Repaso de Informática 1 - Parte 1

Ing José Luis MARTÍNEZ

7 de abril de 2021

Algoritmos

Siempre que se quiera programar se le deberá decir a la computadora, exactamente lo que debe hacer y en el orden que debe hacerlo. Por ello antes de escribir un programa es esencial conocer el problema y como solucionarlo. Cualquier paso que se de en forma equivocada provocará que la computadora no ejecute lo que se desea.

La solución a cualquier problema de cómputo involucra la ejecución de una serie de acciones en un orden específico, es decir:

1- Se debe saber que acciones se deben ejecutar
2- Y el orden en que deben ser ejecutadas

A este procedimiento se le denomina *algoritmo.*

Veamos con un ejemplo la importancia de hacer un buen algoritmo. Consideremos el algoritmo *"levantarse para ir a la escuela",* y supongamos que el alumno es una computadora que hace exactamente lo que se le dice:

Levantarse de la cama
Quitarse el pijama
Bañarse
Vestirse
Desayunar
Tomar el colectivo

Así el algoritmo se encuentra bien planteado, pero si le introducimos un pequeño cambio

Levantarse de la cama
Quitarse el pijama
Vestirse
Bañarse
Desayunar
Tomar el colectivo

El alumno lleva empapado a la escuela. El ejemplo nos ilustra bien la importancia de dar instrucciones precisas y en el orden correcto.

Pseudocódigo

El pseudocódigo es un lenguaje artificial e informal que nos sirve para desarrollar algoritmos. El pseudocódigo es similar al inglés común pero no es un lenguaje de programación y no se puede ejecutar.

EJEMPLO 1.1

Escriba el pseudocódigo de un programa que, al recibir los datos A, B, C y D que representan números enteros, escriba los mismos en orden inverso.

Leer el primer dato entero y guardarlo en A
Leer el primer dato entero y guardarlo en B
Leer el primer dato entero y guardarlo en C
Leer el primer dato entero y guardarlo en D
Escribir "D C B A"

EJEMPLO 1.2

Escriba un pseudocódigo, en el cual reciba la clave de un empleado y los seis primeros sueldos del año; calcule el ingreso total semestral y el promedio mensual, e imprima la clave del empleado, el ingreso total y el promedio mensual.

Leer la clave del empleado y guardarla en Clave
Ingresar el sueldo N° 1 y guardarlo en Mes_1
Ingresar el sueldo N° 2 y guardarlo en Mes_2
Ingresar el sueldo N° 3 y guardarlo en Mes_3
Ingresar el sueldo N° 4 y guardarlo en Mes_4
Ingresar el sueldo N° 5 y guardarlo en Mes_5
Ingresar el sueldo N° 6 y guardarlo en Mes_6
Suma=Mes_1+Mes_2+Mes_3+Mes_4+Mes_5+Mes_6

Promedio=Suma/6
Imprimir Clave, Suma, Promedio

EJEMPLO 1.3
Escriba un pseudocódigo, al recibir como datos la base y la altura de un triángulo, calcule su superficie.

Leer base y guardar en Base
Leer altura y guardar en Altura
*Area=Base*Altura/2*
Imprimir Base, Altura, Area

Problema 1 Escriba un pseudocódigo que, al recibir como datos la longitud y el peso de un objeto expresados en pies y libras, imprima los datos de este objeto pero expresados en metros y kilos, respectivamente.
Datos: PIE, LIB
Consideraciones:
• Un pie equivale a 0.09290 metros.
• Una libra equivale a 0.45359 kilogramos

Problema 2 Escriba el pseudocodigo que, al recibir como datos el radio y la altura de un cilindro sólido, calcule e imprima el área y su volumen.
Datos: RAD, ALT
Consideraciones:
• El volumen de un cilindro lo calculamos aplicando la siguiente fórmula:
Volumen = π* radio^2 * altura
Donde: π= 3.141592.
• La superficie del cilindro la calculamos como:
Área = 2 *π * radio * altura+2*π*radio^2

Problema 3
Una persona compró una estancia. La extensión de la estancia está especificada en acres. Escriba el pseudocódigo que, al recibir como dato la extensión de la estancia en *acres*, calcule e imprima la extensión de la misma en hectáreas.
Dato: ECA (variable de tipo real que especifica la extensión de la estancia en acres).
Consideraciones:
• 1 acre es igual a 4047 m^2.
• 1 hectárea tiene 10,000 m^2.

Problema 4
Escriba el pseudocódigo que, al recibir como datos los tres lados de un triángulo, calcule e imprima su área. Ésta se puede calcular aplicando la siguiente fórmula:

$$AREA = \sqrt{AUX * (AUX - LAI) * (AUX - LA2) * (AUX - LA3)}$$

$$AUX = (LA1 + LA2 + LA3) / 2$$

Problema 5
Escriba el pseudocódigo que, al recibir como datos las coordenadas de los puntos P1, P2 y P3 que corresponden a los vértices de un triángulo, calcule su perímetro.
Consideraciones:
• Para calcular la distancia DIS entre dos puntos dados P1 y P2 aplicamos la siguiente fórmula:

$$DIS = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

Tipos de datos

Los datos que procesa una computadora se clasifican en **simples** y **estructurados**. La principal característica de los tipos de datos simples es que ocupan sólo una casilla de memoria. Dentro de este grupo de datos se encuentran principalmente los **enteros**, los **reales** y los **caracteres**.

Tipo de dato en C	**Descripción**	**Rango**
int	Enteros	-32768 a 32767
float	Reales	$3{,}4x10^{-38}$ a $3{,}4x10^{38}$
long	Enteros de largo alcance	-2147483648 a 2147483647
double	Reales de doble precisión	$1{,}7x10^{-308}$ a $1{,}7x10^{308}$
char	Carácter	Símbolos del abecedario, números o símbolos especiales, que van encerrados entre comillas

Por otra parte, los datos estructurados se caracterizan por el hecho de que con un nombre se hace referencia a un grupo de casillas de memoria. Es decir, un dato estructurado tiene varios componentes. Los **arreglos**, **cadena de caracteres** y **registros** representan los datos estructurados más conocidos

Identificadores

Los datos que procesará una computadora, ya sean simples o estructurados, se deben almacenar en casillas o celdas de memoria para utilizarlos posteriormente. A estas casillas o celdas de memoria se les asigna un nombre para reconocerlas: un **identificador**, el cual se forma por medio de letras, dígitos y el caracter de subrayado (_). Siempre hay que comenzar con una letra. El lenguaje de programación **C** distingue entre minúsculas y mayúsculas, por lo tanto AUX y Aux son dos identificadores diferentes. La longitud más común de un identificador es de tres caracteres, y generalmente no excede los siete caracteres. En **C,** dependiendo del compilador que se utilice, es posible generar identificadores más grandes. Cabe destacar que hay nombres que no se pueden utilizar por ser palabras reservadas del lenguaje **C**. Estos nombres prohibidos se presentan en la siguiente tabla.

TABLA 1.3. Palabras reservadas del lenguaje C

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	float	return	typedef	
default	for	short	union	

Constantes

Las **constantes** son datos que no cambian durante la ejecución del programa. Para nombrar las constantes utilizamos identificadores. Existen tipos de constantes de todos los tipos de datos, por lo tanto puede haber constantes de tipo entero, real, caracter, cadena de caracteres, etc. Las constantes se deben definir antes de

comenzar el programa principal, y éstas no cambiarán su valor durante la ejecución del mismo. Existen dos formas básicas de definir las constantes:

```
const int nu1 = 20; /* nu1 es una constante de tipo entero. */
const int nu2 = 15; /* nu2 es una constante de tipo entero. */
const float re1 = 2.18; /* re1 es una constante de tipo real. */
const char ca1 = 'f'; /* ca1 es una constante de tipo caracter. */
```

Otra alternativa es la siguiente:

```
#define nu1 20; /* nu1 es una constante de tipo entero. */
#define nu2 15; /* nu2 es una constante de tipo entero. */
#define re1 2.18; /* re1 es una constante de tipo real. */
#define ca1 = 'f'; /* ca1 es una constante de tipo caracter. */
```

Otra forma de nombrar constantes es utilizando el método enumerador: enum. Los valores en este caso se asignan de manera predeterminada en incrementos unitarios, comenzando con el cero. enum entonces es útil cuando queremos definir constantes con valores predeterminados. A continuación se presenta la forma como se declara un enum:

```
enum { va0, va1, va2, va3 }; /* define cuatro constantes enteras. */
```

Esta definición es similar a realizar lo siguiente:

```
const int va0 = 0;
const int va1 = 1;
const int va2 = 2;
const int va3 = 3;
```

Variables

Las **variables** son objetos que pueden cambiar su valor durante la ejecución de un programa. Para nombrar las variables también se utilizan identificadores. Al igual que en el caso de las constantes, pueden existir tipos de variables de todos los tipos de datos. Por lo general, las variables se declaran en el programa principal y en las, y pueden cambiar su valor durante la ejecución del programa.
Observemos a continuación la forma como se declaran:

```
void main( void)
{
...
int va1, va2; /* Declaración de variables de tipo entero. */
float re1, re2; /* Declaración de variables de tipo real. */
char ca1, ca2; /* Declaración de variables de tipo caracter. */
...
}
```

Una vez que se declaran las variables, éstas reciben un valor a través de un **bloque de asignación**. La asignación es una operación destructiva. Esto significa que si la variable tenía un valor, éste se destruye al asignar el nuevo valor. El formato de la asignación es el siguiente:

```
variable = expresión o valor;
```

Donde expresión puede representar el valor de una expresión aritmética, constante o variable. Observa que la instrucción finaliza con punto y coma: ; .
Analicemos a continuación el siguiente caso, donde las variables reciben un valor a través de un bloque de asignación.

```
void main( void)
{
...
int va1, va2;
float re1, re2;
char ca1, ca2;
...
va1 = 10; /* Asignación del valor 10 a la variable va1.*/
va2 = va1 + 15; /* Asignación del valor 25 (expresión aritmética) a va2. */
va1 = 15; /* La variable va1 modifica su valor. */
re1 = 3.235; /* Asignación del valor 3.235 a la variable real re1. */
re2 = re1; /* La variable re2 toma el valor de la variable re1. */
ca1 = 't'; /* Asignación del caracter 't' a la variable ca1. */
ca2 = '?'; /* Asignación del caracter '?' a la variable ca2. */
...
}
```

Otra forma de realizar la asignación de un valor a una variable es cuando se realiza la declaración de la misma. Observemos el siguiente caso.

```
void main( void)
{
...
int va1 = 10, va2 = 15;
float re1= 3.25, re2 = 6.485;
char ca1 = 't', ca2 = 's';
```

1. Instrucciones de decisión

1.1. Instrucciones if, if .. else

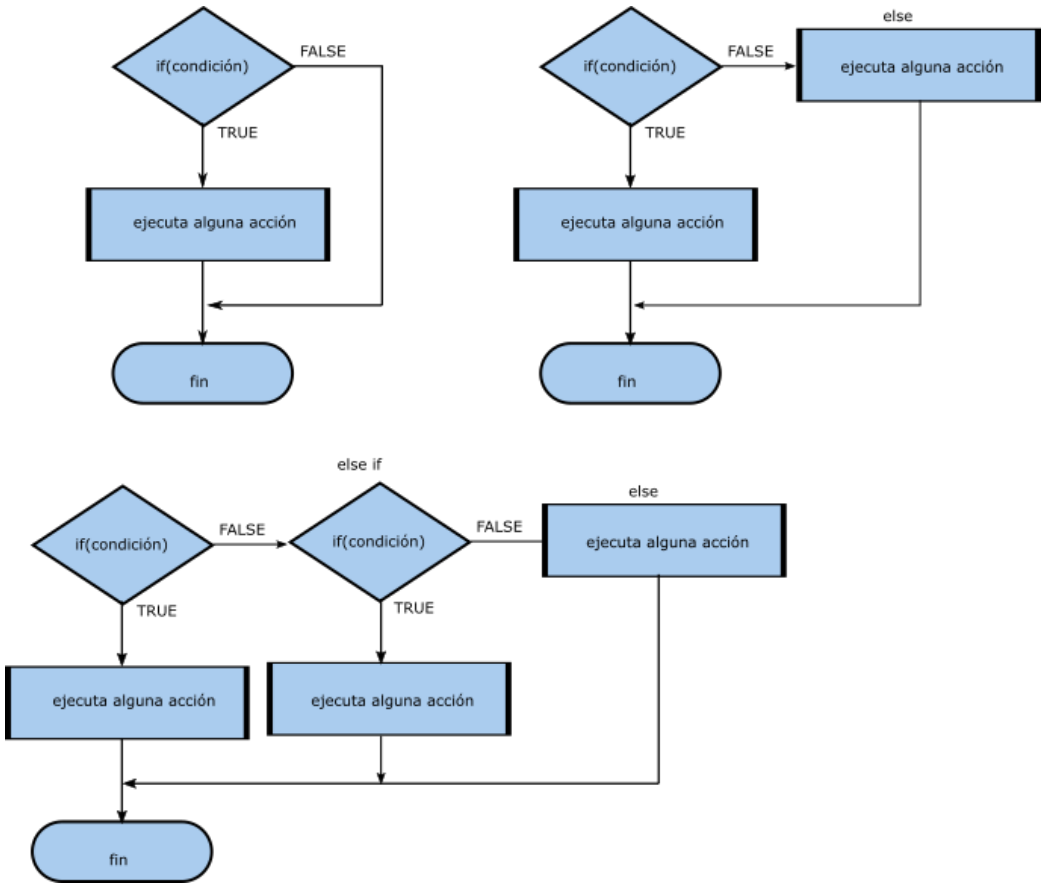


Figura 1: Diagramas de flujo de las estructuras de decisión if

```
if(condicion)
    ejecuta accion;
//************************

if(condicion)
{
    ejecuta accion;
    ejecuta otraAccion;
    .
    .
    .
}
//************************

if(condicion)
{
    ejecuta accion;
}
else
{
    ejecuta accionesPorFalse;
}
```

```
//************************

if(condicion)
{
    ejecuta accion;
}
else if
{
    ejecuta acciones por false 1;

}
else if
{
    ejecuta acciones por false 2;
}
else
{
    ejecuta acciones por defecto
}
```

1.2. Instrucción switch

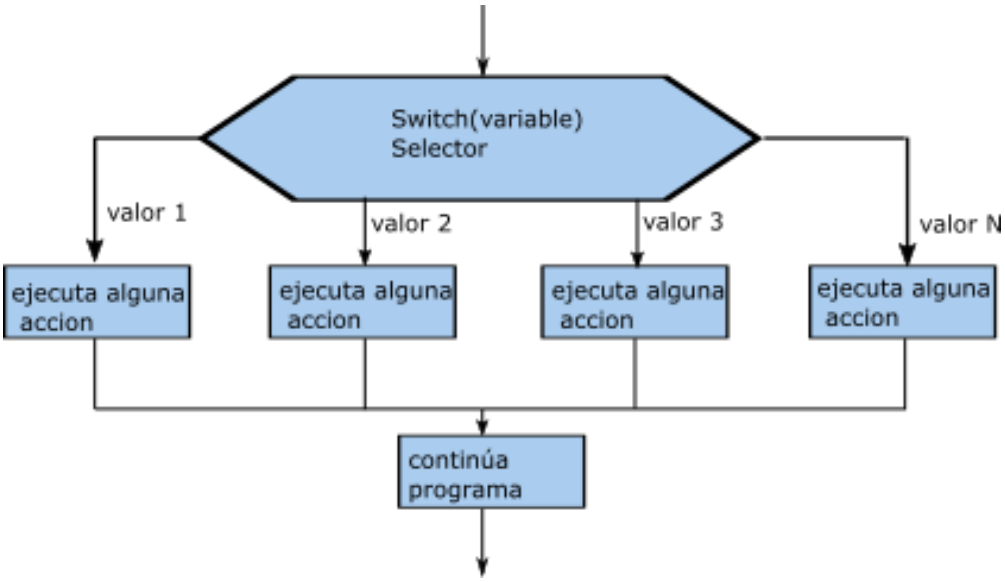


Figura 2: Diagrama de flujo de la estructura de decisión múltiple switch

```
x= a;
switch(x)
{
    case valor1:
        línea de programa1;
        línea de programa2;
        ------------------------;
        break;
    case valor2:
        línea de programa1;
        línea de programa2;
```

```
        ----------------------;
        break;
    default:
        línea de programa1;
        línea de programa2;
        ----------------------;
        break;
}
```

1.3. Operadores lógicos y relacionales

Esta expresión	Es verdadera si
x == y	x es igual a y
x != y	x no es igual a y
x < y	x es menor que y
x > y	x es mayor y
x <= y	x es menor o igual a y
x >= y	x es mayor o igual a y

Operador lógico	Significado
&&	And
||	Or
!	Not

Figura 3: Operadores relacionales y operadores lógicos

2. Bucles de repetición

2.1. Instrucción for()

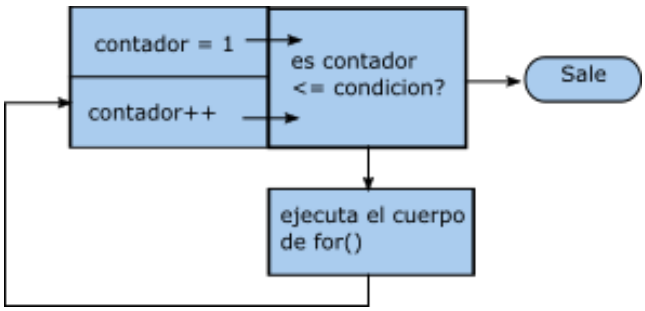


Figura 4: Diagrama de flujo de una instrucción for

```
for(contador=1; contador<=condicion; contador++)
{
    instruccion 1;
    instruccion 2;
    etc;
}
```

2.2. Instrucción while()

```
while ( testea el contador del lazo usando una condicion)
    {
        instruccion 1;
        instruccion 2;
    }
```

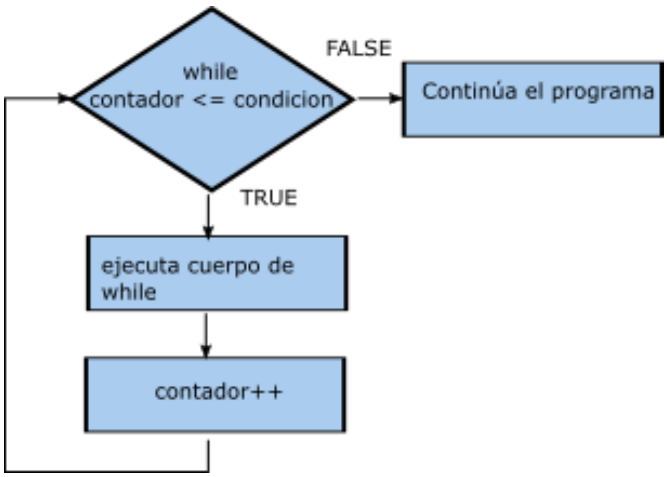


Figura 5: Diagrama de flujo de una instrucción while

2.3. Instruccion do{..} while();

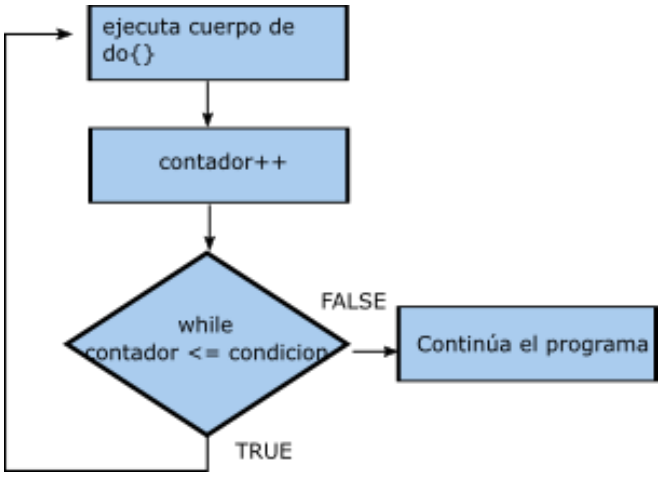


Figura 6: Diagrama de flujo de una instrucción do .. while

```
do
{
    instruccion 1;
    instruccion 2 ;
    .. ;
}while ( condición es TRUE);
```

3. Instruciones break y continue

La instrucción **break** se utiliza para salir de un bucle en forma instantánea sin necesidad de volver a evaluar la condición de entrada de la función.

Ejemplo: Escriba un programa que determine si un número es primo.

```
main( )
{
  int num, i ;
  printf ( "Enter a number " ) ;
  scanf ( "%d", &num ) ;
```

```
  i = 2 ;
  while ( i <= num - 1 )
  {
  if ( num % i == 0 )
  {
  printf ( "No es un numero primo\n" ) ;
  break ;
  }
  i++ ;
  }
  if ( i == num )
  printf ( "Numero primo" ) ;
}
```

La instrucción **continue** nos permite tomar control del lazo y saltar el código hasta la próxima ejecución del bucle.

Ejemplo

```
main( )
{
 int i, j ;
 for ( i = 1 ; i <= 2 ; i++ )
 {
  for ( j = 1 ; j <= 2 ; j++ )
  {
if ( i == j )
  continue ;
  printf ( "\n%d %d\n", i, j ) ;
  }
 }
}
```

4. Funciones

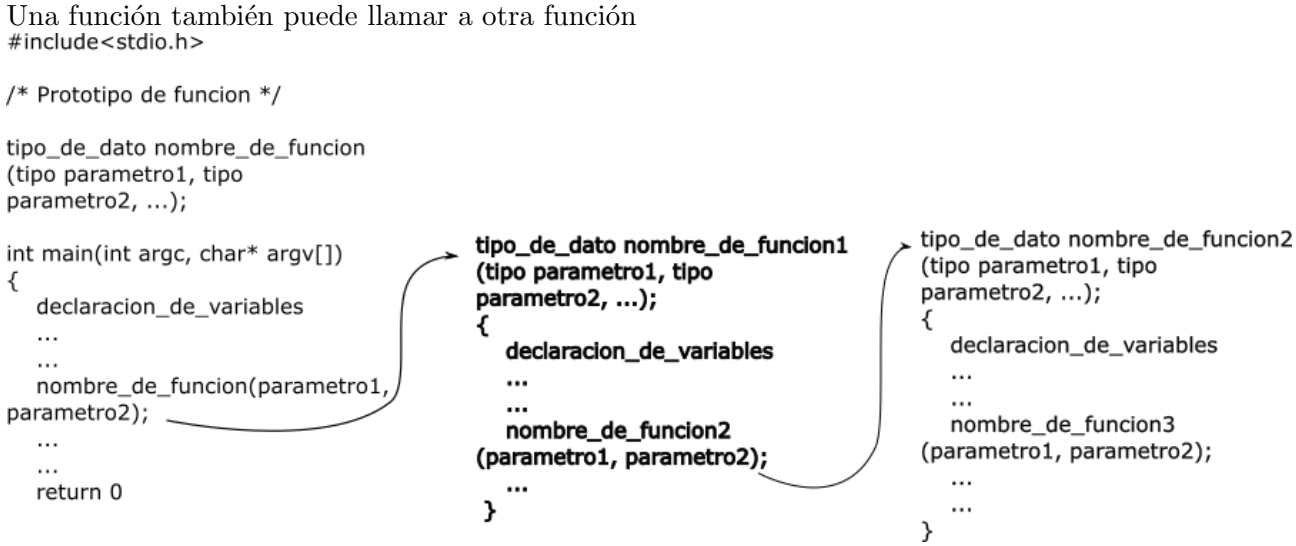
El lenguaje C, como el alumno habrá visto, es un lenguaje basado en funciones que debe tener la función principal *main()* para ser ejecutado. Podemos decir que las funciones son segmentos de programas que ejecutan una o varias tareas. El programador en C debe acostumbrarse a utilizarlas por los siguientes motivos:

1. Brindan claridad en la codificación
2. Permiten reutilizar el software.
3. Permiten aplicar la estrategia de divide y triunfarás, separando un programa complejo en varios problemas más simples.
4. Transportabilidad del código, ya que al solucionar un problema, si se lo hace lo suficientemente general se lo puede llevar a otros programas
5. Posibilidad de crear bibliotecas de funciones, para evitar estar repitiendo programas

El lenguaje C, tiene la función main(), que es la encargada de llamar a las otras funciones que realizarán las acciones, tendrá la forma

```
#include<stdio.h>

/* Prototipo de funcion */

tipo_de_dato nombre_de_funcion(tipo parametro1, tipo parametro2, ...);

int main(int argc, char* argv[])
{
    declaracion_de_variables
    ...
    ...
    nombre_de_funcion(parametro1, parametro2);
    ...
    ...
    return 0

}
```

Una función también puede llamar a otra función



Los valores de los parámetros se le pueden pasar a la función de dos formas

1. **Por valor:** Los valores de las variables son pasados desde la función invocadora (la que llama a la otra función, por ej. main()) a la función invocada.
2. **Por referencia:** Una variable en C tiene asociada un nombre y un valor, al momento de declararla se reserva la memoria necesaria para alojarla, ese lugar de memoria tiene una dirección única para cada variable. En la llamada por referencia las direcciones de las variables son pasadas desde la función invocadora a la función invocada.

Ejemplo. DUP que pase los coeficientes de una ecuación de segundo grado, y devuelva el valor del discriminante. Inicialize el valor del discriminante en la función main(), y verifique si cambiaron luego de invocar a la función, pasando los parámetros por valor y por referencia.

```
#include<stdio.h>

/* Prototipo de funcion */

void discValor(float a, float b, float c, float D);
void discRef(float* a, float* b, float* c, float* D);
```

```
int main(int argc, char* argv[])
{
    float a = 1.0, b = 1.0, c = 1.0, D=0.0;
    printf("El valor de D antes de llamar a la funcion por valor es: %f\n", D);
    discValor(a, b, c, D);
    printf("El valor de D despues de llamar a la funcion por valor y \n antes de llamarla por
    discRef(&a, &b, &c, &D);
    printf("\nEl valor de D despues de llamar a la funcion por por referncia : %f.\n", D);

    getchar();
    return 0;

}

void discValor(float a, float b, float c, float D)
{
    D=b*b - 4.0*a*c;
    printf("\nEl valor de D dentro de la funcion por valor es %f.\n", D);
}


void discRef(float *a, float *b, float *c, float *D)
{
    *D = (*b)*(*b) - 4.0*(*a)*(*c);
    printf("\nEl valor de D dentro de la funcion por referencia es %f.\n", D);//
}
```

5. Ejercicios

. La presente lista de ejercicios resueltos sirven de referencia para practicar. Tenga presente que en algunos casos deberá estar atento para efectuarles modificaciones para lograr una adecuada compilación.

1. Escribir un programa en C, que permita imprimir en la pantalla el típico mensaje introductorio en todos los lenguajes de programación: "Hola Mundo".

```
#include <stdio.h>
/*es igual que "stdio.h", difieren en donde se comienza a buscar el archivo .h*/
int main(void)
{
   printf("Hola mundo\n");
}
```

2. Idem anterior, utilizando una función para ejecutar la misma acción.

```
#include <stdio.h>

void hola(void)

int main( )
```

```
{
   hola ( );
   getchar( );
   return 0;
}

void hola ( )
{
   printf("Hola mundo\n");
}
```

3. DUP que permita imprimir en la pantalla un mensaje con el nombre de esta Universidad.

```
#include <stdio.h>

int main( )
{
   printf("Universidad Tecnológica Nacional\n");
   return 0;
}
```

4. DUP que permita calcular el cuadrado de un número dado como dato, utilizando una función.

```
#include <stdio.h>

void sqr(int);
int main( )
{
   int num;
   num=100; /*100 es el número dado*/
   sqr(num); /*se invoca sqr( ) con num como argumento*/
   getchar( );
   return 0;
}

void sqr(int x) /*en este caso x -parámetro formal- recibe lo que
{
   printf("%d al cuadrado es %d\n",x,x*x);
}
```

5. DUP que permita imprimir en la pantalla, la edad de quien escribe.

```
#include <stdio.h>
int main( )
{
   int edad;
   printf("Ingrese su edad:");
   scanf("%d",&edad);
   printf("Mi edad es %d años\n",edad);
   getchar( );
   return 0;
}
```

6. DUP que permita jugar intentando descubrir un número desconocido, admitiendo el ingreso de propuestas por parte del jugador desde el teclado.

```
#include <stdio.h>
#include<stdlib.h> /*por rand( )  */

int main( )
{
   int aleatorio;
   int intento;
   aleatorio=rand( )%10  /*de 0 a 9*/;
   printf("adivine el numero en juego:");
   scanf("%d",&intento);
   if (intento==aleatorio) /*error si pongo = en vez de = =*/
   {
      printf("correcto,%d es el nro. en juego\n",aleatorio);
   }
   getchar( );
   return 0;
}
```

7. Perfeccionar el programa anterior incorporando opciones que permitan enviar mensajes en caso de error o aproximación.

```
#include <stdio.h>
#include<stdlib.h>
#include<time.h>  /*#include <sys/time.h       */

int main( )
{
   int aleatorio, intento;
   srand(time(0)); //genera valor de referencia basado en el reloj del sistema
   aleatorio=rand( ) %10; /*de 0 a 9*/;
   printf("adivine el numero en juego:");
   scanf("%d",&intento);
   if (intento == aleatorio) /*Cuidado: error si pongo = en vez de ==*/
   {
      printf("correcto,%d es el nro. en juego\n",aleatorio);
   }
   else
   {
      printf("Incorrecto ");
      if (intento>aleatorio)
         printf("demasiado alto,el número era %d\n",aleatorio);
      else
         printf("demasiado bajo,el número era %d\n",aleatorio);
}
getchar( );
return 0;
}
```

8. DUP que permita efectuar la transformación de unidades de medida dadas en valores enteros de pies a metros lineales.

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

```
int main( )
{
   int pies;
   float metros; /*separadas por comas las variables de punto flotante*/
   printf("Introduzca el número de pies: ");
   scanf("%d",&pies); /*se lee un entero*/
   metros=pies *0.3048; /*fórmula de conversion de pies a metros*/
   printf("%d pies son %f metros\n",pies,metros);
   getchar( );
   return 0;
}
```

9. Reescribir el programa anterior para que permita efectuar la transformación de unidades de medida dadas en punto flotante para los pies.

```
#include <stdio.h>
int main( )
{
   float pies,metros; /*separadas por comas las variables*/
   printf("Introduzca el número de pies: ");
   scanf("%f",&pies); /*se lee un float*/
   metros = pies *0.3048; /*fórmula de conversion de pies a metros*/
   printf("%f pies son %f metros\n",pies,metros);
   getchar( );
   return 0;
}
```

10. DUP que permita multiplicar dos números dados.

```
# include <stdio.h>

float mul(float a, float)
int main ( )
{
   float respuesta;
   respuesta = mul(3.0,4.0); /*en mul se ponen los argumentos*/
   printf("La respuesta es %.2f\n", respuesta);
   getchar( );
   return 0;
}

float mul(float a, float b)
{
   return a*b; /*retorna un valor */
}
```

11. DUP que calcule el cociente y el resto de dos números entero que se introduce desde el teclado.

```
#include <stdio.h>
int main( )
{
```

```
   int x,y;
   printf("Introduzca el dividendo y el divisor:");
   scanf("%d%d",&x,&y);
   printf("El cociente de la divisioon es %d y el resto es %d.",x/y, x%y); /*El operador %
   getchar( );
   return 0;
}
```

