Pointer and Struct

Pointer
(指標)

記憶體大小 v.s. 硬碟大小

查看變數的位址

- &a 取得變數a記憶體位址
- cout 輸出指標會顯示 16 進位

```
void print_address() {
    int n;
    vector<int> a;

    cout << &n << '\n';
    cout << &a << '\n';
}
```

輸出結果：

```
0x7ffe4e3a5550
0x7ffe4e3a5560
```

宣告指標變數

- 指標型別 : 可以儲存指標的型別
  - 語法 [type]*
  - 範例 int*
- 指標變數 : 可以儲存指標的變數
  - 語法 [type]* [variable_name];
  - 範例 int* ptr;

```
void pointer_type() {
    int a = 10;
    int *ptr = &a;  // 宣告指標變數
    cout << &a << '\n';
    cout << ptr << '\n';

    // 同一行宣告兩個指標變數
    int b, c;
    int *pb = &b, *pc = &c;
}
```

輸出結果 :

```
0x7ffe4e3a5550
0x7ffe4e3a5550
```

指標取值

- 取出一個指標變數所指向的變數
  - 語法 *[指標變數]
  - 範例 *ptr

```
void get_value() {
    int a = 10;
    int *ptr = &a;
    cout << a << '\n';
    cout << *ptr << '\n'; // 取值

    (*ptr) = 15;
    cout << a << '\n';
    cout << *ptr << '\n';
}
```

輸出結果：

```
10
10
5
5
```

物件指標

- (*a).b 等價於 a->b
- (*a).b() 等價於 a->b()

```
void object() {
    pair<int, int> p(5, 10);
    pair<int, int> *ptr = &p;
    cout << (*ptr).first << '\n';  // 5
    cout << ptr->first << '\n';    // 5

    vector<int> a = {1, 2, 3};
    vector<int> *pa = &a;
    cout << pa->size() << '\n';  // 3
}
```

函數傳指標

- 在函數如果需要改變引數的值
- 可以改成傳入指標

```
void swap_value(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

void swap_ptr(int *ptra, int *ptrb) {
    int temp = *ptra;
    *ptra = *ptrb;
    *ptrb = temp;
}

void func() {
    int a = 1, b = 2;

    swap_value(a, b);
    cout << a << ' ' << b << '\n';

    swap_ptr(&a, &b);
    cout << a << ' ' << b << '\n';
}
```

指標與陣列

指標與陣列

- 陣列中的變數位址是連續的
- 一個 int 是 4 bytes

```
void array_address() {
    int a[8] = {};

    for (int i = 0; i < 8; i++) {
        cout << &(a[i]) << '\n';
    }
}
```

輸出結果：

```
0x7ffe4b67ab80
0x7ffe4b67ab84
0x7ffe4b67ab88
0x7ffe4b67ab8c
0x7ffe4b67ab90
0x7ffe4b67ab94
0x7ffe4b67ab98
0x7ffe4b67ab9c
```

指標與陣列

- 陣列 = 一段連續的指標開頭
- 指標變數 + 1，會根據所型別決定增加的距離

```
void array_head_address() {
    int a[8] = {};

    cout << a << '\n';
    cout << &(a[0]) << '\n';

    cout << a + 1 << '\n';
    cout << &(a[1]) << '\n';
}
```

輸出結果：

```
0x7ffe4b67ab80
0x7ffe4b67ab80
0x7ffe4b67ab84
0x7ffe4b67ab84
```

指標與陣列

- 語法糖
  - a + i = &(a[i])
  - *(a+i) = a[i]
  - *(a+i) = *(i + a) = i[a]

```
void sugar() {
    int a[4] = {2, 1, 4, 7};

    cout << *(a + 1) << '\n';
    cout << *(1 + a) << '\n';
    cout << a[1] << '\n';
    cout << 1[a] << '\n';
}
```

輸出結果：

```
1
1
1
1
```

指標與陣列

- 字元陣列語法糖
  - "abc"[2]

```
void cstring_sugar() {
    char c = *("xyz" + 1);
    cout << c << '\n';
    cout << "xyz"[0] << '\n';
    cout << "xyz"[1] << '\n';

    int n = 4, a[4] = {2, 1, 4, 7};
    for (int i = 0; i < 4; i++) {
        cout << a[i] << " \n"[i == n - 1];
    }
}
```

輸出結果：

```
y
x
y
2 1 4 7
```

陣列/多維陣列/指標

- []是將前面的括號前面和中間位置合併
- 陣列的名稱是一個指標
  - arr[10] = 10[arr] = *(arr + 10) 相同!

```
int arr[5];
for(int i = 0; i < 5 ; i++)
     arr[i] = i + 2;
```

變數	記憶體位置 (&)	數值
arr[0]	0x61fedc	2
arr[1]	0x61fee0	3
arr[2]	0x61fee4	4
arr[3]	0x61fee8	5
arr[4]	0x61feec	6

- 多維陣列也是如此
  - 陣列arr[n][m]中 arr[i][j] = *(arr + j + i*m)

