Assignment 3: Stacks

Jake Gendreau

March 1, 2024

Contents

1 Program Design **2**
- 1.1 Time Estimate . . . 2
- 1.2 Data Structures . . . 2
  - 1.2.1 Stack . . . 2
  - 1.2.2 Queue . . . 2
- 1.3 Program . . . 2
  - 1.3.1 `main()` . . . 2
  - 1.3.2 Functions . . . 3

2 Program Log **5**
- 2.1 Time Requirements . . . 5
- 2.2 Things I encountered . . . 5

3 Source Code Files **6**
- 3.1 `stack.cpp` . . . 6
- 3.2 `stackADT.cpp` . . . 12
- 3.3 `stackADT.h` . . . 16

4 Program output **18**

1 Program Design

1.1 Time Estimate

I estimate that implementing this program will take 2-4 hours. The logic for conversion is provided in the assignment, making it mainly a task of input processing and stack implementation.

1.2 Data Structures

1.2.1 Stack

The stack is implemented in a separate file through the use of `stackADT.h` and its corresponding implementation, `stackADT.cpp`. It uses a linked list under the hood. All of the processing logic is done using a stack

Stack functions:

- `pushFront()` - Adds an item to the top of the stack.
- `pop()` - Returns and deletes the top item from the stack.
- `peek()` - Returns the top item from the stack.
- `print()` - Prints the stack, top to bottom.
- `size()` - Returns the size of the stack.
- `deleteList()` - Deletes the stack.

1.2.2 Queue

The queue is embedded in `stackADT.h` and `stackADT.cpp`, by the inclusion of one more function. All of the input processing and handling is done using a queue.

Queue Function:

- `pushBack()` - Adds an item to the bottom of the stack.

1.3 Program

1.3.1 `main()`

1. Make a queue called infix for input
2. Get infix using `getInfix(queue)`
3. While `getInfix(infix)` is true
   (a) Run `inToPost(infix)`
4. Quit the program

1.3.2 Functions

- `inToPost(infix)`
  1. Make stack
  2. For every token in infix
     (a) If the token is '(', push the token onto the stack
     (b) If the token is a number, push the token onto the stack
     (c) If the token is ')', call `handleClosedParens()`
     (d) If the token is an operator, call `handleOperators()`
  3. Delete the stack
  4. Delete the infix Queue
  5. Print the postfix expression
- `handleOperators(stack, token, postfix)`
  1. While `stack.peek()` has greater precedence than the token:
     (a) If `stack.pop()` is not '(', add it to the postfix expression
  2. Push token to the stack
- `handleClosedParens(stack, postfix)`
  1. Set stackToken = `stack.pop()`
  2. While stackToken is not '('
     (a) Add stackToken to the postfix expression
     (b) If `stack.size()` is greater than 0, stackToken = `stack.pop()`
- `getPrecedence(operator)`
  1. Check that the operator is valid, error out if not
  2. Using PEMDAS, evaluate and return the precedence of the operator, where * and / have the highest, then + and - have the lowest
- `isGreaterPrecedence(stackOperator, token)`
  1. Store the stack precedence and the token precedence using `getPrecedence()`
  2. Return true if the stack precedence is greater than the token precedence
  3. Return false otherwise
- `isNum(token)`
  1. For each character in token
     (a) Using character value comparison, if token < 0 or token > 9, return false

2. return true otherwise

- `getInfix(queue)`
  1. Make string buffer, string tmp, and bool offset
  2. Prompt for input
  3. Use getline to store the input in tmp
  4. Call `cleanInput(tmp)` to clean tmp
  5. If tmp is "quit", return a 0 to main
  6. For every character in tmp
     (a) Empty the buffer string
     (b) While the current character is a number (using `isNum()`)
        i. Add it to the buffer
        ii. Set offset to true
        iii. go the the next character
     (c) If offset is true
        i. Set offset to false
        ii. Go to the previous character
     (d) Store buffer at the end of the queue
  7. Store a ')' at the end of the infix expression
  8. Return a 1 to main

- `cleanExpression(expression)`
  1. If expression is "quit", return "quit"
  2. Make string buffer
  3. Make string containing all of the valid characters
  4. For every character in expression
     (a) If character is in the string of valid characters (using `isInString()`, add it to the buffer
  5. If the size of the buffer is 0, print an error and return "quit"
  6. Otherwise, return buffer

- `isInString(query, string)`
  1. For every character in the string
     (a) If query = the current character, return true
  2. Return false

2 Program Log

2.1 Time Requirements

This program took me about 9 hours to complete. However, the character implementation only took about 5 hours. Getting multi-character numbers to work took a while extra, since I had to convert the stack and queue to work with strings, along with all of the code involved in the logic.

