Unidades I y II - Algoritmos e Introducción al Lenguaje C

Contenido

Herramientas a Utilizar

Para programar en lenguaje C necesitaremos un editor de textos y un compilador.

Editor de Textos

Cualquier editor de textos servirá. Sin embargo, se recomiendan editores de textos que estén pensados para programadores. Por ejemplo:

- Visual Studio Code
- Atom
- Sublime
- Brackets

Compilador

- GCC (Linux)
- MinGW (Windows): https://sourceforge.net/projects/mingw/files/

  Al instalar MinGW en Windows es necesario agregar la carpeta de instalación al path (variables de entorno) del sistema (normalmente: c:\mingw\bin)

Lenguaje de Programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:

- El desarrollo lógico del programa para resolver un problema en particular.
- Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).
- Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.
- Prueba y depuración del programa.
- Desarrollo de la documentación.

Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo HTML (lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación, sino un conjunto de instrucciones que permiten diseñar el contenido de los documentos).

Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa.

Para que la computadora entienda nuestras instrucciones debe usarse un lenguaje específico conocido como código de máquina, el cual la máquina comprende fácilmente, pero que lo hace excesivamente complicado para las personas. De hecho sólo consiste en cadenas extensas de números 0 y 1. Éste es el lenguaje que entienden los microprocesadores y el único que son capaces de ejecutar.

Para facilitar el trabajo, los primeros operadores de computadoras decidieron hacer un traductor para reemplazar los 0 y 1 por palabras o abstracción de palabras y letras provenientes del inglés; éste se conoce como lenguaje ensamblador (o assembler). Por ejemplo, para sumar se usa la instrucción ADD de la palabra inglesa (sumar). El lenguaje ensamblador sigue la misma estructura del lenguaje máquina, pero las letras y palabras son más fáciles de recordar y entender que los números. Ejemplo: la instrucción “ADD 3” (que sumaría 3 al valor de un registro específico (típicamente el *acumulador*) pordría traducirse como 0001 0011 (elección arbitraria donde los primeros 4 números binarios (1 en decimal) se referiría a la instrucción “sumar” y los últimos 4 números binarios (3 en decimal) se referiría al valor que se debe sumar.

La necesidad de recordar secuencias de programación para las acciones usuales llevó a denominarlas con nombres fáciles de memorizar y asociar: ADD (sumar), SUB (restar), MUL (multiplicar), CALL (ejecutar subrutina), etc. A esta secuencia de posiciones se le denominó instrucciones, y a este

conjunto de instrucciones se le llamó lenguaje ensamblador (o assembler). Posteriormente aparecieron diferentes lenguajes de programación, los cuales reciben su denominación porque tienen una estructura sintáctica similar a los lenguajes escritos por los humanos, denominados también lenguajes de alto nivel.

Programación

La programación es el proceso de diseñar, escribir, probar, depurar y mantener el código fuente de programas computacionales. El código fuente es escrito en un lenguaje de programación. El propósito de la programación es crear programas que exhiban un comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas distintas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal.

Para crear un programa, y que la computadora interprete y ejecute las instrucciones escritas en el, debe usarse un Lenguaje de programación.

Como vimos, la computadora sólo entiende lenguaje de máquina, pero para nuestra facilidad y comodidad existen los lenguajes de alto nivel. Escribir un programa en un lenguaje de alto nivel es mucho más sencillo que estar escribiendo ceros y unos.

Una vez que se termina de escribir un programa, sea en assembler o en un lenguaje de alto nivel, es necesario compilarlo. Compilar un programa es traducirlo a lenguaje de máquina; es decir, traducir un programa de un lenguaje de alto nivel a un lenguaje de máquina.

La programación se rige por reglas y un conjunto más o menos reducido de órdenes, expresiones, instrucciones y comandos que tienden a asemejarse a una lengua natural acotada (en inglés); y que además tienen la particularidad de una reducida ambigüedad. Cuanto menos ambiguo es un lenguaje de programación, se dice, es más potente. Bajo esta premisa, y en el extremo, el lenguaje más potente existente es el binario, con ambigüedad nula.

Programas y Algorítmos

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema.

Un programa normalmente implementa (traduce a un lenguaje de programación concreto) uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un diagrama de flujo, en forma de código como en pseudocódigo o un lenguaje de programación, en forma explicativa, etc.

Los programas suelen subdividirse en partes menores, llamadas módulos, de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa. Esta es una práctica muy utilizada y se conoce como "refino progresivo".

Un programa está formado por algoritmos y estructuras de datos.

Por ejemplo, se quiere preparar una tarta de queso. Una receta es una formalización de los pasos para preparar la tarta de queso: es un algoritmo.

Teniendo en cuenta que los ingredientes son queso de untar, huevos, yogur natural, harina y azúcar, la receta para su preparación es:

1. **Encender** el horno a 200º.
2. **Mezclar** todos los ingredientes y **batirlos** hasta que queden bien disueltos y sin ningún grumo.
3. **Untar** el molde con mantequilla y **verter** la mezcla. Meter la preparación en el horno y bajar la temperatura a 170º para su cocción.
4. En 20 o 25 minutos estará lista, **apagar el horno y dejarla dos minutos más**.
5. **Sacar** la tarta del horno, dejarla enfriar unos minutos y luego se lleva al frigorífico hasta el momento de servirse.

Observar que:

- En un número finito de pasos se resuelve la receta. En este caso, son 5 pasos a seguir para que la tarta quede lista.
- Las instrucciones de la receta son precisas. En cada paso, es claro qué acción realizar.

Las recetas culinarias son un claro ejemplo de algoritmo. Existe un problema, elementos para solucionarlos, un procedimiento a seguir y un resultado que es el plato listo. Todas estas características las debe cumplir un algoritmo.

En la vida cotidiana se emplean algoritmos en multitud de ocasiones para resolver diversos problemas. Más ejemplos:

Ejemplo de un algoritmo para cambiar la llanta a un automóvil:

1. Inicio.
2. Traer gato.
3. Levantar el coche con el gato.
4. Aflojar tornillos de las llantas.
5. Sacar los tornillos de las llantas.
6. Quitar la llanta.
7. Poner la llanta de repuesto.
8. Poner los tornillos.
9. Apretar los tornillos.
10. Bajar el gato.
11. Fin

Ejemplo de un algoritmo para preparar una taza de café, tomando en cuenta que se tiene agua caliente, una taza, cuchara, café, crema y azúcar.

