Ex. No. 1a Stack Array
Date:

Aim

To implement stack operations using array.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 5**
3. **Initialize *top* = -1**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
    If top < max -1
        Increment top
        Store element at current position of top
    Else
        Print Stack overflow
   Else If choice = 2 then
    If top < 0 then
        Print Stack underflow
    Else
        Display current top element
        Decrement top
   Else If choice = 3 then
    Display stack elements starting from top
7. **Stop**

Program

```
/* Stack Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int stack[max];
int top = -1;

void push(int x)
{
   stack[++top] = x;
}

int pop()
{
   return (stack[top--]);
}

void view()
{
   int i;
   if (top < 0)
      printf("\n Stack Empty \n");
   else
   {
      printf("\n Top-->");
      for(i=top; i>=0; i--)
      {
         printf("%4d", stack[i]);
      }
      printf("\n");
   }
}

main()
{
   int ch=0, val;
   clrscr();

   while(ch != 4)
   {
      printf("\n STACK   OPERATION \n");
      printf("1.PUSH  ");
      printf("2.POP  ");
      printf("3.VIEW  ");
      printf("4.QUIT \n");
      printf("Enter Choice : ");
```

```
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            if(top < max-1)
            {
               printf("\nEnter Stack element : ");
               scanf("%d", &val);
               push(val);
            }
            else
               printf("\n Stack Overflow \n");
            break;
         case 2:
            if(top < 0)
               printf("\n Stack Underflow \n");
            else
            {
               val = pop();
               printf("\n Popped element is %d\n", val);
            }
            break;
         case 3:
            view();
            break;
         case 4:
            exit(0);
         default:
            printf("\n Invalid Choice \n");
      }
   }
}
```

Output

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 23
```

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 34

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  45  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 2

Popped element is 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 4
```

Result

Thus push and pop operations of a stack was demonstrated using arrays.

Ex. No. 1b Queue Array
Date:

Aim

To implement queue operations using array.

Algorithm

1. **Start**
2. **Define a array *queue* of size *max* = 5**
3. **Initialize *front* = *rear* = –1**
4. **Display a menu listing queue operations**
5. **Accept choice**
6. **If choice = 1 then**
   If rear < max -1
      Increment rear
      Store element at current position of rear
   Else
      Print Queue Full
   Else If choice = 2 then
   If front = –1 then
      Print Queue empty
   Else
      Display current front element
      Increment front
   Else If choice = 3 then
      Display queue elements starting from front to rear.
7. **Stop**

Program

```
/* Queue Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int queue[max];
int front = -1;
int rear  = -1;

void insert(int x)
{
   queue[++rear] = x;
   if (front == -1)
      front = 0;
}

int remove()
{
   int val;
   val = queue[front];
   if (front==rear && rear==max-1)
      front = rear = -1;
   else
      front ++;
   return (val);
}

void view()
{
   int i;

   if (front == -1)
      printf("\n Queue Empty \n");
   else
   {
      printf("\n Front-->");
      for(i=front; i<=rear; i++)
         printf("%4d", queue[i]);
      printf("  <--Rear\n");
   }
}

main()
{
   int ch= 0,val;
   clrscr();
```

```
   while(ch != 4)
   {
      printf("\n QUEUE OPERATION \n");
      printf("1.INSERT  ");
      printf("2.DELETE  ");
      printf("3.VIEW  ");
      printf("4.QUIT\n");
      printf("Enter Choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            if(rear < max-1)
            {
               printf("\n Enter element to be inserted : ");
               scanf("%d", &val);
               insert(val);
            }
            else
               printf("\n Queue Full \n");
            break;
         case 2:
            if(front == -1)
               printf("\n Queue Empty \n");
            else
            {
               val = remove();
               printf("\n Element deleted : %d \n", val);
            }
            break;
         case 3:
            view();
            break;
         case 4:
            exit(0);
         default:
            printf("\n Invalid Choice \n");
      }
   }
}
```

Output

```
 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 12

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 23

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 34

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 45

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 56

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Queue Full

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 3

Front-->  12  23  34  45  56  <--Rear
```

Result

Thus insert and delete operations of a queue was demonstrated using arrays.

Ex. No. 2 **List using Array**
Date:

Aim

To perform various operations on List ADT using array implementation.

Algorithm

1. **Start**
2. **Create a list of n elements**
3. **Display list operations as a menu**
4. **Accept user choice**
5. **If choice = 1 then**
   - **Get position of element to be deleted**
   - **Move elements one position upwards thereon.**
   - **Decrement length of the list**

   Else if choice = 2
   - **Get position of element to be inserted.**
   - **Increment length of the list**
   - **Move elements one position downwards thereon**
   - **Store the new element in corresponding position**

   Else if choice = 3
   - **Traverse the list and inspect each element**
   - **Report position if it exists.**
6. **Stop**

Program

```
/* List operation using Arrays */

#include <stdio.h>
#include <conio.h>

void create();
void insert();
void search();
void deletion();
void display();

int i, e, n, pos;
static int b[50];

main()
{
   int ch;
   char g = 'y';
   create();
   do
   {
      printf("\n List Operations");
      printf("\n 1.Deletion\n 2.Insert\n 3.Search\n
                                         4.Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1:
            deletion();
            break;
         case 2:
            insert();
            break;
         case 3:
            search();
            break;
         case 4:
            exit(0);
         default:
         printf("\n Enter the correct choice:");
      }
      printf("Do you want to continue: ");
      fflush(stdin);
      scanf("\n %c",&g);
   } while(g=='y' || g=='Y');
   getch();
}
```

```
void create()
{
   printf("\n Enter the number of elements:");
   scanf("%d",&n);
   printf("\n Enter list elements: ");
   for(i=0; i<n; i++)
      scanf("%d", &b[i]);
}

void deletion()
{
   printf("\n enter the position you want to delete: ");
   scanf("%d", &pos);
   if(pos >= n)
      printf("\n Invalid location");
   else
   {
      for(i=pos+1; i<n; i++)
         b[i-1] = b[i];
      n--;
      printf("List elements after deletion");
      display();
   }
}

void search()
{
   int flag = 0;
   printf("\n Enter the element to be searched: ");
   scanf("%d", &e);
   for(i=0; i<n; i++)
   {
      if(b[i] == e)
      {
         flag = 1;
         printf("Element is in the %d position", i);
         break;
      }
   }
   if(flag == 0)
   printf("Value %d is not in the list", e);
}

void insert()
{
   printf("\n Enter the position you need to insert: ");
   scanf("%d", &pos);
   if(pos >= n)
      printf("\n Invalid location");
   else
   {
```

```
      ++n;
      for(i=n; i>pos; i--)
      b[i] = b[i-1];
      printf("\n Enter the element to insert: ");
      scanf("%d", &e);
      b[pos] = e;
   }
   printf("\n List after insertion:");
   display();
}

void display()
{
   for(i=0; i<n; i++)
      printf("\n %d", b[i]);
}
```

Output

```
 Enter the number of elements:5
 Enter list elements: 12 23 34 45 56

 List Operations
 1.Deletion
 2.Insert
 3.Search
 4.Exit
 Enter your choice:2
 Enter the position you need to insert: 1
 Enter the element to insert: 99
 List after insertion:
 12
 99
 23
 34
 45
 56
 Do you want to continue: y

 List Operations
 1.Deletion
 2.Insert
 3.Search
 4.Exit
 Enter your choice:1
 Enter the position you want to delete: 3
```

```
Elements after deletion
 12
 99
 23
 45
 56
 Do you want to continue: n
```

Result

Thus various operations was successfully executed on list using array implementation.

