Rajalakshmi Engineering College

Rajalakshmi Nagar, Thandalam, Chennai - 602 105

Department of Computer Science and Engineering

CS19241 - Data Structures

Unit - II

Lecture Notes

(Regulations - 2019)

Prepared by:

B.BHUVANESWARAN

Assistant Professor (SG) / CSE / REC

bhuvaneswaran@rajalakshmi.edu.in

12.1 INTRODUCTION

A stack is a list with the restriction that insertions and deletions can be performed in only one position, namely, the end of the list, called the top. It follows Last-In-First-Out (LIFO) principle.

12.2 OPERATIONS ON STACK

The fundamental operations on a stack are:

- Push - which is equivalent to insert
- Pop - which deletes the most recently inserted element
- Peek - return top of stack
- MakeEmpty – create an empty stack
- IsEmpty – check whether a stack is empty
- IsFull – check whether a stack is full

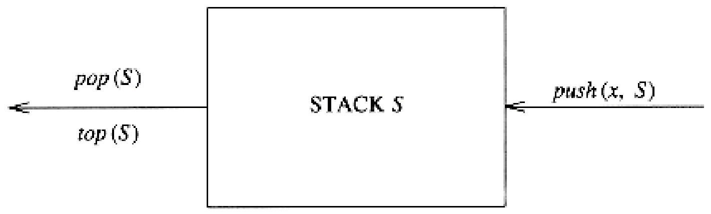


Fig. 12.1 Stack model: input to a stack is by push, output is by pop

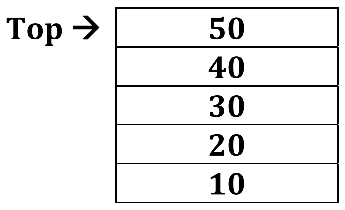


Fig. 12.2 Stack model: only the top element is accessible

12.3 PUSH

- The process of inserting a new element to the top of the stack is called push operation.
- For every push operation the top is incremented by 1.

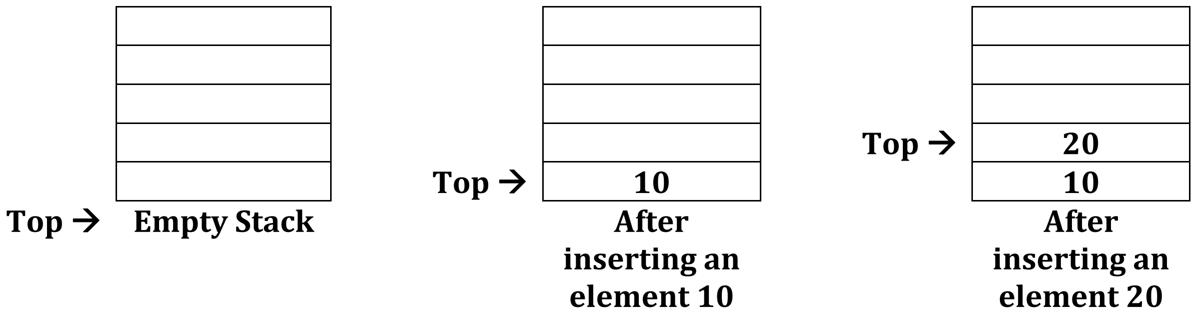


Fig. 12.3 Push Operations

12.4 POP

- The process of deleting an element from the top of stack is called pop operation.
- After every pop operation the top pointer is decremented by 1.

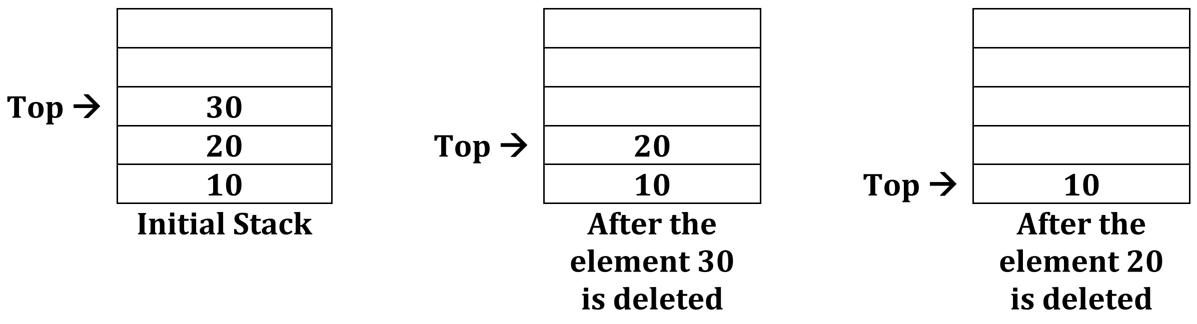

Top →
30
20
10
Initial Stack
Top →
20
10
After the element 30 is deleted
Top →
10
After the element 20 is deleted

Fig. 3.4 Pop Operations

12.5 EXCEPTIONAL CONDITIONS

12.5.1 Overflow

Attempt to insert an element when the stack is full is said to be overflow.

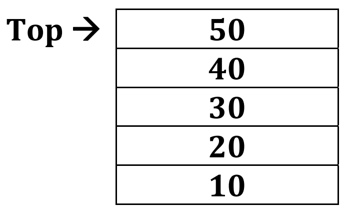


Fig. 12.5 Full Stack (Size = 5)

12.5.2 Underflow

Attempt to delete an element when the stack is empty is said to be underflow.

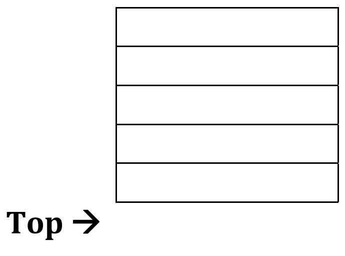


Fig. 12.6 Empty Stack

12.6 IMPLEMENTATION OF STACKS

Stack can be implemented using:

- Array
- Linked list

REVIEW QUESTIONS

1. Define stack. What are the operations performed on a stack?
2. List out the operations on stack.
3. What are the different ways to implement stack?
4. What are the exception conditions of a stack?
5. What is a top pointer? Explain its significance.

13.1 INTRODUCTION

In this implementation each stack is associated with a top pointer, which is -1 for an empty stack.

13.1.1 Push

To push an element X onto the stack, top pointer is incremented by 1 and then set:

Stack [top] = X.

13.1.2 Pop

To pop an element from the stack, the Stack [top] value is returned and the top pointer is decremented by 1.

13.2 STACK FULL

13.2.1 Algorithm to Check whether a Stack is Full

```
IsFull()

Step 1  : Start.
Step 2  : If top = MAX - 1 goto Step 3 else goto Step 4.
Step 3  : Return 1 and Stop.
Step 4  : Return 0.
Step 5  : Stop.
```

13.2.2 Routine to Check whether a Stack is Full

```
int IsFull()
{
	if(top == MAX - 1)
		return 1;
	else
		return 0;
}
```

13.3 STACK EMPTY

13.3.1 Algorithm to Check whether a Stack is Empty

```
IsEmpty()

Step 1  : Start.
Step 2  : If top = -1 goto Step 3 else goto Step 4.
Step 3  : Return 1 and Stop.
Step 4  : Return 0.
Step 5  : Stop.
```

13.3.2 Routine to Check whether a Stack is Empty

```
int IsEmpty()
{
	if(top == -1)
		return 1;
	else
		return 0;
}
```

13.4 PUSH

13.4.1 Algorithm to Push an Element on to the Stack

```
Push(ele)

Step 1 : Start.
Step 2 : If IsFull() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Stack Overflow…!” and goto Step 6.
Step 4 : Set Top = Top + 1.
Step 5 : Set Stack[Top] = ele.
Step 6 : Stop.
```

13.4.2 Routine to Push an Element on to the Stack

```
void Push(int ele)
{
	if(IsFull())
		printf("Stack Overflow...!\n");
	else
	{
		top = top + 1;
		Stack[top] = ele;
	}
}
```

13.5 POP

13.5.1 Algorithm to Pop an Element from the Stack

```
Pop()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Stack Underflow…!” and goto Step 6.
Step 4 : Display Stack[Top].
Step 5 : Set Top = Top – 1.
Step 6 : Stop.
```

13.5.2 Routine to Pop an Element from the Stack

```c
void Pop()
{
	if(IsEmpty())
		printf("Stack Underflow...!\n");
	else
	{
		printf("%d\n", Stack[top]);
		top = top - 1;
	}
}
```

13.6 PEEK

13.6.1 Algorithm to Return Top of Stack

```
Top()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Stack Underflow…!” and goto Step 5.
Step 4 : Display Stack[Top].
Step 5 : Stop.
```

13.6.2 Routine to Return Top of Stack

```c
void Top()
{
	if(IsEmpty())
		printf("Stack Underflow...!\n");
	else
		printf("%d\n", Stack[top]);
}
```

13.7 DISPLAY

13.7.1 Algorithm to Display Stack Elements

```
Display()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Stack Underflow…!” and goto Step 8.
Step 4 : Set i = top.
Step 5 : Repeat Steps 6 to 7 while i >= 0.
Step 6 : Display Stack[i].
Step 7 : Set i = i – 1.
Step 8 : Stop.
```

13.7.2 Routine to Display Stack Elements

```
      void Display()
      {
            int i;
            if(IsEmpty())
                  printf("Stack Underflow...!\n");
            else
            {
                  for(i = top; i >= 0; i--)
                        printf("%d\t", Stack[i]);
                  printf("\n");
            }
      }
```

13.8 PROGRAM

```
#include <stdio.h>

#define MAX 5

int Stack[MAX], top = -1;

int IsFull();
int IsEmpty();
void Push(int ele);
void Pop();
void Top();
void Display();

int main()
{
      int ch, e;
      do
      {
            printf("1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT");
            printf("\nEnter your choice : ");
            scanf("%d", &ch);
            switch(ch)
            {
                  case 1:
                        printf("Enter the element : ");
                        scanf("%d", &e);
                        Push(e);
                        break;
                  case 2:
                        Pop();
                        break;
                  case 3:
                        Top();
                        break;
```

```
                case 4:
                     Display();
                     break;
           }
      } while(ch <= 4);
      return 0;
}

int IsFull()
{
      if(top == MAX - 1)
           return 1;
      else
           return 0;
}

int IsEmpty()
{
      if(top == -1)
           return 1;
      else
           return 0;
}

void Push(int ele)
{
      if(IsFull())
           printf("Stack Overflow...!\n");
      else
      {
           top = top + 1;
           Stack[top] = ele;
      }
}

void Pop()
{
      if(IsEmpty())
           printf("Stack Underflow...!\n");
      else
      {
           printf("%d\n", Stack[top]);
           top = top - 1;
      }
}

void Top()
{
      if(IsEmpty())
           printf("Stack Underflow...!\n");
      else
```

```
            printf("%d\n", Stack[top]);
}

void Display()
{
      int i;
      if(IsEmpty())
            printf("Stack Underflow...!\n");
      else
      {
            for(i = top; i >= 0; i--)
                  printf("%d\t", Stack[i]);
            printf("\n");
      }
}
```

Output

```
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 20
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 30
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 40
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 60
Stack Overflow...!
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 4
50      40      30      20      10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 3
50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
40
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
```

```
30
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
20
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
Stack Underflow...!
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 5
```

REVIEW QUESTIONS

1. How do you push and pop elements in an array stack?
2. Explain push and pop operations with the help of examples.

CHAPTER - 14 - LINKED LIST IMPLEMENTATION OF STACK

14.1 INTRODUCTION

In this implementation:

- Push operation is performed by inserting an element at the front of the list.
- Pop operation is performed by deleting at the front of the list.
- Top operation returns the element at the front of the list.

