**RESUMEN TEORICO PGE**

**Programación Genérica:** Generalizar las funciones para que puedan ser utilizadas en varios casos.

*Ventajas:*

* Reutilización de código.
* Fácil mantenimiento de código.
* Nos concentramos más en la lógica del sistema.

*Desventajas:*

* Pérdida de interés para los amantes de la programación a bajo nivel.
* En C++ requiere el uso de Templates y sobrecarga de operadores, que es dificultoso y poco legible.

**Plantillas:**

* Separa la estructura del contenido.
* Permite construir un diseño predefinido
* Facilita el trabajo de realizar copias idénticas de la estructura.
* Utilización de tipos como parámetros
* Teniendo la función ordena( v ). Dependerá del tipo de v para generar la función.
* Mecanismo que permite usar un tipo como parámetro en una clase o función.
* Clases genéricas: Es un “constructor” (o creador) de clases (no confundir con el constructor de una clase).

**Clase genérica Listado:**

* Una plantilla genera la definición de una clase.
* A la instancia concreta de la clase generada, se la denomina especialización.
* La definición de la clase genérica es la siguiente:

*template< class T > class NombreClase {*

* La definición de métodos off-line tiene la siguiente sintaxis:

*template< class T > bool NombreClase < T >::nombreFuncion( T variable ) {*

**Herencia con clases genéricas:**

La herencia con clase genérica tiene la siguiente sintaxis:

*template< class T > class NombreClase : public NombreClaseHereda< T > {*

**Sobrecarga de operadores:**

Sirve para poder usar operadores y que realicen ciertas acciones dentro de una clase.

La sintaxis para sobrecargar operadores es:

*datoreturn operator+( datorecibe ){*

Ejemplo:

Declaración: *bool operator<( Cliente otroCliente )*  *{*

Uso: *cliente1 < cliente2;*

**Static:**

***Variables estáticas:***

* Al salir de su ámbito no pierde su valor
* Sólo son conocidas dentro de su ámbito (pero igual "recuerdan su valor")
* Se inicializan sólo la primera vez

***Miembros y métodos estáticos:***

* Para cada instancia de una clase existe una copia de los miembros no-estáticos.
* Pero hay una única copia de los estáticos para todas las instancias.
* Pueden ser accedidas sin referencia a ninguna instancia concreta de la clase.
* Los miembros estáticos no dependen de ninguna instancia para su existencia.
* Existen incluso antes que la primera instancia de una clase.
* Los métodos estáticos sólo pueden acceder a miembros estáticos.

***El constructor y miembros estáticos:***

* El constructor puede modificar los valores de los miembros estáticos, pero no inicializarlos.
* Los miembros estáticos pueden ser accedidos con :: con la notación Clase::miembro.
* Los miembros estáticos en el constructor si o si se tienen que poner como:

*D() : x( 20 ) {*

*y = 10; //y es método estático y x no lo es.*

*}*

**Copiar objetos:**

***Operador de asignación:***

* El operador = supone la asignación del operando de la derecha en el izquierdo.
* Ambos objetos deben ser creados previamente.
* La sintaxis para definir el operador de asignación es igual al uso de sobrecarga de operadores:

*Persona & operator=( const Persona & persona );*

* Por otro lado, cuando se crea un nuevo objeto no implica ninguna asignación, sino que se invoca a un constructor especial (constructor copia). Ejemplo:

*Persona p1 = p2; //se llama al constructor copia*

***Constructor copia:***

* Es un constructor que recibe como parámetro un objeto de la misma clase. Ejemplo:

*Persona( const Persona & persona );*

* Se invoca al constructor copia en los siguientes momentos:
  + Cuando se pasa un objeto como argumento
  + Cuando algún método o función devuelve un objeto
  + Cuando se crea un objeto mediante asignación
  + Cuando se lanza una excepción
* El compilador crea un constructor copia oficial cuando no se define explícitamente. Este constructor realiza un clon exacto miembro a miembro.

**Captura de eventos con eventFilter:**

* Sirve para detectar cualquier evento que se produzca en el programa.
* Se devuelve true para indicar que este evento ya lo controlo el eventFilter y no es necesario que se propague para que alguien más lo controle.

**Creación y uso de librerías dinámicas:**

Diagram

Description automatically generated

**Singleton:**

A picture containing text

Description automatically generated

**Función callback:**

* Función que se llama a través de un puntero a función.
* Se puede utilizar como parámetro de otra función.
* Cuando la función que recibe este puntero a función hace uso de este, se dice que hace una retrollamada (callback).

***Declaraciones de punteros a funciones:***

*void ( \* puntero\_funcion )();*

*// puntero a una función sin parámetros que devuelve void.*

*void ( \* puntero\_funcion )( int );*

*// puntero a función que recibe int y devuelve void.*

*int ( \* puntero\_funcion )( int, char );*

*// acepta int y char y devuelve un int.*

*int \* ( \* puntero\_funcion )();*

*// puntero a función, sin argumentos y devuelve puntero a int*

void ( C::\* puntero\_funcion )( int );

// puntero a método de la clase C

* Antes de usar un puntero a función es necesario definirlo (asignarle un valor).
* El valor es la dirección de memoria donde inicia una función concreta.

*puntero\_funcion = &función\_sumar; // Asigna al puntero la dirección de memoria de función\_sumar*

* Para llamar a la función atravez del puntero creo que se hace así:

*( \* puntero\_funcion )(); //acá se llama a la función a la que apunta*

**Excepciones:**

Text, letter

Description automatically generated