Стандартная библиотека шаблонов

Функциональные объекты

Соглашение STL

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

  operator()

- Функторы могут быть параметрами алгоритмов вместо указателей на функции

Пример

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

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>());
```

Пример

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

Адаптеры функциональных объектов

- Если интерфейс функционального объекта не соответствует требуемому в алгоритме интерфейсу, можно использовать специальные адаптеры
- 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 умножить все элементы вектора на одно и то же значение

```
transform(v1.begin(), v1.end(),
                v2.begin(), out, f());
transform(v1.begin(), v1.end(),
                out, f());
```

- Что выбрать?

Пример

- Хотелось бы

```
transform(v1.begin(), v1.end(),
                  out, multiplies());
```

- 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> ()));