Ejercicio 1.1

a) Escriba una función templatizada que reciba un arreglo estático y devuelva la posición del mayor de sus valores.

b) Utilice dicha función en un programa cliente con los tipos *int*, *double* y *string*.

c) Responda: ¿Sería posible utilizar dicha función con un arreglo de tipo *sf::Image*? Justifique su respuesta.

d) Intente realizar la acción que propone el apartado anterior. Observe la salida del compilador.

Ejercicio 1.2

a) Escriba un programa C++ que defina un vector dinámico de 50 elementos enteros aleatorios entre 0 y 100.

b) Muestre el vector generado.

c) Utilice los algoritmos de la biblioteca STL para realizar las siguientes tareas:

   1) Busque, muestre y luego elimine el mayor de los elementos.

   2) Busque, muestre y luego elimine el menor de los elementos.

   3) Ordene en forma ascendente la parte del vector formada por los elementos ubicados entre las posiciones 10 y 40 inclusive y muestre el arreglo resultante.

d) Responda: ¿Es posible realizar el apartado anterior con una lista? Justifique su respuesta.

e) Cree un nuevo vector con los elementos ubicados entre las posiciones 5 y 45 del vector resultante. Elimine del nuevo vector los valores repetidos y muéstrelo.

f) Realice lo mismo del apartado anterior pero esta vez con una lista.

Ejercicio 1.3

En los videjuegos que desarrollaremos nos será de mucha utilidad un parser para leer y generar nuestros propios archivos de configuración. Implemente una clase que provea dicha funcionalidad basándose en los siguientes lineamientos:

- Los archivos de configuración serán archivos de texto conformados por líneas cuya forma se ejemplifica a continuacion.

  unaopcion=valor
  otraopcion=otrovalor
  gravedad=9.8

Es decir, cada linea debe contener el nombre de la opcion y su valor separados por el caracter =.

- Se deberá parsear el archivo de entrada y guardar cada opción con su respectivo valor como strings en una estructura de tipo *map<string, string>*. La conversión a otros tipos será realizada cuando el usuario solicite un determinado valor.

- La clase debe proveer funcionalidades para:

  - ✓ Leer un archivo de texto dado por el usuario y poblar el mapa con las opciones encontradas y sus respectivos valores (utilice la función *getline()*).

  - ✓ Guardar la configuración actual en un archivo de texto.

  - ✓ La clase debe poseer funciones que permitan al usuario agregar una nueva opción con su respectivo valor. Si ya existe una opción con dicho nombre, debe actualizarse su valor. Se recomienda que esta función tenga sobrecargas para soportar diversos tipos de datos y los convierta a *string* para almacenarlos en el mapa.

  - ✓ La clase debe poseer funciones que permitan al usuario proveer el nombre de una opción y una referencia a la variable que debe llenarse con el valor de la opción específica. Dicha función debe recibir como parámetro el valor por defecto con el que será llenada la variable en caso de no encontrar dicha opción. Además debe devolver verdadero o falso según la opción especificada haya sido encontrada o la variable haya sido llenada con el valor por defecto. Se recomienda que esta función tenga sobrecargas para soportar diversos tipos de datos.

- Si lo desea, puede agregar una función que muestre por consola todos los nombres de las opciones con sus respectivos valores para utilizarla con fines de depuración.

- Se propone el siguiente prototipo para la clase:

```
class Config {
private:
    map<string,string> entries;

public:
    Config();
    Config(string filename);

    int Load(string filename);
    int Save(string filename);

    bool GetValue(string option, float &value, float default_value=0);
    bool GetValue(string option, int &value, int default_value=0);
    bool GetValue(string option, string &value, string default_value="");

    bool SetValue(string option, float newValue);
    bool SetValue(string option, int newValue);
    bool SetValue(string option, string newValue);

    void showallentries();
};
```