DSA LAB PROGRAMS

By ASRITHA MEKA, AP19110010224 , CSE-G

Implementing Stacks using Arrays

34)

objective:

Write a C program to implement the STACK operation using array as a data structure. Users must be given the following choices to perform relevant tasks.

1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.

Explanation:

Push the elements: we need to Add an item in the given stack. If the stack is full, then we call it an Overflow condition.

Pop: Removes an item from the given stack. The items are popped. If the stack is empty, then we call it an Underflow condition.

Peek: Returns the top element of the given stack

Pseudo code:

>push the item

1. Read the item
2. if(top == n-1)
3. print(“overflow”)
4. else
5. top++
6. stack[top] = item
7. end

>Pop the item

1. if(top == -1)
2. print(“underflow”)
3. else{
4. top--
5. return stack[top+1] }
6. end

>Display it
1. while(top != -1){
2. print stack[top]
3. top-- }
4. End

CODE:

```
# include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define max 500
typedef struct STACK{
int ar[max];
int top;
}stack;
void push(stack *p, int data){
if(p->top >= max-1){
printf("overflow\n");
return;
}
p->top++;
p->ar[p->top] = data;
}
int pop(stack *p){
if(p->top < 0) {
printf("Underflow\n");
return INT_MIN;
}
int temp = p->ar[p->top];
p->top--;
return temp;
}
int peek(stack p){
return p.ar[p.top];
```

```
}
void display(stack p){
int i;
if(p.top == -1){
printf("Empty\n");
}
for(i =p.top;i>-1;i--){
printf("%d\n",p.ar[i]);
}
printf("\n");
}
int main(int argc, char const *argv[])
{
stack p;
p.top = -1;
int choice,data;
while(1){
printf("\n1. Push an element in to the STACK.\n"
"2. Pop the element from the STACK.\n"
"3. Peek the STACK.\n"
"4. Display the STACK.\n"
"5. Exit from the program.\n");
scanf("%d",&choice);
switch(choice){
case 1:{
printf("\nEnter an element for add\n");
scanf("%d",&data);
push(&p,data);
break;
}
case 2:{
data =pop(&p);
if(data != INT_MIN)
printf("%d is removed\n",data);
break;
}
case 3:{
printf("%d is the top\n",peek(p) );
break;
}
case 4:{
display(p);
break;
```

```
}
case 5:{
exit(0);
break;
}
default: printf("no such option is chosen again\n");
}
}
return 0;
}
```

Output:

```
1. Push an element into the STACK.
2. Pop the element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit from the program.
2
Underflow
1. Push an element into the STACK.
2. Pop the element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit from the program.
1
Enter an element for add
53
1. Push an element into the STACK.
2. Pop the element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit from the program.
4
53

1. Push an element into the STACK.
2. Pop the element from the STACK.
```

3. Peek the STACK.
4. Display the STACK.
5. Exit from the program.
5

Conclusion:
We got outputs when performing any actions in the correct way which are mentioned in the objective.

35)Title:

Reversing a string using Stacks

Objective:

Write a C program to reverse a string using STACK.

Explanation:

C Programming needs to be converted into “gn mmargorp C”. We can achieve this by using STACK data structure since it fools First In Last Out.

CODE:

```
#include <stdio.h>
#include <string.h>
#define max 500
int top,stack[max];
void push(char n){
        if(top == max-1){
        printf("stack overflow");
        }  else {
        stack[++top]=n;
        }

}

void pop(){
        printf("%c",stack[top--]);
}
```

```
main()
{
  char str[]="asritha meka";
  int len = strlen(str);
  int i;

  for(i=0;i<len;i++)
        push(str[i]);

  for(i=0;i<len;i++)
        pop();
}
```

Output:

akem ahtirsa

Conclusion:

The Exception when the string is containing ‘0’ so, it was given as the “underflow” condition in the pop() function. This can be overcomed by using a linked list.

36)

Title:

Conversion of In-Fix to Postfix

Objective:

Write a C program to convert the given infix expression to postfix expression using STACK.

Pseudo code:

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

3. If an operand is encountered, add it to B.
4. If a left parenthesis is encountered, push it onto Stack.
5. Repeatedly pop from Stack and add to the B operator.
6. Repeatedly pop from Stack and add to B each operator. Remove the left Parenthesis.
7. END.

Explanation:

For converting infix expression to postfix expression, we are going to use stack data structure to get code.

