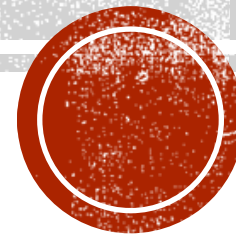


ARGUMENTOS DEL MAIN



Argumentos: argc y argv[]

```
#include <stdio.h>

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

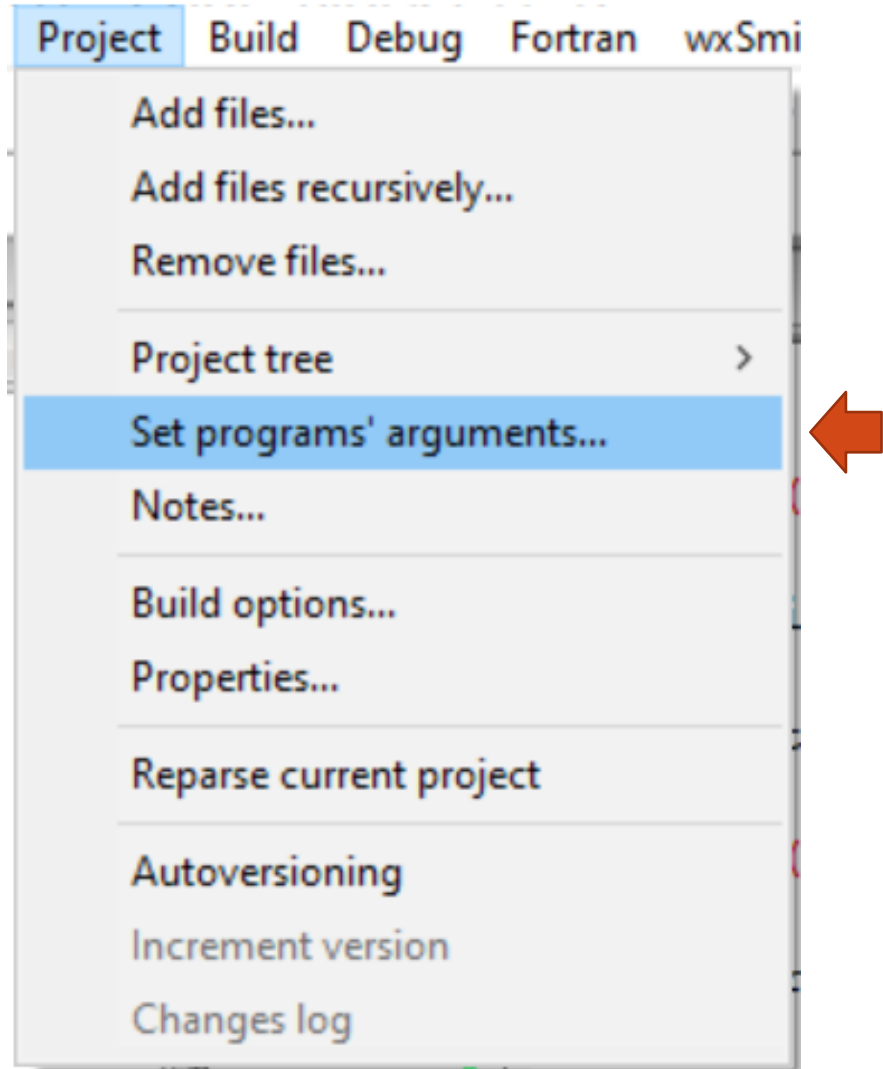
    printf("argc = %d \n", argc);

    for (i=0; i<argc; i++)
    {
        printf("argv[%d] = %s \n", i, argv[i]);
    }
    return 0;
}
```

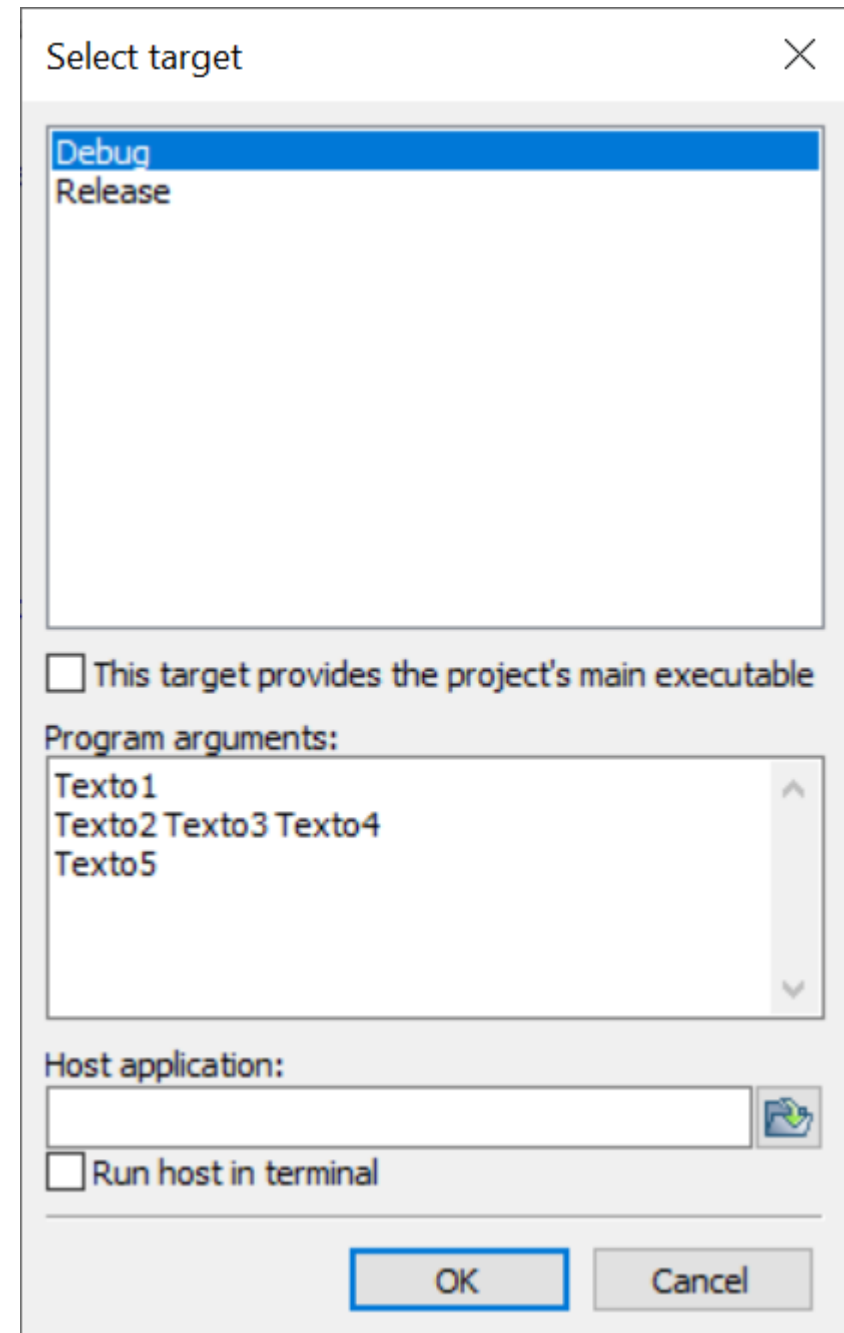
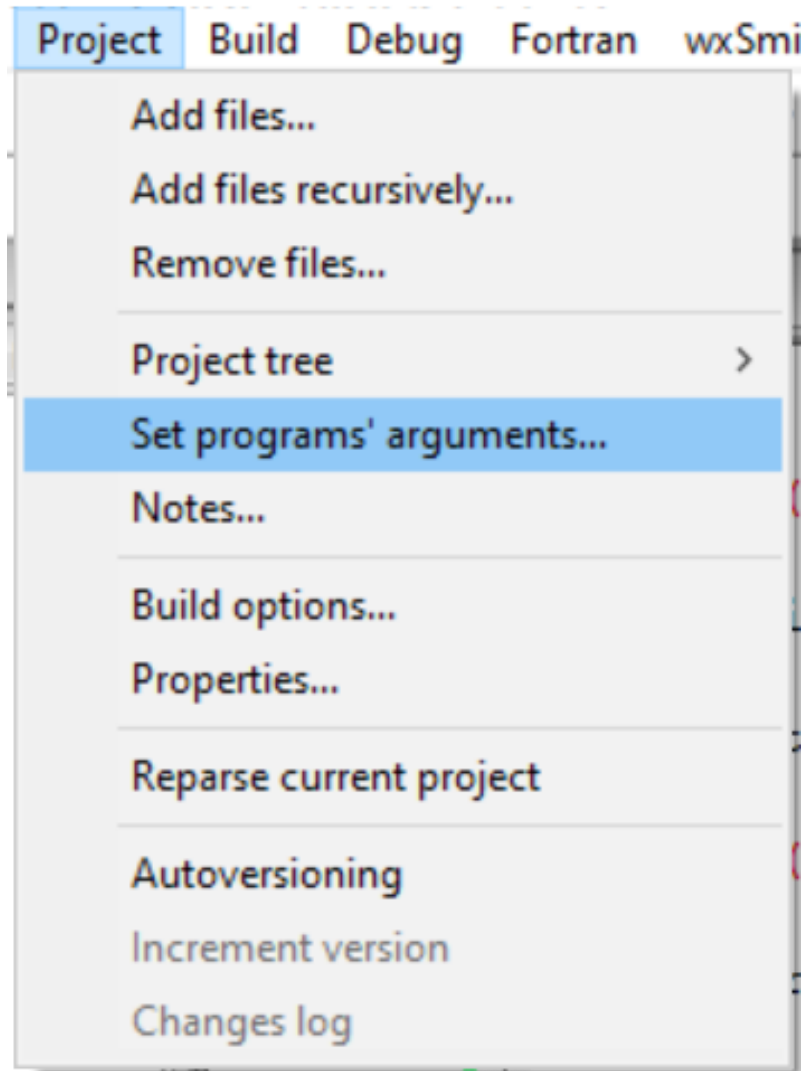
- **argc** indica la cantidad de elementos que contiene argv[].
- **argv[]** contiene el nombre del ejecutable y la lista de parámetros.



DESDE CODE::BLOCKS



DESDE CODE::BLOCKS



Argumentos: argc y argv[]

argMain01.c

```
argc = 6
```

```
argv[0] = C:\\TL1\\Ejemplo1\\bin\\Debug\\Ejemplo1.exe
```

```
argv[1] = Texto1
```

```
argv[2] = Texto2
```

```
argv[3] = Texto3
```

```
argv[4] = Texto4
```

```
argv[5] = Texto5
```

Process returned 0 (0x0) execution time : 0.100 s

Press any key to continue.

Program arguments:

Texto1

Texto2 Texto3 Texto4

Texto5



¿Qué imprime?

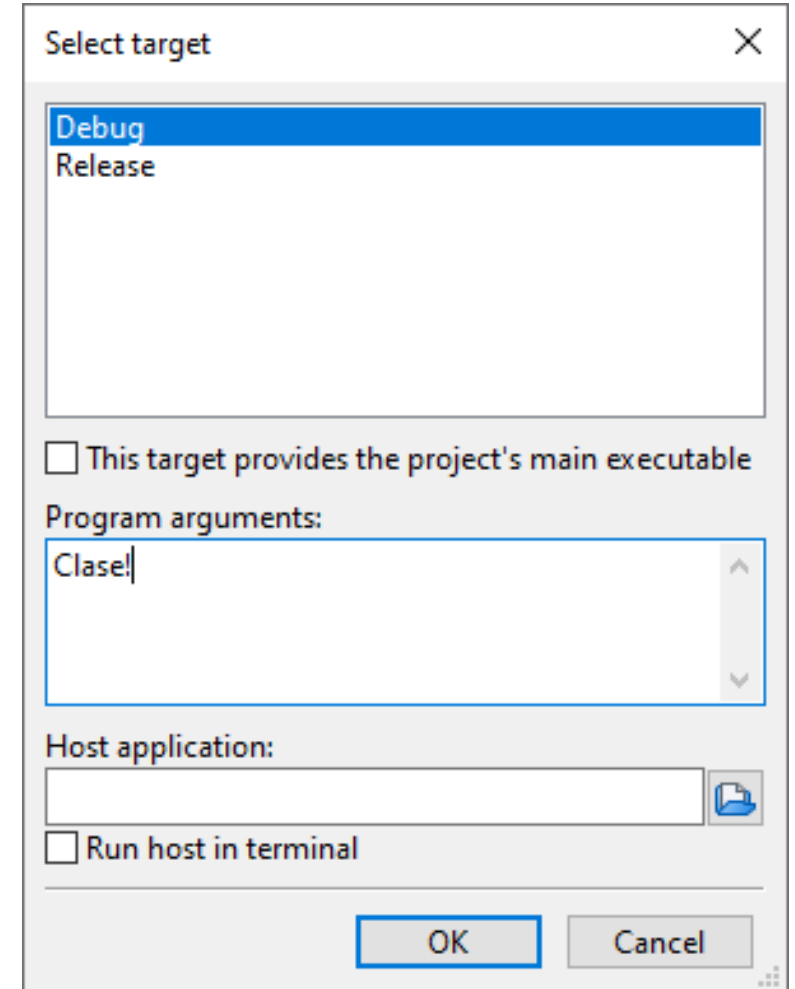
```
#include <stdio.h>

int main(int argc, char * argv[])
{
    if (argc!=2)
        printf("Olvido su nombre!\n");
    else
        printf("Hola %s\n", argv[1]);

    return 0;
}
```

Hola Clase!

Process returned 0 (0x0) execution time : 0.026 s
Press any key to continue.



¿Qué imprime?

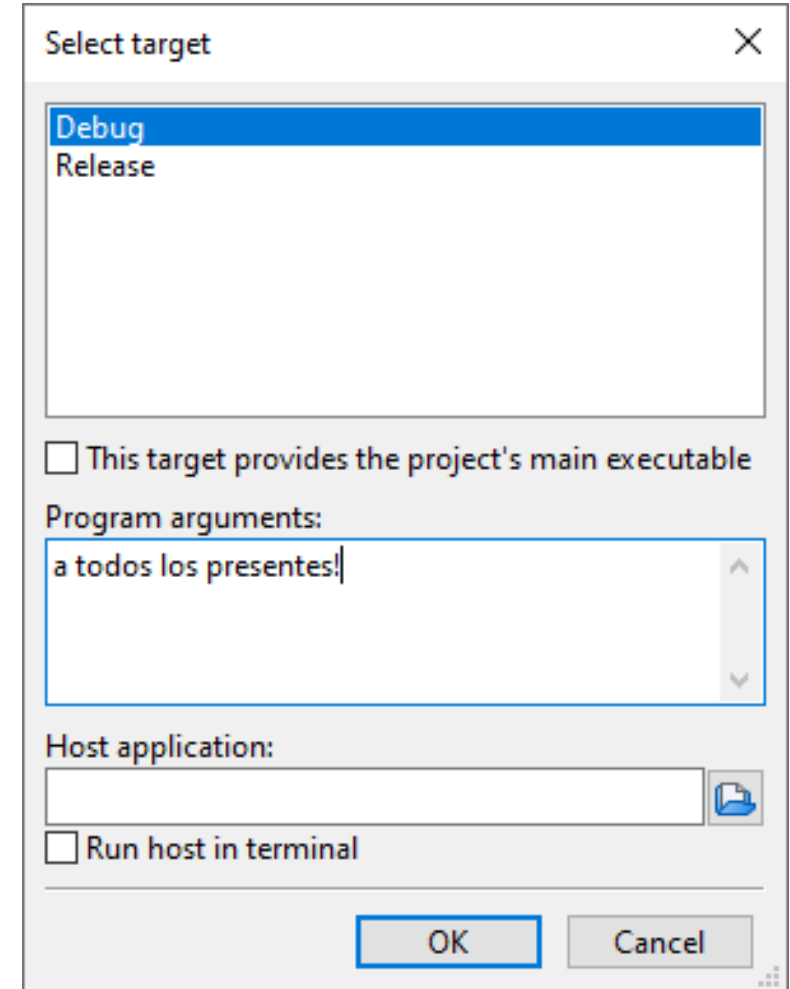
```
#include <stdio.h>

int main(int argc, char * argv[])
{
    if (argc!=2)
        printf("Olvido su nombre!\n");
    else
        printf("Hola %s\n", argv[1]);

    return 0;
}
```

Olvido su nombre!

Process returned 0 (0x0) execution time : 0.009 s
Press any key to continue.



EJERCICIO

- Escriba un programa en C que reciba como parámetros 2 números enteros y un carácter correspondiente a uno de los siguientes operadores: '+', '-', '*' o '/' e imprima el resultado de la operación. Si la cantidad de parámetros es insuficiente debe mostrar el mensaje *“Número incorrecto de parámetros”*.



2Operandos1Operador.c

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char * argv[])
{
    int n1, n2;
    char oper;
    float result;

    if (argc != 4)
        printf("Número incorrecto de parámetros\n");
    else
    {
        n1 = atoi(argv[1]);
        n2 = atoi(argv[2]);
        oper = *argv[3];

        switch (oper){
            case '+': result = n1 + n2; break;
            case '-': result = n1 - n2; break;
            case '*': result = n1 * n2; break;
            default : result = (float)n1 / n2;
        }
        printf ("%d  %c  %d = %.2f", n1, oper, n2, result);
    }
    return 0;
}
```



FUNCIONES DE CONVERSIÓN <stdlib.h>

- `double atof(const char *nptr) ;`

Convierte la posición inicial de la cadena a la cual señala nptr a representación double. La función atof devuelve el valor convertido.

- `int atoi(const char *nptr) ;`

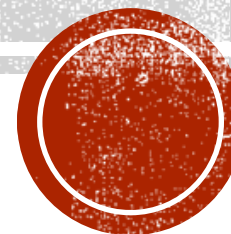
Convierte la posición inicial de la cadena a la cual señala nptr a representación int. La función atoi devuelve el valor convertido.

- `long int atol(const char *nptr) ;`

Convierte la posición inicial de la cadena a la cual señala nptr a representación long. La función atol devuelve el valor convertido.

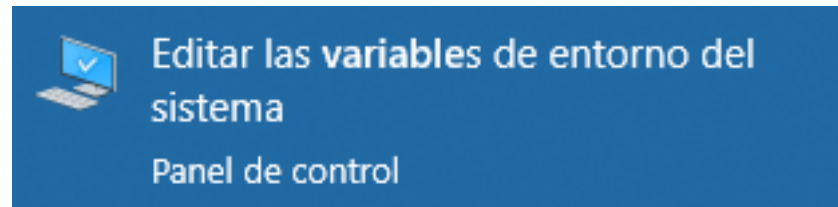


COMPILADOR GCC

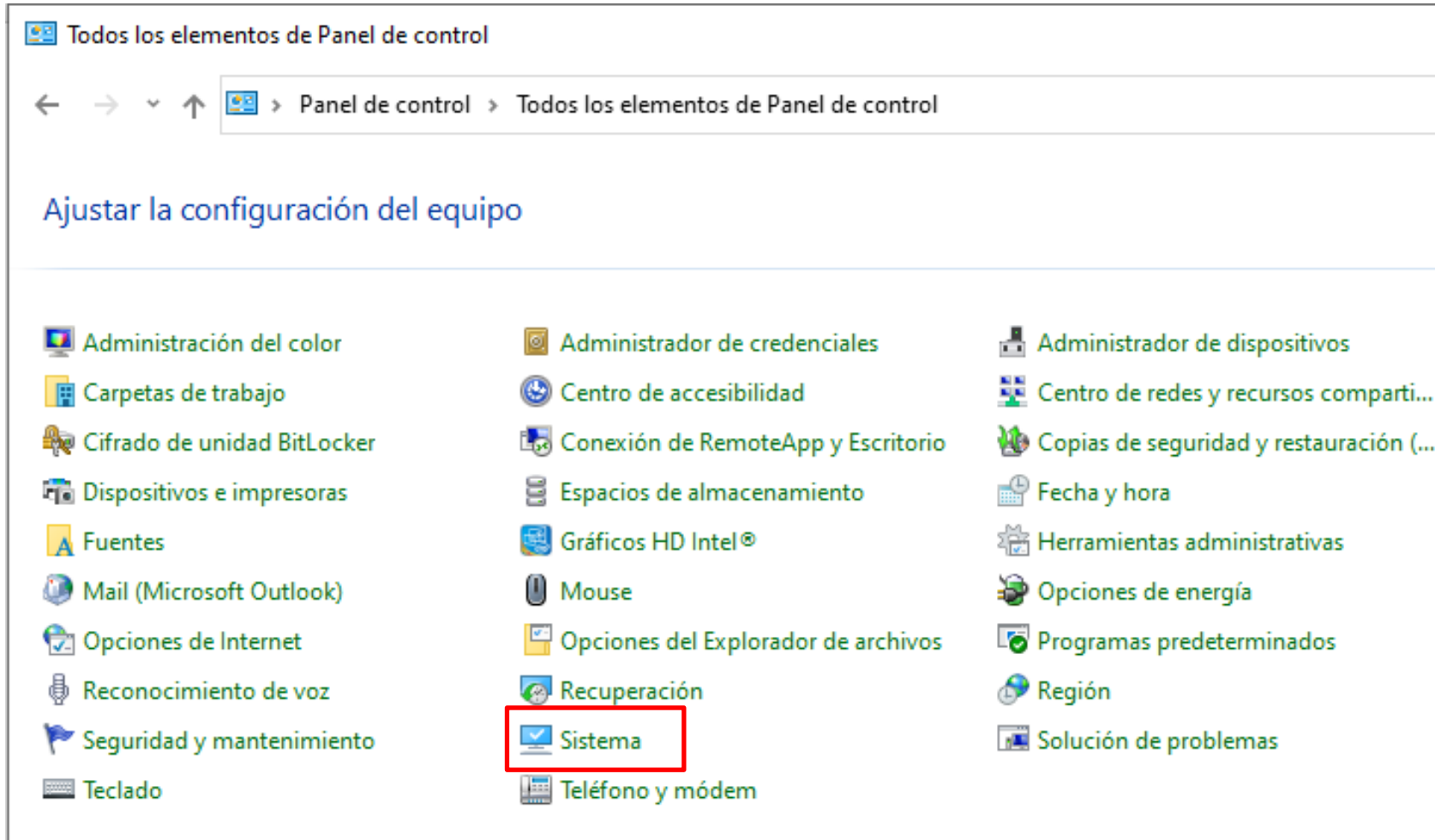


ACCESO A GCC

- Antes de comenzar verifique que la ruta al compilador se encuentra indicada en la variable del sistema PATH.
- Para ello
 - Edite las variables de entorno del sistema ubicadas en la opción SISTEMA del PANEL DE CONTROL



ACCESO A GCC



ACCESO A GCC

Configuración

Inicio

Buscar una configuración

Sistema

Pantalla

Sonido

Notificaciones y acciones

Asistente de concentración

Inicio/apagado y suspensión

Acerca de

Tu equipo está supervisado y protegido.

[Ver detalles en Seguridad de Windows](#)

Especificaciones del dispositivo

Nombre del dispositivo	DESKTOP-BBJS36O
Procesador	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60 GHz
RAM instalada	16,0 GB (15,9 GB usable)
Identificador de dispositivo	6F320F3F-DD98-4735-A806-FFAA306FB5F0
Id. del producto	00331-10000-00001-AA336

Opciones de configuración relacionadas

[Configuración de BitLocker](#)

[Administrador de dispositivos](#)

[Escritorio remoto](#)

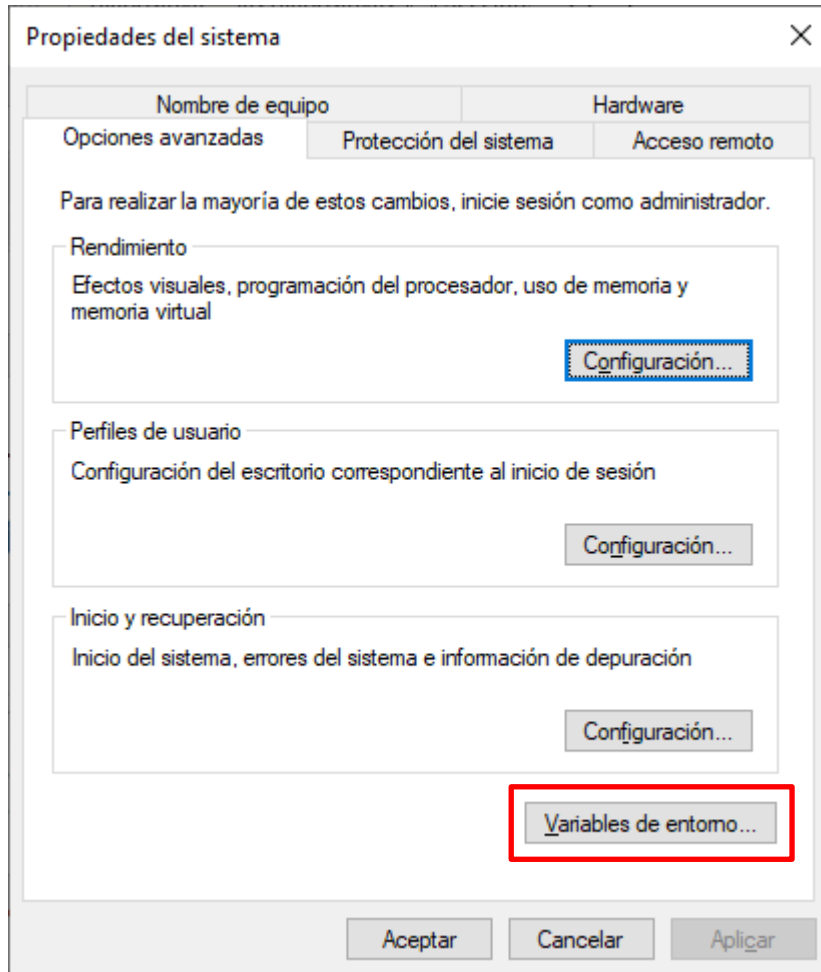
[Protección del sistema](#)

[Configuración avanzada del sistema](#)

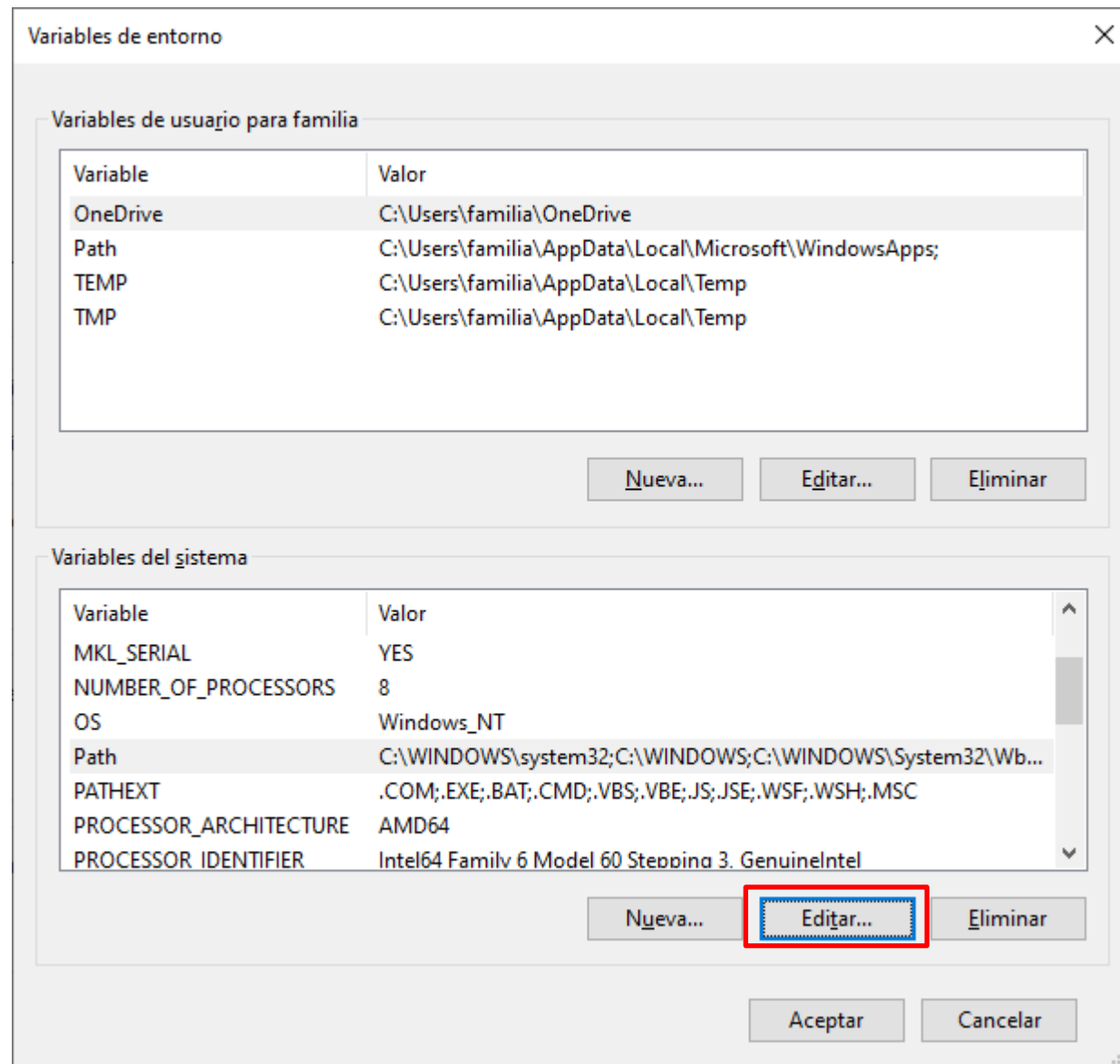
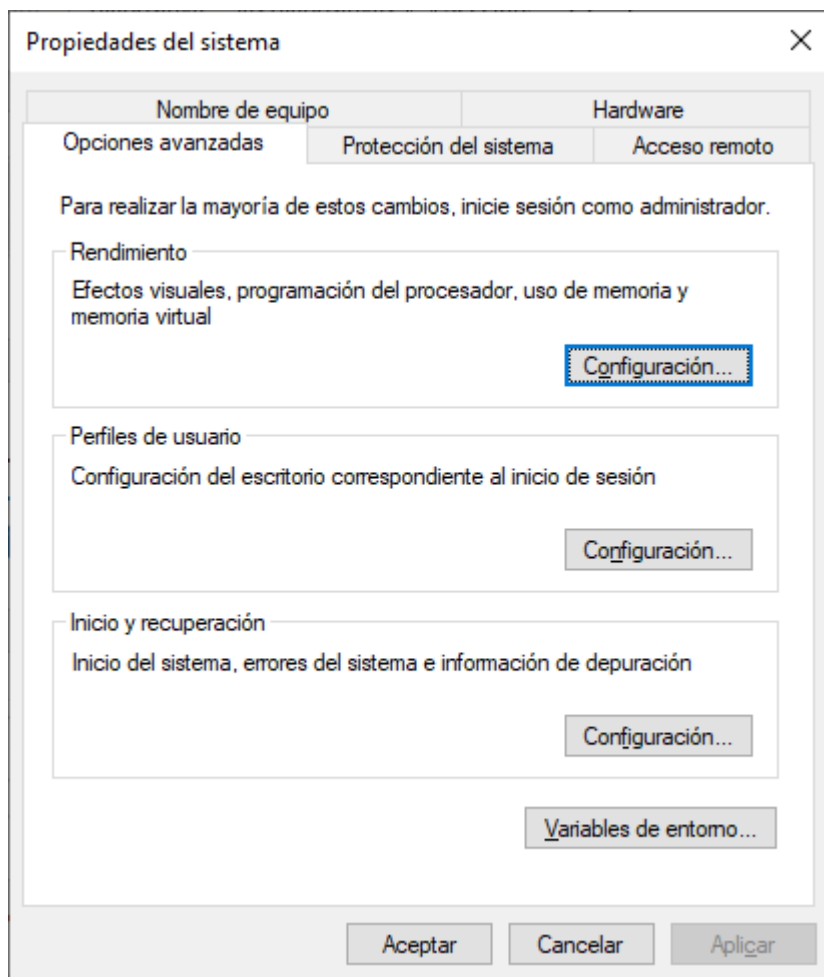
[Cambiar el nombre de este equipo \(avanzado\)](#)



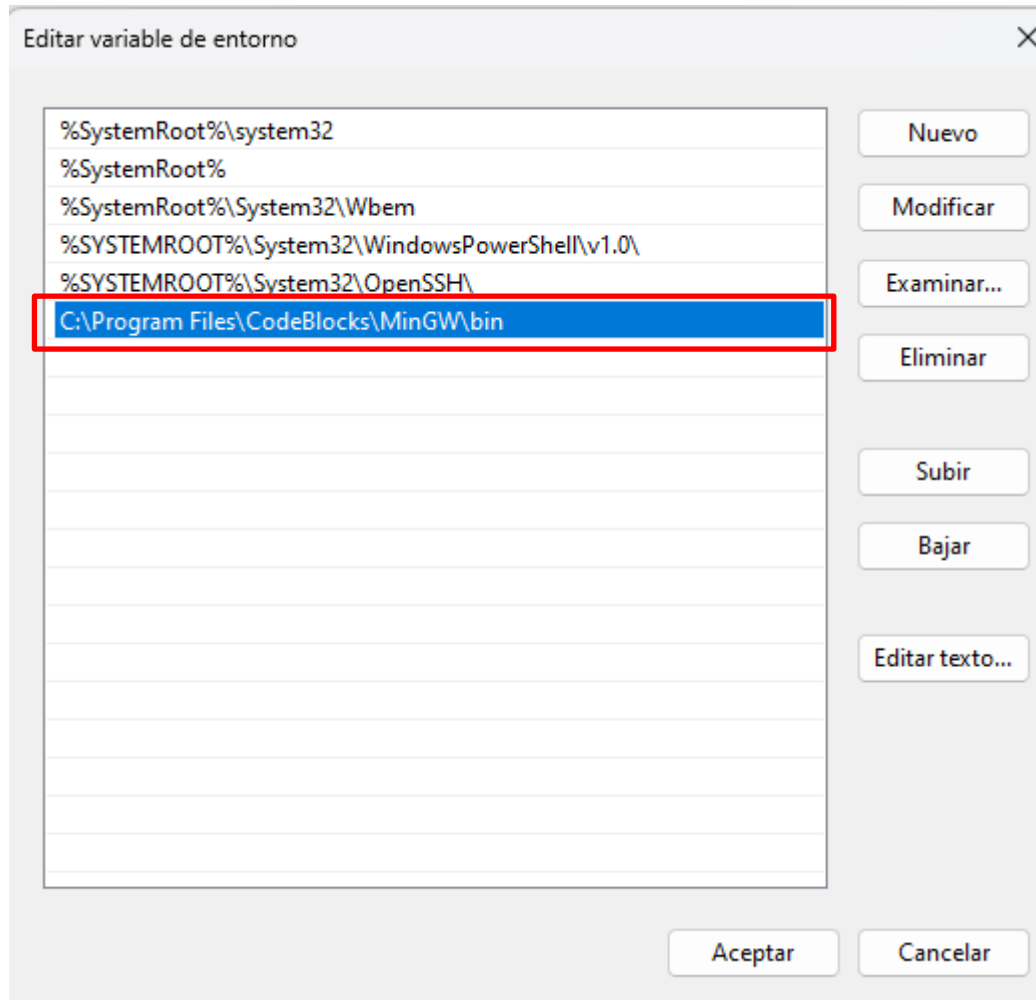
ACCESO A GCC



ACCESO A GCC



ACCESO A GCC



- Verifique que la ruta al compilador se encuentra en la lista.
- Si no está utilice la opción NUEVO y agréguela.
- Para salir elija ACEPTAR



ACCESO A GCC

- El compilador de C de GNU **gcc** es la aplicación que, dado un conjunto de archivos de código C, genera un programa ejecutable. Se puede invocar desde el intérprete de comandos.
- Con la opción **--version** el compilador sólo muestra información sobre su versión. Pruebe el siguiente comando

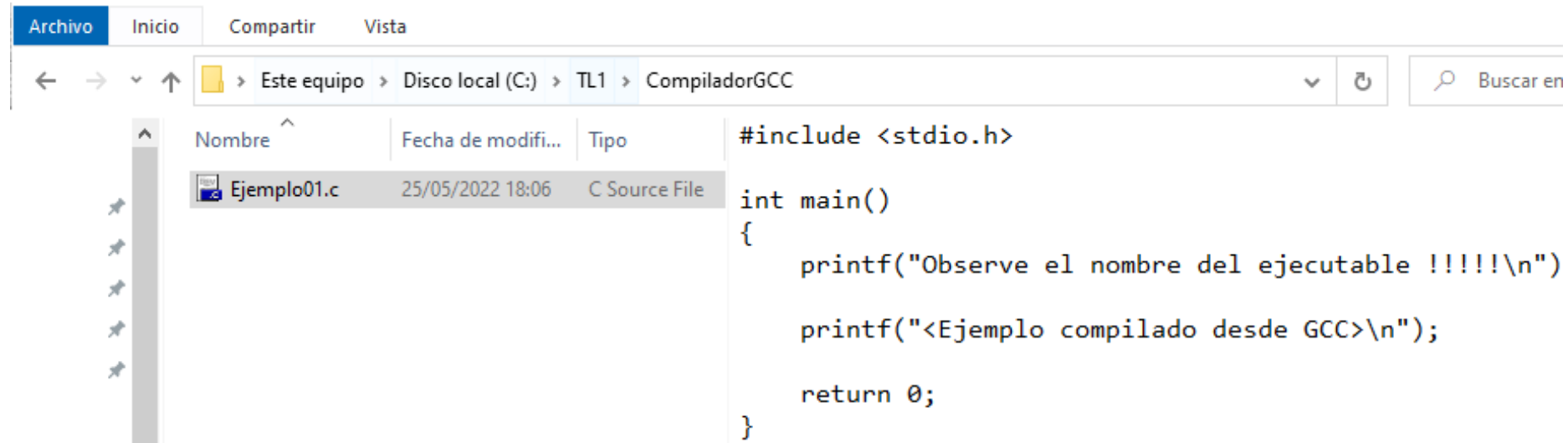
gcc --version

```
c:\TL1\CompiladorGCC>gcc --version
gcc (x86_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```



COMPILANDO UN PROGRAMA

- Cree una carpeta y ubique allí el archivo **Ejemplo01.c**



- Compílelo con el siguiente comando

gcc Ejemplo01.c

- Verifique que el ejecutable generado se llama **a.exe**



COMPILANDO UN PROGRAMA

- Para generar un ejecutable con el nombre Ejemplo01.exe como resultado de compilar el archivo **Ejemplo01.c** debe compilarlo utilizando la opción **-o**

```
gcc -o Ejemplo01 Ejemplo01.c
```

Nombre del
ejecutable

Verifique que se ha
generado un archivo con
nombre **Ejemplo01.exe**



COMPILANDO UN PROGRAMA

- Para generar un ejecutable con el nombre Ejemplo01.exe como resultado de compilar el archivo **Ejemplo01.c** debe compilarlo utilizando la opción **-o**

```
gcc -o Ejemplo01 Ejemplo01.c
```

Verifique que se ha generado un archivo con nombre **Ejemplo01.exe**

- Se recomienda compilar con la opción **-Wall**

```
gcc -Wall -o Ejemplo01 Ejemplo01.c
```



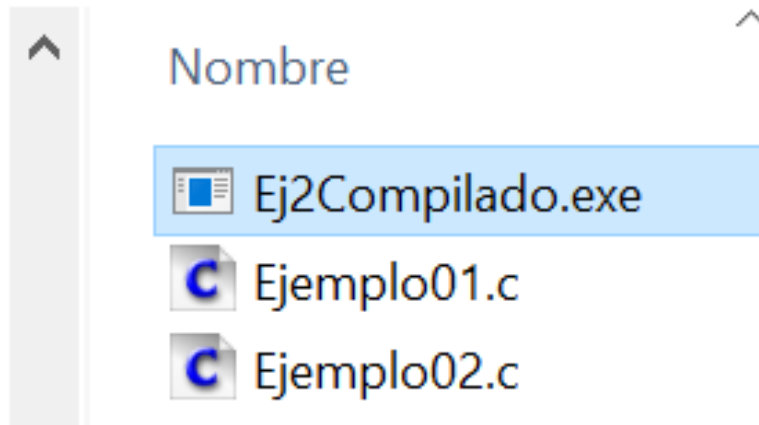
EJERCICIO

- Compile el código del archivo **Ejemplo02.c**. Analice la diferencia de utilizar la opción **-Wall**

```
gcc -o Ej2Compilado Ejemplo02.c
```

¿Cuál será el nombre del ejecutable?

Disco local (C:) > TL1 > CompiladorGCC



EJERCICIO

- Compile el código del archivo **Ejemplo02.c**. Analice la diferencia de utilizar la opción **-Wall**

```
gcc -o Ej2Compilado Ejemplo02.c
```

*¿Se detectó algún
warning?*

```
c:\TL1\CompiladorGCC>gcc -o Ej2Compilado Ejemplo02.c
Ejemplo02.c: In function 'main':
Ejemplo02.c:34:16: warning: implicit declaration of function 'calculate_letter'
[-Wimplicit-function-declaration]
    letter = calculate_letter(index);
```



EJERCICIO

```
gcc -Wall -o Ej2Compilado Ejemplo02.c
```

```
c:\TL1\CompiladorGCC>gcc -Wall -o Ej2Compilado Ejemplo02.c
Ejemplo02.c: In function 'main':
Ejemplo02.c:34:16: warning: implicit declaration of function 'calculate_letter'
[-Wimplicit-function-declaration]
    letter = calculate_letter(index);
               ^
Ejemplo02.c:19:9: warning: variable 'letter' set but not used [-Wunused-but-
set-variable]
    int letter;
       ^
Ejemplo02.c:18:9: warning: unused variable 'not_used' [-Wunused-variable]
    int not_used;
```




```
#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
    int line_width = 80;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(line_width);
}
```

DEFINIENDO SIMBOLOS

- Observe el código del archivo **Ejercicio02.c**



```
#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
    int line_width = 80;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(line_width);
}
```

DEFINIENDO SIMBOLOS

- Observe el código del archivo **Ejercicio02.c**
- La función ***print_line()*** imprime una línea cuya longitud la recibe como parámetro.



```
#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
    int line_width = 80;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(line_width);
```

DEFINIENDO SIMBOLOS

- Observe el código del archivo **Ejercicio02.c**
- La función ***print_line()*** imprime una línea cuya longitud la recibe como parámetro.
- Dentro de la función ***main()*** se la invoca con el valor asignado inicialmente a la variable ***line_width***.



```
#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
    int line_width = 80;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(line_width);
}
```

DEFINIENDO SIMBOLOS

- Es posible indicar la longitud de la línea directamente desde la línea de comandos.



```

#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
int line_width = 90;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(line_width);
}

```

DEFINIENDO SIMBOLOS

- Es posible indicar la longitud de la línea directamente desde la línea de comandos.
- Para ello
 - Borrar la línea donde está declarada la variable ***line_width***



```

#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
int line_width = 90;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(LINEWIDTH);
}

```

DEFINIENDO SIMBOLOS

- Es posible indicar la longitud de la línea directamente desde la línea de comandos.
- Para ello
 - Borrar la línea donde está declarada la variable ***line_width***
 - Modificar las dos invocaciones a la función ***print_line***




```

#include <stdio.h>

/* Dibuja una linea en pantalla */
void print_line(int width)
{
    int i;
    for (i = 0; i < width; i++)
    {
        printf("-");
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
int line_width = 90;
    int index;
    int not_used;
    int letter;

    /* Imprime una linea */
    print_line(LINEWIDTH);
}

```

DEFINIENDO SIMBOLOS

- Es posible indicar la longitud de la línea directamente desde la línea de comandos.
- Para ello
 - Borrar la línea donde está declarada la variable ***line_width***
 - Modificar las dos invocaciones a la función ***print_line***
- Guardar el archivo modificado como **Ej02_Modif.c**



DEFINIENDO SIMBOLOS

- Compile archivo **Ej02_Modif.c** indicando el valor de `LINEWIDTH` mediante la opción `-D`

```
gcc -DLINEWIDTH=80 -o Ej2Modif Ej02_Modif.c
```

```
c:\TL1\CompiladorGCC>gcc -DLINEWIDTH=80 -o Ej02Modif Ej02_Modif.c
Ej02_Modif.c: In function 'main':
Ej02_Modif.c:31:16: warning: implicit declaration of function 'calculate_letter'
[-Wimplicit-function-declaration]
    letter = calculate_letter(index);
                  ^
```

Compile con distintos valores de `LINEWIDTH` y verifique que la longitud de la línea cambia



EJERCICIO

- Corrija los errores de compilación del código **Ejemplo03.c** utilizando la opción **-Wall**



```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int nro, suma=0;
```

```
    printf("Ingrese un numero entero (0=FIN): ");
```

```
    scanf("%d", &nro);
```

```
    while (nro != 0)
```

```
    {
```

```
        suma = suma + nro;
```

```
        printf("Ingrese un numero entero (0=FIN): ");
```

```
        scanf("%d", &nro);
```

```
    }
```

```
    printf("suma = %d", suma);
```

```
    return 0;
```

```
}
```

Ejemplo04.c

Ingrese un numero entero (0=FIN): 1

Ingrese un numero entero (0=FIN): 2

Ingrese un numero entero (0=FIN): 3

Ingrese un numero entero (0=FIN): 4

Ingrese un numero entero (0=FIN): 0

suma = 10

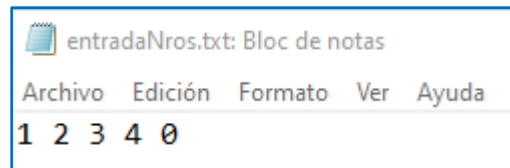
Process returned 0 (0x0) execution time : 5.842 s

Press any key to continue.



REDIRECCIONANDO LA ENTRADA

- Compile el programa Ejemplo04.c utilizando GCC con el nombre **sumaNros.exe**
- Utilizando el bloc de notas genere el archivo de texto **Numeros.txt** con los números que se desean sumar. No olvide poner el cero al final.



- Utilice el siguiente comando

```
sumaNros < Numeros.txt
```



REDIRECCIONANDO LA ENTRADA CON |

- Escriba un programa que imprima una secuencia de números enteros finalizada con cero. El cero también se imprime.
- Compile el código anterior como **generaNros.exe**
- La salida producida por un ejecutable puede ser utilizada como entrada de otro utilizando el símbolo '| '.

generaNros | sumaNros



REDIRECCIONANDO LA SALIDA

- La salida puede redireccionarse usando el símbolo ‘>’

```
generaNros > misNumeros.txt
```

- La salida puede agregarse a un archivo existente con el símbolo ‘>>’

```
generaNros >> misNumeros.txt
```



RESULTADO DEL PREPROCESADOR

```
#define N 10
int main()
{
    int V[N]={0};

    #if N
        V[1]=30;
    #elif
        V[1]=50;
    #endif // N

    return 0;
}
```

```
gcc Ejemplo05.c -E > salida05.c
```

```
# 1 "Ejemplo05.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "Ejemplo05.c"

int main()
{
    int V[10]={0};

    V[1]=30;

    return 0;
}
```



OPTIMIZACIÓN DE CÓDIGO CON GCC

- Un mismo programa puede transformarse en más de una secuencia de instrucciones de máquina. Se puede conseguir un resultado más óptimo si se asignan más recursos al análisis del código durante la compilación.
- Con GCC, puede establecer el nivel de optimización utilizando la opción `-Ox` siendo x el tipo de optimización a realizar.
 - Ejemplo: `gcc hola.c -o hola -O1`



OPTIMIZACIÓN A NIVEL DE CÓDIGO FUENTE

Eliminación de subexpresión común

- La siguiente expresión

$$x = \cos(v) * (1 + \sin(u/2)) + \sin(w) * (1 - \sin(u/2))$$

puede reescribirse como

$$t = \sin(u/2)$$

$$x = \cos(v) * (1 + t) + \sin(w) * (1 - t)$$

- La eliminación de subexpresiones comunes es beneficiosa porque simultáneamente incrementa la velocidad y reduce el tamaño del código.



OPTIMIZACIÓN A NIVEL DE CÓDIGO FUENTE

Expansión de función en línea

- Reduce la sobrecarga de las llamadas a funciones insertando el código de la función en el punto de llamada. Por ejemplo, la sobrecarga de esta función es comparable a realizar una multiplicación fuera de la función

```
double sq (double x) {  
    return x * x;  
}  
for (i = 0; i < 1000000; i++) {  
    sum += sq (i + 0.5);  
}
```



```
for (i = 0; i < 1000000; i++)  
{  
    double t = (i + 0.5);  
    sum += t * t;  
}
```



DILEMA VELOCIDAD-ESPACIO

- Mientras que algunas formas de optimización, tales como la eliminación de subexpresiones comunes, son capaces simultáneamente de incrementar la velocidad y reducir el tamaño de un programa, otros tipos de optimización producen código más rápido a costa de incrementar el tamaño del ejecutable.
- Esta elección entre velocidad y espacio se denomina **dilema velocidad-espacio**.
- Las optimizaciones con el dilema velocidad-espacio se pueden usar en sentido inverso para hacer ejecutables pequeños, a cambio de hacerlos correr más lentos.



DILEMA VELOCIDAD-ESPACIO

- **Desenrollado de bucle**

- Incrementa la velocidad de los bucles eliminando la condición de terminación del bucle en cada iteración.

- Ejemplo

```
for (i = 0; i < 8; i++)  
{  
    y[i] = i;  
}
```

Al final del bucle, esta condición ha sido comprobada 9 veces, y gran parte del tiempo de ejecución fué gastado en la comprobación.



```
y[0] = 0;  
y[1] = 1;  
y[2] = 2;  
y[3] = 3;  
y[4] = 4;  
y[5] = 5;  
y[6] = 6;  
y[7] = 7;
```

Esta optimización incrementa la velocidad pero también incrementa el tamaño del código



DILEMA VELOCIDAD-ESPACIO

- **Desenrollado de bucle**

- Si el límite superior del bucle es desconocido deben controlarse las condiciones de inicio y terminación del bucle.

- Ejemplo

```
for (i = 0; i < n; i++)  
{  
    y[i] = i;  
}
```



```
for (i = 0; i < (n % 2); i++)  
{  
    y[i] = i;  
}  
  
for ( ; i + 1 < n; i += 2)  
{  
    y[i] = i;  
    y[i+1] = i+1;  
}
```

*sin inicializar
¿por qué?*



DILEMA VELOCIDAD-ESPACIO

- **Desenrollado de bucle**

- Si el límite superior del bucle es desconocido deben controlarse las condiciones de inicio y terminación del bucle.

- Ejemplo

```
for (i = 0; i < n; i++)  
{  
    y[i] = i;  
}
```



```
for (i = 0; i < (n % 2); i++)  
{  
    y[i] = i;  
}
```

```
for ( ; i + 1 < n; i += 2)  
{  
    y[i] = i;  
    y[i+1] = i+1;  
}
```

¿Cuál ha sido la optimización?



OPTIMIZACIÓN POR PLANIFICACIÓN

- El menor nivel de optimización es la **planificación**, en el cual el compilador determina **el mejor orden para las instrucciones** individuales.
 - *La mayoría de las CPUs permiten que una o más instrucciones nuevas comiencen a ejecutarse antes de que otras hallan terminado. Muchas CPUs tienen soporte para segmentación, donde múltiples instrucciones se ejecutan en paralelo en la misma CPU.*
- Cuando la planificación está habilitada, las instrucciones pueden disponer de sus resultados llegando a tener disponible la siguiente instrucción en el tiempo correcto, y a permitir un máximo de ejecución en paralelo.
- La planificación aumenta la velocidad de un ejecutable sin incrementar su tamaño, pero requiere memoria y tiempo adicional en sus procesos de compilación (dada su complejidad).



NIVELES DE OPTIMIZACIÓN

‘-O0’ o sin opción ‘-O’ (por defecto)

- **No realiza ninguna optimización** y compila el código fuente de la forma más sencilla posible.
- Cada comando en el código fuente es convertido directamente a sus correspondientes instrucciones en el archivo ejecutable, sin reorganización.
- Esta es la mejor opción a usar cuando se depura un programa y es la opción por defecto si no se especifica una opción de nivel de optimización.

‘-O1’ o ‘-O’

- Este nivel activado es la optimización más frecuente ya que no requiere compensar entre velocidad y espacio.
- Con esta opción los **ejecutables resultantes deberían ser más pequeños y rápidos** que con la opción ‘-O0’. La optimización más costosa, como la planificación de instrucciones, no es usada a este nivel.



