Data Structures Lab 01

Course: Data Structures (CL2001) **Semester**: Fall 2024
Instructor: Sameer Faisal **T.A**: N/A

Note:
- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Revision of Previous Programming Concepts:

Command Line Arguments

The most important function of C/C++ is main() function. It is mostly defined with a return type of int and without parameters:

```
int main() { /* ... */ }
```

Command-line arguments are given after the name of the program in command-line shell of Operating Systems. To pass command line arguments, we typically define main() with two arguments.
The first argument is the number of command line arguments and second is list of command-line arguments.

```
int main(int argc, char *argv[]) { /* ... */ }
```

or

```
int main(int argc, char **argv) { /* ... */ }
```

OOP Programming

Using Header Files

Header files are used for declaration. In OOP you should use header files to declare classes and functions. It will make your program look cleaner and more professional.

Header file for a class will include:
- Include guards.
- Class definition.
  - Member variables
  - Function declarations (only prototype)

Implementation File will include: ● Include directive for "header.h" ● Necessary include directives. ● Function definitions for all the functions of the class.

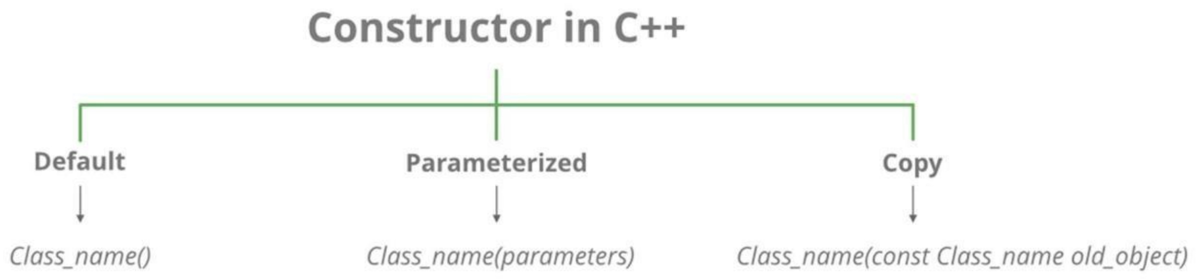
Example:

```
// my_class.h
#ifndef MY_CLASS_H // include guard
#define MY_CLASS_H

namespace N
{
              class my_class
          {
            public:
    void do_something();
          };
}
```

```
// my_class.cpp
#include "my_class.h" // header in local directory
#include <iostream> // header in standard library

using namespace N; using
namespace std;

void my_class::do_something()
{
 cout << "Doing something!" << endl;
}
```

```
// my_program.cpp #include
"my_class.cpp"

using namespace N;

int main()
{
         my_class mc;   mc.do_something();
         return 0;
}
```

Constructors and Destructors



- Default Constructor.
- Parameterized Constructor.
- Copy Constructor ● Initializer List. ● Destructors.

Let us understand the types of constructors in C++ by taking a real-world example. Suppose you went to a shop to buy a marker. When you want to buy a marker, what are the options. The first one you go to a shop and say give me a marker. So just saying give me a marker mean that you did not set which brand name and which color, you didn't mention anything just say you want a marker. So when we said just I want a marker so whatever the frequently sold marker is there in the market or in his shop he will simply hand over that. And this is what a default constructor is! The second method is you go to a shop and say I want a marker a red in color and XYZ brand. So you are mentioning this and he will give you that marker. So in this case you have given the parameters. And this is what a parameterized constructor is! Then the third one you go to a shop and say I want a marker like this (a physical marker on your hand). So the shopkeeper will see that marker. Okay, and he will give a new marker for you. So copy of that marker. And that's what a copy constructor is!

A destructor is also a special member function as a constructor. Destructor destroys the class objects created by the constructor. Destructor has the same name as their class name preceded by a tilde (~) symbol. It is not possible to define more than one destructor. The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded. Destructor neither requires any argument nor returns any value. It is automatically called when the object goes out of scope. Destructors release memory space occupied by the objects created by the constructor. In destructor, objects are destroyed in the reverse of object creation.

Dynamic Memory

C++ supports three types of memory allocation.

➔ Static memory allocation happens for static and global variables. Memory for these types of variables is allocated once when your program is run and persists throughout the life of your program.