Ex. No. 3a **Singly Linked List**
Date:

Aim

To define a singly linked list node and perform operations such as insertions and deletions dynamically.

Algorithm

1. **Start**
2. **Define single linked list *node* as self referential structure**
3. **Create *Head* node with label = -1 and next = NULL using**
4. **Display menu on list operation**
5. **Accept user choice**
6. **If choice = 1 then**
   Locate node after which insertion is to be done
   Create a new node and get data part
   Insert new node at appropriate position by manipulating address
   Else if choice = 2
   Get node's data to be deleted.
   Locate the node and delink the node
   Rearrange the links
   Else
   Traverse the list from Head node to node which points to null
7. **Stop**

Program

```
/* Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
#include <string.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch, fou=0;
   int k;
   struct node *h, *temp, *head, *h1;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->label = -1;
   head->next = NULL;

   while(-1)
   {
      clrscr();
      printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
      printf("1->Add  ");
      printf("2->Delete  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         /* Add a node at any intermediate location */
         case 1:
            printf("\n Enter label after which to add : ");
            scanf("%d", &k);

            h = head;
            fou = 0;

            if (h->label == k)
               fou = 1;
```

```
            while(h->next != NULL)
            {
               if (h->label == k)
               {
                    fou=1;
                    break;
               }
               h = h->next;
            }
            if (h->label == k)
               fou = 1;

            if (fou != 1)
               printf("Node not found\n");
            else
            {
            temp=(struct node *)(malloc(sizeof(struct node)));
               printf("Enter label for new node : ");
               scanf("%d", &temp->label);
               temp->next = h->next;
               h->next = temp;
            }
            break;

         /* Delete any intermediate node */
         case 2:
            printf("Enter label of node to be deleted\n");
            scanf("%d", &k);
            fou = 0;
            h = h1 = head;
            while (h->next != NULL)
            {
               h = h->next;
               if (h->label == k)
               {
                    fou = 1;
                    break;
               }
            }

            if (fou == 0)
               printf("Sorry Node not found\n");
            else
               {
               while (h1->next != h)
                    h1 = h1->next;
               h1->next = h->next;
               free(h);
               printf("Node deleted successfully \n");
            }
            break;
```

```
         case 3:
            printf("\n\n HEAD -> ");
            h=head;
            while (h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : -1
Enter label for new node : 23

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : 23
Enter label for new node : 67

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 3
 HEAD -> 23 -> 67 -> NULL
```

Result

Thus operation on single linked list is performed.

Ex. No. 3b **Stack Using Linked List**
Date:

Aim

To implement stack operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   - **Create a new node with data**
   - **Make new node point to first node**
   - **Make head node point to new node**

   Else If choice = 2 then
   - **Make temp node point to first node**
   - **Make head node point to next of temp node**
   - **Release memory**

   Else If choice = 3 then
   - **Display stack elements starting from head node till null**
7. **Stop**

Program

```
/* Stack using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch = 0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Stack using Linked List \n");
      printf("1->Push  ");
      printf("2->Pop  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            h = head;
            temp->next = h->next;
            h->next = temp;
            break;

         case 2:
            /* Delink the first node */
            h = head->next;
            head->next = h->next;
```

```
            printf("Node %s deleted\n", h->label);
            free(h);
            break;

         case 3:
            printf("\n HEAD -> ");
            h = head;
            /* Loop till last node */
            while(h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL \n");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23
New node added

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 34

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result

Thus push and pop operations of a stack was demonstrated using linked list.

Ex. No. 3c **Queue Using Linked List**
Date:

Aim

To implement queue operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   Create a new node with data
   Make new node point to first node
   Make head node point to new node
   Else If choice = 2 then
   Make temp node point to first node
   Make head node point to next of temp node
   Release memory
   Else If choice = 3 then
   Display stack elements starting from head node till null
7. **Stop**

Program

```
/* Queue using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch=0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Queue using Linked List \n");
      printf("1->Insert  ");
      printf("2->Delete  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);

            /* Reorganize the links */
            h = head;
            while (h->next != NULL)
               h = h->next;
            h->next = temp;
            temp->next = NULL;
            break;
```

```
         case 2:
            /* Delink the first node */
            h = head->next;
            head->next = h->next;
            printf("Node deleted \n");
            free(h);
            break;

         case 3:
            printf("\n\nHEAD -> ");
            h=head;
            while (h->next!=NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL \n");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 12

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 3
HEAD -> 12 -> 23 -> NULL
```

Result

Thus insert and delete operations of a queue was demonstrated using linked list.

Ex. No. 4a **Infix To Postfix Conversion**
Date:

Aim

To convert infix expression to its postfix form using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the infix expression character-by-character**
    - **If character is an operand print it**
    - **If character is an operator**
        - **Compare the operator's priority with the stack[top] operator.**
        - **If the stack [top] has higher/equal priority than the input operator,**
            - **Pop it from the stack and print it.**
        - **Else**
            - **Push the input operator onto the stack**
    - **If character is a left parenthesis, then push it onto the stack.**
    - **If character is a right parenthesis, pop all operators from stack and print it until a left parenthesis is encountered. Do not print the parenthesis.**
    - **If character = $ then Pop out all operators, Print them and Stop**

.

Program

```
/* Conversion of infix to postfix expression */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20

int top = -1;
char stack[MAX];
char pop();
void push(char item);

int prcd(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
         return 2;
         break;
      case '*':
      case '/':
         return 4;
         break;
      case '^':
      case '$':
         return 6;
         break;
      case '(':
      case ')':
      case '#':
         return 1;
         break;
   }
}

int isoperator(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
      case '*':
      case '/':
      case '^':
      case '$':
      case '(':
      case ')':
         return 1;
```

```
            break;
        default:
            return 0;
    }
}

void convertip(char infix[],char postfix[])
{
    int i,symbol,j = 0;
    stack[++top] = '#';
    for(i=0;i<strlen(infix);i++)
    {
        symbol = infix[i];
        if(isoperator(symbol) == 0)
        {
            postfix[j] = symbol;
            j++;
        }
        else
        {
            if(symbol == '(')
                push(symbol);
            else if(symbol == ')')
            {
                while(stack[top] != '(')
                {
                    postfix[j] = pop();
                    j++;
                }
                pop(); //pop out (.
            }
            else
            {
                if(prcd(symbol) > prcd(stack[top]))
                    push(symbol);
                else
                {
                    while(prcd(symbol) <= prcd(stack[top]))
                    {
                         postfix[j] = pop();
                         j++;
                    }
                    push(symbol);
                }
            }
        }
    }

    while(stack[top] != '#')
    {
        postfix[j] = pop();
```

```
      j++;
   }
   postfix[j] = '\0';
}

main()
{
   char infix[20],postfix[20];
   clrscr();
   printf("Enter the valid infix string: ");
   gets(infix);
   convertip(infix, postfix);
   printf("The corresponding postfix string is: ");
   puts(postfix);
   getch();
}

void push(char item)
{
   top++;
   stack[top] = item;
}

char pop()
{
   char a;
   a = stack[top];
   top--;
   return a;
}
```

Output

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/

Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+

Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

Result

Thus the given infix expression was converted into postfix form using stack.

Ex. No. 4b **Postfix Expression Evaluation**
Date:

Aim

To evaluate the given postfix expression using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the postfix expression character-by-character**
   - **If character is an operand push it onto the stack**
   - **If character is an operator**
     - **Pop topmost two elements from stack.**
     - **Apply operator on the elements and push the result onto the stack,**
5. **Eventually only result will be in the stack at end of the expression.**
6. **Pop the result and print it.**
7. **Stop**

Program

