

C++ Jump Start

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

C++ for C Coders

Getting Started

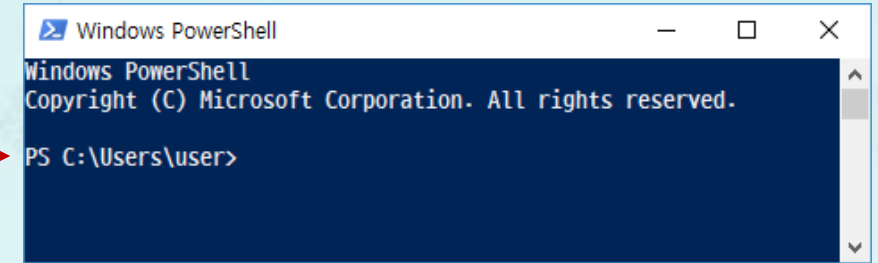
- Creating and using a console
 - cmd, PowerShell
 - bash (Borne-again) shell
 - Linux bash etc.
- Install GNU C/C++ compiler
- Install Atom
- Install Git and GitHub Desktop
- Read
 - github.com/idebtor/nowic/**README**
 - github.com/idebtor/nowic/**GettingStarted**

C++ for C Coders

- C vs C++
- "Hello World!" program
- Scope resolution operator
- Call-by-value vs. Call-by-reference
- Overloading
- Input/Output
- Command line processing
- Labs and Quizzes

