

TPN° 2: Expresiones en C. Entradas y Salidas

Ejercicio 1

- a) Son equivalentes. La constante 'ñ' (o cualquier otro valor ASCII) es de tipo entero, por lo que el casteo (int) es innecesario.
- b) No son equivalentes. En la operación " $x = y / 3.0$ " se utiliza aritmética de punto flotante pues el divisor es un operando de punto flotante (tipo *double*). En la operación " $(float) (y/3)$ " la división se realiza con aritmética de enteros y luego el resultado es promovido a *float*.
- c) No son equivalentes, pues en ambos casos se utiliza aritmética de punto flotante, pero en el segundo caso x es entero y toma sólo la parte entera de la operación (trunca).
- d) El resultado es equivalente, pues x termina con el valor 1. Pero las operaciones no son equivalentes, pues en el primer caso se obtiene 1.8333333 (5.5/3) y en el segundo se obtiene 1.666666 (5/3.0) y ambos son truncados a entero.
- e) Son equivalentes en cuanto a su valor numérico pero no en cuanto a su tipo.
- f) Son equivalentes. En el primer caso efectúa una promoción a entero y en el segundo efectúa una democión.
- g) No son equivalentes, pues dependen de la plataforma. En UNIX / LINUX el entero ocupa 4 bytes, en DOS el tipo entero ocupa dos bytes. Por lo tanto en DOS en el primer caso tenemos 1 bit para el signo y los 15 restantes para representar el número.
- h) Son equivalentes porque el 120 es representable en un char independientemente de que éste sea signado o no. Recordar que el lenguaje C no establece si el char es signado o no, por lo cual se lo debe declarar explícitamente.
- i) Si el char es unsigned (lo que ocurre en algunas arquitecturas), son equivalentes. Si el char es signed no son equivalentes.
- j) No son equivalentes porque el 300 no es representable en un char independientemente de que éste sea signado o no.
- k) No son equivalentes. En el primer caso no se puede determinar cuál de los dos términos se evalúa primero. Si primero evalúa ($x = 2$) y luego ($y = x$) en ambos códigos las variables terminan con el mismo valor. Pero si primero se evalúa ($y = x$) y luego ($x = 2$), en el primer caso tanto x como y toman un valor incierto.

Ejercicio 2

```
#include <stdio.h>

int
main(void)
{
    printf("Este\nes\nun\nprograma\nen\nnC\n");
    return 0;
}
```

Ejercicio 3

En algunas plataformas compila correctamente, pero en otras genera un error de compilación. En caso de querer continuar una cadena de caracteres en otra línea debe hacerse como en el ejemplo siguiente:

```
printf ( "Hola que"  
        "tal \n" );
```

En este caso, en la salida ambas cadenas aparecen en la misma línea.

Ejercicio 4

- ❖ `printf("\a\n");`
Imprime el caracter BELL (suena un "beep")
- ❖ `printf("%c\n", 7);`
Idem al caso anterior, ya que el valor numérico del caracter BELL es 7
- ❖ `printf("%d\n", 7);`
Imprime el entero 7
- ❖ `printf("%f\n", 7);`
La salida es *impredecible*

El número 7 se encuentra almacenado, en el stack, en una dirección de memoria en el formato de complemento a la base y ocupando el tamaño de un entero, la función *printf* tomará del stack la cantidad de bytes necesarios para completar un *double* (ya que la cadena de formato le indica que el dato es de ese tipo) interpretándolo como un valor expresado en notación IEEE de punto flotante de 64 bits. Como el tipo *int* ocupa menos bytes que el tipo *double* (en Linux, el *int* ocupa 4 bytes frente a los 8 del *double*), en este caso la función levantará del stack los cuatro bytes que ocupa el valor 7 más los cuatros bytes que siguen (con contenido desconocido) para representar un *double*.

- ❖ `printf("%g\n", 7);`
La salida es *impredecible*, por las mismas razones que en el ejemplo anterior.

Ejercicio 5

```
#include <stdio.h>  
  
int  
main(void)  
{  
    char c = 't';  
  
    printf("c = %c\n", c);  
    printf("c = %d\n", c);  
    printf("c = %f\n", c);  
    return 0;  
}
```

La función `printf` recibe en el stack el valor entero 116 (valor ASCII de la letra 't'). Tener en cuenta que la asignación `c = 't'` es equivalente a la asignación `c = 116`.

Con `%c` se imprime el caracter que se recibió: 't'

Con `%d` se imprime el ASCII correspondiente al caracter enviado, o sea, 116.

Con `%f` la salida es *impredecible*.

