Object Oriented Programming with C++

6. Classes and Objects

By: Prof. Pandav Patel

Second Semester, 2020-21
Computer Engineering Department
Dharmsinh Desai University

Limitations of C structure (for OOP)

- In C language, structures do not permit data hiding
  - Members of structure variables can be accessed by anywhere in the program (where structure variable is visible)

- C language does not treat struct data type like built-in types

```
struct complex
{
    double i;
    double j;
}c1 = {1, 2}, c2 = {1, 2}, c3;
c3 = c1 + c2;    // This can not be achieved
if(c1 == c2)     // Even this is not allowed
```

Structures in C++

- In C++, structures can be used exactly like C language
- But C++ has extended its capabilities to support OOP
  - Structures can have variables and functions as members
  - It can also declare private members (supports data hiding)
  - While declaring a variable struct keyword can be omited
    - e.g. struct st{ int i}; st s1;

- Under the hood, C++ structures are nothing but classes
  - Only difference is that, by default, members of a class are private while, by default, members of structure are public (for compatibility with C language)
- So we will not discuss structures any further. Lets talk abt class

Classes in C++

- Class is a way to bind data and associated functions together
- Classes allow data and functions to be hidden (prevent access from outside of the class)
- By defining class, we are creating a new Abstract Data Type (ADT), that can be treated like any other built-in type

- Specification of class includes:
  - Class Definition – Describes type and scope of its members
  - Class function definitions

```
#include<iostream>

class MyFirstClass
{
	int foo = 0;
	int set_foo(int foo_value)
	{
		foo = foo_value;
		return foo;
	}

public:
	int bar = 10;
	int get_foo()
	{
		return foo;
	}
	int auth_and_set_foo(int password, int foo_value);
};



int MyFirstClass::auth_and_set_foo(int password, int
foo_value)
{
	if(123 == password)
		return set_foo(foo_value);
	return foo;
}
```

```
int main()
{
	MyFirstClass obj1;

	std::cout << obj1.bar << std::endl;
	obj1.bar = 50;
	std::cout << obj1.bar << std::endl;

	//error: ‘int MyFirstClass::foo’ is private within this context
	//obj1.foo = 20;
	//std::cout << obj1.foo << std::endl;

	//error: ‘int MyFirstClass::set_foo(int)’ is private within this context
	//obj1.set_foo(20);

	std::cout << obj1.get_foo() << std::endl;

	std::cout << obj1.auth_and_set_foo(122, 100) << std::endl;
	std::cout << obj1.auth_and_set_foo(123, 100) << std::endl;
	return 0;

}
```

```
10
50
0
0
10
0
```

Classes and objects in C++

- Class Definition
  - New keyword **class**, it is followed by ***class-name*** (identifier)
  - Enclosed within braces and terminated by semicolon
  - Class definition contains declaration of variables and functions (they are called members of the class)
    - Variables are called **data members**
    - Functions are called **member functions** or **methods**
  - Methods can be defined within class definition or outside class definition
    - Methods defined within class definition become inline by default. Hence only small functions should be defined within class def^{n}
    - Methods defined outside class need to be fully qualified with class name and scope resolution operator. They are not inlined by default, but can be declared inline explicitly using **inline** keyword

Classes and objects in C++

- Access specifiers
  - Two new keywords **private** and **public**
  - Private members (data members and functions) can only be accessed by methods of the same class (private and public)
  - Public members can be accessed by methods of the same class as well as from outside the class (e.g. main function)
  - By default members of the class are private
  - Keywords private and public can be used multiple times in the class definition.
  - Keywords public and private end with colon, and their effect is retained for all the members declared after them, until occurrence of next access specifier
  - Generally (not always), data members are private and methods are public.

Classes and objects in C++

- Objects and member access
  - Object is an instance of class (like variable of structure)
  - Memory is allocated when object is created
  - Once class is defined object can be declared as
    - ***class-name object-name;***
  - From outside the class, members of the objects can be accessed using **dot operator** (**.**)
    - ***object-name.method-name(parameters);***
    - ***object-name.variable-name = some-value;***
  - Methods of the class can access object members without name of the object and dot operator.
    - When method is called using object **x,** within that method members of **x** can be accessed without name of the object and dot operator.

```
#include<iostream>

class Complex
{
    double re;
    double im;
public:
    void add(Complex c)
    {
        std::cout << &c << std::endl;
        re += c.re;
        im += c.im;
    }
    void print()
    {
        std::cout << "real: " << re;
        std::cout << "\timg: " << im <<std::endl;
    }
    void initialize(double _re, double _im)
    {
        re = _re;
        im = _im;
    }
}c3;
int i_global;
```

```
int main()
{
    Complex c1, *c2_ptr = new Complex;

    int i_auto, *ip_dynamic = new int;

    std::cout << "i_global addr\t\t" << &i_global << std::endl;
    std::cout << "c3 addr (global)\t" << &c3 << std::endl;
    std::cout << "i_auto addr\t\t" << &i_auto << std::endl;
    std::cout << "c1 addr (auto)\t\t" << &c1 << std::endl;
    std::cout << "i_dynamic addr\t\t" << ip_dynamic << std::endl;
    std::cout << "c2 addr (dynamic)\t" << c2_ptr << std::endl;

    c1.initialize(1, 1);
    c2_ptr->initialize(2, 2);

    c3.print();
    c3.add(c1);
    c3.print();
    c3.add(*c2_ptr);
    c3.print();
    return 0;
}
```

```
i_global addr           0x56299c0e6150
c3 addr (global)        0x56299c0e6140
i_auto addr             0x7ffc7e2ebd4c
c1 addr (auto)          0x7ffc7e2ebd60
i_dynamic addr          0x56299ce3ee90
c2 addr (dynamic)       0x56299ce3ee70
real: 0      img: 0
0x7ffc7e2ebd00
real: 1      img: 1
0x7ffc7e2ebd00
real: 3      img: 3
```

Classes and objects in C++

- Memory allocation for objects
  - Like other variables, memory for objects is allocated from stack, data or heap section; depending on where it is declared
    - Non-static (local) objects are allocated memory from stack (contain garbage by default)
    - Static (global and local static) objects are allocated memory from data section (contain zero by default)
    - Dynamically created objects (with new keyword), are allocated memory from free store (heap) (contain garbage by default)
- Like other variables, references can also be created for objects
  - Object can be passed to function by value, by pointer (address) or by reference
    - In previous example, objects are passed by value to add function
    - In next example, objects are passed by reference to add function

```
#include<iostream>

class Complex
{
    double re;
    double im;
public:
    void add(Complex &c)
    {
        std::cout << &c << std::endl;
        re += c.re;
        im += c.im;
    }
    void print()
    {
        std::cout << "real: " << re;
        std::cout << "\timg: " << im <<std::endl;
    }
    void initialize(double _re, double _im)
    {
        re = _re;
        im = _im;
    }
}c3;
int i_global;
```

```
int main()
{
    Complex c1, *c2_ptr = new Complex;
    Complex &c2 = *c2_ptr;
    int i_auto, *ip_dynamic = new int;

    std::cout << "i_global addr\t\t" << &i_global << std::endl;
    std::cout << "c3 addr (global)\t" << &c3 << std::endl;
    std::cout << "i_auto addr\t\t" << &i_auto << std::endl;
    std::cout << "c1 addr (auto)\t\t" << &c1 << std::endl;
    std::cout << "i_dynamic addr\t\t" << ip_dynamic << std::endl;
    std::cout << "c2 addr (dynamic)\t" << &c2 << std::endl;

    c1.initialize(1, 1);
    c2.initialize(2, 2);

    c3.print();
    c3.add(c1);
    c3.print();
    c3.add(c2);
    c3.print();
    return 0;
}
```

```
i_global addr           0x55f85d804150
c3 addr (global)        0x55f85d804140
i_auto addr             0x7ffcd6bb51f4
c1 addr (auto)          0x7ffcd6bb5210
i_dynamic addr          0x55f85eb4fe90
c2 addr (dynamic)       0x55f85eb4fe70
real: 0      img: 0
0x7ffcd6bb5210
real: 1      img: 1
0x55f85eb4fe70
real: 3      img: 3
```

Classes and objects in C++

- ***this*** keyword
  - ***this*** keyword is like constant pointer to the object which invoked the method
  - Hence it is generally used with arrow operator
    - e.g. **this->data-member = 7;**
    - e.g. **this->member-function();**
  - It can be used only inside the methods of the class. It can not be used in independent function
  - Can be used to access object members which are hidden behind local variables of the same name.
  - Methods can access other methods and data members even without use of **this** keyword (as far as members are not hidden behind local variable of the same name).

```
#include<iostream>

class Complex
{
    double re;
    double im;
public:
    void add(Complex &c)
    {
        std::cout << &c << std::endl;
        re += c.re;
        im += c.im;
    }
    void print()
    {
        std::cout << "real: " << re;
        std::cout << "\timg: " << im <<std::endl;
    }
    void initialize(double re, double im)
    {
        this->re = re;
        this->im = im;
    }
}c3;
int i_global;
```

```
int main()
{
    Complex c1, *c2_ptr = new Complex;
    Complex &c2 = *c2_ptr;
    int i_auto, *ip_dynamic = new int;

    std::cout << "i_global addr\t\t" << &i_global << std::endl;
    std::cout << "c3 addr (global)\t" << &c3 << std::endl;
    std::cout << "i_auto addr\t\t" << &i_auto << std::endl;
    std::cout << "c1 addr (auto)\t\t" << &c1 << std::endl;
    std::cout << "i_dynamic addr\t\t" << ip_dynamic << std::endl;
    std::cout << "c2 addr (dynamic)\t" << &c2 << std::endl;

    c1.initialize(1, 1);
    c2.initialize(2, 2);

    c3.print();
    c3.add(c1);
    c3.print();
    c3.add(c2);
    c3.print();
    return 0;
}
```

```
i_global addr           0x560816b7b150
c3 addr (global)        0x560816b7b140
i_auto addr             0x7ffc9caae694
c1 addr (auto)          0x7ffc9caae6b0
i_dynamic addr          0x5608181a7e90
c2 addr (dynamic)       0x5608181a7e70
real: 0      img: 0
0x7ffc9caae6b0
real: 1      img: 1
0x5608181a7e70
real: 3      img: 3
```

Classes and objects in C++

- Static data members (a.k.a. Class variables)
  - Declaration inside the class (with static keyword)
  - Must be defined outside the class (without static keyword)
    - Name of the class and scope resolution operator is used
    - Initial value can be provided with definition
      - If not then default value is zero
  - Only one copy is created for static data member
    - All objects of that class share the same copy
    - It is stored in data section
    - Even when class does not have any objects, static data members can be accessed and used
  - Private static data members can be accessed only by member f^n
  - Public static data members can be accessed by outside world as well
    - Using ***object and dot operator*** or ***class name and scope resolution operator***

```
#include<iostream>
class Complex
{
    double re;
    double im;
public:
    void add(Complex &c)
    {
        std::cout << &c << std::endl;
        re += c.re;
        im += c.im;
    }
    void print()
    {
        std::cout << "real: " << re;
        std::cout << "\timg: " << im <<std::endl;
    }
    void initialize(double re, double im)
    {
        this->re = re;
        this->im = im;
    }
    static int instance_count;
};
int Complex::instance_count = 0;
```

```
int main()
{
    Complex c1, c2, c3;

    std::cout << "count: " << Complex::instance_count << std::endl;

    c1.initialize(1, 1);
    c1.instance_count++;
    c2.initialize(2, 2);
    c2.instance_count++;
    c3.initialize(0, 0);
    c3.print();
    c3.add(c1);
    Complex::instance_count++;
    std::cout << "count: " << Complex::instance_count << std::endl;

    c3.print();
    c3.add(c2);
    c3.print();
    return 0;
}
```

```
count: 0
real: 0    img: 0
0x7ffd84af2c60
count: 3
real: 1    img: 1
0x7ffd84af2c70
real: 3    img: 3
```

Classes and objects in C++

- Static member functions (static methods)
  - ***static*** keyword is appended before the method declaration inside class definition
  - Can be defined inside the class (along with declaration) or outside the class
    - When defined outside the class static keyword should not be used
  - It can only access other static data members and static member functions of the class
    - It can not access other non-static data members or non-static member functions of the class
  - Private static member functions can be called only by other static and non-static member functions of the same class
  - Public static methods can be called from outside the class as well
    - Using ***object and dot operator*** or ***class name and scope resolution operator***

```
#include<iostream>

class FirstClass
{
    static int cnt;
public:
    int other_var;
    static void set_cnt(int i)
    {
        // error: invalid use of member ‘FirstClass::other_var’ in static member function
        // other_var = 0; // static method can access only static members of the class
        cnt = i;
    }
    static void print_cnt();
};

void FirstClass::print_cnt()
{
    std::cout << cnt << std::endl;
}

int FirstClass::cnt = 10;
```

```
int main()
{
    FirstClass fc;
    fc.print_cnt();
    FirstClass::set_cnt(0);
    fc.print_cnt();
    fc.set_cnt(100);
    FirstClass::print_cnt();
    return 0;
}
```

```
10
0
100
```

```
#include<iostream>
#include<cstring>

class Person
{
public:
    char name[50];
    int age;
};

void change_name(char name[])
{
    name[0] = 'R';
}

void change_age(int age)
{
    age = 31;
}
```

```
int main()
{
    Person person1;
    strcpy(person1.name, "Sita");
    person1.age = 30;

    change_name(person1.name);    // pass by pointer
    change_age(person1.age);      // pass by value

    std::cout << person1.name << " ";
    std::cout << person1.age << std::endl;

    return 0;
}
```

Rita 30

```
#include<iostream>
#include<cstring>

class Person
{
public:
    char name[50];
    int age;
};

void change_name(Person person)
{
    person.name[0] = 'R';
}

void change_age(Person person)
{
    person.age = 31;
}
```

```
int main()
{
    Person person1;
    strcpy(person1.name, "Sita");
    person1.age = 30;

    change_name(person1);      // pass object by value
    change_age(person1);       // pass object by value

    std::cout << person1.name << " ";
    std::cout << person1.age << std::endl;

    return 0;
}
```

Sita 30

```
#include<iostream>
#include<cstring>

class Person
{
public:
    char name[50];
    int age;
};

void change_name(Person &person)
{
    person.name[0] = 'R';
}

void change_age(Person &person)
{
    person.age = 31;
}
```

```
int main()
{
    Person person1;
    strcpy(person1.name, "Sita");
    person1.age = 30;

    change_name(person1);       // pass object by reference
    change_age(person1);  // pass object by reference

    std::cout << person1.name << " ";
    std::cout << person1.age << std::endl;

    return 0;
}
```

Rita 31

```
#include<iostream>
#include<cstring>

class Person
{
public:
    char name[50];
    int age;
};

void change_name(Person person[])
{
    person[0].name[0] = 'R';
}

void change_age(Person person[])
{
    person[0].age = 31;
}
```

```
int main()
{
    Person person[5];
    strcpy(person[0].name, "Sita");
    person[0].age = 30;

    change_name(person);  // pass object array by pointer
    change_age(person);     // pass object array by pointer

    std::cout << person[0].name << " ";
    std::cout << person[0].age << std::endl;

    return 0;
}
```

```
#include<iostream>
#include<cstring>

class Person
{
public:
    char name[50];
    int age;
    void print()
    {
        std::cout << name << " ";
        std::cout << age << std::endl;
    }
};

void change_name(Person person[])
{
    person[0].name[0] = 'R';
}

void change_age(Person person[])
{
    person[0].age = 31;
}
```

```
int main()
{
    Person person[5];
    strcpy(person[0].name, "Sita");
    person[0].age = 30;

    change_name(person);
    change_age(person);

    // We can access members of an element of object array
    person[0].print();

    return 0;
}
```

```
#include<iostream>
#include<cstring>

class Person
{
    int age;
public:
    char name[50];
    void print()
    {
        std::cout << name << " ";
        std::cout << age << std::endl;
    }

    void set_age(int age)
    {
        this->age = age;
    }

    int get_age()
    {
        return age;
    }
};
```

```
void change_name(Person person)
{
    person.name[0] = 'R';
}

void change_age(Person person)
{
    person.age = 31;
}

int main()
{
    Person person;
    strcpy(person.name, "Sita");
    person.set_age(30);

    change_name(person);
    // error: ‘int Person::age’ is private within this context
    // change_age needs to be method or friend function
    // inorder to access its private members
    change_age(person);

    person.print();

    return 0;
}
```

Classes and objects in C++

- Friend functions
  - We can allow outside functions to access private members of a class
  - To make outside function friendly, we need to declare it as a friend within class definition
    - With use of **friend** keyword
  - Function is defined outside the class definition
    - Function definition does not use keyword friend
    - Can’t use class name and scope resolution operator with definition
    - In other words, function definition does not require any change
      - Unlike methods of that class, friend function can not even access private members without object name and dot operator
  - A function can be declared as friend function in more than one class
  - It can not be called using object of the class in which it has be declared friendly. Like definition, function call requires no change
  - It can be declared in public or private area without affecting its meaning

```
#include<iostream>
#include<cstring>

class Person
{
    int age;
public:
    char name[50];
    void print()
    {
        std::cout << name << " ";
        std::cout << age << std::endl;
    }

    void set_age(int age)
    {
        this->age = age;
    }

    int get_age()
    {
        return age;
    }

    friend void change_age(Person);
};

void change_name(Person person)
{
    person.name[0] = 'R';
}

void change_age(Person person)
{
    person.age = 31;
}

int main()
{
    Person person;
    strcpy(person.name, "Sita");
    person.set_age(30);

    // name ramains Sita, as its pass by value
    change_name(person);
    // age ramains 30, as its pass by value
    change_age(person);

    person.print();

    return 0;
}
```

Sita 30

```
#include<iostream>
#include<cstring>

class Person
{
    int age;
public:
    char name[50];
    void print()
    {
        std::cout << name << " ";
        std::cout << age << std::endl;
    }

    void set_age(int age)
    {
        this->age = age;
    }

    int get_age()
    {
        return age;
    }

    friend void change_age(Person &);
};
```

```
void change_name(Person &person)
{
    person.name[0] = 'R';
}

void change_age(Person &person)
{
    person.age = 31;
}

int main()
{
    Person person;
    strcpy(person.name, "Sita");
    person.set_age(30);

    // Name changes to Rita, as its pass by reference
    change_name(person);
    // age changes to 31, as its pass by reference
    change_age(person);

    person.print();

    return 0;
}
```

Rita 31

```
#include<iostream>

// Class declaration
class Mat;

class Vect
{
    int vctr[3];
    friend Vect product(Mat m, Vect v);
public:
    void initialize()
    {
        int i;
        for(i = 0; i < 3; i++)
            vctr[i] = i + 1;
    }

    void display()
    {
        int i;
        for(i = 0; i < 3; i++)
            std::cout << vctr[i] << std::endl;
    }
};
```

```
Our matrx:
0       1       2
1       2       3
2       3       4
Our vector:
1
2
3
Our resultant
vector:
8
14
20
```

```
class Mat
{
    int matrix[3][3];
public:
    void initialize()
    {
        int i, j;
        for(i = 0; i < 3; i++)
            for(j = 0; j < 3; j++)
                matrix[i][j] = i + j;
    }
    void display()
    {
        int i, j;
        for(i = 0; i < 3; i++)
        {
            for(j = 0; j < 3; j++)
                std::cout << matrix[i][j] << "\t";
            std::cout << std::endl;
        }
    }
    friend Vect product(Mat m, Vect v);
};
```

```
Vect product(Mat m, Vect v)
{
    Vect result;
    int i, j;
    for(i = 0; i < 3; i++)
    {
        result.vctr[i] = 0;
        for(j = 0; j < 3; j++)
        {
            result.vctr[i] += m.matrix[i][j] * v.vctr[j];
        }
    }
    // Returning result object by value
    return result;
}

int main()
{
    Mat m;
    Vect v;
    m.initialize();
    std::cout << "Our matrx:" << std::endl;
    m.display();
    v.initialize();
    std::cout << "Our vector:" << std::endl;
    v.display();

    std::cout << "Our resultant vector:" << std::endl;
    product(m, v).display();

    return 0;
}
```

```
#include<iostream>

// Class declaration
class Mat;

class Vect
{
    int vctr[3];
    friend Vect &product(Mat m, Vect v);
public:
    void initialize()
    {
        int i;
        for(i = 0; i < 3; i++)
            vctr[i] = i + 1;
    }

    void display()
    {
        int i;
        for(i = 0; i < 3; i++)
            std::cout << vctr[i] << std::endl;
    }
};
```

```
class Mat
{
    int matrix[3][3];
public:
    void initialize()
    {
        int i, j;
        for(i = 0; i < 3; i++)
            for(j = 0; j < 3; j++)
                matrix[i][j] = i + j;
    }
    void display()
    {
        int i, j;
        for(i = 0; i < 3; i++)
        {
            for(j = 0; j < 3; j++)
                std::cout << matrix[i][j] << "\t";
            std::cout << std::endl;
        }
    }
    friend Vect &product(Mat m, Vect v);
};
```

```
Vect &product(Mat m, Vect v)
{
    Vect result;
    int i, j;
    for(i = 0; i < 3; i++)
    {
        result.vctr[i] = 0;
        for(j = 0; j < 3; j++)
        {
            result.vctr[i] += m.matrix[i][j] * v.vctr[j];
        }
    }
    // warning: reference to local variable ‘result’ returned
    return result;
}

int main()
{
    Mat m;
    Vect v;
    m.initialize();
    std::cout << "Our matrx:" << std::endl;
    m.display();
    v.initialize();
    std::cout << "Our vector:" << std::endl;
    v.display();

    std::cout << "Our resultant vector:" << std::endl;
    product(m, v).display();

    return 0;
}
```

For me, on g++, it resulted in compilation warning
And during execution, it ends up in segmantation fault

```
#include<iostream>

// Class declaration
class Mat;

class Vect
{
    int vctr[3];
    friend Vect &product(Mat m, Vect v);
public:
    void initialize()
    {
        int i;
        for(i = 0; i < 3; i++)
            vctr[i] = i + 1;
    }

    void display()
    {
        int i;
        for(i = 0; i < 3; i++)
            std::cout << vctr[i] << std::endl;
    }
};
```

```
Our matrx:
0       1       2
1       2       3
2       3       4
Our vector:
1
2
3
Our resultant
vector:
8
14
20
```

```
class Mat
{
    int matrix[3][3];
public:
    void initialize()
    {
        int i, j;
        for(i = 0; i < 3; i++)
            for(j = 0; j < 3; j++)
                matrix[i][j] = i + j;
    }
    void display()
    {
        int i, j;
        for(i = 0; i < 3; i++)
        {
            for(j = 0; j < 3; j++)
                std::cout << matrix[i][j] << "\t";
            std::cout << std::endl;
        }
    }
    friend Vect &product(Mat m, Vect v);
};
```

```
Vect &product(Mat m, Vect v)
{
    Vect &result = *(new Vect);
    int i, j;
    for(i = 0; i < 3; i++)
    {
        result.vctr[i] = 0;
        for(j = 0; j < 3; j++)
        {
            result.vctr[i] += m.matrix[i][j] * v.vctr[j];
        }
    }

    return result;
}

int main()
{
    Mat m;
    Vect v;
    m.initialize();
    std::cout << "Our matrx:" << std::endl;
    m.display();
    v.initialize();
    std::cout << "Our vector:" << std::endl;
    v.display();

    std::cout << "Our resultant vector:" << std::endl;
    product(m, v).display();

    return 0;
}
```

//Like other functions main() can also be a friend of a class.

```
#include<iostream>

class A
{

int pvt_member;

friend int main();

};
```

```
int main() {

    A a1;

    a1.pvt_member=10;

    std::cout<<a1.pvt_member;

    return 0;

}
```

Classes and objects in C++

- Member functions of other classes can also be declared as friend functions

```
class Y
{
    friend int X::fun();
};
```

  - There would be no change in how function **fun** is defined and called
  - Function **fun** is a member function of class X and friend of class Y
    - It needs to be invoked with object of class X
    - If there are any objects of class Y (within scope of **fun**), then function **fun** would be able to access private members of those objects of Y
    - But there reverse is not true
      - Methods of class Y can not access private members of class X, unless those methods are declared as friend within class X

Classes and objects in C++

- Friend class

```
class Y
{
    friend class X;
};
```

  - All the methods of class X, would become friends of class Y
    - So all the methods of class X can access private members of class Y
    - Methods of class Y can not access private members of class X unless
      - Those methods are declared as friend within class X, within definition of X
      - Or if class Y has been declared as friend class of X

  - In other words, friend relationship is not commutative

```
#include<iostream>

// Class declaration
class Mat;

class Vect
{
    int vctr[3];
    friend Vect &product(Mat m, Vect v);
public:
    void initialize()
    {
        int i;
        for(i = 0; i < 3; i++)
            vctr[i] = i + 1;
    }

    void display() const
    {
        int i;
        // error: assignment of read-only location
        // vctr[0] = 100;
        for(i = 0; i < 3; i++)
            std::cout << vctr[i] << std::endl;
    }
};
```

```
class Mat
{
    int matrix[3][3];
public:
    void initialize()
    {
        int i, j;
        for(i = 0; i < 3; i++)
            for(j = 0; j < 3; j++)
                matrix[i][j] = i + j;
    }
    void display() const
    {
        int i, j;
        for(i = 0; i < 3; i++)
        {
            for(j = 0; j < 3; j++)
                std::cout << matrix[i][j] << "\t";
            std::cout << std::endl;
        }
    }
    friend Vect &product(Mat m, Vect v);
};
```

```
Our matrx:
0       1       2
1       2       3
2       3       4
Our vector:
1
2
3
Our resultant
vector:
8
14
20
```

```
Vect &product(Mat m, Vect v)
{
    Vect &result = *(new Vect);
    int i, j;
    for(i = 0; i < 3; i++)
    {
        result.vctr[i] = 0;
        for(j = 0; j < 3; j++)
        {
            result.vctr[i] += m.matrix[i][j] * v.vctr[j];
        }
    }

    return result;
}

int main()
{
    Mat m;
    Vect v;
    m.initialize();
    std::cout << "Our matrx:" << std::endl;
    m.display();
    v.initialize();
    std::cout << "Our vector:" << std::endl;
    v.display();

    std::cout << "Our resultant vector:" << std::endl;
    product(m, v).display();

    return 0;
}
```

Classes and objects in C++

- ***const*** member functions
  - If member function does not alter any data member of the object then it may be declared as const method
  - const member functions does not change the state of the object
  - Keyword **const** is appended at the end of the function prototype (both in declaration and definition)
  - Compiler will generate error if const member function tries to alter state of the object
  - Though such function can have local variables and it can also alter local variable.
  - Benefit of const member functions
    - Developer will not modify state of the object from const member functions by mistake
    - We will see later that const object can call const member functions, but it can not call other non-const member functions

```
#include<iostream>

void fun(int &i)
{
    i = 7;
    std::cout << i << std::endl;
}

class Test
{
    int x = 5;
public:
    void abc()
    {
         // passing private member to non-member function as reference
        fun(x);
        std::cout << x << std::endl;
    }
};

int main()
{
    Test t;
    t.abc();
    return 0;
}
```

- This is valid and does not violate the rules of encapsulation (data hiding), because member function by itself (knowingly) is sending the private member to non-member function
- Member function is knowingly compromising its privacy, and it is allowed
- We have been doing something similar all along, when we use scanf function to scan the values of private members

Classes and objects in C++

- Local classes
  - C++ allows us to define classes within function
  - Methods of the local classes can use
    - Global variables
    - Static variables of the function in which local class is defined
  - Methods of the local classes can not use auto variables of the function in which local class is defined
  - Local classes can not have static data members
  - Member functions of the local class must be defined within the class definition (inline by default)