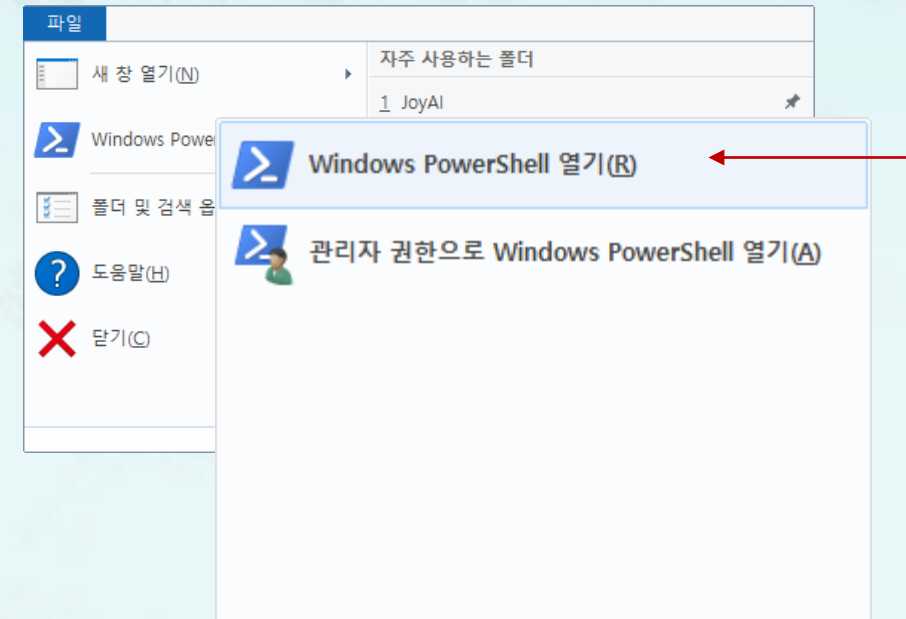
Start a console

1. Start PowerShell through "Start" menu



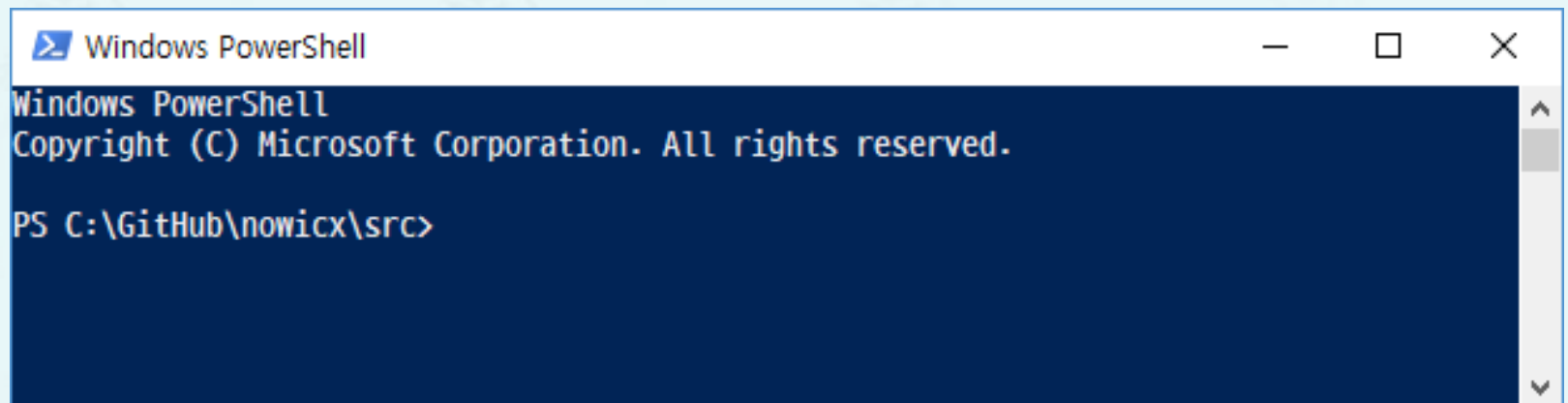
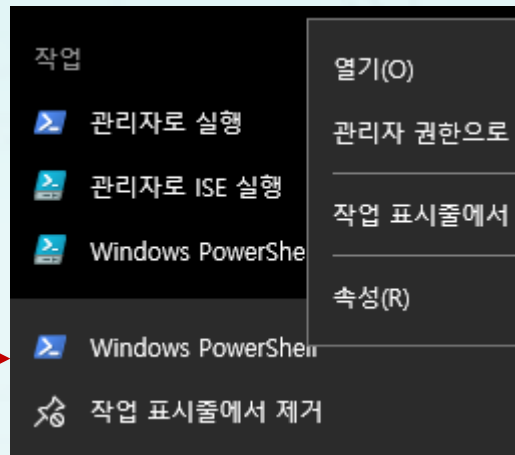
2. Move to the folder you want to start.

- Click the "File" menu at the top left corner of PowerShell or cmd windows
- Enter a command at console. For example, **atom myfile.txt**



Start a console at your favorite folder

1. Start PowerShell if you don't have a PowerShell icon in Taskbar.
2. In the Taskbar right-click and pin to keep a link there.
3. **Again** right click the icon in taskbar and then right-click Windows PowerShell and choose Properties
4. Enter your preferred directory in the Start in: input field and press OK.
5. Start from the taskbar icon



