C++

程式語言（二）

Introduction to Programming (II)

Course Introduction

Joseph Chuang-Chieh Lin

Dept. CSE, NTOU

Platform/IDE/Resources

- Dev-C++

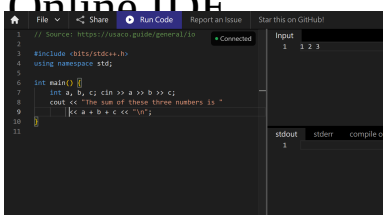
https://www.pngegg.com/en/search?q=Dev-C

- Codeblocks

https://icons8.com/icons/set/code-blocks

- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)



- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

Textbooks (We focusing on C++11)

- ***Learn C++ Programming by Refactoring* (由重構學習 C++ 程式設計). Pang-Feng Liu (劉邦鋒). NTU Press. 2023.**
- ***C++ Primer. 5th Edition.* Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. 2019.**
- *Effective C++*. Scott Meyers. O'Reilly. 2016.
- *Thinking in C++. Vol. 1: Introducing to Standard C++*. 2nd Edition. Bruce Eckel. Prentice Hall PTR. 2000.

Useful Resources

- Tutorialspoint
  - https://www.tutorialspoint.com/cplusplus/index.htm
  - Online C++ Compiler
- Programiz
  - https://www.programiz.com/cpp-programming

Course TAs

- 薛凱駿
  - Email address: 11357063@o365.ntou.edu.tw
- 楊承霖
  - Email address: 11357072@o365.ntou.edu.tw
- 吳奇軒
  - Email address: 11357052@o365.ntou.edu.tw

Grading Policy

- Assignment: 20%
- Midterm Exam: 40%
- Final Exam: 40%

Notes on Homework Submission

- Makeup due date:
  - 1 week after the due date.
  - 40% discount.

If possible...

- Bring your laptops or notebook to the classroom if possible.

Introduction to OOP

Object-Oriented Programming Languages

- [Wikipedia] A programming paradigm based on the concept of "objects", which can contain data and code.
  - Data: in the form of fields (i.e., *attributes* or *properties*)
  - Codes: procedures (i.e., *methods*).
- Examples:
  - C++, JAVA, Python, Perl, Lisp.

- Most popular: *Class*-based.
  - Objects are instances of classes.

C++

- Created by Bjarne Stroustrup (a Danish computer scientist; C++ started in 1979 and has become generally available since 1985).
- Originally C with classes and renamed as C++.
- Efficiency of C is maintained.
- Applications in
  - System software
  - Application software
  - Device drivers
  - Embedded software
  - Games
  - High-performance computing server
  - Used to program FPGAs (Field Programmable Gate Arrays)

Why C++?

- Roughly a superset of C.
- Maintainability and Portability.
- International standard(s)
  - “C with Classes” -> C++03 -> C++11 -> C++14 -> C++17 -> C++20.
- General purpose.
- Powerful yet efficient.
- Low-level access to hardware.
- Easy to move to other OOP languages
  - But NOT in other direction.

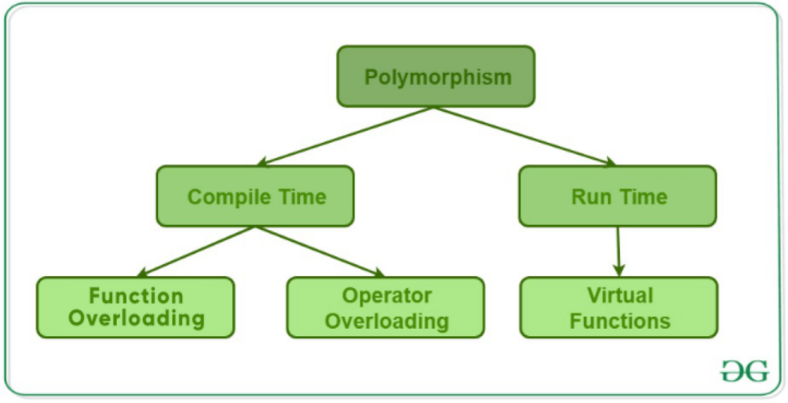
The four basic concepts of OOP

- **Encapsulation (封裝)**
  - **Wrapping up** of data and information under a single unit.
  - Binding together the data and the functions that manipulates them.
  - Emphasize on the "**interface**".
  - Using classes.
  - Protect the data from outside interference and misuse.

