**C++ Notes**

**Last Modified: 2023 September 15**

**Chapter 1: Theory**

**This section includes some theory questions, such as algorithms and understanding of the basic theory. But, some practical understandings (i.e., what is the use of comments?) are discussed in Chapter 3.**

* What is C++?
  + C++ is a cross-platform language that can be used to create high-performance applications.
  + C++ gives programmers a high level of control over system resources and memory.
  + The language was updated 4 major times in 2011, 2014, 2017, and 2020 to C++11, C++14, C++17, C++20.
* Why use C++?
  + C++ is an **object-oriented programming language** which gives a clear structure to programs and allows code to be reused, lowering development costs.
* Difference between C and C++
  + C++ was developed as an extension of C, and both languages have almost the same syntax.
  + The main difference between C and C++ is that C++ support classes and objects, while C does not.
* Why use visual code as an editor or an IDE (Integrated Development Environment)?
  + It is easy to install, and it is cross-platform. It runs well on Windows, Mac, and Linux.
* What is a compiler?
  + A compiler is a software that compiles our codes into code that can directly run on the hardware or binary executable format.
  + Testing our code on multiple compilers can make our code more portable.
  + Example: Mingw (GCC), Msvc, Clang llvm
* What is compile error and runtime error?
  + **Compile Error:** Occurs when you violate the rules of writing syntax, such as missing a semicolon, a parenthesis or using a reserved word. The compiler detects these errors and prevents the code from running.
  + **Runtime Error:** Occurs during the execution of the code, such as dividing by zero, null pointer or logic error. These errors are usually not detected by the compiler but cause the program to produce wrong results or terminate abnormally.
* How is a cpp file executed?
  + When we compile our program, the compiler converts our code into binary format, which is unreadable by humans but readable by machines. It is converted into binary format because it is easily understandable by the CPU.

A computer screen with a computer code

Description automatically generated

* When we run our program, the binary file is loaded up in memory (RAM), which is the program area in the following graph. CPU starts running the program statements by statements. It starts at the top (in our example, allocating space to store our integer ***a***). If a variable value is not specified, some junk value will be stored instead (e.g., ***c***). The CPU keeps a return address for a function to know where it should return after executing the function.

A computer screen shot of a computer

Description automatically generated

* What is Statement?
  + A statement is a basic unit of computation in a C++ program. Every C++ program is a collection of statements organized in a certain way to achieve some goal.
  + Statements end with a semicolon in C++ (***;***).
  + Statements are executed in order from top to bottom when the program is run.
  + Execution keeps going until there is a statement causing the program to terminate or run another sequence of statements.
* What is Variable?
  + A named piece of memory that you use to store specific types of data.
  + All C++ variables must be identified with unique names.These unique names are called **identifiers**.
  + Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
* What are Core Features?
  + Core features are a basic building block of the C++ programming language. For example, How to define variables, How to use a function in C++, etc.
* What is Standard Library?
  + A set of highly specialized components that we can easily used in C++ programs.
  + E.g., ***#include <iostream>***, ***#include <string>***
* What is STL (Standard Template Library)?
  + A part of C++ standard library, but it is collection of container and algorithm types. It is used to implement common data structures and algorithms in C++ programs.
  + Advantage: It provides a way to write generic, reusable code that can be applied to different data types.
  + E.g., ***List***, ***Vector***, ***Map***, ***Sort***, ***Find***, ***Transform***
* What is an Escape Sequence?
  + Escape sequence forces the cursor to change its position to the beginning of the next line on the screen.

|  |  |
| --- | --- |
| **Escape Sequence** | **Description** |
| **\t** | Creates a horizontal tab (4 spaces). |
| **\n** | Inserts a new line. |
| **\\** | Inserts a backslash character. (\) |
| **\'** | Inserts a single quote character. (') |
| **\"** | Inserts a double quote character. (**")** |

* How do we represent data in memory?
  + IEEE 754 is a method used to convert all data into 0 and 1 to represent it in the memory.
* What is ASCII code?
  + ASCII was among the first encodings to represent text in a computer.
  + It falls short when it comes to representing languages other than English (e.g., Chinese)
  + There are better ways to represent text that is meant to be seen in different languages, one of the most common being **Unicode**.
* Why it is recommended to use ***size\_t*** instead of ***unsigned int*** in for loop?
  + Unsigned int **may not be able to hold the size of all objects in bytes**, especially on platforms where int is 32 bits and size\_t is 64 bits. This could cause overflow or truncation errors when using sizeof or indexing large arrays or containers.
* What is the difference between ***struct*** and ***array***?
  + A struct can contain many different data types (int, string, bool, etc.) but an array can only contain same data types.
* Why is it useful to know the **memory address**?
  + Manipulating the data in the computer's memory (by pointers and reference) can **reduce the code and improve the performance**.

**Chapter 2: Configuration Setting**

* How to download C++ compilers
  + Download Mingw and Clang llvm in one place: <https://winlibs.com/>
* How to test whether the compiler GCC has been installed and added to the path (Windows)
  + *g++ --version* & *clang++ --version* in cmd

A screenshot of a computer program

Description automatically generated

**2.1 Visual Studio**

* How to check whether Visual Studio has been appropriately downloaded in Windows
  + Open Developer Command Prompt for VS and type cl.exe

A screenshot of a computer program

Description automatically generated

**2.2 Visual Studio Code**

* How to **run a cpp file** in Visual Studio Code?

1. Install C/C++ Extension.
2. Terminal -> Configure Tasks -> Choose the preferred compiler.

A screenshot of a computer

Description automatically generated

1. A JSON file will be created, which is our configuration to configure which compilers VS code uses to compiler our *main.cpp* file.
2. To run the cpp file in a particular C++ version: google GCC C++ <version>, e.g., GCC C++ 20
3. From <https://stackoverflow.com/questions/66975491/how-to-use-c-20-in-g>, we know we should add *-std=c++20* in JSON file.

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

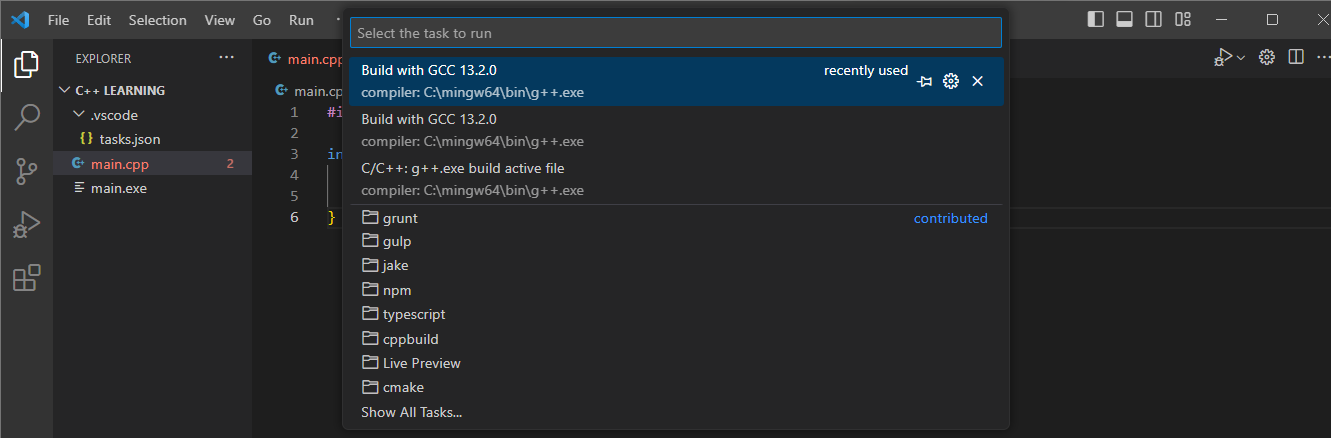
We can also check the tutorial from the C++ extension.

1. Open a new terminal -> Run task -> *main.exe* is created.

A screenshot of a computer

Description automatically generated

1. Select the compiler. (We can change the display name by modifying labels in the JSON file.)



1. If the IDE has been connected to our gcc compiler, we can type *.\main.exe* to run the program.

A black background with white text

Description automatically generated

* It is always good to have stored args so that we can directly copy and paste our configuration into each project's json. *(${workspaceFolder}\\\*.cpp* builds multiple files in the same project.)

"args": [

                "-fdiagnostics-color=always",

                "-g",

                "-std=c++20",

                "${workspaceFolder}\\\*.cpp",

                "-o",

                "${fileDirname}\\${fileBasenameNoExtension}.exe"

         ]

* How can we configure our project to compile it in another compiler?
  1. Terminal -> Configure Task -> Select the new compiler

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. The JSON file will have a new configuration for the new compiler.
  2. The args section can be the same as GCC.
* How to use the compiler from Microsoft (Msvc)
  1. Open Developer Command Prompt for VS
  2. *cd <project dir>*
  3. Open the project in VS code by command *code .*

A screenshot of a computer program

Description automatically generated

* Why do we open our project in this way?
  + Because the terminal can only identify Msvc in this situation as the compiler has not been added into PATH.

A screenshot of a computer

Description automatically generated

1. Terminal -> Configure Task -> C/C++: cl.exe build active file

A screenshot of a computer

Description automatically generated

1. Default args for Msvc json file:

"args": [

                "/Zi",

                "/std:c++latest",

                "/EHsc",

                "/Fe:",

                "${workspaceFolder}\\\*.cpp",

                "${fileDirname}\\${fileBasenameNoExtension}.exe"

         ]

* How to configure C/C++ extension in VS code?
* View -> Command Palette -> C/C++ Edit Configurations (UI)

A screenshot of a computer

Description automatically generated

**Chapter 3: Data Type**

**3.1 Basic Data Types**

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| boolean | 1 byte | Stores true or false values |
| char | 1 byte | Stores a single character/letter/number, or ASCII values |
| int | 2 or 4 bytes | Stores whole numbers, without decimals |
| float | 4 bytes | Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits |
| double | 8 bytes | Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits |

**3.2 Constant**

* Make the variable **unchangeable and read-only.**
* E.g. **const** int myNum = 15;

**3.3 Auto**

* To let the compiler deduce the type.
* There are several suffixes to modify int and fractional number. (e.g., *f*, ***l***, ***u***, ***ul***, ***ll***)
* More examples can be found in Chapter 5.11.

