# ***TP Nº 3: Control de Flujo***

| **Los ejercicios que no están incluídos en este documento, es porque la solución es únicamente un código, que pueden encontrar en el Drive junto a este documento.** |
| --- |

### ***Ejercicio 3***

#include <stdio.h>

#include "getnum.h"

int

main(void)

{

double venta;

do

{

venta = getfloat("\nIngrese la suma vendida: ");

}while (venta < 0);

if (venta > 4000)

printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta \* 0.09);

if (venta >= 2000 && venta <= 4000)

printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta \* 0.07);

if (venta >= 1000 && venta < 2000)

printf("El monto a cobrar es: $%10.2f\n", 300.0 + venta \* 0.05);

if (venta < 1000)

printf("El monto a cobrar es: $%10.2f\n", 300.0);

return 0;

}

El programa anterior satisface la correctitud pero tiene serios defectos:

* no hace preguntas excluyentes. Si ya sabemos que es superior a 4000, no tiene sentido preguntar luego si está entre 2000 y 4000. Y en caso de cambiar el rango (por ejemplo 3000 por 2000) hay que cambiar en dos lados
* Repite código, por lo que de ser necesario cambiar o corregir la impresión, habría que repetir la modificación 4 veces.
* No utiliza constantes simbólicas. De tener que cambiar el sueldo básico se lo debe cambiar en 4 lados.

#include <stdio.h>

#include "getnum.h"

#define BASICO 300.0

#define PORC1000 0.05

#define PORC2000 0.07

#define PORC4000 0.09

int

main(void)

{

double venta, total = BASICO;

do

{

venta = getfloat("\nIngrese la suma vendida: ");

}while (venta < 0);

if (venta > 4000)

total += venta \* PORC4000;

else if (venta >= 2000)

total += venta \* PORC2000;

else if (venta >= 1000)

total += venta \* PORC1000;

printf("El monto a cobrar es: $%10.2f\n", total);

return 0;

}

### ***Ejercicio 4***

1. **imprime “#######” y “\*\*\*\*\*\*\*”**
2. **imprime “#######”**

if ( x < 3 )

{

printf ("x<3");

if ( z < 5 )

printf ("x<3 y z<5");

}

else

printf ("x >= 3");

if ( x = = 1)

{

if ( z > 1)

printf ( "x = = 1 y z > 1");

}

else

if ( x > 1 )

printf ("x > 1");

else

printf("x < 1");

***Ejercicio 6***

1. int a = 0;

while (a <= 4)

**{**

resultado += a;

a++;

**}**

b) if ( sueldo >= 1000 ) **/\* aquí había una coma \*/**

printf("El sueldo es mayor o igual a 1.000\n")**; /\* aquí había una coma \*/**

else

printf("El sueldo es menor a 1.000\n");

c) La variable acumuladora ***var2*** no está inicializada. **No** hay que asumir que las variables se inicializan en cero.





d) while (‘Y’ != (c = getchar()))

printf (“Pulse Y para abandonar el ciclo\n”);

e) **int** c;

El valor retornado por la macro getchar es de tipo **int** debido a que necesita poder devolver además de todos los caracteres posibles (comprendidos en un char) una indicación de error o fin de entrada (EOF). Por lo tanto utiliza un tipo de dato de mayor rango.

Además hay que tener en cuenta que algunos compiladores consideran al char como **unsigned char** y otros como **signed char**, en el primer caso la condición es siempre falsa pues la variable c jamás puede ser igual a un valor negativo. Pensar qué problemas pueden ocurrir en el segundo caso.

f) Para imprimir el nombre de los días, se debe reemplazar la invocación a **printf** por un **switch**. Más adelante veremos una mejor forma de hacerlo (usando vectores) pero por ahora es la mejor solución.

switch(dia) {

case LUN : printf("LUN"); break;

case MAR: printf("MAR"); break;

....

}

***Ejercicio 7***

El programa intenta calcular el promedio de una cantidad *n* de números, ingresados por el usuario, acumulando la suma de los mismos en *suma*. Pero al presentar el promedio en pantalla, divide por el último valor ingresado y no por la cantidad de números *n*.

El único caso en que el programa entrega el valor correcto, es si el último valor ingresado coincide con la cantidad *n* de números.

