**CS8381 DATA STRUCTURES LABORATORY L T P C 0 0 4 2**

**OBJECTIVES**

* To implement linear and non-linear data structures
* To understand the different operations of search trees
* To implement graph traversal algorithms
* To get familiarized to sorting and searching algorithms

**LIST OF EXCERCISE**

1. Array implementation of Stack and Queue ADTs

2. Array implementation of List ADT

3. Linked list implementation of List, Stack and Queue ADTs

4. Applications of List, Stack and Queue ADTs

5. Implementation of Binary Trees and operations of Binary Trees

6. Implementation of Binary Search Trees

7. Implementation of AVL Trees

8. Implementation of Heaps using Priority Queues.

9. Graph representation and Traversal algorithms

10. Applications of Graphs

11. Implementation of searching and sorting algorithms

12. Hashing – any two collision techniques

TOTAL: 60 PERIODS 47

**OUTCOMES:**

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

* Write functions to implement linear and non-linear data structure operations
* Suggest appropriate linear / non-linear data structure operations for solving a given problem
* Appropriately use the linear / non-linear data structure operations for a given problem
* Apply appropriate hash functions that result in a collision free scenario for data storage and retrieval

|  |
| --- |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **EX.No** | **DATE** | **TABLE OF CONTENT** | **STAFF SIGNATURE** |
| 1a |  | Array implementation of Stack ADTs |  |
| 1b |  | Array implementation of Queue ADTs |  |
| 2 |  | Array implementation of List ADT |  |
| 3A |  | Linked list implementation of List |  |
| 3B |  | Linked list implementation of Stack |  |
| 3C |  | Linked list implementation of Queue |  |
| 4a |  | Applications of List-Polynomial Manipulation |  |
| 4b |  | Applications of Stack-Infix to Postfix conversion |  |
| 4c |  | Applications of Queue-FCFS Scheduling |  |
| 5 |  | Implementation of Binary Trees and operations of Binary Trees |  |
| 6 |  | Implementation of Binary Search Trees |  |
| 7 |  | Implementation of AVL Trees |  |
| 8 |  | Implementation of Heaps using Priority Queues. |  |
| 9 |  | Graph representation |  |
| 10 |  | Graph Traversal algorithms-BFS & DFS |  |
| 11 |  | Applications of Graphs-Topological sort |  |
| 12 |  | Implementation of searching and sorting algorithms |  |
| 13 |  | Hashing – Linear and Quadratic probing |  |

**1A Array implementation of Stack ADTs**

**AIM**:

To write a C program to implement stack ADT using array

**ALGORITHM**

Step 1: Start the program.

Step 2: For Push operation, check for stack overflow

Step 3: If Top>=N then print stack overflow else increment Top and insert the element.

Step 4: For Pop operation, check for underflow of the stack.

Step 5: If Top=0 then print stack underflow else decrement Top and delete the element

Step 6: Stop the program.

**PROGRAM**

/\*

\* C program to implement stack. Stack is a LIFO data structure.

\* Stack operations: PUSH(insert operation), POP(Delete operation)

\* and Display stack.

\*/

#include <stdio.h>

#define MAXSIZE 5

struct stack

{

int stk[MAXSIZE];

int top;

};

typedef struct stack STACK;

STACK s;

void push(void);

int pop(void);

void display(void);

void main ()

{

int choice;

int option = 1;

s.top = -1;

printf ("STACK OPERATION\n");

while (option)

{

printf ("------------------------------------------\n");

printf (" 1 --> PUSH \n");

printf (" 2 --> POP \n");

printf (" 3 --> DISPLAY \n");

printf (" 4 --> EXIT \n");

printf ("------------------------------------------\n");

printf ("Enter your choice\n");

scanf ("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

return;

}

fflush (stdin);

printf ("Do you want to continue(Type 0 or 1)?\n");

scanf ("%d", &option);

}

}

/\* Function to add an element to the stack \*/

void push ()

{

int num;

if (s.top == (MAXSIZE - 1))

{

printf ("Stack is Full\n");

return;

}

else

{

printf ("Enter the element to be pushed\n");

scanf ("%d", &num);

s.top = s.top + 1;

s.stk[s.top] = num;

}

return;

}

/\* Function to delete an element from the stack \*/

int pop ()

{

int num;

if (s.top == - 1)

{

printf ("Stack is Empty\n");

return (s.top);

}

else

{

num = s.stk[s.top];

printf ("poped element is = %dn", s.stk[s.top]);

s.top = s.top - 1;

}

return(num);

}

/\* Function to display the status of the stack \*/

void display ()

{

int i;

if (s.top == -1)

{

printf ("Stack is empty\n");

return;

}

else

{

printf ("\n The status of the stack is \n");

for (i = s.top; i >= 0; i--)

{

printf ("%d\n", s.stk[i]);

}

}

printf ("\n");

}

**OUTPUT**

STACK OPERATION

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

1

Enter the element to be pushed

34

Do you want to continue(Type 0 or 1)?

0

$ a.out

STACK OPERATION

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

1

Enter the element to be pushed

34

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

2

poped element is = 34

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

3

Stack is empty

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

1

Enter the element to be pushed

50

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

1

Enter the element to be pushed

60

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

3

The status of the stack is

60

50

Do you want to continue(Type 0 or 1)?

1

------------------------------------------

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

------------------------------------------

Enter your choice

4

RESULT:

Thus the C program to implement stack ADT using array has been implemented successfully.

1B Array implementation of Queue ADTs

**AIM:**

To write a C program to implement Queue ADT using array.

**ALGORITHM:**

Step 1: Start the program.

Step 2: For queue insertion operation, check for queue overflow

Step 3: If Rear>=N then print queue overflow else increment rear pointer and insert the element.

Step 4: For queue deletion operation, check for underflow of the queue.

Step 5: If Front=0 then print queue underflow else delete the element and increment the front pointer

Step 6: Stop the program.

**PROGRAM**

/\*

\* C Program to Implement a Queue using an Array

\*/

#include <stdio.h>

#define MAX 50

int queue\_array[MAX];

int rear = - 1;

int front = - 1;

main()

{

int choice;

while (1)

{

printf("1.Insert element to queue \n");

printf("2.Delete element from queue \n");

printf("3.Display all elements of queue \n");

printf("4.Quit \n");

printf("Enter your choice : ");

scanf("%d", &choice);

switch (choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(1);

default:

printf("Wrong choice \n");

} /\*End of switch\*/

} /\*End of while\*/

} /\*End of main()\*/

insert()

{

int add\_item;

if (rear == MAX - 1)

printf("Queue Overflow \n");

else

{

if (front == - 1)

/\*If queue is initially empty \*/

front = 0;

printf("Inset the element in queue : ");

scanf("%d", &add\_item);

rear = rear + 1;

queue\_array[rear] = add\_item;

}

} /\*End of insert()\*/

delete()

{

if (front == - 1 || front > rear)

{

printf("Queue Underflow \n");

return ;

}

else

{

printf("Element deleted from queue is : %d\n", queue\_array[front]);

front = front + 1;

}

} /\*End of delete() \*/

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\_array[i]);

printf("\n");

}

} /\*End of display() \*/

**OUTPUT**

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 1

Inset the element in queue : 10

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 1

Inset the element in queue : 15

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 1

Inset the element in queue : 20

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 1

Inset the element in queue : 30

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 2

Element deleted from queue is : 10

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 3

Queue is :

15 20 30

1.Insert element to queue

2.Delete element from queue

3.Display all elements of queue

4.Quit

Enter your choice : 4

RESULT:

Thus the C program to implement Queue ADT using array has been implemented successfully.

**2 Array implementation of List ADT**

**AIM:**

To write a C program to implement the List ADT using arrays.

**ALGORITHM:**

Step 1: Start.

Step 2: Declare the necessary functions for implementation.

Step 3: Get the input from the user and store it an array.