12. DUP que permita imprimir los números pares entre 1 y 100.

```
#include <stdio.h>
int main( )
{
   int i;
   for (i=1;i<=100;i++) /*! es el operador lógico NOT, que invierte el resultado*/
   if(!(i%2)) /*la operación % módulo da falso=0, cuando se utilice un número par*/
      printf("%d",i);
   getchar();
   return 0; }
```

13. DUP que permita calcular el producto de dos números, y verificar si uno de ellos es negativo.

```
#include <stdio.h>
int main( )
{
   int x,y, producto;
   printf("Introduzca dos números:");
   scanf("%d%d",&x,&y);
   if((producto= x * y) < 0)
      printf("Uno de los números es negativo\n");
   else
      printf("El producto positivo es %d", producto);
   getchar( );
   return 0;
}
```

14. DUP que permita visualizar el resultado de la aplicación de distintos códigos de barra invertida dentro de printf para la operación de división entre dos números enteros (Ej: 99/2 y enero)

```
#include<stdio.h>
void main( )
{
   int i, j;
   printf("Introduzca dos números:");
   scanf("%d%d", &i,&j);
   printf("%d ", i/ j); /*d: número base diez=49*/
   printf("%e", i/ j); /*e: exponencial4.95+01*/
   printf("%f", i/j); /*f: punto flotante =49.5000*/
   printf("%6.2f", i/ j); /*f: punto flotante =49.50*/
   printf("%g", i/j); /*g: en d y e elimina ceros=49.5*/
   printf("%o", i); /*o: octal =143*/
```

```
   printf("%x", i); /*x: número hexadecimal s/signo=63*/
   printf("%s", "enero"); /*s: cadena=enero*/}
   getchar();
   return 0;
}
```

15. DUP que calcule e imprima, las raíces cuadradas de los 15 primeros números enteros.

```
#include <stdio.h>
#include <math.h>/*inclusión de librería con funciones matemáticas*/

int main( )
{
   int i;
   for (i=1; i<16; i=i+1)
   {
      printf("La raíz cuadrada de %2d es = a %6.4f \n", i, sqrt(i));
   }
   getchar( );
   return 0;
}
```

16. Reescribir el programa anterior utilizando la instrucción while.

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

int main( )
{
   int i; /*declaración de variable:nombre y tipo*/
   i = 1; /*asignación de valor inicial a la variable*/
   while (i<16)
   {
      printf("La raíz cuadrada de %2d es = a %6.4f\n",i,sqrt(i));
      i++;
   }
   getchar( );
   return 0;
}
```

17. DUP que permita efectuar la conversión de temperaturas de grados fahrenheit a grados celsiuss.

```
#include<stdio.h>

int main( )
{
   int fahr,celsius,lower,upper,step;
   lower=0;
   upper=140;
   step=20;
   fahr=lower;
   while (fahr<=upper)
   {
```

```
      celsius=5*(fahr-32)/9;/*fórmula de equivalencia*/
      printf("%d\t%d\n",fahr,celsius);
      fahr=fahr+step;
   }
   getchar( );
   return 0;
}
```

18. Modificar el programa anterior para que permita la manipulación de datos decimales.

```
#include<stdio.h>

int main( )
{
   float fahr,celsius;
   int lower,upper,step;
   lower=0;
   upper=140;
   step=20;
   fahr=lower;
   while (fahr<=upper)
   {
       celsius=(5.0*(fahr-32.0))/9.0;
       printf("%3.0f%6.1f\n",fahr,celsius);
       fahr=fahr+step;
   }
   getchar( );
   return 0;
}
```

19. Diseñar el mismo programa pero utilizando un bucle FOR.

```
#include<stdio.h>

int main( )
{
   int fahr;
   for(fahr=0;fahr<=300;fahr=fahr+20)
      printf("%3d %6.1f\n",fahr,(5.0/9.0)*(fahr-32.0));
   getchar( );
   return 0;
}
```

20. DUP que muestre en pantalla los múltiplos de 2 entre 0 y 40.

```
#include<stdio.h>

int main( )
{
   int mult, tope, cont;
   mult=0;
```

```
   cont=0;
   tope=40;
   while (mult < tope)
   {
      mult = cont *2;
      printf ("%4d\n", mult);
      ++cont;
   }
   getchar( );
   return 0;
}
```

21. DUP que capture una cadena de caracteres de la entrada y la imprima en la pantalla.

```
#include<stdio.h>

int main( )
{
   char cadena;
   while ((cadena =getchar ( )) != '\n')
   {
      putchar (cadena);
   }
   getchar( );
   return 0;
}
```

22. DUP que imprima en pantalla el número de caracteres tecleados y el número de líneas empleadas hasta que se pulse Çtrl Z".

```
#include<stdio.h>

int main( )
{
   int cont, lin, ca;
   cont =0;
   lin=0;
   ca= 0;
   while ((ca =getchar ( )) != EOF)
   {
      ++ cont;
      if (ca == '\n'
      {
         ++ lin;
      }
   }
   printf ("%2d \n", EOF);
   printf ("%3d caracteres tecleados.  \n", cont);
   printf ("En %2d líneas. \n", lin);
   getchar( );
   return 0;
}
```

23. Obtener una tabla del código ASCII.

```
#include<stdio.h>
int main( ) {
int x;
char = ca;

