

Homework 3

班級	姓名	學號	日期
四機械四乙	吳宇昕	B10831020	10/7/2022

Part 1 Problem A

Copy a const int pointer to non-const pointer

[test code](#)

欲複製const pointer，必須將其儲存於另一個const pointer。若新的pointer並非const，則會產生compile time error。

```
1  int main()
2  {
3      int number = 10831020;
4      const int* number_ptr = &number;
5      int* another_ptr = number_ptr;
6  }
```

原本預期c++的mutable關鍵字可以暫時避免compiler設下的這道防護機制，強迫其將const int*複製給int*但是發現這樣做沒有用。看來mutable關鍵字是專門用來讓class裡的const函式修改class member用的，無法像我這樣使用。

```
1  int main()
2  {
3      int number = 10831020;
4      const int* number_ptr = &number;
5      mutable int* another_ptr = number_ptr;
6  }
```

Part 1 Problem B

Why should I pass variables as reference

[test code](#) and [replit](#)

變數被pass by value進到函式時，將會在該函式的stack frame裡產生其數值的副本。複製體積大的變數或物件時，將會耗費CPU資源以及記憶體空間。若是pass by reference，函式接收到的是該物件或變數的記憶體位置，只需要到該位置取值運算，不需要複製整個變數值進自己的stack frame。

我用chrono套件量測將一個含有100000個double的vector pass by value與pass by reference進到函式裡，並修改其值需花費的時間。vector宣告如

```
std::vector<double> largeVtr(100000)
```

Pass by reference 與 pass by value 宣告與參數如

```
void passedByRef(std::vector<double>& _largeVtr)
void passedByValue(std::vector<double> _largeVtr)
```

實驗使用的 [test code](#) 在此。發現在g++ compiler優化前，兩個版本的函式各執行1000次平均時間差0.0113839秒，而開啟g++ compiler優化-O3選項後，差距為0.00413375秒。若是體積更大的object需要被反覆傳入函式，更可以觀察出pass by value與pass by reference的效能差異。

Part 1 Problem C

What are the difference between `int myInt[10]` and `int* myInt[10]`

[test code](#)

前者會在stack上配置一段連續的記憶體，長度40 bytes，儲存int數值，並取得指向[0]的pointer。後者在stack上建立10個連續的int pointer，分別指向零散、非連續的記憶體位置。後者做任何數值運算，需要dereference陣列的每個元素。未經dereference，會出現compile time error

```
4   int arr[10];
5   int* arr_ptr[10];
6
7   arr[0] = 100;
8   arr_ptr[2] = 100;
9   arr_ptr[4] = 100;
10  arr_ptr[6] = 100;
```

然而，原本預期dereference各個元素之後就可以對其賦值，實際操作卻出現runtime error, segmentation fault。不清楚應該如何使用`int* myint[10]`的語法，避免出錯。

```
4   int arr[10];
5   int* arr_ptr[10];
6
7   arr[0] = 100;
8   *arr_ptr[2] = 100;
```

Exception has occurred. ×
Segmentation fault

從vscode檢視記憶體位置，可以看見`int* arr_ptr[10]`配置的10個整數pointer指向記憶體各處，各個整數的記憶體位置凌亂。甚至不知道甚麼原因，`[0]`跟`[2]`指向相同的記憶體位置：

The screenshot shows a C++ IDE with a variable viewer on the left and a code editor on the right. The variable viewer shows the following variables:

- VARIABLES**
 - Locals**
 - `arr`: [100]
 - `arr_ptr`: [100]
 - `[0]`: 0x61fc14
 - `[1]`: 0x769d631c <msvcrt!_get_curre...>
 - `[2]`: 0x61fc14
 - `[3]`: 0x769e6f95 <unlock+21>
 - `[4]`: 0x76a3438c <msvcrt!_aexit_rtn+45...>

The code editor shows the following code:

```
code > HW3 > CODE > C++ HW3C.cpp
1  #include <iostream>
2  int main()
3  {
4      int arr[100];
5      int* arr_ptr[100];
6
7      arr[0] = 100;
8      *arr_ptr[0] = 100;
9      system("pause");
```

對 `int* myInt[10]` 的語法仍然不熟悉，不清楚其應用為何。

Part 1 Problem D

Workshop

[test code](#) and [replit](#)

宣告變數，得到下圖的輸出

```
int number = 10831020;
int* number_ptr = &number;
int* number_ptr2 = number_ptr;
```

