HW3: Struct/Matrix & C++ 視覺化

Name	Student ID	Date	Grades
楊英豪	B10803207	2022/10/16	

Part 1 -- C++ 視覺化 (Chapter 8 Workshop)

請將本作業之workshop極短程式碼,在tutor網站作視覺化練習
註:自行加入合適的cout以顯示(位址/實值)
(part1-VIDEO :: HW3-A, B, C, D, E FIVE Problems)

註: ch8 Workshop -- See also the file marked as "SAMS" (ch8)

HW3-A

Question:

I have a pointer:

```
int number = 30;

const int* pointToAnInt = &number;
```

I understand that I cannot change the value of number using the pointer pointToAnInt due to the const declaration.

Can I assign pointToAnInt to a non-const pointer and then use it to manipulate the value contained in integer number?

Answer:

Once you have a const object, it cannot be assigned to a non-const reference or use functions that are known to be capable of changing the state of the object. This is necessary to enforce the const-ness of the object, but it means you need a way to state that a function should not make changes to an object.

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

int main() {
int number = 30;
const int* pointToAnInt = &number;
cout<<pointToAnInt<<endl;
int* pAnother = pointToAnInt;
  return 0;
}
```

error: invalid conversion from 'const int*' to 'int*' [-fpermissive]

HW3-B

Question: Why should I bother passing values to a function by reference

Answer:

When we use pass by value, the function will copy the value into a new address in RAM, if we pass a large data like struct or hash map, passing value would be an expensive operation. Pass by reference as function arguments is faster than pass by values

HW3-C

Question: What is the difference between these two declarations:

```
int myNumbers[100];
int* myArrays[100];
```

Answer:

Basically the difference in the two is that myArrays is an array that holds 100 pointers, each pointer can point to an integer or an array or integer. While myNumbers is an array that holds 100 integers, It is a static method, pointing to the array's first index (0)

HW3-D

Question

What is the display when these statements are executed:

0: int number = 3;

1: int* pNum1 = &number;

2:*pNum1 = 20;

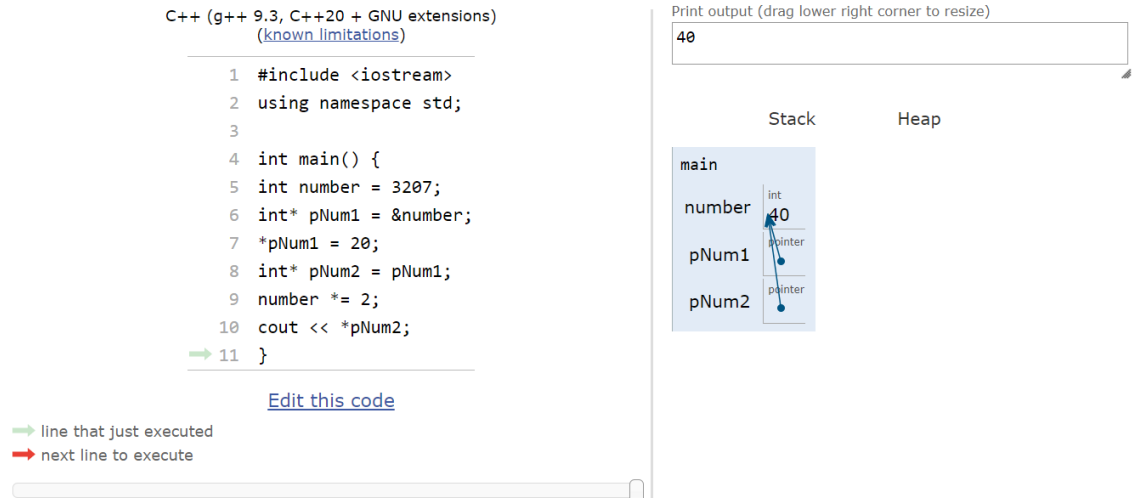
3: int* pNum2 = pNum1;

4: number *= 2;

5: cout << *pNum2;

Answer:

When (0) is executed it means that a variable with type integer is assigned a value of 3. When (1) is executed it means that create a pointer variable pNum1 that points to the address of number. When (2) is executed it means that value of the pointer variable pNum1, that points to number is given a value of 20, when (3) is executed it means create a pointer variable pNum2 that points to the address of pNum1, when (4) is executed multiply the value of variable number by 2. When (5) is executed print out the value of the pointer variable that points to the address of pNum2.



HW3-E

Question:

What are the similarities and differences between these three overloaded functions:

```
int DoSomething(int num1, int num2);
int DoSomething(int& num1, int& num2);
int DoSomething(int* pNum1, int* pNum2);
```

Answer:

The simililarity of the functions is that these functions have the same name but diferrent parameters. The first function parameters is passed by value, the second function parameters is passed by reference using the address, while the third function is basically using pointers as a parameter.

Part 2: Matrix (2D Array ::-C++11 Vector style)

產生2個4×4矩陣(M1, M2),進行矩陣加/減之運算,輸出其結果(MPlus ,MMinus)
M1/M2之每一row資料,每一橫列之四值方式,由亂數(值介於-100~+100)產生代入之。

```
#include <vector>
#include <iostream>
#include<time.h>
using namespace std;

// A type alias declaration for a vector of double value
```

```
using RealVec=vector<double>;

// A type alias for vector dimensions (size)
using Dim = RealVec::size_type;

//A type alias for vector type (e.g. double)
using DataType = RealVec::value_type;

//A type alias for a vector of vectors with doubles= a real matrix
using RealMatrix=vector<RealVec>;

void nonDiagonalSum(RealMatrix m)
{
    //Accumulate elements except the diagonal
    double non_diagonal_sum {};

    for (int r=0; r < m.size();++r)
        for(int c=0;c<m[0].size();++c)
            if (c != r)
                non_diagonal_sum += m[r][c];
    cout<<"Sum of m except the diagonal "<<non_diagonal_sum<<endl;
}

void generateMatrix(double **a){
    int i,j;

    for (i=0 ; i<4 ; i++){
        a[i]=new double[4];
        for (j=0; j<4 ; j++){
            a[i][j] = rand()%(201)-100;
        }
    }
     for( i = 0; i < 4; ++i)
        {for( j = 0; j < 4; ++j)
     cout<<a[i][j]<<'\t';
   cout<<'\n';
 }
}

void matrixAddition(double **a,double **b, double**sum){
int i,j;
for(i = 0; i < 4; ++i){
        sum[i]=new double[4];
        for(j = 0; j < 4; ++j)
            sum[i][j] = a[i][j] + b[i][j];
}

for( i = 0; i < 4; ++i){
        {for( j = 0; j < 4; ++j)
     cout<<sum[i][j]<<'\t';
   cout<<'\n';
 }
}
}
```

```
void matrixSubstraction(double **a,double **b, double**min){
int i,j;
for(i = 0; i < 4; ++i){
        min[i]=new double[4];
        for(j = 0; j < 4; ++j)
            min[i][j] = a[i][j] - b[i][j];
}

for( i = 0; i < 4; ++i){
        {for( j = 0; j < 4; ++j)
     cout<<min[i][j]<<'\t';
   cout<<'\n';
 }
}
}

int main(){
double **M1= new double*[4], **M2=new double*[4];
double **MPLUS=new double*[4], **MMINUS=new double*[4];
// Using RealMatrix as it is
RealMatrix m ={
                { 2, 2, 0, 11 },
                { 3, 4, 5, 0 },
                { -1, 2, -1, 7 }
            };

//Add next row
m.push_back({ 5, 3, 5, -3 });
 srand(time(0));
nonDiagonalSum(m);
cout<<"\nFirst Matrix M1"<<endl;
generateMatrix(M1);
cout<<"\nSecond Matrix M2"<<endl;
generateMatrix(M2);
cout<<"\nAddition"<<endl;
matrixAddition(M1,M2,MPLUS);
cout<<"\nSubstraction"<<endl;
matrixSubstraction(M1,M2,MMINUS);
}
```

Output: