Ingeniería Electrónica

Informática

Apunte de Cátedra

Unidad 2: Introducción al Lenguaje C

Francisco Murcia
Martin Morixe
Germán Tejero

Índice general

2. Introducción al Lenguaje C

2.1 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.

Cuadro 2.1: 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 (condicion)
{
  <instrucciones>
}
```

```
if(numero> 0)
{
  printf("Numero
     positivo");
}
```

Condional

```
if (condicion)
{
  <bloque V>
} else {
  <bloque F>
}
```

```
if (x == 0)
{
  printf ("Error
     division por 0")
} else {
  resultado = 100 / x;
}
```

Ciclo PARA

```
for (var=valor;
    condicion; incr)
{
  <bloque de
     instrucciones>
}
```

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

Ciclo condicional MIENTRAS

```
while (condicion)
{
  <bloque de
     instrucciones>
}
```

```
while (suma < 10 )
{
  suma += x;
}
```

Ciclo hacer/mientras

```
do
{
  <bloque de
     instrucciones>
} while (condicion);
```

```
do
{
  scanf("%d", &opcion);
} while (opcion != 0);
```

2.2 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).

2.2.1 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;
}
```

Ejemplo de código 2.1: Tipo de dato int

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

2.2.2 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;
}
```

Ejemplo de código 2.2: Tipo de dato float

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

2.2.3 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;
}
```

Ejemplo de código 2.3: Tipo de dato char

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

2.3 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

Cuadro 2.2: Operadores aritméticos

Ejercitar 2.1 — Hacer los ejercicios de la sección 2.15.1 Estructuras secuenciales. Para algunos ejercicios es necesario entender la operación cast (a continuación), que permite cambiar el tipo de una variable para realizar operaciones ■

2.4 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;
```

Ejemplo de código 2.4: División entera

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;
```

Ejemplo de código 2.5: División real

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;
```

Ejemplo de código 2.6: Operación cast

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);
```

Ejemplo de código 2.7: Operación cast

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.

2.5 Condicionales

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

2.5.1 Condicional simple

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

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

Ejemplo de código 2.8: Condicional simple

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;
}
```

Ejemplo de código 2.9: Condicional simple

2.5.2 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 de código 2.10: Condicional

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

```
#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;
}
```

Ejemplo de código 2.11: Condicional

Ejercitar 2.2 — Modifique el ejemplo anterior 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.

```
#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;
}
```

Ejemplo de código 2.12: Uso de condicional

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) {`

Cuadro 2.3: Operadores relacionales

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

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

Cuadro 2.4: Operadores lógicos

2.5.3 Conjunción Lógica o Producto Lógico (y, and):

El operador correspondiente se representa mediante los símbolos ’$\hat{}$ (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		
P	**Q**	**P && Q**
true	true	true
true	false	false
false	true	false
false	false	false

Cuadro 2.5: Tabla de verdad AND

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;
}
```

Ejemplo de código 2.13: Ejemplo de uso de AND

2.5.4 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		
P	**Q**	**P || Q**
true	true	true
true	false	true
false	true	true
false	false	false

Cuadro 2.6: Tabla de verdad OR

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;
}
```

Ejemplo de código 2.14: Ejemplo de uso de operador OR

2.5.5 Negación o Complemento Lógico (no, not):

Este Operador representado por un guión sobre la Variable a Complementar ejemplo $\overline{A}$ 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	
P	**!P**
true	false
false	true

Cuadro 2.7: Tabla de verdad NOT

Ejercitar 2.3 — Hacer los ejercicios de las secciones 2.15.2 Condicionales y 2.15.3 Condicionales anidados y operadores lógicos. ■

2.5.6 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.

```c
switch(expresion) {
```

```
  case valor:
  instrucciones
  break;

  case valor-2:
  instrucciones
  break;

  ...

  default:
  instrucciones;
  break;
}
```

Ejemplo de código 2.15: Sintaxis switch

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;
}
```

Ejemplo de código 2.16: Ejemplo de uso de switch

Ejercitar 2.4 — Hacer los ejercicios de la sección 2.15.4 Sentencias de decisión y switch. ■

2.6 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;
}
```

Ejemplo de código 2.17: Ejemplo de ejecución secuencial

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.

2.6.1 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
}
```

Ejemplo de código 2.18: Sintaxis for

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.

```
#include <stdio.h>

int main() {
  int i;

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

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

```
  return 0;
}
```

Ejemplo de código 2.19: Ejemplo de uso de for

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

```
#include <stdio.h>

int main() {
  int i;

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

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

  return 0;
}
```

Ejemplo de código 2.20: Números entre 2 y 100

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

```
int main() {
  int i;

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

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

  return 0;
}
```

Ejemplo de código 2.21: Números del 10 al 1

Ejercitar 2.5 — Hacer los ejercicios de las secciones 2.15.5 Ciclo de repetición exacto (for) y 2.15.6 Contadores, acumuladores, mínimos y máximos. ■

2.6.2 Contadores, acumuladores, mínimos y máximos

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

2.6.3 Ciclo while (mientras)

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

```
while (condicion) {
  instrucciones
}
```

Ejemplo de código 2.22: Sintaxis while

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.

```c
#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 numeros ingresados es: %d", total);

  return 0;
}
```

Ejemplo de código 2.23: Ejemplo de uso de while

2.6.4 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.

```c
do {
  instrucciones;
} while (condicion);
```

Ejemplo de código 2.24: Sintaxis do while

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:

```c
#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", &continuar);
  };

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

  return 0;
}
```

Ejemplo de código 2.25: Ejemplo usando while

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;
}
```

Ejemplo de código 2.26: Ejemplo de uso do while

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).

2.6.5 ¡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

```
#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;
}
```

Ejemplo de código 2.27: Solución 1

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

```
#include <stdio.h>

int main() {
  int numero, suma;

  suma = 0;
  numero = 0;

  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;
}
```

Ejemplo de código 2.28: Solución 2

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;

  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;
}
```

Ejemplo de código 2.29: Solución 3

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;
}
```

Ejemplo de código 2.30: Solución 4

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.

Ejercitar 2.6 — Hacer los ejercicios de la sección 2.15.7 Ciclo de repetición no exacto o condicional (while). ■

2.7 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).

2.7.1 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

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

Ejemplo de código 2.31: Variable global

Variable Local

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

Ejemplo de código 2.32: Variable local

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.

2.7.2 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;
```

Ejemplo de código 2.33: Constante

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
```

Ejemplo de código 2.34: Directiva del preprocesador

2.8 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	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

Cuadro 2.8: Sentencias

Con *, son los principales que utilizaremos en el curso.

2.9 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 (210). 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 (220) bytes. También se conoce como "Mega". Se lo abrevia MB.

Gigabyte: Equivale a 1.024 Megabytes o a 1.073.741.824 (230) 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 240 bytes. También se conoce como "Tera". Se lo abrevia TB.

2.10 Resumen de formatos de printf/scanf

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!

Cuadro 2.9: printf Formatos

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 %

Cuadro 2.10: secuencias de escape

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);`

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

Cuadro 2.11: scanf

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

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

Ejemplo de código 2.35: Limpieza de buffer

2.11 Contadores

Utilizamos los contadores para contar elementos. Los contadores no son más que variables generalmente enteras (int) y se las suele ir incrementado de uno en uno para contar. La forma de incrementar una variable usada como contador es:

```
int contador; /* declara una variable llamada contador */
...
contador = 0; /* siempre inicializar contadores en 0 antes de usarlos
    */
...
contador = contador + 1; /* incrementa contador en 1 */
```

Ejemplo de código 2.36: Contador

o bien, de la siguiente forma (ambas son idénticas);

```
int contador; /* declara una variable llamada contador */
...
contador = 0; /* siempre inicializar contadores en 0 antes de usarlos
    */
```

```
...
contador++; /* incrementa contador en 1 */
```

Ejemplo de código 2.37: Contador con post-incremento

Entonces, las reglas para usar contadores son:

1. Declarar la variable para ser usada como un contador como entero.
2. Inicializar dicha variable en 0 antes de utilizarla por primera vez.
3. Incrementar en uno la variable cada vez que nos interese contar un elemento
4. Mostrar el resulta
5. de la variable usada como contador.

