DATA STRUCTURES – RECORD

# Week 1 – Pointers Recap and Abstraction: Date: 24-01-24

# 1.Sum of Integers

**1.**Aim:

To Write a C++ program to find the sum of 'n' integers using only pointers

**2.Algorithm:**

Input: n,num

Output: sum

1. **Start**.
2. Allocate memory dynamically to store the number of integers n and the sum of the integers (**sum**). Initialize **sum** to 0.
3. Prompt the user to enter the number of integers they wish to sum up and store this value in **n**.
4. While the value of **n** is not equal to 0, repeat steps 5-7.
5. Allocate memory dynamically for a temporary integer pointer (**num**) to store the current integer entered by the user.
6. Prompt the user to enter an integer and store this value in **num**.
7. Add the value pointed to by **num** to **sum**, decrement the value of **n**, and free the memory allocated for **num**.
8. After exiting the loop, print the value of **sum** as the sum of the entered integers.
9. Free the memory allocated for **sum** and **n**.
10. **End**

3.Time complexity analysis- O(n)

4.CODE

// program to find the sum of 'n' integers using only pointers

#include<stdio.h>

#include<stdlib.h>

int main()

{

    int \*n,\*sum;

    n=(int\*)malloc(sizeof(int));

    sum=(int\*)malloc(sizeof(int));

    \*sum=0;

    printf("Enter the number of integers");

    scanf("%d",n);

    while(\*n!=0)

    {

       int \*num;

       num=(int\*)malloc(sizeof(int));

       printf("Enter the number");

       scanf("%d",num);

       \*sum=\*sum+\*num;

       (\*n)--;

    }

    printf("The sum is %d",\*sum);

    free(sum);

    free(n);

    return 0;

}

5.OUTPUT

A screen shot of a computer

Description automatically generated

# 2.Calculator:

## Aim:

To implement a calculator that performs various arithmetic operations.

## Algorithm:

1. Start the program
2. Declare and initialize variables `choice`, `num1`, and `num2` to 0.
3. Prompt the user to enter the first number (`num1`).
4. Read the value of `num1` from the user.
5. Prompt the user to enter the second number (`num2`).
6. Read the value of `num2` from the user.
7. Display the menu with options for arithmetic operations.
8. Read the user's choice (`choice`).
9. Perform the following steps in a loop until the user chooses to exit:
   1. If `choice` is 1, add `num1` and `num2`, and display the result.
   2. If `choice` is 2, subtract `num2` from `num1`, and display the result.
   3. If `choice` is 3, multiply `num1` and `num2`, and display the result.
   4. If `choice` is 4:
      1. If num2==0, display an error message indicating division by zero is not possible.
      2. Else, divide `num1` by `num2` and display the result.
   5. If `choice` is 5, exit the loop.
   6. If `choice` is invalid, display an error message.
10. End the program.

**Complexity:**

1. Addition – O(1)
2. Subtraction – O(1)
3. Multiplication – O(1)
4. Division – O(1)

## Code:

### Header file:

#include<stdio.h>

int sum(int a,int b)

{

    return a+b;

}

int difference(int a,int b)

{

    return a-b;

}

int product(int a,int b)

{

    return a\*b;

}

int division(int a,int b)

{

    return a/b;

}

 void set(int \*a,int \*b)

{

    printf("Enter the first integer:");

    scanf("%d",a);

    printf("Enter the second integer:");

    scanf("%d",b);

}

### Main [cpp] file:

#include<stdio.h>

#include"arithmetic.h"

int main()

{

    int a,b,choice;

    while(1)

    {

        printf("\n(1) Addition");

        printf("\n(2) Subtraction");

        printf("\n(3) Multiplication");

        printf("\n(4) Divsion");

        printf("\n(5) Exit");

        printf("Enter your choice:");

        scanf("%d",&choice);

        getchar();

        switch (choice)

        {

            case 1:

              printf("Enter the first integer:");

              scanf("%d",&a);

              printf("Enter the second integer:");

              scanf("%d",&b);

              printf("\nADDITION:%d",sum(a,b));

              break;

            case 2:

              printf("Enter the first integer:");

              scanf("%d",&a);

              printf("Enter the second integer:");

              scanf("%d",&b);

              printf("\nSUBTRACTION:%d",difference(a,b));

              break;

            case 3:

              printf("Enter the first integer:");

              scanf("%d",&a);

              printf("Enter the second integer:");

              scanf("%d",&b);

              printf("\nPRODUCT:%d",product(a,b));

              break;

            case 4:

              printf("Enter the first integer:");

              scanf("%d",&a);

              printf("Enter the second integer:");

              scanf("%d",&b);

              printf("\nQUOTIENT:%d",division(a,b));

              break;

            case 5:

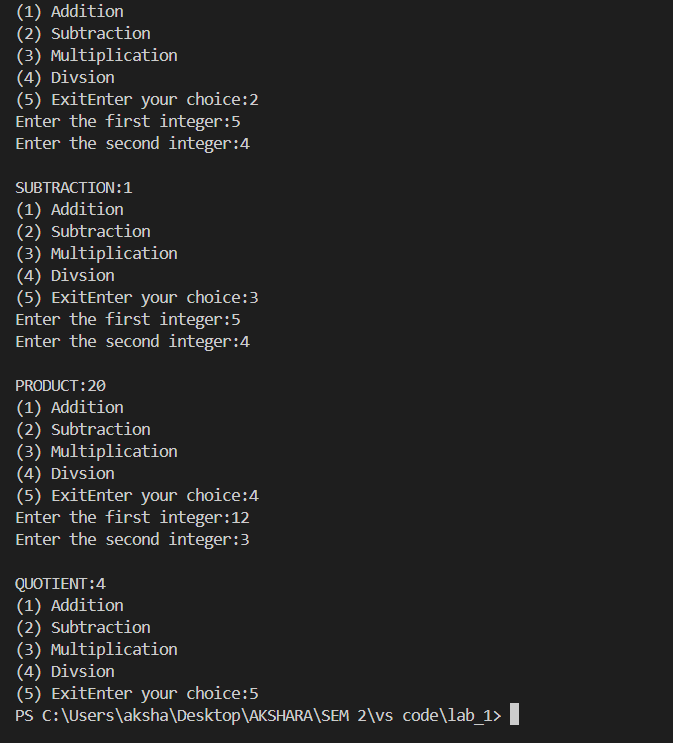
               return 0;

        }

    }

}

**Output:**



# Week 2 – Searching and Sorting: Date: 31-01-24

1. **Searching:**

## Aim:

To check if the given number is present in the array.

**Algorithm:**

* 1. Search:
     1. Set flag to -1.
     2. Initialize i to 0.
     3. Repeat until i is less than n:
     4. If arr[i] equals target:
        1. Set flag to i.
        2. Break the loop.
     5. Increment i by 1.
     6. Return flag.

#### Complexity Analysis:

a. Search – Time Complexity O(n)

#### Code:

#include<stdio.h>

int main()

{

    int i,n,arr[25],num;

    printf("Enter the number of students");

    scanf("%d",&n);

    printf("Enter the elements:");

    for(i=0;i<n;i++)

    {

      scanf("%d",&arr[i]);

    }

    printf("Enter the element to search:");

    scanf("%d",&num);

    for(i=0;i<n;i++)

    {

        if(num==arr[i])

        {

        printf("%d is found at %d index",arr[i],i);

        break;

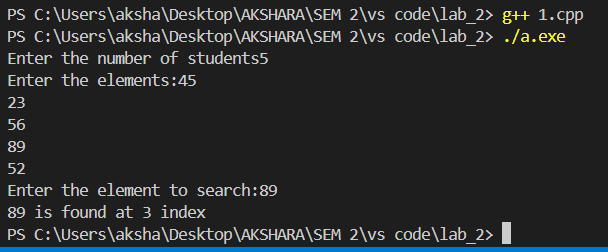
        }

    }

    return 0;

}

**Output:**

****

# Sorting:

## Aim:

To sort an integer array in ascending or descending order

## Algorithm:

* 1. Ascend – sort:

1. Initialize variables i, j, and temp.
2. Start an outer loop from j = 0 to num - 1.
3. Within the outer loop, start an inner loop from i = 0 to num - 1 - j.
4. Inside the inner loop:
5. Compare n[i] and n[i + 1].
6. If n[i] is greater than n[i + 1], swap n[i] and n[i + 1].
7. After each pass through the array (j), the smallest element will be placed at the beginning of the array.
   1. Descend – sort:
8. Initialize variables i, j, and temp.
9. Start an outer loop from j = 0 to num - 1.
10. Within the outer loop, start an inner loop from i = 0 to num - 1 - j.
11. Inside the loop, compare n[i] and n[i + 1].
12. If n[i] is lesser than n[i + 1], swap n[i] and n[i + 1].
13. After each pass through the array (j), the largest element will be placed at the beginning of the array.

**Complexity:**

The program was implemented using *bubble sort* algorithm and it has a time complexity of O(n^2) in the worst case, where n is the number of elements in the array.

1. Ascending – O(n^2)
2. Descending – O(n^2)

## Code:

#include<stdio.h>

int largest(int \*arr,int n)

{

    int max\_index=0;

    for(int i=0;i<n;i++)

    {

      if(arr[i]>arr[max\_index])

      max\_index=i;

    }

    return max\_index;

}

void swap(int \*a,int \*b)

{

    int temp=\*a;

    \*a=\*b;

    \*b=temp;

}

void sort(int \*arr,int n)

{   int i;

    for( i=0;i<n;i++)

    {

        int max\_index=largest(arr+i,n-i)+i;

        swap(&arr[max\_index],&arr[i]);

    }

}

int main()

{

   int choice,n,arr[25];

   printf("Enter the number of elements:");

   scanf("%d",&n);

   printf("Enter the elements:");

   for(int i=0;i<n;i++)

   {

    scanf("%d",&arr[i]);

   }

   while(1)

   {

   printf("\n(1) Ascending");

   printf("\n(2) Descending");

   printf("\n(3) Exit");

   printf("\nEnter your choice:");

   scanf("%d",&choice);

   getchar();

   switch (choice)

   {

      case 1:

      sort(arr,n);

      printf("Elements after sorting:");

      for(int i=n-1;i>=0;i--)

      {

        printf("%d\t",arr[i]);

      }

      break;

      case 2:

      sort(arr,n);

      printf("Elements after sorting in descending order:");

      for(int i=0;i<n;i++)

      {

        printf("%d\t",arr[i]);

      }

      break;

      case 3:

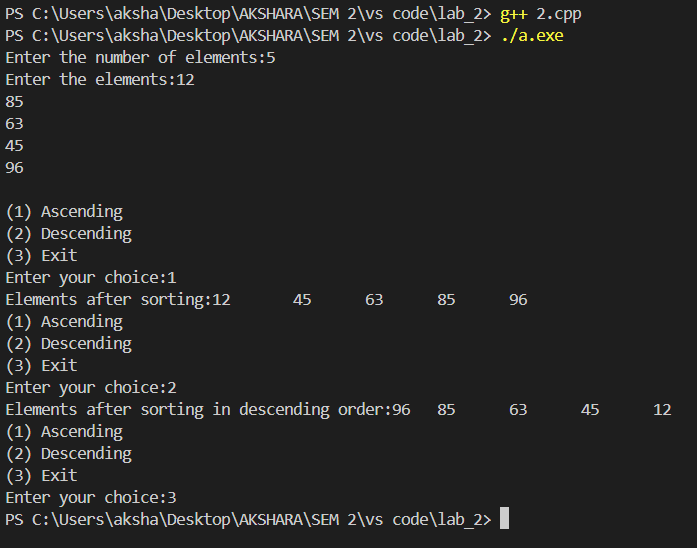
      return 0;

   }

   }

}

**Output:**



# Sorting and Searching [Roll no]:

## Aim:

To search for the given roll number in a sorted array.

## Algorithm:

1. Compare the roll number with the middle element.
2. If present -> return index + 1
3. Otherwise check if the roll number to be searched is bigger than the number in the current index.
4. If so, set the current location as the starting point and repeat 1
5. Otherwise, set the current location as the ending point and repeat 1
6. If there are no more elements to be searched, then return that the roll number is not present in the array.

**Complexity:**

Binary search has a time complexity of O(log n), where n is the number of elements in the array. This is because the search space reduces by half in each step..

## Code:

#include<stdio.h>

void binarysearch(int \*arr,int low,int high,int num)

{

  while(low<=high)

  {

    int middle=low + (high-low)/2;

    if(arr[middle]==num)

    {printf("%d is found at %d index",num,middle);

    return ;}

    else if(arr[middle]<num)

    low=middle+1;

    else

    high=middle-1;

  }

  printf("%d is not found",num);

}

int smallest(int \*arr,int n)

{

    int min\_index=0;

    for(int i=0;i<n;i++)

    {

      if(arr[i]<arr[min\_index])

      min\_index=i;

    }

    return min\_index;

}

void swap(int \*a,int \*b)

{

    int temp=\*a;

    \*a=\*b;

    \*b=temp;

}

void sort(int \*arr,int n)

{   int i;

    for( i=0;i<n;i++)

    {

        int min\_index=smallest(arr+i,n-i)+i;

        swap(&arr[i],&arr[min\_index]);

    }

}

int main()

{

    int n,arr[25],num,og;

    char order;

   printf("Enter the number of students:");

   scanf("%d",&n);

   for(int i=0;i<n;i++)

   {

    printf("\n Enter the roll number for student %d:",i+1);

    scanf("%d",&arr[i]);

   }

   sort(arr,n);

  printf("\nElements after sorting");

   for(int i=0;i<n;i++)

   {

    printf("%d\t",arr[i]);

   }

   printf("\nEnter the element to search:");

   scanf("%d",&num);

   binarysearch(arr,0,n-1,num);

  return 0;

}

## Output:

**A screenshot of a computer

Description automatically generated**

**WEEK 3: List ADT: Array implementation Date:07/02/24**

**Aim:**

**To** Write a C++ menu driven program to implement List ADT using arrays. Maintain proper boundary conditions and follow good coding practices.

TIME COMPLEXITY ANALYSIS:

1. Insertion at beginning : O(n)
2. Insertion at end : O(1)
3. Insertion at position : O(n)
4. Deletion at beginning: O(n)
5. Deletion at end: O(n)
6. Deletion at position: O(n)
7. Search: O(n)
8. Display: O(n)

ALGORITHM:

1. Insert at beginning

Input: num

Output: 1 if element is added else 0

If cur=size-1

Return 0

Else

Temp=cur

While temp!=0

Arr[temp+1]=arr[temp]

Temp--

Arr[0]=num

Inc cur

Return 1

2)Insert at end

Input:num

Output:1 if number is added else 0

If cur=size-1

Return 0

Else

Arr[cur]=num

Return 1

3)Insert at end

Input:num,pos

Output:1 if element is added and 0 if not

If cur=size-1 or pos>cur

Return 0

Else

Temp=pos

Repeat until temp>=cur

Arr[temp+1]=arr[temp]

Arr[pos]=num

Cur++

Return 1

4)Delete at beginning

Input : none

Output: 0 or 1

If cur=-1

Return 0

Else

Temp=0

Repeat until temp=cur

Arr[temp]=arr[temp+1]

Cur - -

Return 1

5) Delete at end

Input: none

Output:0 or 1

If cur=-1

Return 0

Else

Cur - -

Return 1

6) Delete at position

Input: pos

Output:0 or 1

If cur=-1

Return 0;

Else:

If pos=0

Call delbeg

Else if pos=cur

Call delend

Else

Temp=pos

Repeat until temp<=cur

Arr[temp]=arr[temp+1]

Cur- -

Return 1

7) Search

Input: num

Output: 0 or 1

I=0

Repeat until I is not greater than or equal to cur

If arr[i]=num

Return 1

Return 0

8)Display

Input: none

Output: displaying the elements

If cur =-1

Display array is empty

else

Temp=0

Repeat until temp <=cur

Display arr[temp]

TIME COMPLEXITY ANALYSIS:

1. Insertion at beginning -Θ(1)
2. Insertion at end- Θ(1)
3. Insertion at position- O(n)
4. Deletion at beginning- O(n)
5. Deletion at end- Θ(1)
6. Deletion at position O(n)
7. Search - 0(n)
8. Display- O(n)

CODE:

//program to implement Array ADT

#include<stdio.h>

#include<stdlib.h>

#define size 5

class list

{

  int cur;

   int arr[size];

   public:

     list()

      {

      cur=-1;

      }

int indexbeg(int);

void display();

int append(int);

int insert(int ,int );

int delbeg();

int delend();

int delpos(int );

int search(int);

};

int main()

{

  int choice;

  list l1;

  while(1)

  {

printf("\n(1) Insert at beginning");

printf("\n(2) Append");

printf("\n(3) Insert position");

printf("\n(4) Delete beginning");

printf("\n(5) Delete End");

printf("\n(6) Delete position");

printf("\n(7) Search");

printf("\n(8) Display");

printf("\n(9) Exit");

printf("\nChoose choice:");

scanf("%d",&choice);

 switch (choice)

{

   case 1:

     int num1;

     printf("Enter the number to add:");

     scanf("%d",&num1);

       if(l1.indexbeg(num1))

       {

           printf("Number added successfully");

       }

       else

       {

           printf("Array is full");

       }

    break;

    case 2:

    int num2;

    printf("Enter the number to append:");

    scanf("%d",&num2);

    if(l1.append(num2))

    {

        printf("Number added successfully");

    }

    else

    {

        printf("Array is full");

    }

      break ;

      case 3:

      int num3,j;

      printf("Enter the number to add:");

      scanf("%d",&num3);

      printf("Enter the index:");

      scanf("%d",&j);

      if(l1.insert(num3,j))

      {

      printf("Number added successfully");

      }

      else{

        printf("Array is full or index out of range");

      }

      break;

      case  4:

      if(l1.delbeg())

      {

        printf("Element deleted at the beginning successfully");

      }

      else

      {

        printf("Element is not removed");

      }

      break;

     case 5:

     if(l1.delend())

     {

      printf("Element is removed successfully at the end");

     }

     else

     {

      printf("List is empty");

     }

     break;

     case 6:

     int num6;

     printf("Enter the index to remove:");

     scanf("%d",&num6);

     if(l1.delpos(num6))

     {

      printf("Element is removed successfully");

     }

     else

     {

     printf("List is empty");

     }

     break;

    case 7:

    int num7;

    printf("Enter the number to search:");

    scanf("%d",&num7);

    if(l1.search(num7))

    {

      printf("Element is found in the list");

    }

    else

    {

      printf("Element is not found in the list");

    }

    break;

    case 8:

       l1.display();

       break;

    case 9:

    return 0;

   }

}

return 0;

}

//method to insert at beginning

int list::indexbeg(int num)

{

if(cur==size-1)

  {

   return 0;

  }

 else if(cur==-1)

 {

  cur++;

  arr[0]=num;

  return 1;

 }

else

  {

   for(int i= cur;i>=0;i--)

   {

       arr[i+1]=arr[i];

   }

     cur++;

    arr[0]=num;

   return 1;

  }

}

//Method to insert at end

int list::append(int num)

{

    if(cur+1==size)

    {

      return 0;

    }

    else

    {

    cur++;

    arr[cur]=num;

    return 1;

    }

}

//method to display

void list::display()

{

  if(cur==-1)

  {

    printf("Array is empty");

  }

  else

  {

   printf("\nThe contents of the list are:");

   for(int i=0;i<=cur;i++)

   {

    printf("%d\t",arr[i]);

   }

  }

}

//method to insert an element at a given position

int list::insert(int num,int j)

{

  if(cur+1==size||j>cur)

  return 0;

  else

  {

        if(j<=cur)

        {

            for (int k=j;k<=cur;k++)

            {

                arr[k+1]=arr[k];

            }

        }

        arr[j]=num;

        cur++;

        }

  }

//delete at beginning

int list::delbeg()

{

  if(cur==-1)

  {

    return 0;

  }

  else

  {

   for(int i=0;i<=cur;i++)

   {

       arr[i]=arr[i+1];

   }

  cur--;

   return 1;

  }

}

//method to delete the elememt at the end

int list::delend()

{

  if(cur==-1)

  {

    return 0;

  }

  else

  {

    cur--;

    return 1;

  }

}

//method to delete an element at a given position

int list::delpos(int num)

{

  if(cur==-1)

  {

    return 0;

  }

  else if(num==cur)

  {

    delend();

    cur--;

    return 1;

  }

  else if(num==0)

  {

    delbeg();

     cur--;

    return 1;

  }

  else

  {

   for(int i=num;i<=cur;i++)

   {

    arr[i]=arr[i+1];

   }

    cur--;

   return 1;

  }

}

//method to search an element

int list::search(int num)

{

  if(cur==-1)

  {

    return 0;

  }

  else

  {

    for(int i=0;i<size;i++)

    {

      if(arr[i]==num)

      {

        return 1;

      }

      else

      {

        return 0;

      }

    }

  }

}

OUTPUT:

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

WEEK 4:LIST ADT:SINGLY LINKED LIST Date:14/02/24

1.AIM:

To Write a C++ menu-driven program to implement List ADT using a singly linked list.

2.FUNCTIONS USED

1. Insert at beginning :O(1)
2. Insert at end: O(n)
3. Insert at pos: O(n)
4. Delete at beginning: O(1)
5. Delete at pos: O(n)
6. Delete at end : O(n)
7. Search : O(n)
8. Display : O(n)
9. Display reverse : O(n)

10.Reverse link : O(n)

3.ALGORITHM:

1.INSERT AT BEGINNING:

Input:num

Output: 0 or 1

If head=NULL

Head=newnode

Return 1

Else

Newnode->next=head

Head=newnode

Return 1

2.INSERT AT END:

Input:num

Output:0 or 1

If head=NULL

Head=newnode

Return 1

Else

Temp=head

Repeat until temp->next!=NULL

Temp=temp->next

Temp->next=newnode

Return 1

3.INSERT AT POS:

Input:num,pos

Output:0 or 1

If pos=0

Call insertbeg()

Else

Temp=head

Repeat until temp->next!=NULL and pos!=1

Temp=temp->next

Pos- -

Newnode->next=temp->next

Temp->next=newnode

Return 1

4.DELETE AT BEGINNING:

Input: none

Output: 0 or 1

If head==NULL

Return 0

Else

Head=head->next

Return 1

5.DELETE AT END:

Input: none

Output:0 or 1

If head=NULL

Return 0

Else

Temp=head

Repeat until temp->next!=NULL

Temp2=temp

Temp=temp->next

Temp2->next=NULL

Free(temp)

Return 1

6.DELETE AT POS:

Input: pos

Output: 0 or 1

If head=NULL

Return 0

Else

Temp=head

Repeat until temp->next!-NULL and pos!=1

Temp2=temp

Temp=temp->next

Pos- -

Temp2->next=temp->next

Free(temp)

Return 1

7.SEARCH:

Input: num

Output: 0 or 1

If head=NULL

Return 0

Else

Temp=head

Repeat until temp->next!=NULL

If temp->data=num

Return 1

Temp=temp->next

Return 0

8.DISPLAY:

Input:none

Output: All the elements in the list

If head=NULL

Display the list is empty

Else

Temp=head

Repeat until temp!=NULL

Display temp->data

Temp=temp->next

9.DISPLAY REVERSE:

Input:none

Ouput: elements displayed in reverse order

If head=NULL

Return

Else

Call disrev(head->next)

Display head->data

10.REVERSELINK :

Input: none

Output: List is reversed (0 or 1)

If head=NULL

Return 0

Else

Initialize temp1 and temp2 to NULL

Repeat until head!=NULL

Temp2=head->next

Head->next=temp1

Temp1=head;

Head=temp2

Head=temp1

Return 1

CODE:

/\*Program to implement list ADT using singly linked list\*/

#include<stdio.h>

#include<stdlib.h>

class linked

{

  struct node

{

    int data;

    struct node \*next;

};

struct node \*head;

void displayrev(struct node \*temp)

{

    if (temp==NULL)

    {

        return;

    }

    displayrev(temp->next);

    printf("\t%d",temp->data);

}

public:

  void calldis()

  {

    struct node \*temp=head;

    displayrev(temp);

  }

  linked()

  {

    head=NULL;

  }

    int insertbeg(int num);

    int insertend(int num);

    int insertpos(int num,int pos);

    int delbeg();

    int delend();

    int delpos(int pos);

    int search(int num);

    void display();

    int reverselink();

};

int main()

{

    int choice;

    linked l1;

    while(1)

{

printf("\n(1) Insert at beginning");

printf("\n(2) Insert at end");

printf("\n(3) Insert position");

printf("\n(4) Delete beginning");

printf("\n(5) Delete End");

printf("\n(6) Delete position");

printf("\n(7) Search");

printf("\n(8) Display");

printf("\n(9) Display Reverse");

printf("\n(10) Reverse link");

printf("\n(11) Exit");

printf("\nChoose choice:");

scanf("%d",&choice);

    switch(choice)

    {

        case 1:

        int num1;

        printf("Enter the number to insert:");

        scanf("%d",&num1);

        if(l1.insertbeg(num1))

        {

         printf("Element is inserted successfully");

        }

        else

        {

         printf("Operation failed");

        }

        break;

        case 2:

        int num2;

        printf("Enter the number to insert:");

        scanf("%d",&num2);

        if(l1.insertend(num2))

        {

            printf("Element is appended successfully");

        }

        else

        {

            printf("Operation failed");

        }

        break;

        case 3:

        int num3,pos3;

        printf("Enter the number to insert:");

        scanf("%d",&num3);

        printf("Enter the position to insert:");

        scanf("%d",&pos3);

        if(l1.insertpos(num3,pos3))

        {

           printf("Element is inserted");

        }

        else

        {

            printf("The list is empty");

        }

        break;

        case 4:

        if(l1.delbeg())

        {

            printf("Element is deleted successfully");

        }

        else

        {

            printf("The list is already empty");

        }

        break;

        case 5:

        if(l1.delend())

        {

            printf("Element is deleted successfully");

        }

        else

        {

            printf("The list is already empty");

        }

        break;

        case 6:

        int pos6;

        printf("Enter the position to remove the element:");

        scanf("%d",&pos6);

        if(l1.delpos(pos6))

        {

            printf("Element deleted successfully");

        }

        else

        {

            printf("The list is already empty");

        }

        break;

        case 7:

        int num7;

        printf("Enter the element to search:");

        scanf("%d",&num7);

        if(l1.search(num7))

        {

            printf("Element is found in the list");

        }

        else

        {

            printf("Element is not found");

        }

        break;

        case 8:

        l1.display();

        break;

        case 9:

        l1.calldis();

        break;

        case 10:

        if(l1.reverselink())

        {

           printf("The elements are reversed successfully");

        }

        else

        {

            printf("The list is empty");

        }

        break;

        case 11:

        return 0;

    }

    }

    return 0;

    }

//Method to insert a node at beginning

int linked::insertbeg(int num)

{

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    if(head==NULL)

    {

        newnode->next=NULL;

        newnode->data=num;

        head=newnode;

        return 1;

    }

    else

    {

        newnode->next=head;

        newnode->data=num;

        head=newnode;

        return 1;

    }

}

//method to insert a node at end

int linked:: insertend(int num)

{

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    struct node \*temp=head;

    newnode->data=num;

    newnode->next=NULL;

    if(head==NULL)

    {

        head=newnode;

        return 1;

    }

    else

    {

        while(temp->next!=NULL)

        {

            temp=temp->next;

        }

        temp->next=newnode;

        return 1;

    }

}

// method to insert a node at a given position

int linked::insertpos(int num,int pos)

{

    struct node \*temp1=head;

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    newnode->data=num;

    newnode->next=NULL;

    if(pos==1)

    {

        insertbeg(num);

        return 1;

    }

    else

    {

    while(temp1->next!=NULL &&pos!=1)

    {

        temp1=temp1->next;

        pos--;

    }

    newnode->next=temp1->next;

    temp1->next=newnode;

    return 1;

    }

}

//method to delete a node in the beginning

int linked :: delbeg()

{

    struct node \*temp=head;

    if(head==NULL)

    {

        return 0;

    }

    else

    {

        head=head->next;

        free(temp);

        temp=NULL;

    }

}

// method to delete a node at the end

int linked::delend()

{

     struct node \*temp1=head;

     struct node \*temp2=NULL;

     if(head==NULL)

     {

        return 0;

     }

     else

     {

        while(temp1->next!=NULL)

        {

            temp2=temp1;

            temp1=temp1->next;

        }

        temp2->next=NULL;

        free(temp1);

        temp1==NULL;

        return 1;

     }

}

//method to delete a node at a given position

int linked::delpos(int pos)

{

    struct node \*temp1=head;

    struct node \*temp2=(struct node\*)malloc(sizeof(struct node));

    if(head==NULL)

    {

        return 0;

    }

    else if(pos==1)

    {

        delbeg();

        return 0;

    }

    else

    {

        for(int i=0;i<pos-1 && temp1->next!=NULL;i++)

        {

            temp2=temp1;

            temp1=temp1->next;

        }

        temp2->next=temp1->next;

        free(temp1);

        temp1=NULL;

        return 1;

    }

}

//method to search a node

int linked::search(int num)

{

    int count=0;

    struct node \*temp=head;

    if(head==NULL)

    {

        return 0;

    }

    else

    {

        while(temp->next!=NULL)

        {

            if(temp->data==num)

            {

                count=1;

                break;

            }

            temp=temp->next;

        }

        if(count==1)

        {

            return 1;

        }

        else

        {

            return 0;

        }

    }

}

//method to display in reverse

void linked::display()

{

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

    temp=head;

    while(temp!=NULL)

    {

        printf("%d\t",temp->data);

        temp=temp->next;

    }

}

//method to reverse the list

int linked:: reverselink()

{

    struct node \*temp1=NULL;

    struct node \*temp2=NULL;

    if(head==NULL)

    {

        return 0;

    }

    else

    {

        while(head!=NULL)

        {

            temp2=head->next;

            head->next=temp1;

            temp1=head;

            head=temp2;

        }

        head=temp1;

        return 1;

    }

}

5.OUTPUT

A screenshot of a computer program

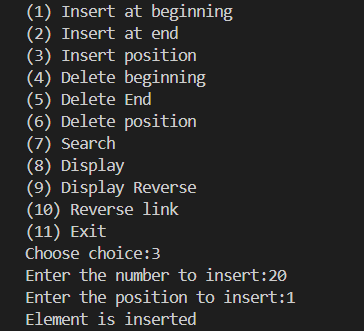
Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated



A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated

**WEEK 5-LIST ADT- DOUBLY LINKED LIST Date-21/02/24**

1.AIM:

To Write a C++ menu-driven program to implement List ADT using a doubly linked list

2.TIME COMPLEXITY ANALYSIS:

1.Insert at beginning : O(1)

2.Insert at end : O(n)

3.Insert at position: O(n)

4.Delete at beginning : O(1)

5.Delete at end : O(n)

6.Delete at position : O(n)

7.Search : O(n)

8.Display : O(n)

3.ALGORITHM

1.INSERT AT BEGINNING:

Input:num

Output: 0 or 1

Initialize newnode

if head=NULL

Head=newnode

Return 1

Else

Head->prev=newnode

Newnode->next=head

Head=newnode

Return 1

2.INSERT AT END:

Input : num

Output : 0 or 1

If head=NULL

Head=newnode

Return 1

Else

Set temp=head

Repeat until temp->next!=NULL

Temp=temp->next

Newnode->prev=temp

Temp->next=newnode

Return 1

3.INSERT AT POS:

Input : num,pos

Output : 0 or 1

If pos=0

Call insertbeg

Else

Set temp=head

Repeat until temp->next!=NULL and pos!=1

Temp2=temp

Temp=temp->next

Pos- -

Temp2->next=newnode

Newnode->prev=temp2

Newnode->next=temp

Return 1

4.DELETE AT BEGINNING:

Input : none

Output : 0 or 1

If head=NULL

Return 0

Else

Head=head->next

Head->prev=NULL

Return 1

5.DELETE AT END:

Input : none

Output : 0 or 1

If head=NULL

Return 0

Else

Set temp=head

Repeat until temp->next!=NULL

Temp=temp->next

Temp2=temp->next

Temp->next=NULL

Free temp2

Return 1

6.DELETE AT POS:

Input : none

Output : 0 or 1

If head==NULL

Return 0

Else if pos=1

Call delbeg

Else

Set temp=head

Repeat until temp->next!=NULL and pos!=1

Temp2=temp

Temp=temp->next

Temp2->next=temp->next

Temp->prev=temp2

Free temp1

7.SEARCH:

Input: num

Output: 0 or 1

If head=NULL

Return 0

Else

Set Temp=head

Repeat until temp->next!=NULL

If temp->data=num

Return 1

Temp=temp->next

Return 0

8.DISPLAY:

Input:none

Output: All the elements in the list

If head=NULL

Display the list is empty

Else

Temp=head

Repeat until temp!=NULL

Display temp->data

Temp=temp->next

4.CODE

/\*Program to implement list ADT using doubly linked list\*/

#include<stdio.h>

#include<stdlib.h>

class doublelinked

{

    struct node

    {

        int data;

        struct node \*next;

        struct node \*prev;

    };

    struct node \*head;

    int length;

    public:

    doublelinked()

    {

        head=NULL;

        length=0;

    }

    int insertbeg(int num);

    int insertend(int num);

    int insertpos(int num,int pos);

    int delbeg();

    int delend();

    int delpos(int pos);

    int search(int num);

    void display();

};

int main()

{

    doublelinked l1;

    while(1)

    {

     getchar();

     int choice;

printf("\n(1) Insert at beginning");

printf("\n(2) Insert at end");

printf("\n(3) Insert position");

printf("\n(4) Delete beginning");

printf("\n(5) Delete End");

printf("\n(6) Delete position");

printf("\n(7) Search");

printf("\n(8) Display");

printf("\n(9) Exit");

printf("\nChoose choice:");

scanf("%d",&choice);

switch (choice)

{

     case 1:

     int num1;

     printf("Enter the number to insert:");

     scanf("%d",&num1);

     if(l1.insertbeg(num1))

     {

        printf("Number is inserted successfully");

     }

     else

     {

        printf("Operation failed");

     }

     break;

     case 2:

     int num2;

     printf("Enter the number to insert:");

     scanf("%d",&num2);

     if(l1.insertend(num2))

     {

        printf("Number is appended successfully");

     }

     else

     {

        printf("Operation failed");

     }

     break;

     case 3:

     int num3,pos3;

     printf("Enter the number to insert:");

     scanf("%d",&num3);

     printf("Enter the position to insert:");

     scanf("%d",&pos3);

     if(l1.insertpos(num3,pos3))

     {

        printf("Number is inserted successfully");

     }

     else

     {

        printf("Operation failed");

     }

     break;

     case 4:

     if(l1.delbeg())

     {

        printf("Element is deleted successfully");

     }

     else

     {

        printf("The list is already empty");

     }

     break;

     case 5:

     if(l1.delend())

     {

        printf("Element is deleted successfully");

     }

     else

     {

        printf("The list is already empty");

     }

     break;

     case 6:

     int pos6;

     printf("Enter the position to delete the element:");

     scanf("%d",&pos6);

     if(l1.delpos(pos6))

     {

        printf("The element is removed successfully");

     }

     else

     {

        printf("The list is already empty");

     }

     break;

     case 7:

     int num7;

     printf("Enter the element to search:");

     scanf("%d",&num7);

     if(l1.search(num7))

     {

        printf("Element is found in the list");

     }

     else

     {

        printf("Element is not found in the list");

     }

     break;

     case 8:

     l1.display();

     break;

     case 9:

     return 0;

}

    }

    return 0;

}

//Methods

// Method to insert an elemnent at the beginning

int doublelinked::insertbeg(int num)

{

    struct node \*newnode=(struct node\*)malloc(sizeof(struct node));

    struct node \*temp=head;

    if(head==NULL)

    {

        newnode->data=num;

        newnode->prev=NULL;

        newnode->next=NULL;

        head=newnode;

        length++;

        return 1;

    }

    else

    {

        newnode->data=num;

        newnode->prev=NULL;

        newnode->next=NULL;

        head->prev=newnode;

        newnode->next=head;

        head=newnode;

        length++;

        return 1;

    }

}

// Method to append an element

int doublelinked::insertend(int num)

{

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    struct node \*temp=head;

    if(head==NULL)

    {

        insertbeg(num);

        length++;

        return 1;

    }

    else

    {

        while(temp->next!=NULL)

        {

            temp=temp->next;

        }

        newnode->next=NULL;

        newnode->data=num;

        newnode->prev=temp;

        temp->next=newnode;

        temp=newnode;

        length++;

        return 1;

    }

}

// Method to insert an element at desired position

int doublelinked::insertpos(int num,int pos)

{

   struct node\*temp1=head;

   struct node\*temp2=(struct node\*)malloc(sizeof(struct node));

   struct node\*newnode=(struct node\*)malloc(sizeof(struct node));

   if(pos==1)

   {

    insertbeg(num);

    length++;

    return 1;

   }

   else if(pos==length)

   {

    insertend(num);

    length++;

    return 1;

   }

   else if(pos>length)

   {

    return 0;

   }

   else

   {

    for(int i =1;i<pos-1 && temp1->next!=NULL;i++)

    {

     temp1=temp1->next;

    }

    temp2=temp1->next;

    newnode->data=num;

    temp1->next=newnode;

    temp2->prev=newnode;

    newnode->next=temp2;

    newnode->prev=temp1;

    length++;

    return 1;

   }

}

// Method to delete the First element

int doublelinked:: delbeg()

{

    struct node \*temp=head;

    if(head==NULL)

    {

        return 0;

    }

    else

    {

       head=head->next;

       free(temp);

       temp=NULL;

       length--;

       return 1;

    }

}

//Method to delete the last element

int doublelinked::delend()

{

    struct node \*temp1=head;

    struct node\*temp2=NULL;

    if(head==NULL)

    {

        return 0;

    }

    else

    {

        while(temp1->next!=NULL)

        {

            temp2=temp1;

            temp1=temp1->next;

        }

        temp2->next=NULL;

        free(temp1);

        temp1=NULL;

        length--;

        return 1;

    }

}

//Method to delete an element at desired position

int doublelinked::delpos(int pos)

{

    struct node \*temp1=head;

    struct node \*temp2=(struct node \*)malloc(sizeof(struct node));

    if(head==NULL)

    {

        return 0;

    }

    else if(pos==length)

    {

        delend();

        return 1;

    }

    else if(pos>length)

    {

        return 0;

    }

    else

    {

     for(int i=0;i<pos-1 && temp1->next!=NULL;i++)

     {

        temp2=temp1;

        temp1=temp1->next;

     }

      temp2->next=temp1->next;

      temp1->prev=temp2;

      free(temp1);

      temp1= NULL;

      length--;

      return 1;

    }

}

//Method to search an element in the list

int doublelinked::search(int num)

{

    struct node \*temp=head;

    int count=0;

    while(temp!=NULL)

    {

        if(temp->data==num ||temp->next->data==num)

        {

            count=1;

            break;

        }

        temp=temp->next;

    }

    if(count==1)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

//Method to display the elements

void doublelinked::display()

{

    struct node \*temp1=head;

    while(temp1!=NULL)

    {

        printf("%d\t",temp1->data);

        temp1=temp1->next;

    }

    printf("\n");

{

    struct node \*temp=head;

    while(temp->next!=NULL)

    {

        temp=temp->next;

    }

    while(temp!=NULL)

    {

        printf("%d\t",temp->data);

        temp=temp->prev;

    }

}

}

5.OUTPUT:

A screenshot of a computer

Description automatically generatedA computer screen shot of a black screen

Description automatically generated

A screen shot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1.AIM

Write a C++ menu-driven program to implement a browser's front and back functionality.

2.TIME COMPLEXITY ANALYSIS:

1.INSERT WEBPAGE : O(n)

2.FRONT: O(1)

3.BACK : O(1)

3.ALGORITHM

1.INSERT WEBPAGE:

Input: webpage

Output : 0 or 1

If head=NULL

Head=newnode

Current=newode

Else if head!=current

Set temp1=head

Repeat until head!=current

Head=head->next

Temp2=temp1

Temp1=temp1->next

Free temp2

Return insert(webpage)

Else

Head->prev=newnode

Newnode->next=newnode

Head=newnode

Return 1

2.FRONT:

Input: none

Output : 0 or 1

If head=NULL

Return 0

Else

Current=current->prev

Return 1

3.BACK:

Input : none

Output : 0 or 1

If head=NULL

Return 0

Else

Current=current->next

Return 1

4.CODE:

HEADER FILE:

//header file to implement a browser's front and back functionality

#include<stdio.h>

#include<stdlib.h>

class doublelinked

{

    struct node

    {

        int data;

        struct node \*prev;

        struct node \*next;

    };

    struct node \*head;

    struct node \*current;

    public:

    doublelinked()

    {

        head=NULL;

        current=NULL;

    }

    int insertwebpage(int num);

    int front();

    int back();

    void display();

    void state();

};

//Method to insert a new webpage

int doublelinked::insertwebpage(int num)

{

    struct node \*newnode=(struct node\*)malloc(sizeof(struct node));

    if(head==NULL)

    {

      newnode->prev=NULL;

      newnode->data=num;

      newnode->next=NULL;

      current=newnode;

      head=newnode;

      return 1;

    }

    else if(head!=current)

    {

        struct node \*temp1=head;

        struct node \*temp2=NULL;

        while(head!=current)

        {

            head=head->next;

            temp2=temp1;

            temp1=temp1->next;

            free(temp2);

            temp2=NULL;

        }

        return insertwebpage(num);

    }

    else

    {

      newnode->prev=NULL;

      newnode->data=num;

      newnode->next=NULL;

      head->prev=newnode;

      newnode->next=head;

      current=newnode;

      head=newnode;

      return 1;

    }

}

// Method to move the browser to the previous page

int doublelinked::back()

{

  if(current==NULL|| current->next==NULL)

  {

    return 0;

  }

  else if(current->next!=NULL)

  {

    current=current->next;

    return 1;

  }

}

// Method to move the browser to the next page

int doublelinked::front()

{

    if(current==NULL||current->prev==NULL)

    {

        return 0;

    }

    else if(current->prev!=NULL)

    {

        current=current->prev;

    }

}

//Method the disply the current webpage

void doublelinked::state()

{

    printf("%d",current->data);

}

//Method to check the elements

void doublelinked::display()

{

    struct node \*temp=head;

    while(temp!=NULL)

    {

        printf("%d\t",temp->data);

        temp=temp->next;

    }

}

CPP FILE:

//program to implement a browser's front and back functionality

#include<stdio.h>

#include"webpage.h"

int main()

{

    doublelinked l1;

    int choice;

    while(1)

    {

        printf("\n(1)Insert New Webpage");

        printf("\n(2)Front");

        printf("\n(3)Back");

        printf("\n(4)Display");

        printf("\n(5)Current page");

        printf("\n(6) Exit)");

        printf("\nEnter the choice:");

        scanf("%d",&choice);

        switch(choice)

        {

           case 1:

           int num1;

           printf("Enter the number to insert in the webpage");

           scanf("%d",&num1);

           if(l1.insertwebpage(num1))

           {

            printf("Inserted successfully");

           }

           else

           {

            printf("Operation failed");

           }

           break;

           case 2:

           if(l1.front())

           {

            printf("Moved to the the next page!");

           }

           else

           {

            printf("Operation failed");

           }

           break;

           case 3:

           if(l1.back())

           {

            printf("Moved to the previous page");

           }

           else

           {

            printf("Operation failed");

           }

           break;

           case 4:

           l1.display();

           break;

           case 5:

           l1.state();

           break;

           case 6:

           return 0;

        }

    }

    return 0;

}

5.OUTPUT:

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

WEEK 6- STACK ADT Date-28/02/24

1.AIM

2.FUNCTIONS USED:

1.PUSH

2.POP

3.PEEK

3.ALGORITHM

4.CODE

5.OUTPUT

WEEK 8- TREE ADT-BINARY TREE Date-20/03//24

1.AIM:

To Write a separate C++ menu-driven program to implement Tree ADT using a character binary tree

2.ALGORITHMS:

1.INSERT:

Input: num

Output: 0 or 1

If root = NULL

Root=newnode

Return 1

Else

Repeat untill node is inserted

If choice= 1

Go left

If choice= 2

Go right

2.DELETE:

Input: root

Output: none

If root=NULL

Return 0

Else if 1 child:

Set temp=child

Free node

Return temp

Else if 2 children

Set leftmost= rightchild

Repeat until leftmost->left=NULL/

Leftmost=leftmost->left

Leftmost->left=leftchild

Return rightchild

Else

Call delete(root->left or root->right)

3.INORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return;

Else

Call inorder(root->left)

Display root->data

Call inorder(root->right)

4.POSTORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return

Else

Call postorder(root->left)

Call postorder(root->right)

Display root->data

5.PREORDER:

Input : root

Output : displays all the elements

if root=NULL

return

else

Display root->data

call preorder(root->left)  
 call preorder(root->right)

6.SEARCH:

Input : num,root

Output : true or false

if root = NULL

return false

else if root->data=num

return true

else

call search(num,root->left) or search(num,root->right)

3.TIME COMPLEXITY ANALYSIS

1.INSERT:

2.DELETE:

3.INORDER:

4.POSTORDER:

5.PREORDER:

6.SEARCH:

4.CODE:

//program to implement binary tree ADT

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

class bintree

{

    struct node

    {

        struct node \*left;

        char data;

        struct node \*right;

    };

    struct node \*root;

    //method to display elements in inorder

    void inorder(struct node \*root)

    {

        if (root==NULL)

        {

            return ;

        }

        else

        {

            inorder(root->left);

            printf("%c\t",root->data);

            inorder(root->right);

        }

    }

    // method to display elements in preorder

    void preorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            printf("%c\t",root->data);

            preorder(root->left);

            preorder(root->right);

        }

    }

    //method to display elements in postorder

    void postorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            postorder(root->left);

            postorder(root->right);

            printf("%c\t",root->data);

        }

    }

   //method to search an element

    bool search(struct node \*root,char num)

    {

    if(root==NULL)

    {

        return false;

    }

    else if(root->data==num)

    {

        return true;

    }

    else

    {

      return search(root->left,num)||search(root->right,num);

    }

    }

    //method to delete a node

     struct node\* deleteRecursively(struct node\* root, char num, struct node\* parent) {

    if (root == NULL) {

        return NULL;

    }

    if (root->data == num)

     {

        // No children

        if (root->left == NULL && root->right == NULL)

        {

            free(root);

            return NULL;

        }

        // One child

        else if (root->left == NULL || root->right == NULL)

         {

            struct node\* temp = root->left ? root->left : root->right;

            free(root);

            return temp; // Return the child to be connected by the parent

        }

        // Two children

        else

        {

            struct node\* rightChild = root->right;

            struct node\* leftchild = root->left;

            free(root);

            struct node\* leftmost = rightChild;

            while (leftmost->left != NULL) {

                leftmost = leftmost->left;

            }

            leftmost->left = leftchild;

            return rightChild;

        }

    } else

     {

        root->left = deleteRecursively(root->left, num, root);

        root->right = deleteRecursively(root->right, num, root);

    }

    return root;

}

    public:

        void callin()

        {

            inorder(root);

        }

        void callpre()

        {

            preorder(root);

        }

        void callpost()

        {

            postorder(root);

        }

        bool callsearch(char num)

        {

           return search(root,num);

        }

        struct node \* calldel(char num)

        {

           root = deleteRecursively(root, num, NULL);

        }

        bintree()

        {

            root=NULL;

        }

        int insert(char num);

        int delete\_node(char num);

};

int main()

{

    bintree b1;

    int choice;

    while(1)

    {

        printf("\n(1) Insert");

        printf("\n(2) Inorder");

        printf("\n(3) Preorder");

        printf("\n(4) Postorder");

        printf("\n(5) Search");

        printf("\n(6) Delete");

        printf("\n(7) EXit");

        printf("\n Enter your choice:");

        scanf("%d",&choice);

        switch (choice)

        {

            case 1:

            char num1;

            printf("Enter the number to insert:");

            scanf("%c",&num1);

            if(b1.insert(num1))

            {

                printf("Number is inserted successfully");

            }

            else

            {

                printf("Operation failed");

            }

            break;

            case 2:

            b1.callin();

            break;

            case 3:

            b1.callpre();

            break;

            case 4:

            b1.callpost();

            break;

            case 5:

            char num5;

            printf("Enter the number to search:");

            scanf("%c",&num5);

            if(b1.callsearch(num5))

            {

                printf("Number is found!");

            }

            else

            {

                printf("Number is not found");

            }

            break;

            case 6:

            int num6;

            printf("Enter the element to delete");

            scanf("%c",&num6);

            if(b1.calldel(num6))

            {

                printf("Node is deleted succesfully");

            }

            else

            {

                printf("The tree is empty");

            }

            break;

            case 7:

            return 0;

        }

    }

}

//method to insert a node

int bintree::insert(char num)

{

    struct node \*temp=root;

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    newnode->data=num;

    newnode->left=NULL;

    newnode->right=NULL;

    if(root==NULL)

    {

       root=newnode;

       return 1;

    }

    else if(temp->left==NULL||temp->right==NULL)

    {

        int choice;

        printf("Left(1) or Right(2)");

        scanf("%d",&choice);

        if(choice==1)

        {

          temp->left=newnode;

          return 1;

        }

        else if (choice==2)

         {

          temp->right=newnode;

          return 1;

        }

    }

    else

    {

        int choice;

        while(1)

        {

        printf("Left(1) or Right(2)");

        scanf("%d",&choice);

        if(choice==1)

        {

            if(temp->left==NULL)

            {

                temp->left=newnode;

                return 1;

            }

            temp=temp->left;

        }

        else if (choice==2)

        {

            if(temp->right==NULL)

            {

                temp->right=newnode;

                return 1;

            }

            temp=temp->right;

        }

        }

       return 0;

    }

}

5.OUTPUT:

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1.AIM:

To Add a "construct expression tree" method to the binary tree data structure

2.ALGORITHMS:

1. INSERT:

Input : string

Output : none

For char in string

If char Is a operator

Newnode->left=pop

Newnode->right=pop

Else

Assign char to a newnode

Push newnode

2.INORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return;

Else

Call inorder(root->left)

Display root->data

Call inorder(root->right)

3.POSTORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return

Else

Call postorder(root->left)

Call postorder(root->right)

Display root->data

4.PREORDER:

Input : root

Output : displays all the elements

if root=NULL

return

else

Display root->data

call preorder(root->left)  
 call preorder(root->right)

3.TIME COMPLEXITY ANALYSIS:

1. INSERT:
2. INORDER:
3. PREORDER:
4. POSTORDER:

4.CODE:

//program to implement binary expression tree

#include<stdio.h>

#include<stdlib.h>

class bintree

{

    struct node

    {

        struct node \*left;

        char data;

        struct node \*right;

        struct node \*next;

    };

    struct node \*root;

//method for inorder traversal

    void inorder(struct node \*root)

    {

        if (root==NULL)

        {

            return ;

        }

        else

        {

            inorder(root->left);

            printf("%c\t",root->data);

            inorder(root->right);

        }

    }

//method for preorder traversal

    void preorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            printf("%c\t",root->data);

            preorder(root->left);

            preorder(root->right);

        }

    }

// method for postorder traversal

    void postorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            postorder(root->left);

            postorder(root->right);

            printf("%c\t",root->data);

        }

    }

//method to push the node

 void  push(struct node \*newnode)

 {

    if(root==NULL)

    {

        root=newnode;

    }

    else

    {

       newnode->next=root;

       root=newnode;

    }

 }

// method to pop the node

struct node \* pop()

{

    if (root==NULL)

    {

      return NULL;

    }

    else

    {

    struct node \*temp=root;

    root=root->next;

    return temp;

    }

}

    public:

        void callin()

        {

            inorder(root);

        }

        void callpre()

        {

            preorder(root);

        }

        void callpost()

        {

            postorder(root);

        }

        bintree()

        {

            root=NULL;

        }

        void insert(char string[25]);

};

int main()

{

    bintree b1;

    char string[25];

    int choice;

    while(1)

    {

        printf("\n(1) Postfix Expression");

        printf("\n(2) Preorder");

        printf("\n(3) Inorder ");

        printf("\n(4) postorder");

        printf("\n(5) Exit");

        printf("\nEnter your choice");

        scanf("%d",&choice);

        getchar();

        switch (choice)

        {

            case 1:

            printf("Enter the expression");

            scanf(" %24s",string);

            b1.insert(string);

            break;

            case 2:

            b1.callpre();

            break;

            case 3:

            b1.callin();

            break;

            case 4:

            b1.callpost();

            break;

            case 5:

            return 0;

        }

    }

}

//method to insert

void bintree:: insert(char string[25])

{

    for(int i=0;string[i]!='\0';i++)

    {

           struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

            newnode->data=string[i];

            newnode->left=NULL;

            newnode->right=NULL;

            newnode->next=NULL;

        if(string[i]=='\*' || string[i]=='/' || string[i]=='+'|| string[i]=='-')

        {

            newnode->right=pop();

            newnode->left=pop();

        }

         push(newnode);

    }

}

5.OUTPUT:

A screenshot of a computer

Description automatically generated

1.AIM:

To identify the **optimal**ADT that can find a given number, its previous smaller element, and the next bigger element.

2.ALGORITHM:

1. Get the input from the user

2. Store it in the array

3.Traverse the array

4. Repeat until array reaches null character

5.If element < key and >lower

Lower=element

6.If element >key and < upper

Upper=element

3.TIME COMPLEXITY ANALYSIS: O(n)

4.CODE:

//program to find the previous and next element of a given element

#include <stdio.h>

int main(){

    int array[] = {4,1,1,2,6,8,3,9,7,5};

    int len = sizeof(array) / sizeof(int);

    int number;

    int lower = 0;

    int upper = 0;

    int flagl=0,flagu=0;

    printf("Enter Number: ");

    scanf("%d", &number);

    for(int i = 0; i < len; i++){

        if(array[i] < number && (lower == 0 || array[i] > lower))

        {

            lower = array[i];

            flagl=1;

        }

        else if(array[i] > number && (upper == 0 || array[i] < upper))

        {

            upper = array[i];

            flagu=1;

        }

    }

    if(flagl==0)

    {

    printf("None %d %d", number, upper);

    }

    else if (flagu==0)

    {

       printf(" %d %d None",lower, number);

    }

    else

    {

        printf(" %d %d %d",lower, number,upper);

    }

    return 0;

}

5.OUTPUT:

A black screen with white text

Description automatically generated

WEEK 9- TREE ADT-BINARY SEARCH TREE Date:27/03/24

1.AIM:

To Write a separate C++ menu-driven program to implement Tree ADT using a character binary tree

2.ALGORITHMS:

1.INSERT:

Input: num

Output: 0 or 1

If root = NULL

Root=newnode

Return 1

Else

Set temp=root

Repeat until node is inserted

If num<key

Go left and insert if temp->left is NULL else repeat

Else

Go right and insert if temp->right is NULL else repeat

2.INORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return;

Else

Call inorder(root->left)

Display root->data

Call inorder(root->right)

3.POSTORDER:

Input : root

Output : Displays all the elements

If root=NULL

Return

Else

Call postorder(root->left)

Call postorder(root->right)

Display root->data

4.PREORDER:

Input : root

Output : displays all the elements

if root=NULL

return

else

Display root->data

call preorder(root->left)  
 call preorder(root->right)

6.SEARCH:

Input : num,root

Output : true or false

if root = NULL

return false

else if root->data=num

return true

else

if key<root

call search(num,root->left)

else

call search(num,root->right)

3.TIME COMPLEXITY ANALYSIS :

1.INSERT:

2.PREORDER:

3.INORDER:

4.POSTORDER:

5.SEARCH:

4.CODE:

//Program to implement Tree ADT using binary search tree

#include<stdio.h>

#include<stdlib.h>

class bst

{

    struct node

    {

        int data;

        struct node \*left;

        struct node \*right;

    };

    struct node \*root;

// method to print inorder

void inorder(struct node \*root)

{

    if(root==NULL)

    {

        return;

    }

    else

    {

        inorder(root->left);

        printf("%d\t",root->data);

        inorder(root->right);

    }

}

//method to print preorder

    void preorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            printf("%d\t",root->data);

            preorder(root->left);

            preorder(root->right);

        }

    }

//method to print postorder

    void postorder(struct node \*root)

    {

        if(root==NULL)

        {

            return;

        }

        else

        {

            postorder(root->left);

            postorder(root->right);

            printf("%d\t",root->data);

        }

    }

//method to search the element

int search(struct node \*root,int num)

{

    if(root==NULL)

    {

        return 0;

    }

    else if (root->data==num)

    {

        return 1;

    }

    else

    {

        if(root->data>num)

        {

            search(root->left,num);

        }

        else

        {

            search(root->right,num);

        }

    }

}

// Utility function to find the node with the minimum value in a subtree

struct node\* delete\_node(struct node\* root, int num) {

    if (root == NULL)

     return NULL;

    if (num < root->data)

    {

        root->left = delete\_node(root->left, num);

    }

    else if (num >= root->data)

    {

        root->right = delete\_node(root->right, num);

    }

    else

    {

        // Node with one child or no child

        if (root->left == NULL &&root->right!=NULL)

        {

            struct node\* temp = root->right;

            free(root);

            return temp;

        }

        else if (root->right == NULL&&root->right!=NULL)

       {

            struct node\* temp = root->left;

            free(root);

            return temp;

        }

        // Node with two children: Get the inorder successor (smallest in the right subtree)

        struct node\* temp = root->right;

        while(temp->left!=NULL)

        {

            temp=temp->left;

        }

        root->data = temp->data;

        root->right = delete\_node(root->right, temp->data);

    }

    return root;

}

    public:

    bst()

    {

        root=NULL;

    }

    void callin()

    {

        inorder(root);

    }

    void callpre()

    {

        preorder(root);

    }

    void callpost()

    {

        postorder(root);

    }

    int callsearch(int num)

    {

        return search(root,num);

    }

    int calldelete(int num)

    {

         delete\_node(root,num);

    }

    int insert(int num);

};

int main()

{

    bst b1;

    int choice;

    while(1)

    {

        printf("\n(1) Insert");

        printf("\n(2) Inorder");

        printf("\n(3) Preorder");

        printf("\n(4) Postorder");

        printf("\n(5) Search");

        printf("\n(6) Delete");

        printf("\n(7) Exit");

        printf("\n Enter your choice:");

        scanf("%d",&choice);

        switch (choice)

        {

            case 1:

            int num1;

            printf("Enter the number to insert:");

            scanf("%d",&num1);

            if(b1.insert(num1))

            {

                printf("Number is inserted successfully");

            }

            else

            {

                printf("Operation failed");

            }

            break;

            case 2:

            b1.callin();

            break;

            case 3:

            b1.callpre();

            break;

            case 4:

            b1.callpost();

            break;

            case 5:

            int num5;

            printf("Enter the number to search:");

            scanf("%d",&num5);

            if(b1.callsearch(num5))

            {

                printf("The number is found in the tree");

            }

            else

            {

                printf("The number is not found");

            }

            break;

            case 6:

            int num6;

            printf("Enter the number to delete:");

            scanf("%d",&num6);

            b1.calldelete(num6);

            break;

            case 7 :

            return 0;

        }

    }

}

//method to insert the element

int bst::insert(int num)

{

    struct node \*temp=root;

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    newnode->data=num;

    newnode->left=NULL;

    newnode->right=NULL;

    if(root==NULL)

    {

        root=newnode;

        return 1;

    }

    else

    {

        while(1)

        {

        if(temp->data>num)

        {

            if(temp->left==NULL)

            {

                temp->left=newnode;

                return 1;

            }

            temp=temp->left;

        }

        else if(temp->data<=num)

        {

            if(temp->right==NULL)

            {

                temp->right=newnode;

                return 1;

            }

            temp=temp->right;

        }

    }

}

}

A screenshot of a computer program

Description automatically generated5.OUTPUT:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

1.AIM:

To write a program to find the count of ideal substrings In a string

2:ALGORITHMS:

Stack headerfile is imported from previous code (lab 6)

3.CODE:

Headerfile:

/\*Program to implement stack ADT using singly linked list\*/

#include<stdio.h>

#include<stdlib.h>

class stack

{

  struct node

{

    char data;

    struct node \*next;

};

struct node \*head;

public:

  stack()

  {

    head=NULL;

  }

    int push(char chr);

    char pop();

    char peek();

};

//Method to push a character into a stack

int stack:: push(char chr)

{

    struct node \*newnode=(struct node \*)malloc(sizeof(struct node));

    newnode->data=chr;

    if(head==NULL)

    {

      newnode->next=NULL;

      head=newnode;

      return 1;

    }

    else

    {

       newnode->next=head;

       head=newnode;

       return 1;

    }

}

// Method to pop a character from the stack

char stack::pop()

{

    struct node \*temp=head;

    if(head==NULL)

    {

        return '\0';

    }

    else

    {

        char ll=head->data;

        head=head->next;

        free(temp);

        temp=NULL;

        return ll;

    }

}

// Method to display the last element in a stack

char stack::peek()

{

   return head->data;

}

Cpp file:

//program to find the count of ideal substring in a string

#include<stdio.h>

#include<string.h>

#include"stack.h"

int check(char string[25],stack s1);

int main()

{

    stack s1;

    char string[25];

    printf("Enter the string");

    scanf(" %24s",string);

    printf("The count of ideal substring=%d",check(string,s1));

    return 0;

}

int check(char string[25],stack s1)

{

    int count=0;

    char a,b,c;

    for(int i =0;string[i+2]!='\0';i++)

    {

        s1.push(string[i]);

        s1.push(string[i+1]);

        s1.push(string[i+2]);

        a=s1.pop();

        b=s1.pop();

        c=s1.pop();

        if(a!=b && b!=c && a!=c)

        {

            count ++;

        }

    }

    return count;

}

4:OUTPUT:

A black background with white text

Description automatically generated

WEEK 10 -PRIORITY QUEUE ADT-HEAP DATE-03/04/24

1.AIM:

To Write a separate C++ menu-driven program to implement Priority Queue ADT using a max heap.

2.FUNCTIONS USED:

1.INSERT

2.HEAPIFY

3.DISPLAY

4.DELETE

5.SORT

6.SEARCH

3.ALGORITHMS:

1.INSERT:

Input : num

Output : 0 or 1

If cur=size-1

Return 0

Else if cur=-1

Arr[cur++]=num

Return 1

Else

Cur++

Arr[cur]=num

Call heapfiy

Return 1

2.HEAPIFY:

Input : none

Output : elements are heapified

Set temp=cur

Repeat until temp > 0

Parent =i-1/2

If arr[i]>arr[parent]

Swap arr[i] and arr[parent]

3.DISPLAY:

Input : none

Output : elements

Set temp=cur

Repeat until temp >=0

Display arr[temp]

4.DELETE:

Input : none

Ouput : the max element

If cur=-1

Return -1

Else

Temp=arr[cur]

Arr[0]=arr[cur]

Call heapify

Return temp

5.SORT:

Input : none

Output : sorted elements

Repeat until cur>=0

Queue->push(call delete)

Repeat until queue is empty

Queue->pop

6.SEARCH:

Input : num

Output : 0 or 1

If cur=-1

Return 0

Else

Set temp=0

Repeat until temp<=cur

If arr[temp]=num

Return 1

Temp++

Return 0

4.CODE

// to implement priority queue ADT using heap data structure

#include<stdio.h>

#include<iostream>

using namespace std;

#include<stdlib.h>

#include<queue>

#define size 50

class heap

{

  int arr[size];

  int cur;

  public:

  heap()

  {

    cur=-1;

  }

  int insert(int num);

  void heapifyup();

  void display();

  int delmax();

  void sort(queue<int>gq);

  void showq(queue<int> gq);

  int search(int num);

};

int main()

{

    queue <int> gq;

    heap h1;

    int choice;

    while (1)

    {

        printf("\n(1) Insert");

        printf("\n(2) Delete");

        printf("\n(3) Display");

        printf("\n(4) Search");

        printf("\n(5) Sort");

        printf("\n(6) Exit");

        printf("\n Enter your choice");

        scanf("%d",&choice);

        getchar();

        switch (choice)

        {

           case 1:

           int num1;

           printf("Enter the number to insert");

           scanf("%d",&num1);

           if(h1.insert(num1))

           {

            printf("Element is inserted successfully");

           }

           else

           {

            printf("Heap is full");

           }

           break;

           case 2:

           if(h1.delmax())

           {

            printf("Element is deleted successfully");

           }

           else

           {

            printf("The heap is already empty");

           }

           break;

           case 3:

           h1.display();

           break;

           case 4:

           int num4;

           printf("Enter a number to search");

           scanf("%d",&num4);

           if(h1.search(num4))

           {

            printf("the number is found at %d",h1.search(num4));

           }

           else

           {

            printf("Element is not found");

           }

           break;

           case 5:

          h1.sort(gq);

          break;

           case 6:

           return 0;

        }

    }

}

//methods

//method to insert a number

int heap:: insert(int num)

{

  if (cur==size-1)

  {

    return 0;

  }

  else if(cur==-1)

  {

    arr[0]=num;

    cur++;

    return 1;

  }

  else

  {

    cur++;

    arr[cur]=num;

    heapifyup();

    return 1;

  }

}

//method to heapify while inserting

void heap:: heapifyup()

  {

     int i=cur;

     while(i>0)

     {

        int parent=(i-1)/2;

        if(arr[i]>arr[parent])

        {

          int temp=arr[i];

          arr[i]=arr[parent];

          arr[parent]=temp;

        }

        i--;

     }

  }

//method to heapify while deleting

/\*void heap::heapifydown() {

    int i = 0;

    while (true) {

        int left = 2 \* i + 1;

        int right = 2 \* i + 2;

        int largest = i;

        if (left <= cur && arr[left] > arr[largest]) {

            largest = left;

        }

        if (right <= cur && arr[right] > arr[largest]) {

            largest = right;

        }

        if (largest != i) {

            int temp = arr[i];

            arr[i] = arr[largest];

            arr[largest] = temp;

            i = largest;

        } else {

            break;

        }

    }

}\*/

//method to display

void heap::display()

{

  for(int i=0;i<=cur;i++)

  {

    printf("%d\t",arr[i]);

  }

}

//method to delete the root

int heap:: delmax()

{

  if(cur==-1)

  {

    return '\0';

  }

  else

  {

    int temp=arr[0];

    arr[0]=arr[cur];

    cur--;

    heapifyup();

    return temp;

  }

}

//method to sort

void heap::sort(queue<int>gq)

{

  while(cur!=-1)

  {

    gq.push(delmax());

  }

   showq(gq);

}

void heap:: showq(queue<int> gq)

{

    queue<int> g = gq;

    while (!g.empty()) {

        cout << '\t' << g.front();

        g.pop();

    }

    cout << '\n';

}

// method to search

int heap:: search(int num)

{

  for(int i=0;i<=cur;i++)

  {

    if(num==arr[i])

    {

      return i+1;

    }

  }

  return 0;

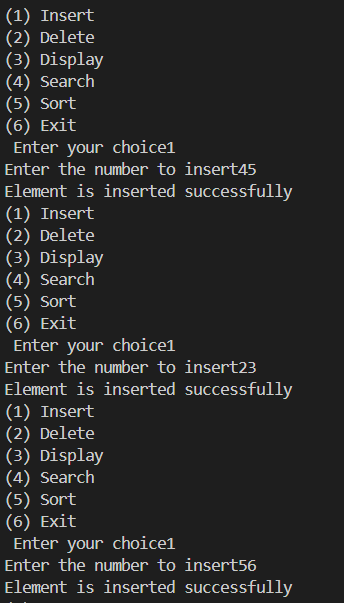
}

5.OUTPUT:

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

WEEK 11- HASH ADT -HASH TABLE Date-10/04/24

1.AIM:

To Write a separate C++ menu-driven program to implement Hash ADT with Separate Chaining.

2.TIME COMPLEXITY ANALYSIS

1.INSERT-

2.DELETE

3.SEARCH

3.ALGORITHM:

1.INSERT:

Input : num

Output : 0 or 1

Index= num mod size

If hashtable[index]=NULL

Hashtable[index]=num

Return 1

Else

Set temp=hashtable[index]

Repeat until temp->next!=NULL

Temp=temp->next

Temp->next=num

Return 1

2.SEARCH:

Input : num

Output: 0 or 1

Index=num mod size

If hashtable[index]=NULL

Return 0

Else

Set temp=hashtable[index]

Repeat until temp->next!=NULL

If temp->data=num

Return 1

Temp=temp->next

Return 0

3.DELETE:

Input : num

Output : 0 or 1

Index= num mod size

If hashtable[index]=NULL

return 0

else if hashtable[index]=num

hashtable[index]=temp->next

Else

Set temp=hashtable[index]

Repeat until temp->!=NULL && temp->data!=num

Temp2=temp

Temp=temp->next

If temp=NULL

Return 0

If temp->data=num

Temp2->next=temp->next

Return 1

4.CODE:

// To implement Hash ADT using separate chaining

#include<stdio.h>

#include<stdlib.h>

#define size 25

class hash

{

    struct node

    {

        int data;

        struct node \*next;

    };

    struct node \*hashtable[size];

    struct node \*head;

    public:

    hash()

    {

        for (int i=0;i<size;i++)

        {

            hashtable[i]=NULL;

        }

    }

    int insert(int num);

    void display();

    void ldis(struct node \*head);

    int del(int num);

    int search(int num);

};

int main()

{

    hash h1;

    int choice;

    while(1)

    {

        printf("\n(1) Insert");

        printf("\n(2) Delete");

        printf("\n(3) Search");

        printf("\n(4) Display");

        printf("\n(5) Exit");

        printf("\n Enter your choice");

        scanf("%d",&choice);

        switch (choice)

        {

            case 1:

            int num1;

            printf("Enter the number to insert");

            scanf("%d",&num1);

            if(h1.insert(num1))

            {

               printf("Element is inserted successfully");

            }

            else

            {

                printf("fail");

            }

            break;

            case 2:

            int num2;

            printf("Enter the number to delete");

            scanf("%d",&num2);

            if(h1.del(num2))

            {

                printf("Number is deleted successfully");

            }

            else

            {

                printf("Element is not present");

            }

            break;

            case 3:

            int num3;

            printf("Enter the element to search");

            scanf("%d",&num3);

            if(h1.search(num3))

            {

                printf("Element is found");

            }

            else

            {

                printf("Element is not found");

            }

            break;

            case 4:

            h1.display();

            break;

            case 5:

            return 0;

        }

    }

}

//method to insert the element

int hash::insert(int num)

{

    struct node \*newnode =(struct node \*)malloc(sizeof(struct node));

    newnode->data=num;

    newnode->next=NULL;

    int index;

    index=num%size;

    if(hashtable[index]==NULL)

    {

        hashtable[index]=newnode;

        return 1;

    }

    else

    {

        struct node \*temp=hashtable[index];

        while(temp->next!=NULL)

        {

            temp=temp->next;

        }

        temp->next=newnode;

        return 1;

    }

}

//method to display the elements

void hash:: display()

{

   for(int i=0;i<size;i++)

   {

    if(hashtable[i]!=NULL)

    {

        ldis(hashtable[i]);

    }

   }

}

void hash::ldis(struct node \*head)

{

    struct node \*temp=head;

    while(temp!=NULL)

    {

        printf("\t%d",temp->data);

        temp=temp->next;

    }

}

//method to delete a element

int hash :: del(int num)

{

    int index=num%size;

    if(hashtable[index]==NULL)

    {

        return 0;

    }

    else

    {

    struct node \*temp1=hashtable[index];

    struct node \*temp2=(struct node \*)malloc(sizeof(struct node));

    if(temp1->data==num)

    {

        hashtable[index]=temp1->next;

        return 1;

    }

    else

    {

        while(temp1!=NULL && temp1->data!=num)

        {

        temp2=temp1;

        temp1=temp1->next;

        }

        if(temp1==NULL)

        {

            return 0;

        }

        if(temp1->data==num)

        {

        temp2->next=temp1->next;

        return 1;

        }

    }

    }

    return 0;

}

//method to search an element

int hash:: search(int num)

{

    for(int i=0;i<size;i++)

    {

        if(hashtable[i]!=NULL)

        {

        struct node \*temp=hashtable[i];

        if(temp->data==num)

        {

            return 1;

        }

        else

        {

          while(temp!=NULL)

          {

           if(temp->data==num)

           {

            return 1;

           }

           temp=temp->next;

          }

        }

        }

    }

    return 0;

}

5.OUTPUT:

A screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated