Семинар 12
Шаблоны классов

Шаблоны классов

Шаблоны классов называют параметризированными типами. Шаблоны классов позволяют конструировать семейства классов.

Формат определения шаблона классов:

template <список_параметров_шаблона>
спецификация шаблонного класса

Шаблоны классов

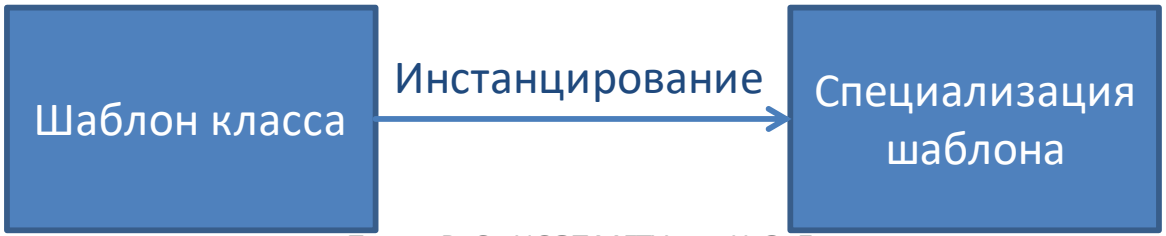
Как и параметры шаблона функций, параметры шаблона классов и соответствующие им аргументы могут быть трёх видов:

- типизирующие (вводятся служебными словами class или template)
- нетипизирующие
- параметры-шаблоны

Шаблоны классов

Специализация шаблона – определение конкретного класса. Создаётся исходя из обращений к шаблонному классу в тексте программы.

Инстанцирование шаблона – процесс генерации специализации шаблона из определения шаблона.



Шаблоны классов

Пример:

```
template<typename A>
class myPair{ //myPair – имя семейства классов
        A x, y;
public:
        //Конструктор:
        myPair(A xx=A(0), A yy=A(1)) :x(xx), y(yy) {}
        //Метод, возвращающий частное:
        A getDiv(void) {return A(x/y);}
        //Перегрузка оператора сложения:
        myPair <A> operator+ (myPair <A> &p)
        {return myPair<A>(x+p.x, y+p.y);}
        //Метод для вывода полей класса:
        void display()
        {cout << "x=" << x << ", y=" << y << endl;}
};
```

Шаблоны классов

В класс myPair<A>, как и в обычный класс, включаются компилятором:

- деструктор

```
~myPair(); // myPair – имя шаблонного класса
```

- конструктор копирования

```
myPair(const myPair<A> &);
```

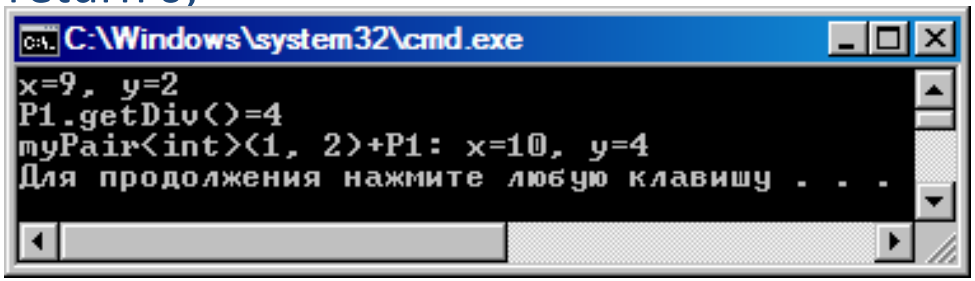
- операцию-функцию присваивания

```
myPair <A> & operator= (const myPair <A> &);
// myPair<A> – имя типа
```

Шаблоны классов

Пример (продолжение):

```
int main()
{
    //Создание экземпляра класса:
    // имя_класса<аргументы_шаблона> имя_объекта(аргументы_констр.);
    myPair <int> P1(9, 2);
    P1.display(); // или P1.myPair<int>::display();
    cout << "P1.getDiv()=" << P1.getDiv() << endl;
    cout << "myPair<int>(1, 2)+P1: ";
    (myPair<int>(1, 2)+P1).display();
    return 0;
}
```



Внешнее определение методов шаблонных классов

Методы шаблонного класса при внешнем определении вводятся как шаблоны функций:

template<список_параметров_шаб._классов>
тип_возвращаемого_значения
имя_класса<список_имён_параметров_шаб.>
::имя_шаблонной_функции
(спецификация_параметров_функции)
тело_шаблонной_функции

Внешнее определение методов шаблонных классов

Пример:

```
template<typename A>
class myPair{
        A x, y;
public:
        //Конструктор:
        myPair(A xx, A yy);
        //Метод, возвращающий частное:
        A getDiv(void);
        //Перегрузка оператора сложения:
        myPair <A> operator+ (myPair <A> &p)
        {return myPair<A>(x+p.x, y+p.y);}
        //Метод для вывода полей класса:
        void display()
        {cout << "x=" << x << ", y=" << y << endl;}
};
template<typename A>
myPair<A>::myPair(A xx=A(0), A yy=A(1)) :x(xx), y(yy) {}
template<typename A>
A myPair<A>::getDiv(void) {return A(x/y);}
```

Дружественные функции шаблонных классов

Как определения, так и описания дружественных функций (даже внутри определения класса) шаблонных классов должны начинаться с template<typename X>.

Вне шаблона классов дружественные функции определяются как обычные шаблоны функций.

Дружественные функции шаблонных классов

Пример.

```
template<typename A>
class myPair{
        A x, y;
public:
        //Конструктор:
        myPair(A xx, A yy);
        //Дружественная функция:
        template<typename AA> // Обратите внимание, что и здесь нужно писать template
        friend ostream & operator<<(ostream &, myPair<AA> &);
};
template<typename A>
myPair<A>::myPair(A xx=A(0), A yy=A(1)) :x(xx), y(yy) {}

template<typename A> // Обратите внимание, что параметры – A и AA не совпадают!
ostream & operator<<(ostream & out, myPair<A> & mp){
        out << "x=" << mp.x << ", y=" << mp.y;
        return out;
}
```

Дружественные функции шаблонных классов

Важное замечание: некоторые компиляторы отказываются компилировать код шаблонных классов, размещённых в cpp-файлах, поэтому старайтесь размещать шаблонные классы в header-файлы.

О пользовательских специализациях

Возможна пользовательская специализация шаблона класса для случаев, когда конкретные типы данных требуют особого подхода при их обработке, либо существуют более подходящие алгоритмы для конкретных типов.

О пользовательских специализациях

Пользовательская специализация может быть:

- явная (полная) или
- частичная

Синтаксис при явной (полной) специализации:

```
template<>
спецификация_параметризованного_класса
```

О пользовательских специализациях

Пример полной пользовательской специализации:

```
template<>
class myPair<char>{ //myPair – имя семейства классов
        char x, y;
public:
        //Конструктор:
        myPair(char xx='a', char yy='b') :x(xx), y(yy) {}
        //Пример метода, тело которого изменено:
        void getDiv(void) {cout<<"Not available";}
        //Перегрузка оператора сложения:
        myPair <char> operator+ (myPair <char> &p)
        {return myPair<char>(x+p.x, y+p.y);}
        //Метод для вывода полей класса:
        void display()
        {cout << "x=" << x << ", y=" << y << endl;}
};
```

О частичных пользовательских специализациях

Пусть определён такой шаблон семейства классов:

```
template<typename T1, typename T2>
class myClass {…}
```

Специализация, в которой оба параметра имеют один тип:

```
template<typename T>
class myClass<T, T> {…}
```

О частичных пользовательских специализациях

Специализация, в которой второй параметр шаблонного класса нетипизирующий:

```
template<typename T>
class myClass<T, int> {…}
```

Специализация, в которой оба параметра указатели:

```
template<typename T1, typename T2>
class myClass<T1 *, T2 *> {…}
```

О частичных пользовательских специализациях

Пример определений объектов:

myClass <int, double> obj1; //myClass<T1, T2>
myClass <double, double> obj2; //myClass<T, T>
myClass <double, int> obj3; //myClass<T, int>
myClass <int *, double *> obj4; //myClass <T1*, T2 *>

О частичных пользовательских специализациях

Неоднозначность указания шаблона:

1. Возможные специализации: myClass <T, T> и myClass <T, int>:

```
myClass <int, int > obj5;
```

2. Возможные специализации: myClass <T1 *, T2 *> и myClass <T, T>:

```
myClass <int*, int*>;
```