Step 4: In Insertion, half of the elements to be shifted upwards and in deletion half of the

elements to be shifted downwards.

Step 5: Display the output using an array.

Step 6: Stop.

WWW.VIDYARTHIPLUS.COM

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define MAX 10

void create();

void insert();

void deletion();

void search();

void display();

int a,b[20], n, p, e, f, i, pos;

void main()

{

clrscr();

int ch;

char g='y';

do

{

printf("\n main Menu");

printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit \n");

printf("\n Enter your Choice");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

WWW.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

case 2:

deletion();

break;

case 3:

search();

break;

case 4:

insert();

break;

case 5:

display();

break;

case 6:

exit();

break;

default:

printf("\n Enter the correct choice:");

}

printf("\n Do u want to continue:::");

scanf("\n%c", &g);

}

while(g=='y'||g=='Y');

getch();

}W.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

void create()

{

printf("\n Enter the number of nodes");

scanf("%d", &n);

for(i=0;i<n;i++)

{

printf("\n Enter the Element:",i+1);

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

}

}

void deletion()

{

printf("\n Enter the position u 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("\n The Elements after deletion");

for(i=0;i<n;i++)

{

printf("\t%d", b[i]);

WWW.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

}

}

void search()

{

printf("\n Enter the Element to be searched:");

scanf("%d", &e);

for(i=0;i<n;i++)

{

if(b[i]==e)

{

printf("Value is in the %d Position", i);

}

}

}

void insert()

{

printf("\n Enter the position u need to insert::");

scanf("%d", &pos);

if(pos>=n)

{

printf("\n invalid Location::");

}

else

{

for(i=MAX-1;i>=pos-1;i--)

{

b[i+1]=b[i];W.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

}

printf("\n Enter the element to insert::\n");

scanf("%d",&p);

b[pos]=p;

n++;

}

printf("\n The list after insertion::\n");

display();

}

void display()

{

printf("\n The Elements of The list ADT are:");

for(i=0;i<n;i++)

{

printf("\n\n%d", b[i]);

}

}

WWW.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

**OUTPUT:**

Main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice: 1

Enter the number of elements: 4

Enter the elements:

10

20

30

40

Do u want to continue(y/n): y

main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice: 2

Enter the element to delete:20

Elements after deletion: 10

30

40

Do u want to continue(y/n): y

WWW.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice: 3

Enter the element to search: 100

Element not found

Do u want to continue(y/n): y

main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice: 4

Enter the element to insert: 15

Enter the position to insert:2

Elements after insertion:

10

15

30

40

Do u want to continue(y/n): y

WWW.VIDYARTHIPLUS.COM

WWW.VIDYARTHIPLUS.COM V+ TEAM

main Menu

1.Create

2.Delete

3.Search

4.Insert

5.Display

6.Exit

Enter your Choice: 4

RESULT:

Thus the C program to implement the List ADT using arrays has been executed successfully.

**3A Linked list implementation of List(Singly Linked List)**

**AIM**

To write a C program for singly linked list using list

**ALGORITHM:**

Step1: Start the program

Step2: Get the values for insertion into the list

Step3: Get the position of the value to be deleted

Step3: Display the values entered into the list

Step4: Display the number of items in the list

Step5: Stop the program

**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 :

100

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 :

200

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 :

300

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 :

400

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 :

500

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 :

# 100 # # 200 # # 300 # # 400 # # 500 #

No Of Items In Linked List : 5

Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:4

No Of Items In Linked List : 5

Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:2

No Of Items In Linked List : 5

Display Linked List :

Enter Position for Delete Element :

3

Deleted Successfully

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 :

# 100 # # 200 # # 400 # # 500 #

No Of Items In Linked List : 4

Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:2

No Of Items In Linked List : 4

Display Linked List :

Enter Position for Delete Element :

6

Invalid Position

Options

1 : Insert into Linked List

2 : Delete from Linked List

3 : Display Linked List

4 : Count Linked List

Others : Exit()

Enter your option:

5

------------------

RESULT:

Thus the C program for singly linked list using list has been executed successfully.

**3B Implementation of Stack Using Linked List**

**AIM:**

To write a c program to implement stack using linked list

**ALGORITHM:**

Step 1: Start the program.

Step 2: For Push operation, check for stack overflow

Step 3: If Top>=N then print stack overflow else increment Top and insert the

element.

Step 4: For Pop operation, check for underflow of the stack.

Step 5: If Top=0 then print stack underflow else decrement Top and delete the

element

Step 6: Stop the program.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int Data;

struct Node \*next;

}\*top;

void popStack()

{

struct Node \*temp, \*var=top;

if(var==top)

{

top = top->next;

free(var);

}

else

printf("\nStack Empty");

}

void push(int value)

{

struct Node \*temp;

temp=(struct Node \*)malloc(sizeof(struct Node));

temp->Data=value;

if (top == NULL)

{

top=temp;

top->next=NULL;

}

else

{

temp->next=top;

top=temp;

}

}

void display()

{

struct Node \*var=top;

if(var!=NULL)

{

printf("\nElements are as:\n");

while(var!=NULL)

{

printf("\t%d\n",var->Data);

var=var->next;

}

printf("\n");

}

else

printf("\nStack is Empty");

}

int main()

{

int i=0;

top=NULL;

clrscr();

printf(" \n1. Push to stack");

printf(" \n2. Pop from Stack");

printf(" \n3. Display data of Stack");

printf(" \n4. Exit\n");

while(1)

{

printf(" \nChoose Option: ");

scanf("%d",&i);

switch(i)

{

case 1:

{

int value;

printf("\nEnter a value to push into Stack: ");

scanf("%d",&value);

push(value);

break;

}

case 2: {

popStack();

printf("\n The last element is popped");

break;

}

case 3:

{

display();

break;

}

case 4:

{

struct Node \*temp;

while(top!=NULL)

{

temp = top->next;

free(top);

top=temp;

}

exit(0);

}

default:

{

printf("\nwrong choice for operation");

}}}}

**OUTPUT**

1. Push to stack

2. Pop from Stack

3. Display data of Stack

4. Exit\

Choose Option:1

Enter a value to push into Stack 5

Choose Option:1

Enter a value to push into Stack 3

Choose Option:1

Enter a value to push into Stack 2

Choose Option:1

Enter a value to push into Stack 9

Choose Option:3

Elements are as :

5

3

2

9

Choose Option:2

The last element is popped

Choose Option:3

Elements are as :

3

2

9

**RESULT:** Thus a C program is written to implement stack using linked list and executed Successfully

**3C Implementation of Queue using Linked List**

**AIM:**

To write a C program to implement queue using linked list

**ALGORITHM:**

Step 1: Start the program.

Step 2: For queue insertion operation, check for queue overflow

Step 3: If R>=N then print queue overflow else increment rear pointer and insert

the element.

Step 4: For queue deletion operation, check for underflow of the queue.

Step 5: If F=0 then print queue underflow else delete the element and increment

the front pointer

Step 6: Stop the program.

**PROGRAM**

#include<stdio.h >

#include<conio.h >

#include<alloc.h >

struct queue

{

int data;

struct queue \*next;

};

struct queue \*addq(struct queue \*front);

struct queue \*delq(struct queue \*front);

void main()

{

struct queue \*front;

int reply,option,data;

clrscr();

front=NULL;

do

{

printf("\n1.addq");

printf("\n2.delq");

printf("\n3.exit");

printf("\nSelect the option");

scanf("%d",&option);

switch(option)

{

case 1 : //addq

front=addq(front);

printf("\n The element is added into the queue");

break;

case 2 : //delq

front=delq(front);

break;

case 3 : exit(0);

}

}while(1);

}

struct queue \*addq(struct queue \*front)

{

struct queue \*c,\*r;

//create new node

c=(struct queue\*)malloc(sizeof(struct queue));

if(c==NULL)

{

printf("Insufficient memory");

return(front);

}

//read an insert value from console

printf("\nEnter data");

scanf("%d",&c->data);

c->next=NULL;

if(front==NULL)

{

front=c;

}

else

{

//insert new node after last node

r=front;

while(r->next!=NULL)

{

r=r->next;

}}

return(front);

}

struct queue \*delq(struct queue \*front)

{

struct queue \*c;

if(front==NULL)

{

printf("Queue is empty");

return(front);

}

//print the content of first node

printf("Deleted data:%d",front->data); //delete first node

c=front;

front=front->next;

free(c);

return(front);

}

**OUTPUT**

1.addq

2.delq

3.exit

Select the option 1

Enter data 8

1.addq

2.delq

3.exit

Select the option 1

Enter data 5

1.addq

2.delq

3.exit

Select the option 1

Enter data 9

1.addq

2.delq

3.exit

Select the option 1

Enter data 1

1.addq

2.delq

3.exit

Select the option 2

Deleted data: 8

1.addq

2.delq

3.exit

Select the option 3

RESULT:

Thus the C program to implement queue using linked list has been executed successfully.

**4A APPLICATION OF LIST (POLYNOMIAL ADITION)**

**AIM:**

To write a C program to implement the polynomial addition.

**ALGORITHM:**

**POLY ADD (POLY1: POLY2: POLY) HEAD: POLY**

Step 1: Assign HEAD+=NULL

Step2: While (POLY! =null)

Step3: HEAD=INSERTNODE (HEAD, COPYNODE ,(POLY1,1))

Step4: POLY1=POLY1\_NEXT

Step5: [End of Step2 while structure]

Step6: While (POLY2 1=NULL)

Step7: HEAD =INSERTNODE (HEAD, COPYNODE(POLY2,1))

Step8: POLY2=POLY2\_NEXT

Step9: [End of Step 6 while Structure]

Step10: Return HEAD

END POLYADD()

**PROGRAM:**

/\*\*

\* Add two polynomials

\* Using Linked List

\* @author Swashata

\* @for Dearest Froggie

\*/

#include<stdio.h>

#include<stdlib.h>

/\*\*

\* The structure for the polynomial

\* This is a linked list with two variable

\* int coeff The Coefficient

\* int pow The power of x

\*/

typedef struct link {

int coeff;

int pow;

struct link \* next;

} my\_poly;

/\*\* The prototypes \*/

void my\_create\_poly(my\_poly \*\*);

void my\_show\_poly(my\_poly \*);

void my\_add\_poly(my\_poly \*\*, my\_poly \*, my\_poly \*);

/\*\*

\* The simple menu driven main function

\*/

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;

}

/\*\*

\* The create polynomial function

\* @param my\_poly \*\* node The pointer to the head of the polynomial

\* We will modify the parameter and will store the base address

\* @return void

\*/

void my\_create\_poly(my\_poly \*\* node) {

int flag; //A flag to control the menu

int coeff, pow;

my\_poly \* tmp\_node; //To hold the temporary last address

tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly)); //create the first node

\*node = tmp\_node; //Store the head address to the reference variable

do {

//Get the user data

printf("\nEnter Coeff:");

scanf("%d", &coeff);

tmp\_node->coeff = coeff;

printf("\nEnter Pow:");

scanf("%d", &pow);

tmp\_node->pow = pow;

//Done storing user data

//Now increase the Linked on user condition

tmp\_node->next = NULL;

//Ask user for continuation

printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");

scanf("%d", &flag);

//printf("\nFLAG: %c\n", flag);

//Grow the linked list on condition

if(flag) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly)); //Grow the list

tmp\_node = tmp\_node->next;

tmp\_node->next = NULL;

}

} while (flag);

}

/\*\*

\* The show polynomial function

\* Prints the Polynomial in user readable format

\* @param my\_poly \* node The polynomial linked list

\* @return void

\*/

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(" + ");

}

}

/\*\*

\* The polynomial add function

\* Adds two polynomial to a given variable

\* @param my\_poly \*\* result Stores the result

\* @param my\_poly \* poly1 The first polynomial expression

\* @param my\_poly \* poly2 The second polynomial expression

\* @return void

\*/

void my\_add\_poly(my\_poly \*\* result, my\_poly \* poly1, my\_poly \* poly2) {

my\_poly \* tmp\_node; //Temporary storage for the linked list

tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node->next = NULL;

\*result = tmp\_node; //Copy the head address to the result linked list

//Loop while both of the linked lists have value

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;

}

//Grow the linked list on condition

if(poly1 && poly2) {

tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

tmp\_node = tmp\_node->next;

tmp\_node->next = NULL;

}

}

//Loop while either of the linked lists has value

while(poly1 || poly2) {

//We have to create the list at beginning

//As the last while loop will not create any unnecessary node

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;

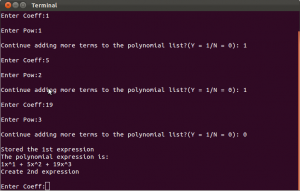
}

}

printf("\nAddition Complete");

}

OUTPUT:

[](https://www.intechgrity.com/wp-content/uploads/2011/11/entering-the-first-expression.png)

**RESULT:**

Thus the program to implement the polynomial addition has been executed successfully.

**4B APPLICATION OF STACK (INFIX TO POSTFIX)**

**AIM:**

To write a C program to convert the infix expression into postfix expression using stack.

**ALGORITHM:**

Step 1: Start the program.

Step 2: Get the infix expression as input.

Step 3: Read the input from left to right.

Step 4: If the input is operand then place it in the postfix expression.

Step 5: Else if the input symbol is an operator then check for the operator type and also the precedence, pop entries from the stack and place them in the postfix expression until the lowest priority operator is encountered.

Step 6: „(„symbol will be popped from stack only when we get a „)‟ symbol.

Step 7: When the input is completely read then pop the elements in stack until it becomes empty.

Step 8: Display the postfix expression.

Step 9: Stop the program.

**PROGRAM**

#include<stdio.h>

char stack[20];

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;

}

main()

{

    char exp[20];

    char \*e, x;

    printf("Enter the expression :: ");

    scanf("%s",exp);

    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());

    }}

**OUTPUT:**

Enter the expression :: a+b\*c

abc\*+

Enter the expression :: (a+b)\*c+(d-a)

ab+c\*da-+

RESULT:

Thus the program to convert the infix expression into postfix expression using stack has been implemented successfully.

**4C APPLICATION OF QUEUE (FCFS SCHEDULING)**

**AIM**

To write a program to implement the FCFS scheduling algorithm

**ALGORITHM**

1. Start the process

2. Declare the array size

3. Get the number of processes to be inserted

4. Get the value

5. Start with the first process from it’s initial position let other process to be in queue

6. Calculate the total number of burst time

7.  Display the values

8. Stop the process

**PROGRAM**

#include<stdio.h>

main()

{

int n,a[10],b[10],t[10],w[10],g[10],i,m;

float att=0,awt=0;

for(i=0;i<10;i++)

{

a[i]=0; b[i]=0; w[i]=0; g[i]=0;

}

printf("enter the number of process");

scanf("%d",&n);

printf("enter the burst times");

for(i=0;i<n;i++)

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

printf("\nenter the arrival times");

for(i=0;i<n;i++)

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

g[0]=0;

for(i=0;i<10;i++)

g[i+1]=g[i]+b[i];

for(i=0;i<n;i++)

{

w[i]=g[i]-a[i];

t[i]=g[i+1]-a[i];

awt=awt+w[i];

att=att+t[i];

}

awt =awt/n;

att=att/n;

printf("\n\tprocess\twaiting time\tturn arround time\n");

for(i=0;i<n;i++)

{

printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

}

printf("the average waiting time is %f\n",awt);

printf("the average turn around time is %f\n",att);

}

**OUTPUT:**

enter the number of process 4

enter the burst times

4  9  8  3

enter the arrival times

0  2  4  3

        process        waiting time     turn arround time

            p0                       0                          4

            p1                       2                         11

            p2                       9                         17

            p3                      18                        21

the average waiting time is 7.250000

the average turn around time is 13.250000

RESULT:

Thus the program to implement the FCFS scheduling algorithm has been implemented successfully.