14.2 TYPE DECLARATIONS FOR LINKED LIST IMPLEMENTATION OF THE STACK

```
struct node
{
	int Element;
	struct node *Next;
}*List = NULL;

typedef struct node Stack;
```

14.3 EMPTY LIST

14.3.1 Algorithm to Check whether a Stack is Empty

```
IsEmpty()

Step 1 : Start.
Step 2 : If List = NULL goto Step 3 else goto Step 4.
Step 3 : Return 1 and Stop.
Step 4 : Return 0 and Stop.
Step 5 : Stop.
```

14.3.2 Routine to Check whether a Stack is Empty

```
int IsEmpty()
{
	if(List == NULL)
		return 1;
	else
		return 0;
}
```

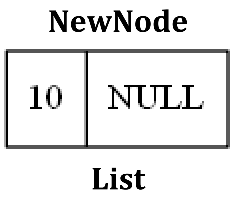
14.4 PUSH

The push is implemented as an insertion into the front of a linked list, where the front serves as the top of the stack.

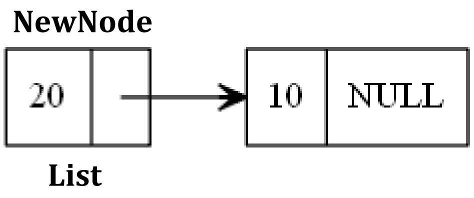
Example

The general idea is shown in Figure. The dashed line represents the old pointer.

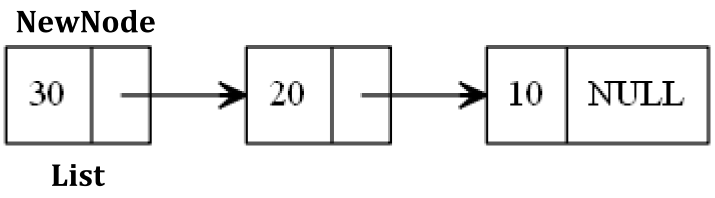
Push(10)



Push(20)



Push(30)



14.4.1 Algorithm to Push an Element on to the Stack

```
Push(e)

Step 1 : Start.
Step 2 : Set NewNode = addressof(Stack).
Step 3 : Set NewNode→Element = e.
Step 4 : If IsEmpty() = True, then goto Step 5 else goto Step 6.
Step 5 : Set NewNode→Next = NULL and goto Step 7.
Step 6 : Set NewNode→Next = List.
Step 7 : Set List = NewNode.
Step 8 : Stop.
```

14.4.2 Routine to Push an Element on to the Stack

```
void Push(int e)
{
      Stack *NewNode = malloc(sizeof(Stack));
      NewNode->Element = e;
      if(IsEmpty())
            NewNode->Next = NULL;
      else
            NewNode->Next = List;
      List = NewNode;
}
```

14.5 POP

Pop as a deletion from the front of the list.

Example

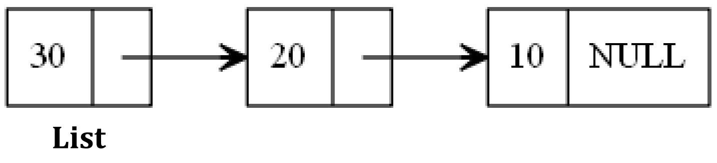
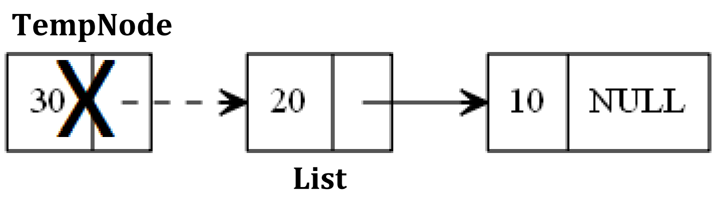


Fig. Initial Stack

Pop()



14.5.1 Algorithm to Pop an Element from the Stack

```
Pop()

Step 1 : Start.
Step 2 : If IsEmpty() = True, then goto Step 3 else goto Step 4.
Step 3 : Display "Stack is Underflow...!" and goto Step 8.
Step 4 : Set TempNode = List.
Step 5 : Set List = List→Next.
Step 6 : Display the TempNode→Element.
Step 7 : Delete TempNode.
Step 8 : Stop.
```

14.5.2 Routine to Pop an Element from the Stack

```c
void Pop()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
	{
		Stack *TempNode;
		TempNode = List;
		List = List->Next;
		printf("%d\n", TempNode->Element);
		free(TempNode);
	}
}
```

14.6 TOP

Top is performed by examining the element in the first position of the list.

Example

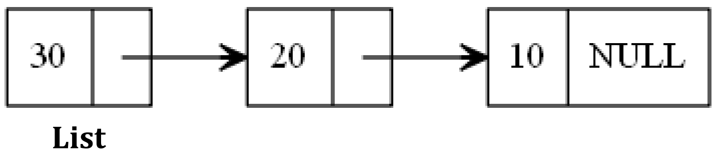
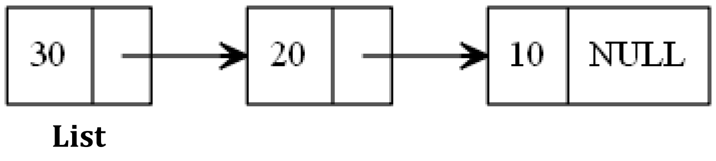


Fig. Initial Stack

Top()



14.6.1 Algorithm to Return Top of Stack

Top(List)

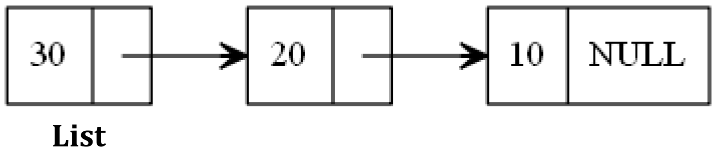
```
Step 1 : Start.
Step 2 : If IsEmpty() = True, then goto Step 3 else goto Step 4.
Step 3 : Display "Stack is Underflow...!" and goto Step 5.
Step 4 : Display the List→ Element.
Step 5 : Stop.
```

14.6.2 Routine to Return Top of Stack

```c
void Top()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
		printf("%d\n", List->Element);
}
```

14.7 DISPLAY

Example



14.7.1 Algorithm to Display Stack Elements

```
Display()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display “Stack is Underflow…!” and goto Step 8.
Step 4 : Set Position = List.
Step 5 : Repeat the Steps 6-7 until Position != NULL.
Step 6 : Display Position→Element.
Step 7 : Set Position = Position→Next.
Step 8 : Stop.
```

14.7.2 Routine to Display Stack Elements

```
void Display()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
	{
		Stack *Position;
		Position = List;
		while(Position != NULL)
		{
			printf("%d\t", Position->Element);
			Position = Position->Next;
		}
		printf("\n");
	}
}
```

14.8 PROGRAM

```
/* Implementation of stack using linked list - STACKLL.C */

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

struct node
{
	int Element;
	struct node *Next;
}*List = NULL;

typedef struct node Stack;

int IsEmpty();
void Push(int e);
void Pop();
void Top();
void Display();
int main()
{
      int ch, e;
      do
      {
            printf("1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT");
            printf("\nEnter your choice : ");
            scanf("%d", &ch);
            switch(ch)
            {
                  case 1:
                        printf("Enter the element : ");
                        scanf("%d", &e);
                        Push(e);
                        break;
                  case 2:
                        Pop();
                        break;
                  case 3:
                        Top();
                        break;
                  case 4:
                        Display();
                        break;
            }
      } while(ch <= 4);
      return 0;
}

int IsEmpty()
{
      if(List == NULL)
            return 1;
      else
            return 0;
}

void Push(int e)
{
      Stack *NewNode = malloc(sizeof(Stack));
      NewNode->Element = e;
      if(IsEmpty())
            NewNode->Next = NULL;
      else
            NewNode->Next = List;
      List = NewNode;
}
```

```
void Pop()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
	{
		Stack *TempNode;
		TempNode = List;
		List = List->Next;
		printf("%d\n", TempNode->Element);
		free(TempNode);
	}
}

void Top()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
		printf("%d\n", List->Element);
}

void Display()
{
	if(IsEmpty())
		printf("Stack is Underflow...!\n");
	else
	{
		Stack *Position;
		Position = List;
		while(Position != NULL)
		{
			printf("%d\t", Position->Element);
			Position = Position->Next;
		}
		printf("\n");
	}
}
```

Output

```
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 20
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 30
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
```

```
Enter your choice : 1
Enter the element : 40
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 1
Enter the element : 50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 4
50      40      30      20      10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 3
50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
50
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
40
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
30
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
20
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
10
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 2
Stack is Underflow...!
1.PUSH 2.POP 3.TOP 4.DISPLAY 5.EXIT
Enter your choice : 5
```

REVIEW QUESTIONS

1. How do you push and pop elements in a linked stack?

CHAPTER - 15 - APPLICATIONS OF STACK

15.1 APPLICATIONS OF STACK

Some of the applications of stack are:

- Balancing symbols
- Infix to postfix conversion
- Evaluating postfix expression
- Function calls
- Towers of Hanoi
- 8 queens problem
- Page-visited history in a Web browser (Back Buttons)
- Undo sequence in a text editor
- Matching Tags in HTML and XML

REVIEW QUESTIONS

1. State the applications of stack data structure. (or) List few applications of stack. (or) Mention the Applications of stack. (or) Write any four applications of stack. (or) List the applications of stack. (or) Give the applications of stack.

16.1 INTRODUCTION

Compilers check your programs for syntax errors, but frequently a lack of one symbol (such as a missing brace or comment starter) will cause the compiler to spill out a hundred lines of diagnostics without identifying the real error.

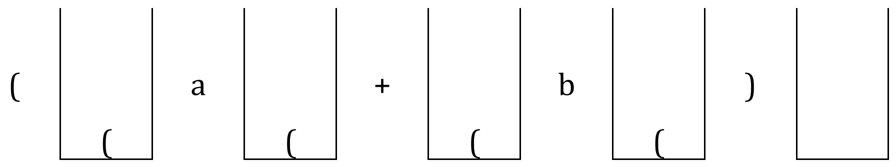
A useful tool in this situation is a program that checks whether everything is balanced. Thus, every right brace, bracket, and parenthesis must correspond to their left counterparts. The sequence [()] is legal, but [(]) is wrong. Obviously, it is not worthwhile writing a huge program for this, but it turns out that it is easy to check these things. For simplicity, we will just check for balancing of parentheses, brackets, and braces and ignore any other character that appears.

The simple algorithm uses a stack and is as follows:

- Make an empty stack.
- Read characters until end of input.
- If the character read is not a symbol to be balanced, ignore it.
- If the character is an opening symbol like (, [, {, push it onto the stack.
- If it is a closing symbol like),], }, then if the stack is empty report an error as "Missing opening symbol". Otherwise, pop the stack.
- If the symbol popped is not the corresponding opening symbol, then report an error as "Mismatched symbol".
- At end of input, if the stack is not empty report an error as "Missing closing symbol". Otherwise, report as "Symbols are balanced".

