	King Saud University	College of Computer and Information Sciences Department of Computer Science
	Data Structures CSC 212	**Final Exam Solution - Fall 2018**
	Date: 15/12/2018	Duration: 3 hours

Guidelines: No calculators or any other electronic devices are allowed in this exam.

Student ID:	Name:
Section:	Instructor:

1	2.1	2.2	3.1	3.2	4	5	6	7	8	Total

Question 1 ....16 (Part (a) and (b): 1pt / question. Part (c): 2pts for 1, 2, 3 and 1pt for 4) points

(a) Choose the correct frequency for every line as well as the total O of the following code:

```
1 int A = 0;
2 for (int i = 1; i <= n; i++)
3   for (int j = 0; j < i; j++)
4     A++;
```

1. Line 1: (A) 0 (B) 1 (C) 2 (D) n (E) A
2. Line 2: (A) A (B) i (C) $i+1$ (D) n (E) $n+1$
3. Line 3: (A) n^2 (B) $n(n+1)/2$ (C) $n(n+1)/2+1$ (D) $(n^2+3n)/2$ (E) $n(n-1)/2-1$
4. Line 4: (A) A^2 (B) n^2 (C) $(n^2+3n)/2$ (D) $n^2(n+1)/2+1$ (E) $n(n+1)/2$
5. Tightest Total O: (A) n (B) n^2 (C) n^3 (D) n^4 (E) None

(b) Choose the correct frequency for every line as well as the total O of the following code:

```
1 int i = 1;
2 while (i < n) {
3   i++;
4   if (i > 7) break;
5 }
```

1. Line 1: (A) 1 (B) 0 (C) i (D) n (E) $n+1$
2. Line 2: (A) 8 (B) 7 (C) n (D) $n-1$ (E) $n+1$
3. Lines 3 (and similarly 4): (A) n (B) $n-1$ (C) 6 (D) 7 (E) 8
4. Tightest Total O: (A) 1 (B) n (C) $\log(n)$ (D) n^2 (E) 2^n

(c) Choose the correct answer:

1. $n^7+n^4+n^2+\log n$ is : (A) $O(n^2)$ (B) $O(n^4)$ (C) $O(n^7)$ (D) $O(\log(n))$ (E) None
2. $2^n+n!$ is : (A) $O(n^2)$ (B) $O(2^n)$ (C) $O(n!)$ (D) $O(n^n)$ (E) None
3. $n+\log n^3+6$ is : (A) $O(n)$ (B) $O(\log n^3)$ (C) $O(n \log n)$ (D) $O(n^3)$ (E) None
4. The time complexity of inserting an element in a heap of n elements is:

   (A) $O(n^2)$ (B) $O(n)$ (C) $O(2^n)$ (D) $O(\log(n))$ (E) None

Question 2 . . . . 10 (0.5pt / question) points

(a) Given the interfaces `Map` and `LocNot` below, write the method `int nbNots(Map<String, Queue<LocNot>> ind, String w, double t1, double g1, double t2, double g2)` which takes as input an index map, where the key is the word and data is a queue containing all notifications where the word appears. The method returns the number of notifications containing the word `w` and located within the rectangle having bottom left corner at `(t1, g1)` and upper right corner at `(t2, g2)`.

```
public interface Map<K extends Comparable<
    K>, T> {
  boolean empty();
  boolean full();
  T retrieve();
  void update(T e);
  boolean find(K key);
  boolean insert(K key, T data);
  boolean remove(K key);
}
```

```
public interface LocNot {
  double getLat(); // Latitude
  double getLng(); // Longitutde
  int getMaxNbRepeats();
  int getNbRepeats();
  String getText();
}
```

Complete the code below by choosing the correct answer:

```
1  int nbNots(Map<String, Queue<LocNot>> ind, String w, double t1, double g1, double t2,
       double g2) {
2    if (...)
3      return 0;
4    int cpt = 0;
5    Queue<LocNot> q = ...;
6    ... {
7      LocNot not = ...;
8      ...;
9      double t = ...;
10     double g = ...;
11     if (...)
12       ...;
13   }
14   ...
15 }
```