- **Abstraction (抽象化)**
  - Displaying only **essential information** and hiding the details.
  - Providing only essential information about the data to the outside world.
  - **Hiding the details and implementation.**
  - Using classes or header files.
  - Reducing complexity and allowing the programmer to focus on interactions at a higher level.

The four basic concepts of OOP

- **Inheritance (繼承)**
  - The ability of **deriving properties and characteristics from another class**.
  - Promoting code **reusability** and establishing a natural **hierarchy** between classes.
- **Polymorphism (多型)**
  - The ability of a message to be displayed **in more than one form.**



Reference:
https://www.geeksforgeeks.org/polymorphism-in-c/?ref=lbp

GNU Compiler Collection (GCC)
C++ Compiler

```
g++ [options] input_file.cpp
```

To object code file:

```
g++ -c file_1.cpp // get file_1.o
g++ -c file_2.cpp // get file_2.o
```

To link object code files and produce a executable file:

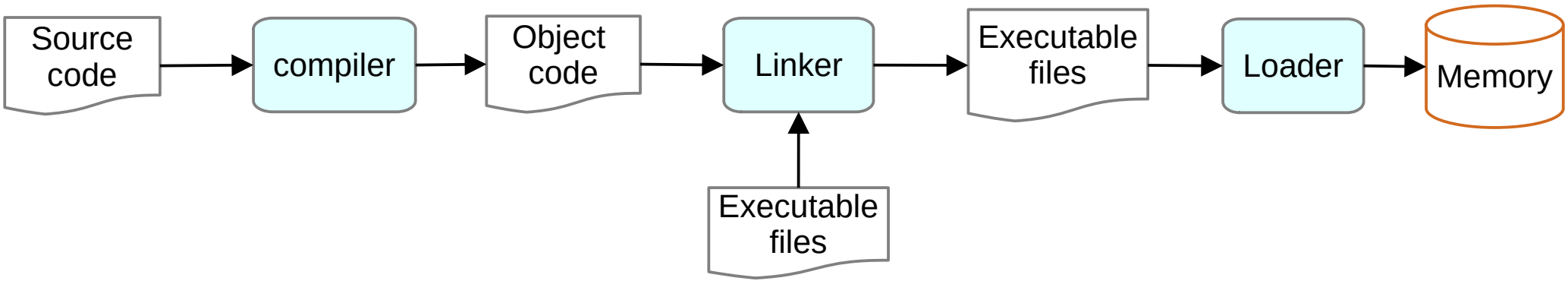
```
g++ -o executable.exe file_1.o file_2.o
```

To directly produce a executable file from a C++ source code:

```
g++ -o executable.exe source.cpp
```

Flowchart

Linker: combines the object files (e.g., generated by the compiler/assembler) and generate executable files.



Loader: load executable files to main memory.

Your first C++ program

```
#include <iostream>

int main() {
    std::cout << "Welcome to C++!\n"; // :: scope resolution operator
    return 0;
}
```

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

int main() {
    cout << "Welcome to C++!\n";
    //cout << "Welcome to C++!" << endl;
    return 0;
}
```

A simple I/O example

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

int main() {
    int n1 = 0, n2 = 0;
    cout << "Enter two numbers: " << endl;
    cin >> n1 >> n2;
    cout << "The sum is " << n1+n2 << endl;

    return 0;
}
```

```
cin >> n1 >> n2; // using cascading; the same as (cin >> n1) >> n2;
```

