Modern C++ - Program structure

3. The structure of C++ programs

C++ programs consist of - Comments - Preprocessor directives - C++ tokens

Comments:

```
1 // single line comment (since C99) from // to end of line
2
3   /* multi
4      line
5      comments
6      // hiding single line comments
7    */
8
9   /*
10       /*  but must not be nested */
11    */
```

No comments inside strings: “/* this is not comment */”

```
1 /************************************\
2 *                                    *
3 *  exist in various style and format  *
4 *                                    *
5 \************************************/
```

Preprocessor directives

starting with ‘#’

```
1 #ifdef
2 #define MY_HEADER
3 #include <header.h>
4
5 #if MY_PLATFORM
6 #define BUFFER 100
```

```
 7 #else
 8 #error platform not defined
 9 #endif
10
11 #endif
```

... more on preprocessor in the next lecture

C++ tokens

keywords

all lowercase

identifyers

starts with letter (incl _underscore) continues with letters or numbers

literals

- 1 decimal integer literal
- 0x14 hexadecimal integer literal
- 3.14 floating point literal
- 'x' character literal
- "Hello world" string literal

operators

- unary e.g. -2
- binary e.g. 2 + 4
- terciary x<y ? x : y

separators

{ } , ;

```
1 //
2 // C++ style
3 //
4 #include <iostream>   <---- preprocessor directvive
5 #include <string>     <---- preprocessor directvive
6
7 std::string myname();    std <-- namespace name: identifier
```

```
 8                          string <-- class name: identifier
 9                          myname <-- function name: identifier
10                          ::     <-- scope: operator
11                          ()     <-- function call: operator
12 int main()               int    <-- type name: keyword
13                          main   <-- function name: identifier
14                          ()     <-- function call: operator
15 {                        {      <-- block begin: separator
16                          cout   <-- variable: identifier
17   std::cout <<           <<     <-- output : operator
18                  "my name is: " <-- string literal, type char[13]
19             << myname()
20             << std::endl;     ; <-- command-end separator
21                          return <-- keyword
22   return 0;              0      <-- decimal int literal, type int
23 }
24 std::string myname()
25 {
26   return "John Cepepe Doe";  // automatic conversion from char * to
std::string
27 }
```

Keywords

we will learn them ...

Identifiers

- starts with letter (underscore '_' is a letter)
- continues with letters and digits
- no maximum length
- but compilers should translate
- only the first 63 letters (internal linkage)
- only the first 31 letters (external linkage)
- must not use keywords as identifiers
- case sensitive

Different conventions:

- camelCaseNotation
- CTypenamesStartsWithUppercase

- under_score_notation

A research on this: https://whathecode.wordpress.com/2011/02/10/camelcase-vs-underscores-scientific-showdown/

results on February 2015: - camelCase: 52.5 % - underscore: 47.5 %

MACRO_NAMES_ARE_ALL_UPPERCASE by convention

A paper on this: http://www.cs.loyola.edu/~binkley/papers/icpc09-clouds.pdf

Literals

void

Notes the return types of functions with no return value.

Integral types

- decimal integral 12 type = int, value = 12
- octal integral 014 type = int, value = 12
- hexadecimal integral 0xC type = int, value = 12
- long integer 12l type = long int, value = 12
- long long integer (since C99)
- unsigned integer 12u type = unsigned int, value = 12
- signed integers are the same as integers
- if an integer literal longer than integer 10000000 –> long or long long
- int size knot known ==> the best for the compiler, machine word

```
sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
// at least 16 bit             at least 32 bit    at least 64 bit
```

- also exist signed and unsigned versions, e.g. long <==> signed long

Characters

- character ‘A’ type = char, value = ascii value of ‘A’, likely 65
- escape sequences:
  - ’’’ single quote
  - ’"’ double quote
  - ’\?’ question mark

    - '\' backslash
    - '\a' bell (audio)
    - '\b' backspace
    - '\f' form feed – new page
    - '\n' newline
    - '\r' carriage return
    - '\t' horizontal tab
    - '\v' vertical tab
- octal value: type = char
    - '\377' -> 11111111
- hexadecimal value type = char
    - '\xff' -> 11111111
- unicode value
    - '\U1234' type = char16_t (min 16bit)
    - '\U12345678 type = char32_t (min 32bit)
- wchar_t is the longest character type
- signed char and unsigned char also exist, but here char not necessary the same as signed char

```
1 == sizeof(char) < sizeof(char16_t) <= sizeof(char32_t) <= sizeof(wchar_t)
```

Boolean

Boolean type **bool** is considered as *integral type*.

Floating point numbers

- float 3.14f
- double 3.14 (as double precision)
- long double 3.14l

```
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

Complex floating points

Declared as a template type in **<complex>** header. The template parameter can be any floating point type.

Size is not fixed! (C++ is NOT Java!)

int ==> best for integral computation, "machine word"

double ==> best for floating point computation

```cpp
//
// This is _very_ compiler/platform specific
//
#include <iostream>

int main()
{
  std::cout << sizeof(char) << std::endl;
  std::cout << sizeof(bool) << std::endl;
  std::cout << sizeof(short) << std::endl;
  std::cout << sizeof(long) << std::endl;
  std::cout << sizeof(long long) << std::endl;  // only from C++11
  std::cout << sizeof(float) << std::endl;
  std::cout << sizeof(double) << std::endl;
  std::cout << sizeof(long double) << std::endl;
  std::cout << sizeof(int*) << std::endl;
  std::cout << sizeof("Hello world") << std::endl;

  return 0;
}
```

```
$ ./a.out
1
1
2
4
8
8
4
8
16
8
12
```

There is a standard library header **<limits>** containing MIN and MAX values for various types.