- Line 2:
  - (A) `if (ind.find(w))`
  - (B) `if (!ind.find(w))`
  - (C) `if (ind.find(w)== null)`
  - (D) `if (ind.retrieve(w)== null)`
  - (E) None

- Line 5:
  - (A) `Queue<LocNot> q = ind.find(w);`
  - (B) `Queue<LocNot> q = ind.remove(w);`
  - (C) `Queue<LocNot> q = ind.retrieve(w);`
  - (D) `Queue<LocNot> q = ind.retrieve();`
  - (E) None

- Line 6:
  - (A) `while (!q.empty()){`
  - (B) `for (int i = 0; i <= q.length(); i++){`
  - (C) `while (!q.last()){`
  - (D) `for (int i = 0; i < q.length(); i++){`
  - (E) None

- Line 7:
  - (A) `LocNot not = q.serve();`
  - (B) `LocNot not = q.head.data;`
  - (C) `LocNot not = q.retrieve();`
  - (D) `LocNot not = q.pop();`
  - (E) None

- Line 8:
  - (A) `q.push(not);`
  - (B) `q.serve();`
  - (C) `q.insert(not);`
  - (D) `q.enqueue();`
  - (E) None
- Line 9:
  - (A) `double t = not.getLat();`
  - (B) `double t = q.retrieve().getLat();`
  - (C) `double t = q.serve().getLat();`
  - (D) `double t = q.pop().getLat();`
  - (E) None
- Line 10:
  - (A) `double g = q.serve().getLng();`
  - (B) `double g = q.retrieve().getLng();`
  - (C) `double g = not.getLng();`
  - (D) `double g = q.pop().getLng();`
  - (E) None
- Line 11:
  - (A) `if (t1<=t && t<=t2 && g1<=g && g<=g2)`
  - (B) `if (t1<=t && t>=t2 && g1<=g && g>=g2)`
  - (C) `if (t1<=t && t<=t2 || g1<=g && g<=g2)`
  - (D) `if (t1<=t && t<=t2 && g1>=g && g>=g2)`
  - (E) None
- Line 12:
  - (A) `return cpt;`
  - (B) `{cpt++; break;}`
  - (C) `cpt++;`
  - (D) `break;`
  - (E) None
- Line 14:
  - (A) `return cpt;`
  - (B) `return q.length()- cpt;`
  - (C) `return q.length()+ cpt;`
  - (D) `return q.length();`
  - (E) None

(b) Write the method `Stack<LocNot> copyNots(Map<String, Stack<LocNot>> ind, String w)` which takes as input an index map, where the key is the word and data is a stack containing all notifications where the word appears. The method returns **a copy** of the stack of notifications where the word `w` appears. If `w` does not exists, an empty stack is returned.

Complete the code below by choosing the correct answer:

```java
1  Stack<LocNot> copyNots(Map<String, Stack<LocNot>> ind, String w) {
2    Stack<LocNot> rs = new LinkedStack<LocNot>();
3    if (...)
4      return ...;
5    Stack<LocNot> ts = ...;
6    Stack<LocNot> st = ...;
7    while (...) {
8      ...;
9    }
10   while (...) {
11     LocNot not = ...;
12     ...;
13     ...;
14   }
15   return rs;
16 }
```

