since a reference point in time
  - Fraction, represents the unit in seconds as a rational number

std::chrono::duration

Defined in header <chrono>

```
template<
    class Rep,
    class Period = std::ratio<1>
> class duration;
```

(since C++11)

Class template std::chrono::duration represents a time interval.

```
duration<int, ratio<1>>         fiveSeconds{5};
cout << "5s:" << fiveSeconds.count() << endl;

duration<float, ratio<1,1000>>  tenMilliSecs{10};
cout << "10ms:" << tenMilliSecs.count() << endl;
```

```
using WorkDay = duration<int, ratio<8*60*60>>;
WorkDay  week{5};
cout << "week:" << week.count() << endl;
```

std::chrono::duration_cast

```
template <class ToDuration, class Rep, class Period>
constexpr ToDuration duration_cast(const duration<Rep,Period>& d);
```

(since C++11)

Converts a std::chrono::duration to a duration of different type ToDuration.

451

Time unit

- Predefined time units, expressed as signed integer durations

Type	Definition
std::chrono::nanoseconds	duration</*signed integer type of at least 64 bits*/, std::nano>
std::chrono::microseconds	duration</*signed integer type of at least 55 bits*/, std::micro>
std::chrono::milliseconds	duration</*signed integer type of at least 45 bits*/, std::milli>
std::chrono::seconds	duration</*signed integer type of at least 35 bits*/>
std::chrono::minutes	duration</*signed integer type of at least 29 bits*/, std::ratio<60>>
std::chrono::hours	duration</*signed integer type of at least 23 bits*/, std::ratio<3600>>

452

Clock

Clocks

A clock consists of a starting point (or epoch) and a tick rate. For example, a clock may have an epoch of January 1, 1970 and tick every second. C++ defines three clock types:

Defined in header <chrono>
Defined in namespace std::chrono

system_clock (C++11)	wall clock time from the system-wide realtime clock (class)
steady_clock (C++11)	monotonic clock that will never be adjusted (class)
high_resolution_clock (C++11)	the clock with the shortest tick period available (class)

```
using any_clock = ...;

auto start   = any_clock::now();
//...perform something...
auto end     = any_clock::now();
auto elapsed = duration_cast<microseconds>(end - start);
```

453

Computing elapsed time

```
#include <iostream>
#include <string>
#include <functional>
#include <tuple>
#include <chrono>

using namespace std;
using namespace std::chrono;

template<typename RetType, typename ArgType>
tuple<RetType, long>
elapsed(function<RetType(ArgType)> f, ArgType arg) {
    auto    start  = high_resolution_clock::now();
    RetType result = f(arg);
    auto    end    = high_resolution_clock::now();
    return make_tuple(result, duration_cast<nanoseconds>(end - start).count());
}

long SUM(int n) { return n * (n + 1) / 2; }

int main() {
    auto N          = 10000;
    auto [sum, ns] = elapsed<long, int>(SUM, N);
    cout << "sum[1.." << N << "] = " << sum << " (elapsed " << ns << " ns)" << endl;
    return 0;
}
```

```
C++> ./cmake-build-debug/elapsed-time
sum[1..10000] = 50005000 (elapsed 865 ns)
C++>
```

454

Time points

- A time point is a duration of time that has passed since the epoch of specific clock

```
1  #include <iostream>
2  #include <iomanip>
3  #include <ctime>
4  #include <chrono>
5  using namespace std;
6  using namespace std::chrono;
7
8  int main() {
9      auto today = system_clock::now();
10     auto t     = system_clock::to_time_t(today);
11     auto tm    = *std::localtime( &t );
12     cout << "Default: " << std::put_time(&tm, "%c %Z") << '\n';
13
14     cout.imbue(std::locale("sv_SE.utf8"));
15     cout << "Swedish: " << std::put_time(&tm, "%c %Z") << '\n';
16     cout << "ISO8601: " << std::put_time(&tm, "%F %T") << '\n';
17 }
18
```

Output:

```
Default: Sat Oct 10 10:07:31 2015 UTC
Swedish: lör 10 okt 2015 10:07:31 UTC
ISO8601: 2015-10-10 10:07:31
```

455

Time unit suffixes

Literals

Defined in inline namespace std::literals::chrono_literals

operator""h (C++14)	A **std::chrono::duration** literal representing hours (function)
operator""min (C++14)	A **std::chrono::duration** literal representing minutes (function)
operator""s (C++14)	A **std::chrono::duration** literal representing seconds (function)
operator""ms (C++14)	A **std::chrono::duration** literal representing milliseconds (function)
operator""us (C++14)	A **std::chrono::duration** literal representing microseconds (function)
operator""ns (C++14)	A **std::chrono::duration** literal representing nanoseconds (function)

```
using namespace std::chrono_literals;

auto boiled_eggs = 10min;
auto working_day = 8h;
auto quarter     = 0.25h;
```

456

Using unit suffixes

```
C++> cat duration-math.cxx
#include <iostream>
#include <chrono>

using namespace std;
using namespace chrono;
using namespace chrono_literals;

int main(int, char**) {
    auto weekend_run = 1h + 12min + 17s + 125ms;
    cout << "last w/e run took " << weekend_run.count() << " seconds" << endl;

    cout << "which is "
         << duration_cast<hours>(weekend_run).count()                    << "h + "
         << duration_cast<minutes>(weekend_run % hours(1)).count()       << "m + "
         << duration_cast<seconds>(weekend_run % minutes(1)).count()     << "s + "
         << duration_cast<milliseconds>(weekend_run % seconds(1)).count() << "ms"
         << endl;

    return 0;
}
```

duration-math

```
/home/jens/Courses/cxx/cxx-embedded/
last w/e run took 4337125 seconds
which is 1h + 12m + 17s + 125ms

Process finished with exit code 0
```

457

CHAPTER SUMMARY

Time & Duration

- Duration
  - duration<int, ratio<1,1000>>
- Clock
  - system_clock
  - steady_clock
  - high_resolution_clock
- Time unit suffixes
  - 12h + 5min + 10s

458

EXERCISE

Number of seconds

- How many seconds are
  - 3 hours, 7 minutes and 49 seconds
- Hint
  - Use time unit literals

459

Intentional Blank

460

461

462

Thread support in C++11

Features

- Thread class
- Asynchronous tasks
- Mutex and Lock
- Condition variable
- Future and Promise

Compilation & Linking

- Includes
  - <thread>
  - <mutex>
  - <condition_variable>
  - <future>
  - . . .
- Link with
  - -pthread

N.B. this is a different library, i.e. not -lpthread

463

API of thread

Member functions

(constructor)	constructs new thread object (public member function)
(destructor)	destructs the thread object, underlying thread must be joined or detached (public member function)
operator=	moves the thread object (public member function)
Observers	
joinable	checks whether the thread is joinable, i.e. potentially running in parallel context (public member function)
get_id	returns the *id* of the thread (public member function)
native_handle	returns the underlying implementation-defined thread handle (public member function)
hardware_concurrency [static]	returns the number of concurrent threads supported by the implementation (public static member function)
Operations	
join	waits for a thread to finish its execution (public member function)
detach	permits the thread to execute independently from the thread handle (public member function)
swap	swaps two thread objects (public member function)

Non-member functions

std::swap (std::thread) (C++11)	specializes the std::swap algorithm (function template)

Functions managing the current thread

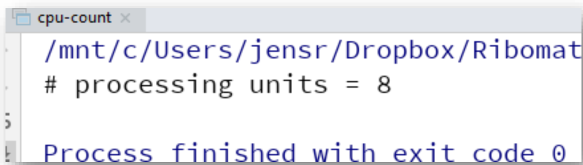
Defined in namespace this_thread

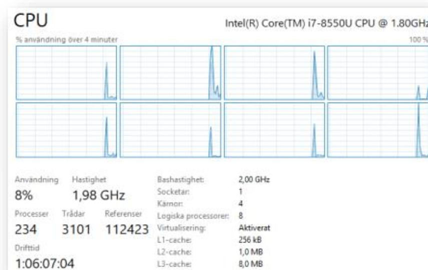
yield (C++11)	suggests that the implementation reschedule execution of threads (function)
get_id (C++11)	returns the thread id of the current thread (function)
sleep_for (C++11)	stops the execution of the current thread for a specified time duration (function)
sleep_until (C++11)	stops the execution of the current thread until a specified time point (function)

464

Number of (virtual) CPUs (processing units)

```
#include <iostream>
#include <thread>
using namespace std;

int main() {
    cout << "# processing units = " << thread::hardware_concurrency() << endl;
    return 0;
}
```





465

Thread life-cycle methods

- The constructor launches a new thread
  - thread thr(*expr*);
- Waiting for termination
  - thr.join();
- If a thread is not joined its destructor invokes std::terminate()
- Possible to detach a thread, which means it do not need to be joined
  - thr.detach();

466

Several ways to create a thread

- Using an ordinary function
  - void run() {…}; &run
- Using a class having a function call operator '()'
  - struct Run { void operator()() {…} }; Run{}
- Using a method pointer and an object
  - struct T { void run() {…} }; &T::run
- Using a lambda expression
  - [](){…}

467

Ordinary function

```
void     run( params );
thread   thr(run, args);
```

thread-by-ordinary-function.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <thread>
5  using namespace std;
6
7  string  operator *(const string& s, int n) {
8      string  result;
9      for (int k=0; k<n; ++k) result += s;
10     return result;
11 }
12
13 void run(int id, int n) {
14     string  msg = string("    ") * id + to_string(id) + "\n";
15     for (int k=0; k<n; ++k) cout << msg << flush;
16 }
17
18 int main(int numArgs, char* args[]) {
19     int numThreads = numArgs > 1 ? stoi(args[1]) : 5;
20     int numMsgs    = numArgs > 2 ? stoi(args[2]) : 100;
21     vector<thread>  threads;
22
23     for (int k=0; k<numThreads; ++k) threads.push_back( thread(run, k+1, numMsgs) );
24     for (auto& t : threads) t.join();
25 }
26
```

```
[jens@vbox3 std-threads]$ g++ --std=c++11 thread-by-ordinary-function.cpp -o thread-by-ordinary-function -pthread
[jens@vbox3 std-threads]$ ./thread-by-ordinary-function 5 3
    1
                4
                4
                4
    1
    1
            3
                    5
                    5
                    5
            3
            3
        2
        2
        2
[jens@vbox3 std-threads]$
```

468

Function call operator

```
class MyClass {
    args
  public:
    MyClass(params) : init-args {}
    void operator()() {. . .}
};
thread thr( MyClass(args) );
```

thread-by-function-call-operator.cpp × thread-by-ordinary-function.cpp ×

```
13 class Runner {
14     int  id, n;
15   public:
16     Runner (int _id, int _n) : id(_id), n(_n) {}
17     void  operator()() {
18         string  msg = string("    ") * id + to_string(id) + "\n";
19         for (int k=0; k<n; ++k) cout << msg << flush;
20     }
21 };
22
23 int main(int numArgs, char* args[]) {
24     int numThreads = numArgs > 1 ? stoi(args[1]) : 5;
25     int numMsgs    = numArgs > 2 ? stoi(args[2]) : 100;
26     vector<thread>  threads;
27
28     for (int k=0; k<numThreads; ++k) threads.push_back( thread( Runner(k+1, numMsgs) ) );
29     for (auto& t : threads) t.join();
30 }
31
```

469

Method pointer

```
class MyClass {
    args
  public:
    MyClass(params) : init-args {}
    void run() {. . .}
};
MyClass* ptr = new MyClass(args);
thread thr( &MyClass::run, ptr );
delete ptr;
```

thread-by-method-pointer.cpp × thread-by-function-call-operator.cpp ×

```
13 class Runner {
14     int  id, n;
15   public:
16     Runner (int _id, int _n) : id(_id), n(_n) {}
17     void  run() {
18         string  msg = string("    ") * id + to_string(id) + "\n";
19         for (int k=0; k<n; ++k) cout << msg << flush;
20     }
21 };
22
23 int main(int numArgs, char* args[]) {
24     int numThreads = numArgs > 1 ? stoi(args[1]) : 5;
25     int numMsgs    = numArgs > 2 ? stoi(args[2]) : 100;
26     vector<Runner>  runners;
27     vector<thread>  threads;
28
29     for (int k=0; k<numThreads; ++k) runners.push_back( Runner(k+1, numMsgs) );
30     for (auto& r : runners) threads.push_back( thread( &Runner::run, &r ) );
31     for (auto& t : threads) t.join();
32 }
33
```

470

Lambda expression

```
thread  thr([](params){. . .}, args);
```

thread-by-lambda.cpp × | thread-by-method-pointer.cpp × | thread-by-function-call-operator.cpp ×

```
12
13 int main(int numArgs, char* args[]) {
14     int numThreads = numArgs > 1 ? stoi(args[1]) : 5;
15     int numMsgs    = numArgs > 2 ? stoi(args[2]) : 100;
16     vector<thread>  threads;
17
18     auto run = [](int id, int n){
19         string  msg = string("   ") * id + to_string(id) + "\n";
20         for (int k=0; k<n; ++k) cout << msg << flush;
21     };
22     for (int k=0; k<numThreads; ++k) threads.push_back( thread(run, k+1, numMsgs) );
23     for (auto& t : threads) t.join();
24 }
25
```

```
[jens@vbox3 std-threads]$ g++ --std=c++11 -pthread thread-by-lambda.cpp -o thread-by-lambda
[jens@vbox3 std-threads]$ ./thread-by-lambda 5 3
                5
                5
                5
      2
      2
      2
    1
    1
    1
              4
              4
              4
            3
            3
            3
[jens@vbox3 std-threads]$
```

471

Sleeping

```
int main() {
    auto body = [](auto name, auto duration) {
        do {
            auto          ts = system_clock::to_time_t(system_clock::now());
            ostringstream buf;
            buf << name << ": " << put_time(localtime(&ts), "%T") << "\n";
            cout << buf.str() << flush;

            this_thread::sleep_for(duration);
        } while (true);
    };

    thread t1(body, "T1"s, 3s);
    thread t2(body, "    T2"s, 750ms);
    thread t3(body, "        T3"s, 0.1min);

    t1.join(); t2.join(); t3.join();
    return 0;
}
```

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <thread>
using namespace std;
using namespace std::literals;
using namespace std::chrono;
using namespace std::chrono_literals;
```

clocks ×

```
/mnt/c/Users/jensr/Drop
        T3: 17:23:20
T1: 17:23:20
    T2: 17:23:20
    T2: 17:23:20
    T2: 17:23:21
    T2: 17:23:22
T1: 17:23:23
    T2: 17:23:23
    T2: 17:23:23
    T2: 17:23:24
    T2: 17:23:25
T1: 17:23:26
        T3: 17:23:26
    T2: 17:23:26
    T2: 17:23:26
    T2: 17:23:27
    T2: 17:23:28
T1: 17:23:29
    T2: 17:23:29
    T2: 17:23:29
    T2: 17:23:30
    T2: 17:23:31
        T3: 17:23:32
```

472

CHAPTER SUMMARY

Threading Support in C++11

- #include <thread>
- Compile with -pthread
- Launch a thread as a local variable, then join or transfer the ownership

473

EXERCISE

Hello C++11 Thread

- Implement the hello threads program using C++11 threads and lambdas

474

C++11 Synchronization

- ✓ mutex, recursive_mutex, timed_mutex
- ✓ lock_guard
- ✓ multi-lock
- ✓ class condition_variable

475

Mutex locks

- Include
  - <mutex>
- Types
  - mutex
  - recursive_mutex
- Operations
  - lock, unlock
  - try_lock

```
mutex  m;
m.lock();
//...
m.unlock();
```

```
class Account {
  recursive_mutex  m;
  int balance = 0;
 public:
  int get() {
    m.lock();
    int value = balance;
    m.unlock();
    return value;
  }

  void update(int amount) {
    m.lock();
    balance = get() + amount;
    m.unlock();
  }
};
```

476

The 'Bank' problem

threads-1.cpp ✖ bank.cpp ✖

```
 1 #include <iostream>
 2 #include <sstream>
 3 #include <string>
 4 #include <vector>
 5 #include <thread>
 6 #include <mutex>
 7 using namespace std;
 8
 9 class Account {
10     int     balance;
11     mutex   exclusive;
12   public:
13     Account(int n=0) : balance(n) {}
14     void  update(int value) {
15         lock_guard<mutex>  s(exclusive);
16         balance += value;
17     }
18     int  getBalance() const { return balance; }
19 };
20
21 void  Updater(int id, Account* acc, int n) {
22     for (int k=0; k<n; ++k) acc->update(+100);
23     for (int k=0; k<n; ++k) acc->update(-100);
24 }
```

```
26 int main(int n, char* args[]) {
27     int             numUpdaters     = (n >= 1 ? stoi(args[1]) : 5);
28     int             numTransactions = (n >= 2 ? stoi(args[2]) : 10000);
29     int             initBalance     = (n >= 3 ? stoi(args[3]) : 0);
30     Account         theAccount(initBalance);
31     cout << "Running with " << numUpdaters << " threads and "
32          << numTransactions << " updates. Initial balance="
33          << theAccount.getBalance() << endl;
34
35     vector<thread>  updaters;
36     for (int id=1; id<=numUpdaters; ++id)
37         updaters.push_back( thread(Updater, id, &theAccount, numTransactions) );
38     for (auto& t : updaters) t.join();
39
40     cout << "Final balance=" << theAccount.getBalance() << endl;
41     return 0;
42 }
```

jens@vbox4: ~/Documents/c++11

```
jens@vbox4:~/Documents/c++11$ g++ -std=c++0x -Wall -pthread bank.cpp -o bank
jens@vbox4:~/Documents/c++11$ ./bank 25 1000000 42
Running with 25 threads and 1000000 updates. Initial balance=42
Final balance=42
jens@vbox4:~/Documents/c++11$
```

477

Mutex lock with timeout

- Include
  - <mutex>
- Types
  - timed_mutex
  - recursive_timed_mutex
- Operations
  - try_lock_for
  - try_lock_until

```
using namespace std::chrono_literals
timed_mutex  m;

if (m.try_lock_for(5s) == false)
    throw Timeout();
. . .
m.unlock();
```

478

Automatic unlocking using a lock guard

```
class Account {
  recursive_mutex  m;
  int balance = 0;

 public:
  int get() {
    lock_guard<recursive_mutex>  g{m};
    return balance;
  }

  void update(int amount) {
    lock_guard<recursive_mutex>  g{m};
    balance = get() + amount;
  }
};
```

479

Deadlock avoiding multi-locking

```
struct Account {
    mutex  lck;
    int    balance = 0;
};

void  transfer(Account& from, Account& to, int amount) {
    scoped_lock  g{from.lck, to.lck};
    from.balance -= amount;
    to.balance   += amount;
}

int main() {
    Account  my(500), your(100);
    thread   t1(transfer, my  , your, 300);
    thread   t2(transfer, your, my  , 100);
    t1.join(); t2.join();
    return 0;
}
```

480

Deferred locking

- General-purpose mutex ownership wrapper allowing deferred locking, time-constrained attempts at locking, recursive locking, transfer of lock ownership, and use with condition variables
- unique_lock<*mutexType*>
- shared_lock<*mutexType*>

481

Read-Write lock

- Include
  - <shared_mutex>
- Lock for reading guard
  - shared_lock
- Lock for writing guard
  - unique_lock

```
class StockTicker {
  const string  name;
  double        value = 0;
  shared_mutex  rwl;

 public:
  Stock(const string& name) : name(name) {}

  double get() {
    shared_lock<shared_mutex>  g{rwl};
    return value;
  }

  void update(double delta) {
    unique_lock<shared_mutex>  g{rwl};
    value += delta;
  }
};
```

482

C++11 Condition variables

- Include
  - <condition_variable>
- Types
  - condition_variable (*used with unique_lock*)
  - condition_variable_any (*used with any type of lock*)
- Operations
  - wait
  - wait_for / wait_until
  - notify_one / notify_all
- Constraints
  - Must be within a mutex lock/unlock context
  - The mutex must be managed by a unique_lock<T> gurad

483

Usage of a condition variable

```
template<typename T>
class MessageQueue {
  mutex                 lck;
  condition_variable    notEmpty;
  queue<T>              inbox;

 public:
  void put(T x) {
    unique_lock<mutex>  g{lck};
    inbox.push(x);
    notEmpty.notify_all();
  }

  T  get() {
    unique_lock<mutex>  g{lck};
    notEmpty.wait(g, []{ return !inbox.empty(); });
    T x = inbox.front();  inbox.pop();
    return x;
  }
}
```

484

CHAPTER SUMMARY

C++11 Synchronization

- C++11 provide an extensive list of synchronization operations
- Different mutex semantics
- Conditions using lambdas

485

EXERCISE

Multi-account Bank using C++11/14

- Create N account objects
  - Let the initial balance be zero
- Create U update threads
  - Loop T number of times and perform in each loop
  - Choose two (random) account objects and transfer the amount A from the first to the second account
    - Hint: use the multi-lock feature
- Print out the sum of all account balances after all updaters have terminated
  - What is the expected value?

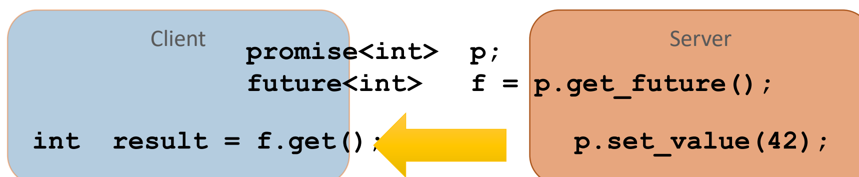
486

Asynchronous Tasks

487

C++11: Promise for the Future

- A *Promise* is the server-side part and a *Future* is the client-side part of a rendezvous connection between two threads
- Makes it easy to implement a whole range of communication patterns between senders and receivers



488

API: promise / future

Getting the result	
get_future	returns a future associated with the promised result (public member function)
Setting the result	
set_value	sets the result to specific value (public member function)
set_value_at_thread_exit	sets the result to specific value while delivering the notification only at thread exit (public member function)
set_exception	sets the result to indicate an exception (public member function)
set_exception_at_thread_exit	sets the result to indicate an exception while delivering the notification only at thread exit (public member function)

Getting the result	
get	returns the result (public member function)
State	
valid	checks if the future has a shared state (public member function)
wait	waits for the result to become available (public member function)
wait_for	waits for the result, returns if it is not available for the specified timeout duration (public member function)
wait_until	waits for the result, returns if it is not available until specified time point has been reached (public member function)

489

Implementing rendezvous using a promise/future

```
template<typename RequestType, typename ResponseType>
struct Message {
    RequestType              requestValue;
    promise<ResponseType>    response;
    Message(RequestType x)  : requestValue{x} {}
};

MessageQueue<Message*>  inputQueue;
```

```
ResponseType businessOperation(RequestType x) {
    Message  message{x};
    inputQueue.put(&message);
    return message.response.get_future().get();
}
```

```
Message*  msg = inputQueue.get();
RequestType  value = msg->requestValue;
//. . .result. . .
msg->response.set_value(result);
```

