Universidad Autónoma Metropolitana

Unidad Iztapalapa

Introducción a la Programación

Guía Básica para el uso de C

Trimestre 2008-I

Profesor: Jorge Luis Ramírez Ortiz

Índice general

Capítulo 1

Primeros pasos con C

1.1. Instalando el Compilador de C

Para nuestro curso de Introducción a la programación utilizaremos el archivo TC.zip, que se puede descargar fácilmente de la página web del curso.

http://mcyti.izt.uam.mx/ gustavo/index.html

A continuación descomprimimos el archivo y lo dejamos en la raíz de nuestro disco duro en la carpeta con el nombre TC. Como se puede observar no se necesita de ningún tipo de instalación especial para utilizar C.

En nuestra unidad raíz en la ruta C:/TC/BIN, tenemos el ejecutable TC.EXE, damos doble click y se nos presentaría una pantalla como la que podemos observar en la figura 1.1. Es necesario aclarar que estaremos utilizando la versión 3.0 de Turbo C++.

1.2. Ajustando los Directorios de Salida

Del menú seleccionamos Options, luego Directories y nos aparecerá una pantalla como la que podemos ver en la figura 1.2.

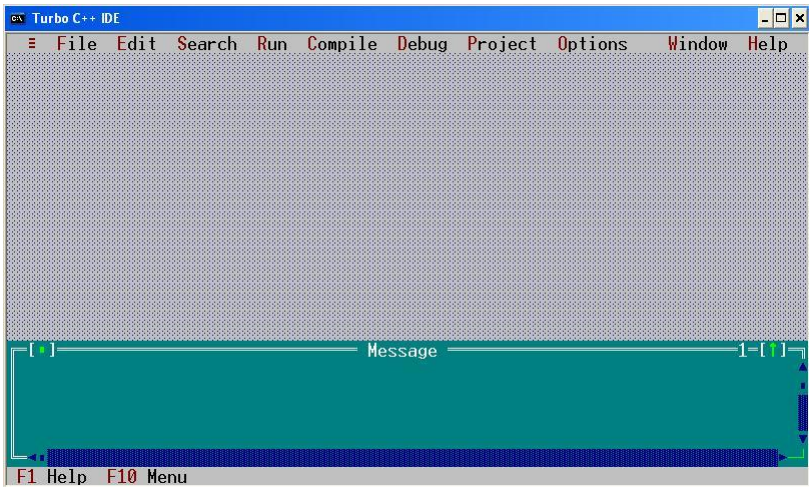

Figura 1.1: Pantalla inicial de C

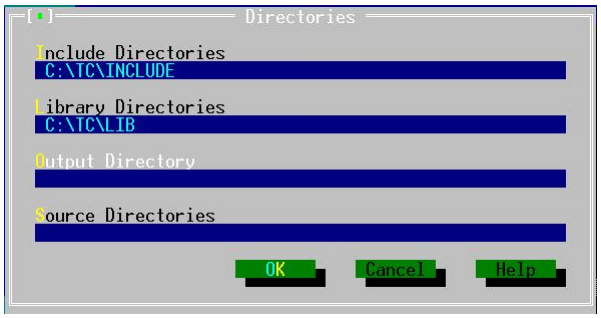

Figura 1.2: Pantalla de directorios

En el campo de texto que se titula Output Directory colocamos la dirección donde estaremos generando todos los archivos que son resultado de la compilación como la creación de archivos .OBJ y .EXE. Como ejemplo podríamos tener la siguiente dirección:

C:/estudio/introduccion/destino

En el campo que se encuentra enseguida que se titula Source directories, aquí colocaremos la dirección desde donde se buscarán nuestros archivos que estaremos tecleando. Como ejemplo ponemos la siguiente dirección:

C:/estudio/introduccion/fuente

Nuestros campos quedarían como se muestra en la figura 1.3

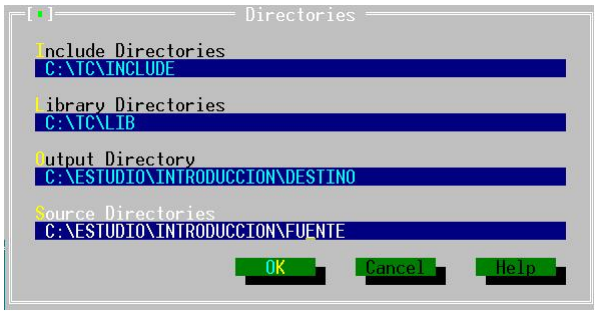

Figura 1.3: Ajuste para fuente y destino en C

Así presionamos OK, y quedará ajustado nuestro entorno para poder programar y encontrar sin ningún problema nuestros archivos.

Despues en el menú de este editor damos clic en File-New, observaremos que nos presenta un ejemplo, lo que aqui se recomienda es cerrarlo y volver a pedir otro nuevo documento. Para que obtengamos un archivo totalmente en blanco y procedamos a escribir nuestro primer ejemplo.

A continuación mostramos el primer programa a compilar y ejecutar con esta versión que estamos utilizando de Turbo C. Guardamos el siguiente código bajo el nombre saludo.c

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

  void main()
  {
    clrscr();//función para limpiar la pantalla

    //mostrará un mensaje en la línea de comandos.
    printf("Hola a todos los Alumnos de Introducción");

    getch();//función que obtiene un carácter de consola
  }
```

Es importante mencionar que debido a la versión que estamos utilizando

de C, es muy conveniente que el nombre del archivo fuente no exceda los ocho caracteres. De lo contrario el nombre no será legible cuando deseemos abrirlo desde el editor de C. Y la extensión de dichos archivos será .c .

El código cuenta con unos detalles importantes a considerar, como lo es include, el cuál me permite traer librerias. Dichas librerias stdio.h y conio.h, contienen los métodos clrscr(), printf() y getch(). En dado caso que no incluyeramos esas librerias el compilador nos marcaría un error. Ya que no encontraría la definición de los métodos que estoy llamando desde mi programa. Aquí es importante hacer notar que en nuestro programa tenemos el método main(), el cuál nos permite comenzar la ejecución de un programa.

1.3. Compilar y Ejecutar

Tomando el código que se mostró anteriormente, procederemos a compilar y ejecutar. Primero para compilar mi archivo fuente, seleccionamos del menú Compile, la opción que se llama compile. O bien podemos presionar ALT+F9. Deberemos ver los resultados de la compilación como se muestra en la figura 1.4.

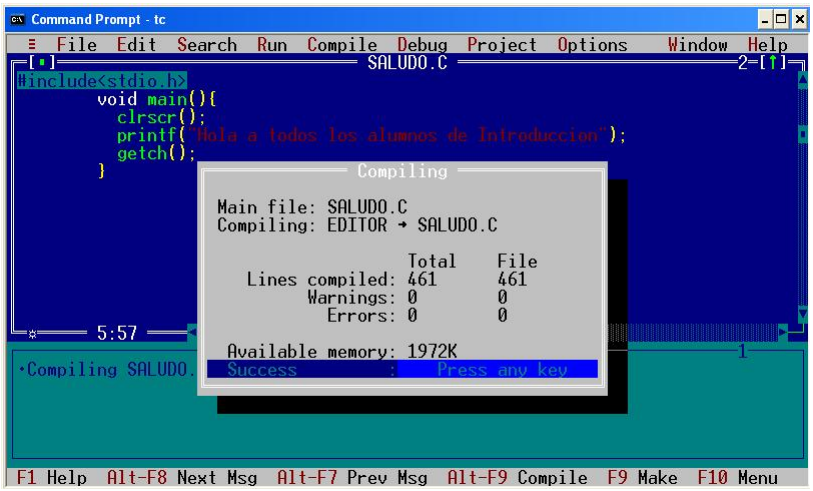

Figura 1.4: Compilando código en C

Para ejecutar el programa seleccionamos Run y de ahí elegimos la misma

opción la opción del mismo nombre (Run). El programa procederá a ejecutarse, y mostrará un mensaje en la línea de comandos como se puede observar en la figura 1.5.

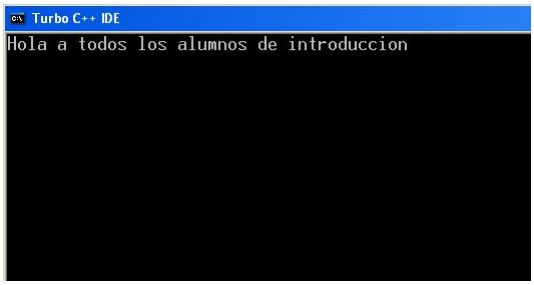

Figura 1.5: Ejecutando un programa en C

También podemos ejecutar nuestro programa, con la combinación de teclas CTRL+F9. Hasta este punto hemos visto el ajuste del entorno de programación así como la compilación y ejecución de un primer programa en C.

1.4. Tipos de Datos y Declaración de Variables

Los tipos de datos básicos que nos ofrece C son los siguintes:

- char carácter
- short entero corto
- int entero
- long entero largo
- float para representar decimales (puntos flotantes)
- double punto flotante de doble precisión

A continuación presentamos un programa que calcula el promedio de calificaciones. Lo guardaremos bajo el nombre promEnt.c

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

  void main()
  {
    clrscr(); //función para limpiar la pantalla

    int valor1=8;
    int valor2=10;
    int valor3=9;
    int valor4=8;
    int valor5=8;
    int resultado=(valor1+valor2+valor3+valor4+valor5);

    printf("El promedio de calificaciones es: \n");
    printf("\n");
    printf("\t %1d ",resultado);

    getch();//función que obtiene un caracter de consola.
  }
```

Los puntos importantes a considerar aquí, son la forma en que declaramos las variables, primero colocamos el tipo de variable que es int y luego el nombre o identificador de la variable que se llama valor1 y con el signo igual, le asignamos el valor de 8.

Procedemos a compilar y ejecutar nuestro programa en C, y obtendríamos algo como lo que se muestra en la figura 1.6.

Como hemos podido observar el resultado del código anterior nos genera un resultado entero, cuyo valor es 8. Podemos ver el resultado gracias a la siguiente línea de código.

```
printf("\t %1d ",resultado);  // %1d le dice a C que queremos en pantalla un e
```

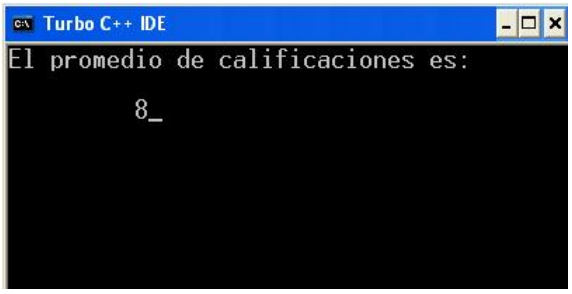

Figura 1.6: Calculando el promedio con un programa en C

¿Que sucedió con la parte fraccionaria? Bueno lo que realmente tenemos que hacer es trabajar con variables de tipo flotante. Asi que modifiquemos nuestro código anterior, cambiando nuestras variables de tipo int a tipo float. Este código lo guardaremos bajo el nombre promFlot.c

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

  void main()
  {
    clrscr();

    float valor1=8;
    float valor1=8;
    float valor2=10;
    float valor3=9;
    float valor4=8;
    float valor5=8;
    float resultado=(valor1+valor2+valor3+valor4+valor5);

    printf("El promedio de calificaciones es: \n");
    printf("\n");
    printf("\t %1.1f ",resultado);

    getch();//función que obtiene un caracter de consola.
  }
```

Ahora si, cuando ejecutamos nuestro programa podemos observar el resultado correcto de calcular el promedio, mostrado en la figura 1.7.

Figura 1.7: Calculando el promedio usando variables flotantes

Aquí con la sentencia:

```
printf("\t %1.1f ",resultado);
```

le estamos diciendo a nuestro programa que nos presente un resultado con punto decimal, con un decimal de precisión, es decir que después del punto décimal sólo un digito se mostrará en pantalla.

1.5. Condicionantes if-else

En el lenguaje C no podría faltar los famosos if-else para poder probar condiciones dentro de nuestro programa. Veamos el siguiente código. Lo guardamos bajo el nombre ifElse.c

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

void main()
```

```
{
  int numero=10;

  if(numero<0){
    printf("nuestro numero es negativo");
  }else{
    printf("nuestro numero es positivo");
  }
  getch();
}
```

En nuestro código anterior podemos ver el uso de los condicionales if-else, como nuestro número es positivo, la condición en el if es falsa, así el programa ejecuta el else y muestra en pantalla un mensaje, como se muestra en la figura 1.8.

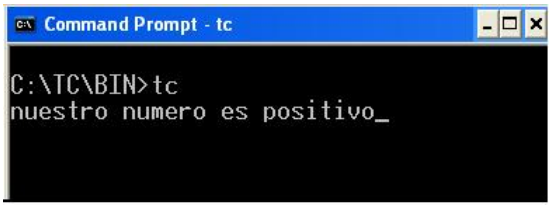

Figura 1.8: Mensaje producido por el else del bloque if-else

1.6. Uso de ciclos while, do-while, for

Para poder ejecutar un determinado número de veces una sentencia, necesitamos del uso de ciclos, en C contamos con 3 ciclos, que estaremos exponiendo en las siguientes secciones.

1.6.1. Utilizando while

La mejor forma en que podemos explicar el funcionamiento de while, sería con el siguiente programa:

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

  void main()
  {
    clrscr();
    int contador=0;

    while(contador<10){
      printf("El valor de contador es: %d",contador);
      contador++;
    }

    getch();
  }
```

Lo guardamos bajo el nombre while.c Compilando y ejecutando el programa anterior obtendríamos lo que se muestra en la figura 1.9.

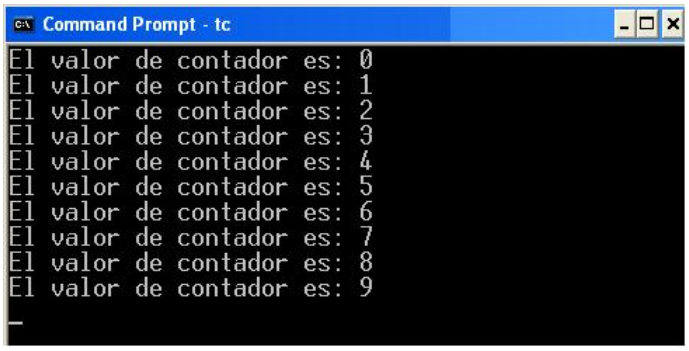

Figura 1.9: Presentación de datos en pantalla utilizando el ciclo while

Del ejemplo el ciclo while lo podríamos leer de la siguiente manera: "Mientras contador sea menor que 10 sigue iterando", Como podemos ver el ciclo termina cuando contador vale 10, y se hace falsa la condición del while. Para que podamos hacer que se incremente el contador usamos el operador unario ++, después de la variable llamada "contador" .

1.6.2. Ciclo for

Ahora daremos un vistaso a lo que es el ciclo for, y presentemos una variante del programa anterior cambiando el while por el for. Tenemos así el siguiente código.

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

  void main()
  {
    clrscr();

    for(int contador=0;contador<10;contador++){
      printf("el valor de contador es: %d",contador);
      printf("\n");
    }

    getch();
  }
```

Lo guardamos bajo el nombre for.c Como podemos observar en el código, la sintáxis de for es diferente a la de while. Primero que nada podenmos definir la variable tanto en tipo como su nombre, después de un ";"va la condición que verifica que mientras sea menor que diez que siga iterando, despues de otro ";", va lo que es el incremento del contador. En esencia el resultado en pantalla es el mismo que el que pudimos observar en la figura 1.9.

1.6.3. Ciclo do-while

Este ciclo es ligeramente diferente a los dos ciclos expuestos anteriormente. Antes de hablar de la diferencia, veamos el siguiente código.

```
#include<conio.h>
```

```
#include<stdio.h>

  void main()
  {
    clrscr();

    int contador=0;

    do{
      printf("El valor de contador es: %d",contador);
      printf("\n");
      contador++;
    }while(contador<10)

    getch();
  }
```

lo guardamos bajo el nombre doWhile.c Ahora expliquemos la diferencia. En caso de que la condición fuera falsa, el código que se encuentra en el bloque do-while al menos se ejecuta una sola vez. El resultado en pantalla se vería igual al que muestra la figura 1.9.

1.7. Arreglos

Un arreglo podemos definirlo como una colección de variables del mismo tipo. Por ejemplo podemos crear arreglos que guarden valores enteros, caracteres o bien valores flotantes. Veamos el código que se muestra a continuación para después aclarar algunos puntos importantes.

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

void main()
{
  clrscr();
```

```
  int arreglo[10];
  //inicializamos el arreglo
  for(int indiceArreglo=0;indiceArreglo<10;indiceArreglo++){
    arreglo[indiceArreglo]=indiceArreglo;
  }
  //mostramos los valores guardados dentro del arreglo
  for(int indice=0;indice<10;indice++){
    printf("%d",arreglo[indice]);
  }
  getch();
}
```

Lo guardamos bajo el nombre arreglos.c En el código declaramos nuestro arreglo como:

int arreglo[10];

Con esto le decimos a C, que estamos declarando un arreglo que guardará enteros y que además el tamaño del arreglo será de 10; El primer for de nuestro código, nos permite inicializar nuestro arreglo de enteros. Ya que debemos recordar que si no hacemos esta operación nuestro arreglo contendrá elementos basura. Todo arreglo dentro de C lo podemos recorrer utilizando un índice, en nuestro primer for dicho índice recibe el nombre de indiceArreglo. Y es utilizado para asignarle a cada una de las casillas de mi arreglo un valor entero desde 0 hasta 9. Nuestro segundo for nos ayuda a recorrer nuestro arreglo y mostrar su contenido en pantalla. El resultado se vería como se muestra en la figura 1.10.

1.8. Funciones

Hasta este momento la única función que se ha estado utilizando es el main(), dentro del cuál se ha estado ejecutando la mayor parte de nuestros ejemplos. Sin embargo, debemos recordar el uso de clrscr(), que es una función

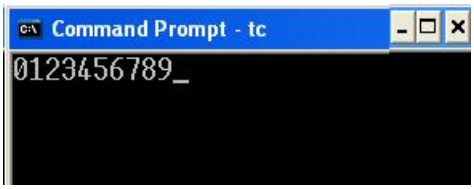

Figura 1.10: Contenido de un arreglo

que llamamos en todos nuestros programas y además nunca nos preocupamos por su implementación, lo único que hicimos fué llamarla y que realizara su trabajo el cuál es limpiar la pantalla. Con esta idea partimos en la creación de funciones dentro de nuestros programas en C. Ahora veamos como podemos crear funciones que sirvan para proporcionar mayor claridad y estructura a nuestros programas.

1.8.1. Utilizando Funciones

Veamos el siguiente código.

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

saludo(){
  printf("Este es un saludo desde un procedimiento en C");
  return 0;
}
void main()
{
  clrscr();
  saludo();
  getch();
}
```

