ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

10-2012

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим работам по дисциплине для студентов направления 230100.62 «Программирование» профиля «Системы автоматизированного проектирования в машиностроении» очной формы обучения

Часть 3

Воронеж 2012

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов,
д-р техн. наук М.И. Чижов,
ст. преп. В.А. Рыжков

УДК 004.9

Методические указания к практическим работам по дисциплине “Программирование” для студентов направления 230100.62 «Информатика и вычислительная техника» профиля «Системы автоматизированного проектирования в машиностроении» очной формы обучения. Ч. 3 / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, М.И. Чижов, В.А. Рыжков. Воронеж, 2012. 31 с.

Методические указания содержат перечень базовых разделов с примерами решения задач на языке С++ по дисциплине «Программирование на языках высокого уровня».

Предназначены для студентов 1 курса.

Методические указания подготовлены в электронном виде в текстовом редакторе Microsoft Word 2007, содержащаяся в файле МУ 2012_ч3.doc

Ил. 5. Библиогр.: 8 назв.

Рецензент канд. техн. наук, доц. Л.А. Иванов

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Разрабатываемое на данный момент программное обеспечение имеет достаточно сложную структуру построения, а отдельные части программы включают в себя алгоритмы реализации, написанные на отличных друг от друга языках программирования. Построить взаимосвязи между частями программы непосредственно в процессе создания программных средств не всегда удается, а внесение изменений в проект является весьма трудоемкой задачей.

Объектно-ориентированное программирование (ООП) предоставляет технологию управления элементами любой сложности. Суть ООП состоит в том, чтобы обращаться с данными и методами, которые выполняют действия над этими данными, как с единым объектом.

Методы ООП позволяют перейти от алгоритмических моделей программ к объектным. При ООП пользователя в первую очередь заботят типы объектов, с которыми приходится иметь дело их программам, свойства этих объектов, а также то, как они взаимодействуют между собой и с другими пользователями.

Основные цели ООП:

-абстрактное представление данных для создания чётко определённого интерфейса всех объектов;

-моделирование объектов и их взаимодействия в программных реализациях.

В методических указаниях рассматриваются вопросы проектирования задач посредством применения концепций ООП, разбираются разделы, охватывающие методы реализации математических алгоритмов, приведены листинги готовых программ на алгоритмическом языке C++. Тексты программных проектов были подготовлены в интегрированной среде разработки Code::Blocks C++, дистрибутив которой можно найти по следующему адресу: www.codeblocks.org.

1. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ

1.1. Общие сведения и возможности интегрированной среды создания приложений Code::Blocks

Code::Blocks-свободная кроссплатформенная среда разработки. Code::Blocks написана на C++ и использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования C и C++. Code::Blocks разрабатывается для Windows, Linux и Mac OS X. Среду можно собрать из исходных текстов системы практически под любую Unix-подобную систему, например FreeBSD.

Среда разработки включает в себя:

-возможности компиляции;

-работу с графическим интерфейсом;

-имеются развитые средства по отладке программных средств;

В рассматриваемой среде разработки выполнена поддержка множества компиляторов. В число используемых реализаций включены следующие решения: MinGW / GCC C/C++, GNU ARM GCC Compiler, GNU AVR GCC Compiler, Digital Mars C/C++,SDCC (Small device C compiler), Microsoft Visual C++ 2005/2008/2010, OpenWatcom, Intel C++ compiler и т.д. Кроме того, в интегрированной среде реализованы многопрофильные проекты, поддержка рабочих пространств, импорт проектов Dev-C++, импорт проектов и рабочих пространств Microsoft Visual Studio (включая 2010). Причем для разных проектов можно подключить разные компиляторы, можно для одного и того же проекта попробовать различные компиляторы. Последнее должно быть особенно удобно для тех, кто разрабатывает open source проекты, которые просто обязаны собираться всеми более-менее распространенными компиляторами. Плюсом можно считать переход на другой компилятор достаточно быстро. Для этого выбирается в списке

другой компилятор, потом производится работа с ним. Если выбранное решение не соответствует выбранному уроню, имеется возможность вернуть все обратно, сохранив все настройки. Никаких дополнительных сред разработки скачивать не надо, привыкать к ним не надо.

К возможности интерфейса среды следует отнести следующие:

-подсветка синтаксиса;

-сворачивание блоков кода;

-автодополнение кода;

-специальные средства для просмотра классов

-система управления проекта с помощью команд скриптого языка на основе Squirrel;

-планировщик под несколько пользователей;

-поддержка установочных пакетов для Dev-C++.