```
/* Evaluation of Postfix expression using stack */

#include <stdio.h>
#include <conio.h>

struct stack
{
   int top;
   float a[50];
}s;

main()
{
   char pf[50];
   float d1,d2,d3;
   int i;
   clrscr();

   s.top = -1;
   printf("\n\n Enter the postfix expression: ");
   gets(pf);
   for(i=0; pf[i]!='\0'; i++)
   {
      switch(pf[i])
      {
         case '0':
         case '1':
         case '2':
         case '3':
         case '4':
         case '5':
         case '6':
         case '7':
         case '8':
         case '9':
            s.a[++s.top] = pf[i]-'0';
            break;

         case '+':
            d1 = s.a[s.top--];
            d2 = s.a[s.top--];
            s.a[++s.top] = d1 + d2;
            break;

         case '-':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1 - d2;
            break;
```

```
         case '*':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1*d2;
            break;

         case '/':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1 / d2;
            break;
      }
   }
   printf("\n Expression value is %5.2f", s.a[s.top]);
   getch();
}
```

Output

```
Enter the postfix expression: 6523+8*+3+*
Expression value is 288.00
```

Result

Thus the given postfix expression was evaluated using stack.

Exp. No. 4c FCFS Scheduling
Date:

Aim

To schedule snapshot of processes queued according to FCFS scheduling.

Process Scheduling

- **CPU scheduling is used in multiprogrammed operating systems.**
- **By switching CPU among processes, efficiency of the system can be improved.**
- **Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.**
- **Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.**

First Come First Serve (FCFS)

- **Process that comes first is processed first**
- **FCFS scheduling is non-preemptive**
- **Not efficient as it results in long average waiting time.**
- **Can result in starvation, if processes at beginning of the queue have long bursts.**

Algorithm

1. **Define an array of structure *process* with members *pid*, *btime*, *wtime* & *ttime*.**
2. **Get length of the ready queue, i.e., number of process (say *n*)**
3. **Obtain *btime* for each process.**
4. **The *wtime* for first process is 0.**
5. **Compute *wtime* and *ttime* for each process as:**
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. **Compute average waiting time *awat* and average turnaround time *atur***
7. **Display the *btime*, *ttime* and *wtime* for each process.**
8. **Display GANTT chart for the above scheduling**
9. **Display *awat* time and *atur***
10. **Stop**

Program

```
/* FCFS Scheduling  - fcfs.c */

#include <stdio.h>

struct process
{
   int pid;
   int btime;
   int wtime;
   int ttime;
} p[10];

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d (in ms) : ",(i+1));
      scanf("%d", &p[i].btime);
      p[i].pid = i+1;
   }

   p[0].wtime = 0;
   for(i=0; i<n; i++)
   {
      p[i+1].wtime = p[i].wtime + p[i].btime;
      p[i].ttime = p[i].wtime + p[i].btime;
   }
   ttur = twat = 0;
   for(i=0; i<n; i++)
   {
      ttur += p[i].ttime;
      twat += p[i].wtime;
   }
   awat = (float)twat / n;
   atur = (float)ttur / n;

   printf("\n       FCFS Scheduling\n\n");
   for(i=0; i<28; i++)
      printf("-");
   printf("\nProcess B-Time T-Time W-Time\n");
   for(i=0; i<28; i++)
      printf("-");
```

```
   for(i=0; i<n; i++)
      printf("\n  P%d\t%4d\t%3d\t%2d",
               p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
   printf("\n");
   for(i=0; i<28; i++)
      printf("-");

   printf("\n\nAverage waiting time     : %5.2fms", awat);
   printf("\nAverage turn around time : %5.2fms\n", atur);

   printf("\n\nGANTT Chart\n");
   printf("-");
   for(i=0; i<(p[n-1].ttime + 2*n); i++)
      printf("-");
   printf("\n");
   printf("|");
   for(i=0; i<n; i++)
   {
      k = p[i].btime/2;
      for(j=0; j<k; j++)
         printf(" ");
      printf("P%d",p[i].pid);
      for(j=k+1; j<p[i].btime; j++)
         printf(" ");
      printf("|");
   }
   printf("\n");
   printf("-");
   for(i=0; i<(p[n-1].ttime + 2*n); i++)
      printf("-");
   printf("\n");
   printf("0");
   for(i=0; i<n; i++)
   {
      for(j=0; j<p[i].btime; j++)
         printf(" ");
      printf("%2d",p[i].ttime);
   }
}
```

Output

```
$ gcc fcfs.c

$ ./a.out
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) : 6

       FCFS Scheduling

----------------------------
Process B-Time T-Time W-Time
----------------------------
  P1      10      10      0
  P2       4      14     10
  P3      11      25     14
  P4       6      31     25
----------------------------

Average waiting time      : 12.25ms
Average turn around time : 20.00ms


GANTT Chart
----------------------------------------
|      P1     |  P2 |      P3      |   P4  |
----------------------------------------
0          10     14            25      31
```

Result

Thus waiting time & turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

Ex. No. 4d **Polynomial Addition**
Date:

Aim

To add any two given polynomial using linked lists.

Algorithm

1. **Create a structure for polynomial with exp and coeff terms.**
2. **Read the coefficient and exponent of given two polynomials p and q.**
3. **While p and q are not null, repeat step 4.**
   - **If powers of the two terms are equal then**
     - **Insert the sum of the terms into the sum Polynomial**
     - **Advance p and q**
   - **Else if the power of the first polynomial> power of second then**
     - **Insert the term from first polynomial into sum polynomial**
     - **Advance p**
   - **Else**
     - **Insert the term from second polynomial into sum polynomial**
     - **Advance q**
4. **Copy the remaining terms from the non empty polynomial into the sum polynomial**
5. **Stop**

Program

```
/* Polynomial Addition */

/* Add two polynomials */

#include <stdio.h>
#include <malloc.h>
#include <conio.h>

struct link
{
     int coeff;
     int pow;
     struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;

void create(struct link *node)
{
     char ch;
     do
     {
          printf("\nEnter coefficient: ");
          scanf("%d", &node->coeff);
          printf("Enter exponent: ");
          scanf("%d", &node->pow);
          node->next = (struct link*)malloc(sizeof(struct
                                                       link));
          node = node->next;
          node->next = NULL;
          printf("\n continue(y/n): ");
          fflush(stdin);
          ch=getch();
     } while(ch=='y' || ch=='Y');
}

void show(struct link *node)
{
     while(node->next!=NULL)
    {
          printf("%dx^%d", node->coeff, node->pow);
          node=node->next;
          if(node->next!=NULL)
               printf(" + ");
     }
}

void polyadd(struct link *poly1, struct link *poly2, struct
                                                  link *poly)
{
```

