

Data Structures-LAB



Lab Manual # 01 & 02

OOP & Pointers in C++

Instructor: Hilal Ahmad

Semester Fall-2024

Course Code: CL2001

**Fast National University of Computer and Emerging
Sciences Peshawar**

**Department of Computer Science
& Software Engineering**

Data Structures-LAB

Table of Contents

Classes and Objects.....	1
Pointers	7
Pointers and Arrays.....	11
Passing Pointers as Arguments to Functions	15
Returning Pointers from Function.....	17
Pointers and Strings	19

Lab # 01

Classes and Objects

- **Definition:**
 - **Class:** A blueprint for creating objects. It defines a data structure, including variables (data members) and functions (member functions) that operate on the data.
 - **Object:** An instance of a class. It represents a specific entity that can use the class's methods and properties.
 - Classes encapsulate data for the object.
 - Objects are instances of classes; they are created using the class as a template.
- **Example:**

```
1 #include <iostream>
2 using namespace std;
3
4 class Car {
5 public:
6     string make;
7     string model;
8     int year;
9
10    void start() {
11        cout << "Car started!" << endl;
12    }
13
14    void displayInfo() {
15        cout << "Make: " << make << ", Model: " << model << ", Year: " << year << endl;
16    }
17 };
18
19 int main() {
20     Car myCar;
21     myCar.make = "Toyota";
22     myCar.model = "Corolla";
23     myCar.year = 2020;
24
25     myCar.displayInfo();
26     myCar.start();
27
28     return 0;
29 }
30
```

Explanation:

- The `Car` class defines a blueprint for a car object. It includes:
 - **Data Members:** `make`, `model`, and `year`—these represent the properties of a car.
 - **Member Functions:** `start()` and `displayInfo()`—these define the behaviors or actions that a car object can perform.
- **Object Creation:**
 - An object named `myCar` is created from the `Car` class. This object can then use the data members and member functions defined in the class.
 - **Accessing Members:**

- Data members are accessed using the dot operator (e.g., `myCar.make = "Toyota";`)
- Member functions are also accessed using the dot operator (e.g., `myCar.start();`).

Constructors

- **Definition:**
 - A special member function of a class that initializes objects of that class. It is called automatically when an object is created.
 - Constructors have the same name as the class and no return type.
 - They can be overloaded to provide multiple ways to initialize objects.
- **Example:**

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

class Car {
public:
    string make;
    string model;
    int year;

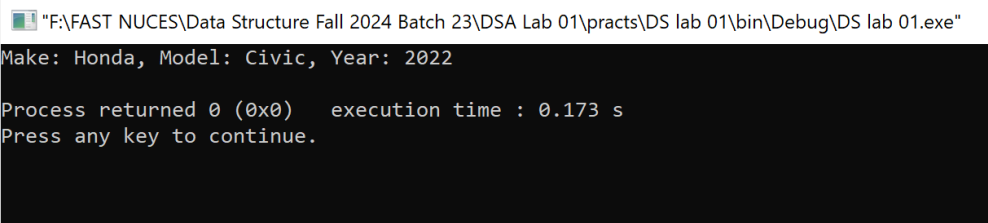
    // Constructor
    Car(string m, string mo, int y) {
        make = m;
        model = mo;
        year = y;
    }

    void displayInfo() {
        cout << "Make: " << make << ", Model: " << model << ", Year: " << year << endl;
    }
};

int main() {
    Car myCar("Honda", "Civic", 2022);

    myCar.displayInfo();

    return 0;
}
```



Explanation:

- A constructor is a special function that is automatically called when an object is created.
- In the modified Car class, the constructor Car(string m, string mo, int y) initializes the data members (make, model, and year) when an object is instantiated.
- When the myCar object is created using Car myCar("Honda", "Civic", 2022);, the constructor is called, and the object's data members are set to "Honda", "Civic", and 2022.

Encapsulation

• Definition:

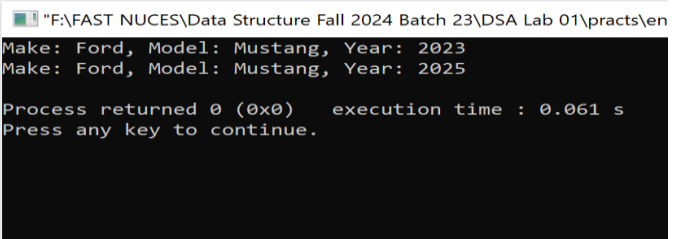
- Encapsulation is the bundling of data and methods that operate on that data within a single unit (class). It restricts direct access to some of the object's components.
- **Private:** Members are accessible only within the class.
- **Public:** Members are accessible from outside the class.
- Encapsulation helps protect the internal state of the object and allows control over how data is accessed or modified.

• Example:

```

.cpp X
1  #include <iostream>
2  using namespace std;
3
4  class Car {
5  private:
6      string make;
7      string model;
8      int year;
9
10 public:
11     Car(string m, string mo, int y) {
12         make = m;
13         model = mo;
14         year = y;
15     }
16
17     void displayInfo() {
18         cout << "Make: " << make << ", Model: " << model << ", Year: " << year << endl;
19     }
20
21     void setYear(int y) {
22         if(y > 1885) { // The first car was made in 1886
23             year = y;
24         } else {
25             cout << "Invalid year!" << endl;
26         }
27     }
28 };
29
30 int main() {
31     Car myCar("Ford", "Mustang", 2023);
32
33     myCar.displayInfo();
34     myCar.setYear(2025);
35     myCar.displayInfo();
36
37     return 0;
38 }
39

```



```

"F:\FAST NUCES\Data Structure Fall 2024 Batch 23\DSA Lab 01\practs\en
Make: Ford, Model: Mustang, Year: 2023
Make: Ford, Model: Mustang, Year: 2025

Process returned 0 (0x0)   execution time : 0.061 s
Press any key to continue.

```

Explanation:

- Encapsulation is the concept of wrapping data (variables) and code (functions) together as a single unit. In C++, this is achieved using classes.
- In the `Car` class, data members like `make`, `model`, and `year` are set to `private` to protect them from being directly accessed from outside the class.
- **Private** members can only be accessed by member functions of the class. In the example, `year` is updated using the `setYear()` function.
- **Public** members can be accessed from outside the class. In the example, `displayInfo()` is a public function that allows controlled access to display the car's information.

Inheritance

- **Definition:**
 - Inheritance allows a new class (derived class) to inherit properties and behavior from an existing class (base class).
 - Promotes code reusability by allowing new classes to use existing code.
 - The derived class can add new members or override existing ones from the base class.
- **Example:**

```

re X main.cpp X
3     model = mo;
4     year = y;
5 }
6
7 void displayInfo() {
8     cout << "Make: " << make << ", Model: " << model << ", Year: " << year <
9 }
10
11 void setYear(int y) {
12     if(y > 1885) { // The first car was made in 1886
13         year = y;
14     } else {
15         cout << "Invalid year!" << endl;
16     }
17 }
18 };
19
20 class SportsCar : public Car {
21 private:
22     int topSpeed;
23
24 public:
25     SportsCar(string m, string mo, int y, int ts) : Car(m, mo, y) {
26         topSpeed = ts;
27     }
28
29     void displayInfo() {
30         Car::displayInfo();
31         cout << "Top Speed: " << topSpeed << " km/h" << endl;
32     }
33 };
34
35 int main() {
36     SportsCar mySportsCar("Ferrari", "F8", 2023, 340);
37
38     mySportsCar.displayInfo();
39
40     return 0;
41 }

```

"F:\FAST NUCES\Data Structure Fall 2024 Batch 23\DSA Lab 01\practs\in
 Make: Ferrari, Model: F8, Year: 2023
 Top Speed: 340 km/h
 Process returned 0 (0x0) execution time : 0.128 s
 Press any key to continue.

Code::Blocks X Search results X Cccc X
 Variable: PATH=C:\Program Files\CodeBlocks\N
 s\System32\WindowsPowerShell\v1.0;C:\Wind
 Data\Local\Programs\Python\Python37;C:\Us

Explanation:

- Inheritance allows a new class to inherit attributes and methods from an existing class. This promotes code reuse and a hierarchical relationship between classes.
- The SportsCar class inherits from the Car class, meaning it has access to all public and protected members of the Car class.
- The SportsCar class extends the Car class by adding a new data member topSpeed and overriding the displayInfo() function to include this new information.

- This allows the `SportsCar` class to build upon the existing functionality of the `Car` class while adding new features specific to sports cars.

Tasks for Students(a):

Lab Task 01

1. **Task 1:** Create a `Student` class with data members like `name`, `rollNumber`, and `grade`. Implement member functions to set and display these values.
2. **Task 2:** Modify the `Student` class to include a constructor that initializes the data members. Implement a function to update the student's grade.
3. **Task 3:** Create a `Teacher` class that has a `name`, `subject`, and `experience`. Then, create a `Principal` class that inherits from `Teacher` and adds a new member `yearsAsPrincipal`. Implement a function in the `Principal` class to display all information.

Lab # 02

Pointers

Pointers are the most powerful feature of C and C++. These are used to create and manipulate data structures such as linked lists, queues, stacks, trees etc. The virtual functions also require the use of pointers. These are used in advanced programming techniques. To understand the use of pointers, the knowledge of memory locations, memory addresses and storage of variables in memory is required.

The variables that is used to hold the memory address of another variable is called a pointer variable or simply pointer.

The data type of the variable (whose address a pointer is to hold) and the pointer variable must be the same. A pointer variable is declared by placing an asterisk (*) after data type or before the variable name in the data type statement.

For example, if a pointer variable “**p**” is to hold memory address of an integer variable, it is declared as:

```
int* p;
```

Similarly, if a pointer variable “**rep**” is to hold memory address of a floating-point variable, it is declared as:

```
float* rep;
```

The above statements indicate that both “**p**” and “**rep**” variable are pointer variables and they can hold memory address of integer and floating-point variable respectively.

Although the asterisk is written after the data type, is usually more convenient to place the asterisk before the pointer variable. i.e. **float *rep;**

```
#include<iostream>

using namespace std;

int main() {

    int a=2, b=3;

    int *x, *y;

    x = &a;

    y = &b;

    cout<<"Memory address of variable a= "<<x<<endl;

    cout<<"Memory address of variable b= "<<y<<endl;

    cout<<"\nValue of pointer x= "<<x<<endl;

    cout<<"Value of pointer y= "<<y<<endl;

    //dereferencing

    cout<<"\nAccessing value of a using its pointer x = "<<*x<<endl;

    cout<<"Accessing value of b using its pointer y = "<<*y<<endl;

    return 0;

}
```

Output:

Memory address of variable a=
0x78fe0c

Memory address of variable b=
0x78fe08

Value of pointer x= 0x78fe0c

Value of pointer y= 0x78fe08

Accessing value of a using its pointer
x = 2

Accessing value of b using its pointer
y = 3

A pointer variable can also be used to access data of memory location to which it points.

In the above program, **x** and **y** are two pointer variables. They hold memory addresses of variables **a** and **b**. To access the contents of the memory addresses of a pointer variable, an asterisk (*) is used before the pointer variable.

For example, to access the contents of **a** and **b** through pointer variable **x** and **y**, an asterisk is used before the pointer variable. For example,

Program

Write a program to input a value to a variable using its pointer variable. Print out the value using the pointer variable.

```
#include<iostream>

using namespace std;

int main() {

    int a;

    int *x=&a;

    cout<<"Enter a value: "<<endl;

    cin>>*x;

    cout<<"Value of a is: "<<*x<<endl;

    cout<<"Memory address of a is: "<<x<<endl;

    return 0;

}
```

Output:

```
Enter a value:

5

Value of a is: 5

Memory address of a is: 0x78fe14
```

Pointers and Arrays

There is a close relationship between pointers and arrays. In Advanced programming, arrays are accessed using pointers.

Arrays consist of consecutive locations in the computer memory. To access an array, the memory location of the first element of the array is accessed using the pointer variable. The pointer is then incremented to access other elements of the array. The pointer is increased in the value according to the size of the elements of the array.

When an array is declared, the array name points to the starting address of the array. For example, consider the following example.

```
int x[5];
```

```
int *p;
```

The array “**x**” is of type **int** and “**p**” is a pointer variable of type **int**.

To store the starting address of array “**x**” (or the address of first element), the following statement is used.

