Capítulo 1

Introducción a la Programación II

1. Paso de parámetros con línea de comandos

Un argumento de línea de comandos es aquella información que sigue al nombre del programa al ejecutarlo. Utilizado por ejemplo cuando se necesita una ejecución con los parámetros que indique el usuario.

int main (int argc, char * argv[ ])

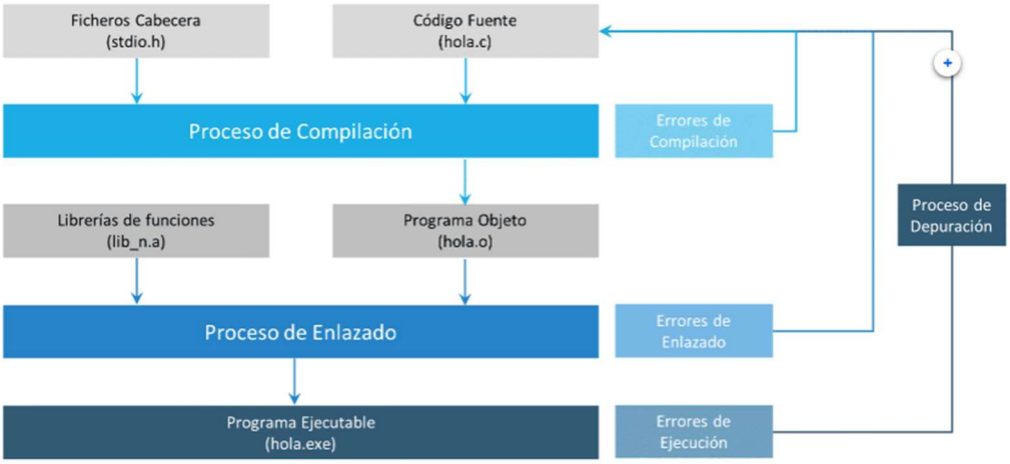
Estos argumentos se pasan cuando ejecutamos el fichero, contamos con un argc que contiene el número de argumentos (contador), que como mínimo siempre valdrá 1 ya que el programa cuenta como primero. Por su parte el parámetro argv es un puntero a un array de caracteres.

$ ./media_alturas.exe 157 180

Para poder visualizar el concepto anterior, ahora podemos crear un programa que muestre los argumentos ingresados por el usuario a través de la línea de comando:

```
#include<stdio.h>

int main (int argc, char * argv[]) {

  int i;

  printf("Argumentos de la linea de ordenes\n\n");


  for(i=0; i<argc; i++)

    printf("El argumento %d es %s\n", i, argv[i]);

    printf("\n\nTerminación normal del programa.\n");

  return 0;

}
```

2. Resumen del proceso de generación de código



La primera etapa del proceso de compilación se conoce como pre-procesamiento. Esta etapa es una traducción previa y básica que tiene como finalidad "acomodar" el código fuente antes de que éste sea procesado por el compilador en sí. El preprocesador modifica el código fuente según las directivas que haya escrito el programador.

En el caso del lenguaje C, estas directivas se reconocen en el código fuente porque comienzan con el carácter #. De esta manera, podemos enunciar aquellas de más frecuente uso:

- #define: Se utiliza para definir una macro que es una especie de 'plantilla de código'. Cuando el preprocesador encuentra esta macro en el código, la reemplaza con su definición.
- #undef: Anula la definición de una macro previamente definida.
- #include: Se utiliza para incluir el contenido de un archivo en otro. Es comúnmente usado para incluir bibliotecas de funciones.
- #error: Esta directiva hace que el preprocesador muestre un mensaje de error y detenga la compilación.
- #if: Permite la compilación condicional. El código dentro de un bloque #if solo se compila si la condición es verdadera.
- #ifdef: Es una abreviatura de "if defined". El código dentro de un bloque #ifdef solo se compila si la macro dada está definida.
- #ifndef: Es una abreviatura de "if not defined". El código dentro de un bloque #ifndef solo se compila si la macro dada no está definida.
- #else: Se utiliza en combinación con #if, #ifdef o #ifndef para proporcionar una alternativa si la condición es falsa.
- #endif: Marca el final de un bloque #if, #ifdef, #ifndef.
- #pragma: Se utiliza para especificar diversas opciones al compilador. La interpretación de #pragma depende del compilador y puede ser ignorada si no se reconoce.

El proceso de enlazado a archivos .lib o .a consiste en enlazar nuestro proyecto con archivos objeto para formar nuestro programa, denominando a esto bibliotecas externas que han sido construidas o bien por nosotros mismos o provienen de terceros como en las bibliotecas estándares de C.

En el proceso de compilación existen una serie de reglas para la compilación en varios pasos con gcc:

- Por cada archivo ".c" de código fuente, generar un archivo de tipo "objeto" (archivos "*.o"). De esta manera, se separan los archivos compilados en varias partes. Este método es más eficiente si hay archivos con alta probabilidad de tener cambios y requiere un menor tiempo en compilación.
- Compilación con gcc flag "-c" que realizará las operaciones de Lectura de un archivo de código, parseo de directivas de precompilado, verificación de código y generación de archivo objeto ".o".
- Además, podemos generar un código pre-procesado con el uso de la flag "-E".
-

```
gcc -E Ejemplo.c > Ejemplo.pp
```

- Para la regla general de generar un archivo objeto por cada archivo ".c".

```
gcc –c Ejemplo.c -o Ejemplo.o
gcc –c main.c -o main.o
```

- En este último caso, en el que tenemos dos objetos por separado, si sólo se modifica el segundo archivo "main.c", sólo se vuelve a compilar ese archivo, quedando el primero sin recompilar.

```
gcc –c main.c -o main.o
```

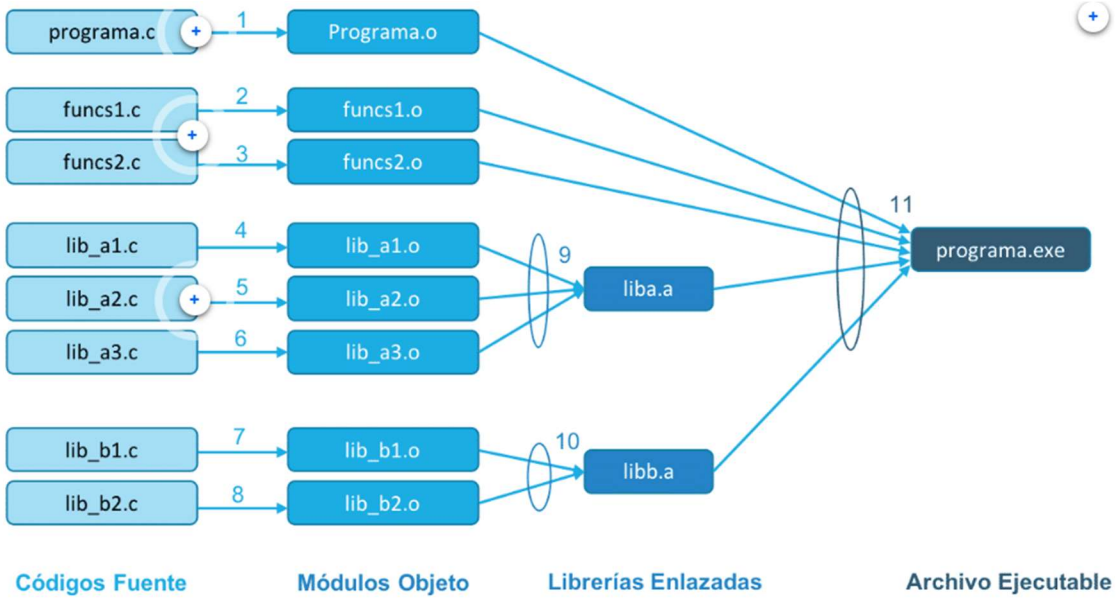
- De esta manera, hemos demostrado la forma básica de proceder con la compilación con GCC. Más adelante es esta unidad, veremos otros casos más complejos, los cuales ocurren cuando existen múltiples ficheros y librerías y que, por lo tanto, la compilación y enlazado requieren un protocolo estricto.

3. Proyectos complejos de más de un fichero

En la construcción de programas complejos es muy común separar el código que se escribe en diferentes archivos y esto tiene muchas ventajas. Para empezar, permite modularizar la funcionalidad y hacer agrupaciones de funciones relacionadas desde un punto de vista lógico, normalmente a cargo de diferentes grupos de programadores.

También, se pueden crear módulos específicos de librerías de funciones que se pueden utilizar en múltiples programas. El uso de los ficheros de cabecera proporciona la interfaz necesaria para poder usar dichas funciones desde cualquier módulo. La compilación de módulos se puede hacer por separado lo que facilita la corrección de los errores en ficheros de muchas líneas de código. En caso de modificación o errores solo es necesaria la recompilación lo que otorga velocidad y eficiencia.

Para mostrar tanto el protocolo como los beneficios de este enfoque de compilación, trabajaremos con un ejemplo concreto a fin de construir un programa ejecutable a partir de varios ficheros objeto y varios bibliotecas o librerías, y también para construir éstos a partir de una serie de ficheros con código fuente. Nuestro programa se construirá a partir de los siguientes ficheros:

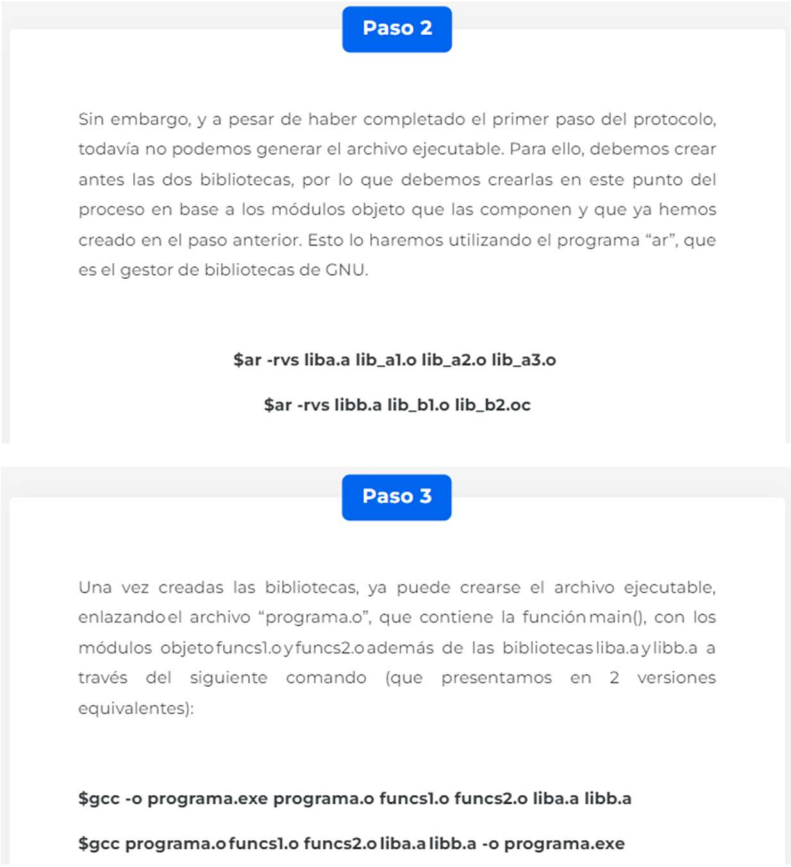


El fichero programa.c contiene la función main y otras del programa, functs1.c serán los ficheros que contienen las funciones y tipos requeridos en el programa. Los lib.c son ficheros que contienen funciones genéricas y de uso estándar en otros programas, o librerías.

Para implementar el protocolo y proceso anterior, procedemos a generar los módulos con código objeto correspondientes a cada uno de los módulos de código fuente. El orden de estas ocho acciones no es importante, pero todas deben realizarse correctamente antes de continuar a la próxima etapa.

Paso 1

```
$gcc -c programa.c -o programa.o
$gcc -c funcs1.c -o funcs1.o
$gcc -c funcs2.c -o funcs2.o
$gcc -c lib_a1.c -o lib_a1.o
$gcc -c lib_a2.c - o lib_a2.o
$gcc -c lib_a3.c -o lib_a3.o
$gcc -c lib_b1.c -o lib_b1.o
$gcc -c lib_b2.c -o lib_b2.o
```



4. Linkado de Librerías y Linkado Dinámico

Vamos a empezar por el linkado estático, este método genera ejecutables más grandes, ya que todo el código se encuentra en un único archivo por eso el hecho de que mover o distribuir el ejecutable es más fácil, pero modificar el ejecutable una vez compilado es difícil porque tendremos que recompilar todo el proyecto y volver a distribuirlo.

Por otro lado, el linkado dinámico genera archivos ejecutables más ligeros, ya que el código generado se mantiene en varios archivos, por lo que es más fácil de modificar el código y generar parches, incluyendo la posibilidad de crear "plugins". Esto conlleva que sea más difícil de distribuir el ejecutable, llegando a situaciones en que se necesitan "instaladores", para lo cual, todas las librerías usadas deben estar disponibles en el sistema.

casos típicos de uso de GCC (GNU Compiler Collection):

- Generar un ejecutable con fuente de un solo archivo:
  $ gcc circulo.c -o circulo
- Crear el fichero pre-procesado:
  $ gcc -E circulo.c > circulo.PP
- Crear un módulo objeto con el mismo nombre del fuente y extensión .o:
  $ gcc -c circulo.c
- Enlazar un módulo objeto en el ejecutable "circulo" :
  $ gcc circulo.o -o circulo
- Enlazar varios módulos objeto (verde.o, azul.o, rojo.o) en el archivo ejecutable "colores" :
  $ gcc verde.o azul.o rojo.o -o colores

5. Proceso de depuración de un programa

La depuración de un programa es el proceso de encontrar los errores de ejecución de un programa y corregir o eliminar dichos errores. En su forma más básica, se realizará una depuración manual, método en el que se proporciona al programa entradas válidas que conducen a una solución conocida. También deben incluirse datos válidos para comprobar la capacidad de detección y generación de errores del programa. Y finalmente, se deben incluir "trazas" en el programa para ir comprobando que los valores intermedios que se van obteniendo son los esperados.

Sin embargo, existen herramientas creadas para facilitar el proceso de depuración, lo que constituye la depuración a partir de herramientas. Dada la complejidad y tamaño de algunos programas, y para ahorro de tiempo en recursos utilizamos los debuggers que poseen todos los IDE. Una vez realizada la depuración, existirá un paso adicional muy importante también. La verificación es la acción de comprobar que el programa está de acuerdo con su especificación o definición de requisitos de diseño. Es decir, se comprueba que el sistema cumple con los requerimientos funcionales y no funcionales que se han especificado.

- Ejecutar paso a paso un programa (stepping).
- Marcar puntos de parada (breakpoints).
- Examinar el contenido de las variables y objetos.
- Conocer el encadenamiento de llamadas de procedimientos.
- Retomar la ejecución hasta un nuevo punto de detención.