```
     while(poly1->next &&  poly2->next)
     {
          if(poly1->pow > poly2->pow)
        {
               poly->pow = poly1->pow;
           poly->coeff = poly1->coeff;
           poly1 = poly1->next;
          }
        else if(poly1->pow < poly2->pow)
          {
               poly->pow = poly2->pow;
               poly->coeff = poly2->coeff;
           poly2 = poly2->next;
          }
        else
        {
               poly->pow = poly1->pow;
               poly->coeff = poly1->coeff + poly2->coeff;
               poly1 = poly1->next;
               poly2 = poly2->next;
          }
        poly->next=(struct link *)malloc(sizeof(struct link));
        poly=poly->next;
        poly->next=NULL;
     }
     while(poly1->next || poly2->next)
     {
          if(poly1->next)
     {
               poly->pow = poly1->pow;
               poly->coeff = poly1->coeff;
               poly1 = poly1->next;
          }
        if(poly2->next)
          {
               poly->pow = poly2->pow;
               poly->coeff = poly2->coeff;
               poly2 = poly2->next;
          }
        poly->next = (struct link *)malloc(sizeof(struct
                                                      link));
          poly = poly->next;
          poly->next = NULL;
     }
}

main()
{
     poly1 = (struct link *)malloc(sizeof(struct link));
     poly2 = (struct link *)malloc(sizeof(struct link));
     poly = (struct link *)malloc(sizeof(struct link));
```

```
     printf("Enter 1st Polynomial:");
     create(poly1);
     printf("\nEnter 2nd Polynomial:");
     create(poly2);
     printf("\nPoly1: ");
     show(poly1);
     printf("\nPoly2: ");
     show(poly2);
     polyadd(poly1, poly2, poly);
     printf("\nAdded Polynomial: ");
    show(poly);
}
```

Output

```
Enter 1st Polynomial:
Enter coefficient: 5
Enter exponent: 2
 continue(y/n): y
Enter coefficient: 4
Enter exponent: 1
 continue(y/n): y
Enter coefficient: 2
Enter exponent: 0
 continue(y/n): n

Enter 2nd Polynomial:
Enter coefficient: 5
Enter exponent: 1
 continue(y/n): y
Enter coefficient: 5
Enter exponent: 0
 continue(y/n): n

Poly1: 5x^2 + 4x^1 + 2x^0
Poly2: 5x^1 + 5x^0
Added Polynomial: 5x^2 + 9x^1 + 7x^0
```

Result

Thus the two given polynomials were added using lists.

Ex. No. 5 **Binary Tree Traversal**
Date:

Aim

To implement different types of traversal for the given binary tree.

Algorithm

1. **Create a structure with key and 2 pointer variable left and right**
2. **Read the node to be inserted.**
   If (root==NULL)
   root=node
   else if (root->key < node->key)
   root->right=NULL
   else
   Root->left=node
3. **For Inorder Traversal**
   Traverse Left subtree
   Visit root
   Traverse Right subtree
4. **For Preorder Traversal**
   Visit root
   Traverse Left subtree
   Traverse Right subtree
5. **For Postorder Traversal**
   Traverse Left subtree
   Traverse Right subtree
   Visit root
6. **Stop.**

Program

```
/* Tree Traversal */

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

typedef struct node
{
   int data;
   struct node *left;
   struct node *right;
}node;

int count=1;

node *insert(node *tree,int digit)
{
   if(tree == NULL)
   {
      tree = (node *)malloc(sizeof(node));
      tree->left = tree->right=NULL;
      tree->data = digit;
      count++;
   }
   else if(count%2 == 0)
      tree->left = insert(tree->left, digit);
   else
      tree->right = insert(tree->right, digit);
   return tree;
}

void preorder(node *t)
{
   if(t != NULL)
   {
      printf(" %d", t->data);
      preorder(t->left);
      preorder(t->right);
   }
}

void postorder(node *t)
{
   if(t != NULL)
   {
      postorder(t->left);
      postorder(t->right);
      printf(" %d", t->data);
   }
}
```

```
void inorder(node *t)
{
   if(t != NULL)
   {
      inorder(t->left);
      printf(" %d", t->data);
      inorder(t->right);
   }
}

main()
{
   node *root = NULL;
   int digit;
   puts("Enter integer:To quit enter 0");
   scanf("%d", &digit);
   while(digit != 0)
   {
      root=insert(root,digit);
      scanf("%d",&digit);
   }
   printf("\nThe preorder traversal of tree is:\n");
   preorder(root);
   printf("\nThe inorder traversal of tree is:\n");
   inorder(root);
   printf("\nThe postorder traversal of tree is:\n");
   postorder(root);
   getch();
}
```

Output

```
Enter integer:To quit enter 0
12 4 6 9 14 17 3 19 0

The preorder traversal of tree is:
 12 4 9 17 19 6 14 3
The inorder traversal of tree is:
 19 17 9 4 12 6 14 3
The postorder traversal of tree is:
 19 17 9 4 3 14 6 12
```

Result

Thus three types of tree traversal was performed on the given binary tree.

Ex. No. 6 **Binary Search Tree**
Date:

Aim

To insert and delete nodes in a binary search tree.

Algorithm

1. **Create a structure with key and 2 pointer variable left and right.**
2. **Read the node to be inserted.**
   If (root==NULL)
   root=node
   else if (root->key<node->key)
   root->right=NULL
   else
   Root->left=node
3. **For Deletion**
   if it is a leaf node
   Remove immediately
   Remove pointer between del node & child
   if it is having one child
   Remove link between del node&child
   Link delnode is child with delnodes parent
   If it is a node with a children
   Find min value in right subtree
   Copy min value to delnode place
   Delete the duplicate
4. **Stop**

Program

```
/* Binary Search Tree */

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

struct node
{
   int key;
   struct node *left;
   struct node *right;
};

struct node *newNode(int item)
{
   struct node *temp = (struct node *)malloc(sizeof(struct
                                                     node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}

void inorder(struct node *root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d ", root->key);
      inorder(root->right);
   }
}

struct node* insert(struct node* node, int key)
{
   if (node == NULL)
      return newNode(key);
   if (key < node->key)
      node->left  = insert(node->left, key);
    else
      node->right = insert(node->right, key);
   return node;
}

struct node * minValueNode(struct node* node)
{
   struct node* current = node;
   while (current->left != NULL)
      current = current->left;
   return current;
}
```

```
struct node* deleteNode(struct node* root, int key)
{
   struct node *temp;
   if (root == NULL)
      return root;
   if (key < root->key)
      root->left = deleteNode(root->left, key);
   else if (key > root->key)
      root->right = deleteNode(root->right, key);
   else
   {
      if (root->left == NULL)
      {
         temp = root->right;
         free(root);
         return temp;
      }
      else if (root->right == NULL)
      {
         temp = root->left;
         free(root);
         return temp;
      }
      temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
   }
   return root;
}

main()
{
   struct node *root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);
   printf("Inorder traversal of the given tree \n");
   inorder(root);
   printf("\nDelete 20\n");
   root = deleteNode(root, 20);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 30\n");
   root = deleteNode(root, 30);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 50\n");
```

```
   root = deleteNode(root, 50);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
}
```

Output

```
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
```

Result

Thus nodes were inserted and deleted from a binary search tree.

Ex. No. 7 **AVL Trees**
Date:

Aim

To perform insertion operation on an AVL tree and to maintain balance factor.

Algorithm

1. **Start**
2. **Perform standard BST insert for w**
3. **Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.**
4. **Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways.**
   - **a) y is left child of z and x is left child of y (Left Left Case)**
   - **b) y is left child of z and x is right child of y (Left Right Case)**
   - **c) y is right child of z and x is right child of y (Right Right Case)**
   - **d) y is right child of z and x is left child of y (Right Left Case)**
5. **Stop**

Program

