2026 / 7 / 2 (Thur.)

String、Array、File、Function

namespace & using

String

Char vs. String

- Unlike C-style character arrays (char[]), std::string automatically manages its memory and size.
- Actually, C only has char arrays.
- Q: how many byte should be allocated for a string ?
- Char is not easy to operate
  - Strcat(a,b) vs. a = a + b

Creating String Objects

- string s; //Empty string
- string s1 = "Hello World"; //Initialization
- string s2("Hello World"); //Constructor

- string part(s1, 6, 10); //World
- string letters(7,a); //aaaaaaa
- string letters2(letters); //copy
- string names[] = {"Alice", "Bob"}; //string array

Basic operation

- Operator +
- Operator !=
- Operator ==
- Operator <
- Operator <=
- Operator <<
- Operator >
- Operator >=
- Operator >>

Operator +

- Return value

Input a concatenated string of characters.

```
string s1= 'Hello';
string s2 = 'Word';
string result = s1 + "  " + s2;
cout << result << endl;   // output:  Hello Word
```

Common Member Functions

- length() //Returns an integer
- size() //Same as length()
- empty() //Checks whether the string is empty
- clear() //Removes all characters
- substr() //Returns part of a string
- find() //Searches for a substring

Common Member Functions

string s = "Programming";

- cout << s.length(); //11
- cout << s.substr(0,4); //Prog
- cout << s.find("gram"); //3

Input a string : cin>> vs. getline()

- cin reads input until it encounters the first whitespace (space, tab, or newline).

- getline() reads an entire line of text, including spaces, until the Enter key (newline character \n) is pressed.

Ex.
Input:
John Smith

Output :
John

```
string name;
getline(cin, name);
```

Input:
John Smith
Output :
John Smith

Input a string : cin>> vs. getline()

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

int main() {
string s1;
string s2;
char s3[20];

cin >> s1;

getline(cin, s2);

cin.getline(s3, 20);
}
```

Accessing strings

- Access a character in a string

```
string sentence = "An apple a day";
for(size_t i = 0 ; i < sentence. length() ; i++ ){
        if(' ' == sentence[i])
        sentence[i] = '*';
}
cout << sentence;
```

- Use the at() member function

```
string sentence = "Too many cooks spil the broth.";
for(size_t i = 0 ; i < sentence.length() ; i++ ){
        if(' ' == sentence.at(i))
        sentence[i] = '*';
}
```

Accessing a substring in a string

- Extract a part of an exising string object as a new string object.
  - string sentence("An apple a day");
  - string w = sentence.substr(3, 5); //Extracts "apple"

Modifying Strings (1) : append()

- Add one or more characters to the end of a string.

```
string phrase("The higher");
string word("fewer");
phrase.append(1, ' ');    // Append one space
phrase.append("the "); // Append a string literal
phrase.append(word);  // Append a string object
phrase.append(2, '!' );   // Append two exclamation marks
```

- When you call append(), the function returns a reference to the object, so the "function chaining" skill encourages you to write the above calls in a single statement:
  - phrase.append(1, ' ').append("the ").append(word).append(2, '!');

Modifying Strings (2) : insert(), swap()

- Insert a string in the interior of another string:

```
string word("ABCDEF");
string test("===");
word.insert(3, test);   // word will become "ABC===DEF“
```

- Swap the contents of two string objects:

```
string phrase("The more the merrier.");
string query("Any");
query.swap(phrase);
```

Modifying Strings (3) : replace()

- Replace part of a string object

```
string text("ABCDEF");
text.replace(2, 3, "->");   // word will become "AB->F"
```

- Replacement with a null string will essentially delete that part.