for (x = 32; x < = 255; ++x) {
ca = x;
printf ("/%d...%c/ \t", x, ca);
}
getchar( );
return 0; }
```

6. Problemas Propuestos

1. En la página web `https://docs.microsoft.com/en-us/cpp/c-language/cpp-integer-limits?view=vs-2019`, se encuentran los valores límites para cada tipo de variable. DUP que muestre los valores límites de su computadora. Por ejemplo:

   ```
   printf("El minimo valor de un int es: ", INT_MIN);
   ```

2. DUP que visualice la aplicación de los distintos operadores relacionales sobre dos enteros ingresados por teclado.(operador = = igual; operador != distinto; operador ¡= menor o igual que;operador ¿= mayor o igual que;operador ¡menor que;operador ¿mayor que.

   ```
   /*las expresiones relacionales y lógicas producen un resultado que es solamente 1 -cierto-
   ```

3. DUP que imprima en pantalla el número de caracteres y el número de líneas ingresados por teclado. La ejecución debe concluir al pulsar las teclas Çontrol+Z".

4. DUP que determine, de un vector de cinco elementos con los números 20, 21, 22, 23, 24, en que elemento del vector se guarda cada número y en que posición de memoria se halla cada uno, mostrando en la pantalla el resultado.

5. DUP que determine si un número introducido por teclado es igual a uno cualquiera de los elementos de un vector determinado, cuyo contenido es 20, 6, 7, 25, 49, 28.

6. DUP un programa que tomados los valores de una ecuación de segundo grado saque sus raíces, teniendo en cuenta que pueden ser imaginarias

7. DUP un programa en C que , al recibir como dato una X cualquiera, calcule el cos(x) utilizando la siguiente serie:

   $$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

   La diferencia entre la serie y un nuevo término debe ser menor o igual a 0,001. Imprima el número de términos requerido para obtener esta precisión

   Ayuda: La serie queda totalmente representada como

   $$\sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k}}{2k!}$$

8. DUP que determine mediante serie de taylor el $\sin(x)$, $\tan(x)$ y $\ln(x)$.

9. Un número N es primo si los únicos enteros positivos que lo dividen son exactamente 1 y N. Escribe un programa en C que , al recibir como dato un número entero positivo, determine si este es un número primo.

10. Se dice que un número es considerado perfecto si la suma de sus divisores excepto el mismo, es igual al propio número. Escriba un programa que obtenga e imprima todos los números erfectos comprendidos entre 1 y N.

11. Escriba una función que dado cualquier número N, imprima sus divisores primos.

12. Escribe un programa en C que, al recibir como dato un arreglo de números enteros:

    a) Lo imprima en forma inversa

    b) Lo ordene de mayor a menor

    c) Lo ordene de menor a mayor

13. DUP que, al recibir un arreglo unidimensional de tipo real que contiene mediciones, calcule lo siguiente:

    a) La media aritmética.

    b) La varianza.

    c) La desviación estándar.

    d) La moda. Se calcula obteniendo el número con mayor frecuencia.

    e) La mediana

    f) Optimize los resultados mediante una regresión lineal de mínimos cuadrados. (Si no los vio en Física I, solicite al docente las ecuaciones)

14. Diseñe un programa que dado un arreglo unidimensional, determine si existe un elemento del arreglo.

    a) Realizando una búsqueda secuencial

    b) Realizando una búsqueda binaria

15. En un experimento de laboratorio se quiere obtener el índice de refringencia de un material, para ello se realizan distintas mediciones del ángulo incidente y del ángulo refractado, que se muestran en la siguiente tabla

Angulo de incidencia (i)	$Angulode refraccion(r)$
$0,0$	$0,0$
$5,0$	$4,0$
$10,0$	$8,0$
$15,0$	$11,0$
$20,0$	$14,0$
$25,0$	$17,0$
$30,0$	$20,0$
$35,0$	$23,5$
$40,0$	$27,0$
$45,0$	$30,0$
$50,0$	$32,5$

    a) Calcule el índice de refracción con su error

    b) Investigue para averiguar que tipo de material es

    c) Realice un ajuste de mínimos cuadrados para obtener su índice de refringencia

d) Calcule el coeficiente de correlación.

e) ¿Que puede decir de la calidad de la medición?

Para la regresión lineal tendremos que adecuar las ecuaciones para que sean las de una recta de la forma $y = ax + b$. Por ley de Snell sabemos que $n_i \sin(\theta_i) = n_r \sin(\theta_r)$, si consideramos que el rayo de luz se desplaza por el aire antes de ser refractado tendremos que $n_i = 1$ y:

$$n_i \sin(\theta_i) = n_r \sin(\theta_r)$$

o

$$\sin(\theta_r) = \frac{1}{n_r} \sin(\theta_)$$

esta ecuación la linealizo en la forma $y = ax+b$ con $b = 0$, donde $y = \sin(\theta_r)$, $a = \frac{1}{n_r}$ y $x = \sin(\theta_i)$.

Las ecuaciones para el cálculo de los coeficientes son:

$$\bar{a} = \frac{N \sum x_i y_i - \sum x_i \sum y_i}{N \sum x_i^2 - \left(\sum x_i\right)^2}$$

$$\bar{b} = \frac{\sum y_i \sum x_i^2 - \sum x_i \sum y_i}{N \sum x_i^2 - \left(\sum x_i\right)^2} = \frac{\sum y_i - a \sum x_i}{N}$$

$$\sigma_y^2 = \frac{\sum (y_i - a\, x_i - b)^2}{N - 2}$$

Conocida la varianza de y, los errores de los parámetros a y b vienen dados por las siguientes expresiones:

$$\sigma_a^2 = \frac{\sum (y_i - a\, x_i - b)^2}{N - 2} \frac{1}{\sum (x_i - \bar{x})^2}$$

$$\sigma_b^2 = \frac{\sum (y_i - a\, x_i - b)^2}{N - 2} \frac{\sum x_i^2}{N \sum (x_i - \bar{x})^2}$$

El coeficiente de correlación se calcula mediante

$$r = \frac{\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{\left[N \sum x_i^2 - \left(\sum x_i\right)^2\right] \left[N \sum y_i^2 - \left(\sum y_i\right)^2\right]}}$$

16. Realice una prueba de escritorio para determinar la salida de las siguientes funciones.

a)
```
main( )
{
   int j ;
   while ( j <= 10 )
   {
      printf ( "\n%d", j ) ;
      j = j + 1 ;
   }
}
```

b)
```
main( )
{
  int i = 1 ;
  while ( i <= 10 ) ;
  {
  printf ( "\n%d", i ) ;
i++ ;
  }
}
```

c)
```
main( )
{
  int j ;
  while ( j <= 10 )
  {
  printf ( "\n%d", j ) ;
  j = j + 1 ;
}
}
```

d)
```
main( )
{
  int x = 1 ;
  while ( x == 1 )
  {
  x = x - 1 ;
  printf( "\n%d", x ) ;
  }
}
```

e)
```
main( )
{
  int x = 1 ;
  while ( x == 1 )
  x = x - 1 ;
  printf ( "\n%d", x ) ;
}
```

f)
```
main( )
{
  char x ;
  while ( x = 0 ; x <= 255 ; x++ )
  printf ( "\nAscii value %d Character %c", x, x ) ;
}
```

g)
```
main( )
{
  int x = 4, y, z ;
  y = --x ;
  z = x-- ;
  printf( "\n%d %d %d", x, y, z ) ;
}
```

h)
```
main( )
{
  int x = 4, y = 3, z ;
  z = x-- -y ;
  printf ( "\n%d %d %d", x, y, z ) ;
```

