C++ for C Coders 2

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

C++ as a better C
namespace
scope resolution operator
GCC compiler options
input/output
inline function

C++ as a better C

- Comment Lines

```
/* This is a comment */
// This is a comment
```

- Declarations and definitions can be placed anywhere.
- A variable lives only in the block, in which it was defined.
  This block is called the **scope** of this variable.

```
int a = 0;
for (int i = 0; i < 100; i++){ // i is declared in for loop
  a++;
  int p = 12;                  // declaration of p ...
  ...                          // scope of p
} // end of scope for i and p
```

namespaces

- Identifiers in a header file used in a program become global.
  - Syntax error occurs if an identifier in a program has the same name as a global identifier in the header file.
- Same problem can occur with third-party libraries.
  - Common solution: third-party vendors begin their global identifiers with _ (underscore)
  - Do not begin identifiers in your program with _
- ANSI/ISO Standard C++ attempts to solve this problem with the **namespace.**
- Syntax:

```
namespace namespace_name {
    members
}
```

  - A member is usually a variable declaration, a named constant, a function, or another namespace.

namespaces

Briefly, it's a fence!

```
namespace namespace_name {
    members
}
```

namespaces

Example

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

namespace one { int var = 5; }
namespace two { int var = 3; }

int main(int argc, char **argv} {
  std::cout << one::var << std::endl;
  std::cout << two::var << std::endl;
  return 0;
}
```

namespaces

Example

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

namespace one { int var = 5; }
namespace two { int var = 3; }

int main(int argc, char **argv} {
  std::cout << one::var << std::endl;
  std::cout << two::var << std::endl;
  return 0;
}
```

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

namespace one { int var = 5; }
namespace two { int var = 3; }

int main(int argc, char **argv} {
  cout << one::var << endl;
  cout << two::var << endl;
  return 0;
}
```

```
5
3
```

namespaces

Example

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

namespace one { int var = 5; }
namespace two { int var = 3; }

int main(int argc, char **argv} {
  std::cout << one::var << std::endl;
  std::cout << two::var << std::endl;
  return 0;
}
```

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

namespace one { int var = 5; }
namespace two { int var = 3; }

int main(int argc, char **argv} {
  cout << one::var << endl;
  cout << two::var << endl;
  return 0;
}
```

5
3

Scope resolution operator ::

The scope resolution operator :: is used for following purposes.

1. To access a global variable when there is a local variable with same name.
2. To define a function outside a class.
3. To access a class's static variables.
4. In case of multiple Inheritance.

Scope resolution operator ::

- A definition in a block can hide a definition in an enclosing block or a global name. It is possible to use a hidden global name by using the **scope resolution operator ::**

```
int y = 0;          // Global y
int x = 1;          // Global x
void f(){           // Function is a new block
  int x = 5;        // Local  x=5, it hides global x
  ::x++;            // Global x=2
  x++;              // Local x=6
  y++;              // Global y=1
}
```

More GCC Compiler Options

- There are some flags available for compiler options:

```
$ g++ -std=c++11 -Wall -g file1.cpp file2.cpp -o prog
```

  - **-o** : specifies the output executable filename.
  - **-std=c++11** : to specify the C++ standard version
    c++11, c++14, c++17, c++2a
  - **-Wall** : enables most warning messages
  - **-g** : for use with gdb debugger.
  - **-DDEBUG** : define "DEBUG" as a macro
  - **-I** : specifies the include file folder
  - **-L** : specifies the library file folder
  - **-l** : specifies the lib (lib~.a) filename

Input/Output

- When a C++ program includes the iostream header, four objects are created and initialized:
  - **cin** handles input from the standard input, **the keyboard**.
  - **cout** handles output to the standard output, **the screen**.
  - **cerr** handles unbuffered output to the standard error device, the screen.
  - **clog** handles buffered error messages to the standard error device.

Input/Output – Using cin object

- cin is used with >> to gather input

```
cin >> variable;
```

- The stream extraction operator is >>

- For example, if miles is a **double** variable

```
cin >> miles;
```

  - Causes computer to get a value of type **double**
  - Places it in the variable miles

Input/Output – Using cin object

- The predefined **cin** stream object is used to read data from the standard input device, usually the keyboard.
- The **cin** stream uses the >> operator, usually called the **"get from" operator**.

```
#include<iostream>
using namespace std;                    // we don't need std:: anymore

int main() {
  int i, j;                             // Two integers are defined
  cout << "Give two numbers \n";        // cursor to the new line

  cin >> i >> j;                        // Read i and j from the keyboard
  cout << "Sum= " << i + j << "\n";
  return 0;
}
```

Input/Output – Avoid using cin >> if you can

- Using `cin` to get user input is convenient sometimes since we can specify a primitive data type. However, it is notorious at causing input issues because it doesn't remove the newline character from the stream or do type-checking. So anyone using `cin >> var;` and following it up with another `cin >> stringtype;` or `std::getline();` will receive empty inputs. It's the best practice not to mix the different types of input methods from `cin`.

- Another disadvantage of using `cin >> stringvar;` is that `cin` has no checks for length, and it will break on a space. So you enter something that is more than one word, only the first word is going to be loaded. Leaving the space, and following word still in the input stream. **This may cause an error.**

- JoyNote: A more elegant solution, much easier to use, is the `std::getline()`.
- Reference https://github.com/idebtor/nowic/blob/master/02GettingInput.md

Input/Output – Using cout object

- The syntax of **cout** and **<<** is:

```
cout << expression or manipulator << ··· ;
```

- Called an output statement
- The stream insertion operator is **<<**
- Expression evaluated and its value is printed at the current cursor position on the screen

Input/Output

TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

inline functions

- In C, macros are defined by using the #define directive of the preprocessor.

```
#define sq(x) ((x)*(x))
#define max(x, y) (y < x ? x : y)
```

- In C++, macros are defined as normal functions.
  Here the keyword **inline** is inserted before the declaration of the function.

```
inline int sq(int x) { return x * x; }
inline int max(int x, int y) { return y < x ? x : y); }
```

inline functions

- An inline function is defined using almost the same syntax as an ordinary function. However, instead of placing the function's machine-language code in a separate location, the compiler simply inserts it into the location of the function call.
- It's appropriate to inline a function when it is short, but not otherwise. If a long or complex function is **inlined**, too much memory will be used and not much time will be saved.
- Advantages
  - Debugging
  - Type checking
  - Readable

(2)

C++ For C Coders 2

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

C++ as a better C
namespace
scope resolution operator
GCC compiler options
input/output
inline function