National Institute of Technology, Calicut

Department of Computer Science and Engineering

CS2092 Programming Lab

Assignment 3

Submission deadline (on or before): 16th October 2019, 10:00:00 PM

Policies for Submission and Evaluation

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Athena server. During evaluation your uploaded programs will be checked in Athena server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program.

Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. Detection of ANY malpractice regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz). The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>.zip (For example: ASSG3_BxxyyyyCS_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive. The source codes must be named as:

ASSG<NUMBER>_<ROLLNO>_<FIRSTNAME>_<PROGRAM-NUMBER>.<extension>
(For example: ASSG3_BxxyyyyCS_LAXMAN_1.c). If there is a part a and a part b or a particular question, then name the source files for each part separately as inASSG3_BxxyyyyCS_LAXMAN_1b.c.

If you do not conform to the above naming conventions, your submission might not be recognized by some automated tools, and hence will lead to a score of 0 for the submission. So, make sure that you follow the naming conventions.Standard of Conduct Violations of academic integrity will be severely penalized.

Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf .

Assignment Questions

General Instructions:

- `The input should be read from the file` ***`input.txt`*** `and output should be written to the file` **`output.txt.`**
- `You are` **`NOT`** `allowed to use` *`global variables`*`.` ***`You have to pass file pointer as an argument to functions appropriately.`***

1. Write a C program to create a singly linked list A and to insert a new element in the middle of A. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must implement the following functions:

main() - repeatedly read a character 'c', 'p', 'i' or 's' from the file and call the given functions appropriately until character 's' is entered.

create(A, fp1) - read the elements of the linked list from the input file pointed by file pointer ***fp1*** and create linked list ***A***.

print(A, fp2) - print the elements of the linked list ***A*** to the output file pointed by file pointer ***fp2***.

insert_middle(A, new_element) - insert the ***new_element*** in the middle of the linked list ***A***.

*Note: If **n** is the number of elements in the linked list, take the middle position as,* $\lceil n/2 \rceil + 1$
Example: When n = 4*, middle position =* $\lceil 4/2 \rceil + 1 = 3$
When n = 5, middle position = $\lceil 5/2 \rceil + 1 = 4$

Input format:

a) The input consists of multiple lines. Each line contains a character from {**'c'**, **'p'**, **'i'**, **'s'**} followed by zero or more integers.
b) Character **'c'**: Character 'c' is followed by integers separated by a space. Create an initial linked list with these integers as elements.
c) Character **'p'**: Print the elements of linked list, separated by a space. If the linked list is empty, print -1.
d) Character **'i'**: Character 'i' is followed by an integer to be inserted in the middle of the linked list.
e) Character **'s'**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

c 4 5 6 8 9

p

i 7

```
p

s
```

Sample Output:

```
4 5 6 8 9

4 5 6 7 8 9
```

2. Given a singly linked list L of integers, write a program to find the k^th^ node from the end of the linked list. Each node in the linked list has a *data* part that stores the integer and a *pointer* that points to the next node of the list. Assume that 'head' is a pointer that points to the start node of the list. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must implement the following functions:

main() - repeatedly read a character 'c', 'p', 'f' or 's' from the file and call the given functions appropriately until character 's' is entered.

create(L, fp1) - read the elements of the linked list from the input file pointed by file pointer ***fp1*** and create linked list ***L***.

print(L, fp2) - print the elements of the linked list ***L*** to the output file pointed by file pointer ***fp2***.

find_knode(L, k) - find and print the ***k***th node from the end of the linked list ***L***.

Input format:

a) The input consists of multiple lines. Each line contains a character from {'**c**', '**p**', '**f**', '**s**'} followed by zero or more integers.
b) Character **'c'**: Character 'c' is followed by integers separated by a space. Create an initial linked list with these integers as elements.
c) Character **'p'**: Print the elements of linked list, separated by a space. If the linked list is empty, print -1.
d) Character **'f'**: Character 'f' is followed by an integer ***k***. Find and print ***k***th node from the end of the linked list. If the linked list is empty or if ***k*** > ***n***, where ***n*** is the number of elements in the linked list, then print -1.
e) Character **'s'**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
c 1 12 43 4 5 15 22

p

f 5

s
```

Sample Output:

```
1 12 43 4 5 15 22

43
```

3. Given a doubly linked list A in which each node has a *data* part that stores an integer value and two pointers *next* and *prev* (*next* and *prev* points to the next and previous nodes respectively). Write a program that removes duplicate elements in the linked list. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must implement the following functions:

main() - repeatedly read a character 'c', 'p', 'r' or 's' from the file and call the given functions appropriately until character 's' is entered.

create(A, fp1) - read the elements of the linked list from the input file pointed by file pointer ***fp1*** and create linked list ***A***.

print(A, fp2) - print the elements of the linked list ***A*** to the output file pointed by file pointer ***fp2***.

remove_duplicate(A) - find and remove duplicate elements in the linked list ***A***.

Input format:

a) The input consists of multiple lines. Each line contains a character from {'**c**', '**p**', '**r**', '**s**'} followed by zero or more integers.
b) Character **'c'**: Character 'c' is followed by integers separated by a space. Create an initial linked list with these integers as elements.
c) Character **'p'**: Print the elements of linked list, separated by a space. If the linked list is empty, print -1.
d) Character **'r'**: Find and remove duplicate elements from the linked list.
e) Character **'s'**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
c  1 2 5 4 5 6 1 7

p

r

p

s
```

Sample Output:

```
1 2 5 4 5 6 1 7

1 2 5 4 6 7
```

4. Given a singly linked list L of integers, write a program to swap the elements in the linked list pairwise. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must implement the following functions:

   main() - repeatedly read a character ‘c’, ‘p’, ‘w’ or ‘s’ from the file and call the given functions appropriately until character ‘s’ is entered.

   create(L, fp1) - read the elements of the linked list from the input file pointed by file pointer ***fp1*** and create linked list ***L***.

   print(L, fp2) - print the elements of the linked list ***L*** to the output file pointed by file pointer ***fp2***.

   swap_pairwise(L) - swap the elements in the linked list ***L*** pairwise.

   Input format:

   a) The input consists of multiple lines. Each line contains a character from **{‘c**’, ‘**p**’, ‘**w**’, ‘**s**’**}** followed by zero or more integers.
   b) Character **‘c’**: Character ‘c’ is followed by integers separated by a space. Create an initial linked list with these integers as elements.
   c) Character **‘p’**: Print the elements of linked list, separated by a space. If the linked list is empty, print -1.
   d) Character **‘w’**: Swap the elements of the linked list pairwise.
   e) Character **‘s’**: Terminate the program.

   Output format:

   The output (if any) of each command should be printed on a separate line.

   Sample Input:

```
c 1 2 3 4 5

p

w

p

s
```

   Sample Output:

```
1 2 3 4 5

2 1 4 3 5
```

5. Write a program to reverse a singly linked list L. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must implement the following functions:

main() - repeatedly read a character 'c', 'p', 'r' or 's' from the file and call the given functions appropriately until character 's' is entered.

create(L, fp1) - read the elements of the linked list from the input file pointed by file pointer ***fp1*** and create linked list ***L***.

print(L, fp2) - print the elements of the linked list ***L*** to the output file pointed by file pointer ***fp2***.

reverse(L) - reverse the elements of the linked list ***L***.

Input format:

a) The input consists of multiple lines. Each line contains a character from **{'c', 'p', 'r', 's'}** followed by zero or more integers.
b) Character **'c'**: Character 'c' is followed by integers separated by a space. Create an initial linked list with these integers as elements.
c) Character **'p'**: Print the elements of linked list, separated by a space. If the linked list is empty, print -1.
d) Character **'r'**: Reverse the elements of the linked list.
e) Character **'s'**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
c 1 2 3 4 5 6 7

p

r

p

s
```

Sample Output:

```
1 2 3 4 5 6 7

7 6 5 4 3 2 1
```

6. Write a program to implement a queue using linked list. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program should include the following functions.

main() – repeatedly read a character 'e', 'd', 'p' or 's' from the file and call the given functions appropriately until character 's' is entered.

enqueue(Q, element) – insert the data specified by ***element*** into queue ***Q***.

dequeue(Q) – remove and return the element at the front of the queue ***Q***. Return -1 if the queue is empty.

print(Q, n, fp) – based on the value of ***n***, print the elements in the queue ***Q*** to the output file pointed by file pointer ***fp***.

- If ***n* = 0**, print all the elements in the queue ***Q***, starting with the element at the front.
- If ***n* > 0**, print the first ***n*** elements in the queue ***Q***, starting with the element at the front. If the number of elements in the queue is less than ***n***, then print all the elements in the queue ***Q*** followed by the string "END".
- If the queue is empty, print -1.

Input format:

Each line in the input file may contain,

a) character **'e'** followed by an integer which is to be enqueued into the queue.
b) character **'d'** to dequeue the element at the front of the queue and print the dequeued element. Print -1, if the queue is empty.
c) character **'p'** followed by an integer ***n***.
   - If ***n* = 0**, print all the elements in the queue starting with the element at the front, separated by a space.
   - If ***n* > 0**, print the first ***n*** elements in the queue starting with the element at the front, separated by a space. If the number of elements in the queue is less than ***n***, then print all the elements in the queue, separated by a space, followed by the string "END".
   - If the queue is empty, print -1.
d) character **'s'** to stop the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
e 2

e 5

e 7

p 0

d
```

```
p 2
e 10
d
p 3
d
d
d
p 6
e 3
e 8
p 1
s
```

