STL

Тема 4. Функциональные объекты

Функциональные объекты

- Любая сущность, применимая к нулю или любому другому количеству элементов для получения значения или изменения состояния
- Применяется для большинства алгоритмов STL

Функциональные объекты

```
template<class InputIterator, class T>
T accumulate( InputIterator First, InputIterator Last, _T Init ) {
    while (first != last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```

Функциональные объекты

```
template<class InputIterator, class T, class BinaryOperator>
T accumulate( InputIterator First, InputIterator Last, T Init,
                        BinaryOperator Binary_Op )
{
    while (first != last) {
            init = binary_function(init, *first);
            ++first;
    }
    return init;
}
```

Функциональные объекты

```
int multfunction(int x, int y)
{
    return x * y;
}

vector<int> v;

int n  = accumulate( v.begin(), v.end(), 1, multfunction);
```

Функциональные объекты

```
class multiply
{
public:
        int operator()(int x, int y) const { return x * y; }
};

multiply multfunobj;
vector<int> v;

int n  = accumulate( v.begin(), v.end(), 1,  multfunobj);
```

Функциональные объекты

```
template <class T>
class multiply : binary_function<T,T,T>
{
public:
    T operator()(const T& x, const T& y) const { return x * y; }
};
multiply<int> multfunobj;
vector<int> v;

int n  = accumulate( v.begin(), v.end(), 1,  multfunobj);
```

ФО предоставляемые STL

- Базовые классы
- Арифметические операции
- Операции сравнения
- Логические операции

ФО. Базовые классы

```
template<class _Arg, class _Result>
struct unary_function
{
    typedef _Arg argument_type; typedef _Result result_type;
};
```

```
template<class _Arg1, class _Arg2, class _Result>
struct binary_function
{
    typedef _Arg1 first_argument_type;
    typedef _Arg2 second_argument_type;
    typedef _Result result_type;
};
```

ФО. Арифметические операторы

```
template<class _Ty>
struct plus : public binary_function<_Ty, _Ty, _Ty>
{
    _Ty operator()(const _Ty& _Left, const _Ty& _Right) const
    {
        return (_Left + _Right);
    }
};
```

- template<class _Ty> struct minus;
- template<class _Ty> struct multiplies
- template<class _Ty> struct divides
- template<class _Ty> struct modulus
- template<class _Ty> struct negate

ФО. Операции сравнения

```
template<class _Ty>
struct less_equal : public binary_function<_Ty, _Ty, bool>
{
    bool operator()(const _Ty& _Left, const _Ty& _Right) const
    {
        return (_Left <= _Right);
    }
};
```

- template<class _Ty> struct equal_to
- template<class _Ty> struct not_equal_to
- template<class _Ty> struct greater
- template<class _Ty> struct less
- template<class _Ty> struct greater_equal

ФО. Логические операции

```
template<class _Ty>
struct logical_not : public unary_function<_Ty, bool>
{
    operator()(const _Ty& _Left) const
    {
        return (!_Left);
    }
};
```

- template<class _Ty> struct logical_and
- template<class _Ty> struct logical_or