Стандартная библиотека шаблонов

Функциональные объекты

Пример

```
class  Fun {
. . .
public:
   Fun() {   }
   int operator() ( int p1, int p2){
      return p1+p2;
   }
. . .
};
Fun f1; // функциональный объект
int k=f1(3,5);
```

Функциональные объекты (функторы)

- Рассмотрим простейшую задачу – поиск наибольшего элемента в массиве.
- Положим, что массив представлен в виде объекта класса **`vector <typename T>`**
- Дополнительно потребуем, чтобы для элементов массива была определена функция **`operator<`**

Шаблон функции

```
template <typename T>
const T& findMax(const vector<T> &a)
{ int maxIndex=0;
  for (int i=1; i<a.size();i++)
        if( a[maxIndex]<a[i])
              maxIndex=i;
  return a[maxIndex];
}
```

Недостаток

- Работает только с массивом объектов, для которых определена функция `operator<`
- Не может для одних и тех же объектов использовать разные варианты сравнений, например, прямоугольники можно сравнивать по площади, ширине, высоте

Решение

- Передавать в **`findMax`** в качестве второго параметра функцию сравнения для элементов массива
- C и C++ допускают передачу указателя на функцию в качестве параметра, но это допустимо не во всех объектно-ориентированных языках
- Использование функционального объекта (функтора) и передача его вторым параметром в функцию **`findMax`**

Функция с функциональным объектом

```
template <typename Object, typename Comparator>
const Object & findMax
        ( const vector<Object> & a, Comparator comp )
{
   int maxIndex = 0;
   for( int i = 1; i < a.size( ); i++ )
   if(comp.isLessThan(a[maxIndex],a[i]))
        maxIndex = i;
   return a[ maxIndex ];
}
```

Функтор для класса **Rectangle**

```
class LessThanByLength
{
  public:
    bool isLessThan(const Rectangle &lhs,
                    const Rectangle & rhs ) const
     {
    return lhs.getLength()<
    rhs.getLength();
}
};
```

Пример использования

```
int main(){
  vector <Rectangle> a;
  . . .
  cout<<findMax(a,LessThanByLength())
          << endl;
  . . .
}
```

Дополнительная возможность C++

- В функциональном объекте перегружается оператор вызова функции **`operator()`**
- Это позволяет улучшить вид функции **`findMax`**

Другой вариант функтора (в стиле C++)

```
class LessThanByArea
{
  public:
    bool operator()(const Rectangle & lhs,
          const Rectangle & rhs ) const
     { return lhs.getArea( )<
              rhs.getArea( );
     }
};
```

Измененная функция с функциональным объектом

```
template <typename Object, typename Comparator>
const Object& findMax(const vector<Object> &a,
                      Comparator isLessThan )
{
   int maxIndex = 0;
   for( int i = 1; i < a.size( ); i++ )
     if( isLessThan(a[maxIndex], a[i]))   maxIndex = i;
   return a[ maxIndex ];
}
```

```
int main( )
{
   vector<Rectangle> a;
. . .
cout << "Largest length:\n\t"
    << findMax( a, LessThanByLength( ) )   << endl;
cout << "Largest area:\n\t"
    << findMax( a, LessThanByArea( ) )   << endl;
   return 0;
}
```

Соглашение STL

- Функциональный объект (функтор) – это экземпляр объекта класса, в котором перегружена операции вызова функции

  operator()

- Функторы могут быть параметрами алгоритмов вместо указателей на функции

Пример

```
class LessThan50
{ public:
   bool operator() (int x) const {
         return x<50;}
};
vector  <int> v;
vector <int>::iterator it;
it=find_if(v.begin(), v.end(), LessThan50());
```

Типы функторов

- Каждый алгоритм может предъявлять определенные требования к виду функтора, который может с ним использоваться
  - *Генератор* – функтор, который вызывается без параметров
  - *Унарная функция* – функтор, имеющий один параметр
  - *Бинарная функция* –функтор, имеющий два аргумента
  - *Предикат* (бывает унарным и бинарным) – функтор, возвращающий значение типа *bool*

Стандартные функциональные объекты

- Эквиваленты всех встроенных операций
- Можно использовать со встроенными типами и с любым пользовательским типом, в котором перегружена соответствующая операция

negate <T> ()	-p
plus <T> ()	p1 + p2
minus <T> ()	p1 - p2
multiplies <T> ()	p1 * p2
divides <T> ()	p1 / p2
modulus <T> ()	p1 % p2
equal_to <T> ()	p1 == p2
not_equal_to <T> ()	p1 != p2
less <T> ()	p1 < p2
greater <T> ()	p1 > p2
less_equal <T> ()	p1 <= p2
greater_equal <T> ()	p1 >= p2
logical_not <T> ()	!p
logical_and <T> ()	p1 && p2
logical_or <T> ()	p1 || p2

Пример

```
vector <double> v1,v2;
ostream_iterator <double, char> out(cout,”  “);
transform(v1.begin(), v1.end(),
                    v2.begin(), out, plus<double>());
```

Адаптеры функциональных объектов

- Если интерфейс функционального объекта не соответствует требуемому в алгоритме интерфейсу, можно воспользоваться специальными адаптерам
- Имеются специальные адаптеры, для приспособления бинарных функций к унарному интерфейсу

Адаптеры функциональных объектов

bind1st() позволяет задать постоянное значение для первого аргумента бинарного функтора, превращая его в унарный

bind2nd() выполняет аналогичное преобразование, но полагая второй аргумент постоянным

Пример

- Пусть необходимо заменить все значения вектора на противоположные

```
transform (v.begin(), v.end(), out,
                                negate<double>());
```

В этой версии алгоритма требуется одноместная функция

Если нужно умножить все элементы вектора на одно и то же значение, то функтор multiplies не подходит под интерфейс алгоритма, изменим интерфейс

Пример

```
transform (v.begin(), v.end(), out,
                bind1st(multiplies<double>(), 2.5));
```

В данном случае не имеет значения, какой адаптер использовать – bind1st() или bind2nd()

- Для несимметричных функторов

```
find_if(v.begin(), v.end(),
                        bind2nd(greater<int>(), 5))
```

Находит первый, больше 5

```
find_if(v.begin(), v.end(),
                        bind1st(greater<int>(), 5))
```

Находит первый, меньше 5

Адаптеры - отрицатели

- Адаптеры

  not1(op) !op(param1)

  not2(op) !op(param1, param2)

  Применяются к одноместным и двуместным предикатам

  int array [4] = { 4, 9, 7, 1 };

  sort (array, array + 4, not2 (greater<int> ()));