- Line 3:
  - (A) `if (ind.find(w))`
  - (B) `if (!ind.find(w))`
  - (C) `if (ind.retrieve(w)== null)`
  - (D) `if (ind.find(w)== null)`
  - (E) None
- Line 4:
  - (A) `return ind.retrieve();`
  - (B) `return null;`
  - (C) `return rs.empty();`
  - (D) `return rs;`
  - (E) None
- Line 5:
  - (A) `Stack<LocNot> ts = null;`
  - (B) `Stack<LocNot> ts = new LinkedStack<LocNot>();`
  - (C) `Stack<LocNot> ts = new LinkedStack<String>();`
  - (D) `Stack<LocNot> ts = new Stack<LocNot>();`
  - (E) None
- Line 6:
  - (A) `Stack<LocNot> st = ind.serve();`
  - (B) `Stack<LocNot> st = ind.retrieve();`
  - (C) `Stack<LocNot> st = ind.pop();`
  - (D) `Stack<LocNot> st = ind.retrieve(w);`
  - (E) None
- Line 7:
  - (A) `while (!st.empty()){`
  - (B) `while (!st.last()){`
  - (C) `while (st.empty()){`
  - (D) `while (st.length()> 0){`
  - (E) None
- Line 8:
  - (A) `ts.pop(st.pop());`
  - (B) `st.push(st.push());`
  - (C) `ts.enqueue(st.serve());`
  - (D) `ts.push(st.pop());`
  - (E) None
- Line 10:
  - (A) `while (!ts.empty()){`
  - (B) `while (!ts.last()){`
  - (C) `while (!st.empty()){`
  - (D) `while (ts.empty()){`
  - (E) None
- Line 11:
  - (A) `LocNot not = ts.pop();`
  - (B) `LocNot not = rs.pop();`
  - (C) `LocNot not = ts.push();`
  - (D) `LocNot not = st.pop();`
  - (E) None
- Line 12:
  - (A) `st.push(st.pop());`
  - (B) `st.push(not);`
  - (C) `st.push(rs.pop());`
  - (D) `st.push(ts.pop());`
  - (E) None
- Line 13:
  - (A) `rs.push(ts.pop());`
  - (B) `rs.push(st.pop());`
  - (C) `rs.push(not);`
  - (D) `ts.push(rs.pop());`
  - (E) None

Question 3 . . . . 10 (Part (a) 1.5pts / question. Part (b) 1pt / question) points

(a) Write the method **public boolean** `isBal()`, member of the BT class, which returns true if the BT is a balanced, and false otherwise. A BT is balanced if for each node, the absolute difference in height of its two subtrees is at most 1. Assume you have a method called **private int** `height(BTNode<T> p)` that returns the height the sub-tree `p`. The method `isBal()` makes a call to the recursive method **private boolean** `isBalRec(BTNode<T> p)`. Choose the correct option to complete the code of these methods:

```
1 public boolean isBal() {
2   ...
3 }
4 private boolean isBalRec(BTNode<T> p) {
5   ...
6   ...
7   ...
8 }
```

1. Line 2:
   - A **return** `isBalRec(root.left)|| isBalRec(root.right);`
   - B **return** `isBalRec(root.left)&& isBalRec(root.right);`
   - C **return** `!isBalRec(root.left)&& !isBalRec(root.right);`
   - D **return** `isBalRec(root);`
   - E None
2. Line 5:
   - A **if** `(p == null)`**return false**;
   - B **if** `(p == null)`**return true**;
   - C **if** `(p != null)`**return true**;
   - D **if** `(p != null)`**return false**;
   - E None
3. Line 6:
   - A **if** `(Math.abs(height(p.right)- height(p.left))>= 1)`**return true**;
   - B **if** `(Math.abs(height(p.right)- height(p.left))>= 2)`**return false**;
   - C **if** `(Math.abs(height(p.right)- height(p.left))<= 2)`**return false**;
   - D **if** `(Math.abs(height(p.right)- height(p.left))!= 0)`**return false**;
   - E None
4. Line 7:
   - A **return** `!isBalRec(p.left)&& !isBalRec(p.right);`
   - B **return** `isBalRec(p.left)+isBalRec(p.right);`
   - C **return** `isBalRec(p.left)&& isBalRec(p.right);`
   - D **return** `isBalRec(p.left)|| isBalRec(p.right);`
   - E None

(b) Consider the function `f` below, member of `DoubleLinkedList`:

```
public void f(int n) {
  Node<T> p = head;
  for(int i = 0; i < n; i++) {
    if (p.next != null)
      p = p.next;
  }
  p.previous.next = p.next;
  if (p.next != null)
    p.next.previous = p.previous;
  p.next = head;
  p.next.previous = p;
  p.previous = null;
  head = p;
}
```

Choose the correct result in each of the following cases:

1. The list `l`: A, B, C, D, E, after calling `l.f(3)`, `l` becomes:
   (A) B, C, D, E **(B) D, A, B, C, E** (C) E, B, C, D (D) A, D, E, B, C (E) None
2. The list `l`: A, B, C, D, E, after calling `l.f(1)`, `l` becomes:
   (A) A, B, C (B) E, A, B, C, D (C) B, C, D, E, A **(D) B, A, C, D, E** (E) None
3. The list `l`: A, B, C, D, E, after calling `l.f(5)`, `l` becomes:
   (A) A **(B) E, A, B, C, D** (C) A, B, C, D, E **(D) E, A, B, C, D** (E) None
4. The list `l`: A, B, C, D, E, after calling `l.f(2)`, `l` becomes:
   (A) *empty* (B) E, D, C, B, A **(C) C, A, B, D, E** (D) E, C, D (E) None

