STL Algorithms - 2

- Sorting and searching algorithms
- Numeric algorithm

Sorting and Searching Algorithms

- `sort`
- `partial_sort`
- `nth-element`
- `binary_search`
- `merge`
- `etc.`

sort

```
1)
template <class RandomAccessIterator>
void
sort(RandomAccessIterator first,
     RandomAccessIterator last);

2)
template <class RandomAccessIterator, class Compare>
void
sort(RandomAccessIterator first,
     RandomAccessIterator last,
     Compare comp);
```

Description

- `[first, last)` կիսահատվածի տարրերը տեսակավորվում են չնվազման կարգով: Առաջին տարբերակում տարրերը համեմատվում են `operator<` գործողության, իսկ երկրորդ տարբերակում՝ `comp` ֆունկցիոնալ օբյեկտի միջոցով: Չի երաշխավորվում, որ համարժեք տարրերը (ոչ մեկը մյուսից փոքր չէ) կպահպանեն իրենց հարաբերական կարգը տեսակավորումից հետո:

sort – (2)

Example

```
int a[] = {3, 10, 5, 9, 15, 21};

//sorting in non-decreasing order
sort(a, a+6);

//sorting in non-increasing order
sort(a, a+6, greater<int>());
```

partial_sort

```
1)
template <class RandomAccessIterator>
void
partial_sort(RandomAccessIterator first,
                  RandomAccessIterator middle,
                    RandomAccessIterator last);
2)
template <class RandomAccessIterator, class Compare>
void
partial_sort(  RandomAccessIterator first,
                 RandomAccessIterator middle,
                     RandomAccessIterator last,
                                    Compare comp);
```

Description

- `[first, last)` կիսահատվածի տարրերը վերադասավորվում են այնպես, որ ըստ մեծության առաջին `(middle-first)` տարրերը տեղադրվում են `[first, middle)` կիսահատվածում չնվազման կարգով: Մնացած տարրերը տեղադրվում են `[middle, last)` կիսահատվածում անորոշ կարգով: Առաջին տարբերակում տարրերը համեմատվում են `operator<` գործողության, իսկ երկրորդ տարբերակում՝ `comp` ֆունկցիոնալ օբյեկտի միջոցով:

partial_sort – (2)

Example

```
int a[] = {3, 10, 5, 1, 9, 15, 6, 30, 25, 20};

partial_sort(a, a+5, a+10);

copy(a, a+5, ostream_iterator<int>(cout, " "));
   //prints:1 3 5 6 9
```

nth_element

```
1)
template <class RandomAccessIterator>
void nth_element( RandomAccessIterator first,
                  RandomAccessIterator nth,
                  RandomAccessIterator last);
2)
template <class RandomAccessIterator, class Compare>
void nth_element(RandomAccessIterator first,
                  RandomAccessIterator nth,
                  RandomAccessIterator last, Compare comp);
```

Description

- `[first, last)` կիսահատվածի տարրերը վերադասավորվում են այնպես, որ `nth` դիրքում տեղադրվում է այն տարրը, որը կտեղադրվեր այդ դիրքում հաջորդականությունը ամբողջությամբ տեսակավորելուց հետո: Բացի դրանից, `[first, nth)` կիսահատվածում անորոշ կարգով տեղադրվում են `nth` դիրքում տեղադրված տարրից ոչ մեծ, իսկ `[nth, last)` կիսահատվածում՝ ոչ փոքր տարրերը: Առաջին տարբերակում տարրերը համեմատվում են `operator<` գործողության, իսկ երկրորդ տարբերակում՝ `comp` ֆունկցիոնալ օբյեկտի միջոցով:

binary_search

```
1)
template <class ForwardIterator, class LessThanComparable>
bool
binary_search(ForwardIterator first, ForwardIterator last,
                              const LessThanComparable& value);
2)
template<class ForwardIterator, class T, class Compare>
bool
binary_search(ForwardIterator first, ForwardIterator last,
                                  const T& value, Compare comp);
```

Description

- Ալգորիթմը վերադարձնում է `true`, եթե `[first, last)` կիսահատվածում կա `value` –ին *համարժեք* տարր, և `false`՝ հակառակ դեպքում: Առաջին տարբերակում տարրերը համեմատվում են `operator<` գործողության, իսկ երկրորդ տարբերակում՝ `comp` ֆունկցիոնալ օբյեկտի միջոցով:

merge

```
1) template <class InputIterator1, class InputIterator2,
                                   class OutputIterator>
OutputIterator
merge( InputIterator1 first1, InputIterator1 last1,
       InputIterator2 first2, InputIterator2 last2,
      OutputIterator result);
2) template <class InputIterator1, class InputIterator2,
             class OutputIterator, class Compare>
OutputIterator
merge( InputIterator1 first1, InputIterator1 last1,
       InputIterator2 first2, InputIterator2 last2,
      OutputIterator result, Compare comp);
```

Description

- Ենթադրվում է, որ `[first1, last1)` և `[first2, last2)` կիսահատվածներում գտնվում են չնվազման կարգով տեսակավորված հաջորդականություններ: Ալգորիթմի աշխատանքի արդյունքում այդ հաջորդականությունների տարրերից կառուցվում է չնվազման կարգով տեսակավորված նոր հաջորդականություն, որը տեղադրվում է `result` դիրքից սկսած: Վերադարձվում է վերջին արտագրված տարրի դիրքին հաջորդող դիրքը՝ `result + (last1 - first1) + (last2 - first2)`:

Numeric Algorithms

- `accumulate`
- `inner_product`
- `etc.`

accumulate

```
1)
template<class InputIterator, class T>
T
accumulate(InputIterator first, InputIterator last, T init);

2)
template<class InputIterator, class T, class BinaryFunction>
T
accumulate(InputIterator first, InputIterator last, T init,
                                    BinaryFunction bin_op);
```

Description

- Վերադարձվում է `[first, last)` կիսահատվածի տարրերի սովորական կամ *ընդհանրացված գումարը*՝ `init` արժեքից սկսած:

accumulate – (2)

Examples

```
int a[] = {4, 1, 3, 2, 1};
/*1*/ accumulate(a, a+5, 0);  //addition
/*2*/ accumulate(a, a+5, 1, multiplies<int>()); //mult.
/*3*/ //greatest common divisor
      int gcd(int x, int y)
      { while(x != y) x > y? x -= y: y -= x; return x; }
      accumulate(a+1, a+5, a[0], gcd);
/*4*/ //polynomial value
      struct ComputationStep{
        int x_;
        ComputationStep(int x): x_(x){}
        int operator()(int result, int coef)const{
           return result * x_ + coef;
        }
      };
      int x = 2;
      accumulate(a+1, a+5, a[0], ComputationStep(x));
```

accumulate – (3)

Implementation (second version)

```
template <class InputIterator, class T, classBinaryFunction>
T
accumulate (InputIterator first,InputIterator last,
            T init,
           BinaryFunction binary_op)
{
  T result = init;
  for(InputIterator it = first; it != last; it++)
     result = binary_op(result, *it);
  return result;
}
```

inner_product

```
1)
template<class InputIterator1, class InputIterator2, class T>
T
inner_product( InputIterator1 first1, InputIterator1 last1,
               InputIterator2 first2, T init);
2)
template<class InputIterator1, class InputIterator2, class T,
         class BinaryFunction1, class BinaryFunction2>
T
inner_product( InputIterator1 first1, InputIterator1 last1,
               InputIterator2 first2, T init,
               BinaryFunction1 op1, BinaryFunction2 op2);
```

Description

- Վերադարձվում է `[first1, last1)` և `[first2, first2+last1-first1)` կիսահատվածներում գտնվող տարրերի հաջորդականությունների սովորական կամ *ընդհանրացված* սկալյարային արտադրյալը՝ `init` արժեքից սկսած:

inner_product – (2)

Examples

```
int a[] = {2, 3, 5};
int b[] = {3, 1, 2};

//scalar product: 2*3+3*1+5*2
inner_product(a, a + 3, b, 0);                          //result: 19

//another way to compute the scalar product
list<int> lst;
transform(a, a + 3, b, back_inserter(lst), multiplies<int>());
accumulate(lst.begin(),lst.end(),0);                    //result: 19

//min-max: min{max{2, 3}, max{3, 1}, max{5, 2}}
inner_product(a, a + 3, b, INT_MAX, min, max);          //result: 3

//max-min: max{min{2, 3}, min{3, 1}, min{5, 2}}
inner_product(a, a + 3, b, INT_MIN, max, min);          //result: 2

//multiplication of powers: 2^3*3^1*5^2
inner_product(a, a+3, b, 1, multiplies<int>(), pow);
                                                        //result: 600
```

inner_product – (3)

Implementation (second version)

```
template <class InputIterator1, class InputIterator2, class T,
                  class BinaryFunction1, class BinaryFunction2>
T
inner_product(InputIterator1 first1, InputIterator1 last1,
              InputIterator2 first2,
               T init,
                BinaryFunction1 binary_op1,
                 BinaryFunction2 binary_op2
)
{
   T result = init;
   InputIterator1 it1 = first1;
   InputIterator2 it2 = first2;
   while( it1 != last1)
      result = binary_op1(result, binary_op2(*it1++,*it2++));
   return result;
}
```