NPR
College of Engineering & Technology
Approved by AICTE, Affiliated to Anna University,
Accredited by NAAC WITH 'A' GRADE | Recognized by UGC under 2 (f)
Natham, Dindigul - 624 401. Web: www.nprcet.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3311 DATA STRUCTRES LABORATORY

Semester III

Lab Manual

Regulation 2021

NPR College of Engineering & Technology, Natham.

VISION

- To develop students with intellectual curiosity and technical expertise to meet the global needs.

MISSION

- To achieve academic excellence by offering quality technical education using best teaching techniques.
- To improve Industry – Institute interactions and expose industrial atmosphere.
- To develop interpersonal skills along with value based education in a dynamic learning environment.
- To explore solutions for real time problems in the society.

NPR College of Engineering & Technology, Natham.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

- To produce globally competent technical professionals for digitized society.

MISSION

- To establish conducive academic environment by imparting quality education and value added training.
- To encourage students to develop innovative projects to optimally resolve the challenging social problems.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates of Computer Science and Engineering Program will be able to:

- Develop into the most knowledgeable professional to pursue higher education and Research or have a successful carrier in industries.
- Successfully carry forward domain knowledge in computing and allied areas to solve complex and real world engineering problems.
- Meet the technological revolution they are continuously upgraded with the technical knowledge.
- Serve the humanity with social responsibility combined with ethics.

PROGRAM SPECIFIC OUTCOMES

At the end of the program students will be able to

1. Deal with real time problems by understanding the evolutionary changes in computing, applying standard practices and strategies in software project development using open-ended programming environments.
2. Employ modern computer languages, environments and platforms in creating innovative career paths by inculcating moral values and ethics.
3. Achieve additional expertise through add-on and certificate programs.

Course Objectives

- To demonstrate array implementation of linear data structure algorithms.
- To implement the applications using Stack.
- To implement the applications using Linked list
- To implement Binary search tree and AVL tree algorithms.
- To implement the Heap algorithm.
- To implement Dijkstra’s algorithm.
- To implement Prim’s algorithm
- To implement Sorting, Searching and Hashing algorithms

Course Outcomes

At the end of this course, the students will be able to:

CO1: Implement Linear data structure algorithms.

CO2: Implement applications using Stacks and Linked lists

CO3: Implement Binary Search tree and AVL tree operations.

CO4: Implement graph algorithms.

CO5: Analyze the various searching and sorting algorithms.

Ex .NO: 1   Array implementation of Stack ,Queue and Circular Queue ADT's

<u>Aim:</u>

To write a c program to implement Stack ,Queue and Circular Queue using array.

<u>Algorithm:</u>

<u>Stack</u>:

1. Start
2. Define a array stack of size max=5, top=-1
3. Display a menu listing stack operation
4. Accept choice
5. If choice=1, then top<max-1, increment top value
6. Else print stack overflow
7. If choice=2, then, top<0, print stack underflow
8. Else display current top element
9. Decrement top
10. If choice=3, then, display the elements
11. Stop

<u>Queue:</u>

1. start
2. define a array queue if size max=5, front=-1, rear=-1
3. display a menu listing queue operation
4. if choice=1, then rear<max-1, increment rear, store the value
5. else print queue is full
6. if choice=2, if front=-1, then queue is empty
7. else increment front
8. if choice=3, then display elements from front to rear
9. stop

<u>Circular Queue:</u>

1. start
2. initialize an array of size, max=5, front=rear=-1
3. display a menu listing circular queue operation
4. get the choice from the user. If it is 1, call enqueue function. If it is 2, call dequeue function. If it is 3, call the display function.
5. Stop

PROGRAM:

Array implementation of stack:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 5
int  stack[SIZE],top=-1;
void push();
void pop();
void display();
void main()
{
int choice;
int isempty();
int length();
clrscr();
while(1)
{
printf("\n 1.Push");
printf("\n 2. POP");
printf("\n 3.Display");
printf("\n 4. Length ");
printf("\n 5.Quit");
printf("\n Enter the choice:");
scanf("\n %d",&choice);
switch(choice)
{
case 1: push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: printf("\n No. of elements in the stack is %d",length());
break;
case 5: exit(0);
break;
default: printf("\n Invalid choice");
}
}
}

void push()
{
int n;
if(top==SIZE-1)
```

```
printf("\n Stack is full");
else
{
        printf("\nEnter the no.");
        scanf("%d",&n);
        top++;
        stack[top]=n;
}
}

void pop()
{
int n;
if(isempty())
{
        printf("\nStack is empty");
        top=-1;
}
else
{
n=stack[top];
        printf("\n %d is popped from the stack \n",n);
        --top;
}
}

void display()
{
int i,temp=top;
if(isempty())
{
        printf("\n Stack Empty");
        return;
}
printf("\n Elements in the stack:");
for(i=temp;i>=0;i--)
printf("%d \n",stack[i]);
}

int isempty()
{
return (top==-1);
}
int length()
{
return (top+1);
```

```
}
getch();
```

output:

```
1.Push
2. POP
3.Display
4. Length
5.Quit
Enter the choice:1
Enter the no:10

1.Push
2. POP
3.Display
4. Length
5.Quit
Enter the choice:3
Elements in the stack: 10
```

Array implementation of Queue:

```
# include<stdio.h>
#include<conio.h>
# define MAX 5
int queue_arr[MAX];
int rear = -1,front = -1;
void insert();
void del();
void display();
void main()
{
int choice;
clrscr();
while(1)
{
printf("1.Enqueue\n");
printf("2.Dequeue\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1 :
insert();
break;
case 2 :
del();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Wrong choice\n");
}}}

void insert()
{
int added_item;
if (rear==MAX-1)
printf("Queue Overflow\n");
else
{
if (front==-1)
front=0;
printf("Input the element for adding in queue : ");
scanf("%d", &added_item);
rear=rear+1;
queue_arr[rear] = added_item ;
}
}
 void del()
{
if (front == -1 || front > rear)
{
printf("Queue Underflow\n");
return ;
}
```

```
else
{
printf("Element deleted from queue is : %d\n", queue_arr[front]);
front=front+1;
}
}
void display()
{
int i;
if (front == -1)
printf("Queue is empty\n");
else
{
printf("Queue is :\n");
for(i=front;i<= rear;i++)
printf("%d ",queue_arr[i]);
printf("\n");
}
 }
getch();
```

output:

```
1 .Enqueue
2. Dequeue
3. display
4. quit
Enter the choice: 1
Input the element to be added in queue:10

1 .Enqueue
2. Dequeue
3. display
4. quit
Enter the choice: 2
Queue is:
10
```

Array implementation of Circular Queue:

```
# include<stdio.h>
# include<conio.h>
# define MAX 5
int queue_arr[MAX];
int rear = -1,front = -1;
void enqueue();
void dequeue();
void display();
void main()
{
int choice;
clrscr();
while(1)
{
printf("1.Enqueue\n");
printf("2.Dequeue\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1 :
enqueue();
break;
case 2 :
dequeue();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Wrong choice\n");
}}}

void enqueue()
{
```

```
int added_item;
if (rear==MAX-1)
printf("Queue Overflow\n");
else
{
if (front==-1)
front=0;
printf("Input the element for adding in queue : ");
scanf("%d", &added_item);
rear=rear+1;
queue_arr[rear] = added_item ;
}
}
 void dequeue()
{
if (front == -1 || front > rear)
{
printf("Queue Underflow\n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n", queue_arr[front]);
front=front+1;
}
}
void display()
{
int i;
if (front == -1)
printf("Queue is empty\n");
else
{
printf("Queue is :\n");
for(i=front;i<= rear;i++)
printf("%d ",queue_arr[i]);
printf("\n");
}
 }
getch();
```

<u>Output:</u>

```
1 .Enqueue
2. Dequeue
3. display
4. quit
Enter the choice: 1
Input the element to be added in queue:10

1 .Enqueue
2. Dequeue
3. display
4. quit
Enter the choice: 2
Queue is:
10
```

Result:

Thus the c program to implement Stack ,Queue and Circular Queue using array was executed successfully.

Ex .NO: 2 Implementation of Singly Linked List

Aim:

To write a c program to implement singly Linked List.

Algorithm:

1. Start
2. If PTR=NULL, print overflow
3. Set NEWNODE = PTR
4. Set PTR=PTR->next
5. Set NEWNODE->data=value
6. Set head=NEWNOE
7. stop

PROGRAM:

```
#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>



struct node {

  int value;

  struct node *next;

};



void insert();

void display();

void delete();

int count();

typedef struct node DATA_NODE;

DATA_NODE *head_node, *first_node, *temp_node = 0, *prev_node, next_node;

int data;
int main() {
int option = 0;
printf("Singly Linked List Example - All Operations\n");
while (option < 5) {
  printf("\nOptions\n");
  printf("1 : Insert into Linked List \n");
  printf("2 : Delete from Linked List \n");
  printf("3 : Display Linked List\n");
  printf("4 : Count Linked List\n");
  printf("Others : Exit()\n");
  printf("Enter your option:");
  scanf("%d", &option);
  switch (option) {
    case 1:
      insert();
      break;
    case 2:
      delete();
      break;
    case 3:
      display();
      break;
    case 4:
      count();
      break;
      default:
        break;
    }
  }
  return 0;
}
void insert() {
  printf("\nEnter Element for Insert Linked List : \n");
  scanf("%d", &data);
  temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));
  temp_node->value = data;
  if (first_node == 0) {
    first_node = temp_node;
  } else {
    head_node->next = temp_node;
  }
  temp_node->next = 0;
  head_node = temp_node;
  fflush(stdin);
}
void delete() {
  int countvalue, pos, i = 0;
  countvalue = count();
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
printf("\nEnter Position for Delete Element : \n");
scanf("%d", &pos);
if (pos > 0 && pos <= countvalue) {
  if (pos == 1) {
    temp_node = temp_node -> next;
    first_node = temp_node;
    printf("\nDeleted Successfully \n\n");
  } else {
    while (temp_node != 0) {
      if (i == (pos - 1)) {
        prev_node->next = temp_node->next;
        if(i == (countvalue - 1))
        {
       head_node = prev_node;
       }
        printf("\nDeleted Successfully \n\n");
        break;
      } else {
        i++;
        prev_node = temp_node;
        temp_node = temp_node -> next;
      }
    }
  }
  } else
    printf("\nInvalid Position \n\n");
}
void display() {
  int count = 0;
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  while (temp_node != 0) {
    printf("# %d # ", temp_node->value);
    count++;
    temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
}
int count() {
  int count = 0;
  temp_node = first_node;
  while (temp_node != 0) {
    count++;
    temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
  return count;
}
```

Output:

```
Singly Linked List Example - All Operations


Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:1

Enter Element for Insert Linked List :

20

Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:3

Display Linked List :

# 20 #

No of Items In Linked List : 1
```

Result:

Thus the c program to implement singly linked list was executed successfully

Ex .NO: 3 Linked List implementation of Stack & Linear Queue ADT’s

Aim:

To write a c program to implement Linked List of stack & linear queue.

Algorithm:

Linked List implementation of stack:

1. start
2. define a singly linked list node, create head node.
3. Display a menu listing stack operations
4. If choice=1, then create a node with data
5. Make newnode point to nextnode & head to newnode.
6. If choice=2, then tempnode to first node
7. Make head node point to next of temp node
8. If choice=3, then display the elements
9. Stop

Linked List implementation of Linear Queue:

1. allocate the space for the newnode PTR
2. set PTR->data=val
3. If front=NULL, set front=rear=PTR
4. Set front->next = rear->next=NULL
5. Else, set Rear->next= PTR, rear=PTR
6. Step rear->next=NULL
7. Stop

PROGRAM:

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *ptr;
}*top,*top1,*temp;
int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stackcount();
void create();
int count = 0;
```

```
void main()
{
int  n, ch, e;
printf("\n 1 - Push");
printf("\n 2 - Pop");
printf("\n 3 - Top");
printf("\n 4 - Empty");
printf("\n 5 - Exit");
printf("\n 6 - Dipslay");
printf("\n 7 - Stack Count");
printf("\n 8 - Destroy stack");
create();
while (1)
{
printf("\n Enter choice : ");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("Enter data : ");
scanf("%d", &n);
push(n);
break;
case 2:
pop();
break;
case 3:
if (top == NULL)
printf("No elements in stack");
else
{
e = topelement();
printf("\n Top element : %d", e);
}
break;
case 4:
empty();
break;
case 5:
exit(0);
case 6:
display();
break;
case 7:
stackcount();
break;
case 8:
destroy();
break;
default :
printf(" Wrong choice, Please enter correct choice  ");
break;
}
}
}
void create()
{
top = NULL;
}
void stackcount()
{
printf("\n No. of elements in stack : %d", count);
}
void push(int data)
{
if (top == NULL)
{
top =(struct node *)malloc(1*sizeof(struct node));
top->ptr = NULL;
top->info = data;
}
else
{
temp =(struct node *)malloc(1*sizeof(struct node));
temp->ptr = top;
temp->info = data;
top = temp;
}
count++;
}
void display()
{
top1 = top;
if (top1 == NULL)
{
printf("Stack is empty");
return;
}
while (top1 != NULL)
{
printf("%d ", top1->info);
top1 = top1->ptr;
```

```
}
}
void pop()
{
top1 = top;

if (top1 == NULL)
{
printf("\n Error : Trying to pop from empty stack");
return;
}
else
top1 = top1->ptr;
printf("\n Popped value : %d", top->info);
free(top);
top = top1;
count--;
}
int topelement()
{
return(top->info);
}
void empty()
{
if (top == NULL)
printf("\n Stack is empty");
else
printf("\n Stack is not empty with %d elements", count);
}
void destroy()
{
top1 = top;

while (top1 != NULL)
{
top1 = top->ptr;
free(top);
top = top1;
top1 = top1->ptr;
}
free(top1);
top = NULL;
printf("\n All stack elements destroyed");
count = 0;
}
getch();
```

Output:

```
1 - Push
 2 - Pop
 3 - Top
 4 - Empty
 5 - Exit
 6 - Dipslay
 7 - Stack Count
 8 - Destroy stack
 Enter choice : 1
 Enter data : 25
 Enter choice : 1
 Enter data : 50
 Enter choice : 2
 Popped value : 50
 Enter choice : 6
 25
```

Linked List implementation of Linear Queue:

```
 #include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
    int data;
   struct node *next;
      };
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
void main ()
{
    int choice;
    while(choice != 4)
    {
```

```
printf("\n*************************Main Menu*****************************\n");
        printf("\n======================================\n");
        printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit);
        printf("\nEnter your choice ?");
        scanf("%d",& choice);
        switch(choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nEnter valid choice??\n");
        }
    }
}
void insert()
{
    struct node *ptr;
    int item;
ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr -> data = item;
        if(front == NULL)
        {
            front = ptr;
            rear = ptr;
            front -> next = NULL;
            rear -> next = NULL;
        }
        else
        {
            rear -> next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
    }
}
void delete ()
{
    struct node *ptr;
    if(front == NULL)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        ptr = front;
        front = front -> next;
        free(ptr);
    }
```

```
}
void display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        while(ptr != NULL)
        {
            printf("\n%d\n",ptr -> data);
            ptr = ptr -> next;
        }
    }
}
```

Output:

```
************************Main Menu****************************

=========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice:1
Enter value?
10
************************Main Menu****************************

=========================================

1.insert an element
2.Delete an element
3.Display the queue
4.Exit
Enter your choice:3
printing values .....
10
```

Result:

Thus the c program to implement linked list of stack and linear queue was executed successfully.

Ex .NO: 4 Implementation of Polynomial Manipulation Using LinkedList

Aim:

To write a c program to implement polynomial manipulation using linked list.

Algorithm:

1. Let p and q be two polynomials represented by kinked list.
2. While p and q are not NULL, repeat step 3
3. If powers of the two terms are equal then, if the terms do not cancel then, insert the sum of terms into the sum polynomial adv p, adv q. Else, if the power of first polynomial > power of 2nd polynomial, then insert the term from 1st polynomial into sum polynomial adv p. else insert the term from 2nd polynomial into sum polynomial adv q
4. Copy the remaining terms from the non empty polynomial into the sum polynomial.
5. stop

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

typedef struct link {

   int coeff;

   int pow;

   struct link * next;

} my_poly;

void my_create_poly(my_poly **);

void my_show_poly(my_poly *);

void my_add_poly(my_poly **, my_poly *, my_poly *);

int main(void) {

   int ch;

   do {
        my_poly * poly1, * poly2, * poly3;

        printf("\nCreate 1st expression\n");

        my_create_poly(&poly1);

        printf("\nStored the 1st expression");

        my_show_poly(poly1);

        printf("\nCreate 2nd expression\n");

        my_create_poly(&poly2);

        printf("\nStored the 2nd expression");

        my_show_poly(poly2);

        my_add_poly(&poly3, poly1, poly2);

        my_show_poly(poly3);

        printf("\nAdd two more expressions? (Y = 1/N = 0): ");

        scanf("%d", &ch);

    } while (ch);

    return 0;

}

void my_create_poly(my_poly ** node) {

    int flag;

    int coeff, pow;

    my_poly * tmp_node;

    tmp_node = (my_poly *) malloc(sizeof(my_poly));
    *node = tmp_node;
    do {
        printf("\nEnter Coeff:");
        scanf("%d", &coeff);
        tmp_node->coeff = coeff;
        printf("\nEnter Pow:");
        scanf("%d", &pow);
        tmp_node->pow = pow;
        tmp_node->next = NULL;
        printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");
        scanf("%d", &flag);
        if(flag) {
            tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
            tmp_node = tmp_node->next;
            tmp_node->next = NULL;
        }
    } while (flag);
}
void my_show_poly(my_poly * node) {
    printf("\nThe polynomial expression is:\n");
    while(node != NULL) {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if(node != NULL)
            printf(" + ");
    }
}
void my_add_poly(my_poly ** result, my_poly * poly1, my_poly * poly2) {
    my_poly * tmp_node;
    tmp_node = (my_poly *) malloc(sizeof(my_poly));
    tmp_node->next = NULL;
    *result = tmp_node;
    while(poly1 && poly2) {
        if (poly1->pow > poly2->pow) {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        else if (poly1->pow < poly2->pow) {
            tmp_node->pow = poly2->pow;
            tmp_node->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        else {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff + poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
        if(poly1 && poly2) {
            tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
            tmp_node = tmp_node->next;
            tmp_node->next = NULL;
        }
    }
    while(poly1 || poly2) {
        tmp_node->next = (my_poly *) malloc(sizeof(my_poly));
        tmp_node = tmp_node->next;
        tmp_node->next = NULL;
        if(poly1) {
            tmp_node->pow = poly1->pow;
            tmp_node->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if(poly2) {
            tmp_node->pow = poly2->pow;
            tmp_node->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
    }
    printf("\nAddition Complete");
}
```

Output:

```
Create 1st expression
Enter Coeff:3
Enter Pow:1
Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1
Enter Coeff:2
Enter Pow:0
Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0
Stored the 1st expression
The polynomial expression is:
3x^1 + 2x^0
Create 2nd expression
```

```
Enter Coeff:4

Enter Pow:0

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 2nd expression

The polynomial expression is:

4x^0

Addition Complete

The polynomial expression is:

3x^1 + 6x^0
```

Result:

Thus the c program to implement polynomial manipulation using linked list was executed successfully.

Ex .NO: 5 Implementation of Evaluating Postfix expressions , Infix to Postfix conversion

Aim:

To write a c program to evaluate postfix expressions, infix to postfix conversion.

Algorithm:

Postfix:

1. start
2. define a array stack of size max=20, top=-1
3. read the expression, If it is a character, push into stack
4. else, pop topmost two elements from stack
5. apply operator on elements, push the results onto stack
6. pop the result and print it.
7. Stop

Infix to postfix:

1. Start
2. Define a array stack of size max=20, top=-1
3. Read the expression. If it is a character, print it
4. If it is a operator, compare the operator's priority
5. If stack[top]>/=, then input operator, pop and print it,
6. Else, push the operator onto the stack
7. If '(', push into stack, if ')', pop all the operators, until '(' is encountered.
8. Stop

PROGRAM:

Implementation of evaluating postfix expression:

```
#include<stdio.h>
#include<conio.h>
int stack[20];
int top = -1;

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

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

void main()
{
   char exp[20];
   char *e;
   int n1,n2,n3,num;
   printf("Enter the expression :: ");
   scanf("%s",exp);
   e = exp;
   while(*e != '\0')
   {
      if(isdigit(*e))
      {
         num = *e - 48;
         push(num);
      }
      else
      {
         n1 = pop();
         n2 = pop();
         switch(*e)
         {
         case '+':
         {
            n3 = n1 + n2;
            break;
         }
         case '-':
```

```
         {
            n3 = n2 - n1;
            break;
         }
         case '*':
         {
            n3 = n1 * n2;
            break;
         }
         case '/':
         {
            n3 = n2 / n1;
            break;
         }
         }
         push(n3);
      }
      e++;
   }
   printf("\nThe result of expression %s  =  %d\n\n",exp,pop());
   getch();
}
```

Output:

```
Enter the expression :: 5+6*
The result of expression 5+6*  =  30
```

Implementation of infix to postfix conversion:

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

char stack[100];
int top = -1;

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

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
    return 0;
}

void main()
{
    char exp[100];
    char *e, x;
    clrscr();
    printf("Enter the expression : ");
    scanf("%s",exp);
    printf("\n");
    e = exp;

    while(*e != '\0')
    {
        if(isalnum(*e))
```

```
            printf("%c ",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c ", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c ",pop());
            push(*e);
        }
        e++;
    }

    while(top != -1)
    {
        printf("%c ",pop( ));
    }
getch( );
}
```

<u>Output:</u>

```
Enter the expression : 6+7*9

6 7 9 * +
```

Result:

Thus the c program to evaluate postfix expressions, infix to postfix conversion was executed successfully.

Ex .NO: 6 Implementation of Binary Search Trees

Aim:

To write a c program to implement binary search tree.

Algorithm:

1. Start
2. Call insert to insert an element into binary search tree
3. Call delete to delete an element from binary search tree
4. Call findmax to find the element with maximum value
5. Call findmin to find the element with minimum value
6. Call find to check the presence of the element
7. Call display to display all the elements
8. Call makeempty to delete the entire tree
9. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct BT
{
int data;
struct BT *right,*left;
};
void insert(struct BT **ptr,int d)
{
if((*ptr)==NULL)
{
(*ptr)=(struct BT*)malloc(sizeof(struct BT));
(*ptr)->data=d;
(*ptr)->left=(*ptr)->right=NULL;
}
else
{
if((*ptr)->data>d)
insert(&((*ptr)->left),d);
else
insert(&((*ptr)->right),d);
}
return;
}
int search(struct BT *ptr,int no)
{
if(ptr==NULL)
return(0);
if(ptr->data==no)
return(1);
if(ptr->data>no)
return(search(ptr->left,no));
else
return(search(ptr->right,no));
}
void inorder(struct BT *ptr)
{
if(ptr==NULL)
return;
else
{
inorder(ptr->left);
printf("\t%d",ptr->data);
inorder(ptr->right);
}
}
void preorder(struct BT *ptr)
```

```
{
if(ptr==NULL)
return;
else
{
printf("\t%d",ptr->data);
preorder(ptr->left);
preorder(ptr->right);
}
}
void postorder(struct BT *ptr)
{
if(ptr==NULL)
return;
else
{
postorder(ptr->left);
postorder(ptr->right);
printf("\t%d",ptr->data);
}
}
void main()
{
struct BT *root;
int ch,d,no,f;
clrscr();
root=NULL;
while(ch!=6)
{
printf("\n1.Insert\n2.Search\n3.Inorder\n4.Preorder\n5.Postorder\n6.Exit\n");
printf("\n Enter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("Enter the data:");
scanf("%d",&d);
insert(&root,d);
break;
case 2:
printf("Enter the node:");
scanf("%d",&no);
f=search(root,no);
if(f==0)
printf("Node is not present");
else
printf("Node is present");
break;
case 3:
inorder(root);
break;
case 4:
preorder(root);
break;
case 5:
postorder(root);
break;
case 6:
break;
}
}
getch();}
```

Output:

```
1.Insert
2.Search
3.Inorder
4.Preorder
5.Postorder
6.Exit

 Enter the choice:1
 Enter the data:10
 1.Insert
2.Search
3.Inorder
4.Preorder
5.Postorder
6.Exit

 Enter the choice:1
 Enter the data:20
 1.Insert
2.Search
3.Inorder
4.Preorder
5.Postorder
6.Exit
 Enter the choice:2
 Enter the node:10
 Node is present
```

Result:

Thus the c program to implement binary search tree was executed successfully.

Ex .NO: 7 Implementation of AVL Trees

Aim:

To write a c program to implement AVL tree.

Algorithm:

Insertion

1. insert the node in the AVL true using binary search tree
2. once the node is added, the balance factor is updated.
3. Now check if only nodes violate the range of balance factor
4. If violated, perform rotations

Deletion

1. Find the element in the tree
2. Delete the node as per the BST deletion
3. Two case
    1. Deletion from right subtree
    2. Deletion from left subtree

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};
int max(int a, int b);
int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}
int max(int a, int b) {
  return (a > b) ? a : b;
```

```
}
struct Node *newNode(int key) {
  struct Node *node = (struct Node *) malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;
  x->right = y;
  y->left = T2;
  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;
  return x;
}
struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;
  y->left = x;
  x->right = T2;
  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;
  return y;
}
int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
```

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

  node->height = 1 + max(height(node->left),
               height(node->right));
  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);
  if (balance < -1 && key > node->right->key)
    return leftRotate(node);
  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }
  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }
  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) {
  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) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;
      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
    }
  }
  if (root == NULL)
    return root;
  root->height = 1 + max(height(root->left),
               height(root->right));
  int balance = getBalance(root);
  if (balance > 1 && getBalance(root->left) >= 0)
```

```
    return rightRotate(root);
  if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
  }
  if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);
  if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);  }

  return root;
}
void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}
int main() {
  struct Node *root = NULL;
  root = insertNode(root, 2);
  root = insertNode(root, 1);
  root = insertNode(root, 7);
  root = insertNode(root, 4);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 8);
  printPreOrder(root);
  root = deleteNode(root, 3);
  printf("\nAfter deletion: ");
  printPreOrder(root);
  return 0;
getch( );
}
```

Output:

```
4 2 1 3 7 5 8
After deletion: 4 2 1 7 5 8
```

Result:

Thus the c program to implement AVL tree was executed successfully.

Ex .NO: 8 Implementation of Heap Using Priority Queues

Aim:

To write a c program to implement heap using priority queues.

Algorithm:

1. To insert an element, attach the new element to any leaf
2. Swap the incorrectly placed node with its parent until the heap is properly satisfied
3. The max value is stored at root of the tree, it cant be directly removed. So it is repeated with any one of the leaves and removed
4. To remove an amount, change its priority to a value larger than the current . then shift it up

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void insert();
void del();
void display();
struct node
{
int priority;
int info;
struct node*next;
}*start,*q,*temp,*new;
typedef struct node *N;
void main()
{
int ch;
clrscr();
do
{
printf("\n [1]insertion \t [2]deletion \t [3]display [4]exit \t:");
scanf("%d",&ch);
switch(ch)
{
case 1:insert();
break;
case 2:del(); break;
case 3:display(); break;
case 4: break;
} }
```

```
while(ch<4);
}
void insert()
{
int item,itprio;
new=(struct node*) malloc(sizeof(struct node));
printf("enter the elt to be inserted:\t");
scanf("%d",&item);
printf("enter its priority:\t");
scanf("%d",&itprio);
new->info=item;
new->priority=itprio;
if(start==NULL||itprio<start->priority)
{
new->next=start;
start=new;
}
else
{
q=start;
while(q->next!=NULL&&q->next->priority<=itprio)
q=q->next;
new->next=q->next;
q->next=new;
}
}
void del()
{
if(start==NULL)
{
printf("\n queue underflow\n");
}
else
{
new=start;
printf("\n deleted item is %d \n",new->info);
start=start->next;
free(start);
}
}
void display()
{
temp=start;
if(start==NULL)
printf("queue is empty \n");
else
{
printf("queue is: \n");
while(temp!=NULL)
{
printf("\t %d [priority=%d]",temp->info,temp->priority);
temp=temp->next;
}
getch();
}
}
```

<u>Output:</u>

```
[1]insertion     [2]deletion     [3]display [4]exit      :1
enter the elt to be inserted:     20
enter its priority:        1
[1]insertion     [2]deletion     [3]display [4]exit      :1
enter the elt to be inserted:     10
enter its priority:        1
[1]insertion     [2]deletion     [3]display [4]exit      :1
enter the elt to be inserted:     87
enter its priority:        2
[1]insertion     [2]deletion     [3]display [4]exit      :2
deleted item is 20
```

Result:

Thus the c program to implement heap using priority queues was executed successfully.

Ex .NO: 9 Implementation of Dijkstra’s Algorithm

Aim:

To write a c program to implement Dijkstra’s algorithm.

Algorithm:

1. Create a set shortpath to store vertices that come in the way of the shortest path tree.
2. Initialize all distance value as infinite and assign distance values as 0 for source vertex. So that is picked
3. Loop until all vertices of the graph are in the shortpath
4. Take a new vertex that is not visited and is nearest
5. Add this vertex to shortpath

For all adjacent vertices of this vertex update distances. Now check every adjacent vertex of v, if sum of distance of U and weight of edge, else update it

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

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

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

void dijikstra(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 of %d = %d", i, distance[i]);
                        printf("\nPath = %d", i);
                        j=i;
                        do
                        {
                                j=pred[j];
                                printf(" <-%d", j);
                        }
                        while(j!=startnode);
                }
}
```

Result:

Thus the c program to implement Dijkstra’s algorithm was executed successfully.

Ex .NO: 10 Implementation of Prim's Algorithm

Aim:

To write a c program to implement Prim's algorithm.

ALGORITHM:

```
Step 1: Select a starting vertex
Step 2: Repeat Steps 3 and 4 until there are fringe vertices
Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight
Step 4: Add the selected edge and the vertex to the minimum spanning tree T
[END OF LOOP]
Step 5: EXIT
```

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

 void main()

{

clrscr();

 printf("\nEnter the number of nodes:");

 scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

 for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

 {

scanf("%d",&cost[i][j]);

 if(cost[i][j]==0)

 cost[i][j]=999;

            }

            visited[1]=1;

     printf("\n");

 while(ne < n)

            {

for(i=1,min=999;i<=n;i++)
```

```
 for(j=1;j<=n;j++)
if(cost[i][j]< min)
if(visited[i]!=0)
 {
 min=cost[i][j];
a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
 {
 printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
getch();
}
```

Result:

Thus the c program to implement Prim’s algorithm was executed successfully.

Ex .NO: 11 Implementation of Linear Search & Binary Search

Aim:

To write a c program to implement linear search & binary search.

Algorithm:

Linear search :

1. Start with index 0 and compare each element with the target
2. .If the target is found to be equal to the element, return its index
3. If the target is not found, return -1

Binary Search:

- Compare the target element with the middle element of the array.
- If the target element is greater than the middle element, then the search continues in the right half.
- Else if the target element is less than the middle value, the search continues in the left half.
- This process is repeated until the middle element is equal to the target element, or the target element is not in the array

If the target element is found, its index is returned, else -1 is returned.

PROGRAM:

Linear Search:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],i,no,srchno;
clrscr();
printf("\n Enter the number of elements\n");
scanf("%d",&no);
printf("\n Enter %d numbers\n",no);
for(i=0;i<no;++i)
scanf("%d",&a[i]);
printf("Enter the search number\n");
scanf("%d",&srchno);
for(i=0;i<no;++i)
if(a[i]==srchno)
{
printf("\n search number is present");
exit(0);
}
printf("\n Search number is not present");
getch( );
}
```

Output:

```
Enter the number of elements
3
Enter 3 numbers
1
2
3
Enter the search number
3
search number is present
```

Binary Search:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],i,no,srchno,top,bottom,mid,j,temp;
clrscr();
printf("\n Enter the number of elements\n");
scanf("%d",&no);
printf("\n Enter %d numbers\n",no);
for(i=0;i<no;++i)
scanf("%d",&a[i]);
printf("Enter the search number\n");
scanf("%d",&srchno);
for(i=0;i<no-1;++i)
{
for(j=i+1;j<no;++j)
{
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
printf("\n Sorted array in ascending order\n");
for(i=0;i<no;++i)
printf("%5d",a[i]);
bottom=0;
top=no-1;
while(top!=bottom+1)
{
mid=(bottom+top)/2;
if (a[mid]<=srchno)
bottom=mid;
else
top=mid;
}
if(a[bottom]==srchno)
printf("\n search number is present");
else
printf("\n Search number is not present");
getch( );
}
```

Output:

```
Enter the number of elements
3
Enter 3 numbers
1
2
3
Enter the search number:
3
Sorted array in ascending order
1 2 3
Search number is present
```

Result:

To write a c program to implement linear search & binary search

Ex .NO: 12 Implementation of Insertion sort & Selection sort

Aim:

To write a c program to implement insertion sort & selection sort .

Algorithm:

Insertion sort:

Step 1 – If it is the first element, it is already sorted. return 1;

Step 2 – Pick next element

Step 3 – Compare with all elements in the sorted sub-list

Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

Selection sort:

Step 1 – Set MIN to location 0

Step 2 – Search the minimum element in the list

Step 3 – Swap with value at location MIN

Step 4 – Increment MIN to point to next element

Step 5 – Repeat until list is sorted

PROGRAM:

Insertion Sort:

```
#include<stdio.h>
#include<conio.h>
void insert(int a[],int n)
{
int i,j,t;
for(i=n-2;i>=0;i--)
{
for(j=0;j<=i;j++)
{
if(a[j]>a[j+1])
{
t=a[j];
a[j]=a[j+1];
a[j+1]=t;
}
}
}
}
void main()
{
int a[100],n,i;
clrscr();
printf("\n\n Enter integer value for total no.s of elements to be sorted: ");
scanf("%d",&n);
for( i=0;i<=n-1;i++)
{ printf("\n\n Enter integer value for element no.%d : ",i+1);
scanf("%d",&a[i]);
}
insert(a,n);
printf("\n\n Finally sorted array is: ");
for( i=0;i<=n-1;i++)
printf("%3d",a[i]);
getch( );
}
```

Output:

```
Enter integer value for total no.s of elements to be sorted: 4
Enter integer value for element no.1 : 65
Enter integer value for element no.2 : 11
Enter integer value for element no.3 : 99
Enter integer value for element no.4 : 22
Finally sorted array is:  11 22 65 99
```

Selection Sort:

```
#include <stdio.h>
#include<conio.h>
#define MAXSIZE 500
void selection(int elements[], int maxsize);
int elements[MAXSIZE],maxsize;
void main()
{
int i;
printf(“\nHow many elements you want to sort: “);
scanf(“%d”,&maxsize);
printf(“\nEnter the values one by one: “);
for (i = 0; i < maxsize; i++)
{
printf (“\nEnter element %i :”,i);
scanf(“%d”,&elements[i]);
}
printf(“\nArray before sorting:\n”);
for (i = 0; i < maxsize; i++)
printf(“[%i], “,elements[i]);
printf (“\n”);
selection(elements, maxsize);
printf(“\nArray after sorting:\n”);
for (i = 0; i < maxsize; i++)
printf(“[%i], “, elements[i]);
}
void selection(int elements[], int array_size)
{
int i, j, k;
int min, temp;
for (i = 0; i < maxsize-1; i++)
min = i;
for (j = i+1; j < maxsize; j++) {
if (elements[j] < elements[min])
min = j;
}
temp = elements[i];
elements[i] = elements[min];
elements[min] = temp;
}
getch( );
```

Output:

```
How many elements you want to sort: 4
Enter the values one by one:
Enter element 1: 22
Enter element 2: 10
Enter element 3: 88
Enter element 4: 20
Array before sorting
22 10 88 20
Array after sorting:
10 20 22 88
```

Result:

Thus the c program to implement insertion sort & selection sort was executed successfully.

Ex .NO: 13 Implementation of Merge sort

Aim:

To write a c program to implement merge sort.

Algorithm:

Step 1 – if it is only one element in the list it is already sorted, return.

Step 2 – divide the list recursively into two halves until it can no more be divided.

Step 3 – merge the smaller lists into new list in sorted order

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define MAX 50

void mergeSort(int arr[],int low,int mid,int high);
void partition(int arr[],int low,int high);

int main(){

   int merge[MAX],i,n;
clrscr( );

   printf("Enter the total number of elements: ");
   scanf("%d",&n);

   printf("Enter the elements which to be sort: ");
   for(i=0;i<n;i++){
        scanf("%d",&merge[i]);
   }

   partition(merge,0,n-1);

   printf("After merge sorting elements are: ");
   for(i=0;i<n;i++){
        printf("%d ",merge[i]);
   }

  return 0;
```

```
getch( );
}

void partition(int arr[],int low,int high){

   int mid;

   if(low<high){
       mid=(low+high)/2;
       partition(arr,low,mid);
       partition(arr,mid+1,high);
       mergeSort(arr,low,mid,high);
   }
}

void mergeSort(int arr[],int low,int mid,int high){

   int i,m,k,l,temp[MAX];

   l=low;
   i=low;
   m=mid+1;

   while((l<=mid)&&(m<=high)){

       if(arr[l]<=arr[m]){
          temp[i]=arr[l];
          l++;
       }
       else{
          temp[i]=arr[m];
          m++;
       }
       i++;
   }

   if(l>mid){
       for(k=m;k<=high;k++){
          temp[i]=arr[k];
          i++;
       }
   }
   else{
       for(k=l;k<=mid;k++){
          temp[i]=arr[k];
          i++;
```

```
            }
        }

        for(k=low;k<=high;k++){
            arr[k]=temp[k];
        }
    }
```

<u>Output:</u>

```
Enter the total number of elements: 5
Enter the elements which to be sort: 10
23
1
88
65
After merge sorting elements are: 1 3 10 65 88
```

Result:

Thus the c program to implement merge sort was executed successfully.

Ex .NO: 14 Implementation of Open Addressing

Aim:

To write a c program to implement open addressing.

Algorithm:

1. Start
2. A structure that represented by hashcode function which returns the value of key%size where size is assumed as 20
3. Insert function is called
4. While inserting an element if the hashcode function returns a key that has been previously assigned to an element in the hash table
5. If there is a collision, then the hash tkey is incremented
6. This process is repeated until a space if found for current element or the hash table is full
7. Delete function is called to delete an element from the hash table
8. Stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void linear_prob(int[],int,int);
void display(int[]);
int create(int);
int a[100],num,key,i;
int tablesize=100,ch;
char ans;
void main()
{
clrscr();
printf("\n H ashing _collision handling by linear probing\n");
printf("\n ***************************");
printf("\n Enter the hash table size:");
scanf("%d",&tablesize);
for(i=0;i<tablesize;i++)
a[i]=-1;
printf("\n menu\n");
printf("\n 1.insertion\n 2.dipplay\n 3.exit\n");
do
{
printf("Enter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n Enter the number:");
scanf("%d",&num);
key=create(num);
linear_prob(a,key,num);
break;
case 2:
display(a);
break;
default:
printf("\n Invalid choice:");
}
}
while(ch!=3);
}
int create(int num)
{
int key;
key=num%10;return key;
}
void linear_prob(int a[100],int key,int num)
{
int flag,i,count=0;
flag=0;
if(a[key]==-1)
a[key]=num;
else
{
i=0;while(i<tablesize)
{
if(a[i]!=-1)
count++;
i++;
}
if(count==tablesize)
```

```
{
printf("\n Hash table is full:");
display(a);
getch();
}
for(i=key+1;i<tablesize;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[100])
{
int i;
printf("\n The has table is:\n");
for(i=0;i<tablesize;i++)
printf("\n %d%d",i,a[i]);
}
```

Output:

Hashing _collision handling by linear probing

```
 ***************************
 Enter the hash table size:2
 menu

 1.insertion
 2.display
 3.exit

 Enter your choice:1
 Enter the number:10
 Enter your choice:1
 Enter the number:20
 Enter your choice:2
 The hash table is:

 0 - 10
 1 - 20
```

Result:

Thus the c program to implement open addressing was executed successfully.