C++, objects, generic programming functional programming and clang/LLVM, and transpilers

Adam Boult (www.bou.lt)

April 29, 2024

Contents

I Basic changes from C **2**

1 Default function parameters **3**

2 Increments and decrements **4**

3 rvalues in C++ **5**

4 auto **6**

5 Reference data types in C++ **7**

6 Control flow **8**

7 Exception handling **9**

II Structs in C++ **10**

8 Adding methods to structs in C++ **11**

9 Struct inheritance in C++ **12**

10 Static variables in structs in C++ **14**

III Objects and classes **15**

11 Objects **16**

12 Object-Oriented Programming **18**

IV Generic programming **19**

13 Generic functions **20**

14 Generic classes **21**

15 Casting in C++ **22**

V Compiling C++ **23**

16 g++ **24**

17 cmake **25**

VI C++ libraries **26**

18 Packages and namespaces **27**

19 C standard library in C++ **28**

20 C++ Standard Library **29**

VII Parallel programming in C++ **30**

VIII Clang and LLVM **31**

21 Clang and LLVM **32**

Part I

Basic changes from C

Chapter 1

Default function parameters

1.1 Introduction

1.1.1 Introduction

can put default function parameters in function. unlike c can have default parameters for functions, have to be trailing parameters

Chapter 2

Increments and decrements

2.1 Introduction

2.1.1 Introduction

```
--a; // this was introduced in c++, not in c. c just has a-- (ditto for ++)

y=x++;
y=--x;
```

if x is 2, result of both is x=3 but top y=2, bottom y=3. order of evaluation.
does this apply to regular c?

Chapter 3

rvalues in C++

3.1 Introduction

3.1.1 Introduction

Chapter 4

auto

4.1 Introduction

4.1.1 Introduction

auto keyword in c++ mean don't have to label type if implied.

auto x=1L different meaning from auto in c

Chapter 5

Reference data types in C++

5.1 Introduction

5.1.1 Introduction

reference as variable type in c++ (c has pointers, and uses & operator, but can't do int &r, but can in cpp?)

Chapter 6

Control flow

6.1 Introduction

6.1.1 Introduction

same as c, also have foreach to iterate over arrays

```
    int vals[] {1, 2, 3, 4, 5};

    for (auto val : vals) {

        std::cout << val << std::endl;
    }
```

```
can use this over strings
  for (char c : str)
  {
    cout << "[" << c << "]";
  }
```

Chapter 7

Exception handling

7.1 Introduction

7.1.1 Introduction

c++ exception handling (not in c)

try

catch

throw

uses Resource acquisition is initialization (RAII) to implement?

for object to be initialised, it must have resources allocated.

all stack objects are destroyed (stack unwinding) if an exception is found

Part II

Structs in C++

Chapter 8

Adding methods to structs in C++

8.1 Introduction

8.1.1 Introduction

CPP constructor

Destructor

these can be done on structs? as can methods more generally? what is difference between structs and objects then? priv/pub stuff?

Chapter 9

Struct inheritance in C++

9.1 Introduction

9.1.1 Introduction

```
struct point_2d {
   int x;
   int y;
};

struct point_3d: point_2d {
    int z;
};

point_3d my_point;
my_point.x = 1;
my_point.y = 2;
my_point.z = 3;
```

9.1.2 Multiple inheritance

```
struct point_2d {
   int x;
   int y;
};

struct colour {
  char red;
  char green;
```

```
  char blue;
};

struct point_3d_colour: point_2d, colour {
  int z;
};
```

Chapter 10

Static variables in structs in C++

10.1 Introduction

10.1.1 Introduction

can do static on variable in struct in c++, can't in c

Part III

Objects and classes

Chapter 11

Objects

11.1 Introduction

11.1.1 Keys and values

11.1.2 Classes

11.1.3 Integer caching

If we set $x = 2$ we can either create 2 in memory, or simply point x to 2, which is already in memory

That means if we do $x = 2$ $y = 2$ they have the same pointer.

Can also cache some other common data values, eg empty lists.

Makes sense if pointer is smaller in memory than value.

11.2 Representing objects

11.2.1 Representing a single object

11.2.2 Null in objects

11.2.3 Representing a class with a multiple array (ie 2d)

11.2.4 Representing a class with a single array (ie 1d)

11.3 Functions with objects

11.3.1 Creating new objects

11.3.2 Getting values by field

11.3.3 Adding fields

11.3.4 Changing values in fields

11.4 Hierarchies of objects

11.4.1 Inheritance

Chapter 12

Object-Oriented Programming

12.1 Introduction

12.1.1 Introduction

in objects, OOP. essentially, all variable types are objects. inc integers, floats, lists etc

Part IV

Generic programming

Chapter 13

Generic functions

13.1 Introduction

13.1.1 Introduction

using multiple classes in a generic function function templates

```
template <class myType>
myType GetMax (myType a, myType b) {
 return (a>b?a:b);
}

int x,y;
GetMax <int> (x,y);
```

note: can use

```
template <class myType>
template <typename myType>
```

interchangeably

Chapter 14

Generic classes

14.1 Introduction

14.1.1 Introduction

using multiple classes in a generic class

Chapter 15

Casting in C++

15.1 Introduction

15.1.1 Introduction

in c had casting

casting

```
int value = 1;
float y = (float) value
```

cpp can also do

```
static_cast<float>(value)
```

other options in c++

```
reinterpret_cast<>()
const_cast<>()
dynamic_cast<>()
```

Part V

Compiling C++

Chapter 16

g++

16.1 Introduction

16.1.1 Introduction

gnu compiler collection includes gcc (gnu c compiler) and g++

Chapter 17

cmake

17.1 Introduction

17.1.1 Introduction

Part VI

C++ libraries

Chapter 18

Packages and namespaces

18.1 Introduction

18.1.1 Introduction

cpp double colon meaning

when do use by eg

```
#include <iostream>
std::cout
```

if want to just use eg cout

```
using namespace std;
cout
```

can use namespace in a specific scope, eg a function.

Chapter 19

C standard library in C++

19.1 Introduction

19.1.1 Introduction

Chapter 20

C++ Standard Library

20.1 Introduction

20.1.1 Introduction

Part VII

Parallel programming in C++

Part VIII

Clang and LLVM

Chapter 21

Clang and LLVM

21.1 Introduction

21.1.1 Introduction