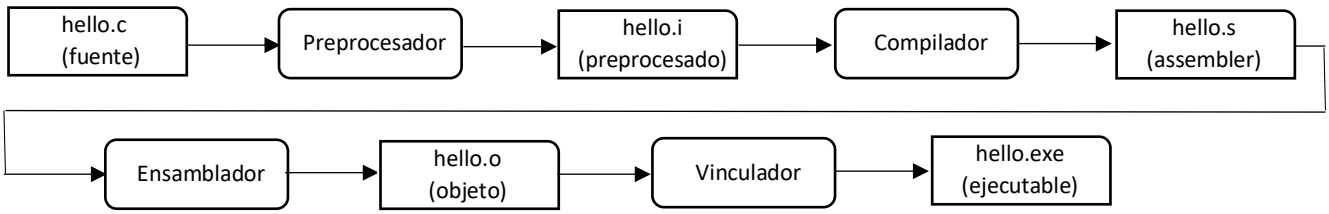
Trabajo Práctico 5

Fases de la Traducción y Errores

1. **Objetivos:**
   Este trabajo tiene como objetivo identificar las fases del proceso de traducción o Build y los posibles errores asociados a cada fase. Para lograr esa identificación se ejecutan las fases de traducción una a una, se detectan y corrigen errores, y se registran las conclusiones en readme.md. No es un trabajo de desarrollo; es más, el programa que usamos como ejemplo es simple, similar a hello.c pero con errores que se deben corregir. La complejidad está en la identificación y comprensión de las etapas y sus productos.



2. **Temas:**
   - Fases de traducción.
   - Preprocesamiento.
   - Compilación.
   - Ensamblado.
   - Vinculación (Link).
   - Errores en cada fase.
   - Compilación separada.

3. **Tareas:**
   1. La primera tarea es investigar las funcionalidades y opciones que su compilador presenta para limitar el inicio y fin de las fases de traducción.
   2. La siguiente tarea es poner en uso lo que se encontró. Para eso se debe transcribir al readme.md cada comando ejecutado y su resultado o error correspondiente a la siguiente secuencia de pasos. También en readme.md se vuelcan las conclusiones y se resuelven los puntos solicitados. Para claridad, mantener en readme.md la misma numeración de la secuencia de pasos.

Secuencia de Pasos:

Se parte de un archivo fuente que es corregido y refinado en sucesivos pasos. Es importante no saltearse pasos para mantener la correlación, ya que el estado dejado por el paso anterior es necesario para el siguiente.

1. *Preprocesador*
   a. Escribir hello2.c, que es una variante de hello.c:

```c
#include <stdio.h>
int/*medio*/main(void){
int i=42;
 prontf("La respuesta es %d\n");
```

b. Preprocesar hello2.c, no compilar, y generar hello2.i.
Analizar su contenido. ¿Qué conclusiones saca?

c. Escribir hello3.c, una nueva variante:

```c
int printf(const char * restrict s, ...);

int main(void){
  int i=42;
  prontf("La respuesta es %d\n");
```

d. Investigar e indicar la semántica de la primera línea.

e. Preprocesar hello3.c, no compilar, y generar hello3.i.
Buscar diferencias entre hello3.c y hello3.i.

2. *Compilación*

a. Compilar el resultado y generar hello3.s, no ensamblar.

b. Corregir solo los errores, no los warnings, en el nuevo archivo hello4.c y empezar de nuevo, generar hello4.s, no ensamblar.

c. Leer hello4.s, investigar sobre lenguaje ensamblador, e indicar de forma sintética cual es el objetivo de ese código.

d. Ensamblar hello4.s en hello4.o, no vincular.

3. *Vinculación*

a. Vincular hello4.o con la biblioteca estándar y generar el ejecutable.

b. Corregir en hello5.c y generar el ejecutable. Solo corregir lo necesario para que vincule.

c. Ejecutar y analizar el resultado.

4. *Corrección de Bug*

a. Corregir en hello6.c y empezar de nuevo; verificar que funciona como se espera.

5. *Remoción de prototipo*

a. Escribir hello7.c, una nueva variante:

```c
int main(void){
    int i=42;
    printf("La respuesta es %d\n", i);
}
```

b. Explicar por qué funciona.

6. *Compilación Separada: Contratos y Módulos*

a. Escribir studio1.c (sí, studio1, no stdio) y hello8.c.

La unidad de traducción studio1.c tiene una implementación de la función prontf, que es solo un wrappwer[1] de la función estándar printf:

```c
void prontf(const char* s, int i){
    printf("La respuesta es %d\n", i);
}
```

La unidad de traducción hello8.c, muy similar a hello4.c, invoca a prontf, pero no incluye ningún header.

```c
int main(void){
    int i=42;
    prontf("La respuesta es %d\n", i);
}
```

b. Investigar como en su entorno de desarrollo puede generar un programa ejecutable que se base en las dos unidades de traducción (i.e., archivos fuente, archivos con extensión .c). Luego generar ese ejecutable y probarlo.

c. Responder ¿qué ocurre si eliminamos o agregamos argumentos a la invocación de prontf? Justifique.

d. Revisitar el punto anterior, esta vez utilizando un contrato de interfaz en un archivo header.

i. Escribir el contrato en studio.h

```c
#ifndef STUDIO_H_INCLUDED
#define STUDIO_H_INCLUDED

void prontf(const char*, int);

#endif // STUDIO_H_INCLUDED
```

ii. Escribir hello9.c, un cliente que sí incluye el contrato.

```c
#include "studio.h" // Interfaz que importa

int main(void){
    int i=42;
    prontf("La respuesta es %d\n", i);
}
```

iii. Escribir studio2.c, el proveedor que sí incluye el contrato.

```c
#include "studio.h" // Interfaz que exporta
#include <stdio.h> // Interfaz que importa

void prontf(const char* s, int i){
        printf("La respuesta es %d\n", i);
}
```

iv. Responder: ¿Qué ventaja da incluir el contrato en los clientes y en el proveedor.

[1] https://en.wikipedia.org/wiki/Wrapper_function

4. **Restricciones**

   El programa ejemplo debe enviar por stdout la frase La respuesta es 42, el valor 42 debe surgir de una variable.