```
}
```

i)
```
main( )
{
  while ( 'a' < 'b' )
  printf ( "\nmalyalam es un palindromo" ) ;
}
```

j)
```
main( )
{
  int i = 10 ;
  while ( i = 20 )
  printf ( "\nLoop infinito!" ) ;
}
```

k)
```
main( )
{
 int i ;
 while ( i = 10 )
 {
  printf ( "\n%d", i ) ;
  i = i + 1 ;
 }
}
```

l)
```
main( )
{
 float x = 1.1 ;
 while ( x == 1.1 )
 {
  printf ( "\n%f", x ) ;
  x = x { 0.1 ;
 }
}
```

m)
```
main( )
{
 char x ;
 for ( x = 0 ; x <= 255 ; x++ )
  printf ( "\nValor Ascii %d caracter %c", x, x ) ;
}
```

n)
```
main( )
{
 int x = 4, y = 0, z ;
 while ( x >= 0 )
 {
  x-- ;
  y++ ;
  if ( x == y )
continue ;
  else
  printf ( \\n%d %d", x, y ) ;
 }
}
```

ñ)
```
main( )
{
 int x = 4, y = 0, z ;
```

```
        while ( x >= 0 )
        {
         if ( x == y )
         break ;
         else
         printf ( \\n%d %d", x, y ) ;
         x-- ;
         y++ ;
        }
       }
```

7. Anexo

7.1. Ingreso de caracteres especiales en Linux.

La forma de insertar caracteres especiales, no es en formato ASCII sino Unicode.

-Una manera de insertarlos es con la combinación:

`CTRL+SHIFT+u+(código Unicode)`

El código Unicode es de cuatro números en hexadecimal manteniendo presionadas continuamente sólo las teclas CTRL y SHIFT(mayúsculas), y tipeando las demás. Al soltar CTRL y SHIFT, el caracter se insertará.

Para conocer el código Unicode de un caracter, se puede ejecutar el Mapa de caracteres (desde el menú Aplicaciones de Gnome o con el comando charmap, ó Menú $->$ Accesorios $->$ Mapa de Caracteres), buscar el caracter deseado y fijarse en la barra de estado que va a aparecer algo como Ü+004A LATIN CAPITAL LETTER J"(por ejemplo para la letra J). Esos números después del U+ son los que deben escribir con la combinación de teclas anteriores.

-Otra forma es copiando los caracteres deseados desde el mismo Mapa de caracteres.

-También Utilizando [Alt Gr], como en Windows: Utilice este método para obtener:
$[Alt + Gr]$+[Ñ] $= \sim$
$[Alt + Gr]$+[Z] $= \ll$
$[Alt + Gr]$+[X] $= \gg$
$[Alt + Gr]$+[V] = "
$[Alt + Gr]$+[B] = "
$[Alt + Gr]$+[,]= guión largo

7.2. Comandos basicos para crear un programa en Linux

1. Mediante el comando pwd averiguamos en cual directorio estamos ubicados

```
clasesUTN@clases:~$ pwd
/home/JLM
```

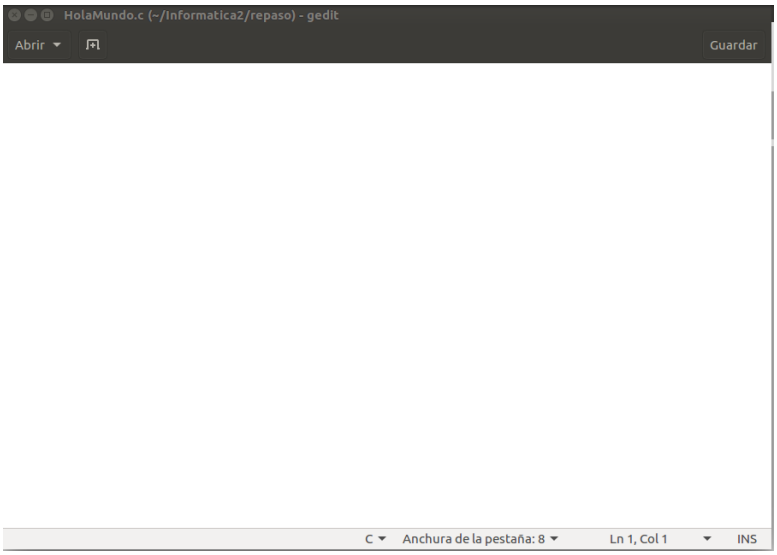
2. A continuación creamos el directorio donde vamos a guardar nuestro programa con el comando mkdir

```
clasesUTN@clases:~$ mkdir /home/JLM/Informatica2/Repaso
clasesUTN@clases:~$ cd /home/JLM/Informatica2/Repaso
clasesUTN@clases:~/Informatica2/Repaso$
```

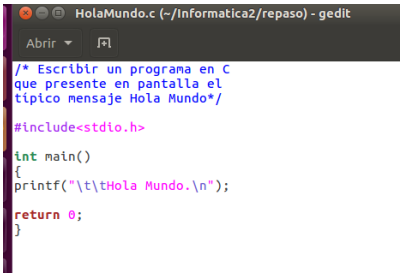
3. Utilizamos el editor de texto gedit

```
clasesUTN@clases:~/Informatica2/Repaso$ gedit HolaMundo.c
```

4. Se abre la ventana del programa



5. Escribimos nuestro programa



6. Para compilar el programa volvemos al terminal y ejecutamos la siguiente orden

```
clasesUTN@clases:~/Informatica2/Repaso$ gcc -Wall HolaMundo.c -o HolaMundo
clasesUTN@clases:~/Informatica2/Repaso$ ls
HolaMundo  HolaMundo.c
```

7. Vemos que al ejecutar el comando ls para que muestre el contenido del directorio, ahora también está presente el archivo HolaMundo, que va a ser nuestro archivo ejecutable.
Para correr el programa se antepone ./ al nombre del archivo.

```
clasesUTN@clases:~/Informatica2/Repaso$ ./HolaMundo
                Hola Mundo.
```

8. El comando cat permite ver el interior del archivo

```
clasesUTN@clases:~/Informatica2/Repaso$ cat HolaMundo.c
/* Escribir un programa en C que presente el tipico
mensaje Hola Mundo*/
#include<stdio.h>

int main()
{
        printf("\t\t Hola Mundo.\n");
        return 0;
}
```

9. Cuando utilizamos funciones de la biblioteca matemática, puede suceder que al compilar no encuentra la función sqrt() por ejemplo. Para solucionar este error se debe agregar -lm al final de la instrucción de compilación.

```
clasesUTN@clases:~/Informatica2/Repaso$ gcc -Wall ej_0_16.c -o ej_0_16 lm
```