1. Inicio.
2. Verter agua caliente en la taza.
3. Tomar con la cuchara el café.
4. Poner el café en la taza.
5. Si quiere azúcar,
   5.1. Tomar el azúcar con la cuchara
   5.2. Poner el azúcar en la taza.
6. Si quiere Crema
   6.1. Tomar la crema con la cuchara.
   6.2. Poner crema en la taza
7. Revolver.
8. Fin

Estos ejemplos muestran la formalización de los pasos a seguir para solucionar problemas de la vida cotidiana. Se plantea una secuencia de pasos precisos que, ejecutándolas en orden, se llega a la solución del problema.

Características fundamentales

- Un algoritmo debe ser *preciso* e indicar el orden de realización de cada paso.
- Un algoritmo debe estar *definido*. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez (ante la misma entrada de datos).
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento, debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida

- Datos de Entrada: un algoritmo tiene cero o más entradas, es decir cantidades que le son dadas antes de que el algoritmo comience, o dinámicamente mientras el algoritmo corre.
- Procesamiento de Datos: aquí incluye operaciones aritmético-lógicas, selectivas y repetitivas; cuyo objetivo es obtener la solución del problema.
- Salida de Resultados: permite comunicar al exterior el resultado. Puede tener una o más salidas, es decir cantidades que tienen una relación única respecto a las entrantes.

Compilación

El programa escrito en un lenguaje de programación (fácilmente comprensible por el programador) es llamado programa fuente y no se puede ejecutar directamente en una computadora. La opción más común es compilar el programa obteniendo un módulo objeto, aunque también puede ejecutarse en forma más directa a través de un intérprete informático.

El código fuente del programa se debe someter a un proceso de traducción para convertirlo en lenguaje máquina (que es único ejecutable por el procesador). A este proceso se le llama compilación.

Normalmente la creación de un programa ejecutable lleva dos pasos. El primer paso se llama compilación (propiamente dicho) y traduce el código fuente escrito en un lenguaje de programación almacenado en un archivo a código en bajo nivel (normalmente en código objeto, no directamente a lenguaje máquina). El segundo paso se llama enlazado en el cual se enlaza el código de bajo nivel generado de todos los ficheros y subprogramas que se han mandado compilar y se añade el código de las funciones que hay en las bibliotecas del compilador para que el ejecutable pueda comunicarse directamente con el sistema operativo, traduciendo así finalmente el código objeto a código máquina, y generando un módulo ejecutable.

Estos dos pasos se pueden hacer por separado, almacenando el resultado de la fase de compilación en archivos objetos; para enlazarlos en fases posteriores, o crear directamente el ejecutable; con lo que la fase de compilación se almacena sólo temporalmente. Un programa podría tener partes escritas en varios lenguajes (por ejemplo C, C++ y ensamblador), que se podrían compilar de forma independiente y luego enlazar juntas para formar un único módulo ejecutable.

Algunos Conceptos fundamentales

Procesador

Entidad capaz de entender una secuencia finita de acciones y ejecutarlas en la forma en que se especifican.

Código Fuente

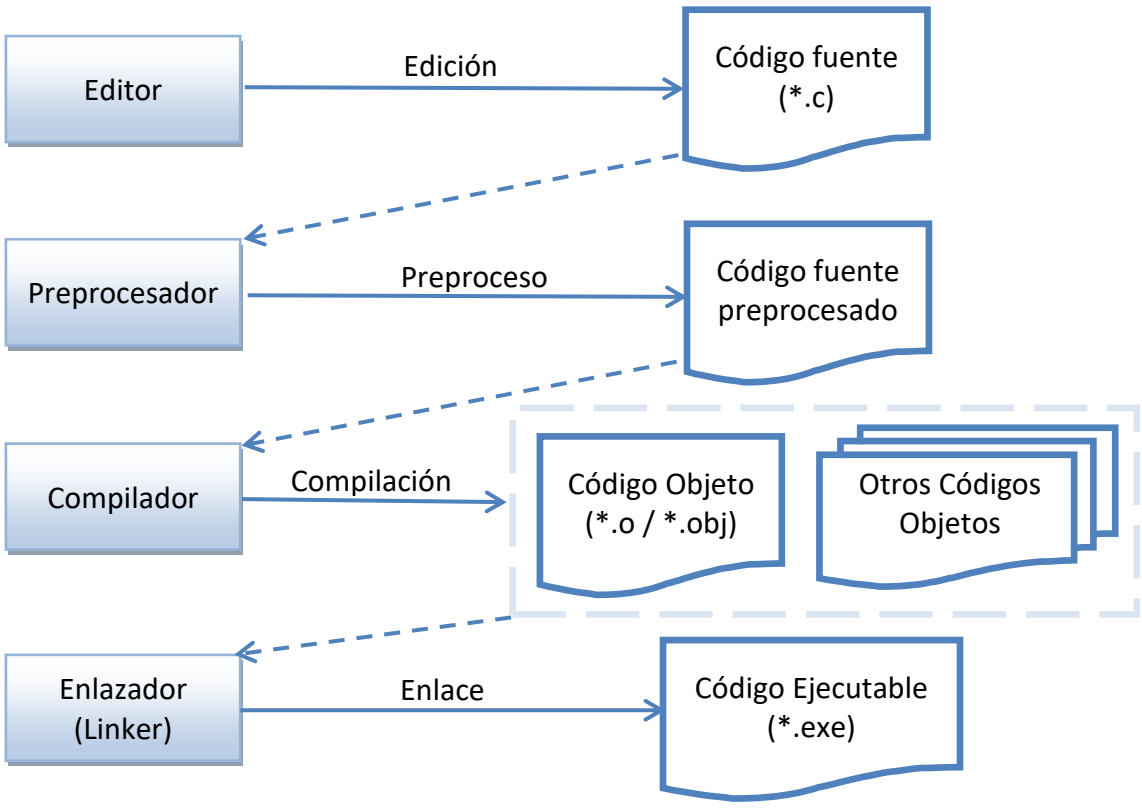
El código fuente de un programa es el escrito por un programador en algún lenguaje de programación. Como vimos, en este primer estado no es directamente ejecutable por la computadora, sino que debe ser traducido a lenguaje de máquina. Para esta traducción se usan los llamados compiladores.

Código Objeto

En programación, se llama código objeto al código que resulta de la compilación del código fuente. Consiste en lenguaje de máquina y se distribuye en varios archivos que corresponden a cada código fuente compilado. Para obtener un programa ejecutable se han de enlazar todos los archivos de código objeto con un programa llamado enlazador (linker).

Código Ejecutable

El código que es capaz de ejecutar un procesador.



Llevando estos conceptos a la práctica

Escribamos nuestro primer programa en lenguaje C, en un archivo llamado hola.c.

```
#include <stdio.h>

int main() {
  printf("Hola <su-nombre>!");

  return 0;
}
```

Para generar el código objeto debemos tipear en la consola:

gcc –c hola.c

Esto genera el archivo hola.o. Este archivo no será ya más comprensible por nosotros. Para generar el código ejecutable a partir del código objeto:

gcc -o hola hola.o

Esto genera el archivo ejecutable hola. Para ejecutarlo en la computadora bastará con tipear:

./hola

Es posible realizar todos estos pasos automáticamente, es decir, generar un archivo ejecutable a partir de un archivo fuente (tener en cuenta que el proceso siempre generará todos los pasos anteriores, pero será transparente a nuestros ojos). Para esto:

gcc –o hola hola.c

Preprocesador

El lenguaje C realiza un paso más a los recién vistos: antes de la compilación se realiza un pre-procesamiento del archivo fuente. Esta tarea es realizada por el preprocesador y se utiliza para resolver características que permite realizar el lenguaje (inclusión de archivos, definición de constantes, compilación condicional, etc). Para ver el código fuente luego de ser preprocesado debemos tipear en la consola:

gcc -E hola.c

IDE

Un IDE es un entorno de desarrollo integrado (Integrated Development Environment) que permite desde un programa corriendo en el entorno gráfico, crear, editar, compilar y ejecutar programas desarrollados por nosotros.

Algunos IDEs son: CodeBlocks, Dev C++, Visual Studio y Visual Studio Community, C++ Builder o C++ Builder Community Edition, Eclipse, Netbeans, etc.

Introducción al Lenguaje C

Para comenzar a escribir programas en lenguaje C tenga en cuenta:

- La indentación es muy importante para poder comprender mejor los programas escritos en C.
- C es sensible a mayúsculas y minúsculas, por lo tanto, a != A. Todas las palabras claves vistas son con minúsculas, while no es lo mismo que While.
- En C, todas las instrucciones finalizan con ";" (punto y coma). Notar que existen estructuras que no finalizan con ";". Un ejemplo, es la condición de un while no finaliza con ";" ya que no es la finalización de una instrucción.

Todo programa escrito en C está en el contexto de una función principal, llamada main (principal en inglés). Entonces, todos los programas estarán en el contexto de esta función.

Variables: permiten almacenar datos en la computadora, para ser utilizados en un programa.

Resumen de instrucciones que veremos		
Tipo	**Lenguaje C**	**Ejemplo**

Comienzo y fin de un programa	int main()
	{
	return 0;
	}
		int main()
		{
		printf("Hola\n");
		return 0;
		}

Asignación de variable	variable = valor;	x = 4;
Ingreso de datos teclado	scanf("formato", &variables);	scanf("%d", &x); scanf("%d %d", &x, &y);
Impresión por pantalla	printf("formato"); printf("formato",variables);	printf("Hola"); printf("Edad: %d Peso: %d", edad, peso);

Condional	if (condición)
	{
	<instrucciones>
	}
		if(numero> 0)
		{
		printf("Numero positivo");
		}

Condional	if (condición)
	{
	<bloque V>
	} else {
	<bloque F>
	}
		if (x == 0)
		{
		printf ("Error división por 0")
		} else {
		resultado = 100 / x;
		}

Ciclo PARA	for (var=valor; condición; incr)
	{
	<bloque de instrucciones>
	}
		for(i=0; i<10; i++)
		{
		printf("i: %d\n",i);
		}

Ciclo condicional MIENTRAS	while (condición)
	{
	<bloque de instrucciones>
	}
		while (suma < 10)
		{
		suma += x;
		}

Ciclo hacer/mientras	do { <bloque de instrucciones> } while (condición);	do { scanf("%d", &opción); } while (opción != 0);

Variables, Ingreso de Datos por teclado, Impresión de Datos por pantalla

Declaración de variables: las variables deben ser declaradas al comienzo del programa, dentro de la función main(), precedidas por el tipo (int, float, char).

Tipo de dato Int

Ejemplo: escriba un programa que lea un número entero ingresado por teclado, lo multiplique por dos, y luego lo muestre por pantalla:

```
#include <stdio.h>

int main() {
  int x, mult;

  scanf("%d", &x);
  mult = x * 2;
  printf("%d * 2 = %d\n", x, mult);

  return 0;
}
```

- para números enteros usar variable int y leer/imprimir utilizando %d.

Tipo de dato Float

Ejemplo: escriba un programa que lea una suma y una cantidad en números reales ingresado por teclado, y luego calcule e imprima por pantalla el promedio:

```
#include <stdio.h>

int main() {
  float promedio, suma, cantidad;

  scanf("%f", &suma);
  scanf("%f", &cantidad);

  promedio = suma / cantidad;
  printf("Suma: %f   Cantidad %f  Promedio: %f\n", suma, cantidad,
promedio);

  return 0;
}
```

- para números reales usar float y leer/imprimir utilizando %f.

Tipo de dato Char

Ejemplo: escriba un programa que pida al operador ingresar una letra, y luego imprima dicha letra:

```
#include <stdio.h>

int main() {
  char letra;

  scanf(" %c", &letra); // notar el espacio entre la comilla y el %
  printf("Ud. Ingreso la letra: %c\n", letra);

  return 0;
}
```

- para números reales usar float y leer/imprimir utilizando %f.

Expresiones aritméticas en lenguaje C

Operador	Nombre	Ejemplo
+	Suma	a = b + 4
-	Resta	a = 3 – 4
*	Multiplicación	a = b * c
/	División	a = 4 / b
%	Módulo (operación resto)	a = numero % 2
++	Incremento en 1	a++
--	Decremento en 1	a--
+=	Incremento en n	a += 5
-=	Decremento en n	a -= 5

Hacer los ejercicios del práctico secciones:

- I- Estructuras secuenciales.
- Para algunos ejercicios es necesario entender la operación cast (a continuación), que permite cambiar el tipo de una variable para realizar operaciones.

Operación cast

El lenguaje C realiza las operaciones de acuerdo al tipo de datos de las variables involucradas. Asi, si las variables son enteras, entonces C realizará las operaciones utilizando enteros. Entendiendo esto, ¿qué valor dará a la variable c el siguiente fragmento de código?

```
int a = 5;
int b = 2;
float c = a / b;
```

Debido a que a y b son enteros, la operación es realizado utilizando aritmética de entero y por lo tanto a / b dará por resultado 2. Este valor será asignado luego a la variable c. Es decir, c = 2.

Pero si en cambio alguna de las variables es real, C realiza la operación como aritmética real. Entendiendo esto, ¿qué valor dará a la variable c el siguiente fragmento de código?

```
int a = 5;
float b = 2;
float c = a / b;
```

Debido a que b es real, la operación es realizado utilizando aritmética real y por lo tanto a / b dará por resultado 2,5 que es el valor esperado. Este valor será asignado luego a la variable c. Es decir, c = 2.5.

Resumiendo, C trata de resolver las operaciones de la forma más sencilla (enteros) a no ser que haya algún operando de algún tipo más complejo. Si lo hubiere, C realizará la operación en la aritmética del tipo más complejo.

Muchas veces tenemos variables enteras pero necesitamos asignar el resultado a una variable de tipo real (float o double). Para esto se utiliza la operación cast. Para que el primer fragmento de código asigne a c el valor correcto 2,5 pero manteniendo las variables enteras, se utiliza la operación cast:

```
int a = 5;
int b = 2;
float c = (float)a / b;
```

La operación cast (en este caso es poner la palabra float a la izquierda de algún operando), lo que hace es convertir la variable a al tipo float antes de realizar la operación. El lenguaje C verificará que uno de los operando es de tipo entero pero el otro es de tipo float, por lo que la operación será realizada en tipo float.

Otra posibilidad es usar la operación cast para pasar un resultado de punto flotante a entero. Esto es lo que hace el siguiente framento de código:

```
float a = 5.0;
float b = 2.0;
int c = (int)(a / b);
```

En este caso la operación se realizará con aritmética real, pero antes de asignar el valor a la variable c, el mismo será convertido a entero. De esta manera el valor 2,5 será convertido a su equivalente entero 2 y este valor será asignado a la variable c.

Condicionales

Determinan la ejecución de una u otra parte del programa dependiendo de que se cumpla o no una condición.

Condicional simple

Si se cumple una determinada condición se ejecutará un conjunto de instrucciones (que en caso contrario no se ejecutarían).

```
if (condicion) {
  instrucciones
}
```

Donde:
- *condición* es cualquier expresión que devuelva un valor lógico (true/false).
- *instrucciones* es un bloque de una o más instrucciones que se ejecutarán solo en el caso de que se cumpla la condición.

Ejemplo: escriba un programa que indique si un número ingresado por teclado es positivo.

```
#include <stdio.h>

int main() {
  int a;

  printf("Ingrese un numero: ");
  scanf("%d", &a);

  if (a > 0) {
    printf("El numero ingresado es positivo.\n");
  }

  return 0;
}
```

Condicional si/sino

Si se cumple la condición se ejecutará el primer bloque de instrucciones, si no se cumple se ejecutará el segundo bloque.

```
if (condicion) {
  instrucciones
} else {
  instrucciones
}
```

Ejemplo: escriba un programa que indique si un número ingresado por teclado es positivo no.

```c
#include <stdio.h>

int main() {
  int a;

  printf("Ingrese un numero: ");
  scanf("%d", &a);

  if (a > 0) {
    printf("El numero ingresado es positivo.\n");
  } else {
    printf("El numero ingresado no es positivo.\n");
  }

  return 0;
}
```

Modifique el ejercicio para indicar si es positivo, cero o negativo:

Ejemplo: escriba un programa que, dada la edad de una persona, determine si es mayor o menor de edad.

```c
#include <stdio.h>

int main() {
  int edad;

  printf("Ingrese su edad: ");
  scanf("%d", &edad);

  if (edad < 18) {
    printf("Ud. es menor de edad.\n");
  } else {
    printf("Ud. Es mayor de edad.\n");
  }

  return 0;
}
```

Expresiones relacionales que se pueden utilizad para formar condiciones:

Operador	Nombre	Ejemplo
>	mayor que	if(a > b) {
>=	mayor o igual que	if(a >= 10) {
<	menor que	if(a + 3< b) {
<=	menor o igual que	if(a <= b + 10) {
==	igual	if(a % 2== 0) {
!=	distinto	if(a != 0) {

Las condiciones se pueden combinar usando los operadores lógicos indicados a continuación.
Expresiones lógicas:

Operador	Nombre	Ejemplo
&&	y, conjunción (and)	if(a < 10 && a > 0) {
||	o, disyunción (or)	if(a < 10 || a > 20) {
!	no, negación (not)	a = ! b

Conjunción Lógica o Producto Lógico (y, and):

El operador correspondiente se representa mediante los símbolos **'^'** (propio de la Lógica Proposicional) 'AND' y '.' En el Lenguaje de Programación C se utiliza el símbolo **'&&'**.

El efecto de este Operador es la evaluación simultánea del Estado de Verdad de las variables lógicas involucradas.

Así tendremos, por ejemplo, que la expresión: **P && Q**, será **Verdadera** únicamente si **P** y **Q** lo son. Cualquier otro arreglo de Estados de Verdad para ambas variables, dará como resultado el Valor Falso, puesto que basta con que una de la dos variables tenga valor Falso, para que ambas no sean simultáneamente Verdaderas.

Variables Lógicas		Resultado
P	Q	P && Q
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso
Falso	Verdadero	Falso
Falso	Falso	Falso

Ejemplo: escriba un programa que, dada la hora del día, indique si es horario matutino (de 5 a 11 hs).

```c
#include <stdio.h>

int main() {
  int hora;

  printf("Ingrese la hora del dia: ");
  scanf("%d", &hora);

  if (hora >= 5 && hora <= 11) {
    printf("Es horario matutino.\n");
  } else {
    printf("No es horario matutino.\n");
  }

  return 0;
}
```

Disyunción Lógica Inclusiva o Suma Lógica (o, or):

El Operador correspondiente, se representa mediante los símbolos **'v'** (propio de la Lógica Proposicional), **'OR'** y **'+'** En el Lenguaje de Programación C se utiliza el símbolo '||'.

El efecto de este operador, es la evaluación no simultánea del Estado de Verdad de las variables lógicas involucradas. Esto implica que al tener Estado Verdadero por lo menos una de las variables afectadas, la operación dará un resultado verdadero.

Así tendremos que la expresión **A || B** será Falsa únicamente cuando el Estado de Verdad de ambas variables sea Falso. En cualquier otro caso, la operación será Verdadera.

Variables Lógicas		Resultado
P	Q	P || Q
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Ejemplo: escriba un programa que, dada la hora del día, indique si es horario nocturno (de 21 a 4 hs).

```c
#include <stdio.h>

int main() {
  int hora;

  printf("Ingrese la hora del dia: ");
  scanf("%d", &hora);

  if (hora >= 21 || hora <= 4) {
    printf("Es horario nocturno.\n");
  } else {
    printf("No es horario nocturno.\n");
  }

  return 0;
}
```

Negación o Complemento Lógico (no, not):

Este Operador representado por un guión sobre la Variable a Complementar ejemplo **Ā** y también la palabra **'NOT'** al aplicarse a un predicado lógico (simple o compuesto) , devuelve el valor opuesto; es decir : si el predicado en cuestión es Falso (valor lógico F), el resultado será Verdadero (valor lógico T) y recíprocamente. En el Lenguaje de Programación C se utiliza el símbolo '**!**'.

Variable Lógica	Resultado
P	! P
Verdadero	Falso
Falso	Verdadero

Hacer los ejercicios del práctico secciones:

- II- Condicionales
- III- Condicionales anidados y operadores lógicos

Condicionales: uso de switch/case

La sentencia switch es un condicional que evalúa una expresión numérica entera y, de acuerdo a su valor, selecciona un bloque de instrucciones y los ejecuta.

```
switch(expresión) {
    case valor:
        instrucciones
        break;

    case valor-2:
        instrucciones
        break;

    ...

    default:
        instrucciones;
        break;
}
```

De acuerdo al valor de x se elije el conjunto de instrucciones indicado por la sentencia case. Las sentencias involucradas son:

- switch(*expresión*) evalúa la expresión y selecciona el caso correspondiente de acuerdo al valor obtenido
- case *valor:* indica el comienzo del bloque de instrucciones que se deben ejecutar si *expresión* es igual a *valor*
- break finaliza el bloque de instrucciones / termina la ejecución del switch
- default indica el comienzo del bloque de instrucciones que se deben ejecutar si el valor de la *expresión* no se corresponde con ningún caso especificado

Ejemplo: escriba un programa que escriba en letras el número ingresado por teclado. El número debe ser 0, 1 o 2.

```
#include <stdio.h>

int main() {
    int numero;

    printf("Ingrese un numero: ");
    scanf("%d", &numero);

    switch(numero) {
        case 0:
            printf("cero\n");
            break;

        case 1:
            printf("uno\n");
            break;

        case 2:
            printf("dos\n");
            break;

        default:
            printf("Numero invalido\n");
            break;
    }

    return 0;
}
```

Hacer los ejercicios del práctico 1 secciones:

- IV- Sentencias de decisión y switch

Estructuras repetitivas (ciclos)

En ocasiones necesitaremos que un bloque de instrucciones se ejecute varias veces seguidas; en estos casos utilizaremos estructuras repetitivas o bucles:

- estructura for (para)
- estructura while (mientras)
- estructura do … while (hacer … mientras)

Supongamos un ejercicio donde tengamos que escribir los números del 1 al 5. La solución es bastante sencilla teniendo en cuenta las instrucciones que ya hemos aprendido:

```
#include <stdio.h>

int main() {
    printf("1\n");
    printf("2\n");
    printf("3\n");
    printf("4\n");
    printf("5\n");

    return 0;
}
```

Pero qué sucedería si tuviéramos que imprimir los los números del 1 al 100, o al 10 mil, o al 100 mil. Claramente la solución propuesta no escala. Es por esto que en los lenguajes de programación existen los ciclos.

Ciclo for (para)

La estructura for ejecuta un bucle un número determinado de veces controlando automáticamente el número de iteraciones. La utilizaremos siempre que sepamos previamente el número de veces que se ejecutará el bucle. Su formato genérico es el siguiente:

```
for (i=0; i <= 10; i++) {
   instrucciones
}
```

Donde:

- ***i***: es una variable interna que se utiliza normalmente como contador del numero de ejecuciones del bucle en cada momento.
- ***0***: es el valor inicial que tomará la *i.*
- ***i < 10***: es una condición que será comprobada antes de realizar cada ejecución del bucle. Si se cumple, se ejecutará el bloque de instrucciones; en caso contrario pasará el control a la línea siguiente al final de la estructura.
- **i++**: es una expresión que modificará el valor de la variable de control. Normalmente se trata de un valor positivo o negativo.

Ejemplo: escriba un programa que muestre los números entre 0 y 10.

```c
#include <stdio.h>

int main() {
  int i;

  printf("Los números entre 0 y 10 son:");

  for (i=0; i <= 10; i++) {
     printf("\t%d", i);
  }
  printf("\n");

  return 0;
}
```

Ejemplo: escriba un programa que muestre los números pares entre 2 y 100.

```c
#include <stdio.h>

int main() {
  int i;

  printf("Los números pares entre 2 y 100 son:");

  for (i=2; i <= 100; i += 2) {
     printf("\t%d", i);
  }
  printf("\n");

  return 0;
}
```

Ejemplo: escriba un programa que muestre los números del 10 al uno

```c
int main() {
  int i;

  printf("Los números entre el 10 y el 1 son:");

  for (i=10; i > 0; i--) {
     printf("\t%d", i);
  }
  printf("\n");

  return 0;
}
```

Hacer los ejercicios del práctico secciones:

- V- Ciclo de repetición exacto (for)
- VI- Contadores, acumuladores, mínimos y máximos

Contadores, acumuladores, mínimos y máximos

Resuelvas los primeros 4 (cuatro) ejercicios del práctico, sección VI.

Hacer los ejercicios del práctico secciones:

- VI- Contadores, acumuladores, mínimos y máximos

Ciclo while (mientras)

La estructura while ejecuta un bloque de instrucciones y repite dicha ejecución mientras que se cumpla una condición.

```
while (condición) {
   instrucciones
}
```

Donde:

- ***condición*** es la condición cuyo valor deberá ser true para que se produzca la entrada en el bucle y que será comprobado antes de cada nueva ejecución del bloque de instrucciones.
- ***instrucciones*** es el bloque de instrucciones que se ejecutará mientras que la condición sea verdadera.

Funcionamiento:
1. Al encontrar la estructura while lo primero que hace (antes de entrar por primera vez en el bucle) es evaluar la condición: si es verdadera entra en el bucle y ejecuta el bloque de instrucciones, pero si es falsa ni siquiera llegará a entrar en el bucle.

2. Una vez ejecutadas las instrucciones del bucle se evalúa de nuevo la condición para determinar si se vuelve a ejecutar el bloque o no (si es verdadera se ejecuta, si es falsa deja de ejecutarse). Este punto se repite hasta que la condición deja de ser verdadera.

3. Cuando al evaluar la condición el resultado es false, el flujo del programa va a la línea siguiente al final del bucle.

Observaciones: Debemos asegurarnos de que en algún momento se produzca la salida del bucle ya que de lo contrario estaríamos ante un bucle sin fin (bucle infinito). Por ejemplo, si en lugar de la condición while (nota < 0 || nota > 10) hubiésemos escrito: while (nota < 0 && nota > 10) el bucle hubiese estado iterando constantemente y no finalizaría nunca.

Ejemplo: escriba un programa que sume números ingresados por teclado hasta que el número ingresado sea cero.

```
#include <stdio.h>

int main() {
  int total, dato;

  total = 0;
  scanf("%d", dato);
  while (dato > 0) {
    total = total + dato;
    scanf("%d", dato);
  }

  printf("La suma de los nros ingresados es: %d", total);

  return 0;
}
```

Ciclo do/while (hacer/mientras)

La estructura do..while es similar a la anterior pero en este caso la comprobación se produce después de ejecutar el bloque de instrucciones.

```
do {
  instrucciones;
} while (condición);
```

Ejemplo: escriba un programa que sume números ingresados por teclado y que pida al usuario la opción de continuar o finalizar luego de cada numero ingresado. Con lo que hemos visto hasta ahora una solución posible es utilizar el ciclo while:

```
#include <stdio.h>

int main() {
  int suma, nro, continuar;

  suma = 0;
  continuar = 1; /* para ingresar al ciclo la 1era vez */
  while (continuar) {
    printf("Ingrese un nro: ");
    scanf("%d", &nro);
    suma = suma + nro;

    printf("Continuar? 1=SI, otro=NO: ");
    scanf("%d", &nro);
  } while (continuar == 1);

  printf("La suma es %d\n", suma);

  return 0;
}
```

También es posible utilizar el ciclo do..while:

```
#include <stdio.h>

int main() {
  int suma, nro, continuar;

  suma = 0;
  do {
    printf("Ingrese un nro: ");
    scanf("%d", &nro);
    suma = suma + nro;

    printf("Continuar? 1=SI, otro=NO: ");
    scanf("%d", &continuar);
  } while (continuar == 1);

  printf("La suma es %d\n", suma);

  return 0;
}
```

La única diferencia entre el ciclo *while* y el ciclo *do..while* está en la primera vez que se ejecuta el bucle:

- la estructura *while* comprueba la condición antes de entrar por primera vez en el bucle y si la condición no se cumple, no entrará.
- la estructura *do..while* ejecuta el bucle la primera vez sin comprobar la condición. Para las demás iteraciones el funcionamiento es idéntico en ambas estructuras (únicamente se producen variaciones en el caso de utilizar la cláusula *continue*).

¡Un ejercicio, múltiples soluciones!

Intente hallar diferentes soluciones para el siguiente ejercicio: acumular números hasta que el usuario ingrese el número 999.

Solución 1: solicitando dos veces el ingreso del número, antes del while y dentro del while

```c
#include <stdio.h>

int main() {
    int numero, suma;

    suma = 0;

    printf("Ingrese numero: ");
    scanf("%d", &numero);

    while (numero != 999) {
        suma += numero;

        printf("Ingrese numero: ");
        scanf("%d", &numero);
    }

    printf("La suma de todos sus numeros es: %d\n\n", suma);

    return 0;
}
```

Solución 2: usando ciclo while e inicializando variables, pero teniendo cuidado de no sumar el valor 999 que indica salir.

```c
#include <stdio.h>

int main() {
    int numero, suma;

    suma = 0;
    numero = 0; /* inicializar con un valor para ingresar al ciclo */

    while (numero != 999) {
        printf("Ingrese numero: ");
        scanf("%d", &numero);

        if(numero != 999) {
            suma += numero;
        }
    }

    printf("La suma de todos sus numeros es: %d\n\n", suma);
```

```
    return 0;
}
```

Solución 3: inicializando con un valor neutro para evitar el doble pedido de ingreso de datos de la solución 1.

```
#include <stdio.h>

int main() {
    int numero, suma;

    suma = 0;
    numero = 0; /* valor neutro para sumar y permite ingresar al ciclo */

    while (numero != 999) {
        suma += numero;

        printf("Ingrese numero: ");
        scanf("%d", &numero);
    }

    printf("La suma de todos sus numeros es: %d\n\n", suma);

    return 0;
}
```

Solución 4: utilizando el ciclo do..while.

```
#include <stdio.h>

int main()
{
    int numero, suma;

    suma = 0;
    numero = 0;

    do {
        suma += numero;

        printf("Ingrese numero: ");
        scanf("%d", &numero);
    } while (numero != 999);

    printf("La suma de todos sus numeros es: %d\n\n", suma);

    return 0;
}
```

Note que las últimas dos soluciones son las más claras ya que no es necesario tratar casos especiales como en las primeras dos soluciones. Sin embargo, si modificamos el ejercicio de manera que el número a ingresar para finalizar sea 0 (cero), entonces la única solución válida es la última (ciclo do..while). Pruébelo modificando las últimas dos soluciones.

Hacer los ejercicios del práctico secciones:

- VII- Ciclo de repetición no exacto o condicional (while)

Variables y Constantes

Un identificador es una Variable, cuando su valor puede modificarse y además posee un Nombre que lo identifica y un Tipo que describe su uso.

Un identificador es una Constante, cuando su valor no puede modificarse y además posee un Nombre que lo identifica y un Tipo que describe su uso.

Cuando definimos una Variable, estamos creando un objeto para el Procesador. La diferencia respecto de la definición de una constante, es que en el momento de su creación, el valor del objeto es desconocido, mientras que para una constante no solamente es conocido sino que permanece inalterado durante la ejecución del procedimiento resolvente.

Las variables son palabras en el código, que al ejecutar el programa toman valores concretos.

Nombres de las variables: sólo están permitidas letras de la 'a' a la 'z' (la ñ no se incluye), números y el símbolo '_'. Las variables no pueden comenzar con un número.

Ejemplos de nombres válidos:

camiones
numero
a1
j10hola29
num_alumnos

Ejemplos de nombres no válidos:

1abc
nombre?
num/alumnos
num-alumnos
#variable

Tampoco es posible utilizar como nombres de variables las palabras reservadas que usa el compilador. Por ejemplo: for, main, do, while, etc.

El lenguaje C distingue mayúsculas y minúsculas. Por lo tanto, las siguientes son variables diferentes:

Nombre
nombre
NOMBRE

Por convención, las variables se escriben siempre en minúsculas (pueden tener alguna mayúscula si hace falata), y las constantes se escriben siempre enmayúsculas (y no pueden tener minúsculas).

Alcance de una variable

Tenemos dos posibilidades respecto del alcance de una variable:

- declarar una variable como global

- declarar una variable como local

Es global aquella variable que se declara fuera de las funciones (main es una función) y local la que se declara dentro de alguna:

Variable Global	**Variable Local**

#include <stdio.h>
int x;
int main()
{
…
}
	#include <stdio.h>
	int main()
	{
	int x;
	…
	}

Las variables globales se pueden usar en cualquier función y las locales sólo pueden usarse en la función en el que se declaran. Es buena costumbre usar variables locales en vez de globales.

Podemos declarar más de una variable en una sola línea:

```
int x, y;
```

Nota: el uso de variables globales quedará más claro al ver Funciones. Por ahora utilizaremos variables locales.

Constantes

Las constantes se declaran igual que una variable normal, pero añadiendo la palabra const delante. Esta forma de definir constantes está disponible desde el standard C99 en adelante. Por ejemplo, para declarar una constante con valor 14 y de tipo entero:

```
const int NUMERO = 14;
```

Esta constante no puede ser modificada a lo largo del programa. Por eso deben ser definidas al mismo tiempo que declaradas.

Otra forma utilizada es definir constantes al comienzo del programa utilizando la directiva #define. Esta forma de definir constantes hace uso del preprocesador de C, que está disponible desde las primeras versiones del lenguaje y aún continúa vigente. Por ejemplo:

```
#define PI 3.141592
```

Tipos de Datos

El lenguaje de programación C proporciona *un conjunto fijo de Tipos de Datos*, llamados *Datos Primitivos o Básicos*. Cada *Tipo de Dato Primitivo o Básico*, define el conjunto de valores que puede asumir una variable. De aquí se desprende *que el Tipo de Dato, fija el conjunto de Operaciones aplicables a todos los Datos de su Clase*.

Los Tipos de Datos Básicos que manejan la mayoría de los Procesadores son:

Tipo NUMERICO
Tipo LOGICO
Tipo CARACTER

Estos Tipos se aplican tanto a las Variables como a las Constantes, pero debe observarse que *una variable de un determinado Tipo, sólo puede tomar como valor una constante del mismo Tipo*.

Tipo Numérico

Básicamente se distinguen dos Subtipos:

Tipo Numérico Entero.
Tipo Numérico Real.

Los Datos Numéricos Enteros, deberán siempre estar comprendidos entre los valores mínimo y máximo que cada computador establezca.

Los Datos Numéricos Reales, tendrán en general dos formas de representación: apelando a un punto decimal que separe la mantisa de la parte entera, o bien según la notación científica, utilizable para expresar números muy grandes o muy chicos. Esto entraña a dos maneras de considerar a las expresiones decimales de los números Reales: Notación Científica o Exponencial, y Coma o Punto Flotante, en la cual se fija la cantidad de dígitos significativos (sin considerar ceros a izquierda). En realidad, la notación de punto flotante, es un tipo de notación científica en un formato particular, denominado normalizado en base 2 (dos).

No se hará incapié aquí sobre estos mecanismos de representación, ya que se supone que el lector está familiarizado con ellos. En cuanto a la representación interna de números reales dentro de un ordenador, dada su complejidad respecto de la representación de números enteros, se hará alguna referencia más adelante.

Tipo Lógico

A este Tipo de Dato, se lo suele llamar también Tipo Booleano, en honor al matemático inglés George Boole, quien creó un Algebra para el tratamiento de los mismos. Por tal razón muchas veces se presentará a tales variables bajo la denominación de Variables Booleanas.