**3.4 String**

* A string variable contains a collection of characters surrounded by double quotes.
* Two ways to concatenate different strings.
  + Concatenation
    - The ***+*** operator can be used between strings to add them together to make a new string. For example,

string firstName = "John ";  
string lastName = "Doe";  
string fullName = firstName + " " + lastName;

* + Append
    - A string in C++ is actually an object, which contain functions that can perform certain operations on strings. For example, you can also concatenate strings with the ***append()*** function. For example,

string firstName = "John ";  
string lastName = "Doe";  
string fullName = firstName.append(lastName);

* Length
  + To get the length of a string, use the ***length()*** or ***size()*** function, the following example string’s length is 26:

string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
cout << "The length of the txt string is: " << txt.length();

cout << "The length of the txt string is: " << txt.size();

* Access/Slice
* The characters in a string by referring to its index number inside square brackets ***[]***. E.g.,

string myString = "Hello";  
cout << myString[0]; // prints the first character in myString

myString[0] = 'J'; // "Hello" -> "Jello"

* GetLine
  + To read a line of text. It takes cin as the first parameter, and the string variable as second.
  + Example:

string fullName;  
cout << "Type your full name: ";  
getline (cin, fullName);  
cout << "Your name is: " << fullName;  
  
// Type your full name: John Doe  
// Your name is: John Doe

* + More example checks Chapter 5.4 Inputs and Outputs

**3.5 Pointers**

* Pointer is a special kind of variable that stores addresses of other variables. (e.g., ***int \*p\_var {&var};***)

A screen shot of a computer

Description automatically generated

* A pointer can only store an address of its same type variable.
* Pointer Assignment Syntax:

string food = "Pizza";  // A food variable of type string  
**string\* ptr = &food;**    // A pointer variable, with the name ptr, that stores the address of food

* Pointer Deferencing Syntax:

// Dereference: Output the value of food with the pointer (Pizza)  
**cout << \*ptr << "\n";**

* A pointer can be initialized by ***nullptr***.
* All pointer variables have the **same size**.
* The position of ***\**** does not matter.

**Chapter 4: Operation**

**4.1 Basic Data Operations**

* add
* subtract
* divide
* modulus
* multiply

**4.2 Flow Control**

|  |  |
| --- | --- |
| **If** | Doing things conditionally |
| **Else If** | Testing for several different conditions. |
| **Switch** | Testing for several different conditions. |
| **Ternary Operator** | An alternative way to do test with the if statements. |

**4.3 Loops**

* All loop is used to perform repetitive tasks.

|  |  |
| --- | --- |
| **For loop** | When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop. |
| **Range based for loop** |  |
| **While loop** | The while loop loops through a block of code as long as a specified condition is true. |
| **Do while loop** | The do/while loop is a variant of the while loop. **This loop will execute the code block once, before checking if the condition is true**, then it will repeat the loop as long as the condition is true. |

**Chapter 5: Code Examples**

**5.1 First Program**

#include <iostream>

// using namespace std;

consteval int get\_value(){

    return 3;

}

int main(){

    constexpr int value = get\_value();

    std::cout << "value : " << value << std::endl;

    return 0;

}

***#include <iostream>***: To include a standard library header file, which provides input and output functionality, such as ***std::cout*** and ***std::endl***.

***int main()***: The entry point of a C++ program.

***std::cout***: Print something.

***std::endl***: Print a new line. This can also be replaced by ***cout << "\n"***.

***using namespace std***: Can be used to omit ***std::***

***return 0***: Send message to the operation system if the program is finished successfully or if there is some kind of problem.

**5.2 Comments**

#include <iostream>

// Entry point main function

int main(int argc, char \*\*argv)

{

    //One line comment

    /\*

        Multi-line block comment

        Another line

        Oh! And another one!

    \*/

   //Print out some text

   std::cout << "Hello World in C++20!" << std::endl;

   return 0;

}

* What is the use of comments?
  + Comments can be used to explain C++ code, and to make it more readable.
  + It can also be used to prevent execution when testing alternative code.
  + The compiler will ignore comments.

**5.3 Functions**

#include <iostream>

// A function that adds two numbers and returns the result

int addNumbers(int first\_number, int second\_number){

    int sum = first\_number + second\_number;

    return sum;

}

int main(int argc, char \*\*argv){

    int firstNumber = 12;

    int secondNumber {9}; // Another way to initialize variables

    int sum = firstNumber + secondNumber;

    std::cout << "The sum of " << firstNumber << " and " << secondNumber <<" is " << sum << std::endl;

    std::cout << "The sum of " << firstNumber << " and " << secondNumber <<" is " << addNumbers(firstNumber, secondNumber) << std::endl;

    return 0;

}

***int addNumbers***: A function that adds two numbers and returns the result.

* A function must be defined before it is used.
* The advantage of function is reusable.

**5.4 Inputs and Outputs**

#include <iostream>

#include <string> // To use std::string

int main(int argc, char \*\*argv){

    // std::cout: Printing stuff to the console

    std::cout << "Hello World!" << std::endl;

    // Error

    std::cerr << "std::cerr output: Something went wrong." << std::endl;

    // Log message

    std::clog << "std::clog output: This is a log message." << std::endl;

    int age;

    std::string name;

    std::cout << "Please type in your Last Name: " << std::endl;

    // Input

    std::cin >> name;

    std::cout << "Please type in your age: " << std::endl;

    std::cin >> age;

    std::cout << "Hello " << name << "! You are " << age << " years old." << std::endl;

    // Chaining std::cin

    std::cout << "Please type in your Last Name and age, separated by spaces: " << std::endl;

    std::cin >> name >> age;

    std::cout << "Hello " << name << "! You are " << age << " years old." << std::endl;

    // Reading data with spaces

    std::string full\_name;

    std::cout << "Please type in your full name and age: " << std::endl;

    std::getline(std::cin, full\_name); // Enter then input age

    std::cin >> age;

    std::cout << "Hello " << full\_name << "! You are " << age << " years old." << std::endl;

    return 0;

}

***std::cout***: To print/output the variable to the console(terminal). The output is stored in a buffer until it is flushed. The buffer is typically flushed when a newline character (***\n***) is encountered or the program exits.

***std::cerr***: An unbuffered stream that is useful for printing error messages or other critical information that needs to be displayed immediately. The output is immediately sent to the console without being stored in a buffer.

***std::log***: Printing log messages to the console. It is buffered like ***cout***. This makes it useful for logging information that does not need to be displayed immediately, but still needs to be stored in a buffer for later retrieval.

***std::cin***: Reading data from the terminal. The example above shows three usages of ***cin***.

***#include <string>***: To include a standard library header file, which provides ***std::string***

**5.5 Number Systems**

#include <iostream>

int main(){

    int number1 = 15; // Decimal

    int number2 = 017; // Octal

    int number3 = 0x0F; // Hexadecimal

    int number4 = 0b00001111; // Binary

    std::cout << "number 1:" << number1 << std::endl;

    std::cout << "number 2:" << number2 << std::endl;

    std::cout << "number 3:" << number3 << std::endl;

    std::cout << "number 4:" << number4 << std::endl;

    return 0;

}

* The above program represents the same number in four different representations.
  + Decimal:
  + Octal:
  + Hexadecimal:
  + Binary:
* All data is represented by a bunch of grouped cells of 0’s and 1’s in memory.
* As the range of your data grows, so will the number of digits you need to represent the data in memory.
* Hexadecimal system makes it a little easier for humans to handle data streams with 1’s and 0’s.

N

* Octal has the same goal as Hexadecimal, but it’s almost no longer used in modern times. It is just mentioned here for awareness.

**5.6 Initialization**

#include <iostream>

int main(){

    // Braced Initialization

    int elephant\_count; // Variable may contain random garbage value. Warning

    int lion\_count{}; // Initialized to 0

    int dog\_count{10}; // Initializes to 10

    int cat\_count{15}; // Initializes to 15

    // Can use expression as initializer

    int domesticated\_animals{dog\_count + cat\_count};

    // int narrowing\_conversion{2.9}; // Warning: Some of the compiler will chop off the decimal part or throw an error

    std::cout << "Elephant count: " << elephant\_count << std::endl;

    std::cout << "Lions count: " << lion\_count << std::endl;

    std::cout << "Dogs count: " << dog\_count << std::endl;

    std::cout << "Cats count: " << cat\_count << std::endl;

    // Functional Initialization

    int apple\_count(5);

    int orange\_count(10);

    int fruit\_count{apple\_count + orange\_count};

    int narrowing\_conversion\_functional(2.9); // Information lost, 2 is stored instead of 2.9.

    std::cout << "Apples count: " << apple\_count << std::endl;

    std::cout << "Oranges count: " << orange\_count << std::endl;

    std::cout << "Fruits count: " << fruit\_count << std::endl;

    std::cout << "Narrowing conversion (Functional Initialization): " << narrowing\_conversion\_functional << std::endl;

    // Assignment Initialization

    int bike\_count = 2;

    int truck\_count = 7;

    int vehicle\_count = bike\_count + truck\_count;

    int narrowing\_conversion\_assignment = 2.9; // Information lost, 2 is stored instead of 2.9.

    std::cout << "Bikes count: " << bike\_count << std::endl;

    std::cout << "Trucks count: " << truck\_count << std::endl;

    std::cout << "Vehicles count: " << vehicle\_count << std::endl;

    std::cout << "Narrowing conversion (Assignment Initialization): " << narrowing\_conversion\_assignment << std::endl;

    // Check the size with sizeof

    std::cout << "sizeof int: " << sizeof(int) << std::endl; // 4 bytes

    std::cout << "sizeof truck\_count: " << sizeof(truck\_count) << std::endl; // 4 bytes

    return 0;

}

* There are three ways to initialize a variable.
  + Braced Initialization (***{}*** can only be used in initialization but not in assignment.)
  + Functional Initialization(***()*** can only be used in initialization but not in assignment.)
  + Assignment Initialization
* To declare more than one variable of the same type, use a comma-separated list:

***int x = 5, y = 6, z = 50;***

***sizeof***: Used to measure the memory size of a variable. For integer, it should be 4 bytes.

**5.7 Integer**

#include <iostream>

int main(){

    int value1{10};

    int value2{-300};

    std::cout << "value1: " << value1 << std::endl;

    std::cout << "value2: " << value2 << std::endl;

    std::cout << "sizeof(value1):" << sizeof(value1) << std::endl; // 4 bytes

    std::cout << "sizeof(value2):" << sizeof(value2) << std::endl; // 4 bytes

    std::cout << "---------------------" << std::endl;

    unsigned int value3{4};

    // unsigned int value4{-5}; // Compiler Error: negative value assigned to unsigned int

    // short and long

    short short\_var{-32768}; // 2 bytes

    short int short\_int{455};

    signed short signed\_short{122};

    signed short int signed\_short\_int{-456};

    unsigned short int unsigned\_short\_int{456};

    int int\_var {55} ; // 4 bytes

    signed signed\_var {66};//

    signed int signed\_int {77};//

    unsigned int unsigned\_int{77};

    long long\_var{88}; // 4 or 8 bytes

    long int long\_int{33};

    signed long signed\_long{44};

    signed long int signed\_long\_int{44};

    unsigned long int unsigned\_long\_int{44};

    long long long\_long{888}; // 8 bytes

    long long int long\_long\_int{999};

    signed long long signed\_long\_long{444};

    signed long long int signed\_long\_long\_int{1234};

    unsigned long long int unsigned\_long\_long\_int{1234};

    std::cout << "Short variable : " << short\_var<<  " , size : "

