Unidad Nº6: Plantillas

Preguntas orientadoras

1) ¿Por qué utilizar plantillas cuando se tienen las macros?

2) ¿Cuál es la diferencia entre el tipo parametrizado de una función de plantilla y y los parámetros para una función normal?

3) ¿Para qué sirve generar un template o plantilla de función?

4) ¿Es posible proporcionar un comportamiento especial para una instancia de una plantilla, pero no para otras instancias? Si es posible, mencione un ejemplo.

5) ¿Cuántas variables estáticas se crean si se coloca un miembro estático en la definición de una clase de plantilla?

6) Explique el motivo por el cual se genera un error al separar en el proyecto la declaración y la implementación de un template. ¿Cómo se puede solucionar este inconveniente?

Ejercicios

1) A continuación se da una lista de descripciones de funciones y clases de plantillas, escriba una declaración apropiada para las mismas.

a) Declarar una función que toma dos parámetros de plantilla distintos de los cuales uno es el tipo de retorno y el otro es argumento.

b) Declarar una clase que toma un parámetro de plantilla, el cual es una variable miembro (atributo) de la misma.

c) Declarar una clase que toma dos parámetros de plantilla, uno como argumento al constructor y otro como tipo de retorno de una función miembro (método) sin argumentos.

d) Declarar una clase de plantilla con un parámetro entero con un valor por defecto cualquiera y que sirva como el tamaño de un atributo array.

e) Declarar una clase que tiene como amiga a una clase de plantilla distinta (con tantos parámetros como se desee).

f) Declarar una clase en la que uno de los parámetros es a su vez una clase de plantilla.

2) Crear una clase de plantilla Vector que facilite el manejo de arrays y los dote de seguridad en el acceso a índices fuera de rango. La clase Vector a implementar recibirá un único parámetro de plantilla para el tipo a almacenar. Las funciones a implementar serán:

a) El constructor inicializa los atributos a valores seguros.
b) Un método redimensionar que cambia el tamaño máximo del vector, reserva espacio cuando sea necesario (por ej, primera reserva de memoria) y copia los datos del array viejo cuando sea necesario (por ej, redimensionar a un tamaño mayor).
c) El destructor, que liberará la memoria.
d) El Operador [] para acceder a un elemento debe devolver el valor por referencia para que el siguiente código sea válido: Vector v; v.redimensionar(1); v[0] = 10; // Modificación std::cout << v[0] << std::endl; //Acceso. Si se intenta acceder a un elemento por encima del máximo se debe imprimir un mensaje de error. Nota: Un diseño más correcto incluiría una sobrecarga adicional del operador[] con modificadores const.
e) No hemos definido constructor de copia. ¿Qué peligros tendría asignar un Vector a otro? Definir el constructor de copia para que no haya peligro alguno (Pista: por simplicidad, el array interno no puede ser compartido, debe copiarse)
f) Un constructor de copia desde otro tipo de vector distinto que funcione cuando existe una conversión válida entre ambos tipos. Preocuparse sólo de los tipos escalares (int, float, double) porque entre ellos sí hay conversión.