Templates

1. Implementar un template llamado StaticArray que tome como el tipo de dato del arreglo y su size.

Los métodos que tendrá son:

* insert() que inserta un elemento en la próxima posición disponible
* delete() que borra el ultimo elemento
* getElement() que toma el índice del elemento y devuelve el elemento en ese índice
* reverse() que revierte el contenido del arreglo
* print() imprime su contenido
* totalSize() devuelve la cantidad de elementos
* availableSize() retorna la cantidad de lugar disponible para seguir agregando elementos
* swap() que toma dos índices e intercambia los valores de los elementos en dichos índices
* Especializar parcialmente el template, donde la cantidad de elementos es 1000
* Especializar totalmente el template, donde el tipo es float y la cantidad de elementos es 1000

Todo el manejo de errores y chequeo de precondiciones tiene que ser realizado por su clase, no por el usuario de la misma.

2)Enumere las principales diferencias entre “class template” y “function template”, con respecto al tratamiento de los tipos.

3)Utilice templates para cambiar la siguiente clase a una clase generica que reemplace “double” -> “Template Parameter T”

class vector  
{  
 int sz; // number of elements  
 double\* elem; // address of first element  
 int space; // number of elements plus free-space  
 // slots for new elements  
public:  
 vector() : sz(0), elem(0), space(0) {}  
 vector(int s) : sz(s), elem(new double[s]), space(s) {  
 for(int i = 0; i < sz; ++i) elem[i] = 0;  
 }  
 vector(const vector&);  
 vector& operator=(const vector & v);  
  
 ~vector() { delete[] elem; }  
  
 double& operator[](int n);  
 const double& operator[](int n) const;  
  
 int size() const { return sz; }  
 int capacity() const { return space; }  
  
 void reserve(int alloc\_size);  
 void resize(int resize\_size);  
 void push\_back(double d);  
};  
  
void vector::reserve(int alloc\_size) {  
 if(alloc\_size <= space) return;  
 double \*p = new double[alloc\_size];  
 for(int i = 0; i < sz; ++i) p[i] = elem[i];  
 delete[] elem;  
 elem = p;  
 space = alloc\_size;  
}  
  
vector& vector::operator=(const vector & v) {  
 if(this == &v;) return \*this;  
  
 if(v.sz <= space) {  
 for(int i = 0; i < v.sz; ++i)   
 elem[i] = v.elem[i];  
 sz = v.sz;  
 return \*this;  
 }  
  
 double \*p = new double[v.sz];  
 for(int i = 0; i < v.sz; ++i)   
 p[i] = v.elem[i];  
 delete[] elem;  
 space = sz = v.sz;  
 elem = p;  
 return \*this;  
}  
  
void vector::resize(int resize\_size) {  
 reserve(resize\_size);  
 for(int i = 0; i < resize\_size; ++i) elem[i] = 0;  
 sz = resize\_size;  
}  
  
void vector::push\_back(double d){  
 if(space == 0)   
 reserve(10);  
 else if(sz == space)   
 reserve(2\*space);  
 elem[sz] = d;  
 ++sz;  
}