```
/* AVL Tree */

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

#define CHANGED 0
#define BALANCED 1

typedef struct bnode
{
   int data,bfactor;
   struct bnode *left;
   struct bnode *right;
}node;

int height;

void displaymenu()
{
   printf("\nBasic Operations in AVL tree");
   printf("\n0.Display menu list");
   printf("\n1.Insert a node in AVL tree");
   printf("\n2.View AVL tree");
   printf("\n3.Exit");
}

node* getnode()
{
   int size;
   node *newnode;
   size = sizeof(node);
   newnode = (node*)malloc(size);
   return(newnode);
}

void copynode(node *r, int data)
{
   r->data = data;
   r->left = NULL;
   r->right = NULL;
   r->bfactor = 0;
}

void releasenode(node *p)
{
   free(p);
}
```

```
node* searchnode(node *root, int data)
{
   if(root!=NULL)
      if(data < root->data)
         root = searchnode(root->left, data);
      else if(data > root->data)
         root = searchnode(root->right, data);
   return(root);
}

void lefttoleft(node **pptr, node **aptr)
{
   node *p = *pptr, *a = *aptr;
   printf("\nLeft to Left AVL rotation");
   p->left = a->right;
   a->right = p;
   if(a->bfactor == 0)
   {
      p->bfactor = 1;
      a->bfactor = -1;
      height = BALANCED;
   }
   else
   {
      p->bfactor = 0;
      a->bfactor = 0;
   }
   p = a;
   *pptr = p;
   *aptr = a;
}

void lefttoright(node **pptr, node **aptr, node **bptr)
{
   node *p = *pptr, *a = *aptr, *b = *bptr;
   printf("\nLeft to Right AVL rotation");
   b = a->right;
   b->right = p;
   if(b->bfactor == 1)
      p->bfactor = -1;
   else
      p->bfactor = 0;
   if(b->bfactor == -1)
      a->bfactor = 1;
   else
      a->bfactor = 1;
   b->bfactor = 0;
   p = b;
   *pptr = p;
   *aptr = a;
   *bptr = b;
```

```
}

void righttoright(node **pptr, node **aptr)
{
   node *p = *pptr, *a = *aptr;
   printf("\nRight to Right AVL rotation");
   p->right = a->left;
   a->left = p;
   if(a->bfactor == 0)
   {
      p->bfactor = -1;
      a->bfactor = 1;
      height = BALANCED;
   }
   else
   {
      p->bfactor = 0;
      a->bfactor = 0;
   }
   p = a;
   *pptr = p;
   *aptr = a;
}

void righttoleft(node **pptr, node **aptr, node **bptr)
{
   node *p = *pptr, *a = *aptr, *b = *bptr;
   printf("\nRight to Left AVL rotation");
   b = a->left;
   a->left = b->right;
   b->right = a;
   p->right = b->left;
   b->left = p;
   if(b->bfactor == -1)
      p->bfactor = 1;
   else
      p->bfactor = 0;
   if(b->bfactor == -1)
      a->bfactor = 0;
   b->bfactor = 0;
   p = b;
   *pptr = p;
   *aptr = a;
   *bptr = b;
}

void inorder(node *root)
{
   if(root == NULL)
      return;
   inorder(root->left);
```

```
   printf("\n%4d", root->data);
   inorder(root->right);
}

void view(node *root, int level)
{
   int k;
   if(root == NULL)
      return;
   view(root->right, level+1);
   printf("\n");
   for(k=0; k<level; k++)
      printf("  ");
   printf("%d", root->data);
   view(root->left, level+1);
}

node* insertnode(int data, node *p)
{
   node *a,*b;
   if(p == NULL)
   {
      p=getnode();
      copynode(p, data);
      height = CHANGED;
      return(p);
  }
  if(data < p->data)
  {
      p->left = insertnode(data, p->left);
      if(height == CHANGED)
      {
         switch(p->bfactor)
         {
            case -1:
               p->bfactor = 0;
               height = BALANCED;
               break;
            case 0:
               p->bfactor = 1;
               break;
            case 1:
               a = p->left;
               if(a->bfactor == 1)
                    lefttoleft(&p, &a);
               else
                    lefttoright(&p, &a, &b);
               height = BALANCED;
               break;
         }
      }
```

```
    }
    if(data > p->data)
    {
       p->right = insertnode(data, p->right);
       if(height == CHANGED)
       {
          switch(p->bfactor)
          {
             case 1:
                p->bfactor = 0;
                height = BALANCED;
                break;
             case 0:
                p->bfactor = -1;
                break;
             case -1:
                a=p->right;
                if(a->bfactor == -1)
                     righttoright(&p, &a);
                else
                     righttoleft(&p, &a, &b);
                height=BALANCED;
                break;
          }
       }
    }
    return(p);
}

main()
{
    int data, ch;
    char choice = 'y';
    node *root = NULL;
    clrscr();
    displaymenu();
    while((choice == 'y') || (choice == 'Y'))
    {
       printf("\nEnter your choice: ");
       fflush(stdin);
       scanf("%d",&ch);
       switch(ch)
       {
          case 0:
             displaymenu();
             break;
          case 1:
             printf("Enter the value to be inserted ");
             scanf("%d", &data);
             if(searchnode(root, data) == NULL)
                root = insertnode(data, root);
```

```
            else
               printf("\nData already exists");
               break;
         case 2:
            if(root == NULL)
            {
               printf("\nAVL tree is empty");
               continue;
            }
            printf("\nInorder traversal of AVL tree");
            inorder(root);
            printf("\nAVL tree is");
            view(root, 1);
            break;
         case 3:
            releasenode(root);
            exit(0);
      }
   }
   getch();
}
```

Output

```
Basic Operations in AVL tree
0.Display menu list
1.Insert a node in AVL tree
2.View AVL tree
3.Exit
Enter your choice: 1
Enter the value to be inserted 1

Enter your choice: 1
Enter the value to be inserted 2

Enter your choice: 1
Enter the value to be inserted 3

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 4

Enter your choice: 1
Enter the value to be inserted 5

Right to Right AVL rotation
```

```
Enter your choice: 1
Enter the value to be inserted 6

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 7

Right to Right AVL rotation

Enter your choice: 1
Enter the value to be inserted 8

Enter your choice: 2
Inorder traversal of AVL tree
   1
   2
   3
   4
   5
   6
   7
   8
AVL tree is
        8
      7
    6
      5
  4
      3
    2
      1
Enter your choice: 3
```

Result

Thus rotations were performed as a result of insertions to AVL Tree.

Ex. No. 8 **Binary Heap**
Date:

Aim

To build a binary heap from an array of input elements.

Algorithm

1. **Start**
2. **In a heap, for every node *x* with parent *p*, the key in *p* is smaller than or equal to the key in *x*.**
3. **For insertion operation**
    a. **Add the element to the bottom level of the heap.**
    b. **Compare the added element with its parent; if they are in the correct order, stop.**
    c. **If not, swap the element with its parent and return to the previous step.**
4. **For deleteMin operation**
    a. **Replace the root of the heap with the last element on the last level.**
    b. **Compare the new root with its children; if they are in the correct order, stop.**
    c. **If not, Swap with its smaller child in a min-heap**
5. **Stop**

Program

```
/* Binary Heap */

#include <stdio.h>
#include <limits.h>

int heap[1000000], heapSize;

void Init()
{
   heapSize = 0;
   heap[0] = -INT_MAX;
}

void Insert(int element)
{
   heapSize++;
   heap[heapSize] = element;
   int now = heapSize;
   while (heap[now / 2] > element)
   {
      heap[now] = heap[now / 2];
      now /= 2;
   }
   heap[now] = element;
}

int DeleteMin()
{
   int minElement, lastElement, child, now;
   minElement = heap[1];
   lastElement = heap[heapSize--];
   for (now = 1; now * 2 <= heapSize; now = child)
   {
      child = now * 2;
      if (child != heapSize && heap[child + 1] < heap[child])
         child++;
      if (lastElement > heap[child])
         heap[now] = heap[child];
      else
         break;
   }
   heap[now] = lastElement;
   return minElement;
}

main()
{
   int number_of_elements;
   printf("Program to demonstrate Heap:\nEnter the number of
```

```
                                        elements: ");
   scanf("%d", &number_of_elements);
   int iter, element;
   Init();
   printf("Enter the elements: ");
   for (iter = 0; iter < number_of_elements; iter++)
   {
      scanf("%d", &element);
      Insert(element);
   }
   for (iter = 0; iter < number_of_elements; iter++)
      printf("%d ", DeleteMin());
   printf("\n");
}
```

Output

```
Program to demonstrate Heap:
Enter the number of elements: 6
Enter the elements: 3 2 15 5 4 45

2 3 4 5 15 45
```

Result

Thus a binary heap is constructed for the given elements.

Ex. No. 9a **Breadth First Search**
Date:

Aim

To create adjacency matrix of the given graph and to perform breadth first search traversal.

Algorithm

1. **Start**
2. **Obtain Adjacency matrix for the given graph**
3. **Define a Queue of size total number of vertices in the graph.**
4. **Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.**
5. **Visit all the adjacent vertices of the verex which is at front of the Queue which is not visited and insert them into the Queue.**
6. **When there is no new vertex to be visit from the vertex at front of the Queue then delete that vertex from the Queue.**
7. **Repeat step 5 and 6 until queue becomes empty.**
8. **When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph.**
9. **Stop**

Program

```
/* Graph Traversal – BFS */

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

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}

void BF_Traversal()
{
    int v;
    for(v=0; v<n; v++)
        state[v] = initial;
    printf("Enter Start Vertex for BFS: ");
    scanf("%d", &v);
    BFS(v);
}

void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;
    printf("BFS Traversal : ");
    while(!isEmpty_queue())
    {
        v = delete_queue( );
```

```
        printf("%d ", v);
        state[v] = visited;
        for(i=0; i<n; i++)
        {
            if(adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}

void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}

int isEmpty_queue()
{
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}

int delete_queue()
{
    int delete_item;
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }
    delete_item = queue[front];
    front = front+1;
    return delete_item;
}

void create_graph()
{
    int count,max_edge,origin,destin;
```

```
    printf("Enter number of vertices : ");
    scanf("%d", &n);
    max_edge = n * (n-1);
    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d", &origin, &destin);
        if((origin == -1) && (destin == -1))
            break;
        if(origin>=n || destin>=n || origin<0 || destin<0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
            adj[origin][destin] = 1;
    }
}
```

Output

```
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0 1
Enter edge 2( -1 -1 to quit ) : 0 3
Enter edge 3( -1 -1 to quit ) : 0 4
Enter edge 4( -1 -1 to quit ) : 1 2
Enter edge 5( -1 -1 to quit ) : 1 4
Enter edge 6( -1 -1 to quit ) : 2 5
Enter edge 7( -1 -1 to quit ) : 3 4
Enter edge 8( -1 -1 to quit ) : 3 6
Enter edge 9( -1 -1 to quit ) : 4 5
Enter edge 10( -1 -1 to quit ) : 4 7
Enter edge 11( -1 -1 to quit ) : 6 4
Enter edge 12( -1 -1 to quit ) : 6 7
Enter edge 13( -1 -1 to quit ) : 7 8
Enter edge 14( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS: 0
BFS Traversal is : 0 1 3 4 2 6 5 7 8
```

Result

Thus Breadth First Traversal is executed on the given graph.

Ex. No. 9b **Depth First Search**
Date:

Aim

To create adjacency matrix of the given graph and to perform depth first search traversal.

Algorithm

1. **Start**
2. **Obtain Adjacency matrix for the given graph**
3. **Define a Stack of size total number of vertices in the graph.**
4. **Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.**
5. **Visit any one of the adjacent vertex of the verex which is at top of the stack which is not visited and push it on to the stack.**
6. **Repeat step 5 until there are no new vertex to be visit from the vertex on top of the stack.**
7. **When there is no new vertex to be visit then use back tracking and pop one vertex from the stack.**
8. **Repeat steps 5, 6 and 7 until stack becomes Empty.**
9. **When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph.**
10. **Stop**

Program

```
/* DFS on undirected graph */

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

#define true 1
#define false 0
#define MAX 5

struct Vertex
{
   char label;
   int visited;
};

int stack[MAX];
int top = -1;

struct Vertex* lstVertices[MAX];
static int adjMatrix[MAX][MAX];
int vertexCount = 0;

void push(int item)
{
   stack[++top] = item;
}

int pop()
{
   return stack[top--];
}

int peek()
{
   return stack[top];
}

int isStackEmpty()
{
   return top == -1;
}

void addVertex(char label)
{
   struct Vertex* vertex = (struct Vertex*)
                              malloc(sizeof(struct Vertex));
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
```

```
}

void addEdge(int start, int end)
{
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}

void displayVertex(int vertexIndex)
{
   printf("%c ", lstVertices[vertexIndex]->label);
}

int getAdjUnvisitedVertex(int vertexIndex)
{
   int i;
   for(i = 0; i < vertexCount; i++)
   {
      if(adjMatrix[vertexIndex][i] == 1 &&
                         lstVertices[i]->visited == false)
         return i;
   }
   return -1;
}

void depthFirstSearch()
{
   int i;
   lstVertices[0]->visited = true;
   displayVertex(0);
   push(0);
   while(!isStackEmpty())
   {
      int unvisitedVertex = getAdjUnvisitedVertex(peek());
      if(unvisitedVertex == -1)
         pop();
      else
     {
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         push(unvisitedVertex);
      }
   }
   for(i = 0;i < vertexCount;i++)
      lstVertices[i]->visited = false;
}

main()
{
   int i, j, n, edges, orgn, destn;
   char ch;
```

```
   printf("Enter no. of vertices : ");
   scanf("%d", &n);
   edges = n * (n - 1);

   printf("Enter Vertex Labels : \n");
   for (i=0; i<n; i++)
   {
      fflush(stdin);
      scanf("%c", &ch);
      addVertex(ch);
   }
    for(i=0; i<edges; i++)
    {
        printf("Enter edge ( -1 -1 to quit ) : ");
        scanf("%d %d", &orgn, &destn);
        if((orgn == -1) && (destn == -1))
            break;
        if(orgn>=n || destn>=n || orgn<0 || destn<0)
            printf("Invalid edge!\n");
        else
         addEdge(orgn, destn);
    }
   printf("\nDepth First Search: ");
   depthFirstSearch();
}
```

Output

```
Enter no. of vertices : 5
Enter Vertex Labels :
S
A
B
C
D
Enter edge ( -1 -1 to quit ) : 0 1
Enter edge ( -1 -1 to quit ) : 0 3
Enter edge ( -1 -1 to quit ) : 0 2
Enter edge ( -1 -1 to quit ) : 1 4
Enter edge ( -1 -1 to quit ) : 2 4
Enter edge ( -1 -1 to quit ) : 3 4
Enter edge ( -1 -1 to quit ) : -1 -1