Sample Output:

```
2 5 7
2
5 7
5
7 10 END
7
10
-1
-1
3
```

7. Write a program to implement a stack using array. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must support the following functions:

   main() - repeatedly read a character 'p', 't', 'd' or 's' from the file and call the given functions appropriately until character 's' is entered.

   push(stk, element) - put the data specified by ***element*** on top of the stack ***stk***. Return 999 if the stack is full.

pop(stk) - remove and return the topmost element of the stack ***stk***. Return -1, if the stack is empty.

print(stk, n, fp) – based on the value of ***n***, print the elements in the stack ***stk*** to the output file pointed by file pointer ***fp***.

- If ***n*** **= 0**, print all the elements in the stack ***stk***, starting with the element at the top.
- If ***n*** **> 0**, print the first ***n*** elements in the stack ***stk***, starting with the element at the top. If the number of elements in the stack is less than ***n***, then print all the elements in the stack ***stk*** followed by the string "END".
- If the stack is empty, print -1.

Input format:

First line of the input file contains an integer value ***c***, 0 < ***c*** < 1000, which is the capacity of the stack. Subsequent lines in the input file may contain,

a) character **'p'** followed by an integer which is to be pushed into the stack. Print 999, if the stack is full.
b) character **'t'** to pop and print the top most element of the stack. Print -1, if the stack is empty.
c) character **'d'** followed by an integer ***n***.
    - If ***n*** **= 0**, print all the elements in the stack starting with the element at the top, separated by a space.
    - If ***n*** **> 0**, print the first ***n*** elements in the stack starting with the element at the top, separated by a space. If the number of elements in the stack is less than ***n***, then print all the elements in the stack, separated by a space, followed by the string "END".
    - If the stack is empty, print -1.
d) character **'s'** to stop the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
3

p 2

p 5

p 7

d 0

t

d 1

p 10
```

```
p 11
t
d 3
t
t
t
d 1
p 3
d 5
s
```

Sample Output:

```
7 5 2
7
5
999
10
5 2 END
5
2
-1
-1
3 END
```

8. Write a program to implement a stack using Linked list. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must support the following functions:

main() - repeatedly read a character 'p', 't', 'd' or 's' from the file and call the given functions appropriately until character 's' is entered.

push(stk, element) - put the data specified by ***element*** on top of the stack ***stk***.

pop(stk) - remove and return the topmost element of the stack ***stk***. Return -1, if the stack is empty.

print(stk, n, fp) – based on the value of ***n***, print the elements in the stack ***stk*** to the output file pointed by file pointer ***fp***.

- If ***n*** **= 0**, print all the elements in the stack ***stk***, starting with the element at the top.
- If ***n*** **> 0**, print the first ***n*** elements in the stack ***stk***, starting with the element at the top. If the number of elements in the stack is less than ***n***, then print all the elements in the stack ***stk*** followed by the string "END".
- If the stack is empty, print -1.

Input format:

Each line in the input file may contain,

a) character **'p'** followed by an integer which is to be pushed into the stack.
b) character **'t'** to pop and print the top most element of the stack. Print -1, if the stack is empty.
c) character **'d'** followed by an integer ***n***.
    - If ***n*** **= 0**, print all the elements in the stack starting with the element at the top, separated by a space.
    - If ***n*** **> 0**, print the first ***n*** elements in the stack starting with the element at the top, separated by a space. If the number of elements in the stack is less than ***n***, then print all the elements in the stack, separated by a space, followed by the string "END".
    - If the stack is empty, print -1.
d) character **'s'** to stop the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
p 2

p 5

p 7

d 0

t

d 3

p 10

t

d 1
```

```
t
t
t
d 5
p 3
d 3
s
```

Sample Output:

```
7 5 2
7
5 2 END
10
5
5
2
-1
-1
3 END
```

9. Write a menu driven program to implement circular queue Q using array. The input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program must support the following functions:

   main() – repeatedly read a character ‘e’, ‘d’, ‘p’ or ‘s’ from the file and call the given functions appropriately until character ‘s’ is entered.

   enqueue(Q, element) – insert the data specified by ***element*** into circular queue ***Q***. Return -999, if the queue is full.

   dequeue(Q) – remove and return the element at the front of the circular queue ***Q***. Return -1, if the queue is empty.

   print(Q, n, fp) – based on the value of ***n***, print the elements in the circular queue ***Q*** to the output file pointed by file pointer ***fp***.