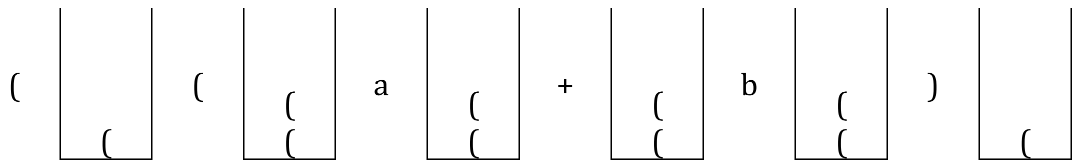
16.2 EXAMPLES

16.2.1 (a+b)



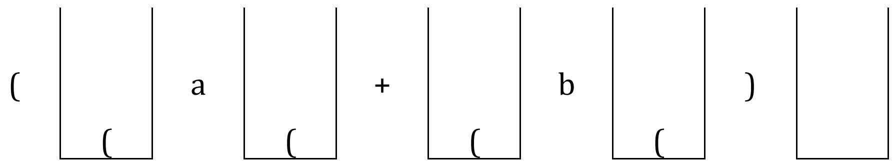
Stack is empty → Symbols are balanced.

16.2.2 ((a+b)



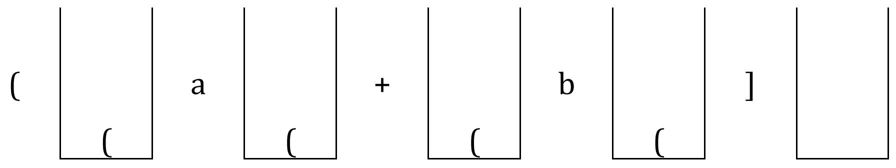
Stack is not empty → Missing closing symbol.

16.2.3 (a+b))



Stack is empty → Missing opening symbol.

16.2.4 (a+b]



Not corresponding to the opening symbol → Mismatched symbol.

16.3 PROGRAM

```
#include <stdio.h>
#include <string.h>

#define MAX 20
char Stack[MAX];
int top = -1;

int IsEmpty();
void Push(char sym);
char Pop();

int main()
{
	char exp[20], c;
	int i;
	printf("Enter the expression : ");
	scanf("%s", exp);
	for(i = 0; i < strlen(exp); i++)
	{
		if(exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
		{
			Push(exp[i]);
		}
		else if((exp[i]==')'||exp[i]==']'||exp[i]=='}')&&IsEmpty())
		{
			printf("Missing opening symbol...!");
			return 0;
		}
		else if(exp[i] == ')' || exp[i] == ']' || exp[i] == '}')
		{
			c = Pop();
```

```
                    if(c == '(' && exp[i] != ')')
                    {
                          printf("Mismatched symbol...!");
                          return 0;
                    }
                    else if(c == '[' && exp[i] != ']')
                    {
                          printf("Mismatched symbol...!");
                          return 0;
                    }
                    else if(c == '{' && exp[i] != '}')
                    {
                          printf("Mismatched symbol...!");
                          return 0;
                    }
              }
        }
        if(!IsEmpty())
              printf("Missing closing symbol...!");
        else
              printf("Symbols are balanced...!");
        return 0;
}

int IsEmpty()
{
        if(top == -1)
              return 1;
        else
              return 0;
}
void Push(char sym)
{
        Stack[++top] = sym;
}
char Pop()
{
        return Stack[top--];
}
```

Output

```
Enter the expression : (a+b)
Symbols are balanced...!
Enter the expression : ((a+b)
Missing closing symbol...!
Enter the expression : (a+b))
Missing opening symbol...!
Enter the expression : (a+b]
Mismatched symbol...!
```

REVIEW QUESTIONS

1. What is the role of stack in balancing symbol?
2. Write the algorithm for balancing symbol.

CHAPTER - 17 - NOTATIONS TO REPRESENT ARITHMETIC EXPRESSIONS

17.1 INTRODUCTION

There are 3 different ways of representing the algebraic expression. They are:

- Infix notation
- Postfix notation
- Prefix notation

17.2 INFIX NOTATION

In infix notation, the arithmetic operator appears between the two operands to which it is being applied.

For example:

A/B+C

17.3 POSTFIX NOTATION

In postfix notation, the arithmetic operator appears directly after the two operands to which it applies. It is also called as reverse polish notation.

For example:

((A/B) + C) → AB/C+

17.4 PREFIX NOTATION

In prefix notation, the arithmetic operator is placed before the two operands to which it applies. It is also called as polish notation.

For example:

((A/B) + C) → +/ABC

REVIEW QUESTIONS

1. What are the different types of notations to represent arithmetic expressions?
2. What is an infix notation?
3. What is a postfix notation?
4. What is a prefix notation?

CHAPTER - 18 - INFIX TO POSTFIX CONVERSION

18.1 INTRODUCTION

To evaluate an arithmetic expression, first convert the given infix expression to postfix expression and then evaluate the postfix expression using stack.

18.2 INFIX TO POSTFIX CONVERSION

Not only can a stack be used to evaluate a postfix expression, but we can also use a stack to convert an expression in standard form (otherwise known as infix) into postfix.

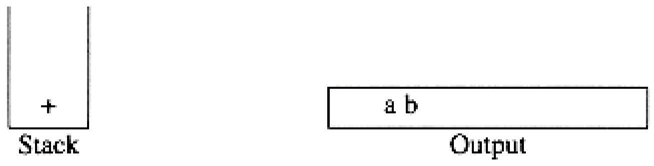
The algorithm uses a stack and is as follows:

- Make an empty stack.
- Read the infix expression one character at a time until it encounters end of expression.
- If the character is an operand, place it onto the output.
- If the character is an operator, push it onto the stack. If the stack operator has a higher or equal priority than input operator, then pop that operator from the stack and place it onto the output.
- If the character is a left parenthesis, push it onto the stack.
- If the character is a right parenthesis, pop all the operators from the stack till it encounters left parenthesis, discard both the parenthesis in the output.

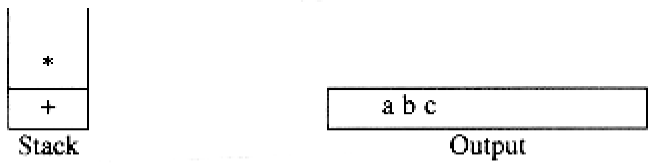
18.3 EXAMPLE-I

a + b * c + (d * e + f) * g

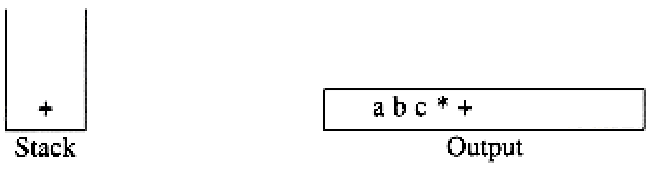
First, the symbol a is read, so it is passed through to the output. Then '+' is read and pushed onto the stack. Next b is read and passed through to the output. The state of affairs at this juncture is as follows:



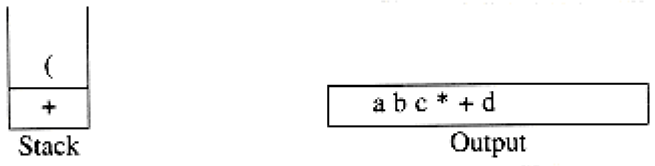
Next a '*' is read. The top entry on the operator stack has lower precedence than '*', so nothing is output and '*' is put on the stack. Next, c is read and output. Thus far, we have



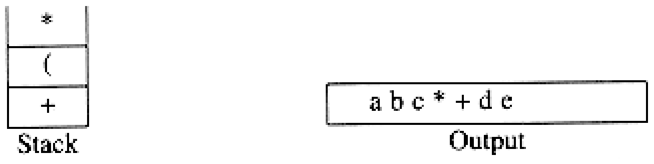
The next symbol is a '+'. Checking the stack, we find that we will pop a '*' and place it on the output, pop the other '+', which is not of lower but equal priority, on the stack, and then push the '+'.



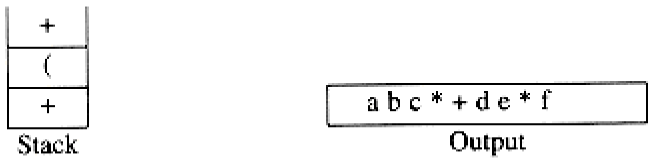
The next symbol read is an '(', which, being of highest precedence, is placed on the stack. Then d is read and output.



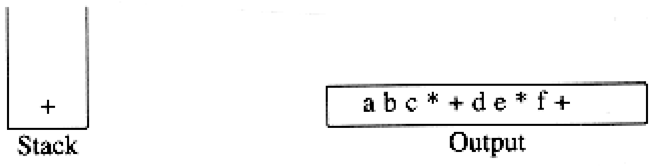
We continue by reading a '*'. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, e is read and output.



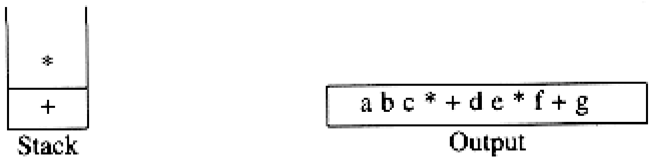
The next symbol read is a '+'. We pop and output '*' and then push '+'. Then we read and output.



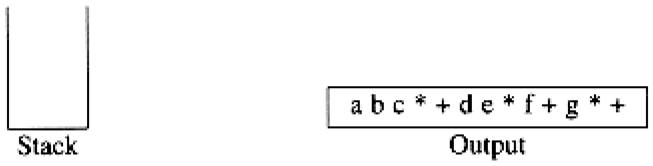
Now we read a ')', so the stack is emptied back to the '('. We output a '+'.



We read a '*' next; it is pushed onto the stack. Then g is read and output.

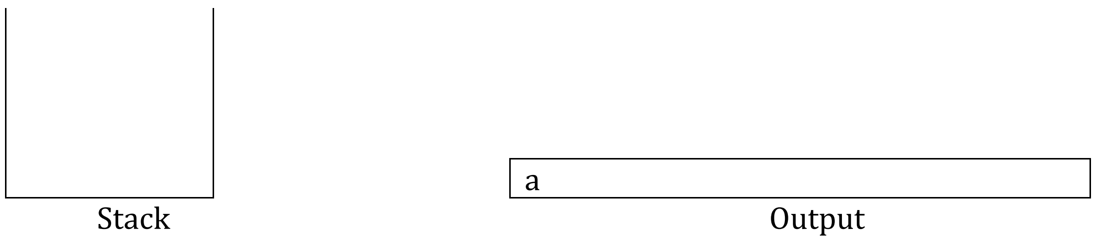


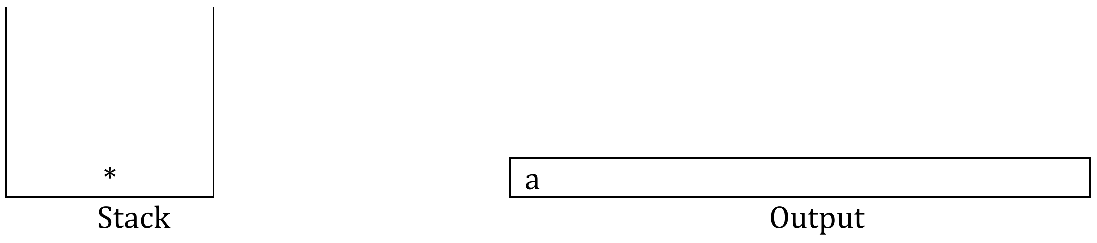
The input is now empty, so we pop and output symbols from the stack until it is empty.

