TP Nº4: Macros y Funciones

Ejercicio 1 / 2

a) Al finalizar la ejecución:

a = 1 ***b = 3***

b) El programa no compila debido a que se pretende aplicar un operador de incremento a ***PI*** que es una constante. Esto es incorrecto ya que las constantes no son un l-value, es decir, no pueden estar del lado izquierdo de una asignación porque su valor no es modificable.

c) La macro ***CUBO*** se debe escribir:

#define CUBO(x) ((x) * (x) * (x))

ya que sin los paréntesis, al realizar el preprocesamiento, la macroexpansión quedaría de la siguiente forma:

```
b = a + 1 * a + 1 * a + 1;
```

y sería evaluada con la siguiente precedencia:

a + (1 * a) + (1 * a) + 1 es decir 4 + 4 + 4 + 1 = 13

que no es lo deseado.

La función ***cubo*** es correcta ya que al ser una función, los parámetros con los cuales es invocada son evaluados en tiempo de ejecución, y recibe a través del stack el valor de los mismos:

cubo(a+1) es 5 * 5 * 5 = 125

d) En este ejercicio, el valor final de las variables a las que se les asignan las invocaciones a la macro CUBO (***b*** y ***c***) es IMPREDECIBLE. Esto se debe a que el lenguaje ANSI C no especifica el orden en que se evalúan los operadores de igual precedencia dentro de una expresión.

Por ejemplo, en algún compilador el resultado en la variable ***b*** puede llegar a ser 150. La forma en que llega a este valor es la siguiente :

Macroexpansión: b = ++a * ++a * ++a;

Inicialmente el valor de la variable ***a*** es 3. Al evaluar el primer término, se detecta que primero hay que hacer un incremento y entonces el valor de la variable ***a*** queda en 4. Luego se requiere evaluar la misma variable para el segundo término entonces nuevamente lo incrementa (antes de utilizar su valor), por lo que el valor de ***a*** es 5. Una vez que incrementó la variable ya la puede usar, por lo que la multiplicación es 5 * 5 = 25. Luego se multiplica por el tercer término, lo incrementa (queda en 6) y lo multiplica por 25 (5 * 5 * 6 = 150).

En otros compiladores se podría obtener resultados distintos.

En el caso de invocaciones a funciones el parámetro se evalúa una única vez y su valor es enviado a través del stack a la función cubo. Los valores finales son:

d = 4 ***e = 64*** ***f = 27***

e) Primera invocación: Es correcta

a = 20	Mantiene su valor original
b = 5	Mantiene su valor original
c = 4	Tiene el resultado de la división

Segunda invocación: No es correcta. Al usar como parámetros de una macro post/pre incrementos/decrementos no sabemos cuántas veces se utilizará el parámetro dentro de la macro.

m = 6	Aparece incrementado en 1, lo que resulta lógico porque en la invocación está postincrementado.
n = 2	Aparece decrementado en 2. Este efecto no es previsible, ya que en la invocación se realiza un único postdecremento.
p = 1	Tiene el resultado de la división

Tercera invocación: Lo marcado en negrita son proposiciones inválidas ya que no se puede realizar una asignación sobre un valor constante y tampoco un postincremento. No son l-values.

```
{int i;
   for (0 = 0 , i = x –y ; i >= 0; 0++, i - = y);
 }
```

Corregido este error (reemplazando 0 por z en la invocación) los valores finales son los siguientes:

x = 15	Mantiene el valor original
y = 3	Mantiene el valor original
z = 5	Tiene el resultado de la división

De este ejercicio surge como recomendación no usar operaciones de pre/post incremento/decremento como parámetros en la invocación de macros. Tampoco sería correcto usar cualquier otra operación que implique la alteración del contenido de una variable que aparezca más de una vez en la expresión.

Ejercicio 3

```
#include <stdio.h>

#define     PI          3.1416
#define     VOLUMEN(r)  (4.0 / 3.0 * PI * (r) * (r) * (r))

int
main(void)
{
      int i;

      for(i=1; i<=10; i++)
            printf("Radio %2d -> Volumen %8.2f\n",i, VOLUMEN(i));
      return 0;
}
```

Ejercicio 4

```
#include <stdio.h>

#define SWAP(tipo, x, y) {tipo temp; temp= x; x= y; y= temp;}

int
main(void)
{
      int a = 5, b = 3;

      printf("Antes del swap: a=%d, b=%d\n",a,b);

      /* El primer parámetro corresponde al tipo de las variables a intercambiar */
      SWAP(int, a, b);

      printf("Despues del swap: a=%d, b=%d\n",a,b);
      return 0;
}
```

Ejercicio 5

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

/* IMPORTANTE: Los paréntesis no están de más. a  y b pueden ser expresiones */
#define MAXIMO2(a,b,resp)  (resp = ((a)>(b)) ? (a) : (b))

int
main(void)
{
      int num1,num2,maximo;

      num1 = getint("Ingrese primer numero:");
      num2 = getint("Ingrese segundo numero:");

      MAXIMO2(num1,num2,maximo);
      printf("El mayor entre %d y %d es %d\n",num1,num2,maximo);

      /* Otra forma de hacerlo */
      printf("El mayor entre %d y %d es %d\n", num1,num2,
             MAXIMO2(num1,num2,maximo));

      return 0;
}
```

Es importante notar que la macro MAXIMO2 está escrita de forma tal que puede formar parte de una expresión (ver segundo ejemplo).

Ejercicio 6

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

#define MAXIMO2(a,b,resp)  (resp = ((a)>(b)) ? (a) : (b))

/* Esta macro se basa en la anterior, calculando el máximo de los dos
** primeros, y luego comparando este máximo con el tercer número
*/
#define MAXIMO3(a,b,c,resp) (resp = (MAXIMO2(a,b,resp)>(c)) ? resp : (c))
```

```
int
main(void)
{
     int num1,num2,num3,maximo;

     num1= getint("Ingrese el primer numero: ");
     num2= getint("Ingrese el segundo numero: ");
     num3= getint("Ingrese el tercer numero: ");

     MAXIMO3(num1,num2,num3,maximo);
     printf("El mayor entre %d, %d y %d es %d\n",num1,num2,num3, maximo);

     /* Otra forma de hacerlo */
     printf("El mayor entre %d, %d y %d es %d\n",num1,num2,num3,
                                        MAXIMO3(num1,num2,num3,maximo));

     return 0;
}
```

Ejercicio 7

Presentamos una primera solución con operador condicional.

```
#include <stdio.h>

#define isdigit(digit) ((digit>='0' && digit<='9') ? 1 : 0)

int
main(void)
{
     int letra;

     letra=getchar();

     if ( isdigit(letra) )
            printf("Es un dígito\n");
     else
            printf("No es un dígito\n”);

     return 0;
}
```

Una segunda solución posible es:

```
#define isdigit(digit) (digit>='0' && digit<='9')
```

Presentamos una solución NO válida, que el alumno puede llegar a estar tentado de realizar. Queda como tarea para el alumno explicar por qué no funciona.

```
#define isdigit(digit) { if(digit>='0' && digit <='9')  \
                                return 1;               \
                         else                           \
                                return 0;               \
                        }
```

Ejercicio 8

a) No se puede definir una función dentro de otra.
b) Falta el retorno de la función: **return respuesta;**
c) El punto y coma luego del encabezado de la función es incorrecto. Además falta el retorno de la función para la rama del else: **return n + 1**;
d) No se puede redefinir "a". "a" es el nombre del parámetro de la función, por lo tanto no puede ser a la vez el nombre de una variable.
e) ***Letra****:* debe ir entre comillas dobles y ***n*** entre comillas simples.

Ejercicio 9

No funciona correctamente. Por ejemplo: si el horario de entrada es a las 9:30hs, entonces:

ENT_HORA=9 y ENT_MINUTOS=30

Si un empleado llega a las 8:45, la función devuelve falso:

```
(8<= ENT_HORA && 45 <= ENT_MINUTOS)
```

La solución es la siguiente:

```
((hora < ENT_HORA)  || (hora == ENT_HORA && minutos <= ENT_MINUTOS))
```

Otra solución podría ser construir un número que tenga el "peso" de las horas y los minutos.

```
(hora * 100 + minutos  <= ENT_HORA * 100 +  ENT_MINUTOS)
```

Ejercicio 10

```
int
dcm ( int a, int b)
{
      int auxi ;

      auxi = a;          ──────►  Falta inicializar auxi

      while (auxi>0)
      {
            a = b;
            b = auxi;
            auxi = a % b ;
      }
      return b;          ──────►  El resultado queda en b, no en a.
}
```

