UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS
DEPARTAMENTO DE INFORMÁTICA

Ingeniería en Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

Ingeniería en Inteligencia Artificial

PROGRAMACIÓN II

UNIDAD 8
Biblioteca STL

Guía de Trabajos Prácticos

EJERCICIOS BÁSICOS

Ejercicio 1

Escriba un programa que permita al usuario ingresar 15 valores por teclado, los almacene en un vector y luego:

1. Muestre el vector generado utilizando 3 mecanismos de iteración diferentes:
    - utilizando el operador `[]`
    - utilizando iteradores
    - utilizando el for basado en rangos
    - ¿en qué caso es conveniente utilizar cada mecanismo?
2. Calcule y muestre:
    - los valores de los elementos máximo y mínimo
    - la suma de todos los elementos del arreglo
    - el promedio y la mediana de los elementos del arreglo
3. Permita al usuario ingresar un valor, e informe si se encuentra en el vector, y en caso afirmativo, en qué posición.

Ejercicio 2

Escriba una función C++ genérica llamada `dosmayores_up(...)` que reciba un vector como parámetro. La función debe modificadar el vector: debe obtener los 2 mayores y ubicar el primer mayor en el 1er lugar (al comienzo del vector) intercambiando los elementos correspondientes, y al 2do mayor en el 2do lugar, el resto de los elementos quedan en sus posiciones originales.

Nota: no utilice estructuras de control iterativas, solo recorra la secuencia a través de las funciones de la biblioteca `<algorithm>`.

Ejercicio 3

Escriba un programa que defina un vector dinámico de 30 enteros aleatorios menores que 25. Luego:

1. Ordene en forma descendente los elementos ubicados entre las posiciones 10 y 20 inclusive, y muestre el vector.
2. Inserte en un nuevo vector los elementos que fueron ordenados en el apartado anterior, y quítelos del vector original.
3. Permita ingresar un valor por teclado, y muestre cuántas veces aparece dicho valor en el vector.
4. Elimine todas las ocurrencias del valor ingresado en el punto anterior, utilizando la función remove. Responda: ¿Pueden las funciones globales de la STL eliminar realmente los elementos (liberando la memoria de un contenedor)?

Ejercicio 4

Cree un programa que lea valores flotantes por teclado y los inserte en una lista. Muestre la lista, Inserte en medio de cada par de elementos contiguos el promedio de dichos elementos y muestre la lista modificada.

Responda: ¿es posible ordenar solamente una porción de la lista de la misma manera que se hizo con el vector en el ejercicio 2?

Ejercicio 5

Declare un `std::vector` de enteros de tamaño 20 y luego:

1. Implemente una función `int rand10()` que genere un entero aleatorio entre -10 y 10, y utilícela como argumento para el algoritmo `generate` para inicializar el arreglo con valores aleatorios.
2. Implemente una función `bool es_par(int x)` que retorne true si el entero que recibe es par; y utilícela en combinación con `count_if` para contar cuantos elementos pares hay en el contenedor generado.
3. Implemente una función `bool menor_abs(int a, int b)` que reciba dos enteros y retorne verdadero cuando el valor absoluto del primero sea menor que el valor absoluto del segundo; y utilice esta función en como argumento de `sort` para ordenar el vector por valor absoluto.
4. Elimine los elementos repetidos utilizando los algoritmos genéricos de la STL, y luego muestre el arreglo resultante.
    - *Ayuda*: recuerde que el algoritmo `unique` requiere que los valores repetidos estén contiguos en la secuencia para funcionar correctamente. Si en el punto 3 ordenó solo por valor absoluto, valores repetidos podría estar separados. Por ejemplo, el vector *{1,**-2,2,-2***,-3,3,3}* debería quedar *{1,-2,2,-3,3}* (eliminar el 2do -2, y los dos últimos -3, pero `unique` no lo hará bien).
        - Pruebe usar la sobrecarga de `unique` que recibe el comparador con `menor_abs`. ¿Qué problema tiene para este caso?
        - Intente ordenar primero de la forma habitual, y luego por valor absoluto (para ello investique la diferencia entre `sort` y `stable_sort`).
        - Modifique su función `menor_abs` para que en caso de recibir dos nro con igual valor absoluto pero diferente signo, considere menor al negativo, y resuelva ahora el problema utilizando un solo `sort`.

Ejercicio 6

Realice las mismas operaciones que en el ejericio anterior, pero utilizando ahora un **arreglo estático de 20 elementos** enteros (`int v[20]`).

Ejercicio 7

Implemente una función genérica `unique_no_sort` que elimine los elementos repetidos de un contenedor (de cualquier tipo) **sin modificar el orden** de los elementos que no se eliminan (es decir, sin ordenar previamente, por lo que no podrá usar `unique`).

CUESTIONARIO

1. ¿Qué es la STL? ¿Para qué sirve? ¿Qué ventajas trae su uso?
2. Qué es un contenedor? ¿Cuáles son los tipos de contenedores?
3. ¿Qué es un iterador? ¿Para qué sirve?
4. ¿En qué se diferencian los algoritmos de la STL de otros algoritmos convencionales?
5. ¿Cuáles son las diferencias entre un vector y una lista? ¿En que casos conviene usar uno o el otro?
6. ¿Por qué la clase `std::list` implementa sus propios algoritmos como `sort()` en lugar de utilizar los algoritmos genéricos de la STL?

EJERCICIOS ADICIONALES

Ejercicio 1

Diseñe una clase que permita manejar un archivo de configuración con el formato que se muestra en el recuadro. Las opciones deberán ser almacenadas en un mapeo de tipo string a string, y los valores de las opciones serán convertidos a otros formatos (int, float, etc) cuando el usuario lo solicite.

```
unaopcion=5
otraopcion=hola
fuerzagravedad=9.8
```

La clase debe proveer funcionalidades para:

1. Leer un archivo de texto dado por el usuario y poblar el mapa con las opciones encontradas.
2. Agregar una opción con su respectivo valor, si dicha opción ya existe su valor debe actualizarse. Se recomienda que esta función tenga varias sobrecargas para soportar distintos tipos de datos (`int`, `float`, etc) y los convierta a string para poder almacenarlos en el mapa.
3. Devolver por referencia el valor asociado a una determinada opción y un booleano indicando si dicho valor se encontró o no en el mapa. Se recomienda que esta función tenga varias sobrecargas para soportar los diversos tipos de datos (`int`, `float`, etc).
4. Guardar la configuración modificada en un archivo de texto.

Se propone el siguiente prototipo para la clase:

```
class Config {
  public:
    // constructores
    Config();
    Config(string filename);
    // cargar y guardar en archivo de texto
    int Load(string filename);
    int Save(string filename);
    // pedir un valor
    bool GetValue(string option, float &value, float def_val=0);
    bool GetValue(string option, int &value, int def_val=0);
    bool GetValue(string option, string &value,
                                 string def_val="");
    // agregar o modificar un valor
    bool SetValue(string option, float newValue);
    bool SetValue(string option, int newValue);
    bool SetValue(string option, string newValue);
  private:
    map<string,string> entries;
};
```

Analice: ¿Podría reemplazar las sobrecargas de `GetValue` y `SetValue` por funciones genéricas?

Ejercicio 2

El archivo *"datos.txt"* contiene una lista de valores flotantes dispuestos uno por línea. Lea el contenido del archivo y almacénelo en una lista utilizando el algoritmo `copy`. Para esto haga lo siguiente:

1. Abra el archivo para lectura y cree un iterador de lectura para leer valores flotantes que apunte al principio del archivo de la siguiente forma: `stream_iterator<float> p(archi)`.
2. Cree otro iterador que apunte al final del archivo e indique el fin del archivo.
3. Cree un contenedor para almacenar lo que se leerá.
4. Utilice el algoritmo `copy` para copiar los datos del archivo al contenedor.

Luego, realice las siguientes operaciones:

1. Calcule el promedio de los elementos del contenedor.
2. Reste dicho valor a cada uno de los elementos.
3. Guarde los datos modificados en el archivo utilizando nuevamente el algoritmo `copy`.