Ésta condición NO es suficiente como para considerar que el programa funciona correctamente.

***Ejercicio 10***



a) float x = **1.0 / 3.0;** 

while (x **<** 0.52)

x += .01;

La conversión de 0.52 al formato de punto flotante se pierde precisión. Por ejemplo 0.519999… ( probarlo en <https://www.h-schmidt.net/FloatConverter/IEEE754.html>). Y lo mismo con 0.01

Como x es float, la cantidad de dígitos decimales será menor que la cantidad de dígitos decimales de la constante double 0.52, al hacer la promoción de x a double completará con ceros en la parte decimal, por eso nunca será igual al double 0.52.

b) #define EPSILON 0.001

....

/\* No se puede preguntar por la igualdad en los reales por el

\*\* error de representación. Se debe usar un margen de error (en

\*\* este caso de un milésimo \*/

for ( y = .1; **y -1.0 > EPSILON || y -1.0 < -EPSILON**; y += .1)

c) switch (n) {

case 1:

printf(“El número es 1\n”);

**break; /\* Falta el break \*/**

case 2:

...

1. Hay que encerrar entre paréntesis la asignación, pues la comparación tiene mayor precedencia que la asignación.

**while ( (c = getchar() ) != EOF )**

***Ejercicio 12***

#include <stdio.h>

int

main(void)

{

int i;



for (i = 3; i <= 23; i+=5)

printf("%d,",i);

putchar('\n');



for (i = 20; i >= -10; i-=6)

printf("%d,",i);

putchar('\n');



for (i = 19; i <= 51; i+=8)

printf("%d,",i);

putchar('\n');

return 0;

}

***Ejercicio 14 (resuelto en videos)***

#include <stdio.h>

#define DELTA ('a' - 'A')

int

main(void)

{

int car;

int menor='z';

int cantMay=0, cantMin=0;

int eraMay=0, esMay;

*/\* Lee caracteres de la entrada estándar mientras no sea fin de entrada y*

*\* sea una letra ( mayúscula o minúscula ) del alfabeto ingles*

*\*/*

while ((car=getchar())>= 'A' && car<= 'Z' || car>= 'a' && car<= 'z'

|| car ==' ')

{

if ( car != ' ' )

{

if (esMay = (car >= 'A' && car <= 'Z'))

{

cantMay++;

car += DELTA;

}

else

cantMin++;

if (car < menor)

{

menor = car;

eraMay = esMay;

}

}

}

if ( cantMin + cantMay > 0 )

printf("\nLa letra menor era: %c\n Cantidad de mayúsculas: %d\n"

"Cantidad de minúsculas: %d\n",

eraMay ? menor-DELTA:menor, cantMay, cantMin);

}

else

printf ("\nNo se han ingresado letras\n");

return 0;

}

***Ejercicio 15***

#include <stdio.h>

#include "getnum.h"

*/\* Solo se consideran binarios positivos \*/*

int

main(void)

{

int binario;

int decimal=0;

int base=1;

int resto;

binario = getint("Ingrese un número binario (hasta 10 digitos):");

for (base = 1; binario > 0; binario /=10, base \*=2)

{

if ( (resto = binario % 10) > 1 ) 

{

printf("\nNo es un número binario !!!\n");

return 1;

}

else if (resto)

decimal += base;

}

printf("\nEquivalente decimal = %d\n",decimal);

return 0;

}

***Ejercicio 17***

#include <stdio.h>

int

main(void)

{

long i=1;

*/\* Ejecutar el ciclo mientras los números que se obtienen sean múltiplos*

*\*\* de 10. Dejarán de ser múltiplos cuando el resultado esté fuera del rango*

*\*/*

while ( (i\*=10) % 10 == 0 )

printf("%ld\n",i);

return 0;

}

## ***Para pensar: ¿ Es un método seguro ?***

***Ejercicio 19***

a) Números enteros

#include <stdio.h>

#include "getnum.h"

#define DIGITO 5

int

main(void)

{

int n;

int contador=0;

n = getint("Ingrese un entero: ");

*/\* Si es negativo lo convertimos para trabajar en positivo \*/*

if ( n < 0 )

n \*= -1;

while ( n != 0 )

{

if ( (n % 10) == DIGITO )

contador++;

n /= 10 ;

}

printf("\nCantidad de dígitos iguales a %d: %d\n", DIGITO,contador);

return 0;

}

b) Números reales

Presentamos dos soluciones alternativas para el caso de números reales.