Ejercicio 6

Se obtienen las siguientes salidas:

```
num1=          53
num2= 0000000004
num1= 53
num1+num2=      57
num1+num2=     ....      (salida impredecible!!!)
num3= 6.87
num1= 53
num2= 4
num3= 6.9
num3(como entero)=...      (salida impredecible!!!)
num1 / num2 = 13
num2 / num1 = 0
esta es la ultima prueba
```

Ejercicio 7

```
#include <stdio.h>

int
main(void)
{
    int edad= 25;
    float longitud= 185.654;
    char letra= 'Z';

    /* a) */
    printf("%-5d\n",edad);

    /* b) */
    printf("%10d\n",edad);

    /* c) */
    printf("%-10.2f\n",longitud);

    /* d) */
    printf("%8d\n",letra);

    return 0;
}
```

Ejercicio 8

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

int
main(void)
{
    float velocidad ;

    velocidad = getfloat("\nIngrese una velocidad en m/s: ");

    printf ("\nEn km/h = %f\n", (velocidad * 3600) / 1000 );
    return 0;
}
```

Los valores enteros son promovidos a float.

Ejercicio 9

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

int
main(void)
{
    int num ;

    num = getint("Ingrese un entero positivo: ");
    printf ("\n%d / 2 = %d\n", num, num >>1 );
    return 0;
}
```

Ejercicio 10

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

int
main(void)
{
    int seg;

    seg = getint("Ingrese una cantidad de segundos: ");
    printf ("\n%d seg. = %d h %d m %d s\n",
            seg, seg / 3600, (seg % 3600)/60, seg % 60);

    return 0;
}
```

Ejercicio 11

```
#include <stdio.h>

int
main(void)
{
    char letra1,letra2 ;

    printf("\nIngrese dos caracteres (uno al lado del otro): ");
    letra1 = getchar();
    letra2 = getchar();
    printf ("\nEl caracter de mayor valor ASCII es %c\n",
            (letra1>letra2) ? letra1 : letra2);
    return 0;
}
```

Se deberá redireccionar con: **a.out < entrada.txt**, y las salidas serán:

- a) B
- b) A
- c) A

Ejercicio 12

```
#include <stdio.h>

int
main(void)
{
    char lt ;

    printf("Ingrese un carácter:");
    lt = getchar();
    printf ("El caracter %s es letra\n",
        (lt>='A' && lt<='Z' || lt>='a' && lt<='z') ? "SI" : "NO" );

    return 0;
}
```

Ejercicio 13

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

int
main(void)
{
    int a,b ;

    a = getint("\nIngrese el numero1:");
    b = getint("Ingrese el numero2:");

    printf ("\n%d %ses multiplo de %d\n",b,(!a || b % a)? "no ": "",a);
    return 0;
}
```

Ejercicio 14


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

int
main(void)
{
    int n1,n2 ;

    n1 = getint("\nIngrese el numero1:");
    n2 = getint("\nIngrese el numero2:");

    printf("\nEl promedio es %.1f\n", (n1+n2)/2.0);
    printf("La suma es %d\n",n1+n2);
    printf("El menor es %d\n", (n1<n2)? n1 : n2);
    printf("El mayor es %d\n", (n1<n2)? n2 : n1);
    printf("%sson iguales\n", (n1==n2)? "" : "no ");
    return 0;
}
```

Queremos que el resultado de la división sea de tipo float, también se podía castear (n1+n2).



Ejercicio 15

```
#include <stdio.h>

#define DELTA 'A' - 'a'

int
main(void)
{
    char letra ;

    printf("Ingrese una letra :");
    letra = getchar();

    printf("\n%c en mayusculas = %c\n",letra,
          (letra>='a' && letra<='z')? letra+DELTA : letra);

    return 0;
}
```

Ejercicio 16

```
#include <stdio.h>

int
main(void)
{
    char a,b ;

    printf("Ingrese dos caracteres seguidos: ");
    a = getchar();
    b = getchar();

    printf("\nEl caracter '%c' es %s al caracter '%c'\n",
          a, (a==b)? "igual" : (a>b)? "mayor" : "menor",b);

    return 0;
}
```

Ejercicio 17

"3" + "4";	no compila (no se pueden sumar cadenas de caracteres)
'3' + '4';	compila (se obtiene 51+52, es decir 103)
3 + 4;	compila (se obtiene 7)
'3' + 4;	compila (se obtiene 51+4, es decir 55)
3 + '4';	compila (se obtiene 3+52, es decir 55)
"3" + '4';	la operación es válida, pero no es de utilidad
"3" + 4;	la operación es válida, pero no es de utilidad

Ejercicio 18

- a) FALSO. Comienza a escribir en donde se encuentra el cursor.
- b) VERDADERO
- c) FALSO. Las declaraciones de variables deben encontrarse al comienzo de los bloques.
- d) FALSO. Se las puede imprimir en un solo *printf*, usando la secuencia de escape '\n' (ver el código del *Ejercicio 1*)

Ejercicio 19

Ej 1.2: Si el valor de *c* hace que *\c* sea una secuencia de escape, se imprime el carácter correspondiente a la misma (ver ejercicio 7), sino se imprime el mismo carácter *c* enviado, como un carácter común.

Ej: 1.6: La expresión *c = getchar() != EOF* primero lee un carácter desde la entrada, luego lo compara con *EOF* y según sean o no iguales asigna a la variable *c* un 1 o un 0 (verdadero o falso).

Ej 1.7: Para imprimir el valor EOF usamos: *printf("El valor de EOF es %d", EOF);*