Основы объектно-ориентированного программирования

Часть 3

Вопросы к Части 2

- Как осуществляется контроль доступа к членам класса
  - При описании класса?
  - При наследовании?
- Для чего применяется описание friend?
- В каких случаях необходимо уточнение имени?
- Что обычно делают конструкторы и деструкторы?
- Когда они вызываются?
- Зачем переопределять операторы new и delete?

Конструкторы и деструкторы производных классов

```
class Employee {
 char name[128]; int age;
 public:
  Employee(char* N, int A) { … }
};
class Manager : public Employee  {
 int level;
 public:
  Manager(char* N, int A, int L)
    : Employee(N,A), level(L) {…}
};
```

Иерархия классов

- Производный класс сам в свою очередь может быть базовым классом:

```
class Employee { /* ... */ };
class Manager : public Employee { /* ... */ };
class Director : public Manager { /* ... */ };
```

Множественное наследование в C++

- Обычно иерархия классов представляется деревом, но в C++ бывают иерархии с более общей структурой в виде ориентированного графа без циклов:

```
class Temporary { /* ... */ };
class Secretary : public Employee { /* ... */ };
class TempSec: public Temporary,
                    public Secretary { /* ... */ };
class Consultant: public Temporary,
                    public Manager { /* ... */ };
```

Интерфейс в Java

```
public interface ITree {
public void DeleteChild
        (int VertexID, int ChildIndex);
//…
}
//**********************
public class MyTree: public Vector
implements ITree, IVector {
public void   DeleteChild
        (int VertexID, int ChildIndex) {/*…*/}
//…
```

Интерфейс в Delphi

```
AttributeList = interface(IUnknown)
function  GetStrAttr
            (const AttrName:string):string;
//…
end;
//**********************
TAttributeList =
class(TStringList,IAttributeList)
function  GetStrAttr
            (const AttrName:string):string;
//…
end;
```

ВОПРОС

- Когда оправдано применение множественного наследования или интерфейсов?

Контроль типа и виртуальные методы

- Указатели на базовые классы обычно используются при проектировании контейнерных классов (множество, вектор, список и т.д.)
- 4 способа понять тип объекта:
  - Обеспечить, чтобы указатель мог ссылаться на объекты только одного типа
  - Поместить в базовый класс поле типа, которое смогут проверять функции
  - Использовать виртуальные методы
  - Использовать RTTI

УПРАЖНЕНИЕ: печать списка сотрудников

```
class Employee {
 char name[128]; int age;
//…};
class Manager : public Employee  {
 int level;
//…};
//*****************************************
class EmployeeList {
 Employee **v,**p;
 public: void printAll();
};
```

Абстрактные классы

```
class MyShape {
 public:
  virtual void rotate(int) = 0;
  virtual void draw() = 0;
 };
```

- Класс, в котором есть абстрактные функции, называется абстрактным
- abstract в Java и Delphi
- ВОПРОС: в чем преимущество абстрактных методов по сравнению с «пустой» реализацией?

Операторные функции в C++

```
class complex {
 double re, im;
public:
 complex(double r, double i) { re=r; im=i; }
 friend complex operator+(complex, complex);
 friend complex operator*(complex, complex);
};
//**************************
complex a(2,1), c=a+complex(1.5,0);
//**************************
complex operator+(complex a, complex b) {
  return complex(a.re + b.re, a.im + b.im);
}
```

Список операторов

- Арифметические, логические и др. операторы:
  - + - * / % ++ --
  - ^ & | ~ !
  - = += -= *= /= %= ^= &= |=
  - < > == != <= >= && ||
  - << >> >>= <<=
  - ->* -> () []
  - new delete
- Операторы преобразования типа

Бинарные и унарные операции как члены класса

```
struct X {
 X* operator&();       // префиксная унарная
                       // операция & (взятие адреса)
 X operator&(X);       // бинарная операция &
                       // (И поразрядное)
 X operator+=(int);    // прибавление целого числа
 X operator&(X,X);     // ошибка: слишком много
                        // операндов
X operator/();         // ошибка: слишком мало
                       // операндов
};
```

Бинарные и унарные операции как глобальные функции

- Как правило, объявляются друзьями класса X

```
X operator-(X);    // префиксный унарный минус
X operator-(X,X);          // бинарный минус
X operator+=(X&,int);    // прибавление целого числа
X operator-();                 // ошибка: нет операнда
X operator-(X,X,X);  // ошибка: много операндов
X operator%(X);         // ошибка: мало операндов
```

УПРАЖНЕНИЕ: Отладочная печать

- Реализовать класс DebugInt, выполняющий и распечатывающий арифметические операции с целыми числами
- Если определена функция
  ```
  INT F(INT a, INT b, INT c)
  { return a*b+b*c; }
  ```
  и
  ```
  #define INT DebugInt
  ```
  , то вызов `F(INT(1),INT(2),INT(3))` должен напечатать
  `1*2=2  2*3=6  2+6=8`

Большие объекты как параметры операторов

```
class matrix {
  double m[4][4];
 public:
  matrix();
  friend matrix operator+
            (const matrix&, const matrix&);
  friend matrix& operator*
            (const matrix&, const matrix&);
};
```

ВОПРОС: что должны возвращать операторы: matrix или matrix& ?

Пример реализации класса Vector

```
class Vector
{
 Shape* *data;
 public:
  Vector(int sz) { data=new (Shape*)[sz]; }
  ~Vector() { delete[] data; }
  void put(int index, Shape* x)
   { data[index]=x; }
  Shape* get(int index)
   { return data[index]; }
};
```

Вопрос: Что происходит в этой программе?

```
void main()
{
 Vector v1(10),v2(20);
 v1.put(0,1.); v2.put(0,2.); v1=v2;
 v2.put(0,3.);
 printf("v1.get(0)=%f\n",v1.get(0));
}
```

Присваивание и инициализация

- Для многих типов задача управления ими сводится к построению и уничтожению связанных с ними объектов, но есть типы, для которых этого мало. Иногда необходимо управлять всеми операциями копирования:

```
Vector v1 ( 100 );
Vector v2 = v1;
// построение нового вектора v2,
// инициализируемого v1
v1 = v2;  // v2 присваивается v1
```

Присваивание и инициализация: пример

```
class Vector {
int * v; int sz;
public:
  void operator = ( const Vector & );  // присваивание
 Vector(int);
 Vector ( const Vector & );  // инициализация
};
```

- Присваивание и инициализация являются РАЗНЫМИ операторами
- УПРАЖНЕНИЕ: реализовать operator = и второй конструктор

Индексация

- Операторная функция operator[] () задает для объектов классов интерпретацию индексации
- УПРАЖНЕНИЕ: реализовать operator[] для класса Vector как глобальную функцию.
- Второй параметр этой функций (индекс) может иметь произвольный тип. Это позволяет, например, определять ассоциативные массивы
- Delphi также позволяет определить оператор индексации (об этом позже)

Предостережение

- Как и всякое другое языковое средство, перегрузка операций может использоваться разумно и неразумно.
- В частности, возможностью придавать новый смысл обычным операциям можно воспользоваться так, что программа будет совершенно непостижимой.
- К счастью, нельзя изменить:
  - смысл операций над основными типами данных, такими, как int,
  - синтаксис выражений
  - приоритеты операций

Шаблоны

- Зачем программисту может понадобиться определить такой тип, как вектор целых чисел?
- Как правило, ему нужен вектор из элементов, тип которых неизвестен создателю класса Vector.
- Следовательно, надо суметь определить тип вектора так, чтобы тип элементов в этом определении участвовал как параметр, обозначающий "реальные" типы элементов

Пример шаблона: вектор элементов типа T

```
template < class Elem > class Vector  {
Elem * v;  int sz;
public:
 Vector ( int s ):sz(s),v(new Elem [s]) {}
Elem & operator [] ( int i );
 int size () { return sz; }
};
```

Использование шаблона: пример

```
void f () {
 Vector < int > v1 ( 100 );
  // вектор из 100 целых
 Vector < complex > v2 ( 200 );
 // вектор из 200 комплексных
 v2 [ i ] = complex ( v1 [ x ], v1 [ y ] );
// ...
}
```

Определение методов шаблона внутри и вне описания класса

```
template<class T> class Stack {
 T* v; int top; int sz;
public:
 Stack(int s): top(0),sz(s) { v=new T[sz]; }
 ~Stack() { delete[] v; }
 void push(T);
 T pop() { return v[--top]; }
 int size() const { return sz; }
};
//*****************************
template<class T> void Stack<T>::push(T a)
{ v[top++]=a; }
```

Шаблон для глобальной функции

```
template<class T> void sort(Vector<T>&);
//**********************
Vector<int> vi;
Vector<MyString> vs;
Vector<char*> vc;
sort(vi);       // sort<int>(Vector<int>& v);
sort(vs);       // sort(Vector<MyString>& v);
sort(vc);       // sort(Vector<char*>& v);
```

Контейнерные классы в Delphi

- За некоторыми исключениями хранят ссылки на TObject или Pointer
- Одна из причин того, что все классы VCL выведены из TObject

TBucketList,…	TList	TClassList
TOrderedList		
TObjectList	TQueue	TObjectQueue
TComponentList	TStack	TObjectStack