➔ Automatic memory allocation happens for function parameters and local variables. Memory for these types of variables is allocated when the relevant block is entered, and freed when the block is exited, as many times as necessary.

➔ Dynamic memory allocation is a way for running programs to request memory from the operating system when needed.

new Operator

- This operator is used to allocate a memory of a particular type.
- This creates an object using the memory and returns a pointer containing the memory address.
- The return value is mostly stored in a pointer variable.

```
// new_op.cpp int main()
{
                        int *ptr = new int; // allocate memory
                    *ptr = 7; // assign value

      // allocated memory and assign value           int *ptr2 = new
int(5);
}
```

delete Operator

- When we allocate memory dynamically, we need to explicitly tell C++ to deallocate this memory.
- delete Operator is used to release / deallocate the memory.

```
// delete_op.cpp
#include
<iostream> int
main() {
int *ptr = new int; // dynamically allocate an integer int *otherPtr
= ptr; // otherPtr is now pointed at that same memory delete ptr; //
ptr and otherPtr are now dangling pointers.
ptr = 0; // ptr is now a nullptr
// however, otherPtr is still a dangling pointer!
return 0; }
```

Rule of Three:

If you need to explicitly declare either the destructor, copy constructor or copy assignment operator yourself, you probably need to explicitly declare all three of them.

The default constructors and assignment operators do shallow copy and we create our own constructor and assignment operators when we need to perform a deep copy (For example when a class contains pointers pointing to dynamically allocated resources).

First, what does a destructor do? It contains code that runs whenever an object is destroyed. Only affecting the contents of the object would be useless. An object in the process of being destroyed cannot have any changes made to it. Therefore, the destructor affects the program's state as a whole.

Now, suppose our class does not have a copy constructor. Copying an object will copy all of its data members to the target object. In this case when the object is destroyed the destructor runs twice. Also the destructor has the same information for each object being destroyed. In the absence of an appropriately defined copy constructor, the destructor is executed twice when it should only execute once. This duplicate execution is a source for trouble.

```
class Array  {
private:
   int size;   int*vals;

public:
 ~Array();
   Array( int s, int* v );
};
Array::~Array()
{    delete vals;
vals = NULL;
}

Array::Array( int s, int* v )
{
   size = s;     vals = new int[ size ];     std::copy(
v, v + size, vals );
}

int main()  {     int vals[ 4 ] = {
11, 22, 33, 44 };
  Array a1( 4, vals );
  // This line causes problems.
  Array a2( a1 );

  return 0;
}
```

Lab Exercises:

1. Suppose you are developing a bank account management system, and you have defined the BankAccount class with the required constructors. You need to demonstrate the use of these constructors in various scenarios.

   a) Default Constructor Usage:
   Create a default-initialized BankAccount object named account1. Print out the balance of account1.

   b) Parameterized Constructor Usage:
   Create a BankAccount object named account2 with an initial balance of $1000. Print out the balance of account2.

   c) Copy Constructor Usage:
   Using the account2 you created earlier, create a new BankAccount object named account3 using the copy constructor. Deduct $200 from account3 and print out its balance. Also, print out the balance of account2 to ensure it hasn't been affected by the transaction involving account3. Note: assume the variables in your case and print out the details.

2. Create a C++ class named "Exam" designed to manage student exam records, complete with a shallow copy implementation? Define attributes such as student name, exam date, and score within the class, and include methods to set these attributes and display exam details. As part of this exercise, intentionally omit the implementation of the copy constructor and copy assignment operator. Afterward, create an instance of the "Exam" class, generate a shallow copy, and observe any resulting issues?

3. You're tasked with designing a Document class for a document editor program. The class should handle text content, ensuring that copying a document creates a deep copy of the content to maintain data integrity. Follow the Rule of Three to manage resource allocation and deallocation correctly.
Here are the key requirements:

   (a) Create a constructor that takes initial text content and allocates memory for it.
   (b) Implement a destructor to deallocate memory used for the text content.
   (c) Create a copy constructor that performs a deep copy of the text content, preventing unintended sharing.
   (d) Create a copy assignment operator that ensures a deep copy of the text content, maintaining separation between objects.
   (e) Provide a sample program that showcases your Document class. Create an original document, generate copies using both the copy constructor and copy assignment operator, modify the original's content, and show that the copies remain unaffected.