```
string text = "AB->F";
text.replace(2, 2, ""); // word will become "ABF"
```

Comparing Strings

- Operator overloading has been implemented for
  - == != < <= > >=
- When two corresponding characters are compared, their ASCII codes determine which one is less than the other.
  - “USA” < “unique " //true
- If no character pairs are found to be different, the string with fewer characters is less.
  - "book" < "books" //true

Search Strings

- Four versions of the find() function:
  - string phrase("Alfa Bravo");
  - string s("Bravo");
  - cout << phrase.find("Alfa") << endl; // Outputs 0 (starting position)
  - cout << phrase.find(s) << endl; // Outputs 5
  - cout << phrase.find("Charlie") << endl; // Outputs string::npos = 4294967295 on MS VC++
- The function returns the value string::npos if the item was not found.
  - The value of string::npos may vary with different C++ compilers, so you should always use string::npos and not the explicit value.

Search Strings (2)

- Searching from a specified position:
  - string phrase("ABCDEABCDEABCDE");
  - cout << phrase.find("A"); // Outputs 0
  - cout << phrase.find("A", 3); // Outputs 5
  - cout << phrase.find("A", 11); // string::npos = 4294967295

Conversion to C-Style Pointer-Based char* Strings

- For functions which expect legacy C-style char* parameters, this is convenient.
- Copy from string to char*
  - string city("Kings");
  - char ptr1[6] = { 0 };
  - city.copy(ptr1, city.length());
- Get the pointer char
  - const char* ptr2 = city.c_str();
  - city = "Landing"; // Modify the contents
  - cout << city << endl;
  - cout << ptr1 << endl; // Are ptr1 and ptr2
  - cout << ptr2 << endl; // affected?

Handling Chinese Characters (VS2022)

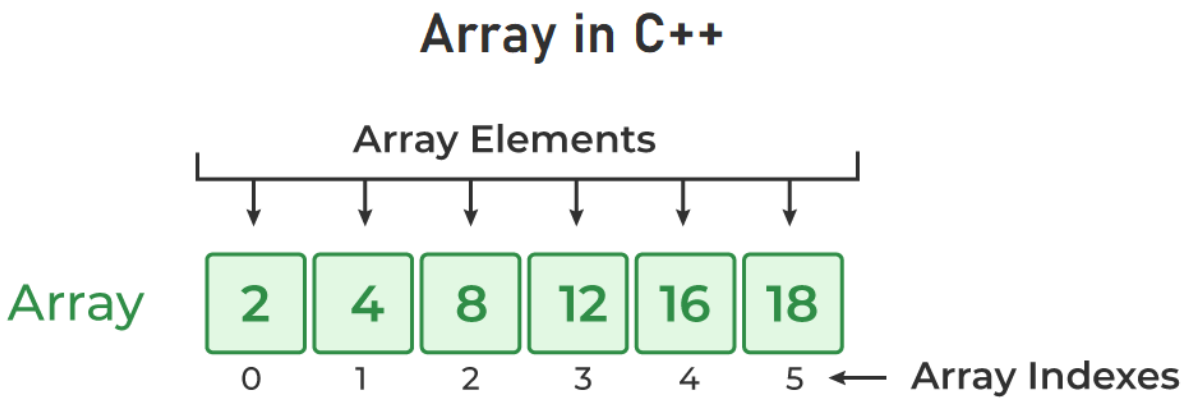
```
#pragma execution_character_set("utf-8")
#include <iostream>
#include <string>
#include <Windows.h>

int main() {
SetConsoleOutputCP(65001); //設定控制台輸出 UTF - 8
SetConsoleCP(65001); //設定控制台輸入為 UTF-8 (建議也加上)
std::string myString = "暨南大學";
std::string str2 = "中文";
std::cout << myString << myString.length() << '\n';
std::cout << str2.substr(3, 3) << myString.substr(6, 3) << '\n';
return 0;
}
```

Array

Initialize array

- An array is a collection of elements with the same data type, stored in contiguous memory locations.
  - int num[6] = {2, 4, 8, 12, 16,18};
  - int num[ ] = {2, 4, 8, 12, 16,18};

  //complier automatically detects the number of elements and sets the size to 10



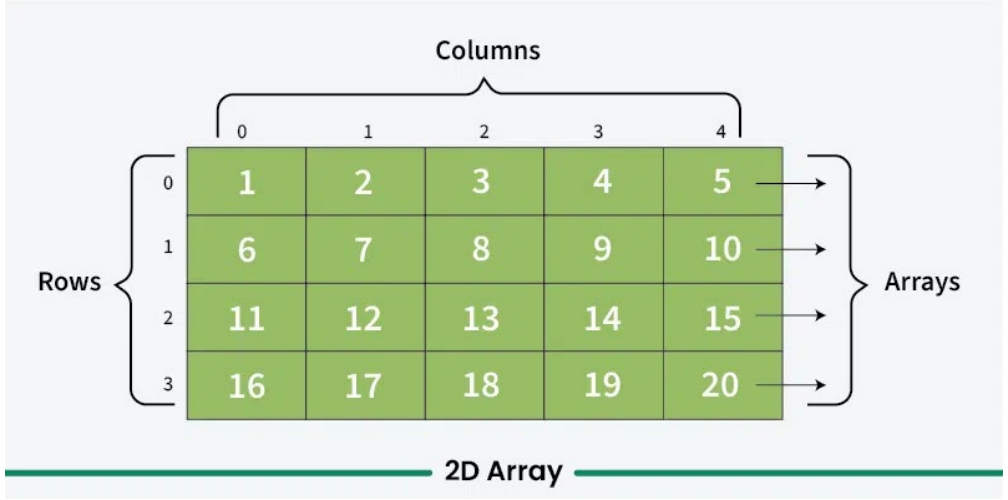
Basic Operations

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

int main() {
int score[5] = { 80, 90, 75, 88, 95 };
score[0] = 100;

std::cout << score[2];

for (int i = 0; i < 5; i++)
{
std::cout << score[i];
}
return 0;
}
```

