

Lecture #2 | C/C++: types, variables and control flows

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

A few announcements

- Recommend books
 - Programming: Principles and Practice Using C++, 2nd ed.
 - Jumping into C++
 - The C++ Programming Language, 4th ed.
 - Learn C++: <http://www.learncpp.com/>
- Penalty for absence: $\text{pow}(2, \max(n - 3, 0))$

Today's topic

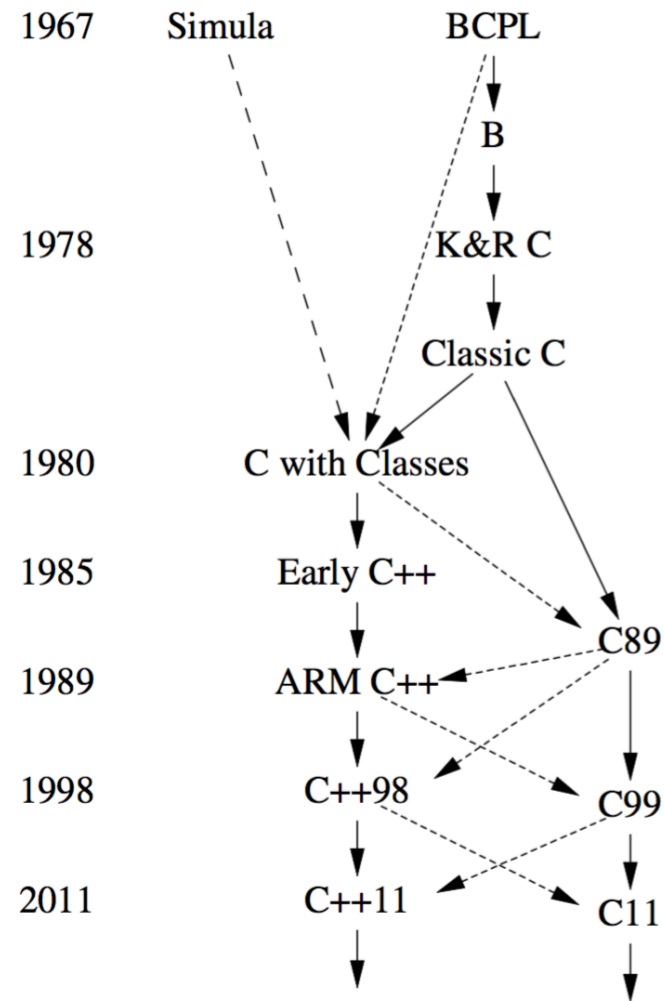
- Hello, world!
 - Comments
 - `main()` function
 - Header files
- Basic language features
 - Values and statements
 - Data types
 - Variables
 - Operators
 - Control flows: `if`, `while`, `do`, `for`, `switch/case`

History of C/C++

- B.C. Early languages, e.g., Fortran, Cobol, Algol, PL/I
- 1970 Kernigham and Ritchie invents C (K&R C)
- 1980 Bjarne Stroustrup creates "C with classes"
- 1983 "C with classes" is renamed as "C++"
- 1989 The C standard is ratified (ANSI C, C89)
- 1995 The ANSI committee releases a draft of the C++ standard
- 1998 An official C++ standard is adopted (C++98)
- 1999, 2011 The C standard is updated (C99, C11)
- 2003, 2011, 2014 The C++ standard is updated (C++03, C++11, C++14)

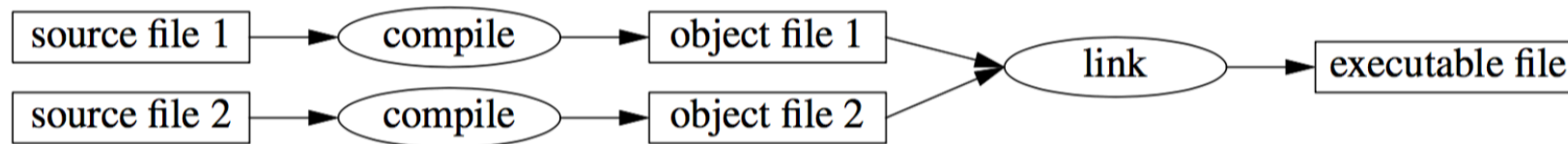


Revisit C/C++ history: C and C++ are siblings



C++ is a compiled language

- C++ source codes should be compiled (and linked) into machine language for execution



