程式設計(C++) – 完整課程

Source: [https://www.youtube.com/playlist?list=PLMHSr8fseBzUvwjKtR6pX0Vv9Q9H-V3lY](%20https:/www.youtube.com/playlist?list=PLMHSr8fseBzUvwjKtR6pX0Vv9Q9H-V3lY)

Ch1 Intro

\*Variables and data types

- Each variable must be specified a data type:

It tells the system how to allocate memory spaces.

It tells the system how to interpret those 0s and 1s stored there.

- Four attributes of a (typical) variable: type, name, value, address.

\*Basic data types

- The number of bytes is compiler-dependent.

Ch2 Selection and repetition

\*Preprocessor and namespaces

- We may define our own variables and functions into self-defined header files and include them by ourselves. Ex. **#include “C:\myHeader.h”;**

- The scope resolution operator ( :: ). Ex. std::cout << “Hello World! \n”;

\*If-else

- The ternary(三元) if operator ? : . Ex. (condition) ? (operation A) : (operation B);

Ch3 Digital system

Ch4 Variables and arrays

\*Data types, literals, and variables

- Literals: items whose contents are fixed.

- Variables: items whose values may change.

\*Int

- The C++ standard only requires a compiler to ensure that:

- The space for a **long** variable ≥ the space for an **int** one.

- The space for an **int** variable ≥ the space for a **short** one.

\*Limits of **int**

- The limits of C++ basic data types are stored in **<climits>**.

\*Char

- Using single quotation marks to make your **char** literal. Ex. **char c = ‘c’ ;**

\*Float and double

- The function **setprecision()** in **<iomanip>** is capable of setting the precision of a **float** number.

- Precision can be a big issue:

- Remedy: “imprecise” comparisons. Ex. **abs(f \* f – i) > 0.0001** -> Set a range that is acceptable. (Further information: Search Numerical Methods)

\*Casting

- Changing the type of a variable or literal is called casting.

- There are two types of casting:

- Implicit casting: from a small type to a large type. -> No loss.

- Explicit casting: from a large type to a small type. -> Need proper processing.

Ex. **int a = static\_cast<int>(5.6);**

- Why bother casting?

- To make sure that, at the run time, the program runs as we expect.

- To notify other programmers (or the future ourselves).

- There are four different explicit casting operators: **static\_cast**, **dynamic\_cast**, **reindivter\_cast**, and **const\_cast**.

Ch5 Function

\*External variables (Should be avoided)

- If a program wants to access a variable defined in another program, it can declare the variable with the keyword **extern**.

**- extern int a**;

- **a** must has been defined in another program.

- These programs must run together.

\*Static variables

- A local variable will be recycled (its memory space will be released) immediately when it is “dead”.

- A global variable will not be recycled until the end of a program.

- A static variable, declared inside a block, also will not be recycled until the program terminates.

\*Call-by-value mechanism

- The default way of invoking a function is the “call-by-value” (pass-by-value) mechanism. -> Copy the value of the argument into local variable.

\*Function overloading

- Default arguments

- In general, we may assign default values for some parameters in a function.

Ex. **double circleArea (double, double = 3.14) ;**

**double circleArea (double radius, double pi) ;**

If only one argument is delivered to the function, the compiler will use the

upper case.

\*Inline function

- When the compiler finds an inline function, it will replace the invocation by the function statement.

- The function thus does not exist.

- Statements will be put in the caller and executed directly.

- While this saves some time, it also expands the program size.

Ch6 Algorithms and Recursion

Ch7 Complexity and Graphs

\*Complexity

- Time complexity and space complexity:

- Time: We hope an algorithm takes a short time to complete the task.

- Space: We hope an algorithm uses a small space to complete the task.

\*Big O notation

- Worst-case time complexity

- Avareage-case: the expected number of operations required for a randomly

drawn input. The probability distribution matters.

- Worst-case: the maximum possible number of operations required for a

randomly drawn input.

- There are other measurements (small O, theta, big omega, small omega).

\*Graphs/Networks

- A graph has nodes (vertices) and edges (arcs/links). -> Nodes are locations and arcs are roads.

- Two nodes are adjacent if there is an edge between them. -> Neighbors.

- A node’s degree is its number of neighbors.

- Edges may be directed or undirected.

- For an edges from u to v, we denote as (u,v) if it is directed or [u,v] if it is

undirected.

\*Path

- A path (route) from node s to node t is a set of directed edges (s,v1), (s,v2), …, and (vk,t) such that’s and t are connected.

- s is called the source and t is called the destination.

\*Cycles

- A cycle (equivalent to circuit in some textbooks) is a path whose destination node is the source node.

- A path is a simple path if it is not a cycle.

- A graph is a acyclic graph if it contains no cycle.

\*Weights

- An edge may have a weight.

- A weight may be a distance, a cost per unit item shipped, etc.

- A weighted graph is a graph whose edges are weighted.

- A node may also have a weight.

\*Adjacency Matrices and Adjacency Lists

- Adjacency matrix:

- For a graph with n nodes, we construct an n×n array A.

