HCSLAB

C++ Basics

Lecture 10-2

Optimism is the faith that leads to achievement.

Overview

- C++ Introduction
- Print/Console Input
- Variables/Const Keyword/Data Types
- Operators
- Namespaces
- Arrays
- Conditions/Loops
- Function
- File I/O
- Pointers/References

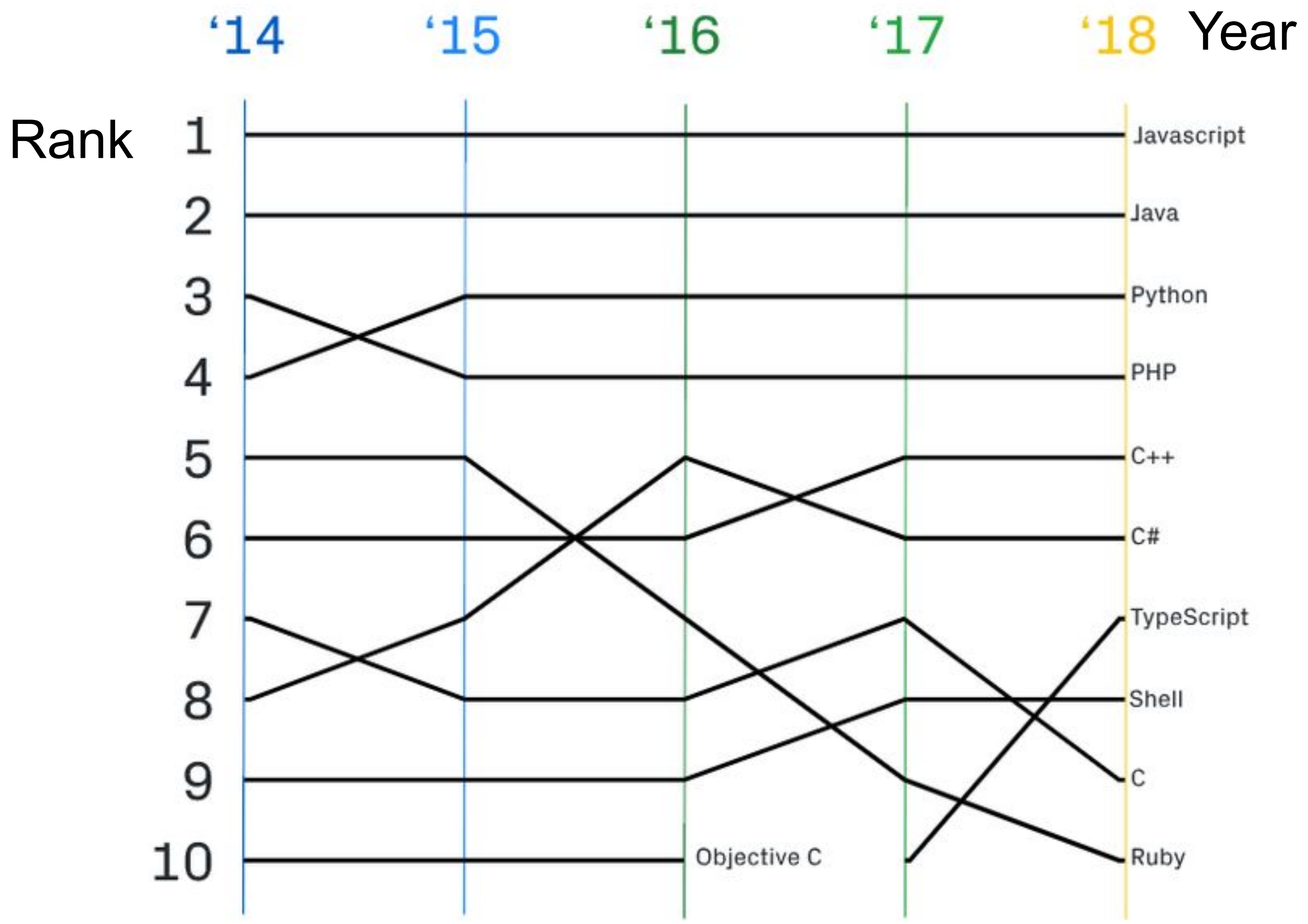
What is C++?

- C++ is a general-purpose programming language created as a an extension of the C programming language.
- C++ has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation.
    - C++ is also considered as “C with Classes”.
- C++ was designed with a focus on system programming for embedded software and large systems, which need to achieve high performance.

Why Use C++?

- You can do everything with C++: from low-level programming (memory manipulation, etc.) to high-level programming (generic data structure).
  - Programmer can choose their own programming style.
- Compatibility with C programming language.
- Generally, C++ is much faster than Java when there are lots of memory allocations.

