Declaración vs. Definición

Declaración Interfaz de un módulo. Definición del QUE Archivos .h (header)

Define es que es lo que hace: IMPORTA AL QUE LO USA

H: void CalcularDiametro(float radio, CIRCULO circulo);, se tiene que cumplir esto, este acuerdo que no se puede romper

Se arma primero y decimos que hacemos

Definición Implementación de un módulo cumpliendo la interfaz declarada. Definición del COMO Archivos .cpp

IMPORTA AL QUE CODEA

Se arma segundo y decimos como lo hacemos

C: void CalcularDiametro(float radio, CIRCULO circulo){

……………………………………………

}

Mientras tenga el .h mi main lo va compilar

TODO CODIGO QUE SE PUEDE REUTILIZAR, SE DEBE HACER UNA LIBRERÍA .H y .CPP TAMBIEN POR TEMA DE MANTENIMIENTO Y PAR AHORRAR LINES DE CODIGO Y SER ORDENADO

No es obligatorio usar las funciones del .h al main, puedo usarlo en otro .h o .c

SI ROMPO EL .h TODO EL CPP DEJA DE FUNCIONAR

Podemos tener muchos .h para dividir el trabajo, vos pibe, triangulo, vos circulo.

Mientras tenga el .h mi main lo va compilar

El .cpp incluye al .h

COMPILACION

Pre procesamiento

#include“.h”

#ifndef PI=3.14

#define PI=3.14

PUNTEROS

UN PUNTERO GUARDA DIRECCIONES

Esto para no crearlo de nuevo o por si no está creado. Esto se hace al principio para no usar 2 veces distintas variables

SI VOS PEDIS MEMORIA MANUALMENTE CON NEW, ES OBLIGACION DEL PROGRAMADOR HACER UN DELETE

LA MAQUINA SE LIBERA REINICIANDO LA MAQUINA, AUNQUE ESTO NO ES UNA BUENA PRACTICA

AL MOMENTO DE NO USARLO MAS, HAY QUE LIBERARLO

Referenciar

Significa tomar la dirección de una variable existente (usando &) para establecer una variable de puntero. Para que sea válido, un puntero debe establecerse en la dirección de una variable del mismo tipo que el puntero, sin el asterisco:

Int c1;//1234

Int \*p1;//2345

C1=5;

P1=&c1; // ASI GUARDAMOS (referenciamos) la dirección al puntero //1234

//p1 es un puntero, este puntero tiene la dirección de memoria de c1

//&DIRECCION DE MEMORIA

Ahora bien, y si quiero acceder al valor del p1?

TENGO QUE DISFERENCIAR

Significa usar el operador \* (carácter de asterisco) para acceder al valor almacenado en un puntero.

Int n1;

n1= \*p1; // a n1 le doy el contenido de p1, el contenido que está en la dirección del puntero

LA ABSTRACCION EVITA QUE SI ALGUIEN CAMBIA EL .CPP NO TENGA QUE CAMBIAR EL HEADER, LA IMPLEMENTACION NO SE DEBE TOCAR

ABSTRACCION ES CREAR LAS FUNCIONES DEL PUNTO H. Supon

La abstracción es una herramienta que clasifica aquellos atributos del struct del TDA que de verdad necesito. Esta abstracción se trae consigo solo aquellos campos que voy a utilizar en el código y descarta aquellos campos innecesarios a estudiar (codear) en el TDA.

EJEMPLO:

Struct Alumno{

Int edad; //SI

char nombre [20]; //SI

int dni; //SI

char seriefavorita[30]; //NO

};

En este caso los primeros 3 atributos son necesarios si tengo que codear, obvio faltan más, pero la idea esa solo poner aquellos que me importan para el sistema.

Por otra parte tenemos el char seriefavorita, es un campo innecesario si tengo que usarlo en el sistema. Ya que no me aporta nada, y es solo irrelevante para mí uso.

El proceso de abstracción presenta dos aspectos complementarios:

1. Destacar los aspectos relevantes del objeto.
2. Ignorar los aspectos irrelevantes del mismo (la irrelevancia depende del nivel de abstracción, ya que si se pasa a niveles más concretos, es posible que ciertos aspectos pasen a ser relevantes).

Permite al usuario (otra persona que va tocar el código) a manejar las funciones que necesita sin tener que saberse el código, el solo se maneja con esas funciones. Nos permite también reutilizar esos métodos que abstraemos para que el usuario se le haga más fácil, cómodo y no tenga que crear métodos repetidos.