Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Summer, Year:2022), B.Sc. in CSE (Day)

LAB REPORT NO # 04

Course Title: Data Structure Lab

Course Code: CSE 106 Section: PC-213DA

Student Details

Name	ID
Pankaj Mahanto	213902002

Lab Date	: 25/07/2022
Submission Date	: 07/08/2022
Course Teacher's Name	: Farhana Akter Sunny

Farhana Akter Sunny

Senior Professor

Green University of Bangladesh

[For Teachers use only: Don't Write anything inside the box]

Lab Report Status

Marks: . . . . Signature: . . . .

Comments: . . . . Date: . . . .

1. TITLE OF THE LAB EXPERIMENT [1]

- Implement a program of Circular Queue?
- Implement a program of infix to postfix expression using stack?

2. OBJECTION [1]

In this problem I will discuss Circular Queue and how it use?

3. PROCEDURE /ANALYSIS/DESIGN/PSEUDOCODE [2]

Algorithm to insert an element in a circular queue

```
Step 1: IF (REAR+1)%MAX = FRONT
            Write " OVERFLOW "
            Goto step 4
        [End OF IF]

Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT ! = 0
            SET REAR = 0
        ELSE
            SET REAR = (REAR + 1) % MAX
        [END OF IF]

Step 3: SET QUEUE[REAR] = VAL

Step 4: EXIT
```

Algorithm to delete an element from the circular queue

```
Step 1: IF FRONT = -1
            Write " UNDERFLOW "
            Goto Step 4
        [END of IF]

Step 2: SET VAL = QUEUE[FRONT]

Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                SET FRONT = 0
            ELSE
```

```
SET FRONT = FRONT + 1
[END of IF]
[END OF IF]
```

Step 4: EXIT

4. IMPLEMENTATION

Circular Queue:

```
// Circular Queue implementation in C

#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

int items[SIZE];
int front = -1, rear = -1;


// Adding an element
void enQueue()
{
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
        printf("\n Queue is full!! \n");
    else
    {
        int value;

        if (front == -1)
            front = 0;
        rear = (rear + 1) % SIZE;
        printf("\nwhich value enqueue :\n");
        scanf("%d", &value);
        items[rear] = value;
        printf("\n Inserted -> %d", value);
    }
}

// Removing an element
```

```
int deQueue()
{
    int value;
    if (front == -1)
    {
        printf("\n Queue is empty !! \n");
        return (-1);
    }
    else
    {
        value = items[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        // Q has only one element, so we reset the
        // queue after dequeing it. ?
        else
        {
            front = (front + 1) % SIZE;
        }
        printf("\n Deleted element -> %d \n", value);
    }
}

// Display the queue
void Display()
{
    int i;
    if (front == -1)
        printf(" \n Empty Queue\n");
    else
    {
        printf("\n Front -> %d ", front);
        printf("\n Items -> ");
        for (i = front; i != rear; i = (i + 1) % SIZE)
        {
            printf("%d  ", items[i]);
        }
        printf("%d ", items[i]);
        printf("\n Rear -> %d \n", rear);
    }
}

int main()
{
    int n;
    while (1)
    {
        printf("\nAll Item Here!!\n");
        printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");

        printf("\nchoice any item for above.\n");
        scanf("%d", &n);
        switch (n)
        {

        case 1:
            enQueue();
            break;
        case 2:
            deQueue();
            break;
        case 3:

            Display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nInvalid Choice.\n");
            break;
        }
    }
    return 0;
}
```

5. TEST RESULT

Output Circular Queue :

```
All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit

choice any item for above.
1

which value enqueue :
50

 Inserted -> 50
All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit

choice any item for above.
1

which value enqueue :
30

 Inserted -> 30
All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit

choice any item for above.
1

which value enqueue :
20

 Inserted -> 20
All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit

choice any item for above.
3

 Front -> 0
 Items -> 50  30  20
 Rear -> 2

All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit
```

```
choice any item for above.
2

 Deleted element -> 50

All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit

choice any item for above.
3

 Front -> 1
 Items -> 30  20
 Rear -> 2

All Item Here!!

1.Insert
2.Delete
3.Display
4.Exit
choice any item for above.
4
PS D:\Batch_213_Semester_03\Data_Structure& ALgorithm_lab\new_file\lab_report_4>
```

6. ANALYSIS AND DISCUSSION

In first problem we get the proper use of circular queue and how to use it.In these problem first of all use queue and push or pop element then use circular queue and finally solved this problem.

1.OBJECTION [1]

In this problem I will discuss Stack and how it use?

2.PROCEDURE /ANALYSIS/DESIGN/PSEUDOCODE [2]

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.

```
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of
       Stack) which has the same precedence as or higher precedence than
       operator.
    2. Add operator to Stack.
       [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of
       Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
       [End of If]
       [End of If]
7. END.
```

3.IMPLEMENTATION

Stack:

```
// Infix to Postfix Expression using stack implementation in C
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>

#define SIZE 100

char stack[SIZE];
int top = -1;



void push(char item)
{
   if(top >= SIZE-1)
   {
      printf("\nStack Overflow.");
   }
   else
```

```
    {
        top = top+1;
        stack[top] = item;
    }
}
char pop()
{
    char item ;

    if(top <0)
    {
        printf("stack under flow: invalid infix expression");
        getchar();

        exit(1);
    }
    else
    {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

int is_operator(char symbol)
{
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
    {
        return 1;
    }
    else
    {
    return 0;
    }
}

int precedence(char symbol)
{
    if(symbol == '^' )
    {
        return(3);
    }
    else if(symbol == '*' || symbol == '/')
```

```
   {
      return(2);
   }
   else if(symbol == '+' || symbol == '-')
   {
      return(1);
   }
   else
   {
      return(0);
   }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
   int i, j;
   char item;
   char x;

   push('(');
   strcat(infix_exp,")");

   i=0;
   j=0;
   item=infix_exp[i];

   while(item != '\0')
   {
      if(item == '(')
      {
         push(item);
      }
      else if( isdigit(item) || isalpha(item))
      {
         postfix_exp[j] = item;
         j++;
      }
      else if(is_operator(item) == 1)
      {
         x=pop();
         while(is_operator(x) == 1 && precedence(x)>= precedence(item))
         {
            postfix_exp[j] = x;
```

```
                j++;
                x = pop();
            }
            push(x);


            push(item);
        }
        else if(item == ')')
        {
            x = pop();
            while(x != '(')
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        i++;

        item = infix_exp[i];
  }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }
    if(top>0)
    {
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
    }

    postfix_exp[j] = '\0';
```

```
}


int main()
{
    char infix[SIZE], postfix[SIZE];
    printf("ASSUMPTION: The infix expression contains single letter variables and
single digit constants only.\n");
    printf("\nEnter Infix expression : ");
    gets(infix);

    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");
    puts(postfix);

    return 0;
}
```

4.TEST RESULT

Output infix to postfix expression:

```
 ASSUMPTION: The infix expression contains single letter variables
and single digit constants only.

Enter Infix expression: A+(B*C-(D/E^F)*G)*H
Postfix Expression: ABC*DEF^/G*-H*+
PS D:\Batch_213_Semester_03\Data_Structure&
```

5.ANALYSIS AND DISCUSSION

In this problem we will be solved infix to postfix expression using a stack. In this particular problem computer easy handle postfix notation so it is very important in daily basis life in computer.