GitHub's Programming Language Ranks



Revisiting Course Connections

- Java and C++ are used in many other courses.
- JAVA
  - Data Structure
  - Algorithm
  - Introduction to Data Mining
  - Mobile Computing and Its Applications
- C/C++
  - Systems Programming
  - Operating Systems
  - Computer Graphics
  - Compiler

Hello World with C++

main.cpp

```
#include <iostream>


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

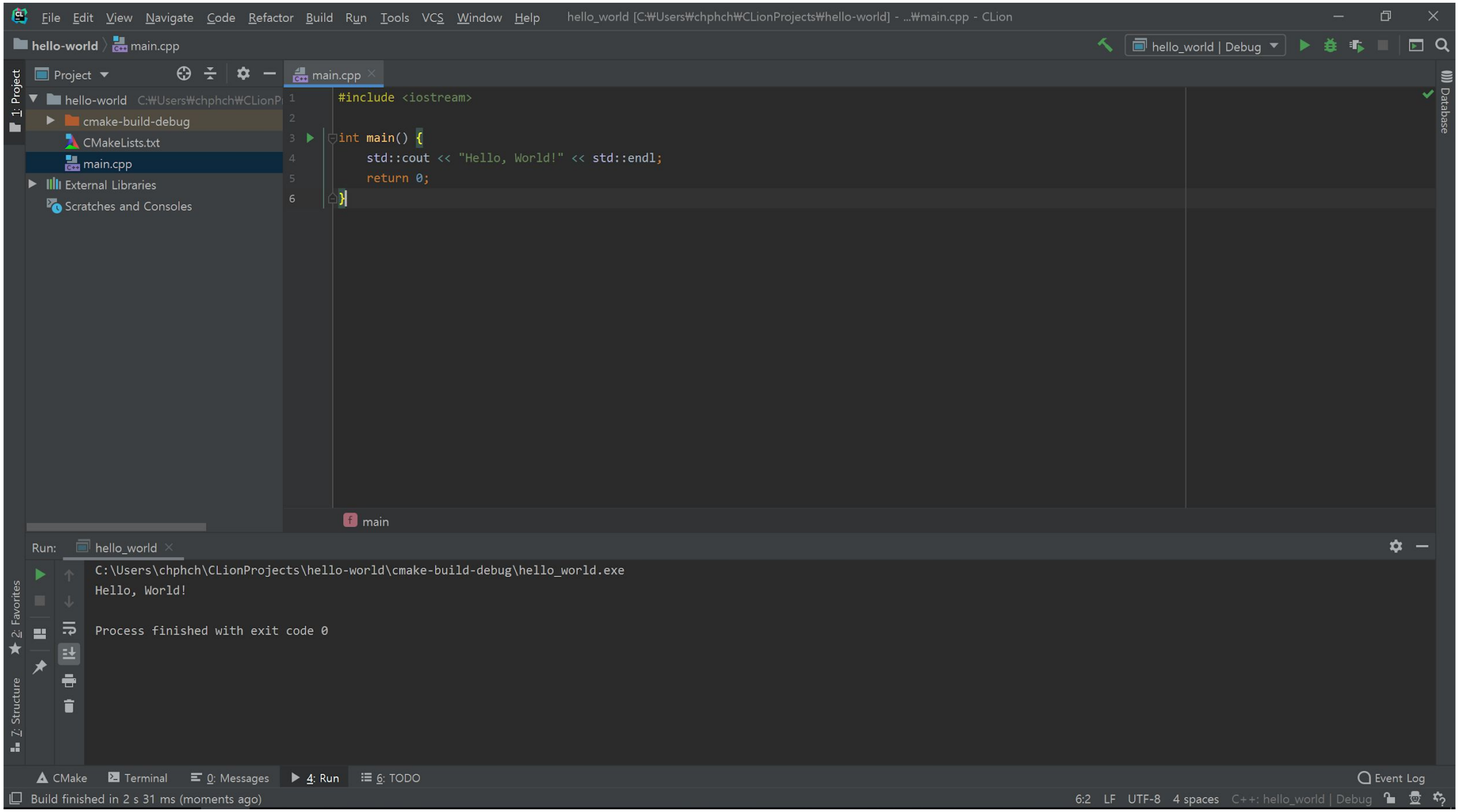
HCSLAB

How to Compile & Run?

1. Write a C++ source code. Use .cpp extensions.
2. Use “g++” to compile. Specify the source code file path, and the output binary file path with -o option.
3. Type the path of the output file to run.

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

$ ./main

Hello World!
```

