Тема: Обобщенное программирование. Шаблоны

- Понятие обобщенного программирования
- Шаблоны функций
- Контейнерные классы. Класс «стек», как пример контейнерного класса
- Шаблоны описания класса и шаблоны методов
- Параметры шаблонов

Понятие обобщенного программирования

- Обобщенным программированием называется создание универсального программного кода, инвариантного к типу обрабатываемых данных.
  Например:
  - функция выбора максимального или минимального из двух значений одинаково реализуются для разных числовых типов данных (int, float, double)
  - организация данных в виде *стека* или *множества* одинакова для элементов разных типов и т.д.
- Английское название ***generic programming*** - “родовое программирование”
- Непосредственно не связано с ООП, но замечательным образом сочетается
- Языковые средства обобщенного программирования еще не «устоялись»

Обобщенное программирование в С

- *Макросы с параметрами* - возможность простейшего обобщенного программирования функций

  Пример: функция выбора максимального из двух значений

  #define max(x,y) ((x)>(y) ? (x) : (y))

- Данный макрос пригоден для аргументов всех числовых типов (int, float, double)

Функция сортировки массива qsort()

void *qsort*(**void*** *base*, **size_t** *nelem*, **size_t** *width*,
int(**fcmp*)(**const void***, **const void***));

- Выполняет сортировку массива данных любого типа
- *base* - указатель на сортируемый массив (или подмассив)
- *nelem* - количество элементов, подлежащих сортировке (например, весь массив)
- *width* - длина данных для элемента массива в байтах
- *fcmp* - указатель на функцию сравнения элементов

Шаблоны классов. Параметризованные классы

- ***Контейнерные классы*** — классы, объединяющие атомарные объекты - элементы некоторого заданного типа. Примеры:
  - *Список*
  - *Стек*
  - *Множество*
  - и т.п

Стек – описание класса

файл stack.h

```
class stack_char {
  public:
    stack_char(unsigned);
   ~stack_char();
     void push(char);
     char pop();
     unsiged size() const;
  private:
     char* base;
     char* top;
     unsigned max_size;
};
```

```
class stack_int {
  public:
    stack_int(unsiged);
   ~stack_int();
    void push(int);
    int pop();
    unsigned size() const;
  private:
     int* base;
     int* top;
     unsigned max_size;
};
```

Стек – реализация методов

файл stack.cpp

```
#include “stack.h”
#include <iostream.h>
#include <stdlib.h>
stack_char::stack_char(unsigned sz) {
      if ((top=base=new char[max_size=sz])==NULL) {
       cerr << “Нет свободной памяти для стека.\n”;
              exit(1);
       }
}

stack_char::~stack_char() {
     delete [ ]base;
}
```

Стек – реализация методов (продолжение)

файл stack.cpp

```
void stack_char::push(char a) {
        if (size() == max_size) cerr << “Переполнение стека.\n”;
       *top++ = a;
}
char stack_char::pop(void) {
    if (size() == 0) cerr << “Исчерпание стека.\n”;
    return *--top;
}
unsigned stack_char::size(void) const {
    return top-base;
}
```

Шаблон класса *stack*

файл stack.h

```
#ifndef  __STACK_H
  #define __STACK_H
template<typename T> class stack {
      public:
        stack(unsigned);
        ~stack();
        void push(const T&);
        T pop(void);
        unsigned size(void) const;
      private:
        T* base;
        T* top;
        unsigned max_size;
};
#endif
```

Шаблоны методов класса *stack*

файл stack.hpp

```
#ifndef  __STACK_HPP
  #define __STACK_HPP
#include “stack.h”
#include <iostream.h>
#include <stdlib.h>
template<typename T> stack<T>::stack(unsigned sz) {
    if ((top=base=new T[max_size=sz])==NULL) {
      cerr << “Нет свободной памяти для стека.\n”;
      exit(1);
    }
}
template<typename T> stack<T>::~stack() {
    delete [ ]base;
}
```

Шаблоны методов класса *stack* (продолжение)

файл stack.hpp

```
template<typename T> void stack<T>::push(const T& a) {
    if (size() == max_size) cerr << “Переполнение стека.\n”;
    *top++ = a;
}
template<typename T> T stack<T>::pop(void) {
    if (size() == 0) cerr << “Исчерпание стека.\n”;
    return *--top;
}
template<typename T> unsigned stack<T>::size(void) const {
    return top-base;
}
#endif
```

Использование шаблона класса *stack*

```
#include “point.h”
#include “stack.h”
#include “stack.hpp”
void main(void)
{
    stack<char> sch(100);
    stack<int> sint(50);
    stack<Point> sp(400);
    sch.push(‘A’);
    sint.push(111);
    sp.push(Point(1,1));
    .....................
}
```

Параметры шаблонов

- Шаблон может иметь несколько параметров
- Параметры шаблона не обязательно являются именами типов
- Кроме имен типов можно использовать константные выражения

Параметры шаблонов

файл stack.cpp

```
template<typename T, unsigned max_size> class stack {
    public:
      stack();
      void push(const T&);
      T pop(void);
      unsigned size(void) const;
    private:
      T  base[max_size];
      T* top;
};
```

Параметры шаблонов

```
#include <iostream.h>
template<typename T, unsigned max_size>
    class stack<T, max_size>::stack() { top=base; }
template<typename T, unsigned max_size>
    void stack<T, max_size>::push(const T& a) {
      if (size() == max_size) cerr << "Переполнение стека.\n";
      *top++ = a;
    }
template<typename T, unsigned max_size>
    T stack<T, max_size>::pop(void) {
      if (size() == 0) cerr << "Исчерпание стека.\n";
      return *--top;
}
template<typename T, unsigned max_size>
    unsigned stack<T, max_size>::size(void) const {
    return top-base;
}
```

Использование

```
#include “point.h”
void main(void) {
    stack<char, 100> sch;
    stack<int, 50> sint1;
    stack<int, 10> sint2;
    stack<Point,400> sp;

    sch.push('A');
    sint1.push(111);
    sint2.push(111);
    // sint1 = sint2; - ошибка, т.к. переменные различных типов!
    sp.push(Point(1,1));
}
```

РЕЗЮМЕ

- Шаблон есть средство обобщенного *описания* класса, а не *определение* класса. (Компиляция файлов stack.h и stack.hpp не влечет распределения памяти. Реальное распределение памяти происходит при генерации параметризованных классов stack<char>, stack<int> и т.д. в тот момент, когда компилятор встречает определение переменной соответствующего типа.)
- Шаблон класса должен быть видим компилятору в момент использования. (Содержимое как файла stack.h, так и файла stack.hpp д. б. видимо внутри файла main.cpp. Поэтому, либо надо включать файлы, содержащие шаблоны классов директивой #include, либо задавать при компиляции глобальную область видимости шаблонов.)
- В первом примере размер стека являлся атрибутом переменных-экземпляров одного и того же класса. (Распределение памяти происходит на этапе выполнения. Присваивание переменных-стеков разного размера определено по умолчанию).
- Во втором примере размер стека является атрибутом, определяющим класс. (Он известен во время компиляции программы и распределение памяти происходит на этапе компиляции. Параметризованные классы с различными значениями параметра max_size задают *различные типы*.)