**5 Implementation of Binary Trees and operations of Binary Trees**

**AIM:**

To write a C program to implement Binary Trees and operations of Binary Trees.

**ALGORITHM:**

### Creation of binary tree:

* Check first if tree is empty, then insert node as root.
* Check if node value to be inserted is lesser than root node value, then
* Call insert() function recursively while there is non-NULL left node
* When reached to leftmost node as NULL, insert new node.
* Check if node value to be inserted is greater than root node value, then
* Call insert() function recursively while there is non-NULL right node
* When reached to rightmost node as NULL, insert new node

**Searching:**

* Check first if tree is empty, then return NULL.
* Check if node value to be searched is equal to root node value, then return node
* Check if node value to be searched is lesser than root node value, then call search() function recursively with left node
* Check if node value to be searched is greater than root node value, then call search() function recursively with right node
* Repeat step 2, 3, 4 for each recursion call of this search function until node to be searched is found.

**Deletion of binary tree**

* Check first if root node is non-NULL, then
* Call deltree() function recursively while there is non-NULL left node
* Call deltree() function recursively while there is non-NULL right node
* Delete the node.

**Displaying binary tree**

* Binary tree can be displayed in three forms – pre-order, in-order and post-order.
* Pre-order displays root node, left node and then right node.
* In-order displays left node, root node and then right node.
* Post-order displays left node, right node and then root node.

**Pre-order display**

* Display value of root node.
* b.Call print\_preorder() function recursively while there is non-NULL left node
* Call print\_preorder() function recursively while there is non-NULL right node

**In-order display**

* Call print\_inorder() function recursively while there is non-NULL left node
* Display value of root node.
* Call print\_inorder() function recursively while there is non-NULL right node

**Post-order display**

* Call print\_postorder() function recursively while there is non-NULL left node
* Call print\_postorder() function recursively while there is non-NULL right node
* Display value of root node.

**PROGRAM:**

#include<stdlib.h>

#include<stdio.h>

struct bin\_tree {

int data;

struct bin\_tree \* right, \* left;

};

typedef struct bin\_tree node;

void insert(node \*\* tree, int val)

{

node \*temp = NULL;

if(!(\*tree))

{

temp = (node \*)malloc(sizeof(node));

temp->left = temp->right = NULL;

temp->data = val;

\*tree = temp;

return;

}

if(val < (\*tree)->data)

{

insert(&(\*tree)->left, val);

}

else if(val > (\*tree)->data)

{

insert(&(\*tree)->right, val);

}

}

void print\_preorder(node \* tree)

{

if (tree)

{

printf("%d\n",tree->data);

print\_preorder(tree->left);

print\_preorder(tree->right);

}

}

void print\_inorder(node \* tree)

{

if (tree)

{

print\_inorder(tree->left);

printf("%d\n",tree->data);

print\_inorder(tree->right);

}

}

void print\_postorder(node \* tree)

{

if (tree)

{

print\_postorder(tree->left);

print\_postorder(tree->right);

printf("%d\n",tree->data);

}

}

void deltree(node \* tree)

{

if (tree)

{

deltree(tree->left);

deltree(tree->right);

free(tree);

}

}

node\* search(node \*\* tree, int val)

{

if(!(\*tree))

{

return NULL;

}

if(val < (\*tree)->data)

{

search(&((\*tree)->left), val);

}

else if(val > (\*tree)->data)

{

search(&((\*tree)->right), val);

}

else if(val == (\*tree)->data)

{

return \*tree;

}

}

void main()

{

node \*root;

node \*tmp;

//int i;

root = NULL;

/\* Inserting nodes into tree \*/

insert(&root, 9);

insert(&root, 4);

insert(&root, 15);

insert(&root, 6);

insert(&root, 12);

insert(&root, 17);

insert(&root, 2);

/\* Printing nodes of tree \*/

printf("Pre Order Display\n");

print\_preorder(root);

printf("In Order Display\n");

print\_inorder(root);

printf("Post Order Display\n");

print\_postorder(root);

/\* Search node into tree \*/

tmp = search(&root, 4);

if (tmp)

{

printf("Searched node=%d\n", tmp->data);

}

else

{

printf("Data Not found in tree.\n");

}

/\* Deleting all nodes of tree \*/

deltree(root);

}

**OUTPUT**

Pre Order Display

9

4

2

6

15

12

17

In Order Display

2

4

6

9

12

15

17

Post Order Display

2

6

4

12

17

15

9

Searched node=4

RESULT:

Thus the program to implement Binary Trees and operations of Binary Trees has been implemented successfully.

6 Implementation of Binary Search Trees

**AIM:**

To construct a binary search tree and perform various operation.

**ALGORITHM:**

STEP: 1 [Include all the necessary header files.]

STEP: 2 [Declare the structure with all necessary variables.]

STEP: 3 Read x;

STEP: 4 Call INORDER().

STEP: 5 Call PREORDER().

STEP: 6 Call POSTORDER().

STEP: 7 Call display().

**Algorithm For INSERT(P,X)**

1. If (p🡨NULL)

Create P

P🡨data🡨x.

P-->lchild🡨P-->rchild<--NULL

Else

1.WHILE(TEMP!=NULL)

2.Temp2🡨Temp1

3.If(temp1-->data-->x)

4.Else Temp1🡨Temp1-->rchild

5.[End of while structure]

6.If(temp2-->data-->x)

7.Temp 2🡨Temp2-->lchild

8.Temp 2-->data<--x

9.Temp2-->data🡨slchild🡨temp2-->rchild Null

10.Else

11.Temp 2-->Temp2🡨Temp2-->rchild🡨null

12.Temp2-->data<--x

13.Temp 2-->lchild🡨Temp 2-->rchild🡨null

14.[Return P]

**Algorithm For INORDER(p)**

1. If(p!=Null)

2. CALL INORDER (p🡪xdhild)

3. WRITE (D🡪data)

4. CALL INORDER (p🡪rchild)

5. [End the function]

**Algorithm for PREORDER**

1. If (pl =NULL)

2. WRITE (P\_Data)

3. CALL PREORDER (P🡪lCHILD)

4. CALL PREORDER (P🡪 Rchild)

5. [END OF FUNTION]

**Algorithm for POSTORDER**

1. If (P!=NULL)

2. Call POSTORDER (P🡪lchild)

3. Call POSTORDER (P🡪rchild)

4. Write (P🡪data)

5. [End of function]

**Algorithm for COUNT**

If (P==NULL)

1. Return 0

2. Else

3. [Return (1+count(P🡪lchild)+call count(P🡪rchild)) ]

4. Algorithm for postorder

**Algorithm for DISPLAY**

If (T!=NULL)

1.X\_(lm+rm)/2

2.Call goto xy (x,4\*y)

3.Write (t.data)

4.Call display (t🡪lchild, lm,x, l+1)

5.Call display (t🡪rchild, x, rm,l+1)

6.[END THE FUNCTION}

**Algorithm for SEARCH**

1. while(temp!=NULL)

2. If (temp🡪data🡨t)

[Return temp]

3.If (Temp🡪data🡨x)

Temp\_temp\_lchild

4. ELSE

\_temp\_rchild

5.[RETURN NULL]

PROGRAM

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define NULL 0

struct treenode

{

int element;

struct treenode \*left;

struct treenode \*right;

};

typedef struct treenode \*position,\*searchtree;

searchtree insert(int x,searchtree t)

{

if(t==NULL)

{

t=(struct treenode \*)malloc(sizeof(struct treenode));

if(t==NULL)

exit(0);

else

{

t->element=x;

t->left=t->right=NULL;

}

}

else

if(x<t->element)

t->left=insert(x,t->left);

else

if(x>t->element)

t->right=insert(x,t->right);

return t;

}

position findmin(searchtree t)

{

if(t==NULL)

return NULL;

else

return findmin(t->left);

}

position findmax(searchtree t)

{

if(t==NULL)

return NULL;

else

if(t->right==NULL)

return t;

else

return findmax(t->right);

}

searchtree rem(int x,searchtree t)

{

position temp;

if(t==NULL)

printf("\n Element not found\t");

else

if(x<t->element)

t->left=rem(x,t->left);

else

if(x>t->element)

t->right=rem(x,t->right);

else

if(t->left&&t->right)

{

temp=findmin(t->right);

t->element=temp->element;

t->right=rem(t->element,t->right);

}

else

{

temp=t;

if(t->left==NULL)

t=t->right;

else

if(t->right==NULL)

t=t->left;

free(temp);

}

return t;

}

void intrav(searchtree head)

{

if(head==NULL)

return;

if(head->left!=NULL)

intrav(head->left);

printf("%d\t",head->element);

if(head->right!=NULL)

intrav(head->right);

}

void main()

{

int n,i,dat,ch;

searchtree t=NULL;

position node;

clrscr();

printf("\nElement no.of element:\t");

scanf("%d",&n);

printf("\nEnter the elements:\t");

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

{

scanf("%d",&dat);

t=insert(dat,t);

}

intrav(t);

do

{

printf("\n\n");

printf("\n\*\*\*\*MENU\*\*\*\*\n");

printf("\nEnter 1->Insert a node \n");

printf(" 2->Delete a node \n");

printf(" 3->Find Minimum\n");

printf(" 4->Find Maximum\n");

printf(" 5->Display(Inorder Traversal\n");

printf(" 6->Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\n Enter the element to be inserted:\t");

scanf("%d",&dat);

t=insert(dat,t);

break;

case 2:

printf("\n Enter the node to be deleted:\t");

scanf("%d",&dat);

t=rem(dat,t);

break;

case 3:

node=findmin(t);

printf("\nThe minimum element is %d",node->element);

break;

case 4:

node=findmax(t);

printf("\n The maximum element is %d",node->element);

break;

case 5:

intrav(t);

break;

case 6:

exit(0);

}

}

while(ch!=6);

getch();

}

**OUTPUT:**

Enter no of elements: 3

Enter the elements:

5

2

9

2 5 9

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:1

Enter the element to be inserted: 4

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:1

Enter the element to be inserted:6

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:2

Enter the element to be deleted:5

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice: 5

2 4 6 9

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:3

The minimum element is:2

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:4

The maximum element is:9

\*\*\*\*MENU\*\*\*\*

1 >Insert a node

2 >Delete a node

3 >Find Minimum

4 >Find Maximum

5 >Display(Inorder Traversal)

6 >Exit

Enter your choice:6

RESULT:

Thus the program to construct a binary search tree and perform various operations has been executed successfully.

7 Implementation of AVL Trees

AIM:

To write a C program for the implementation of AVL trees.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare all the functions and using the switch case select the operation to be performed.

Step 3: Declare a structure with all the required variables and allocate the memory using the malloc function.

Step 4: Check if (search(root,info)==NULL),if so assign the value to the variable root .

Step 5: Insert an element into the tree.

Step 6: After insertion check if the tree is balanced or not .If not balance the tree.

Step 7: Stop the program.

**PROGRAM**

#include<stdio.h>

typedef struct node

{

int data;

struct node \*left,\*right;

int ht;

}node;

node \*insert(node \*,int);

node \*Delete(node \*,int);

void preorder(node \*);

void inorder(node \*);

int height( node \*);

node \*rotateright(node \*);

node \*rotateleft(node \*);

node \*RR(node \*);

node \*LL(node \*);

node \*LR(node \*);

node \*RL(node \*);

int BF(node \*);

int main()

{

node \*root=NULL;

int x,n,i,op;

do

{

printf("\n1)Create:");

printf("\n2)Insert:");

printf("\n3)Delete:");

printf("\n4)Print:");

printf("\n5)Quit:");

printf("\n\nEnter Your Choice:");

scanf("%d",&op);

switch(op)

{

case 1: printf("\nEnter no. of elements:");

scanf("%d",&n);

printf("\nEnter tree data:");

root=NULL;

for(i=0;i<n;i++)

{

scanf("%d",&x);

root=insert(root,x);

}

break;

case 2: printf("\nEnter a data:");

scanf("%d",&x);

root=insert(root,x);

break;

case 3: printf("\nEnter a data:");

scanf("%d",&x);

root=Delete(root,x);

break;

case 4: printf("\nPreorder sequence:\n");

preorder(root);

printf("\n\nInorder sequence:\n");

inorder(root);

printf("\n");

break;

}

}while(op!=5);

return 0;

}

node \* insert(node \*T,int x)

{

if(T==NULL)

{

T=(node\*)malloc(sizeof(node));

T->data=x;

T->left=NULL;

T->right=NULL;

}

else

if(x > T->data) // insert in right subtree

{

T->right=insert(T->right,x);

if(BF(T)==-2)

if(x>T->right->data)

T=RR(T);

else

T=RL(T);

}

else

if(x<T->data)

{

T->left=insert(T->left,x);

if(BF(T)==2)

if(x < T->left->data)

T=LL(T);

else

T=LR(T);

}

T->ht=height(T);

return(T);

}

node \* Delete(node \*T,int x)

{

node \*p;

if(T==NULL)

{

return NULL;

}

else

if(x > T->data) // insert in right subtree

{

T->right=Delete(T->right,x);

if(BF(T)==2)

if(BF(T->left)>=0)

T=LL(T);

else

T=LR(T);

}

else

if(x<T->data)

{

T->left=Delete(T->left,x);

if(BF(T)==-2) //Rebalance during windup

if(BF(T->right)<=0)

T=RR(T);

else

T=RL(T);

}

else

{

//data to be deleted is found

if(T->right!=NULL)

{ //delete its inorder succesor

p=T->right;

while(p->left!= NULL)

p=p->left;

T->data=p->data;

T->right=Delete(T->right,p->data);

if(BF(T)==2)//Rebalance during windup

if(BF(T->left)>=0)

T=LL(T);

else

T=LR(T);\

}

else

return(T->left);

}

T->ht=height(T);

return(T);

}

int height(node \*T)

{

int lh,rh;

if(T==NULL)

return(0);

if(T->left==NULL)

lh=0;

else

lh=1+T->left->ht;

if(T->right==NULL)

rh=0;

else

rh=1+T->right->ht;

if(lh>rh)

return(lh);

return(rh);

}

node \* rotateright(node \*x)

{

node \*y;

y=x->left;

x->left=y->right;

y->right=x;

x->ht=height(x);

y->ht=height(y);

return(y);

}

node \* rotateleft(node \*x)

{

node \*y;

y=x->right;

x->right=y->left;

y->left=x;

x->ht=height(x);

y->ht=height(y);

return(y);

}

node \* RR(node \*T)

{

T=rotateleft(T);

return(T);

}

node \* LL(node \*T)

{

T=rotateright(T);

return(T);

}

node \* LR(node \*T)

{

T->left=rotateleft(T->left);

T=rotateright(T);

return(T);

}

node \* RL(node \*T)

{

T->right=rotateright(T->right);

T=rotateleft(T);

return(T);

}

int BF(node \*T)

{

int lh,rh;

if(T==NULL)

return(0);

if(T->left==NULL)

lh=0;

else

lh=1+T->left->ht;

if(T->right==NULL)

rh=0;

else

rh=1+T->right->ht;

return(lh-rh);

}

void preorder(node \*T)

{

if(T!=NULL)

{

printf("%d(Bf=%d)",T->data,BF(T));

preorder(T->left);

preorder(T->right);

}

}

void inorder(node \*T)

{

if(T!=NULL)

{

inorder(T->left);

printf("%d(Bf=%d)",T->data,BF(T));

inorder(T->right);

}

}

**Output**

*1)Create:*  
*2)Insert:*  
*3)Delete:*  
*4)Print:*  
*5)Quit:*

*Enter Your Choice:1*

*Enter no. of elements:4*

*Enter tree data:7 12 4 9*

*1)Create:*  
*2)Insert:*  
*3)Delete:*  
*4)Print:*  
*5)Quit:*

*Enter Your Choice:4*

*Preorder sequence:*  
*7(Bf=-1)4(Bf=0)12(Bf=1)9(Bf=0)*

*Inorder sequence:*  
*4(Bf=0)7(Bf=-1)9(Bf=0)12(Bf=1)*

*1)Create:*  
*2)Insert:*  
*3)Delete:*  
*4)Print:*  
*5)Quit:*

*Enter Your Choice:3*

*Enter a data:7*

*1)Create:*  
*2)Insert:*  
*3)Delete:*  
*4)Print:*  
*5)Quit:*

*Enter Your Choice:4*

*Preorder sequence:*  
*9(Bf=0)4(Bf=0)12(Bf=0)*

*Inorder sequence:*  
*4(Bf=0)9(Bf=0)12(Bf=0)*

*1)Create:*  
*2)Insert:*  
*3)Delete:*  
*4)Print:*  
*5)Quit:*

*Enter Your Choice:5*

8 Implementation of Heaps using Priority Queues.

AIM

To write a C program for the implementation of heaps using priority queues.

ALGORITHM:

Step: Start the program

Step2: Declare the necessary variables

Step3: Write the functions for Insert, Delete, Display and Exit

Step4: Do the deletion for the higher order element and display it.

Step5: End the program

PROGRAM

/\* C program to implement Priority Queue using heap \*/

#include<stdio.h>

#include<math.h>

#define MAX 100/\*Declaring the maximum size of the queue\*/

void swap(int\*,int\*);

main()

{

int choice,num,n,a[MAX],data,s;

void display(int[],int);

void insert(int[],int,int,int);

int del\_hi\_priori(int[],int,int);

int lb=0;/\*Lower bound of the array is initialized to 0\*/

clrscr();

n=0;

while(1)

{

printf(".....MAIN MENU.....\n");

printf("\n1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:/\*choice to accept an elemnt and insert it in the queue\*/

printf("Enter data to be inserted : ");

scanf("%d",&data);

insert(a,n,data,lb);

n++;

break;

case 2:

s=del\_hi\_priori(a,n+1,lb);

if(s!=0)

printf("\nThe deleted value is : %d \n",s);

if(n>0)

n--;

break;

case 3:/\*choice to display the elements of the queue\*/

printf("\n");

display(a,n);

break;

case 4:/\*choice to exit from the program\*/

return;

default:

printf("Invalid choice.n");

}

printf("\n\n");

}

}

/\*This function inserts an element in the queue\*/

void insert(int a[],int heapsize,int data,int lb)

{

int i,p;

int parent(int);

if(heapsize==MAX)

{

printf("Queue Is Full!!\n");

return;

}

i=lb+heapsize;

a[i]=data;

while(i>lb&&a[p=parent(i)]<a[i])

{

swap(&a[p],&a[i]);

i=p;

}

}

/\*This function deletes an element from the queue\*/

int del\_hi\_priori(int a[],int heapsize,int lb)

{

int data,i,l,r,max\_child,t;

int left(int);

int right(int);

if(heapsize==1)

{

printf("Queue Is Empty!!\n");

return 0;

}

t=a[lb];

swap(&a[lb],&a[heapsize-1]);

i=lb;

heapsize--;

while(1)

{

if((l=left(i))>=heapsize)

break;

if((r=right(i))>=heapsize)

max\_child=l;

else

max\_child=(a[l]>a[r])?l:r;

if(a[i]>=a[max\_child])

break;

swap(&a[i],&a[max\_child]);

i=max\_child;

}

return t;

}

/\*Returns parent index\*/

int parent(int i)

{

float p;

p=((float)i/2.0)-1.0;

return ceil(p);

}

/\*Returns leftchild index\*/

int left(int i)

{

return 2\*i+1;

}

/\*Returns rightchild index\*/

int right(int i)

{

return 2\*i+2;

}

/\*This function displays the queue\*/

void display(int a[],int n)

{

int i;

if(n==0)

{

printf("Queue Is Empty!!\n");

return;

}

for(i=0;i<n;i++)

printf("%d ",a[i]);

printf("\n");

}

/\*This function is used to swap two elements\*/

void swap(int\*p,int\*q)

{

int temp;

temp=\*p;

\*p=\*q;

\*q=temp;

}

OUTPUT:  
  
.....MAIN MENU.....  
  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 1  
Enter data to be inserted : 52  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 1  
Enter data to be inserted : 63  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 1  
Enter data to be inserted : 45  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 1  
Enter data to be inserted : 2  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 1  
Enter data to be inserted : 99  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 3  
99 63 45 2 52  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 2  
The deleted value is : 99  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 3  
63 52 45 2  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 2  
The deleted value is : 63  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 2  
The deleted value is : 52  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 3  
45 2  
  
.....MAIN MENU.....  
1.Insert.  
2.Delete.  
3.Display.  
4.Quit.  
Enter your choice : 4

RESULT:

Thus the program for the implementation of heaps using priority queues has been executed successfully.

**9 Graph representation and Traversal algorithms**

**AIM:**

To write a C program to implement graph representation(Adjacency matrix) and traversal algorithm(BFS&DFS).

**ALGORITHM:**

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

The BFS algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

**PROGRAM:**

#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];

int delete();

void add(int item);

void bfs(int s,int n);

void dfs(int s,int n);

void push(int item);

int pop();

void main()

{

int n,i,s,ch,j;

char c,dummy;

printf("ENTER THE NUMBER VERTICES ");

scanf("%d",&n);

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

{

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

{

printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);

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

}

}

printf("THE ADJACENCY MATRIX IS\n");

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

{

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

{

printf(" %d",a[i][j]);

}

printf("\n");

}

do

{

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

vis[i]=0;

printf("\nMENU");

printf("\n1.B.F.S");

printf("\n2.D.F.S");

printf("\nENTER YOUR CHOICE");

scanf("%d",&ch);

printf("ENTER THE SOURCE VERTEX :");

scanf("%d",&s);

switch(ch)

{

case 1:bfs(s,n);

break;

case 2:

dfs(s,n);

break;

}

printf("DO U WANT TO CONTINUE(Y/N) ? ");

scanf("%c",&dummy);

scanf("%c",&c);

}while((c=='y')||(c=='Y'));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*BFS(breadth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bfs(int s,int n)

{

int p,i;

add(s);

vis[s]=1;

p=delete();

if(p!=0)

printf(" %d",p);

while(p!=0)

{

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

if((a[p][i]!=0)&&(vis[i]==0))

{

add(i);

vis[i]=1;

}

p=delete();

if(p!=0)

printf(" %d ",p);

}

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

if(vis[i]==0)

bfs(i,n);

}

void add(int item)

{

if(rear==19)

printf("QUEUE FULL");

else

{

if(rear==-1)

{

q[++rear]=item;

front++;

}

else

q[++rear]=item;

}

}

int delete()

{

int k;

if((front>rear)||(front==-1))

return(0);

else

{

k=q[front++];

return(k);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DFS(depth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void dfs(int s,int n)

{

int i,k;

push(s);

vis[s]=1;

k=pop();

if(k!=0)

printf(" %d ",k);

while(k!=0)

{

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

if((a[k][i]!=0)&&(vis[i]==0))

{

push(i);

vis[i]=1;

}

k=pop();

if(k!=0)

printf(" %d ",k);

}

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

if(vis[i]==0)

dfs(i,n);

}

void push(int item)

{

if(top==19)

printf("Stack overflow ");

else

stack[++top]=item;

}

int pop()

{

int k;

if(top==-1)

return(0);

else

{

k=stack[top--];

return(k);

}

}

OUTPUT:

ENTER THE NUMBER VERTICES 2

ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0 1

ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0 1

ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0 1

ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0 1

THE ADJACENCY MATRIX IS

1 1

1 1

MENU

1.B.F.S

2.D.F.S

ENTER YOUR CHOICE1

ENTER THE SOURCE VERTEX :1

1 2 DO U WANT TO CONTINUE(Y/N) ? y

MENU

1.B.F.S

2.D.F.S

ENTER YOUR CHOICE2

ENTER THE SOURCE VERTEX :2

2 1 DO U WANT TO CONTINUE(Y/N) ? n

RESULT:

Thus the C program to implement graph representation (Adjacency matrix) and traversal algorithm (BFS&DFS) has been executed successfully.

10 APPLICATION OF GRAPH (TOPOLOGICAL SORT)

AIM:

To write a C program to implement topological sort.

ALGORITHM:

Step1: Include all the header files.

Step2: Get the number of vertices.

Step3: Get the edge weights for each vertex.

Step4: Print the adjacent vertices and topological order after sorting.

Step5: End

PROGRAM

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

void create\_graph();

void add(int vertex);

int del();

int isEmpty();

int find\_indegree\_of\_vertex(int vertex);

int total\_vertices;

int adjacent\_matrix[MAX][MAX];

int queue[MAX];

int front = -1;

int rear = -1;

int main()

{

int i, vertex, count, topological\_sort[MAX], indegree[MAX];

create\_graph();

for(i = 0; i < total\_vertices; i++)

{

indegree[i] = find\_indegree\_of\_vertex(i);

if(indegree[i] == 0)

{

add(i);

}

}

count = 0;

while(!isEmpty() && count < total\_vertices)

{

vertex = del();

topological\_sort[++count] = vertex;

for(i = 0; i < total\_vertices; i++)

{

if(adjacent\_matrix[vertex][i] == 1)

{

adjacent\_matrix[vertex][i] = 0;

indegree[i] = indegree[i] - 1;

if(indegree[i] == 0)

{

add(i);

}

}

}

}

if(count < total\_vertices)

{

printf("Graph is Cyclic. Therefore, Topological Ordering Not Possible\n");

exit(1);

}

printf("Topological Order of Vertices\n");

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

{

printf("%3d", topological\_sort[i]);

}

printf("\n");

return 0;

}

void add(int vertex)

{

if(rear == MAX - 1)

{

printf("Queue Overflow\n");

}

else

{

if(front == -1)

{

front = 0;

}

rear = rear + 1;

queue[rear] = vertex ;

}

}

int isEmpty()

{

if(front == -1 || front > rear)

{

return 1;

}

else

{

return 0;

}

}

int del()

{

int element;

if(front == -1 || front > rear)

{

printf("Queue Underflow\n");

exit(1);

}

else

{

element = queue[front];

front = front + 1;

return element;

}

}

int find\_indegree\_of\_vertex(int vertex)

{

int count, total\_indegree = 0;

for(count = 0; count < total\_vertices; count++)

{

if(adjacent\_matrix[count][vertex] == 1)

{

total\_indegree++;

}

}

return total\_indegree;

}

void create\_graph()

{

int count, maximum\_edges, origin\_vertex, destination\_vertex;

printf("Enter number of vertices:\t");

scanf("%d", &total\_vertices);

maximum\_edges = total\_vertices \* (total\_vertices - 1);

for(count = 1; count <= maximum\_edges; count++)

{

printf("Enter Edge [%d] co-ordinates (-1 -1 to quit)\n", count);

printf("Enter Origin Vertex:\t");

scanf("%d", &origin\_vertex);

printf("Enter Destination Vertex:\t");

scanf("%d", &destination\_vertex);

if((origin\_vertex == -1) && (destination\_vertex == -1))

{

break;

}

if(origin\_vertex >= total\_vertices || destination\_vertex >= total\_vertices || origin\_vertex < 0 || destination\_vertex < 0)

{

printf("Edge Co-ordinates are Invalid\n");

count--;

}

else

adjacent\_matrix[origin\_vertex][destination\_vertex] = 1;

}

}

OUTPUT

Enter number of vertices: 4

Enter Edge [1] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: 0

Enter Destination Vertex: 1

Enter Edge [2] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: 0

Enter Destination Vertex: 2

Enter Edge [3] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: 1

Enter Destination Vertex: 3

Enter Edge [4] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: 1

Enter Destination Vertex: 2

Enter Edge [5] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: 2

Enter Destination Vertex: 3

Enter Edge [6] co-ordinates (-1 -1 to quit)

Enter Origin Vertex: -1

Enter Destination Vertex: -1

Topological Order of Vertices

0 1 2 3

RESULT:

10A SEARCHING-LINEAR AND BINARY SEARCH

PROGRAM

#include <stdio.h>

/\* Function for sequential search \*/

void sequential\_search(int array[], int size, int n)

{

int i;

for (i = 0; i < size; i++)

{

if (array[i] == n)

{

printf("%d found at location %d.\n", n, i+1);

break;

}

}

if (i == size)

printf("Not found! %d is not present in the list.\n", n);

}

/\* End of sequential\_search() \*/

/\* Function for binary search \*/

void binary\_search(int array[], int size, int n)

{

int i, first, last, middle;

first = 0;

last = size - 1;

middle = (first+last) / 2;

while (first <= last) {

if (array[middle] < n)

first = middle + 1;

else if (array[middle] == n) {

printf("%d found at location %d.\n", n, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last) / 2;

}

if ( first > last )

printf("Not found! %d is not present in the list.\n", n);

}

/\* End of binary\_search() \*/

/\* The main() begins \*/

int main()

{

int a[200], i, j, n, size;

printf("Enter the size of the list:");

scanf("%d", &size);

printf("Enter %d Integers in ascending order\n", size);

for (i = 0; i < size; i++)

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

printf("Enter value to find\n");

scanf("%d", &n);

printf("Sequential search\n");

sequential\_search(a, size, n);

printf("Binary search\n");

binary\_search(a, size, n);

return 0;

}

OUTPUT

Enter the size of the list:10

Enter 10 Integers in ascending order

2 5 8 12 16 25 31 45 48 55

Enter value to find

16

sequential search

16 found at location 5.

Binary search

16 found at location 5.

10 SORTING(INSERTION,SELECTION,BUBBLE,SHELL,RADIX SORT)

* 1. INSERTION SORT

AIM:

To write a C program for sorting an array of N numbers using insertion sort.

ALGORITHM:

1. Start the program
2. Initialize arr[max],i,j,tmp,n,p
3. Read the number of elements from the user
4. Read the elements
5. Display the unsorted elements to the user
6. Invoke function call insertionsort(arr,n)
   1. In function block, using for loop,for(p=1;p<n;p++)
   2. Initialize tmp=a[p]
   3. for(j=p;j>0&&a[j-1]>tmp;j--),if it is true go to next step, else go to step 6.5
   4. Calculate a[j]=a[j-1]
   5. Calculate a[j]=tmp
   6. Return back to main function
7. Display sorted list to the user
8. Stop the program

PROGRAM:

#include<stdio.h>

#include<conio.h>

void insertionsort(int[],int);

#define max 20

void main()

{

int arr[max],i,j,tmp,n,p;

clrscr();

printf(“\n enter the size of the sorting”);

scanf(“%d”,&n);

for(i=0;i<n;i++)

{

printf(“\n Enter elements %d”,i+1);

scanf(“%d”,&arr[i]);

}

printf(“unsorted list are:”);

for(i=0;i<n;i++)

printf(“%d \t”,arr[i]);

insertionsort(arr,n);

printf(“sorted list:”);

for(i=0;i<n;i++)

printf(“%d\t”,arr[i]);

getch();

}

void insertionsort(int a[],int n)

{

int p,j,tmp;

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

{

tmp=a[p];

for(j=p;j>0&&a[j-1]>tmp;j--)

a[j]=a[j-1];

a[j]=tmp;

}

}

OUTPUT:

Enter the number of elements: 5

Enter element 1: 8

Enter element 2: 5

Enter element 3: 1

Enter element 4: 3

Enter element 5: 4

Unsorted list are: 8 5 1 3 4

Sorted list are: 1 3 4 5 8

* 1. SELECTION SORT

AIM:

To write a C program for sorting an array of N numbers using selection sort.

