#include<iostream>

using namespace std;

class matrix {

int t[10][10];

int m, n;

public:

matrix() {

m = 0;

n = 0;

}

matrix(int r, int c) {

m = r;

n = c;

t[m][n];

create(t, m, n);

}

void create(int t[][10], int, int);

void display();

friend matrix operator+(matrix, matrix);

friend matrix operator-(matrix, matrix);

friend matrix operator\*(matrix, matrix);

};

void matrix :: create(int t[][10], int m, int n) {

int i, j;

cout<<"Enter elements for matrix.\n";

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

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

cin>>t[i][j];

}

}

}

void matrix :: display() {

int i, j;

cout<<"Matrix is: \n";

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

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

cout<<t[i][j];

cout<<"\t";

}

cout<<endl;

}

}

matrix operator+(matrix a, matrix b) {

int i, j;

matrix c;

if(a.m == b.m&&a.n == b.n) {

c.m = a.m; c.n = a.n;

for(i=0;i<a.m;i++) {

for(j=0;j<a.n;j++) {

c.t[i][j] = a.t[i][j] + b.t[i][j];

}

}

}

else {

cout<<"Matrices cannot be added!\n";

}

return c;

}

matrix operator-(matrix a, matrix b) {

int i, j;

matrix c;

if(a.m == b.m&&a.n == b.n) {

c.m = a.m; c.n = a.n;

for(i=0;i<a.m;i++) {

for(j=0;j<a.n;j++) {

c.t[i][j] = a.t[i][j] - b.t[i][j];

}

}

}

else {

cout<<"Matrices not subtractable!\n";

}

return c;

}

matrix operator\*(matrix a, matrix b) {

int i, j, k;

matrix c;

if(a.n == b.m) {

c.m = a.m; c.n = b.n;

for(i=0;i<a.m;i++) {

for(j=0;j<b.n;j++) {

c.t[i][j] = 0;

for(k=0;k<a.n;k++) {

c.t[i][j] += a.t[i][k] \* b.t[k][j];

}

}

}

}

else {

cout<<"Matrices not multiplicable!\n";

}

return c;

}

int main(void) {

matrix a(2,2), b(2,2), c, d, e;

c = a + b;

d = a - b;

e = a \* b;

a.display();

b.display();

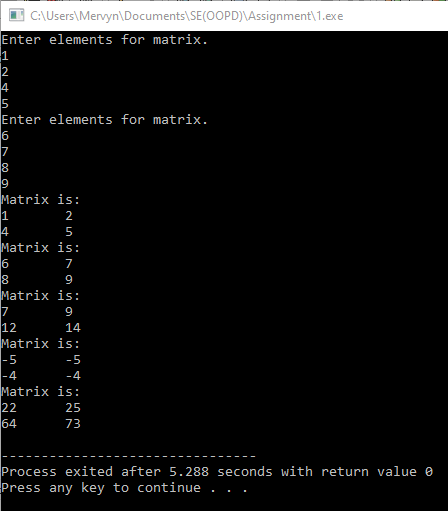
c.display();

d.display();

e.display();

}

**Output:**



**2.**

A friend function will be compulsory if you are performing an operation involving one built in data type and an object of class type

Eg:- t1 = 5 + t2;

Here t1 and t2 are objects of class type, whereas 5 is in integer.

**3.**

Time t2(t1) and Time t2 = t1 are both calls to Copy Constructor function.

Time t2(t1) is an “EXPLICIT” call.

Time t2 = t1 is an “IMPLICIT” call.

**4.**

#include<iostream>

#include<string.h>

using namespace std;

class String {

char \*s;

int len;

public:

String() {

s = NULL;

len = 0;

}

String(const char \*x) {

len = strlen(x);

s = new char[len + 1];

strcpy(s, x);

}

void display() {

cout<<"String: ";

puts(s);

cout<<endl;

}

friend String operator+(const String &, const String &);

};

String operator+(const String &a, const String &b) {

String c;

c.len = a.len + b.len;

c.s = new char[c.len + 1];

c.s = strcpy(c.s, a.s);

c.s = strcat(c.s, b.s);

return c;

}

int main(void) {

String a = "New ";

String b = "Delhi";

String c;

c = a + b;

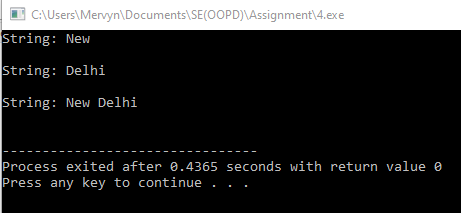
a.display();

b.display();

c.display();

}

**Output.**



**5.**

* A constructor has same name as class name.
* A constructor does not have any return value.
* A constructor is called as soon as an object is created.
* If no constructor is specified the C++ compiler generates a default constructor.