        << sizeof (short) << " bytes" << std::endl;

    std::cout << "Short Int : " << short\_int << " , size : "

    << sizeof (short int) << " bytes" << std::endl;

    std::cout << "Signed short : " << signed\_short

    << " , size : " << sizeof (signed short) << " bytes" << std::endl;

    std::cout << "Signed short int :  " << signed\_short\_int

    <<  " , size : " << sizeof (signed short int) << " bytes" << std::endl;

    std::cout << "unsigned short int :  " << unsigned\_short\_int

    <<  " , size : " << sizeof (unsigned short int) << " bytes" << std::endl;

    std::cout << "---------------------" << std::endl;

    std::cout << "Int variable :  " << int\_var <<" , size : "

        << sizeof (int) << " bytes" << std::endl;

    std::cout << "Signed variable " << signed\_var <<" , size : "

        << sizeof (signed) << " bytes" << std::endl;

    std::cout << "Signed int :  " << signed\_int <<" , size : "

        << sizeof (signed int) << " bytes" << std::endl;

    std::cout << "unsigned int :  " << unsigned\_int <<" , size : "

        << sizeof (unsigned int) << " bytes" << std::endl;

    std::cout << "---------------------" << std::endl;

    std::cout << "Long variable :  " << long\_var << " , size : "

        << sizeof (long) << " bytes" <<std::endl;

    std::cout << "Long int :  " << long\_int <<" , size : "

        << sizeof (long int) << " bytes" << std::endl;

    std::cout << "Signed long :  " << signed\_long <<" , size : "

        << sizeof (signed long) << " bytes" << std::endl;

    std::cout << "Signed long int : " << signed\_long\_int <<" , size : "

        << sizeof (signed long int) << " bytes" << std::endl;

    std::cout << "unsigned long int : " << unsigned\_long\_int <<" , size : "

        << sizeof (unsigned long int) << " bytes" << std::endl;

    std::cout << "---------------------" << std::endl;

    std::cout << "Long long : " << long\_long <<" , size : "

        << sizeof (long long) << " bytes" << std::endl;

    std::cout << "Long long int : " << long\_long\_int <<" , size : "

        << sizeof (long long int) << " bytes" << std::endl;

    std::cout << "Signed long long : " << signed\_long\_long <<" , size : "

        << sizeof (signed long long) << " bytes" <<std::endl;

    std::cout << "Signed long long int : " << signed\_long\_long\_int <<" , size : "

        << sizeof (signed long long int) << " bytes" << std::endl;

    std::cout << "unsigned long long int : " << unsigned\_long\_long\_int <<" , size : "

        << sizeof (unsigned long long int) << " bytes" << std::endl;

    std::cout << "---------------------" << std::endl;

    return 0;

}

|  |  |  |
| --- | --- | --- |
| **Type with modifier** | **Bytes in memory** | **Range** |
| Unsigned int | 4 |  |
| Signed int | 4 |  |

* is the number of bits for a type in memory.
* An unsigned integer can only store positive numbers.
* If the range of the stored value is larger than the range, we should consider use another type to store the value.
* Size of an integer with ***short*** type is 2 bytes.
* Size of an integer with ***long*** type is 4 bytes.
* Size of an integer with ***long long*** type is 8 bytes.

**5.8 Fractional Numbers**

#include <iostream>

#include <iomanip> // For std::setprecision()

int main(){

    // Declare and initialize the variables

    //Declare and initialize the variables

    float number1{1.12345678901234567890f};  // Size: 4bytes; Precision : 7

    double number2{1.12345678901234567890};  // Size: 8bytes; Precision : 15

    long double number3{1.12345678901234567890L}; // Size: 16bytes; Precision: 15+

    // Print out the sizes

    std::cout << "sizeof float : " << sizeof(float) << std::endl;

    std::cout << "sizeof double : " << sizeof(double) << std::endl;

    std::cout << "sizeof long double : " << sizeof(long double) << std::endl;

    //Precision

    std::cout << std::setprecision(20); // Control the precision from std::cout.

    std::cout << "number1 is : " << number1 << std::endl; //7 digits

    std::cout << "number2 is : " << number2 << std::endl; // 15'ish digits

    std::cout << "number3 is : " << number3 << std::endl; // 15+ digits

    //Float problems : The precision is usually too limited

    //for a lot of applications

    // float number4 {192400023.0f};               // Error : narrowing conversion, junk will be stored in number4

    double number4 {192400023.0};               // OK

    std::cout << "number4 : " << number4 << std::endl;

    //Scientific notation

    //What we have seen so far in terms of floating point types

    //is fixed notation. There is another notation, scientific

    //that is handy if you have really huge numbers or small numbers

    //to represent

    std::cout << "-------------------------" << std::endl;

    double number5 {192400023};

    double number6 {1.92400023e8};

    double number7 {1.924e8};           // Can ommit the lower 00023

                                        // for simplicity if our application allows that.

    double number8 {0.00000000003498};

    double number9 {3.498e-11}; // multiply with 10 exp(-11)

    std::cout << "number5 is : " << number5 << std::endl;

    std::cout << "number6 is : " << number6 << std::endl;

    std::cout << "number7 is : " << number7 << std::endl;

    std::cout << "number8 is : " << number8 << std::endl;

    std::cout << "number9 is : " << number9 << std::endl;

    //Infinity and Nan

    std::cout << std::endl;

    std::cout << "Infinity and NaN" << std::endl;

    double number10{ -5.6 };

    double number11{};//Initialized to 0

    double number12{};  //Initialized to 0

    //Infinity

    double result { number10 / number11 };

    std::cout << number10 << "/" << number11 << "  yields " << result << std::endl;

    std::cout << result << " + " << number10 << " yields " << result + number10 << std::endl;

    //NaN

    result = number11 / number12;

    std::cout << number11 << "/" << number12 << " = " << result << std::endl;

    return 0;

}

***std::setprecision***: Control the precision from std::cout. Need to ***#include <iomanip>***.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Size** | **Precision** | **Comment** |
| float | 4 | 7 | - |
| double **(Default Type)** | 8 | 15 | Recommended default |
| long double | 12 | > double |  |

* Scientific Notation
* ***double number4 {2.5f}*** still equals ***float number 4 {2.5f}***. Hence, we need to remove ***f*** to convert it to double type: ***double number4 {2.5}***.
* Special Case

**5.9 Booleans**

#include <iostream>

int main(){

    bool red\_light{false};

    bool green\_light{true};

    if(red\_light == true){

        std::cout << "Stop!" << std::endl;

    }

    else{

        std::cout << "Go through!" << std::endl;

    }

    if(green\_light == true){

        std::cout << "The light is green!" << std::endl;

    }

    else{

        std::cout << "The light is NOT green!" << std::endl;

    }

    // sizeof()

    std::cout << "sizeof(bool): " << sizeof(bool) << std::endl; // 8 bytes

    //Printing out a bool

    //1 -->> true

    //0 -->> false

    std::cout << std::endl;

    std::cout << "red\_light : " << red\_light << std::endl; // 1

    std::cout << "green\_light : " << green\_light << std::endl; // 0

    std::cout << std::boolalpha; // Print out true or false instead of 1 or 0

    std::cout << "red\_light : " << red\_light << std::endl;

    std::cout << "green\_light : " << green\_light << std::endl;

    return 0;

}

* Boolean is often used to create condition statements.

***std::cout<<std::boolalpha***: Allows printing out true or false instead of 1 or 0.

**5.10 Characters and Text**

#include <iostream>

int main(){

    char character1 {'a'};

    char character2 {'r'};

    char character3 {'r'};

    char character4 {'o'};

    char character5 {'w'};

    std::cout << character1 << std::endl;

    std::cout << character2 << std::endl;

    std::cout << character3 << std::endl;

    std::cout << character4 << std::endl;

    std::cout << character5 << std::endl;

    std::cout << "sizeof(char): " << sizeof(char) << std::endl; // 1 byte (256 different values ( 0 - 255))

    char value = 65; // ASCII character code for 'A'

    std::cout << "value: " << value << std::endl; // print A

    std::cout << "value(int): " << static\_cast<int>(value) << std::endl; // print 65

    return 0;

}

***static\_cast<int>(char)***: Convert char variable to int. Usually will be converted to ASCII.

**5.11 Auto**

#include <iostream>

int main(){

    auto var1{12}; // int

    auto var2{13.0}; // double

    auto var3{14.0f}; // float

    auto var4{15.01l}; // long double

    auto var5{'e'}; // char

    // int modifier suffixes

    auto var6{123u};    // unsigned

    auto var7{123ul};   // unsigned long

    auto var8{123ll};   // long long

    std::cout << "var1 occupies : " << sizeof(var1) << " bytes" << std::endl; // 4 bytes

    std::cout << "var2 occupies : " << sizeof(var2) << " bytes" << std::endl; // 8 bytes

    std::cout << "var3 occupies : " << sizeof(var3) << " bytes" << std::endl; // 4 bytes

    std::cout << "var4 occupies : " << sizeof(var4) << " bytes" << std::endl; // 16 bytes

    std::cout << "var5 occupies : " << sizeof(var5) << " bytes" << std::endl; // 1 byte

    std::cout << "var6 occupies : " << sizeof(var6) << " bytes" << std::endl; // 4 bytes

    std::cout << "var7 occupies : " << sizeof(var7) << " bytes" << std::endl; // 4 bytes

    std::cout << "var8 occupies : " << sizeof(var8) << " bytes" << std::endl; // 8 bytes

    return 0;

}

* To let the compiler deduce the type.
* There are several suffixes to modify int and fractional number. (e.g., *f*, ***l***, ***u***, ***ul***, ***ll***)

**5.12 Data Assignment**

#include <iostream>

int main(){

    int var1{123}; // Declare and initialize

    std::cout << "var1 : "  << var1 << std::endl;

    var1 = 55; // Assign

    std::cout << "var1 : "  << var1 << std::endl;

    std::cout << "----------------" << std::endl;

    double var2 {44.55}; // Declare and initialize

    std::cout << "var2 : " << var2 << std::endl;

    var2 = 99.99; // Assign

    std::cout << "var2 : " << var2 << std::endl;

    std::cout << "----------------" << std::endl;

    bool state{false}; // Declare and initialize

    std::cout << std::boolalpha;

    std::cout << "state : " << state << std::endl;

    state = true; // Assign

    std::cout << "state : " << state << std::endl;

    //Auto type deduction

    //Careful about auto assignments

    auto var3 {333u}; // Declare and initialize with type deduction

    var3 = -22; // Assign negative number. DANGER! (Junk will be assigned instead)

    std::cout << "var3 : " << var3 << std::endl;

    return 0;

}

* Be careful about **assignments to auto variable**, the assignment should not be different type of the auto initialized data type. Otherwise, a random number will be assigned to the variable instead.