ALGORITHM:

1. Start the program
2. Initialize a[max],i,j,n
3. Read the number of elements from the user
4. Read the elements
5. Display the unsorted elements to the user
6. Invoke function call selectionsort(a,n)

6.1 In function block, using for loop for(i=0;i<n-1;i++)

6.2 index=i;

6.3 using for loop, for(j=i+1;j<n;j++)

6.4 Check the condition,if(a[j]<a[index]), if it is true go to next step else go to step 6.6

6.5 index=j;

6.6 Calculate

smallno=a[index];

a[index]=a[i];

a[i]=smallno;

* 1. Return back to main function

1. Display sorted list to the user
2. Stop the program

PROGRAM:

#include<stdio.h>

#include<conio.h>

void selectionsort(int[],int);

#define max 20

void main()

{

int a[max],i,j,n;

clrscr();

printf(“\n enter the size of the sorting”);

scanf(“%d”,&n);

for(i=0;i<n;i++)

{

printf(“\n Enter elements %d”,i+1);

scanf(“%d”,&a[i]);

}

printf(“unsorted list are:”);

for(i=0;i<n;i++)

printf(“%d \t”,a[i]);

selectionsort(a,n);

printf(“sorted list:”);

for(i=0;i<n;i++)

printf(“%d\t”,a[i]);

getch();

}

void selectionsort(int a[],int n)

{

int i,j,index,smallno;

for(i=0;i<n-1;i++)

{

index=i;

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

if(a[j]<a[index])

index=j;

smallno=a[index];

a[index]=a[i];

a[i]=smallno;

}

}

OUTPUT:

Enter the number of elements: 5

Enter element 1: 8

Enter element 2: 5

Enter element 3: 1

Enter element 4: 3

Enter element 5: 4

Unsorted list are: 8 5 1 3 4

Sorted list are: 1 3 4 5 8

* 1. BUBBLE SORT

AIM:

To write a C program for sorting an array of N numbers using bubble sort.

ALGORITHM:

1. Start the program
2. Initialize i,j,n,temp,a[max]
3. Read the number of elements from the user
4. Read the elements
5. Display the unsorted elements to the user
6. Invoke function call bubblesort(a,n)
   1. Using for loop, for(i=0;i<n-1;i++)
   2. Using for loop,for(j=0;j<n-1-i;j++)
   3. Check the condition, if(a[j]>a[j+1]), if it is true perform the following

6.3.1 Calculate

temp=a[j]

a[j]=a[j+1]

a[j+1]=temp

* 1. Return back to main function

1. Display sorted list to the user
2. Stop the program

PROGRAM:

#include<stdio.h>

#include<conio.h>

void bubblesort(int[],int);

#define max 20

void main()

{

int i,j,n,temp,a[max];

clrscr();

printf(“\n enter the size of the sorting”);

scanf(“%d”,&n);

for(i=0;i<n;i++)

{

printf(“\n Enter elements %d”,i+1);

scanf(“%d”,&a[i]);

}

printf(“unsorted list are:”);

for(i=0;i<n;i++)

printf(“%d \t”,a[i]);

bubblesort(a,n);

printf(“sorted list:”);

for(i=0;i<n;i++)

printf(“%d\t”,a[i]);

getch();

}

void bubblesort(int a[],int n)

{

int i,j,tmp;

for(i=0;i<n-1;i++)

for(j=0;j<n-1-i;j++)

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

OUTPUT:

Enter the number of elements: 5

Enter element 1: 8

Enter element 2: 5

Enter element 3: 1

Enter element 4: 3

Enter element 5: 4

Unsorted list are: 8 5 1 3 4

Sorted list are: 1 3 4 5 8

* 1. SHELL SORTING

PROGRAM

#include<stdio.h>

#include<conio.h>

Void shelsort(int[],int)

#define max 20

Void main()

{

int arr[max],i,j,temp,n,p;

clrscr();

printf(“\n enter the size of the sorting”);

scanf(“%d”,&n);

for(i=0;i<n;i++)

{

printf(“\n enter elements %d”,i+1);

scanf(“%d”,&arr[i]);

}

printf(“unsorted list are:”);

for(i=0;i<n;i++)

printf(“%d\t”,arr[i]);

shellsort(arr,n);

printf(“sorted list:”);

for(i=0;i<n;i++)

printf(“%d\t”,arr[i]);

getch();

}

void shellsort(int a[],int n)

{

int i,j,increment,tmp;

for(incremenr=n/2;increment.0;increment/=2)

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

{

tmp=a[i];

for(j=I;j>=increment;j=increment)

if(tmp<a[j-increment])

a[j]=a[j-increment];

else

break;

a[j]=tmp;

}

}

OUTPUT

Enter the number of elements : 5

Enter the element 1: 8

Enter the element 2: 5

Enter the element 3: 1

Enter the element 4: 3

Enter the element 5: 4

Unsorted list are: 8 5 1 3 4

Sorted list : 1 3 4 5 8

* 1. RADIX SORT

#include<stdio.h>

#include<conio.h>

radix\_sort(int array[], int n);

void main()

{

 int array[100],n,i;

 clrscr();

 printf("Enter the number of elements to be sorted: ");

 scanf("%d",&n);

 printf("\nEnter the elements to be sorted: \n");

 for(i = 0 ; i < n ; i++ )

 {

  printf("\tArray[%d] = ",i);

  scanf("%d",&array[i]);

 }

 printf("\nArray Before Radix Sort:");  //Array Before Radix Sort

 for(i = 0; i < n; i++)

 {

  printf("%8d", array[i]);

 }

 printf("\n");

 radix\_sort(array,n);

 printf("\nArray After Radix Sort: ");  //Array After Radix Sort

 for(i = 0; i < n; i++)

 {

  printf("%8d", array[i]);

 }

 printf("\n");

 getch();

}

radix\_sort(int arr[], int n)

{

 int bucket[10][5],buck[10],b[10];

 int i,j,k,l,num,div,large,passes;

 div=1;

 num=0;

 large=arr[0];

 for(i=0 ; i<n ; i++)

 {

  if(arr[i] > large)

   {

    large = arr[i];

   }

  while(large > 0)

  {

   num++;

   large = large/10;

  }

  for(passes=0 ; passes<num ; passes++)

  {

   for(k=0 ; k<10 ; k++)

   {

    buck[k] = 0;

   }

   for(i=0 ; i<n  ;i++)

   {

    l = ((arr[i]/div)%10);

    bucket[l][buck[l]++] = arr[i];

   }

   i=0;

   for(k=0 ; k<10 ; k++)

   {

    for(j=0 ; j<buck[k] ; j++)

    {

     arr[i++] = bucket[k][j];

    }

   }

   div\*=10;

  }

 }

}

OUTPUT:

Enter the number of elements to be sorted: 5

Enter the elements to be sorted:

Array[0] = 76

Array[1] = 23

Array[2] = 89

Array[3] = 54

Array[4] = 90

Array Before Radix Sort: 76 23 89 54 90

Array After Radix Sort: 23 54 76 89 90