Question 4 . . . . 14 (2pts / question) points

(a) Consider the following heap represented as an array: 2, 7, 5, 8, 20, 10, 12. Choose the correct answer for every operation (all operations are done on the above heap).

1. Heap after inserting 1: **(A) 1, 2, 5, 7, 20, 10, 12, 8** (B) 1, 2, 5, 7, 20, 10, 8, 12 (C) 2, 5, 7, 20, 10, 12, 8, 1 (D) 2, 5, 7, 20, 10, 12, 1, 8 (E) None
2. Heap after inserting 3 then 4: (A) 2, 3, 4, 5, 20, 10, 12, 8, 7 **(B) 2, 3, 5, 4, 20, 10, 12, 8, 7** (C) 2, 3, 4, 5, 8, 7, 20, 10, 12 (D) 2, 3, 4, 5, 8, 10, 12, 7, 20 (E) None
3. Heap after inserting 11 then deleting one key: (A) 11, 2, 7, 5, 8, 20, 10, 12 (B) 5, 3, 4, 20, 10, 12, 8, 7 (C) 5, 7, 11, 8, 20, 10, 12 (D) 5, 7, 10, 8, 20, 12, 11 **(E) None**
4. Heap after deleting two keys: (A) 2, 7, 5, 8, 20 (B) 2, 5, 7, 20, 8 **(C) 7, 8, 10, 12, 20** (D) 7, 10, 8, 12, 20 (E) None

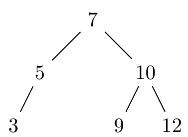
(b) Suppose we have two heaps (5, 9, 6) and (7, 8, 10) represented as arrays and a key 12, what will be the resultant heap after merging them? (A) 12, 5, 9, 6, 7, 8, 10 (B) 5, 9, 6, 12, 7, 8, 10 **(C) 5, 6, 7, 9, 12, 8, 10** (D) 5, 9, 6, 7, 8, 10, 12 (E) None

(c) What is the result of a bottom-up min-heap construction of the array 5, 11, 2, 7, 16, 15, 4? **(A) 2, 7, 4, 11, 16, 15, 5** (B) 5, 11, 2, 7, 16, 15, 4 (C) 2, 7, 5, 11, 16, 15, 4 (D) 2, 4, 7, 11, 16, 15, 5 (E) None

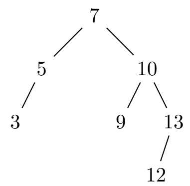
(d) What is the height of a heap containing 10 elements? **(A) 3** (B) 10 **(C) 4** (D) 5 (E) None.

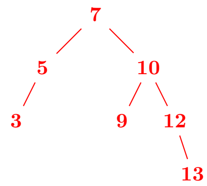
Question 5 . . . . 14 (2pts / question) points

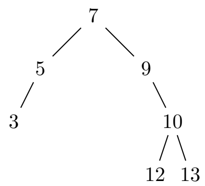
Choose the correct result in each of the following cases (follow the the convention of replacing with the smallest key in the right sub-tree when necessary):

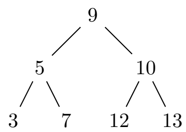
1. After inserting the key 13 in the AVL 

, the tree becomes:

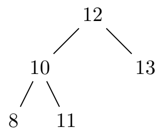
(A) 

(B) 

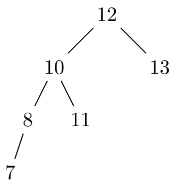
(C) 

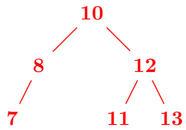
(D) 

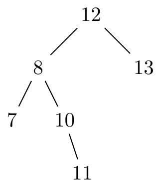
(E) None

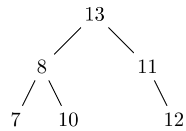
2. After inserting the key 7 in the AVL 

, the tree becomes:

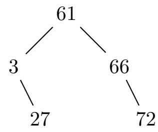
(A) 

(B) 

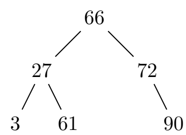
(C) 

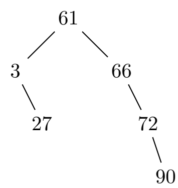
(D) 

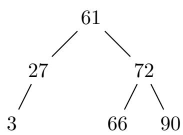
(E) None

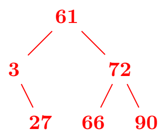
3. After inserting the key 90 in the AVL 

, the tree becomes:

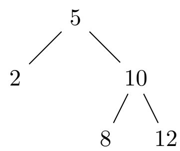
(A) 

(B) 

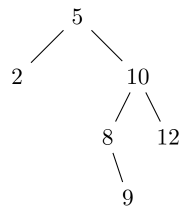
(C) 

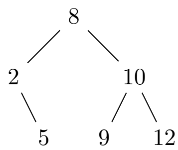
(D) 

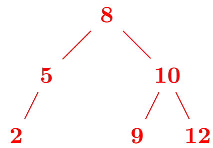
(E) None

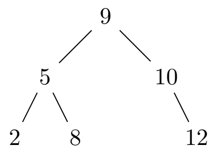
4. After inserting the key 9 in the AVL 

, the tree becomes:

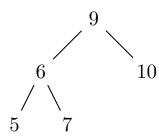
(A) 

(B) 

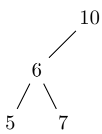
(C) 

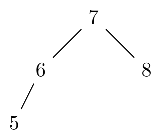
(D) 

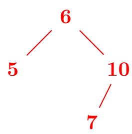
(E) None

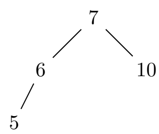
5. After deleting the key 9 from the AVL 

, the tree becomes:

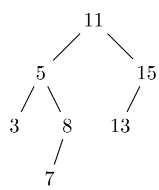
(A) 

(B) 

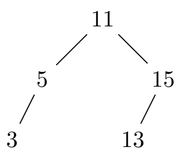
(C) 

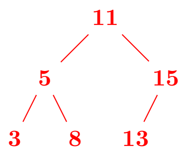
(D) 

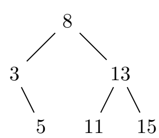
(E) None

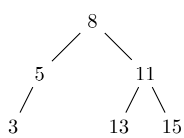
6. After deleting the key 7 from the AVL 

, the tree becomes:

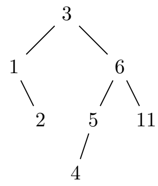
(A) 

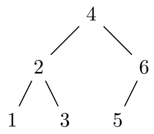
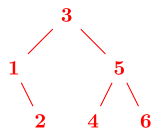
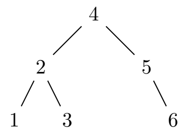
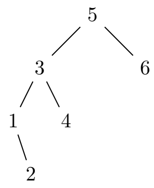
(B) 

(C) 

(D) 

(E) None

7. After deleting the key 11 from the AVL  , the tree becomes:

Ⓐ  Ⓑ  Ⓒ  Ⓓ  Ⓔ None

Question 6 . . . . 14 ((45 - number of mistakes) / 45 * 14 then rounded up to the nearest 0.5) points

Use the hash function $H(key) = key\%9$ to store the sequence of keys $21, 15, 18, 12, 27, 30, 35, 19, 10$ in a hash table of size 9. Use the following collision resolution strategies:

1. Linear rehashing (c=1). Fill in the following table:

Key	21	15	18	12	27	30	35	19	10
Position	3	6	0	4	1	5	8	2	7
Number of probes	1	1	1	2	2	3	1	2	7

2. External chaining. Fill in the following table:

Key	21	15	18	12	27	30	35	19	10
Index of the list	3	6	0	3	0	3	8	1	1

3. Coalesced chaining with cellar size 3 and address region size 7 (you must change the hash function to $H(key) = key\%7$.) Fill in the following table (put -1 if there is no next element):

