Язык программирования C++

Лекция 9

Объектно-ориентированный подход:

- шаблоны
- наследование классов

Класс SimpleFloatArray

Напоминание: заголовочный файл SimpleFloatArray.h

```
class SimpleFloatArray {
  public:
        SimpleFloatArray(int n);                  // init to size n
        SimpleFloatArray();                       // init to zero size
        SimpleFloatArray(const SimpleFloatArray&);
        ~SimpleFloatArray();                      // destroy
        float& operator[](int i);                 // subscript
        int numElems();
        SimpleFloatArray& operator=(const SimpleFloatArray&);
        SimpleFloatArray& operator=(float);       // set values
        void setSize(int n);
  private:
        int num_elems;
        float *ptr_to_data;
        void copy(const SimpleFloatArray&);
};
```

Но если не ...Float... ?

А если нам нужен такой же класс, описывающий массив, но из элементов другого типа, например int, т.е. SimpleIntArray ?

А если double ? Или Hep3Vector ?

➡ дублировать весь код, заменяя в тексте float на другой тип ?

➡ или использовать операцию подмены типа переменной из языка C – т.е. применять void* вместо float и возвращать результат через (cast) ?

В языке C++ нам не нужно создавать *N* разных классов для описания *N* разных типов данных – можно использовать механизм *шаблонов* (*template*)

Шаблон класса SimpleArray

Шаблон позволяет использовать *параметризацию* типа данных – т.е. при создании класса или функции можно передать имя типа в качества аргумента

```
template<class T>
class SimpleArray {

public:
  SimpleArray(int n);
  SimpleArray();
  SimpleArray(const SimpleArray<T>&);
  ~SimpleArray();
  T& operator[ ](int i);
  int numElems();
  SimpleArray<T>& operator=(const SimpleArray<T>&);
  SimpleArray<T>& operator=(T);
  void setSize(int n);
private:
  int num_elems;
  T *ptr_to_data;
  void copy(const SimpleArray<T>& a);
};
```

эта строчка означает, что далее следует шаблон класса

<class T> - аргумент шаблона. В последних стандартах языка C++ введено новое ключевое слово typename вместо class в аргументе шаблона, т. е. <typename T>

T – произвольный символ (или несколько), обозначающий некоторый тип данных, либо встроенный (int, float и т.д.), либо определенный самим программистом (не обязательно класс)

Пример использования шаблона

Линейный фит:

```
void linefit( )  {

        int n;
        cin >> n;
        SimpleArray<float> x(n);
        SimpleArray<float> y(n);

        // read the data points
        for (int i = 0; i < n; i++) {
                cin >> x[i] >> y[i];
        }
        // the rest of data processing
        . . .
}
```

SimpleArray<float> – теперь класс

<class T> заменен на float

шаблон класса может быть использован также, как и класс

В шаблоне можно использовать любой тип данных, который уже существует в программе:

```
SimpleArray<Hep3Vector>  z(n);
```

Шаблон функции

```
double square(double x)  {
        return  x * x;
}
```

обычная функция

```
template<class T>
T square (T x)  {
        return  x * x;
}
```

шаблон функции

теперь можем написать:

```
int  i = 1;
float  f = 3.1;
Hep3Vector  v (1, 1, 1);

cout << square(i) << endl;
cout << square(f) << endl;
cout << square(v) << endl;
```

Наследование

Вспомним 3-х вектор

```
class Hep3Vector  {
public:
        Hep3Vector ( );
        Hep3Vector (double x, double y, double z);
        double x ( );
        doubly y ( );
        double z ( );
        double Phi ( );
        double cosTheta ( );
        double mag ( );
private:
        double dx, dy, dz;
};
```

А если нам нужен 4-х вектор – как может выглядеть такой класс ?

```
class Hep4Vector  {
public:
        Hep4Vector ( );
        Hep4Vector (double x, double y, double z,
double t);
        double x ( );
        doubly y ( );
        double z ( );
        doublt  t ( );
        double Phi ( );
        double cosTheta ( );
        double mag ( );
private:
        double dx, dy, dz, dt;
};
```

- некоторые методы – те же самые
- некоторые – добавлены
- некоторые могут быть заново реализованы с тем же именем
- то же самое с элементами данных

Наследование

можно записать так:

```
class Hep4Vector  {
public:
        Hep4Vector ( );
        Hep4Vector (double x, double y, double z,
double t);
        double x ( );
        doubly y ( );
        double z ( );
        double t ( );
        double Phi ( );
        double cosTheta ( );
        double mag ( );
private:
        Hep3Vector vec3;
        double dt;
};
```

конструкторы:

```
Hep4Vector::Hep4Vector ( )  :
  vec3(), dt(0.0) { }

Hep4Vector::Hep4Vector (double x, double y,
double z, double t) :
  vec3(x, y, z), dt (t)  { }
```

некоторые функции:

```
double Hep4Vector::mag ( ) {
  return sqrt ( dt*dt – vec3.mag() );
}
double Hep4Vector::x () {
  return vec3.x ( );
}
```

Наследование

наиболее корректно будет создать класс-наследник Hep4Vector от базового класса Hep3Vector

```
class Hep4Vector  : public Hep3Vector {
public:
        Hep4Vector ( );
        Hep4Vector (double x =0, double y =0,
double z =0, double t =0);
        double t ( );
        double mag ( );
private:
        double dt;
};
```

- все публичные члены Hep3Vector будут публичными в Hep4Vector
- добавлена функция t( )
- функция mag( ) будет перегружена, т.е. заменена на другую по сравнению с классом Hep3Vector с тем же именем
- новый элемент данных dt

Наследование

пример использования:

```
int main ( ) {
   double x, y, z, t;
   while ( cin >> x >> y >> z >> t ) {
        Hep3Vector a3Vec(x, y, z);
        Hep4Vector a4Vec(x, y, z, t);

        cout << a3Vec.x() << " " << a3Vec.mag() << endl;
        cout << a4Vec.x() << " " << a4Vec.mag() << endl;
   }
   return 0;
}
```

- реализация функции a4Vec.x ( ) отсутствует, но она наследуется из базового класса Hep3Vec
- a4Vec.mag ( ) отличается от a3Vec.mag ( )

Наследование

конструктор:

```
Hep4Vector::Hep4Vector (double x, double y, double z, double t) :
   Hep3Vector(x, y, z), dt (t)  { }
```

реализация новой функции Hep4Vector::mag ( )

```
double Hep4Vector::mag ( ) {
   return dt*dt – (dx*dx + dy*dy + dz*dz);
```

НЕПРАВИЛЬНО !

dx, dy и dz – приватные члены класса Hep3Vector

```
double Hep4Vector::mag ( ) {
   return dt*dt – Hep3Vector::mag ( );
```

ПРАВИЛЬНО !

для использования скрытых членов в классах-наследниках идентификатор private в базовом классе можно поменять на protected

```
class Hep3Vector {
public:
// same as before
protected:
   double dx, dy, dz;
};
```

protected разрешает доступ к элементам данных и методам как своему классу, так и всем классам-наследникам

Практическое задание (№ С9)

Повторить практическое задание из лекции 8, используя механизм шаблонных классов :

- разработать дизайн и написать реализацию методов класса template<class T> class SimpleArray
  - int numElems() - вывод количества элементов в данном объекте класса
  - void setSize(int n) - изменение текущего размера объекта (если размер уменьшается по сравнению с первоначальным, то лишние элементы отбрасываются, а оставшиеся сохраняют свои значения. Если размер увеличивается, то добавляемые элементы зануляются)
  - еще один конструктор, который создает объект класса заданного размера и присваивает всем элементам заданное значение

Проверить сделанную реализацию, написав программу, которая будет использовать класс SimpleArray, сначала создав объект определенного размера, а потом изменив его размер в сначала в большую, а потом в меньшую стороны. Повторить выполненную проверку, используя 2 других типа данных для этого шаблонного класса.