2.2 Things I encountered

- I didn't understand how the provided process works until I tried to implement it from a more vague set of instructions. Once I understood it, implementing it was significantly easier.
- I initially had it reading character by character, which does work for the provided problems, but I wanted to make it work with all of the natural numbers instead. Getting that to work was a little harder, and I ended up implementing a que to hold the tokens.
- I learned how to use valgrind to check for memory leaks, and how to use the GDB CLI to debug programs. It was very helpful for keeping track of where bugs were coming from.
- I forgot how to make my .h and its implementation to compile, but a quick text to friends helped me out there.
- My pop() function kept throwing errors. It turns out that I didn't have any sort of integrated protection for reading outside of bounds. GDB helped me to identify and fix the issue.
- I finally broke down and learned LaTeX. I don't like it as much as markdown, but it's definitely better than HTML and has better organizational features than HTML or markdown.

3 Source Code Files

3.1 stack.cpp

Listing 1: stack.cpp

```
/*
*-------------------------------------------------------------
*Stack.cpp
*
*CS121.Bolden.............GCC 11.4.0.............Jake Gendreau
*Feb 27, 2024....Pop!_OS 22.04 / Intel Core i9-13900H.......
*
*..............gend0188@vandals.uidaho.edu..................
*
*Use a stack to convert a given infix
*expression to a postfix expression
*-------------------------------------------------------------
*/

//dependencies
#include <iostream>
#include "stackADT.h"

using namespace std;

//prototypes
string cleanExpression(string);

int getPrecedence(char);

void initStack(Stack&);

void handleClosedParens(Stack&, string&);
void handleOperators(Stack&, string, string&);
void inToPost(Stack&);

bool isNum(string);
bool isGreaterPrecedence(string, string);
bool isInString(char, string);
bool getInfix(Stack&);

int main()
{

    //prompt user
    Stack infix = Stack();

    //repeat until quit keyword is met
    while(getInfix(infix))
```

```
45      {
46          inToPost(infix);
47      }
48
49      cout << "Exiting Program..." << endl;
50
51      infix.deleteStack();
52  }
53
54  void inToPost(Stack &infix)
55  {
56      //init stack
57      Stack stack = Stack();
58      initStack(stack);
59
60      //init needed variables
61      string token;
62      string postfix;
63
64      //for every character in the infix string
65      while(infix.size() > 0)
66      {
67          token = infix.pop();
68
69          //handle open parens
70          if(token == "("
71              stack.pushFront("(");
72
73          //if token is an number, append it and move on
74          else if(isNum(token))
75              postfix += token + " ";
76
77          //handle closed parens
78          else if(token == ")" && stack.size() > 0)
79              handleClosedParens(stack, postfix);
80
81          //handle operators
82          else
83              handleOperators(stack, token, postfix);
84
85      }
86
87      //free the stack
88      stack.deleteStack();
89      infix.deleteStack();
90
91      //print the converted expression
92      cout << "Converted to Postfix: " << postfix << endl;
93  }
94
```

```
95 void handleOperators(Stack &stack, string token, string &postfix)
96 {
97     string stackToken;
98 
99     //iterate through whole stack, organizing operators as it goes
100     while(stack.size() > 0 && isGreaterPrecedence(stack.peek(), token))
101     {
102         stackToken = stack.pop();
103         if(stackToken != "(")
104             postfix += stackToken + " ";
105     }
106 
107     stack.pushFront(token);
108 }
109 