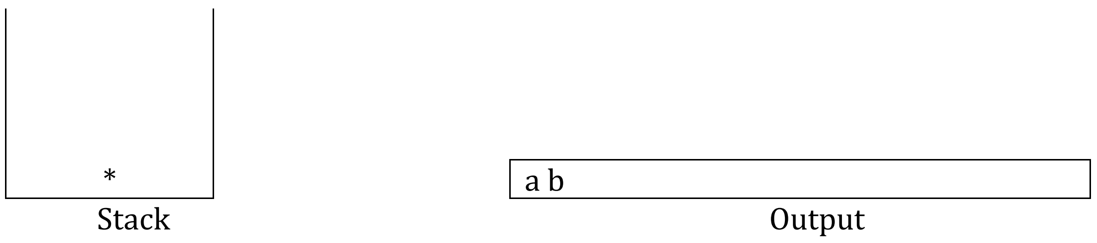


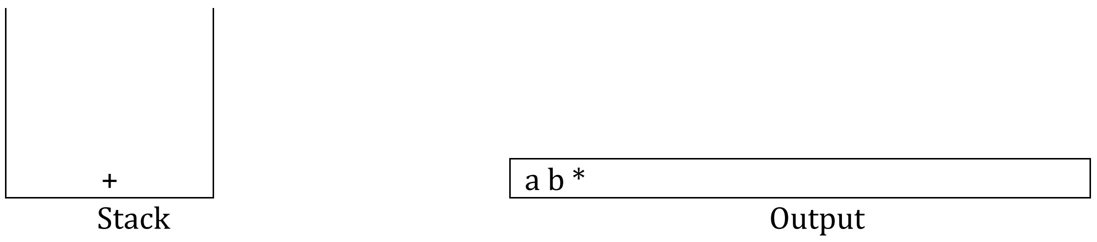
18.4 EXAMPLE-II

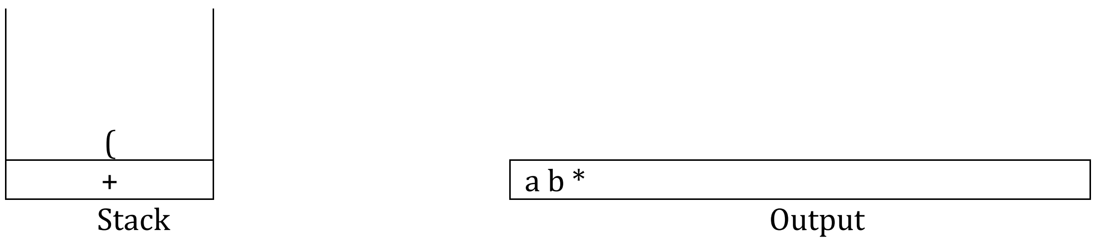
a * b + (c – d / e)

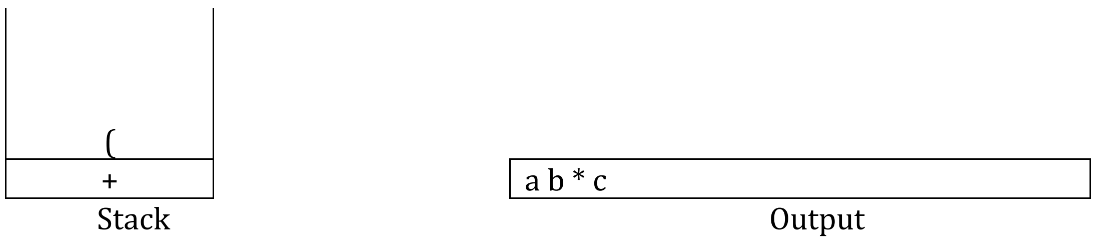


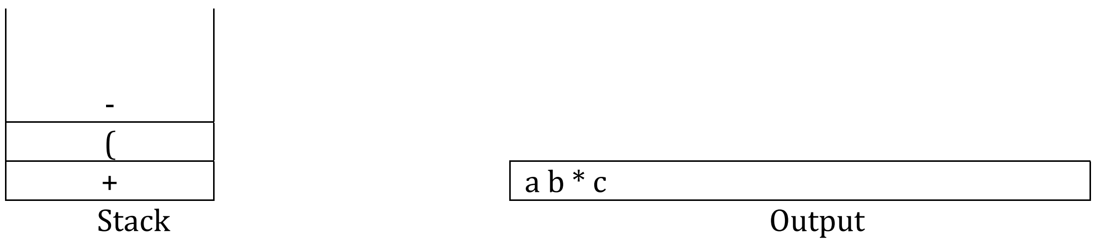


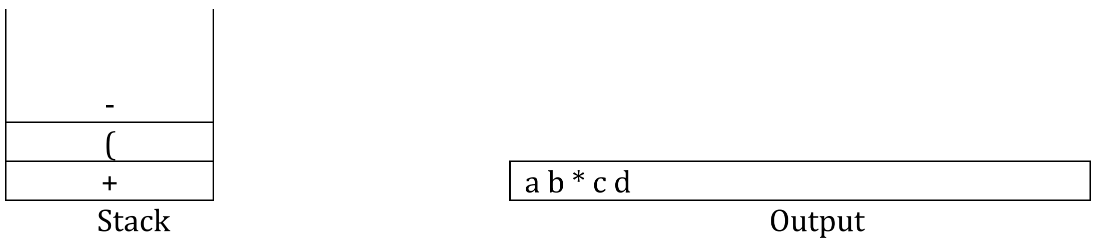


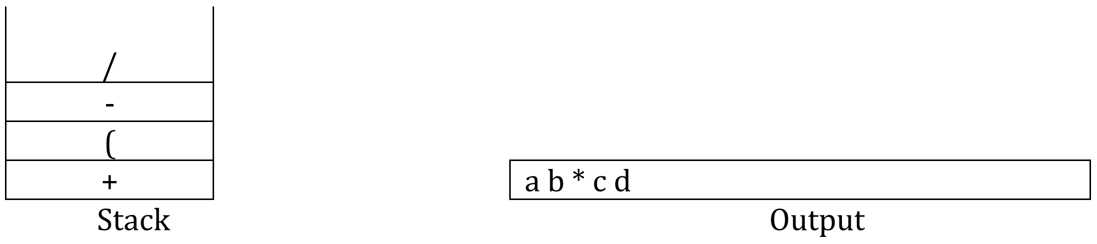


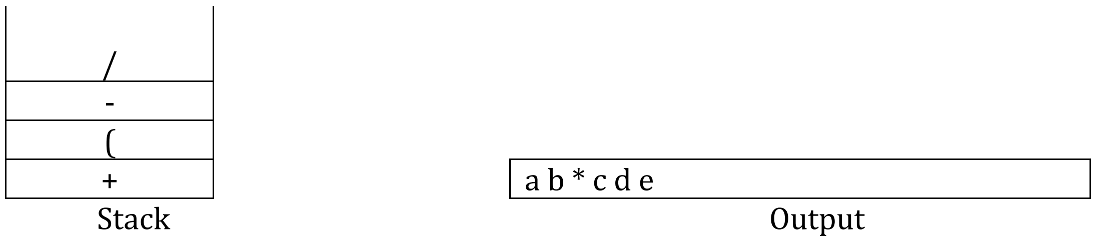


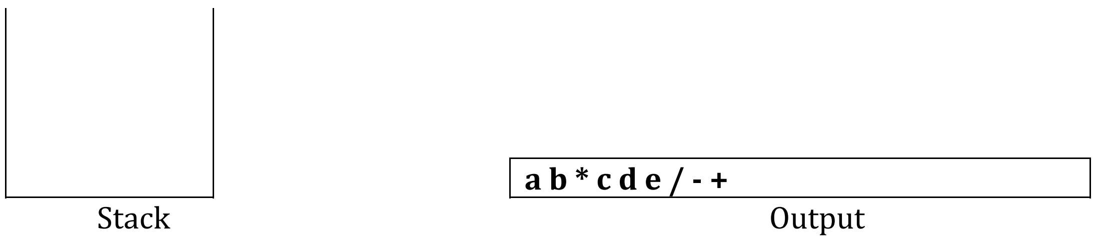












18.5 EXAMPLE-III

Figure 18.1 shows the conversion algorithm working on the expression A * B + C * D. Note that the first * operator is removed upon seeing the + operator. Also, + stays on the stack when the second * occurs, since multiplication has precedence over addition. At the end of the infix expression the stack is popped twice, removing both operators and placing + as the last operator in the postfix expression.

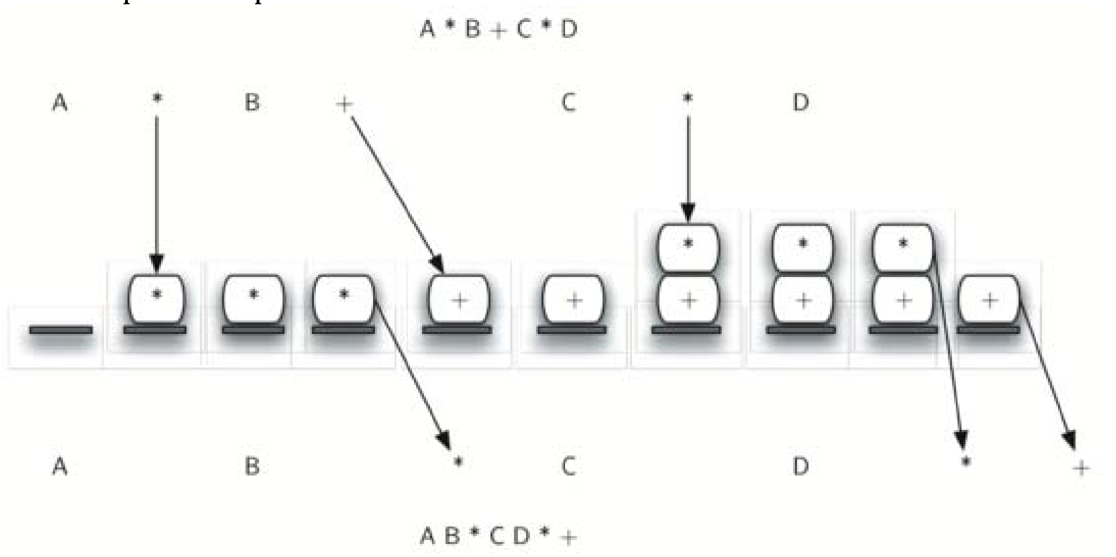


Fig. 18.1 Converting A*B + C *D to Postfix Notation

18.5 ADDITIONAL EXAMPLES

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +
(A + B) * C	* + A B C	A B + C *
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

18.6 PROGRAM (USING ARRAY)

```
#include <stdio.h>
#include <string.h>

#define MAX 20

int Stack[MAX], top = -1;
char expr[MAX], post[MAX];

void Push(char sym);
char Pop();
char Top();
int Priority(char sym);

int main()
{
	int i;
	printf("Enter the infix expression : ");
	gets(expr);
	for(i = 0; i < strlen(expr); i++)
	{
		if(expr[i] >= 'a' && expr[i] <= 'z')
			printf("%c", expr[i]);
		else if(expr[i] == '(')
			Push(expr[i]);
		else if(expr[i] == ')')
		{
			while(Top() != '(')
				printf("%c", Pop());
			Pop();
		}
		else
		{
			while(Priority(expr[i])<=Priority(Top()) && top!=-1)
				printf("%c", Pop());
			Push(expr[i]);
		}
	}
	for(i = top; i >= 0; i--)
		printf("%c", Pop());
	return 0;
}

void Push(char sym)
{
	top = top + 1;
	Stack[top] = sym;
}

char Pop()
{
	char e;
	e = Stack[top];
	top = top - 1;
	return e;
}
char Top()
{
	return Stack[top];;
}

int Priority(char sym)
{
	int p = 0;
	switch(sym)
	{
		case '(':
			p = 0;
			break;
		case '+':
		case '-':
			p = 1;
			break;
		case '*':
		case '/':
		case '%':
			p = 2;
			break;
		case '^':
			p = 3;
			break;
	}
	return p;
}
```

Output

```
Enter the infix expression : a/b^c+d*e-f*g
abc^/de*+fg*-
```

REVIEW QUESTIONS

1. How will you evaluate an arithmetic expression?
2. Write the algorithm for infix to postfix conversion.
3. Convert the infix expression a + b * c + (d * e + f)* g to its equivalent postfix expression.
4. 19. What are the postfix and prefix forms of the expression?
   A + B * (C – D) / (P – R)
5. Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.
6. Given the infix for an expression, write its prefix a*b/c+d.