- We need to generate machine language for each processor, OS (and often times) and library combination
- During this course, each student is expected to set up his/her own development environment
- You can also use elice.io for easier access/assignment submission

Hello, world!

```
01 /*
02  * hello_world.cpp
03  */
04 #include <iostream>
05
06 int main(void)
07 {
08     // outputs "Hello, World!" and changes a line
09     std::cout << "Hello, World!" << std::endl;
10     return 0;
11 }
```

Anatomy of Hello, world!

- Comments
 - `//`: single line comment (until the end of line)
 - `/* */`: multiline (or block) comment, cannot be nested
- Statement: a unit of code that does something. Ends with `;`
- Code block: collection of statements. Denoted by `{ }`
- `main()` function: the starting point of C++ program
- Header files: pre-/user-defined source codes, typically with declaration of functions, classes, etc. Include by `#include` compiler directive
- return statement: specifies a return value of a function
- Standard output with `iostream` library
 - `std::cout`: standard out, a.k.a., console
 - `std::endl`: insert a new line character and flushed the stream (similar to `'\n'`)
 - `<<`: "put to" operator writes its second argument onto its first, while returning its first

Tokens: minimal chunk of program

Token type	Description/Purpose	Examples
Keywords	Words with special meaning to the compiler	int, double, for, auto
Identifiers	Names of things (e.g., variables, classes) that are not built into the language	cout, std, x, myFunction
Literals	Basic constant values whose value is specified directly in the source code	"Hello, world!", 24.3, 0, 'c'
Operators	Mathematical or logical operations	+, -, &&, %, <<
Punctuation/ Separators	Punctuation defining the structure of a program	{ } () , ;
Whitespace	Spaces of various sorts; ignored by the compiler	Spaces, tabs, newlines, comments

Statements, expressions and values

- A *statement* is a program element that control how and in what order objects are manipulated, which ends with ;
- An *expression* is a sequence of operators and operands that specifies a computation. An expression can result in a *value* and can cause side effects.
 - A number: 4, 1.3
 - A string: "Hello, world!"
 - Operations: 4+3
 - Function call: `sqrt()`

Built-in data types of C++

Category	Type	Min. Size (byte)	Example	Note
boolean	bool	1	true, false	
character	char	1	'a', 'b', ...	signed or unsigned
	wchar_t	1		
	char16_t	2		C++11 type
	char32_t	4		C++11 type
integer	short	2	0, -1, 1, 123, ...	
	int	2		
	long	4		
	long long	8		C99/C++11 type
floating point	float	4	1.0, 0.1, 3.14, 3e3, 2.4e-10	
	double	8		
	long double	8		

What is a bit and a byte?

- bit: stores 1 binary value, i.e., 0 or 1

- byte: 8 bits

--	--	--	--	--	--	--	--

Variables

- Variable declaration (type variable_name)

```
double d;
```

```
int i;
```

- A variable names is
 - Composed of alphabets, numbers, _
 - Starting with alphabets or _
 - Case-sensitive

- Assignment

```
d = 2.3;
```

```
i = 2 * 2;
```

```
i = 2 * 2.2; // evaluates to 4 (implicit conversion)
```

Variables (cont.)

- A variable can be initialized when it is declared with = or {}
 - = dates back to C, more commonly used
 - {} is new for all data types; it is more flexible and prevents you from conversions that lose information (since C++11)

- Examples

```
double d1 = 2.3;
```

```
double d2 {2.3};
```

```
double d3 = {2.3}; // = is redundant
```

```
int i1 = 7.2;    // i1 becomes 7
```

```
int i2 {7.2};    // error: compile time error
```

Operators (not comprehensive)

