Rta 1: En C++, las funciones devuelven valores y se utilizan para cálculos y retornos de resultados. Los procedimientos, definidos con void, realizan acciones sin devolver valores y son útiles para efectos secundarios como impresión en pantalla o manipulación de datos sin necesidad de retorno. La elección depende de si se necesita devolver un valor (función) o simplemente ejecutar una acción (procedimiento).

Rta 2:

En C++, una función está compuesta por:

Tipo de Retorno: Especifica qué tipo de dato devuelve la función al final de su ejecución.

Nombre: Identificador único que se usa para llamar a la función desde otros lugares del programa.

Parámetros: Variables opcionales que la función puede recibir para realizar su tarea.

Cuerpo: Contiene las instrucciones que definen qué hace la función.

Instrucción de Retorno: Devuelve un valor al punto desde donde fue llamada, si es necesario.

Rta 3:
Arreglo de Caracteres (`char[]`): Estructura estática con tamaño fijo para almacenar secuencias de caracteres. Requiere gestión manual de memoria y utiliza el carácter nulo (`'\0'`) para marcar el final de la cadena.
Tipo de Dato `string`: Tipo dinámico que representa cadenas de caracteres en C++. Proporciona gestión automática de memoria, métodos integrados para manipulación de cadenas y es parte de la biblioteca estándar de C++.

Rta 4:

| Metodo/OperadorD | Descripción | Ejemplo de uso |
|---|---|---|
| max_size | Devuelve el tamaño máximo que puede tener la cadena. | string str = "Hola"; cout << str.max_size(); |
| compare | Compara dos cadenas. | string str1 = "Hola"; string str2 = "Mundo"; int res = str1.compare(str2); |
| copy | Copia una parte de la cadena a un array de caracteres. | string str = "Hola Mundo"; char arr[5]; str.copy(arr, 4, 0); arr[4] = '\0'; cout << arr; |
| c_str | Devuelve un puntero a un array que contiene una versión terminada en nulo de la cadena. | string str = "Hola"; const char* cstr = str.c_str(); cout << cstr; |
| data | Devuelve un puntero a un array que contiene la cadena. | string str = "Hola"; const char* data = str.data(); cout << data; |

| | | |
|---|---|---|
| empty | Comprueba si la cadena está vacía. | string str = ""; bool isEmpty = str.empty(); cout << isEmpty; |
| erase | Elimina una parte de la cadena. | string str = "Hola Mundo"; str.erase(5, 5); cout << str; |
| find | Encuentra la primera aparición de una subcadena. | string str = "Hola Mundo"; size_t pos = str.find("Mundo"); cout << pos; |
| find_last_of | Encuentra la última aparición de cualquiera de los caracteres especificados. | string str = "Hola Mundo"; size_t pos = str.find_last_of("o"); cout << pos; |
| front | Devuelve una referencia al primer carácter de la cadena. | string str = "Hola"; char& ch = str.front(); cout << ch; |
| insert | Inserta una subcadena en la cadena. | string str = "Hola Mundo"; str.insert(5, "querido "); cout << str; |
| replace | Reemplaza una parte de la cadena con otra cadena. | string str = "Hola Mundo"; str.replace(5, 5, "amigo"); cout << str; |
| reserve | Reserva capacidad en la cadena. | string str; str.reserve(100); cout << str.capacity(); |
| resize | Cambia el tamaño de la cadena. | string str = "Hola"; str.resize(10, 'x'); cout << str; |
| substr | Devuelve una subcadena. | string str = "Hola Mundo"; string sub = str.substr(5, 5); cout << sub; |
| swap | Intercambia el contenido de dos cadenas. | string str1 = "Hola"; string str2 = "Mundo"; str1.swap(str2); cout << str1 << " " << str2; |

Rta 5:

Un vector en C++ es una estructura de datos dinámica proporcionada por la biblioteca estándar de C++. Actúa como un array dinámico, permitiendo almacenar una colección de elementos cuyo tamaño puede cambiar automáticamente cuando se añaden o eliminan elementos.

## Principales características

1. Dinámico: Cambio automático de tamaño.
2. Acceso aleatorio**:** Acceso directo a cualquier elemento mediante un índice.
3. Eficiencia: Inserción y eliminación eficientes al final.
4. Seguridad: Gestión automática de memoria.
5. Integración con STL: Compatible con algoritmos y funcionalidades de STL.

Declaracion e Inicializacion de un Vector:

```
#include <vector>
using namespace std;

int main() {
   // Declaración de un vector vacío de enteros
   vector<int> vec1;

   // Inicialización con valores específicos
   vector<int> vec2 = {1, 2, 3, 4, 5};

   // Inicialización con tamaño y valor por defecto
   vector<int> vec3(10, 0); // 10 elementos, todos a 0

   // Inicialización con otro vector
   vector<int> vec4(vec2); // copia los elementos de vec2

   return 0;
}
```

Rta 6:

En C++ se puede modificar el tamaño de un vector usando los métodos resize, push_back y pop_back de la clase std::vector.

Con Resize:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
```

```
    vector<int> vec = {1, 2, 3, 4, 5};

    // Cambiar el tamaño del vector a 3 (se eliminan elementos al final)
    vec.resize(3);
    // Ahora vec contiene: 1, 2, 3

    // Cambiar el tamaño del vector a 6 (se agregan elementos inicializados a 0)
    vec.resize(6);
    // Ahora vec contiene 1, 2, 3, 0, 0, 0

    for (int i : vec) {
        cout << i << " ";
    }

    return 0;
}
```

Con push_back:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> vec = {1, 2, 3};

    // Añadir un elemento al final del vector
    vec.push_back(4);
    // Ahora vec contiene: 1, 2, 3, 4
```

```
    for (int i : vec) {
        cout << i << " ";
    }

    return 0;
}
```

Con pop_back:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> vec = {1, 2, 3, 4};

    // Eliminar el último elemento del vector
    vec.pop_back();
    // Ahora vec contiene: 1, 2, 3

    for (int i : vec) {
        cout << i << " ";
    }

    return 0;
}
```

Rta 7:

Acceso con []:

Si se intenta acceder a un elemento fuera del rango usando el operador [], el comportamiento no está definido por el estándar de C++. Esto significa que el programa puede comportarse de manera impredecible. Puede devolver un valor incorrecto, sobrescribir datos importantes en memoria (corrupción de memoria), o incluso provocar un fallo del programa (segmentation fault) que termina la ejecución abruptamente.

Acceso con at():

Si se usa el método at() para acceder a un elemento fuera del rango, se va a lanzar una excepción std::out_of_range. Esto ocurre porque at() realiza una comprobación de límites y detecta que estás intentando acceder a un índice que no existe en el vector. Esta excepción te permite manejar el error de manera controlada en tu programa, en lugar de dejar que ocurra un comportamiento indefinido.

Rta 8: La clase vector en C++ es parte de la biblioteca estándar (std::vector) y representa una estructura de datos que proporciona un contenedor dinámico de elementos. Es esencialmente un array dinámico que permite almacenar una secuencia de elementos de manera eficiente, con capacidades de gestión automática de memoria y operaciones optimizadas para inserción, eliminación y acceso a elementos.

Funcionalidades:

Dinamismo: Capacidad de cambiar de tamaño dinámicamente.
Acceso Aleatorio: Acceso eficiente a elementos mediante índices.
Eficiencia: Operaciones optimizadas de inserción, eliminación y acceso.
Gestión Automática de Memoria: Evita desbordamientos y fugas de memoria.
Compatibilidad con STL: Integración con algoritmos y funciones estándar de C++.

Rta 9:

| Metodo | Descripcion | Ejemplo de uso |
|---|---|---|
| size() | Obtiene el tamaño del vector. | vector<int> v = {1, 2, 3}; <br> int size = v.size(); |
| capacity() | Obtiene la capacidad de almacenamiento actual del vector. | int capacity = v.capacity(); |
| empty() | Comprueba si el vector está vacío. | bool is_empty = v.empty(); |
| push_back(valor) | Añade un elemento al final del vector. | v.push_back(4); |

| | | |
|---|---|---|
| pop_back() | Elimina el último elemento del vector. | v.pop_back(); |
| front() | Devuelve una referencia al primer elemento del vector. | int& first_element = v.front(); |
| back() | Devuelve una referencia al último elemento del vector. | int& last_element = v.back(); |
| at(posición) | Devuelve una referencia al elemento en la posición especificada. | int& element = v.at(2); |
| assign(inicio, fin) | Reemplaza el contenido del vector con un rango de elementos. | `vector<int> new_elements = {5, 6};` <br> `v.assign(new_elements.begin(), new_elements.end());` |
| erase(posición) | Elimina el elemento en la posición especificada. | v.erase(v.begin() + 1); |
| erase(inicio, fin) | Elimina un rango de elementos del vector. | v.erase(v.begin() + 1, v.begin() + 3); |
| insert(posición, valor) | Inserta un elemento en la posición especificada. | v.insert(v.begin() + 2, 7); |
| insert(posición, inicio, fin) | Inserta un rango de elementos en la posición especificada. | `vector<int> new_elements = {8, 9};` <br> `v.insert(v.begin() + 2, new_elements.begin(), new_elements.end());` |
| clear() | Elimina todos los elementos del vector. | v.clear(); |
| swap(otro_vector) | Intercambia el contenido del vector con otro vector. | `vector<int> another_vector = {10, 11};` <br> `v.swap(another_vector);` |

Rta 10: Una clase en programación es una estructura que define el comportamiento y las propiedades de un tipo de objeto. Proporciona un modelo a partir del cual se pueden crear objetos concretos.

La estructura básica de una clase C++ es:

class NombreDeLaClase {

```
public:
    // Declaración de variables miembro (propiedades)
    TipoDeDato variableMiembro1;
    TipoDeDato variableMiembro2;
    // ...

    // Declaración de métodos (funciones miembro)
    TipoDeRetorno metodo1(TipoDeParametro parametro1);
    TipoDeRetorno metodo2(TipoDeParametro parametro2);
    // ...

private:
    // Opcional: variables y métodos privados (accesibles solo desde dentro de la clase)
};
```

Variables miembro: Son las propiedades o datos que pertenecen a cada objeto creado a partir de la clase.
Métodos: Son funciones que definen el comportamiento de los objetos creados a partir de la clase.
Accesibilidad: Los miembros pueden ser públicos (public), accesibles desde fuera de la clase; privados (private), accesibles solo desde dentro de la clase; o protegidos (protected), accesibles por la clase y sus subclases.

Un objeto es una instancia concreta de una clase. Se crea utilizando la plantilla proporcionada por la clase y tiene estado (datos almacenados en sus variables miembro) y comportamiento (definido por los métodos de la clase). Los objetos permiten modelar entidades del mundo real como usuarios, productos, vehículos, etc., en el contexto de la programación orientada a objetos.

El propósito principal de las clases y los objetos es facilitar la reutilización de código y la organización de datos y funciones relacionadas. Las clases permiten abstraer características comunes de varios objetos en un solo lugar (la clase), promoviendo así la modularidad y la estructura ordenada del código.