Объектно-ориентированное программирование на языке C++

Шаблоны

Шаблоны

Шаблон (англ. *template*) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов без привязки к типам данных.

Шаблоны функций

Родовая функция – определяет обобщенный набор операций, которые будут применены к различным типам данных. Также может называться шаблонной функцией.

Синтаксис:

```
template <class тип>  // may be typename
тип_возврата имя_функции(параметры) {
  // тело функции
}
```

Шаблоны функций

Пример:

```
template <class X>
void swap_any(X &a, X &b){
  cout << ”Обмен\n”;
  X temp_object = a;
  a = b;
  b = temp_object;
}
```

```
int main() {
  int a=3, b=4;
  char c=’c’, d=’d’;
  double e=0.3, f=0.567;
  swap_any(a, b);
  swap_any(c, d);
  swap_any(e, f);
  return 0;
}
```

Несколько родовых типов

Для шаблона допустимо использовать несколько типов:

```
template <class T1, class T2>
void show(T1 x, T2 y){
  cout << x << endl;
  cout << y << endl;
}
// …
char str[] = ”Hello”;
float a = -1.0;
show(a, str);
```

Для каждого набора параметров компилятор генерирует новый экземпляр функции. Процесс создания нового экземпляра называется *инстанцированием шаблона*

Явная перегрузка родовых функций

Используется, если для некоего типа поведение шаблонной функции должно отличаться.

Синтаксис:

```
template <>
тип имя <перегружаемый тип> (параметры) {
        // тело перегруженной функции
}
```

Пример:

```
template <>
void swap_any <int> (int &x, int &y) {
        cout << “Работа перегруженной ”
             << “шаблонной функции\n”;
        x = x + y;
        y = x – y;
        x = x – y;
}
```

Родовые классы

Синтаксис:

```
template <class тип>
class имя_класса {
    // Тело класса
};
```

Объявление объекта:

```
имя_класса <тип> имя_переменной;
```

Зачем: контейнеры для хранения данных (очередь, стек, массив и т.д.)

Родовые классы

```
template <class T> class MyArray {
  T array[10];
  int size;
public:
  MyArray() { this->size = 0; };
  MyArray(MyArray &c): array(c.array), size(c.size) {};
  void Add(T &element) { array[size++] = element; }
  T getElement(int index) { return array[index]; };
}

// …
MyArray <int> mas;
mas.Add(5);
mas.Add(6);
int tmp = mas.getElement(1);
```

Несколько типов и явная специализация

Так же, как и в родовых функциях, в родовых классах допустимо использование нескольких типов

```
template <class T1, class T2>
class MyClass {
     // тело класса
};
```

Явная специализация (перегрузка)

```
template<>
class MyClass<int, float> {
     // тело перегруженного класса
};
```

Шаблоны в модулях C++

Foo.h

```
template<typename T>
class foo {
public:
    void bar(const T &t);
};
```

main.cpp

```
#include ”Foo.h”

int main() {
  foo <int> a;
  a.bar(3);
  return 0;
}
```

Foo.cpp

```
#include ”Foo.h”
template <class T>
void foo<T>::bar(const T &t){
  // do smth useful
}

// Принудительное инстанцирование
template class foo<int>;
```

На оценку

1. Родовая функция вычисления квадрата элемента. Реализовать для чисел и для векторов (векторное произведение)
2. Где используется ключевое слово typename? Привести пример.
3. Перегрузка операторов ввода-вывода в шаблонах.

Задание:

1. Написать функцию, которая будет выводить на экран любые данные, переданные через параметр.
2. Реализовать класс MyVector, позволяющий хранить данные любого типа. Реализовать функции: добавления элемента, получения элемента, удаления элемента.
3. Реализовать класс MyStack, позволяющий хранить данные любого типа в виде стека. Реализовать функции: добавления элемента в стек, удаления из стека, проверки состояния стека (пуст/полон). Переопределить оператор вывода на экран.

Домашнее задание

Реализовать класс MyQueue, позволяющий хранить данные любого типа в виде очереди. Реализовать функции: добавления элемента в очередь, удаления из очереди, проверки состояния очереди (пуста/полна).