PowerShell, Cmd, vs Bash (Borne-again shell), sh, ksh, csh

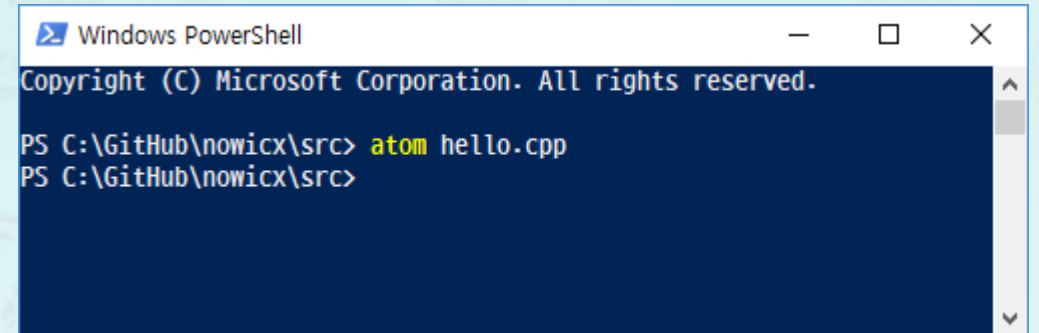
PowerShell (Cmdlet)	PowerShell (Alias)	Windows Cmd	Unix shell	Description
Get-ChildItem	gci, dir, ls	dir	<u>ls</u>	Lists all files and folders in the current or given folder
Get-Content	gc, type, cat	type	cat	Gets the content of a file
Get-Command	gcm	help	type, which	Lists available commands
Get-Help	help, man	help	apropos, man	Prints a command's documentation on the console
Clear-Host	cls, clear	cls	clear	Clears the screen
Copy-Item	cpi, copy, cp	copy, xcopy	<u>cp</u>	Copies files and folders to another location
Move-Item	mi, move, mv	move	mv	Moves files and folders to a new location
Remove-Item	ri, del, rmdir, rd, rm	del, erase, rmdir, rd	rm, rmdir	Deletes files or folders
Rename-Item	rni, ren, mv	ren, rename	mv	Renames a single file, folder, hard link or symbolic link
Get-Location	gl, cd, pwd	cd	pwd	Displays the working path (current folder)
Pop-Location	popd	popd	popd	Changes the working path to the location most recently pushed onto the stack
Push-Location	pushd	pushd	pushd	Stores the working path onto the stack
Set-Location	sl, cd, chdir	cd, chdir	cd	Changes the working path
Write-Output	echo, write	echo	echo	Prints strings or other objects to the standard output
Get-Process	gps, ps	tlist, tasklist	ps	Lists all running processes
Stop-Process	spps, kill	kill, taskkill	kill	Stops a running process
Select-String	sls	findstr	find, grep	Prints lines matching a pattern
Set-Variable	sv, set	set	env, export, set, setenv	Creates or alters the contents of an environment variable

Install GNU C/C++ Compiler

- Install **MinGW-w64**
<https://sourceforge.net/projects/mingw-w64/>
- Install MSYS2
<http://www.msys2.org/>
- Add PATH
 - c:/msys64/user/bin