Using CLion

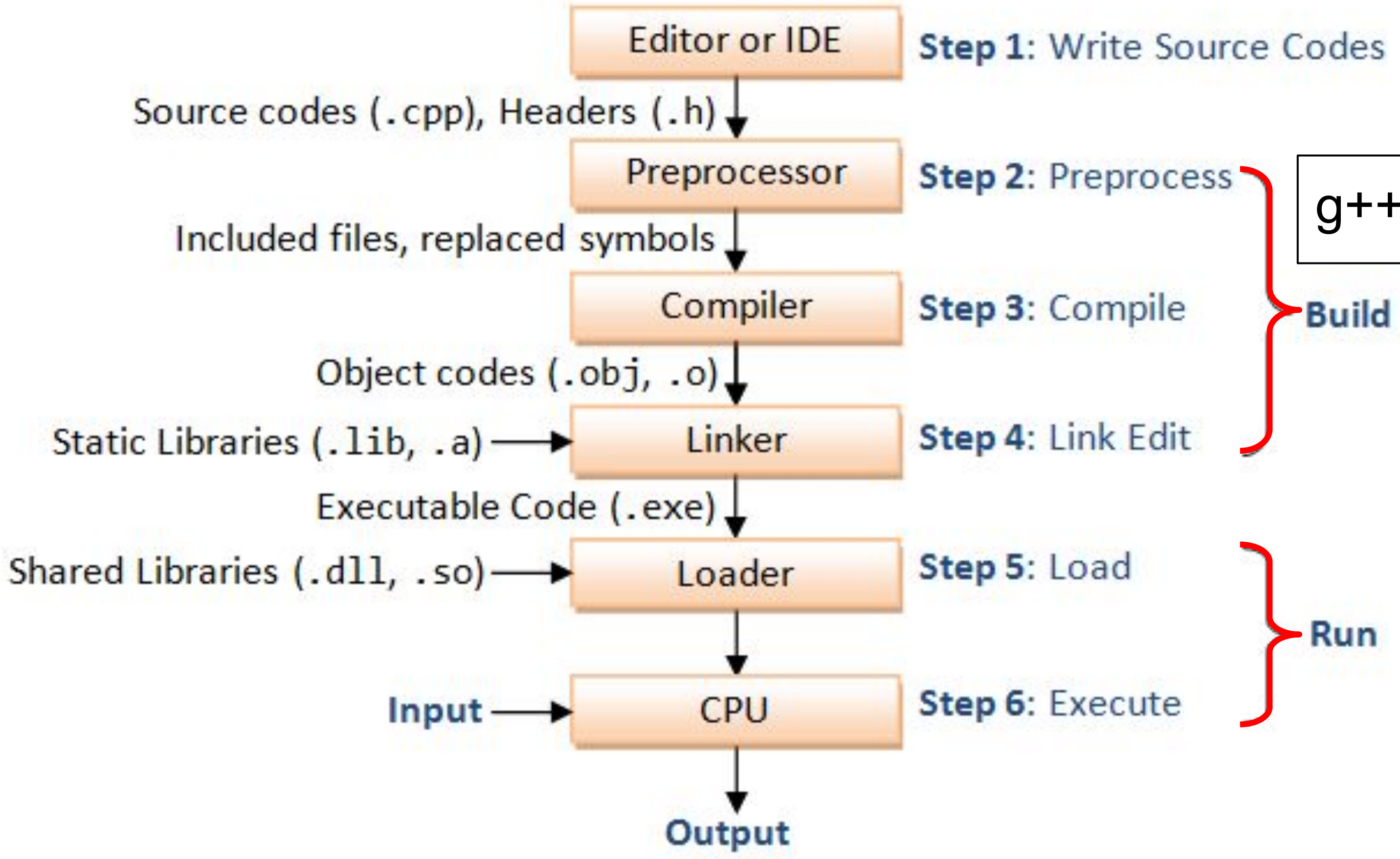

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
hello_world [C:₩Users₩chphch₩CLionProjects₩hello-world] - ...₩main.cpp - CLion
hello-world
main.cpp
hello_world | Debug
Project
hello-world C:₩Users₩chphch₩CLionP
cmake-build-debug
CMakeLists.txt
main.cpp
External Libraries
Scratches and Consoles
main
Run:
hello_world
CMake
Terminal
0: Messages
4: Run
6: TODO
Event Log
6:2 LF UTF-8 4 spaces C++: hello_world | Debug

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

```
C:\Users\chphch\CLionProjects\hello-world\cmake-build-debug\hello_world.exe
Hello, World!

Process finished with exit code 0
```

Build finished in 2 s 31 ms (moments ago)

Deeper Look



```
g++ -o main main.cpp
```

```
$ ./main
```

What Does it Mean?

main.cpp

```
#include <iostream>

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

Import I/O library “iostream” to use “std::cout” and “std::endl”.

The main function is a starting point of a c++ program. Exceptionally, you don’t have to return a value from the main function.

“std::cout” followed by one or more “<< VARIABLE” prints the variables to the standard output (console).

String expression with “ ”

“std::endl” breaks a line. You can use ‘\n’ instead.

Print (`std::cout`):

- `std::cout` followed by one or more `<<` variables prints the variables to the standard output (console).
- Use `std::endl` or `'\n'` to break a line.

```
#include <iostream>

int main() {
  int i = 3;
  float f = 4.5;
  char c = 'd';
  std::cout << i << ' ' << f << ' ' << c << std::endl;
}
```

Output

```
3 4.5 d
```

Print:String Formatting

- Use sprintf to write a data to a format string.

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

int main() {
    char str[50];
    sprintf(str, "Hi, I am %d years old", 3);
    cout << str << endl;
}
```

Output

```
Hi, I am 3 year old.
```

User Input (`std::cin`)

- Use `std::cin` and extraction operator `>>` to get input and save it to the given variable.
- The extraction operator can be used multiple times to get multiple inputs. `std::cin` assume that the multiple inputs are seperated with a space or in different lines.
- Import `<iostream>` to use `std::cin`.