- Operators act on expressions to form a new expression (e.g., $(4 + 2) / 3$)
- Types of operators
 - Arithmetic: $+$, $-$, $*$, $/$, $\%$,
 - Increment, decrement: $++$, $--$
 - Comparison (or relational): $==$, $!=$, $>$, $<$, $>=$, $<=$
 - Logical: $\&\&$, $||$, $!$
 - Bitwise: $\&$, $|$, $^$
 - Assignment: $=$, $+=$, $-=$, $*=$, $/=$, \dots
 - Member access: $[]$, $*$, $\&$, $->$, $.$
 - Other: `sizeof()`, conditional ($? :$), comma ($,$)

Increment & decrement operators

- Pre-increment: increment the given variable first, and then return the new value
- Post-increment: return the current value (copy it if needed), and increment the given variable

```
int x, y;  
// Increment operators  
x = 1;  
y = ++x;    // x is now 2, y is also 2  
y = x++;    // x is now 3, y is 2  
// Decrement operators  
x = 3;  
y = x--;    // x is now 2, y is 3  
y = --x;    // x is now 1, y is also 1
```


Logical operators: &&, ||, !

- The position of operands matters when they have side effects


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

int main(void)
{
    int result, x = 1, y = 2; // not recommended
    ++x > 2 && ++y > 2;
    cout << "x=" << x << ", y=" << y << endl;
    ++x > 2 || ++y > 2;
    cout << "x=" << x << ", y=" << y << endl;
}
```

sizeof(): returns the size of a variable/data type

```
#include <iostream>
using namespace std;
int main(void)
{
    cout << "char: " << sizeof(char) << endl;
    cout << "int: " << sizeof(int) << endl;
    cout << "long: " << sizeof(long) << endl;
    cout << "long long: " << sizeof(long long) << endl;
    cout << "float: " << sizeof(float) << endl;
    cout << "double: " << sizeof(double) << endl;
    cout << "long double: " << sizeof(long double) << endl;
}
```

char: 1
int: 4
long: 8
long long: 8
float: 4
double: 8
long double: 16


***This result may differ
system by system!!!***

Conditional operator: ? :

- *condition* ? *X* : *Y* → If *condition* is true, evaluates to *X*; otherwise *Y*

```
int result;
```

```
int i = 5;
```

```
int j = 3;
```

```
result = i > j ? i : j; // result takes 5
```

```
// equivalent code w/o conditional operator
```

```
if (i > j)
```

```
    result = i;
```

```
else
```

```
    result = j;
```

Control flows: if

- `if (condition) statement;`
- `if (condition) statement; else statement;`

```
int i = 2;
```

```
int j = 3;
```

```
if (i > j)
```

```
    cout << "i is greater than j" << endl;
```

```
else {
```

```
    cout << "i is NOT greater than j" << endl;
```

```
    cout << "i is either smaller than or equal to j" << endl;
```

```
}
```

Blocks

- A sequence of statements delimited by curly braces ({ and }) is called a *block* or a *compound statement*
- Since control flows in C/C++ takes only one statement, you need to use blocks to have more than one statement
- Note: Unlike python, white spaces (or indentations) are ignored by compilers. But it is **STRONGLY** recommended put adequate and consistent indentations in your code!

Control flows: while & do

- `while (condition) statement;`
→ *statement* may not be executed
- `do statement while (expression);`
→ *statement* is executed at least once

```
i = 3
```

```
while (i > 0) {  
    cout << "i=" << i-- << endl;  
}
```

```
do {  
    cout << "i=" << i++ << endl;  
} while (i < 5);
```

Control flows: for

- `for (init-statement; conditionopt; expressionopt) statement;`

```
for (i = 0; i < 5; i++)  
    cout << "i=" << i << endl;
```

```
cout << "----" << endl;
```

```
// equivalent code
```

```
i = 0;  
while (i < 5) {  
    cout << "i=" << i << endl;  
    i++;  
}
```

Control flows: switch/case

```
switch (i) {  
case 1:  
    cout << "i is 1" << endl;  
    break;  
case 2:  
case 3:  
    cout << "i is 2 or 3" << endl;  
    break;  
default: // default is NOT required  
    cout << "i is neither 1, 2, nor 3" << endl;  
}
```




ANY QUESTIONS?