//1) WAP to use binary operator + add two object of class Numbers having num1 and num2 as its data members and display result.

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

class Numbers{
    int num1, num2;

public:
    Numbers(){
        num1 = 0;
        num2 = 0;
    }

    Numbers(int a, int b){
        num1 = a;
        num2 = b;
    }

    Numbers operator+(Numbers obj){
        Numbers temp;
        temp.num1 = num1 + obj.num1;
        temp.num2 = num2 + obj.num2;
        return temp;

    }
```

```
        void display(){
                cout<< "num = " << num1 << ", num2 = " << num2 << endl;
        }

};

int main(){
        Numbers n1(5,10) , n2(3,7), result;

        cout << "First object: ";
        n1.display();

        cout<< "Second object: ";
        n2.display();

        result = n1 + n2;

        cout << "Result after addition: ";
        result.display();

        return 0 ;

}

//output:
//First object: num = 5, num2 = 10
//Second object: num = 3, num2 = 7
//Result after addition: num = 8, num2 = 17
```

//2) WAP to overload operator * which multiply a number to each element of an array within a class arrayContainer and display the result.

```
#include<iostream>

using namespace std;


class arrayContainer{

        int arr[5];


public:

        arrayContainer(int a,int b, int c,int d,int e){

                arr[0] = a;

                arr[1] = b;

                arr[2] = c;

                arr[3] = d;

                arr[4] = e;

        }


        arrayContainer operator*(int num){

                arrayContainer temp(0,0,0,0,0);

                for(int i=0; i<5; i++){

                        temp.arr[i] = arr[i]*num;

                }

                return temp;

        }

        void dispaly(){

                cout << "Array elements: ";

                for(int i=0; i<5; i++){

                        cout << arr[i] << " ";
```

```
                }
                cout << endl;
        }
};

int main(){
        arrayContainer A(1,2,3,4,5);

        cout << "Original array:";
        A.dispaly();

        arrayContainer B=A*3;

        cout << "After multiplying by 3:";
        B.dispaly();

        return 0;
}



//output:

//Original array:Array elements: 1 2 3 4 5
//After multiplying by 3:Array elements: 3 6 9 12 15
```

```
//3) WAP to Overload the *, +,-, ==, ! = and = operators for the complex class.

#include<iostream>

using namespace std;


class Complex{

        float real,image;


public:

        Complex(){

                real = 0;

                image = 0;


        }

        Complex(float r, float i){

                real = r;

                image = i;

        }


        Complex operator+(Complex c){

                return Complex(real + c.real, image + c.image);

        }


        Complex operator-(Complex c){

                return Complex(real - c.real , image - c.image);

        }


        Complex operator*(Complex c){
```

```
        return Complex((real *c.real - image * c.image),(real * c.image + image *
c.real));
    }

    Complex& operator=(const Complex &c){
        if (this!= &c){
            real = c.real;
            image = c.image;
            }
        return *this;
    }

    bool operator==(Complex c){
        return(real == c.real && image==c.image);
    }

    bool operator!=(Complex c){
        return!(*this == c);
    }

    void display(){
        if(image >= 0)
            cout << real << " + " << image << "i" << endl;
        else
            cout << real << " - " << -image << "i" << endl;

    }
};
```

```
int main(){
        Complex c1(3,2), c2(1,7),result;

        cout << "c1 = ";
        c1.display();
        cout << "c2 = ";
        c2.display();

        result = c1+c2;
        cout << "c1 +c2 = ";
        result.display();


        result = c1-c2;
        cout << "c1 - c2";
        result.display();


        result = c1*c2;
        cout << "c1 * c2";
        result.display();

        Complex c3;
        c3=c1;
        cout << "After assignment c3=";
        c3.display();
```

```
        if(c1 == c3)
                cout<< "c1 and c3 are equal" << endl;
        else
                cout<< "c1 and c3 are not equal"<< endl;

        if(c1 != c3)
                cout<< "c1 and c3 are not equal"<< endl;
        else
                cout<<          "c1 and c3 are equal" << endl;

        return 0;

}

//output

//c1 = 3 + 2i
//c2 = 1 + 7i
//c1 +c2 = 4 + 9i
//c1 - c22 - 5i
//c1 * c2-11 + 23i
//After assignment c3=3 + 2i
//c1 and c3 are equal
//c1 and c3 are equal
```

//4) WAP to define an object m1 of matrix class, use m1<<cout.

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

class Matrix{
	int mat[2][2];

public:
	Matrix(int a, int b, int c, int d){
		mat[0][0] = a;
		mat[0][1] = b;
		mat[1][0] = c;
		mat[1][1] = d;
	}

	void operator << (ostream &out){
		out << "Matrix:" << endl;
		for(int i = 0 ; i < 2; i++){
			for(int j=0;j<2;j++){
				cout<<mat[i][j] << " ";
			}
			cout << endl;
		}
	}
};

int main(){
```

```
        Matrix m1(1,2,3,4);


        m1 << cout;


        return 0;
}


//output
//Matrix:
//1 2
//3 4
```

//5) WAP to define a matrix class and overload the * operator to multiply a number with matrix (Example: 5*Matrix should be possible).

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

class Matrix{
    int mat[2][2];

public:
    Matrix(int a=0, int b=0, int c=0, int d=0){
        mat[0][0] = a;
        mat[0][1] = b;
        mat[1][0] = c;
        mat[1][1] = d;
    }

    friend Matrix operator*(int num,Matrix m);

    void display(){
        cout << "Matrix : " << endl;
        for(int i =0 ; i<2 ; i++){
            for(int j =0; j<2 ; j++){
                cout << mat[i][j] << " ";
            }
            cout << endl;
        }
    }
```

```
};

Matrix operator*(int num, Matrix m){
	Matrix temp;
	for(int i=0 ; i<2 ; i++){
		for(int j=0 ; j<2 ; j++){
			temp.mat[i][j] = num*m.mat[i][j];
		}
	}
	return temp;
}

int main(){
	Matrix m1(1,2,3,4);

	cout<< "Original matrix:" << endl;
	m1.display();

	Matrix m2 = 5* m1;

	cout << "After multiplying by 5: " << endl;
	m2.display();

	return 0;
}

//output :
```

OOCP Practical Assignment 2 MCA 1 Name : vivek jadav
Roll no : 09

//Original matrix:

//Matrix :

//1 2

//3 4

//After multiplying by 5:

//Matrix :

//5 10

//15 20

//6) WAP to define a class Date with properties int month; int day; int year; overload the following operators.

//5.1) + operator [a+b] (a is of date type and b is an integer), use the assumption that all years all years have 360 days and months 30 days.

//5.2) – operator [a-b(same as above)]

//5.3) = operator

//5.4) <,<=,>,>=

//5.5) ++,--[post and pre both]

```
#include<iostream>

using namespace std;


class Date{

        int day, month, year;


public:

        Date(int d=1, int m=1, int y=2000){

                day =d;

                month = m;

                year = y;

        }


        int toDays() const{

                return year*360 + month*30 + day;

        }


        void fromDays(int total){

                year = total/360;

                total = total %360;
```

```
        month = total/30;
        day = total % 30;

        if (day == 0){
            day = 30;
            month --;
        }
        if(month == 0){
            month = 12;
            year --;
        }
    }

    Date operator+(int days){
        Date temp;
        int total = this->toDays() + days;
        temp.fromDays(total);
        return temp;
    }

    Date operator-(int days){
        Date temp;
        int total = this->toDays() - days;
        temp.fromDays(total);
        return temp;

    }
```

```
Date& operator=(const Date &d){
        if(this !=&d){
                day = d.day;
                month = d.month;
                year = d.year;

        }
        return *this;
}

bool operator<(const Date &d){
        return this->toDays() <d.toDays();
}

bool operator<=(const Date &d){
        return this->toDays() <=d.toDays();
}

bool operator>(const Date &d){
        return this->toDays() >d.toDays();
}

bool operator>=(const Date &d){
        return this->toDays() >=d.toDays();
}

Date& operator++(){
        *this = *this+1;
```

```
            return *this;
        }

        Date operator++(int) {
    Date temp = *this;
    *this = *this + 1;
    return temp;
  }

        Date operator--(){
            *this = *this - 1;
            return *this;
        }

        Date operator--(int){
            Date temp = *this;
            *this = *this - 1;
            return temp;
        }

        void display() const{
            cout << day << "/" << month << "/" << year << endl;
        }
};

int main(){
        Date d1(25,12,2024);
        cout << "Original date";
```

```
d1.display();

Date d2 = d1 + 10;
cout <<"After adding 10 days: ";
d2.display();

Date d3 = d1-40;
cout << "After subtracting 40 days:";
d3.display();

Date d4;
d4=d1;
cout << "After assignment (d4=d1)";
d4.display();

if (d2 >d1) cout << "d2 is later than d1" << endl;
if (d3 < d1) cout<< "d3 is earlier then d1" << endl;

cout << "Pre-increment (++d1):";
(++d1).display();

cout << "Post-increment(d1++):";
(d1++).display();

cout<<"Now d1:";
d1.display();
```

```
        cout<<"Pre-decrement (--d1)";
        (--d1).display();

        cout<<"Post-decrement (d1--)";
        (d1--).display();

        cout<<"Now d1;";
        d1.display();

        return 0;

}

//Output :

//Original date25/12/2024
//After adding 10 days: 5/1/2025
//After subtracting 40 days:15/11/2024
//After assignment (d4=d1)25/12/2024
//d2 is later than d1
//d3 is earlier then d1
//Pre-increment (++d1):26/12/2024
//Post-increment(d1++):26/12/2024
//Now d1:27/12/2024
//Pre-decrement (--d1)26/12/2024
//Post-decrement (d1--)26/12/2024
//Now d1;25/12/2024
```

//7) WAP to define a class Time with properties int hour; int minute; int second; overload the following operators.

//6.1) + operator [a+b] (a is of time type and b is an integer)

//6.2) – operator [a-b(same as above)]

// 6.3) = operator

//6.4) <,<=,>,>=

//6.5) ++,--[post and pre both]

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

class Time{
    int hour,minute,second;

public:
    Time(int h=0,int m=0,int s=0){
        hour = h;
        minute = m;
        second = s;
        normalize();
    }

    int toSeconds() const{
        return hour*3600 + minute*60 + second;
    }

    void fromSeconds(int total){
        if(total < 0)
```

```
                total = 0;
        hour = total / 3600;
        total %= 3600;
        minute = total /60;
        second = total %60;
}

void normalize(){
        int total = toSeconds();
        fromSeconds(total);
}

Time operator+(int sec){
        Time temp;
        int total = this->toSeconds()+sec;
        temp.fromSeconds(total);
        return temp;
}

Time operator-(int sec){
        Time temp;
        int total = this->toSeconds() - sec;
        temp.fromSeconds(total);
        return temp;
}

Time& operator=(const Time &t){
        if(this != &t){
```

```
            hour = t.hour;
            minute = t.minute;
            second = t.second;
        }
        return *this;
    }

    bool operator<(const Time &t){
        return this->toSeconds() < t.toSeconds();
    }

    bool operator<=(const Time &t){
        return this->toSeconds() <= t.toSeconds();
    }

    bool operator>(const Time &t){
        return this->toSeconds() >t.toSeconds();
    }

    bool operator >=(const Time &t){
        return this->toSeconds()>=t.toSeconds();
    }

    Time& operator++(){
        *this = *this+1;
        return *this;
    }
```

```
        Time operator++(int){
                Time temp = *this;
                *this = *this + 1;
                return temp;
        }
        Time& operator--(){
                *this = *this - 1;
                return *this;
        }
        Time operator--(int){
                Time temp = *this;
                *this = *this - 1;
                return temp;
        }

        void display() const{
                cout<< hour << "h : " << minute << "m : "<< second << "s" << endl;
        }
};

int main(){
        Time t1(1,59,50);
        cout<<"Original Time:";
        t1.display();

        Time t2 = t1+20;
        cout<<"After adding 20 seconds: ";
        t2.display();
```

```
Time t3 = t1-100;
cout<<"After subtracting 100 second :";
t3.display();

Time t4;
t4 = t1;
cout << "Ater assignment(t4 = t1)";
t4.display();

if (t2>t1)
        cout<<"t2 is later then t1" << endl;
if(t3<t1)
        cout<<"t3 is earlier then t1" << endl;

cout << "Pre-increment(++t1): ";
(++t1).display();
cout << "Post-increment(t1++): ";
(t1++).display();
cout <<"Now t1:";
t1.display();

cout<<"Pre-decrement (--t1): ";
(--t1).display();
cout<<"Post-decrement (t1--): ";
(t1--).display();
cout<<"Now t1: ";
t1.display();
```

```
        return 0;

}
```

//output :

//Original Time:1h : 59m : 50s

//After adding 20 seconds: 2h : 0m : 10s

//After subtracting 100 second :1h : 58m : 10s

//Ater assignment(t4 = t1)1h : 59m : 50s

//t2 is later then t1

//t3 is earlier then t1

//Pre-increment(++t1): 1h : 59m : 51s

//Post-increment(t1++): 1h : 59m : 51s

//Now t1:1h : 59m : 52s

//Pre-decrement (--t1): 1h : 59m : 51s

//Post-decrement (t1--): 1h : 59m : 51s

//Now t1: 1h : 59m : 50s

//8) Write a menu driven program that can perform the following functions on strings. (Use overloaded operators where possible).(Do not use predefined string function or class.)

//1. Compare two strings for equality (== operator)

//2. Check whether first string is smaller than the second (<= operator)

//3. Copy the string to another

//4. Extract a character from the string (Overload [])

//5. Reverse the string

//6. Concatenate two strings (+ operator)

```
#include<iostream>

using namespace std;


class MyString{

	char str[100];


public:

	MyString(){

		str[0] = '\0';

	}


	MyString(const char s[]){

		int i=0;

		while(s[i] != '\0'){

			str[i] = s[i];

			i++;

		}

		str[i] = '\0';
```

```
}

void display() const{
        cout<< str;
}

bool operator==(const MyString &s){
        int i=0;
        while(str[i] != '\0' && s.str[i] != '\0'){
                if(str[i] != s.str[i])
                        return false;
                i++;
        }
        return(str[i] == '\0' && s.str[i] == '\0');
}

bool operator<=(const MyString &s){
        int i=0;
        while(str[i] != '\0' && s.str[i] != '\0'){
                if(str[i] < s.str[i])
                        return true;
                else if(str[i] > s.str[i])
                        return false;
                i++;
        }
        return(str[i] == '\0');
}
```

```
    MyString& operator=(const MyString &s){
        if(this != &s){
            int i=0;
            while(s.str[i] != '\0'){
                str[i] = s.str[i];
                i++;
            }
            str[i] = '\0';
        }
        return *this;
    }

    char operator[](int index){
        return str[index];
    }

    MyString reverse(){
         MyString temp;
int len=0;
while(str[len] != '\0')
            len++;

for(int i=0; i<len; i++) {
   temp.str[i] = str[len-1-i];
}
temp.str[len] = '\0';
return temp;
    }
```

```
        MyString operator+(const MyString &s) {
    MyString temp;
    int i=0, j=0;
    while(str[i] != '\0') {
      temp.str[i] = str[i];
      i++;
    }
    while(s.str[j] != '\0') {
      temp.str[i] = s.str[j];
      i++; j++;
    }
    temp.str[i] = '\0';
    return temp;
  }

};

int main() {
  MyString s1, s2, s3;
  int choice;
  char input[100];

  cout << "Enter first string: ";
  cin >> input;
  s1 = MyString(input);

  cout << "Enter second string: ";
```

```
  cin >> input;
  s2 = MyString(input);

  do {
    cout << "\n--- MENU ---\n";
    cout << "1. Compare two strings for equality (==)\n";
    cout << "2. Check if first string <= second string\n";
    cout << "3. Copy first string into another\n";
    cout << "4. Extract a character using []\n";
    cout << "5. Reverse first string\n";
    cout << "6. Concatenate two strings (+)\n";
    cout << "7. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch(choice) {
      case 1:
        if(s1 == s2)
                                    cout << "Strings are equal\n";
        else
                                    cout << "Strings are not equal\n";
        break;

      case 2:
        if(s1 <= s2)
                                    cout << "First string is smaller or equal\n";
        else
                                    cout << "First string is greater\n";
```

```
        break;

    case 3:
        s3 = s1;
        cout << "Copied string: ";
        s3.display();
        cout << endl;
        break;

    case 4: {
        int index;
        cout << "Enter index: ";
        cin >> index;
        cout << "Character at index " << index << ": " << s1[index] << endl;
        break;
    }

    case 5:
        s3 = s1.reverse();
        cout << "Reversed string: ";
        s3.display();
        cout << endl;
        break;

    case 6:
        s3 = s1 + s2;
        cout << "Concatenated string: ";
        s3.display();
```

```
            cout << endl;
            break;

        case 7:
            cout << "Exiting..." << endl;
            break;

        default:
            cout << "Invalid choice!" << endl;
      }
   } while(choice != 7);

   return 0;
}
/*
output :

Enter first string: abcd
Enter second string: efgh

--- MENU ---
1. Compare two strings for equality (==)
2. Check if first string <= second string
3. Copy first string into another
4. Extract a character using []
5. Reverse first string
6. Concatenate two strings (+)
7. Exit
```

Enter your choice: 1

Strings are not equal

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice: 2

First string is smaller or equal

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice: 3

Copied string: abcd

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice: 4

Enter index: 1

Character at index 1: b

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice: 5

Reversed string: dcba

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice: 6

Concatenated string: abcdefgh

--- MENU ---

1. Compare two strings for equality (==)

2. Check if first string <= second string

3. Copy first string into another

4. Extract a character using []

5. Reverse first string

6. Concatenate two strings (+)

7. Exit

Enter your choice:7*/

```
//9) WAP to Overload the New and Delete for Stack Class.
#include <iostream>
#include <cstdlib>
using namespace std;

class Stack {
  int *arr;
  int top;
  int size;

public:

  Stack(int s = 5) {
    size = s;
    arr = new int[size];
    top = -1;
  }



  ~Stack() {
    delete[] arr;
  }



  void push(int x) {
    if (top == size - 1) {
      cout << "Stack Overflow!" << endl;
      return;
```

```
    }
    arr[++top] = x;
}

int pop() {
    if (top == -1) {
        cout << "Stack Underflow!" << endl;
        return -1;
    }
    return arr[top--];
}

void display() {
    if (top == -1) {
        cout << "Stack Empty!" << endl;
        return;
    }
    cout << "Stack: ";
    for (int i = 0; i <= top; i++)
        cout << arr[i] << " ";
    cout << endl;
}

void* operator new(size_t sz) {
    cout << "[Custom new called] Allocating " << sz << " bytes" << endl;
```

```
    void* p = malloc(sz);
    if (!p) throw bad_alloc();
    return p;
  }

  void operator delete(void* p) {
    cout << "[Custom delete called] Freeing memory" << endl;
    free(p);
  }
};

int main() {

  Stack* s = new Stack(5);

  s->push(10);
  s->push(20);
  s->push(30);
  s->display();

  cout << "Popped: " << s->pop() << endl;
  s->display();

  delete s;
```

```
    return 0;
}

/*
output:

[Custom new called] Allocating 16 bytes
Stack: 10 20 30
Popped: 30
Stack: 10 20
[Custom delete called] Freeing memory
*/
```

//10) Write a template function to make sum of two numbers.

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



template <typename T>
T sum(T a, T b) {
  return a + b;
}



int main() {
  cout << "Sum of integers: " << sum(5, 10) << endl;
  cout << "Sum of floats: " << sum(2.5f, 3.7f) << endl;
  cout << "Sum of doubles: " << sum(4.123, 7.456) << endl;
  cout << "Sum of characters: " << sum('A', 'B') << " (ASCII sum)" << endl;

  return 0;
}

/*
Output :
Sum of integers: 15
Sum of floats: 6.2
Sum of doubles: 11.579
Sum of characters: â (ASCII sum)
*/
```

```
//11) Write a program to generate templates function for swapping values of
//variables and show its use with integer, float and character type of data as
//input.

#include <iostream>
using namespace std;



template <typename T>
void swapValues(T &a, T &b) {
  T temp = a;
  a = b;
  b = temp;
}

int main() {
  int x = 10, y = 20;
  float p = 2.5f, q = 7.8f;
  char c1 = 'A', c2 = 'Z';

  cout << "Before swapping:" << endl;
  cout << "x = " << x << ", y = " << y << endl;
  cout << "p = " << p << ", q = " << q << endl;
  cout << "c1 = " << c1 << ", c2 = " << c2 << endl;



  swapValues(x, y);
  swapValues(p, q);
```

```
    swapValues(c1, c2);

    cout << "\nAfter swapping:" << endl;
    cout << "x = " << x << ", y = " << y << endl;
    cout << "p = " << p << ", q = " << q << endl;
    cout << "c1 = " << c1 << ", c2 = " << c2 << endl;

    return 0;
}

/*
Output :

Before swapping:
x = 10, y = 20
p = 2.5, q = 7.8
c1 = A, c2 = Z

After swapping:
x = 20, y = 10
p = 7.8, q = 2.5
c1 = Z, c2 = A

*/
```

```
//12) Write an object-oriented program to implement a generic Number Class that
//can accept either int or float data type and perform basic calculation like +,-,/
//and *.

#include <iostream>
using namespace std;


template <typename T>
class Number {
  T value;

public:

  Number(T v = 0) {
    value = v;
  }


  T getValue() const {
    return value;
  }


  Number operator+(const Number &obj) {
    return Number(value + obj.value);
  }
```

```
    Number operator-(const Number &obj) {
        return Number(value - obj.value);
    }

    Number operator*(const Number &obj) {
        return Number(value * obj.value);
    }

    Number operator/(const Number &obj) {
        if (obj.value == 0) {
            cout << "Error: Division by zero!" << endl;
            return Number(0);
        }
        return Number(value / obj.value);
    }

    void display() const {
        cout << value;
    }
};


int main() {

    Number<int> n1(20), n2(10);
    cout << "Integer Operations:" << endl;
    cout << "n1 + n2 = "; (n1 + n2).display(); cout << endl;
    cout << "n1 - n2 = "; (n1 - n2).display(); cout << endl;
```

```
    cout << "n1 * n2 = "; (n1 * n2).display(); cout << endl;
    cout << "n1 / n2 = "; (n1 / n2).display(); cout << endl;



    Number<float> f1(5.5f), f2(2.2f);
    cout << "\nFloat Operations:" << endl;
    cout << "f1 + f2 = "; (f1 + f2).display(); cout << endl;
    cout << "f1 - f2 = "; (f1 - f2).display(); cout << endl;
    cout << "f1 * f2 = "; (f1 * f2).display(); cout << endl;
    cout << "f1 / f2 = "; (f1 / f2).display(); cout << endl;


    return 0;
}


/*
Output :
Integer Operations:
n1 + n2 = 30
n1 - n2 = 10
n1 * n2 = 200
n1 / n2 = 2
Float Operations:
f1 + f2 = 7.7
f1 - f2 = 3.3
f1 * f2 = 12.1
f1 / f2 = 2.5
*/
```

//13) Write an object-oriented program to implement a generic Stack. Incorporate

//all the possible operation on Stack in the program.

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


template <typename T>
class Stack {
  T *arr;
  int top;
  int capacity;

public:

  Stack(int size = 10) {
    capacity = size;
    arr = new T[capacity];
    top = -1;
  }


  ~Stack() {
    delete[] arr;
  }

  void push(T value) {
```

```
    if (isFull()) {
        cout << "Stack Overflow! Cannot push " << value << endl;
        return;
    }
    arr[++top] = value;
}


T pop() {
    if (isEmpty()) {
        cout << "Stack Underflow!" << endl;
        return T();
    }
    return arr[top--];
}


T peek() {
    if (isEmpty()) {
        cout << "Stack is Empty!" << endl;
        return T();
    }
    return arr[top];
}


bool isEmpty() {
    return top == -1;
```

```
  }



  bool isFull() {
    return top == capacity - 1;
  }



  void display() {
    if (isEmpty()) {
      cout << "Stack is Empty!" << endl;
      return;
    }
    cout << "Stack elements: ";
    for (int i = 0; i <= top; i++) {
      cout << arr[i] << " ";
    }
    cout << endl;
  }
};



int main() {

  Stack<int> intStack(5);
  intStack.push(10);
  intStack.push(20);
  intStack.push(30);
```

```
    intStack.display();
    cout << "Top element = " << intStack.peek() << endl;
    cout << "Popped: " << intStack.pop() << endl;
    intStack.display();


    Stack<float> floatStack(3);
    floatStack.push(1.1f);
    floatStack.push(2.2f);
    floatStack.display();
    cout << "Popped: " << floatStack.pop() << endl;
    floatStack.display();


    Stack<char> charStack(4);
    charStack.push('A');
    charStack.push('B');
    charStack.push('C');
    charStack.display();
    cout << "Top element = " << charStack.peek() << endl;

    return 0;
}

/*
Output :

Stack elements: 10 20 30
```

Top element = 30

Popped: 30

Stack elements: 10 20

Stack elements: 1.1 2.2

Popped: 2.2

Stack elements: 1.1

Stack elements: A B C

Top element = C

*/

//14) Write a generic function that will sort a character string, integer and float
//value. Create a menu with appropriate options and accept the values from the
//user.

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


template <typename T>
void sortArray(T arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        T temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}


template <typename T>
void displayArray(T arr[], int n) {
  for (int i = 0; i < n; i++)
    cout << arr[i] << " ";
```

```
    cout << endl;
}



int main() {
    int choice;

    do {
        cout << "\n====== MENU ======" << endl;
        cout << "1. Sort Integer Array" << endl;
        cout << "2. Sort Float Array" << endl;
        cout << "3. Sort Character String" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            int n;
            cout << "Enter number of integers: ";
            cin >> n;
            int *arr = new int[n];
            cout << "Enter " << n << " integers: ";
            for (int i = 0; i < n; i++)
                cin >> arr[i];

            sortArray(arr, n);
            cout << "Sorted Integers: ";
            displayArray(arr, n);
```

```
        delete[] arr;

    } else if (choice == 2) {
        int n;
        cout << "Enter number of floats: ";
        cin >> n;
        float *arr = new float[n];
        cout << "Enter " << n << " floats: ";
        for (int i = 0; i < n; i++)
            cin >> arr[i];

        sortArray(arr, n);
        cout << "Sorted Floats: ";
        displayArray(arr, n);
        delete[] arr;

    } else if (choice == 3) {
        string str;
        cout << "Enter a string: ";
        cin >> str;
        int n = str.length();
        char *arr = new char[n];

        for (int i = 0; i < n; i++)
            arr[i] = str[i];

        sortArray(arr, n);
        cout << "Sorted String: ";
```

```
        for (int i = 0; i < n; i++)
            cout << arr[i];
        cout << endl;

        delete[] arr;

    } else if (choice == 4) {
        cout << "Exiting program..." << endl;
    } else {
        cout << "Invalid choice! Try again." << endl;
    }

  } while (choice != 4);

  return 0;
}

/*
Output :

====== MENU ======
1. Sort Integer Array
2. Sort Float Array
3. Sort Character String
4. Exit
Enter your choice: 1
Enter number of integers: 5
Enter 5 integers: 10
```

50

80

90

60

Sorted Integers: 10 50 60 80 90

====== MENU ======

1. Sort Integer Array

2. Sort Float Array

3. Sort Character String

4. Exit

Enter your choice: 2

Enter number of floats: 5

Enter 5 floats: 1.1

2.5

1.0

2.6

9.8

Sorted Floats: 1 1.1 2.5 2.6 9.8

====== MENU ======

1. Sort Integer Array

2. Sort Float Array

3. Sort Character String

4. Exit

Enter your choice: 3

Enter a string: vivek

Sorted String: eikvv

OOCP Practical Assignment 2 MCA 1 Name : vivek jadav
Roll no : 09

====== MENU ======

1. Sort Integer Array

2. Sort Float Array

3. Sort Character String

4. Exit

Enter your choice:

*/

```
//15) Write a template function called find(). This function searches an array for an
//object. It returns either the index of the matching object (if one is found) or
//-1 if no match is found.



#include <iostream>
using namespace std;



template <typename T>
int find(T arr[], int n, T key) {
  for (int i = 0; i < n; i++) {
    if (arr[i] == key) {
      return i;
    }
  }
  return -1;
}

int main() {
  int intArr[] = {10, 20, 30, 40, 50};
  int intSize = 5;
  int intKey = 30;
  cout << "Searching " << intKey << " in intArr ? Index: "
    << find(intArr, intSize, intKey) << endl;



  float floatArr[] = {1.1f, 2.2f, 3.3f, 4.4f};
```

```
    int floatSize = 4;
    float floatKey = 4.4f;
    cout << "Searching " << floatKey << " in floatArr ? Index: "
         << find(floatArr, floatSize, floatKey) << endl;



    char charArr[] = {'a', 'e', 'i', 'o', 'u'};
    int charSize = 5;
    char charKey = 'o';
    cout << "Searching '" << charKey << "' in charArr ? Index: "
         << find(charArr, charSize, charKey) << endl;



    int missingKey = 99;
    cout << "Searching " << missingKey << " in intArr ? Index: "
         << find(intArr, intSize, missingKey) << endl;

    return 0;
}

//Output :

//Searching 30 in intArr ? Index: 2
//Searching 4.4 in floatArr ? Index: 3
//Searching 'o' in charArr ? Index: 3
//Searching 99 in intArr ? Index: -1
```