HCSLAB

User Input: More Examples

```
#include <iostream>

int main() {
  int var, ivar;
  char cvar;
  std::cout << "Put an integer" << std::endl;
  std::cin >> var;
  std::cout << "The first input is " << var << std::endl;
  std::cout << "Put an integer and a character"
            << std::endl;
  std::cin >> ivar >> cvar;
  std::cout << "The second input is " << ivar
            << ", " << cvar << std::endl;
}
```

Console

```
output  Put an integer
 input  4
output  The first input is 4
output  Put an integer and a character
 input  2 d
output  The second input is 2, d
```

Comments

- Use `//`, `/* */` for comments in C++ like Java.
    - `//` for single-line comments.
    - `/* */` for multi-line comments.

```
#include <iostream>

int main(void) {
  // This is a comment
  std::cout << "Hello World!" << std::endl; // This is a
comment
  /* The code above will print the words Hello World!
to the screen, and it is amazing */
}
```

Variables

- Containers for storing data values.
- All C++ variables must be identified with unique names like Java variables.
- A C++ variable should be a specified data type.
- There are different types of C++ variables like Java variables. (following slide…)

`const` Keywords

- `const` keyword specifies that the object or variable is not modifiable.

```cpp
#include <iostream>
int main() {
  const int i = 5;
  i = 10;
}
```

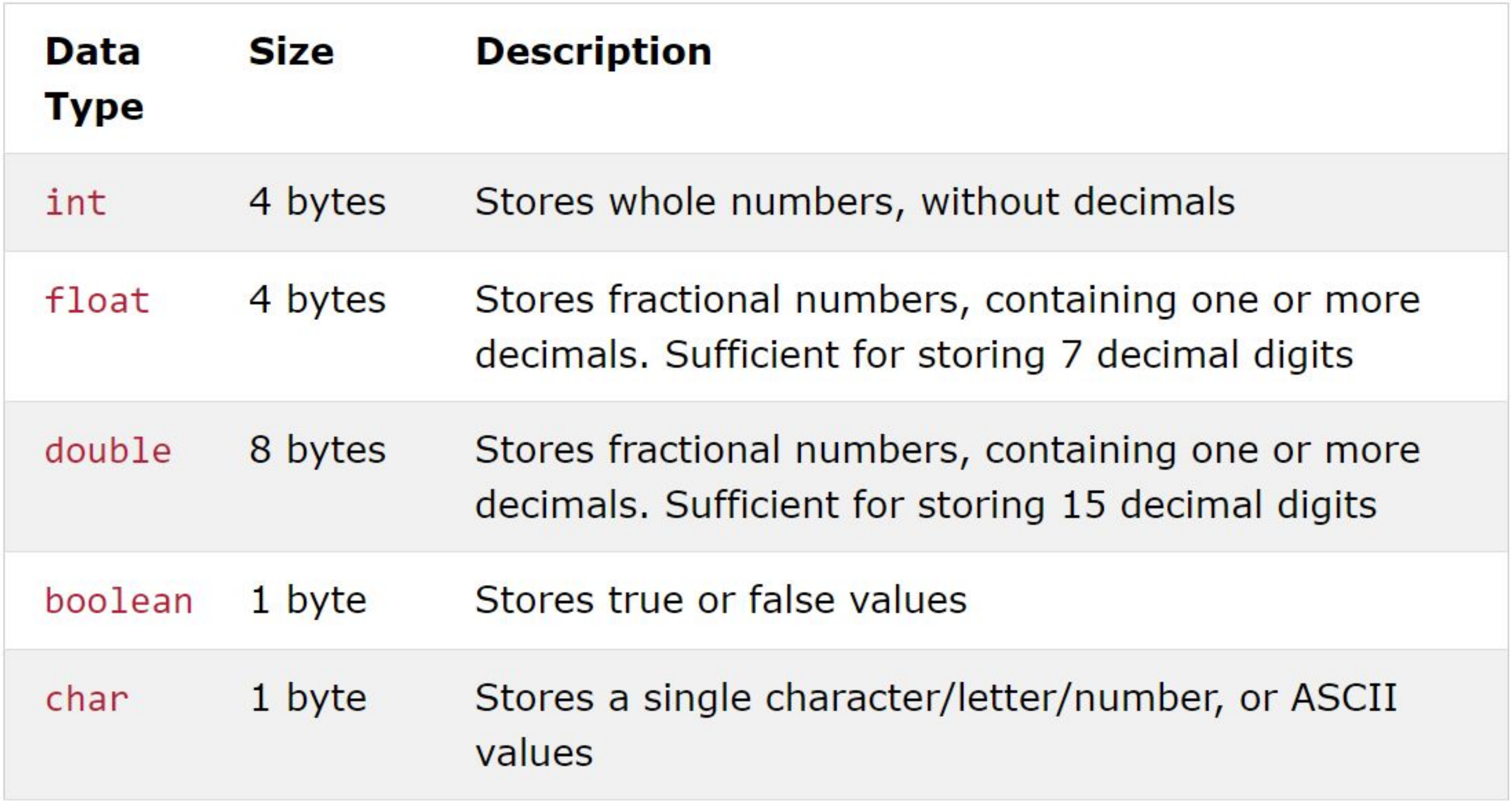
Output

```
main.cpp:6:7: error: cannot assign to variable 'i' with
const-qualified type
      'const int'
    i = 10;
    ~ ^
1 error generated.
```

Data Types

- C++ provides multiple data types as in JAVA.
- `int`, `float`, `double`, and `char` types of C++ are similar to those of Java.
- `bool` of C++ stores `true` or `false.`
    - When a `bool` variable is printed, 1 or 0 is printed (for `true` or `false` respectively).
    - A `bool` can be cast to `int` 1 or 0.
- `string` of C++ is similar to that of Java.
    - You should import `<string>` library as it is not a built-in data type.