A typical syntax error (語法錯誤)

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

int main() {
    int n1 = 0, n2 = 0;
    cin >> a >> b >> c;
    cout << a * b / c << endl

    return 0;
}
```

What's wrong with the above example?

`main()` with no return type

```
#include <iostream>

using namespace std;

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

```
#include <iostream>

using namespace std;

void main()
{
    cout << "Hello World";
}
```

`main()` with no return type

```
#include <iostream>

using namespace std;

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

warning: ISO C++ forbids declaration of ‘main’ with no type [-Wreturn-type]

```
#include <iostream>

using namespace std;

void main()
{
    cout << "Hello World";
}
```

Continually getting input numbers

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

int main() {
    int sum = 0, value = 0;
    // continually getting numbers and sum over them until
    // reaching end-of-file
    while (cin >> value)
        sum += value;
    cout << "The sum is " << sum << endl;

    return 0;
}
```

```
while (!cin.eof()) {
    // do something on s
    cin >> s;
}
```

Note:

In Windows OS, use `Ctrl+z` to represent end-of-file.

In Mac OS or Unix/Linux OS, use `Ctrl+d` to represent end-of-file.

String I/O

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

int main()
{
    char str[100];

    cout << "Enter a string: ";
    cin >> str;
    cout << "You entered: " << str << endl;

    return 0;
}
```

String I/O: read a line of text

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

int main()
{
    char str[100];
    //string str;

    cout << "Enter a string: ";
    cin.get(str, 100);//or "cin.getline(str, 100);"
    //or using "getline(cin, str);"
    cout << "You entered: " << str << endl;

    return 0;
}
```

getline

- The prototype of `getline` function

```
istream& getline(istream& is, string & str);
```

`istream`: a class (input stream); so `ostream` is for output string

"&": similar to "*" (pointer); in C++ it's called a **reference** declarator

- **The referenced object will NOT be copied.**

Supplementary

- Difference between

  ```
  cout << "blah blah ..." << endl;
  ```

  and

  ```
  cout << "blah blah ..." << "\n";
  ```

- `cout << "blah blah ..." << endl;`

  is equivalent to

  `cout << "blah blah ..." << "\n"` **and then flush the buffer.**

- Let’s review and renew some features we have learned in C.

Variables

Mostly the same as C language.

Basic C++ variables vs. Modern C++ Variables

```
int age = 25;
double pi = 3.14159;
char rank = 'A';
```

The conventional way

Basic C++ variables vs. Modern C++ Variables

```
int age = 25;
double pi = 3.14159;
char rank = 'A';
```

The conventional way

```
auto age = 25;
auto pi = 3.14159;
auto rank = 'A';
//deduce the type automatically
```

Modern C++

Basic C++ variables vs. Modern C++ Variables

```
int age = 25;
double pi = 3.14159;
char rank = 'A';
```

The conventional way

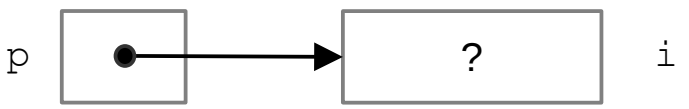
```
auto age {25};
auto pi {3.14159};
auto rank {'A'};
//deduce the type automatically
//using uniform initialization
// using {} from C++11
```

Modern C++

https://onlinegdb.com/ZGFO-8-NQ

The address operator

```
int i, *p;
...
p = &i;
```

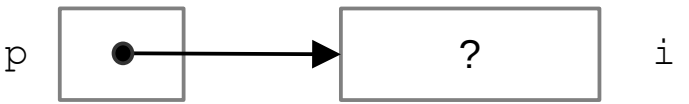



```
cout << p << &i;
//print the address of i
```

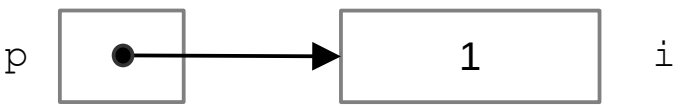
The indirection operator (間接取值)

It can be used to obtain the **value** stored at the memory location **referenced by a pointer variable**.

```
int i, *p;
...
p = &i;
```




```
i = 1;
```




```
cout << i << endl;
cout << *p << endl;
```

What's are the outputs?

```
*p = 2;
```

```
cout << i << endl;
cout << *p << endl;
```

What's are the outputs?

