Instructor: Dr. Jie Shen
Release date: Mar 08, 2021
Due date: Mar 22, 2021

Student Name: Demetrius Johnson

*Special note: see my lab uploads of the .CPP and .EXE files for ease of access and testing any of the programs for any question.

Table of Contents

Question 1

Source code (USED C++ COMPILER on Microsoft Windows 10)

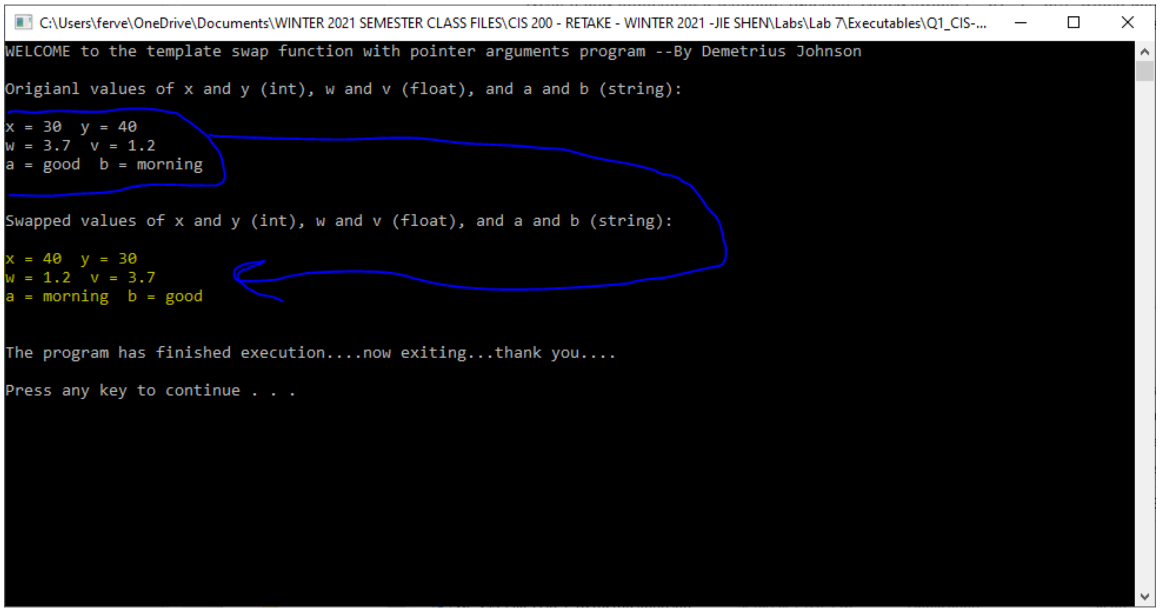
```
// CIS-200-LAB 7 -DemetriusJohnson.cpp : This file contains the 'main' function. Program execution
begins and ends there.
//

/*
//Author: Demetrius E Johnson
//Date: 16 MAR 2021
//Last Modification Date: 03-16-2021
//Purpose: use a template function that can swap many data types


*/

/*
Question 1:

//Design and implement a template function, called swapt(T *p1, T *p2),
//which takes two parameters with the same generic data type.
//The function swaps the values of these two parameters.


*/




#include <iostream>
//#include<assert.h>
using namespace std;


//FUNCTION DECLARATIONS

template<typename T>
void swapT(T* p1, T* p2);

//FUNCTION DECLARATIONS



int main()
{

    cout << "WELCOME to the template swap function with pointer arguments program --By Demetrius
Johnson\n\n";

    //originally declared values
    int x = 30, y = 40;
    float w = 3.7, v = 1.2;
    string a = "good", b = "morning";

    cout << "Origianl values of x and y (int), w and v (float), and a and b (string):\n\n";
    cout << "x = " << x << "  y = " << y << endl;
    cout << "w = " << w << "  v = " << v << endl;
    cout << "a = " << a << "  b = " << b << endl;



    //swapping values
    swapT(&x, &y);
    swapT(&w, &v);
    swapT(&a, &b);

    cout << "\n\nSwapped values of x and y (int), w and v (float), and a and b (string):\n\n";
    cout << "x = " << x << "  y = " << y << endl;
    cout << "w = " << w << "  v = " << v << endl;
    cout << "a = " << a << "  b = " << b << endl;


    cout << endl << endl << "The program has finished execution....now exiting...thank you....\n\n";
    system("pause");

    return 0;
}


//FUNCTION DEFINITIONS BELOW THIS LINE

template<typename T>
void swapT(T* p1, T* p2) {

    T holder;
    holder = *p1;//let holder store the value stored at the address to which p1 points to

    *p1 = *p2; //set the value stored at p1 equal to the value stored at p2
    *p2 = holder; //now set the value stored by p2 equal to the original data value p1 stored

    return;
}
```

Test data and expected results

Test Table:

Test #	Valid / Invalid Data	Description of test	Input Value	Expected Output	Actual Output	Test Pass / Fail
1	valid	Test the swap template function on variables of type int, float, and string	Int values: 30 and 40 Float values: 1.2 and 3.7 String values “morning” and “ good	All three data types swapped.	See screenshot	pass

TEST 1:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q1_CIS-...

```
WELCOME to the template swap function with pointer arguments program --By Demetrius Johnson

Origianl values of x and y (int), w and v (float), and a and b (string):

x = 30  y = 40
w = 3.7  v = 1.2
a = good  b = morning

Swapped values of x and y (int), w and v (float), and a and b (string):

x = 40  y = 30
w = 1.2  v = 3.7
a = morning  b = good


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

Question 2

Source code (USED C++ COMPILER on Microsoft Windows 10)

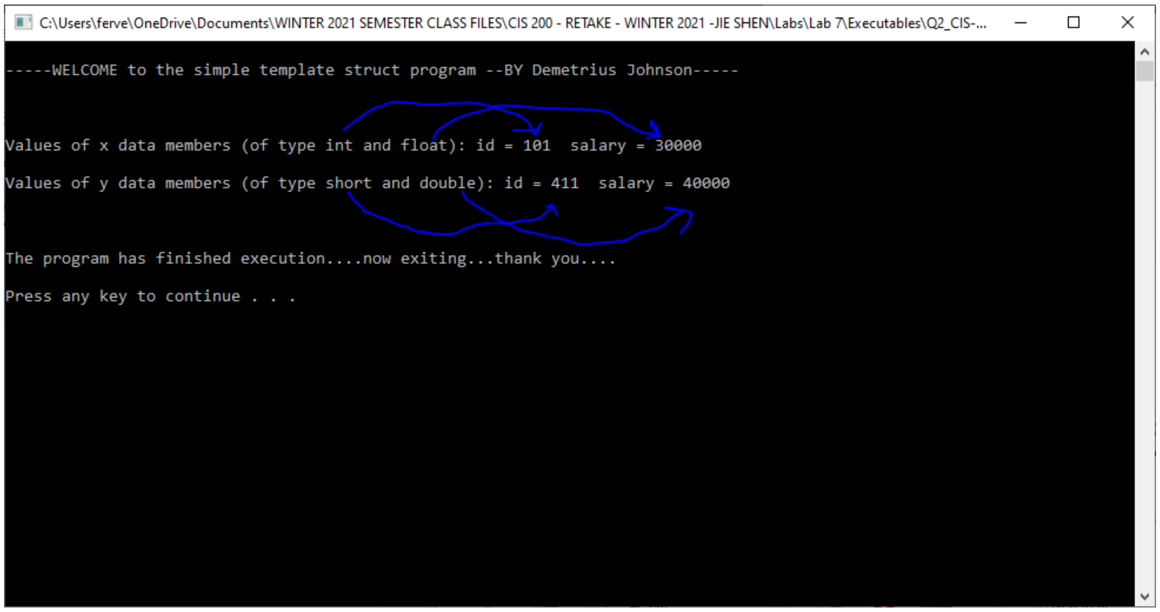
```
// CIS-200-LAB 7-DemetriusJohnson.cpp : This file contains the 'main' function. Program execution begins
and ends there.
//

/*
//Author: Demetrius E Johnson
//Date: 16 MAR 2021
//Last Modification Date: 03-16-2021
//Purpose: implement a template struct


*/

/*
Question 2:

Design and implement a template struct, called Employeet,
//which contains two data members (id and salary).
//The data types of id and salary are T1 and T2, respectively.

*/



#include <iostream>
//#include<assert.h>
using namespace std;


//FUNCTION DECLARATIONS
template<typename T1, typename T2>
struct EmployeeT { T1 id; T2 salary; };

//FUNCTION DECLARATIONS



int main()
{

    cout << "\n-----WELCOME to the simple template struct program --BY Demetrius Johnson-----\n\n";


    //declare an instance of Employeet, x, with data types:  int and float
    EmployeeT<int, float> x;

    //Assign int value (101) and float value (30000.00) to this instance
    x.id = 101;
    x.salary = 30000.00;

    //Print out the values of the data members to computer screen
    cout << "\n\nValues of x data members (of type int and float): id = " << x.id << "  salary = " << x.salary;

    //declare an instance of Employeet, y, with data types:  short and double
    EmployeeT<short, double> y;

    //Assign int value (411) and float value (40000.00) to this instance
    y.id = 411;
    y.salary = 40000.00;

    //Print out the values of the data members to computer screen
    cout << "\n\nValues of y data members (of type short and double): id = " << y.id << "  salary = " <<
y.salary;


    cout << endl << endl << "\n\nThe program has finished execution....now exiting...thank you....\n\n";
    system("pause");

    return 0;
}


//FUNCTION DEFINITIONS BELOW THIS LINE
```

Test data and expected results

Test Table:

Test #	Valid / Invalid Data	Description of test	Input Value	Expected Output	Actual Output	Test Pass / Fail
1	valid	Test if the template struct can handle various data types for its initialization	Using int and float, then using short and double	Id and salary should output properly and be assigned to the created struct	See screenshot	pass

TEST 1:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q2_CIS-...

```
-----WELCOME to the simple template struct program --BY Demetrius Johnson-----


Values of x data members (of type int and float): id = 101  salary = 30000

Values of y data members (of type short and double): id = 411  salary = 40000


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

Question 3

Source code (USED C++ COMPILER on Microsoft Windows 10)

```
// CIS-200-LAB 7 -DemetriusJohnson.cpp : This file contains the 'main' function. Program execution
begins and ends there.
//

/*
//Author: Demetrius E Johnson
//Date: 16 MAR 2021
//Last Modification Date: 03-16-2021
//Purpose: practice the use of dynamic allocation using a multi-dimensional array of 3 Dimensions


*/

/*
Question 3:

Dynamic memory allocation is an efficient and flexible way for software development. Design and
implement a function, bar( ... ),
which dynamically allocates a three-dimensional arrays. This function has four parameters:
1) a parameter for transmitting the array between main( ) and bar( );
2) an integer parameter for specifying the length of the first index of the array;
3) an integer parameter for specifying the length of the second index of the array;
4) an integer parameter for specifying the length of the third index of the array.

In the function of bar( ), you need to do the following tasks:

i) dynamically allocate a three-dimensional array (say x[][][] ) with length (length1, length2, length3)

ii) perform the following loops:
    for(int i=0; i< length1; i++)
        for(int j=0; j<length2; j++)
            for(int k=0; k<length3; k++)
                x[i][j][k]= sin(i*j*k);

  In the function of main( ), you need to do the following tasks:

i) assume the three-dimensional array is of float type;
ii) ask users to input three integer parameters: s1, s2, and s3 for lengths of three indices of the array to be
created.
iii) call bar(…)  by passing s1, s2 and s3 to length1, length 2 and length3 in bar( )
iv) after finishing the calling of bar(…), perform the following loops:
    for(int i=0; i< s1; i++)
        for(int j=0; j<s2; j++)
           for(int k=0; k<s3; k++)
                cout << "i= " << i << "  j= " << j  <<  " k= " << k << "y[i][j][k]=  " <<
                          y[i][j][k] << endl;
v) deallocate the array created in bar( )
```

```
*/



#include <iostream>
//#include<assert.h>
using namespace std;


//FUNCTION DECLARATIONS

template<typename T>
int Triple_P_alloc(T*** Triple_P, int length1, int length2, int length3);
template<typename T>
int Triple_P_dealloc(T*** Triple_P, int length1, int length2); //note length3 is not used in this function
because it is the length of
                                                            //the single pointers allocation which point only to addresses of
data variables - not other pointer variables;




//FUNCTION DECLARATIONS

//**SPECIAL NOTE: THERE ARE 2 WAYS TO PASS BY REFERENCE:
//1) USING POINTER ARGUMENTS [(var_type* var_name) -->pass in argument is &var_name]
//2) USING REFERENCE ARGUMENTS [(var_type& var_name) -->pass in argument is var_name]

int main()
{
   cout << "---WELCOME: This Program implements the dynamic allocation and deallocation of a 3D
Array --BY Demetrius Johnson\n\n\n";

   int L1 = 0;
   int L2 = 0;
   int L3 = 0;
   int num_delete_calls = 0;

   //ask user for size of 3D array
   cout << "Select the array size of the first dimension (L1 = array of double pointers): ";
   cin >> L1;
   cout << "Select the array size of the second dimension (L2 = array of single pointers): ";
   cin >> L2;
   cout << "Select the array size of the third dimension (L3 = array of non-pointer data): ";
   cin >> L3;
   cout << "\n*NOTE: each array value [i][j][k] is set to a value: sin(i * j * k) = radians.\n\n";

   //3D array is of float type
   float*** triple_float = new float**[L1]; //triple pointer points to (stores) a double pointer [address] or an
array of double pointers [addresses]
   int num_dynamic_alloc = 1;
   num_dynamic_alloc += Triple_P_alloc(triple_float, L1, L2, L3); //call allocation function
   cout << "\nDuring allocation, the number of 'new' calls was: 1 + L1 + (L1 * L2) == " << 1 << " + " <<
L1 << " + (" << L1 << " * " << L2 << ") = " << num_dynamic_alloc;
   cout << "\n\nThe forumula comes from:\n\n";
   cout << " 1 --> allocation of memory for triple pointer." << endl << endl;
```

```
    cout << " + L1 --> each element of the triple pointer is a double pointer,\n and each double pointer needs
also to be assigned memory --> of type single pointer." << endl << endl;
    cout << " + (L1 * L2) --> the number of single pointers pointed to by \n each double pointer all need also
to be assigned memory." << endl << endl;
    cout << " ***NOTE: L3, the size of the single pointer memory allocation is\n irrelevant since it only
contains an array of normal data addresses;";
    cout << "\n it only matters how many times 'new' operator is called regardless of size allocated for each
single pointer.\n\n";
    cout << "~Total number of elements in the 3D array is: L1 * L2 * L3 = " << L1 << " * " << L2 << " * "
<< L3 << " = " << (L1 * L2 * L3) << endl;
    // after finishing the calling, perform the following loops :
    for (int i = 0; i < L1; i++)
        for (int j = 0; j < L2; j++)
            for (int k = 0; k < L3; k++)
                cout << "i = " << i << "  j = " << j << " k = " << k << "  triple_float[i][j][k] = " <<
                triple_float[i][j][k] << endl;

    num_delete_calls = Triple_P_dealloc(triple_float, L1, L2); //deallocate the triple pointer
    cout << "\n\nDuring deallocation, the number of 'delete[]' calls was: (L2 * L1) + L1 + 1 == ("<< L2 << "
* " << L1 << ") + " << L1 << " + " << 1 << " = " << num_delete_calls;
    cout << "\n\nThe forumula comes from:\n\n";
    cout << " (L2 * L1) --> the number of single pointers pointed to by \n each double pointer all need also
to be deallocated memory." << endl << endl;
    cout << " + L1 --> each element of the triple pointer is a double pointer,\n and each double pointer array
needs also to be deallocated its memory of single pointers." << endl << endl;
    cout << " + 1 --> the deallocation of memory for the single triple pointer which was storing an array of
single pointers." << endl << endl;
    cout << " ***NOTE: L3, the size of the single pointer memory allocation is\n irrelevant since it only
points to a single array of normal data variable addresses;";
    cout << "\n it is only necessary to call 'delete[]' once for all single pointers regardless of address size
pointed to.\n\n";

    cout << endl << endl << "The program has finished execution....now exiting...thank you....\n\n";
    system("pause");

    return 0;
}


//FUNCTION DEFINITIONS BELOW THIS LINE


//Dynamic memory allocation is an efficientand flexible way for software development. Design and
implement a function, bar(...),
//which dynamically allocates a three - dimensional array.This function has four parameters :
//1) a parameter for transmitting the array between main() and bar();
//2) an integer parameter for specifying the length of the first index of the array;
//3) an integer parameter for specifying the length of the second index of the array;
//4) an integer parameter for specifying the length of the third index of the array.

template<typename T>
int Triple_P_alloc(T*** Triple_P, int length1, int length2, int length3) {

    int num_allocations = 0;//use this to keep track of allocations so that I know how many deallocations
there should be later
```

```
    for (int i = 0; i < length1; i++) { //remember element 0 is the first element; thus we say i < length1 //this
loop allocates memory for all double pointer elements from Triple_P

        Triple_P[i] = new T*[length2]; //double pointer points to (stores) a single pointer [address] or an array
of single pointers [addresses]
        num_allocations++;
        for (int j = 0; j < length2; j++) {//remember element 0 is the first element; thus we say j < length2
//this loop allocates memory for all single pointer elements from double Pointers in Triple_P

            Triple_P[i][j] = new T[length3]; //single pointer points to the address of a data variable or an array
of data variables [addresses]
            num_allocations++;
        }
    }

    //now assign values to all elements of the 3D array
    for (int i = 0; i < length1; i++)
        for (int j = 0; j < length2; j++)
            for (int k = 0; k < length3; k++)
                Triple_P[i][j][k] = sin(i * j * k);

    return num_allocations;
}

template<typename T>
int Triple_P_dealloc(T*** Triple_P, int length1, int length2) {

    int num_delete_called = 0; //use this to track num of deallocations; should match the number of
allocations made from when the 3D array was allocated memory

    for (int i = 0; i < length1; i++) {

        for (int j = 0; j < length2; j++) {//use this loop to deallocate single pointers, and then double pointers

            delete[] Triple_P[i][j]; //dealloc single pointers
            num_delete_called++; //used to keep track/test if proper number of deletes called based on 3D array
size
        }

        delete[] Triple_P[i]; //dealloc double pointers
        num_delete_called++;
    }

    delete[] Triple_P; //finally, dealloc the triple pointer
    num_delete_called++;

    return num_delete_called;
}
```

Test data and expected results

Test Table:

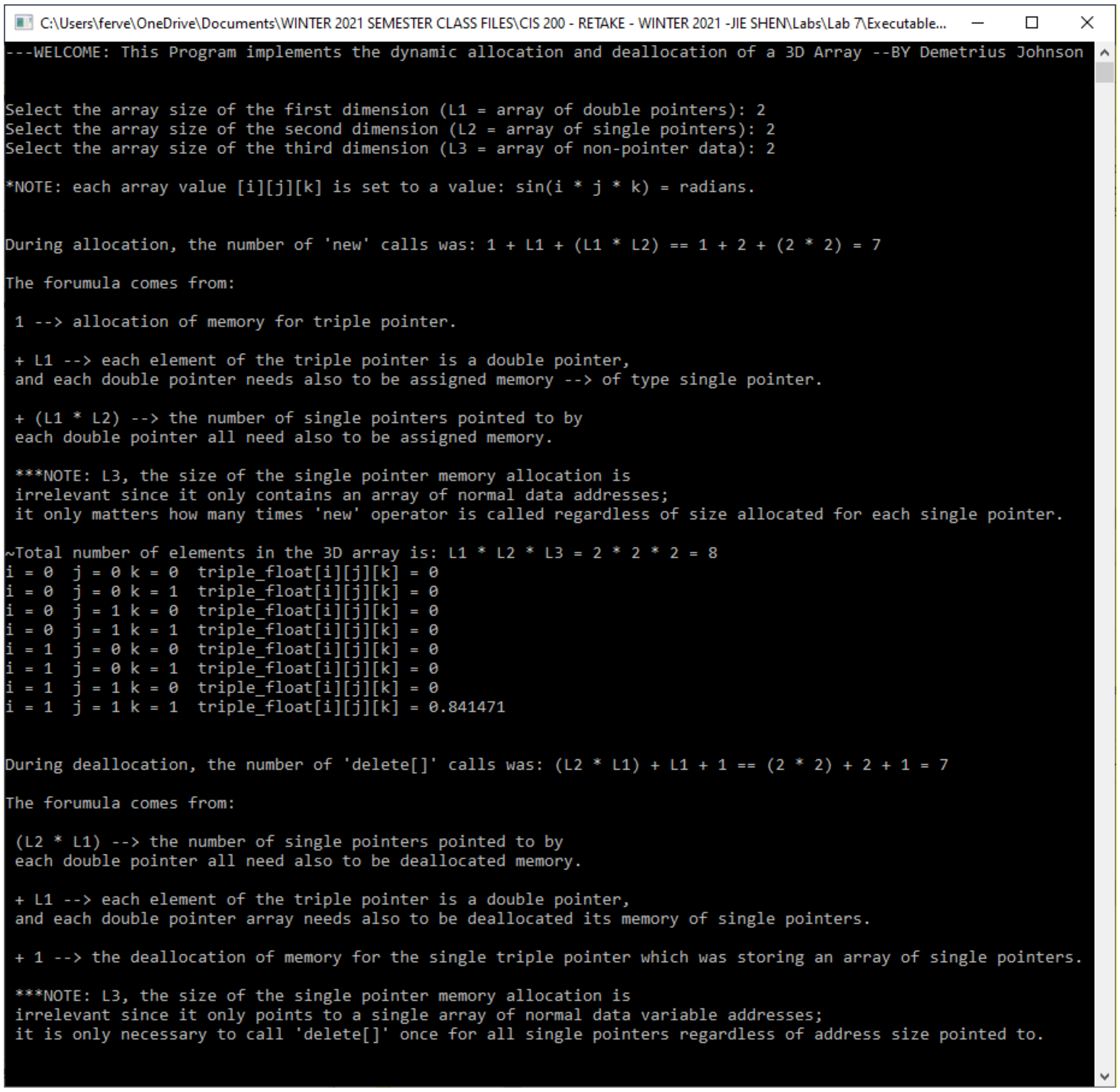
Test #	Valid / Invalid Data	Description of test	Input Value	Expected Output	Actual Output	Test Pass / Fail
1	valid	Give a full view of the program using small input values for array dimensions	L1, L2, L3 == 2, 2, 2	Allocate, output, and deallocate the 3D array of size 8	See screenshot	pass
2	valid	Show that my program works for various dimension sizes and that my prediction formula is accurate	1, 2, 3	Allocate, output, and deallocate the 3D array of size 6	See screenshot	pass
3	valid	Show that my program works for various dimension sizes and that my prediction formula is accurate	1, 3, 3	Allocate, output, and deallocate the 3D array of size 9	See screenshot	pass
4	valid	Show that my program works for various dimension sizes and that my prediction formula is accurate	3, 1, 1	Allocate, output, and deallocate the 3D array of size 3	See screenshot	pass

*Question 3-1: If length1 = n, length2 = n and length3 = n, what is time complexity of the above for loops?

Time complexity is equal to L1 * L2 * L3 = n * n * n = n^3.
Thus; BIG O(N^3).

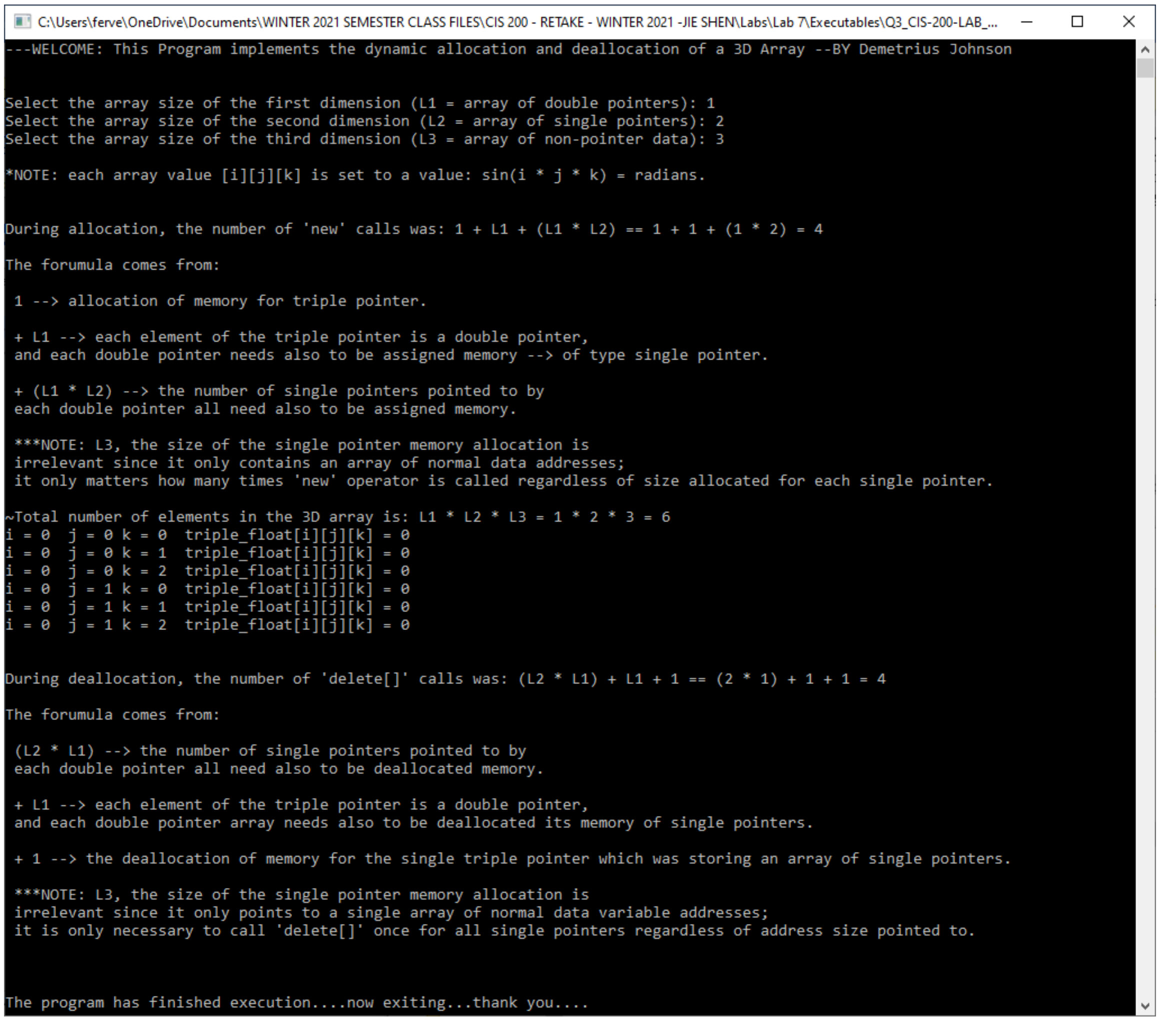
**NOTE: I developed a formula to predict the exact number of ‘new’ and ‘delete’ operators would need to be called for any given lengths for all 3 dimensions of the 3D array. Using a data variables to track whenever I called ‘new’ and ‘delete’, it was able to help me analyze what formula could be put together that will correspond to the value evaluated for any set of dimension sizes. So you will notice in my program how I developed the formula.

TEST 1:



```
C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executable...
---WELCOME: This Program implements the dynamic allocation and deallocation of a 3D Array --BY Demetrius Johnson


Select the array size of the first dimension (L1 = array of double pointers): 2
Select the array size of the second dimension (L2 = array of single pointers): 2
Select the array size of the third dimension (L3 = array of non-pointer data): 2

*NOTE: each array value [i][j][k] is set to a value: sin(i * j * k) = radians.


During allocation, the number of 'new' calls was: 1 + L1 + (L1 * L2) == 1 + 2 + (2 * 2) = 7

The forumula comes from:

 1 --> allocation of memory for triple pointer.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer needs also to be assigned memory --> of type single pointer.

 + (L1 * L2) --> the number of single pointers pointed to by
 each double pointer all need also to be assigned memory.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only contains an array of normal data addresses;
 it only matters how many times 'new' operator is called regardless of size allocated for each single pointer.

~Total number of elements in the 3D array is: L1 * L2 * L3 = 2 * 2 * 2 = 8
i = 0  j = 0 k = 0  triple_float[i][j][k] = 0
i = 0  j = 0 k = 1  triple_float[i][j][k] = 0
i = 0  j = 1 k = 0  triple_float[i][j][k] = 0
i = 0  j = 1 k = 1  triple_float[i][j][k] = 0
i = 1  j = 0 k = 0  triple_float[i][j][k] = 0
i = 1  j = 0 k = 1  triple_float[i][j][k] = 0
i = 1  j = 1 k = 0  triple_float[i][j][k] = 0
i = 1  j = 1 k = 1  triple_float[i][j][k] = 0.841471


During deallocation, the number of 'delete[]' calls was: (L2 * L1) + L1 + 1 == (2 * 2) + 2 + 1 = 7

The forumula comes from:

 (L2 * L1) --> the number of single pointers pointed to by
 each double pointer all need also to be deallocated memory.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer array needs also to be deallocated its memory of single pointers.

 + 1 --> the deallocation of memory for the single triple pointer which was storing an array of single pointers.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only points to a single array of normal data variable addresses;
 it is only necessary to call 'delete[]' once for all single pointers regardless of address size pointed to.
```

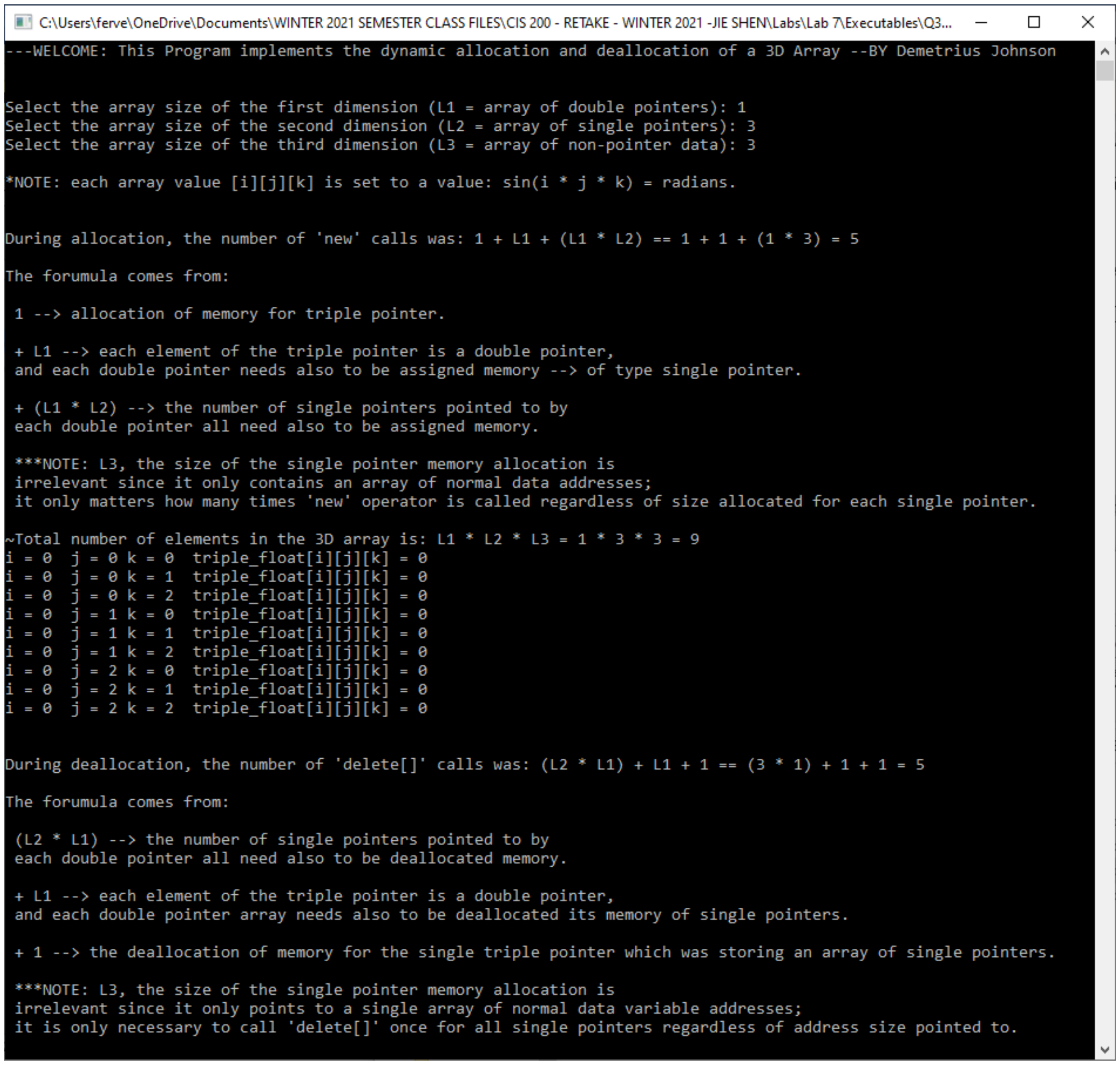
TEST 2:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q3_CIS-200-LAB_...

```
---WELCOME: This Program implements the dynamic allocation and deallocation of a 3D Array --BY Demetrius Johnson


Select the array size of the first dimension (L1 = array of double pointers): 1
Select the array size of the second dimension (L2 = array of single pointers): 2
Select the array size of the third dimension (L3 = array of non-pointer data): 3

*NOTE: each array value [i][j][k] is set to a value: sin(i * j * k) = radians.


During allocation, the number of 'new' calls was: 1 + L1 + (L1 * L2) == 1 + 1 + (1 * 2) = 4

The forumula comes from:

 1 --> allocation of memory for triple pointer.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer needs also to be assigned memory --> of type single pointer.

 + (L1 * L2) --> the number of single pointers pointed to by
 each double pointer all need also to be assigned memory.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only contains an array of normal data addresses;
 it only matters how many times 'new' operator is called regardless of size allocated for each single pointer.

~Total number of elements in the 3D array is: L1 * L2 * L3 = 1 * 2 * 3 = 6
i = 0  j = 0 k = 0  triple_float[i][j][k] = 0
i = 0  j = 0 k = 1  triple_float[i][j][k] = 0
i = 0  j = 0 k = 2  triple_float[i][j][k] = 0
i = 0  j = 1 k = 0  triple_float[i][j][k] = 0
i = 0  j = 1 k = 1  triple_float[i][j][k] = 0
i = 0  j = 1 k = 2  triple_float[i][j][k] = 0


During deallocation, the number of 'delete[]' calls was: (L2 * L1) + L1 + 1 == (2 * 1) + 1 + 1 = 4

The forumula comes from:

 (L2 * L1) --> the number of single pointers pointed to by
 each double pointer all need also to be deallocated memory.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer array needs also to be deallocated its memory of single pointers.

 + 1 --> the deallocation of memory for the single triple pointer which was storing an array of single pointers.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only points to a single array of normal data variable addresses;
 it is only necessary to call 'delete[]' once for all single pointers regardless of address size pointed to.



The program has finished execution....now exiting...thank you....
```

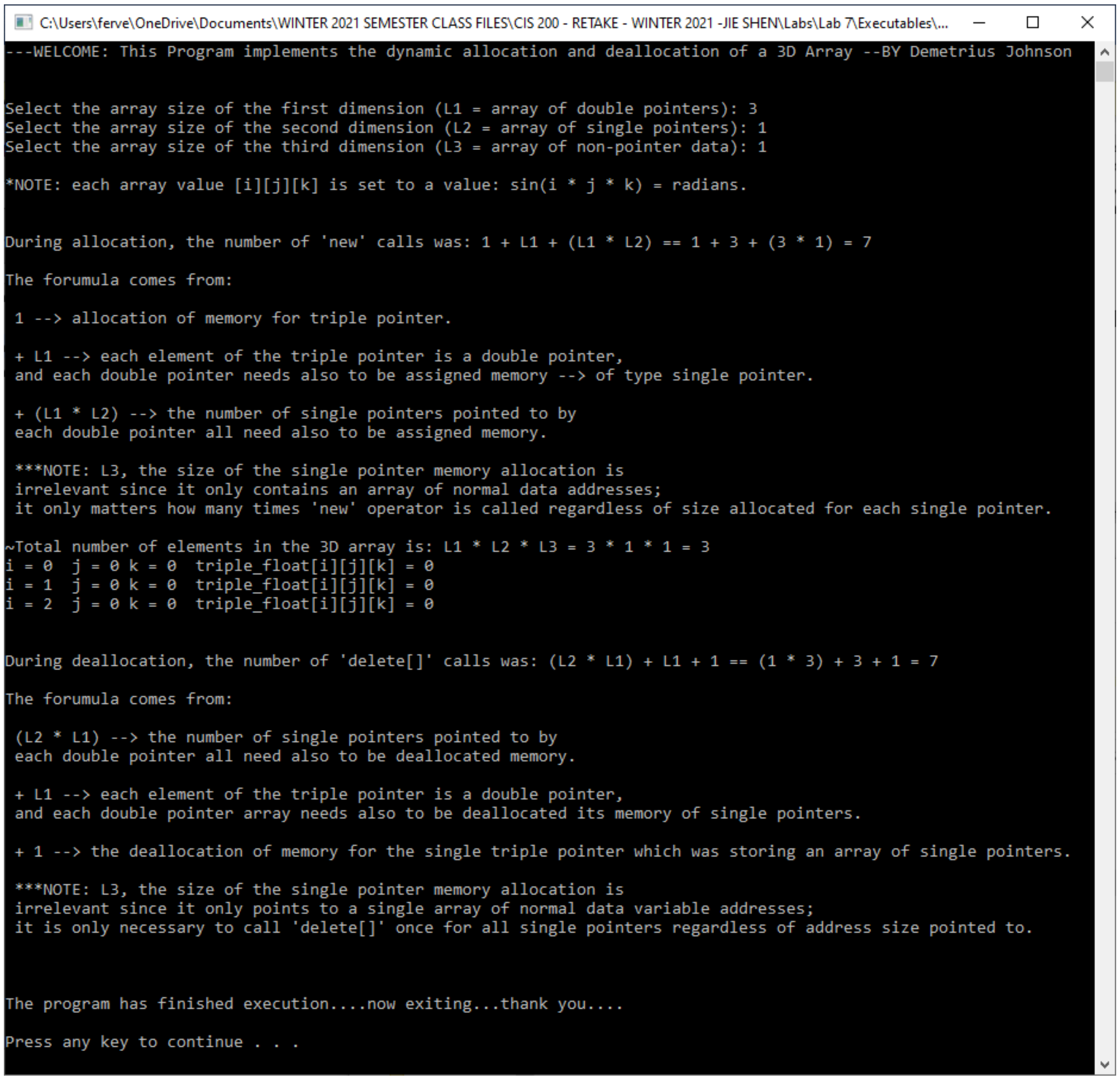
TEST 3:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q3...

```
---WELCOME: This Program implements the dynamic allocation and deallocation of a 3D Array --BY Demetrius Johnson


Select the array size of the first dimension (L1 = array of double pointers): 1
Select the array size of the second dimension (L2 = array of single pointers): 3
Select the array size of the third dimension (L3 = array of non-pointer data): 3

*NOTE: each array value [i][j][k] is set to a value: sin(i * j * k) = radians.


During allocation, the number of 'new' calls was: 1 + L1 + (L1 * L2) == 1 + 1 + (1 * 3) = 5

The forumula comes from:

 1 --> allocation of memory for triple pointer.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer needs also to be assigned memory --> of type single pointer.

 + (L1 * L2) --> the number of single pointers pointed to by
 each double pointer all need also to be assigned memory.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only contains an array of normal data addresses;
 it only matters how many times 'new' operator is called regardless of size allocated for each single pointer.

~Total number of elements in the 3D array is: L1 * L2 * L3 = 1 * 3 * 3 = 9
i = 0  j = 0 k = 0  triple_float[i][j][k] = 0
i = 0  j = 0 k = 1  triple_float[i][j][k] = 0
i = 0  j = 0 k = 2  triple_float[i][j][k] = 0
i = 0  j = 1 k = 0  triple_float[i][j][k] = 0
i = 0  j = 1 k = 1  triple_float[i][j][k] = 0
i = 0  j = 1 k = 2  triple_float[i][j][k] = 0
i = 0  j = 2 k = 0  triple_float[i][j][k] = 0
i = 0  j = 2 k = 1  triple_float[i][j][k] = 0
i = 0  j = 2 k = 2  triple_float[i][j][k] = 0


During deallocation, the number of 'delete[]' calls was: (L2 * L1) + L1 + 1 == (3 * 1) + 1 + 1 = 5

The forumula comes from:

 (L2 * L1) --> the number of single pointers pointed to by
 each double pointer all need also to be deallocated memory.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer array needs also to be deallocated its memory of single pointers.

 + 1 --> the deallocation of memory for the single triple pointer which was storing an array of single pointers.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only points to a single array of normal data variable addresses;
 it is only necessary to call 'delete[]' once for all single pointers regardless of address size pointed to.
```

TEST 4:



```
C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\...
---WELCOME: This Program implements the dynamic allocation and deallocation of a 3D Array --BY Demetrius Johnson


Select the array size of the first dimension (L1 = array of double pointers): 3
Select the array size of the second dimension (L2 = array of single pointers): 1
Select the array size of the third dimension (L3 = array of non-pointer data): 1

*NOTE: each array value [i][j][k] is set to a value: sin(i * j * k) = radians.


During allocation, the number of 'new' calls was: 1 + L1 + (L1 * L2) == 1 + 3 + (3 * 1) = 7

The forumula comes from:

 1 --> allocation of memory for triple pointer.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer needs also to be assigned memory --> of type single pointer.

 + (L1 * L2) --> the number of single pointers pointed to by
 each double pointer all need also to be assigned memory.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only contains an array of normal data addresses;
 it only matters how many times 'new' operator is called regardless of size allocated for each single pointer.

~Total number of elements in the 3D array is: L1 * L2 * L3 = 3 * 1 * 1 = 3
i = 0  j = 0 k = 0  triple_float[i][j][k] = 0
i = 1  j = 0 k = 0  triple_float[i][j][k] = 0
i = 2  j = 0 k = 0  triple_float[i][j][k] = 0


During deallocation, the number of 'delete[]' calls was: (L2 * L1) + L1 + 1 == (1 * 3) + 3 + 1 = 7

The forumula comes from:

 (L2 * L1) --> the number of single pointers pointed to by
 each double pointer all need also to be deallocated memory.

 + L1 --> each element of the triple pointer is a double pointer,
 and each double pointer array needs also to be deallocated its memory of single pointers.

 + 1 --> the deallocation of memory for the single triple pointer which was storing an array of single pointers.

 ***NOTE: L3, the size of the single pointer memory allocation is
 irrelevant since it only points to a single array of normal data variable addresses;
 it is only necessary to call 'delete[]' once for all single pointers regardless of address size pointed to.



The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

Question 4

Source code (USED C++ COMPILER on Microsoft Windows 10)

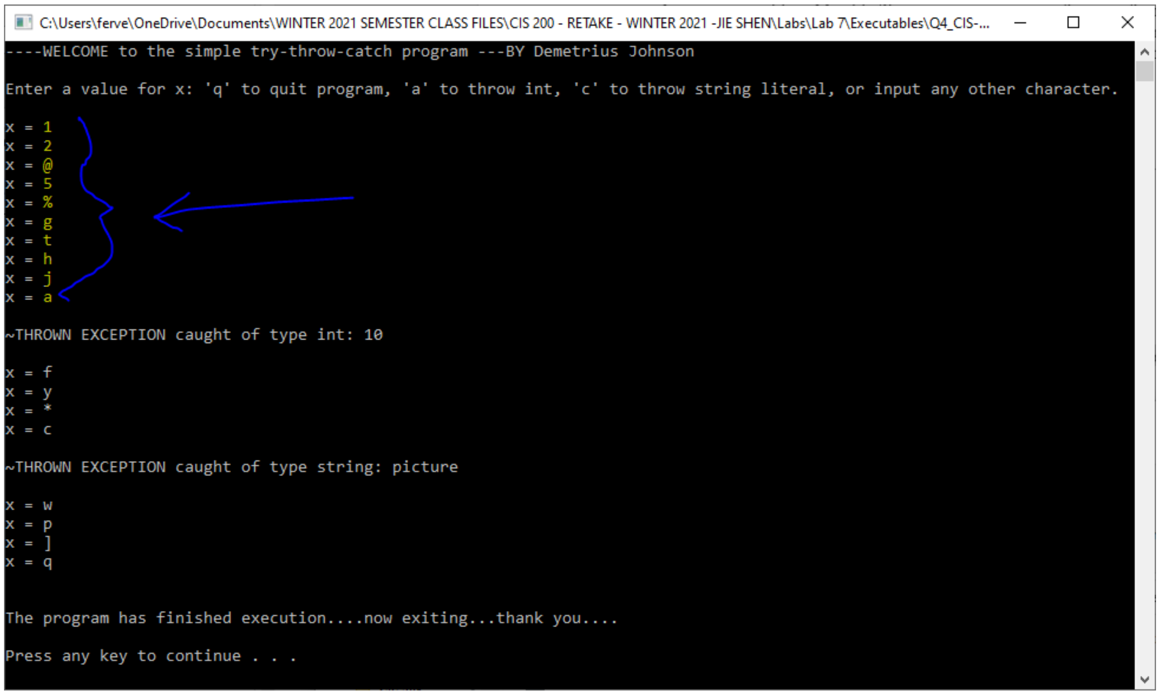
```
// CIS-200-LAB 7 -DemetriusJohnson.cpp : This file contains the 'main' function. Program execution
begins and ends there.
//

/*
//Author: Demetrius E Johnson
//Date: 19 MONTH 2021
//Last Modification Date: 03-21-2021
//Purpose: demontrate the use of try-catch-throw exception handling blcoks


*/

/*
Question 4:



*/




#include <iostream>
//#include <string>
//#include<stdexcept>
//#include<assert.h>
using namespace std;


//FUNCTION DECLARATIONS



//FUNCTION DECLARATIONS




int main()
{
        cout << "----WELCOME to the simple try-throw-catch program ---BY Demetrius Johnson\n\n";
        cout << "Enter a value for x: 'q' to quit program, 'a' to throw int, 'c' to throw string literal, or input
any other character.\n\n";

        char x9;

        cout << "x = ";
        cin >> x9;

        while (x9 != 'q'){

                try {
                        switch (x9)
                        {
                        case 'a':
                                throw 10; //throw an int literal
                                break;
                        case 'c':
                                throw "picture"; //throw a string literal
                                break;
                        }
                }

                // Write two exception handlers here. Inside each exception handler,
                // use cout to print a message
                catch (int x9_int_catch) {

                        cout << "\n~THROWN EXCEPTION caught of type int: " << x9_int_catch <<
endl << endl;

                }
                catch (const char* x9_string_catch) { //need to use const char* since a string literal is
techinically a const char[] array

                        cout << "\n~THROWN EXCEPTION caught of type string: " <<
x9_string_catch << endl << endl;

                }

                cout << "x = ";
                cin >> x9;

        }



  cout << endl << endl << "The program has finished execution....now exiting...thank you....\n\n";
  system("pause");

  return 0;
}


//FUNCTION DEFINITIONS BELOW THIS LINE
```

Test data and expected results

Test Table:

Test #	Valid / Invalid Data	Description of test	Input Value	Expected Output	Actual Output	Test Pass / Fail
1	valid	Show the use of various non exception characters	See screenshot	No exception to be throw. Keep looping	See screenshot	pass
2	valid	Throw an exception of type int	‘a’	Throw int	See screenshot	pass
3	valid	Throw an exception of type string literal (const char[])	‘c’	Throw const char[]	See screenshot	pass
4	valid	Show that q quits program	‘q’	Exit program	See screenshot	pass

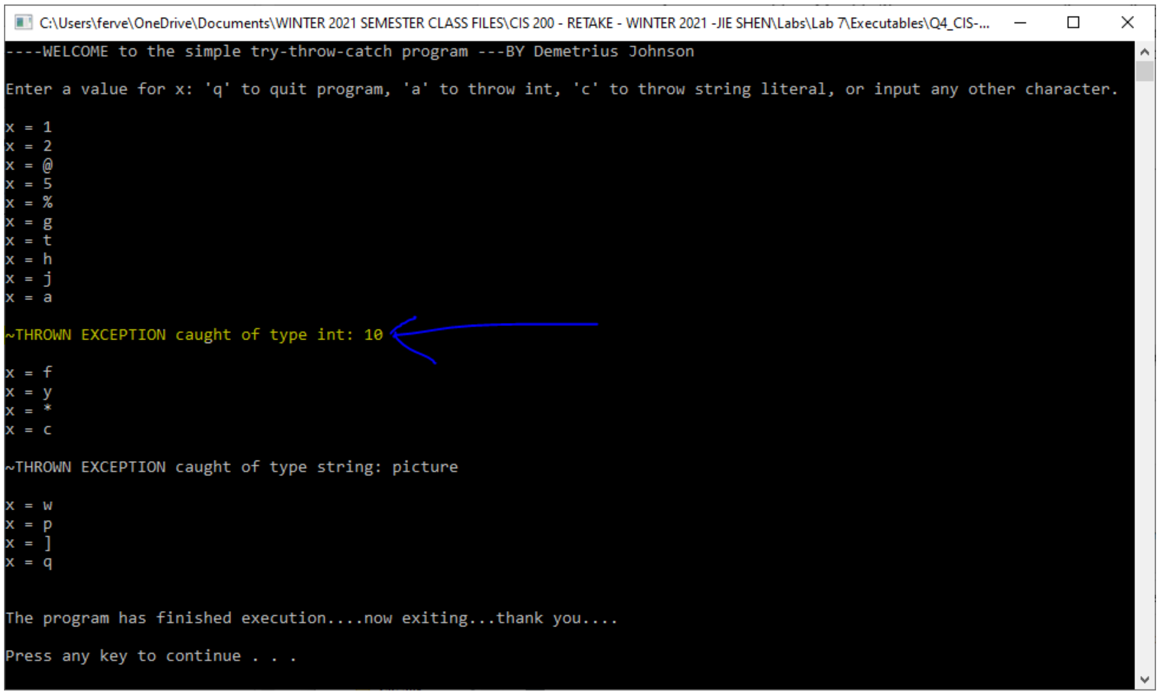
TEST 1:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q4_CIS-...

```
----WELCOME to the simple try-throw-catch program ---BY Demetrius Johnson

Enter a value for x: 'q' to quit program, 'a' to throw int, 'c' to throw string literal, or input any other character.

x = 1
x = 2
x = @
x = 5
x = %
x = g
x = t
x = h
x = j
x = a

~THROWN EXCEPTION caught of type int: 10

x = f
x = y
x = *
x = c

~THROWN EXCEPTION caught of type string: picture

x = w
x = p
x = ]
x = q


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

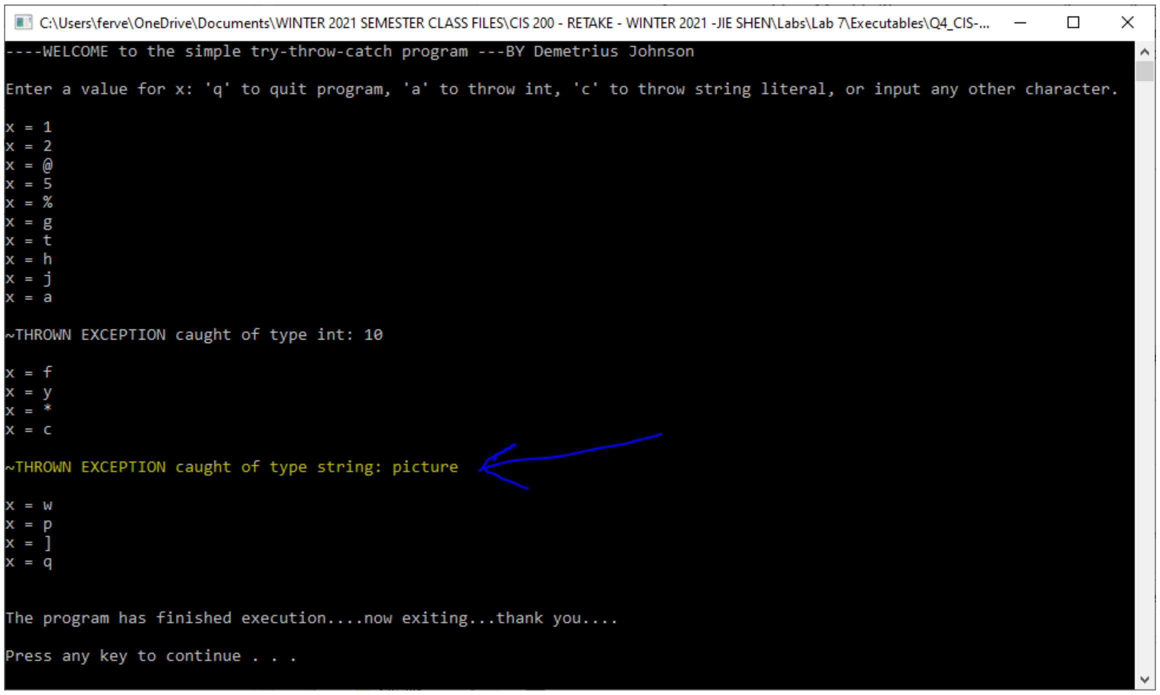
TEST 2:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q4_CIS-...

```
----WELCOME to the simple try-throw-catch program ---BY Demetrius Johnson

Enter a value for x: 'q' to quit program, 'a' to throw int, 'c' to throw string literal, or input any other character.

x = 1
x = 2
x = @
x = 5
x = %
x = g
x = t
x = h
x = j
x = a

~THROWN EXCEPTION caught of type int: 10

x = f
x = y
x = *
x = c

~THROWN EXCEPTION caught of type string: picture

x = w
x = p
x = ]
x = q


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

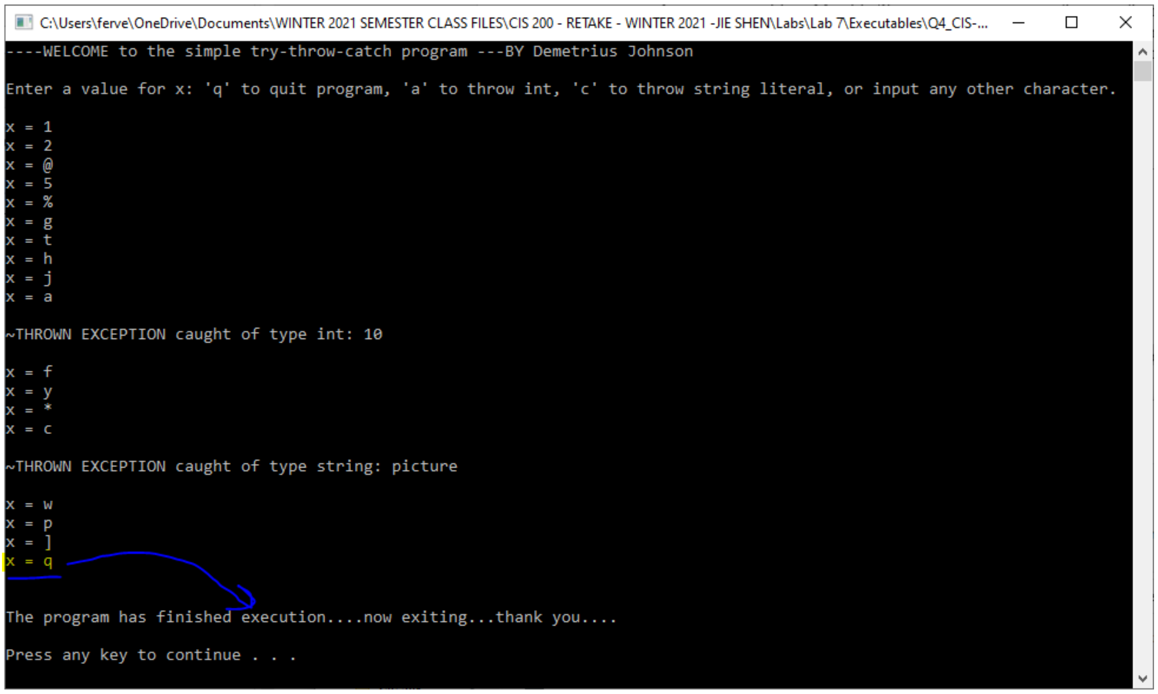
TEST 3:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q4_CIS-...

```
----WELCOME to the simple try-throw-catch program ---BY Demetrius Johnson

Enter a value for x: 'q' to quit program, 'a' to throw int, 'c' to throw string literal, or input any other character.

x = 1
x = 2
x = @
x = 5
x = %
x = g
x = t
x = h
x = j
x = a

~THROWN EXCEPTION caught of type int: 10

x = f
x = y
x = *
x = c

~THROWN EXCEPTION caught of type string: picture

x = w
x = p
x = ]
x = q


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

TEST 4:



C:\Users\ferve\OneDrive\Documents\WINTER 2021 SEMESTER CLASS FILES\CIS 200 - RETAKE - WINTER 2021 -JIE SHEN\Labs\Lab 7\Executables\Q4_CIS-...

```
----WELCOME to the simple try-throw-catch program ---BY Demetrius Johnson

Enter a value for x: 'q' to quit program, 'a' to throw int, 'c' to throw string literal, or input any other character.

x = 1
x = 2
x = @
x = 5
x = %
x = g
x = t
x = h
x = j
x = a

~THROWN EXCEPTION caught of type int: 10

x = f
x = y
x = *
x = c

~THROWN EXCEPTION caught of type string: picture

x = w
x = p
x = ]
x = q


The program has finished execution....now exiting...thank you....

Press any key to continue . . .
```

Submission

Provide an MS word document or a pdf file with the following information:

- Machine type (Unix, Mac, Linux or PC machine ?)
- Compiler type
- Description of your code design and implementation
- Inclusion of your source
- A reasonable number of comment lines in your source code
- Screen shot of your test run