PRACTICAL FILE

DATA STRUCTURES

Data Structures Lab

Submitted By

Name: Khushi Wadhawan

Roll No.: R2142210423

SAP ID: 500093673

B. Tech CSE 2nd Sem
School of Computer Science

Submitted to

Mr. Amit Verma

School of Computer Science, UPES

School of Computer Science

University of Petroleum and Energy Studies,

Dehradun – 248007: Uttarakhand

EXPERIMENT 1:

Data Abstraction:

Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

For example, a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, but you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

CODE:

```
#include <iostream>
using namespace std;

class Adder {
   public:
      // constructor
      Adder(int i = 0) {
         total = i;
      }

      // interface to outside world
      void addNum(int number) {
         total += number;
      }

      // interface to outside world
      int getTotal() {
         return total;
      };

   private:
      // hidden data from outside world
      int total;
};

int main() {
   Adder a;

   a.addNum(10);
   a.addNum(20);
   a.addNum(30);

   cout << "Total " << a.getTotal() <<endl;
   return 0;
}
```

Constructor:

A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object is created. It is special member function of the class because it does not have any return type.

1. *Default constructor:*

Default constructor is the constructor which doesn't take any argument. It has no parameters.

Eg:

```
#include <iostream>
using namespace std;

class construct
{
public:
    int a, b;

    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};

int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```

2. *Parameterized constructor*:
   It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function.

   Eg:

```
#include <iostream>
using namespace std;

class Point
{
private:
    int x, y;

public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    int getX()
    {
```

```
            return x;
        }
        int getY()
        {
            return y;
        }
    };

    int main()
    {
        // Constructor called
        Point p1(10, 15);

        // Access values assigned by constructor
        cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

        return 0;
    }
```

Destructor:

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

CODE:

```
#include<iostream>
using namespace std;
class Demo {
   private:
   int num1, num2;
   public:
   Demo(int n1, int n2) {
      cout<<"Inside Constructor"<<endl;
      num1 = n1;
      num2 = n2;
   }
   void display() {
      cout<<"num1 = "<< num1 <<endl;
      cout<<"num2 = "<< num2 <<endl;
   }
   ~Demo() {
      cout<<"Inside Destructor";
   }
};
int main() {
   Demo obj1(10, 20);
   obj1.display();
   return 0;
}
```

Constructor overload:

Overloaded constructors have the same name (name of the class) but the different number of arguments. Depending upon the number and type of arguments passed, the corresponding constructor is called.

CODE:

```
#include <iostream>
using namespace std;

class Person {
   private:
    int age;

   public:
    // 1. Constructor with no arguments
    Person() {
        age = 20;
    }

    // 2. Constructor with an argument
    Person(int a) {
        age = a;
    }

    int getAge() {
        return age;
    }
};

int main() {
    Person person1, person2(45);

    cout << "Person1 Age = " << person1.getAge() << endl;
    cout << "Person2 Age = " << person2.getAge() << endl;

    return 0;
}
```

EXPERIMENT 2:

1. Write a program to find the sum of the elements of an array using recursion.

CODE:

```
# include <iostream>
using namespace std;
int sumarr(int *a,int n, int size)
{
        int sum;
        if(n>=size)
        {
            return 0;
        }
        else
        {
            sum = a[n] + sumarr(a,n+1,size);
        }
        return sum;
}
int main()
{
        int size, sum;
        cout << "\n Enter the size of the array :";
        cin >> size;
        int arr[size];
        cout << "\n Enter the Array :";
        for(int i=0; i<size; i++)
        {
            cin >> arr[i];
        }
        sum = sumarr(arr,0,size);
        cout << "\n The sum of array is : " << sum;
        return 0;
}
```

2. Program to convert uppercase string to lowercase using for loop.

CODE:

```
# include <iostream>
using namespace std;

int main()
{
    char str[100];
    int i, size =0;
    cout << "\n Enter the string : ";
    cin.getline(str,100);
    for(i=0; str[i] != '\0'; i++)
    {
        size ++;
    }
```

```
    for( i =0; i< size; i++)
    {
        if((str[i] >= 65 && str[i] <= 90))
        {
            str[i] =   str[i] + 32;

        }
    }
    cout << "The new string is : " << str;
    return 0;
}
```

3. Find the transpose and inverse of the matrix.

TRANSPOSE CODE:

```
# include <iostream>
using namespace std;
int main()
{
        int m,n, count =0;
        cout << "\nEnter the number of rows :";
        cin >> m;
        cout << "\nEnter the number of columns :";
        cin >>n;
        int arr[m][n],b[n][m];
        cout << "\nEnter the matrix :";
        for(int i=0; i<m; i++)
        {
            for(int j=0; j<n; j++)
            {
                  cin >> arr[i][j];
            }
        }
        cout << "\n \t **** THE MATRIX IS : ****\n";
        for(int i=0; i<m; i++)
        {
            for(int j=0; j<n; j++)
            {
                  cout << arr[i][j] << "\t";
                  b[j][i] = arr[i][j];
            }
            cout << endl;
        }
        cout << "\n \t ***** TRANSPOSE MATRIX : ****\n";
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<m; j++)
            {
                  cout << b[i][j] << "\t";
            }
            cout << endl;
        }
        return 0;
}
```

INVERSE CODE:

```
# include <stdio.h>
int determinant(float [10][10], int);
int main()
{
        float a[10][10];
        int size, d=0;
        printf("\n Enter size of square matrix : ");
        scanf("%d", &size);
        printf("\n Enter the %d * %d matrix : ",size,size);
        for(int i=0; i<size; i++)
        {
            for(int j=0; j<size; j++)
            {
                  printf("\n Enter element A[%d][%d] : ", i+1, j+1);
                  scanf("%f", &a[i][j]);
            }
        }
        float b[10][10], c[10][10], in[10][10];
        int x, y ,sign;
        for(int w=0; w<size; w++)
        {
            (w%2 == 0)? sign =1 : (sign=-1);

            for(int z=0; z<size ;z++)

            {
                  x =0, y=0;
                  for(int i =0; i<size; i++)
                  {
                        for(int j=0; j<size; j++)
                        {
                              if(i!=w && j!=z)
                              {
                                    b[x][y] = a[i][j];
                                    if(y<(size-1)) y++;
                                    if(y>=(size-1))
                                    {
                                          y =0;
                                          x++;
                                    }
                              }
                        }
                  }
                  c[w][z] = determinant(b,size-1) * sign;
                  sign *= (-1);
            }

        }
        for( int i =0; i<size; i++)
        {
            for(int j=0; j<size; j++)
            {
                  b[j][i] = c[i][j];
            }
        }
```

```
        d = determinant(a, size);
        if(d==0)
        {
            printf("\n Inverse doesn't exist");
            return 0;
        }
        if(size == 1)
        {
            printf("\n Inverse of martrix : %d", a[0][0]);
            return 0;
        }
        printf("\n Inverse of given matrix : \n");
        for(int i=0; i<size;i++)
        {
            for(int j=0; j<size;j++)

            {
                  in[i][j] = (b[i][j])/(d);

                  printf("  %f  ", in[i][j]);
            }
            printf("\n");
        }
        return 0;
}
int determinant(float a[10][10], int size)
{
        float b[10][10];
        int d = 0, x, y , sign =1;
        if(size == 1)  return a[0][0];
        else{
            for(int z=0; z<size ;z++)
            {
                  x =0; y=0;
                  for(int i=0; i<size; i++)
                  {
                        for(int j=0; j<size; j++)
                        {
                              if(i!=0 && j!=z)
                              {
                                    b[x][y] = a[i][j];
                                    y++;
                                    if(y>=(size-1))
                                    {
                                          y=0;
                                          x++;
                                    }
                              }
                        }
                  }
                  d = d + sign*(a[0][z]*determinant(b,size-1));
                  sign = sign*(-1);
            }

            return d;
        }
}
```

*4. Find if the given matrix of order (m*n) is a sparse matrix of not.*

CODE:

```
# include <iostream>
using namespace std;
int main()
{
        int m,n, count =0;
        cout << "\nEnter the number of rows :";
        cin >> m;
        cout << "\nEnter the number of columns :";
        cin >>n;
        int arr[m][n];
        cout << "\nEnter the matrix :";
        for(int i=0; i<m; i++)
        {
            for(int j=0; j<n; j++)
            {
                  cin >> arr[i][j];
            }
        }
        cout << "\n \t ******* THE MATRIX IS : *******\n";
        for(int i=0; i<m; i++)
        {
            for(int j=0; j<n; j++)
            {
                  cout << arr[i][j] << "\t";
                  if(arr[i][j]==0)
                  {
                        count ++;
                  }
            }
            cout << endl;
        }
        if(count > m)
        {
            cout << "\nThe given matrix is sparse matrix ..";
        }
        else
        {
            cout << "\nThe given matrix is NOT sparse matrix ..";
        }
        return 0;
}
```

5. Find the largest and the smallest number in the given array.

CODE:

```
# include <iostream>
using namespace std;
int main()
{
        int size, sum;
        cout << "\n Enter the size of the array :";
```

```
    cin >> size;
    int arr[size];
    cout << "\n Enter the Array :";
    for(int i=0; i<size; i++)
    {
        cin >> arr[i];
    }
    int max = arr[0], min = arr[0];
    cout << "\nThe Array is : ";
    for(int i=0; i<size; i++)
    {
        cout << arr[i] << " ";
        if(max < arr[i])
        {
              max = arr[i];
        }
        if(min > arr[i])
        {
              min = arr[i];
        }
    }
    cout << "\n The maximun element of the Array is :" <<
  max;
    cout << "\n The minimun element of the Array is :" <<
  min;
    return 0;
}
```

EXPERIMENT 3:

1. Design a structure with (int) and one (float) data member and display their sum as the result.

CODE:

```
# include <iostream>
using namespace std;
struct ab{
        int a;
        float b;
        float sum ;
        void input()
        {
            cout << "\n Enter value of a :";
            cin >>a;
            cout << "\n Enter the value of b :";
            cin >> b;
            sum = cal_sum(a,b);
        }
        float cal_sum(int x, float y)
        {
            float sum = x+y;
            return sum;
        }
        void display()
        {
            cout << "\n Sum of both : " << sum;
        }
};
int main()
{
        struct ab v;
        v.input();
        v.display();
        return 0;
}
```

2. Design a structure with (int) and one (float) data member and display their sum as the result.

CODE:

```
# include <iostream>
using namespace std;
struct ab{
        int a;
        float b;
};
int main()
{
        struct ab v;
        struct ab *v1  = &v;
        cout << "\n Enter value of a :";
        cin >> v1->a;
```

```
        cout << "\n Enter the value of b :";
        cin >> v1->b;
        float sum = v1->a + v1->b;
        cout << "\n Sum : " << sum;
}
```

3. Design a structure with two integer data members and an array of structure of size 4. Input the data into the structure and display it.

CODE:

```
# include <iostream>
using namespace std;
struct ab{
        int a;

        int b;
        void input()
        {
            cout << "\n Enter value of a :";
            cin >>a;
            cout << " Enter the value of b :";
            cin >> b;
        }
        void display()
        {
            cout << "\n Value of a and b is : " << a << "and"<<b;
        }
};
int main()
{
        struct ab arr[4];
        int i;
        for(i=0; i<4;i++)
        {
            cout << "\n Enter data " << i+1 << ":";
            arr[i].input();
        }
        for(i=0; i<4;i++)
        {
            cout << "\n Data " << i+1 <<":";
            arr[i].display();
        }

        return 0;
}
```

4. Design a structure to maintain a record of the students with roll number, sap id, and semester. Create an array of structure to input data of 5 students and display them.

CODE:

```
# include <iostream>
using namespace std;
struct ab{
        int roll_no, semester;
        char sap_id[10];
```

```
        void input()
        {
            cout << "\n Enter roll no :";
            cin >> roll_no;
            cout << " Enter sap id :";
            cin >> sap_id
            ;cout << " Enter semester : ";
            cin >> semester;
        }
        void display()
        {
            cout << "\n Roll no :" << roll_no;
            cout << "\n Sap id :" << sap_id;
            cout << "\n Semester : " << semester;
        }
};
int main()
{
        struct ab arr[5];
        int i;
        for(i=0; i<5;i++)
        {
            cout << "\n Enter details of student " << i+1 << ":";
            arr[i].input();
        }
        for(i=0; i<5;i++)
        {
            cout << "\n \t\tStudent " << i+1 <<":";
            arr[i].display();
        }

        return 0;
}
```

5. Design a structure with two data members (integer and float) dynamically allocate the memory to data members and display the data of structure.

CODE:

```
# include <iostream>
# include <malloc.h>
using namespace std;
struct ab{
        int a;
        float b;
};
int main()
{
        struct ab *v;
        int i,n;
        cout <<"\n Enter number of records : ";
        cin >>n;
        v = (struct ab*) malloc(n*sizeof(struct ab));

        for(i=0; i<n;i++){
            cout << "\n For " << i+1 << ":";
            cout << "\n Enter any integer : ";
            cin >> (v+i)->a;
            cout << " Enter any float value : ";
```

```
            cin >> (v+i)->b;
        }
        for (i=0; i<n;i++)
        {
            cout << "\n Values are : " << (v+i)->a << " and " <<
        (v+i)->b;
        }
        free(v);
        return 0;
}
```

EXPERIMENT 4:

1. What are Structures? Explain with the help of examples

A structure is a key word that create user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Eg:

```
#include <stdio.h>
#include <string.h>

// create struct with person1 variable
struct Person {
  char name[50];
  int citNo;
  float salary;
} person1;

int main() {

  // assign value to name of person1
  strcpy(person1.name, "George Orwell");

  // assign values to other person1 variables
  person1.citNo = 1984;
  person1. salary = 2500;

  // print struct variables
  printf("Name: %s\n", person1.name);
  printf("Citizenship No.: %d\n", person1.citNo);
  printf("Salary: %.2f", person1.salary);

  return 0;
}
```

2. What are pointers? Explain with the help of examples.

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

Eg:

```
#include <stdio.h>
int main()
{
   int* pc, c;

   c = 22;
   printf("Address of c: %p\n", &c);
   printf("Value of c: %d\n\n", c);  // 22

   pc = &c;
   printf("Address of pointer pc: %p\n", pc);
   printf("Content of pointer pc: %d\n\n", *pc); // 22
```

```
    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc); // 11

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c); // 2
    return 0;
}
```

3. What is Dynamic Memory Allocation? Explain with an example.

Dynamic memory allocation is a way to allocate memory to a data structure during the runtime we can use DMA function available in C to allocate and free memory during runtime.

Following functions are available in C to perform dynamic memory allocation:

1. malloc()
2. calloc()
3. free()
4. realloc()

malloc() function:

Malloc stands for memory allocation. It takes number of bytes to be allocated as an input and returns a pointer of type void.

Eg:

```
#include <iostream>
# include <malloc.h>
using namespace std;
int main()
{
        int *ptr = (int*)malloc(30*sizeof(int));
        cout << endl << ptr;
        cout << endl << sizeof(ptr);
        return 0;

}
```

4. Write a program to input and print an array using pointers.

CODE:

```
#include <iostream>
# include <malloc.h>
using namespace std;
int main()
{
        int a[5], i;
```

```
        int *ptr;
        ptr  = &a[0];
        cout << "\n Enter an Array : ";
        for(i = 0; i<5; i++)
        {
            cin >> *(ptr+i);
        }
        cout << "\n The Array is :";
        for( i=0; i<5 ;i++)
        {
            cout << *(ptr+i) << " ";
        }
        return 0;
}
```

5. Write a program to create an array of structure having two integers as data members. Input/ Print the element of array.

CODE:

```
# include <iostream>
using namespace std;
struct ab{
        int a;
        int b;
        void input()
        {
            cout << "\n Enter value of a :";
            cin >>a;
            cout << " Enter the value of b :";
            cin >> b;
        }
        void display()
        {
            cout << "\n Value of a & b is : " << a << "and" << b;
        }
};
int main()
{
        struct ab arr[4];
        int i;
        for(i=0; i<4;i++)
        {
            cout << "\n Enter data " << i+1 << ":";
            arr[i].input();
        }
        for(i=0; i<4;i++)
        {
            cout << "\n Data " << i+1 <<":";
            arr[i].display();
        }
        return 0;
}
```

6. Write a program to create and perform following operation on single linked list.

a. *Insertion at the beginning.*
b. *Insertion at the end.*

c. *Deletion from the beginning.*
d. *Deletion from the end.*
e. *Traversing.*

CODE:

```
# include <stdio.h>
# include <malloc.h>
#include <stdlib.h>
struct node{
        int a;
        struct node *link;
};
struct node *head = NULL;
struct node *temp;
void insert_beg(struct node*);
void insert_end(struct node*);
void delete_beg();
void traverse();
int main()
{
        struct node *ptr;
        int input,i;
        while(1)
        {
            printf("\n Enter Your choice :\n1. Insert_Beginning
         \n2. Insert_End\n3. Delete_Beg \n4. Traverse \n5.
          Exit\n ");
            scanf("%d",&input);

            switch(input)
            {
                  case 1:
                        ptr = (struct node*)malloc(sizeof(struct  node));
                        printf("\n Enter the data :");
                        scanf("%d", &ptr->a);
                        ptr->link = NULL;
                        insert_beg(ptr);
                        break;

                  case 2:
                        ptr = (struct node*)malloc(sizeof(struct node));
                        printf("\n Enter the data :");
                        scanf("%d", &ptr->a);
                        ptr->link = NULL;
                        insert_end(ptr);
                        break;
                  case 3:
                        delete_beg();
                        break;
                  case 4:
                        traverse();
                        break;
                  default:
                        exit(1);
            }

        }
        return 0;
```

```
}
void insert_beg(struct node *ptr)
{
        if(head == NULL)
        {
            head = ptr;
        }
        else{
            ptr->link = head;
            head = ptr;
        }
}
void insert_end(struct node *ptr)
{
        if(head == NULL)
        {
            head = ptr;
        }
        else{
            temp = head;
            while(temp->link != NULL)
            {
                  temp = temp -> link;
            }
            temp->link = ptr;
        }
}

void delete_beg(ptr)
{
        if(head == NULL)
        {

            printf("\n No data to delete ..");
        }
        else{
            head = head->link;
        }
}
void traverse(ptr)
{
        temp = head;
        while(temp != NULL)
        {
            printf("%d ",temp->a);
            temp = temp->link;
        }
}
```

EXPERMIENT 5:

1. What is linked list? Mention one drawback of single linked list.

It is a data structure made up of a chain of nodes in which each node contains a value and a pointer to the nex node in the chain.

Drawback:
More memory is required in linked list as compared to an array. Because in linked list, a pointer is also required to store the address of the next element and requires extra memory for itself.
Also, we can't access the previous element in single linked list.

2. What is the difference between single and circular linked list?

In circular linked list, the pointer part of the last node points to the first node i.e. head and not NULL

as in the single linked list.

3. Implement circular linked list:

a) *Insertion at beginning*
b) *Insertion at end*
c) *Deletion from the end*
d) *Traversing the list*

CODE:

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

struct node{
      int data;
      struct node *next;
      struct node *pre;
};
struct node *head = NULL;
struct node *end = NULL;
struct node *temp;

void ins_beg(struct node*);
void ins_end(struct node*);
void del_beg();
void del_end();
void traverse();
int main(){
      int choice;
      struct node *new1;
      while(1){
            printf("\nEnter the choice\n");
            printf("1. ins_beg\n2. ins_end\n3. del_beg\n4. del_end \n5.
traverse \n6. exit\n");
            scanf("%d",&choice);
```

```
            switch(choice){
                  case 1:
                        new1 = (struct node*)malloc(sizeof(struct node));
                        printf("enter the data\n");
                        scanf("%d",&new1->data);
                        new1->next = new1;
                        new1->pre = new1;
                        ins_beg(new1);
                        break;
                  case 2:
                        new1 = (struct node*)malloc(sizeof(struct node));
                        printf("enter the data\n");
                        scanf("%d",&new1->data);
                        new1->next = new1;
                        new1->pre = new1;
                        ins_end(new1);
                        break;
                  case 3:
                        del_beg();
                        break;
                  case 4:
                        del_end();
                        break;
                  case 5:
                        traverse();
                        break;
                  default:
                        exit(1);
            }
      }
}
void ins_beg(struct node *new1){
      if(head == NULL)
      {
            head = new1;
            end = new1;
      }
      else{
            new1->next = head;
            head->pre = new1;
            new1->pre = end;
            end->next = new1;
            head = new1;
      }
}

void ins_end(struct node *new1){
      if(head == NULL)
      {
            head = new1;
            end = new1;
      }
      else{
            end->next = new1;
            new1->pre = end;
            new1->next = head;
            head->pre = new1;
            end = new1;
      }
}
void del_beg(){
```

```
      if(head == NULL)
            printf("No item to delete\n");
      else{
            printf("%d Deleted", head->data);
            head = head->next;
            head->pre = end;
            end->next = head;
    }

}
void del_end()
{
      if(head == NULL)
      {
            printf(" No item to delete\n");
      }
      else{
            printf("%d Deleted", end->data);
            head->next = end->pre;
            end->pre->next = head;
            end = end->pre;
      }
}
void traverse()
{
      temp = head;
      do{
            printf("%d ",temp->data);
            temp = temp->next;
      }
      while(temp!=end->next);
}
```

EXPERIMENT 6:

1. Read about Stack. Implement stack using arrays and perform following functions:

a. *push()*
b. *pop()*
c. *traverse()*
d. *exit()*

CODE:

```
# include <stdio.h>
# include <stdlib.h>
# define MAX 10
int a[MAX];
int top = -1;
void push(int );
void pop();
void traverse();
int main()
{
          int choice;
          int n;
          while(1)
          {
             printf("\nEnter the choice: \n1.PUSH \n2.POP \n3.TRAVERSE
\n4.exit\n");
             scanf("%d",&choice);
             switch(choice)
             {
                   case 1:
                         printf("\nEnter the data :");
                         scanf("%d",&n);
                         push(n);
                         break;
                   case 2:
                         pop();
                         break;
                   case 3:
                         traverse();
                         break;
                   case 4:
                         exit(1);
             }
          }
}

void push(int n)
{
          if(top == MAX-1)
          {
             printf("\nStack Overflow ..");
          }
          else
          {
             a[top+1] = n;
             top ++;
          }
```

```
}
void pop()
{
          if(top == -1)
          {
             printf("\nStack Underflow ..");
          }
          else
          {
             printf("\n%d DELETED !",a[top]);
             top --;
          }
}
void traverse()
{
          if(top == -1)
          {
             printf("\nStack Underflow ..");
          }
          else
          {
             int temp = top;
             while(temp > -1)
             {
                   printf("%d ",a[temp]);
                   temp --;
             }
          }
}
```

---

2. Implement all the above functions using Linked list.

CODE:

```
# include <stdio.h>
# include <stdlib.h>
# define MAX 10
struct node{
          int data;
          struct node *next;
};
struct node *top = NULL;
void push(struct node *);
void pop();
void traverse();
int main()
{
          struct node *ptr;
          int choice;
          int n;
          while(1)
          {
             printf("\nEnter the choice: \n1.PUSH \n2.POP \n3.TRAVERSE
\n4.exit\n");
             scanf("%d",&choice);
             switch(choice)
             {
```

```
                case 1:
                      ptr = (struct node*)malloc(sizeof(struct node));
                      printf("\nEnter the data :");
                      scanf("%d",&ptr->data);
                      ptr->next = NULL;
                      push(ptr);
                      break;
                case 2:
                      pop();
                      break;
                case 3:
                      traverse();
                      break;
                case 4:
                      exit(1);
            }
          }
}
void push(struct node *ptr)
{

          if(top == NULL)
          {
            top = ptr;
          }
          else
          {
            ptr->next = top;
            top = ptr;
          }
}
void pop()
{
          if(top == NULL)
          {
            printf("\nEmpty Stack ..");
          }
          else
          {
            printf("\n%d DELETED !",top->data);
            top = top->next;
          }
}
void traverse()
{
          if(top == NULL)
          {
            printf("\nEmpty Stack ..");
          }
          else
          {
            struct node *temp = top;
            while(temp != NULL)
            {
                  printf("%d ",temp->data);
                  temp = temp->next;
            }
          }
}
```

EXPERIMENT 7:

1. Write a code to create binary tree.

CODE:

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

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

// Inorder traversal
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ", root->item);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ", root->item);
}

// Create a new Node
struct node* newnode(int value) {
  struct node* new1 = malloc(sizeof(struct node));
  new1->item = value;
  new1->left = NULL;
  new1->right = NULL;

  return new1;
}

// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = newnode(value);
  return root->left;
}

// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = newnode(value);
  return root->right;
```

```
}

int main() {
     int ch;
     struct node* root = newnode(1);
     insertLeft(root, 2);
     insertRight(root, 3);
     insertLeft(root->left, 4);
     insertRight(root->left, 5);
     insertLeft(root->right, 6);
     insertRight(root->right, 7);


     while(1){
        printf("1. Inorder \n2. pre-order  \n3. post-order traversal\n");
        scanf("%d", &ch);
        switch(ch){
             case 1:

                     inorderTraversal(root);
                     printf("\n");
                     break;
            case 2:
                     preorderTraversal(root);
                     printf("\n");
                     break;
            case 3:
                     postorderTraversal(root);
                     printf("\n");
                     break;
            default:
                     exit(0);

         }
    }
    return 0;
}
```