- If the graph is unweighted, make the array a Boolean array. Let Aij = 1 if thereis

an edge (I,j) (or [I,j] if undirected). Let Aii = 1 for either case.

- If the graph is weighted, make the array an integer/float/double array. Let Aij

be the weight of the edge (I,j) (or [I,j] if undirected). Use a specially chosen value

(-1, ∞, node+1, etc.) to indicate the nonexistence of edges.

- An adjacency matrix is simple and straightforward. However, it is space inefficient if the graph has only few edges. -> Solution is using an adjacency list.

\*Breadth-first Search (BFS) & Depth-first Search (DFS)

Ch8 Pointers

\*Pointers

- A pointer is a variable which stores a memory address.

- An array variable also stores a memory address.

\*Pointer Assignment

- We use the address-of operator & to obtain a variable’s address:

**pointer\_name = &variable\_name**

-Ex. **int a = 5; int\* ptr = &a;**

- When assigning an address, the two types must match.

\*Null Pointers

- A pointer point to nothing should be assigned **nullptr**, **NULL**, or **0**.

\*Reference

- A reference is a variable’s alias.

- Call by reference.

- Call by pointer

- Returning a pointer is returning an address. With the address, we also know the value and index of the variable.

\*Static memory allocation

- Memory allocation to an array can be determined during the compilation time, which is called “static memory allocation”. And we may decide the length of an array “dynamically”, that is, during the run time.

\*Dynamic memory allocation (DMA)

- The operator new allocates a memory space and returns the address.

- Ex. **new int** allocates 4 bytes, and the returned address is not recorded.

- **int\* a = new** int makes **a** store the address of the 4-byte space.

- Dynamically allocated arrays cannot be initialized with a single statement.

- A loop, for example, is needed.

- **int\* array = new int [10]** declares an array of integers.

- **int\*\* array = new int\* [10]** declares an array of integer pointers.

Ch9 C strings

\*Char

- Use one byte (-128 to 127) to store English letters, numbers, symbols, and special characters (e.g, the new character).

- Character literals should be placed in between a pair of single quotation marks.

- The C++ standard library <cctype> contains some useful functions for processing characters.

- Ex. **int islower (int c) ;**

**int isupper (int c) ;**

**int tolower (int c) ;**

**int toupper (int c) ;**

**(以上輸出為數字而非字元)**

\*A special way to input a string

- Character arrays are special, and many operations are overloaded for character arrays in a special way. In particular, **cin >>** and **cout <<**.

- Ex. **char str[10] ;**

**cin >> str ;** -> 可直接輸入字串

- For an array **A**, if we do **cout << A**.

- If **A** is of other types, this will print out it memory address.

- But for a character array, this prints out the whole string.

- Ex. **int values[5] = {0};**

**cout << values ;** //an address

**char str[10];**

**cin >> str ;**

**cout << str ;** // output the string

\*The null character

- When we use **cin >>** to input string, a null character \0 will be appended at the end automatically.

- A C string may be initialized with a double quotation.

- **char s[100] = “abc”;**

- The assignment operator is overloaded for character arrays.

- A null character will also be appended if a C string is initialized in this way.

- **char s[100] = { ‘a’, ‘b’, ‘c’ };** -> No null character appended.

\*String assignment

- Assignments with double quotations are allowed only for initialization.

- Ex. **char s[100];**

**s = “this is a string”;** -> compilation error! **s** stores a memory address.

- One may assign values to a string by assigning values to individual characters.

\*Array boundary

- C++ does not check array boundary!

\*cin >> vs. cin.getline()

- cin splits the input stream into pieces according to white spaces.

- The same thing happens for the newline character and tab.

- To input a string with white spaces, use cin.getline().

- Ex. **char a[100];**

**cin.getline(a, 100);** -> the first parameter a is the container’s name, and the

second one is its capacity.

\*String literals and character pointers

- A character pointer may also be initialized as a string literal.

- Ex. **char\* p = “12345”;**

**cout << p + 2 << “\n”;** //345

- That space is read-only and **p** stores the address of that space.

\*Passing a string to a function

- To pass a string to a function, let the parameter type to be a character pointer or a character array. Then pass an address to it.

\*Main function argument

- In fact, we may pass arguments to the main function.

- Ex. **int main (int argc, char\* argc[]) {…}** -> argc: number of strings/ argc[]: storing the pointers

\*A dynamic character arrays as a C string

- A dynamic character array (pointed by a pointer) can store a C string.

- Ex. **char\* p = new char[100];**

**cin >> p; cout << p; delete [] p;**

\*C String processing functions

- The C++ standard library **<cstring>** contains many useful pointer-based string processing functions.

- Query and searching **strlen, strchr, strstr.**

- Comparison: **strcmp, strncmp**.

- Concatenation: **strcat, strncat**.

- Copying: **strcpy, strncpy**.

- Splitting: **strtok.**

- The C++ standard library **<cstdlib>** contains some more.

- String-number conversion: **atoi, atof, itoa**.

\*String length query

- The function **strlen** returns the string length of a given C string.

- Ex. **unsigned int strlen (const char\* str);**

\*Searching in a string

- To find the location of a character in a string (or conclude that it does not exist in the string), use **strchr**.

- Ex. **char\* strchr (char\* str, int character);**

- It returns the address of the first occurrence of the character.

- If the character does not exist, it returns **nullptr**.

\*Searching for the next occurrence

- Two advanced techniques:

- The returned address may be used to modify the given string.

- The returned address may be used as the string location of a “new string” to

search for the next occurrence of the character.

\*Searching for a substring

- If we want to search for a substring, we use **strstr**.

- Ex. **char\* strstr (char\* str1, const char\* str2);**

- This returns the address of the first occurrence of **str2** in **str1**.

\*String-number conversion

- In **<cstdlib>**, two functions converts a character array into a number.

- Ex. **int atoi (const char\* str) ;**

**double atof (const char\* str) ;**

- For atoi, **str** should contain only digits (but the first character can be ‘ **–** ‘).

- For atof, **str** may contain at most one ‘ **.** ‘.

- A function converts a number into a character array.

- Ex. **char\* itoa (int value, char\* str, int base) ;** value是待轉換數字,str是分配給轉換後儲存的空間,base是想轉換的進位制

\*String comparisons

- String may be compared alphabetically.

- Ex. **int strcmp (const char\* str1, const char\* str2) ;**

**Int strncmp (const char\*str1, const char\* str2, unsigned int num) ;**

- They return 0 if the two strings are identical, a negative number if **str1** is in

front of **str2**, and a positive number if **str2** is in front of **str1**.

- **strcmp** compares the entire strings. **strncmp** does up to **num** characters.

\*String copying

- Ex. **char\* strcpy (char\* dest, const char\* source) ;**

- It copies the string at **source** into the array at **dest**, including the terminating null character in **source**. It returns **dest**.

\*String concatenation

- A similar task is to concatenate two strings. We may use **strcat** to do this.

- Ex. **char\* strcat (char\* dest, const char\* source) ;**

\*Splitting a string into substrings

- We often want to split a string into substrings based on some characters.

- These characters are called delimiters. These substrings are called token.

- A way is to use strtok.

- **char\* strtok (char\* str, const char\* delimiters );**

CH 10 Self Defined Data Types in C

- In C, there are many ways if creating self-defined data types.

- **typedef**, **struct**, **union**, and **enum**.

\*Struct

- Ex. **struct Point**

**{**

**int x;**

**int y;**

**};** -> Here **Point** becomes a new type.

- With the new data type, the program can now be written in this way.

- Declare variables with the self-defined type name.

- Assign values to both attributes by grouping values by curly brackets.

- Access attributes through the dot operator.

\*Struct definition

- Ex. **struct struct\_name**

**{**

**type1 field1;**

**type2 field2;**

**type3 field3;**

**}**

\*struct variable declaration

- To declare a variable defined as a structure, use

- **struct\_name variable\_name;**

\*typedef

- typedef is the abbreviation of “type definition”.

- It allows to create a new type from another data type.

- Ex. **typedef old\_type new\_type;**

- To avoid modifying lots of declaration statements and to help readers understand easily.

- Put them globally unless you really use them locally.

\*A member-function implementation

- We may redefine Point to include a member function.

- Ex. **struct Point**

**{**

**int x;**

**int y;**

**double distori()**

**{**

**return sqrt( pow(x,2) + pow(y,2) );**

**}**

**};** -> To use the function, simply type **Point.distori();**

- Another way to write a member-function implementation

- Ex. **struct Point**

**{**

**int x;**

**int y;**

**double distori();**

**};**

**Double Point::distori() //scope resolution is required**

**{**

**return sqrt( pow(x,2) + pow(y,2) );**

**}**

\*Random numbers

- In C++, randomization can be done with two functions, **srand()** and **rand()**.

\*rand()

- **int rand();**

- It “randomly” returns an integer between 0 and **RAND\_MAX** (in **<cstdlib>**, typically 32767).

- rand() returns a “pseudo-random” integer. They just look like random numbers. But they are not really random. There is a formula to produce each number.

\*srand()

- We use srand to determine the seed.

- Ex. **void srand(unsigned int);**

- We must give **srand()** different arguments. In many cases, we use **time(nullptr)** to be the argument of **srand()**.

- The function **time(0)**, defined in **<ctime>**, returns the number of seconds that have past since 0:0:0, Jan, 1st, 1970.

- Ex. **time\_t time(time\_t\* timer);**

\*Random numbers in a range

- If you want to produce random numbers in a specific range, use %.

-Ex. **rn = ((rand() % 10)) + 100;** %10 -> In the range from 0 to 9, then plus the desired number.

- More powerful random number generators are provided in **<random>**.

* Supplementary Information

1. Vector

Reference:<https://mropengate.blogspot.com/2015/07/cc-vector-stl.html>

Vector 是 C++ 標準程式庫中的一個 class，可視為會自動擴展容量的陣列，是C++標準程式庫中的眾多容器（container）之一，以循序 (Sequential) 的方式維護變數集合，使用前預先 #include <vector> 即可。

2. Hashmap