19.1 INTRODUCTION

The easiest way to do this is to use a stack. The algorithm uses a stack and is as follows:

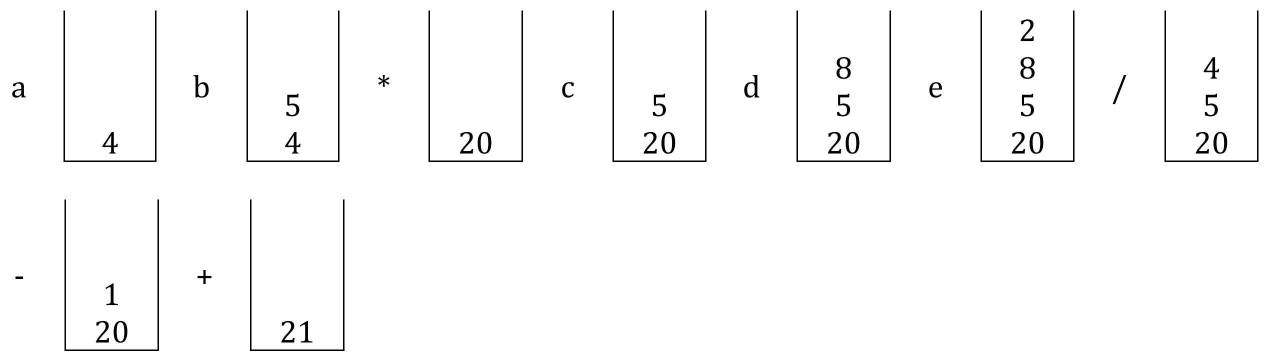
- Make an empty stack.
- Read the postfix expression one character at a time until it encounters end of expression.
- If the character is an operand, push its associated value onto the stack.
- If the character is an operator, pop two values from the stack, apply the operator to them and push the result onto the stack.

19.2 EXAMPLE-I

a b * c d e / - +

Let us consider the symbols a, b, c, d, e had the associated values:

a → 4 b → 5 c → 5 d → 8 e → 2



19.3 EXAMPLE-II

Consider the postfix expression 4 5 6 * +. As you scan the expression from left to right, you first encounter the operands 4 and 5. Placing each on the stack ensures that they are available if an operator comes next.

In this case, the next symbol is another operand. So, as before, push it and check the next symbol. Now we see an operator, *. This means that the two most recent operands need to be used in a multiplication operation. By popping the stack twice, we can get the proper operands and then perform the multiplication (in this case getting the result 30).

We can now handle this result by placing it back on the stack so that it can be used as an operand for the later operators in the expression. When the final operator is processed, there will be only one value left on the stack. Pop and return it as the result of the expression. Fig. 9.1 shows the stack contents as this entire example expression is being processed.

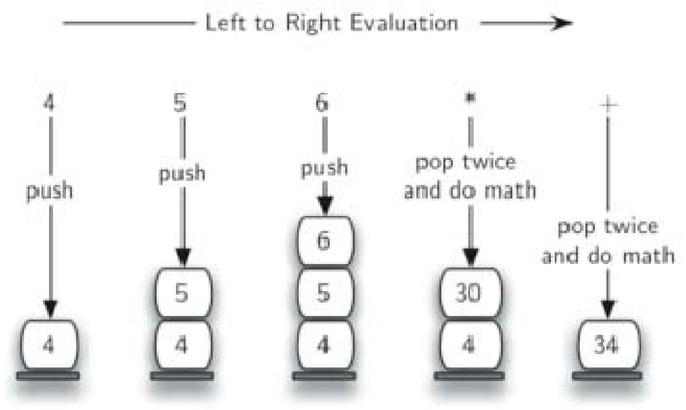


Fig. 19.1 Sack Contents During Evaluation

19.4 EXAMPLE-III

Fig. 19.2 shows a slightly more complex example, 7 8 + 3 2 + /. There are two things to note in this example. First, the stack size grows, shrinks, and then grows again as the sub-expressions are evaluated. Second, the division operation needs to be handled carefully. Recall that the operands in the postfix expression are in their original order since postfix changes only the placement of operators. When the operands for the division are popped from the stack, they are reversed. Since division is not a commutative operator, in other words 15/515/5 is not the same as 5/155/15, we must be sure that the order of the operands is not switched.

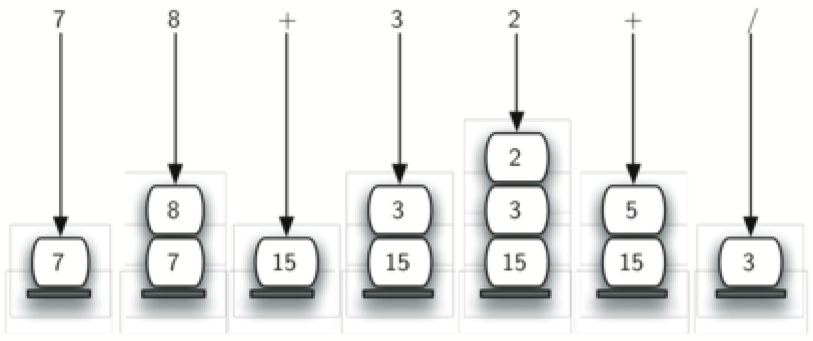


Fig. 19.2 A More Complex Example of Evaluation

19.5 PROGRAM (USING ARRAY)

```
#include <stdio.h>
#include <string.h>

#define MAX 20

int Stack[MAX], top = -1;
char expr[MAX];

void Push(int ele);
int Pop();

int main()
{
     int i, a, b, c, e;
```

```
	printf("Enter the postfix expression : ");
	gets(expr);
	for(i = 0; i < strlen(expr); i++)
	{
		if(expr[i]=='+'||expr[i]=='-'||expr[i]=='*'||expr[i]=='/')
		{
			b = Pop();
			a = Pop();
			switch(expr[i])
			{
				case '+':
					c = a + b;
					Push(c);
					break;
				case '-':
					c = a - b;
					Push(c);
					break;
				case '*':
					c = a * b;
					Push(c);
					break;
				case '/':
					c = a / b;
					Push(c);
					break;
			}
		}

		else
		{
			printf("Enter the value of %c : ", expr[i]);
			scanf("%d", &e);
			Push(e);
		}
	}
	printf("The result is %d", Pop());
	return 0;
}
void Push(int ele)
{
	top = top + 1;
	Stack[top] = ele;
}
int Pop()
{
	int e;
	e = Stack[top];
	top = top - 1;
	return e;
}
```

Output

```
Enter the postfix expression : abc+*d*
Enter the value of a : 2
Enter the value of b : 3
Enter the value of c : 4
Enter the value of d : 5
The result is 70
```

19.6 PROGRAM (LINKED LIST)

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

#define MAX 20

struct node
{
	int Element;
	struct node *Next;
}*List = NULL;

typedef struct node Stack;

void Push(int e);
int Pop();

int main()
{
	int i, a, b, c, e;
	char expr[MAX];
	printf("Enter the postfix expression : ");
	gets(expr);
	for(i = 0; i < strlen(expr); i++)
	{
		if(expr[i]=='+'||expr[i]=='-'||expr[i] =='*'||expr[i]=='/')
		{
			b = Pop();
			a = Pop();
			switch(expr[i])
			{
				case '+':
					c = a + b;
					Push(c);
					break;
				case '-':
					c = a - b;
					Push(c);
					break;
```

```
                        case '*':
                             c = a * b;
                             Push(c);
                             break;

                        case '/':
                             c = a / b;
                             Push(c);
                             break;
                   }
              }
              else
              {
                   printf("Enter the value of %c : ", expr[i]);
                   scanf("%d", &e);
                   Push(e);
              }
         }
         printf("The result is %d", Pop());
         return 0;
}

void Push(int e)
{
         Stack *NewNode = malloc(sizeof(Stack));
         NewNode->Element = e;
         if(List == NULL)
              NewNode->Next = NULL;
         else
              NewNode->Next = List;
         List = NewNode;
}
int Pop()
{
         int e;
         Stack *TempNode;
         TempNode = List;
         List = List->Next;
         e = TempNode->Element;
         free(TempNode);
         return e;
}
```

Output

```
Enter the postfix expression : abc+*d*
Enter the value of a : 2
Enter the value of b : 3
Enter the value of c : 4
Enter the value of d : 5
The result is 70
```

REVIEW QUESTIONS

1. How will you evaluate a postfix expression?

CHAPTER - 20 - QUEUE ADT

20.1 INTRODUCTION

A queue is list with the restrictions that insertion is done at one end (rear), whereas deletion is performed at the other end (front). It follows First-In-First-Out (FIFO) principle.

Example: Waiting line in a reservation counter.

20.2 OPERATIONS ON QUEUE

The basic operations on a queue are:

- Enqueue - which inserts an element at the end of the list (called rear).
- Dequeue - which deletes (and returns) the element at the start of the list (known as front).

Fig. 20.1 shows the abstract model of a queue.

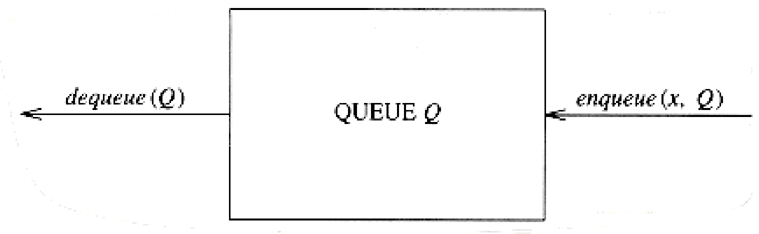


Fig. 20.1 Model of a Queue

Fig. 20.2 Queue of People

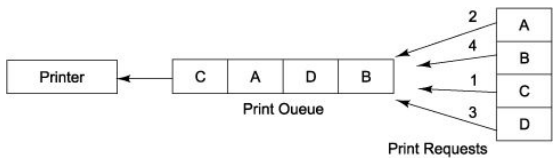


Fig. 20.3 Queue Implementation

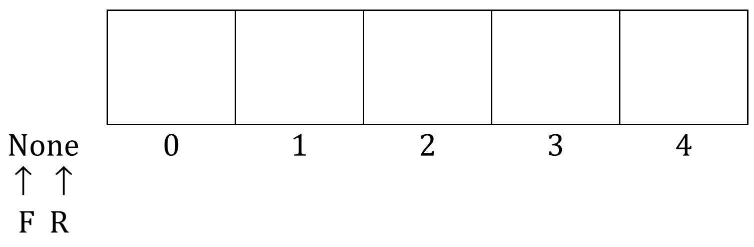


Fig. 20.3 Empty Queue (Initial state)

20.3 ENQUEUE

- The process of inserting a new element on to the rear of the queue is called enqueue operation.
- For every enqueue operation the rear pointer is incremented by 1.

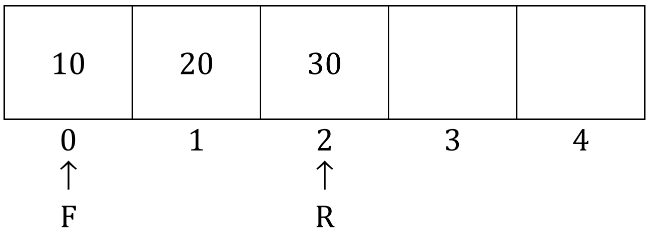


Fig. 20.4 Enqueue Operations

10.4 DEQUEUE

- The process of deleting an element from the front of queue is called dequeue operation.
- After every dequeue operation the front pointer is incremented by 1.

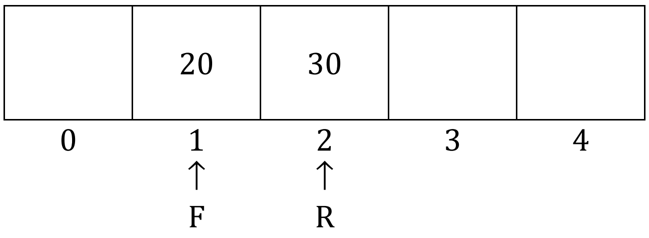


Fig. 20.5 Dequeue Operations

20.5 EXCEPTIONAL CONDITIONS

20.5.1 Overflow

Attempt to insert an element when the queue is full is said to be overflow.

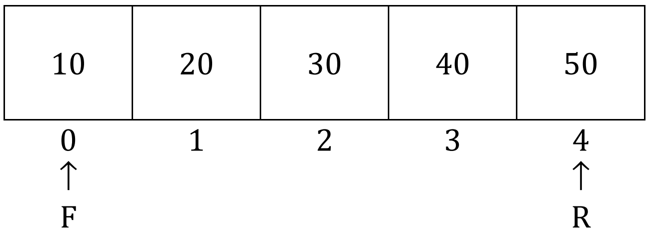


Fig. 20.6 Full Queue (Size = 5)

20.5.2 Underflow

Attempt to delete an element, when the queue is empty is said to be underflow.

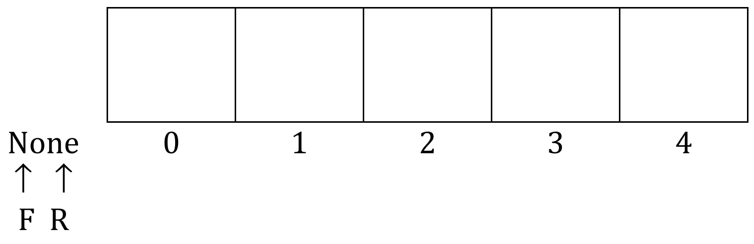


Fig. 20.7 Empty Queue

20.6 IMPLEMENTATION OF QUEUE

Queue can be implemented using:

- Array
- Linked list

20.7 APPLICATIONS OF QUEUE

The applications of queue data structure are:

- Batch processing in an operating system
- To implement priority queues
- Mathematics user queuing theory
- Computer networks where the server takes the jobs of the client as per the queue strategy

REVIEW QUESTIONS

1. Define a queue model. (or) Define Queue.
2. List out the operations on queue.
3. What are the different ways to implement queue?
4. What are the exception conditions of a queue?
5. What are front and rear pointers? Explain their significance.
6. What are the different application areas of queue data structure?

CHAPTER - 21 - ARRAY IMPLEMENTATION OF QUEUE

21.1 INTRODUCTION

In this implementation each queue is associated with two pointers namely front and rear, which is -1 for an empty queue.

21.1.1 Enqueue

To insert an element X onto the queue, the rear pointer is incremented by 1 and then set:

Queue[rear] = X.

21.1.2 Dequeue

To delete an element from the queue, the Queue[front] value is returned and the front pointer is incremented by 1.

21.2 QUEUE FULL

21.2.1 Algorithm to Check whether a Queue is Full

```
IsFull()

Step 1  : Start.
Step 2  : If rear = MAX - 1 goto Step 3 else goto Step 4.
Step 3  : Return 1 and Stop.
Step 4  : Return 0.
Step 5  : Stop.
```

21.2.2 Routine to Check whether a Queue is Full

```
int IsFull()
{
	if(rear == MAX - 1)
		return 1;
	else
		return 0;
}
```

21.3 QUEUE EMPTY

21.3.1 Algorithm to whether a Queue is Empty

```
IsEmpty()

Step 1  : Start.
Step 2  : If front = -1 goto Step 3 else goto Step 4.
Step 3  : Return 1 and Stop.
Step 4  : Return 0.
Step 5  : Stop.
```

21.3.2 Routine to whether a Queue is Empty

```
int IsEmpty()
{
	if(front == -1)
		return 1;
	else
		return 0;
}
```

21.4 ENQUEUE

21.4.1 Algorithm to Enqueue an Element on to the Queue

```
Enqueue(ele)

ele     : int

Step 1  : Start.
Step 2  : If IsFull() = True goto Step 3 else goto Step 4.
Step 3  : Display message "Queue is Overflow...!" and goto Step 8.
Step 4  : Set rear = rear + 1.
Step 5  : Set Queue[rear] = ele.
Step 6  : If front = -1 goto Step 7 else goto Step 8.
Step 7  : Set front = 0.
Step 8  : Stop.
```

21.4.2 Routine to Enqueue an Element on to the Queue

```
void Enqueue(int ele)
{
	if(IsFull())
		printf("Queue is Overflow...!\n");
	else
	{
		rear = rear + 1;
		Queue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}
```

21.5 DEQUEUE

21.5.1 Algorithm to Dequeue an Element from the Queue

```
Dequeue()

Step 1  : Start.
Step 2  : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3  : Display message "Queue is Underflow...!" and goto Step 8.
Step 4 : Display Queue[front].
Step 5 : If front = rear goto Step 6 else goto Step 7.
Step 6 : Set front = rear = -1 and goto Step 8.
Step 7 : Set front = front + 1.
Step 8 : Stop.
```

21.5.2 Routine to Dequeue an Element from the Queue

```c
void Dequeue()
{
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", Queue[front]);
		if(front == rear)
			front = rear = -1;
		else
			front = front + 1;
	}
}
```

21.6 DISPLAY

21.6.1 Algorithm to Display Queue Elements

```
Display()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message "Queue is Underflow...!" and goto Step 8.
Step 4 : Set i = front.
Step 5 : Repeat Steps 6 to 7 while i <= rear.
Step 6 : Display Queue[i].
Step 7 : Set i = i + 1.
Step 8 : Stop.
```

21.6.2 Routine to Display Queue Elements

```c
void Display()
{
	int i;
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		for(i = front; i <= rear; i++)
			printf("%d\t", Queue[i]);
		printf("\n");
	}
}
```

21.7 PROGRAM

```
#include <stdio.h>

#define MAX 5

int Queue[MAX], front = -1, rear = -1;

int IsFull();
int IsEmpty();
void Enqueue(int ele);
void Dequeue();
void Display();

int main()
{
	int ch, e;
	do
	{
		printf("1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT");
		printf("\nEnter your choice : ");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				printf("Enter the element : ");
				scanf("%d", &e);
				Enqueue(e);
				break;
			case 2:
				Dequeue();
				break;
			case 3:
				Display();
				break;
		}
	} while(ch <= 3);
	return 0;
}

int IsFull()
{
	if(rear == MAX - 1)
		return 1;
	else
		return 0;
}
```

```c
int IsEmpty()
{
	if(front == -1)
		return 1;
	else
		return 0;
}

void Enqueue(int ele)
{
	if(IsFull())
		printf("Queue is Overflow...!\n");
	else
	{
		rear = rear + 1;
		Queue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}

void Dequeue()
{
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", Queue[front]);
		if(front == rear)
			front = rear = -1;
		else
			front = front + 1;
	}
}

void Display()
{
	int i;
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		for(i = front; i <= rear; i++)
			printf("%d\t", Queue[i]);
		printf("\n");
	}
}
```

Output

```
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 60
Queue is Overflow...!
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
10      20      30      40      50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
Queue is Underflow...!
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
Queue Underflow...!
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 4
```

REVIEW QUESTIONS

1. How do you enqueue and dequeue elements in an array queue?
2. Write down a function to insert an element into a queue, in which the queue is implemented as an array.

CHAPTER - 22 - LINKED LIST IMPLEMENTATION OF QUEUE

22.1. INTRODUCTION

In this implementation:

- Enqueue operation is performed by inserting an element at the end of the list.
- Dequeue operation is performed by deleting at the front of the list.

22.2 TYPE DECLARATIONS FOR LINKED LIST IMPLEMENTATION OF THE QUEUE

```
struct node
{
     int Element;
     struct node *Next;
}*Front = NULL, *Rear = NULL;

typedef struct node Queue;
```

22.3 EMPTY LIST

22.3.1 Algorithm to whether a Queue is Empty

```
IsEmpty(List)
Step 1 : Start.
Step 2 : If List = NULL goto Step 3 else goto Step 4.
Step 3 : Return 1 and Stop.
Step 4 : Return 0 and Stop.
Step 5 : Stop.
```

22.3.2 Routine to whether a Queue is Empty

```
int IsEmpty(Queue *List)
{
     if(List == NULL)
          return 1;
     else
          return 0;
}
```

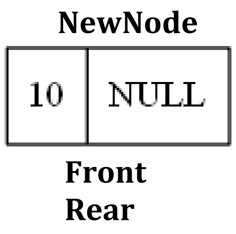
22.4 ENQUEUE

The enqueue is implemented as an insertion into the end of a linked list.

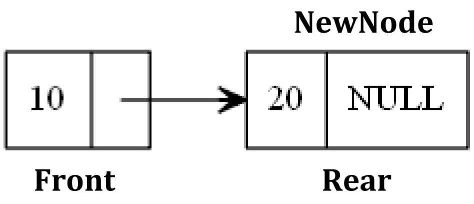
Example

The general idea is shown in Figure.

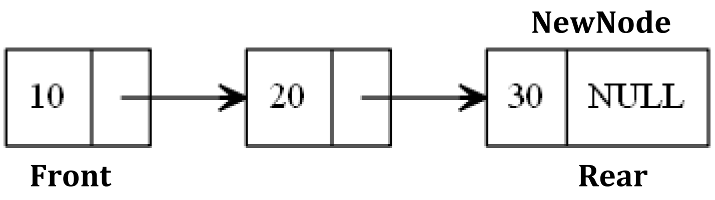
Enqueue(10)



Enqueue(20)



Enqueue(30)



22.4.1 Algorithm to Enqueue an Element on to the queue

```
Enqueue(e)

e       : int

Step 1 : Start.
Step 2 : Set NewNode = addressof(Queue).
Step 3 : Set NewNode→Element = e.
Step 4 : Set NewNode→Next = NULL.
Step 5 : If Rear = NULL, then goto Step 6 else goto Step 7.
Step 6 : Set Front = Rear = NewNode and goto Step 9.
Step 7 : Set Rear→Next = NewNode.
Step 8 : Set Rear = NewNode.
Step 9 : Stop.
```

22.4.2 Routine to Enqueue an Element on to the Queue

```c
void Enqueue(int e)
{
	Queue *NewNode = malloc(sizeof(Queue));
	NewNode->Element = e;
	NewNode->Next = NULL;
	if(Rear == NULL)
		Front = Rear = NewNode;
	else
	{
		Rear->Next = NewNode;
		Rear = NewNode;
	}
}
```

22.5 DEQUEUE

Dequeue as a deletion from the front of the list.

Example

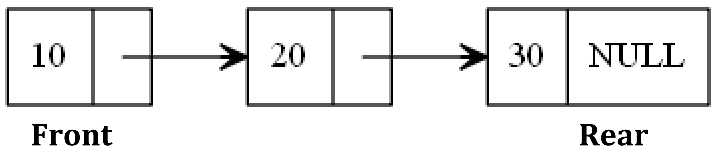
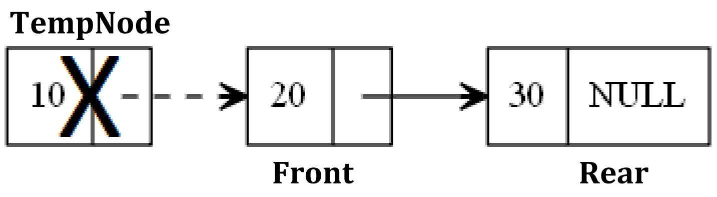


Fig. Initial Queue

Dequeue()



22.5.1 Algorithm to Dequeue an Element from the Queue

Dequque()

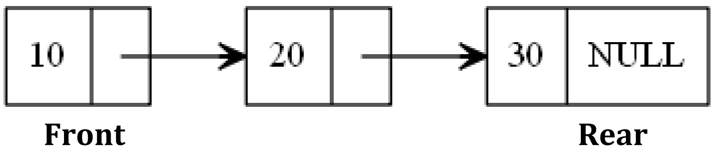
```
Step 1  : Start.
Step 2  : If IsEmpty(Front) = True, then goto Step 3 else goto Step 4.
Step 3  : Display "Queue is Underflow...!" and goto Step 10.
Step 4  : Set TempNode = Front.
Step 5  : If Front = Rear goto Step 6 else goto Step 7.
Step 6  : Set Front = Rear = NULL and goto Step 8.
Step 7  : Set Front = Front→Next.
Step 8  : Display TempNode→Element.
Step 9  : Delete TempNode.
Step 10: Stop.
```

22.5.2 Routine to Dequeue an Element from the Queue

```c
void Dequeue()
{
      if(IsEmpty(Front))
            printf("Queue is Underflow...!\n");
      else
      {
            Queue *TempNode;
            TempNode = Front;
            if(Front == Rear)
                  Front = Rear = NULL;
            else
                  Front = Front->Next;
            printf("%d\n", TempNode->Element);
            free(TempNode);
      }
}
```

22.6. DISPLAY

Example



22.6.1 Algorithm to Display Queue Elements

```
Display(List)

Step 1 : Start.
Step 2 : If IsEmpty(Front) = TRUE goto Step 3 else goto Step 4.
Step 3 : Display "Queue is Underflow...!" and goto Step 8.
Step 4 : Set Position = Front.
Step 5 : Repeat the Steps 6-7 until Position != NULL.
Step 6 : Display Position→Element.
Step 7 : Set Position = Position→Next.
Step 8 : Stop.
```

22.6.2 Routine to Display Queue Elements

```
void Display()
{
	if(IsEmpty(Front))
		printf("Queue is Underflow...!\n");
	else
	{
		Queue *Position;
		Position = Front;
		while(Position != NULL)
		{
			printf("%d\t", Position->Element);
			Position = Position->Next;
		}
		printf("\n");
	}
}
```

22.7 PROGRAM

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

struct node
{
	int Element;
	struct node *Next;
}*Front = NULL, *Rear = NULL;
typedef struct node Queue;
```

```
int IsEmpty(Queue *List);
void Enqueue(int e);
void Dequeue();
void Display();
int main()
{
	int ch, e;
	do
	{
		printf("1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT");
		printf("\nEnter your choice : ");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				printf("Enter the element : ");
				scanf("%d", &e);
				Enqueue(e);
				break;
			case 2:
				Dequeue();
				break;
			case 3:
				Display();
				break;
		}
	} while(ch <= 3);
	return 0;
}

int IsEmpty(Queue *List)
{
	if(List == NULL)
		return 1;
	else
		return 0;
}

void Enqueue(int e)
{
	Queue *NewNode = malloc(sizeof(Queue));
	NewNode->Element = e;
	NewNode->Next = NULL;
	if(Rear == NULL)
		Front = Rear = NewNode;
	else
	{
		Rear->Next = NewNode;
		Rear = NewNode;
	}
}
```

```
void Dequeue()
{
	if(IsEmpty(Front))
		printf("Queue is Underflow...!\n");
	else
	{
		Queue *TempNode;
		TempNode = Front;
		if(Front == Rear)
			Front = Rear = NULL;
		else
			Front = Front->Next;
		printf("%d\n", TempNode->Element);
		free(TempNode);
	}
}

void Display()
{
	if(IsEmpty(Front))
		printf("Queue is Underflow...!\n");
	else
	{
		Queue *Position;
		Position = Front;
		while(Position != NULL)
		{
			printf("%d\t", Position->Element);
			Position = Position->Next;
		}
		printf("\n");
	}
}
```

Output

```
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 50
```

```
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
10      20      30      40      50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
Queue is Underflow...!
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 4
```

REVIEW QUESTIONS

1. How do you enqueue and dequeue elements in a linked list queue?
2. Write down a function to insert an element into a queue, in which the queue is implemented as a linked list.

23.1 INTRODUCTION

A circular queue is a queue whose start and end locations are logically connected with each other. That means, the start location comes after the end location. If we continue to add elements in a circular queue till its end location, then after the end location has been filled, the next element will be added at the beginning of the queue.

Fig. 23.1 shows the logical representation of a circular queue.

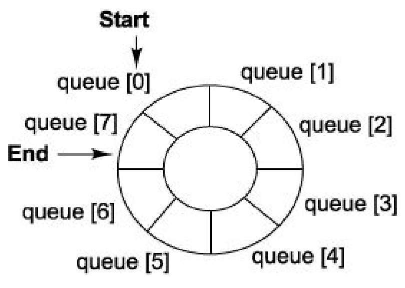


Fig. 23.1 Circular Queue

As we can see in Fig. 23.2, the start location of the queue comes after its end location. Thus, if the queue is filled till its capacity, i.e., the end location, then the start location will be checked for space, and if it is empty, the new element will be added there.

Figure shows the different states of a circular queue during insert and delete operations.

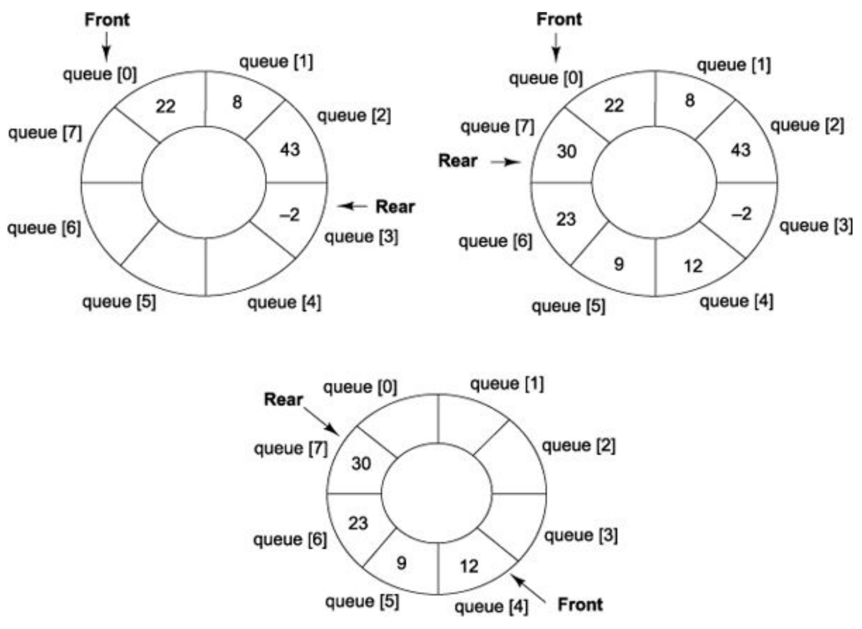


Fig. 23.2 Inserting and deleting elements in a circular queue

23.1.1 Advantages

Circular queues remove one of the main disadvantages of array implemented queues in which a lot of memory space is wasted due to inefficient utilization.

23.2 QUEUE FULL

23.2.1 Algorithm to Check whether a Queue is Full

```
IsFull()

Step 1 : Start.
Step 2 : If front = (rear + 1) % MAX goto Step 3 else goto Step 4.
Step 3 : Return 1 and Stop.
Step 4 : Return 0.
Step 5 : Stop.
```

11.2.2 Routine to Check whether a Queue is Full

```c
int IsFull()
{
	if(front == (rear + 1) % MAX)
		return 1;
	else
		return 0;
}
```

23.3 QUEUE EMPTY

23.3.1 Algorithm to whether a Queue is Empty

```
IsEmpty()

Step 1 : Start.
Step 2 : If front = -1 goto Step 3 else goto Step 4.
Step 3 : Return 1 and Stop.
Step 4 : Return 0.
Step 5 : Stop.
```

23.3.2 Routine to whether a Queue is Empty

```c
int IsEmpty()
{
	if(front == -1)
		return 1;
	else
		return 0;
}
```

23.4 ENQUEUE

23.4.1 Algorithm to Enqueue an Element on to the Queue

```
Enqueue(ele)

ele     : int

Step 1 : Start.
Step 2 : If IsFull() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Overflow…!” and goto Step 8.
Step 4 : Set rear = (rear + 1) % MAX.
Step 5 : Set CQueue[rear] = ele.
Step 6 : If front = -1 goto Step 7 else goto Step 8.
Step 7 : Set front = 0.
Step 8 : Stop.
```

23.4.2 Routine to Enqueue an Element on to the Queue

```c
void Enqueue(int ele)
{
	if(IsFull())
		printf("Queue is Overflow...!\n");
	else
	{
		rear = (rear + 1) % MAX;
		CQueue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}
```

23.5 DEQUEUE

23.5.1 Algorithm to Dequeue an Element from the Queue

```
Dequeue()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Underflow…!” and goto Step 8.
Step 4 : Display CQueue[front].
Step 5 : If front = rear goto Step 6 else goto Step 7.
Step 6 : Set front = rear = -1 and goto Step 8.
Step 7 : Set front = (front + 1) % MAX.
Step 8 : Stop.
```

23.5.2 Routine to Dequeue an Element from the Queue

```
void Dequeue()
{
      if(IsEmpty())
            printf("Queue is Underflow...!\n");
      else
      {
            printf("%d\n", CQueue[front]);
            if(front == rear)
                  front = rear = -1;
            else
                  front = (front + 1) % MAX;
      }
}
```

23.6 DISPLAY

23.6.1 Algorithm to Display Queue Elements

```
Display()

Step 1 : Start.
Step 2 : If IsEmpty() = True goto Step 3 else goto Step 4.
Step 3 : Display message "Queue is Underflow...!" and goto Step 9.
Step 4 : Set i = front.
Step 5 : Repeat Steps 6 to 7 while i != rear.
Step 6 : Display CQueue[i].
Step 7 : Set i = (i + 1) % MAX.
Step 8 : Display CQueue[i].
Step 9 : Stop.
```

23.6.2 Routine to Display Queue Elements

```
void Display()
{
      int i;
      if(IsEmpty())
            printf("Queue is Underflow...!\n");
      else
      {
            for(i = front; i != rear; i = (i + 1) % MAX)
                  printf("%d\t", CQueue[i]);
            printf("%d\n", CQueue[i]);
      }
}
```

23.7 PROGRAM

```
#include <stdio.h>

#define MAX 5

int CQueue[MAX], front = -1, rear = -1;

int IsFull();
int IsEmpty();
void Enqueue(int ele);
void Dequeue();
void Display();

int main()
{
      int ch, e;
      do
      {
            printf("1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT");
            printf("\nEnter your choice : ");
            scanf("%d", &ch);
            switch(ch)
            {
                  case 1:
                        printf("Enter the element : ");
                        scanf("%d", &e);
                        Enqueue(e);
                        break;
                  case 2:
                        Dequeue();
                        break;
                  case 3:
                        Display();
                        break;
            }
      } while(ch <= 3);
      return 0;
}

int IsFull()
{
      if(front == (rear + 1) % MAX)
            return 1;
      else
            return 0;
}
```

```c
int IsEmpty()
{
	if(front == -1)
		return 1;
	else
		return 0;
}

void Enqueue(int ele)
{
	if(IsFull())
		printf("Queue is Overflow...!\n");
	else
	{
		rear = (rear + 1) % MAX;
		CQueue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}

void Dequeue()
{
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", CQueue[front]);
		if(front == rear)
			front = rear = -1;
		else
			front = (front + 1) % MAX;
	}
}

void Display()
{
	int i;
	if(IsEmpty())
		printf("Queue is Underflow...!\n");
	else
	{
		for(i = front; i != rear; i = (i + 1) % MAX)
			printf("%d\t", CQueue[i]);
		printf("%d\n", CQueue[i]);
	}
}
```

Output

```
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 20
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 30
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 40
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 60
Queue is Overflow...!
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
10      20      30      40      50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 2
10
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
20      30      40      50
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 1
Enter the element : 60
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 3
20      30      40      50      60
1.ENQUEUE 2.DEQUEUE 3.DISPLAY 4.EXIT
Enter your choice : 4
```

REVIEW QUESTIONS

1. Define circular queue.
2. What is the advantage of circular queue?

CHAPTER - 24 - DOUBLE ENDED QUEUE

24.1 INTRODUCTION

A double-ended queue is a special type of queue that allows insertion and deletion of elements at both ends, i.e., front and rear. In simple words, a double-ended queue can be referred as a linear list of elements in which insertion and deletion of elements takes place at its two ends but not in the middle. This is the reason why it is termed as double-ended queue or deque.

Fig. 24.1 shows the logical representation of a deque.

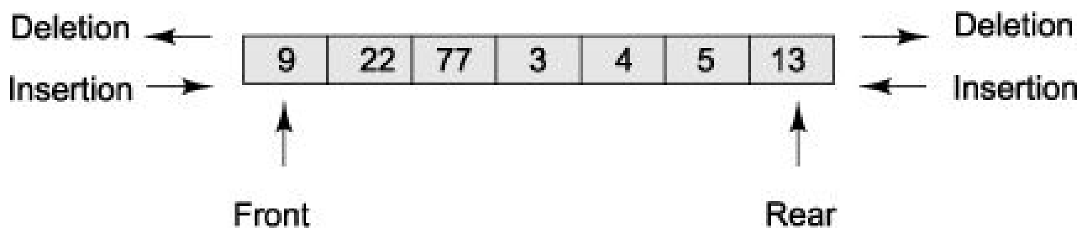


Fig. 24.1 Double-ended Queue

24.2 TYPES

Based on the type of restrictions imposed on insertion and deletion of elements, a double-ended queue is categorized into two types:

1. Input-restricted deque
2. Output-restricted deque

24.2.1 Input-restricted Deque

It allows deletion from both the ends but restricts the insertion at only one end.

24.2.2 Output-restricted Deque

It allows insertion at both the ends but restricts the deletion at only one end.

24.3 OPERATIONS

As shown in Fig., insertion and deletion of elements is possible at both front and rear ends of the queue. As a result, the following four operations are possible for a double-ended queue:

1.	EnqueueFront	:	Insertion at front end of the queue.
2.	DequeueFront	:	Deletion from front end of the queue.
3.	EnqueueRear	:	Insertion at rear end of the queue.
4.	DequeueRear	:	Deletion from rear end of the queue.

24.4 INSERTION AT FRONT END OF THE QUEUE

24.4.1 Algorithm to Enqueue an Element at Front

```
EnqueueFront(ele)

ele     : int

Step 1  : Start.
Step 2  : If front = 0 goto Step 3 else goto Step 4.
Step 3  : Display message “Cannot Insert at Front…!” and goto Step 11.
Step 4  : If front = –1 goto Step 5 else goto Step 9.
Step 5  : Set front = front + 1.
Step 6  : Set DQueue[front] = ele.
Step 7  : If rear = -1 goto Step 8 else goto Step 11.
Step 8  : Set rear = 0 and goto Step 11.
Step 9  : Set front = front – 1.
Step 10: Set DQueue[front] = ele.
Step 11: Stop.
```

24.4.2 Routine to Enqueue an Element at Front

```
void EnqueueFront(int ele)
{
	if(front == 0)
		printf("Cannot Insert at Front...!\n");
	else
	{
		if(front == -1)
		{
			front = front + 1;
			DQueue[front] = ele;
			if(rear == -1)
				rear = 0;
		}
		else
		{
			front = front - 1;
			DQueue[front] = ele;
		}

	}
}
```

24.5 DELETION FROM FRONT END OF THE QUEUE

24.5.1 Algorithm to Dequeue an Element from Front

```
DequeueFront()

Step 1 : Start.
Step 2 : If front = -1 goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Underflow...!” and goto Step 8.
Step 4 : Display DQueue[front].
Step 5 : If front = rear goto Step 6 else goto Step 7.
Step 6 : Set front = rear = -1 and goto Step 8.
Step 7 : Set front = front + 1.
Step 8 : Stop.
```

24.5.2 Algorithm to Dequeue an Element from Front

```c
void DequeueFront()
{
	if(front == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", DQueue[front]);
		if(front == rear)
			front = rear = -1;
		else
			front = front + 1;
	}
}
```

24.6 INSERTION AT REAR END OF THE QUEUE

24.6.1 Algorithm to Enqueue an Element at Rear

```
EnqueueRear(ele)

ele    : int

Step 1 : Start.
Step 2 : If rear = MAX – 1 goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Overflow...!” and goto Step 8.
Step 4 : Set rear = rear + 1.
Step 5 : Set DQueue[rear] = ele.
Step 6 : If front = -1 goto Step 7 else goto Step 8.
Step 7 : Set front = 0.
Step 8 : Stop.
```

24.6.2 Routine to Enqueue an Element at Rear

```
void EnqueueRear(int ele)
{
	if(rear == MAX - 1)
		printf("Queue is Overflow...!\n");
	else
	{
		rear = rear + 1;
		DQueue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}
```

24.7 DELETION FROM REAR END OF THE QUEUE

24.7.1 Algorithm to Dequeue an Element from Rear

```
DequeueRear()

Step 1 : Start.
Step 2 : If rear = -1 goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Underflow...!” and goto Step 8.
Step 4 : Display DQueue[front].
Step 5 : If front = rear goto Step 6 else goto Step 7.
Step 6 : Set front = rear = -1 and goto Step 8.
Step 7 : Set rear = rear - 1.
Step 8 : Stop.
```

24.7.2 Routine to Dequeue an Element from Rear

```
void DequeueRear()
{
	if(rear == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", DQueue[rear]);
		if(front == rear)
			front = rear = -1;
		else
			rear = rear - 1;
	}
}
```

24.8 DISPLAY THE ELEMENTS IN THE QUEUE

24.8.1 Algorithm to Display Queue Elements

```
Display()

Step 1 : Start.
Step 2 : If front = -1 goto Step 3 else goto Step 4.
Step 3 : Display message “Queue is Underflow…!” and goto Step 8.
Step 4 : Set i = front.
Step 5 : Repeat Steps 6 to 7 while i <= rear.
Step 6 : Display DQueue[i].
Step 7 : Set i = i + 1.
Step 8 : Stop.
```

24.8.2 Routine to Display Queue Elements

```c
void Display()
{
	int i;
	if(front == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		for(i = front; i <= rear; i++)
			printf("%d\t", DQueue[i]);
		printf("\n");
	}
}
```

13.9 PROGRAM

```c
#include <stdio.h>

#define MAX 5

int DQueue[MAX], front = -1, rear = -1;

void EnqueueRear(int ele);
void EnqueueFront(int ele);
void DequeueFront();
void DequeueRear();
void Display();

int main()
{
	int ch, e;
	do
	{
		printf("1.ENQUEUE-REAR 2.ENQUEUE-FRONT ");
		printf("3.DEQUEUE-FRONT 4.DEQUEUE-REAR 5.DISPLAY 6.EXIT");
		printf("\nEnter your choice : ");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1:
				printf("Enter the element : ");
				scanf("%d", &e);
				EnqueueRear(e);
				break;
			case 2:
				printf("Enter the element : ");
				scanf("%d", &e);
				EnqueueFront(e);
				break;
			case 3:
				DequeueFront();
				break;
			case 4:
				DequeueRear();
				break;
			case 5:
				Display();
				break;
		}
	} while(ch <= 5);
	return 0;
}
```

```
void EnqueueRear(int ele)
{
	if(rear == MAX - 1)
		printf("Queue is Overflow...!\n");
	else
	{
		rear = rear + 1;
		DQueue[rear] = ele;
		if(front == -1)
			front = 0;
	}
}

void EnqueueFront(int ele)
{
	if(front == 0)
		printf("Cannot Insert at Front...!\n");
	else
	{
		if(front == -1)
		{
			front = front + 1;
			DQueue[front] = ele;
			if(rear == -1)
				rear = 0;
		}
		else
		{
			front = front - 1;
			DQueue[front] = ele;
		}

	}
}

void DequeueFront()
{
	if(front == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", DQueue[front]);
		if(front == rear)
			front = rear = -1;
		else
			front = front + 1;
	}
}
```

```
void DequeueRear()
{
	if(rear == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		printf("%d\n", DQueue[rear]);
		if(front == rear)
			front = rear = -1;
		else
			rear = rear - 1;
	}
}

void Display()
{
	int i;
	if(front == -1)
		printf("Queue is Underflow...!\n");
	else
	{
		for(i = front; i <= rear; i++)
			printf("%d\t", DQueue[i]);
		printf("\n");
	}
}
```

Output

```
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 10
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 20
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 30
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 40
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 50
```

```
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 1
Enter the element : 60
Queue is Overflow...!
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 5
10      20      30      40      50
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 3
10
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 3
20
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 5
30      40      50
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 2
Enter the element : 22
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 2
Enter the element : 11
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 2
Enter the element : 1
Cannot Insert at Front...!
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 5
11      22      30      40      50
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 4
50
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 4
40
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 5
11      22      30
```

```
1.ENQUEUE-REAR  2.ENQUEUE-FRONT  3.DEQUEUE-FRONT  4.DEQUEUE-REAR  5.DISPLAY
6.EXIT
Enter your choice : 6
```

REVIEW QUESTIONS

1. Define double ended queue (deque). (or) How do you define double ended queue? (or) What is double ended queue?
2. What are the types of deque?
3. What is an input-restricted deque?
4. What is an output-restricted deque?
5. What are the operations of deque?
6. Explain the significance of double-ended queue.
7. List the differences between stack and queue data structures.

RAJALAKSHMI
ENGINEERING COLLEGE

Website : www.rajalakshmi.org

Elearning : www.rajalakshmicolleges.net/moodle

Give a man a fish and you feed him for a day.

Teach him how to fish and you feed him for a lifetime.