Key	21	15	18	12	27	30	35	19	10
Position	0	1	4	5	6	2	9	8	3
Index of next element	9	-1	-1	8	-1	-1	-1	-1	-1

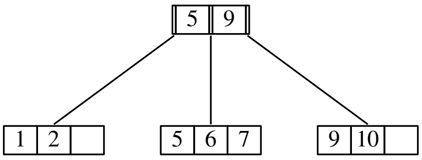
Question 7 . . . . 14 (2pts / question) points

Choose the correct result in each of the following cases (when possible, always borrow and transfer to the left):

1. After inserting the key 2 in the B+ tree [1 | 3 | 6], the **root** of tree becomes:

(A) [3 |] (B) [3 | 6] **(C)** [3 |] (D) [6 |] (E) None

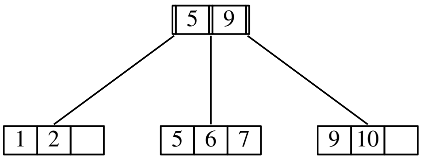
2. After inserting the key 4 in the B+ tree



, the **root** of the tree becomes:

(A) [4 | 7] **(B)** [5 | 9] (C) [6 |] (D) [4 |] (E) None

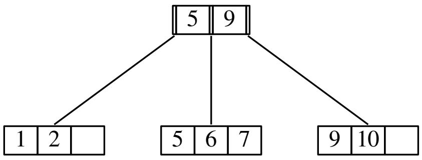
3. After inserting the key 8 in the B+ tree



, the **root** of the tree becomes:

(A) [5 | 9] **(B)** [6 | 9] (C) [6 |] (D) [6 | 10] (E) None

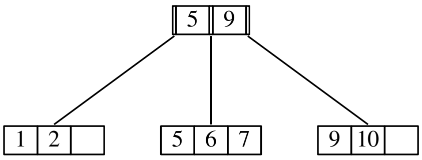
4. After deleting the key 2 from the B+ tree



, the **root** of the tree becomes:

(A) [9 |] (B) [2 | 9] **(C)** [6 | 9] (D) [7 | 9] (E) None

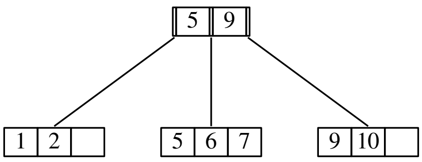
5. After deleting the key 10 from the B+ tree



, the **root** of the tree becomes:

(A) [9 |] (B) [7 |] (C) [6 |] **(D)** [5 | 7] (E) None

6. After deleting the key 6 from the B+ tree



, the **root** of the tree becomes:

(A) [2 | 9] **(B)** [5 | 9] (C) [6 |] (D) [5 |] (E) None

7. The leaves of a B+ tree of order 5 can contain the following number of data elements:

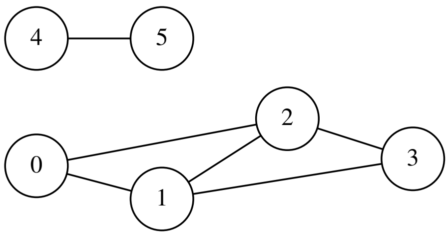
(A) 2 to 5 elements **(B) 3 to 5 elements** (C) 4 to 5 elements (D) 0 to 5 elements (E) None

Question 8 .... 8 points

(1) 3pts: -0.5pts for every incorrect edge (missing or extra); (2) 3pts: -0.25pts for every incorrect edge (missing or extra) and round up to the nearest 0.5; (3) 1pt; (4) 1pt.

1. Given the following adjacency list, draw the graph it represents.

0	→ 1 → 2
1	→ 0 → 2 → 3
2	→ 0 → 1 → 3
3	→ 1 → 2
4	→ 5
5	→ 4



2. Give the adjacency matrix representation of the graph.

	0	1	2	3	4	5
0		1	1			
1	1		1	1		
2	1	1		1		
3		1	1			
4						1
5					1	

3. This graph is connected: [**False**]

4. This graph has a cycle: [**True**]

ADT Queue Specification

- enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.

ADT Stack Specification

- push(Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- pop(Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
- empty(boolean flag): **requires**: none. **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
- full(boolean flag): **requires**: none. **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.