

# Objects in C++

## Encapsulation



# C++ Object System

- Object-oriented features
  1. Classes and Data Abstraction
  2. Encapsulation
  3. Inheritance
    - Single and multiple inheritance
    - Public and private base classes
  4. Objects, with dynamic lookup of virtual functions
  5. Subtyping
    - Tied to inheritance mechanism

# Encapsulation

- **Encapsulation** means that implementation details are hidden inside a program unit with a specific interface.
- A way to provide **abstraction**: the *interface* of objects usually consist of a set of public functions that manipulate hidden data.
- Incapsulation involves *restricting access* to a program component according to its specified interface.

# Struct and Class in C++ (1)

- A **struct** is a way to collect a group of variables, like in C.

```
struct Structure1 {  
    char c;  
    int i;  
};
```

```
int main() {  
    struct Structure1 s1, s2;  
    // the keyword struct is optional in C++  
    ...  
}
```

# Struct and Class in C++ (2)

- In C++ **struct** and **class** have been made similar
- In C++, a struct can contain
  - member functions
  - private fields
- By default, **all members of a struct are public**
- By default, **all members of a class are private**
- Similar considerations also apply to **union**

# Visibility

- Public, private, protected levels of visibility
  - **Public:** visible everywhere
  - **Protected:** within class and subclass declarations
  - **Private:** visible only in class where declared, inherited private members exist in the derived class, but cannot be named directly in code written as part of the derived class.
- Friend functions and classes
  - Friend allows special access
  - Careful attention to visibility and data abstraction
  - Are executed faster

# Private, protected, public levels of visibility

- Member data is made **private**, so that changes do **not affect** the way that other classes (including derived classes) depend on this class.
- **Members that modify private data are made protected**, so that derived classes may change the value of member data, but external code is not allowed to do so.
- Finally, member functions that read the value of member data and provide **useful operations on objects are declared public**.

# Friend functions (1)

- A class may declare friend functions
- The friend designation is used **to allow visibility to the private and protected part of a class**
- A friend function can be
  - a *public* member function of another class
  - an external function



# Friend functions (2)

```
class A {  
private:  
    int i;  
public:  
friend int B::f(int n, A* a);  
...  
};
```

```
class B {  
private:  
    int i;  
public:  
    int f(int n, A* a);  
...  
};
```

```
int B::f(int n, A* a) {  
    return i + a->i + n;  
}
```

# Friend classes

- If a class B has the declaration **friend class A**, then code written as part of A has access to the private/private part of B.
- The friend mechanism is **used when a pair of classes is closely related**, such as matrices and vectors.

```
class A {  
    int a;  
    friend class B;  
};
```

```
class B {  
public: void foo();  
};
```

```
B::foo() {  
    A a_obj;  
    a_obj.a = 10;  
}
```