NIVELES DE OPTIMIZACIÓN

‘-O2’

- Esta opción activa más optimizaciones, además de las empleadas por ‘-O1’. Estas optimizaciones adicionales incluyen la planificación de instrucciones.
- Sólo se emplearán optimizaciones que no requieren compensación entre velocidad y espacio, así el ejecutable no aumentará de tamaño.
- Al compilador le costará más compilar programas y necesitará más memoria que con la opción ‘-O1’.
- **Esta opción es la más elegida en el desarrollo de programas, porque suministra un máximo de optimización sin incrementar el tamaño del ejecutable.**
- Es la opción del nivel de optimización por defecto en las versiones de paquetes GNU.



NIVELES DE OPTIMIZACIÓN

‘-O3’

- Esta opción activa las más costosas optimizaciones, tales como funciones en-línea, además de todas las optimizaciones de niveles inferiores ‘-O2’ y ‘-O1’.
- El nivel de optimización ‘-O3’ incrementa la velocidad del ejecutable.
- Bajo algunas circunstancias donde esta optimización no es posible, esta opción podría hacer un programa muy lento.

‘-Os’

- Esta opción selecciona las optimizaciones que **reducen el tamaño del ejecutable**.
- Su propósito es producir el ejecutable menor posible, para sistemas con restricciones de memoria o espacio de disco.
- En algunos casos un pequeño ejecutable es más rápido, debido al mejor uso de memoria caché.



NIVELES DE OPTIMIZACIÓN

‘-funroll-loops’

- Esta opción activa el desenrollado de bucles, y es independiente de otras opciones de optimización.
- Se incrementará el tamaño del ejecutable.
- Si esta opción produce o no un resultado beneficioso será comprobado sobre una base caso por caso.



OPTIMIZACIÓN DE CÓDIGO CON GCC

Opción	Descripción
0	Optimizar para la velocidad de compilación - sin optimización del código (por defecto).
1,2,3	Optimizar para aumentar la velocidad de ejecución del código (cuanto mayor sea el número, mayor será la velocidad).
s	Optimizar el tamaño del archivo.
funroll-loops	Optimizar activando el desenrollado de bucles. Es independiente de otras opciones de optimización.



EJEMPLO

- Compile **Ejemplo02conTiempo.c** usando las distintas opciones de optimización.
- Observe el tiempo de ejecución en cada caso.

Opción	Tiempo (en mseg)
-O0	2934
-O1	273
-O2	11
-O3	9
-O3 -funroll-loops	6



EJERCICIO: COMPILE CON DISTINTOS NIVELES DE OPTIMIZACIÓN

```
#include <stdio.h>
#include <time.h>
```

```
double powern (double , unsigned);
```

```
int main ()
```

```
{ clock_t start_t, end_t;
  double total_t, sum = 0.0;
  unsigned i;
```

```
  start_t = clock();
```

```
  for (i = 1; i <= 100000000; i++) {
    sum += powern (i, i % 5);
```

```
  }
```

```
  end_t = clock();
```

```
  printf ("suma = %g\n", sum);
```

```
  total_t = 1000.0* (end_t - start_t) / CLOCKS_PER_SEC;
```

```
  printf("\nTiempo de CPU (en mseg) = %.0f", total_t);
```

```
  return 0;
```

```
}
```

```
double powern (double d, unsigned n)
{
  double x = 1.0;
  unsigned j;

  for (j = 1; j <= n; j++)
    x *= d;

  return x;
}
```

Ejemplo06.c



MÚLTIPLES ARCHIVOS DE CODIGO

main.c

```
void hola (const char *);  
int main ()  
{  
    hola ("mundo");  
    return 0;  
}
```

Los archivos de código sólo
comparten el nombre de la función

funcionHola.c

```
#include <stdio.h>  
void hola (const char * nombre)  
{  
    printf ("¡Hola, %s!\n", nombre);  
}
```



MÚLTIPLES ARCHIVOS DE CODIGO

main.c

```
#include "hola.h"
int main ()
{
    hola ("mundo");
    return 0;
}
```

hola.h

```
#ifndef HOLA_H_INCLUDED
#define HOLA_H_INCLUDED
void hola (const char * nombre);
#endif // HOLA_H_INCLUDED
```

funcionHola.c

```
#include <stdio.h>
#include "hola.h"
void hola (const char * nombre)
{
    printf ("¡Hola, %s!\n", nombre);
}
```



COMPILANDO MÚLTIPLES ARCHIVOS FUENTE

- Para compilar estos archivos con GCC se utiliza el siguiente comando

```
gcc -Wall main.c FuncionHola.c -o nuevoHola
```

- El archivo de cabecera 'hola.es.h' no está especificado en la línea de comandos ya que la directiva `#include "hola.h"` en los archivos fuente ordena al compilador incluirlo de forma automática en los puntos apropiados.

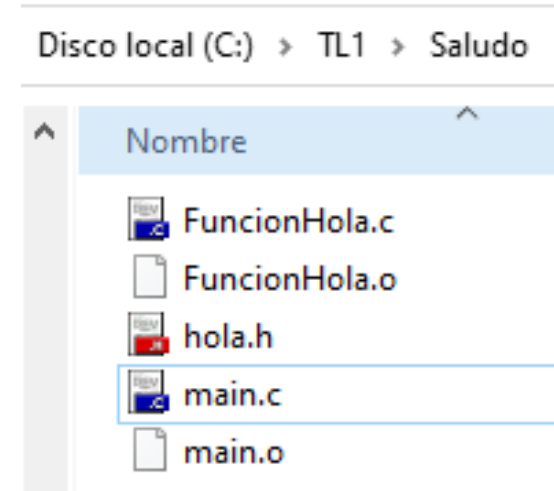


COMPILACION SEPARADA

- La opción **-c** se usa para compilar un módulo fuente y generar el módulo objeto correspondiente.
- Ejemplo:

```
gcc -Wall -c main.c
```

```
gcc -Wall -c FuncionHola.c
```



COMPILACION SEPARADA

- La opción **-c** se usa para compilar un módulo fuente y generar el módulo objeto correspondiente.

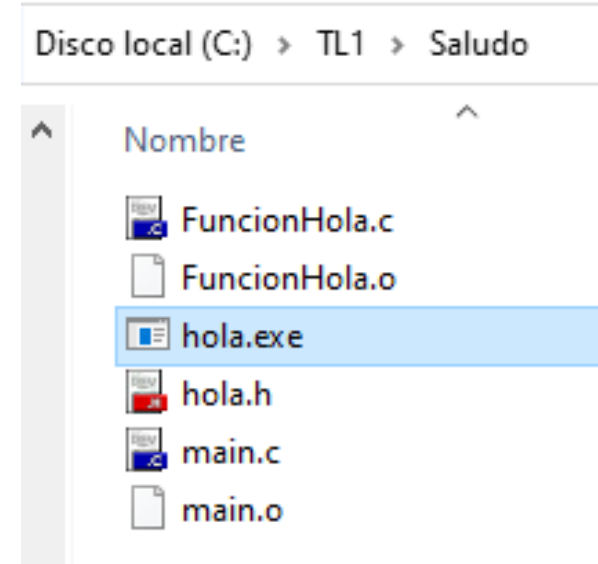
- Ejemplo:

```
gcc -Wall -c main.c
```

```
gcc -Wall -c FuncionHola.c
```

- El ejecutable se genera enlazando los módulos objeto

```
gcc main.o FuncionHola.o -o hola
```



COMPILACIÓN SEPARADA

- Si se producen cambios en el código fuente, puede compilarse sólo aquellos módulos que hayan tenido modificaciones.
- Ejemplo: Se cambia el saludo del archivo **main.c** como se indica a continuación y se lo graba con el nombre **main2.c**

```
#include "hola.h"
int main ()
{
    hola ("cualquiera");
    return 0;
}
```

`gcc -Wall -c main2.c`

`gcc main2.o FuncionHola.o -o hola2`

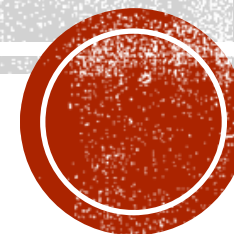
Se compila sólo el módulo modificado

Se enlaza el nuevo módulo objeto con el que se tenía de la función hola.



MAKE

Archivo Makefile



MAKE

- **Make** es una herramienta de gestión de dependencias que controla la **generación de ejecutables** y otros archivos no fuente de un programa a partir de los archivos fuente.
 - Generalmente su función básica es determinar automáticamente qué partes de un programa requieren ser recompiladas y ejecutar los comandos necesarios para hacerlo.
 - También puede usarse para borrar archivos temporales entre otras cosas.
- Make obtiene su conocimiento sobre cómo compilar un programa a partir de un archivo llamado ***Makefile***.



MAKEFILE

- Un archivo Makefile utiliza una sintaxis relativamente sencilla para definir variables y reglas.
- Las reglas utilizan un formato para indicar: el objetivo, los requerimientos y las acciones correspondientes.
 - Las acciones deben comenzar con el carácter TAB.
- Típicamente, un Makefile contiene
 - reglas para compilar archivos fuente.
 - una regla para enlazar los archivos objeto resultantes.
 - un objetivo que sirve como punto de entrada en la parte superior de la jerarquía.



EJEMPLO 1: Contenido del archivo Makefile para compilar un el código escrito en **hola.c** y generar el ejecutable **hola.exe**

```
all: hola.exe
```

```
hola.exe: hola.o  
    gcc hola.o -o hola.exe
```

```
hola.o: hola.c  
    gcc -c hola.c -o hola.o
```

Ejecutar **make** o **make all** es lo mismo

Formato de regla

<Objetivo> : <requerimientos>
[acciones precedidas por TAB]

```
hola.c x
1  #include <stdio.h>
2  int main()
3  {
4      printf("\nHola clase!\n\n");
5
6      return 0;
7  }
```



EJEMPLO 2: Contenido del archivo Makefile para compilar un el código escrito en **hola.c** y generar el ejecutable **hola.exe**

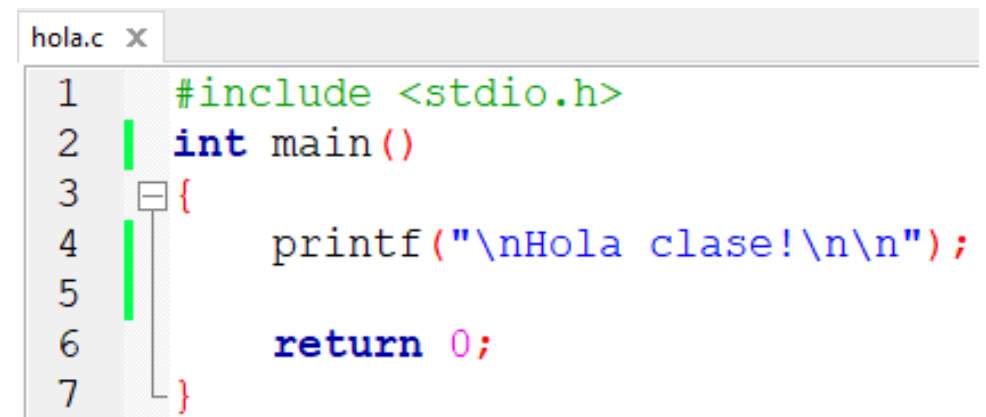
PROGRAM := saludos

COMP := gcc

FLAGS := -O2 -o

Compilar:

`$(COMP) saludos.c $(FLAGS) $(PROGRAM)`

A screenshot of a code editor window titled 'hola.c'. The editor shows a C program with 7 lines of code. Line 1: #include <stdio.h>. Line 2: int main(). Line 3: {. Line 4: printf("\nHola clase!\n\n");. Line 5: (blank). Line 6: return 0;. Line 7: }. A green vertical bar is on the left margin, and a small square icon is next to line 3.

```
1  #include <stdio.h>
2  int main()
3  {
4      printf("\nHola clase!\n\n");
5
6      return 0;
7  }
```



EJEMPLO 3: Contenido del archivo Makefile para compilar un el código escrito en **hola.c** y generar el ejecutable **hola.exe**

```
PROGRAM := hola
```

```
COMP := gcc
```

```
FLAGS := -O2 -o
```

```
Compilar: hola.o
```

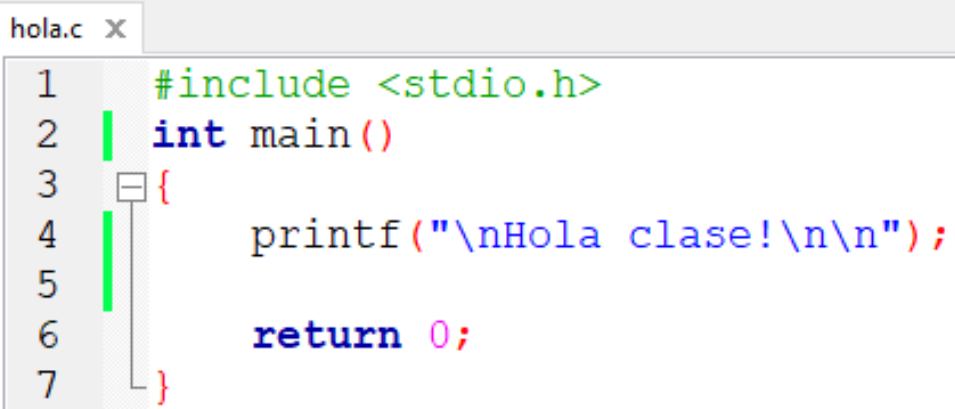
```
$(COMP) hola.o $(FLAGS) $(PROGRAM)
```

```
hola.o: hola.c
```

```
gcc -c hola.c -o hola.o
```

```
clean:
```

```
del -f $(PROGRAM) *.o
```



The screenshot shows a code editor window titled 'hola.c'. The code is as follows:

```
1  #include <stdio.h>
2  int main()
3  {
4      printf("\nHola clase!\n\n");
5
6      return 0;
7  }
```



VARIABLES AUTOMÁTICAS

\$@	Se sustituye por el nombre del objetivo de la regla.
\$*	Se sustituye por la raíz de un nombre de archivo.
\$<	Se sustituye por la primera dependencia de la presente regla.
\$^	Se sustituye por una lista separada por espacios de cada una de las dependencias de la presente regla.
\$?	Se sustituye por una lista separada por espacios de cada una de las dependencias de la presente regla que sean más nuevas que el objetivo de la regla.



EJEMPLO: Contenido del archivo Makefile para compilar el código escrito en **hola.c** y generar el ejecutable **hola.exe**

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -O3
```

```
hola: hola.o
```

```
$(CC) $(LDFLAGS) -o $@ $^
```

\$@ se refiere al objetivo de la regla
En este caso "hola"

\$^ equivale a todos los requerimientos.
En este caso "hola.o"

```
hola.o: hola.c
```

```
$(CC) $(CFLAGS) -c $<
```

\$< corresponde al 1er. requerimiento.
En este caso "hola.c"

```
clean:
```

```
del -f hola.exe hola.o
```

Utilice **rm** en Linux



MÚLTIPLES ARCHIVOS DE CODIGO

main.c

```
#include "hola.h"
int main ()
{
    hola ("mundo");
    return 0;
}
```

hola.h

```
#ifndef HOLA_H_INCLUDED
#define HOLA_H_INCLUDED
void hola (const char * nombre);
#endif // HOLA_H_INCLUDED
```

funcionHola.c

```
#include <stdio.h>
#include "hola.h"
void hola (const char * nombre)
{
    printf ("¡Hola, %s!\n", nombre);
}
```



MAKEFILE

```
all: programa.exe
```

```
programa.exe: main.o funcionHola.o  
    gcc main.o funcionHola.o -o programa.exe
```

```
main.o: main.c  
    gcc -c main.c -o main.o
```

```
funcionHola.o: funcionHola.c  
    gcc -c funcionHola.c -o funcionHola.o
```

hola.h

```
#ifndef HOLA_H_INCLUDED  
#define HOLA_H_INCLUDED  
void hola (const char * nombre);  
#endif // HOLA_H_INCLUDED
```

main.c

```
#include "hola.h"  
int main ()  
{ hola ("mundo");  
  return 0;  
}
```

funcionHola.c

```
#include <stdio.h>  
#include "hola.h"  
void hola (const char * nombre)  
{  
    printf ("¡Hola, %s!\n", nombre);  
}
```



```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hola.c
OBJ=$(SOURCE:.c=.o)
EXE=hola.exe
```

Crea una variable "OBJ" que toma el valor de "SOURCE" pero reemplaza la extensión ".c" por ".o".
De esta forma, el nombre del archivo objeto es igual al del archivo fuente.

```
all: $(EXE)
```

```
$(EXE): $(OBJ)
    $(CC) $(OBJ) -o $@
```

```
%.o: %.c
    $(CC) $(CFLAGS) $< -o $@
```

```
clean:
    del -f $(EXE) $(OBJ)
```



```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hola.c
OBJ=$(SOURCE:.c=.o)
EXE=hola.exe
```

```
all: $(EXE)
```

```
$(EXE): $(OBJ)
    $(CC) $(OBJ) -o $@
```

```
%.o: %.c
    $(CC) $(CFLAGS) $< -o $@
```

```
clean:
    del -f $(EXE) $(OBJ)
```

Es una regla que compila cualquier archivo .c en un archivo .o

\$< corresponde al 1er. requerimiento.
En este caso "hola.c"

\$@ se refiere al objetivo
En este caso "hola.o"



```
CC = gcc
```

```
CFLAGS = -c -Wall
```

```
SOURCES = $(wildcard *.c)
```

```
OBJS = $(SOURCES:.c=.o)
```

```
EXE = hola.exe
```

```
all: $(EXE)
```

```
$(EXE): $(OBJS)
```

```
$(CC) $(OBJS) -o $@
```

```
%.o: %.c
```

```
$(CC) $(CFLAGS) $< -o $@
```

```
clean:
```

```
rm -f $(OBJS) $(EXE)
```

Busca todos los archivos .c en el directorio actual

Genera la lista de archivos objeto

\$@ se refiere al objetivo
En este caso "hola.o"

\$< corresponde al 1er. requerimiento.
En este caso "hola.c"