There are 3 types of constructors.

* Default Constructor:

This constructor is automatically called as soon as an object of the class is created. It does not take any arguments.

* Parametrised Constructor:

This constructor initializes values to members via the arguments passed to it. It can take any number of arguments.

* Copy Constructor:

This constructor is used to copy the contents from one object to another object of same class.

**6.**

A friend function is a function which has access to all private members of a class to which it is a friend of.

Merits: No need for inheritance

Demerit: Extra line of code which affects dynamic binding

**7.**

#include<iostream>

using namespace std;

class complex {

int x, y;

public:

complex() {

x = 0;

y = 0;

}

complex(int a, int b) {

x = a;

y = b;

}

void display(int p) {

cout<<"Complex No "<<p<<": "<<x<<" + "<<y<<"i"<<endl;

}

friend complex operator+(complex, complex);

friend complex sum(complex, complex, complex);

};

complex sum(complex a, complex b, complex c) {

c = a + b;

return c;

}

complex operator+(complex a, complex b) {

complex temp;

temp.x = a.x + b.x;

temp.y = a.y + b.y;

return temp;

}

int main() {

complex a(1, 2), b(4, 5), c;

c = sum(a, b, c);

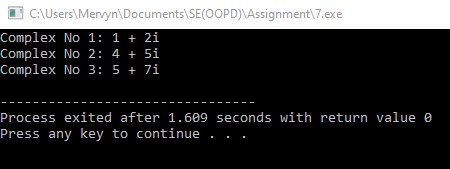
a.display(1);

b.display(2);

c.display(3);

}

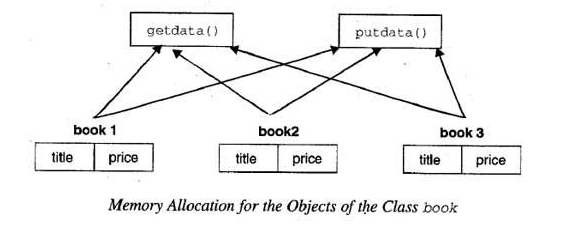
**Output:**

****

**8.**

Data members of a class are allocated memory only when an object of that class is created and not when first declared in class.

Member functions are allocated memory when they are declared in the class itself. A single copy of the function may be shared by different objects of the same class.



**9.**

#include<iostream>

using namespace std;

class vector {

int x, y, z;

public:

vector() {

x = y = z = 0;

}

vector(int a, int b, int c) {

x = a;

y = b;

z = c;

}

friend ostream& operator<<(ostream&, vector&);

friend istream& operator>>(istream&, vector&);

friend int operator==(vector, vector);

};

istream& operator>>(istream& a, vector &v) {

a>>v.x;

a>>v.y;

a>>v.z;

}

ostream& operator<<(ostream& a, vector &v) {

a<<v.x;

cout<<"i\t";

a<<v.y;

cout<<"j\t";

a<<v.z;

cout<<"k.";

cout<<endl;

}

int operator==(vector v1, vector v2) {

if(v1.x == v2.x && v1.y == v2.y && v1.z == v2.z)

return 1;

else

return 0;

}

int main(void) {

vector v1, v2;

cout<<"Enter values for vector.\n";

cin>>v1;

cout<<"Enter values for vector.\n";

cin>>v2;

cout<<"Values for vector are 1: ";

cout<<v1;

cout<<"Values for vector are 2: ";

cout<<v2;

if(v1 == v2)

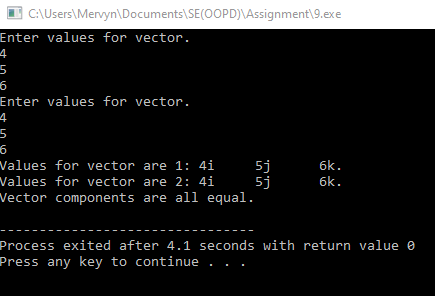
cout<<"Vector components are all equal.\n";

else

cout<<"Vector components are not equal.\n";

}

**Output:**

****

**10.**

#include<iostream>

using namespace std;

class centi;

class meters;

class centi {

float cm;

public:

centi() {

cm = 0.0;

}

centi(float x) {

cm = x;

}

void display() {

cout<<"Value of centimeters is: "<<cm<<endl;

}

float getcm() {

return cm;

}

};

class meters {

float m;

public:

meters() {

m = 0.0;

}

meters(centi x) {

float c;

c = x.getcm()/100.0;

m = m + c;

cout<<"Centimeters converted to meters: "<<m<<endl;

}

};

int main() {

centix(10.5); meters y;

x.display();

y = x;

}

**Output:**

