Шаблоны

2013

- Шаблоны (*template*) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию)
- В C++ возможно создание шаблонов функций и классов

- Шаблоны позволяют создавать параметризованные классы и функции
- Параметром может быть любой тип или значение одного из допустимых типов

Шаблоны функций

- Перегруженные функции обычно используются для выполнения похожих по синтаксису, но разных по семантике операций.
- Если же для каждого типа данных должны выполняться идентичные операции, то более компактным и удобным решением является использование шаблонов функций

Шаблоны функций

- *Шаблоны* дают возможность определять при помощи одного фрагмента кода целый набор взаимосвязанных функций, называемых *шаблонными функциями*.
- Шаблонная функция – это «реализация функции по шаблону».

Синтаксис

- Шаблон функции начинается с ключевого слова template, за которым в угловых скобках следует список параметров. Затем следует объявление функции

Пример

```
template <typename T>
void swap ( T& a, T&b)
{
   T r = a;
    a=b;
    b=r;
}
```

Допустимо использовать

template< class T >

T – параметр типа

Использование шаблона функции

- При вызове

```
int x, y;
double w, t;
Complex c1(2,3), c2(2.5,0);
swap ( x, y );  // шаблонная функция для int
swap ( w, t ); // шаблонная функция для double
swap ( c1, c2 ); //для Complex
```

- Процесс конструирования шаблонной функции называется *созданием экземпляра* шаблона
- После создания экземпляра шаблонная функция компилируется
- Если создание экземпляра выполняется по типу фактических параметров, оно называется неявным

- Если по типу фактических параметров невозможно определить тип для шаблона, необходимо задать его при вызове

swap <double> (x, t);

// шаблонная функция для double

- При многофайловой организации проекта может создаваться слишком много одинаковых шаблонных функций для одинаковых списков параметров

Явное создание экземпляров

template void swap <int> (<int>& , <int>&);

- Процесс создания шаблонных функций (экземпляров) компилятором называется конкретизацией

Специализация

- Иногда общее определение, предоставляемое шаблоном, для некоторых типов вообще не работает, а для некоторых работает неэффективно
- В этом случае необходимо предоставить специализированное определение для конкретизации шаблона (*специализацию*) или использовать перегруженную функцию с нужным списком параметров.

```
template <> void swap (char* s1, char * s2)
{
 // специализация для строк в стиле C
}
```

Компиляция

- Шаблон не определяет никакой функции, поэтому не подвергается компиляции
- В многофайловых проектах шаблоны включаются в заголовочные файлы
- Явные конкретизации и специализация шаблона определяют конкретную функцию, поэтому должны включаться в компилируемый файл

- Возникает множество перегруженных имен
  - имена функций
  - имена явных специализаций шаблонов
  - имена явных экземпляров шаблонных функций (конкретизаций)
  - имена неявных конкретизаций шаблонных функций
  - имена функций с учетом автоматического преобразования типов

