UNIVERSITY OF ENGINEERING AND TECHNOLOGY

(NAROWAL CAMPUS)

Object-Oriented Programming Lab Manual

Created by: Muhammad Abdullah

Registration Number: 2022-CS-525

Topics: Pointers, 1-D Array using pointers,

2-D Array using double pointers, Recursion Functions,

Structures, Memory Leak, Dangling Pointer,

Lab Manual

(Object-Oriented Programming Lab)

- **Pointers:**

Pointer variables contain memory addresses as their values. Normally, a variable directly contains a specific value. A pointer contains the memory address of the variable that, in turn, contains a specific value. A variable name directly references the value and a pointer indirectly references the value. Referencing a value through a pointer is called indirection. The declaration

```
int* ptr,
```

declares the variable ptr to be of type int* and is read, "ptr is a pointer to int". The * in the declaration applies only to the first variable. Each variable being declared as a pointer must be preceded by an asterisk (*). When * appears in a declaration, it's not an operator; rather, it indicates that the variable being declared is a pointer.

Pointer Operators:

The unary operators & and * are used to create pointer values and "dereference" pointers. The & is the address operator and * is the indirection operator.

Address (&) Operator:

The address operator is a unary operator that obtains the memory address of its operand. The statement

```
int* ptr = &y;
```

declares a pointer and assigns the address value of the integer variable y to the pointer variable ptr. The variable ptr is said to "point to" y. Now, ptr indirectly references the variable y value.

Indirection (*) Operator:

The unary * operator, commonly called the indirection operator or dereferencing operator, returns a value representing the object to which its pointer operand points. The statement

```
cout<<*ptr<<endl;
```

displays the value of variable y. Using * in this manner is called dereferencing a pointer. A dereferenced pointer may also be used as a value on the left side of an assignment, as in

```
*ptr = 9;
```

which would assign 9 to the variable y. *ptr is an alias for y. The dereferenced pointer may also use to receive an input value as in

```
cin>>*ptr;
```

which places the input value in y.

Uses of Pointer:

Here are some common uses:

1. Accessing array elements
2. Passing arguments to a function when the function needs to modify the original argument
3. Passing arrays and strings to a function
4. Obtaining memory from the system
5. Creating and manipulating data structures that can grow and shrink such as linked lists, queues, stacks, and trees

- ## Dynamic Memory Allocation:

Dynamic memory allocation means we allocate or delete memory at the runtime or execution time. The dynamically allocated memory is kept on the heap which is also known as the free store. Declarations are used for statically allocate memory but for dynamic allocation we use the new operator. The statement

```
int* ptr = new int;
```

declares a pointer ptr of int and allocates memory for int. The ptr pointer variable points to memory which is dynamically allocated. By using pointers in such a manner we access, allocate, deallocate, and manipulate the data of dynamically allocated memory. There is also an operator which is used to delete dynamically allocated memory. The delete operator frees up the space in memory which is pointed by a pointer. The delete operator doesn't delete the memory which is dynamically allocated and we make our pointer point to the NULL or some other value. Since the memory pointed is no longer available, such a pointer is said to be a dangling pointer. We should return our memory to the heap before making dangling our pointer. When the memory is allocated from the heap but not returned to the heap using the delete operator such a process is known as a memory leak.

The difference between Dynamic Memory Allocation and Static Memory Allocation is given in below table:

Static Memory Allocation	Dynamic Memory Allocation
Also called compile time allocation	Also called run-time allocation
The size and location where the variable will be stored are fixed during compile time	Memory requirements should be defined during the execution of the program
Both allocation and deallocation of memory are done by the compiler itself	The programmer needs to write a proper code for allocation and deallocation
Slightly faster	Slightly slower
Memory allocation on the stack	Memory allocation on the heap

- ## Recursion:

  Recursion is a programming technique that involves a function calling itself. The function which calls itself during its execution is known as Recursive Function. Every recursive function must be provided with a way to end the recursion. Otherwise, it will call itself forever and crash the program. We should provide some base condition in the function body that stops recursion at some point. The base condition plays an important role and stops execution. Deeply nested recursion creates many stored variables which are initialized on function call, which can pose a problem to the system if it doesn't have enough space.

- ## Structure:

  A structure is a collection of simple variables. The variables in a structure can be of different data types. The data items in a structure are called the members of the structure. Simply a structure is a collection of data. We can access the members of structure variables using the dot operator (.) which is also known as the member accessing operator. Structure members are treated just like other variables. We can also nest structures within other structures but in the case of nested structures we use the dot operator the number of times according to the depth of the structure nesting.