Supplement: void pointer

```
void* p; cout << *p; /* invalid */
 // error: 'void*' is not a pointer-to-object type
```

```
void* p;
int *px = (int*)p; /*forced type-transformation first!*/
cout << *px; /*valid now*/
```

Pointer variables

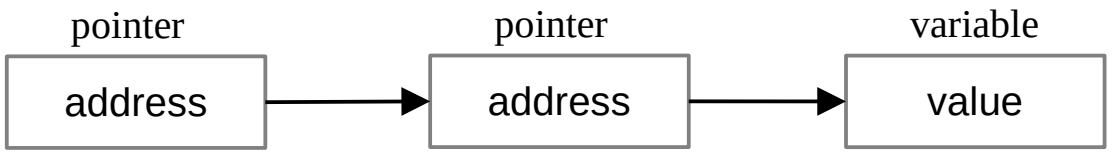
- Pointer variables can appear in declaration along with other variables.

```
int i, j, a[10], b[20], *p, *q;
```

- Every pointer variable points only to objects of a particular type.

```
int *p; /*points only to integers */
double *q; /*points only to doubles */
char *r; /*points only to characters */
```

- A pointer variable can even point to another pointer!



Example

- Try the right-hand side example.
- What are the outputs?

```
int  var;
int  *ptr;
int  **pptr;

var = 3000;

/* take the address of var */
ptr = &var;

/* take the address of ptr using address of operator & */
pptr = &ptr;

/* take the value using pptr */
cout << "Value of var = " << var << endl;
cout << "Value available at *ptr = " << *ptr << endl;
cout << "Value available at **pptr = " << **pptr << endl;
cout << "What's the value of pptr? => " << ptr << endl;
cout << "What's the value of *pptr? => " << *pptr << endl;
```

Example (Revised)

- Try the right-hand side example.
- What are the outputs?

```
auto var {3000};

/* take the address of var */
auto *ptr {&var};

/* take the address of ptr using address of operator & */
auto **pptr {&ptr};
/* take the value using pptr */
cout << "Value of var = " << var << endl;
cout << "Value available at *ptr = " << *ptr << endl;
cout << "Value available at **pptr = " << **pptr << endl;
cout << "What's the value of pptr? => " << ptr << endl;
cout << "What's the value of *pptr? => " << *pptr << endl;
```

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
What's the value of pptr? => 94203732
What's the value of *pptr? => 94203732
```

Null pointer

- `NULL` is defined as `(void*)0`
- Usage:
  - Initialization.
  - Error handling.
    - Dereference a pointer variable only if it's not NULL.
  - Pass a null pointer to a function argument when we don't want to pass any valid memory address.

```
int *p = NULL;
```

Null pointer

```
int *p = NULL;
```

The conventional way
(also in C)

```
int *p = nullptr;
```

Modern C++

Boolean variable

- In C++, we can use the type "`bool`".

```
int a = 5, b = 3;
bool check {a == b};
cout << check << endl;
```

https://onlinegdb.com/FenGBnjhR

Manipulator

- Note: We need to add:

```
#include<iomanip>
```

```
int i = 5
cout << i << endl;
cout << setw(4) << i << endl;
```

```
5
   5
```

dec, oct, hex, and setfill()

