STL: Algorithms

Instituto Balseiro

STL – Algorithms

- Alrededor de 80 algoritmos implementados en **`<algorithm>`**.
- No hay que reinventar la rueda.
- Implementación muy bien pensada y optimizada.
- Basados en secuencias **`[begin:end)`**.
- Las secuencias en la realidad están representadas por iteradores, principalmente de contenedores.
- Obviamente son funciones template.
- Las operaciones sobre los elementos pueden ser especificadas con una política como argumento.

STL – Algorithms – Complejidad

O(1)	`swap()`, `iter_swap()`
O(log(n))	`lower_bound()`, `upper_bound()`, `equal_range()`, `binary_search()`, `push_heap()`, `pop_heap()`
O(n log(n))	`inplace_merge()`, `stable_partition()`, `sort()`, `stable_sort()`, `partial_sort()`, `partial_sort_copy()`, `sort_heap()`
$O(n^2)$	`find_end ()`, `find_first_of()`, `search()`, `search_n()`
O(n)	El resto

STL – Nonmodifiying Algorithms

for_each	
`f = for_each(b,e,f)`	Aplica `f(x)` para cada `x` en `[b:e)`, retorna `f`

```
void increment_all(vector<int> &v) {
    for_each(v.begin(), v.end(), [](int &x){++x;});
}
```

Predicados sobre secuencias	
`all_of(b,e,f)`	Es `f(x)!=0` para todo `x` en `[b:e)`?
`any_of(b,e,f)`	Es `f(x)!=0` para algún `x` en `[b:e)`?
`none_of(b,e,f)`	Es `f(x)==0` para todo `x` en `[b:e)`?

```
vector<double> scale(const vector<double> &val, const vector<double> &div) {
    assert(val.size() == div.size());
    assert(all_of(div.begin(), div.end(), [](double x){ return x!=0; }));
    vector res(val.size());
    for( int i = 0; i < val.size(); ++i )
        res[i] = val[i] / div[i];
    return res;
}
```

STL – Nonmodifiying Algorithms

count	
`n=count(b,e,v)`	Cantidad de elementos `x` en `[b:e)` tal que `x==v`
`n=count(b,e,f)`	Cantidad de elementos `x` en `[b:e)` tal que `f(x)!=0`

```
void f(const string &s) {
    auto n_space = count(s.begin(), s.end(), ' ');
    auto n_whitespace = count_if(s.begin(), s.end(), isspace);
}
```

equal y mismatch	
`equal(b,e,b2)`	Es `v==v2` para todos los elementos correspondientes de `[b:e)` y `[b2:b2+(e-b))`?
`equal(b,e,b2,f)`	Es `f(v,v2)!=0` para todos... ?
`pair(p1,p2)= mismatch(b,e,b2)`	`p1` y `p2` apuntan a los primeros elementos en `[b:e)` y `[b2:b2+(e-b))`, respectivamente, que `*p1!=*p2` o `p1==e`
`pair(p1,p2)= mismatch(b,e,b2,f)`	... que `!f(*p1,*p2)` o `p1==e`

STL – Nonmodifiying Algorithms

`find` family	
`p=find(b,e,v)`	`p` apunta al primer elemento en `[b:e)` tal que `*p==v`
`p=find_if(b,e,f)`	... tal que `f(*p)!=0`
`p=find_if_not(b,e,f)`	... tal que `f(*p)==0`
`p=find_first_of(b,e,b2,e2)`	`p` apunta al primer elemento en `[b:e)` tal que `*p==*q` para `q` en `[b2:e2)`
`p=find_first_of(b,e,b2,e2,f)`	...tal que `f(*p,*q)!=0` para `q` en `[b2:e2)`
`p=adjacent_find(b,e)`	`p` apunta al primer elemento en `[b:e)` tal que `*p==*(p+1)`
`p=adjacent_find(b,e,f)`	`p` apunta al primer elemento en `[b:e)` tal que `f(*p,*(p+1))!=0`
`p=find_end(b,e,b2,e2)`	`p` apunta al último elemento en `[b:e)` tal que `*p==*q` para `q` en `[b2:e2)`
`p=find_end(b,e,b2,e2,f)`	... tal que `f(*p,*q)!=0` para `q` en `[b2:e2)`

STL – Modifiying Algorithms

Modifying sequence operations	
`generate_n()`	Reemplaza primeros n elementos por el resultado de operación
`remove()`	Remueve elementos con un determinado valor
`remove_if()`	Remueve elementos que matcheen un predicado
`remove_copy()`	Copia secuencia removiendo elementos de un valor
`remove_copy_if()`	Copia secuencia removiendo elementos que matcheen
`unique()`	Remueve elementos adyacentes iguales
`unique_copy()`	Copia secuencia eliminando elementos iguales adyacentes
`reverse()`	Invierte el orden de elementos
`reverse_copy()`	Copia una secuencia en orden inverso
`rotate()`	Rota elementos
`rotate_copy()`	Copia elementos en una secuencia rotada
`random_shuffle()`	Mueve elementos según una distribución uniforme

STL – Sorting Algorithms

Sorting operations	
`sort()`	Ordena secuencia eficientemente
`stable_sort()`	Ordena manteniendo el orden de elementos iguales
`partial_sort()`	Ordena la primer parte de una secuencia
`partial_sort_copy()`	Copia la primer parte de una secuencia en orden
`nth_element()`	Pone el elemento n en su lugar
`lower_bound()`	Primer ocurrencia de un valor
`upper_bound()`	Primer elemento mayor que un valor
`equal_range()`	Subsecuencia con un determinado valor
`binary_search()`	Busca un elemento en una secuencia ordenada
`merge()`	Mezcla 2 secuencias ordenadas
`inplace_merge()`	Mezcla 2 secuencias consecutivas ordenadas
`partition()`	Ubica elementos que matcheen un predicado
`stable_partition()`	Ubica elementos que matcheen manteniendo el orden relativo

STL – Algorithms

Set algorithms	
`includes()`	Verdadero si una secuencia es una subsecuencia de otra
`set_union()`	Construye una unión ordenada
`set_intersection()`	Construye una intersección ordenada
`set_difference()`	Construye secuencia ordenada de elementos una secuencia pero no en la otra
`set_simmetric_ difference()`	Construye secuencia ordenada de elementos una secuencia pero no en ambas secuencias

Heap operations	
`make_heap()`	Alista una secuencia para se utilizada como un heap
`push_heap()`	Agrega elementos a un heap
`pop_heap()`	Remueve elemento de un heap
`sort_heap()`	Ordena un heap

STL – Algorithms

Minimum and maximum	
`min()`	Mínimo de 2 valores
`max()`	Máximo de 2 valores
`min_element()`	Menor elemento en una secuencia
`max_element()`	Mayor elemento en una secuencia
`lexicographical_compare()`	Compara 2 secuencias lexicográficamente

Permutations	
`next_permutation()`	Siguiente permutación en orden lexicográfico
`prev_permutation()`	Permutación previa en orden lexicográfico

STL – Function Objects

```
// Búsqueda de un valor

void f(list<int> &c) {
    list<int>::iterator p = find(c.begin(), c.end(), 7);
}


// Búsqueda según un criterio

bool less_than_7(int v) {
    return v < 7;
}

void f(list<int> &c) {
    list<int>::iterator p = find_if(c.begin(), c.end(), less_than_7);
}
```

STL – Function Objects

```
template<class T> class Sum {
        T res;
    public:
        Sum(T i = 0) : res(i) { }
        void operator()(T x) { res += x; };
        T result() const { return res; };
};


void f(list<double> &ld) {
    Sum<double> s;
    s = for_each(ld.begin(), ld.end(), s);
    cout << "La suma es " << s.result() << endl;
}
```

STL – Function Objects Bases

```
template<class Arg, class Res> struct unary_function {
    typedef Arg argument_type;
    typedef Res result_type;
};


template<class Arg, class Arg2, class Res> struct binary_function {
    typedef Arg first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Res result_type;
};
```

STL – Predicates

```
template<class Arg, class Res> struct unary_function {
    typedef Arg argument_type;
    typedef Res result_type;
};

template<class Arg, class Arg2, class Res> struct binary_function {
    typedef Arg first_argument_type;
    typedef Arg2 second_argument_type;
    typedef Res result_type;
};

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

template<class T> struct less : public binary_function<T, T, bool> {
    bool operator()(const T &x, const T &y) const { return x < y; }
};
```