Data Types



Data Type	Size	Description
`int`	4 bytes	Stores whole numbers, without decimals
`float`	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
`double`	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
`boolean`	1 byte	Stores true or false values
`char`	1 byte	Stores a single character/letter/number, or ASCII values

Data Types

```
#include <iostream>
#include <string>

int main() {
  int myNum = 5;                // Integer (whole number)
  float myFloatNum = 5.99;      // Floating point number
  double myDoubleNum = 9.98;    // Floating point number
  char myLetter = 'D';          // Character
  bool myBoolean = true;        // Boolean
  string myText = "Hello";      // String
  std::cout << myNum << ' ' << myFloatNum << ' '
       << myDoubleNum << ' ' << myLetter << ' '
       << myBoolean << ' ' << myText << std::endl;
}
```

Output

```
5 5.99 9.98 D 1 Hello
```

String Compare

- Check string equality with `==` operator. (Different from Java string comparison)

```
#include <iostream>

int main() {
    string str1 = "abcde",
           str2 = "abcde";
    bool is_equal = (str1 == str2);
    std::cout << is_equal << std::endl;
}
```

Output

```
1
```

# Meaning true

Type Conversion

- Type conversion are automatically performed when a value is copied to a compatible type.
- Both widening casting and narrowing casting are done automatically.
  - *Different from the type casting of JAVA
- Converting to the smaller data type may lose some data.

HCSLAB

Converting to the Smaller Data Type

```cpp
#include <iostream>

int main() {
    double d = .12345678901234567;
    float f = d;
    std::cout.precision(17);
    std::cout << d << std::endl;
    std::cout << f << std::endl;
}
```

Output

```
0.1234567890123456
0.1234567910432815
```

Trash value: float variable can store at most 7 digits.

Explicit Type Conversion

- You can also force conversions with () as Java.
- There are various types of explicit type conversions in C++ such as `dynamic_cast`, `static_cast` , `reinterpret_cast`, and `const_cast`. We may cover some of them later.

Explicit Type Conversion

```
#include <iostream>

int main() {
    int i = 100;
    float f1 = i / 3,
          f2 = (float) i / 3;
    std::cout << f1 << std::endl;
    std::cout << f2 << std::endl;
}
```

Output

```
33
33.3333
```

Type Checking

- Check type with typeid() like typeof in Java.

```
#include <iostream>
using namespace std;
class MyClass { };
int main() {
    int value = 3;
    MyClass myObject;
    cout << typeid(value).name() << endl;
    cout << typeid(myObject).name() << endl;
}
```

Output

```
i
7MyClass
```

Operators

- There are various operators in C++.
- There is also operator precedence in C++.

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
3	++a, -a, ...	Prefix Increment/decrement, plus/minus, ...	Right-to-Left
5	a*b, a/b, ...	Multiplication, division, ...	Left-ro-right
6	a+b, a-b	Addition and subtraction	Left-ro-right
9	<, <=, ...	Relation operators	Left-ro-right
10	==, !=	Relation operators	Left-ro-right

Other operators and full operator precedences: https://en.cppreference.com/w/cpp/language/operator_precedence

Namespaces

- As packages help manage the scope of various classes in JAVA, namespaces provide a way to define a scope for identifiers (variables, functions, etc.) in C++.
- Namespaces are used to systematize code in logical groups which prevents naming conflict, which can occur especially if there are multiple libraries with single names in your code base.

HCSLAB

Namespace Declaration

- Declare a namespace with `namespace` keyword.
- Access identifiers in a namespace using `::` .

```
#include <iostream>

namespace my_namespace {
    int x = 123;
}

int main() {
  std::cout << my_namespace::x << std::endl;
}
```

Output

```
123
```

`using namespace` Directive

- Avoid prepending of namespaces with `using namespace` directive. The directive tells the compiler that the subsequent code is making use of names in the specified namespace.
- A `using namespace` directive can be applied to multiple libraries. For example, `using namespace std;` enables the use of all std namespaces in multiple c++ files.

using namespace Directive

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

int main() {
  int i;
  clock_t start, end; // std::clock_t of <ctime>
  start = clock(); // std::clock of <ctime>
  for (i = 0; i < 100000; i++) { }
  end = clock();
  cout << i << endl; // std::cout, std::endl of <iostream>
  cout << end - start << endl;
}
```

Output

```
100000
169
```

Global Variables and Functions

- In Java, all variables and functions are members of classes.
- However, in C++, there are global variables and global functions which are not a member of any class.
- A global variable can be accessed by any function. That is, a global variable is available for use throughout the entire program.
- The main function is a typical global function.

Global Variables and Functions

```
#include <iostream>

int glob = 123; // Global variable declaration

int func(int i) { // Global function declaration
    return glob + i;
}

int main () {
   int local = 111; // Local variable declaration
   cout << func(local) << endl;
}
```

Output

```
234
```

Arrays Declaration

- Like Java, arrays are used to store multiple values in a single variable.
- Declare an array with the variable type, the name of the array followed by square brackets and specify the number of elements it should store. (It's different from Java array declaration).
- Use can also declare an array with new keyword like Java array, but we will handle this later.