490

Asynchronous tasks

- Spawn a task executing a lambda expression asynchronously
  - `auto result = async(policy, [](){ expr } )`
- Launch policies
  - launch::async
    - A new thread is launched to execute the task asynchronously.
  - launch::deferred
    - The task is executed on the calling thread the first time its result is requested by a first call to a non-timed wait function on the std::future that async returned to the caller. Then the function is invoked.
- N.B.)
  - There is currently no support for thread pools. However, eventually there will be
    - http://chriskohlhoff.github.io/executors/
    - https://github.com/chriskohlhoff/executors

491

Asynchronous Fibonacci

```
11 long fibonacci(int n) {
12     return n <= 2 ? 1 : fibonacci(n-1) + fibonacci(n-2);
13 }
14
15 tuple<long,int> sequential(int n1, int n2, int n3) {
16     auto start  = system_clock::now();
17     long result = fibonacci(n1) + fibonacci(n2) + fibonacci(n3);
18     auto end    = system_clock::now();
19     int elapsed = duration_cast<milliseconds>(end - start).count();
20     return make_tuple(result, elapsed);
21 }
22
23 tuple<long,int> parallel(int n1, int n2, int n3) {
24     auto start  = system_clock::now();
25     auto r1     = async(launch::async, fibonacci, n1);
26     auto r2     = async(launch::async, fibonacci, n2);
27     auto r3     = async(launch::async, fibonacci, n3);
28     long result = r1.get() + r2.get() + r3.get();
29     auto end    = system_clock::now();
30     int elapsed = duration_cast<milliseconds>(end - start).count();
31     return make_tuple(result, elapsed);
32 }
```

492

Asynchronous task

```
34 int main(int numArgs, char* args[]) {
35     int n1 = numArgs > 1 ? stoi(args[1]) : 35;
36     int n2 = numArgs > 2 ? stoi(args[2]) : 35;
37     int n3 = numArgs > 3 ? stoi(args[3]) : 35;
38
39     cout << "Computing: fib("<<n1<<") + fib("<<n2<<") + fib("<<n3<<")"<<endl;
40     auto par = parallel(n1, n2, n3);
41     cout << "PAR: result=" << get<0>(par) << " (elapsed " << get<1>(par) << " ms)" << endl;
42     auto seq = sequential(n1, n2, n3);
43     cout << "SEQ: result=" << get<0>(seq) << " (elapsed " << get<1>(seq) << " ms)" << endl;
44     cout << "PAR/SEQ = " << fixed << setprecision(1) << (100.0 * get<1>(par) / get<1>(seq)) <<"%"<< endl;
45 }
46
```

```
[jens@vbox3 std-threads]$ g++ --std=c++11 -pthread thread-by-task.cpp -o thread-by-task
[jens@vbox3 std-threads]$ ./thread-by-task 40 40 40
Computing: fib(40) + fib(40) + fib(40)
PAR: result=307002465 (elapsed 700 ms)
SEQ: result=307002465 (elapsed 1948 ms)
PAR/SEQ = 35.9%
[jens@vbox3 std-threads]$
```

493

Scanning the file systems using async tasks

```
int main(int argc, char** argv) {
    const auto baseDir = fs::path{argc == 1 ? "." : argv[1]};
    if (!fs::is_directory(baseDir)) {
        throw invalid_argument{"not a directory: "s + baseDir.string()};
    }

    cout << "Base dir: " << fs::canonical(baseDir) << endl;
    auto result = aggregate(baseDir);
    for (const auto& [ext, stats] : map{result.cbegin(), result.cend()})
        if (ignoredExt.count(ext) == 0 && stats.text)
            cout << stats << endl;

    return 0;
}
```

```
C++> ./cmake-build-debug/file-stats ../../..
Base dir: "/mnt/c/Users/jensr/Dropbox/Ribomation/Ribomation-Training-2017-Autumn/cxx"
    .bat:     2181 bytes (32 lines)
      .c:  2462501 bytes (83331 lines)
    .cpp:  2383751 bytes (77677 lines)
    .cxx:  1593303 bytes (64602 lines)
      .h:  5061656 bytes (147639 lines)
    .hpp:    53538 bytes (2258 lines)
   .html:    18064 bytes (330 lines)
    .hxx:   656423 bytes (11204 lines)
    .iml:    11559 bytes (236 lines)
   .java:      622 bytes (23 lines)
   .json:     1659 bytes (61 lines)
    .log:  6577647 bytes (81274 lines)
     .md:    38794 bytes (994 lines)
     .sh:     6150 bytes (369 lines)
    .txt: 71719106 bytes (1547460 lines)
    .xml:  1812502 bytes (38317 lines)
C++>
```

494

Scanning the file systems using async tasks

```cpp
FileStats aggregate(const fs::path& dir) {
    FileStats                   stats;
    vector<future<FileStats>> dirStats;

    for (const auto& e : fs::directory_iterator{dir}) {
        if (fs::is_regular_file(e)) {
            auto ext = e.path().extension().string();
            update(stats, ext, e.path());
        } else if (fs::is_directory(e)) {
            dirStats.emplace_back( async(launch::async, [=]() {
                return aggregate(e.path());
            }) );
        }
    }
    for (auto& f : dirStats) update(stats, f.get());

    return stats;
}
```

Launch async task and append future

Wait for async result and aggregate.

495

Scanning the file systems using async tasks

```cpp
struct Count {
    const string ext;
    const bool   text;
    unsigned     size  = 0;
    unsigned     lines = 0;

    Count(string ext, bool text) : ext{move(ext)}, text{text} {}
    Count() : Count{"", false} {}

    void update(const fs::path& file) {
        ifstream f{file};
        f.seekg(0, ios::end);
        size += f.tellg();
        if (text) {
            f.seekg(0);
            for (string line; getline(f, line);) ++lines;
        }
    }

    void update(const Count& cnt) {
        size += cnt.size;
        lines += cnt.lines;
    }

    ~Count() = default;
    Count(const Count&) = default;
    Count& operator =(const Count&) = delete;

    friend ostream& operator <<(ostream& os, const Count& cnt) {
        return os << setw(8) << cnt.ext << ": "
                  << setw(8) << cnt.size << " bytes "
                  << (cnt.text ? "("s + to_string(cnt.lines) + " lines)"s : ""s);
    }
};
```

```cpp
using FileStats = unordered_map<string, Count>;

static const unordered_set<string> textExt = {
        ".txt", ".cxx", ".hxx", ".sh", ".xml", ".iml", ".log",
        ".md", ".java", ".json", ".html", ".hpp", ".cpp", ".c", ".h", ".bat"
};

static const unordered_set<string> ignoredExt = {
        ".internal", ".includecache", ".check_cache", ".marks", ".cbp", ".dir"
};

void update(FileStats& stats, const string& ext, const fs::path& file) {
    if (stats.count(ext) == 0) {
        stats.emplace(make_pair(ext, Count{ext, textExt.count(ext) > 0}));
    }
    stats[ext].update(file);
}

void update(FileStats& stats, const string& ext, const Count& cnt) {
    if (stats.count(ext) == 0) {
        stats.emplace(make_pair(ext, Count{ext, textExt.count(ext) > 0}));
    }
    stats[ext].update(cnt);
}

void update(FileStats& stats, const FileStats& dirStat) {
    for (const auto&[ext, s] : dirStat) {
        update(stats, ext, s);
    }
}
```

496

Packaged task

- A std::packaged_task is a std::function linked to a std::future
  - std::async wraps and calls a std::packaged_task (possibly in a different thread)
- More info
  - https://stackoverflow.com/questions/18143661/what-is-the-difference-between-packaged-task-and-async

A `packaged_task` won't start on it's own, you have to invoke it:

```
std::packaged_task<int()> task(sleep);

auto f = task.get_future();
task(); // invoke the function

// You have to wait until task returns. Since task calls sleep
// you will have to wait at least 1 second.
std::cout << "You can see this after 1 second\n";

// However, f.get() will be available, since task has already finished.
std::cout << f.get() << std::endl;
```

```
//! sleeps for one second and returns 1
auto sleep = [](){
    std::this_thread::sleep_for(std::chrono::seconds(1));
    return 1;
};
```

```
std::packaged_task<int(int,int)> task(f);
std::future<int> result = task.get_future();

std::thread task_td(std::move(task), 2, 10);
task_td.join();

std::cout << "task_thread:\t" << result.get() << '\n';
```

On the other hand, `std::async` with `launch::async` will try to run the task in a different thread:

```
auto f = std::async(std::launch::async, sleep);
std::cout << "You can see this immediately!\n";

// However, the value of the future will be available after sleep has finished
// so f.get() can block up to 1 second.
std::cout << f.get() << "This will be shown after a second!\n";
```

497

Atomic variables

Operations on atomic types

atomic_is_lock_free (C++11)	checks if the atomic type's operations are lock-free (function template)
atomic_store (C++11) atomic_store_explicit (C++11)	atomically replaces the value of the atomic object with a non-atomic argument (function template)
atomic_load (C++11) atomic_load_explicit (C++11)	atomically obtains the value stored in an atomic object (function template)
atomic_exchange (C++11) atomic_exchange_explicit (C++11)	atomically replaces the value of the atomic object with non-atomic argument and returns the old value of the atomic (function template)
atomic_compare_exchange_weak (C++11) atomic_compare_exchange_weak_explicit (C++11) atomic_compare_exchange_strong (C++11) atomic_compare_exchange_strong_explicit (C++11)	atomically compares the value of the atomic object with non-atomic argument and performs atomic exchange if equal or atomic load if not (function template)
atomic_fetch_add (C++11) atomic_fetch_add_explicit (C++11)	adds a non-atomic value to an atomic object and obtains the previous value of the atomic (function template)
atomic_fetch_sub (C++11) atomic_fetch_sub_explicit (C++11)	subtracts a non-atomic value from an a... obtains the previous value of the atomi... (function template)

Type alias	Definition
std::atomic_bool	std::atomic<bool>
std::atomic_char	std::atomic<char>
std::atomic_schar	std::atomic<signed char>
std::atomic_uchar	std::atomic<unsigned char>
std::atomic_short	std::atomic<short>
std::atomic_ushort	std::atomic<unsigned short>
std::atomic_int	std::atomic<int>
std::atomic_uint	std::atomic<unsigned int>
std::atomic_long	std::atomic<long>
std::atomic_ulong	std::atomic<unsigned long>
std::atomic_llong	std::atomic<long long>
std::atomic_ullong	std::atomic<unsigned long long>

498

Readers / Writer using atomic data

```
void reader(int id)
{
    for(;;)
    {
        // lock
        while(std::atomic_fetch_sub(&cnt, 1) <= 0)
            std::atomic_fetch_add(&cnt, 1);
        // read
        if(!data.empty())
            std::cout << (  "reader " + std::to_string(id)
                          + " sees " + std::to_string(*data.rbegin()) + '\n');
        if(data.size() == 25)
            break;
        // unlock
        std::atomic_fetch_add(&cnt, 1);
        // pause
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
}
```

```
void writer()
{
    for(int n = 0; n < 25; ++n)
    {
        // lock
        while(std::atomic_fetch_sub(&cnt, N+1) != N)
            std::atomic_fetch_add(&cnt, N+1);
        // write
        data.push_back(n);
        std::cout << "writer pushed back " << n << '\n';
        // unlock
        std::atomic_fetch_add(&cnt, N+1);
        // pause
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
}
```

```
int main()
{
    std::vector<std::thread> v;
    for (int n = 0; n < N; ++n) {
        v.emplace_back(reader, n);
    }
    v.emplace_back(writer);
    for (auto& t : v) {
        t.join();
    }
}
```

```
// meaning of cnt:
// 5: there are no active readers or writers.
// 1...4: there are 4...1 readers active, The writer is blocked
// 0: temporary value between fetch_sub and fetch_add in reader lock
// -1: there is a writer active. The readers are blocked.
const int N = 5; // four concurrent readers are allowed
std::atomic<int> cnt(N);

std::vector<int> data;
```

