Нижегородский государственный университет
им. Н. И. Лобачевского

Методы ООП

Тема 17. Введение в шаблоны

Вопросы:

1. Шаблоны функций
2. Шаблоны классов
3. Специализация классов с шаблоном
4. Пример: класс **`Container`** с шаблоном

Шаблоны функций (функции-шаблоны)

Перегрузка: специальный полиморфизм

```
int mymin(int a, int b)                                          {
      if ( a < b ) return a; else return b;
}
double mymin(double a, double b)                                 {
      if ( a < b ) return a; else return b;
}
Cmystring mymin(const Cmystring& a, const Cmystring& b) {
      if ( a < b ) return a; else return b;
}
//-------------------------------------------------
int _tmain(int argc, _TCHAR* argv[])
{
      int i1 = 4, i2 = 7, i3;
      i3 = mymin(i1, i2);

      double d1 = 4.67, d2 = 1.89, d3;
      d3 = mymin(d1, d2);

      Cmystring s1("Волк"), s2("Воля"), s3;
      s3 = mymin(s1, s2);
      return 0;
}
```

Шаблоны: параметрический полиморфизм

```
template <class Type>
Type mymin(const Type& a, const Type& b)
{
      if ( a < b ) return a; else return b;
}
//-------------------------------------------------
int _tmain(int argc, _TCHAR* argv[])
{
      int i1 = 4, i2 = 7, i3;
      i3 = mymin(i1, i2);

      double d1 = 4.67, d2 = 1.89, d3;
      d3 = mymin<double>(d1, d2); // можно явно указать тип

      Cmystring s1("Волк"), s2("Воля"), s3;
      s3 = mymin(s1, s2);

      return 0;
}
```

Шаблоны функций

- Синтаксис объявления функции-шаблона:

```
template <class    <тип>> <имя_функ>(<параметры>) {<тело_функ>}
template <typename <тип>> <имя_функ>(<параметры>) {<тело_функ>}
```

- Конкретизация (инстанцирование instantiation) – порождение компилятором версии функции для каждого ее вызова с различными типами:
  - компилируется не по описанию функции-шаблона, а по типу обнаруженного компилятором вызова
- Шаблон может иметь несколько параметров:

```
template <class T1, typename T2> T1 min(T1, T2);
```

Шаблоны функций

- При вызове функции с шаблоном можно явно указывать типы аргументов:

```
i1 = min<int, double>(1, 2.0);
```

- Если в качестве параметра шаблона передается класс, то в нем должны быть:
  - конструктор копирования
  - перегружены операции, применяющиеся в функции
- Объявление шаблона перекрывает глобально объявленный тип:

```
deftype double Type;
template <class Type>
Type mymin(const Type& a, const Type& b)   {
  if ( a < b ) return a; else return b;
}
```

Перегрузка функций-шаблонов

- Шаблоны функций могут быть перегружены как при помощи обычных функций, так и при помощи шаблонов:

```
template <class T>
T   mymin(const T& a, const T& b)
{
  if ( a < b ) return a; else return b;
}
template <class T1, class T2>
T1  mymin(const T1& a, const T2& b)
{
  if ( a < b ) return a; else return b;
}
int mymin(const int& a, const int& b)
{
  if ( a < b ) return a; else return b;
}
```

при прочих равных, предпочтение отдается обычным функциям

Простые параметры шаблонов

- Шаблон может иметь в качестве параметра не только тип данных, но и обычные параметры:

```
template <class T, int size>
T mymin(const T (&a)[size])
{
  T rez = a[0];
  for (int i = 1; i < size; i++ )
    if ( rez > a[i] ) rez = a[i];
  return rez;
}
int main()
{
  int a[] = {12, 8, 2, 18, 5, 9};
  int a1[] = {12, 8, 2, 18, 5, 9, 1};
  double m[] = {12., 8., 2., 18., 5., 9.};
  int n;
  int d = mymin<int, 6>(a);
  int e = mymin(a);
  int e1= mymin(a1);
  double dd= mymin(m);
}
```

Специализация шаблонов

- Шаблон может быть специализирован:

```
template <class T> T max( T p1, T p2)
{
  return p1 > p2 ? p1 : p2;
}
typedef const char* CCP;
template <> CCP max<CCP>( CCP s1, CCP s2)  {
  return strcmp(s1, s2) > 0 ? s1 : s2;
}
int main()
{
  int c = max(3, 7);
  char* s = max("Вова", "Валя");  // ошибка!!!
}
```

Пример: SortAny

```
template<class T> void SortAny(T* a, int size)
{
    T tmp;
    int i, j;
    for ( i = 0; i < size - 1; i++ )
        for ( j = i + 1; j < size; j++ )
            if ( a[i] > a[j] )
            {
                tmp  = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
    }
```

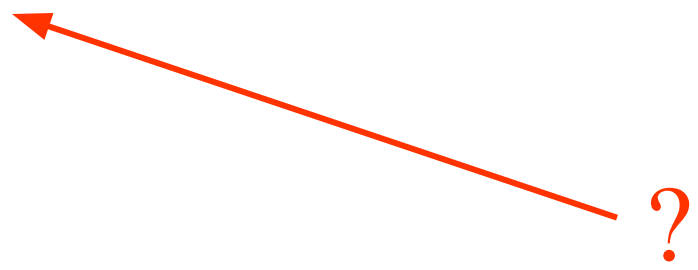
Пример: SortAny

```
int main()
{
   int ma[] = { 8, 2, 7, 4 };
   SortAny(ma, 4);

   double d[5];
   for ( int i = 0; i < 5; i++ )
      d[i] = (5-i)*0.1;
   SortAny(d, 5);

   Complex c[6];
   for ( int i = 0; i < 6; i++ )
      c[i] = Complex(i * 0.1, i * 0.2);
   SortAny(c, 6);

   return 0;
}
```



В **Complex** должны быть перегружены операции

= >

Создание и освобожд. двумерн. массива

```
// Создание двумерного динамического массива ----------
template <class T> T** newA2T(int n, int m)
{
      T **ms = new T*[n];
      for ( int i = 0; i < n; i++ )
            ms[i] = new T[m];
      return ms;
}
// Освобождение двумерного динамического массива -------
template <class T> void delA2T(T **ms, int n)
{
      for ( int i = 0; i < n; i++ )
            delete[] ms[i];
      delete[] ms;
}

int main()
{
      int n1, n2;
      cout << "n1 = "; cin >> n1;
      cout << "n2 = "; cin >> n2;
      double **md = newA2T<double>(n1, n2);  // Создание массива
      int    **mi = newA2T<int>   (n1, n2);  // Создание массива
      . . . . . .
      delA2T(md, n1); // Освобождение массива
      delA2T(mi, n1); // Освобождение массива
      return 0;
}
```

Модели компиляции шаблонов

- Шаблон компилируется при конкретизации (вызове) – описание шаблона должно быть видимо при компиляции вызова
- Две модели компиляции:
  - Компиляция с включением
    - ✔ описание шаблонов включено в заголовочные файлы
  - Компиляция с разделением (в C++ VisualStudio не работает):
    - ✔ объявление шаблона в заголовочном файле:

```
template <class T> min(T a, T b);
```

    - ✔ описание шаблона в исполняемом файле:

```
export template <class T> min(T a, T b) { . . . }
```

- Явное объявление конкретизации:

```
template int min<int>(int a, int b);
```

  - ✔ выполняется только один раз и блокирует дублирование конкретизаций в разных файлах программы

Шаблоны классов (классы-шаблоны)

Класс с шаблоном:

- Объявление класса с шаблоном:

```
template <class T, …> class A { … };
```

- Описание методов класса с шаблоном:
  - включается в заголовочный файл
  - описание каждого метода:

```
template <class T, …> тип A<T,…>:: имя{…}
```

  - все упоминания имени класса (кроме имен конструкторов и деструктора):

```
A<T,…>      A<T,…>*      A<T,…>&
```

- Объявление объектов класса с шаблоном:

```
A<type,…> a;      A a;
```

Пример: объявление класса с шаблоном

```
template <class T>
class A
{
  protected:
        T data;  // данные типа T
  public:
     A();
     A(T _data);
     A(const A& _a);
     ~A();
     A f(A& _a);
     A& operator=(const A& _a);
     . . . . . .
};
```

Пример: описание методов класса с шаблоном

```
//---------------------------------------
template <class T>
A<T>::A(T _data): data(_data) { }
//---------------------------------------
template <class T>
A<T>::A(const A<T>& _a) { data = _a.data; }
//---------------------------------------
template <class T>
A<T> A<T>::operator+(const A<T>& _a)
{
      A<T> tmp;
      tmp.data = data + _a.data;
      return tmp;
}
//---------------------------------------
template <class T> A<T> A<T>::f(A<T>* _a)
{
      //.....................
}
```

Пример: объявление объектов класса с шаблоном

```
A<int> ai1, ai2(5);
A<double> ad1, ad2(7.);
A<Cmystring> ac1, ac2("Вова");
```

Пример: класс вектор - double

```
class Vector
{
private:
  double* data;
  int n;
public:
  Vector(int n_, double* data_);
  explicit Vector(int n_, double filler = 0.0);
  Vector(const Vector& v);
  ~Vector();
  Vector& operator=(const Vector& v);
  int Length() const;
  double& operator[](int index);
  double  operator[](int index) const;
  Vector& operator+=(const Vector& v);
  Vector& operator-=(const Vector& v);
  Vector& operator*=(double alpha);
  double  operator* (const Vector& v) const;
};
```

Пример: класс вектор – с шаблоном

```
template <class T>
class Vector
{
private:
  T* data;
  int n;
public:
            Vector(int n_, T* data_       );
  explicit  Vector(int n_, T filler = T());
            Vector(const Vector& v        );
  ~Vector();
  int Length() const;
  Vector&  operator= (const Vector& v);
        T& operator[](int index         );
  const T& operator[](int index         ) const;
  Vector&  operator+=(const Vector& v);
  Vector&  operator-=(const Vector& v);
  Vector&  operator*=(double alpha      );
  T        operator* (const Vector& v) const;
};
#include "Vector.cc"
```

Пример: класс вектор – с шаблоном

```
// ------------------------------------------------------
template <class T> Vector<T>::Vector(const Vector<T>& v)
{
    n = v.n;  data = new T[n];
    for ( int i = 0; i < n; i++) data[i] = v.data[i];
}
// ------------------------------------------------------
template <class T> Vector<T>::~Vector()
{
    delete[] data;
}
// ------------------------------------------------------
template <class T>
Vector<T>& Vector<T>::operator=(const Vector<T>& v)
{
    if(&v != this)      {
        delete[] data;
        n = v.n;  data = new T[n];
        for ( int i = 0; i < n; i++) data[i] = v.data[i];
    }
    return *this;
}
```

Пример: класс вектор – с шаблоном

```
int main()
{
    // Тестирование класса вектор
    Vector<double> v1(10, 2.0), v2(10, 3.0);
    v1 += v2;
    cout << v1 << endl;
    Vector<double> v3(10, 5.0);
    cout << v1 - v3 << endl;

    Vector<int> v4(10, 2), v5(10, 3);
    v4 += v5;
    cout << v4 << endl;
    Vector<int> v6(10, 5);
    cout << v4 - v6 << endl;

    return 0;
}
```

Проект примера

Специализация класса с шаблоном

Класс A с шаблоном

Файл **ATemp.h**

```
#ifndef HATEMP
#define HATEMP
template <class T> class A
{
      T a;
public:
      A(T _a): a(_a) {}
      T amin(const A<T>& _a);
      T amax(const A<T>& _a);
};
template <class T> T A<T>::amin(const A<T>& _a)
{
      if ( a < _a.a ) return a; else return _a.a;
}
template <class T> T A<T>::amax(const A<T>& _a)
{
      if ( a > _a.a ) return a; else return _a.a;
}
#endif
```

Использование класса A с шаблоном

Файл **mainfile.cpp**

```
#include "ATemp.h"

int main()
{
	A<double> d1(1.2), d2(3.7);
	double dd1 = d1.amax(d2);
	double dd2 = d1.amin(d2);

	A<int> i1(1), i2(3);
	int ii1 = i1.amax(i2);
	int ii2 = i1.amin(i2);

	A<char> c1(1), c2(3);
	char cc1 = c1.amax(c2);
	char cc2 = c1.amin(c2);

	return 0;
}
```

Специализация метода класса A с шаблоном

Файл **AIntSp.h**

```
#ifndef HAINTSP
#define HAINTSP
#include "ATemp.h"
// явная специализация метода amin класса A<T>
template <> int A<int>::amin(const A<int>& _a);
#endif
```

Файл **AIntSp.cpp**

```
#include "AIntSp.h"
// явная специализация метода amin класса A<T> --
int A<int>::amin(const A<int>& _a)
{
      if ( a < _a.a ) return a; else return _a.a;
}
```

Использ. класса A с шаблоном и специализацией метода amin

Файл **mainfile.cpp**

```
#include "ATemp.h"
#include "AIntSp.h" // Специализация метода для int

int main()
{
      A<double> d1(1.2), d2(3.7);
      double dd1 = d1.amax(d2);
      double dd2 = d1.amin(d2);

      A<int> i1(1), i2(3);
      int ii1 = i1.amax(i2);
      int ii2 = i1.amin(i2);

      A<char> c1(1), c2(3);
      char cc1 = c1.amax(c2);
      char cc2 = c1.amin(c2);

      return 0;
}
```

Специализация класса A с шаблоном

Файл **AChar.h**

```
#ifndef HACHAR
#define HACHAR
#include "ATemp.h"
template <> class A<char>
{
      char a;
public:
      A(char _a): a(_a) {}
      char amin(const A<char>& _a);
      char amax(const A<char>& _a);
};
#endif
```

Файл **AChar.cpp**

```
#include "AChar.h"
char A<char>::amin(const A<char>& _a)        {
      if ( a < _a.a ) return a; else return _a.a;
}
char A<char>::amax(const A<char>& _a)        {
      if ( a > _a.a ) return a; else return _a.a;
}
```

Использ. класса A с шаблоном, специализацией метода amin и специализацией класса

Файл **mainfile.cpp**

```
#include "ATemp.h"
#include "AIntSp.h" // Специализация метода для int
#include "AChar.h"  // Специализация для char

int main()
{
      A<double> d1(1.2), d2(3.7);
      double dd1 = d1.amax(d2);
      double dd2 = d1.amin(d2);

      A<int> i1(1), i2(3);
      int ii1 = i1.amax(i2);
      int ii2 = i1.amin(i2);

      A<char> c1(1), c2(3);
      char cc1 = c1.amax(c2);
      char cc2 = c1.amin(c2);

      return 0;
}
```

Проект примера

Использование шаблонов. Заключение

- Для шаблонов классов действуют те же соображения по поводу конкретизации (инстанцирования) и специализации
- Можно организовывать **иерархии наследования** как для обычных классов
- Использование шаблонов:
  - представляет собой синтаксический механизм размножения кода
  - приводит к увеличению размера исполняемого приложения и времени выполнения

Класс `Container` с шаблоном

ПроектПроект

Листинг