Mapúa University

School of EECE

Department of Computer Engineering

Lab 1- Pointers and Dynamic Memory Allocation

Santos, Joaquin S. II

2018130832

CPE104L (Data Structures and Algorithms)

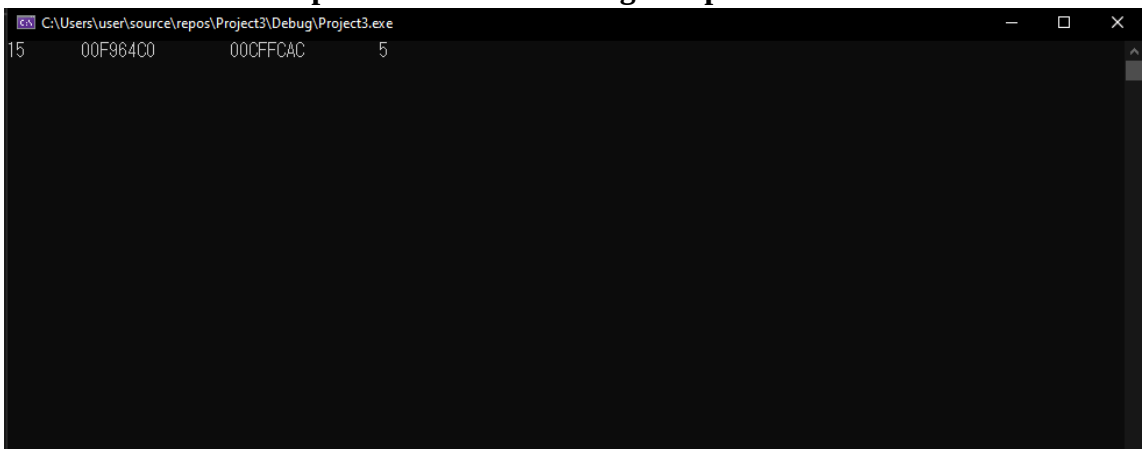
C2

II. Answer To Question

How is the array allocated in Sample Code 1?

We could see that in sample code 1, the program asks the user input. This is where dynamic memory allocation takes place. As we could see, the user is the one that set values to each element of the array. Starting from 0 to 9.

Illustrate/show the output seen after executing Sample Code 2



The output shows the value of two variables, and the address of two variables.

From Sample Code 2, discuss how the program arrived at such an output.

Since *p has been assigned the value of 15, and r has been assigned to the address of s, in the cout command, we could see that it is written as *x (*p) which gives us the value of p. and then member has been assigned the value of *z (or the value of s) which gives us the value of 5. And when the cout command showed &member, it shows the address of the variable s.

For the variable y, we can see that q from new int, became an array and has assigned the value of 4 to the 5^{th} element of q (y). now in the cout command, it is written as y and not *y. therefore it also showed its address instead of its value.

Try to include arithmetic operations to manipulate pointer variables. What are the possible operations allowed?

The possible operations would be to increment, decrement, and to compare (==, >, <) two pointers of the same type.

What are the major purposes of having the new and the delete operators?

The main purpose for these operators is mainly for memory management. Specifically used for dynamic allocation. For the new operator, when memory is available, it will initialize its memory and will return the address of a newly allocated memory. For the

delete operator, it would destroy and array or a pointer which was created by the new operator.

6. Given the declarations and assignment,
int nums[15];
int *pt;
pt = &nums;
is the code above invalid or not? If invalid, correct the given code
The code is invalid, because the value of &nums cannot be assigned to pt. To fix this code, we just need to remove the ampersand.

```
#include <iostream>

int main()
{
    int nums[15];
    int *pt;
    pt = nums;

    return 0;
}
```

Can pointers be considered an efficient and semantic-wise data structure? Why or why not?

Yes, pointers can be considered an efficient and semantic-wise data structure. Lets say for example you would want to pass an array to a function or class. You cannot pass the values of an array as efficient as using a pointer. With pointers, you can access the whole array list since you can go from *(array+i) or array++. Incrementing the array will shift it to the next address. Unlike when you use array[i] or array[0], you are only limited to one element.

III. Screenshots of Experiment Output

Sample Code 1

```
PS C:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L\Laboratory 1\sample codes> & .\"drill1.exe"
Enter input 1:1
Enter input 2:2
Enter input 3:3
Enter input 4:4
Enter input 5:5
Enter input 6:6
Enter input 7:7
Enter input 8:8
Enter input 9:9
Enter input 10:10
1
2
3
4
5
6
7
8
9
10
PS C:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L\Laboratory 1\sample codes>
```

Sample Code 2

```
C:\Users\user\source\repos\Project3\Debug\Project3.exe
15      00F964C0        00CFFCAC        5
```

Sample Code 3

```
Enter the radius: 65
Area:13273.2
Circumference:408.407
C:¥Users¥user¥source¥repos¥Project3¥Debug¥Project3.exe (process 24812) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

IV. Answer To Program Exercise

1. Create a program that has a declaration in main( ) to store the string " I'm so great! " in an array named message. Include a member function of any name from your defined class that accepts the message in an argument named strng, then displays the contents of message using the pointer notation *(strng + i).

```
#include <iostream>

class labExercise1
{
    public:
        void displayMessage(std::string *strng)
        {
            for(int i=0;i<sizeof(strng);i++)
            {
                std::cout<<*(strng+i)<<" ";
            }
        }
};


int main()
{
    labExercise1 exercise;
    std::string message[3]={"I'm","so","great!"};
    exercise.displayMessage(message);
    return 0;
}
```

```
PS C:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L> cd "c:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L\Laboratory 1\Lab 1-1"
PS C:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L\Laboratory 1\Lab 1-1> & .\"Laboratory1_Santos_II.exe"
I'm so great!
```

V. Discussion of Report

For this experiment, the researchers are required to use pointers and dynamic memory allocation. Pointers are variables which take the address of the memory. For example, if we have an array arr[5]={1,2,3,4,5}the value 1 will have a different address from 2,3,4, and 5. In this case: if we use **int p=*arr ,** this means the variable **p** will be getting the values of **arr** in a specific address. If we use **&p** this will show use the memory address instead.

```
#include <iostream>

int main()
{
    int arr[5]={1,2,3,4,5}, p=*arr;
    std::cout<<"Memory Address: ";
    for(int i=0;i<5;i++)
    {
        std::cout<<&p+i<<" ";
    }
    std::cout<<std::endl<<"Actual Values: ";
    for(int i=0;i<5;i++)
    {
        std::cout<<arr[i]<<" ";
    }
    return 0;
}
```

OUTPUT:

```
Memory Address: 0x61fdfc 0x61fe00 0x61fe04 0x61fe08 0x61fe0c
Actual Values: 1 2 3 4 5
```

As we could see, each element of the array a different address. Pointers help with a more efficient way on addressing elements on an array. Dynamic allocation is when values are assigned to an array during the execution time. Let's take this code for example.

```
#include <iostream>
int main()
{
    int arr[5]={0};
    for(int i=0;i<5;i++)
    {
        std::cout<<"Enter value for ["<<i<<"] : ";
```

```
        std::cin>>arr[i];
    }
    std::cout<<"The new array values are: ";
    for(int i=0;i<5;i++)
        std::cout<<arr[i]<<" ";
}
```

OUTPUT:

```
Enter value for [0] : 2
Enter value for [1] : 3
Enter value for [2] : 6
Enter value for [3] : 4
Enter value for [4] : 10
The new array values are: 2 3 6 4 10
PS C:\Users\user\Desktop\MAIN MENU\Assignments\2nd Years\CPE104L\Laboratory 1\sample codes>
```

As we could see, we assigned values to the array while we were running the code. This is the most basic concept of dynamic memory allocation.

VI. Conclusion

As we could see, dynamic memory allocation and pointers can be used to make the code more efficient and optimized. Just like how pointers can speed the process of going to an element. Instead of looping from a[0] to a[4], we can just directly go to it by using *(a+4). And for dynamic allocation, it would make our code more flexible.