**5.13 Basic Operations**

#include <iostream>

int main(){

    //Addition

    int number1{2};

    int number2{7};

    int result = number1 + number2;

    std::cout << "result : " << result << std::endl; // 9

    //Subtraction

    result = number2 - number1 ;

    std::cout << "result : " << result << std::endl; // 5

    result = number1 - number2;

    std::cout << "result : " << result << std::endl; // -5

    //Multiplication

    result = number1 \* number2;

    std::cout << "result : " << result << std::endl; // 14

    //Division

    result = number2 / number1;

    std::cout << "result : " << result << std::endl; // 3

    //Modulus

    result = number2 % number1; // 7 % 2

    std::cout << "result : " << result << std::endl; // 1

    result = 31 % 10;

    std::cout << "result : " << result << std::endl; // 1

    return 0;

}

* Integer Division ***/*** can only get Integer.
* Modulus ***%*** returns the remainder of the division.

**5.14 Precedence and Associativity**

#include <iostream>

int main(){

    int a {6};

    int b {3};

    int c {8};

    int d {9};

    int e {3};

    int f {2};

    int g {5};

    int result = a + b \* c -d/e -f + g; //  6 +  (3\*8)  -   (9/3) - 2 + 5

    std::cout << "result : " << result << std::endl; // 30

    result = a/b\*c +d - e + f;  //   (6/3)\*8 + 9 - 3 + 2

    std::cout << "result : " << result << std::endl; // 24

    result = (a + b) \* c -d/e -f + g; // (6+3)\*8 -(9/3) -2 + 5

    std::cout << "result () : " << result << std::endl; // 72

    return 0;

}

* A rule to follow when multiple operations exist in the same expression.
  + Precedence: which operation to do first.
  + Associativity: which direction or which order.
* It is better to make the intent in code as clear as possible by clearly using ***()*** to indicate which operations should be done first.
* C++ operator precedence table: <https://en.cppreference.com/w/cpp/language/operator_precedence>

**5.15 Prefix and Postfix + and -**

#include <iostream>

int main(){

    int value { 5 };

    //Increment by one

    value = value + 1; //6

    std::cout << "The value is : " << value << std::endl; // 6

    value = 5; // Reset value to 5

    //Decrement by one

    value = value - 1; // 4

    std::cout << "The value is : " << value << std::endl; //4

    //===================================================================

    std::cout << "======Postfix increment and decrement======"<< std::endl;

    //Reset value to 5

    value = 5;

    std::cout << "The value is (incrementing) : " << value++ << std::endl; // 5

    std::cout << "The value is : " << value << std::endl; // 6

    std::cout << std::endl;

    //Decrement with postfix

    //Reset value to 5

    value = 5;

    std::cout << "The value is (decrementing) : " << value-- << std::endl; //5

    std::cout << "The value is : " << value << std::endl; // 4

    //===================================================================

    std::cout << "======Prefix increment and decrement======"<< std::endl;

    //Reset value to 5

    value = 5;

    ++value;

    std::cout << "The value is (prefix++) : " << value << std::endl; // 6

    //Reset value to 5

    value = 5;

    std::cout << "The value is (prefix++ in place) : " << ++value << std::endl; // 6

    std::cout << std::endl;

    //Prefix : Decrementing

    //Reset value to 5;

    value = 5;

    --value;

    std::cout << "The value is (prefix--) : " << value << std::endl; // 4

    //Reset value to 5;

    value = 5;

    std::cout << "The value is (prefix-- in place) : " << --value << std::endl;//4

    return 0;

}

* Prefix and postfix increment/decrement operators can only increment/decrement by 1.
* Prefix and postfix operators are only available for + and -.
* Difference between the prefix and postfix operator is the execution order. For example,
  + ***a++***: Returns ***a*** and change ***a = a + 1***. (Postfix)
  + ***++a***: Returns ***a + 1*** and change ***a = a + 1***. (Prefix)

**5.16 Logical Operators**

#include <iostream>

int main(){

    bool a {true};

    bool b {false};

    bool c {true};

    std::cout << std::boolalpha; // print bools as true or false instead of 1 or 0

    std::cout << "a : " << a << std::endl;

    std::cout << "b : " << b << std::endl;

    std::cout << "c : " << c << std::endl;

    //AND : Evaluates to true when all operands are true.

    //      A single false operand will drag

    //      the entire expression to evaluating false.

    std::cout << std::endl;

    std::cout << "Basic AND operations" << std::endl;

    std::cout << " a && b : " << (a && b) << std::endl; // false

    std::cout << " a && c : " << (a && c ) << std::endl; // true

    std::cout << " a && b && c :" << (a && b && c) << std::endl; // false

    //OR : Evaluates to true when at least one operand true.

    //      A single true operand will push

    //      the entire expression to evaluating true.

    std::cout << std::endl;

    std::cout << "Basic OR operations" << std::endl;

    std::cout << " a || b : " << (a || b) << std::endl;

    std::cout << " a || c : " << (a || c ) << std::endl;

    std::cout << " a ||b || c :" << (a || b || c) << std::endl;

    //NOT : Negates whateve operand you put it with

    std::cout << std::endl;

    std::cout << "Basic NOT operations" << std::endl;

    std::cout << "!a : " << !a << std::endl;

    std::cout << "!b : " << !b << std::endl;

    std::cout << "!c : " << !c << std::endl;

    //Combinations of all these operators

    std::cout << std::endl;

    std::cout << "Combining logical operators" << std::endl;

    // !(a &&b) || c = !(true && false) || true = true || true = true

    std::cout << "!(a && b) || c : " << (!(a &&b) || c) << std::endl;

    //Combining logical operators with relational operators

    int d{45};

    int e{20};

    int f{11};

    std::cout << std::endl;

    std::cout << "Relational and logic operations on integers" << std::endl;

    std::cout << "d : " << d << std::endl;

    std::cout << "e : " << e << std::endl;

    std::cout << "f : " << f << std::endl;

    std::cout << std::endl;

    std::cout << "(d > e) && (d > f) : " << ((d > e) && (d > f)) << std::endl; // true

    std::cout << "(d == e) || (e <= f ) : " << ((d==e) || (e <= f ) ) << std::endl;

    std::cout << "(d < e) || (d > f) : " << ((d < e) || (d > f)) << std::endl;

    std::cout << "(f > e) || (d < f) : " << ((f > e) || (d < f)) << std::endl;

    std::cout << "(d > f) && (f <= d) : " << ((d > f) && (f <= d)) << std::endl;

    std::cout << "(d > e) && (d <= f) : " << ((d > e) && (d <= f)) << std::endl;

    std::cout << "(! a) && (d == e) : " << ((! a) && (d == e)) << std::endl;

    std::cout << "(! a) && (d == e) : " << ((! a) && (d == e)) << std::endl;

    return 0;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Logical Table** | | | | |
| **a** | **b** | **a && b (AND)** | **a || b**  **(OR)** | **!a**  **(NOT)** |
| **False** | **False** | **False** | **False** | **True** |
| **False** | **True** | **False** | **True** | **True** |
| **True** | **False** | **False** | **True** | **False** |
| **True** | **True** | **True** | **True** | **False** |

**5.17 Output Formatting**

#include <iostream>

#include <iomanip>

int main(){

    //std::endl : places a new line character on the output stream.

    //       This is identical to placing '\n' on the output stream.

    std::cout << "Hello";

    std::cout << "World";

    std::cout << std::endl;

    std::cout << "-------------" << std::endl;

    std::cout << "Hello" << std::endl;

    std::cout << "World" << std::endl;

    std::cout << std::endl;

    std::cout << "Hello\n";

    std::cout << "World\n";

    //===================================================================

    std::cout << std::endl;

    //std::flush : flushes the output buffer to its final destination.

    std::cout << "This is a nice message...." << std::endl << std::flush;

    //After this std::flush, we're sure that at this line, the message has been sent

    //to the stream. This may be important in some applications.

    //===================================================================

    std::cout << std::endl;

    //std::setw() : Adjusts the field with for the item about to be printed.

    //The setw() manipulator only affects the next value to be printed.

    std::cout << "Unformatted table : " << std::endl;

    std::cout << "Daniel" << " " << "Gray" << " 25" << std::endl;

    std::cout << "Stanley" <<" "  << "Woods" << " 33" << std::endl;

    std::cout << "Jordan" << " "  << "Parker" << " 45" << std::endl;

    std::cout << "Joe" << " " << "Ball" << " 21" << std::endl;

    std::cout << "Josh" << " " << "Carr" << " 27" << std::endl;

    std::cout << "Izaiah" << " " << "Robinson" << " 29" << std::endl;

    std::cout << std::endl;

    std::cout << "Formatted table : " << std::endl;

    std::cout << std::setw(10) <<  "Lastname"  << std::setw(10) << "Firstname" << std::setw(5) << "Age" << std::endl;

    std::cout << std::setw(10) << "Daniel"  << std::setw(10) << "Gray" << std::setw(5) << "25" << std::endl;

    std::cout << std::setw(10) << "Stanley" << std::setw(10)  << "Woods" << std::setw(5) <<  "33" << std::endl;

    std::cout << std::setw(10) <<  "Jordan" << std::setw(10)  << "Parker" << std::setw(5) << "45" << std::endl;

    std::cout << std::setw(10) <<  "Joe" << std::setw(10) << "Ball" << std::setw(5) << "21" << std::endl;

    std::cout << std::setw(10) << "Josh" << std::setw(10) << "Carr" << std::setw(5) <<"27" << std::endl;

    std::cout << std::setw(10) << "Izaiah" << std::setw(10) << "Robinson" << std::setw(5) << "29" << std::endl;

    std::cout << std::endl;

    std::cout << "Formatted table with variables: " << std::endl;

    int col\_width{14};

    std::cout << std::setw(col\_width) <<  "Lastname"  << std::setw(col\_width) << "Firstname" << std::setw(col\_width/2) << "Age" << std::endl;

    std::cout << std::setw(col\_width) << "Daniel"  << std::setw(col\_width) << "Gray" << std::setw(col\_width/2) << "25" << std::endl;

    std::cout << std::setw(col\_width) << "Stanley" << std::setw(col\_width)  << "Woods" << std::setw(col\_width/2) <<  "33" << std::endl;

    std::cout << std::setw(col\_width) <<  "Jordan" << std::setw(col\_width)  << "Parker" << std::setw(col\_width/2) << "45" << std::endl;

    std::cout << std::setw(col\_width) <<  "Joe" << std::setw(col\_width) << "Ball" << std::setw(col\_width/2) << "21" << std::endl;

    std::cout << std::setw(col\_width) << "Josh" << std::setw(col\_width) << "Carr" << std::setw(col\_width/2) <<"27" << std::endl;

    std::cout << std::setw(col\_width) << "Izaiah" << std::setw(col\_width) << "Robinson" << std::setw(col\_width/2) << "29" << std::endl;

    //===================================================================

    std::cout << std::endl;

    //Justify : Values can be justified in their fields. There are three manipulators

    //          for adjusting the justification: left, right, and internal.

    //right justified

    std::cout << std::endl;

    std::cout << "Right justified table(default) :  " << std::endl;

    col\_width = 20;

    std::cout << std::right;

    std::cout << std::setw(col\_width) <<  "Lastname"  << std::setw(col\_width) << "Firstname" << std::setw(col\_width/2) << "Age" << std::endl;

    std::cout << std::setw(col\_width) << "Daniel"  << std::setw(col\_width) << "Gray" << std::setw(col\_width/2) << "25" << std::endl;

    std::cout << std::setw(col\_width) << "Stanley" << std::setw(col\_width)  << "Woods" << std::setw(col\_width/2) <<  "33" << std::endl;

    std::cout << std::setw(col\_width) <<  "Jordan" << std::setw(col\_width)  << "Parker" << std::setw(col\_width/2) << "45" << std::endl;

    std::cout << std::setw(col\_width) <<  "Joe" << std::setw(col\_width) << "Ball" << std::setw(col\_width/2) << "21" << std::endl;

    std::cout << std::setw(col\_width) << "Josh" << std::setw(col\_width) << "Carr" << std::setw(col\_width/2) <<"27" << std::endl;

    std::cout << std::setw(col\_width) << "Izaiah" << std::setw(col\_width) << "Robinson" << std::setw(col\_width/2) << "29" << std::endl;

    //Left justified

    std::cout << std::endl;

    std::cout << "Left justified table :  " << std::endl;

    col\_width = 20;

    std::cout << std::left;

    std::cout << std::setw(col\_width) <<  "Lastname"  << std::setw(col\_width) << "Firstname" << std::setw(col\_width/2) << "Age" << std::endl;

    std::cout << std::setw(col\_width) << "Daniel"  << std::setw(col\_width) << "Gray" << std::setw(col\_width/2) << "25" << std::endl;

    std::cout << std::setw(col\_width) << "Stanley" << std::setw(col\_width)  << "Woods" << std::setw(col\_width/2) <<  "33" << std::endl;

    std::cout << std::setw(col\_width) <<  "Jordan" << std::setw(col\_width)  << "Parker" << std::setw(col\_width/2) << "45" << std::endl;

    std::cout << std::setw(col\_width) <<  "Joe" << std::setw(col\_width) << "Ball" << std::setw(col\_width/2) << "21" << std::endl;

    std::cout << std::setw(col\_width) << "Josh" << std::setw(col\_width) << "Carr" << std::setw(col\_width/2) <<"27" << std::endl;

    std::cout << std::setw(col\_width) << "Izaiah" << std::setw(col\_width) << "Robinson" << std::setw(col\_width/2) << "29" << std::endl;

    //Internal justified : sign is left justified , data is right justified

    std::cout << std::endl;

    std::cout << "Internal justified : " << std::endl;

    std::cout << std::right;

    std::cout << std::setw(10) << -123.45 << std::endl;

    std::cout << std::internal;

    std::cout << std::setw(10) << -123.45 << std::endl;

    //===================================================================

    std::cout << std::endl;

    //setfill

    std::cout << std::endl;

    std::cout << "Table with fill characters :  " << std::endl;

    col\_width = 20;

    std::cout << std::left;

    std::cout << std::setfill('\*'); // The fill character

    std::cout << std::setw(col\_width) <<  "Lastname"  << std::setw(col\_width) << "Firstname" << std::setw(col\_width/2) << "Age" << std::endl;

    std::cout << std::setw(col\_width) << "Daniel"  << std::setw(col\_width) << "Gray" << std::setw(col\_width/2) << "25" << std::endl;

    std::cout << std::setw(col\_width) << "Stanley" << std::setw(col\_width)  << "Woods" << std::setw(col\_width/2) <<  "33" << std::endl;

    std::cout << std::setw(col\_width) <<  "Jordan" << std::setw(col\_width)  << "Parker" << std::setw(col\_width/2) << "45" << std::endl;

    std::cout << std::setw(col\_width) <<  "Joe" << std::setw(col\_width) << "Ball" << std::setw(col\_width/2) << "21" << std::endl;

    std::cout << std::setw(col\_width) << "Josh" << std::setw(col\_width) << "Carr" << std::setw(col\_width/2) <<"27" << std::endl;

    std::cout << std::setw(col\_width) << "Izaiah" << std::setw(col\_width) << "Robinson" << std::setw(col\_width/2) << "29" << std::endl;

    //===================================================================

    std::cout << std::endl;

    //boolalpha and noboolapha : control bool output format : 1/0 or true/false

    bool condition {true};

    bool other\_condition {false};

    std::cout << "condition : " << condition << std::endl;

    std::cout << "other\_condition : " << other\_condition << std::endl;

    std::cout << std::endl;

    std::cout << std::boolalpha;

    std::cout << "condition : " << condition << std::endl;

    std::cout << "other\_condition : " << other\_condition << std::endl;

    std::cout << std::endl;

    std::cout << std::noboolalpha;

    std::cout << "condition : " << condition << std::endl;

    std::cout << "other\_condition : " << other\_condition << std::endl;

    //===================================================================

    std::cout << std::endl;

    //showpos and noshowpos : show or hide the +  sign for positive numbers

    int pos\_num {34};

    int neg\_num {-45};

    std::cout << "pos\_num : " << pos\_num << std::endl;

    std::cout << "neg\_num : " << neg\_num << std::endl;

    std::cout << std::endl;

    std::cout << std::showpos;

    std::cout << "pos\_num : " << pos\_num << std::endl;

    std::cout << "neg\_num : " << neg\_num << std::endl;

    std::cout << std::endl;

    std::cout << std::noshowpos;

    std::cout << "pos\_num : " << pos\_num << std::endl;

    std::cout << "neg\_num : " << neg\_num << std::endl;

    //===================================================================

    std::cout << std::endl;

    //different number systems : std::dec, std::hex, std::oct

    int pos\_int {717171};

    int neg\_int {-47347};

    double double\_var {498.32};

    std::cout << std::endl;

    std::cout << "default base format : " << std::endl;

    std::cout << "pos\_int : " << pos\_int << std::endl;

    std::cout << "neg\_int : " << neg\_int << std::endl;

    std::cout << "double\_var : " << double\_var << std::endl;

    std::cout << std::endl;

    std::cout << "pos\_int in different bases : " << std::endl;

    std::cout << "pos\_int (dec) : " << std::dec << pos\_int << std::endl;

    std::cout << "pos\_int (hex) : " << std::hex << pos\_int << std::endl;

    std::cout << "pos\_int (oct) : " << std::oct << pos\_int << std::endl;

    std::cout << std::endl;

    std::cout << "neg\_int in different bases : " << std::endl;

    std::cout << "neg\_int (dec) : " << std::dec << neg\_int << std::endl;

    std::cout << "neg\_int (hex) : " << std::hex << neg\_int << std::endl;

    std::cout << "neg\_int (oct) : " << std::oct << neg\_int << std::endl;

    std::cout << std::endl;

    std::cout << "double\_var in different bases : " << std::endl;

    std::cout << "double\_var (dec) : " << std::dec << double\_var << std::endl;

    std::cout << "double\_var (hex) : " << std::hex << double\_var << std::endl;

    std::cout << "double\_var (oct) : " << std::oct << double\_var << std::endl;

    std::cout << "pos\_int (showbase):" << std::endl;

    std::cout << std::showbase;

    std::cout << "pos\_int (dec) : " << std::dec << pos\_int << std::endl; // 717171

    std::cout << "pos\_int (hex) : " << std::hex << pos\_int << std::endl; // 0xaf173

    std::cout << "pos\_int (oct) : " << std::oct << pos\_int << std::endl; // 02570563

    //===================================================================

    std::cout << std::endl;

    //uppercase and nouppercase

    pos\_int = 717171;

    std::cout << "pos\_int (nouppercase : default) : " << std::endl;

    std::cout << "pos\_int (dec) : " << std::dec << pos\_int << std::endl;

    std::cout << "pos\_int (hex) : " << std::hex << pos\_int << std::endl;

    std::cout << "pos\_int (oct) : " << std::oct << pos\_int << std::endl;

    std::cout << std::endl;

    std::cout << "pos\_int (uppercase) : " << std::endl;

    std::cout << std::uppercase;

    std::cout << "pos\_int (dec) : " << std::dec << pos\_int << std::endl;

    std::cout << "pos\_int (hex) : " << std::hex << pos\_int << std::endl;

    std::cout << "pos\_int (oct) : " << std::oct << pos\_int << std::endl;

    //===================================================================

    std::cout << std::endl;

    //fixed and scientific : for floating point values

    double a{ 3.1415926535897932384626433832795 };

    double b{ 2006.0 };

    double c{ 1.34e-10 };

    std::cout << std::endl;

    std::cout << "double values (default : use scientific where necessary) : " << std::endl;

    std::cout << "a : " << a << std::endl;

    std::cout << "b : " << b << std::endl;

    std::cout << "c : " << c << std::endl;

    std::cout << std::endl;

    std::cout << "double values (fixed) : " << std::endl;

    std::cout << std::fixed;

    std::cout << "a : " << a << std::endl;

    std::cout << "b : " << b << std::endl;

    std::cout << "c : " << c << std::endl;

    std::cout << std::endl;

    std::cout << "double values (scientific) : " << std::endl;

    std::cout << std::scientific;

    std::cout << "a : " << a << std::endl;

    std::cout << "b : " << b << std::endl;

    std::cout << "c : " << c << std::endl;

    std::cout << std::endl;

    std::cout << "double values (back to defaults) : " << std::endl;

    std::cout.unsetf(std::ios::scientific | std::ios::fixed); // Hack

    std::cout << "a : " << a << std::endl;

    std::cout << "b : " << b << std::endl;

    std::cout << "c : " << c << std::endl;

    //===================================================================

    std::cout << std::endl;

    //setprecision() : the number of digits printed out for a floating point. Default is 6

    a = 3.1415926535897932384626433832795;

    std::cout << std::endl;

    std::cout << "a (default precision(6)) : " << a <<  std::endl;

    std::cout << std::setprecision(10);

    std::cout << "a (precision(10)) : " << a << std::endl;

    std::cout << std::setprecision(20);

    std::cout << "a (precision(20)) : " << a << std::endl;

    //If the precision is bigger than supported by the type, you'll just print garbage.

    //===================================================================

    std::cout << std::endl;

    //showpoint and noshowpoint : show trailing zeros if necessary

    //Force output of the decimal point

    double d {34.1};

    double e {101.99};

    double f {12.0};

    int    g {45};

    std::cout << std::endl;

    std::cout << "noshowpoint (default) : " << std::endl;

    std::cout << "d : " << d << std::endl;

    std::cout << "e : " << e << std::endl;

    std::cout << "f : " << f << std::endl; // 12

    std::cout << "g : " << g << std::endl;

    std::cout << std::endl;

    std::cout << "showpoint: " << std::endl;

    std::cout << std::showpoint;

    std::cout << "d : " << d << std::endl;

    std::cout << "e : " << e << std::endl;

    std::cout << "f : " << f << std::endl; // 12.0

    std::cout << "g : " << g << std::endl;

    return 0;

}

**5.18 Numeric Limits**

#include <iostream>

#include <limits>

int main(){

    std::cout << "The range for short is from " << std::numeric\_limits<short>::min() << " to "

        << std::numeric\_limits<short>::max() << std::endl;

    std::cout << "The range for unsigned short is from " << std::numeric\_limits<unsigned short>::min() << " to "

        << std::numeric\_limits<unsigned short>::max() << std::endl;

    std::cout << "The range for int is from " << std::numeric\_limits<int>::min() << " to "

        << std::numeric\_limits<int>::max() << std::endl;

    std::cout << "The range for unsigned int is from " << std::numeric\_limits<unsigned int>::min() << " to "

        << std::numeric\_limits<unsigned int>::max() << std::endl;

    std::cout << "The range for long is from " << std::numeric\_limits<long>::min() << " to "

        << std::numeric\_limits<long>::max() << std::endl;

    std::cout << "The range for float is from " << std::numeric\_limits<float>::min() << " to "

        << std::numeric\_limits<float>::max() << std::endl;

    std::cout << "The range(with lowest) for float is from " << std::numeric\_limits<float>::lowest() << " to "

        << std::numeric\_limits<float>::max() << std::endl;

    std::cout << "The range(with lowest) for double is from " << std::numeric\_limits<double>::lowest() << " to "

        << std::numeric\_limits<double>::max() << std::endl;

    std::cout << "The range(with lowest) for long double is from " << std::numeric\_limits<long double>::lowest() << " to "

        << std::numeric\_limits<long double>::max() << std::endl;

    //Other facilities

    //More info : https://en.cppreference.com/w/cpp/types/numeric\_limits

    std::cout << "int is signed : " << std::numeric\_limits<int>::is\_signed << std::endl;

    std::cout << "int digits : " << std::numeric\_limits<int>::digits << std::endl;

    return 0;

}

* ***#include <limits>*** are necessary.

|  |  |
| --- | --- |
| **<limits> functions** | |
| ***std::numeric\_limits<T>::min()*** | returns the **minimum finite value** representable by the numeric type T. For floating-point types with denormalization, it returns the minimum positive normalized value. |
| ***std::numeric\_limits<T>::max()*** | returns the **maximum finite value** representable by the numeric type T. |
| ***std::numeric\_limits<T>::lowest()*** | returns the **lowest finite value** representable by the numeric type T, which is equal to -max() for signed types and to min() for unsigned types1. |

* More information of limits library: <https://en.cppreference.com/w/cpp/types/numeric_limits>

**5.19 Math Functions**

#include <iostream>

#include <cmath>

int main(){

    double weight { 7.7 };

    //floor

    std::cout << "Weight rounded to floor is : " << std::floor(weight) << std::endl;

    //ceil

    std::cout << "Weight rounded to ceil is : " << std::ceil(weight) << std::endl;

    //abs

    double savings {-5000 };

    std::cout << "Abs of weight is : " << std::abs(weight) << std::endl;

    std::cout << "Abs of savings is : " << std::abs(savings) << std::endl;

    //exp : f(x) = e ^ x , where e = 2.71828 . Test the result here against a calculator

    double exponential = std::exp(10);

    std::cout << "The exponential of 10 is : " << exponential << std::endl;

    //pow

    std::cout << "3 ^ 4 is : " << std::pow(3,4) << std::endl;

    std::cout << "9^3 is : " << std::pow(9,3) << std::endl;

    //log : reverse function of pow. if 2^3 = 8 , log 8 in base 2 = 3.  Log is like asking

    // to which exponent should we elevate 2 to get eight ? Log, by default computes the log

    // in base e. There also is another function which uses base 10 called log10

    // Try the reverse operation of  e^4 = 54.59 , it will be log 54.59 in base e = ?

    std::cout << "Log ; to get 54.59, you would elevate e to the power of : "

             << std::log(54.59) << std::endl;

    //log10 , 10 ^ 4 = 10000  , to get 10k , you'd need to elevate 10 to the power of ? , this is log in base 10

    std::cout << "To get 10000, you'd need to elevate 10 to the power of : "

                 << std::log10(10000) << std::endl; // 4

    //sqrt

    std::cout << "The square root of 81 is : " << std::sqrt(81) << std::endl;

    //round. Halfway points are rounded away from 0. 2,5 is rounded to 5 for example

    std::cout << "3.654 rounded to : " << std::round(3.654) << std::endl;

    std::cout << "2.5 is rounded to : " << std::round(2.5) << std::endl;

    std::cout << "2.4 is rounded to : " << std::round(2.4) << std::endl;

    std::cout << std::max(5, 10) << std::endl; // 10

    std::cout << std::min(5, 10) << std::endl; // 5

    return 0;

}

|  |  |
| --- | --- |
| **Common Math Functions** | |
| ***std::floor(x)*** | Nearest integer not greater than the given value |
| ***std::ceil(x)*** | Nearest integer not less than the given value |
| ***std::abs(x)*** | Absolute value of a floating point value () |
| ***std::exp(x)*** | Exponential integral |
| ***std::pow(x,index)*** | Raises a number to the given power () |
| ***std::log(x)*** | Computes natural (base ) logarithm () |
| ***std::log10(x)*** | Computes common (base ) logarithm () |
| ***std::sqrt(x)*** | computes square root () |
| ***std::round(x)*** | Nearest integer, rounding away from zero in halfway cases |
| ***std::max(x,y)*** | Find the highest value of and |
| ***std::min(x,y)*** | Find the lowest value of and |

* Most Math functions live in the ***<cmath>*** library.
* More functions from <https://en.cppreference.com/w/cpp/header/cmath>

**5.20 Weird Integral Types**

#include <iostream>

int main(){

    short int var1 {10}; // 2 bytes

    short int var2 {20};

    char var3 {40}; //1 bytes

    char var4 {50};

    std::cout << "size of var1 : " << sizeof(var1) << std::endl; // 2 bytes

    std::cout << "size of var2 : " << sizeof(var2) << std::endl; // 2 bytes

    std::cout << "size of var3 : " << sizeof(var3) << std::endl; // 1 bytes

    std::cout << "size of var4 : " << sizeof(var4) << std::endl; // 1 bytes

    short int result1 = var1 + var2 ;

    auto result2 = var3 + var4;

    std::cout << "size of result1 : " << sizeof(result1) << std::endl; // 2 bytes

    std::cout << "size of result2 : " << sizeof(result2) << std::endl; // 4 bytes

    return 0;

}

* Integral types less than 4 bytes (e.g., char, short int) in size don’t support arithmetic operations (e.g., addition, subtraction).
* Using ***auto*** can convert integral types for these operations.

**5.21 If Statements**

#include <iostream>

int main(){

    //Nesting if statements

    std::cout << std::endl;

    std::cout << "Nesting if statements" << std::endl;

    bool red = false;

    bool green {true};

    bool yellow {false};

    bool police\_stop{true};

     std::cout << std::endl;

     std::cout << "Police officer stops(less verbose)" << std::endl;

     if(green && !police\_stop){

         std::cout << "Go" << std::endl;  // green and police has not stopped them

     }else{

         std::cout << "Stop" << std::endl;

     }

    return 0;

}

* The example above prints "Stop".

**5.22 Else If Statements**

#include <iostream>

// Tools

const int Pen{ 10 };

const int Marker{ 20 };

const int Eraser{ 30 };

const int Rectangle{ 40 };

const int Circle{ 50 };

const int Ellipse{ 60 };

int main(){

    int tool {Eraser};

    if (tool == Pen) {

        std::cout << "Active tool is pen" << std::endl;

        //Do the actual painting

    }

    else if (tool == Marker) {

        std::cout << "Active tool is Marker" << std::endl;

    }

    else if (tool == Eraser) {

        std::cout << "Active tool is Eraser" << std::endl;

    }

    else if (tool == Rectangle) {

        std::cout << "Active tool is Rectangle" << std::endl;

    }

    else if (tool == Circle) {

        std::cout << "Active tool is Circle" << std::endl;

    }

    else if (tool == Ellipse) {

        std::cout << "Active tool is Ellipse" << std::endl;

    }

    std::cout << "Moving on" << std::endl;

    return 0;

}

* Syntax:

if (condition1) {  
  *// block of code to be executed if condition1 is true*  
} else if (condition2) {  
  *// block of code to be executed if the condition1 is false and condition2 is true*  
} else {  
  *// block of code to be executed if the condition1 is false and condition2 is false*  
}

* Only one block in the whole chain of statements is going to execute.
* The example above executes the ***else if (tool == Ellipse)*** block.

**5.23 Switch**

#include <iostream>

#include <string>

// Tools

const int Pen{ 10 };

const int Marker{ 20 };

const int Eraser{ 30 };

const int Rectangle{ 40 };

const int Circle{ 50 };

const int Ellipse{ 60 };

int main(){

    int tool {Eraser};

    switch (tool)

    {

        case Pen : {

             std::cout << "Active tool is Pen" << std::endl;

        }

        break;

        case Marker : {

             std::cout << "Active tool is Marker" << std::endl;

        }

        break;

        case Eraser :

        case Rectangle :

        case Circle : {

             std::cout << "Drawing Shapes" << std::endl;

        }

        break;

        case Ellipse : {

             std::cout << "Active tool is Ellipse" << std::endl;

        }

        break;

        default: {

            std::cout << "No match found" << std::endl;

        }

            break;

    }

    std::cout << "Moving on" << std::endl;

    /\*

    // Condition can only be integer of enum (We'll learn about enums later in the course)

    std::string name {"John"};

    switch (name) // Compiler error!

    {

    }

    \*/

    return 0;

}

* Syntax:

switch(expression) {  
  case x:  
    *// code block*  
    break;  
  case y:  
    *// code block*  
    break;  
  default:  
    *// code block*  
}

* The example above is the same as he example in 5.22.
* It is a much more compact way than Else If to test for several different conditions. Hence, the switch expression is **evaluated once**.
* **The condition can only be integral types and enums**: int, long, unsigned short, etc.
* **Default** blocks are needed for executing when no conditions match.

***break;*** The **break** statement after each case is very important. It stops processing the switch block when a successful case has been found. If the break statement is not there, all the cases following the current case will be executed.

**5.24 Ternary Operaters**

#include <iostream>

int main(){

     int max{};

     int a{35};

     int b{200};

     std::cout << std::endl;

     // std::cout << "using regular if " << std::endl;

     /\*

     if(a >  b){

          max = a;

     }else{

          max = b;

     }

     \*/

     max = (a > b)? a : b; // Ternary operator

     std::cout << "max : " << max << std::endl; // 200

     return 0;

}

|  |
| --- |
| **Ternary Expression** |
| ***result = (condition) ? option1 : option2;*** |

* An alternative way (a short-hand) to do test with if statement.
* Types of option1 and option2 must match or be convertible. (***auto*** can be used for convertible for two different types.)

**5.25 For Loop**

#include <iostream>

int main(){

    //Print I love C++ 10 times : The bad way

    /\*

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    \*/

   //for loop : the good way

    /\*

   for( unsigned int i{0} ; i < 10000 ;++i){

       //Whatever we want the loop to run

       std::cout << i <<  " : I love C++" << std::endl;

   }

   std::cout << "Loop done!" << std::endl;

   \*/

  //Use size\_t : a representation of some unsigned int for positive numbers [sizes]

  /\*

  for(size\_t i{0} ; i < 10 ; ++i){

      std::cout << i << " : I love C++" << std::endl;

  }

  std::cout << "Loop done!" << std::endl;

  \*/

    /\*

    //sizeof(size\_t)

    std::cout << "sizeof(size\_t) : " << sizeof(size\_t) << std::endl;

    \*/

    //Scope of the iterator

    /\*

    for(size\_t i{0} ; i < 10 ; ++i){

        std::cout << i << " : I love C++" << std::endl;

    }

    std::cout << "Loop done!" << std::endl;

    // std::cout << "i : " << i << std::endl;Compiler error : i is not in scope

    \*/

   //Iterator declared outside the loop

   /\*

    size\_t i{0}; // Iterator defined outside

    for(i ; i < 10 ; ++i){

        std::cout << i << " : I love C++" << std::endl;

    }

    std::cout << "Loop done!" << std::endl;

    std::cout << "i : " << i << std::endl;

    \*/

    //Leave out the iterator declaration part

    /\*

    size\_t i{0}; // Iterator defined outside

    for(  ; i < 10 ; ++i){

        std::cout << i << " : I love C++" << std::endl;

    }

    std::cout << "Loop done!" << std::endl;

    std::cout << "i : " << i << std::endl;

    \*/

   //Don't hard code values : BAD!

    const size\_t COUNT{10}; // Initialize values outside the loop

    for(size\_t i{0} ; i < COUNT ; ++i){

        std::cout << i << " : I love C++" << std::endl;

    }

    std::cout << "Loop done!" << std::endl;

    return 0;

}

* Syntax of for loop:

const size\_t COUNT{10}; // Initialize values outside the loop

    for(size\_t i{0} ; i < COUNT ; ++i){

        std::cout << i << " : I love C++" << std::endl;

    }

* 1. Pillars of any loop
     + Iterator (e.g., unsigned int i;)
     + Starting Point (e.g., 0)
     + Test (Controls when the loop stops) (e.g., i < 10;)
     + Increment (Decrement) (e.g., ++i)
     + Loop body (e.g., std::cout << “I love C++” << std::endl;)
* Iterator can also be represented by **size\_t**.

***size\_t***: Not a type, just a type alias for some unsigned int representations. It can only represent positive numbers.

* It can leave out ***{}*** on the loop body for a single statement in the body.
* If an iterator is defined inside the loop, it can’t be used outside the loop.
* Steps of For Loop:

1. Initialization is performed (i=0)
2. The check is performed (i < n)
3. The code in the loop is executed.
4. The value is incremented.
5. Repeat steps 2 - 4

* It is bad to hard code the value of Test, we should initialize the count variable outside the loop.

***break;***: Jump out of a loop

***continue;***: **Breaks one iteration** (in the loop), if a specified condition occurs and continues with the next iteration in the loop.

**5.26 While Loop**

#include <iostream>

int main(){

    //Print I love C++ 10 times

    /\*

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    std::cout << "I love C++" << std::endl;

    \*/

    const size\_t COUNT{10};

    size\_t i{0}; // Iterator declaration

    while(i < COUNT ){ // Test

       std::cout << i << " : I love C++" << std::endl;

       ++i; // Incrementation

    }

    std::cout << "Loop done!" << std::endl;

    return 0;

}

* Syntax of while loop:

while (condition) {  
*// code block to be executed*  
}

**5.27 Do While Loop**

#include <iostream>

int main(){

    //Print I love C++ 10 times

    const int COUNT{10};

    size\_t i{0}; // Iterator declaration

    do{

        std::cout << i << " : I love C++" << std::endl;

        ++i; // Incrementation

    }while( i < COUNT);

    std::cout << "Loop done!" << std::endl;

    return 0;

}

* Syntax:

const int COUNT{10};

size\_t i{0}; // Iterator declaration

do{

    std::cout << i << " : I love C++" << std::endl;

    ++i; // Incrementation

  }while( i < COUNT);

* **This loop will execute the code block once before checking if the condition is true**, then it will repeat the loop as long as the condition is true.

**5.28 Declaring and Using Arrays**

#include <iostream>

int main(){

    // Declare an array of ints

    int scores[10];

    // Read data (garbage value before initialization)

    // std::cout << "scores[0]:" << scores[0] << std::endl;

    // std::cout << "scores[1]:" << scores[1] << std::endl;

    // Input with a loop

    for (size\_t i{0}; i < 10; i++){

        scores[i] = i \* 10;

    }

    // Output with a loop

    for (size\_t i{0}; i < 10; i++){

        std::cout << "scores[" << i << "]:" << scores[i] << std::endl;

    }

    std::cout << "---------------------------" << std::endl;

    //Declare and initialize at the same time

    double salaries[5] {12.7, 7.5, 13.2, 8.1, 9.3};

    for(size\_t i{0}; i < 5; ++i){

        std::cout << "salary[" << i << "] : " << salaries[i] << std::endl;

    }

    std::cout << "---------------------------" << std::endl;

    //If you don't initialize all the elements, those you leave out

    //are initialized to 0

    int families[5] {12, 7, 5};

    for(size\_t i{0}; i < 5; ++i){

        std::cout << "families[" << i << "] : " << families[i] << std::endl;

    }

    std::cout << "---------------------------" << std::endl;

    //Omit the size of the array at declaration

    int class\_sizes[] {10,12,15,11,18,17,23,56};

    // Will print this with a range based for loop

    for(auto value : class\_sizes){

        std::cout << "value : " << value << std::endl;

    }

    std::cout << "---------------------------" << std::endl;

   // Read only arrays (Can't change the value of the array)

   const int birds[] {10,12,15,11,18,17,23,56};

   //birds[2] = 8;

    //Sum up scores array, store result in sum

    int scoress [] {2,5,8,2,5,6,9};

    int sum {0};

    for( int element : scoress){

        sum += element;

    }

    std::cout << "Score sum : " << sum << std::endl;

    std::cout << "---------------------------" << std::endl;

    return 0;

}

* Arrays are used to **store multiple values in a single variable** instead of declaring separate variables for each value.
* An array can only store elements of the **same type.**
* Once the array is declared, some garbage data is used to fill the unused place.
* Syntax (Declare):

data type variableName[array size]

* Syntax (Declare and initialize at the same time):

data type variableName[array size] = {element 1, element 2,…, element n};

**OR**

data type variableName[] = {element 1, element 2,…,element n};

* Syntax (For-Each loop): loop through elements in an array

for (data type variableName : arrayName) {  
  // code block to be executed  
}

**5.29 Size of an Array**

#include <iostream>

int main(){

    int scores [] {1,2,5};

    int count1 { std::size(scores)}; // std::size(C++17)

    std::cout << "count1 : " << count1 << std::endl;    // 3

    std::cout << "-------------------------" << std::endl;

    std::cout << "sizeof(scores) : " << sizeof(scores) << std::endl;    // 12

    std::cout << "sizeof(scores[0]) : " << sizeof(scores[0]) << std::endl;// 4

    int count2 {sizeof(scores)/sizeof(scores[0])};

    std::cout << "count2 : " << count2 << std::endl; // 3

    std::cout << "-------------------------" << std::endl;

    for(size\_t i {0} ; i < count2 ; ++i){

        std::cout << "scores [" << i << "] : " << scores[i] << std::endl;

    }

    std::cout << "-------------------------" << std::endl;

   //Range based for loop (Same as the above for loop)

    for ( auto i : scores){

            std::cout << "value  : " << i << std::endl;

    }

    std::cout << "-------------------------" << std::endl;

    return 0;

}

* Two ways to find the size of an array.
  1. ***std::size(array)***: Get the number of elements in an array. Introduced since C++17.
  2. ***sizeof(array)***: Returns the size of array in bytes.

***sizeof(array[0])***: Returns the size of an element in bytes.

***sizeof(array)/sizeof(array[0])***: Get the number of elements in an array.

* Using size in for loop can **dynamically change** the ending point of the loops (e.g., Sometimes the array may be changed dynamically.). It is more sustainable.

**5.30 Multi-Dimensional Arrays**

#include <iostream>

#include <string>

int main(){

    // Initialization

    std::string letters[2][4] = {

    { "A", "B", "C", "D" },

    { "E", "F", "G", "H" }

    };

    std::cout << letters[0][2] << std::endl; // Outputs "C"

    std::cout << "-------------------------" << std::endl;

    // Change Elements in a multidimensional array

    letters[0][0] = "Z";

    std::cout << letters[0][0] << std::endl;  // Now outputs "Z" instead of "A"

    std::cout << "-------------------------" << std::endl;

    // Loop through a multidimensional array

    for (size\_t i = 0; i < 2; i++){

        for (size\_t j = 0; j < 4;j++){

            std::cout << letters[i][j] << std::endl;

        }

    }

    std::cout << "-------------------------" << std::endl;

    // Spaceship game

    // We put "1" to indicate there is a ship.

    bool ships[4][4] = {

    { 0, 1, 1, 0 },

    { 0, 0, 0, 0 },

    { 0, 0, 1, 0 },

    { 0, 0, 1, 0 }

    };

    // Keep track of how many hits the player has

    // and how many turns they have played in these variables

    int hits = 0;

    int numberOfTurns = 0;

    // Allow the player to keep going until they have hit all four ships

    while (hits < 4) {

        int row, column;

        std::cout << "Selecting coordinates\n";

        // Ask the player for a row

        std::cout << "Choose a row number between 0 and 3: ";

        std::cin >> row;

        // Ask the player for a column

        std::cout << "Choose a column number between 0 and 3: ";

        std::cin >> column;

        // Check if a ship exists in those coordinates

        if (ships[row][column]) {

            // If the player hit a ship, remove it by setting the value to zero.