Printing all 3 symbols of number:

&number	*number	number
0xffff000bd4	N/A	10831020

Printing all 3 symbols of number_ptr:

&number_ptr	*number_ptr	number_ptr
0xffff000bd8	10831020	0xffff000bd4

可以發現 `&number` 與 `number_ptr` 顯示相同的記憶體位置，而 `number` 與 `*number_ptr` 顯示相同值。
`number_ptr` 本身也占據記憶體位置，但是它自身的記憶體位置與 `number` 的記憶體位置無關，為任意數。然而其指向的記憶體位置必與 `&number` 相同。由於 `*number` 沒有意義，圖中 `print*number` 的欄位以 `N/A` 代替。

Part 1 Problem E

Overloading functions

[test code](#)

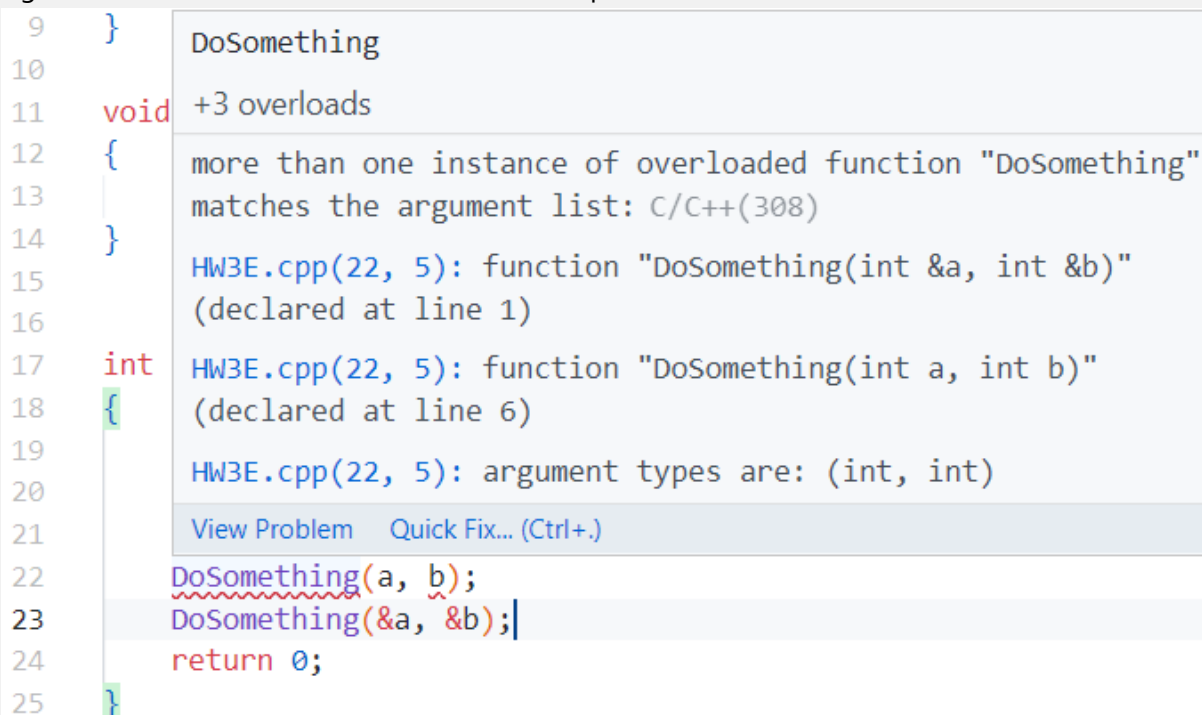
宣告三個版本的函式，positional arguments 接採用不同的 type 產生 overload 效果。呼叫函式時，compiler 會以傳入的變數的 type 自動決定應該使用哪個版本的函式。

```
int DoSomething(int a, int b);
void DoSomething(int& a, int& b);
void DoSomething(int* a, int* b);
```

三個版本分別傳入變數值、變數的reference，以及變數的記憶體位置。以三種call signature分別呼叫

```
int result = DoSomething(a, b);
DoSomething(a, b);
DoSomething(&a, &b);
```