- ## Programs:

  Task 1: Create a 1-D array using pointers in Dynamic Memory

  Task 2: Create a 2-D array using pointers in Dynamic Memory

  Task 3: Write a program of Dangling Pointer

  Task 4: Write a program for Memory Leak

  Task 5: Write a Program of Recursive Function that reverse an array

  Task 6: Create a program that takes the values of structure members and prints the value

Task 1 Program:

```cpp
#include <iostream>
using namespace std;
int main(){
    int length;
    cout<<"Enter the length of Array: ";
    cin>>length;
    int* ptr = new int[length];
    cout<<"  --: Enter Array Elements :--  "<<endl;
    for(int i =0; i<length;i++){
        cout<<"Enter the "<<i+1<<" element of array: ";
        cin>>*(ptr+i);
    }
    cout<<" --: The Array Elements :-- "<<endl;
    for(int i =0; i<length ; i++){
        cout<<"The "<<i+1<<" elements of array is "<<*(ptr+i)<<endl;
    }
    delete[] ptr;
    return 0;
}
```

Task 1 Output:

```
Enter the length of Array: 5
  --: Enter Array Elements :--
Enter the 1 element of array: 901
Enter the 2 element of array: 130
Enter the 3 element of array: 463
Enter the 4 element of array: 947
Enter the 5 element of array: 425
 --: The Array Elements :--
The 1 elements of array is 901
The 2 elements of array is 130
The 3 elements of array is 463
The 4 elements of array is 947
The 5 elements of array is 425

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 12620) 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 . . .
```

Task 2 Program:

```cpp
#include <iostream>
using namespace std;
int main(){
    int rows, columns;
    cout<<"Enter the rows length: ";
    cin>>rows;
    cout<<"Enter the columns length: ";
    cin>>columns;
    int **matrix = new int*[rows];
    for(int i=0;i<rows;i++){
        matrix[i] = new int[columns];
    }
    cout<<" --: Enter Matrix Elements :-- "<<endl;
    for(int i=0;i<rows;i++){
        for(int j=0;j<columns;j++){
            cout<<"Enter the element at ("<<i+1<<","<<j+1<<") : ";
            cin>>matrix[i][j];
        }
    }
    cout<< " --: The Array Elements :--"<<endl;
    for(int i=0;i<rows;i++){
        for(int j=0;j<columns;j++){
            cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
    }
    for(int i =0; i<rows;i++){
        delete[] matrix[i];
    }
    delete[] matrix;
    return 0;
}
```

Task 2 Output:

```
Enter the rows length: 2
Enter the columns length: 3
 --: Enter Matrix Elements :--
Enter the element at (1,1) : 12
Enter the element at (1,2) : 22
Enter the element at (1,3) : 32
Enter the element at (2,1) : 13
Enter the element at (2,2) : 23
Enter the element at (2,3) : 33
 --: The Array Elements :--
12 22 32
13 23 33

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 14184) 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 . . .
```

Task 3 Program:

```cpp
#include <iostream>
using namespace std;
int main(){
    int* arr = new int[6];
    int* ptr = &arr[5];
    delete[] arr;
    cout<<*ptr<<endl;
    return 0;
}
```

Task 3 Output:

```
-572662307

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 12256) 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 . . .
```

Task 4 Program:

```cpp
#include <iostream>
using namespace std;
int main(){
    int* ptr = new int;
    *ptr = 10;
    ptr = new int;
    *ptr = 20;
    cout<<*ptr;
    delete ptr;
    return 0;
}
```

Task 4 Output:

```
20
D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 16492) 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 . . .
```

Task 5 Program:

```
#include <iostream>
using namespace std;
void reverseArray(int [], int,int);
int main(){
    int size = 5;
    int arr[5] = {1,2,3,4,5};
    for(int i=0; i<size;i++){
        cout << arr[i] << "\t";
    }
    cout<<endl;
    reverseArray(arr, 0, size-1);
    for(int i=0; i<size;i++){
        cout << arr[i] << "\t";
    }
    return 0;
}
void reverseArray(int array[], int start, int end){
    if(start >= end){
        return;
    }
    int temp = array[start];
    array[start] = array[end];
    array[end] = temp;
    reverseArray(array, start+1, end-1);
}
```

Task 5 Output:

```
1       2       3       4       5
5       4       3       2       1
D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 4640) 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 . . .
```

Task 6 Program:

```cpp
#include <iostream>
using namespace std;
struct Person{
    string name;
    int age;
    double height;
    string origin;
};
int main(){
    Person person1;
    cout<<"Enter the name of person: ";
    cin>>person1.name;
    cout<<"Enter the age of person: ";
    cin>>person1.age;
    cout<<"Enter the height of person: ";
    cin>>person1.height;
    cout<<"Enter the origin of person: ";
    cin>>person1.origin;
    cout<<"The name of person is: "<<person1.name<<endl;
    cout<<"The age of person is: "<<person1.age<<endl;
    cout<<"The height of person is: "<<person1.height<<endl;
    cout<<"The origin of person is: "<<person1.origin<<endl;
    return 0;
}
```

Task 6 Output:

```
Enter the name of person: Abdullah
Enter the age of person: 18
Enter the height of person: 5.9
Enter the origin of person: Pakistan
The name of person is: Abdullah
The age of person is: 18
The height of person is: 5.9
The origin of person is: Pakistan

D:\UET Narowal\2nd Semester\Object Oriented Programming\Lab\OOP Lab\x64\Debug\OOP Lab.exe (process 18376) 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 . . .
```