            ships[row][column] = 0;

            // Increase the hit counter

            hits++;

            // Tell the player that they have hit a ship

// and how many ships are left

            std::cout << "Hit! " << (4-hits) << " left.\n\n";

        }

        else {

            // Tell the player that they missed

            std::cout << "Miss\n\n";

        }

        // Count how many turns the player has taken

        numberOfTurns++;

    }

    std::cout << "Victory!\n";

    std::cout << "You won in " << numberOfTurns << " turns";

    return 0;

}

* A multi-dimensional array is **an array of arrays**.
* Example (2-dimensional):

string letters[2][4] = {  
  { "A", "B", "C", "D" },  
  { "E", "F", "G", "H" }  
};

* To access an element of a multi-dimensional array, specify an index number in each of the array's dimensions.
* The following statement accesses the element's value in the letters array's first row (0) and third column (2).

string letters[2][4] = {  
  { "A", "B", "C", "D" },  
  { "E", "F", "G", "H" }  
};  
  
cout << letters[0][2];  // Outputs "C"

**5.31 Array Of Characters**

#include <iostream>

int main(){

    //Declare array

    char message [5]  {'H','e','l','l','o'};

    //int data[5] {1,2,3,3,3};

    //std::cout << "data : " << data << std::endl;

    //Print out the array through looping

    std::cout << "message : ";

    for( auto c : message){

        std::cout << c ; // Hello

    }

    std::cout << std::endl;

    std::cout << "------------------" << std::endl;

    //Change characters in our array

    message[1] = 'a';

    //Print out the array through looping

    std::cout << "message : ";

    for( auto c : message){

        std::cout << c ; // Hallo

    }

    std::cout << std::endl;

    std::cout << "------------------" << std::endl;

    // Will probably print garbage after your char array

    std::cout << "message : " << message << std::endl; // Hallooo■²┐J☻

    std::cout << "------------------" << std::endl;

    //If a character array is null terminated, it's called as C-String

    char message1 [] {'H','e','l','l','o','\0'};

    std::cout << "message1 : " << message1 << std::endl; // Hello

    std::cout << "sizeof(message1) : " << sizeof(message1) << std::endl; // 6 bytes

    std::cout << "------------------" << std::endl;

    // Implicitly null terminated

    char message2 [6] {'H','e','l','l','o'};

    std::cout << "message2 : " << message2 << std::endl; // Hello

    std::cout << "sizeof(message2) : " << sizeof(message2) << std::endl; // 6 bytes

    std::cout << "------------------" << std::endl;

    char message3 [] {'H','e','l','l','o'}; // This is not a c string ,

                                            //as there is not null character (Don't do it)

    std::cout << "message3 : " << message3 << std::endl; // HelloHello

    std::cout << "sizeof(message3) : " << sizeof(message3) << std::endl; // 5 bytes

    std::cout << "------------------" << std::endl;

    //String literal (Implicitly null terminated)

    char message4 [] {"Hello"};

    std::cout << "message4 : " << message4 << std::endl; // Hello

    std::cout << "sizeof(message4) : " << sizeof(message4) << std::endl;   // 6 bytes

    std::cout << "------------------" << std::endl;

    //Can't safely print out arrays other than those of characters

    int numbers [] {1,2,3,4,5};

    std::cout << "numbers :  " << numbers << std::endl;

    std::cout << "------------------" << std::endl;

    return 0;

}

* Array of characters **can be printed out directly**.
* If a character array is null terminated, it's called as C-String. A null character ***\0*** is appended to the end of the string to identify the end of the array. Otherwise, garbage may store be printed. For example,

***char message1 [] {'H','e','l','l','o','\0'};)***

* **String** array can be **directly print out** by ***cout***, but not for other type (int).
* The strings belows are all C-String.

***char message2 [6] {'H','e','l','l','o'};***

***char message4 [] {"Hello"};***

***char message5 [] {"Hello!" };***

* The strings below is not C-String (Don’t do it).

***char message3 [] {'H','e','l','l','o'};***

**5.32 Declaring and Using Pointers**

#include <iostream>

int main(){

    // Declare and initialize pointer

    int \*p\_number{}; // WIll initialize with nullptr

    double \*p\_fractional\_number{};

    // Explicitly initialize with nullptr

    int \*p\_number1{nullptr};

    int \*p\_fractional\_number1{nullptr};

    //Pointers to different variables are  of the same size

    std::cout << "sizeof(int) : " << sizeof(int) << std::endl; // 4 bytes

    std::cout << "sizeof(double) : " << sizeof(double) << std::endl; // 8 bytes

    std::cout << "sizeof(double\*) : " << sizeof(double\*) << std::endl; // 8 bytes

    std::cout << "sizeof(int\*) : " << sizeof(int\*) << std::endl; // 8 bytes

    std::cout << "sizeof(p\_number1) : " << sizeof(p\_number1) << std::endl; // 8 bytes

    std::cout << "sizeof(p\_fractional\_number1) : " << sizeof(p\_fractional\_number1) << std::endl; // 8 bytes

    std::cout << "----------------------------------" << std::endl;

    //It doesn't matter if you put the \* close to data type or to varible name

    int\*  p\_number2{nullptr};

    int \* p\_number3{nullptr};

    int  \*p\_number4{nullptr};

    int \*p\_number5{}, other\_int\_var{};

    int \*p\_number6{}, other\_int\_var6{}; // Confusing as you wonder if other\_int\_var6

                                        // is also a pointer to int. It is not

                                        // The structure is exactly the same for the

                                        // previous statement

    std::cout << "sizeof(p\_number5) : " << sizeof(p\_number5) << std::endl; // 8 bytes (Int\*)

    std::cout << "sizeof(other\_int\_var) : " << sizeof(other\_int\_var) << std::endl; // 4 bytes (Int)

    std::cout << "sizeof(p\_number6) : " << sizeof(p\_number6) << std::endl; // 8 bytes (Int\*)

    std::cout << "sizeof(other\_int\_var6) : " << sizeof(other\_int\_var6) << std::endl; // 4 bytes (Int)

    std::cout << "----------------------------------" << std::endl;

    //It is better to separate these declarations on different lines though

    int \*p\_number7{};

    int other\_int\_var7{}; // No room for confusion this time

    // Initializing pointers and assigning them data

    // We know that pointers store addresses of variables

    int int\_var{43};

    int \*p\_int{&int\_var}; // The address of operator (&)

    std::cout << "Int var: " << int\_var << std::endl;

    std::cout << "p\_int (Address in memory): " << p\_int << std::endl;

    std::cout << "----------------------------------" << std::endl;

    //You can also change the address stored in a pointer any time

    int int\_var1 {65};

    p\_int = &int\_var1; // Assign a different address to the pointer

    std::cout << "p\_int (with different address)  : " << p\_int << std::endl;

    std::cout << "----------------------------------" << std::endl;

    //Can't cross assign between pointers of different types

    int \*p\_int1{nullptr};

    double double\_var{33};

    //p\_int = &double\_var; // Compiler error

    //Dereferencing a pointer (Read the value stored at the address)

    int\* p\_int2 {nullptr};

    int int\_data {56};

    p\_int2 = &int\_data;

    std::cout << "value : " << \*p\_int2 << std::endl; // Dereferencing a pointer

    return 0;

}

* Pointer is a special kind of variable that stores addresses of other variables. (e.g., ***int \*p\_var {*&*var};***)

A screen shot of a computer

Description automatically generated

* A reference variable is a "reference" to an existing variable, and it is created with the **&** operator(e.g., **&*var***), in the following example, ***food*** and ***meal*** are the same, while **&*meal*** is the memory address.

string food = "Pizza";  
string &meal = food;  
  
cout << food << "\n";  // Outputs Pizza  
cout << meal << "\n";  // Outputs Pizza

cout << &food; // Outputs 0x6dfed4

* A pointer can only store an address of its same type variable.
* Pointer Assignment Syntax:

string food = "Pizza";  // A food variable of type string  
**string\* ptr = &food;**    // A pointer variable, with the name ptr, that stores the address of food

* Pointer Deferencing Syntax:

// Dereference: Output the value of food with the pointer (Pizza)  
**cout << \*ptr << "\n";**

* A pointer can be initialized by ***nullptr***.
* All pointer variables have the **same size**.
* The position of ***\**** does not matter.
* Note that the ***\**** sign can be confusing here, as it does **two different things** in our code:
  + When used in **declaration** (***string\* ptr***), it creates a **pointer variable**.
  + When **not used in declaration**, it act as a **dereference** operator.
* The original variable can be changed by changing the pointer’s value. The following example changes the value of ***food*** to “Hamburger” by changing the pointer ***ptr***.

string food = "Pizza";  
string\* ptr = &food;  
  
// Output the value of food (Pizza)  
cout << food << "\n";  
  
// Output the memory address of food (0x6dfed4)  
cout << &food << "\n";  
  
// Access the memory address of food and output its value (Pizza)  
cout << \*ptr << "\n"; // Pizza  
  
// Change the value of the pointer  
\*ptr = "Hamburger";  
  
// Output the new value of the pointer (Hamburger)  
cout << \*ptr << "\n"; // Hamburger  
  
// Output the new value of the food variable (Hamburger)  
cout << food << "\n"; // Hamburger

**5.33 C++ Structures (struct)**

* Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.
* Unlike an array, **a structure can contain many different data types** (int, string, bool, etc.).
* Syntax (***Initialization***):

struct {             // Structure declaration  
  int myNum;         // Member (int variable)  
  string myString;   // Member (string variable)  
} myStructure;       // Structure variable

* To **access members** of a structure, use the dot syntax (***.***):

// Create a structure variable called myStructure  
struct {  
  int myNum;  
  string myString;  
} myStructure;  
  
// Assign values to members of myStructure  
myStructure.myNum = 1;  
myStructure.myString = "Hello World!";  
  
// Print members of myStructure  
cout << myStructure.myNum << "\n";  
cout << myStructure.myString << "\n";

* You can use a comma (***,***) to use one structure in many variables:

struct {  
  int myNum;  
  string myString;  
} myStruct1, myStruct2, myStruct3; // Multiple structure variables separated with commas

* By giving a name to the structure, you can treat it as a data type. This means **you can create variables with this structure anywhere** in the program at any time. E.g.,

struct myDataType { // This structure is named "myDataType"  
  int myNum;  
  string myString;  
};

myDataType myVar;

**References**

* [**C++ Programming Course - Beginner to Advanced**](https://www.google.com/url?q=https://www.youtube.com/watch?v%3D8jLOx1hD3_o%26ab_channel%3DfreeCodeCamp.org&sa=D&source=calendar&usd=2&usg=AOvVaw32IHq1-uzixQnuCtxAm8fA)
* [**W3schools C++ Tutorial**](https://www.w3schools.com/cpp/cpp_intro.asp)