Depth First Search: S A D B C
```

Result

Thus depth first traversal is executed on the given undirected graph.

Ex. No. 10 **Dijkstra's Shortest Path**
Date:

Aim

To find the shortest path for the given graph from a specified source to all other vertices using Dijkstra's algorithm.

Algorithm

1. **Start**
2. **Obtain no. of vertices and adjacency matrix for the given graph**
3. **Create cost matrix from adjacency matrix. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity**
4. **Initialize visited[] to zero**
5. **Read source vertex and mark it as visited**
6. **Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex**
   distance[i]=cost[0][i];
7. **Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.**
8. **Recalculate the shortest distance of remaining vertices from the source.**
9. **Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v**
   if(visited[v]==0)
   distance[v]=min(distance[v]
   distance[w]+cost[w][v])
10. **Stop**

Program

```
/* Dijkstra's Shortest Path */

#include <stdio.h>
#include <conio.h>

#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);

main()
{
    int G[MAX][MAX], i, j, n, u;
    printf("Enter no. of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%d", &G[i][j]);
    printf("Enter the starting node: ");
    scanf("%d", &u);
    dijkstra(G, n, u);
}

void dijkstra(int G[MAX][MAX], int n,int startnode)
{
    int cost[MAX][MAX], distance[MAX], pred[MAX];
    int visited[MAX],count, mindistance, nextnode, i, j;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            if(G[i][j] == 0)
                cost[i][j] = INFINITY;
            else
                cost[i][j] = G[i][j];
    for(i=0; i<n; i++)
    {
        distance[i] = cost[startnode][i];
        pred[i] = startnode;
        visited[i] = 0;
    }
    distance[startnode] = 0;
    visited[startnode] = 1;
    count = 1;
    while(count < n-1)
    {
        mindistance = INFINITY;
        for(i=0; i<n; i++)
            if(distance[i] < mindistance && !visited[i])
            {
```

```
                mindistance = distance[i];
                nextnode=i;
            }
            visited[nextnode] = 1;
            for(i=0; i<n; i++)
                if(!visited[i])
                    if(mindistance + cost[nextnode][i] <
                                                   distance[i])
                    {
                        distance[i] = mindistance +
                                             cost[nextnode][i];
                        pred[i] = nextnode;
                    }
        count++;
    }
    for(i=0; i<n; i++)
        if(i != startnode)
        {
            printf("\nDistance to node%d = %d", i,
                                             distance[i]);
            printf("\nPath = %d", i);
            j = i;
            do
            {
                j = pred[j];
                printf("<-%d", j);
            } while(j != startnode);
    }
}
```

Output

```
Enter no. of vertices: 5
Enter the adjacency matrix:
0   10  0  30  100
10  0   50 0   0
0   50  0  20  10
30  0   20 0   60
100 0   0  60  0
Enter the starting node: 0
Distance to node1 = 10
Path = 1<-0
Distance to node2 = 50
Path = 2<-3<-0
Distance to node3 = 30
Path = 3<-0
Distance to node4 = 60
Path = 4<-2<-3<-0
```

Result

Thus Dijkstra's algorithm is used to find shortest path from a given vertex.

Ex. No. 11a **Linear Search**
Date:

Aim

To perform linear search of an element on the given array.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i, $i = 0,1,2,\ldots n-1$**
4. **Read *search* value**
5. **Assign 0 to *found***
6. **Check each array element against *search***
   If A_i = *search* then
      ***found* = 1**
      Print "Element found"
      Print position *i*
      Stop
7. **If *found* = 0 then**
   print "Element not found"
8. **Stop**

Program

```
/* Linear search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, val, found;
   clrscr();

   printf("Enter number of elements : ");
   scanf("%d", &n);
   printf("Enter Array Elements : \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   printf("Enter element to locate : ");
   scanf("%d", &val);
   found = 0;
   for(i=0; i<n; i++)
   {
      if (a[i] == val)
      {
         printf("Element found at position %d", i);
         found = 1;
         break;
      }
   }
   if (found == 0)
      printf("\n Element not found");
   getch();
}
```

Output

```
Enter number of elements : 7
Enter Array Elements :
23  6  12  5  0  32  10
Enter element to locate : 5
Element found at position 3
```

Result

Thus an array was linearly searched for an element's existence.

Ex. No. 11b **Binary Search**
Date:

Aim

To locate an element in a sorted array using Binary search method

Algorithm

1. **Start**
2. **Read number of array elements, say *n***
3. **Create an array *arr* consisting *n* sorted elements**
4. **Get element, say *key* to be located**
5. **Assign 0 to *lower* and *n* to *upper***
6. **While (*lower* < *upper*)**

   Determine middle element *mid* = (upper+lower)/2
   If key = arr[mid] then
   Print mid
   Stop
   Else if key > arr[mid] then
   lower = mid + 1
   else
   upper = mid – 1

7. **Print "Element not found"**
8. **Stop**

Program

```
/* Binary Search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, upper, lower, mid, val, found;
   clrscr();

   printf("Enter array size  : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
      a[i] = 2 * i;

   printf("\n Elements in Sorted Order \n");
   for(i=0; i<n; i++)
      printf("%4d", a[i]);

   printf("\n Enter element to locate : ");
   scanf("%d", &val);
   upper = n;
   lower = 0;
   found = -1;
   while (lower <= upper)
   {
      mid = (upper + lower)/2;
      if (a[mid] == val)
      {
         printf("Located at position %d", mid);
         found = 1;
         break;
      }
      else if(a[mid] > val)
         upper = mid - 1;
      else
         lower = mid + 1;
   }

   if (found == -1)
      printf("Element not found");
   getch();
}
```

Output

```
Enter array size  : 9
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16
 Enter element to locate : 12
 Located at position 6



Enter array size  : 10
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16  18
 Enter element to locate : 13
 Element not found
```

Result

Thus an element is located quickly using binary search method.

Ex. No. 11c **Bubble Sort**
Date:

Aim

To sort an array of N numbers using Bubble sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Index *i* varies from 0 to *n*-2**
5. **Index *j* varies from *i*+1 to *n*-1**
6. **Traverse the array and compare each pair of elements**
   If $A_i > A_j$ then
   Swap A_i and A
7. **Stop**

Program

```
/* Bubble Sort */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, j, n, t;
   clrscr();
   printf("Enter number of elements : ");
   scanf("%d", &n);
   printf("Enter Array Elements \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if (a[i] > a[j])
         {
            t = a[i];
            a[i] = a[j];
            a[j] = t;
         }
      }
   }

   printf("\n Elements in Sorted order :");
   for(i=0; i<n; i++)
      printf("%d  ", a[i]);
   getch();
}
```

Output

```
Enter number of elements : 5
Enter Array Elements
3 7 -9 0 2

Elements in Sorted order :
-9  0  2  3  7
```

Result

Thus an array was sorted using bubble sort.

Ex. No. 11d **Quick Sort**
Date:

Aim

To sort an array of N numbers using Quick sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Select an pivot element x from A_i**
5. **Divide the array into 3 sequences: elements < x, x, elements > x**
6. **Recursively quick sort both sets ($A_i < x$ and $Ai > x$)**
7. **Stop**

Program