   - If ***n*** **= 0**, print all the elements in the queue ***Q***, starting with the element at the front.

- If ***n*** **> 0**, print the first ***n*** elements in the queue ***Q***, starting with the element at the front. If the number of elements in the queue is less than ***n***, then print all the elements in the queue ***Q*** followed by the string “END”.
- If the queue is empty, print -1.

Input format:

First line of the input file contains an integer value ***c***, 0 < ***c*** < 1000, which is the capacity of the queue. Subsequent lines in the input file may contain,

a) character **'e'** followed by an integer which is to be enqueued into the queue. Print -999 if the queue is full.
b) character **'d'** to dequeue the element at the front of the queue and print the dequeued element. Print -1, if the queue is empty.
c) character **'p'** followed by an integer ***n***.
    - If ***n*** **= 0**, print all the elements in the queue starting with the element at the front, separated by a space.
    - If ***n*** **> 0**, print the first ***n*** elements in the queue starting with the element at the front, separated by a space. If the number of elements in the queue is less than ***n***, then print all the elements in the queue, separated by a space, followed by the string “END”.
    - If the queue is empty, print -1.
d) character **'s'** to stop the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

```
3

e 2

e 5

e 7

p 0

d

p 1

e 10

e 11

d

p 3

d
```

d

d

p 7

e 3

p 4

s

Sample Output:

2 5 7

2

5

-999

5

7 10 END

7

10

-1

-1

3 END

10. Checking balanced parentheses: Write a program to examine whether the pairs and the orders of {, }, (,), [,] are correct in the given string expression. Input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program should include the following functions.

main() - repeatedly read a character ‘e’ or ‘s’ from the file and call the given function until character ‘s’ is entered.

checkParantheses(exp) - check whether the pairs and the orders of {, }, (,), [,] are correct in the given string expression ***exp***.

Input format:

a) The input consists of multiple lines. Each line contains a character from **{‘e’, ‘s’}**.
b) Character **‘e’**: Character ‘e’ is followed by a string expression, which is a combination of characters from {, }, (,), [,]. Check whether the pairs and the orders are correct in the given expression and print ‘YES’ if the parentheses are balanced. Else print ‘NO’.
c) Character **‘s’**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

e [()]{}{[()()]()}

e [(])

s

Sample Output:

YES
NO

11. Write a program to convert an infix expression into postfix expression.

Infix expression:

The expression of the form **a op b**, where an operator is in-between every pair of operands. Example: `a + b`

Postfix expression:

The expression of the form **a b op**, where the operands appear before their operators. Example: `a b +`

Input should be read from the file ***input.txt*** and output should be written to the file ***output.txt***. Your program should include the following functions.

main() - repeatedly read a character 'e' or 's' from the file and call the given function until character 's' is entered.

infixToPostfix(exp) - convert the given infix expression ***exp*** to postfix expression and print it.

Input format:

a) The input consists of multiple lines. Each line contains a character from **{'e', 's'}**.
b) Character **'e'**: Character 'e' is followed by an infix expression, which is a combination of characters(operands) from {a to z} and symbols(operator) like +, -, *, /, (,), ^. Convert it into postfix expression and print it.
c) Character **'s'**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

e a+b*c

e (a+b)*c

e (a+b)*c+d/(e+f*g)-h

s

Sample Output:

abc*+

ab+c*

ab+c*defg*+/+h-

12. Write a program for the evaluation of a postfix expression. Input should be read from the file ***input.txt*** and output should be written to the file ***output.txt*** Your program should include the following functions.

main() - repeatedly read a character ‘e’ or ‘s’ from the file and call the given function until character ‘s’ is entered.

evaluatePostfix(exp) - evaluate the given postfix expression ***exp*** and print the result.

Input format:

a) The input consists of multiple lines. Each line contains a character from **{‘e**’, ‘**s’}**.
b) Character **‘e’**: Character ‘e’ is followed by a postfix expression, which is a combination of integers(operands) and symbols(operator) like +, -, *, /, (,), ^, separated by a space. For negative integers, operator '-' followed by integer and no space is provided between '-' and the integer. Evaluate the postfix expression and print the result.
c) Character **‘s’**: Terminate the program.

Output format:

The output (if any) of each command should be printed on a separate line.

Sample Input:

e 2 3 1 * + 9 -

e 100 200 + 2 / 5 * 7 +

e -20 30 *

s

Sample Output:

-4

757

-600