陣列		
arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

記憶體位置 (&)		
0x61fee8	0x61feec	0x61fef0
0x61fef4	0x61fef8	0x61fefc

函數傳遞多維指標陣列

- 給定第二維度大小

```
//給定第二維度大小
void func(int arr[3][]){

}

int main(){
    int arr[5][3];
    func(arr);

    return 0;
}
```

new / delete

- 利用 new 取得一個新的變數，可使用範圍為**全域**
- 利用new可以宣告動態陣列
- delete可以釋放空間

```
void new_del() {
    int x = 10;
    int *ptrx = &x;

    int *ptry = new int(15);
    delete ptry;
}

void new_del_array() {
    int *a = new int[10];
    cout << a[0] << endl;
    delete[] a;
}
```

參照
reference

reference

- 在函數中傳遞vector非常耗時間
- 利用reference vector
  - 可以節省複製vector時間
  - 也可以修改vector數值

```
#include <bits/stdc++.h>
using namespace std;

void func(vector<int> &vec) {
    vec.push_back(30);
}

int main() {
    vector<int> vec;
    vec.push_back(10), vec.push_back(20);
    func(vec);
    for (int i = 0; i < vec.size(); i++)
        cout << vec[i] << " ";
}
```

輸出結果：

```
10 20 30
```

Iterator

iterator/迭代器 - vector

- vec.begin()
  - vec的開始
- vec.end()
  - vec的結束
- lower_bound(vec.begin(), vec.end() , x)
  - x在vec中的記憶體位置

```
vector<int> a = {1, 2, 3, 4, 5};

for (vector<int>::iterator i = a.rbegin(); i != a.rend(); i++) {
    cout << *i;
}
```

iterator/迭代器 - map/set

- begin(), end()
  - 整個set/map的開始/結束
- lower_bound
  - 數值在STL中的位置
- erase
  - 可以傳入要刪除的數值或是iterator

```
set<int> myset;
set<int>::iterator it;

for (int i = 1; i < 10; i++) {
    myset.insert(i * 10);
}

it = myset.begin();
++it;  // it 指向 20
myset.erase(it);
myset.erase(40);
```

struct

Struct

- C++ 或一些 library 有內建型別(type) `int`, `double`, `string` ...
- 如果這些不足以表示我們的資料呢？
- 用 `struct` 來表示一個箱子的資料，`red` 和 `blue` 分別代表這個箱子裡面有幾個紅色的球和幾個藍色的球

```
struct Box {
  int red, blue;
};

void test_box() {
    Box a;
    a.red = 10;
    a.blue = 5;

    Box b = a;
    cout << b.red << '\n';  // 10
    cout << b.blue << '\n'; // 5
}
```

Constructor

- 初始化當前物件的函式
- 名字和 struct 名稱一樣
- 沒有回傳型別

```
struct Box {
    int red, blue;

    // 建構子
    Box() {
        red = 0, blue = 0;
    }
    // 建構子
    Box(int n_red, int n_blue) {
        red = n_red, blue = n_blue;
    }
};

void test_box() {
    // 宣告一個空的箱子
    Box box1 = Box();

    // 宣告一個有 1 顆紅球 2 顆藍球的箱子
    Box box2 = Box(1, 2);
}
```

Functions in Struct

- 可以在 struct 內寫函式

```
struct Box {
    int red, blue;

    Box(int n_red = 0, int n_blue = 0) {
        red = n_red, blue = n_blue;
    }
    void addRedBall(int n) { red += n; }
    void addBlueBall(int n) { blue += n; }
};

void test_box() {
    Box x = Box(2, 3);

    x.addRedBall(3);
    x.addBlueBall(1);

    cout << x.red << '\n';  // 5
    cout << x.blue << '\n'; // 4
}
```

Operation Overloading

- 兩個 Box 相加會是甚麼?
  - 兩個 int 相加是數值相加
  - 兩個 string 相加是字串連接
- 我們自己定義一個型別的運算

```
Box operator+(Box a, Box b) {
    Box ret;
    ret.red = a.red + b.red;
    ret.blue = a.blue + b.blue;
    return ret;
}

void test_box() {
    Box a = Box(2, 3);
    Box b = Box(4, 7);

    Box c = a + b;
    cout << c.red << '\n';   // 6
    cout << c.blue << '\n';  // 10
}
```

struct 綜合練習

- 用 struct 定義一個叫做 Point3D 的類別，包含三個型別為 double 的成員變數 x, y, z
- 實作 distance 函式，傳入兩個 Point 3D 物件，回傳兩點距離
- 實作建構函數將 x, y, z 初始值設定為 0
- 實作建構函數可以傳入欲設定的 x, y, z 初始值
- 實作成員函式 abs2 回傳 x*x + y*y + z*z
- 實作 operator + 將兩個 Point3D 的物件的 x, y, z 個別相加
- 實作 operator - 將兩個 Point3D 的物件的 x, y, z 個別相加