# GCC

- Herramientas de desarrollo de software
- Programas con varios archivos fuentes
- Etapas de compilación
- El preprocesador: macros y compilación condicional


Palomeque Nestor Levi

# Herramientas de desarrollo de software

Compilación es el proceso esencial de traducir código escrito en lenguajes de alto nivel, como C o C++, a un lenguaje de máquina que pueda entender y ejecutar un computador. GCC, conocido como Compiler Collection, es un conjunto robusto de herramientas de código abierto diseñado para soportar varios lenguajes de programación. Entre ellos, g++ se destaca como la interfaz principal dentro de GCC para compilar programas específicamente escritos en C++.

# Compilación de un programa en C

Para compilar un programa en C, por ejemplo, debemos usar cualquiera de estas opciones:

```
1) gcc main.c -o miPrograma
2) gcc -o miPrograma main.c
```

Y para ejecutar

```
./miPrograma
```

# Programas con varios archivos fuentes

En programación, los módulos y archivos fuentes permiten dividir programas en múltiples archivos para mejorar la organización y facilitar la reutilización del código. Este enfoque ayuda a mantener el código ordenado y modular. Además, existe una distinción importante entre declaraciones (archivos .h o .hpp) y definiciones (archivos .c o .cpp): las declaraciones especifican la interfaz de funciones y variables, mientras que las definiciones proporcionan su implementación concreta. Esta separación es clave para una programación eficiente y estructurada.

Por ejemplo, tenemos tres archivos: **main.c, operaciones.c** y **operaciones.h**

**main.c**

```c
#include <stdio.h>
#include "operaciones.h"

int main() {
    int a = 5, b = 3;
    printf("Suma: %d\n", suma(a, b));
    printf("Resta: %d\n", resta(a, b));
    return 0;
}
```

**operaciones.c**

```c
#include "operaciones.h"

int suma(int a, int b) {
    return a + b;
}

int resta(int a, int b) {
    return a - b;
}
```

**operaciones.h**

```c
#ifndef OPERACIONES_H
#define OPERACIONES_H

int suma(int a, int b);
int resta(int a, int b);

#endif
```

# Compilación con varios archivos fuentes

Para compilar un programa en C debemos hacer:

```
gcc main.c operaciones.c -o miPrograma
```

A veces es útil compilar cada archivo por separado en archivos objeto (.o) y luego enlazarlos juntos. Esto es especialmente útil en proyectos grandes donde no quieres recompilar todo cada vez que haces un cambio pequeño. Para ello **compilaremos por etapas**:

```
gcc -c main.c
gcc -c operaciones.c
```

Luego debemos enlazar los archivos main.o y operaciones.o generados:

```
gcc main.o operaciones.o -o miPrograma
```

# Etapas de compilación

- **Preprocesado**:  El preprocesado es la primera fase, donde el preprocesador modifica el código fuente antes de que comience la compilación real, procesando directivas **#include**, reemplazando **macros** definidas con **#define**, evaluando directivas condicionales como **#if**, y eliminando comentarios del código fuente
- **Compilado**: en esta fase el compilador traduce el código fuente en lenguaje de alto nivel (como C o C++) a lenguaje ensamblador, verificando la sintaxis y estructura del código, generando un código intermedio independiente de la máquina, realizando optimizaciones para mejorar el rendimiento, y finalmente produciendo el código ensamblador.
- **Ensamblado**: aquí el ensamblador traduce este código ensamblador a lenguaje de máquina (código objeto), creando archivos objeto que contienen el código binario
- **Enlazado**: el linker combina las funciones y variables de varios archivos objeto en un solo programa ejecutable, resolviendo las referencias a funciones y variables entre diferentes archivos objeto, uniendo todos los archivos objeto en un ejecutable y añadiendo el código necesario de las bibliotecas estándar y otras librerías

# Etapas de compilación

- **Preprocesado**:

  `gcc -E hello.c -o hello.i`

- **Compilado**:

  `gcc -S hello.i -o hello.s`

- **Ensamblado**:

  `gcc -c hello.s -o hello.o`

- **Enlazado**:

  `gcc hello.o -o hello`

# El preprocesador: macros

Las macros en C son herramientas poderosas que permiten definir fragmentos de código que serán sustituidos por el preprocesador antes de la fase de compilación. Esto puede incluir desde simples constantes hasta fragmentos de código más complejos. A tener en cuenta:

- **Definición de Macros (#define)**: Se utiliza para definir una macro junto con su sustitución. Por ejemplo: `#define PI 3.14159`
- **Macros con Parámetros**: Permiten crear fragmentos de código que se expanden según los argumentos proporcionados. Por ejemplo: `#define CUADRADO(x) ((x) * (x))`
- **Operadores en Macros**: Pueden utilizarse operadores como '##' (concatenación) para concatenar tokens y crear nombres de funciones o variables dinámicamente ó el operador '#' (stringificación) que convierte un argumento de macro en una cadena literal. Por ejemplo: `#define CONCATENAR(x, y) x ## y`

  Esto permite la creación de identificadores compuestos como: `CONCATENAR(nombre, 1)` que se expandirá a **nombre1**.