110 void handleClosedParens(Stack &stack, string &postfix)
111 {
112     string stackToken = stack.pop();
113 
114     //add everything in the stack on a closed parens
115     while(stackToken != "(")
116     {
117         postfix += stackToken + " ";
118 
119         if(stack.size() > 0)
120             stackToken = stack.pop();
121     }
122 }
123 
124 int getPrecedence(string op)
125 {
126     if(op.size() != 1){
127         cout << "ERROR: Invalid size in getPrecedence" << endl;
128         exit(-1);
129     }
130     //use PEMDAS to define priority
131     if(op[0] == '/' || op[0] == '*')
132         return 2;
133 
134     if(op[0] == '+' || op[0] == '-')
135         return 1;
136 
137     return 0;
138 }
139 
140 bool isGreaterPrecedence(string stackChar, string token)
141 {
142     //grab precedence values
143     int tokenVal = getPrecedence(token);
144     int stackVal = getPrecedence(stackChar);
```

```
145
146     //compare and return
147     return(stackVal >= tokenVal);
148 }
149
150 bool isNum(string token)
151 {
152     for(int i = 0; token[i] != '\0'; i++)
153     {
154         //true if token is a num, false otherwise
155         if(token[i] < '0' || token[i] > '9')
156             return false;
157     }
158
159     return true;
160 }
161
162 void initStack(Stack &stack)
163 {
164     stack.pushFront("(");
165 }
166
167 bool getInfix(Stack &stack)
168 {
169     string tmpString;
170     string buffer;
171     int offset = true;
172
173     //prompt the user and get the expression
174     cout << "Enter your Infix expression or type \"quit\" to exit: ";
175
176     //get the string
177     getline(cin, tmpString);
178
179     //clean it
180     tmpString = cleanExpression(tmpString);
181
182     //error checking
183     if(tmpString == "quit"){
184         return 0;
185     }
186
187     //create substrings and add them to the temp stack
188     for(int i = 0; tmpString[i] != '\0'; i++)
189     {
190         buffer = "";
191
192         //handle non-numbers
193         if(!isNum(string(1, tmpString[i])))
194         {
```

```
195             buffer = tmpString[i];
196         }
197 
198         //group numbers together
199         while(isNum(string(1, tmpString[i])) && tmpString[i] != '\0')
200         {
201             buffer += tmpString[i];
202 
203             //iterate through rest of list
204             i++;
205             offset++;
206         }
207 
208         //set i to where it should be
209         if(offset > 0){
210             offset = 0;
211             i--;
212         }
213 
214         //write it to the stack
215         stack.pushBack(buffer);
216     }
217 
218     stack.pushBack(")");
219 
220     return 1;
221 }
222 
223 string cleanExpression(string expression)
224 {
225     //check for quit
226     if(expression == "quit")
227         return("quit");
228 
229     string buffer;
230 
231     //define valid characters
232     string validChars = "()0123456789*+-/";
233 
234     //iterate through expression, comparing to validChars along the way
235     for(int i = 0; expression[i] != '\0'; i++)
236     {
237         if(isInString(expression[i], validChars))
238         {
239             //add to buffer if validChar
240             buffer += expression[i];
241         }
242     }
243 
244     //error checking
```

```
245     if(buffer.size() == 0)
246     {
247         cout << "Invalid statement." << endl;
248         return("quit");
249     }
250 
251     return buffer;
252 }
253 
254 bool isInString(char query, string string)
255 {
256     //iterate through string, return true if query in string
257     for(int i = 0; string[i] != '\0'; i++)
258     {
259         if(string[i] == query)
260             return true;
261     }
262     return false;
263 }
```