Two-dimensional Arrays

- int a[3][4] = { {0, 1, 2, 3} ,
  {4, 5, 6, 7} ,
  {8, 9, 10, 11}
  };

- int a[3][4] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };



C++ Streams

1. I/O Stream
2. File Stream
3. String Stream

Compare cout and printf

- cout is convenient. You don't need to worry about those conversion specifiers:
  - %s for string
  - %d for decimal integer
  - %x for hexadecimal
  - %f for floating point
- For your defined data type (e.g., CRational), cout knows how to print it out, as long as you have defined operator<<() for CRational.
  - For each printf(), you must work out the details to print out its numerator and denominator.

Q: printf is good at formatting

- Sometimes I still want to specify the format:
  - %4d
    - width of an integer
    - padding ‘0’ if the integer is shorter
  - %5.2f
    - number of digits after the decimal number
  - %-10s
    - left-justify (default is right-justify, including strings)

File Stream

ifstream
ofstream

Q: fprintf() can output to a file

```
#include <stdio.h>
using namespace std;
int main() {
    FILE* fp = fopen("a.txt", "w");
    for (int i = 1;i <= 5;i++) {
        for (int j = 0;j < i;++j) {
            fprintf(fp, "*");
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    return 0;
}
```

A: C++ File Stream Does the Same

- With the same syntax as I/O Stream.

```
#include <iostream>
#include <fstream>
int main() {
    std::ofstream fout("b.txt");
    for (int i = 1; i <= 5; ++i) {
        for (int j = 0; j < i; ++j)
            fout << '*';
        fout << '\n';
    }
    fout.close();
    return 0;
}
```

Q: If I want to extend the file

- A: Open the file using the Append mode.

```
#include <iostream>
#include <fstream>
int main() {
    std::ofstream fout("b.txt", std::ios::app);
    for (int i = 1; i <= 5; ++i) {
        for (int j = 0; j < i; ++j)
            fout << '*';
        fout << '\n';
    }
    fout.close();
    return 0;
}
```

Q: How do I read input from a file?

```
#include <iostream>
#include <fstream>
int main() {
    std::ifstream f("c.txt");
    int n;
    while (f >> n)
        std::cout << n << std::endl;
    f.close();
    return 0;
}
a
```

Q: If I did not close the file?

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main() {
remove("a.txt");    // Remove the file
FILE* fp = fopen("a.txt", "a");
char msg[] = "First\n"; fwrite(msg, sizeof(msg), 1, fp);
// fclose(fp);      // If you do not close the file properly ...

fp = fopen("a.txt", "a");
char msg2[] = "Second\n";
fwrite(msg2, sizeof(msg2), 1, fp);
fclose(fp);
return 0;
 }
```

String Stream

String Streams

- A string stream allows a string to be treated like an input/output stream.
- Instead of reading data from the keyboard (cin) or writing data to the screen (cout), a string stream reads from or writes to a string.

Class	Purpose
`istringstream`	Read data from a string (input string stream)
`ostringstream`	Write data to a string (output string stream)
`stringstream`	Both read and write (input & output string stream)

Using istringstream

- istringstream read data from a string as if it were input from cin.

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

using namespace std;

int main() {
string data("123 45.6 hello");
istringstream iss(data);  //create a string
int a;
double b;
string s;

iss >> a >> b >> s; //extract data
cout << a << " " << b << " " << s << endl;
return 0;
}
```

