TP 10

Alumnos: Eric Galera

1.¿Cuál es la principal diferencia entre un arreglo y un vector en C++? Elaborar un cuadro comparativo
2. ¿Cuál es la diferencia entre un vector tradicional y un vector de clase en C++? Elaborar un cuadro comparativo con las principales características de cada uno
B- Resolver las actividades planteadas en el TP Nº10.

Rta 1:

Característica	Arreglo	Vector (std::vector)
Tamaño	Fijo al momento de la declaración.	Dinámico; puede crecer o reducirse según sea necesario.
Declaración	Se declara con un tamaño fijo. Ejemplo: int arr[10];	Se declara sin necesidad de especificar el tamaño inicial. Ejemplo: std::vector<int> vec;
Redimensionamiento	No se puede redimensionar después de su declaración.	Puede redimensionarse con funciones como push_back(), resize(), etc.
Acceso a Elementos	Se accede mediante índices. Ejemplo: arr[0]	Igual que un arreglo, se accede con índices. Ejemplo: vec[0]
Gestión de Memoria	La memoria es estática o en el stack (si se declara sin punteros).	La memoria es gestionada dinámicamente en el heap.
Funcionalidades Adicionales	No hay funciones incorporadas para manipular arreglos.	Tiene muchas funciones útiles: push_back(), size(), clear(), etc.
Seguridad	No realiza verificaciones de límites en tiempo de ejecución.	Tiene verificaciones de límites si se usa at(), pero no con operator[].
Asignación y Copia	Puede ser complicado (especialmente con punteros).	Fácil de copiar o asignar usando operaciones de asignación estándar.
Requiere Biblioteca	No requiere bibliotecas adicionales.	Requiere la inclusión de la biblioteca <vector>.
Coste en Términos de Rendimiento	Mejor rendimiento en operaciones simples (por ejemplo, si no necesitas redimensionar).	Puede ser más lento debido a la gestión dinámica de la memoria, pero ofrece más flexibilidad.

Rta 2ª:

Característica	Vector Tradicional (Arreglo Dinámico)	Vector de Clase (std::vector)
Gestión de Memoria	Se maneja manualmente con new y delete.	Automática; gestionada por la clase con RAII (destruye y libera memoria).
Redimensionamiento	No es sencillo; debe ser manualmente redimensionado (creando un nuevo arreglo).	Fácil redimensionamiento dinámico con métodos como push_back(), resize().
Declaración	Se usa un puntero para asignar memoria dinámicamente. Ejemplo: int* arr = new int[n];	Simple, solo se declara el vector. Ejemplo: std::vector<int> vec;
Acceso a Elementos	Acceso con operadores de índice (arr[i]).	Igual que un arreglo, pero también se puede usar at() para verificación de límites.
Asignación de Tamaño Inicial	Se asigna manualmente, por ejemplo, usando new y especificando el tamaño inicial.	Se puede especificar o no, y puede crecer dinámicamente según sea necesario.
Funcionalidades Incorporadas	No tiene funcionalidades como la gestión de tamaño o inserción automática.	Tiene funciones como push_back(), size(), capacity(), clear(), entre otras.
Verificación de Límites	No realiza verificaciones de límites, lo que puede causar errores de desbordamiento de memoria.	Usa at() para verificaciones de límites o operator[] para acceso sin verificación.
Reasignación	Requiere crear un nuevo bloque de memoria y copiar los datos manualmente.	Se maneja automáticamente, expandiendo la capacidad cuando es necesario.
Sobrecarga de Operadores	No tiene sobrecarga de operadores, deben implementarse manualmente si se requieren.	std::vector sobrecarga varios operadores (=, [], etc.) para facilitar su uso.
Destrucción Automática	La memoria debe ser liberada manualmente con delete[].	La clase std::vector automáticamente libera la memoria al destruirse.
Copia y Asignación	Debe hacerse manualmente copiando los elementos uno por uno.	Se puede copiar directamente con la asignación estándar o el constructor de copia.
Rendimiento	Puede ser más rápido si la gestión de memoria y las operaciones se realizan correctamente.	Ligeramente más lento debido a las comprobaciones internas y gestión de memoria, pero mucho más seguro y flexible.
Requiere Biblioteca	No requiere bibliotecas estándar adicionales.	Requiere la inclusión de la biblioteca <vector>.