**Software Testing Project**

**CMP461**

**Team “Sunday”**

**Team Members:**

* Radwa Samy Khattab 1 – 18
* Salma Ibrahim 1 – 26
* Mariam Hesham 2 – 21
* Nehal Abdelkader 2 – 28
* Nourhan Gamal 2 – 31

**Delivered to**

Dr. Ahmed Sobeih

Eng. Ali El-Sedeek

[I. Software Under Test (SUT) 3](#_Toc59149908)

[II. Coverages and Approach in Testing 3](#_Toc59149909)

[III. User-defined Map 4](#_Toc59149910)

[IV. User-define List 6](#_Toc59149911)

[V. User-Defined Linked Stack 9](#_Toc59149912)

[VI. User-Defined Linked Queue 10](#_Toc59149913)

[VII. Graph 12](#_Toc59149914)

# Software Under Test (SUT)

We choose various user implemented data structures to implement in C++. For example: pair, map, linked list, stack, queue, binary trees, graphs…etc.

In addition, we choose multiple sorting and searching algorithms and implemented them too.

All code is written in C++.

Then we choose Google Test (G-Test) tool to test these codes and their results.

Starting III. we would show each Data Structure/Algorithm we implemented and the code used for testing using GTest framework, and the results of these tests. Mostly all would be correct as we implemented the original code in a clean clear way and were almost sure it had no errors.

The expected output of each test case is shows in the testing code.

For example:

EXPECT\_EQ(queue.top(), 0);

Here it shows that the expected output of “queue.top()” should be equal to 0.

The actual output is showed after running the test code. If all passed that means the actual output is same as the expected output.

GTest documentation can be shown here: <https://github.com/nordlow/gtest-tutorial>

But generally, name of functions can be easy to understand their behavior, as in the previous example, “EXPECT\_EQ” means EXPECT EQUAL.

# Coverages and Approach in Testing

We mainly focused on call coverage in data structure testing, which was mentioned in one of the first lectures. When testing a class, we have to make sure to test all the member functions of that class. And in each member function, which is called at least once, we make sure of satisfying Prime Path Coverage, which subsumes Edge Pair Coverage, which subsumes Edge Coverage, which also subsumes Node Coverage.

All-Defs coverage is covered in our test cases too. And in some cases, so is All-Coupling-Defs coverage is covered too or even AUC, and ACUC.

When needed, we try out different input for same function. Those values can be chosen like we learnt in Input Space Partition. However, it won’t be needed in all functions.

In Sorting and Searching algorithms, PPC is also guaranteed in our test cases.

Each Data Structure or algorithm are shown in more details in following sections.

# G-Test Used Functions

|  |  |  |
| --- | --- | --- |
| Functions | Parameters | Explanation |
| TEST | Test Class Name, Test Name | The main test function for a test case, in this function we use the following functions to test out specific values. |
| EXPECT\_EQ | Val1, Val2 | Expect val1 to be val2 |
| EXPECT\_TRUE | Condition | Expect condition to be True |
| EXPECT\_FALSE | Condition | Expect condition to be False |
| EXPECT\_STREQ | Str1, Str2 | Expect Str1 to equal Str2 |

# User-defined Map

## User-defined “Pair” Class Header Code:

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Key | --- | --- |
| Value | --- | --- |
| Pair | Void | Void |
| Pair | Int Key, Int value | Void |
| setPair | Int Key, Int value | Void |
| setValue | Int Value | Void |
| getKey | Void | Int |
| getValue | Void | Int |
| Operator== | Pair &p | Boolean |

## User-defined Map Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Pair pairs[MAXSIZE] | --- | --- |
| Count | --- | --- |
| Map | Void | Void |
| addPair | Int Key, Int value | Void |
| getValue | Int Key | Int |
| getCount | Void | Int |
| updateValue | Int Key, Int &value | Boolean |
| deleteKey | Int key | Boolean |

## Map Test Code

### Constructor & Get Count tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Map initialized  Count = 0 | getCount | Void | 0 | 0 |

### Add Pair and Get Pair Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Map initialized  Count = 0 | addPair | Key=1, Value=-4 | True | True |
| Map has 1 element | getCount | Void | 1 | 1 |
| Map has pair (1, -4) | getValue | Key=1 | -4 | -4 |
| Map doesn’t have key 2 | getValue | Key=2 | NULL | NULL |
| Map initialized | addPair | Key=1.4, Value=0 N.B. 1.4 will be rounded to 1, because datatype is Integer | True | True |
| Map has pair (1, 4) | getValue | Key=1.4  N.B. will be considered 1 | -4 | -4 |

### Max Capacity (Size) of Map Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Map has count = MAXSIZE | getCount | Void | MAXSIZE | MAXSIZE |
| Map has count = MAXSIZE | addPair | Key=1, value=1 | FALSE  (no place to add) | FALSE |
| Previous state | getCount | Void | MAXSIZE | MAXSIZE |

### Update Value Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Map has pair (5, 30) | updateValue | Key=5, Value=50 | TRUE | TRUE |
| Map has supposedly updated pair (5, 30) to (5, 50) | getValue | Key=5 | 50 | 50 |
| Map doesn’t have key 3 | updateValue | Key=3 | FALSE | FALSE |

### Delete Key Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Map has pair (5, 50) and (3, 30) | deleteKey | Key=3 | TRUE | TRUE |
| After Previous Test case | getCount | Void | 1 | 1 |
| Map doesn’t have key 7 | deleteKey | Key=7 | FALSE | FALSE |

## Map Tests Results

# User-define List

You can set any value at any specific index in a List, as long as it’s below MAX\_SIZE

## Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Int List[MAXSIZE] | --- | --- |
| Int Size | --- | --- |
| Char \* nameOfList | --- | --- |
| List | Void | Void |
| List | Int maxSize, char\* name | Void |
| Set | Int index, Int data | Void |
| setName | Char\* name | Void |
| getIndex | Int data | Int |
| Sort | Void | Void |
| getSize | Void | Int |
| getName | Void | Char\* |
| Operator [] | Int index | Int |
| Operator = | List &l | List |
| Operator == | List &l | Boolean |

## Test Code

### Empty Constructor Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| List initialized by empty constructor | getSize | Void | MaxSize | MaxSize |
| List initialized by empty constructor | getName | Void | “” | “” |
| List initialized by empty constructor | Operator[] | Any index from 0→MaxSize | 0 | 0 |

### Initialize Constructor Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| List initialized by init constructor, setting size=3 and name= “smallList”  i.e. List(size, name) | getSize | Void | 3 | 3 |
| Same as above | getName | Void | “smallList” | “smallList” |
| Same as above | Operator[] | Any index from 0→MaxSize | 0 | 0 |

### Change Name Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| List initialized  Size=3, name= “smallList” | setName | Name= “newname” | Void | Void |
| After previous test case | getName | Void | “newName” | “newname” |

### Setting Values Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| List initialized, size = 10 | set | Index=4, data=5 | TRUE | TRUE |
| List initialized, size = 10 | set | Index=-2, data=2 | FALSE | FALSE |
| List initialized, size = 10 | set | Index=13, data=3 | FALSE | FALSE |

### Index Operator Overloading Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| List init, size=20 | Set [] | L[3] = 30 | Void | Void |
| After previous test case | Get [] | L[3] | 30 | 30 |
| After previous test case | == | L[3] == 30 | TRUE | TRUE |

### Equal Operator Overloading Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| L initialized, size=6, name= “testList”  L1 initialized, size=5, name= “list1”, has values from 1 to 5 | = | L = L1 | Void | Void |
| After Previous Test case | L.getSize | Void | 5 | 5 |
| After Previous Test case | L.getName | Void | “List1” | “List1” |
| After Previous Test case | [] | L[0] | 1 | 1 |
| After Previous Test case | [] | L[2] | 3 | 3 |
| After Previous Test case | [] | L[4] | 5 | 5 |

### Sort Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| L initialized, size=5 l = [4, 2, 0, -4, 1] | sort | Void | Void | Void |
| After Previous Test case | [] | L[0] | -4 | -4 |
| After Previous Test case | [] | L[2] | 1 | 1 |
| After Previous Test case | [] | L[4] | 4 | 4 |

### Get Index Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| L initialized, size=5, L = [10, 20, 30, 40, 50] | getIndex | 10 | 0 | 0 |
| L initialized, size=5, L = [10, 20, 30, 40, 50] | getIndex | 30 | 2 | 2 |
| L initialized, size=5, L = [10, 20, 30, 40, 50] | getIndex | 50 | 4 | 4 |
| L initialized, size=5, L = [10, 20, 30, 40, 50] | getIndex | 70 | -1 | -1 |

## List Tests Results

# User-Defined Linked Stack

## Node Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Int item | --- | --- |
| Node \* next | --- | --- |
| Char \* nameOfList | --- | --- |
| Node | Void | Void |
| Node | Int item | Void |
| Node | Int item, Node \* nextNodePtr | Void |
| setItem | Int item | Void |
| setNextNode | Node \* nextNodePtr | Void |
| getItem | Void | Int |
| getNextNode | Void | Node \* |

## LinkedStack Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Node \* top | --- | --- |
| LinkedStack | Void | Void |
| LinkedStack (copy constructor) | LinkedStack &stack | Void |
| isEmpty | Void | Boolean |
| Push | Int data | Void |
| Pop | Void | Boolean |
| Peek | Void | Int |

## Linked Stack Test Code

### Constructor & isEmpty Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Stack initialized | isEmpty | Void | TRUE | TRUE |

### Push and Peek Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Stack initialized | push | 1 | Void | Void |
| After previous test case | peek | Void | 1 | 1 |
| After previous test case | push | 3 | Void | Void |
| After previous test case | peek | Void | 3 | 3 |

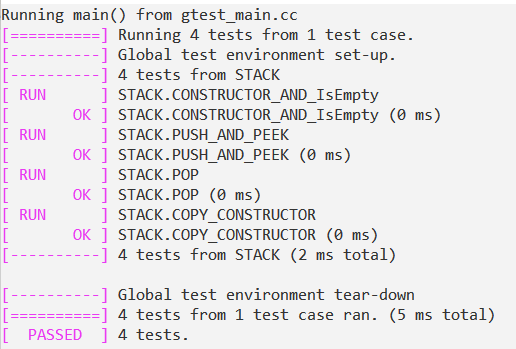
### Pop Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Stack initialized,  1, 3 are pushed | pop | Void | True | True |
| After previous test case | isEmpty | Void | FALSE | FALSE |
| After previous test case | peek | Void | 1 | 1 |
| After previous test case | pop | Void | True | True |
| After previous test case | isEmpty | Void | TRUE | TRUE |
| After previous test case | pop | Void | FALSE | FALSE |

### Copy Constructor Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Stack s initialized, 1 pushed  Stack s2 = s | S2.peek | Void | 1 | 1 |
| same | s.peek | Void | 1 | 1 |

## Linked Stack Tests Results



# User-Defined Linked Queue

## Linked Queue Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Node \* backPtr | --- | --- |
| Node \* frontPtr | --- | --- |
| LinkedQueue | Void | Void |
| LinkedQueue (copy constructor) | LinkedQueue &queue | Void |
| isEmpty | Void | Boolean |
| Enqueue | Int data | Void |
| Dequeue | Void | Boolean |
| PeekFron | Void | Int |

## Linked Queue Test Code

### Constructor & isEmpty Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Queue initialized | isEmpty | Void | TRUE | TRUE |

### Push and Peek Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Queue initialized | Enqueue | 1 | Void | Void |
| After previous test case | peekFront | Void | 1 | 1 |
| After previous test case | Enqueue | 3 | Void | Void |
| After previous test case | peekFront | Void | 1 | 1 |

### Dequeue

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Queue initialized,  1, 3 are enqueued | Dequeue | Void | TRUE | TRUE |
| After previous test case | isEmpty | Void | FALSE | FALSE |
| After previous test case | peekFront | Void | 3 | 3 |
| After previous test case | Dequeue | Void | TRUE | TRUE |
| After previous test case | isEmpty | Void | TRUE | TRUE |
| After previous test case | Dequeue | Void | FALSE | FALSE |

### Copy Constructor Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Queue q initialized, 3, 2, 1 are enqueued  Queue q2 = q | q2.peekFront | Void | 3 | 3 |
| same | q.peekFront | Void | 3 | 3 |

## Linked Queue Tests Results

# Graph

## Edge Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Int destination | --- | --- |
| Edge \* nextEdge | --- | --- |
| Long weight | --- | --- |
| Edge | Void | Void |
| Edge | Void | Void |
| Edge | Void | Void |
| Edge | Void | Void |
| setDestination | Int d | Void |
| setNext | Edge \* next | Void |
| setWeight | Int w | Void |
| getDestnation | Void | Int |
| getNext | Void | Edge \* |
| getWeight | Void | Int |

## Vertex Class Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Int data | --- | --- |
| Vertex \* nextVertex | --- | --- |
| Int inDegree | --- | --- |
| Int outDegree | --- | --- |
| Edge \* firstEdge | --- | --- |
| Int processed | --- | --- |
| Vertex | Int data | Void |
| setData | Int data | Void |
| setNextVertex | Vertex \* vertex | Void |
| setFirstEdge | Edge\* edge | Void |
| getData | Void | Int |
| getNextVertex | Void | Vertex \* |
| getFirstEdge | Void | Edge \* |
| getProcessedCount | Void | Int |
| incProcessed | Void | Void |
| resetProcessed | Void | Void |
| isConnected | Int vertex2 | Boolean |
| addEdge | Int vertex, int weight | Void |
| removeEdge | Int destination | Void |
| getEdgeWeight | Int destination | Int |
| HasEdges | Void | Boolean |

## Graph Header Code

|  |  |  |
| --- | --- | --- |
| Data Member/ Member Function | Parameters | Return value |
| Vertex \* headVertex | --- | --- |
| Vertex \* tailVertex | --- | --- |
| Int numVertices | --- | --- |
| Int numEdges | --- | --- |
| getVertexPtr (private utility function) | Int v | Vertex \*& |
| getPrevVertex (private utility function) | Int v | Vertex \*& |
| Graph | Void | Void |
| Add | Int v1, Int v2, int w | Boolean |
| Remove | Int v1, int v2 | Boolean |
| Clear | Void | Void |
| isExists | Int vertex | Boolean |
| isEmpty | Void | Boolean |
| isConnected | Int v1, int v2 | Boolean |
| getNumVertices | Void | Int |
| getNumEdges | Void | Int |
| getEdgeWeight | Void | Int |
| depthFirstTraversal | Int start | String |
| breadthFirstTravesal | Int start | String |

## Graph Tests Code

### Constructor & Empty Checks Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized | isEmpty | Void | TRUE | TRUE |
| Graph initialized | getNumEdges | Void | 0 | 0 |
| Graph initialized | getNumVertices | Void | 0 | 0 |

### isExists Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized and have:  1 – 2, w=10 | isExists | vertex=1 | TRUE | TRUE |
| Same as above | isExists | vertex=3 | False | False |

### isConnected Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized & have:  1 – 2, w=10 | isConnected | v1=1, v2=2 | TRUE | TRUE |
| Same as above | isConnected | v1=3, v2=2 | False | False |

### Add Vertex Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized | Add | v1=1, v2=2, w=10 | TRUE | TRUE |
| After previous test case | Add | v1=1, v2=3, w=20 | TRUE | TRUE |
| After previous test case | Add | v1=4, v2=2, w=30 | TRUE | TRUE |
| After previous test case | Add | v1=3, v2=2, w=40 | TRUE | TRUE |
| After previous test case | Add | v1=5, v2=6, w=50 | TRUE | TRUE |
| After previous test case | Add | v1=1, v2=2, w=60 | FALSE | FALSE |

### Check Size Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized & have:  1—2, w=10  1—3, w=20 | isEmpty | Void | TRUE | TRUE |
| same | getNumEdges | Void | 2 | 2 |
| same | getNumVertices | Void | 3 | 3 |

### Clear Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized & had:  1—2, w=10  Then function “clear” is called | isEmpty | Void | TRUE | TRUE |
| After previous test case | getNumEdges | Void | 0 | 0 |
| After previous test case | getNumVertices | Void | 0 | 0 |

### Remove Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized and have:  1 – 2, w=10  1—3, w=20 | remove | v1=4, v2=5  (don’t exists) | FALSE | FALSE |
| After previous test case | remove | v1=3, v2=2  (not connected) | FALSE | FALSE |
| After previous test case | remove | v1=1, v2=2  (v2 will be removed from graph) | TRUE | TRUE |
| After previous test case | getNumEdges | Void | 1 | 1 |
| After previous test case | getNumVertices | Void | 2 | 2 |
| After previous test case | isConnected | v1=1, v2=2 | FALSE | FALSE |

### Edge Weight Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized & have:  1 – 4, w=10 | getEdgeWeight | v1=1, v2=4 | 10 | 10 |
| Same as above | getEdgeWeight | v1=1, v2=3  (v3 doesn’t exists) | -1 | -1 |

### Add Vertex Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized and is as shown in figure below | depthFirstTraversal | Start=1 | 143256 | 143256 |

### Check Size Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Previous State | Tested function | Inputs | Expected Output | Actual Output |
| Graph initialized and is as shown in figure below | breadthFirstTraversal | Start=1 | 143256 | 143256 |

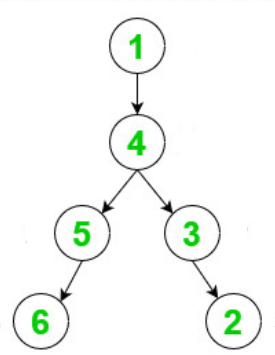


Figure : Graph used in test 9 and 10

## Graph Tests Results