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 300
char stack[SIZE];
int n = -1;
void push(char item)
{
 if(n >= SIZE-1)
 {
  printf("\nStack Overflow.");
 }
 else
 {
  n = n+1;
  stack[n] = item;
 }
}
char pop()
{
 char item ;
 if(n <0)
 {
```

```
  printf("stack underflow: invalid infix expression");
  getchar();
  exit(1);
 }
 else
 {
  item = stack[n];
  n = n-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 a, b;
char item;
char y;
push('(');
strcat(infix_exp,")");
a=0;
b=0;
item=infix_exp[a];
while(item != '\0')
{
 if(item == '(')
 {
  push(item);
 }
 else if( isdigit(item) || isalpha(item))
 {
  postfix_exp[b] = item;
  b++;
 }
 else if(is_operator(item) == 1)
 {
  y=pop();
  while(is_operator(y) == 1 && precedence(y)>= precedence(item))
  {
   postfix_exp[b] = y;
   b++;
   y = pop();
  }
  push(y);
  push(item);
 }
 else if(item == ')')
 {
  y = pop();
  while(y != '(')
  {
   postfix_exp[b] = y;
   b++;
   y = pop();
  }
 }
 else
 {
```

```
   printf("\nInvalid infix Expression.\n");
   getchar();
   exit(1);
  }
  a++;

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

 postfix_exp[b] = '\0';
}

int main()
{
 char infix[SIZE], postfix[SIZE];

 printf("The infix expression is contains single letter variables.\n");
 printf("\nEnter the Infix expression : ");
 gets(infix);
 InfixToPostfix(infix,postfix);
 printf("Postfix Expression: ");
 puts(postfix);
 return 0;
}
```

OUTPUT:

The infix expression contains single letter variables.
Enter the Infix expression : 23
Postfix Expression: 23

CONCLUSION:

The code gives desired outputs as expected. The time complexity of the program is O(n^2) in best, average, worst cases. Not very reliable for large expressions. Space complexity is O(n).

37)

Title:

Conversion of In-Fix to Prefix

Objective:

Write a C program to convert the given infix expression to prefix expression using STACK.

Explanation:

We need to convert the infix to prefix by using the stack

Pseudo Code:

>Push

1. Read string, INITIALIZE stack, i = 0
2. FOR i < string.size()
3. push(string[i])/*pushes into the stack*/
4. WHILE (a = pop() != '0')
5. string[i] = a
6. PRINT the string
7. END

CODE:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define max 700
typedef struct Stack{
int n;
char arr[max];
```

```
}stack;
char pop(stack *s){
if (s->n!= -1)
return s->arr[s->n--] ;
return '#';
}
void push(stack *s, char op){
s->arr[++s->n] = op;
}
int Prec(char ch){
switch (ch) {
case '+': return 1;
case '-': return 1;
case '*': return 2;
case '/': return 2;
case '^': return 3;
}
return -1;
}
void display(stack s){
int i;
if(s.n == -1){
printf("Empty\n");
}
for(i =s.n;i>-1;i--){
printf("%c",s.arr[i]);
}
printf("\n");
}
void in2pre(char* exp) {
int i, n;
for(n = 0;exp[n];n++);
stack s; s.n = -1;
stack pre;pre.n = -1;
for (i = n-1; i>=0; i--) {
if ((exp[i] >= 'a' && exp[i] <= 'z') || (exp[i] >= 'A' && exp[i]
<= 'Z'))
push(&pre,exp[i]);
else if (exp[i] == ')')
push(&s, exp[i]);
else if (exp[i] == '('){
while (s.n!= -1 && s.arr[s.n] != ')')
push(&pre,pop(&s));
```

```
if (s.n == -1 && s.arr[s.n] != ')')
printf("Invalid expression\n");
else
pop(&s);
 }
else {
while (s.n!= -1 && Prec(exp[i]) <= Prec(s.arr[s.n]))
push(&pre,pop(&s));
push(&s, exp[i]);
}
}
push(&pre,pop(&s));
display(pre);
}
int main() {
char exp[] = "(a+b/c)*(a/k+l)";
printf( "Infix- expression: %s\n", exp );
printf("%s","prefix expression:" );
in2pre(exp);
return 0;
}
```

OUTPUT:

Infix- expression: (a+b/c)*(a/k+l)
prefix expression:*+a/bc+/akl

CONCLUSION:

The code has the expected outputs. Not every reliable for the large expressions but the space complexity is O(n)

38):

Title:

Evaluation of Postfix and Prefix expressions.