Output:

```
writer pushed back 0
reader 2 sees 0
reader 3 sees 0
reader 1 sees 0
<...>
reader 2 sees 24
reader 4 sees 24
reader 1 sees 24
```

https://en.cppreference.com/w/cpp/atomic/atomic_fetch_add

499

CHAPTER SUMMARY

Asynchronous Tasks

- Promise<T> p;
- Future<T> f = p.get_future();
- auto result = async(launch::async , [](){ *expr* })
  - Do not use launch::deferred

Use async calls to ensure that the tasks are always launched in fresh threads. The authors of GCC came to realize this as well, and switched the libstdc++ default launch policy to std::launch::async in mid-2015. In fact, as the discussion in that bug highlights, std::async came close to being deprecated in the next C++ standard, since the standards committee realized it's not really possible to implement real task-based parallelism with it without non-deterministic and undefined behavior in some corner cases.

https://eli.thegreenplace.net/2016/the-promises-and-challenges-of-stdasync-task-based-parallelism-in-c11/

500

EXERCISE

Word frequencies

- Load a large text file into memory (e.g. gitlab/files/Shakespeare.txt)
- Create T async tasks (where T > 1) that process the file in T different segments
- For each task, extract all the words (sequence of letters) and update the word frequency in an unordered_multiset<string>
- Wait for all partial results and combine into one container
- Pour over all data into a map<unsigned, string> container
- Print out the N most frequent words (where N > 10) together with counts

501

Intentional Blank

502

503

504

What is (template) meta-programming

- Meta-programming
  - A program that writes another program
- Template meta-programming
  - Meta-programming using templates
- In other words;
  - Possible to let the compiler "pre-compute" values

505

Classic compile-time function

```
using UL = unsigned long;

template<unsigned N>
struct Factorial {
    constexpr static UL value = N * Factorial<N - 1>::value;
};

template<>
struct Factorial<1> {
    constexpr static UL value = 1;
};

int main(int, char**) {
    UL values[] = {
            Factorial<1>::value,
            Factorial<5>::value,
            Factorial<10>::value,
            Factorial<12>::value,
            Factorial<20>::value,
    };
    for (auto x : values) cout << x << endl;
    return 0;
}
```

```
factorial
/home/jens/Courses/cxx/cxx-embedded/s
1
120
3628800
479001600
2432902008176640000

Process finished with exit code 0
```

506

All values are computed by the compiler

```
objdump --source --line-numbers --demangle cmake-build-debug/factorial
```

```
 916:   31 c0                   xor    %eax,%eax
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/factorial.cxx:18
   UL values[] = {
 918:   48 c7 45 d0 01 00 00    movq   $0x1,-0x30(%rbp)
 91f:   00
 920:   48 c7 45 d8 78 00 00    movq   $0x78,-0x28(%rbp)
 927:   00
 928:   48 c7 45 e0 00 5f 37    movq   $0x375f00,-0x20(%rbp)
 92f:   00
 930:   48 c7 45 e8 00 fc 8c    movq   $0x1c8cfc00,-0x18(%rbp)
 937:   1c
 938:   48 b8 00 00 b4 82 7c    movabs $0x21c3677c82b40000,%rax
 93f:   67 c3 21
 942:   48 89 45 f0             mov    %rax,-0x10(%rbp)
/home/jens/Courses/cxx/cxx-embedded/src/explorations/templates/factorial.cxx:25
          Factorial<5>::value,
```

```
Factorial<1>::value,
Factorial<5>::value,
Factorial<10>::value,
Factorial<12>::value,
Factorial<20>::value,
```

```
1
120
3628800
479001600
2432902008176640000
```

37 5F00

HEX 37 5F00
DEC 3 628 800
OCT 15 657 400
BIN 0011 0111 0101 1111 0000 0000

QWORD MS M˅

507

Compile-time expressions and functions

- Possible to define variables and functions that provide/evaluate values during compile time, by prefixing the return value with constexpr

```
constexpr unsigned KB = 1024;
constexpr unsigned MB = 1024 * KB;
```

```
template<typename T> auto PI = T{3.141'592'653'589'793'238};
constexpr auto PI_F = PI<float>;
```

```
constexpr long  SUM(long n) {return n * (n + 1)/2;}
constexpr static long  value = SUM(100);
```

508

Compile-time function using the modern style (constexpr)

```
using XXL = unsigned long long;

constexpr XXL fibonacci(unsigned n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    XXL tbl[] = {
            fibonacci(2),
            fibonacci(5),
            fibonacci(10),
            fibonacci(20),
            fibonacci(30),
            fibonacci(40)
    };
    for (auto n : tbl) std::cout << n << "\n";
    return 0;
}
```

objdump --source --demangle fibonacci-constexpr

```
  XXL tbl[] = {
8b0:   48 c7 45 c0 01 00 00    movq   $0x1,-0x40(%rbp)
8b7:   00
8b8:   48 c7 45 c8 05 00 00    movq   $0x5,-0x38(%rbp)
8bf:   00
8c0:   48 c7 45 d0 37 00 00    movq   $0x37,-0x30(%rbp)
8c7:   00
8c8:   48 c7 45 d8 6d 1a 00    movq   $0x1a6d,-0x28(%rbp)
8cf:   00
8d0:   48 c7 45 e0 28 b2 0c    movq   $0xcb228,-0x20(%rbp)
8d7:   00
8d8:   48 c7 45 e8 cb 7e 19    movq   $0x6197ecb,-0x18(%rbp)
8df:   06
          fibonacci(10),
          fibonacci(20),
          fibonacci(30),
          fibonacci(40)
  };
  for (auto n : tbl) std::cout ‹
```

HEX	619 7ECB
DEC	102 334 155
OCT	606 277 313
BIN	0110 0001 1001 0111 1110 1100 1011

```
fibonacci-constexpr
/mnt/c/Us
1
5
55
6765
832040
102334155

Process finished with exit code 0
```

509

Compile-time if statement

- Introduced in C++17 to simplify template function design

```
template<typename T>
constexpr T ABS(T x) { return x < 0 ? -x : x; }

template<typename T>
constexpr auto TOLERANCE = T{0.000'001};

template<typename T>
constexpr bool EQUALS(T lhs, T rhs) {
    cout << "{" << __PRETTY_FUNCTION__ << "} = ";
    if constexpr (is_floating_point_v<T>)
        return ABS(lhs - rhs) < TOLERANCE<T>;
    else if constexpr (is_integral_v<T>)
        return lhs == rhs;
    else if constexpr (is_convertible_v<T, std::string>)
        return lhs == rhs;
    else
        return false;
}
```

```
#include <iostream>
#include <iomanip>
#include <type_traits>
using namespace std;
using namespace std::literals;
```

```
int main() {
    cout << boolalpha << "int   : " << EQUALS(42, 2 * 21) << "\n";
    cout << boolalpha << "double: " << EQUALS(42.0, 2.0 * 21.0) << "\n";
    cout << boolalpha << "string: " << EQUALS("42"s, "4"s + "2"s) << "\n";

    return 0;
}
```

```
equals
/mnt/c/Users/jensr/Dropbox/Ribomation/Ribomation-Training-2017-Autumn/cxx/cxx-embedded/sr
int   : {constexpr bool EQUALS(T, T) [with T = int]} = true
double: {constexpr bool EQUALS(T, T) [with T = double]} = true
string: {constexpr bool EQUALS(T, T) [with T = std::__cxx11::basic_string<char>]} = true

Process finished with exit code 0
```

510

Flexible return type

```
template<typename T>
T maximum(T a, T b) { return a >= b ? a : b; }

template<typename T1, typename T2>
typename common_type<T1, T2>::type
MAXIMUM(T1 a, T2 b) { return a >= b ? a : b; }


int main(int, char**) {
    int   x = 10, y = 42;
    float z = 21.5;

    cout << "max(x,y) = " << maximum(x, y) << endl;

    //cout << "max(x,z) = " << maximum(x, z) << endl;
    //error: no matching function for call to 'maximum(int&, float&)'

    cout << "MAX(x,y) = " << MAXIMUM(x, y) << endl;
    cout << "MAX(x,z) = " << MAXIMUM(x, z) << endl;

    return 0;
}
```

flexible-return

```
/home/jens/Courses/cxx/cxx-embedded/s
max(x,y) = 42
MAX(x,y) = 42
MAX(x,z) = 21.5

Process finished with exit code 0
```


Getting the same type: `decltype` (*expr*)

```
struct PersonFactory { Person create() {return {};} };
struct AddressFactory { Address create() {return {};} };

template<typename Factory>
auto mkObject(Factory& f) -> decltype(f.create()) {
    auto obj = f.create();
    return obj;
}

int main() {
    PersonFactory pf;
    AddressFactory af;
    cout << "obj-1: " << mkObject(pf).toString() << endl;
    cout << "obj-2: " << mkObject(af).toString() << endl;
    cout << "obj-3: " << mkObject(pf).toString() << endl;
    cout << "obj-4: " << mkObject(af).toString() << endl;
    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;

string operator++(string& s) { return s = to_string(stoi(s) + 1); }

string cnt = "0";

class Person {
    string name;
public:
    Person()
      : name("prs" + ++cnt) { }
    string toString() const { return name; }
};

class Address {
    string street, zip, city;
public:
    Address()
      : street("str" + ++cnt), zip("zip" + ++cnt), city("cty" + ++cnt) { }
    string toString() const { return street + ", " + zip + " " + city; }
};
```

object-factory

```
/home/jens/.CLion12/system
obj-1: prs1
obj-2: str2, zip3 cty4
obj-3: prs5
obj-4: str6, zip7 cty8

Process finished with exit
```

Significant safety improvement of machine-level code

- When doing C style programming, CPP macros are used extensively

```
#define PACK(hi, lo)  ((uint16_t)(((uint16_t)(hi) << 8) | (lo)))

void doit() {
uint8_t  lo = 0x123;
uint8_t  hi = 0x456;
uint16_t w  = PACK(hi, lo);
```

However, the compiler cannot prevent subtle bugs

```
int8_t   lo = 0x123;
int16_t  hi = 0x456;
uint16_t w  = PACK(hi, lo);  //oops, unpredicted result
```

513

Type-safe word packing

In C++ we can check the types, their bit sizes and if they are unsigned

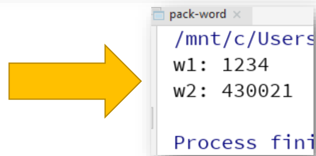
```
template<typename DstType, typename SrcType>
DstType pack(SrcType hiHalf, SrcType loHalf) {
    static_assert(is_unsigned_v<SrcType>, "src-type must be unsigned");
    static_assert(is_unsigned_v<DstType>, "dst-type must be unsigned");

    constexpr int SRC_DIGITS = numeric_limits<SrcType>::digits;
    constexpr int DST_DIGITS = numeric_limits<DstType>::digits;
    static_assert((2 * SRC_DIGITS) == DST_DIGITS, "src-type not half of dst-type");

    return static_cast<DstType>(
            static_cast<DstType>(hiHalf) << SRC_DIGITS | loHalf
    );
}
```