1.2. Разработка проектов в Code::Blocks

Среда Code::Blocks обладает достаточными возможностями по разработке и поддержке проекта программного средства. На рисунке 1 показан интерфейс рассматриваемой интегрированной среды.

Среда представлена оконным интерфейсом с интегрированными средствами управления данными. Наличие текстового меню, а также графических средств быстрого доступа к командам системы позволяет значительно повысить производительность работы при создании программного средства. Опции по компиляции проекта, отладочные средства, меню по правке листинга и т.д. вынесены на первый план. Текстовый редактор информативен, а синтаксис программы, как упоминалось ранее, “подсвечен” разными цветами. Вкладка с проектом вынесена в отдельное окно, где перечислены все составляющие проектного решения (файлы с исходным текстом .cpp, заголовки библиотек и классов .h и т.д.).

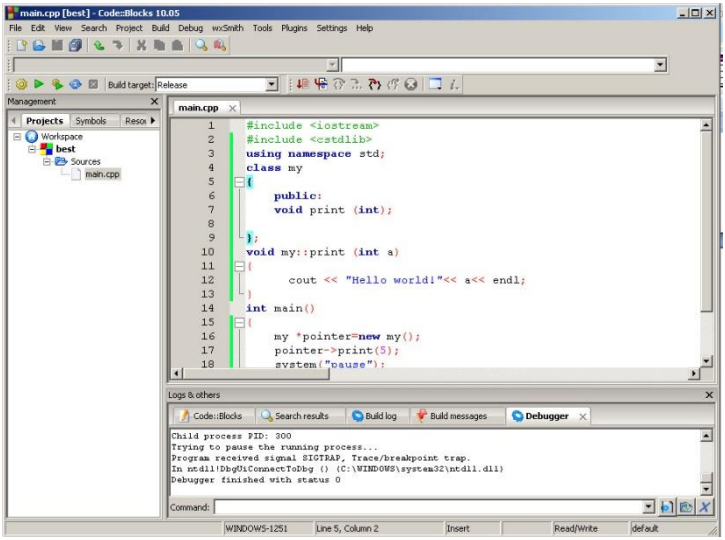

Рис. 1. Интегрированная среда Code::Blocks

В отдельное информационное окно вынесена система сообщений и результаты действий при разработке проекта. Для создания консольного приложения необходимо выбрать соответствующий шаблон проекта, как показано на рисунке 2.

Рис.2. Выбор типа создаваемого приложения

Следуя указаниям мастера по созданию приложения, система подготовит проект консольной программы (рисунок 3). Проект при создании включает файл main.cpp и содержит минимальный программный код по выводу приветственного сообщения на экран.

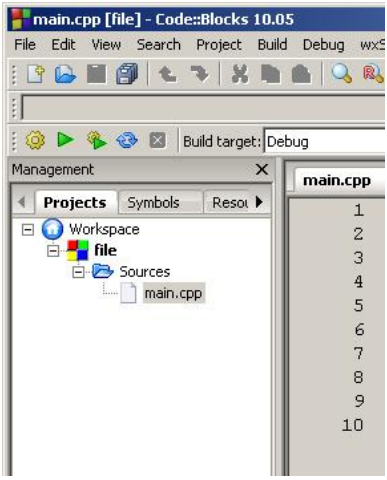

Рис. 3. Проект консольного приложения, созданного посредством мастера шаблонов

Чтобы подготовить исполняемый файл для ОС Windows, требуется собрать проект с помощью средств компиляции интегрированной среды, как показано на рисунке 4. Кнопка, обозначенная 1 позицией, позволяет построить проект. По кнопке 2 производится вызов полученного исполняемого файла. Кнопка под номером три выполняет два действия – собирает проект и производит его запуск. По кнопке 4 имеется возможность перестроить проект согласно внесенным изменениям в него - данная процедура работает быстрее в том случае, когда проект содержит несколько решений и нужно пересобрать лишь часть из них. Кнопка, обозначенная 5 позицией, дает возможность подготовить отладочный или финальный проект. Отладочный проект, в процессе разработки программного средства, имеет ряд преимуществ: просмотр значений переменных на заданном

шаге выполнения, возвращаемые значения функций, инициализацию и связывание указателей и т.д.

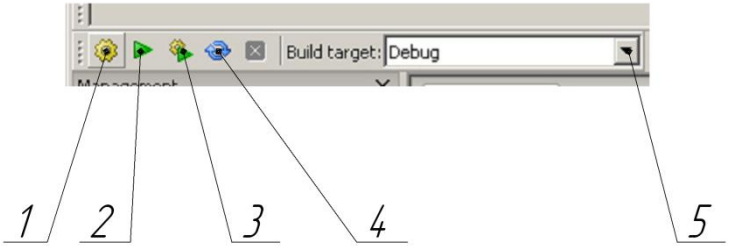

Рис. 4. Компоновочные опции проекта

При отладке проекта в интегрированной среде разработки появляются дополнительные возможности по управлению проектом (рисунок 5).

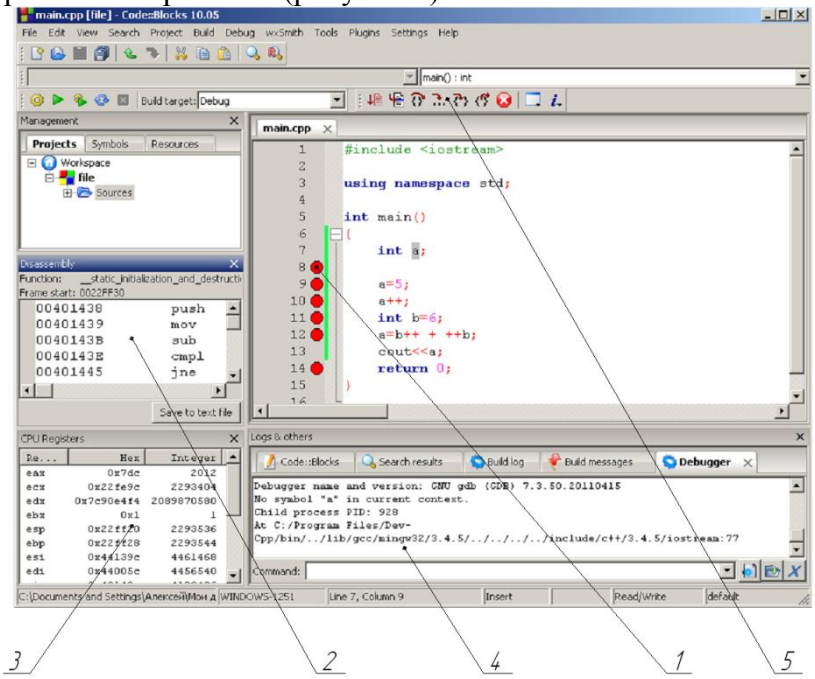

Рис. 5. Отладочные возможности системы Code::Blocks

К ним относятся: использование точек прерывания при построчном анализе данных (позиция 1), например слежение за значением переменной, просмотр программы в собранном виде посредством опции дизассемблирования (позиция 2), просмотр и изменение регистров процессора (позиция 3), отображения окна состояния (позиция 4), использование инструментальной панели для отладки приложения (позиция 5).

2. ОБЪЕКТЫ И КЛАССЫ

2.1. Объявление и спецификация класса

Для описания предмета методами объектно-ориентированного программирования в языке C++ используют классы. Класс-это некоторый объект, в котором объявлены данные и указан порядок действий, приемы работы с ними. Определение класса состоит из двух частей: заголовка, включающего ключевое слово class, за которым следует имя класса, и тела, заключенного в фигурные скобки. После такого определения должны стоять точка с запятой или список объявлений:

```
class Screen { /* ... */ };
class Screen { /* ... */ } myScreen, yourScreen;
```

Пример листинга программы с описанием класса в одном файле среды Code::Blocks

```
#include <iostream>
 using namespace std;
  class newclass
   {
    //Заполняется согласно условию задачи
   }n;
int main()
  {
    newclass m;
    return 0;
  }
```

Внутри тела объявляются данные-члены и функции-члены, которые именуются как методы класса, а также указываются уровни доступа к ним. Уровни доступа будут рассмотрены подробным образом в следующих разделах. Таким образом, тело класса определяет список его членов.

Каждое определение вводит новый тип данных. Даже если два класса имеют одинаковые списки членов, они все равно считаются разными типами:

```
class First
{
   int a;
   double b;
};
class Second
{
   int a;
   double b;
};
int main()
{
 First obj1;
 //Second obj2 = obj1;   //ошибка: obj1 и obj2
 return 0;
}
```

Тело класса определяет отдельную область видимости. Объявление членов внутри тела помещает их имена в область видимости класса. Наличие в двух разных классах членов с одинаковыми именами – не ошибка, эти имена относятся к разным объектам.

После того как тип класса определен, на него можно ссылаться двумя способами:

-написать ключевое слово class, а после него – имя класса. В предыдущем примере объект obj1 класса First объявлен именно таким образом;

-указать только имя класса. Так объявлен объект obj2 класса Second из приведенного примера.

Оба способа сослаться на тип класса эквивалентны. Первый заимствован из языка С и остается корректным методом задания типа класса; второй способ введен в С++ для упрощения объявлений.

2.2. Элементы класса

К элементам класса относятся данные, имеющие заданный тип согласно специфике языка, и методы, которые реализуют те или действия для данного класса.

Методы класса объявляются в его теле. Это объявление выглядит точно так же, как объявление функции в области видимости пространства имен. (Напомним, что глобальная область видимости – это тоже область видимости пространства имен. Например:

```
class Screen {
public:
   void home();
   void move( int, int );
   char get();
   char get( int, int );
   void checkRange( int, int );
   // ...
};
```

Определение метода-члена для указанного класса также можно поместить внутрь тела класса:

```
class Screen {
    int _x,_y;
public:
   // определения функций home() и add()
   void home() { _x = 10;_y=20; }
   int add() { return (_x+_y); }
   // ...
};
```

Методы класса отличаются от обычных функций следующим:

-функция-метод объявлена в области видимости своего класса, следовательно, её имя не видно за пределами этой

области. К функции-методу можно обратиться с помощью одного из операторов доступа к членам – точки (.) или стрелки (->):

```
ptrScreen->home();
myScreen.home();
```

-функции-методы имеют право доступа, как к открытым, так и к закрытым членам класса, тогда как обычным функциям доступны лишь открытые. Конечно, функции-методы одного класса, как правило, не имеют доступа к данным-методам другого класса.

2.3 Ассоциативная связь элементов с классом

Методы реализации вычислительных или иных действий могут быть описаны в классе - в таком случае после заголовка функции следует конструкция в фигурных скобках. Другим вариантом принадлежности функции к методам класса является использование внесение имени класса в прототип метода за пределами уже созданного объекта. Выглядит такая конструкция следующим образом:

```
#include <iostream>
using namespace std;
class my_class {
    int posx;
    int posy;
    int posz;
public:
   void show(int,int,int);
};
void my_class:: show(int a,int b,int c)
{
   posx=a;
   posy=b;
   posz=c;
   cout<<a<<"\t"<<b<<"\t"<<c<<endl;
}
int main()
{
    my_class h;
```

```
    h.show(5,7,9);
    return 0;
}
```

О том, что метод show () принадлежит рассматриваемому классу, в функции присутствует ссылка на имя класса с двумя знаками ::.

```
void my_class:: show(int a,int b,int c)
```

Для переменных класса применяется похожий механизм, но перед использованием необходимо объявить переменную с модификатором static.

```
static int all;
int my_class ::all=4;
```

2.4 Доступ к элементам класса

Прототипы функций и объявления элементов данных включаются в объявлении класса в разделы public (открытый) или private (закрытый). Ключевые слова public и private говорят компилятору о доступности элементов-функций и данных. Например, функция SetRadius() определена в разделе public, и это означает, что любая функция программы может вызвать функцию SetRadius(). Функция CalculateArea() определена в разделе private, и эту функцию можно вызвать только в коде функций-элементов класса Circle.

Аналогично, поскольку элемент данных radius объявлен в разделе private, прямой доступ к нему (для установки или чтения его значения) возможен только в коде функций-элементов класса Circle. Если бы элемент данных radius был объявлен в разделе public, то любая функция программы имела бы доступ (для чтения и присваивания) к элементу данных radius.

Задачи на самостоятельное решение

1.Посчитать в классе число точек, находящиеся внутри круга радиусом r с центром в точке с координатами (1,1); координаты заданы массивами X(10), Y(10).

2. Построить класс, в котором определить динамический массив, ввести данные и получить индексы отрицательных элементов.

3. Определить класс, в котором размещена информация о студентах заданного факультета. Предусмотреть ввод-вывод данных.

3. КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ В КЛАССАХ

3.1. Назначение и порядок использования конструкторов (деструкторов)

Конструктором называется специальная функция, назначение которой заключается в присвоении элементам класса начальных значений или выделение памяти под создаваемые объекты.

Конструкторы отличаются от прочих методов функций тем, что имеют то же самое имя, что и класс, к которому они относятся. При создании или копировании объекта данного класса происходит неявный вызов соответствующего конструктора. Функция-конструктор не имеет возвращаемых параметров.

Конструктор запускается при создании объекта данного класса. Пример программы с конструктором представлен на следующем листинге.

```
#include <cstdlib>
#include <iostream>
using namespace std;
class sample
{
    public:
    sample()
    {
      cout<<"Constructor"<<endl;
    }
};
```

```
int main()
{   //1 форма объявления
    sample var1;
    //2 форма объявления
    sample var2=sample();
    //3 форма объявления
    sample var3=sample::sample();
    //4 форма объявления
    sample *var4=new sample();
    //delete var4;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3.2. Конструкторы с параметрами

Параметры конструкторы могут быть любого типа, за исключением класса, компонентом которого является данный конструктор. Конструктор может принимать в качестве параметра ссылку на свой собственный класс; в таком случае он называется конструктором копирования. Конструктор, не принимающий параметров вообще, называется конструктором по умолчанию.

Конструктором по умолчанию для класса X называется такой конструктор, который не принимает никаких аргументов. Если для класса не существует конструкторов, определяемых пользователем, то C++ генерирует конструктор по умолчанию. При таких объявлениях, как X x, конструктор по умолчанию создает объект x.

Как и все функции, конструкторы могут иметь аргументы по умолчанию.

```
#include <cstdlib>
#include <iostream>
using namespace std;
class number
{
    public:
```

```
    number(int var=0)
    {
      cout<<"Number is..."<<var<<endl;
    }
};
int main()
{
    number a;
    number b(5);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3.3. Перегруженные конструкторы

Конструкторы могут быть перегружены, что позволяет создание объектов в зависимости от значений, использованных при инициализации.

```
#include <cstdlib>
#include <iostream>
using namespace std;
class x
{
      int integer_part;
      double double_part;
      public:
      x(int i)
        { cout<<i<<endl; }
      x(double d)
        { cout<<d<<endl; }
    };

    main()
{
    x one1(10);
    x one2(3.14);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3.4. Деструкторы в классах

Деструктор решает задачу, обратную задаче конструктора. Он вызывается всякий раз, когда объект уничтожается. Имя деструктора состоит из знака тильды ”~” и имени класса. Например, ~date().

Деструктор не имеет параметров и не возвращает значения. Компилятор сам генерирует деструктор, если программист его не определил. Когда объект выходит из области видимости, деструктор вызывается автоматически. Явный вызов деструктора никогда не требуется.

Задачи на самостоятельное решение

1.Задана функция вида y=1-x. Определить, принадлежит ли точка с координатами (-0.5, 1.5), передав указанные координаты в конструктор класса.

2. В класс передать одномерный массив и подготовить метод по определению номера первого по порядку нулевого элемента.

3. Составить класс по нахождению в одномерном массиве A(N) элемента, встречающегося наибольшее число раз и, если таких групп несколько, выбрать группу с наименьшим из них.

4. ПРОСТОЕ И МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ КЛАССОВ

4.1. Назначение и общие принципы наследования

Наследование - один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым

классом. Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

4.2. Простое наследование классов

Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class). В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Пример простого наследования классов и области видимости элементов базового класса за его пределами при заданных спецификаторах.

```
class some
{
friend void f(some&);
 public:
 int a_;
 protected:
 int b_;
```

```
 private:
 int c_;
};
 void f(some& obj) {
 obj.a_ = 0; // ok
 obj.b_ = 0; // ok
 obj.c_ = 0; // ok
}
 void g(some& obj) {
 obj.a_ = 0; // ok
 //obj.b_ = 0; // CT error
 //obj.c_ = 0; // CT error
}
class derived : public some
{
 derived() {
 a_ = 0; // ok
 b_ = 0; // ok
 //c_ = 0; // CT error
 }
};
```

4.3. Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Пример множественного наследования в программе

```
#include <cstdlib>
#include <iostream>
using namespace std;
class c1
{   int a1;
    protected:
```

```
    void cout_c1(int a2)
    {a1=a2;
    cout<<a1<<endl;}
};
class c2
{   int a3;
    protected:
    void cout_c2(int a4)
    {a3=a4;
    cout<<a3<<endl;}
};
class c3:public c1,public c2
{ int i;
  public:
  void init()
  { cin>>i;
    cout_c1(i);
    cout_c2(i*2); }
};
int main(int argc, char *argv[])
{   c3 *adress=new c3();
    adress->init();
    delete adress;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

4.4.Модификаторы доступа при наследовании

В C++ существует три типа наследования: public, protected, private. Спецификаторы доступа членов базового класса меняются в потомках следующим образом:

-при public-наследовании все спецификаторы остаются без изменения.

-при protected-наследовании все спецификаторы остаются без изменения, кроме спецификатора public, который меняется на спецификатор protected (то есть public-члены базового класса в потомках становятся protected).

-при private-наследовании все спецификаторы меняются на private.

Теперь кратко, кому какой доступ они предоставляют.

-public – доступ открыт всем, кто видит определение данного класса.

-private – доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса, как функциям, так и классам.

-protected – доступ открыт классам, производным от данного.

Задачи на самостоятельное решение

1.В функциях независимых классов реализовать по одному из математических действий (+, -, /,*), а в наследуемом классе вывести результаты решений в указанных функциях.

2.В одном классе определен одномерный массив, а в другом классе - многомерный. Определить в унаследованном от 2 классов классе размерность каждого из массивов.

3.В одном классе определить значения трех констант, а в другом осуществить вызов перечисленных констант в произвольном выражении.

5. ВИРТУАЛЬНЫЕ МЕТОДЫ ПРИ НАСЛЕДОВАНИИ КЛАССОВ

5.1. Раннее и позднее связывание в программах

Для того чтобы освоить методы использования виртуальных механизмов на языке C++, необходимо рассмотреть такие понятия как раннее и позднее связывание.

Рассмотрим два подхода к решению задач. В одном случае требуется объявить массив, состоящий из десяти целочисленных значений. В данном случае массив жестко определен заданным условием, поэтому форма записи с

учетом синтаксиса языка C++ будет выглядеть следующим образом: `int array[10]`. Во втором случае требуется объявить целочисленный массив, размерность которого определит пользователь в процессе эксплуатации программного средства. Для примера со стороны пользователя должен поступить запрос, в котором указывается число элементов, равное 10. В таком случае форма записи текста программы будет выглядеть следующим образом:

```
int a,*p;
cin>>a;
p=new int [a];
delete [] p;
```

Приведенные подходы в определенной мере отражают смысл применения раннего и позднего связывания при разработке программ. Очевидно, что для решения конкретной задачи первый подход оптимален. Однако при изменении размерности массива на 100, 1000 и т.д. оптимальным является второй подход, так как в предыдущем случае программу изменить невозможно, если проект выполнен виде исполняемого файла.

Таким образом, во время раннего связывания вызывающий и вызываемый методы связываются при первом удобном случае, обычно при компиляции.

При позднем связывании вызываемого метода и вызывающего метода они не могут быть связаны во время компиляции. Поэтому реализован специальный механизм, который определяет, как будет происходить связывание вызываемого и вызывающего методов, когда вызов будет сделан фактически.

Очевидно, что скорость и эффективность при раннем связывании выше, чем при использовании позднего связывания. В то же время, позднее связывание обеспечивает некоторую универсальность связывания.

5.2. Виртуальные функции в классах

Виртуальные методы существуют для того, чтобы производный класс вел себя отлично от базового класса. При этом сохранялось свойство совместимости с базовой реализацией указанного класса.

Виртуальный метод - это метод, который, будучи описан в потомках, замещает собой соответствующий метод везде, даже в методах, описанных для предка, если он вызывается для потомка.

Адрес виртуального метода известен только в момент выполнения программы. Когда происходит вызов виртуального метода, его адрес берется из таблицы виртуальных методов своего класса. Таким образом, вызывается та конструкция, которая требуется для решения задач. Преимущество применения виртуальных методов заключается в том, что при вызове функций используется именно механизм позднего связывания, который допускает обработку объектов, тип которых неизвестен во время компиляции.

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Base
{
 public:
 virtual
   int f(const int &d)
   {return 2*d;}
   int CallFunction(const int &d)
    {
      return f(d)+1;
    }
};
 class Next: public Base
{   public:
    int f(const int &d)
    { return d*d; }
```

```
};
int main()
{
    Base a;
    cout << a.CallFunction(5)<< endl;
    Next b;
    cout << b.CallFunction(8)<< endl;
    system("pause");
    return 0;
}
```

5.3 Чисто виртуальные методы

Виртуальная функция-элемент некоторого класса может быть объявлена чистой. Это выглядит так:

```
virtual тип имя функции(список параметров) = 0;
```

Другими словами, тело функции объявляется нулем (задается чистым спецификатором). Действительная реализация ее не нужна (хотя и возможна). Предполагается, что чисто виртуальная функция будет переопределяться в классах, производных от него. Класс, объявляющий одну или несколько чисто виртуальных функций, является абстрактным базовым классом. Нельзя создать представитель такого класса. Он может служить только в качестве базового для порождения классов, которые полностью реализуют его чисто виртуальные функции.

Если класс не определяет чисто виртуальные функции абстрактного базового класса, то он также является абстрактным.

Абстрактные классы обладают следующими свойствами:

-не могут, как уже сказано, иметь представителей;

-не могут использоваться в качестве типа параметров или возвращаемых значений;

-не могут участвовать в явных приведениях типа.

Тем не менее, можно объявлять указатели или ссылки на абстрактный класс. Смысл абстрактных базовых классов в

том, что они способствуют лучшей концептуальной организации классовой иерархии и позволяют тем самым в полной мере использовать преимущества виртуальных механизмов C++.

Правила описания и использования виртуальных функций-методов следующие:

1. Виртуальная функция может быть только методом класса.

2. Любую перегружаемую операцию-метод класса можно сделать виртуальной, например, операцию присваивания или операцию преобразования типа.

3. Виртуальная функция, как и сама виртуальность, наследуется.

4. Виртуальная функция может быть константной.

5.Если в базовом классе определена виртуальная функция, то метод производного класса с такими же именем и прототипом (включая тип возвращаемого значения и константность метода) автоматически является виртуальным (слово virtual указывать необязательно) и замещает функцию-метод базового класса.

6. Конструкторы не могут быть виртуальными.

7. Статические методы не могут быть виртуальными.

8. Деструкторы могут (чаще — должны) быть виртуальными — это гарантирует корректный возврат памяти через указатель базового класса.

Подводя итог вышесказанному, виртуальные функции - важный и, по большому счету, необходимый механизм любой более-менее сложной системы иерархии классов,- его отсутствие, скорее всего, говорит о неправильном проектировании программы. В противоположность сказанному необходимо знать, что нецелесообразно определять новые виртуальные функции в конечных классах, т.е. классах, от которых не будет происходить наследование.

Задачи на самостоятельное решение

1.В функциях независимых классов реализовать по одному из математических действий (+, -, /,*), а в наследуемом классе вывести результаты решений в указанных функциях.

2.В одном классе определен одномерный массив, а в другом классе - многомерный. Определить в унаследованном от 2 классов классе размерность каждого из массивов.

3.В одном классе определить значения трех констант, а в другом осуществить вызов перечисленных констант в произвольном выражении.

6. ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ

Обобщенное программирование достигается не только с помощью наследования и полиморфизма, но и применение шаблонов. Шаблоны позволят определить параметризованные классы и функции, в которых параметрами служат не только переменные, но и типы.

Шаблоны - это механизм генерации типов. Сами по себе они не являются типами, никак не представлены во время выполнения и не оказывают влияния на модель размещения объектов в памяти.

Сначала в языке C++ была подготовлена реализация шаблонов исключительно для классов, позже были добавлены шаблоны функций.

6.1. Обобщенные (шаблонные) функции

Рассмотрим следующую ситуацию. В программе необходимо произвести сравнение пары чисел и, если выявлено минимальное, возвести его в квадрат. При этом не сказано, являются ли числа вещественными, целыми или же принадлежат к типу double. Решение задачи простое - для этих

целей следует определить функцию и использовать условный оператор if, в котором, в зависимости от выполнения условия, возвести полученное значение в степень, н-р, с помощью функции pow(), входящей в описание заголовочного файла <cmath>. Программа, которая производит вычисления без использования математической библиотеки, будет иметь следующий вид:

```
#include <iostream>
using namespace std;
int MinAndPow (int x,int y)
{
  if (x<y) return x*x;
  else return y*y;
}
float MinAndPow (float x,float y)
{
  if (x<y) return x*x;
  else return y*y;
}
double MinAndPow (double x,double y)
{
  if (x<y) return x*x;
  else return y*y;
}
int main()
{
    cout<< MinAndPow (2.5,3.5)<<endl;
    cout<< MinAndPow (2,3)<<endl;
    return 0;
}
```

Из примера видно, что предложено решение, которое представляет собой механизм перегруженных функций, но при этом функции возвращают отличный друг от друга тип. В своей основе функции похожи.

Запись листинга программы с помощью функции-шаблона выглядит проще.

```
#include <cstdlib>
#include <iostream>
using namespace std;
template <class T> T MinAndPow(T,T);
```

```
int main(int argc, char *argv[])
{
    int a,b;
    cin>>a>>b;
    cout<<MinAndPow(b,a);
    char a1,b1;
    cin>>a1>>b1;
    cout<<MinAndPow(b1,a1);
    system("PAUSE");
    return EXIT_SUCCESS;
}
template <class T> T MinAndPow(T x,T y)
{
  if (x<y) return x*x;
  else return y*y;
}
```

6.2 Шаблоны классов

Наряду с использованием шаблонов в функциях аналогичным образом возможна реализация универсальных подходов по созданию абстрактных данных в классах. Для демонстрации всех преимуществ шаблонов рассмотрим пример, в котором требуется посчитать значение функции $y=x^2$ при x=2 и 2.2. Посмотрим на реализацию задачи, когда используется обычный подход к созданию программы – для этих целей подготовим два класса.

```
#include "iostream"
using namespace std;
class funcInt
{
public:
      funcInt(int value)
      {
            x=value;
      }
      int SetValue()
      {
            return x*x;
      }
```

```
private:
	int x;
};
class funcDouble
{
public:
	funcDouble(double value)
	{
		x=value;
	}
	double SetValue()
	{
		return x*x;
	}
private:
	double x;
};
int main()
{
	funcInt fi(2);
	funcDouble fd(2.2);
	cout << "if int..." << fi.SetValue() << endl;
	cout << "if double..." << fd.SetValue() << endl;
	return 0;
}
```

Так как шаблоны-классы похожи на шаблоны-функции, то они реализуют аналогичные решения, т.е позволяют производить одинаковые операции с разными типами данных. Теперь рассмотрим указанную задачу с учетом вышесказанного.

```
	#include "iostream"
	using namespace std;

	template <class T>
	class func
	{
	public:
		func(T value)
		{
			x=value;
		}
```

```
            T SetValue()
            {
            return x*x;
            }
      private:
            T x;
      };
      int main()
      {
      func <int> fi(2);
      func <double> fd(2.2);
cout << "if template int..." << fi.SetValue() << endl;
cout << "if template double..."<<fd.SetValue()<< endl;
      return 0;
      }
```

Задачи на самостоятельное решение

1.Подготовить функцию-шаблон по переводу чисел из восьмеричной системы счисления в десятичную.

2. Подготовить класс, представляющий собой шаблон, а также определить в нем функцию, в которой производится упорядочивание элементов массива по возрастанию.

3. В функции-шаблоне поменять местами две переменные, не используя промежуточную переменную, и вывести большую из них на экран, не прибегая к операторам if и switch.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Довбуш Г.Ф. Visual C++ на примерах / Г.Ф. Довбуш, А.Д. Хомоненко. – СПб.: БХВ-Петербург, 2007. – 528 с.

2. Мюссер Д.Р. C++ и STL: справочное руководство / Д.Р. Мюссер, Ж.Дж. Дердж, А. Сейни. 2-е изд. - М.: ООО ”И.Д. Вильямс”, 2010. – 430 с.

3. Прата С. Язык программирования C++. Лекции и упражнения / С. Прата. 5-е изд. – М.: ООО ”И.Д. Вильямс”, 2007. – 1184 с.

4. Страуструп Б Язык программирования C++ /Б. Страуструп. - М.: Бином, 2011. – 1136 с.

5. Шилдт Г. C++ Базовый курс / Г. Шилдт. 3-е изд. – М.: ООО ”И.Д. Вильямс”, 2010. – 624 с.

6. Коплиен Дж. Программирование на C++ / Дж. Коплиен. – СПб.: Питер, 2005. – 480 с.

7. Roberge J. A laboratory course in C++ structures. 2ed./ J. Roberge, S. Brandl, D. Whittington. Jones and Bartlett, 2003. -411 p.

8. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. -841p.

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим работам по дисциплине “Программирование” для студентов направления 230100.62 «Информатика и вычислительная техника», профиля «Системы автоматизированного проектирования в машиностроении» очной формы обучения

Часть 3

Составители:
Юров Алексей Николаевич
Паринов Максим Викторович
Чижов Михаил Иванович
Рыжков Владимир Анатольевич

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 20.01.2012.
Уч.-изд. л. 1,9. «С»

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14