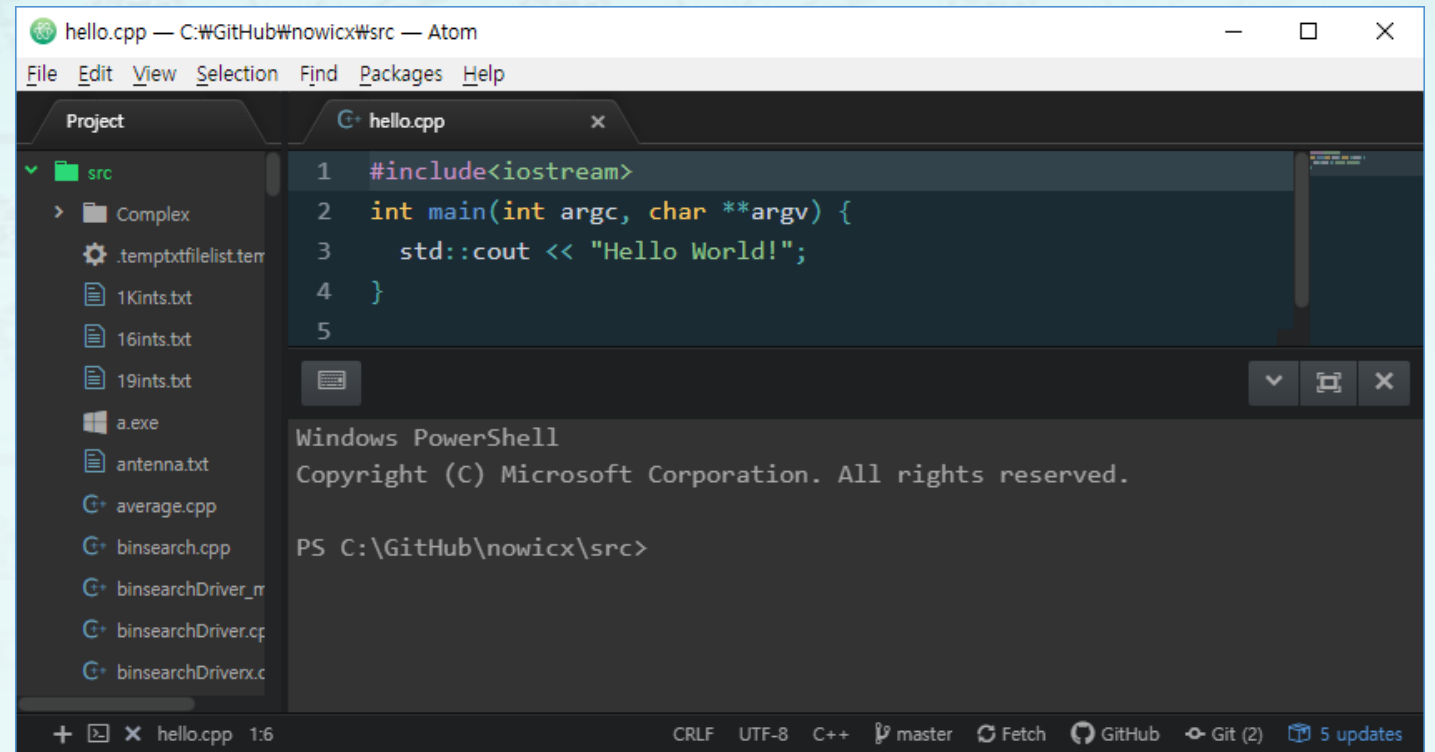
Install Atom

1. Install **Atom**.
2. Start Atom to edit a text file at a console.
 - start a console (or console window)
 - select Windows PowerShell
 - start the following command at console
atom hello.cpp
3. Add the following packages
 - gpp-compiler
 - Platformio-ide-terminal
 - File-icons
 - Markdown-preview
 - Mini-maps



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\GitHub\nowicx\src> atom hello.cpp
PS C:\GitHub\nowicx\src>
```



Install Git and GitHub Desktop

- **Git**
 - a revision control system,
 - a tool to manage your source code history
- **Github**
 - a hosting service for Git repositories.
- **Github Desktop**
 - a Windows interface for Git/GitHub,
 - a subset of Git commands supported

1. Install Git.
2. Install Github Desktop
3. Clone the following repository
 - github/idebtor/nowic




Install Git and GitHub Desktop

Setting the starting folder of Git

1. Right click on the Git Bash shortcut icon
→ *Start in:* → C:\github\nowic
2. Remove the **--cd-to-home** part from the Target box,



C++ for C Coders

- C
 - Dennis Ritchie
 - 1972
 - 29 Keywords
 - Imperative programming
 - Procedural programming
 - C++
 - Stroustrup
 - 1985
 - 63 Keywords by 1996
 - C++ as a better C
 - Imperative programming
 - Object-oriented programming
 - Generic programming
- Declarative programming
- 

명령형 (C, C++, Java) vs 선언형 (SQL, HTML)

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
```

Write "Hello World" program in C++

- Open Atom editor
- Open a new file
 - Filename: **hello.cpp**
- Add the source code.
- Save the file.
- Compile and Execute

```
$ atom hello.cpp
```

```
$ g++ hello.cpp
```

```
$ ./a.exe
```

```
$ ./a
```

```
$ a
```

PowerShell

OSX

cmd

```
// file: hello.cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

Write "Hello World" program in C++

- Open Atom editor
- Open a new file
 - Filename: **hello.cpp**
- Add the source code.
- Save the file.
- Compile and Execute

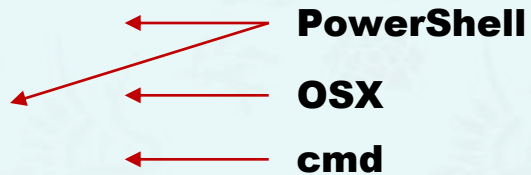
```
$ atom hello.cpp
```

```
$ g++ hello.cpp
```

```
$ ./a.exe
```

```
$ ./a
```

```
$ a
```



```
// file: hello.cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

```
// file: hello.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
}
```

Write "Hello World" program in C++

- Open Atom editor
- Open a new file
 - Filename: **hello.cpp**
- Add the source code.
- Save the file.