514

Usage of pack<Dst,Src>(hi,lo)

```
int main() {
    uint8_t hi = 0x12;
    uint8_t lo = 0x34;
    auto    w  = pack<uint16_t>(hi, lo);
    cout << "w1: " << hex << w << "\n";

    cout << "w2: " << hex << pack<uint32_t, uint16_t>(0x43, 0x21) << "\n";
    return 0;
}
```

Re-use: no need to create auxiliary functions

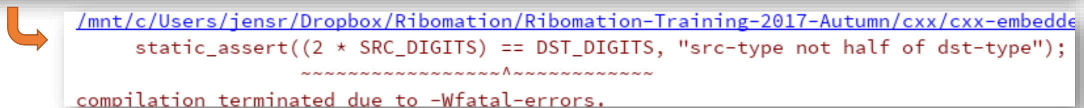


515

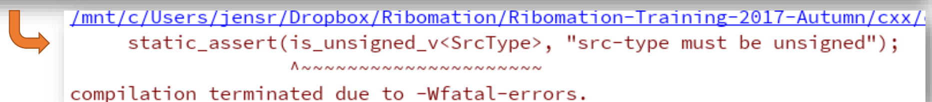
Catching errors in compile time

The compiler will politely safeguard you and spare you spending endless of hours running the debugger. Hey, you might even forget how to use a debugger ☺

```
auto mis_match = pack<uint16_t>(static_cast<uint16_t>(0x12),
                                static_cast<uint16_t>(0x34));
```




```
auto not_unsigned = pack<uint16_t>(static_cast<int8_t>(0x12),
                                   static_cast<int8_t>(0x34));
```



516

Variadic templates

- Unbounded number of template parameters

```
template <size_t... Entries>
struct MyArray {
  size_t array[] = {Entries...};
};
```

```
MyArray<2,3,5,7,11> firstPrimes;
```

```
struct MyArray {
  size_t array[] = {2,3,5,7,11};
};
```

517

Template fold expressions

- Apply a binary operator on a set of arguments, as an unrolled expression

```
template<typename ... T>
auto SUM(T ... args) {
    return (... +args);
}

int main() {
    cout << "numb: " << SUM(1, 2, 3, 4, 5) << endl;
    cout << "text: " << SUM("Hi"s, "-"s, "Fi"s) << endl;

    auto s1 = "Foo"s;
    auto s2 = "Boo"s;
    cout << "vars: " << SUM(s1, s2) << endl;
    return 0;
}
```

```
/mnt/c/Users/jen
numb: 15
text: Hi-Fi
vars: FooBoo

Process finished
```

```
SUM(1,2,3,4) → return 1 + 2 + 3 + 4;
```

518

Fold expression syntax

- Left fold
  - `(... @ args)` → `((arg1 @ arg2) @ arg3) @ arg4 ...`
- Right fold
  - `(args @ ...)` → `arg1 @ (arg2 @ ... (argN-1 @ argN))`
- Left fold with initial value (supports empty parameter pack)
  - `(init @ ... @ args)` → `((init @ arg1) @ arg2) @ arg3 ...`
- Right fold with initial value (supports empty parameter pack)
  - `(args @ ... @ init)` → `arg1 @ (arg2 @ ... (argN @ init))`

The symbol @ denotes an arbitrary binary operator.

All binary operators can be used in a fold expression, except '.', '->' and '[]'.

519

Compile-time type-safe CSV function

```
int main() {
    auto name   = "Anna Conda"s;
    auto age    = 42U;
    auto weight = 53.7;
    auto female = true;
    cout << "CSV: \"" << CSV(name, age, weight, female, "A char* string") << "\"\n";
    cout << "CSV: \"" << CSV<'#'>(name, age, weight, female, "A char* string") << "\"\n";
    return 0;
}
```

```
template<auto SEP = ';', typename First, typename... Rest>
string CSV(const First& first, const Rest& ... rest) {
    auto append = [&](const auto& arg) {
        return SEP + asString(arg);
    };
    return (asString(first) + ... + append(rest));
}
```

```
csv
/mnt/c/Users/jensr/Dropbox/Ribomation/Riboma
CSV: "Anna Conda;42;53.70;T;A char* string"
CSV: "Anna Conda#42#53.70#T#A char* string"

Process finished with exit code 0
```

520

Compile-time type-safe CSV function

```
template<typename T>
string toString(T x, int decimals) {
    static_assert(is_arithmetic_v<T>, "toString(T), T is not numeric");
    ostringstream buf;
    buf << fixed << setprecision(decimals) << x;
    return buf.str();
}

template<typename T>
string asString(T arg) {
    //cerr << __PRETTY_FUNCTION__ << endl;
    if constexpr (is_same_v<T, bool>) return arg ? "T" : "F";
    if constexpr (is_arithmetic_v<T>) return toString(arg, 2);
    if constexpr (is_convertible_v<T, std::string>) return arg;
    return "??";
}
```

521

CHAPTER SUMMARY

Enhanced template support

- Type predicates and modifiers are helpful in combination with compile-time conditional evaluation
- Functions marked as constexpr, simplifies template meta-programming
- Variadic templates and fold expressions as well

522

Common C++ Idioms

523

What is an idiom?

- A reusable code snippet provided as a viable solution of a particular programming problem
- The code snippet need to be adapted to the code environment it should be inserted into
- More complex idioms are usually referred to as (design) patterns

524

RAII – Resource Acquisition Is Initialization

- A.k.a. resource allocation objects (RAO)
- In the constructor
  - Open/allocate the resource
- In the destructor
  - Close/deallocate the resource

```
void copy(const string& from, const string& to) {
  ifstream in{from}; if(!in)  throw from;
  ofstream out{to} ; if(!out) throw to;

  for (char ch; in.get(ch); ) out.put(ch);
}
```

No need to close the files at the end of the function.
The infile is closed, if the outfile failed to open.

525

Debug tracing - usage

```
int main() {
  Trace  t("main");
  cout << "create person" << endl;
  Person p("Anna Conda", 31);
  cout << "p.name=" << p.getName() << ", p.age=" << p.getAge() << endl;
  {
    Trace b;
    Person q(p);
    cout << "q.name=" << q.getName() << ", q.age=" << q.getAge() << endl;
  }
  return 0;
}
```

```
class Person {
    Trace    trace;
    string    name;
    int      age;
 public:
  Person(const char* name_, int age_=0)
      : trace("Person",this), name(name_), age(age_) {}
  Person(const Person& p)
      : trace("Person",this), name(p.name), age(p.age) {}
  string  getName() const {return name;}
  int    getAge()  const {return age;}
};
```

526

Debug tracing - output

```
$ g++ tstTrace.cpp -o tstTrace
$ ./tstTrace
[main] ENTER
create person
[Person@0x28cd00] ENTER
p.name=Anna Conda, p.age=31
[BLOCK] ENTER
[Person@0x28cd42] ENTER
q.name=Anna Conda, q.age=31
[Person@0x28cd42] EXIT
[BLOCK] EXIT
[Person@0x28cd00] EXIT
[main] EXIT
$
```

527

Thread mutex locks

```
class Account {
  mutex    m;
  int      balance = 0;
 public:
  Account() = default;
  int  update(int amount) {
    lock_guard<mutex> g(m);
    balance += amount;
    return balance;
  }
};
```

528

Prevent mixed lines output, when multi-threading

- A single write(2) system call is atomic
- However, shifting to an ostream is not
  - cout << "x=" << x << ", y=" << y << "\n";
- Need to create the final string (inclusive '\n') first, then write it
  - cout << ("x=" + to_string(x) + ", y="s + to_string(y) +"\n"s);
  - The expression above is clumsy and relies on the to_string(*) to be defined

529

Mixed lines in output

```
void Worker(unsigned id, unsigned n) {
    for (auto k = 0U; k < n; ++k) {
        cout << string(id * 2, ' ') << "[" << id << "] " << k << "\n";
    }
}

int main() {
    auto           T = 10U;
    auto           N = 10'000U;
    vector<thread> workers;

    workers.reserve(T);
    for (auto k = 0U; k < T; ++k) {
        workers.emplace_back(&Worker, k + 1, N);
    }
    for (auto& w : workers) w.join();
    return 0;
}
```

atomic-output

```
                    [10] 7613
7736
                [8] 7737
                [8] 7738
                [8                      ] 7739
                [8] 7740
                [8] 7741
```

Mixed output lines

530

class AtomicOutput

```
class AtomicOutput : public ostringstream {
    using super = ostringstream;
    ostream& os;
public:
    AtomicOutput(ostream& os) : os(os) {}

    ~AtomicOutput() {
        os << super::str() << std::flush;
    }
};
```

```
void Worker(unsigned id, unsigned n) {
    for (auto k = 0U; k < n; ++k) {
        AtomicOutput{cout} << string(id
    }
}
```

Each line is wholly printed

```
atomic-output
    [5] 8138
[4] 8500
    [6] 8868
            [10] 8706
        [8] 8492
      [7] 8051
          [9] 7454
  [5] 8139
[4] 8501
    [6] 8869
            [10] 8707
        [8] 8493
      [7] 8052
          [9] 7455
  [5] 8140
[4] 8502
    [6] 8870
    [6] 8871
    [6] 8872
```

531

User-defined io-manip

- The stream (left) shift operator has one override that takes a function pointer
  - In the expression: `cout << endl;` the right operand is technically a function name

```
inline ostream& dump(ostream& os) {
    if (auto buf = dynamic_cast<AtomicOutput*>(&os); buf != nullptr) {
        buf->os << buf->str();
        os.flush();
        ostringstream emptyBuf;
        buf->swap(emptyBuf);
    }
    return os;
}
```

```
void Worker(unsigned id, unsigned n) {
    AtomicOutput buf{cout};
    for (auto k = 1U; k <= n; ++k) {
        buf << string(id * 3, ' ') << "[" << id << "] " << k << "\n" << dump;
    }
}
```

```
Terminal
        [3] 9977
                 [6] 9888
                    [7] 9917
                             [10] 9930
                       [8] 9975
        [3] 9978
                 [6] 9889
                    [7] 9918
                             [10] 9931
                       [8] 9976
```

532

Implementation of for-each support

- Provide an iterator class overloading
  - `T          operator *()`
  - `iterator&  operator ++()`
  - `bool       operator !=(const iterator& that)`
- Provide two functions in the class supporting for-each
  - `iterator   begin()`
  - `iterator   end()`

```
for (obj::iterator it = obj.begin(); it != obj.end(); ++it) {
    Type  element = *it;
    //...
}
```

533

class Range

```
class Range {
    int start, stop, step;

public:
    Range(int stop = 10, int start = 1, int step = 1)
            : start(start), stop(stop), step(step)
    {
        if (step == 0) throw std::invalid_argument("step must not be zero");
    }

    struct iterator {
        int current, step;

        iterator(int current, int step) : current(current), step(step) { }
        iterator(int current) : current(current) { }

        int operator *() { return current; }

        iterator& operator ++() {
            current += step;
            return *this;
        }

        bool operator !=(const iterator& that) {
            if (step > 0) {
                return this->current < that.current;
            } else {
                return this->current > that.current;
            }
        }
    };

    iterator begin() { return {start, step}; }
    iterator end()   { return {stop + (step > 0 ? +1 : -1)}; }
};
```

```
void to_5_should_pass() {
    ostringstream buf;
    for (auto k : Range(5)) { buf << k << " "; }
    assert(buf.str(), "1 2 3 4 5 ");
}

void from_5_to_15_should_pass() {
    ostringstream buf;
    for (auto k : Range(15, 5)) { buf << k << " "; }
    assert(buf.str(), "5 6 7 8 9 10 11 12 13 14 15 ");
}

void from_10_to_30_with_step_4_should_pass() {
    ostringstream buf;
    for (auto k : Range(30, 10, 4)) { buf << k << " "; }
    assert(buf.str(), "10 14 18 22 26 30 ");
}
```

