C/C++: Lecture 1

Vorobev D.V

04.09.2020

Introduction

C was developed at Bell Labs company by Dennis Ritchie in 1972.

Introduction

Seven years later Bjourne Stroustroup (a software engineer at Bell Labs at that moment) started to develop a C extension - ”C with classes”

Introduction

The key requirements for the new language were:

1. the support of the high-level abstraction
2. closeness to the hardware

Introduction

Key dates:

1. 1998: C+98 standard (778 pages)
2. 2003: C+03 standard (786 pages)
3. 2007: Technical Report 1
4. 2011: C+11 standard (1350 pages)
5. 2014: C+14 standard (1380 pages)
6. 2017: C+17 standard (1580 pages)
7. 2020: C+20 standard (1780 pages)

Introduction

For 30 years the committee membership has increased from 46 up to 252 people

Nowadays the standard is released each 3 years

The main conferences in the scope of our course are CppCon and C++ Russia

Simple program structure

```
// the program entry point
int main() {
    return 0;
}
```

Scope

Definition

Potential Scope is a piece of the program from the point of the **definition** up to the end of the block (the first occurrence of the **}** symbol)

Definition

Actual Scope is the Potential Scope that does not take into consideration nested blocks with the definitions of the same name

Note

We have considered **Block Scope** above.
More formal definition you can find in the standard / cppreference

Scope

```
int main()
{
    int x = 0;
    {
        // the above x is hidden
        int x = 1;
    }
}
```

Errors

There are 2 classes of errors:

Compilation error

It is an error that does not allow you to get an executable file.

Runtime error

It is an error that occurs during the program execution. In more detail in the lecture about exceptions.

Compilation error

There are 3 types of compilation errors:

Lexical

It is an error related with usage of a symbol outside the **alphabet** of the language. It is detected at the first phase of the compilation pipeline - lexical analysis.

Syntactical

It is an error related with usage of **incorrect** syntactical construction. It is detected at the second phase of the compilation pipeline where the analyzer checks that the program belongs to the language produced by CFG (Context Free Frammar).

Semantical

It is an error related with use of construction not in accordance with its semantic meaning.

CE: Lexical

```
int main() {
    // cyrillic symbols
    int икс
    return 0;
}
```

CE: Syntactical

```
int main() {
    // missing character ;
    int x
    return 0;
}
```

CE: Semantical

```
int main() {
    // different types: "int" , "const char[2]"
    int x = "x";
    return 0;
}
```

Runtime error

Segmentation fault

It is an error that rises when we attempt to make r/w operation on restricted area of memory.

Stack overflow

It is an error that occurs after exceeding stack memory limitation as a consequences it leads to the segfault.

Runtime error

```
// Stack overflow
int main() {
    int x = 0;
    main();
    x++;
    return 0;
}
```

```
// Segmentation fault
int main() {
    int x[10];
    x[20000] = 10;
    return 0;
}
```

Identifier

Definition

Identifier is an arbitrarily long sequence of 0-9, _ , a-z, A-Z and most Unicode characters that does not started with 0-9.

```
int main() {
    // valid identifier
    int num_cars;

    // invalid identifier
    int 100500num_cars;

    return 0;
}
```

Variable

Definition

Variable = identifier + memory area.

```
int main() {
    int num_cars = 0;
    return 0;
}
```

Fundamental types

Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed		-128 to 127
		unsigned		0 to 255
	16	unsigned		0 to 65535
	32	unsigned		0 to 1114111 (0x10ffff)
integer	16	signed	$\pm 3.27 \cdot 10^{4}$	-32768 to 32767
		unsigned	0 to $6.55 \cdot 10^{4}$	0 to 65535
	32	signed	$\pm 2.14 \cdot 10^{9}$	-2,147,483,648 to 2,147,483,647
		unsigned	0 to $4.29 \cdot 10^{9}$	0 to 4,294,967,295
	64	signed	$\pm 9.22 \cdot 10^{18}$	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
		unsigned	0 to $1.84 \cdot 10^{19}$	0 to 18,446,744,073,709,551,615
floating point	32	IEEE-754	• min subnormal: $\pm$ 1.401,298,4 $\cdot 10^{-45}$ • min normal: $\pm$ 1.175,494,3 $\cdot 10^{-38}$ • max: $\pm$ 3.402,823,4 $\cdot 10^{38}$	• min subnormal: ±0x1p-149 • min normal: ±0x1p-126 • max: ±0x1.fffffep+127
	64	IEEE-754	• min subnormal: $\pm$ 4.940,656,458,412 $\cdot 10^{-324}$ • min normal: $\pm$ 2.225,073,858,507,201,4 $\cdot 10^{-308}$ • max: $\pm$ 1.797,693,134,862,315,7 $\cdot 10^{308}$	• min subnormal: ±0x1p-1074 • min normal: ±0x1p-1022 • max: ±0x1.fffffffffffffp+1023

Data model and specifiers

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least 16	16	16	16	16
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least 16	16	32	32	32
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least 32	32	32	32	64
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						
`long long`	`long long int` (C++11)	at least 64	64	64	64	64
`long long int`						
`signed long long`						
`signed long long int`						
`unsigned long long`	`unsigned long long int` (C++11)					
`unsigned long long int`						

Signed numbers representation

Let's n be the number of digits.
For certainty we put $n = 4$.
Methods:

- Signed magnitude representation
- Offset binary
- One's complement
- Two's complement

Signed magnitude representation

Requirements

The most significant bit is associated with the sign.
We introduce positive and negative zeros.