原本預期三種call signature都可以順利執行，卻發現compiler無法判別有無return type決定前兩種call signature該呼叫哪個版本的函式，因此發生compile time error。



```
9   }
10  }
11  void +3 overloads
12  {
13      more than one instance of overloaded function "DoSomething"
14      matches the argument list: C/C++(308)
15      HW3E.cpp(22, 5): function "DoSomething(int &a, int &b)"
16      (declared at line 1)
17      HW3E.cpp(22, 5): function "DoSomething(int a, int b)"
18      (declared at line 6)
19      HW3E.cpp(22, 5): argument types are: (int, int)
20      View Problem Quick Fix... (Ctrl+.)
21
22  DoSomething(a, b);
23  DoSomething(&a, &b);
24  return 0;
25  }
```

Part 2

[source code](#) and [replit](#)

盡可能讓main()內容簡潔，13行以內塞下完整的程序。之前老師建議把print等函式寫在main()裡面，但這次需要另外寫PrintMatrix函式在main()之外，讓矩陣印出來比較美觀。二維矩陣，以及main()定義如下：

```
6  using INT_COL = std::vector<int>; //vector of ints
7  using INT_MATRIX = std::vector<INT_COL>; //vector of vectors of ints
```

```

14  ✓ int main()
15  {
16      printf("Student B10831020\n");
17      INT_MATRIX m1; //create a vector of vectors of ints
18      INT_MATRIX m2; //create a vector of vectors of ints
19      INT_MATRIX matrixAdditionResult; //the matrix to store calculation results
20      INT_MATRIX matrixSubtractionResult; //the matrix to store calculation results
21      AssignRandomValues(m1, m2, 4, 4); //give random values to the two matrices
22      PrintMatrix(m1, "Matrix m1", 4, 4);
23      PrintMatrix(m2, "Matrix m2", 4, 4);
24      MatrixAdd(m1, m2, 4, 4, matrixAdditionResult);
25      PrintMatrix(matrixAdditionResult, "Addition of the two matrices", 4, 4);
26      MatrixSubtract(m1, m2, 4, 4, matrixSubtractionResult);
27      PrintMatrix(matrixSubtractionResult, "Subtraceton of the two matrices", 4, 4);
28      system("pause");
29      return 0;
30  }

```

不知道在命名習慣上，`typedef`與`using`自訂義的type name需不需要全部大寫？我假設需要，因此把二維矩陣取名為`INT_MATRIX`，卻發現程式碼裡太多大寫字母顯得雜亂。

作業中遇到困難的，是`rand()`函式每次程式執行皆輸出相同的一系列數字。後來才知道需要用`srand(time())`將執行當下的時間點作為random seed，才能避免此狀況。這樣一來，就可以確保每次`rand()`會輸出不同的數值。

Part 3

[source code](#) and [replit](#)

自訂義`struct hozRow`與`struct struct_mat`，將`hozRow`作為`struct_mat`矩陣的一橫列。每個矩陣含有4列，每列有4個int。宣告如下：

```

struct hozRow
{
    int w, x, y, z;
};
struct struct_mat
{
    hozRow row0, row1, row2, row3;
};

```

由於struct裡各個member的記憶體位置前後相連，且順序固定，可以用pointer arithmetic技巧走訪陣列各整數位置

```

40 void AssignRandomValue(struct_mat& _m1, struct_mat& _m2)
41 {
42     srand(time(0));
43     for(int i=0; i<N_ROW; i++){
44         for(int j=0; j<N_COL; j++){
45             //get the beginning memory location of _m1 and cast the pointer to int* to operate on each value
46             int* m1_ptr = (int*)&_m1 + (N_COL*i + j);
47             int* m2_ptr = (int*)&_m2 + (N_COL*i + j);
48             *(m1_ptr) = rand() % 100 - 200;
49             *(m2_ptr) = rand() % 100 - 200;
50         }
51     }
52 }

```

我們土炮做出來的二維矩陣，不知道效能有沒有比一般的 `int 2Darray[][]` 更好。這樣宣告的二維矩陣程式碼相當易讀，但是 `compiler` 並不會把矩陣的每一列放在記憶體相鄰處；矩陣的各列將會四散於記憶體各處，而我們的 `struct` 就可以確保整個陣列各列的記憶體位置必定相鄰。

然而，這樣用 `struct` 寫二維矩陣大幅降低程式易讀性。若真的需要讓矩陣的各列相鄰，我應該會把二維陣列寫成一維，像是：

```

constexpr n_rows, n_cols;
int matrix[n_rows * n_cols]; //declare 2D matrix as 1D

//iterate through matrix
for(int i=0; i<n_rows; i++){
    for(int j=0; j<n_cols; j++){
        matrix[j + i*n_cols] = //do stuff;
    }
}

```