```
#include <string>
int iarr[5];
string sarr[5];
```

Array Initialization

- Use array literal to declare an array with initialization.
- Place the values in a comma-separated list, inside curly braces.
- The size of the array can be omitted.

```
#include <string>

int nums1[3] = {10, 20, 30},
    nums2[] = {10, 20, 30};
string cars1[4] = {"Volvo", "BMW", "Ford", "Mazda"},
    cars2[] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Access/Change an Array Element

- Access/change an array element by referring to the index number.

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

string cars[] = {"Volvo", "BMW", "Ford", "Mazda"};
cout << cars[0] << endl;
// This statement changes the value of the first
element in cars
cars[0] = "Opel";
cout << cars[0] << endl;
```

Output

```
Volvo
Opel
```

Loop Through an Array

- Loop through array elements with a loop.

```
#include <string>
using namespace std;
int main() {
  string cars[] = {"Volvo", "BMW", "Ford", "Mazda"};
  // Outputs all elements in the cars array:
  for (string car : cars) { cout << car << endl; }
}
```

Output

```
Volvo
BMW
Ford
Mazda
```

Array Length

- Unlike Java, there is no magic keyword length to get the size of an array.
- There are several ways to get the size of an array, but the most safe way is to use “size” function in “iterator” library. (From C++17)

```
#include <iostream>
#include <iterator>
using namespace std;
int main() {
  string cars[] = {"Volvo", "BMW", "Ford", "Mazda"};
  cout << size(cars) << endl;
}
```

Conditions (Boolean Expression)

- Less than: a < b
- Less than or equals to: a <= b
- Greater than: a > b
- Equal to: a == b
- Not equal to: a != b

if else

- Use if, else, and else if to specify a block of code to execute depending on a condition.

```
int time = 22;
if (time < 10) {
  cout << "Good morning." << endl;
} else if (time < 20) {
  cout << "Good day." << endl;
} else {
  cout << "Good evening." << endl;
}
```

switch

- Use the switch statement to select one of many code blocks to be executed. (The syntax same with switch of Java)

```
int day = 4;
switch (day) {
  case 6:
    cout << "Today is Saturday" << endl;
    break;
  case 7:
    cout << "Today is Sunday" << endl;
    break;
  default:
    cout << "Looking forward to the Weekend" << endl;
}
```

Ternary Operator

- Shorthand if-else like Java ternary operator.

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

int main() {
  int a = 10, b = 20;
  cout << a > b ? "a" : "b" << " is greater" << endl;
}
```

Output

```
b is greater
```

Loop

- Like Java, there are four types of loops, while, do-while, for, and for-each.
- C++ loop syntax is same with the syntax of Java.

while & do-while Loop

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

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

for & for-each Loop

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

int arr[] = {1, 2, 3, 4, 5};

for (int i = 0; i < size(arr); i++) {
  cout << i << endl;
}

for (int i : arr) {
  cout << i << endl;
}
```

Breaking Loop

- Use break to break the whole loop statement.
- Use continue to break the current iteration, and go to the next iteration.

```cpp
#include <iostream>
using namespace std;
int main(void) {
  for (int i = 0; i < 10; i++) {
    cout << i;
    if (i == 1) { continue; }
    cout << ',';
    if (i == 6) { break; }
  }
  cout << endl;
}
```

Output

```
0,12,3,4,5,6,
```

One-line if/else/while/for/for-each

- You can omit curly brackets {} of a `if`/`else` statement or `while`/`for`/`for`-each loop containing only a single statement, just like Java.
- Not recommended, because this may increase your chances of writing buggy code. However, some codes are already in this style, so you have to be aware of this syntax.

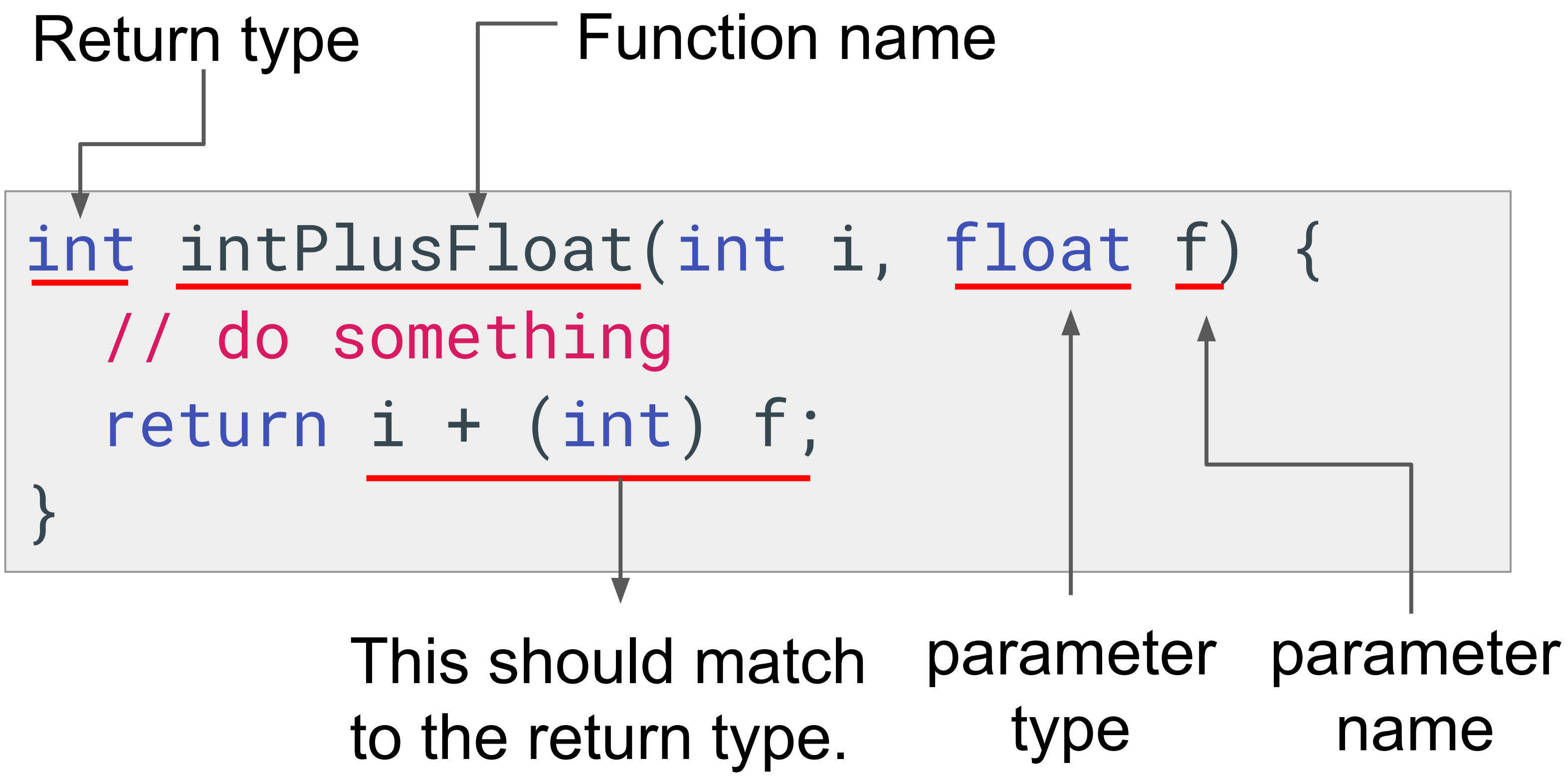
```
int hour = 14;
if (hour < 12)
  cout << "It's before noon." << endl;
// else (accidentally commented)
  cout << "It's after noon." << endl;
```

Output

```
It’s before noon.
It’s after noon.
```

Function

- A function is a block of code which only runs when it is called.



```
int intPlusFloat(int i, float f) {
  // do something
  return i + (int) f;
}
```

Function Overloading

- Multiple functions can have the same name with different parameters and return types.

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

int plus(int x, int y) { return x + y; }
double plus(double x, double y) { return x + y; }

int main() {
  cout << plus(1, 2) << endl;
  cout << plut(1.2, 3.4) << endl;
}
```

Output

```
3
4.6
```

Header (.h) File

- Lots of libraries separates declarations and implementations of elements (variables, functions, and classes) as header files and body (.cpp) files.
- Only importing a header file enables to use all implementations of the declarations in the header file.
- Header file also prevent multiple inclusion.

Header (.h) File

- In C++, to call a function, the function should be written in the previous lines, which is different from Java.
- However, the written positions of function bodies isn’t important when all declarations of the functions are written in a header file.

Macro

- A macro is a fragment of code which has been given a name. Whenever the name is used, it is replaced by the contents of the macro.
- There are two kinds of macros, object-like macros and function-like macros. Use `#define` directive to give symbolic names to numeric constants.

Object-Like Macro Example

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

#define PI 3.14

int main() {
    double radius = 10;
    double circumference = 2 * PI * radius;
    cout << circumference << endl;
}
```

Output

```
62.8
```

Function-like Macro Example

```
#define SUB(x,y) x-y
#define PRINT(x) cout << x << endl;

int main() {
    int k = 10;
    int m = 5;
    int diff = SUB(k,m);
    PRINT(diff);
}
```

Output

```
5
```

inline Function

- Whole code of the inline function gets inserted or substituted at the point of inline function call.
- This substitution is performed by the C++ compiler at compile time.

```
inline int min(int x, int y) {
   return x > y ? y : x;
}
int main() {
   cout << min (10, 5) << endl;
   // min (10, 5) is not really a function call, but a
text replacement
}
```

typedef

- typedef gives an alias (nickname) to an existing type.
- Note that it does not create a new type.

```
typedef long miles_t;
typedef long speed_t;
typedef int int16_t;

typedef std::vector<std::pair<std::string, int>>
pairlist_t;

miles_t distance = 5 ;
speed_t mhz = 3200 ;
```

Files

- fstream library provide features to work with files.
- To use fstream library, include both iostream and fstream.
- Three objects included in the fstream library, which are used to create, write, or read files.
    - ofstream: Creates and writes to files.
    - ifstream: Reads from files.
    - fstream: A combination of ofstream and ifstream: creates, reads, and writes to files.

Create and Write to a File

- To create a file, use either ofstream or fstream object, and specify the name of the file.
- Use insertion operator `<<` to write to the file.
- Close the stream object when the writing is done.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
  // Create and open a text file
  ofstream my_file("filename.txt");
  // Write to the file
  my_file << "Files are fun!" << endl;
  my_file.close(); // Close the file
}
```

Output filename.txt

```
Files are fun!
```

Read a File

- To read from a file, use either the ifstream or fstream object, and the name of the file.
- Use getline function defined in fstream library with a ifstream/fstream object, and a string variable.
- You can get the whole lines of a file by using getline and while statement together. In while statement, the return value of getline is casted to true or false by whether there are any more lines left to read.
- Close the stream object when the reading is done.

Read a File

```
#include <iostream>
#include <fstream>

using namespace std;
int main() {
  string str;
  ifstream my_file("filename.txt");
  while (getline(my_file, str)) {
    cout << str << endl;
  }
  my_file.close();
}
```

filename.txt

```
1st line
2nd line
3rd line
```

Output

```
1st line
2nd line
3rd line
```

Motivation: Pointers and References

- It is not straightforward to swap two variables with a function when only values are passed to the the function.
- We can swap two variables after we learn pointers.

```
#include <iostream>
using namespace std;
void swap(int var1, int var2) {
  int temp = var1; var1 = var2; var2 = temp;
}
int main() {
  int var1 = 1, var2 = 2;
  swap(var1, var2);
  cout << var1 << ',' << var2 << endl;
}
```

Output

```
1,2 # Not swapped!!!
```

Address-of Operator &

- Get memory address with address-of operator &.

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

int main() {
  int var = 3;
  cout << var << endl;
  cout << &var << endl;
}
```

Output

```
3
0x7ffeeeee1d7fc   # This can be different
at each run
```

Pointers

- A pointer variable can store the memory address.
- A pointer data type is created with `*` on the existing data type. In the example below, ptr stores the address of an integer variable, var.

```cpp
#include <iostream>
using namespace std;
int main() {
  int var = 3;
  int* ptr = &var;
  cout << var << endl;
  cout << &var << endl;
  cout << ptr << endl;
}
```

Output

```
3
0x7ffeeaeb67fc
0x7ffeeaeb67fc
```

Pointer Expressions

- There are three types of pointer expressions.

```
string* mystring;
string *mystring;
string * mystring;
```

The first expression is recommended.

Misuse of Pointer Syntax

```
char *str1, str2;

char* str1, str2;
```

- Two lines are equivalent.
- Be careful! `str1` is a `char` pointer type variable, but `str2` is a `char` type variable.

Access Pointer Value

- You can access a value of a pointer by using the dereference operator (`*`).

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string food = "Pizza";
  string* ptr = &food;
  // Output the value of food
  cout << food << "\n";
  // Access the memory address
of food and output its value
  cout << *ptr << "\n";
}
```

Output

```
Pizza
Pizza
```

Change Pointer Value

- You can change the value of a variable that the pointer points to by using the dereference operator.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
  string food = "Pizza";
  string* ptr = &food;
  // Change the value of the pointer
  *ptr = "Hamburger";
  // Output the new value of the pointer
  cout << *ptr << endl;
  // Output the new value of the food variable
  cout << food << endl;
}
```

Output

```
Hamburger
Hamburger
```

Swap Values with Pointers

- We can swap values of two variables with pointers.

```cpp
#include <iostream>
using namespace std;
void swap(int* ptr1, int* ptr2) {
  int temp = *ptr1;
  *ptr = *ptr2;
  *ptr2 = temp;
}
int main() {
  int var1 = 1, var2 = 2;
  cout << var1 << ',' << var2 << endl;
  swap(&var1, &var2);
  cout << var1 << ',' << var2 << endl;
}
```

Output

```
1,2
2,1
```

new & delete keyword

- “new” keyword allocate a memory in the heap space, and return the pointer of the memory.
- delete keyword delete the allocated memory which the pointer points.

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

int main() {
    int *ptr = new int;
    cout << ptr << endl;
    delete ptr;
}
```

Output

```
0x7fbc6d4006a0
```

References

- A reference variable is a reference to an existing variable, and it is created with the & operator.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string my_home = "My Home";
  string &my_house = my_home
  cout << my_home << endl;
  cout << my_house << endl;
}
```

Output

```
My Home
My Home
```

Swap with References

- Easier swap with references.

```
#include <iostream>
using namespace std;
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}
int main() {
  int var1 = 1, var2 = 2;
  cout << var1 << ',' << var2 << endl;
  swap(var1, var2);
  cout << var1 << ',' << var2 << endl;
}
```

Output

```
1,2
2,1
```