Объектно-ориентированное программирование

Шаблоны функций, шаблоны классов

Шаблоны функций

Шаблоны функций

- При создании функций иногда возникают ситуации, когда две функции выполняют одинаковую обработку, но работают с разными типами данных
  - например, одна функция сортирует массивы типа string, а другая типа float.
- Часто используют шаблоны функций для быстрого определения нескольких функций, которые с помощью одинаковых операторов работают с параметрами разных типов или имеют разные типы возвращаемых значений.

Описание шаблона функции

- Шаблон функции определяет типонезависимую функцию.

```
template< typename T >
void sort( T array[], int size );  // прототип: шаблон sort
                                   //объявлен, но не определён
template< typename T >
void sort( T array[], int size )   // объявление и определение
{
  T t;
  for (int i = 0; i < size - 1; i++)
    for (int j = size - 1; j > i; j--)
      if (array[j] < array[j-1])
      {
        t = array[j];
        array[j] = array[j-1];
        array[j-1] = t;
      }
}
```

- Буква T данном случае представляет собой общий тип шаблона.

Вызов шаблонной функции

- После определения шаблона внутри вашей программы вы объявляете прототипы функций для каждого требуемого вам типа.

```
int i[5] = { 5, 4, 3, 2, 1 };
sort< int >( i, 5 );

char c[] = "бвгда";
sort< char >( c, strlen( c ) );

// ошибка: у sort< int > параметр int[] а не char[]
sort< int >( c, 5 );
```

Выведение шаблонной функции

- В некоторых случаях компилятор может сам вывести (логически определить) значение параметра шаблона функции из аргумента функции.

```
int i[5] = { 5, 4, 3, 2, 1 };
sort( i, i + 5 );                    // вызывается sort< int >

char c[] = "бвгда";
sort( c, c + strlen( c ) );          // вызывается sort< char >
```

Генерация шаблонных функций

- Для каждого набора параметров компилятор генерирует новый экземпляр функции. Процесс создания нового экземпляра называется *инстанцированием шаблона*.
- Например, компилятор создал две специализации шаблона функции sort (для типов char и int)
- Для каждого возможного значения параметра компилятор будет создавать новые и новые экземпляры функций, которые будут отличаться лишь одной константой.

Параметры шаблонов

- Параметрами шаблонов могут быть: параметры-типы, параметры обычных типов, параметры-шаблоны.
- Для параметров любого типа можно указывать значения по умолчанию.

```
template< class T1,                    // параметр-тип
          typename T2,                 // параметр-тип
          int I,                       // параметр обычного типа
          T1 DefaultValue,             // параметр обычного типа
          template< class > class T3,  // параметр-шаблон
          class Character = char       // параметр по умолчанию
            >
```

Ошибки шаблонов

- Ошибки, связанные с использованием конкретных параметров шаблона, нельзя выявить до того, как шаблон использован.
- Например, шаблон **sort** сам по себе не содержит ошибок, однако использование его с типами, для которых операция '<' не определена, приведёт к ошибке

```
struct A
{
  int a;
};
A mas[5] = …;
sort( mas, 5 );
```

- Если ввести операцию '<' до первого использования шаблона, то ошибка будет устранена.

Шаблоны классов

Шаблон класса

- Шаблон класса позволяет задать класс, параметризованный типом данных.
- Передача классу различных типов данных в качестве параметра создает семейство родственных классов.
- Наиболее широкое применение шаблоны находят при создании *контейнерных классов*.

Контейнерный класс

- Контейнерным называется класс, который предназначен для хранения каким-либо образом организованных данных и работы с ними.
- Преимущество использования шаблонов состоит в том, что как только алгоритм работы с данными определен и отлажен, он может применяться к любым типам данных без переписывания кода.

Синтаксис описания шаблона класса

```
template <описание_параметров_шаблона> class имя
{ /* определение класса */ };
```

- Например:

```
template <typename Type>
class Queue {
public:
   Queue();
   ~Queue();

   Type& remove();
   void add( const Type & );
   bool is_empty();
   bool is_full();
private:
   // ...
};
```

Создание классов на основе шаблонов

- Определение методов шаблона вне класса:

```
template <typename T> Queue<T>::add (const T &a)
{ /*реализация класса*/ }
```

- Пример создания классов Очереди (Queue) для целых чисел, комплексных чисел, строк:

```
Queue<int> qi;
Queue<complex<double>> qc;
Queue<string> qs;
```

Специализация шаблонов

- Генерация конкретного класса из обобщенного определения шаблона называется конкретизацией шаблона.
- При такой конкретизации Queue для объектов типа `int` каждое вхождение параметра Type в определении шаблона заменяется на `int`, так что определение класса Queue принимает вид.
- Части <`int`> и <`string`>, следующие за именем Queue, называются фактическими аргументами шаблона.

Специализация шаблонов

```
template <>
class Queue <int> {
public:
   Queue() : front( 0 ), back (
0 ) { }
   ~Queue();

   int& remove();
   void add( const int & );
   bool is_empty() const {
      return front == 0;
   }
private:
   QueueItem<int> *front;
   QueueItem<int> *back;
};
```

Частичная специализация шаблонов

- Если у шаблона класса есть несколько параметров, то можно специализировать его только для одного или нескольких аргументов, оставляя другие неспециализированными.

```
template <int hi, int wid>
class Screen {
   // ...
};
// частичная специализация шаблона класса Screen
template <int hi>
class Screen <hi, 80> {
public:
   Screen();
   // ...
private:
   string            _screen;
   string::size_type _cursor;
   short             _height;
   // для экранов с 80 колонками используются специальные алгоритмы
};
```