534

More test cases

```
void negatives_should_pass() {
    ostringstream buf;
    for (auto k : Range(+10, -10, 5)) { buf << k << " "; }
    assert(buf.str(), "-10 -5 0 5 10 ");
}

void backwards_should_pass() {
    ostringstream buf;
    for (auto k : Range(2, 10, -2)) { buf << k << " "; }
    assert(buf.str(), "10 8 6 4 2 ");
}

void zero_step_should_fail() {
    try {
        Range(10, 1, 0);
        assert("expected std::invalid_argument exception", false);
    } catch (logic_error& e) {
        assert(e.what(), "step must not be zero");
    }
}
```

```
int main() {
    to_5_should_pass();
    from_5_to_15_should_pass();
    from_10_to_30_with_step_4_should_pass();
    from_10_to_28_with_step_4_should_pass();
    negatives_should_pass();
    backwards_should_pass();
    zero_step_should_fail();

    if (numErrors == 0) { cout << "All tests passed\n"; }
    return numErrors;
}
```

```
Range
C:\Users\jens\.clion10\system\cmake\generated\
All tests passed

Process finished with exit code 0
```

535

Implementation of static initializer support

- Include
  - <initializer_list>
- Let the class parameterized
  - template<typename T>
- Use parameter of type
  - initializer_list<T> args
- Iterate over the arguments using
  - size()
  - begin() / end()

536

Example of using static initializer_list

```
init-list.cpp  init-list-2.cpp
 1 #include <iostream>
 2 #include <numeric>
 3 #include <initializer_list>
 4 using namespace std;
 5
 6 template<typename T=int>
 7 class Stats {
 8     T       sum;
 9     int     cnt;
10   public:
11     Stats(initializer_list<T> args) : sum(0), cnt(0) {
12         cnt = args.size();
13         sum = accumulate(args.begin(), args.end(), 0);
14     }
15     double  mean()  const {return cnt==0 ? 0 : (double)sum/cnt;}
16     int     count() const {return cnt;}
17 };
18
19 int main() {
20     Stats<> s = {10,20,30,40,70,40,30,20,10};
21     cout << "Mean  = " << s.mean() << endl;
22     cout << "Count = " << s.count() << endl;
23     return 0;
24 }
```

```
jens@vbox4: ~/Documents/c++11
jens@vbox4:~/Documents/c++11$ g++ -std=c++0x -Wall init-list-2.cpp -o init-list-2
jens@vbox4:~/Documents/c++11$ ./init-list-2
Mean  = 30
Count = 9
jens@vbox4:~/Documents/c++11$
```

537

Overriding the index operator

- The index operator is supposed to return a reference to the payload, which works fine as long as its already at a memory location

```
class MyVector {
  int  storage[100];
//...
  int&  operator [](int idx) {
    if (idx < 0 || 100 <= idx) throw out_of_range("idx");
    return storage[idx];
  }
};
//...
MyVector  v;
v[10] = 42; //→ v.operator[](10) = 42;
```

How can we separate a READ from a WRITE, using the index operator? →

538

Separating read and write for operator[]

- Define a struct for transient use containing context data
- Let the index operator return that struct
- Implement an assignment operator and type-conversion operator

```
struct Position {
  unsigned idx;
  Position(unsigned idx) : idx(idx) {}
  operator T()                 {T t; READ(idx, t); return t;}
  Position&  operator =(T& t) {WRITE(idx, t); return *this;}
};

Position operator [](unsigned idx) {CHECK(idx); return {idx};}
```

539

How it translates

```
MyDatabaseFile<Account>  db{filename};

(1) Account  a = db[42];
    a += 1000;
(2) db[42]  = a;
```

```
(1) Account  a = db.operator[](42).operator Account();
    a += 1000;
(2) db.operator[](42).operator=(a);
```

540

CHAPTER SUMMARY

Common C++ Idioms

- Resource manager objects (RAII)
- Iterator and for-each support
- Static initializer support
- User-defined stream manipulators
- Separating read and write when using the index operator

541

Intentional Blank

542

Translating C++ into C

543

A typical C++ class

thing.cpp

```
1 #include <iostream>
2 #include <string>
3 #include <cstdlib>
4 using namespace std;
5
6 class Thing {
7     string    name;
8     int       value;
9   public:
10    Thing() {name = "None"; value = 17;}
11    Thing(string n, int v) : name(n), value(v) {}
12    ~Thing() {cout << "~Thing:" << name<<':'<<value << endl;}
13
14    string    getName() const {return name;}
15    int       getValue() const {return value;}
16    void      setValue(int v) {value = v;}
17
18    bool      operator !() {return name != "None";}
19 };
20
21 Thing     operator +(const Thing& left, const Thing& right)  {
22     return Thing(left.getName()+"SUM", left.getValue() + right.getValue());
23 }
24
25 ostream&  operator <<(ostream& os, const Thing& t) {
26     return os << t.getName() << ":" << t.getValue();
27 }
```

544

Invocation of a typical C++ class

```
29 int main(int numArgs, char* args[]) {
30     if (numArgs != 3) {
31         cerr << "usage: " << args[0] << " <name> <value>" << endl; exit(1);
32     }
33
34     string  s = args[1];
35     int     i = stoi(args[2]);
36
37     Thing  t1(s, i);
38     Thing  t2(s + "2nd", 2 * i);
39     Thing  t3;
40
41     int  n = t3.getValue();
42     t3.setValue(2 * n);
43     cout << "t3 = " << t3 << endl;
44
45     if (!t1) t3 = t1 + t2;
46     cout << "t3 = " << t3 << endl;
47 }
48
```

```
[jens@vbox3 cpp-behind-the-scenes]$ g++ --std=c++11 -g thing.cpp -o thing
[jens@vbox3 cpp-behind-the-scenes]$ ./thing foobar 10
t3 = None:34
~Thing:foobarSUM:30
t3 = foobarSUM:30
~Thing:foobarSUM:30
~Thing:foobar2nd:20
~Thing:foobar:10
[jens@vbox3 cpp-behind-the-scenes]$
```

545

A class is a just a plain C struct

C++

```
class Thing {
  string     name;
  int        value;
  . . .
};
```

```
Thing  t;
```

C

```
struct Thing {
  struct string  name;
  int            value;
};
```

```
struct Thing  t;
```

546

Methods are ordinary functions

```
class Thing {
  int      getValue() {return value;}
  void     setValue(int v) {value = v;}
  . . .
};
```

```
int  getValue__5ThingFv(struct Thing* this) {
  return this->value;
}

void setValue__5ThingFi(struct Thing* this, int v) {
  this->value = v;
}
```

However, in practice; things are slightly more complex

```
[jens@vbox3 cpp-behind-the-scenes]$ nm --format=posix --line-numbers ./thing | egrep -i '(get|set)(name|value)'
_ZN5Thing8setValueEi W 080492d4 0000000e
_ZNK5Thing7getNameEv W 080492a8 0000001f
_ZNK5Thing8getValueEv W 080492c8 0000000b
[jens@vbox3 cpp-behind-the-scenes]$
```

547

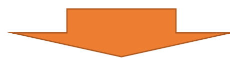
C++ name mangling

- The C++ compiler translates each identifier into a linker unique name
  - The name is based on the types involved

```
#include <iostream>
using namespace std;

class Whatever {
  string  name = "hepp";
  int     number = 42;
  static double PI;
public:
  Whatever();
  ~Whatever();
  int     update(int);
  string change(const string&);
};
```

```
#include "Whatever.hxx"
Whatever::Whatever() {}
Whatever::~Whatever() {}
int Whatever::update(int n) {number += n; return number;}
string Whatever::change(const string& s) {name = s; return name;}
double Whatever::PI = 3.141592653;
```

`Whatever::update(int)`



`_ZN8Whatever6updateEi`

548

Demangling names

- Use c++filt(1) to demangle names

```
jens@apollo:~/tmp$ c++filt _ZN8Whatever6updateEi
Whatever::update(int)
jens@apollo:~/tmp$
```

549

Name mangling

- C do not have function name overloading. The C++ compiler must therefore generate unique function names, which is does based on the function signature
- When linking C++ and C code, you must prevent name mangling for C functions
  - Use an **extern "C" {. . .}** block

```
extern "C" {
  int printf(const char *format, ...);
  int close(int fd);
  int stat(const char *path, struct stat *buf);
};
```

550

Method invocations

```
Thing  t;

int  n = t.getValue();
t.setValue(42);
```

```
struct Thing  t;
//. . .
int  n = getValue__5ThingFv(&t);
setValue__5ThingFi(&t, 42);
```

551

Operators are ordinary functions too

```
Thing  operator +(const Thing& left, const Thing& right) {
  Thing result;
  result.setValue(left.getValue() + right.getValue());
  return result;
}
```

```
struct Thing
_pl__FCR5ThingCR5Thing(struct Thing* left, struct Thing* right) {
  struct Thing result;
  setValue__5ThingFi(&result,
         getValue__5ThingFv(left) + getValue__5ThingFv(right));
  return result; //struct copy
}
```

```
Thing  t1,t2,t3;
//. . .
t3 = t1 + t2;
```

```
struct Thing  t1,t2,t3;
//. . .
t3 = _pl__FCR5ThingCR5Thing(&t1, &t2);
```

552

Constructors/destructors are just C functions too

```
{
  Thing  t1, t2("foo", 42);
  //. . .
}
```

```
{
  struct Thing  t1, t2;
  _ct__5ThingFv(&t1);
  _ct__5ThingFCPci(&t2, "foo", 42);
  //. . .
  _dt__5ThingFv(&t2);
  _dt__5ThingFv(&t1);
}
```

553

Constructors/destructor

```
class Thing {
  string  name;
  int     value;
 public:
  Thing() {name="None"; value=17;}
  Thing(const char* n, int v) : name(n), value(v) {}
  ~Thing() {cerr << "Bye";}
};
```

```
struct Thing*  _ct_5ThingFv(struct Thing* this) {
  _ct_6stringFv(&(this->name));
  _as_6stringFCPc(&(this->name), "None");
  this->value = 17;
  return this;
}
```

name is initialized twice

```
struct Thing*  _ct_5ThingFCPci(struct Thing* this, char* n, int v) {
  _ct_6stringFCPc(&(this->name), n);
  this->value = v;
  return this;
}
```

```
struct Thing*  _dt_5ThingFv(struct Thing* this) {
  _ls__FR6ostreamCPc(&cerr, "Bye")
  return this;
}
```

554

Dynamic objects act in 2-steps

```
Thing*  ptr = new Thing("abc", 42);
//. . .
delete ptr;
```

```
struct Thing*  ptr;
ptr = _ct__5ThingFCPci(
        _nw__Fui( sizeof(struct Thing) ),
        "abc", 42);
//. . .
_dl__FPv( _dt__5ThingFv(ptr) );
```

