University of Nevada, Las Vegas Computer Science 477/677 Fall 2024

Assignment 2: Due Tuesday January 30 2024 11:59 PM

Upload your homework to Canvas. Miss Wallace will explain how that is done.

Name:____

You are permitted to work in groups, get help from others, read books, and use the internet. Your answers must be written in a pdf file and uploaded to canvas, by midnight September 2nd. Your file must not be unnecessarily long. If you have any questions, or you are having trouble uploading the assignment you may email the grader, Sebrina Wallace, at `wallace4@unlv.nevada.edu`. You may also send me email to ask questions.

1. Fill in the Blanks

   (a) The operators of type stack are **push**, **pop**, and **empty**.

   (b) The operators of type array are **fetch** and **store**.

   (c) Three kinds of priority queues are **stack**, **queue**, and **heap**.

2. Implementations of Stack

   (a) Write a C++ program that contains an array implementation of stack of integers. You may use either a struct or a class. Your stack should have a capacity of 10 integers. For ease of grading, do not submit a handwritten solution.

   (b) Write a C++ program that contains a linked list implementation of stack of integers. You may use either a struct or a class. For ease of grading, do not submit a handwritten solution.

   For both implementations, start with an empty stack, then execute the following, in this order, where s is a variable of type stack.

   (i) push(s,5)

   (ii) push(s,7)

   (iii) push(s,-3)

   (iv) write pop(s)

   (v) push(s,9);

   (vi) while(not empty(s))

           write pop(s)

   // stackarray.cpp

```
#include<iostream>
#include<cstdio>
#include<iomanip>
#include<cassert>
```

```
  #include<string>
  #include<cmath>
  #include<iostream>
  #include<cmath>
  #include<sstream>
  #include<stdio.h>
  #include<stdlib.h>
   using namespace std;

int const N = 10;

struct stack
 {
  int item[N];
  int size = 0;;
 };

void push(stack&s,int newitem) // pushes newitem onto s
 {
  assert(s.size < N);
  s.item[s.size] = newitem;
  s.size++;
 }

bool empty(stack s) // returns true if s is empty, false otherwise
 {
  return s.size == 0;
 }

int pop(stack&s) // returns and deletes most recently inserted item
 {
  assert(not empty(s)); // error if you try to pop an empty stack
  s.size --;
  return s.item[s.size];
 }

int main()
 {
  stack s; // defaults to s.size == 0
  push(s,5);
  push(s,7);
  push(s,-3);
  cout << pop(s) << endl;
  push(s,9);
```

```
    while(not empty(s))
     cout << pop(s) << endl;
    return 1;
   }
//
```

// stacklink.cpp

```
    #include<iostream>
    #include<cstdio>
    #include<iomanip>
    #include<cassert>
    #include<string>
    #include<cmath>
    #include<iostream>
    #include<cmath>
    #include<sstream>
    #include<stdio.h>
    #include<stdlib.h>
     using namespace std;

  struct stacknode;

  typedef stacknode*stack;

  struct stacknode
   {
    int item;
    stack link;
   };

  void push(stack&s,int newitem) // pushes newitem onto s
   {
    stack temp;
    temp = new stacknode;
    temp -> item = newitem;
    temp -> link = s;
    s = temp;
   }

  bool empty(stack s) // returns true if s is empty, false otherwise
   {
    return s == nullptr;
   }
  int pop(stack&s) // returns and deletes most recently inserted item
   {
    assert(not empty(s)); // error if you try to pop an empty stack
    int rslt = s->item;
    s = s->link;
    return rslt;
   }

  int main()
   {
    stack s; // defaults to s == NULL;
    push(s,5);
    push(s,7);
    push(s,-3);
    cout << pop(s) << endl;
    push(s,9);
    while(not empty(s))
     cout << pop(s) << endl;
    return 1;
   }
//
```

3. Implementations of Queue

   (a) Write a C++ program that contains an array implementation of queue of integers. You may use either a struct or a class. Your queue should hold 10 integers. For ease of grading, do not submit a handwritten solution.

   (b) Write a C++ program that contains a linked list implementation of queue of integers. You may use either a struct or a class.

   For both implementations, start with an empty queue, then execute the following, in this order, where q is a variable of type queue. For ease of grading, do not submit a handwritten solution.