```
int i;
cin >> i;
cout << dec << setw(6) << i << endl;
cout << oct << setw(7) << i << endl;
cout << hex << setw(8) << i << endl;

cin >> hex >> i;
cout << setfill('0');
cout << dec << setw(6) << i << endl;
cout << oct << setw(7) << i << endl;
cout << hex << setw(8) << i << endl;
cout << setfill(' ');
cout << left << setw(15) << "hello, world\n";
cout << right << setw(15) << "hello, world\n";
```

```
100
   100
    144
      64
100
000256
0000400
00000100
hello, world
    hello, world
```

Set precision

```
double pi {3.1415926};
double goldRatio {1.618};

cout << setprecision(4) << pi << endl;
cout << setprecision(3) << pi << endl;
cout << setprecision(2) << pi << endl;
cout << setprecision(1) << pi << endl;
cout << setprecision(0) << pi << endl;
cout << setprecision(2) << goldRatio << endl;
cout << setprecision(1) << goldRatio << endl;
cout << setprecision(0) << goldRatio << endl;
```

```
3.142
3.14
3.1
3
3
1.6
2
2
```

Standard integer types (C++11)

- `#include <cstdint>`

```
int8_t  // sizeof(int8_t) = 1
int16_t // sizeof(int16_t) = 2
int32_t // sizeof(int32_t) = 4
int64_t // sizeof(in64_t) = 8
```

- Maximum values:

```
INT8_MAX // 127
INT16_MAX // 32757
INT32_MAX // 2147483647
INT64_MAX // 9223372036854775807
```

```
UINT8_MAX // 255
UINT16_MAX // 65535
UINT32_MAX // 4294967295
UINT64_MAX // 18446744073709551615
```

Array

Initialization for an array

```
int array1[] {1, 2, 3, 4, 5};
int array2[5] { 1, 2, 3 };
int array3[5] {1, 2, 3, 0, 0 }; // the same as array2
```

Processing an array using the for loop

```
#define ARRAYSIZE 5
int a[ARRAYSIZE]{ 1, 2, 3, 4, 5 };

for (int i = 0; i < ARRAYSIZE; i++) {
    cout << a[i];
}
```

- What's the benefit using `#define ARRAYSIZE 5`?

Processing an array using the for loop (Method #2)

```
#include <iostream>
#include <iomanip>

#define ARRAYSIZE 5

using namespace std;

int main() {
    int a[ARRAYSIZE]{ 1, 2, 3, 4, 5 };
    int sum{0};

    for (int v : a) {
        sum += v;
    }
    cout << sum << endl;
    return 0;
}
```

```
15
```

assert

- Note: We need to add:

  ```
  #include<cassert>
  ```

- Usage: check if the condition holds

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

#define N 10

int main() {
    int a[N];
    for (int i=0; i<N; i++)
        a[i] = i;

    int j;
    while (cin >> j) {
        assert( j >= 0 && j < N);
        cout << a[j] << endl;
    }
    return 0;
}
```