(1) Allocate block
(2) Initialize object

(3) Clean-up object
(4) De-allocate block

555

Subclasses inherits all variables, in one large block

```
class Super {
    long  lv;
  public:
    Super(long v=42) : lv(v) {}
};

class Sub : public Super {
    string s;
    short  sv;
  public:
    Sub(short v=17) : Super(2*v), sv(v) {}
};
```

```
struct Super {
  long lv;
};

struct Sub {
  long   lv;
  struct string  s;
  short  sv;
};
```

556

The subclass constructor invokes the superclass' constructor

```
class Sub : public Super {
    string s;
    short  sv;
  public:
    Sub(short v=17) : Super(2*v), sv(v) {}
};

Sub  obj;
```

```
struct Sub*  _ct__3SubFs(struct Sub* this, short v) {
  _ct__5SuperFi( (struct Super*)this, 2*v );
  _ct__6stringFv( &(this->s) );
  this->sv = v;
  return this;
}

struct Sub  obj;
_ct__3SubFs(&obj, 17);
```

557

Virtual methods

```
class A {
  public:
    virtual int  compute(int n) {return n*n;}
};
class B : public A {
  public:
    virtual int  compute(int n) {return n-10;}
};
class C : public B {
  public:
    virtual int  sum(int n) {return n*(n+1)/2;}
};
```

```
struct A {
  VirtualMethod*  _vptr;
};
struct B {
  VirtualMethod*  _vptr;
};
struct C {
  VirtualMethod*  _vptr;
};
```

A::_vtbl: &A::compute

B::_vtbl: &B::compute

C::_vtbl: &B::compute, &C::sum

_vtbl sometimes are called vtable

_vtbl are global arrays located in the DATA area

_vptr is assigned in the constructor

558

Invocation of a virtual method

```
A*  p1 = new A();
A*  p2 = new B();
A*  p3 = new C();

int r1 = p1->compute(10);
int r2 = p2->compute(20);
int r3 = p3->compute(30);
```

```
struct A* p1 = _ct_1AFv( _nw_FUi(sizeof(A)) );
struct A* p2 = _ct_1BFv( _nw_FUi(sizeof(B)) );
struct A* p3 = _ct_1CFv( _nw_FUi(sizeof(C)) );

int  r1 = (*(p1->_vptr[0]))(p1, 10);
int  r2 = (*(p2->_vptr[0]))(p2, 20);
int  r3 = (*(p3->_vptr[0]))(p3, 30);
```

559

Lambdas are structs with a function call operator

```
int C = 42;
auto  f = [=](int n){return n + C;};
int sq = f(5);
```

```
struct lambda0 {
  int C;
  lambda0(int _C) : C(_C) {}
  int  operator()(int n) {return n + C;}
};

int C = 42;
lambda0  f(C);
int sq = f(5); //f.operator()(5);
```

560

Lambdas with free variables

```
vector<int>  v = {1,2,3,4,5};
int sum = 0;
for_each(v.begin(),v.end(), [&](int n){
      sum += n;
});
cout << "sum = " << sum << endl;
```

```
struct lambda1 {
  int& sum;
  lambda1(int& sum) : sum(_sum) {}
  void operator()(int n) {sum += n;}
};

int sum = 0;
for_each(v.begin(),v.end(), lambda1(sum));
// → for (; first != last; ++first) lambda1(*first);
//   → for (...) lambda1.operator()( first.operator*() )
cout << "sum = " << sum << endl;
```

561

Exceptions uses setjmp() / longjmp()

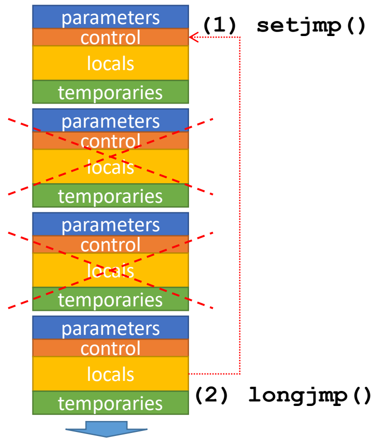
```
try {
  //…business logic…
} catch (Exception1 x) {
  //error handling 1
} catch (Exception2 x) {
  //error handling 2
} catch (...) {
  //unknown error
}
```

```
jmp_buf exception;
int rc = setjmp(exception);
if (rc < 0) syserror(rc);
if (rc == 0) {
  //…business logic…
} else if (rc == 1) {
  //error handling 1
} else if (rc == 2) {
  //error handling 2
} else {
  //unknown error
}
```

562

What is setjmp() / longjmp()

- Plus a stack of destructor pointers



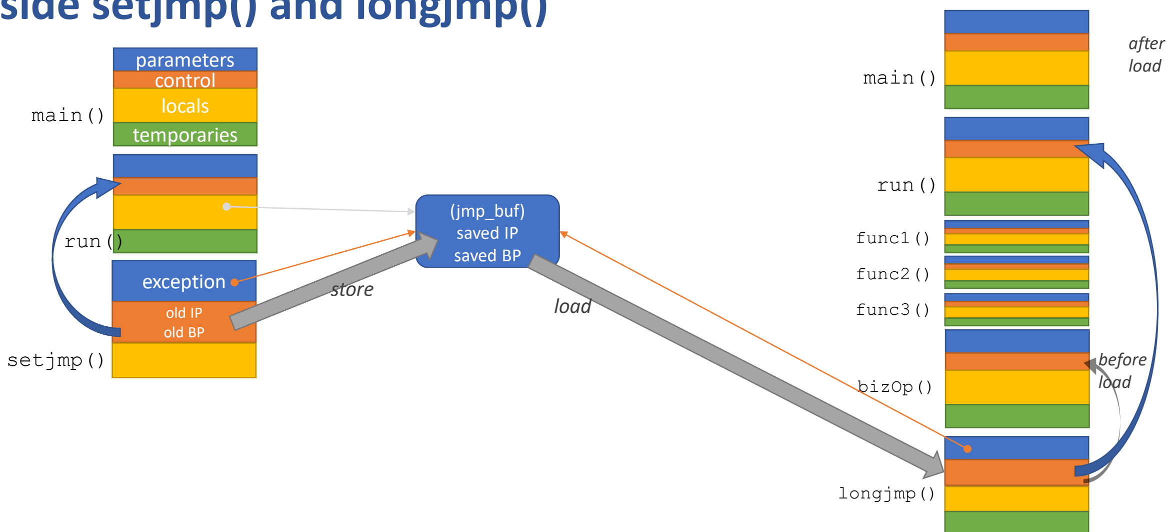

```
int main() {
  ...
  if ((rc = setjmp(exception)) != 0)
  ...
}
```

Invoked onced, but returned twice

```
void someFunc() {
  ...
  longjmp(exception, ERROR_2);
  ...
}
```

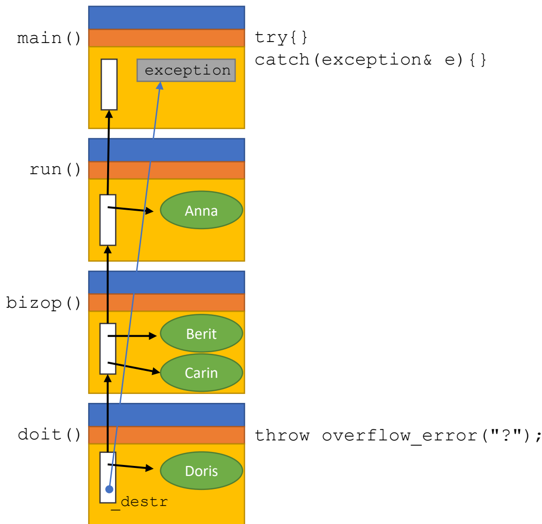
563

Inside setjmp() and longjmp()



564

Realization of `throw e;`




```
try{}
catch(exception& e){}
```

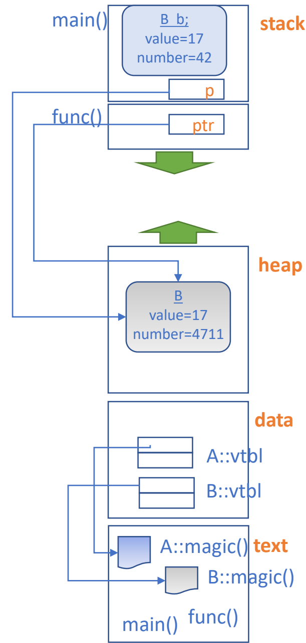
```
const int excType = 1;

for (DestrInfo d = _destr; d; ++d) {
  _invoke_destructor(d.object);
}
longjmp(*exception, excType);
```

```
throw overflow_error("?");
```

565

The complete picture

```
struct A{
  int  value=17;
  virtual int magic() {return value;}
};

struct B: public A{
  short  number=42;
  B(int n) : numer(n) {}
  int magic() {return number;}
};

void func(A* ptr) {
  cout << ptr->magic();
}

int main() {
  B  b;
  A*  p = new B(4711);
  func(p);
  return 0;
}
```



566

CHAPTER SUMMARY

Translating C++ into C

- It's important to understand that –in reality– C++ is just a thin wrapper around plain ordinary C
- Member variables are struct variables
- Methods are ordinary functions with funny names
- The 'this' pointer is the first parameter of each member function
- Virtual methods are function pointers
- Exceptions are based on longjmp and destruction of all local objects
- Lambdas are just plain structs with a function-call operator

567

Intentional Blank

568

Upcoming Features of C++

569

Parallel versions of STL algorithms

- Part of C++17, but not yet generally implemented

```
#include <experimental/execution_policy>
#include <experimental/numeric>

int main()
{
    std::vector<double> v(10'000'007, 0.5);

    {
        auto t1 = std::chrono::high_resolution_clock::now();
        double result = std::accumulate(v.begin(), v.end(), 0.0);
        auto t2 = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double, std::milli> ms = t2 - t1;
        std::cout << std::fixed << "std::accumulate result " << result
                  << " took " << ms.count() << " ms\n";
    }

    {
        auto t1 = std::chrono::high_resolution_clock::now();
        double result = std::experimental::parallel::reduce(
                            std::experimental::parallel::par,
                            v.begin(), v.end());
        auto t2 = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double, std::milli> ms = t2 - t1;
        std::cout << "parallel::reduce result "
                  << result << " took " << ms.count() << " ms\n";
    }
}
```

Possible output:

```
std::accumulate result 5000003.50000 took 12.7365 ms
parallel::reduce result 5000003.50000 took 5.06423 ms
```

The TS provides parallelized versions of the following 69 algorithms from <algorithm>, <numeric> and <memory>:

Standard library algorithms for which parallelized versions are provided [Collapse]		
std::adjacent_difference	std::is_heap_until	std::replace_copy_if
std::adjacent_find	std::is_partitioned	std::replace_if
std::all_of	std::is_sorted	std::reverse
std::any_of	std::is_sorted_until	std::reverse_copy
std::copy	std::lexicographical_compare	std::rotate
std::copy_if	std::max_element	std::rotate_copy
std::copy_n	std::merge	std::search
std::count	std::min_element	std::search_n
std::count_if	std::minmax_element	std::set_difference
std::equal	std::mismatch	std::set_intersection
std::fill	std::move	std::set_symmetric_difference
std::fill_n	std::none_of	std::set_union
std::find	std::nth_element	std::sort
std::find_end	std::partial_sort	std::stable_partition
std::find_first_of	std::partial_sort_copy	std::stable_sort
std::find_if	std::partition	std::swap_ranges
std::find_if_not	std::partition_copy	std::transform
std::generate	std::remove	std::uninitialized_copy
std::generate_n	std::remove_copy	std::uninitialized_copy_n
std::includes	std::remove_copy_if	std::uninitialized_fill
std::inner_product	std::remove_if	std::uninitialized_fill_n
std::inplace_merge	std::replace	std::unique
std::is_heap	std::replace_copy	std::unique_copy

570

Ranges

- Support for pipeline data streams in STL
- More info
  - https://ericniebler.github.io/std/wg21/D4128.html
  - https://github.com/ericniebler/range-v3/

```cpp
#include <iostream>
#include "range/v3/core.hpp"
#include "range/v3/view/filter.hpp"
#include "range/v3/view/transform.hpp"
#include "range/v3/view/iota.hpp"
using namespace std;
using namespace ranges::view;

int main() {
    auto      results = ints(1, 100)
                      | filter([](int n) { return n % 2 == 1; })
                      | transform([](int n) { return n * n; });
    for (auto n : results) cout << n << "  ";
    cout << endl;
    return 0;
}
```

```
simple-pipeline
/mnt/c/Users/jensr/Dropbox/Ribomation/Ribomation-Training-2017-Autumn/cxx/cxx-17/src
 -exploration/ranges/cmake-build-debug/simple-pipeline
1  9  25  49  81  121  169  225  289  361  441  529  625  729  841  961  1089  1225
 1369  1521  1681  1849  2025  2209  2401  2601  2809  3025  3249  3481  3721  3969
 4225  4489  4761  5041  5329  5625  5929  6241  6561  6889  7225  7569  7921  8281
 8649  9025  9409  9801

Process finished with exit code 0
```

571

Concepts

- Boolean predicates on template parameters, evaluated at compile time
  - May be associated with a template, in which case it serves as a constraint: it limits the set of arguments that are accepted as template parameters
- More info
  - https://en.wikipedia.org/wiki/Concepts_(C%2B%2B)
  - https://accu.org/index.php/journals/2157

```cpp
template <class T>
concept bool EqualityComparable() {
    return requires(T a, T b) {
        {a == b} -> Boolean; // Boolean
        {a != b} -> Boolean;
    };
}
```

```cpp
void f(const EqualityComparable&); // constrained function template declaration
```

```cpp
f(42); // OK, int satisfies EqualityComparable
```

572

Calendar operations

Calendar

Defined in header <chrono>
Defined in namespace std::chrono

last_spec (C++20)	tag class indicating the *last* day or weekday in a month (class)
day (C++20)	represents a day of a month (class)
month (C++20)	represents a month of a year (class)
year (C++20)	represents a year in the Gregorian calendar (class)
weekday (C++20)	represents a day of the week in the Gregorian calendar (class)
weekday_indexed (C++20)	represents the n-th weekday of a month (class)
weekday_last (C++20)	represents the last weekday of a month (class)
month_day (C++20)	represents a specific day of a specific month (class)
month_day_last (C++20)	represents the last day of a specific month (class)
month_weekday (C++20)	represents the n-th weekday of a specific month (class)
month_weekday_last (C++20)	represents the last weekday of a specific month (class)
year_month (C++20)	represents a specific month of a specific year (class)
year_month_day (C++20)	represents a specific year, month, and day (class)
year_month_day_last (C++20)	represents the last day of a specific year and month (class)
year_month_weekday (C++20)	represents the n-th weekday of a specific year and month (class)
year_month_weekday_last (C++20)	represents the last weekday of a specific year and month (class)
operator/ (C++20)	conventional syntax for Gregorian calendar date creation (function)

573

Time-zone operations

Time zone

Defined in header <chrono>
Defined in namespace std::chrono

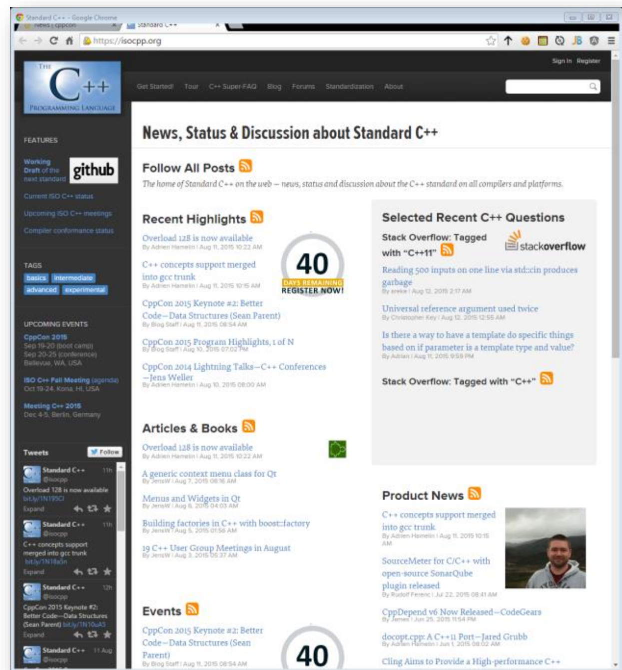
tzdb (C++20)	describes a copy of the IANA time zone database (class)
tzdb_list (C++20)	represents a linked list of tzdb (class)
get_tzdb get_tzdb_list reload_tzdb remote_version (C++20)	accesses and controls the global time zone database information (function)
locate_zone (C++20)	locates a time_zone based on its name (function)
current_zone (C++20)	returns the current time_zone (function)
time_zone (C++20)	represents a time zone (class)
sys_info (C++20)	represents information about a time zone at a particular time point (class)
local_info (C++20)	represents information about a local time to UNIX time conversion (class)
choose (C++20)	selects how an ambiguous local time should be resolved (enum)
zoned_traits (C++20)	traits class for time zone pointers used by zoned_time (class template)
zoned_time (C++20)	represents a time zone and a time point (class)
leap (C++20)	contains information about a leap second insertion (class)
link (C++20)	represents an alternative name for a time zone (class)
nonexistent_local_time (C++20)	exception thrown to report that a local time is nonexistent (class)
ambiguous_local_time (C++20)	exception thrown to report that a local time is ambiguous (class)

574

575

Standard C++

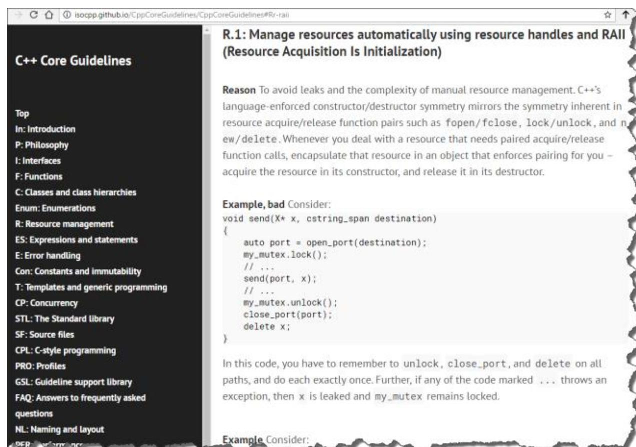
Keep track of the evolving C++ standard at isocpp.com



576

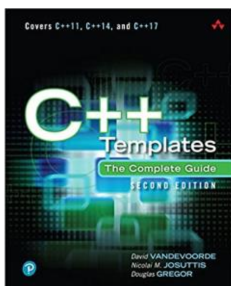
C++ Core Guidelines

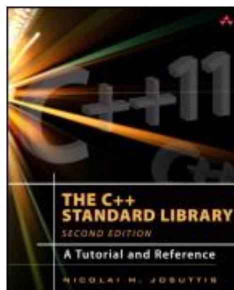
"The C++ Core Guidelines are a collaborative effort led by Bjarne Stroustrup, much like the C++ language itself. They are the result of many person-years of discussion and design across a number of organizations. Their design encourages general applicability and broad adoption but they can be freely copied and modified to meet your organization's needs."

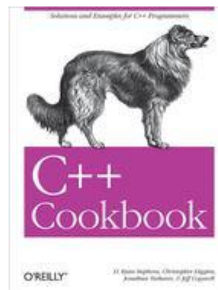


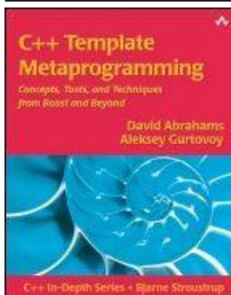
https://github.com/isocpp/CppCoreGuidelines

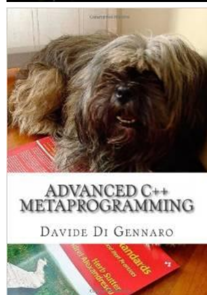
Books

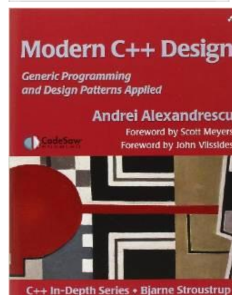








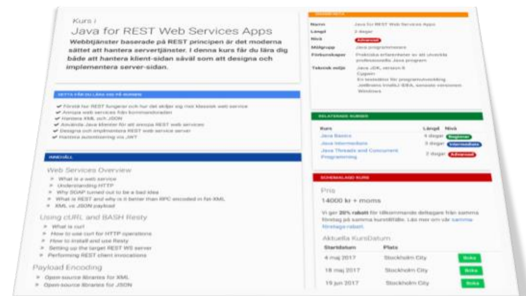




Related courses at Ribomation

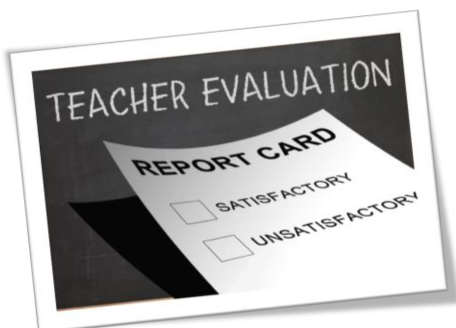
`https://www.ribomation.se/{topic}/{name}.html`

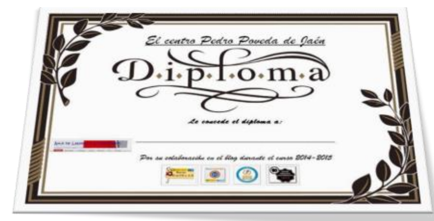
- Linux Systems Programming
  - cxx/cxx-systems-programming
- C++ Threads
  - cxx/cxx-threads
- C++ 11/14/17
  - cxx/cxx-17
- Unit Testing with Google Test
  - cxx/cxx-unit-testing
- Make for C/C++ Development
  - build/make



579

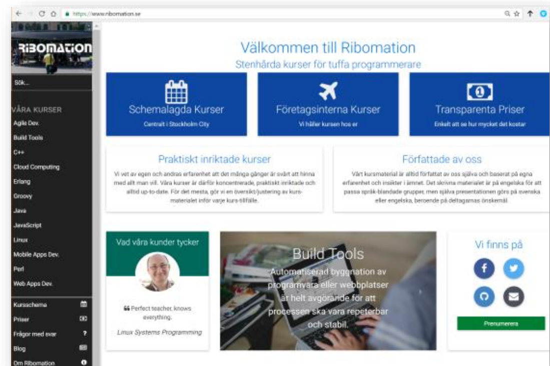
Course Evaluation & Diploma





580

Thanks & Goodbye



Email	jens.riboe@ribomation.se
Mobile	+46 (0)730-314-040
Web	www.ribomation.se
Skype	jensriboe
Twitter	twitter.com/jens_riboe
LinkedIn	linkedin.com/in/jensriboe

581