Программирование

Лекция 7

Объектно-ориентированное программирование (ООП)

- (ООП) — это особый концептуальный подход к проектированию программ, и C++ расширяет язык C средствами, облегчающими применение такого подхода.
- Наиболее важные характеристики ООП:
  - абстракция;
  - инкапсуляция и сокрытие данных;
  - полиморфизм;
  - наследование;
  - повторное использование кода.

Процедурное программирование и ООП

- При процедурном подходе вы сначала концентрируетесь на **процедурах**, которым должны следовать, а только потом думаете о том, как представить данные.
- При объектно-ориентированном подходе вы концентрируетесь на объекте, как его представляет пользователь, думая о **данных**, которые нужны для описания объекта, и операциях, описывающих взаимодействие пользователя с данными. После разработки описания интерфейса вы перейдете к выработке решений о том, как реализовать этот интерфейс и как организовать хранение данных. И, наконец, вы соберете все это вместе в программу, соответствующую новому проекту.

Абстракции и классы

- Жизнь полна сложностей, и единственный способ справится со сложностью — это ограничиться упрощенными абстракциями.
- В компьютерных вычислениях абстракция — это ключевой шаг в представлении информации в терминах ее интерфейса с пользователем. То есть вы абстрагируете основные операционные характеристики проблемы и выражаете решение в этих терминах.
- Например, тип char занимает 1 байт памяти, a double — обычно 8 байт.
- Например, к типу int можно применять все арифметические операции. Целые числа можно складывать, вычитать, умножать, делить.

Спецификация базового типа выполняет три вещи:

- Определяет, сколько памяти нужно объекту.
- Определяет, как интерпретируются биты памяти.
- Определяет, какие операции, или методы, могут быть применены с использованием этого объекта данных.

- Но когда вы определяете пользовательский тип в C++, то должны предоставить эту информацию самостоятельно.

Классы в C++

- Класс — это двигатель C++, предназначенный для трансляции абстракции в пользовательские типы.

- Обычно спецификация класса состоит из двух частей:
- *Объявление класса,* описывающее компоненты данных в терминах членов данных, а также открытый интерфейс в терминах функций-членов, называемых *методами*.
- *Определения методов класса,* которые описывают, как реализованы определенные функции-члены.

Что такое интерфейс?

- Интерфейс — это совместно используемая часть, предназначенная для взаимодействия двух систем, например, между компьютером и принтером или между пользователем и компьютерной программой.
- Например, чтобы определить количество символов в объекте string, вам не нужно открывать этот объект и смотреть что у него внутри. Вы просто используете метод size () класса, предоставленный его разработчиком. Таким образом, метод size () является частью открытого интерфейса между пользователем и объектом класса string.
- Обычно программисты на C++ помещают интерфейс, имеющий форму определения класса, в заголовочный файл, а реализацию в форме кода для методов класса — в файл исходного кода.

Создание класса

- Давайте взглянем на класс, представляющий акционерный капитал.
- Операции:
  - приобретение пакета акций компании;
  - приобретение дополнительных акций в имеющийся пакет;
  - продажа пакета;
  - обновление объема доли в пакете акций;
  - отображения информации о пакетах, находящихся во владении.
- Сведения для сохранения:
  - название компании;
  - количество акций, находящихся во владении;
  - объем каждой доли;
  - общий объем всех долей.

Объявление класса под именем Stock

```
#ifndef STOCK00_H_
#define STOCK00_H_

#include <string>

class Stock  // объявление класса
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    void acquire(const std::string & co, long n, double pr);
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
}; // Обратите внимание на точку с запятой в конце.

#endif
```

#ifndef применяется для защиты против многократного включения файла

Соглашение о написании имен классов с заглавной буквы

Экземпляры класса:

```
Stock sally;
Stock solly;
```

Управление доступом

- Слова private и public позволяют управлять доступом к членам класса. Любая программа, которая использует объект определенного класса, может иметь непосредственный доступ к членам из раздела **public**. Доступ к членам объекта из раздела **private** программа может получить только через открытые функции-члены из раздела public.
- Открытые функции-члены действуют в качестве посредников между программой и закрытыми членами объекта; они предоставляют интерфейс между объектом и программой. Эта изоляция данных от прямого доступа со [illegible] называется **сокрыт**

Сокрытие данных предохраняет целостность данных!

Инкапсуляция

- Открытый интерфейс представляет абстрактный компонент проектного решения.
- Собрание деталей реализации в одном месте и отделение их от абстракции называется **инкапсуляцией**. *Сокрытие данных* (помещение данных в раздел private класса) является примером инкапсуляции, и поэтому оно скрывает функциональные детали реализации в разделе private.

public или private?

- Поскольку одним из главных принципов ООП является сокрытие данных, то единицы данных обычно размещаются в разделе private.
- Использовать ключевое слово private в объявлении класса не обязательно, поскольку это спецификатор доступа к объектам класса по умолчанию:

```
class World
{
    float mass;              // по умолчанию private
    char name[20];           // по умолчанию private
public:
    void tellall(void);
    ...
}
```

Классы и структуры

- Описания классов выглядят очень похожими на объявления структур с дополнениями в виде функций-членов и меток видимости private и public.
- Фактически C++ расширяет на структуры те же самые свойства, которые есть у классов. Единственная разница состоит в том, что типом доступа по умолчанию у структур является public, в то время как у классов — private.

Реализация функций-членов класса

- Определения функций-членов очень похожи на определения обычных функций.
- При определении функции-члена для идентификации класса, которому принадлежит функция, используется операция разрешения контекста (: :).
- Методы класса имеют доступ к private-компонентам класса

```
void Stock::update(double price)

void Button::update(double price)
```

Реализация класса Stock

```
#include <iostream>
#include "stock00.h"
void Stock::acquire(const std::string & co, long n, double pr)
{
    company = co;
    if (n < 0)
    {
        // Количество пакетов не может быть отрицательным; устанавливается в 0.
        std::cout << "Number of shares can't be negative; "
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
void Stock::buy(long num, double price)
{
    if (num < 0)
    {
        //Количество приобретаемых пакетов не может быть отрицательным. Транзакция прервана.
        std::cout << "Number of shares purchased can't be negative. "
                  << "Transaction is aborted.\n";
    }
    else
    {
        shares += num;
        share_val = price;
        set_tot();
    }
}
```

```
void Stock::sell(long num, double price)
{
    using std::cout;
    if (num < 0)
    {
        // Количество продаваемых пакетов не может быть отрицательным. Транзакция прервана.
        cout << "Number of shares sold can't be negative. "
             << "Transaction is aborted.\n";
    }
    else if (num > shares)
    {
        // Нельзя продать больше того, чем находится во владении. Транзакция прервана.
        cout << "You can't sell more than you have! "
             << "Transaction is aborted.\n";
    }
    else
    {
        shares -= num;
        share_val = price;
        set_tot();
    }
}
void Stock::update(double price)
{
    share_val = price;
    set_tot();
}
void Stock::show()
{
    // Вывод названия компании, количества пакетов, цены пакета и общей стоимости.
    std::cout << "Company: " << company
              << " Shares: " << shares << '\n'
              << " Share Price: $" << share_val
              << " Total Worth: $" << total_val << '\n';
}
```

Использование классов

- Целью языка C++ является сделать применение классов насколько возможно простым — подобно базовым встроенным типам вроде int и char.
- Создавать объект класса можно за счет объявления переменной этого класса либо использования операция new для размещения в памяти объекта этого класса. Объекты можно передавать в аргументах, возвращать их из функций, присваивать один объект другому.
- Создим объект типа Stock по имени f luffy_the_cat.

```
// Компилируется вместе с stock00.cpp
#include <iostream>
#include "stock00.h"
int main()
{
    Stock fluffy_the_cat;
    fluffy_the_cat.acquire("NanoSmart", 20, 12.50);
    fluffy_the_cat.show();
    fluffy_the_cat.buy(15, 18.125);
    fluffy_the_cat.show();
    fluffy_the_cat.sell(400, 20.00);
    fluffy_the_cat.show();
    fluffy_the_cat.buy(300000,40.125);
    fluffy_the_cat.show();
    fluffy_the_cat.sell(300000,0.125);
    fluffy_the_cat.show();
    return 0;
}
```

```
Company: NanoSmart  Shares: 20
  Share Price: $12.5  Total Worth: $250
Company: NanoSmart  Shares: 35
  Share Price: $18.125  Total Worth: $634.375
You can't sell more than you have! Transaction is aborted.
Company: NanoSmart  Shares: 35
  Share Price: $18.125  Total Worth: $634.375
Company: NanoSmart  Shares: 300035
  Share Price: $40.125  Total Worth: $1.20389e+007
Company: NanoSmart  Shares: 35
  Share Price: $0.125  Total Worth: $4.375
```

Клиент-серверная модель

- **Клиентом** является программа, которая использует класс.
- Объявление класса, включая его методы, образует **сервер**, который является ресурсом, доступным нуждающейся в нем программе.
- Клиент взаимодействует с сервером только через открытый (public) интерфейс. Это означает, что единственной ответственностью клиента и, как следствие — программиста, является знание интерфейса.
- Ответственностью сервера и, как следствие — его разработчика, является обеспечение того, чтобы его реализация надежно и точно соответствовала интерфейсу. Любые изменения, вносимые разработчиком сервера в класс, должны касаться деталей реализации, но не интерфейса. Это позволяет программистам разрабатывать клиент и сервер независимо друг от друга, без внесения в сервер таких изменений, которые нежелательным образом отобразятся на поведении клиента.

Конструкторы и деструкторы классов

```
int year = 2001;    // допустимая инициализация
struct thing
{
    char * pn;
    int m;
};

thing amabob = {"wodget", -23};                    // допустимая инициализация
Stock hot = {"Sukie's Autos, Inc.", 200, 50.25};  // ошибка компиляции
```

```
Stock gift;
gift.buy(10, 24.75);
```

- Для автоматической инициализации объектов при их создании в C++ предлагаются специальные функции-члены, называемые **конструкторами класса**, которые предназначены для создания новых объектов и присваивания значений их членам-данным.
- Имя метода конструктора совпадает с именем класса. Например, возможный конструктор для класса Stock — это функция-член Stock ().
- Конструкторы не имеют возвращаемого значения, но они не объявляются с типом void. Фактически конструкторы не имеют объявленного типа.

Конструктор для класса Stock

```
// Прототип конструктора с несколькими аргументами по умолчанию
Stock(const string & co, long n = 0, double pr = 0.0);

// Определение конструктора
Stock::Stock(const string & co, long n, double pr)
{
    company = co;
    if (n < 0)
    {
        std::cerr << "Number of shares can't be negative; "
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
```

Это тот же код, который использовался для функции acquire (). Разница в том, что в данном случае программа автоматически вызовет конструктор при объявлении объекта!

Использование конструкторов

```
Stock food = Stock("World Cabbage", 250, 1.25);   // вызов конструктора явно

Stock garment("Furry Mason", 50, 2.5);            // вызов конструктора неявно

Stock *pstock = new Stock("Electroshock Games",18,19.0);   // динамическое выделение памяти
```

Деструкторы

- Деструктор призван очищать всяческий «мусор».
- Деструктор формируется из имени класса и предваряющего его символа тильды (~).
- Подобно конструктору, деструктор не имеет ни возвращаемого значения, ни объявляемого типа. Однако в отличие от конструктора, деструктор не должен иметь аргументы.

```
~Stock();        // прототип деструктора
```

```
Stock::~Stock()      // реализация
{                    деструктора
}
```

```
Stock::~Stock()
{
    cout << "Bye, " << company << "!\n";
}
```

Усовершенствование класса Stock

```
// stock10.h -- объявление класса Stock с добавленными конструкторами и деструктором
#ifndef STOCK10_H_
#define STOCK01_H_
#include <string>

class Stock
{

private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }

public:
    // Два конструктора
    Stock();          // конструктор по умолчанию
    Stock(const std::string & co, long n = 0, double pr = 0.0);
    ~Stock();         // деструктор
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};

#endif
```

Файл реализации

```
// stock10.cpp -- реализация класса Stock с добавленными конструкторами и деструктором
#include <iostream>
#include "stock10.h"

// Конструкторы (версии с выводом сообщений)
Stock::Stock()        // конструктор по умолчанию
{
    std::cout << "Default constructor called\n";
    company = "no name";
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}

Stock::Stock(const std::string & co, long n, double pr)
{
    std::cout << "Constructor using " << co << " called\n";
    company = co;
    if (n < 0)
    {
        std::cout << "Number of shares can't be negative; "
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}

// Деструктор класса
Stock::~Stock()      // деструктор класса, отображающий сообщение
{
    std::cout << "Bye, " << company << "!\n";
}
```

Файл клиентской программы

```
#include <iostream>
#include "stock10.h"

int main()
{
    {
        using std::cout;
        // Использование конструкторов для создания новых объектов
        cout << "Using constructors to create new objects\n";
        Stock stock1("NanoSmart", 12, 20.0);              // первый синтаксис
        stock1.show();
        Stock stock2 = Stock ("Boffo Objects", 2, 2.0);   // второй синтаксис
        stock2.show();
        // Присваивание stock1 объекту stock2
        cout << "Assigning stock1 to stock2:\n";
        stock2 = stock1;
        // Вывод stock1 и stock2
        cout << "Listing stock1 and stock2:\n";
        stock1.show();
        stock2.show();
        // Использование конструктора для сброса объекта
        cout << "Using a constructor to reset an object\n";
        stock1 = Stock("Nifty Foods", 10, 50.0);          // временный объект
        cout << "Revised stock1:\n";
        stock1.show();
        cout << "Done\n";
    }
    return 0;
}
```

Вывод программы

```
Using constructors to create new objects
Constructor using NanoSmart called
Company: NanoSmart  Shares: 12
  Share Price: $20.00  Total Worth: $240.00
Constructor using Boffo Objects called
Company: Boffo Objects  Shares: 2
  Share Price: $2.00  Total Worth: $4.00
Assigning stock1 to stock2:
Listing stock1 and stock2:
Company: NanoSmart  Shares: 12
  Share Price: $20.00  Total Worth: $240.00
Company: NanoSmart  Shares: 12
  Share Price: $20.00  Total Worth: $240.00
Using a constructor to reset an object
Constructor using Nifty Foods called
Bye, Nifty Foods!
Revised stock1:
Company: Nifty Foods  Shares: 10
  Share Price: $50.00  Total Worth: $500.00
Done
Bye, NanoSmart!
Bye, Nifty Foods!
```

Указатель this

Создадим функцию, работающую с двумя объектами с целью их сравнения

```
const Stock & topval(const Stock & s) const;        // прототип функции

const Stock & Stock::topval(const Stock & s) const  // реализация функции
{
   if (s.total_val > total_val)
      return s;
   else
      return *this;
}
```

Массив объектов

```
Stock mystuff[4];          // создание массива из 4 объектов Stock

mystuff[0].update();       // применяет update() к первому элементу
mystuff[3].show();         // применяет show() к 4-му элементу
const Stock * tops = mystuff[2].topval(mystuff[1]);
    // сравнивает 2-й и 3-й элементы и устанавливает tops
    // в указатель на тот из них, у которого больше значение total_val
```

Для инициализации элементов массива можно использовать конструктор. В этом случае необходимо вызывать конструктор для каждого индивидуального элемента:

```
const int STKS = 4;
Stock stocks[STKS] = {
    Stock("NanoSmart", 12.5, 20),
    Stock("Boffo Objects", 200, 2.0),
    Stock("Monolithic Obelisks", 130, 3.25),
    Stock("Fleep Enterprises", 60, 6.5)
};
```

Вопросы

- Что такое класс?
- Что такое инкапсуляция?
- Каково отношение между объектом и классом?
- Определите класс для представления банковского счета. Данные-члены должны включать имя вкладчика, номер счета (используйте строку) и баланс. Функции-члены должны позволять следующее:
  - создание объекта и его инициализация;
  - отображение имени вкладчика, номера счета и баланса;
  - добавление на счет суммы денег, переданной в аргументе;
  - снятие суммы денег, переданной в аргументе.
- Приведите объявление класса без реализации методов.

Вопросы

- Когда вызываются конструкторы класса? Когда вызываются деструкторы?
- Напишите код конструктора для класса банковского счета, описанного в вопросе 5.
- Что такое конструктор по умолчанию?