**Lab 01: Pointers and malloc**

Name: **Raja Hasnain Anwar**

Registration: **192135**

Date: **11th Sept, 2017**

**Link:** https://github.com/rhasnainanwar/DSAlab/tree/master/Lab01\_pointers\_malloc

# Task # 1:

## Code:

#include<iostream>

using namespace std;

int main(void)

{

int \*salary = new int[20];

int i;

for (i = 0; i<20; ++i)

{

cout << "Enter Salary: ";

cin >> \*(salary + i);

}

for (i = 0; i < 20; ++i){

\*(salary + i) = \*(salary + i) + \*(salary + i) / (i + 1);

cout << \*(salary + i) << endl;

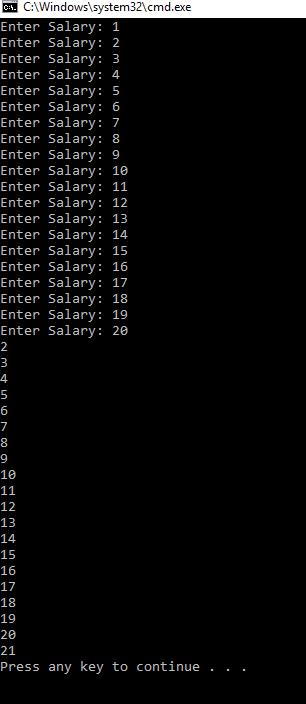
}

delete [] salary; //clear memory

return 0;

}

## Output:



# Task #2:

## Code:

/\*

\* Kindly fill this information.

\* Name: Raja Hasnain Anwar

\* Student #: 192135

\*/

#include <iostream>

using namespace std;

/\*

\* PROBLEM #2: Analyze pointers

\*/

/\*

\* PART #1: Write a function that does two things:

\* >> Write the memory location pointed by the pointer to the console.

\* >> Write the value of the integer (which the pointer points to) to the console.

\*/

void analyze\_pointer(int \*ptr)

{

cout << "Pointer: " << ptr << endl;

cout << "Value: " << \*ptr << endl;

}

/\*

\* PROBLEM #3: Add a couple more functions to the mix

\* >> call them int\_pointer1 and int\_pointer2

\* >> both will return int pointers.

\*/

/\*

\* Follow the steps:

\* >> the function will allocate an int on the heap (via new int)

\* >> assign a value to it

\* >> and return that value (an int pointer type).

\*/

int\* int\_pointer1()

{

int \*ptr = new int;

\*ptr = 425;

return ptr;

}

/\*

\* Follow the steps:

\* >> the function will allocate an int on the stack (via "int iValue;" or something similar)

\* >> assign a value to it

\* >> and return its memory location (an int pointer type).

\*/

int\* int\_pointer2()

{

int something = 435;

return &something;

}

int main()

{

/\*

\* PART #2: Use the analyze\_pointer function to complete two TASKS

\*/

/\*

\* TASK #1

\* >> allocate an int on the stack (e.g., "int iValue;")

\* >> assign a value to it

\* >> get its memory location (with the reference operator---&)

\* >> and pass this value to analyze\_pointer.

\*/

cout << "Part #2: Memory on stack..." << endl;

// Add your code here!

int iValue; //declared

iValue = 12; //assign

int \*ptr = &iValue; //memory location

analyze\_pointer(ptr); //pass

/\*

\* TASK #2

\* >> allocate an int on the heap (with the new operator)

\* >> assign a value to it

\* >> and pass it to analyze\_pointer.

\*/

cout << "Memory on heap..." << endl;

// Add your code here!

int \*new\_int; //for heap

new\_int = new int; //in heap

\*new\_int = 65; //assign

analyze\_pointer(new\_int); //pass

delete new\_int; //clear mem

/\*

\* PROBLEM #3: Call analyze\_pointer on the return of functions int\_pointer1() and int\_pointer2()

\*/

cout << "Memory on heap versus on stack..." << endl;

int \*temp = int\_pointer1();

analyze\_pointer(temp);

analyze\_pointer(int\_pointer2());

/\*

\* Remember that int was allocated on the heap for function "int\_pointer1()"

\* Delete it

\* and call analyze\_pointer on it after deleting it.

\*/

cout << "Memory on heap after delete..." << endl;

delete temp;

analyze\_pointer(temp);

/\*

\* Call: analyze\_pointer(new int);

\*/

// Add your code here!

analyze\_pointer(new int);

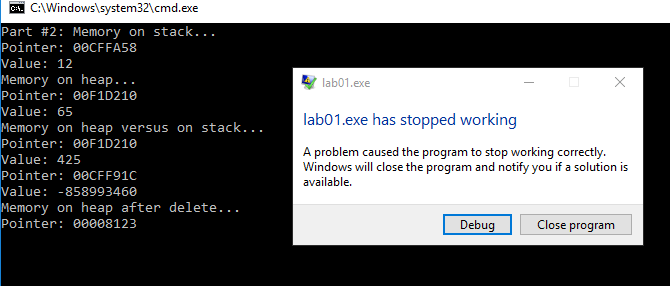
return 0;

}

## Q/A:

1. In first case, the variable is declared on stack. Then a value is assigned to it. By using &, we pass it’s address to the analyze\_pointer() function. There, first the address which is passed is printed, then using dereferencing, the value at that address is printed.  
   In second case, a pointer is declared using new keyword which declares a variable on heap and gives its address. Now we have only the address. Again using dereferencing, the value at the given address is printed.
2. int\_pointer1() returns a pointer to a heap memory location. This means, the variable will still be there even after the function exists. For this reason, we are still able to get its value.  
   On contrary, int\_pointer2() declares a variable on stack which will vanish from memory when the function exists. For this reason, we get a garbage value for that variable because the required value is no longer on that location.
3. Every time a variable is dynamically declared in heap using new keyword, it is a leak threat. That memory is allocated until freed. Hence, heap variables must be deleted or freed. There are three heap variables but one was deleted as a task hence two memory leaks. Although, I have freed them while doing the task.

## Output:



# Task #3:

## Code:

/\*

\* Kindly fill this information.

\* Name:

\* Student #:

\*/

#include <iostream>

#include <string>

using namespace std;

/\*

\* Define a struct Area that has two private variable members;

\* units of type string and area\_value of type float.

\*/

class Area

{

private:

string units;

float area\_value;

public:

void set\_units(string u){

units = u;

}

void set\_area(float area){

area\_value = area;

}

string get\_units(){

return units;

}

float get\_area(){

return area\_value;

}

};

int main()

{

/\*

\* Modify program to create a dynamic variable of type Area.

\*/

Area \*area1 = new Area;

/\*

\* Input from the keyboard the area\_value and its units.

\*/

float area\_value;

cout << "Enter area value: ";

cin >> area\_value;

(\*area1).set\_area(area\_value);

string units;

cout << "Enter Units: ";

cin >> units;

(\*area1).set\_units(units);

/\*

\* Compute one-half and one-quarter of the area

\* and display the results

\*/

cout << (\*area1).get\_area() / 2.0 << " " << (\*area1).get\_units() << endl;

cout << (\*area1).get\_area() / 4.0 << " " << (\*area1).get\_units() << endl;

/\*

\* Destroy the dynamic variable at the end

\*/

delete area1;

return 0;

}

## Output:

