МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Уральский федеральный университет имени первого Президента России
Б.Н.Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ
Кафедра Информационных технологий

Реализация принципов инкапсуляции и полиморфизма. Классы вектор и матрица

Методические указания к лабораторной работе
по дисциплине «Технология проектирования и тестирования
программного обеспечения»
для студентов всех форм обучения по направлению
230100 – Информатика и вычислительная техника

Екатеринбург

2012

УДК 004.432.2

Составитель С.П.Трофимов
Научный редактор доц., канд. техн. наук В.П.Битюцкий

РЕАЛИЗАЦИЯ ПРИНЦИПОВ ИНКАПСУЛЯЦИИ И ПОЛИМОРФИЗМА. КЛАССЫ ВЕКТОР И МАТРИЦА: Метод. разработка к лабораторной работе по дисциплине «Технология проектирования и тестирования программного обеспечения» / сост. С.П. Трофимов. Екатеринбург: УрФУ, 2012. 14 с.

Даются основные понятия, связанные с созданием классов на зыке Си++, с ограничением доступа к элементам классов. Рассматриваются конструкторы и другие специальные методы, перегрузка операций, организация взаимодействия классов с потоками.

Приводятся контрольные вопросы, указания по подготовке и выполнению лабораторных заданий.

Указания предназначены для студентов всех форм обучения направления 230100 – «Информатика и вычислительная техника».

Библиогр.: 5 назв.

Подготовлено кафедрой «Информационные технологии»

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Ссылки

Ссылки используются, как правило, в списке аргументов функции и в качестве возвращаемого значения. В первом случае, в функцию передается оригинал и любые изменения ссылки внутри функции изменяют оригинал, хотя мы не разыменовываем ссылку. Ссылки упрощают использование переменных, переданных по адресу. Ссылка - это договоренность между компилятором и программистом, что такую-то переменную надо использовать, как оригинал.

Использование ссылок предпочтительнее, нежели передача указателей на оригинал.

Параметры - ссылки

Пример 1. Написать функцию нахождения минимального и максимального значений из двух целых чисел.

```
#include <stdio.h>
void MaxMin(int a, int b, int& max, int& min);
void MaxMin(int a, int b, int& max, int& min){
 max=(a<b)?b:a;
 min=(a<b)?a:b;
}
void main(){
```

```
 int max, min, a=2, b=3;
 MaxMin(a, b, max, min);
 printf("\nmax(%d, %d) = %d", a, b, max);
 printf("\nmin(%d, %d) = %d", a, b, min);
}
```

Пример 2. Изменим программу так, чтобы структуры передавались по адресу.

```
#include <graphics.h>
#include <conio.h>
struct Pixel{ int x, y;};
void DrawLine(const struct Pixel &a, const struct Pixel &b);
void DrawLine(const struct Pixel &a, const struct Pixel &b){
 line(a.x, a.y, b.x, b.y);
}
void main(){
 int gd=DETECT, gm;
 struct Pixel A={100, 100}, B={200, 300};
 initgraph(&gd, &gm,"c:\\bc2\\bgi");
 DrawLine(A, B);
 if(!getch()) getch();
 closegraph();
}
```

Функция, возвращающая значение типа ссылки

Если функция, возвращает значение типа ссылки, то она может стоять в левой части операции присваивания. Это приведет к изменению значения того, что функция возвращает. Еще раз. То, что возвращается, как ссылка может стоять и в левой, и в правой части присваивания. Такие объекты называются Lvalue. Объекты, которые могут стоять только в правой части операции присваивания, называются Rvalue.

Пример 3.

```
int i =1;
int& f(void);
int& f(void){ return i;}
void main(){
 int j=f();  // после этого оператора j=1
 f() = 2;    // после этого оператора i=2
}
```

В этом примере функции не будет разрешено вернуть локальную переменную, т.к. это лишено смысла - она все равно будет уничтожена по окончании функции.

Встроенные (inline) функции

Заставляют компилятор вставлять вместо вызова функции ее тело. Это ускоряет вычисления.

Пример 4. Сделаем функцию swap встроенной.

```
#include <stdio.h>
inline void swap(int &a, int &b){
 a=a+b; b=a-b; a=a-b;  return;
```

```
 //  a^=b^=a^=b;     //тот же эффект !
}
void main(){
 int a=2, b=3;
 printf("a=%d, b=%d", a, b);
 swap(a, b);
 printf("a=%d, b=%d", a, b);
}
```

Замечания. 1. Ошибочно передать функции swap параметры по значению - тогда изменятся лишь копии. Ограничения на inline - функции:

- нельзя использовать циклы;
- должны иметь тип void и оператор return;
- код должен стоять до первого вызова функции.

Операция разрешения видимости ::

Позволяет обратиться к глобальной переменной или функции, которая скрыта одной или несколькими локальными переменными.

Пример 5.

```
#include <stdio.h>
int i=10;                  //глобальная
void main(){
 int i=100;  //внешняя локальная
 if( i > 0 ){
     int i=1000;   //внутренняя локальная
     printf("\nлокальная i=%d", i);
     printf("\nглобальная i=%d", ::i);
 }
 return;
}
```

Перегруженные функции

Функции, у которых одинаковое имя, но разный список параметров, является разными, и называются перегруженными. Тип возвращаемого значения в расчет не принимается. Перегруженные функции реализуют принцип полиморфизма объектно-ориентированного подхода в программировании.

Пример 6. Следующие функции перегружены и являются разными.

```
void swap(int& a, int& b) {
 int buf = a;
 a=b; b=buf;
}
```

```
void swap(float& a, float& b){
    float buf = a;
    a=b; b=buf;
}
```

Новые операторы распределения памяти

Вместо функций malloc, calloc, free можно использовать опреаторы new, delete. Для проверки на ошибку можно создать свой обработчик с помощью функции set_new_handler, которой надо передать имя своего обработчика.

Пример 7. Записи на Си

```
int *pi = (int *)malloc(sizeof(int));
```

```
int *arrint = (int *)malloc(Dim*sizeof(int));
free(pi);
free(arrint);
```

эквивалентны записям на Си++

```
int *pi = new int;
int *arrint = new int[Dim];
delete pi;
dlete[] arrint;
```

Можно создать свой обработчик ошибки , возникающей при нехватке динамической памяти

Пример 8.

```
#include <iostream.h>
#include <new.h>
void MyHandler(void){
 cout << "Нет памяти !";
}
unsigned int Dim = 60000l;
void main(){
 set_new_handler(MyHandler);
 int *arrint = new int[Dim];
}
```

Объявление класса

class [имя_класса]{...} [имя_переменной];

При отсутствии имени_класса объявляется безымянный класс, для которого в другом месте невозможно определить переменные.

Классы являются типами, поэтому их объявляют. Переменные типа класса являются структурными переменными, под них сразу выделяется память, поэтому они определяются.

Пример 9. Объявим класс для хранения фамилии, имени и отчества.

```
class FIO{
 char fam[30],im[20],otch[20];
};
```

Определим переменную данного типа

```
FIO man;
```

Другой вариант класса

```
class FIO{
 char *fam,*im, *otch;
};
```

Синонимами являются: переменная типа класса, объект класса, представитель класса, экземпляр класса.

Элементы класса

Элементами класса являются элементы-функции и элементы-данные. Для краткости элементы-функции называют методами, а элементы-данные просто данными. Присутствие в теле класса элементов различной природы реализует принцип инкапсуляции объектно-ориентированного подхода в программировании.

Пример 10.

```
class Pixel{
public:int x, y, col;
     void put(void);
     int getcolor(void){
         return col;
     }
};
```

Определение метода может находиться в теле класса или за его областью действия

```
void Pixel::Put(void){
 putpixel(x, y, col);
};
```

Элементы-данные могут являться переменными базовых типов, массивами, объектами других классов, указатели на данный или другой класс.

Пример 11. class Ortezok{Pixel A, B;};

Задание. Определите метод Draw для класса Otrezok. Создайте объект этого класса и вызовите метод Draw.

Доступ к элементам класса

Доступ к элементам класса осуществляется так же, как к элементам структур. Используются две операции:

операция **.** ("точка") – обращение к элементу объекта по имени объекта,

операция **->** ("стрелка") – обращение к элементу объекта по указателю на объект.

Пример 12.

```
Point C, *D=&C;
C.x=C.y=0;
C.Draw();
D->x=639;
(*D).y=479;
D->Draw();
```

Управление доступом к элементам класса регулируется спецификаторами public, protected, private. По умолчанию действует private.

К публичным элементам объекта можно обращаться в любом месте области действия объекта по имени этого объекта.

К защищенным элементам объекта можно обращаться в методах производных классов.

К частным элементам можно обращаться только в методах этого класса. Также к частным элементам имеют доступ друзья класса.

Желательно данные объявлять в секции private, а функции доступа к ним – в секции public.

Пример 13.

```
class Point{
 private:
     float x, y;
 public:
     float getx(void){
```

```
        return x;
    };

;
```

Задание. Определите методы доступа gety, setx, sety.

Специальные методы класса

Некоторые методы являются обязательными или крайне желательными при объявлении классов. Как правило, они определяются первыми.

Таблица 1

Специальные методы класса

Метод	Описание
Конструктор по умолчанию	Определяет объект класса без фактических параметров
Конструктор инициализации	Определяет объект класса с фактическими параметрами
Конструктор копии	Определяет объект класса, используя данные уже существующего объекта класса
Операция присваивания	Присваивает содержимое одного существующего объекта другому существующему объекту
Деструктор	Реализует удаление представителя класса
Функция явного преобразования типа	Преобразует представитель класса к другому типу

Конструкторы вызываются при создании представителя класса. Они ничего не возвращают, и для них не указывается возвращаемый тип.

Конструктор по умолчанию в основном обнуляет данные создаваемого объекта. При отсутствии этого конструктора компилятор определит его сам с неинициализированными данными.

Конструкторов инициализации может быть несколько. Они отличаются списком формальных параметров.

Конструктор копии в классе один и вызывается наиболее часто из всех конструкторов. Безымянные копии существующих объектов создаются при передаче объектов в функцию по значению и при возвращении объектов из функции оператором return.

Пример 14.

```
class Pixel{
 int x, y, col;
public:
 Pixel(){
     x=0; y=0;col=0;
 }        //К-р по умолчанию
 Pixel(int _x, int _y){
     x=_x; y=_y;
 } //К-р инициализации
 Pixel(int _x, int _y, int _col){
     x=_x; y=_y; col=_col;
 } //К-р инициализации
```

```
      Pixel(Pixel& P){
          x=P.x; y=P.y; col=P.col;
      } //К-р копии
     };
```

В результате можно писать

```
Pixel A, B(10, 10), C(20, 20, 15), D(B);
```

Задание. Постройте заново классы Pixel, Otrezok. Задайте для них конструкторы. Нарисуйте главную диагональ красным цветом. Для этого в функции Otrezok::draw() нужно установить цвет рисования, сохранив старый. Затем восстановить старый цвет.

Деструктор вызывается всегда, когда объект класса выходит из своей области действия.

Операция присвоения - это функция с именем operator= , она получает единственный аргумент тип_класса&. Присвоение, по умолчанию производит побитовое копирование.

Пример 15.

```
#include <stdio.h>
class Vector{
      int Dim;
      int *dat;
      public:
      Vector(){Dim=0; dat=NULL;}
      Vector(int Dim);
      Vector(Vector& V);
      Vector& operator= (Vector& V);
      ~Vector();
      float& operator[](int j){
          if(j < 0 || j >= Dim){
              printf("выход за диапазон");
              exit(1);
          }
          return dat[j];
      }
};
// К-р инициализации
Vector::Vector(int D){
      Dim = D;
      dat = new int(Dim);
};
Vector::~Vector(){
      delete[] dat;
}

Vector& Vector::operator= (Vector& V){
      if( Dim !=V.Dim){
              printf("Error in dimensions");
              exit(1);
      }
      for(int i=0; i<Dim; i++)
              dat[i] = V.dat[i];
      return *this;
```

```
};
Vector::Vector (Vector& V){
      Dim=V.Dim;
      dat = new int(Dim);
      for(int i=0; i<Dim; i++)
              dat[i] = V.dat[i];
};
void main(){
      Vector A, B(3), C(B);
}
```

Пример 16.

```
#include <stdio.h>
class Matrix{
      public:
      int M, N;
      float **A;
      Matrix(){M=N=0; A=NULL;}
      Matrix(int _m, int _n);
      Matrix(int _m, int _n, float *_A);
      Matrix(Matrix& V);
      Matrix& operator= (Matrix& V);
      ~ Matrix ();
};
// К-р инициализации
Matrix:: Matrix (int _m, int _n){
      M=_m;
      N=_n;
      A = (float **)malloc(M*sizeof(float*));
      // проверка
      for(int  = 0; i< M; i++){
          A[i] = (float *)malloc(N*sizeof(float));
          // проверка
      }
};
Matrix::~Matrix(){
      for(int i=0; i<M; i++)
          free(A[i]);
      free(A);
}

Matrix& Matrix::operator= (Matrix & V){
      if( M != V.M || N != V.N){
              printf("Error in dimensions");
              exit(1);
      }
      for(int i=0; i<M; i++)
          for(int j=0; j<N; j++)
              A[i][j] = V.A[i][j];
      return *this;
};
Matrix::Matrix (Matrix& V){
      M=V.M;
      N=V.N;
```

```
      A = (float **)malloc(M*sizeof(float*));
      // проверка
      for(int  = 0; i< M; i++){
          A[i] = (float *)malloc(N*sizeof(float));
          // проверка
      }
      for(int i=0; i<M; i++)
          for(int j=0; j<N; j++)
              A[i][j] = V.A[i][j];
};
void main(){
      Matrix A, B(3, 5), C(B);
      float r[2][3]={{2, 4, 5}, {1, 2, 3}};
      Matrix D(2, 3, (float *)r);
}
```

Указатель this

Если использовать указатель this в методе объекта класса, то он означает указатель на этот объект. Таким образом, *this означает сам объект, внутри которого находится this. Поэтому в методах класса Vector записи Dim, (*this).Dim и this->Dim эквивалентны.

Вопрос. Как реализовать последовательное присвоение?

Ответ. Надо, чтобы операция присвоения возвращала ссылку на этот же класс. Пример.

```
Vector& Vector::operator= (Vector& V){  ......
 return *this;
};
```

Конструктор явного преобразования типа преобразует данный класс к какому-либо другому типу. Таких конструкторов может быть несколько.

Пример 16.

```
class MyInt{
 int n;
 public:
 int(){return n;}
};
void main(){
 MyInt A(5);
 int m;
 // m=A;  ошибка
 m=(int)A;
}
```

Если конструктор класса A принимает один аргумент типа B, то говорят, что B может быть приведен к типу A.

Друзья класса

Друзьями класса могут быть глобальные функции и другие классы. Друзья имеют доступ ко всем элементам этого класса независимо от спецификаторов доступа. Пример.

```
class MyClass{
```

```
 private: float x;
 friend classA;
 friend Afunc(int );
} TheMyClass;
```

Основное отличие метода класса от дружественной функции состоит в способе передаче фактических параметров. Метод класса в качестве первого неявного параметра получает указатель this на объект класса, для которого он вызывается. Дружественная функция все фактические аргументы получает явным образом. Другими словами, два списка: список формальных аргументов в прототипе и список фактических аргументов при вызове, - совпадают по количеству и типам аргументов.

Перегрузка операций

При перегрузке операций соблюдаются следующие правила:

- сохраняются приоритеты, арность и ассоциативность операций,
- нельзя придумать новую операцию
- перегруженная операция должна быть либо методом класса, либо дружественной функцией класса. Во втором случае она должна получать первый аргумент с типом данного класса.

Пример 17.

```
#include <stdio.h>
#include <conio.h>
class MyInt{
 int n;
 MyInt(){n=0;}
 MyInt(int _n){n=_n;}
 MyInt(MyInt &V){n=V.n;}
 MyInt operator+(MyInt V){   // бинарное сложение
     MyInt buf;
     buf.n=n+V.n;
     return buf;
 }
 friend MyInt operator+(int x, MyInt V); // бинарное сложение
 MyInt operator-(){   // унарный минус
     MyInt buf;
     buf.n=-n;
     return buf;
 }
 int operator!(){   // унарная операция логического отрицания
     return (n==0)?1:0;
 }
 frienf ostream& operator<<( ostream& os, MyInt V);
 frienf istream& operator>>( istream& os, MyInt V);

};
 MyInt operator+(int x, MyInt V)
 {
     MyInt buf;
     buf.n=x+V.n;
     return buf;
```

```
}
ostream& operator<<( ostream& os, MyInt V){
    os<<n;
    return os;
}
istream& operator>>( istream& is, MyInt V) {
    is>>n;
    return is;
}
```

Контрольные вопросы.

ЛАБОРАТОРНЫЕ ЗАДАНИЯ

1. Класс Vector

Объявите класс Vector, являющийся надстройкой над одномерным массивом вещественных чисел. Предотвратите возможность выхода индекса массива за диапазон массива. Перегрузите операции, необходимые для выполнения функции

```
void main(){
  float x[]={3, 5, 2, 6};
  float y[]={1, 1, 1, -1.4};
  Vector A(sizeof(x), x), B(sizeof(y), y), C(sizeof(x));
  C = -A+2*A-A+B*2;
  C[0] = (A,B); //скалярное произведение
  // C[6] = 1; ошибка!!
  cout << C;
}
```

2. Класс Matrix

Объявите класс Matrix, являющийся надстройкой над двумерным динамическим массивом вещественных чисел. Предотвратите возможность выхода первого индекса массива за диапазон массива. Перегрузите операции, необходимые для выполнения функции

```
void main(){
  float x[][4]={{3, 5, 2, 6}, {3, 5, 2, 6}};
  float y[][4]={{1, 1, 1, 1}, {0, 0, 0, 0}};
  Matrix A(2, 4, (float *)x), B(2, 4, (float *)y);
  Matrix C(2, 4);

  C = -A+2*A-A+B*2;
  C[0][0] = 100;
  // C[6][0] = 1; ошибка!!
  cout << C << endl;
  //использование класса вектора
  float z[]={0, 0, 0, 1};
  Vector D(sizeof(z), z), R(2);
  R=A*D;
  Cout << R;
```

```
}
```

ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ

1. Реализуйте класс Matrix таким образом, чтобы он содержал массив векторов. В этом случае можно предотвратить возможность выхода обоих индексов массива за диапазон массива.

```
#include <stdio.h>
class Matrix{
      public:
      int M, N;
      Vector *A;
      // ……
      Vector& operator[](int i); // i – номер строки
};
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бруно Бабэ. Просто и ясно о Borland C++. - М.: Бином, 1995, 395с.
2. Страуструп Б. Язык программирования Си++. - М.: Радио и связь, 1991, 348с.
3. Подбельский В. В., Фомин С. С. Программирование на языке Си. Учеб. Пособие. – 2-е доп. изд. М.: Финансы и статистика, 2004, 600 с.
4. Программирование в Си. Организация ввода–вывода: метод.указания / сост. С.П. Трофимов. Екатеринбург: УГТУ, 1998. 14 с.
5. Программирование в Си. Динамическое распределение памяти: метод.указания / сост. С.П. Трофимов. Екатеринбург: УГТУ, 1998. 13с.