   (i) enqueue(q,5)

   (ii) enqueue(s,7)

   (iii) enqueue(q,-3)

   (iv) write dequeue(q)

   (v) enqueue(q,9)

   (vi) while(not empty(q))

     write dequeue(q)

   // queuearray.cpp

```
    #include<iostream>
```

```
  #include<cstdio>
  #include<iomanip>
  #include<cassert>
  #include<string>
  #include<cmath>
  #include<iostream>
  #include<cmath>
  #include<sstream>
  #include<stdio.h>
  #include<stdlib.h>
   using namespace std;

int const N = 10;

struct queue
 {
  int item[N];
  int front = 0;
  int rear = 0;
 };

void enqueue(queue&q,int newitem) // inserts newitem into q
 {
  assert(q.rear < N-1);
  q.item[q.rear] = newitem;
  q.rear++;
 }

bool empty(queue&q) // returns true if q is empty, false otherwise
 {
  return q.front == q.rear;
 }

int dequeue(queue&q) // returns and deletes least recently inserted item
 {
  assert(not empty(q)); // error if you try to dequeue an empty queue
  int rslt = q.item[q.front];
  q.front++;
  return rslt;
 }

int main()
 {
  queue q; // defaults to q.front = q.rear = 0
```

```
    enqueue(q,5);
    enqueue(q,7);
    enqueue(q,-3);
    cout << dequeue(q) << endl;
    enqueue(q,9);
    while(not empty(q))
     cout << dequeue(q) << endl;
    return 1;
   }
//
```

// queuelink.cpp

```
    #include<iostream>
    #include<cstdio>
    #include<iomanip>
    #include<cassert>
    #include<string>
    #include<cmath>
    #include<iostream>
    #include<cmath>
    #include<sstream>
    #include<stdio.h>
    #include<stdlib.h>
     using namespace std;

  struct queuenode;

  typedef queuenode*queue;

  struct queuenode
   {
    int item;
    queue link; // points to the dummy queuenode
   };

  void initialize(queue&q)
   {
    q->link = q;
   }

  void enqueue(queue&q,int newitem) // inserts newitem at rear of q
   {
    queue temp;
    temp = new queuenode;
   q->item = newitem;
   temp->link = q->link;
   q->link = temp;
   q = temp;
  }

 bool empty(queue q) // returns true if q is empty, false otherwise
  {
   return q->link == q; // the dummy points to itself
  }

 int dequeue(queue&q) // returns and deletes least recently inserted item
  {
   assert(not empty(q)); // error if you try to dequeue an empty queue
   int rslt = q->link->item;
   q->link = q->link->link;
   return rslt;
  }

 int main()
  {
   queue q;
   q = new queuenode;
   initialize(q);
   enqueue(q,5);
   enqueue(q,7);
   enqueue(q,-3);
   cout << dequeue(q) << endl;
   enqueue(q,9);
   while(not empty(q))
    cout << dequeue(q) << endl;
   return 1;
  }

//
```

4. There is a stack algorithm for converting an infix expression to postfix. (You can find it if you look.) Write that algorithm in pseudocode. Hint: in postfix or prefix expressions, subtraction is indicated with a minus sign, but negation by a tilde, such as $\sim x$ (prefix) or $x \sim$ (postfix). (This algorithm is used every day by compilers.) You may write your answer by hand if you wish.

   The algorithm operates by examining the top of the stack and the next unread input symbol, called lookahead. (In reality, the lookahead is read and stored in a buffer.)

   1. Initialize the stack to be empty.
   2. Execute the following until both the stack and the input file are empty.

(a) If the lookahead is an identifier or a numeral, read it and write it to output.
(b) If the lookahead is an operator symbol, and the stack is empty, or the top of the stack is a left parenthesis, read the lookahead and push it onto the stack.
(c) If the lookahead is an operator symbol, and the top of the stack is an operator symbol of lower priority, read the lookahead and push it onto the stack.
(d) If the lookahead is an operator symbol and the top of the stack is an operator symbol of higher priority, pop the symbol from the stack and write it to output.
(e) If the lookahead is a left parenthesis, read it and push it onto the stack.
(f) If the lookahead is a right parenthesis and the top of the stack is an operator symbol, pop that operator symbol and write it.
(g) If the lookahead is a right parenthesis and the top of the stack is a left parentheses, read the lookahead and pop the stack and write nothing.
(h) If the input file is empty, pop the top of the stack and write it to output.

Use the algorithm to convert the infix expression $x * (-y - z) + x$ to postfix, showing the stack at each step.

Use \$ for both the bottom of the stack and the end of file symbol. Here is the execution of the algorithm with input file $x * (-y - z) + x$.

stack	input	output	action
\$	x∗(−y−z)+x\$		1
\$	∗(−y−z)+x\$	x	2(a)
\$∗	(−y−z)+x\$	x	2(b)
\$∗(	−y−z)+x\$	x	2(e)
\$∗(∼	y−z)+x\$	x	2(b)
\$∗(∼	−z)+x\$	xy	2(a)
\$∗(	−z)+x\$	xy∼	2(d)
\$∗(−	z)+x\$	xy∼	2(b)
\$∗(−	)+x\$	xy∼z	2(a)
\$∗(	)+x\$	xy∼z−	2(f)
\$∗	+x\$	xy∼z−	2(g)
\$	+x\$	xy∼z−∗	2(d)
\$+	x\$	xy∼z−∗	2(b)
\$+	\$	xy∼z−∗x	2(a)
\$	\$	xy∼z−∗x+	2(h)