**Trabajo Práctico Nº4**

**Funciones / Strings / Vectores / Clases**

Alumno: Nahuel Vilca

📝**ACTIVIDADES:**  
Resolver las siguientes actividades utilizando el material adjunto, investigar en Internet en caso de ser necesario:

1. Explica la diferencia entre funciones y procedimientos en C++.

¿Cuándo es más adecuado utilizar una u otra?

1. Describe los componentes de una función.
2. Explica las diferencias entre un arreglo de caracteres y el tipo de dato string en C++
3. Investiga la funcionalidad de los siguientes operadores y métodos de la clase string, organizarlos en una tabla con un ejemplo de uso:  
   **max\_size** / **compare** / **copy** / **c\_str** / **data** / **empty** / **erase** / **find** / **find\_last\_of** / **front** / **insert** / **replace** / **reserve** / **resize** / **substr** / **swap**
4. ¿Qué es un vector en C++ y cuáles son sus principales características? ¿Cómo se declara y se inicializa un vector en C++?
5. ¿Cómo se modifica el tamaño de un vector en C++?
6. ¿Qué sucede cuando se intenta acceder a un elemento fuera del rango del vector?
7. ¿Qué es la clase vector en C++ y qué funcionalidades proporciona?
8. Para los siguientes métodos de la clase vector busca un ejemplo de uso, organizando la información en una tabla:

* **size():** Obtiene el tamaño del vector.
* **capacity():** Obtiene la capacidad de almacenamiento actual del vector.
* **empty():** Comprueba si el vector está vacío.
* **push\_back(valor):** Añade un elemento al final del vector.
* **pop\_back():** Elimina el último elemento del vector.
* **front():** Devuelve una referencia al primer elemento del vector.
* **back():** Devuelve una referencia al último elemento del vector.
* **at(posición):** Devuelve una referencia al elemento en la posición especificada.
* **assign(inicio, fin):** Remplaza el contenido del vector con un rango de elementos.
* **erase(posición):** Elimina el elemento en la posición especificada.
* **erase(inicio, fin):** Elimina un rango de elementos del vector.
* **insert(posición, valor):** Inserta un elemento en la posición especificada.
* **insert(posición, inicio, fin):** Inserta un rango de elementos en la posición especificada.
* **clear():** Elimina todos los elementos del vector.
* **swap(otro\_vector):** Intercambia el contenido del vector con otro vector.

1. ¿Qué es una clase? ¿cuál es su estructura? ¿Qué es un objeto? ¿Cuál es su propósito?
2. En general, funciones y procedimientos en C++ son bastante similares ambos son bloques de código que permiten agrupar instrucciones y facilitar la reutilización del código. Sin embargo, la diferencia principal radica en su **resultado objetivo**:

**Funciones:** Son bloques de código diseñados para realizar una acción específica y devolver un resultado, El valor que devuelven dependiendo de las necesidades del programador.

**Por Ejemplo:**

**Int** Suma **(int** A**, int** B**)**

**{**

**Return** A**+**B**;**

**}**

**Procedimientos**: Son bloques de código que realizan una o varias acciones, pero no devuelven un valor directamente, Su objetivo es que se ejecute el conjunto de instrucciones, no necesariamente que se obtenga un resultado.

**Por Ejemplo**:

**Void** Mostrar\_mensaje **()**

**{**

**Cout<<” Bienvenido”;**

**}**

**Cuando usar cada uno:**

**Funciones:** se utilizan cuando necesitas que el bloque de código calcule y devuelva un valor, como en operaciones matemáticas o cuando quieres que algo te dé un resultado para utilizarlo después.

**Procedimientos:** se utilizan cuando solo quieres que el bloque de código realice acciones, por ejemplo, imprimir un mensaje, llenar un arreglo, o modificar algo que ya existe, sin necesidad de devolver un valor.

1. **1. Tipo de retorno**

Indica el tipo de dato que la función devolverá (por ejemplo, int, double, char, string, etc.).  
Si no devuelve nada, se usa void.

**Ejemplo: int** suma (int num1, int num2)

**2. Nombre de la función**

Es el identificador único que le das a la función para poder llamarla después.

**Ejemplo:** int **suma** (int num1, int num2)

**3. Lista de parámetros (o argumentos)**  
Son las variables que la función recibe cuando se llama.  
Se escriben entre paréntesis () y cada parámetro tiene un tipo y un nombre.

**Ejemplo:** intsuma (**int num1**, **int num2**)

**4. Cuerpo de la función**  
Está formado por un bloque de código delimitado por llaves { }, Contiene las instrucciones que se ejecutan cuando se llama a la función.

**Ejemplo:**

intsuma (int num1, int num2) **{**

**int resultado = num1 + num2;**

**return resultado;**

**}**

1. En C++, un arreglo de caracteres (char) es una secuencia de caracteres de tamaño fijo que termina con un carácter nulo ('\0'), En cambio el tipo de dato string permite manejar texto de forma más flexible, puede cambiar de tamaño y ofrece varios métodos incorporados para manipular cadenas de manera más fácil y segura.
2. Un vector en C++ es un tipo de dato incluida en la biblioteca <vector> que permite almacenar una variedad de elementos del mismo tipo, como si fuera un arreglo, pero con la ventaja de que puede cambiar de tamaño automáticamente durante la ejecución del programa.

**Características principales de los vectores:**

* **Tamaño dinámico**: se expanden o reducen automáticamente según se agreguen o eliminen elementos.
* **Acceso por índice**: se puede acceder a los elementos como en un arreglo ( vector[i] ).
* **Seguro y eficiente**: maneja memoria automáticamente y evita errores comunes de punteros.
* **Métodos útiles**: como push\_back(), size(), at(), clear(), insert(), etc.

1. La siguiente tabla muestra el metodo su descripción y ejemplo:

|  |  |  |
| --- | --- | --- |
| **Metodo** | **Descripción** | **Ejemplo de uso** |
| **max\_size()** | Devuelve el número máximo de caracteres que puede tener | cout << str.max\_size(); |
| **compare()** | Compara dos cadenas (0 si son iguales) | str1.compare(str2) |
| **copy()** | Copia caracteres a un array de char | str.copy(buffer, 4); |
| **c\_str()** | Devuelve un puntero const char\* con la cadena tipo C | puts(str.c\_str()); |
| **data()** | Similar a c\_str(), devuelve puntero a los datos internos | cout << str.data(); |
| **empty()** | Retorna true si la cadena está vacía | if (str.empty()) |
| **erase()** | Elimina parte de la cadena | str.erase(2, 3); |
| **find()** | Busca la primera aparición de un carácter o subcadena | str.find("lo") |
| **find\_last\_of()** | Busca la última aparición de caracteres dados | str.find\_last\_of("aeiou") |
| **front()** | Devuelve el primer carácter | char c = str.front(); |
| **insert()** | Inserta caracteres o subcadena en una posición | str.insert(3, "XYZ"); |
| **replace()** | Reemplaza parte de la cadena por otra | str.replace(0, 4, "Hola"); |
| **reserve()** | Reserva espacio en memoria | str.reserve(100); |
| **resize()** | Cambia el tamaño de la cadena | str.resize(5); |
| **substr()** | Devuelve una subcadena desde una posición | str.substr(0, 4) |
| **swap()** | Intercambia contenidos entre dos cadenas | str1.swap(str2); |

1. En general cuando hablamos de la biblioteca <vector> hay varias formas de hacerlo algunos ejemplos son:

* **push\_back ():** Agrega un elemento al final del vector (lo agranda):

**vector<int> v;**

**v.push\_back (10); // ahora tiene 1 elemento**

**v.push\_back (20); // ahora tiene 2**

* **pop\_back ():** Elimina el último elemento del vector (lo achica).

**vector<int> v = { 1, 2 };**

**v.pop\_back (); // ahora tiene 1 elemento**

* **resize(n):** Cambia directamente el tamaño del vector.

**//Si el nuevo tamaño es mayor, agrega elementos con valor por defecto (0 para int, "" para string, etc.).**

**//Si es menor, recorta el vector.**

**v.resize (5); // Agranda a 5 elementos**

**v.resize (2); // Achica a 2 elementos**

* **clear ():** Elimina todos los elementos (lo deja vacío).

**//Si el nuevo tamaño es mayor, agrega elementos con valor por defecto (0 para int, "" para string, etc.).**

**//Si es menor, recorta el vector.**

**v.resize (5); // Agranda a 5 elementos**

**v.resize (2); // Achica a 2 elementos**

1. Realmente depende de cómo accedas a la memoria del vector por ejemplo si usas **corchetes** no tira una advertencia de error perro accede a la memoria no válida, mostrara valores basura, causara fallos raros o incluso hacer que el programa se cuelgue más adelante, en cambio sí utilizas él **.at()** mostrara el error **“std::out\_of\_range”** esto ocurre por que si bien usar corchetes es mucho más rápido este mismo no verifica si esta accediendo a una parte de la memoria inexistente en cambio el .at() si y mostrar un error es por eso que a pesar de ser mas lento es mucho mas seguro que usar simplemente corchetes.
2. La clase vector en C++ es una estructura dinámica que almacena elementos del mismo tipo y puede cambiar de tamaño automáticamente. Permite acceder, agregar o eliminar elementos fácilmente Y manejar la memoria internamente. Algunos de sus métodos más útiles son push\_back (), pop\_back (), resize (), y at ().
3. La siguiente tabla:

|  |  |  |
| --- | --- | --- |
| **Método** | **Descripción** | **Ejemplo de uso** |
| **size()** | Obtiene el tamaño actual del vector | v.size (); // devuelve número de elementos |
| **capacity()** | Obtiene la capacidad reservada del vector | v.capacity (); // devuelve capacidad interna |
| **empty()** | Comprueba si el vector está vacío | v.empty (); // true si no tiene elementos |
| **push\_back(valor)** | Añade un elemento al final | v.push\_back (10); // agrega 10 al final |
| **pop\_back()** | Elimina el último elemento | v.pop\_back (); // elimina el último elemento |
| **front()** | Devuelve referencia al primer elemento | int x = v.front (); |
| **back()** | Devuelve referencia al último elemento | int x = v.back (); |
| **at(posición)** | Devuelve referencia al elemento en posición | int x = v.at (2); |
| **assign(inicio, fin)** | Reemplaza contenido con rango de elementos | v.assign(other.begin(), other.end()); |
| **erase(posición)** | Elimina elemento en posición especificada | v.erase(v.begin() + 1); // elimina segundo elemento |
| **erase(inicio, fin)** | Elimina rango de elementos | v.erase(v.begin(), v.begin() + 2); // elimina los dos primeros |
| **insert(posición, valor)** | Inserta elemento en posición especificada | v.insert(v.begin() + 1, 99); // inserta 99 en segunda posición |
| **insert(posición, inicio, fin)** | Inserta rango de elementos en posición | v.insert(v.begin(), other.begin(), other.end()); |
| **clear()** | Elimina todos los elementos | v.clear(); |
| **swap(otro\_vector)** | Intercambia contenido con otro vector | v.swap(other); |

1. **¿Qué es una clase? ¿cuál es su estructura? ¿Qué es un objeto? ¿Cuál es su propósito?**

* **Una Clase:** en C++ es un molde o plantilla que define un tipo de dato complejo, Describe las propiedades (atributos) y las acciones (métodos o funciones) que pueden tener los objetos creados a partir de ella.
* **La estructura básica de una clase incluye:**
* **Atributos:** variables que describen las características del objeto.
* **Métodos:** funciones que definen el comportamiento o acciones del objeto.
* **Modificadores de acceso:** como public, prívate o protected para controlar la visibilidad de atributos y métodos.
  + - **public:** accesible desde cualquier parte del programa.
    - **prívate:** solo accesible desde dentro de la clase.
    - **protected:** accesible desde la clase y sus clases derivadas (más usado con herencia).

**Ejemplo básico:**

**//Declaración de clase**

**class Persona {**

//**Modificador de acceso**

**public:**

**// Tipos de atributos:**

**string nombre;**

**int edad;**

**// método**

**void saludar() {**

**cout << "Hola, soy " << nombre << endl;**

**}**

**};**

* **Un objeto:** es una instancia de una clase, es decir, un ejemplar concreto que tiene sus propios valores para los atributos y puede usar los métodos definidos en la clase.

**Ejemplo conectado del anterior:**

**// p1 es un objeto de la clase Persona**

**Persona p1;**

**p1.nombre = "Ana";**

**p1.edad = 25;**

**// Muestra: Hola, soy Ana**

**p1.saludar();**

* **El propósito:** de las clases y objetos es organizar y modelar datos y comportamientos en programas de forma estructurada, reutilizable y cercana a cómo funcionan las cosas en el mundo real. Permiten manejar programas complejos dividiéndolos en partes pequeñas y manejables.