GEPROC-ULA

Grupo de Entrenamiento de Programación Competitiva ULA

Universidad de Los Andes

Biblioteca Estandar de plantillas STL c++

STL

- Es un conjunto de clases parametrizadas de **C++** que proveen estructuras de datos comunes y funciones.
- Es una biblioteca de contenedores,algoritmos e iteradores.
- Es una biblioteca genérica por lo que sus componentes están parametrizados.
- Para trabajar con los contenedores de la **STL** es necesario conocer las plantillas.
- La **STL** provee ciertos contenedores y funciones que son útiles para la programación competitiva.

Componentes de la STL

Componentes de la STL: Algoritmos

- La STL posee una gran variedad de algoritmos que pueden ser usados en cualquier contenedor, independientemente de su tipo.
- Están especialmente diseñados para trabajar sobre rangos de elementos.
- Tipos de algoritmos:
  - Algoritmos de ordenamiento
  - Algoritmos de búsqueda
  - Algoritmos no modificadores.
  - Algoritmos modificadores.
  - Algoritmos numéricos.
  - Operaciones Mínimo y Máximo.

Componentes de la STL: Contenedores

- Los contenedores o clases contenedoras almacenan objetos y datos.
- Estos contenedores son las estructuras de datos mas comunes como arreglos, listas, arboles (entre otros).
- Estos contenedores son genéricos, pueden almacenar elementos de cualquier **tipo**.
- **Contenedores de secuencias**: Implementan estructuras de datos que se acceden de manera secuencial.
  - vector
  - list
  - deque
  - array
  - foward_list (c++11)

Componentes de la STL: Contenedores

- **Contenedores adaptadores**: Proveen una interfaz diferente para contenedores secuenciales.
  - queue
  - priority_queue
  - stack
- **Contenedores asociativos**: Implementan estructuras de datos orientadas a la búsqueda eficiente.
  - set
  - multiset
  - map
  - multimap
  - unordered_set (c++14)
  - unordered_map (c++14)

Componentes de la STL: Iteradores

- Utilizados para trabajar sobre una secuencia de valores.
- Son la característica que provee mayor generalidad en la STL.
- Son usados para apuntar las direcciones de memoria de los contenedores de la STL.
- Reducen la complejidad y el tiempo de ejecución de un programa.
- Los algoritmos en la STL no trabajan sobre contenedores directamente, en lugar de eso trabajan sobre iteradores.
- Los iteradores permiten realizar operaciones que son **independientes del tipo de contenedor**.

Ejemplos: Algoritmos

SORT

- Esta función de la STL, ordena los elementos en un rango dado.
- Hay dos versiones de la función sort.
  - **sort(start_iterator, end_iterator)** : Ordena el rango definido por el iterador de inicio y el iterador de fin de manera ascendente.
  - **sort(start_iterator, end_iterator, compare_function)**: Ordena el rango definido por el iterador de inicio y el iterador de fin segun el criterio de comparación.

```cpp
#include <cstdio>
#include <algorithm>

int N = 10;
void show(int * array,int N)
{
        for(int i = 0; i < N; ++i)
                printf("%d ",a[i]);
}

int main()
{
        int a[N]= {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
        printf("\n El arreglo antes de ordenar es : ");
        show(a,N);

        // ordenar
        std::sort(a, a+N);

        printf("\n\n El arreglo después de ordenar es : ");
        show(a,N);

        return 0;
}
```

```cpp
#include <cstdio>
#include <algorithm>

int N = 5;
void show(int * array,int N)
{
        for(int i = 0; i < N; ++i)
                printf("%d ",a[i]);
}
bool compare_function(int i, int j)
{
        return i > j;    // return 1 if i>j else 0
}

int main()
{
        int arr[N] = {1,5,8,4,2};
         // sorts arr[0] to arr[4] in ascending order
        std::sort(arr , arr+5,compare_function);
        show(arr,N);

        return 0;
}
```

PARTIAL SORT

- Esta función de la STL, ordena una parte de los elementos en un rango dado.
- Hay dos versiones de la función **partial_sort**.
  - **partial_sort(start, middle, end)** : Ordena el rango desde el iterador de inicio hasta iterador de fin de tal forma que los elementos antes de **middle** estan en orden ascendente y son los menores elementos del rango.
  - **sort(start_iterator, end_iterator, compare_function)**: Ordena el rango desde el iterador de inicio hasta el iterador fin de tal forma que los elementos antes de **middle** están en orden según el criterio de comparación.

```cpp
#include <cstdio>
#include <algorithm>
void show(int * array,int N)
{
        for(int i = 0; i < N; ++i)
                printf("%d ",a[i]);
        printf("\n");
}
int main()
{
        int a[] = {9,8,7,6,5,4,3,2,1};

        std::partial_sort(a, a+4, a+9);
        show(a,9);

        int b[] = {1,5,6,2,4,8,9,3,7};

        std::partial_sort(b, b+4, b+9);
        show(b,9);
}
```

IS_SORTED

- Esta función de la **STL** retorna verdadero si el rango dado esta ordenado.
- Exiten dos versiones de la función **is_sorted()**
  - **is_sorted(start_iterator, end_iterator)**: Verifica que el rango definido por el iterador de inicio e iterador de fin esta en orden ascendente.
  - **is_sorted(start_iterator, end_iterator, compare_function)**:: Verifica que el rango definido por el iterador de inicio e iterador de fin según el criterio de ordenamiento.

```cpp
#include <cstdio>
#include <algorithm>

using namespace std;

int main()
{
        int a[5] = {1,5,8,4,2};
        if(! std::is_sorted(a, a+5)){
                std::sort(a,a+5);
                printf("Esta ordenado\n");
        }
        else
                printf("Esta ordenado\n");

        return 0;
}
```