- Compile and Execute

```
$ g++ hello.cpp
```

```
$ ./a.exe
```

```
$ ./a
```

- Compile and Execute

```
$ g++ hello.cpp -o hello
```

```
$ ./hello.exe
```

```
$ ./hello
```

```
// file: hello.cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
}
```

```
// file: hello.cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
}
```

namespaces

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

```
namespace namespace_name {  
    members  
}
```


namespaces

Briefly, it's a fence!



```
namespace namespace_name {  
    members  
}
```

namespaces

Example

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

namespace first { int var = 5; }
namespace second { int var = 3; }

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

namespaces

Example

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

namespace first { int var = 5; }
namespace second { int var = 3; }

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

5 3

namespaces

Example

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

namespace first { int var = 5; }
namespace second { int var = 3; }

int main(int argc, char **argv) {
    std::cout << first::var << std::endl << second::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 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
}
```

Scope Operator ::

- Lab

```
int i = 1;
int main(){
    int i = 2; {
        int n = i;
        int i = 3;
        cout << i << " " << ::i << endl;
        cout << n << "\n" ;
    }
    cout << i << " " << ::i << endl;
    return 0;
}
```


Scope Operator ::

- Lab

```
int i = 1;
int main(){
    int i = 2; {
        int n = i;
        int i = 3;
        cout << i << " " << ::i << endl;
        cout << n << "\n" ;
    }
    cout << i << " " << ::i << endl;
    return 0;
}
```

3	1
2	
2	1

Multiple Source Files

- Open Atom editor
- Open two new files
 - Filename: **greet.cpp**
 - Filename: **greet_func.cpp**
- Add the source code.
- Save files.

```
// file: greet_func.cpp
#include <iostream>
void greet_func() {
    std::cout << "Hello World!";
}
```

```
// file: greet.cpp
#include <iostream>
extern greet_func();
int main() {
    greet_func();
}
```

- Compile and Execute

```
$ g++ greet.cpp greet_func.cpp -o greet
```

```
$ ./greet
```

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

Command line processing

- Open Atom editor
- Open a new file
 - Filename: **argcargv.cpp**
- Add the source code.
- Save the file.
- Compile and Execute

```
$ g++ argcargv.cpp -o arg
$ ./arg command line args
```

```
// Name of program argcargv.cpp
#include <iostream>
using namespace std;

int main(int argc, char** argv) {
    cout << "You entered: "
         << argc << " arguments:" << endl;

    for (int i = 0; i < argc; ++i)
        cout << argv[i] << endl;

    return 0;
}
```

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.
- 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
<code>\n</code>	Newline	Cursor moves to the beginning of the next line
<code>\t</code>	Tab	Cursor moves to the next tab stop
<code>\b</code>	Backspace	Cursor moves one space to the left
<code>\r</code>	Return	Cursor moves to the beginning of the current line (not the next line)
<code>\\</code>	Backslash	Backslash is printed
<code>\'</code>	Single quotation	Single quotation mark is printed
<code>\"</code>	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

Default Function Arguments

- In calling of the function, if the arguments are not given, default values are used.

```
int exp(int n, int k = 2) {  
    if (k == 2) return (n * n);  
    return (exp(n, k - 1) * n);  
}
```


Default Function Arguments

- In calling a function argument must be given from left to right without skipping any parameter

```
void foo(int i, int j=7);      // right
void goo(int i=3, int j);      // wrong
void hoo(int i, int j=3, int k=7); // right
void moo(int i=1, int j=2, int k=3); // right
void noo(int i=2, int j, int k=3); // right? wrong
```


Reference Operator &

- A reference allows to declare an alias to another variable.
- As long as the aliased variable lives, you can use indifferently the variable or the alias.

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

int main() {
    int x;
    int& foo = x;
    foo = 49;
    cout << x << endl;
    return 0;
}
```

Reference Operator &

- A reference allows to declare an alias to another variable.
- References are extremely useful when used with function arguments since it saves the cost of copying parameters into the stack when calling the function.

Reference Operator &

- Swap() example in C/C++

```
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main() {  
    int i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

5 3

Reference Operator &

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

void whatIsOutput(int& x, int y, int& z) {
    cout << x << " " y << " " << z << endl;
    x = 1;
    y = 2;
    z = 3;
    cout << x << " " y << " " << z << endl;
}

int main() {
    int a = 10, b = 20, c = 30;
    whatIsOutput(a, b, c);
    cout << a << " " << b << " " << c << endl;
    return 0;
}
```

What is the output?

Swap in C

```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    printf("in: %d, %d\n", a, b);  
    swap(&a, &b);  
    printf("out: %d, %d\n", a, b);  
  
}
```

& is an address operator.

Swap in C

```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    printf("in: %d, %d\n", a, b);  
    swap(&a, &b);  
    printf("out: %d, %d\n", a, b);  
  
}
```

& is an address operator.

```
void swap(int *i, int *j) {  
    int temp = *i;  
    *i = *j;  
    *j = temp;  
}
```

i is a pointer (or memory address) to an int.
*i is the memory contents that i points.

Swap in C

```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    printf("in: %d, %d\n", a, b);  
    swap(&a, &b);  
    printf("out: %d, %d\n", a, b);
```

& is an address operator.

```
    printf("in: %lf, %lf\n", a, b);  
    swap(&x, &y);  
    printf("out: %lf, %lf\n", x, y);  
}
```

```
void swap(int *i, int *j) {  
    int temp = *i;  
    *i = *j;  
    *j = temp;  
}
```

Any problem here?

No overloading in C

```
void swap(int *i, int *j) {  
    int temp = *i;  
    *i = *j;  
    *j = temp;  
}
```

```
void swap_double(double *i, double *j) {  
    double temp = *i;  
    *i = *j;  
    *j = temp;  
}
```

No overloading in C

```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    printf("in: %d, %d\n", a, b);  
    swap(&a, &b);  
    printf("out: %d, %d\n", a, b);  
  
    printf("in: %lf, %lf\n", a, b);  
    swap(&x, &y);  
    printf("out: %lf, %lf\n", x, y);  
}
```


```
void swap(int *i, int *j) {  
    int temp = *i;  
    *i = *j;  
    *j = temp;  
}  
  
void swap_double(double *i, double *j) {  
    double temp = *i;  
    *i = *j;  
    *j = temp;  
}
```

Swap in C++

& is a reference to int



```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    cout << "in:" << a << ", " << b << endl;  
    swap(a, b);  
    cout << "out:" << a << ", " << b << endl;  
  
    call by reference  
  
}
```



```
void swap(int& i, int& j) {  
    int temp = i;  
    i = j;  
    j = temp;  
}
```

Swap in C++

& is a reference to int



```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    cout << "in:" << a << ", " << b << endl;  
    swap(a, b);  
    cout << "out:" << a << ", " << b << endl;  
  
    cout << "in:" << x << ", " << y << endl;  
    swap(x, y);  
    cout << "out:" << x << ", " << y << endl;  
}
```

```
void swap(int& i, int& j) {  
    int temp = i;  
    i = j;  
    j = temp;  
}
```

C++ has both overloading and call by reference.

Swap in C++

& is a reference to int

```
int main() {  
    int a = 5, b = 6;  
    double x = 6.7, y = 8.9;  
  
    cout << "in:" << a << ", " << b << endl;  
    swap(a, b);  
    cout << "out:" << a << ", " << b << endl;  
  
    cout << "in:" << x << ", " << y << endl;  
    swap(x, y);  
    cout << "out:" << x << ", " << y << endl;  
}
```

call by reference

```
void swap(int& i, int& j) {  
    int temp = i;  
    i = j;  
    j = temp;  
}
```

```
void swap(double& i, double& j) {  
    double temp = i;  
    i = j;  
    j = temp;  
}
```

function overloading

C++ has both overloading and call by reference.

Reference Operator &

- Summary: swap() example in C/C++

C

```
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int main() {  
    int i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

C++

```
void swap(int& a, int& b) {  
    int temp = a;  
    a = b;  
    b = temp;  
}
```

```
int main() {  
    int i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

Reference Operator &

- Lab

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

int main(){
    int x = 2, y = 3, z = 4;
    squareByPointer(&x);
    cout<< x << endl;

    squareByReference(y);
    cout<< y << endl;

    z = squareByValue(z);
    cout<< z << endl;
}
```


Overloading

- Function overloading refers to the possibility of creating multiple functions with the same name as long as they have different parameters (type and/or number) which is called a signature of function.

C

```
int main() {  
    int i = 3, j = 5;  
    swap(&i, &j);  
    cout << i << " " << j << endl;  
}
```

C++

```
int main() {  
    int i = 3, j = 5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
}
```

const Reference

- To prevent the function from changing the parameter accidentally, we pass the argument as **constant reference** to the function.

```
struct Person {  
    char name[40];  
    int age;  
};  
  
void print(const Person& k) {  
    cout << "Name: " << k.name << endl;  
    cout << "Age: " << k.age << endl;  
}  
  
int main(){  
    Person man{"Adam", 316};  
    print(man);  
    return 0;  
}
```

← C style coding in C++

← k is constant reference parameter

Instead of 44 bytes, only 4 bytes (address) are sent to the function.

const Reference

- To prevent the function from changing the parameter accidentally, we pass the argument as **constant reference** to the function.

```
struct Person {  
    string name;  
    int age;  
};  
  
void print(const Person& k) {  
    cout << "Name: " << k.name << endl;  
    cout << "Age: " << k.age << endl;  
}  
  
int main(){  
    Person man{"Adam", 316};  
    print(man) ;  
    return 0;  
}
```

← **k is constant reference parameter**

Return by reference

- By default in C++, when a function returns a value, it is copied into stack. The calling function reads this value from stack and copies it into its variables.
- An alternative to “return by value ”is “return by reference”, in which the value returned is not copied into stack.
- One result of using “return by reference” is that the function which returns a parameter by reference **can be used on the left side** of an assignment statement.

Return by reference

- Example: Modify the following programs such that it sets the maximum element to zero.

```
int max(int a[], int N) {  
    int i=0;  
    for (int j=0 ; j<N; j++)  
        if (a[j] > a[i]) i = j;  
    return a[i];  
}  
  
int main() {  
    int array[] = {12, 42 , 33 , 99, 63};  
    int n = 5;  
  
    for (int i = 0; i < n; i++)  
        cout << a[i] << " ";  
}
```

12 42 33 0 63

Return by reference

- Example: Modify the following programs such that it sets the maximum element to zero.

```
int& max(int a[], int N) {    // returns an integer reference of the max element
    int i=0;
    for (int j=0 ; j<N; j++)
        if (a[j] > a[i]) i = j;
    return a[i];
}

int main() {
    int array[] = {12, 42 , 33 , 99, 63};
    int n = 5;

    max(array,5) = 0;        // overwrite the max element with 0
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
}
```

12 42 33 0 63

const return value

- **To prevent** the calling function from changing the return parameter accidentally, **const** qualifier can be used.

```
const int& max(const int a[], int N){
    int i=0;
    for (int j=0 ; j<N ; j++) {
        if (a[j] > a[i]) i = j;
    }
    return a[i];
}

int main(){
    int array[ ] = {12, 42 , 33 , 99, 63};
    int largest = max(array,5);
    cout << "The largest is " << largest << endl;
    return 0;
}
```


Never return a local variable by reference

- Since a function that uses “return by reference” returns an actual memory address, it is important that the variable in this memory location remain in existence after the function returns.