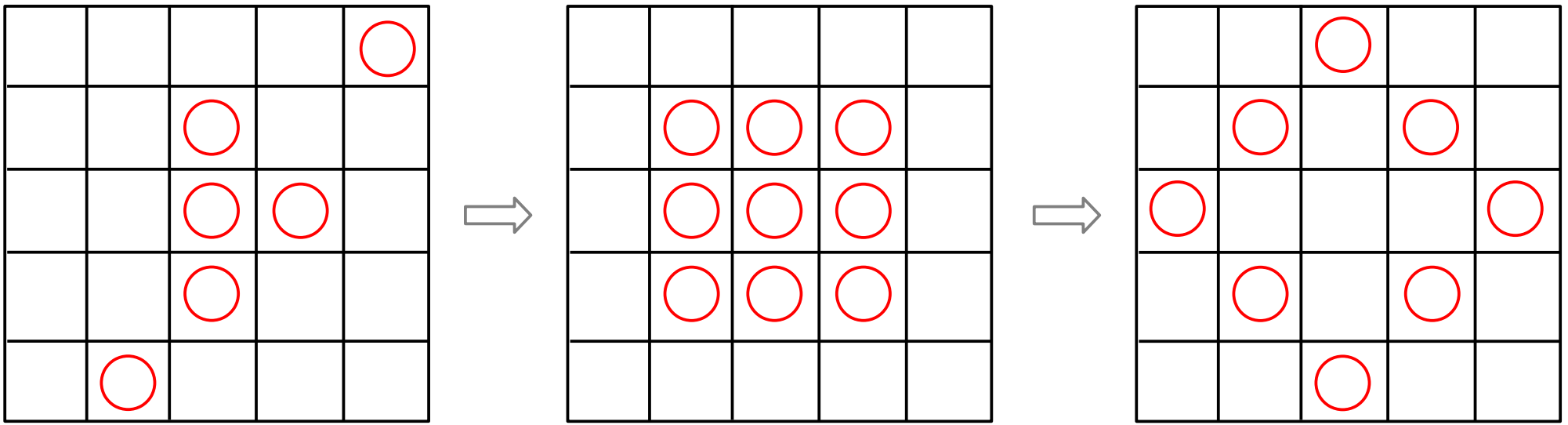
Multi-dimensional array

```
int arr1[10][20][30];

int arr2[3][3] {{1, 2}, {4}};

int arr3[3][3] {{1, 2, 0}, {4, 0, 0}, {0, 0, 0}};

// arr2 and arr3 are the same

int arr4[100][100] {{0}};
//initialize the whole array with zero entries
```

Game of Life (Exercise)

- The game is represented as an *n*-by-*n* 2D array (given by the user). Each element represents a cell which could be “dead” or “alive”. An element has at most 8 neighbors.
- Rule:
  - An alive cell will be still alive in the next stage if it has **two or three alive neighbors** surrounding, otherwise it will be dead in the next stage.
  - A dead cell will become alive in the next stage if it has **exactly three alive neighbors**, otherwise it is still dead.
- Compute the stage after ***k*** iterations.

Illustration



Sample input/output

Sample code by the author:

https://github.com/pangfengliu/Cplusplus-refactor/blob/main/array/life.cc
https://github.com/pangfengliu/Cplusplus-refactor/blob/main/function/life.cc

```
bool life[2][n][n];
int from = 0, to = 1;
```

```
for (int row=0; row<n; row++) // get input
    for (int col=0; col<n; col++) {
        int temp;
        cin >> temp;
        life[from][row][col] = (temp == 1);
}
```

Sample input

```
5 3
0 0 0 0 1
0 0 1 0 0
0 0 1 1 0
0 0 1 0 0
0 1 0 0 0
```

Sample output

```
0 0 1 0 0
0 0 1 0 0
0 1 0 1 0
0 0 1 0 0
0 0 1 0 0
```

Type casting or type conversion

Static cast (靜態轉型)

- Usage:

```
static_cast<type>(expression);
```

- Example:

```
int count {8};
int sum_grade { 120 };
double average = static_cast<double>(sum) / count;
```

In compile time

Comparing with type conversion in C

https://onlinegdb.com/5cODWYI71

```
char *c1;
int *p1 = static_cast<int *>(&c1);
*p1 = 5;
cout << *p1 << endl;
```

```
char c2[2] { 15, 16 };
int *p2 = (int *)(c2);
*p2 = 5;
cout << *p2 << endl;
```

Integer to string

```
#include <string>

std::string s = std::to_string(42);
```

More on String in C++ (1/5)

```
#include <string>
string s; // in C, perhaps we use char *s or char s[20]…
cin >> s;
cout << s;
```

More on String in C++ (2/5)

https://onlinegdb.com/96tSeJs1T

```
string s1, s_cp, s5;
string s2 {"main()\n{\n}\n"};
string s3 {"is the best in TW."};
string s4 {"NTOU CSE"};
s_cp = s3;
s5 = s4 + " " + s_cp;
if (s1.empty())
    cout << "empty string" << endl;
cout << "length of s2: " << s2.length() << endl;
cout << s5 << endl;
```

```
empty string
length of s2:11
NTOU CSE is the best in TW.
```