El primero presenta la solución matemáticamente correcta, pero que en la práctica no funciona debido al error de representación del punto flotante, que pierde precisión en la parte decimal.

Por ejemplo, si ingresamos 34.456, el número almacenado en realidad es el 34.456001282.....A esto se le agrega el error de la multiplicación en cada paso.

#include <stdio.h>

#include "getnum.h"

#define DIGITO 5

#define ERROR 0.000000001

int

main(void)

{

float x;

long parteEntera;

int contador=0;

x = getfloat("Ingrese un número real: ");

printf("\nNúmero leído: %20.9f\n",x);

if ( x < 0 )

x \*= -1;

parteEntera = (long) x;

x = x - parteEntera;

while ( parteEntera != 0 )

{

if ( (parteEntera % 10) == DIGITO )

contador++;

parteEntera /= 10 ;

}

printf ("\nDígitos iguales a %d en parte entera: %d\n",

DIGITO,contador);

contador = 0;

while ( x > ERROR )

{

x \*= 10;

parteEntera = (long) x;

if ( parteEntera == DIGITO )

contador++;

x = x - parteEntera;

}

printf ("\nDígitos iguales a %d en parte decimal: %d\n",

DIGITO, contador);

return 0;

}

La siguiente solución lee los dígitos como caracteres, sin almacenarlos. A medida que se van leyendo, se van contando cuántos de ellos son igual a '5'. Al estar basada en la lectura de caracteres no hay errores de representación.

#include <stdio.h>

#define DIGITO '5'

int

main(void)

{

int c;

int huboPunto = 0;

int cntInt = 0;

int cntDec = 0;

printf ("\nIngrese un número real: ");

while ( ( (c=getchar()) >= '0' && c <= '9') ||

( c == '.' && !huboPunto ) )

{

if ( c >= '0' && c <= '9' )

{

if ( c == DIGITO )

( huboPunto ) ? cntDec++ : cntInt++;

}

else

huboPunto = 1;

}

printf ("\nDigitos iguales a %c en parte entera: %d\n",

DIGITO,cntInt);

printf ("Digitos iguales a %c en parte decimal: %d\n",DIGITO,cntDec);

return 0;

}

***Ejercicio 20 (resuelto en videos)***

Presentamos primero una versión que, si bien funciona, no es aceptable pues es ineficiente.

#include <stdio.h>

#define EPSILON 0.0000001

int

main(void)

{

long factorial=1;

float e=1, anterior=0;

int i=1, j;

printf("%-10s %10s\n","N","e");

while ( e - anterior > EPSILON )

{

printf ("%-10d %10.7f\n",i,e);

anterior = e;

e += 1.0 / factorial;

i++;

for (j=2, factorial=1; j<=i; j++)

factorial \*= j;

}

return 0;

}

Presentamos ahora una versión correcta

#include <stdio.h>

#define EPSILON 0.0000001

int

main(void)

{

long factorial=1;

float e=1, anterior=0;

int i=1;

printf("%-10s %10s\n","N","e");

while ( e - anterior > EPSILON )

{

printf ("%-10d %10.7f\n",i,e);

anterior = e;

e += 1.0 / factorial;

i++;

factorial \*= i; 

}

return 0;

}

***Ejercicio 23***

| int n,c;  */\* al salir del ciclo n debe ser un número par mayor que cero \*/*  do  n=getint("");  while(**n <= 0 || n % 2 != 0** );  */\* otra forma de hacerlo \*/*  while(**!(n > 0 && n % 2 == 0)**);    */\* este ciclo debe incluir la lectura de caracteres y se debe*  *\*\* ejecutar mientras no sea fin de archivo y los caracteres sean*  *\*\* dígitos o minúsculas*  *\*/*  while (**(c=getchar())!=EOF && (c>='0' && c<='9' || c>='a' && c<='z'**)  putchar(c); |
| --- |

***Ejercicio 24***

La salida que se obtiene al ejecutar el programa es la siguiente:

***NO SON iguales: a vale 0.1 que no es igual a 0.1***

Obviamente es absurda e involucra conceptos **MUY IMPORTANTES**:

* Como ya se explicó en Introducción a la Informática, los valores reales se almacenan en memoria según la norma IEEE 754. Es decir que se almacenan expresados como mantisa y exponente. Por este motivo una gran cantidad de valores no tienen representación exacta **como por ejemplo el 0.1**. A la pérdida de precisión debido a este error de representación se le debe agregar que 0.1 es una constante de ***tipo double*** que al ser almacenada en una variable de ***tipo float*** pierde aún más precisión.
* La función **printf**  *por default* muestra los valores reales redondeados a una cierta precisión, si uno desea ver una precisión mayor debe colocar la precisión deseada explícitamente como se muestra en el siguiente ejemplo:

| #include <stdio.h>  int  main(void)  {  float a=0.1;  if (a==0.1)  printf("SON iguales\n");  else  printf("NO SON iguales:a vale %.20g que no es igual a 0.1\n", a);  return 0;  } |
| --- |

Y su salida es :

***NO SON iguales: a vale 0.10000000149011611938 que no es igual a 0.1***

***Ejercicio 26***

1. Como cada if es mutuamente excluyente se debería usar else. En el código presentado siempre se harán 3 comparaciones (3 restas), mientras que en el fragmento propuesto se harán a lo sumo 3, pero si a es mayor que cero, entonces se hará solo esa comparación.

| int a = getint("Ingrese un número entero:");  if ( a > 0 )  printf("Es mayor que cero\n");  else if ( a < 0 )  printf("Es menor que cero\n");  else if ( a == 0)  printf("Es igual a cero\n"); |
| --- |

1. A la variable a se le asigna 0 (falso) si b > 0 es verdadero, y se le asigna 1 (verdadero) si b > 0 es falso, por lo que se podría escribir así

| int a = ! ( b > 0 ); |
| --- |

Ahora bien, más sencillo que decir que no es mayor que cero, es decir que es menor o igual que cero, o sea

| int a = ( b <= 0 ); |
| --- |

Los paréntesis no son necesarios, están por claridad.

1. El ciclo tiene una clara condición de salida, que es cuando a llega a 10, por lo que se lo puede reescribir de la siguiente manera

| int a = 0;  while( a < 10 ) {  f(a); // f es una función  a++; // más claro que a = a + 1  } |
| --- |

Teniendo en cuenta que a tomará los valores de 0 a 10, quedará aún más claro si usamos un for

| int a;  for( a=0; a < 10; a++) {  f(a); // f es una función  } |
| --- |

Si la variable a no será usada luego del ciclo for, entonces se puede escribir de la siguiente forma

| for(int a=0; a < 10; a++) {  f(a); // f es una función  } |
| --- |

1. Similar al punto b, podemos abreviar asignando directamente el resultado de la comparación. Los paréntesis no son necesarios, están por claridad.

| a = ( b > 0 ); |
| --- |

1. Para que a c se le asigne 10 tiene que cumplirse a > 0 y b < 0. En cualquier otro caso se le asigna 20

| if ( a > 0 && b < 0 )  c = 10;  else  c = 20; |
| --- |

o mejor aún, usando un condicional

| c = ( a > 0 && b < 0 ) ? 10 : 20; |
| --- |

1. Queda más claro inicializar las variables en el for, sobre todo si hubiera más líneas de código entre la inicialización y el for. También el incremento de la variable i colocarlo en el for.

| int i, a;  for ( i=0, a=10; i < 10; i++ ) {  a += getint("");  }  printf("El valor de a es %d\n", a); |
| --- |

1. Si bien en este caso se podría reemplazar el ciclo por una cuenta, la idea es ver ejemplos de estilo.

En este caso estamos preguntando 100 veces lo mismo, cuando sabemos que sólo será verdadero la primera vez, por lo que se debe sacar fuera del ciclo el caso especial

| int suma = getint("Ingrese la base:");  for ( int i=1; i < 100; i++) {  suma += i;  }  printf("La suma es %d\n", a); |
| --- |

1. El mismo error que en el punto anterior, tendría que ser

| cant = 1;  last = v[0];  for(int i=1; i<dim; i++){  if(last != v[i]) {  last=v[i];  cant++;  }  } |
| --- |