Ejercicio 11

Se presentan varias soluciones, algunas más claras que otras.

```
int
maximo ( int a, int b, int c)
{
      int m;
      m = (a > b)? ((a > c)? a : c) : ((b > c)? b : c);
      return m;
}
```

```
int
maximo ( int a, int b, int c)
{
      return (a > b)? ((a > c)? a : c) : ((b > c)? b : c);
}
```

```
int
maximo ( int a, int b, int c)
{
      int m;
      m = ( a > b )? a : b;
      return ( m > c )? m : c;
}
```

Ejercicio 12

Al ejecutar el programa del punto a) se obtiene el valor esperado (-2), al ejecutar el programa de punto b) se obtiene un valor erróneo. Esto se debe a que en ambos casos falta la inclusión, en el primer archivo, del prototipo de la función **neg** por lo que el compilador no conoce el tipo de datos que recibe y retorna la función. Al no conocer los tipos de datos involucrados el compilador siempre asume *por default* que todos los tipos desconocidos son **int**. Cuando se compila el archivo que contiene la función **main**, asume *por default* que la función **neg** recibe un entero y retorna un entero entonces en el caso a) por casualidad asume los tipos correctos y en el caso b) NO.

NUNCA SE DEBE DEJAR LIBRADA LA INTERPRETACION DE TIPOS AL DEFAULT DEL COMPILADOR (Normas Indian Hill).

Ejercicio 13

En el archivo tp4_13a.c no se declararon las funciones **getfloat**, **f1** y **f2**, por lo que el compilador, al igual que en el ejercicio anterior, asume que los tipos de datos involucrados son de tipo entero, lo que provocará un error de ejecución.

Para solucionarlo se debe incluir en el archivo tp4_13a.c el archivo de encabezado "getnum.h" por un lado y por otro definir un archivo "tp4_13.h" con el prototipo de las funciones f1 y f2.

Ejercicio 14

No hay errores de preprocesamiento ni compilación. Sí habrá un error al linkeditar ambos archivos pues la función funAuxiliar está definida en más de una oportunidad, la solución para este error es eliminar una de las dos definiciones. Al igual que en el archivo anterior se produce un error de ejecución tanto al invocar la función getfloat como al invocar la función f2.

Ejercicio 15

Al preprocesar el archivo tp4_15a.c se produce un error pues no existe el archivo tp4_15.h.

Al compilar el archivo tp4_15main.c se producirá un error pues la función fAuxiliar está declarada más de una vez y con distinto prototipo, lo cual provoca un error de "conflicto de tipos".

Al linkeditar los 3 archivos se producirá un error por la múltiple definición de la función fAuxiliar.

Los dos últimos errores están relacionados y derivan del hecho que la función fAuxiliar es una función auxiliar tanto para promedio3 como para mayor3, pero que si bien tienen el mismo nombre son dos funciones distintas. La solución es no incluir el prototipo en el archivo de encabezados y declararla como static, tanto en tp4_15a.c como en tp4_15b.c. Los códigos finales deben ser:

Archivo tp4_15main.c

```
#include "getnum.h"
#include <stdio.h>
#include "tp4_15a.h"
#include "tp4_15b.h"

int
main(void)
{
    int x,y,z;

    x = getint("Ingrese un numero: ");
    y = getint("\nIngrese otro numero: ");
    z = getint("\nIngrese ultimo numero: ");

    printf("\nEl mayor es %d\n", mayor3(x,y,z));
    printf("El promedio es %.2g\n", promedio3(x,y,z));

    return 0;
}
```

Archivo tp4_15a.h

```
/* Biblioteca  para obtener el mayor de 3 numeros */

int mayor3 (int n, int m, int p);
```

Archivo tp4_15b.h

```
/* Biblioteca para obtener el promedio de 3 enteros */

float promedio3 (int n, int m, int p);
```

Archivo tp4_15a.c

```
/* Biblioteca  para obtener el mayor de 3 numeros */

static int fAuxiliar (int m, int n);


int
mayor3 (int n, int m, int p)
{
    return fAuxiliar( fAuxiliar(n,m), p);
}

static int
fAuxiliar (int m, int n)
{
    int resp;

    if ( m > n )
        resp = m;
    else
        resp = n;

    return resp;
}
```

Archivo tp4_15b.c

```
/* Biblioteca para obtener el promedio de 3 enteros */

static int fAuxiliar (int n, int m, int p);

float
promedio3 (int n, int m, int p)
{
    return fAuxiliar(n,m,p) / 3.0;
}

static int
fAuxiliar (int n, int m, int p)
{
    return n + m + p;
}
```