Ejemplo: indicar cuantos ratones pesan más de 250 gramos, dado un conjunto de 5 ratones.

```
#include <stdio.h>

#define CANTIDAD_RATONES 5

int main() {
  int i, contador, peso;

  contador = 0;

  for(i = 0; I < CANTIDAD_RATONES; i++) {
    printf("Ingrese el peso de  raton (gramos): ");
    scanf("%d", &peso);

    if(peso > 250) {
      contador = contador + 1;
    }
  }

  printf("La cantidad de ratones de mas de 250 gramos es: %d\n\n",
      contador);

  return 0;
}
```

Ejemplo de código 2.38: Ejemplo de un contador

2.12 Acumuladores

Utilizamos los acumuladores para sumar (acumular) algún atributo de un elemento. Los acumuladores no son más que variables, enteras (int) o reales (float) dependiendo de lo que se quiera acumular (ambas deben ser del mismo tipo). Se les debe ir sumando el valor del atributo del elemento en cuestión:

```
int acumulador; /* declara una variable llamada contador */
...
acumulador = 0; /* siempre inicializar en 0 antes de usarlos */
...
acumulador = acumulador + atributoParaAcumular; /* suma o acumula */
```

Ejemplo de código 2.39: Acumulador

o bien, de la siguiente forma (ambas son idénticas);

```
int acumulador, atributoParaAcumular; /* variables*/
...
acumulador = 0; /* siempre inicializar en 0 antes de usarlos */
...
acumulador += atributoParaAcumular; /* suma o acumula */
```

Ejemplo de código 2.40: Acumulador

Entonces, las reglas para usar contadores son: 1. Declarar la variable para ser usada como un acumulador, y otra para contener el atributo del elemento a acumular. Ambas del mismo tipo. 2. Inicializar el acumulador en 0 antes de utilizarla por primera vez. 3. Sumar al acumulador la variable que contiene el atributo a acumular 4. Mostrar el resultado de la variable usada como acumulador.

Ejemplo: indicar el peso total de los ratones que pesan más de 250 gramos, dado un conjunto de 5 ratones. Nota: En este caso el atributo a acumular es el peso, y como se ingresa en gramos la vamos a usar como entero.

```
#include <stdio.h>
#define CANTIDAD_RATONES    5
int main() {
  int i, acumulador, peso;

  acumulador = 0;
  for(i = 0; i < CANTIDAD_RATONES; i++) {
    printf("Ingrese el peso del raton (gramos): ");
    scanf("%d", &peso);

    if(peso > 250) {
      acumulador = acumulador + peso;
    }
  }

  printf("El peso total de los ratones de mas de 250 gramos es: %d\n\n
      ", acumulador);

  return 0;
}
```

Ejemplo de código 2.41: Ejemplo de acumulador

2.13 Mínimos

Utilizamos los mínimos para obtener el valor mínimo a partir de un conjunto de valores. Los acumuladores no son más que variables, enteras (int) o reales (float) dependiendo del valor contra el que se esté comparando (ambas deben ser del mismo tipo). Se las debe ir comparando uno a uno con cada elemento del conjunto, determinando en cada paso el valor mínimo. La forma de obtener un mínimo es:

```
int minimo; /* variable donde se almacenara el valor minimo */
...
minimo = 999999; /* siempre inicializar el minimo con un valor grande
    */
...
if (valor < minimo) { /* en un ciclo, obtiene valor minimo */
  minimo = valor;
}
```

Ejemplo de código 2.42: Cálculo de un mínimo

Entonces, las reglas para usar contadores son: 1. Declarar la variable para ser usada como mínimo, y otra para ir leyendo el valor a evaluar. Ambas del mismo tipo. 2. Inicializar dicha variable en un valor más grande que cualquier valor posible (en nuestro curso, 9999999) antes de utilizarla por primera vez. 3. Dentro del ciclo que recorre los valores, comparar el mínimo con el valor, y actualizarlo si corresponde 4. Mostrar el resultado de la variable usada para obtener el mínimo.

Ejemplo: indicar el peso mínimo de un conjunto de 5 ratones.

```
#include <stdio.h>

#define CANTIDAD_RATONES    5

int main() {
  int i, pesoMinimo, peso;

  pesoMinimo = 999999;

  for(i = 0; i < CANTIDAD_RATONES; i++) {
    printf("Ingrese el peso del raton (gramos): ");
    scanf("%d", &peso);

    if(peso < pesoMinimo) {
      pesoMinimo = peso;
    }
  }

  printf("El raton mas liviano pesa: %d\n\n", pesoMinimo);

  return 0;
}
```

Ejemplo de código 2.43: Ejemplo del cálculo de un mínimo

2.14 Máximos

Utilizamos los máximos para obtener el valor máximo a partir de un conjunto de valores. Los máximos no son más que variables, enteras (int) o reales (float) dependiendo del valor contra el que se esté comparando (ambas deben ser del mismo tipo). Se las debe ir comparando uno a uno con cada elemento del conjunto, determinando en cada paso el valor máximo. La forma de obtener un mínimo es:

```
int maximo; /* variable donde se almacenara el valor maximo */
...
maximo = -999999; /* siempre inicializar maximo con un valor chico */
...
if (valor > maximo) { /* en un ciclo, obtiene valor maximo */
  maximo = valor;
}
```

Ejemplo de código 2.44: Cálculo de un máximo

Entonces, las reglas para usar contadores son: 1. Declarar la variable para ser usada como máximo, y otra para ir leyendo el valor a evaluar. Ambas del mismo tipo. 2. Inicializar dicha variable en un valor más grande que cualquier valor posible (en nuestro curso, 9999999) antes de utilizarla por primera vez. 3. Dentro del ciclo que recorre los valores, comparar el máximo con el valor, y actualizarlo si corresponde 4. Mostrar el resultado de la variable usada para obtener el máximo.

Ejemplo: indicar el peso máximo de un conjunto de 5 ratones.

```
#include <stdio.h>

#define CANTIDAD_RATONES    5

int main() {
  int i, pesoMaximo, peso;

  pesoMaximo = -999999;
```

```
    for(i = 0; i < CANTIDAD_RATONES; i++) {
      printf("Ingrese el peso del raton (gramos): ");
      scanf("%d", &peso);

      if(peso > pesoMaximo) {
        pesoMaximo = peso;
      }
    }

    printf("El raton mas pesado pesa: %d\n\n", pesoMaximo);

    return 0;
}
```

Ejemplo de código 2.45: Ejemplo de cálculo de un máximo

2.15 Guía Práctica

2.15.1 Estructuras secuenciales (asignación, entrada de datos, salida de datos)

1. Escribir un programa que imprima la suma de dos números ingresados.
2. Escribir un programa que imprima el producto de dos números ingresados.
3. Escribir un programa que imprima el número siguiente al ingresado.
4. Escribir un programa que imprima el número anterior al ingresado.
5. Escribir un programa que imprima el área y el perímetro de un rectángulo cuya base y altura se ingresan por teclado con números enteros.
6. . Escribir un programa que imprima el área de un triángulo cuya base y altura se ingresan por teclado con números enteros (área = b * h / 2). [1]
7. Escribir un programa que imprima el perímetro de un triángulo cuyos 3 lados se ingresan por teclado. (perímetro = a + b + c)
8. Escribir un programa que, dados tres números enteros, calcule e imprima el promedio. [2]
9. Escribir un programa que, dado el peso de un objeto en Kg, calcule y muestre dicho peso en libras (1 libra es igual a 0.45 Kg.).
10. Escribir un programa para calcular el porcentaje de hombres y de mujeres que hay en un grupo, dados el total de hombres y total de mujeres.
11. Un profesor prepara tres cuestionarios para una evaluación final: A, B y C. Se sabe que se tarda 5 minutos en revisar el cuestionario A, 8 en revisar el cuestionario B y 6 en el C. La cantidad de exámenes de cada tipo se entran por teclado (cantidad de A, de B y de C). ¿Cuántas horas y cuántos minutos se tardará en revisar todas las evaluaciones?

2.15.2 Condicionales

12. Escribir un programa que pida dos números y muestre el menor, suponiendo que son distintos.
13. Escribir un programa que calcule el valor absoluto de un número. (valor absoluto de un número = valor numérico de dicho número sin tener en cuenta su signo).
14. Escribir un programa que indique si se puede lograr hielo con una temperatura ambiente dada. El programa deberá imprimir “si” o “no” de acuerdo a la temperatura ingresada. (El agua se congela en 0°).
15. Escribir un programa que imprima “buenos noches” si la hora ingresada corresponde a un horario nocturno (21 a 06 horas). En caso contrario que imprima “buenos días”.

[1] Utilizar la operación cast
[2] Utilizar la operación cast

16. Escribir un programa que indique si el número ingresado es par o impar. El n° ingresado es >0.
17. Ingresar un número y luego mostrar el número consecutivo siguiente al ingresado que sea par.

2.15.3 Condicionales anidados y operadores lógicos

18. Ingresar dos números naturales y mostrar el menor o "iguales" en caso de que sean iguales.
19. Ingresar tres números y mostrar el mayor (asuma que los n° son distintos).
20. Escribir un programa que imprima el número de docena ("primera", "segunda" o "tercera") del resultado de una jugada de ruleta. Utilizar el operador lógico Y.
21. Escribir un programa que imprima "par" si el valor ingresado es 2, 4, o 6; "impar" si es 1, 3, o 5; y en cualquier otro caso "error". Utilizar el operador lógico O.
22. Escribir un programa que, dado un número, imprima "válido" si está entre 0 y 10 e "inválido" si es mayor a 10 o menor a cero.
    a. Resolver el ejercicio con el operador lógico Y.
    b. Resolver el ejercicio con el operador lógico O.

2.15.4 Sentencias de decisión y switch

23. Ingresar un número del 1 al 7 e imprimir el día correspondiente en letras (1=lunes). Utilizar sentencia if.
24. Ingresar un número del 1 al 7 e imprimir el día correspondiente en letras (1=lunes). Utilizar sentencia switch.
25. Hacer un programa que pida una letra e indique si es una vocal o no.
26. Hacer un programa que pidiendo base y altura pueda informar lo siguiente de acuerdo a la opción elegida por el usuario: 1=área rectángulo, 2=perímetro rectángulo, 3=área triángulo, 4=área de un paralelogramo (base x altura). Utilizar sentencia switch.

2.15.5 Ciclo de repetición exacto (for)

27. Imprimir los números del 1 al 10.
28. Imprimir los números del 10 al 1.
29. Hacer un programa en el que el usuario ingrese dos números enteros y se muestren todos los números comprendidos entre estos dos, incluyéndolos.
30. Hacer un programa en el que el usuario ingrese dos números reales y se muestren los números reales comprendidos entre estos dos, incluyéndolos, con incrementos de 0,5.
31. Dado un número imprimir su tabla de multiplicar (de 0 a 10).
32. Imprimir los números múltiplos de 3 que hay entre el 3 y el 100. No utilizar sentencia if.
33. Escriba un programa que calcule el área de un terreno rectangular de lados [100 - 2x] y [x] respectivamente, para los valores de x entre 10 y 30. Imprima todas las áreas para cada valor de x.
34. Escribir un programa en el que se introduce un número entero y se crea una pirámide de asteriscos. A continuación, algunos ejemplos (los espacios se muestran con un punto): Si se introduce 1:

```
*
```

Si se introduce 2:

```
.*
***
```

Si se introduce 3:

```
..*
.***
```

```
*****
```

Si se introduce 4:

```
...*
..***
.*****
*******
```

2.15.6 Contadores, acumuladores, mínimos y máximos

35. Ingresar el peso de 10 ratones e indicar cuantos pesan menos de 600 gramos (contadores).
36. Ingresar el peso de 10 ratones e indicar el peso acumulado de todos los ratones (acumulador).
37. Ingresar el peso de 10 ratones e indicar el peso del ratón más liviano (mínimo).
38. Ingresar el peso de 10 ratones e indicar el peso del ratón más pesado (máximo).
39. Calcular el peso acumulado de 40 personas.
40. Calcular el peso promedio de 20 personas.
41. Ingresar la edad de 10 personas e indicar cuántas son mayores de edad (>=18).
42. Ingresar la edad de 10 personas e indicar cuántas son mayores y cuántos menores.
43. Ingresar la edad de 10 personas e indicar mayores, menores, porcentaje de mayores, porcentaje de menores.
44. Ingresar género y edad de 10 personas e indicar: a) cantidad de mujeres, b) cantidad de varones, c) cantidad de mujeres menores de edad, d) porcentaje de varones y e) porcentaje de mujeres.
45. Calcular el peso acumulado de 15 materiales metálicos en libras (los datos se ingresan en Kg). Nota: 1 libra = 0,4536 kg.
46. En un depósito se reciben 10 barriles de lubricantes para una fábrica de rulemanes. Teniendo en cuenta que existen 4 tipos de barriles (de 25, 40, 50 y 100 litros): calcular el volumen total que ocuparán.
47. Ingresar 10 pares de temperaturas (Tmin y Tmax). Calcular el promedio de cada una.
48. Ingresar 10 pares de temperaturas (Tmin y Tmax). Indicar el valor de Tmax del día más frío.
49. Ingresar 10 pares de temperaturas (Tmin y Tmax). Indicar el valor de Tmin del día más cálido.
50. Ingresar 10 pares de temperaturas (Tmin y Tmax). Calcular el promedio de las temperaturas máximas cuando la mínima está entre -5° y 5°.
51. Se ingresan 10 números. Se pide calcular: a) la cantidad de números negativos, b) la suma de los números que se encuentran entre el 1 y el 10, c) el promedio de los 10.
52. Ingresar peso y edad de 5 ratones, mostrar la edad del ratón de mayor peso y la edad del de menor peso.
53. En la universidad se registran los siguientes datos de los alumnos de segundo año: edad, cantidad de materias aprobadas (regular), cantidad de materias aprobadas por finales y el género. Se pide:
    a Promedio de edad de los varones.
    b Cantidad de alumnos que aprobaron más de 3 finales
    c Porcentaje de alumnas y alumnos en la universidad.
    d Promedio de materias regularizadas.
54. Repetir ejercicio anterior, pero se pide:
    a Cantidad de alumnos aprobaron más de 3 materias (finales).
    b Porcentaje de materias aprobadas (finales) entre alumnos y alumnas.
    c Promedio de edad de alumnos (varones) que hasta el momento no aprobaron ningún final

2.15.7 Ciclo de repetición no exacto o condicional (while)

55. Ingresar el peso de ratones hasta que el peso acumulado supere los 4 kg, e indicar cuantos pesan menos de 600 gramos (contador).
56. Ingresar el peso de ratones hasta que el peso acumulado supere los 4 kg, e indicar a) cuantos pesan menos de 600 gramos (contador), b) el peso acumulado de todos los ratones (acumulador), c) el peso del ratón más liviano (mínimo), d) el peso del ratón más pesado (máximo).
57. Calcular el promedio de edad de un grupo de personas hasta que haya 5 menores, e indicar cuántas son mayores de 18 años.
58. Ingresar la edad de personas hasta que haya 5 mujeres, e indicar a) cuántas personas son mayores, b) cuántas mujeres son mayores, c) cuántos hombres se procesaron, d) suma de edades de los hombres.
59. Ingresar el peso de vacas que van siendo cargadas a un camión que soporta 40 mil kg de carga como máximo. Hacer un programa que muestre el total de vacas cargadas en el camión y su peso (¡¡cuidado!! no ingresar la última vaca que hace que el peso exceda el soportado por el camión).
60. Ingresar temperaturas hasta que el promedio sea mayor que 20 grados, y mostrar la menor y la mayor.
61. Ingresar temperaturas mientras el promedio sea menor que 20 grados, y mostrar la menor y la mayor.
62. 2. Ingresar pares de temperaturas (T1 y T2) hasta que la suma del par ingresado sea mayor que 40 grados. Hallar el promedio de las temperaturas T1 y el promedio de las temperaturas T2.
63. Ingresar pares de temperaturas (T1 y T2) mientras que T1 no sea 0° (este par no se procesa). Indicar la cantidad de temperaturas T1 que están entre 5° y 15° y la mayor T2.
64. Ingresar números hasta que la sume alcance los 1000. Hallar: a) la cantidad de números ingresados, b) la cantidad de números negativos, c) la suma de los números que se encuentran entre el 1 y el 10, d) el promedio de los números mayores a 10.