Objective:

Write a C program to evaluate the given prefix expression, post-fix expression.

Explanation:

When reading the expression from left to the right. We have to push the element in the given stack if it is an operand only. We had to pop 2 operands from the given stack. Push back the result of the evaluation. We had to repeat this upto end of the given expression.

Pseudo code:

```
1.{
2.    Read the next element
3.    If element is operand in the given stack
4.          Push the element
5.    If element id operator then
    {
6.        Pop the element
7.        Evaluate the expression formed by 2 operands
8.        Push the result
        }
  }
9. END
```

CODE:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define max 500
typedef struct Stack{
 int n;
 int arr[max];
}stack;
int pop(stack *s){
 if (s->n > -1)
```

```
  return s->arr[s->n--] ;
 return -1;
}
void push(stack *s, int op){
 s->arr[++s->n] = op;
}
void display(stack s){
 int i;
 if(s.n == -1){
  printf("Empty\n");
 }
 for(i =s.n;i>-1;i--){
  printf("%d\n",s.arr[i]);
 }
 printf("\n");
}
void evaluatePost(char *exp){
 int i, num = 0;
 stack operand; operand.n = -1;
 for(i= 0;exp[i];i ++){
  if(exp[i]-'0' >= 0 && exp[i]-'0' <=9){
   if(num == 0) num = exp[i]-'0';
   else num = num*10 + exp[i]-'0';
  }
  else if(exp[i] == ' '){
   push(&operand, num);
   num = 0;
  }
  else{
   int op2 = pop(&operand);
   int op1 = pop(&operand);
   if(op2 == -1 || op1 == -1){
    printf("invalid\n");
    return;
   }
   switch(exp[i]){

    case '*':{
     push(&operand, (op1*op2));
     break;
    }
    case '/':{
     push(&operand,(op1/op2));
```

```
     break;
    }
    case '+':{
     push(&operand,(op1+op2));
     break;
    }
    case '-':{
     push(&operand,(op1-op2));
     break;
    }

   }
  }
 }
 if(operand.n!= 0){ printf("invalid\n");
 }
 display(operand);
}
void evaluatePre(char *exp){
 int n;
 for(n=0;exp[n];n++);
 int i, num = 0;
 stack operand; operand.n = -1;
 for(i = n-1;i>-1;i--){
  if(exp[i]-'0' >= 0 && exp[i]-'0' <=9){
   if(num == 0) num = exp[i]-'0';
   else num = num*10 + exp[i]-'0';
  }
  else if(exp[i] == ' '){
   push(&operand, num);
   num = 0;
  }
  else{
   int op2 = pop(&operand);
   int op1 = pop(&operand);
   if(op2 == -1 || op1 == -1){
    printf("invalid\n");
    return;
   }
   switch(exp[i]){

    case '*':{
     push(&operand, (op1*op2));
```

```
     break;
    }
    case '/':{
     push(&operand,(op1/op2));
     break;
    }
    case '+':{
     push(&operand,(op1+op2));
     break;
    }
    case '-':{
     push(&operand,(op1-op2));
     break;
    }

   }
  }
 }
 if(operand.n!= 0){ printf("invalid\n");
 return;
 }
 display(operand);
}
int main(){
 char exp[] = "7 9 +3 1 +/";
 printf("expression = %s\n",exp);
 evaluatePost(exp);
 char exp2[] = "+ 9* 1 6";
 printf("expression = %s\n",exp2);
 evaluatePre(exp2);
 return 0;
}
```

OUTPUT:

```
expression = 7 9 +3 1 +/
4
expression = + 9* 1 6
15
```

CONCLUSION:

The input is restricted and complex in the pseudo code. The time complexity will be on both evaluation of prefix and postfix .

39)

Title: IMPLEMENTATION OF LINEAR QUEUE USING STACKS

OBJECTIVE:

Write a c program to i =mplement a linear queue, the user must choose the following options.

1. Add an element on queue-enqueue.
2. Remove an element from the queue- dequeue.
3. Display the element of the queue.
4. Terminate the program.

EXPLANATION:

Enqueue will add the item to the given queue. While the queue is full then we can say it is overflow condition.
Dequeue will remove the item from the given queue. The items are popped in the same order while they are pushing. If the queue is empty then we can say that it is underflow condition.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
# define MAX_SIZE 300
void display(int *queue, int rear, int front){
        int i;
        printf("now queue (front.....to .....back:)\n");
        for(i=front;i<rear;i++){
        printf("%d ",*(queue+i));
        }
```

```
        printf("%d ",*(queue+rear));
}
void enqueue(int *queue, int *rear,int *front){
        if(*rear  == MAX_SIZE-1){
        printf("Overflow\n Aborting...");
        return;
        }
        int ele;
        printf("Enter the element to insert");
        scanf("%d",&ele);
        if (*rear == -1)
        {
        *front = 0;
        }
        *rear += 1;
        *(queue+*rear) = ele;

}
void dequeue(int *queue, int *front,int *rear){
        if((*front)==-1){
        printf("\nunderflow.. aborting");
        return;
        }

        int temp = *(queue+(*front));
        if(*front== *rear){*front= -1; *rear= -1;}
        else{*front = *front + 1;}
        printf("\n%d is deleted",temp);
}
int main(){
        int ans, queue[MAX_SIZE], front = -1, rear = -1;
        while(1){
        printf("\nMENU\n"
        "\n1.Insert element "
        "\n2. delete element "
        "\n3.Display queue"
        "\n4.Then Exit");
        scanf("%d",&ans);
        switch(ans){
        case 1: enqueue(queue, &rear, &front);break;
        case 2: dequeue(queue,&front,&rear);break;
        case 3: display(queue, rear, front);break;
        case 4: exit(0);break;
```

```
        }
        }
        return 0;
}
```

OUTPUT:

MENU
1.Insert element
2. delete element
3.Display queue
4. Then Exit^C

CONCLUSION:

Will got expected output. For deletion and insertion time complexity is constant.

40)

Title:

IMPLEMENTATION OF CIRCULAR QUEUE

OBJECTIVE:

Write a c program to implement a circular queue

1. Add an element on queue-enqueue.
2. Remove an element from the queue- dequeue.
3. Display the element of the queue.
4. Terminate the program.

EXPLANATION:

A circular queue is a linear data structure. The last position will be connected as a back to the first position and it is formed as a circle.

CODE:

```
 #include <stdio.h>
#include <stdlib.h>
# define MAX_SIZE 8
void enqueue(int *queue, int *rear,int *front){
        if((*rear  == MAX_SIZE-1 && *front == 0)|| (*front== (*rear)+1)){
        printf("Overflow\n Aborting...");
        return;
        }
        int ele;
        printf("Enter the element to insert");
        if (*front == -1){
        *front = 0;
        }
        *rear = ((*rear)+1)%MAX_SIZE;
        printf("%d is rear\n",*rear );
        scanf("%d",&ele);
        *(queue+*rear) = ele;

}
void display(int queue[], int rear, int front){
        int i;
        printf("now queue (front.....to .....back:)\n");
        for(i=(front);i!=rear;i= (i+1)%MAX_SIZE){
        printf("%d ",queue[i]);
        }
        printf("%d ",queue[i]);
}
void dequeue(int *queue, int *front,int *rear){
        if((*front)==-1){
        printf("\nunderflow.. aborting");
        return;
        }

        int temp = *(queue+(*front));
        if(*front== *rear){*front= -1; *rear= -1;}
        else{*front = (*front + 1)%MAX_SIZE;}
        printf("\n%d is deleted",temp);
}
int main(){
        int ans, queue[MAX_SIZE], front = -1, rear = -1;
```

```
        //menu
        while(1){
        printf("\nMENU:circular queue\n"
        "\n1.Insert the element "
        "\n2.delete the element "
        "\n3.Display the queue"
        "\n4. Then Exit");
        scanf("%d",&ans);
        switch(ans){
        case 1: enqueue(queue, &rear, &front);break;
        case 2: dequeue(queue, &front,&rear);break;
        case 3: display(queue, rear, front);break;
        case 4: exit(0);break;
        }
        }
        return 0;
}
```

OUTPUT:

```
MENU:circular the queue
1.Insert the element
2.delete the element
3.Display the queue
4.then Exit
Enter the element to insert0 is rear
3
MENU:circular queue
1.Insert an element
2.delete an element
3.Display queue
4.Exit23
MENU:circular queue
1.Insert an element
2.delete an element
3.Display queue
4.Exit23
MENU:circular queue
1.Insert an element
2.delete an element
3.Display queue
```

4.Exitm 6
MENU:circular queue
1.Insert an element
2.delete an element
3.Display queue
4.Exit

CONCLUSION:

This is also the same as linear queue.it is constant for the enqueue and dequeue O(n).