# El preprocesador: macros desde linea de comandos

En C, se pueden definir macros utilizando la opción **-D** seguida del nombre de la macro y su valor correspondiente al invocar el compilador. Esto se realiza típicamente desde la línea de comandos del sistema operativo antes de compilar el programa. Veamos un ejemplo práctico para ilustrar este concepto:

```c
#include <stdio.h>
#ifndef OPERACION
 // Por defecto si no se define OPERACION
    #define OPERACION (a + b)
#endif

int main() {
    int a = 5, b = 3;
    int resultado;

    resultado = OPERACION;
    printf("El resultado de la operación es: %d\n", resultado);
    return 0;
}
```

Para compilar:

```
gcc -o miPrograma main.c -DOPERACION="(a * b)"
```

- Al definir -DOPERACION="(a * b)", la macro OPERACION se expandirá a (a * b), donde a y b son las variables locales del programa.
- La directiva **#ifndef** verifica si la macro OPERACION no está definida. En ese caso, se define una operación por defecto (a + b).
- Si no se usa -D la macro se expande a “nada”, lo que si no se controla puede generar errores.

# El preprocesador: compilación condicional

La compilación condicional en C se basa en directivas de preprocesamiento que permiten incluir o excluir partes del código según condiciones definidas. Las más comunes son:

- **#define** para configuraciones: Se utilizan para gestionar configuraciones del programa de manera centralizada. Por ejemplo:

```
#define DEBUG     // Activar modo de depuración
#define VERSION 2 // Número de versión del programa
```

- **#ifdef** y **#ifndef**: Verifican si una macro (o “valor”) está definida o no, respectivamente. Por ejemplo:

```
#ifdef DEBUG
   // Código que se incluirá solo si DEBUG está definida como: #define DEBUG
#endif
```

- **#if**, **#elif**, **#else**, **#endif**: Permite la evaluación de expresiones constantes para la inclusión condicional de código. Por ejemplo:

```
#if VERSION >= 2
   // Código que se incluirá si la VERSION es mayor o igual que 2
#elif VERSION == 1
   // Código alternativo si la VERSION es exactamente 1
#else
   // Código para otros casos
#endif
```

# Actividad

Crear un módulo que contenga las funciones setBit(), resetBit() y getBit(). Las mismas deben setear un bit, resetear un bit y obtener el estado de un bit de una variable entera sin signo. Para ello:

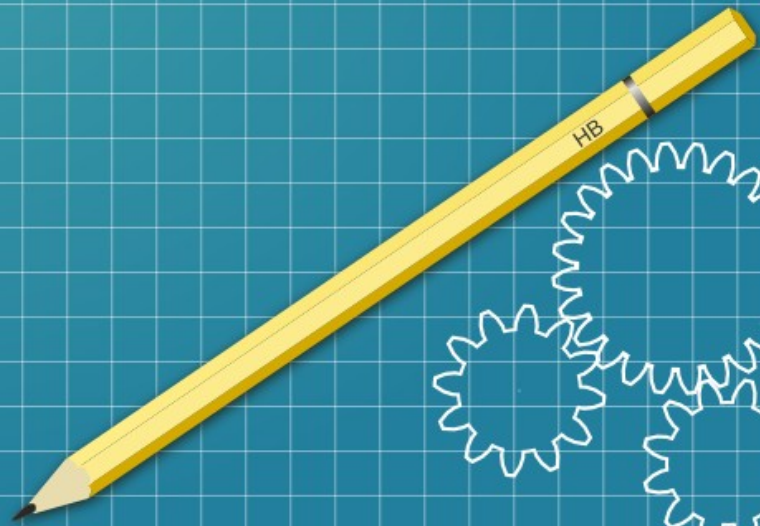
1) Crear el archivo fuente util.c y su correspondiente cabecera util.h
   - Crear un programa que utilice dicho módulo e intercambie el estado del bit 4 por el 2.
   - Empleando un arreglo de caracteres desencriptar el texto, cuyos bits 4 y 2 están intercambiados: "Buff{ qx euq xqq qx yqzgajq"
   - Compilar el proyecto y ejecutarlo
2) Utilizar una macro desde línea de comandos para pasar el dato encriptado. Ten en cuenta que para pasar una cadena como una macro se debe emplear:

- DMARCO="\"Hola mundo\""
- DMARCO='\"Hola mundo\"'

Es decir, encerrando la macro por comillas simples o dobles y luego emplear \ para escapar las comillas dobles que encierran la cadena, para que dichas comillas sean interpretadas literalmente.

3) ¿Cómo sería el programa que genera dicha codificación?





HB