More on String in C++ (3/5)

```
string s {"NTU CSIE is fantastic!"};
s[2] = 'O', s[7] = ' ';
cout << s << endl;
s.at(3) = 'U'; s.at(6) = 'E';
cout << s << endl;
```

```
NTO CSIE is fantastic!
NTOUCSE  is fantastic!
```

An example of reversing a string:
https://github.com/pangfengliu/Cplusplus-refactor/blob/main/string/string-reverse.cc

More on String in C++ (4/5)

```
string zodiac[12];
for (string &s : zodiac)
  cin >> s;

for (int i = 10; i >= 1; i--)
  for (int j = 0; j <= i; j++)
    if (zodiac[j] > zodiac[j + 1])
   swap(zodiac[j], zodiac[j + 1]);

for (string &s : zodiac)
  cout << s << endl;
```

Comparing string by the ASCII code

- "abc" < "def", "a" < "abc", "ABC" < "abc",...

An example of string sorting:

https://github.com/pangfengliu/Cplusplus-refactor/blob/main/string/string-sort.cc

More on String in C++ (5/5)

Prototype:

```
string string::substr(size_t pos, size_t len);
```

```
string s {"NTOU CSE is fantastic!"};
cout << s.substr(5, 3) << endl; // CSE
cout << s.substr(12) << endl; // fantastic!
```

Prototype:

```
size_t string::find(const string &s) const;
```

```
string s {"NTOU CSE is fantastic!"};
auto pos = s.find("CSE");
if (pos != string::npos) // npos is just like nullpointer
    cout << pos << endl; // 5 is the starting position!
pos = s.find("CSIE");
if (pos != string::npos)
    cout << pos << endl; // None!
```

Structure

Define a customized data type: **struct**

```
struct Person {
    char name[50];
    int age;
    float salary;
};
```

Three members of Person:

- name
- age
- Salary

No memory is allocated so far.
Just a blueprint for creating variables of such a datatype.

Define a customized data type: **struct**

```
struct Person {
    char name[50];
    //string name;
    int age;
    float salary;
};
```

```
Person bill;
bill.age = 50;
bill.salary = 10000;

cout << bill.age;
cout << bill.salary;
```

Three members of Person:

- name
- age
- Salary

No memory is allocated so far.
Just a **blueprint** for creating variables of such a datatype.

Another Example: Student

```
struct Student {
    string name;
    int id;
    string phone;
    // why use string?
    float GPA;
    int birthYear;
    int birthMonth;
    int birthDay;
};
```

```
Student josef;
```

```
josef.name = "Josef Kunze";
josef.id = 12345;
josef.phone = "+49-01245678";
josef.GPA = 3.5;
josef.birthYear = 1965;
josef.birthMonth = 9;
josef.birthDay = 30;

cout << "Student: " << josef.name << endl;
cout << "ID: " << josef.id << endl;
cout << "Phone: " << josef.phone << endl;
cout << "GPA: " << josef.GPA << endl;
cout << "Birthday: " << josef.birthYear << "/"
                     << josef.birthMonth << "/"
                     << josef.birthDay
                     << endl;
```

List initialization

```
struct Student {
    string name;
    int id;
    string phone;
    // why use string?
    float GPA;
    int birthYear;
    int birthMonth;
    int birthDay;
};
```

```
Student josef { "Josef Kunze",
                12345,
                "+49-01245678",
                3.5,
                1965,
                9,
                30 };

cout << "Student: " << josef.name << endl;
cout << "ID: " << josef.id << endl;
cout << "Phone: " << josef.phone << endl;
cout << "GPA: " << josef.GPA << endl;
cout << "Birthday: " << josef.birthYear << "/"
                     << josef.birthMonth << "/"
                     << josef.birthDay << "/"
                     << endl;
```