Output:
123 45.6 hello

Using ostringstream

- ostringstream : write data to a string

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdio.h>

using std::cout;
using std::string;
using std::ostringstream;

int main()
{
    int a = 10, b = 20, c = a + b;

    char buffer[80];
    sprintf(buffer, "%d + %d = %d", a, b, c);

    ostringstream oss;
    oss << a << " + " << b << " = " << c;

    string result = oss.str();

    cout << buffer << '\n'
        << result << '\n';

    return 0;
}
```

Output:
The answer is 42, pi is 3.14

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

int main()
{
    string line;
    getline(cin, line); //Reads an entire line of input
    stringstream ss(line); //Convert the string into a stream.
    int num, sum = 0;

    while (ss >> num) //Extract one integer at a time
        sum += num;

    cout << "Sum = " << sum << endl;
    return 0;
}
```

Function

Main()

```
int main() {
        cout << "Hello" << '\n';
        return 0;
}
```

1. A function is a group of code that can do some work and (usually) return a value.
2. A C++ programs must have a function called main().
3. The execution starts from main().

Code between two curly braces is called a block and is usually indented.

Pass by Value, Reference, and Pointer

- Pass by Reference

```
void addString(string& str) {
    str += "hello world";
}
```

- Pass by Pointer

```
void addString(string* str) {
    *str += "hello world";
}
```

- Pass by Value

```
void addString(string str) {
    str += "hello world";
}
```

```
int main() {
    string txt = "Alice,";
    addString(txt);
    cout << txt << '\n';
}
```

namespace & using

namespace

- When multiple programmers work in a project, it is quite often that they choose the same function names or global variable names in their code:

- undergraduate.h

```
#include <math.h>
const int THRESHOLD_TO_PASS = 60;
int adjust(int n) {
        return (int) sqrt(n) * 10;
}
```

- graduate.h

```
const int THRESHOLD_TO_PASS = 70;
int adjust(int n) {
        return (int)n * 0.7 + 70;
}
```

```
#define N      10
#include "undergraduate.h"
#include "graduate.h"
int main() {
        int grade[] = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81};
        for (int i=0; i<N; ++i)
                grade[i] = adjust(grade[i]);
        return 0;
}
```

namespace

```
#include <iostream>
#define N       10

namespace Undergraduate
{ #include "undergraduate.h" }
namespace Graduate
{ #include "graduate.h" }

int main() {
int grade[] = { 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 };
for (int i = 0; i < N; ++i)
std::cout << Undergraduate::adjust(grade[i]) << std::endl;
std::cout << Graduate::THRESHOLD_TO_PASS << std::endl;
return 0;
}
```

Employing a using Directive

- This allows all the names in a namespace to be used without the namespace-name as an explicit qualifie

```
// Hello NCNU
// A first C++ program

#include <iostream>
int main() {
    std::cout << "Hello NCNU!" <<
std::endl;
    return 0;
}
```

```
// Hello NCNU 2.0
// Demonstrates a using directive

#include <iostream>
using  namespace std;
int main() {
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

Employing using Declarations

- You may also write two using declarations to precisely specify which elements from the std namespace you want to use in your program.

```
// Hello NCNU 2.0
// Demonstrates a using directive

#include <iostream>
using  namespace std;
int main() {
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

```
// Hello NCNU
// A first C++ program

#include <iostream>
using std::cout;
using std::endl;
int main() {
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

Exercise #1

- Input : Hello NCNU C++
- Output : Hello_NCNU_C++
- Concepts
  - getline()
  - length()
  - [] / at()

Exercise #2

- The input is a single line containing several integers separated by spaces.
- Use **getline()** and **stringstream** to calculate their sum.

- Input : 10 20 30 40
- Output : 100
- Concepts
  - getline()
  - Stringstream
  - while(ss >> x)