Lo guardamos bajo el nombre funciones.c Aquí tenemos algo importante que observar, primero que nada creamos una función llamada saludo, en la cuál

en su interior tiene un printf, que nos estará ayudando a presentar un mensaje en pantalla. Otra cosa importante a mencionar es que se recomienda colocar todas nuestras funciones, que definamos en nuestro archivo fuente, antes de la función main(), o el compilador nos generará un error. Si compilamos y ejecutamos el código que mostramos anteriormente generariamos una salida como la que se muestra en la figura 1.11 Como podemos observar en este ejemplo, la función main(), se va volviendo más fácil de leer, y lo podemos leer de la siguiente forma: "main manda llamar primero a la función que limpia la pantalla: clrscr(), después manda llamar a la función que imprime un mensaje en pantalla: saludo(), y por último manda llamar a la función que nos permitirá visualizar nuestro resultado hasta que demos enter: getch()"

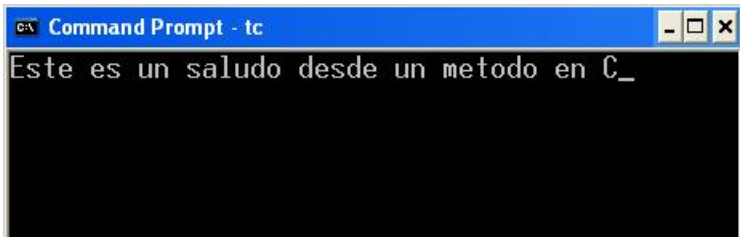

Figura 1.11: Llamada al método saludo

1.8.2. Pasando datos y Devolviendo Resultados

Las funciones en un lenguaje de programación pueden recibir datos, y devolver un resultado tras haber procesado alguna información. Utilicemos como siempre un ejemplo para dar seguimiento a este nuevo tema. Guardamos el código bajo el nombre param.c

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

suma(int primero, int segundo){

  return (primero + segundo);
```

```
}

resta(int primero,int segundo){

  return (primero - segundo);

}

void main()
{
  clrscr();
  int primero = 10;
  int segundo = 8;
  int resultado;
  //Mandamos llamar metodo suma

  resultado = suma(primero,segundo);
  printf("El metodo suma da como resultado: %d\n",resultado);
  resultado = resta(primero,segundo);
  printf("El metodo resta da como resultado: %d",resultado);
  getch();
}
```

El código que se mostró anteriormente, nos enseña como podemos crear funciones que hagan algo más que sólo presentar mensajes en pantalla, estas funciones reciben datos y los procesan en su interior. En nuestro código especificamos el nombre de la función, así como los parámetros que recibe, en el caso la función suma, recibe dos int, primero y segundo. Los cuáles en su interior procederá a sumarlos. Aquí es importante observar que no especificamos el tipo de retorno (el tipo de dato que devolverá nuestra función) y al no especificarlo en C por default se tiene que devuelve un entero.

Los resultados de ejecutar el código anterior son los valores 18 y 2 para suma y resta respectivamente. El resultado se muestra en la figura 1.12

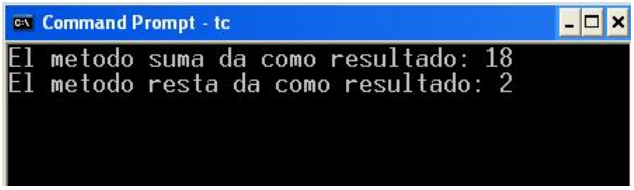

Figura 1.12: Resultados de los métodos suma y resta