La principal característica que posee este Tipo, es que su valor no es *cuantitativo*, sino *cualitativo*. Además mientras que una variable del tipo numérico puede asumir un gran conjunto de valores (idealmente infinitos) como contenido, una variable Booleana, es binaria y por consiguiente podrá asumir dos valores solamente : Verdadero (True) o Falso (False). De este dominio impuesto para el Tipo en estudio, se explica su caracter cualitativo. Estas variables permiten describir Estados del Ambiente mediante un enunciado implícito en las mismas que puede resultar verdadero o falso, dependiendo del instante de la observación.

Las operaciones posibles de definir entre ellas así como también sus aplicaciones serán luego estudiadas detalladamente.

Tipo Caracter

Involucran a un *conjunto ordenado y finito de símbolos que el procesador puede reconocer*. Si bien no existe un conjunto estandar, podemos decir que dicho conjunto está básicamente integrado por:

1) Letras mayúsculas (desde la A hasta la Z), sin incluir la CH y la LL (eventualmente puede no ser incluida la Ñ).

2) Letras minúsculas (desde la a hasta la z), con las mismas restricciones que para las mayúsculas.

3) Dígitos (del 0 al 9).

4) Caracteres especiales. Están incluidos aquí símbolos tales como: *,+,-,:,;,,,,",/,<,>,=,|,\,~,@,#,$,.,%,^,&,(,),{,},[,],`,!,?,' y otros. El espacio en blanco, también puede ser considerado como un caracter.

A continuación mostramos una tabla con los tipos de datos básicos y sus rangos de valores en lenguaje C:

	Tipo	Nombre	Ancho en bits / bytes	Rango de Valores
*	Númerico	char	8 bits / 1 byte	-128 a 127
	Numérico Entero	short	16 bits / 2 bytes	-32.768 a 32.767
*	Numérico Entero	int	32 bits / 4 bytes	-2.147.483.648 a 2.147.483.647
	Numérico Entero	long	32 bits / 4 bytes	-2.147.483.648 a 2.147.483.647
*	Numérico Real	float	32 bits / 4 bytes	
	Numérico Real	double	64 bits / 8 bytes	
*		void	0	sin valores

(*) Estos son los principales que utilizaremos en el curso.

Unidades de Medida de la Información

Bit Dígito binario. Es el elemento más pequeño de información del ordenador. Un bit es un único dígito en un número binario (0 o 1). Los grupos forman unidades más grandes de datos en los sistemas de ordenador – siendo el Byte (ocho Bits) el más conocido de éstos.

Byte Se describe como la unidad básica de almacenamiento de información, generalmente equivalente a 8 bits. En español, a veces se le llama octeto. Cada byte puede representar, por ejemplo, una letra.

Kilobyte Equivale a 1.024 bytes (2^{10}). Se trata de una unidad de medida común para la capacidad de memoria o almacenamiento de las computadoras. Se lo abrevia kb.

Megabyte Equivale a 1.024 Kilobytes o a 1.048.576 (2^{20}) bytes. También se conoce como "Mega". Se lo abrevia MB.

Gigabyte Equivale a 1.024 Megabytes o a 1.073.741.824 (2^{30}) bytes. Es la unidad de medida más utilizada en los discos duros. También se conoce como "Giga". Se lo abrevia GB.

Terabyte Equivale a 1.024 Gigabytes o 2^{40} bytes. También se conoce como "Tera". Se lo abrevia TB.

Resumen de expresiones y formatos de printf/scanf

Expresiones aritméticas

Operador	Nombre	Ejemplo
+	Suma	a = b + 4
-	Resta	a = 3 – 4
*	Multiplicación	a = b * c
/	División	a = 4 / b
%	Módulo (operación resto)	a = numero % 2
++	Incremento	a++
--	Decremento	a--
+=	Incremento en n	a += 5
-=	Decremento en n	a -= 5

Expresiones relacionales

Operador	Nombre	Ejemplo
>	mayor que	if(a > b) {
>=	mayor o igual que	if(a >= 10) {
<	menor que	if(a + 3< b) {
<=	menor o igual que	if(a <= b + 10) {
==	igual	if(a % 2== 0) {
!=	distinto	if(a != 0) {

Expresiones lógicas

Operador	Nombre	Ejemplo
&&	y, conjunción (and)	if(a < 10 && a > 0) {
||	o, disyunción (or)	if(a < 10 || a > 20) {
!	no, negación (not)	a = ! b

La función printf: formatos

Formato	Descripción	Ejemplo	Salida
%c	Salida de un carácter	`char a = 'X';` `printf("%c", a);`	X
%d	Salida de un número entero	`int a = 1234;` `printf("%d", a);`	1234

%f	Salida de un número real float	`float a = 12.3456789`
		`printf("%f\n", a);`
		`printf("%6.2f\n", a);`
		`printf("%06.2f\n", a);`
			12.345679
			12.35
			012.35

%lf	Salida de un número real double	`double a = 12.3456789`
		`printf("%lf\n", a);`
		`printf("%6.2lf\n", a);`
		`printf("%06.2lf\n", a);`
			12.345679
			12.35
			012.35

%s	Salida de una cadena de caracteres	`char s[] = "Hola!"` `printf("%s", a);`	Hola!

La función printf: secuencias de escape

Formato	Ejemplo
\n	Nueva línea: mueve el cursor al comienzo de la línea siguiente.
\t	Tabulado horizontal: mueve el cursor a la posición siguiente del tabulado horizontal
\"	Imprime el carácter comillas
\'	Imprime el carácter comilla simple
\\	Imprime el carácter \

%%	Imprime el carácter %

La función scanf:

Formato	Descripción	Ejemplo
%c	Entrada de un carácter	`char a;` `scanf(" %c", &a); // blanco entre " y %c`
%d	Entrada de un número entero	`int a;` `scanf("%d", &a);`
%f	Entrada de un número real	`float a;` `scanf("%f", &a);`
%s	Entrada de una palabra (cadena de caracteres)	`char s[100]` `scanf ("%s", &s);`

%[^\n]	Entrada de una oración (cadena de caracteres) hasta el \n	`char s[100]`
		`scanf ("%[^\n]", &s);`
		`scanf("%c", &basura); // elimina \n`

La limpieza del buffer de teclado se puede lograr utilizando alguna de las sig. funciones:

```
setbuf(stdin, NULL);
setvbuf(stdin, NULL, _IOFBF, BUFSIZ);
```