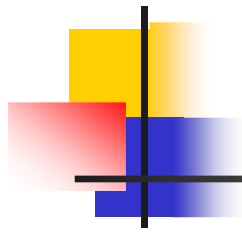


# Lecture-4

---

- friend functions
- friend classes
- Inheritance
- Miscellaneous topics
  - static members
  - this keyword
  - Setting member values
  - const member functions
  - Function overriding and function overloading



# friend functions

---

- What if a function genuinely needs to have access to private data?
  - E.g. `showAccountInfo (Account acct )`
- Need to give access **ONLY** to that function, not others.
- Use **friend** function definition
- **friend** functions of a class have access to private members of the class.



# Example - friend function

---

```
class account
{
    private:
        int user_SSN;
        int accountNumber;
    public:
        void deposit (int amount)
        void withdraw (int amount);
    friend showAccountInfo
        (class Account)
};
```

```
void showAccountInfo
    (Account acct)
{
    cout << user_SSN << endl;
    cout << accountNumber <<
        endl;
}
```

This is valid.  
Friend function can access  
private members.



# friend class

---

- Concept of **friend** can be extended to a class from a function.
- A class gives access to its private members to its **friend** classes.

class account	class bank
{	{
...	...
friend class bank	}
}	

Members of bank have access to private members of account



# Inheritance

---

- Let's take the account example again
- There are many types of accounts
  - Checking, saving, money market, IRA, etc.
- All accounts may have
  - Some common members.
    - Account number, user SSN, etc.
  - Some class specific members.
- Method implementation may be
  - Same in different classes
  - Different in different classes.



# Inheritance - base class & derived class

- **Base class**

```
class account
{
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount);
    void withdraw (int amount);
    double computeInterest ( );
};
```

- **Derived class or child class**

```
class checkingAccount : public account // checkingAccount is
{                                     // derived from account
    int lastCheckCleared;             // not present in account
    void showAllChecksCleared( );// not present in account
    double computeInterest( ); // defined in both classes
};
```



# Inheritance - base class and derived classes

## ■ Base Class

```
class account
{
private:
    int user_SSN;
    int accountNumber;
public:
    account ( ) { }
    account (int ssn, acctNum);
    ~account( ) { }
    void deposit (int amount)
    void withdraw (int amount);
    double computeInterest( );
};
```

## ■ Derived (or child) class-1

```
class checkingAccount : public account
{
public:
    int lastCheckCleared;
    void showChecksCleared ( );
    double computeInterest ( );
};
```

## ■ Derived (or child) class-2

```
class IRA_account : public account
{
public:
    void buyFund (int fund_ID);
    void sellFund (int fund_ID);
    double computeInterest ( );
};
```



# Inheritance - continued.

---

- Important points to note:
  - Derived classes have access to members of base classes in this example.
  - Derived classes can have their own members.
    - E.g. `lastCheckCleared`, `showAllChecksCleared( )`, `buyFund( )`, `sellFund( )`, etc.
    - Members of one derived class are not accessible to another





# Examples

---

- Valid usage in an external function
  - `account acct(123456, 5672);`
  - `checkingAccount ca;`
  - `acct.deposit (700);`
  - `acct.withdraw (300);`
  - `ca.deposit (1000);`
  - `ca.showAllChecksCleared( )`
- Invalid usage in derived class
  - `ca.user_SSN = 1234; // Can't access user_SSN`
  - `ca.accountNumber = 567;`



# C++ static members

---

- static members in C++
  - Shared by all the objects of a class
  - Specific to a class, **NOT** object of a class
  - Access them using `className::static_member`
  - E.g., `myClass::staticVar`, or `myClass::f1`
  - **NOT** `myClassObj.staticVar` or `myClassObj.f1( )`

```
class myClass
{
    public:
        static int staticVar;
        static void f1( );
};
```



# this keyword

---

- **this** refers to the address of the current object

- E.g.

```
class Account
{
    private:
        int balance;
    public setBalance (int amount)
    {
        this->balance = amount;
    }
};
```



# Initializing member values

---

```
class Account
{
    private:
        int balance;
    public:
        Account ( ) : balance (0)
        { }
        Account (int amount) :
            balance (amount) { }
};
```

```
class checkingAccount : public
    Account
{
    checkingAccount (int amount)
        : Account (amount) { }
}
```



# const member functions

---

```
class myClass
```

```
{
```

```
    int a;
```

```
    ...
```

```
    void f1( ) const
```

```
    { a = 3; } // Not allowed
```

```
};
```

- **const** functions can't change any attributes of myClass.
- f1 can't change a in the above example



# Making arguments const

---

```
class myClass
{
    int a;

    ...

    void f1(const int& i )
    { i = 3; } // not allowed
```

- Cannot change the value of const arguments

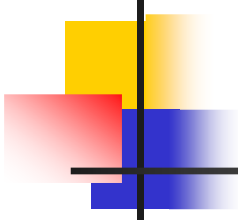


## Passing args. to a function ... by value

- Compiler creates its own copy.
- Any changes made inside the function are not reflected after the function.

```
class myClass
{
    void f1(int i ) // i is passed by value
    { i = 3; }
};

int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is still 5
```



# Passing args. to a function ... by reference

---

- Compiler takes the original object.
- Any changes made inside the function are **reflected** after the function.

```
class myClass
{
    void f1(int& i ) // i is passed by reference.
    { i = 3; }
};

int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is 3
```





# Function overriding

- Derived class can redefine (override) any function defined in the base class.
- E.g. computeInterest below is overridden by checkingAccount class.

```
class Account
{
    protected:
        double balance;
    public:
        void computeInterest( )
        {
            balance = balance +
                0.01 * balance;
        }
};
```

```
class checkingAccount : public
    Account
{
    public:
        void computeInterest ( )
        {
            balance = balance +
                0.03 * balance;
        }
};
```



# Function overloading

---

- Possible to have multiple member functions of the **same** name with **different** parameters  
⇒ Function overloading

```
class myClass
{
    // f1 - overloaded function
    void f1 (int i);
    void f1 (int i, int j);
    void f2 (int i, double j);
}
```