- $0000_2 = +0_{10}$
- $1000_2 = -0_{10}$

- $0101_2 \ = \ 5_{10}$
- $0101_2 \ = \ -5_{10}$

Problem

"+" in the decimal numeral system does not correspond "+" in the binary numeral system

$$5_{10} \ + \ (-2)_{10} \ = \ 3_{10}$$
$$0101_2 \ + \ 1010_2 \ = \ 1111_2$$
$$3_{10} \neq -7_{10}$$

Offset binary

Requirements

- $1000_2 = 0_{10}$
- **negative** number = number $< 1000_2$
- **positive** number = number $> 1000_2$

- $1011_2 \;=\; 3_{10}$
- $0011_2 \;=\; -3_{10}$

Problem

We can't obtain positive integer by raising digits to the power of 2.
In signed magnitude representation method we have this advantage.

$$3_{10} =\mid \text{def.} \mid= 1011_2 \;\neq 2^3 + 2^1 + 2^0 = 11_{10}$$

One's complement

Requirements

Negative number is an inversion of the positive number.
We introduce positive and negative zeros.

- $0000_2 = +0_{10}$
- $1111_2 = -0_{10}$

- $0101_2 \;=\; 5_{10}$
- $1010_2 \;=\; -5_{10}$

Problem

"+" in the decimal numeral system does not correspond "+" in the binary numeral system

$$5_{10} \;+\; (-1)_{10} \;=\; 4_{10}$$
$$0101_2 \;+\; 1110_2 \;=\; 0011_2$$
$$4_{10} \neq 3_{10}$$

Two's complement

Requirements

Negative number is an inversion of positive number + 1.

- $01100_2 \ = \ 12_{10}$
- $10100_2 \ = \ -12_{10}$

Benefits

The fact $x + \neg x = 2^n$ allows us to simplify summators designing.
The subtraction is calculated on the same hardware.

Implicit conversions

Numeric promotion

It is an **expression** conversion to more "general" type

Numeric conversion

It is an **expression** conversion to more "concrete" type

Implicit conversions

- Integral promotion
  - char → int
  - unsigned char → unsigned int
  - wchar_t → to the first type from the list able to hold the value range [int, unsigned int, long, unsigned int]
  - bool → int
- Float-point promotion
  - float → double

Implicit conversions

- Integral conversion to the unsigned destination value is the source value modulo 2^n, where n is a number of bits of the destination type
- Floating-integral conversion is a truncation of the fractional part.

Integral conversions

```
//  Integral conversion
int main()
{
    unsigned int x = 128000;
    unsigned short int y = x;
    // 62464
    std::cout << y;
    return 0;
}
```

```
//  Floating-integral conversion
int main()
{
    double x = 12.8;
    int y = x;
    // 12
    std::cout << y;
    return 0;
}
```

Constants

```
int main() {
    // a pointer to constant data
    const int* ptr1;

    // a constant pointer to data
    int* const ptr2 = new int(1);

    // a constant pointer to constant data
    const int* const ptr3 = new int(1);

    return 0;
}
```

One definition rule

Translation unit

It is a source file with literally included header files that are listed in #include

One definition rule

There is only one definition of any variable, function or class type is allowed in any one translation unit

Note

The same class can be defined in different translation units.

One definition rule: Not Allowed

```cpp
// file1.cpp
#include <iostream>

struct A {};
struct A {};

int main() {
    return 0;
}
```

```cpp
// file1.cpp
#include <iostream>

int x = 0;
int x = 0;

int main() {
return 0;
}
```

```cpp
// file1.cpp
#include <iostream>

void f() {};
void f() {};

int main() {
return 0;
}
```

One definition rule: Not Allowed

```cpp
// file1.cpp
#include <iostream>

int x = 0;
```

```cpp
// file2.cpp
#include <iostream>

int x = 0;
int main()
{
    return 0;
}
```

One definition rule: Not Allowed

```cpp
// file1.cpp
#include <iostream>

void f() {};
```

```cpp
//  file2.cpp
#include <iostream>

void f() {};

int main()
{
    return 0;
}
```

One definition rule: Allowed

```cpp
// file1.cpp
#include <iostream>

struct A {
    A() {std::cout << "file1";}
};
```

```cpp
//  file2.cpp
#include <iostream>

struct A {
    A() {std::cout << "file2";}
};
int main()
{
    A a;
    return 0;
}
```

Memory layout