```
/* Quick Sort */

#include <stdio.h>
#include <conio.h>

void qsort(int arr[20], int fst, int last);

main()
{
   int arr[30];
   int i, size;
   printf("Enter total no. of the elements : ");
   scanf("%d", &size);
   printf("Enter total %d elements : \n", size);
   for(i=0; i<size; i++)
      scanf("%d", &arr[i]);
   qsort(arr,0,size-1);
   printf("\n Quick sorted elements \n");
   for(i=0; i<size; i++)
      printf("%d\t", arr[i]);
   getch();
}

void qsort(int arr[20], int fst, int last)
{
   int i, j, pivot, tmp;
   if(fst < last)
   {
      pivot = fst;
      i = fst;
      j = last;
      while(i < j)
      {
         while(arr[i] <=arr[pivot] && i<last)
         i++;
         while(arr[j] > arr[pivot])
         j--;
         if(i <j )
         {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
         }
      }
      tmp = arr[pivot];
      arr[pivot] = arr[j];
      arr[j] = tmp;
      qsort(arr, fst, j-1);
      qsort(arr, j+1, last);
```

```
   }
}
```

Output

```
Enter total no. of the elements : 8
Enter total 8 elements :
1
2
7
-1
0
4
-2
3

Quick sorted elements
-2      -1      0       1       2       3       4       7
```

Result

Thus an array was sorted using quick sort's divide and conquer method.

Ex. No. 11e **Merge Sort**
Date:

Aim

To sort an array of N numbers using Merge sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Divide the array into sub-arrays with a set of elements**
5. **Recursively sort the sub-arrays**
6. **Merge the sorted sub-arrays onto a single sorted array.**
7. **Stop**

Program

```
/* Merge sort */

#include <stdio.h>
#include <conio.h>

void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;

main()
{
   int i, arr[30];
   printf("Enter total no. of elements : ");
   scanf("%d", &size);
   printf("Enter array elements : ");
   for(i=0; i<size; i++)
      scanf("%d", &arr[i]);
   part(arr, 0, size-1);
   printf("\n Merge sorted list : ");
   for(i=0; i<size; i++)
      printf("%d ",arr[i]);
   getch();
}

void part(int arr[], int min, int max)
{
   int i, mid;
   if(min < max)
   {
      mid = (min + max) / 2;
      part(arr, min, mid);
      part(arr, mid+1, max);
      merge(arr, min, mid, max);
   }
   if (max-min == (size/2)-1)
   {
      printf("\n Half sorted list : ");
      for(i=min; i<=max; i++)
         printf("%d ", arr[i]);
   }
}

void merge(int arr[],int min,int mid,int max)
{
   int tmp[30];
   int i, j, k, m;
   j = min;
   m = mid + 1;
```

```
   for(i=min; j<=mid && m<=max; i++)
   {
      if(arr[j] <= arr[m])
      {
         tmp[i] = arr[j];
         j++;
      }
      else
      {
         tmp[i] = arr[m];
         m++;
      }
   }
   if(j > mid)
   {
      for(k=m; k<=max; k++)
      {
         tmp[i] = arr[k];
         i++;
      }
   }
   else
   {
      for(k=j; k<=mid; k++)
      {
         tmp[i] = arr[k];
         i++;
      }
   }
   for(k=min; k<=max; k++)
      arr[k] = tmp[k];
}
```

Output

```
Enter total no. of elements : 8
Enter array elements : 24 13 26 1 2 27 38 15

 Half sorted list : 1 13 24 26
 Half sorted list : 2 15 27 38
 Merge sorted list : 1 2 13 15 24 26 27 38
```

Result

Thus array elements was sorted using merge sort's divide and conquer method.

Ex. No. 11f **Insertion Sort**
Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. **Start**
2. **Read number of array elements** ***n***
3. **Read array elements** A_i
4. **Sort the elements using insertion sort**
   In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
   Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.
5. **Stop**

Program

```
/* Insertion Sort */

main()
{
   int i, j, k, n, temp, a[20], p=0;

   printf("Enter total elements: ");
   scanf("%d",&n);
   printf("Enter array elements: ");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   for(i=1; i<n; i++)
   {
      temp = a[i];
      j = i - 1;
      while((temp<a[j]) && (j>=0))
      {
         a[j+1] = a[j];
         j = j - 1;
      }
      a[j+1] = temp;

      p++;
      printf("\n After Pass %d: ", p);
      for(k=0; k<n; k++)
         printf("  %d", a[k]);
   }

   printf("\n Sorted List : ");
   for(i=0; i<n; i++)
      printf(" %d", a[i]);
}
```

Output

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
 After Pass 1:   8  34  64  51  32  21
 After Pass 2:   8  34  64  51  32  21
 After Pass 3:   8  34  51  64  32  21
 After Pass 4:   8  32  34  51  64  21
 After Pass 5:   8  21  32  34  51  64
 Sorted List :  8 21 32 34 51 64
```

Result

Thus array elements was sorted using insertion sort.

Ex. No. 12 **Open Addressing Hashing Technique**
Date:

Aim

To implement hash table using a C program.

Algorithm

1. **Create a structure, data (hash table item) with key and value as data.**
2. **Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).**
3. **A menu is displayed on the screen.**
4. **User must choose one option from four choices given in the menu**
5. **Perform all the operations**
6. **Stop**

Program

```
/* Open hashing */

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

#define MAX 10

main()
{
   int a[MAX], num, key, i;
   char ans;
   int create(int);
   void linearprobing(int[], int, int);
   void display(int[]);

   printf("\nCollision handling by linear probing\n\n");
   for(i=0; i<MAX; i++)
      a[i] = -1;
   do
   {
      printf("\n Enter number:");
      scanf("%d", &num);
      key = create(num);
      linearprobing(a, key, num);
      printf("\nwish to continue?(y/n):");
      ans = getch();
   } while( ans == 'y');
   display(a);
}

int create(int num)
{
   int key;
   key = num % 10;
   return key;
}

void linearprobing(int a[MAX], int key, int num)
{
   int flag, i, count = 0;
   void display(int a[]);
   flag = 0;
   if(a[key] == -1)
      a[key] = num;
   else
   {
      i=0;
```

```
      while(i < MAX)
      {
         if(a[i] != -1)
         count++;
         i++;
      }
      if(count == MAX)
      {
         printf("hash table is full");
         display(a);
         getch();
         exit(1);
      }
      for(i=key+1; i<MAX; i++)
         if(a[i] == -1)
         {
            a[i] = num;
            flag = 1;
            break;
         }
      for(i=0; i<key && flag==0; i++ )
      if(a[i] == -1)
      {
         a[i] = num;
         flag = 1;
         break;
      }
   }
}

void display(int a[MAX])
{
   int i;
   printf("\n Hash table is:");
   for(i=0; i<MAX; i++)
      printf("\n %d\t\t%d",i,a[i]);
}
```

Output

```
Collision handling by linear probing
 Enter number:1
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:62
wish to continue?(y/n):
 Enter number:93
wish to continue?(y/n):
```

```
 Enter number:84
wish to continue?(y/n):
 Enter number:15
wish to continue?(y/n):
  Enter number:76
wish to continue?(y/n):
 Enter number:98
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:199
wish to continue?(y/n):
 Enter number:1234
wish to continue?(y/n):
 Enter number:5678
hash table is full
 Hash table is:
 0              1234
 1              1
 2              62
 3              93
 4              84
 5              15
 6              26
 7              76
 8              98
 9              199
```

Result

Thus hashing has been performed successfully.