Шаблоны классов

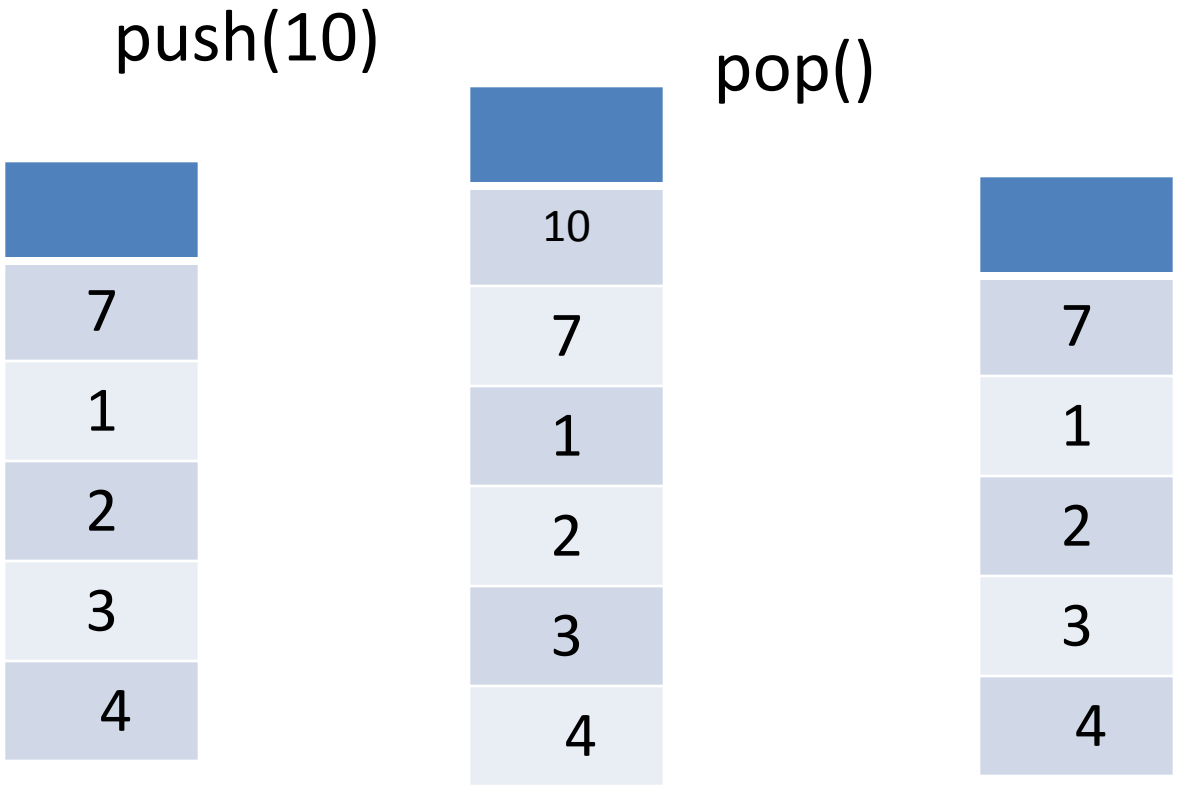
- *Шаблоны классов*, так же как и шаблоны функций, являются трафаретами, по которым компилятор создает *шаблонные классы* и шаблонные методы
- Шаблоны классов часто называют *параметризованными типами*

- Шаблоны классов удобно использовать для создания типов коллекций без определения типа данных, хранящихся в коллекции
- Например, «список элементов одного типа», вместо «список целых»

Стек

- Стек – абстрактная структура данных

  Принцип LIFO
- Позиция совершения операций – вершина стека
- Операции
  - поместить элемент в стек (push)
  - удалить элемент из стека (pop)
  - посмотреть элемент на вершине (top)
  - проверить стек на пустоту (isEmpty)


push(10)
7
1
2
3
4
10
7
1
2
3
4
pop()
7
1
2
3
4

- Создадим шаблон класса Stack, используя для хранения элементов массив
- Чтобы иметь возможность настраивать размер стека для разных задач, будем создавать массив в динамической памяти
- Так как все равно размер стека будет ограничен, дополним еще одной операцией – «стек полон» isFull.

- При неправильной работе со стеком возможны ошибки при выполнении операций push, pop и top.
- Будем генерировать в случае ошибки исключение.

Шаблон класса

```
template <typename T>
class Stack {
private:
  T * start;  //указатель на стек
  int t;      //положение вершины стека
  int size;   //размер стека
```

```
public:
  class StackERR
  {  int code;
       public :
        StackERR ( int n=0):  code(n) {}
        int getCode()
       {
          return code;
        }
   };
```

```
//конструктор с параметром– размер стека
  Stack (int n= 10);
  ~Stack() { delete [] start; }  //деструктор
// операции
void  push (const T&);          //в стек
T pop  ();                      //из стека
T top   () const;               //вершина стека
```

- проверки

```
bool isEmpty() const {return t==-1;}
  //true, если стек пустой
  bool isFull() const {return t==size-1;}
  //true, если стек полон
};
```

Реализация

- помещается как и описание в заголовочном файле!!!

```
//конструктор
template < typename T>
Stack<T> ::  Stack(int n)
{
  //«разумный» размер стека
  size = n>0 && n<1000 ? n : 10;
  start=new T [ size ];
  t=-1;
}
```

```
//помещение объекта в стек
template < typename T>
void Stack<T>::push (const T& x){
  if (!isFull()) {
    start[++t]=x;  //сдвигаем вершину
   return;
  }
  throw StackERR(1); //стек полон
}
```

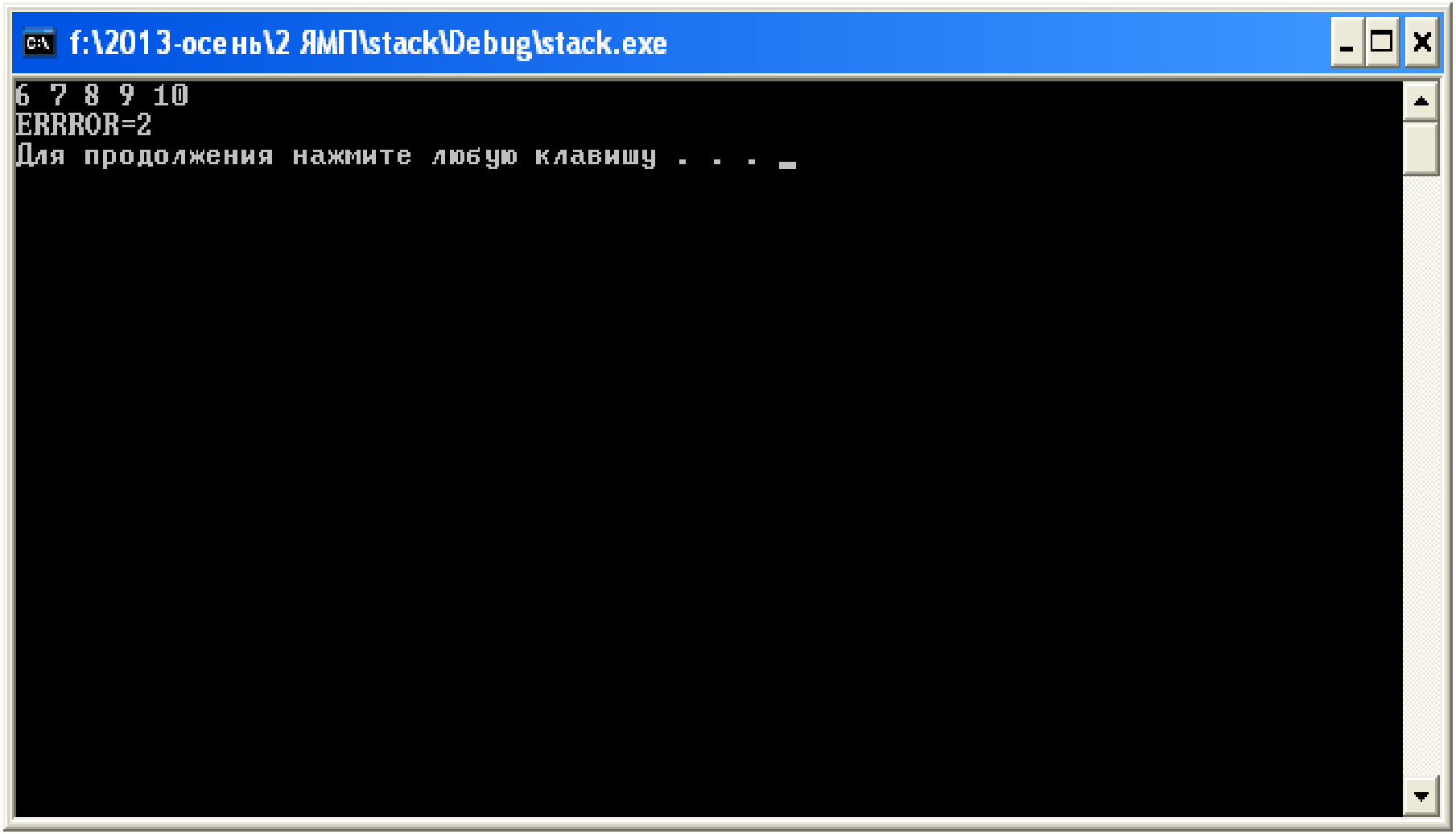
```
//выталкивание элемента из стека
template < typename T>
T Stack<T>::pop () {
  if (!isEmpty()) {
    return start[t--];
 //взяли элемент, опустили вершину
   }
  throw StackERR(2); //в стеке нет элементов
}
```

```
//просмотр элемента из вершины стека
template < typename T>
T Stack<T>::top ()const{
  if (!isEmpty()) {
    return start[t];
  }
  throw StackERR(2);
}
```

Проверка работы

```
#include "header1.h"
#include <iostream>
using namespace std;
```

```
int main()
{
  Stack<int> intStack(5);
try{
   for (int i=0; i<5; i++)   intStack.push(10-i);
   while (!intStack.isEmpty())   cout<<intStack.pop()<<'  ' ;
     cout<<endl;
    cout<<intStack.top(); // для проверки исключения
  }
  catch(Stack<int>::StackERR e)  {
      cout<<"ERRROR="<<e.getCode()<<endl;
  }
    return 0;
}
```


f:\2013-осень\2 ЯМП\stack\Debug\stack.exe
6 7 8 9 10
ERRROR=2
Для продолжения нажмите любую клавишу . . .