Using pointers to assess object values

```
struct Student {
    string name;
    int id;
    string phone;
    // why use string?
    float GPA;
    int birthYear;
    int birthMonth;
    int birthDay;
};
```

```
Student josef { "Josef Kunze",
                12345,
                "+49-01245678",
                3.5,
                1965,
                9,
                30 };
auto ptr { &josef };
cout << "Student: " << ptr->name << endl;
cout << "ID: " << ptr->id << endl;
cout << "Phone: " << ptr->phone << endl;
cout << "GPA: " << ptr->GPA << endl;
cout << "Birthday: " << ptr->birthYear << "/"
                     << ptr->birthMonth << "/"
                     << ptr->birthDay << "/"
                     << endl;
```

Passing a structure to a function

```
struct Person {
    string name;
    int age;
    float salary;
};

void displayData(Person);

int main() {
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
    displayData(p);

    return 0;
}
```

```
void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

Passing a structure to a function

```
struct Person {
    string name;
    int age;
    float salary;
};

void displayData(Person);

int main() {
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
    displayData(p);

    return 0;
}
```

```
void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

Exercise:
How to make it run successfully?

Return a structure from a function

```
struct Person {
    string name;
    int age;
    float salary;
};

void displayData(Person);
Person getData(Person);

int main() {
    Person p, temp;

    temp = getData(p);
    p = temp;
    displayData(p);

    return 0;
}
```

```
void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    Cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

```
Person getData(Person p) {

    cout << "Enter Full name: ";
    getline(cin, p.name);

    cout << "Enter age: ";
    cin >> p.age;

    cout << "Enter salary: ";
    cin >> p.salary;

    return p;
}
```

Return a structure from a function (passing by reference)

```
struct Person {
    string name;
    int age;
    float salary;
};

void displayData(Person &);
void getData(Person &);

int main() {
    Person p;

    getData(p);
    displayData(p);
    return 0;
}
```

```
void displayData(Person &p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    Cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

```
void getData(Person &p) {

    cout << "Enter Full name: ";
    getline(cin, p.name);

    cout << "Enter age: ";
    cin >> p.age;

    cout << "Enter salary: ";
    cin >> p.salary;
}
```

Benefit of using `struct`

leftUpCorner_x, leftUpCorner_y,

rightDownCorner_x, rightDownCorner_y,

```
double area(double leftUpCorner_x,
            double leftUpCorner_y,
            double rightDownCorner_x,
            double rightDownCorner_y) {
    ...
}
```

Benefit of using `struct`

leftUpCorner_x, leftUpCorner_y,

rightDownCorner_x, rightDownCorner_y,

```
double area(double leftUpCorner_x,
            double leftUpCorner_y,
            double rightDownCorner_x,
            double rightDownCorner_y) {
    ...
}
```

```
struct Rect {
    double leftUpCorner_x;
    double leftUpCorner_y;
    double rightDownCorner_x;
    double rightDownCorner_y;
};
```

Exercise

- Using `struct` to compute the area of the input axis-parallel rectangle

```
struct Rect {
    double leftUpCorner_x;
    double leftUpCorner_y;
    double rightDownCorner_x;
    double rightDownCorner_y;
};
```

SAMPLE INPUT:
```
10 20 30 -10
```

SAMPLE OUTPUT:
```
600
```

Pointer to a structure

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

struct Distance {
    int feet;
    float inch;
};

int main() {
    Distance *ptr, d;

    ptr = &d;

    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> (*ptr).inch;

    cout << "Displaying information." << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches";

    return 0;
}
```

Lambda expressions

- The traditional fashion:

```
struct ADD {
    int operator()(int a, int b) {
        return a + b; }
};
ADD add;
std::cout << add(2, 3) << endl;
```

- In Modern C++: https://onlinegdb.com/bn20980R2

```
auto add = [](int a, int b) { return a+b; };
std::cout << add(2, 3) << endl;
```