CSC2302

Data Structures using C language

Spring 2019

Project 2, 3

You need to submit two C programs; this work is counted as Project 2 and 3. In these projects, you will practice your CSC1401 knowledge and also the material we discussed in class about **stacks and queues implemented using Linked Lists**. The first C program will be about stacks and the second will be about queues. These are important points to consider:

- All your **4** homework count 15% of your overall grade
- You need to submit your homework Wednesday April 15th before midnight (April 15th is included)
  - Please note that any homework sent after 00:00 will not be considered: you get a zero as grade
- I need 1 submission from a team where you put your partner in CC
  - First program call it: lastNameStudent1_ lastNameStudent2_Stack.c
  - Second program call it: lastNameStudent1_ lastNameStudent2_Queue.c
    - Example: Moussa_Saber_Stack.c and Moussa_Saber_Queue.c
- Team => work has to be done together
- Any form of plagiarism will lead to a WF a course grade so please maintain the privacy of your work.

First Problem: Implementing a stack using Linked lists

For this problem you are asked to implement a stack using linked list in order to convert an infix expression to a prefix one! Remember, LIFO and the basic functions push(), pop() and you can also use check_top() to check the element in the top of the stack.

Refer to this two examples to understand what a prefix expression means:

Infix	Prefix
A+B	+AB
A+B*C	+A*BC

The steps you need to follow are the followings:

1. Put an infix expression (with parenthesis) in a file called infix.txt and make sure that it is balanced (you don't need to write a C code to check that)
2. Provide the user with the following menu
    1. Load infix expression to an array
    2. Reverse infix
    3. Convert infix to prefix
    4. Quit
3. Make sure that choices 2, 3 are not executed unless 1 is already chosen. Also make sure that choice 3 is not executed unless choices 1 and 2 are chosen
4. For choice 1, use a function called from_file_to_array() that will load the content of infix.txt into an array in the main that we will call infix_arr: (use DMA to allocate memory for this array)
5. For choice 2, you need to use a function called reverse_array() this function will use a stack implemented using LL to reverse the content of the array infix_arr: you don't need to use another temporary array!
6. For choice 3 you need to use a function called infix_prefix() that will produce a prefix expression using the new content of infix_arr. Let's use an array that we will call prefix_arr (declared in the main and use DMA to

allocate memory needed) that will hold the final result. Here is the algorithm that you need to use:

a. Scan the characters in infix_arr one by one
b. If the character is an operand, write down to prefix_arr
c. If the character is a closing parenthesis, then push it to the stack
d. If the character is an opening parenthesis, pop the elements in the stack (if it is an operator write it to prefix_arr) until a closing parenthesis is popped (do not write a parenthesis into prefix_arr)
e. If the character scanned is an operator check this:

- If stack empty, push it to stack
- If the operator has precedence greater than or equal to the top of the stack (here use check_top ()), push the operator to the stack.
- If the operator has precedence lesser than the top of the stack, pop the operator and output it to the prefix array then check the above condition again with the new top of the stack.

f. After all the characters are scanned make sure that the stack is empty otherwise pop characters from the stack and write them to your prefix array (if it is an operator)
g. The last thing: use the stack to reverse the prefix array without using any temporary array. **Print the new content of prefix_arr to screen**

7. For priorities, use the following function:

```
 int priority(char value){
if (value==’+’ || value==’-’)
  return(1);
else if (value==’*’ || value==’/’)
  return(2);
else if (value==’)’ || value==’(’)
  return(0);
}
```

8. Test your program for different infix expressions:
    a. (A-(B*C)+(D/E))
    b. A-(B*C)+D
    c. ……
9. No global variables and handle possible errors

Second Problem: Implementing a queue using Linked lists

When you get into a bank requesting a service, you get a ticket number corresponding to what service you are requesting. We assume that the bank provide you with the following services:

1. **Money transactions**
2. **Account Issues**
3. **Meet the bank director**

You need to implement a C program that manages the above services by providing the user with the following menu options:

1. Add a client to a queue
2. Serve a client
3. Switch tickets
4. Bank Service is over

Important Points to Consider:

- You need to implement 3 **queues (1 for each bank service) using Linear Linked List (not doubly)**
- Again no global variables
- Every menu choice will be solved using a user defined function
- Respect the rules to design a function
- A client will be represented by the following features: client name, ticket number, requested_service, and next pointer
- Besides the functions that you will create for every menu option, you need create another function used to create and fill the element of a new client: call this function create_fill_node()
    - In this function you need to ask for the client name and the requested service (should be one of these values: Money, Account, or Meeting.)
    - For the ticket number, your program should automatically assign an ordered ticket number to every new user without asking the user for

    the value: first user in a service X should have a ticket number 1, the second in the same queue will be assigned 2 as ticket number…..

- When the user chooses 1 from the menu you need to call create_fill_node() first then call a second function called enqueue_client() that will add the new client to the corresponding queue; this operation should take O(1). Remember you are working with three queues but you will use only 1 function to add a new client to the right queue! Make sure that you respect FIFO
- For the second menu option, you need to ask first for the service name then release any client (if any! Using dequeuer_client()) from the corresponding queue service
    - You need to use three files named: Money_S.txt, Account_S.txt and Meeting_S.txt where you need to put the name of every client served. Well possibly once or more of these files might be empty at the end!
- For the third choice: well sometimes when you have an X ticket number and you notice that another client in the same queue has an X+Y ticket number and he/she is in a hurry, you can volunteer to switch your ticket number. That's exactly what you need to do in option 3.
- For the last menu option: it's time for the bank office to end their services, yet some clients might be still waiting in the queues. For that you are asked to release/destroy from every queue any left client but make sure that you put the client name and service requested in a file called: clients_not_served.txt
- Handle possible errors