```
p = x;
```

The address operator (&) is not used when only the array name is used. If an element of the array is used, the & operator is used. For example, if memory address of first element of the array is to be assigned to a pointer, the statement is written as:

```
p = &x[0];
```

when integer value 1 is added to or subtracted from the pointer variable “**p**”, the content of pointer variable “**p**” is incremented or decremented by (1 x size of the object or element), it is incremented by 1 and multiplied with the size of the object or element to which the pointer refers.

For example, the memory size of various data types is shown below:

- The array of **int** type has its object or element size of **2 bytes**. It is **4 bytes** in Xenix System.
- The array of type **float** has its object or element size of **4 bytes**.
- The array of type **double** has its object or element size of **8 bytes**.
- The array of type **char** has its object or element size of **1 byte**.

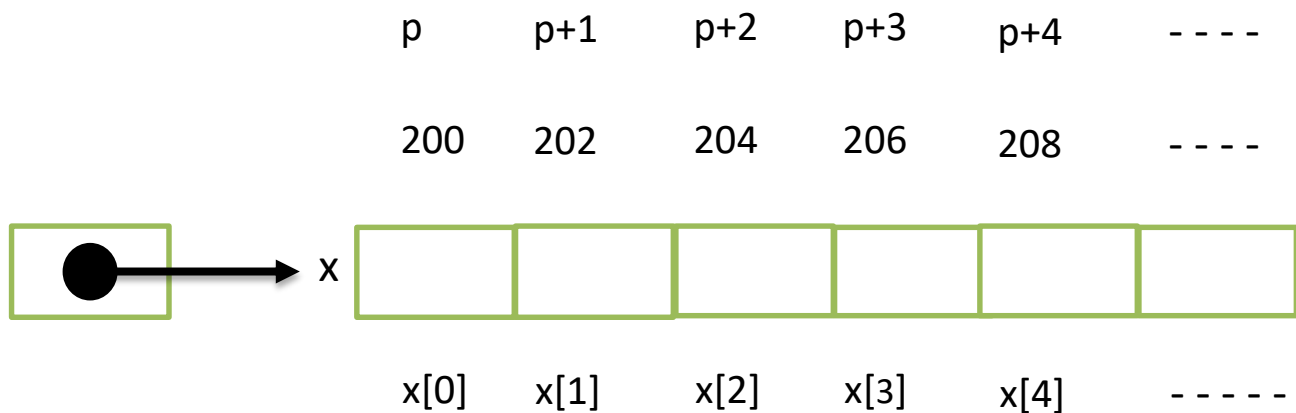
Suppose the location of first element in memory is 200. i.e., the value of pointer variable “p” is 200, and it refers to an integer variable.

When the following statement is executed,

$p = p + 1;$

the newly value of “p” will be $200 + (1 * 2) = 202$. All elements of an array can be accessed by using this technique.

The logical diagram of an integer type array “x” and pointer variable “p” is that refers to the elements of the array “x” is given below:



Program

In the below program, an integer array with a size of 5 and an integer pointer p are initialized. The pointer p is initialized to point to the memory location of the first element of the array. In the subsequent for loop, user input values are sequentially stored in the array using the pointer p, with the pointer being incremented to point to the next memory location after each input. Following this, the pointer p is reset to the start of the array. In the second for loop, the values stored in the array are accessed and displayed using the pointer p.

```
#include<iostream>

using namespace std;

int main() {

    int size=5;

    int arr[size];

    int *p=arr;

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

        cout<<"Enter value at index: "<<i<<endl;

        cin>>*p;

        p++;

    }

    p=arr;

    cout<<"Values in array: "<<endl;

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

        cout<<*p<<" ";

        p++;

    }

}
```


Output:

Enter value at index: 0

1

Enter value at index: 1

2

Enter value at index: 2

3

Enter value at index: 3

4

Enter value at index: 4

5

Values in array:

1 2 3 4 5

Passing Pointers as Arguments to Functions

The pointer variables can also be passed to functions as arguments. When pointer variable is passed to a function, the address of the variable is passed to the function. Thus, a variable is passed to a function not by its value but by its reference.

The below program defines a function func that takes an integer pointer p as its parameter. Inside the function, the integer value pointed to by p is modified to its square value. In the main function, an integer variable a is declared and assigned a value of 5. An integer pointer p is then assigned the address of a. The original value of a is displayed, showing "Value of a before function: 5". The func function is called with the p pointer as an argument, resulting in the value pointed to by p being squared and modified. The modified value of a is then displayed, demonstrating "Value of a after function: 25". This code illustrates how modifying the value through a pointer inside a function affects the original value outside the function, showcasing the concept of passing values by reference using pointers.

Program

