4 Constructors

- A **constructor** is a special member function and can be used to initialize member variables when an object is declared
  - A constructor is a member function that is usually public
  - A constructor is automatically called when an object of the class is declared
  - A constructor's name must be the name of the class
  - A constructor cannot return a value
    - No return type, not even void, is used in declaring or defining a constructor

4.1 Constructor Declaration

- A constructor for the **BankAccount** class could be declared as:

```
class BankAccount
{
  public:
        BankAccount(int dollars, int cents, double rate);
        //initializes the balance to $dollars.cents
        //initializes the interest rate to rate percent
        :
        //The rest of the BankAccount definition
};
```

4.2 Constructor Definition

- The constructor for the **BankAccount** class could be defined as

```
BankAccount::BankAccount(int dollars, int cents, double rate)
{
        if ((dollars < 0) || (cents < 0) || ( rate < 0 )) {
                cout << "Illegal values for money or rate\n";
                exit(1);
        }
        balance = dollars + 0.01 * cents;
        interest_rate = rate;
}
```

  - Note that the class name and function name are the same

4.3 Calling A Constructor

- A constructor is **not** called like a normal member function:

```
BankAccount account1;
account1.BankAccount(10, 50, 2.0); /// ERROR!!!
```

- A constructor is called in the object declaration

```
BankAccount account1(10, 50, 2.0);
```

  - Creates a **BankAccount** object and calls the **constructor** to initialize the member variables

4.4 Overloading Constructors

- Constructors can be overloaded by defining constructors with different parameter lists
  - Other possible constructors for the **BankAccount** class might be

```
BankAccount (double balance, double interest_rate);
BankAccount (double balance);
BankAccount ( );
```

4.5 The Default Constructor

- A **default constructor** uses no parameters
- A default constructor for the BankAccount class could be declared in this way

```
class BankAccount
{
  public:
        BankAccount( );
        // initializes balance to $0.00
        // initializes rate to 0.0%
        :
        // The rest of the class definition
};
```

4.5.1 Default Constructor Definition

- The default constructor for the **BankAccount** class could be defined as

```
BankAccount::BankAccount( )
{
        balance = 0;
        rate = 0.0;
}
```

- It is a good idea to always include a default constructor even if you do not want to initialize variables

4.5.2 Calling the Default Constructor

- The default constructor is called during declaration of an object
  - An argument list is not used

```
BankAccount account1; // uses the default BankAccount constructor
```

```
BankAccount account1( ); // Is not legal
```

4.5.3 New BankAccount class

```
class BankAccount
{
public:
    BankAccount(int dollars, int cents, double rate);
    // Postcondition: The account balance has been set $dollars.cents;
    // The interest rate has been set to rate percent.

    BankAccount(int dollars, double rate);
    // Postcondition: The account balance has benn set to $dollars.00.
    // The interest rate has been set to rate percent.

    BankAccount(); // Default constructor
    // Initializes the account balance to $0.00 and the interest rate to 0.0%

    void update();
    // Postcodition: One year of simple interest has been
    // added to the account balance.

    double get_balance();
    // Returns the current account balance.

    double get_rate();
    // Returns the current account interest rate as a percentage.

    void output(std::ostream& outs);
    // Precondition: If outs is a file output stream, outs has already
    // been connected to a file.
    // Postcondition: Account balance and interest rate have been
    // written to the stream outs.

private:
    double balance;
    double interest_rate;

    double fraction(double percent);
    // Converts a percentage to a fraction.
};
```

4.6 Initialization Sections

- An initialization section in a function definition provides an alternative way to initialize member variables

```
BankAccount::BankAccount( ): balance(0), interest_rate(0.0)
{
        // No code needed in this example
}
```

  - The values in parenthesis are the initial values for the member variables listed

4.6.1 Parameters and Initialization

- Member functions with parameters can use initialization sections

```
BankAccount::BankAccount(int dollars, int cents, double rate)
                        : balance (dollars + 0.01 * cents), interest_rate(rate)
{
        if (( dollars < 0) || (cents < 0) || (rate < 0)) {
                cout << "Illegal values for money or rate\n";
                exit(1);
        }
}
```

- Notice that the parameters can be arguments in the initialization

4.6.2 sample26.cpp