***TP N° 1: Compilación y Linkedición***

**Nota:**

Muchas veces al compilar un código, aparece un mensaje de error del tipo ***“Parse error before...”***. Este mensaje indica que existe un error de sintaxis, como por ejemplo la falta de un “;” en la instrucción anterior a la línea indicada como error.

A diferencia de Z80, donde cada fin de línea indicaba fin de instrucción, en C una instrucción puede ocupar más de una línea, por lo que es necesario indicar explícitamente el fin de una instrucción, y esto se hace con el ";"

***Ejercicio 1***

int

main(void)

{

/\* Esto es un primer comentario

**\*/  
1 \\* Que buen comentario! \*\** 



**2 /\* Es valido /\* o no \*/ este comentario? \*/**

**3 / \* Esto es un ultimo comentario \*/**

**4 return 0**

}

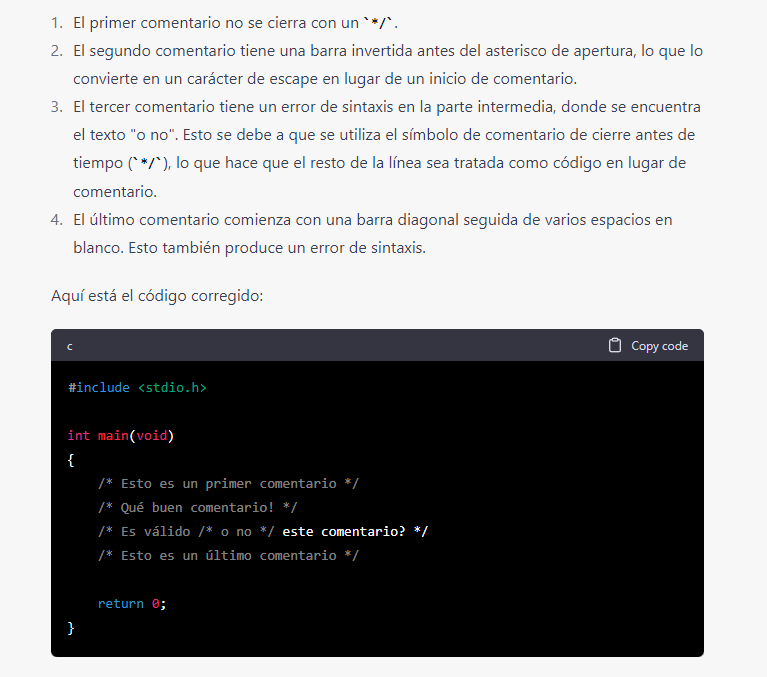
En la línea indicada como 1, las dos barras son incorrectas. Por lo tanto el compilador interpreta que no es un comentario y supone que la palabra “Que” corresponde a una variable que no está declarada.

En la línea 2, aparece un comentario anidado (que no están permitidos), entonces el compilador ignora el segundo “/\*” y considera que el comentario termina en el “\*/” que está después de la palabra “no” y que “este” corresponde a una variable no declarada.

En la línea 3, el error es dejar espacios entre la barra y el asterisco.

Por último, en la línea 4, falta el “;” después del cero.

Copiamos debajo la respuesta que da chatGPT. Conclusión: si bien chatGPT puede ser útil, no es muy confiable, sobre todo en lenguajes de programación de no muy alto nivel, suele ser más confiable en lenguajes como Python, JavaScript, Dart, etc



***Ejercicio 2***

int

main( void /\* programa que no hace nada \*/)

{

return 0

}

En este caso el error es la falta del ‘;’ después de “return 0”.

***Ejercicio 3***

#include <stdio.h>

**int**

**1 Main(void)**

{

**2 printf ("Estamos escribiendo un mensaje \n")**

**3 Return = 0**

}

En la línea 1 aparece “Main”. Como el lenguaje C diferencia mayúsculas de minúsculas entonces no reconoce a esta función como “main”, y supone que puede ser una función propia del usuario. Por lo tanto si este fuera el único error del programa, no generaría errores de compilación (ya que lo interpreta como una función de librería) y sí generaría errores de linkedición ya que no existe la función “main” (con minúscula).

En la línea 2 falta el “;”.

En la línea 3 “Return” no es reconocido como palabra reservada por tener la inicial en mayúscula y el compilador lo interpreta como una variable no declarada. El signo “=” no corresponde y además falta el “;” final.

***Ejercicio 4***

**1** #include <stdioh>

**2**

**3 int**

**4 main[void]**

5 {

**6 int a, b = c = 5**

**7 a = b + c**

**8**

**9 return: 0**

10 }

En la línea 1 el nombre correcto es <stdio.h>

En las líneas 6, 7 y 9 falta el punto y coma al final de la sentencia

En la línea 4 se colocaron corchetes en lugar de paréntesis después de “main”.

En la línea 6 debería decir “int a, b=5, c=5;” ya que en la inicialización de la variable b se utiliza la variable c que no está definida.

En la línea 9, no corresponde colocar los dos puntos.

***Ejercicio 5***

**1 # include <stdio.h>**

int

main(void)

{

int i, j, max;

i = j = 2;

**2 max = (i>j? I : j ) ;**

**3 end**

}

En la línea 1, si bien no es un error conviene no dejar espacios entre el “#” y el “include”.

En la línea 2, la “I” se considera una variable no declarada (la que fue declarada es la ***i*** minúscula).

En la línea 3, aparece un “end” que no es una palabra del lenguaje.

Falta agregar “return 0;”.

***Ejercicio 6***

Si **solamente** se compila**,** se crea un archivo con código objeto (**tp1\_06.o**). Como los archivos objeto NO son ejecutables, se generan automáticamente con permisos de lectura y escritura.

Si se realiza la linkedición, se puede ver que el archivo resultante tiene permisos de ejecución. Los archivo ejecutables en Unix/Linux no llevan ninguna extensión específica, se reconocen por tener el permiso de ejecución. Si a un archivo no ejecutable se le cambian los atributos, autorizando la ejecución, el sistema operativo intentará ejecutarlo y el resultado es impredecible.

***Ejercicio 7***

El proceso falla en la linkedición ya que el código no tiene la función “main”. Al compilar no se detectan errores.

***Ejercicio 8***

Falla la compilación porque la función “main” está duplicada, y un programa no puede tener dos funciones con el mismo nombre (ni siquiera “main”).

***Ejercicio 9***

Al compilar cada programa, aunque cada uno tiene una función “main”, no se producen errores porque la compilación de cada archivo es un proceso independiente.

Al linkeditarlos juntos, surge como error que ambos tienen una función “main”.

***Ejercicio 10***

El programa no tiene errores de compilación ni de linkedición.

* Ejecutando **tp1\_10**  no se nota ningún efecto especial.
* En este caso **tp1\_10 | cat** , al enviar la salida estándar como entrada del comando ***cat***, el mensaje ***“Ingrese una letra”*** del programa ***tp1\_10*** no aparece por pantalla, sino que lo recibe el ***cat***. Es por eso que el programa se queda esperando una letra (sin el mensaje que la pide) y luego de ingresarla emite un mensaje (que tampoco se visualiza ya que no va a salida estándar sino a la entrada estándar de ***cat***) y termina. Al terminar ***tp1\_10,*** se ejecuta ***cat*** mostrando por pantalla (salida estándar) todo el texto generado por ***tp1\_10***.
* Ejecutando **tp1\_10 > out.txt** no veremos ninguna salida en la pantalla pues la salida estándar (por defecto la pantalla) fue redirigida al archivo out.txt. Abrir el archivo out.txt y verificar que contiene lo que se vería por pantalla.

***Ejercicio 11***

El archivo se incluye a sí mismo, lo que provocaría un bucle infinito de inclusiones. Depende del preprocesador puede detectar el error o mostrar un ciclo infinito de mensajes de error, hasta que se interrumpe el proceso, por ejemplo con CTRL-C

Una vez corregido el error (eliminado la línea con el #include) los resultados son:

* Al ejecutar tp1\_11 < tp1\_11.txt se redirige la entrada estándar, por lo que la función getchar no se quedará esperando que el usuario ingrese un caracter sino que será tomado del archivo tp01\_11.txt e imprimirá lo siguiente

| Ingrese una letra:  a es una letra minúscula |
| --- |

* Al ejecutar tp1\_11 < tp1\_11.txt > out.txt se creará el archivo out.txt con el contenido mostrado en el cuadro de arriba

***Ejercicio 12***

Al preprocesar el programa, se puede ver cómo desaparece la directiva “**#include <tp1\_13.h>**” y en su lugar aparece el contenido del archivo **tp1\_13.h**. Este archivo también contiene las directivas de definición de las constantes simbólicas TRUE y FALSE, pero éstas no aparecen en el código ya que sus ocurrencias son reemplazadas por sus valores asociados.

***Ejercicio 13***

En el primer ejemplo se define una constante de tipo ***long*** fuera de rango. Como dicha constante **no es utilizada** en el programa el compilador **no detecta ningún error**. En el segundo ejemplo la constante es usada para inicializar una variable, y en ese caso el compilador genera un “warning”, que no es un error (por lo que puede generar el código objeto) pero indica que hay algo que puede ser mal interpretado o que puede producir un resultado diferente al esperado. En este caso el warning indica que el valor de la constante está fuera de rango, por lo que la variable no va a quedar correctamente inicializada pues va a “ciclar”.

***Ejercicio 14***

Este es un ejemplo similar al anterior. Se define una constante octal (porque comienza con 0) y uno de sus dígitos es nueve, que es un error. En el primer caso, al no usarla, no provoca error.

En el segundo caso, a diferencia del ejercicio anterior, se genera un **error de compilación** ya que no puede interpretar el valor de la constante.

***Ejercicio 15***

El código falla en compilación por el uso de la palabra reservada auto como nombre de una variable. El calificador auto está relacionado con el modo de almacenamiento de una variable y no será utilizado en el transcurso de la materia. Este ejemplo invita a tener presente las palabras reservadas del lenguaje y no utilizarlas como nombres de variables.

***Ejercicio 16***

1. Falla en **preprocesamiento** por no encontrar el archivo de encabezado
2. Falla en **compilación** por redefinir una función en un mismo archivo fuente. Esto ocurre con cualquier función, no sólo con la función main.
3. Falla en **linkedición**, ya que el linkeditor encuentra más de una función main (el ejecutable tiene más de un punto de inicio).
4. Falla en **linkedición**, ya que el linkeditor no encuentra una función main en el conjunto de archivos fuente (el ejecutable no tiene un punto de inicio).
5. Falla en **compliación** por utilizar una palabra reservada como nombre de variable.
6. Falla en **compilación** por redefinir una variable dentro de un bloque o función.
7. **No falla**. Se obtiene una advertencia o warning (*control reaches end of non-void function*) por no incluir un valor de retorno entero a una función (en este caso main) que dice retornar entero en su prototipo (**int** main()).
8. Falla en **compilación** por redefinir una función dentro de un mismo archivo fuente.