```
#include<iostream>

using namespace std;

void func(int *p)
{
    *p=*p * *p;
}

int main() {

    int a=5;

    int *p=&a;

    cout<<"Value of a before function: "<<*p<<endl;

    func(p);

    cout<<"\nValue of a after function: "<<*p<<endl;

}
```

Output:

```
Value of a before function: 5
Value of a after function: 25
```

Returning Pointers from Function

The below program defines a function that takes an integer pointer `p` as its parameter. Inside the function, a static integer `sq` is declared to store the square of the integer pointed to by `p`. The `sq` retains its value across function calls. An integer pointer `p2` is assigned the address of the static variable `sq`, allowing it to be returned from the function. In the main function, an integer variable `a` is declared with a value of 5, and an integer pointer `p` is assigned the address of `a`. The value of `a` is displayed. The `calcSquare` function is called with `p` as an argument, leading to the calculation of the square of `a` and its storage in the static variable. The square value is displayed using the pointer `p2`.

```
#include<iostream>

using namespace std;

int * calcSquare(int *p)
{
    static int sq=*p * *p;
    int *p2=&sq;
    return p2;
}

int main() {

    int a=5;

    int *p=&a;

    cout<<"Value of a: "<<*p<<endl;

    int *p2=calcSquare(p);

    cout<<"\nSquare of a: "<<*p2<<endl;

}
```

Output:

```
Value of a: 5
Square of a: 25
```

Pointers and Strings

A String is a sequence of characters. A string type variable is declared in the same manner as an array type variable is declared. This is because a string is an array of characters type variables.

Since a string is like an array, pointer variables can also be used to access it. For example:

```
char st1[] = "Pakistan";
```

```
char *st2 = "Pakistan"
```

In the above statements, two string variables “**st1**” and “**st2**” are declared. The variable “**st1**” is an array of character type. The variable “**st2**” is a pointer also of character type. These two variables are equivalent. The difference between string variables “**st1**” and “**st2**” is that:

- string variable “**st1**” represents a pointer constant. Since a string is an array of character type, the “**st1**” is the name of the array. Also, the name of the array represents its address its which is a constant. Therefore, **st1** represents a pointer constant.
- string variable “**st2**” represents a pointer variable.

In the following program example, a string is printed by printing its character on by one.

```
#include<iostream>

using namespace std;

void display(char *s)
{
    while(*s!='\0')
    {
        cout<<*s<<endl;
        s++;
    }
}

int main() {
    char word[]="Pakistan";
    display(word);
}
```

The above program defines a function `display` that takes a pointer to a character (`char *s`) as its parameter. The function uses a `while` loop to traverse through the characters of the input string pointed to by `s`. It displays each character followed by a newline and then increments the pointer `s` to move to the next character. In the `main` function, a character array `word` containing the string "Pakistan" is declared. The `display` function is called with the `word` array as the argument. Consequently, the function iterates through the characters of the string and prints each character on a new line.

Tasks for Students (b):**Lab Task 02****Problem 01:**

- Write a program to swap two numbers using pointers. The pointers must be passed to the function.

Problem 02:

- Write a menu driven program in which you should input a string without space from the user, and the program should perform the following operations using pointers:
 - Check if the string is palindrome.
 - Count the frequency of a certain character.

Problem 03 (CHALLENGING ONE):

- Write a program in which using a 2D array store the weekly temperatures of a month. Each row represents a week. A visual representation is given below.

Week 1						
Week 2						
Week 3						
Week 4						

- Populate the array using a pointer variable with random numbers (make use of rand() function) between 10-30 degree Centigrade.
- Display the temperature of the whole month using a pointer.
- Find the hottest day of each week using a pointer, and also show the temperature on that day.

Sample Output:

```
Temperatures of each week:
```

```
12      18      5      11      30      5      19
19      23      15      6      6      2      28
2       12      26      3      28      7      22
25      3       4      23      23      22      27
```

```
Hottest day of week: 1 is:  Friday having a temperature: 30
```

```
Hottest day of week: 2 is:  Sunday having a temperature: 28
```

```
Hottest day of week: 3 is:  Friday having a temperature: 28
```

```
Hottest day of week: 4 is:  Sunday having a temperature: 27
```
