Data Structures and Algorithms

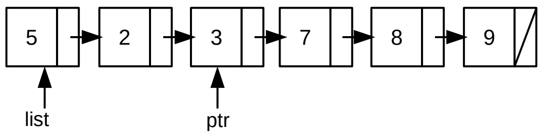
Time: 90 Minutes — Sessional 1 – March 27, 2019 — Total Marks: 50

Q1: Convert the following expression into postfix. {10}

```
((3+2-1)/2*(5-1))/2
```

Input	**Stack**	**Output**
(	(	
(	((	
3	((	3
+	((+	3
2	((+	32
-	((-	32+
1	((-	32+1
)	(	32+1-
/	(/	32+1-
2	(/	32+1-2
*	(*	32+1-2/
(	(*(	32+1-2/
5	(*(	32+1-2/5
-	(*(-	32+1-2/5
1	(*(-	32+1-2/51
)	(*	32+1-2/51-
)		32+1-2/51-*
/	/	32+1-2/51-*
2	/	32+1-2/51-*2
		32+1-2/51-*2/

Q2: Consider the following linked list with two pointers *list* and *ptr* pointing to the nodes 5 and 3 respectively. Write the result of the following statements, if you think there is an error in the statement, discuss it: {10}



a) `list→next == 2`
Error, because list→next is a pointer and 2 is an integer

b) `ptr→next→next→next→next`
NULL

c) `list→next→next != ptr`
False

d) `ptr→prev→data`
Error, because *prev* is not part of the structure node.

e) `list→next→next→data == ptr`
Error, because *data* is of type integer and *ptr* is a pointer

Output for the parts b and c:

```
b) 0x0
c) 0
```

Errors generated by the compiler for the other parts:

```
a)
s1-q2.cpp:29:10: error: comparison between pointer and integer ('Node *' and 'int')
    list == 2;
    ~~~~ ^  ~

d)
s1-q2.cpp:38:10: error: no member named 'prev' in 'Node'
    ptr->prev->data;
    ~~~  ^

e)
s1-q2.cpp:42:28: error: comparison between pointer and integer ('int' and 'Node *')
    list->next->next->data == ptr;
```

Q3: Implement a function (client code) *void ReverseQueue(Queue &q)*, which takes a queue as a parameter and reverses it. {10}

```
void ReverseQueue(queue<T> &q) {
 stack<T> s;

 while(!q.empty()) {
    T v;
    v = q.front(); //read the value at front of the queue
    q.pop();        //deque
    s.push(v);   //push on to a stack
 }

 while(!s.empty()) {
    T v;
    v = s.top();    //read the top most element
    s.pop();        //pop it from the stack
    q.push(v);   //enque it in the queue
 }
}
```

Q4: Write a program that takes an arithmetic expression as input. The program outputs whether the expression contains matching grouping symbols. For example, the arithmetic expressions `{2+(3−6)*8}` and `7+8*2` contains matching grouping symbols. However, the expression `5+{(1+7)/8-2*9` and `{(2+3})` do not contain matching grouping symbols. {10}

```
bool CheckBalanced(string expr) {
 stack<char> s;
 char top;
 for (int i=0;i<expr.length();i++) {
    switch (expr[i]) {
       //If it is an opening bracket, put it onto the stack.
       case '(': case '{': case '[':
          s.push(expr[i]);
          break;

       //If it is a closing bracket, pop the bracket from the stack and
       //compare it with the closing bracket, they should match.

       //If there is a closing bracket and the stack is empty,
       //there are more closing brackets than the opening brackets,
       //return false.
       case ')':
          if (s.empty())
             return false;
          top = s.top();
          s.pop();
          if (top != '(')
             return false;
          break;
       case '}':
          if (s.empty())
             return false;
          top = s.top();
          s.pop();
          if (top != '{')
             return false;
          break;
       case ']':
          if (s.empty())
             return false;
          top = s.top();
          s.pop();
          if (top != '[')
             return false;
          break;
    }
 }

 //If there is no bracket left on the stack, that means all brackets are matched, return true.
 if (s.empty())
    return true;
 //If there are some unmatched brackets on top of the stack, return false.
 else
    return false;
}
```

Q5: Write a function *void Stack::MoveToTop(T val)* for the linked structures implementation of the Stack ADT. The function should take a parameter *val*, if *val* is found in the stack, it should swap *val* with the value on *top* of the stack. For example, if the stack contains `5, 3, 2, 7` and *MoveToTop(3)* is called, the stack should become `5,` **`7,`** `2,` **`3.`** {10}

```
void MoveToTop(T val) {
    Node<T> *temp;
    temp = top;
    while (temp != NULL) {
        if (temp->data == val) {
            temp->data = top->data;
            top->data = val;
            break;
        }
        temp = temp->next;
    }
}
```