Lab Manual

CL2001 – Data Structures

Fall-2022 | Lab_01

National University of Computer and Emerging Sciences, Karachi Campus

Data Structures Lab
Lab_01

Course: Data Structures (CL2001) **Semester:** Fall 2022
Instructor: Shoaib Rauf **T.A:** N/A
Deadline: Friday, August 26, 2022 (02:00 PM After noon) (Submit on Google Classroom)
Points: 70

Submission Guidelines:

1. Solve each problem in separate file, Name the code file with problem no (Task_01, Task_02,.)
2. Copy these files (Task_01, Task_02,.) in a folder and name the folder like that K21XXXX. where XXXX is your 4-digit Student Id.
3. Now compress that folder and submit on google-classroom.
4. Do not attach .exe file, otherwise it will show a threat or virus and not allow me to download.
5. Make sure you must Press the Turn-In button after uploading the solution folder. Otherwise, it will not be submitted.

Task # 1

Using Command Line arguments

The main() in C/C++ is a very vital function. It is the main entry point for a program. There will be only one main in a program. It is generally defined with a return type of int and without parameters:

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

We can also give command-line arguments in C and C++. Command-line arguments are given after the name of the program in command-line shell of Operating Systems.

C:>\ myProgram.exe I am Rafi

To pass command line arguments, we typically define main() with two arguments : 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) { /* ... */ }

Task # 2

Using Command Line arguments to read from the file and write to the file.

Example:

```
int main () {
  ofstream myfile;
  myfile.open ("example.txt");
```

```
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

There will be 2 text file for this task input_file.txt and output_file.txt. these two file names come as some arguments to the main program. You need to open the input_file.txt for reading and output_file for writing. If the file is not created already do create a file with this name. Now read the file character by character and write it back to output_file. After completing the entire input_file to output_file
Do close the 2 files.

Debugging in C++:
Steps to Remember:
Consider the following Code as an example

The following coding example illustrates how does a debugger executes the code, steps into which method's body, how does it steps out and when does it step over a function.

```
int a =5; int b=6; int c =3;
 if( a> b && a < c){
    cout << "A is the greatest of all three integers "<< a << endl;
}
  else if (b> a && b > c){
    cout << "B is the greatest of all three integers "<< b << endl;
}
  else {
    cout << "C is the greatest of all three integers << c << endl;
}

 return 0;
 }
```

The below task is to understand the Watching and Debugging the code segments.

Task # 3
The problem is to get a positive integer from the keyboard and find all its factors by division method. After completing all factor also produce the sum of all the factors. The program should stop when a user entered -1 as input.

Creating a watch for the iteration on factor and sum to find whether the process is correct or not.

Task # 4

This task is to understand the Watching and Debugging of functions and their effects.

Write the task # 3 code as function to return factors and sum. See effectively how step into a function and step out of a function works.

Pointers:

A pointer variable is one that contains the address, not the value, of a data object. A pointer can be declared as,

Int *intptr;

There are few important operations, which we will do with the pointers very frequently. (a) We define a pointer variable. (b) Assign the address of a variable to a pointer. (c) Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations.

```
int main()

{

int var = 20; // actual variable declaration.
int *ip;       // pointer variable
ip = &var;
 // store address of var in pointer variable
// print the address stored in ip pointer
//Code
// access the value at the address available in pointer
   //Code
return 0;
}
```

Task # 5

Find the largest and the smallest element in an array using pointers. Create a function named MinMax having addresses of Min and Max from main function, use pointers so that it will directly affect the values of the variables defined inside the main function.

Dynamic Allocation of Objects:

[Dynamically allocate memory to 5 objects of a class.]

Just like basic types, objects can be allocated dynamically, as well.

But remember, when an object is created, the constructor runs. Default constructor is invoked unless parameters are added:

```
Fraction * fp1, * fp2, * flist;
fp1 = new Fraction;              // uses default constructor
fp2 = new Fraction(3,5);         // uses constructor with two parameters
```

```
flist = new Fraction[20];        // dynamic array of 20 Fraction objects
// default constructor used on each
```

Deallocation with delete works the same as for basic types:

```
delete fp1;
delete fp2;
delete [] flist;
```

Notation: dot-operator vs. arrow-operator:

dot-operator requires an object name (or effective name) on the left side

```
objectName.memberName                // member can be data or function
```

The arrow operator works similarly as with structures.