3.2 stackADT.cpp

Listing 2: stackADT.cpp

```
/*
stackADT.cpp
Class for a linked list of strings
*/

#include <iostream>

#include "stackADT.h"

using namespace std;

//add an item to the front of the list
void Stack::pushFront(string x)
{
    nodePtr n;

    //allocate new node
    n = new node;

    //error checking
    if(n == NULL)
    {
        cout << "ERROR: PUSH IS INVALID" << endl;
        exit(-1);
    }

    //set node
    n -> data = x;

    //set head = n if head is null
    if(head == NULL){
        head = n;
        n -> next = NULL;
    }

    //otherwise, append to the front of the list
    else{
        n -> next = head;
        head = n;
    }

    count++;
}

void Stack::pushBack(string x){
    nodePtr p = new node();
```

```
47      p->next = NULL;
48      p->data = x;
49
50      if (head == NULL) {
51          // If the list is empty, make the new node the head
52          head = p;
53      } else {
54          // Otherwise, find the last node and update its next pointer
55          nodePtr n = head;
56          while (n->next != NULL) {
57              n = n->next;
58          }
59          n->next = p;
60      }
61
62      count++;
63  }
64
65  string Stack::pop()
66  {
67      nodePtr n = head;
68
69      //error checking
70      if(n == NULL)
71      {
72          cout << "ERROR: POP ON EMPTY STACK" << endl;
73          exit(-1);
74      }
75
76      //store data to return
77      string returnChar = n -> data;
78
79      //reposition head
80      head = head -> next;
81
82      //cut link and delete
83      n -> next = NULL;
84      delete(n);
85
86      count--;
87
88      //return data
89      return returnChar;
90  }
91
92  //return the data in the first node
93  string Stack::peek()
94  {
95      //error checking
96      if(count == 0)
```

```
97      {
98          cout << "ERROR: PEEK ON EMPTY STACK" << endl;
99          exit(-1);
100     }
101
102     //return data @ head
103     return head -> data;
104 }
105
106 //print list
107 void Stack::print()
108 {
109     nodePtr p = head;
110
111     //traverse
112     while(p != NULL)
113     {
114         //print data @ current node
115         cout << p -> data << endl;
116         p = p -> next;
117     }
118 }
119
120 //identify if element is in list
121 bool Stack::isInList(string query)
122 {
123     nodePtr p = head;
124
125     //traverse the list
126     while(p != NULL)
127     {
128         //return true if query is found
129         if(p -> data == query)
130             return true;
131         p = p -> next;
132     }
133
134     //otherwise, return false
135     return false;
136 }
137
138 int Stack::size()
139 {
140     return count;
141 }
142
143 void Stack::deleteStack()
144 {
145     nodePtr p;
146
```

```
147     //traverse through the list
148     while(head != NULL)
149     {
150         //move head to the next node and delete current node
151         p = head;
152         head = head -> next;
153         delete(p);
154     }
155 
156     //finally, delete head
157     delete(head);
158 }
```

3.3 stackADT.h

Listing 3: stackADT.h

```
/*
stack.h
A header file for interfacing with a linked list
Jake Gendreau
Feb 26, 2024
*/

#ifndef STACK_H
#define STACK_H

using namespace std;

#include <iostream>

class Stack{
    private:
        struct node{
            string data;
            node* next;
        };
        typedef node* nodePtr;

        nodePtr head;

        int count;

    public:
        //constructor
        Stack(){
            //init
            head = NULL;
            count = 0;
        }

        //deconstructor
        ~Stack(){
            nodePtr p = head;
            nodePtr n;

            while(p != NULL){
                n = p;
                p = p -> next;
                delete n;
            }
        }

```

```
47          //add node onto the front of the list
48          void pushFront(string x);
49
50          //add node onto the front of the list
51          void pushBack(string x);
52
53          string pop();
54
55          //delete the first node found with the value x if one exists
56          void deleteNode(string x);
57
58          //return the first node found in the list
59          string peek();
60
61          //output the values in the nodes, one char per line
62          void print();
63
64          //return true if there is a node with the value x
65          bool isInList(string x);
66
67          //return count of the number of nodes in the list
68          int size();
69
70          //delete list
71          void deleteStack();
72  };
73
74  #endif
```

4 Program output

```
Enter your Infix expression or type "quit" to exit: 3 + 4
Converted to Postfix: 3 4 +
Enter your Infix expression or type "quit" to exit: (3 + 4)
Converted to Postfix: 3 4 +
Enter your Infix expression or type "quit" to exit: (2 + 4) * 3
Converted to Postfix: 2 4 + 3 *
Enter your Infix expression or type "quit" to exit: 2 + ((5) * 3)
Converted to Postfix: 2 5 3 * +
Enter your Infix expression or type "quit" to exit: 2 * ((3 + 4 * 7) / 3 + 2)
Converted to Postfix: 2 3 4 7 * + 3 / 2 + *
Enter your Infix expression or type "quit" to exit: 10 * 4
Converted to Postfix: 10 4 *
Enter your Infix expression or type "quit" to exit: quit
Exiting Program...
```