Eye-User Manual

Sahil Shah, Preey Shah and Aman Bansal

October 2017

1 Introduction

Our language is very similar to C++ with some in-built data structures like stacks, queues, and others mentioned below. It has most of the implementations of C++ except classes. Given below are some of the rules of the language.

2 General Rules

- Every statement has a semicolon at its end
- The language is not whitespace sensitive
- The main program begins with a main-program tag
- All the control flow statements have to come within a bracket

3 Declarations

The declarations are exactly similar to C++. The type of the variable to be declared is followed by the names of the variables separated which can be given a value eg $int\ a = 10, b = 5, c;$. The primitive datatypes are

- bool
- int
- float

- string

The arrays can also be initialized by mentioning the type. eg. int a[10];

4 Assignment

The name of a variable can be written in any expression(in the rhs of an assignment) and the value of the variable at that point is substituted. The expression can be parenthesised or not, it will be implemented using the rules of BODMAS.The LHS of an assignment has to be a variable that is already initialized.eg. a=x*y-3. The algebraic and boolean operators are similar to C++, with only the ++ construct not defined. The arrays can be accesed similarly.

5 Functions

The functions are exactly similar to C++, but everything is passed by value and arrays cannot be passed as arguments.

- The function has to have a return statement or it should be of the type void.
- The function can be called at the RHS of an assignment if it has a non-void return type
- Else it can be called just as a statement
- Global variables have been defined too. Their value can be changed anywhere in the program
- Normal rules of scoping and blocks have been followed.

6 Control-flow

- If-Elseif-else statements-They are implemented as in C++. The block of every condition should be inside a bracket
- The for-loops and while-loop have been implemented with their usual meaning in C++.

7 Data Structures

We have defined several of the common data structures used by programmers and we go on to explain how to declare instances of these data structures and the member functions that it contains.

7.1 Stack

This declares the data structure called stack, which is basically a first-in-first-out or FIFO kind of data structure. To define it use the following signature:

```
stack<dataType> StackName
```

where dataType can refer to int, float, string, etc. The member functions include functions to access the top element of the stack, check the size, check if it is empty, insert to the stack and pop the top element.

```
myStack.push(10) //pushes 10 to the top of the stack
myStack.pop() //pops the top element of the stack
myStack.top() //returns the top element of the stack
myStack.empty() //returns true if the stack is empty
myStack.size() //returns the number of elements in the stack
```

7.2 Queue

This data structure is a first in last out or LIFO queue. It is a double sided queue, that is, you can push and pop from and to both ends of the queue. The declaration is as follows:

```
queue<float>q;
```

The member functions are as follows:

```
q.pushFront(0.3) //push to the front of the queue
q.popFront() //pop the element at the front of the queue
q.pushBack(0.3) //push to the back of the queue
q.popBack() //pop from the back of the queue
q.empty() //return true if the queue is empty
q.size() //return the size of the queue
q.front() //return the element at the front of the queue
q.back() //return the element at the back of the queue
```

7.3 Binary Search Tree

We have implemented a binary search tree which allows insertion, deletion and finding operations in logarithmic time.
The declaration is as follows:

```
binarySearchTree<double>bst;
```

The member functions for searching, insertion and deletion are as follows:

```
bst.insert(2.3) //insert an element into the binary tree
bst.find(2.4) //return true if the element is in the binary tree
bst.delete(2) // deletes the element if it is found in the tree
```

7.4 Linked List

This implements a singly linked list which is declared as follows:

```
linkedList<long long int> ll;
```

This supports functions like inserting an element at an index, searching for an element, deleting an element and returning the index of an element.

```
ll.insert(10, 12000) //inserts 12000 at index 10
ll.indexOf(12000) //returns the index at which an element is present
ll.erase(10000) //erase an element from the list
ll.find(100) //returns true if an element is found
```

7.5 Hash Table

This implements a hashing table which using open hashing and uses a hash function that uses the remainder function. It is similar to the unordered set of the C++ STL. The declaration is as follows:

```
hashTable<int>ht(29) // where 29 is the divisor we want to use
```

It supports the functions of push, erase and find. We use them as follows:

```
ht.push(12) // insert 12 into the hash table
ht.find(12) //return true if 12 is in the hash table
ht.erase(13) // erase 13 if it is present in the hash table
```

7.6 Doubly Linked List

This implements a doubly linked list which is declared as follows:

```
doublyLinkedList<long long int> dl;
```

This supports functions like inserting an element at an index, searching for an element, deleting an element and returning the index of an element.

```
dl.insert(10, 12000) //inserts 12000 at index 10
dl.indexOf(12000) //returns the index at which an element is present
dl.erase(10000) //erase an element from the list
dl.find(100) //returns true if an element is found
```