```
pointerToObject->memberName
```

Remember that if you have a pointer to an object, the pointer name would have to be dereferenced first, to use the dot-operator:

```
(*fp1).Show();
```

Arrow operator is a nice shortcut, avoiding the use or parentheses to force order of operations:

```
fp1->Show(); // equivalent to (*fp1).Show();
```

When using dynamic allocation of objects, we use pointers, both to single object and to arrays of objects. Here's a good rule of thumb:

For pointers to single objects, arrow operator is easiest:

```
fp1->Show();
fp2->GetNumerator();
fp2->Input();
```

For dynamically allocated arrays of objects, the pointer acts as the array name, but the object "names" can be reached with the bracket operator. Arrow operator usually not needed:

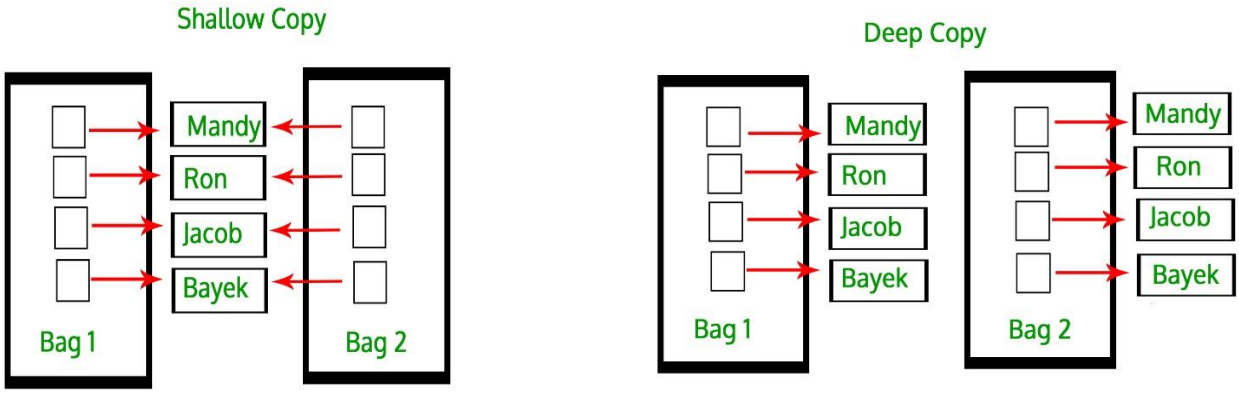
```
flist[3].Show();
flist[5].GetNumerator();

// note that this would be INCORRECT, flist[2] is an object, not a pointer flist[2]->Show();
```

Task # 6 create an Animal class (having appropriate attributes and functions) and do following steps.

1. Dynamically allocate memory to 5 objects of a class.

2. Order data in allocated memories by Animal names in ascending order.

Rule of Three:
The Rule of Three is a rule of thumb in C++ (prior to C++ 11) which states that if a class defines one or more of the following, then it should probably explicitly define all three of these. The rule is also called Law of Big Three or The Big Three, three special functions which are:
1) Destructor
2) Copy Constructor
3) Overloaded/Copy Assignment Operator.



Example:

```
class Numbers{
//private members:

//public members

  //copy constructor
  /*
  Numbers(const ClassName &obj)
  {
Some code
  }*/

  ~Numbers() //destructor
  {
    delete ptr;
  }
};

int main(){

  int arr[ 4 ] = { 11, 22, 33, 44 };
  Numbers Num1( 4, arr );
  // this creates problem (if not using the copy constructor)
  Numbers Num2( Num1 );

  return 0;
}
```

Task # 7 The task is to understand the concept of shallow and deep copies, and to understand the logic behind Rule of Three.

Write a code that create a Class named Numbers, with private size and pointer variables , a public function to assign values to private member and a destructor to destroy the pointer and memory. Understand what happens when you define an object by providing one object as an argument to another or simply assign one object to another.

Reference:

Using command line arguments in C++

- **argc :** Argument Count (argc) is integer type and cannot be negative. It represents number of command-line arguments passed by the user including the absolute name of the program.
- **argv:** ARGument Vector (argv) is array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings. Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

Filling as Command Line arguments:

- Write a program that read a file containing only one line as an input argument (input_file.txt)
- The program writes the line back into a new file as output. (output_file.txt)
- File operations in C/C++

Watching and Debugging C/C++ Programs:

- Creating a watch list of variables, seeing the values during the execution of the program.
- Stepping into a function and Stepping out of the function.
- Identifying potential value watch for debugging.

Pointers:

- Use of pointers, storing the address of a variable into a pointer variable and accessing the value it points.
- Value of the variables changes when the address hold by pointer variables changes.

Dynamic Allocation of Objects:

- Memory allocation and deallocation of objects in variables and arrays using default and copy constructors.

Rule of Three:

- Creating a shallow copy and then calling a destructor without explicit definition of copy constructor and assignment operator.
- Creating deep copy by explicitly defining copy constructor and assignment operator to resolve the anomaly caused when the destructor is called twice or more.

Lab1: Programming and Debugging		
Std Name:		**Std_ID:**
Lab1-Tasks	**Completed**	**Checked**
Task #1		
Task #2		
Task #3		
Task# 4		
Task# 5		
Task# 6		
Task# 7		

Instructor signature: _____