```
int& foo() {  
    int i;  
    ...  
    return i;                // ERROR! does not exist anymore
```

- Local variables can be return by their values

```
int foo() {  
    int i;                    // Local variable, created in stack  
    ...  
    return i;                 // OK! does not exist anymore
```

malloc & free vs new & delete

- In C, dynamic memory allocation is done with malloc() and free().
- The C++ new and delete operators performs dynamic memory allocation.

```
int *p = (int *)malloc(sizeof(int) * N);
```

```
for (int i = 0; i < N; i++)  
    p[i] = i;
```

```
free(p);
```

```
int *p = new int[N];
```

```
for (int i = 0; i < N; i++)  
    p[i] = i;
```

```
delete[] p;
```

Using new & delete

- The **new** operator allocates memory, and **delete** frees memory allocated by **new**.

```
int *pi = new int;           // pi points to uninitialized int
int *pi = new int(7);        // which pi points has value 7
string *ps = new string;     // empty string
```

Using new & delete

- The **new** operator allocates memory, and **delete** frees memory allocated by **new**.

```
int *pi = new int;           // pi points to uninitialized int
int *pi = new int(7);        // which pi points has value 7
string *ps = new string;     // empty string

int *pia = new int[7];       // block of seven uninitialized ints
int *pia = new int[7]();     // block of seven ints values initialized to 0
```

Using new & delete

- The **new** operator allocates memory, and **delete** frees memory allocated by **new**.

```
int *pi = new int;           // pi points to uninitialized int
int *pi = new int(7);        // which pi points has value 7
string *ps = new string;     // empty string

int *pia = new int[7];       // block of seven uninitialized ints
int *pia = new int[7]();     // block of seven ints values initialized to 0

string *psa = new string[5]; // block of 5 empty strings
string *psa = new string[5](); // block of 5 empty strings
int *pia = new int[5]{0, 1, 2, 3, 4}; // block of 5 ints initialized
string *psa = new string[2]{"a", "the"}; // block of 2 strings initialized
delete pi;
delete[] pia;
```

Lab 1: Convert a C program to C++

```
#include <stdio.h>
#define N 40
void sum(int d[], int n, int* p) {
    *p = 0;
    for(int i = 0; i < n; ++i) *p = *p + d[i];
}

int main() {
    int total = 0;
    int data[N];
    for(int i = 0; i < N; ++i) data[i] = i;
    sum(data, N, &total);
    printf("total is %d\n", total);
    return 0;
}
```

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

Lab 1: Convert a C program to C++

```
#include <stdio.h>
#define N 40
void sum(int d[], int n, int* p) {
    *p = 0;
    for(int i = 0; i < n; ++i) *p = *p + d[i];
}

int main() {
    int total = 0;
    int data[N];
    for(int i = 0; i < N; ++i) data[i] = i;
    sum(data, N, &total);
    printf("total is %d\n", total);
    return 0;
}
```

```
#include <iostream>
using namespace std;
void sum(int d[], int n, int& p) {
    p = 0;
    for(int i = 0; i < n; ++i) p = p + d[i];
}

int main() {
    const int N = 40;
    int total = 0;
    int *data = new int[N];
    for(int i = 0; i < N; ++i) data[i] = i;
    sum(data, N, total);
    cout << "total is " << total << endl;
    return 0;
}
```


Command line processing

- Open Atom editor
- Open a new file
 - Filename: **arg.cpp**
- Add the source code.
- Save the file.
- Compile and Execute

```
$ g++ arg.cpp -o arg
```

```
$ ./arg command line args
```

```
// Name of program argcargv.cpp
#include <iostream>
using namespace std;

int main(int argc, char** argv) {
    cout << "You entered: "
         << argc << " arguments:" << endl;

    for (int i = 0; i < argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```