lOMoARcPSD|43337963

[CS3311 - DATA Structures LAB Manual](https://www.studocu.com/en-us/document/rvs-institute-of-management-studies/business-administration/cs3311-data-structures-lab-manual/48574828?utm_campaign=shared-document&utm_source=studocu-document&utm_medium=social_sharing&utm_content=cs3311-data-structures-lab-manual)

**UNIVERSITY COLLEGE OF ENGINEERING PATTUKKOTTAI**

**RAJAMADAM**

**CS3311- DATA STRUCTURES LABORATORY**

**NAME : ………………………………**

**REGNO : ………………………………**

**DEGREE & SEMESTER : ………………………………**

**YEAR & BRANCH : ………………………………**

**UNIVERSITY COLLEGE OF ENGINEERING PATTUKKOTTAI**

**RAJAMADAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**BONAFIDE CERTIFICATE**

**This is to certify that this ………..............................................................**

**........................................................... (Laboratory Name) record work done**

**by Mr/Ms ........................................................................... for the course B.E**

**Computer Science and Engineering during (Year/Semester)**

**of academic year ................................ is Bonafide.**

# Staff in-charge Head of the Department

**University Register Number : ...........................................**

**This record is submitted for ……....................... Semester B.E.**

**Practical Examination of Anna University conducted on .......................**

# INTERNAL EXAMINER EXTERNAL EXAMINER

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Name of the Exercise** | **Page No.** | **Marks** | **Signature** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Date** | **Name of the Exercise** | **Page No.** | **Marks** | **Signature** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## CS3311 DATA STRUCTURES LABORATORY

|  |  |  |  |
| --- | --- | --- | --- |
| **L** | **T** | **P** | **C** |
| **0** | **0** | **3** | **1.5** |

**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.

## LIST OF EXERCISES:

1. Array implementation of Stack, Queue and Circular Queue ADTs
2. Implementation of Singly Linked List
3. Linked list implementation of Stack and Linear Queue ADTs
4. Implementation of Polynomial Manipulation using Linked list
5. Implementation of Evaluating Postfix Expressions, Infix to Postfix conversion
6. Implementation of Binary Search Trees
7. Implementation of AVL Trees
8. Implementation of Heaps using Priority Queues
9. Implementation of Dijkstra’s Algorithm
10. Implementation of Prim’s Algorithm
11. Implementation of Linear Search and Binary Search
12. Implementation of Insertion Sort and Selection Sort
13. Implementation of Merge Sort
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing)

**TOTAL:** 45 PERIODS

## 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.

## ARRAY IMPLEMENTATION OF STACK

**AIM:**

To write a ‘C’ PROGRAM to implement the stack using array.

## ALGORITHM:

* 1. Start the program
  2. According to switch case the operations are selected.
  3. In case 1, data are added using PUSH operation.
  4. In case 2, data are deleted from stack using POP operation.
  5. In case 3, elements of array in stack are displayed using display Function.
  6. End the program.

## PROCEDURE FOR PUSH: -

* + 1. Initialize pointer variable and assign top is equal to -1
    2. Get the data to be pushed from user.
    3. If the pointer head IS pointed to maximum of Stack or above the maximum.
    4. Print Stack is full.
    5. Otherwise Increment the top pointer.
    6. Put the data into the top of the array of stack.
    7. End the Program.

## PROCEDURE FOR POP: -

* + 1. Initialize the pointer variable.
    2. Check weather Stack is full or not.
    3. IF top is equal to 0, Print “Stack is EMPTY”
    4. Otherwise, top element of array is assigned to item.
    5. Decrement the top pointer
    6. Print the deleted data
    7. End POP program

## PROCEDURE FOR DISPLAY: -

* + 1. Initialize Pointer variable
    2. By using for loop top is initializing to i Print data[i] until I >= condition is False
    3. When every looping process I is decremented in for loop
    4. End DISPLAY Program.
  1. Stop the Program.

## PROGRAM

#include<stdio.h> #include<conio.h> #include<stdlib.h> #define MAXSIZE 5 int stack[MAXSIZE]; int top=-1;

void push()

{

int element; if(top>=MAXSIZE-1)

{

printf("\nStack Overflow\n\n");

}

else

{

printf("\nEnter the Element to be inserted:\n"); scanf("%d",&element);

top=top+1; stack[top]=element;

}

}

void pop()

{

int item; if(top==-1)

{

printf("\nThe Stack is Empty\n\n");

}

else

{

item=stack[top]; top=top-1;

printf("\nThe Deleted Item is %d",item);

}

}

void print()

{

int i=0; if(top==-1)

{

printf("\nThe Stack is Empty\n\n");

}

else

{

while(i<=top)

{

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

}

}

}

void main()

{

int choice; char ch; clrscr();

while(choice!=5)

{

printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT"); printf("\nEnter Your Choice: "); scanf("%d",&choice);

switch(choice)

{

case 1:push(); break;

case 2:pop(); break;

case 3:print(); break;

case 4:exit(0); break; default:

printf("\n Wrong Choice");

}

getch();

}

}

## OUTPUT:

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter Your Choice: 1

Enter the Element to be inserted:

10

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter Your Choice: 1

Enter the Element to be inserted:

20

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter Your Choice: 3 10 20

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter Your Choice: 2

The Deleted Item is 20 1.PUSH

1. POP
2. DISPLAY
3. EXIT

Enter Your Choice: 3 10

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter Your Choice: 4

## RESULT:

Thus, the program for implementation of stack is executed and its output is verified.

## IMPLEMENTATION OF QUEUE USING ARRAY

**AIM:**

To write a ‘C’ PROGRAM to implement the queue using array.

## ALGORITHM:

* 1. Start the program
  2. According to switch case the operations are selected.
  3. In case 1, data are added using insert operation.
  4. In case 2, data are deleted from stack using delete operation.
  5. In case 3, elements of array in stack are displayed using display Function.
  6. End the program.

## PROGRAM:

#include<stdio.h> #include<stdlib.h> #define maxsize 5 void insert();

void delete(); void display();

int front = -1, rear = -1; int queue[maxsize]; void main ()

{

int choice; while(choice != 4)

{

printf("\n1.Insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\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(0); break; default:

printf("\nEnter valid choice??\n");

}

}

}

void insert()

{

int item;

printf("\nEnter the element:"); scanf("%d",&item);

if(rear == maxsize-1)

{

printf("\nOVERFLOW"); return;

}

if(front == -1 && rear == -1)

{

front = 0;

rear = 0;

}

else

{

rear = rear+1;

}

queue[rear] = item; printf("\nValue inserted ");

}

void delete()

{

int item;

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

{

printf("\nUNDERFLOW"); return;

}

else

{

item = queue[front]; if(front == rear)

{

front = -1; rear = -1 ;

}

else

{

front = front + 1;

}

printf("\nvalue deleted ");

}

}

void display()

{

int i;

if(rear == -1)

{

printf("\nEmpty queue");

}

else

{ printf("\nprinting values \n");

for(i=front;i<=rear;i++)

{

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

}

}

}

## OUTPUT:

1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:1 Enter the element:10 Value inserted 1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:1 Enter the element:20 Value inserted 1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:3 printing values .....

10 20

1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:2 value deleted 1.Insert an element 2.Delete an element

3.Display the queue 4.Exit

Enter your choice:3 printing values .....

20

1. Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:

## RESULT:

Thus, the program for implementation of queue is executed and its output is verified.

## ARRAY IMPLEMENTATION OF CIRCULAR QUEUE ADTS

**AIM:**

To write a ‘C’ PROGRAM to implement the array-based circular queue.

## ALGORITHM:

* 1. Start the program.
  2. Declare the necessary variables.
  3. Use a do while loop and switch statement to call the functions.
  4. Insert:
     1. Read the value to be inserted
     2. Increment rear and front if it is the first element.
     3. Else increment the rear by 1.
  5. Delete:
     1. Increment front by 1.
     2. Print the deleted elements.
  6. Display:
     1. If the both front=rear=1, print empty queue.
     2. Else use for loop varying from front to rear and print the elements.
  7. Stop the program.

## PROGRAM

#include<stdio.h> #include<conio.h> #include<stdlib.h> #define size 5

int queue[size]; int front= -1; int rear=0;

int qfull()

{

if(front==(rear+1)%size) return 1;

else return 0;

}

int qempty()

{

if(front==-1) return 1; else

return 0;

}

void add(int item)

{

if(qfull())

{

printf("\nThe Insert is full");

}

else

{

if(front==-1) front=rear=0; else

rear=(rear+1)%size; queue[rear]=item;

}

}

void delet()

{

int item; if(qempty())

{

printf("\nThe Insert doesn't have any item !"); printf("\nWe can't Delete items.\n");

}

else

{

item=queue[front];

if(front==rear)

{

front=rear=-1;

}

else

{

front=(front+1)%size;

}

printf("\nThe Deleted item is %d",item);

}

}

void display()

{

int i; if(qempty())

{

printf("\nThe Delete doesn't have any item !"); return;

}

printf("\nThe Element are:\n"); i=front;

while(i!=rear)

{

printf(" %d",queue[i]); i=(i+1)%size;

}

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

}

void main()

{

int choice,item; char ans; clrscr(); while(choice!=4)

{

printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit"); printf("\nEnter your choice:"); scanf("%d",&choice);

switch(choice)

{

case 1:printf("\nEnter the Element:"); scanf("%d",&item);

add(item); break;

case 2:delet(); break;

case 3:display(); break;

case 4:exit(0); break; default:

printf("\nWrong Choice\n"); break;

}

}

getch();

}

## OUTPUT:

1. Insert 2.Delete 3.Display 4.Exit

Enter your choice:1 Enter the Element:10 1.Insert

1. Delete 3.Display 4.Exit

Enter your choice:1

Enter the Element:20 1.Insert

2.Delete 3.Display 4.Exit

Enter your choice:3 The Element are:

10 20

1. Insert 2.Delete 3.Display 4.Exit

Enter your choice:2 The Deleted item is 10 1.Insert

1. Delete 3.Display 4.Exit

Enter your choice:3 The Element are:

20

1. Insert 2.Delete 3.Display 4.Exit

Enter your choice:4

## RESULT:

Thus, the program for circular using array is executed and its output is verified.

## SINGLY LINKED LIST

**AIM:**

To write a ‘C’ program to implement the singly linked list.

## ALGORTHIM:

* 1. Start the program
  2. Get the variables
  3. Create a head node and with help of the switch case option add delete display exit we going to perform this single linked list.
  4. Insertion:
     1. Enter the choice for add in the switch case.
     2. The function moves the calls to void add (), if given list is NULL the given data is added or else it’s shift the address to the next node.
  5. Deletion:
     1. Enter the choice for delete in the switch case.

1. The function moves the call to the void Del ().
2. Check whether the list is empty or not if empty mean” data not found” may

display.

1. Using of the while loop check the data which is present in the list.
   1. Display:
      1. Enter the choice for display in the switch case.
      2. The function moves the call to the void display ().
      3. Check whether the list is empty or not if empty mean” data not found” may

display.

* + 1. List is not equal to NULL than its display the element and move to next pointer for next element till condition fails.
  1. Stop the program and Exit

## PROGRAM

#include<stdio.h> #include<conio.h> #include<stdlib.h> #include<alloc.h> typedef struct node node; struct node

{

int data;

struct node \*link;

}\*head,\*temp,\*New,\*old; void display()

{

temp=head; if(temp==NULL)

{

printf("\nLinked list is Empty");

}

else

{

while(temp!=NULL)

{

printf("%d ",temp->data); temp=temp->link;

}

}

}

void insert()

{

int item;

printf("\nENTER THE NUMBER TO BE ADDED:");

scanf("%d",&item); if(head==NULL)

{

temp=(node \*)malloc(sizeof(node)); temp->data=item;

temp->link=NULL; head=temp;

}

else

{

temp=head;

while(temp->link!=NULL)

{

temp=temp->link;

}

New=(node \*)malloc(sizeof(node)); New->data=item;

New->link=NULL; temp->link=New;

}

}

void del()

{

int num,flag=0; temp=head;

printf("\nEnter the element to be deleted:"); scanf("%d",&num);

if(temp==NULL)

{

printf("\nLinked List is Empty");

}

else

{

while(temp!=NULL)

{

if(temp->data==num)

{

if(temp==head)

{

head=temp->link;

}

else

{

old->link=temp->link;

}

free(temp); flag=1;

}

old=temp; temp=temp->link;

}

if(flag==0)

{

printf("\nELEMENT IS NOT FOUND");

}

else

{

printf("\nELEMENT IS DELETED");

}

}

}

void main()

{

int ch; char cho; clrscr(); do

{

printf("\n1.Insert \n2.Delete \n3.Display \n4.Exit"); printf("\nEnter your Choice:");

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

{

case 1:insert(); break;

case 2:del(); break;

case 3:display(); break;

case 4:exit(0); break; default:

printf("\nINVALID CHOICE");

}

printf("\nDo u want to continue(y for YES/n for NO) :"); cho=getche();

}while(cho=='y'); getch();

}

# OUTPUT:

1. Insert 2.Delete 3.Display 4.Exit

Enter your Choice: 1

ENTER THE NUMBER TO BE ADDED: 20

Do u want to continue (y for YES/n for NO): y 1.Insert

1. Delete 3.Display 4.Exit

Enter your Choice: 1

ENTER THE NUMBER TO BE ADDED: 25

Do u want to continue (y for YES/n for NO): y 1.Insert

1. Delete
2. Display 4.Exit

Enter your Choice: 3 20 25

Do u want to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Display 4.Exit

Enter your Choice: 2

Enter the element to be deleted: 34 ELEMENT NOT FOUND

Do u want to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Display 4.Exit

Enter your Choice: 2

Enter the element to be deleted: 25 ELEMENT IS DELETED

Do u want to continue (y for YES/n for NO): y 1.Insert

1. Delete 3.Display 4.Exit

Enter your Choice: 3 20

Do u want to continue (y for YES/n for NO): N

## RESULT:

Thus, the program for implementation of singly linked list is executed and its output is verified.



## IMPLEMENTATION OF STACK USING LINKED LIST

**AIM:**

To write a C program to implement stack using array.

## ALGORITHM:

* 1. Start the program
  2. According to switch case the operations are selected.
  3. In case 1, data are added using PUSH operation.
  4. In case 2, data are deleted from stack using POP operation.
  5. In case 3, elements of array in stack are displayed using display Function.
  6. End the program.

## PROCEDURE FOR PUSH: -

1. Initialize pointer variable and assign top is equal to -1
2. Get the data to be pushed from user.
3. If the pointer head IS pointed to maximum of Stack or above the maximum.
4. Print Stack is full.
5. Otherwise Increment the top pointer.
6. Put the data into the top of the array of stack.
7. End the Program.

## PROCEDURE FOR POP: -

1. Initialize the pointer variable.
2. Check weather Stack is full or not.
3. IF top is equal to 0, Print “Stack is EMPTY”
4. Otherwise, top element of array is assigned to item.
5. Decrement the top pointer
6. Print the deleted data
7. End POP program

## PROCEDURE FOR DISPLAY: -

1. Initialize Pointer variable
2. By using for loop top is initializing to i Print data[i] until I >= condition is False
3. End DISPLAY Program.
4. Stop the Program.

## PROGRAM:

#include<stdio.h> void push();

void pop(); void display(); main()

{

int n;

printf("\tMENU\n1.PUSH\n2.POP \n3.DISPLAY\n4.EXIT"); do

{

printf("\nEnter your choice:"); scanf("%d",&n);

switch(n)

{

case 1:

push(); break; case 2:

pop(); break; case 3:

display(); break; case 4:

break; default:

printf("\nInvalid choice"); break;

}

}

while(n!=4);

}

typedef struct node

{

int data;

struct node \*link;

}n;

n \*top=NULL; void push()

{

int item; n \*temp;

printf("\nEnter the item:"); scanf("%d",&item); temp=(n\*)malloc(sizeof(n)); temp->data=item;

temp->link=top; top=temp;

}

void pop()

{

n \*temp; if(top==NULL)

printf("\nStack is empty"); else

{

temp=top;

printf("\nThe element deleted = %d",temp->data); free(temp);

top=top->link;

}

}

void display()

{

n \*save; if(top==NULL)

printf("\nStack is empty"); else

{

save=top;

printf("\nThe elements of the stack are :"); while(save!=NULL)

{

printf("%d\t",save->data); save=save->link;

}

printf("\nTopmost element = %d",top->data);

}

}

## OUTPUT:

MENU 1.PUSH

1. POP
2. DISPLAY
3. EXIT

Enter your choice:1 Enter the item:10 Enter your choice:1 Enter the item:20 Enter your choice:3

The elements of the stack are :20 10 Topmost element = 20

Enter your choice:2

The element deleted = 20 Enter your choice:3

The elements of the stack are :10 Topmost element = 10

Enter your choice: 4

## RESULT:

Thus, the program for implementation of stack is executed and its output is verified.

## IMPLEMENTATION OF QUEUE ADTS USING LINKED LIST

**AIM:**

To write a C program for the Queue Implementation.

## ALGORITHM:

* 1. Start the program.
  2. Declare the necessary variables.
  3. Use a do while loop and switch statement to call the functions.

## Insertion:

* + 1. **Step 1:** Allocate the space for the new node PTR
    2. **Step 2:** SET PTR -> DATA = VAL
    3. **Step 3:** IF FRONT = NULL SET FRONT = REAR = PTR

SET FRONT -> NEXT = REAR -> NEXT = NULL ELSE

SET REAR -> NEXT = PTR SET REAR = PTR

SET REAR -> NEXT = NULL [END OF IF]

* + 1. **Step 4:** END

## Delete:

* + 1. Step 1: IF FRONT = NULL Write " Underflow "

Go to Step 5 [END OF IF]

* + 1. Step 2: SET PTR = FRONT
    2. Step 3: SET FRONT = FRONT -> NEXT
    3. Step 4: FREE PTR
    4. Step 5: END

## Display:

* 1. Stop the program.

## PROGRAM:

#include<stdio.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("\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");

}

}

}

void insert()

{

struct node \*ptr; int item;

ptr = (struct node \*) malloc (sizeof(struct node)); if(ptr == NULL)

{

printf("\nOVERFLOW"); return;

}

else

{

printf("\nEnter value:"); 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"); return;

}

else

{

ptr = front;

front = front -> next; free(ptr);

}

printf("The Element is Deleted");

}

void display()

{

struct node \*ptr; ptr = front; if(front == NULL)

{

printf("\nEmpty queue");

}

else

{

printf("\nprinting values \n");

while(ptr != NULL)

{

printf("%d \t",ptr -> data); ptr = ptr -> next;

}}

}

## OUTPUT:

1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:1 Enter value:10 1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:1 Enter value:20 1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:3 printing values .....

10 20

1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:2 The Element is Deleted 1.Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:3 printing values .....

20

1. Insert an element 2.Delete an element 3.Display the queue 4.Exit

Enter your choice:4

## RESULT:

Thus, the program for simulation of queue using linked list implementation is executed and its output is verified.

## POLYNOMIAL MANIPULATION USING LINKED LIST

**AIM:**

To write a ‘C’ PROGRAM to represent a polynomial as a linked list and perform the addition

of polynomial.

## ALGORITHM:

* 1. Create Link polynomials
  2. Read the coefficients and exponent values of polynomials.
  3. Set the pointers P1 and P2 to both polynomials respectively to traverse them.
  4. Start from first node and compare the exponent of two polynomials
     1. If ( P1 -- exp = = P2 -- exp )

P3- coeff = P1 – coeff + P2 --- coeff P3 --- exp = P1 – exp

* + 1. If ( P1 -- exp < P2 -- exp ) P3 --- exp = P2 – exp;

P3- -- coeff = P2 --- coeff Move q to next node

* + 1. Else

P3 --- exp = P1 – exp:

P3- -- coeff = P1 --- coeff

* + 1. Move p to next node
  1. Append the remaining nodes of either of the polynomials to the resultant linked list.
  2. End

## PROGRAM

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

{

int coeff; int pow;

struct link \*next;

};

struct link \*poly1=NULL,\*poly2=NULL,\*poly=NULL; void create(struct link \*node)

{

char ch; int i=3;

while(i<=3&&i>=0)

{

printf("\nEnter the coefficient of X^%d :",i); scanf("%d",&node->coeff);

node->pow=i;

node->next=(struct link\*)malloc(sizeof(struct link)); node=node->next;

node->next=NULL; i--;

}

}

void show(struct link \*node)

{

while(node->next!=NULL)

{

printf("%dx^%d",node->coeff,node->pow); node=node->next;

if(node->next!=NULL) printf(" + ");

}

}

void polyadd(struct link \*poly1,struct link \*poly2,struct link \*poly)

{

while(poly1->next && poly2->next)

{

if(poly1->pow>poly2->pow)

{

poly->pow=poly1->pow; poly->coeff=poly1->coeff; poly1=poly1->next;

}

else if(poly1->pow<poly2->pow)

{

poly->pow=poly2->pow; poly->coeff=poly2->coeff; poly2=poly2->next;

}

else

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff+poly2->coeff; poly1=poly1->next;

poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link)); poly=poly->next;

poly->next=NULL;

}

while(poly1->next || poly2->next)

{

if(poly1->next)

{

poly->pow=poly1->pow; poly->coeff=poly1->coeff; poly1=poly1->next;

}

if(poly2->next)

{

poly->pow=poly2->pow; poly->coeff=poly2->coeff; poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link)); poly=poly->next;

poly->next=NULL;

}

}

void main()

{

char ch; do

{

clrscr();

poly1=(struct link \*)malloc(sizeof(struct link)); poly2=(struct link \*)malloc(sizeof(struct link)); poly=(struct link \*)malloc(sizeof(struct link)); printf("\nThe First Polynomial Number:"); create(poly1);

printf("\nThe Second Polynomial Number:"); create(poly2);

printf("\nResult\n\nThe First Polynomial Number:"); show(poly1);

printf("\nThe Second Polynomial Number:"); show(poly2);

polyadd(poly1,poly2,poly); printf("\nAdded polynomial Number:"); show(poly);

printf("\nDo u want to continue ("y" for YES "n" for NO):"); ch=getch();

}while(ch=='y');

}

## OUTPUT:

The First Polynomial Number:

Enter the coefficient of X^3 :2 Enter the coefficient of X^2 :5 Enter the coefficient of X^1 :1 Enter the coefficient of X^0 :7 The Second Polynomial Number:

Enter the coefficient of X^3 :7 Enter the coefficient of X^2 :2 Enter the coefficient of X^1 :3 Enter the coefficient of X^0 :5 Result

The First Polynomial Number: 2x^3 + 5x^2 + 1x^1 + 7x^0 The Second Polynomial Number: 7x^3 + 2x^2 + 3x^1 + 5x^0 Added polynomial Number: 9x^3 + 7x^2 + 4x^1 + 12x^0

Do u want to continue ("y" for YES "n" for NO): N

## RESULT:

Thus, the program for implementation of polynomial addition is executed and its output is verified.

## CONVERSION FROM INFIX TO POST EXPRESSION

**AIM:**

To write a ‘C’ PROGRAM to implement the conversion from infix to postfix expression

using stack.

## ALGORITHM:

Step 1: Read the expression from left to right.

Step 2: If the input symbol red is ‘(‘then push it on to the stack.

Step 3: If the input symbol read is an operand, then place it in the postfix expression. Step 4: If the input symbol read is an operator, then,

Check if precedence (stack operator) >= precedence (input operator) If so, remove all elements from stack and place it in postfix Otherwise, push the operator being read onto the stack.

Step 5: If the input symbol read is a closing parenthesis ‘)’ then,

pop all the elements from the stack, place them in postfix expression till ‘(’ is not popped.

Step 6: Finally print the postfix expression.

## PROGRAM

#include<stdio.h> #include<conio.h> #include<string.h> #define MAX 25 int s[MAX],top=-1;

char post[MAX],b[MAX]; int pre(char a)

{

int b; switch(a)

{

case '(':b=0; break;

case '+': case '-':b=1;

break;

case '\*':

case '/':b=2; break;

case '^':b=3; break;

}

return b;

}

int alphanum(char a)

{

return ((a>='a'&&a<='z')||(a>='A'&&a<='Z'));

}

int opr(char a)

{

int b=0; switch(a)

{

case '+':

case '-':

case '\*':

case '/':

case '^':

b=1;

break;

}

return b;

}

void intopost()

{

char a,in[25]; int i,ti=-1,tp=-1; s[++top]='(';

printf("\nEnter the Infix Expression:\n"); scanf("%s",&in);

while(in[++ti]!='\0')

{

if(alphanum(in[ti]))

{

post[++tp]=in[ti];

}

else if(opr(in[ti]))

{

while(pre(in[ti])<=pre(s[top]))

{

post[++tp]=s[top--];

}

s[++top]=in[ti];

}

else if(in[ti]=='(')

{

s[++top]='(';

}

else if(in[ti]==')')

{

while(s[top]!='(')

{

post[++tp]=s[top--];

}

top--;

}

else

{

printf("\nInvalid Expression\n");

}

}

while(s[top]!='(')

{

post[++tp]=s[top--];

}

post[++tp]='\0';

printf("\nPostFix Expression is :%s",post); for(i=0;i<strlen(post);i++)

{

b[i]=post[i];

}

}

void main()

{

char a,post[25]; clrscr(); intopost(); getch();

}

## OUTPUT:

Enter the Infix Expression:

(A+B)\*(C-D)

PostFix Expression is: AB+CD-\*

## RESULT:

Thus, the program for implementation of conversion from infix to postfix expression is executed and its output is verified

## EVALUATING POSTFIX EXPRESSIONS

**AIM:**

To write a C program for evaluating infix to postfix expression using array implementation

of Stack

## ALGORTHIM:

1. Start the program
2. Declare the variables.
3. Read the infix expression one character at a time
4. If the character is an operand push its associated value into the stack

## PUSH ()

* 1. Increment top by 1
  2. Assign the top to the stack [top].
  3. If the character is an operator pop two values from the stack.

## POP ()

* 1. Decrement the top by 1
  2. print the pop element
  3. Apply the operator to them and push the value into the stack.
  4. Using the switch statement, the operator is applied to the operand.
  5. Print the value of the postfix expression by popping the last value from the stack.

1. Stop the program.

## PROGRAM:

#include<stdio.h> #include<conio.h> #include<stdlib.h> #include<string.h> #include<math.h> #define size 80

/\*declaration of stack data structure\*/ struct stack

{

double s[size]; int top;

}st;

void main()

{

char exp[size]; int len;

double result; double post(); clrscr();

printf("enter the postfix expression\n"); scanf("%s",exp);

len=strlen(exp);

exp[len]='$';/\*append $ at the end as a endmarker\*/ result=post(exp);

printf("the value of the expression is %f\n",result); getch();

exit(0);

}

double post(char exp[])

{

char ch,\*type;

double result,val,op1,op2; void push(double); double pop();

int i; st.top=0; i=0;

ch=exp[i]; while(ch!='$')

{

if(ch>='0' && ch<='9') type="operand";

else if(ch=='+'||ch=='-'|| ch=='\*'||ch=='/'||ch=='^') type="operator";

if(strcmp(type,"operand")==0)

{

val=ch-48; push(val);

}

else

if(strcmp(type, "operator")==0)

{

op2=pop(); op1=pop(); switch(ch)

{

case '+': result=op1+op2; break;

case '-': result=op1-op2; break;

case '\*': result=op1\*op2; break;

case '/': result=op1/op2; break;

case '^': result=pow(op1,op2); break;

}

push(result);

} i++;

ch=exp[i];

}

result=pop(); return(result);

}

void push(double val)

{

if(st.top+1>=size) printf("\nstack is full\n"); st.top++;

st.s[st.top]=val;

}

double pop()

{

double val; if(st.top==-1)

printf("\nstack is empty\n"); val=st.s[st.top];

st.top--; return(val);

}

## OUTPUT:

Enter the postfix expression 12+34\*+

The value of the expression is 15.000000

## RESULT:

Thus, the program for implementation of conversion from infix to postfix expression is executed and its output is verified.

## BINARY SEARCH TREE

**AIM:**

To write a ‘C’ PROGRAM to implement the binary search tree.

## ALGORITHM:

1. Start the program.
2. Get the root node and initialized the pointer.
3. Get the Values of the child nodes.
4. Call the function “Create” to construct tree.
5. According to the switch statement the Traversals was selected.

## PROCEDURE – CREATE:

* 1. Use IF loop to check whether the mode is Null.
  2. Create a Node using malloc function.
  3. Store the value in the Node function.
  4. Make the right and left Pointers to Null.
  5. Else if (n>node-> data).
  6. Node->right = Create (node->right, n);
  7. Else if (n< node-> data)
  8. Node -> left = Create (node-> left, n);
  9. Return the node value to main program.

1. End the process.

## PROGRAM

#include<stdio.h> #include<conio.h> struct treenode;

typedef struct treenode \*position,\*searchtree,ptrtonode; struct treenode

{

int element;

struct treenode \*left; struct treenode \*right;

};

searchtree insert(int x,searchtree t)

{

if(t== NULL)

{

t=(ptrtonode\*)malloc(sizeof(struct treenode)); if(t== NULL)

{

printf("\nOut of Space");

}

else

{

t->element=x; t->left=NULL;

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 find(int x,searchtree t)

{

if(t== NULL)

{

return NULL;

}

else if(x<t->element)

{

return find(x,t->left);

}

else if(x>t->element)

{

return find(x,t->right);

}

else

{

return t;

}}

searchtree delet(int x,searchtree t)

{

position tmp; if(t== NULL)

{ }

else if(x<t->element)

{

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

}

else if(x>t->element)

{

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

}

else

{

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

{

tmp=find(x,t->right);

t->element=tmp->element;

t->right=delet(tmp->element,t->right);

}

else

{

tmp=t;

if(t->left== NULL)

{

t=t->right;

}

else if(t->right== NULL)

{

t=t->left;

}

free(tmp);

}}

return t;

}

void display(searchtree t)

{ if(t!=NULL)

{

display(t->left); printf("%d\t",t->element); display(t->right);

}}

void main()

{

int op,item; char ch; searchtree t; position p; clrscr(); t=NULL;

do

{

printf("\n1.Insert\n2.Delete\n3.Find\n4.Display"); printf("\nEnter your choice:");

scanf("%d",&op); switch(op)

{

case 1:printf("\nEnter the item to be inserted:"); scanf("%d",&item);

t=insert(item,t); printf("\nItem is inserted");

break;

case 2:printf("\nEnter the item to be deleted:"); scanf("%d",&item);

if(find(item,t))

{

t=delet(item,t); printf("\nItem is deleted");

}

else

{

printf("\nThe item is not found");

}

break;

case 3:printf("\nEnter the item to be found:"); scanf("%d",&item);

if(find(item,t))

{

printf("\nItem is found");

}

else

{

printf("\nThe item is not found");

}

break; case 4:if(t)

{

printf("The Tree is \n");

display(t);

}

else

{

printf("\nThe Tree is empty");

}

break; default:

printf("\nWrong Choice"); exit(1);

}

printf("\nDo u wish to continue (y for YES/n for NO):"); ch=getche();

}while(ch== 'y'); getch();

}

## OUTPUT:

1. Insert 2.Delete 3.Find 4.Display

Enter your choice: 1

Enter the item to be inserted: 2 Item is inserted

Do u wish to continue (y for YES/n for NO): y 1.Insert

1. Delete 3.Find 4.Display

Enter your choice: 1

Enter the item to be inserted: 4 Item is inserted

Do u wish to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Find 4.Display

Enter your choice: 1

Enter the item to be inserted: 6 Item is inserted

Do u wish to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Find 4.Display

Enter your choice: 4 The Tree is ....

2 4 6

Do u wish to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Find 4.Display

Enter your choice: 3

Enter the item to be found: 4 Item is found

Do u wish to continue (y for YES/n for NO): y 1.Insert

2.Delete 3.Find 4.Display

Enter your choice: 2

Enter the item to be deleted: 4 Item is deleted

Do u wish to continue (y for YES/n for NO): y 1.Insert

1. Delete 3.Find 4.Display

Enter your choice: 4 The Tree is ....

2 6

Do u wish to continue (y for YES/n for NO): n

## RESULT:

Thus, the Program for implementation of binary search tree is executed and its output is verified.

## AVL TREE

**AIM:**

To write a ‘C’ PROGRAM to implement the insertion in AVL tree.

## ALGORTHIM:

* 1. Start the program
  2. Get the variables to declare.
  3. Get the position of the height through left and as well as right.
  4. If the height doesn’t match with the balance factor make a single rotation. Through right

side.

* 1. Do the same step for left side too.
  2. If the balance factor doesn’t match the height means go for double rotation in left as well

as right.

* 1. Display the AVL tree.
  2. Stop the program.

## PROGRAM

#include<stdio.h> #include<conio.h> struct treenode;

typedef struct treenode \*position,\*avltree,ptrtonode; struct treenode

{

int element; avltree left; avltree right; int height;

};

int height(position p)

{ if(p!=NULL)

return p->height; else

return 0;

}

int max(int h1,int h2)

{

if(h1>h2) return h1; else return h2;

}

position singlelerightrotation(position k2)

{

position k1; k1=k2->left;

k2->left=k1->right; k1->right=k2;

k2->height=max(height(k2->left),height(k2->right))+1; k1->height=max(height(k1->left),height(k1->right))+1; return k1;

}

position singlerileftrotation(position k1)

{

position k2; k2=k1->right;

k1->right=k2->left; k2->left=k1;

k1->height=max(height(k1->left),height(k1->right))+1; k2->height=max(height(k2->left),height(k2->right))+1; return k2;

}

position doublelerightrotation(position k3)

{

k3->left=singlerileftrotation(k3->left); return singlerileftrotation(k3);

}

position doublerileftrotation(position k3)

{

k3->right=singlerileftrotation(k3->right); return singlerileftrotation(k3);

}

avltree insert(int x,avltree t)

{

if(t= = NULL)

{

t=(ptrtonode \*)malloc(sizeof(struct treenode)); if(t= = NULL)

{

printf("\nOut of space");

}

else

{

t->element=x; t->left=NULL;

t->right=NULL; t->height=0;

}

}

else if(x<t->element)

{

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

if(height(t->left)-height(t->right)= = 2) if(x<t->left->element) t=singlelerightrotation(t);

else t=doublelerightrotation(t);

}

else if(x>t->element)

{

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

if(height(t->right)-height(t->left)= = 2) if(x<t->right->element)

t=singlerileftrotation(t);

else t=doublerileftrotation(t);

}

1. >height=max(height(t->left),height(t->right))+1; return t;

}

void display(avltree root)

{

if(root!=NULL)

{

display(root->left);

if(root->element= = NULL)

{

printf("0");

}

else

{

printf("\n%d",root->element);

}

display(root->right);

}

}

void main()

{

int item; char ch; avltree t; position p; clrscr(); t=NULL;

do

{

printf("\nEnter the item to be inserted:"); scanf("%d",&item);

t=insert(item,t);

printf("\nThe item is inserted");

printf(“\nDo u want to insert the item again:(y for YES/n for NO)”);

ch=getche();

}while(ch= ='y');

printf("\nThe AVL Tree is ");

display(t); getch();

}

## OUTPUT:

Enter the item to be inserted:4 The item is inserted

Do u want to insert the item again:(y for YES/n for NO) y

Enter the item to be inserted:3 The item is inserted

Do u want to insert the item again:(y for YES/n for NO) y

Enter the item to be inserted:9 The item is inserted

Do u want to insert the item again:(y for YES/n for NO) y

Enter the item to be inserted:1 The item is inserted

Do u want to insert the item again:(y for YES/n for NO) y

Enter the item to be inserted:5 The item is inserted

Do u want to insert the item again:(y for YES/n for NO) n

The AVL Tree is .....

1

3

4

5

9

## RESULT:

Thus, the Program for implementation of insertion in AVL tree is executed and its output is verified.

## PRIORITY QUEUE USING HEAPS

**AIM:**

To write a ‘C’ PROGRAM to implement the priority queue using heaps.

## ALGORTHIM:

* 1. Start the program
  2. Declare the variables
  3. Check the condition of for qfull() and we can insert the elements if(rear==SIZE-1) we can insert the element
  4. Check the condition for qempty and delete the elements in the queue if(front==-1)||(front>rear)) we can delete the element.
  5. Check the condition for qempty and display the elements in the queue if(front==-1)||(front>rear)) we can dispaly the element.
  6. Stop the program

## PROGRAM

#include<stdio.h> #include<conio.h> #define SIZE 5

int rear= -1,front=0,que[SIZE],choice; void insert()

{

int item,j;

printf("\nEnter the Element: "); scanf("%d",&item);

if(front== -1)

{

front++;

}

j=rear; while(j>=0&&item<que[j])

{

que[j+1]=que[j]; j--;

}

que[j+1]=item; rear=rear+1;

}

int Qfull()

{

if(rear==SIZE-1)

{

return 1;

}

else

{

return 0;

}

}

void delet()

{

int item; item=que[front];

printf("\nThe item deleted is: %d",item); front++;

}

int Qempty()

{

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

{

return 1;

}

else

{

return 0;

}

}

void display()

{

int i;

printf("\nThe queue is:\n"); for(i=front;i<=rear;i++)

{

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

}

}

void main()

{

char ans; clrscr();

do

{

printf("\n Priority Queue"); printf("\n Main Menu");

printf("\n1.Insert\n2.Delete\n3.Display"); printf("\nEnter your Choice:"); scanf("%d",&choice);

switch(choice)

{

case 1:if(Qfull())

{

printf("\nQueue is Full");

}

else

{

insert();

}

break;

case 2:if(Qempty())

{

printf("\nQueue is Empty");

}

else

{

delet();

}

break;

case 3:if(Qempty())

{

printf("\nQueue is Empty");

}

else

{

display();

}

break; default:

printf("\nWrong Choice"); break;

}

printf("\nDo u want to continue?(y for YES/n for NO): "); ans=getche();

}while(ans=='y'); getch();

}

## OUTPUT:

Priority Queue Main Menu 1.Insert 2.Delete 3.Display

Enter your Choice1 Enter the Element: 40

Do u want to continue?(y for YES/n for NO): y Priority Queue

Main Menu 1.Insert

1. Delete 3.Display

Enter your Choice1 Enter the Element: 50

Do u want to continue?(y for YES/n for NO): y Priority Queue

Main Menu 1.Insert 2.Delete 3.Display

Enter your Choice1 Enter the Element: 10

Do u want to continue?(y for YES/n for NO): y Priority Queue

Main Menu 1.Insert 2.Delete 3.Display

Enter your Choice3 The queue is:

10

40

50

Do u want to continue?(y for YES/n for NO): y Priority Queue

Main Menu 1.Insert 2.Delete 3.Display

Enter your Choice2 The item deleted is 10

Do u want to continue?(y for YES/n for NO): y Priority Queue

Main Menu 1.Insert 2.Delete 3.Display

Enter your Choice3 The queue is:

40

50

## RESULT:

Thus, the Program for implementation of priority queue using heaps is executed and its output is verified.

## DIJKSTRA’S ALGORITHM

**AIM:**

To write a ‘C’ PROGRAM to implement the Dijkstra’s algorithm.

## ALGORITHM

Step1: Start the program.

Step2: Read the number of vertices

Step3: Read the weight of every pair of vertices. Step4: Get the source vertex & destination.

Step5: Construct the graph.

Step6: Start finding the start node to all other neighboring nodes.

Step7: Nearest path is selected using array until the end node is reached. Step8: Print the shortest path.

Step9: Terminate the program.

## PROGRAM

#include<stdio.h> #include<conio.h> #define INFINITY 9999

#define MAX 10

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

int G[MAX][MAX],i,j,n,u; clrscr();

printf("Enter 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);

dijkstra(G,n,u); getch();

}

void dijkstra(int G[MAX][MAX],int n,int startnode){ int cost[MAX][MAX],distance[MAX],pred[MAX]; int visited[MAX],count,mindistance,nextnode,i,j; for(i=0;i<n;i++)

for(j=0;j<n;j++) if(G[i][j]==0) cost[i][j]=INFINITY; else cost[i][j]=G[i][j]; for(i=0;i<n;i++){

distance[i]=cost[startnode][i]; pred[i]=startnode; visited[i]=0;

}

distance[startnode]=0; visited[startnode]=1; count=1; while(count<n-1){

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

if(distance[i]<mindistance&&!visited[i]){ mindistance=distance[i];

nextnode=i;

}

visited[nextnode]=1; for(i=0;i<n;i++) if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i]){ distance[i]=mindistance+cost[nextnode][i]; pred[i]=nextnode;

}

count++;

}

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

if(i!=startnode){

printf("\nDistance of node%d=%d \n",i,distance[i]); printf("\nPath=%d \n",i);

j=i; do{

j=pred[j]; printf("<-%d",j);

}while(j!=startnode);

}}

## OUTPUT:

Enter no. of vertices:4 Enter the adjacency matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 3 | 0 | 0 |
| 0 | 0 | 0 | 2 |
| 4 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |

Enter the starting node:0 Distance of node1=3 Path=1

<-0

Distance of node2=6 Path=2

<-3<-1<-0

Distance of node3=5 Path=3

<-1<-0

## RESULT:

Thus, the Program for implementation of Dijkstra ‘s algorithm is executed and its output is verified.

## PRIM’S ALGORITM

**AIM:**

To write a ‘C’ PROGRAM to implement the Prim’s algorithm.

## ALGORTHIM:

* 1. Start the program
  2. Declare the variables.
  3. Enter the edges and its edges in the graph.
  4. Construct the graph.
  5. Pick one arbitrary vertex and consider it as visited.
  6. The unvisited vertices in the new edge are considered.
  7. Repeat the step 6 until u cover all the edges.
  8. Print the minimum spanning with the corresponding weights.
  9. End the program.

## 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("\nEdge %d:(%d %d) cost:%d",ne++,a,b,min); mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\nMinimun cost=%d",mincost); getch();

}

## OUTPUT:

Enter the number of nodes:6 Enter the adjacency matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 4 | 4 | 0 | 0 | 0 |
| 4 | 0 | 2 | 0 | 0 | 0 |
| 4 | 2 | 0 | 3 | 2 | 4 |
| 0 | 0 | 3 | 0 | 0 | 3 |
| 0 | 0 | 2 | 0 | 0 | 3 |
| 0 | 0 | 4 | 3 | 3 | 0 |

Edge 1:(1 2) cost:4

Edge 2:(2 3) cost:2

Edge 3:(3 5) cost:2

Edge 4:(3 4) cost:3

Edge 5:(4 6) cost:3 Minimum cost=14

## RESULT:

Thus, the Program for implementation of Prim’s algorithm is executed and its output is verified.

## IMPLEMENTATION OF LINEAR SEARCH

**AIM:**

To write a ‘C’ program to implement linear search.

## ALGORITHM:

Step 1: Start the program

Step 2: Define function for linear search as

* + 1. Read the data to be searched ‘X’
    2. Scan the array from the left to right
    3. Compare ‘X’ with the first element
    4. If equal then

Print ‘The number is found’ and return

Else

Compare ‘X’ with second element and so on.

Step 3: Stop the program

## PROGRAM:

#include<stdio.h> int a[10],i,n,flag=0; void insert()

{

printf("Enter the size of an array: "); scanf("%d",&n);

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

{

printf("Enter the elements of the array %d:",i+1); scanf("%d",&a[i]);

}

}

void delet()

{

int del;

printf("Enter the number to be delete: "); scanf("%d",&del);

for(i=0;i<n;i++){ if(a[i]==del){

a[i]=-1;

flag=1; break;

}

}

if(flag==0)

printf("The number is not in the list"); else

printf("The number is deleted");

}

void display()

{

printf("\nThe Element are:"); for(i=0;i<n;i++)

{

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

}

}

void search()

{

int key;

printf("Enter the number to be search: "); scanf("%d",&key);

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

{

if(a[i]==key)

{

flag=1; break;

}

}

if(flag==0)

printf("The number is not in the list");

else

printf("The number is found");

}

int main()

{

int ch; while(ch!=5)

{

printf("\n1. Insert \n2. Delete \n3. Display \n4. Search\n5. Exit"); printf("\n Enter your Choice:");

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

{

case 1:

insert(); break; case 2:

delet(); break; case 3:

display(); break; case 4:

search(); break; case 5:

exit(0); break; default:

printf("\nInvalid Choice");

}

}

}

## OUTPUT:

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your Choice:1

Enter the size of an array: 5

Enter the elements of the array 1:10 Enter the elements of the array 2:20 Enter the elements of the array 3:30 Enter the elements of the array 4:40 Enter the elements of the array 5:50

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your Choice:2

Enter the number to be delete: 30 The number is deleted

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your Choice:3

The Element are:10 20 -1 40 50

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your Choice:4

Enter the number to be search: 50 The number is found

1. Insert
2. Delete
3. Display
4. Search
5. Exit

Enter your Choice:

## RESULT:

Thus, the C program to implement Linear Search was executed successfully and the output was verified.

## IMPLEMENTATION OF BINARY SEARCH

**AIM:**

To write a ‘C’ program to implement binary search.

## ALGORITHM:

Step 1: Start the program

Step 2: Define function for Binary Search as

* 1. Sort the array in ascending order
  2. Let lb=0 and ub=n-1
  3. Read the data to be searched ‘X’
  4. Find the mid position of the given array Mid=(lb+ub)/2
  5. Compare X with a[mid] If equal then

Goto step (g)

Else

If X less than a[mid] then ub=mid-1

If X greater than a[mid] then lb=mid+1

* 1. If lb<=ub

Repeat steps (d) and (e) for the sub array lb to ub Else

Goto step (g)

* 1. If(lb>ub)

Print “Search Success”

Else

Print “Search Failed”

* 1. Return Step3: Stop the program.

## PROGRAM

#include<stdio.h> int main()

{

int a[10],i,n,m,c=0,l,u,mid;

printf("\nEnter the size of an array: "); scanf("%d",&n);

printf("\nEnter the elements in ascending order"); for(i=0;i<n;i++){

printf("\nEnter the %d elements:", i+1); scanf("%d",&a[i]);

}

printf("\nEnter the number to be search: "); scanf("%d",&m);

l=0,u=n-1; while(l<=u){ mid=(l+u)/2; if(m==a[mid]){ c=1;

break;

}

else if(m<a[mid]){ u=mid-1;

}

else l=mid+1;

}

if(c==0)

printf("\nThe number is not found."); else

printf("\nThe number is found."); return 0;

}

## OUTPUT:

Enter the size of an array: 5

Enter the elements in ascending order Enter the 1 elements:10

Enter the 2 elements:20 Enter the 3 elements:30 Enter the 4 elements:40 Enter the 5 elements:50

Enter the number to be search: 50 The number is found.

## RESULT:

Thus, the C program to implement Binary Search was executed successfully and the output was verified.

## INSERTION SORT

**AIM:**

To write a ‘C’ program to implement insertion sort.

## ALGORITHM:

Step 1: Read the elements into the array

Step 2: Take the second element. Compare it with the first element. If the second element less than the first element interchange them

Step 3: Take the third element compare it first and second element and insert it in the correct position by shifting the elements in the array. So that the first, second and third elements are in sorted array

Step 4: In general, take the ith element and compare it with the all the elements before it and place it in the proper position by shifting the elements one position right.

Step 5: When the ith element is placed, the elements in the array from the 0th to the ith position will be in the sorted order

Step 6: The above process is continued for all the elements in the array. Step 7: Stop.

## PROGRAM:

#include<stdio.h> #include<conio.h> void insert(int[],int); void main(){

int a[20],i,n; clrscr();

printf("\nEnter the size of an array:"); scanf("%d",&n);

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

printf("\nEnter the %d element in the array:",i+1); scanf("%d",&a[i]);

}

insert(a,n);

getch();

}

void insert(int a[],int n){ int i,j,temp; for(i=1;i<n;i++){ temp=a[i];

for(j=i-1;j>=0;j--){ if(a[j]>temp){

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

}

else break;

}

a[j+1]=temp;

}

printf("\nData After Insertion sort\n"); for(i=0;i<n;i++)

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

}

## OUTPUT:

Enter the size of an array:5

Enter the 1 element in the array:8 Enter the 2 element in the array:2 Enter the 3 element in the array:6 Enter the 4 element in the array:4 Enter the 5 element in the array:1 Data After Insertion sort

1 2 4 6 8

## RESULT:

Thus, the C program to implement Insertion Sort was executed successfully and the output was verified.

## SELECTION SORT

**AIM:**

To write a “C++‟ program to implement Selection Sort.

## ALGORITHM:

**Step1 :** Start the program.

**Step2 :** Set MIN to location 0

**Step3 :** Search the minimum element in the list

**Step4 :** Swap with value at location MIN

**Step5 :** Increment MIN to point to next element

**Step6 :** Repeat until list is sorted

**Step7 :** Stop the program.

## PROGRAM:

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

void SelectionSort (int arr[], int n)

{

int i, j;

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

{

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

{

if (arr[i] > arr[j])

{

arr[i] = arr[i]+arr[j];

arr[j] = arr[i]-arr[j];

arr[i] = arr[i]-arr[j];

}

}

}

}

int main()

{

clrscr(); int n, i;

printf("\nEnter the number of data element to be sorted: "); scanf("%d",&n);

int arr[10];

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

{

printf("Enter %d element:",i+1); scanf("%d",&arr[i]);

}

SelectionSort(arr, n); printf("\nSorted Data:\n"); for (i = 0; i < n; i++) printf("%d \t ",arr[i]); getch();

}

## OUTPUT:

Enter the number of data element to be sorted: 5 Enter 1 element:10

Enter 2 element:2

Enter 3 element:8

Enter 4 element:4

Enter 5 element:6 Sorted Data:

2 4 6 8 10

## RESULT:

Thus, the C++ program to implement Selection Sort was executed successfully and the output was verified.

## MERGE SORT

**AIM:**

To write a ‘C’ program to implement merge sort.

## ALGORITHM:

Step 1: Read the elements into the array

Step 2: Take the second element. Compare it with the first element. If the second element less than the first element interchange them

Step 3: Take the third element compare it first and second element and insert it in the correct position by shifting the elements in the array. So that the first, second and third elements are in sorted array

Step 4: In general take the ith element and compare it with the all the elements before it and place it in the proper position by shifting the elements one position right.

Step 5: When the ith element is placed, the elements in the array from the 0th to the ith position will be in the sorted order

Step 6: The above process is continued for all the elements in the array. Step 7: Stop.

## PROGRAM:

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

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

int main()

{

int arr[30]; int i,size;

printf("Enter total no. of elements : "); scanf("%d",&size);

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

{

printf("Enter %d element: ",i+1); scanf("%d",&arr[i]);

}

part(arr,0,size-1); printf("\n\tSorted elements\n"); for(i=0; i<size; i++) printf("%d\t",arr[i]);

getch(); return 0;

}

void part(int arr[],int min,int max)

{

int mid; if(min<max)

{

mid=(min+max)/2; part(arr,min,mid); part(arr,mid+1,max); merge(arr,min,mid,max);

}

}

void merge(int arr[],int min,int mid,int max)

{

int tmp[30]; int i,j,k,m; j=min; m=mid+1;

for(i=min; j<=mid && m<=max ; i++)

{

if(arr[j]<=arr[m])

{

tmp[i]=arr[j]; j++;

}

else

{

tmp[i]=arr[m]; m++;

}

}

if(j>mid)

{

for(k=m; k<=max; k++)

{

tmp[i]=arr[k]; i++;

}

}

else

{

for(k=j; k<=mid; k++)

{

tmp[i]=arr[k]; i++;

}

}

for(k=min; k<=max; k++) arr[k]=tmp[k];

}

## OUTPUT

Enter total no. of elements: 5 Enter 1 element: 10

Enter 2 element: 2

Enter 3 element: 8

Enter 4 element: 4

Enter 5 element: 6 Sorted elements

2 4 6 8 10

## RESULT:

Thus, the C program to implement Merge Sort was executed successfully and the output was verified.

## OPEN ADDRESSING LINEAR PROBING

**AIM:**

To write a ‘C’ PROGRAM to implement the linear probing hashing technique.

## ALGORTHIM:

1. start the program
2. Declare the variables
3. Check the condition for loop for linear probing and set FLAG as 0 and COUNT as 0
4. By increment of count we can insert the element in the hash table.
5. check the condition for full and display the elements in the hash table if(count==MAX), by help of for loop we can display the elements.
6. stop the program

## PROGRAM

#include<stdio.h> #include<conio.h> #include<stdlib.h> #define MAX 10 int create(int num)

{

int key; key=num%10; return key;

}

void display(int a[MAX])

{

int i;

printf("\nThe Hash Table is \n");

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

{

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

}

}

void linearprob(int a[MAX],int key,int num)

{

int flag=0,i,count=0; if(a[key]= = -1)

{

a[key]=num;

}

else

{ i=0;

while(i<MAX)

{

if(a[i]!=-1)

{

count++;

} i++;

}

if(count= =MAX)

{

printf("\nHash Table is Full"); display(a);

getch();

exit(1);

}

for(i=key+1;i<MAX;i++)

{

if(a[i]= = -1)

{

a[i]=num; flag=1; break;

}

}

for(i=0;i<key&&flag= = 0;i++)

{

if(a[i]= = -1)

{

a[i]=num; flag=1; break;

}

}

}

}

void main()

{

int a[max],num,key,i; char ans;

clrscr();

printf("\nCollision Handling by Linear Probing"); for(i=0;i<MAX;i++)

{

a[i]=-1;

}

do

{

printf("\nEnter the Number "); scanf("%d",&num); key=create(num); linearprob(a,key,num);

printf("\nDo u want to continue?(y for YES/n for NO): "); ans=getche();

}while(ans= = 'y'); display(a); getch();

}

## OUTPUT:

Collision Handling by Linear Probing Enter the Number 131

Do u want to continue? (y for YES/n for NO): y Enter the Number 21

Do u want to continue? (y for YES/n for NO): y Enter the Number 3

Do u want to continue? (y for YES/n for NO): y Enter the Number 4

Do u want to continue? (y for YES/n for NO): y Enter the Number 5

Do u want to continue? (y for YES/n for NO): y Enter the Number 8

Do u want to continue? (y for YES/n for NO): y Enter the Number 9

Do u want to continue? (y for YES/n for NO): y Enter the Number 18

Do u want to continue? (y for YES/n for NO): n The Hash Table is ....

0 18

1 131

2 21

3 3

4 4

5 5

6 -1

7 -1

8 8

9 9

## RESULT:

Thus, the Program for implementation of linear probing hashing technique is executed and its output is verified.

## OPEN ADDRESSING QUADRATIC PROBING

**AIM:**

To write a ‘C’ PROGRAM to implement the quadratic probing hashing technique.

## ALGORTHIM:

1. start the program
2. Create an array of structure (i.e a hash table).
3. Take a key and a value to be stored in hash table as input.
4. Corresponding to the key, an index will be generated i.e every key is stored in a particular array index.
5. Using the generated index, access the data located in that array index.
6. In case of absence of data, create one and insert the data item (key and value) into it and increment the size of hash table.
7. In case the data exists, probe through the subsequent elements (looping back if necessary) for free space to insert new data item.
8. Note: This probing will continue until we reach the same element again (from where we began probing)
9. Note: Here, unlike Linear Probing, probing will be done according to the following formula –
10. (currentPosition + h) % arraySize => Linear Probing
11. (currentPosition + (h \* h)) % arraySize => Quadratic Probing
12. where h = 1, 2, 3, 4 and so on.
13. To display all the elements of hash table, element at each index is accessed (via for loop).
14. To remove a key from hash table, we will first calculate its index and delete it if key matches, else probe through elements until we find key or an empty space where not a single data has been entered (means data does not exist in the hash table).
15. Stop

## PROGRAM:

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

#define MAX 10

void printArray(int arr[], int n)

{

printf("The Quadratic Probing Hash table"); for (int i = 0; i < n; i++)

{

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

}

}

void hashing(int table[], int tsize, int arr[], int N)

{

for (int i = 0; i < N; i++)

{

int hv = arr[i] % tsize; if (table[hv] == -1) table[hv] = arr[i];

else

{

for (int j = 0; j < tsize; j++)

{

int t = (hv + j \* j) % tsize; if (table[t] == -1)

{

table[t] = arr[i]; break;

}}

}}

printArray(table, N);

}

void main()

{

int arr[MAX],i,n,s=10; clrscr();

printf("\nEnter the size of n:"); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("\nEnter the Number: "); scanf("%d",&arr[i]);

}

int hash\_table[MAX]; for (int i = 0; i < s; i++)

{

hash\_table[i] = -1;

}

hashing(hash\_table, s, arr, n); getch();

}

## OUTPUT:

Enter the size of n:7 Enter the Number: 50 Enter the Number: 700 Enter the Number: 76 Enter the Number: 85 Enter the Number: 92 Enter the Number: 73 Enter the Number: 101

The Quadratic Probing Hash table [0] 50

[1] 700

[2] 92

[3] 73

[4] -1

[5] 85

[6] 76

## RESULT:

Thus, the Program for implementation of quadratic probing hashing technique is executed and its output is verified.