

## TPN°5: Funciones y Biblioteca Estándar

### Ejercicio 1

Se indica para cada variable, persistencia, alcance y visibilidad. Al ser un único código fuente en todos los casos las variables son **definidas**.

a)

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

```
int a, b ;  
void local ( void );
```

estáticas, globales,  
enlace externo

```
int  
main ( void )  
{  
    a = 2;  
    b = 3;  
    local();  
    printf( " a vale : %d\tb vale %d\n",  a, b );  
    return 0;  
}
```

```
void  
local ( void )  
{
```

```
    int a;  
  
    a = -5;  
    b = 10;
```

```
    return ;
```

```
}
```

automática, local,  
sin enlace

La salida del programa es: **a vale : 2   b vale 10**

b)

```
#include <stdio.h>
int a, b, c;

void primero ( void );
void segundo ( void );

int
main ( void )
{
    a = 1;

    printf("El valor de c es %d", c );

    segundo();
    printf("El valor de a es %d, el de b es %d", a, b );

    primero();
    printf("El valor de a es %d, el de c es %d", a, c );

    return 0;
}

void
primero ( void )
{
    int a ;

    a = 3;
    c = 0 ;
}

void
segundo ( void )
{
    int a ;

    c = 0;
    a = 2;
    b = -a;
}
```

estáticas, globales,  
enlace externo

Automática, local,  
sin enlace

Automática, local,  
sin enlace

La salida de el programa es:

**El valor de c es 0 El valor de a es 1, el de b es -2 El valor de a es 1, el de c es 0**

c)

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

```
char letra;
void segundo ( void );
```

estática, global,  
enlace externo

```
int
main( void )
{
    printf("Ingrese un carácter : ");

    letra = getchar();
    segundo();
    printf(" letra es : %c\n", letra );

    return 0;
}
```

```
void
segundo( void )
{
```

```
    char letra;
```

```
    letra = 'X';
    return;
```

```
}
```

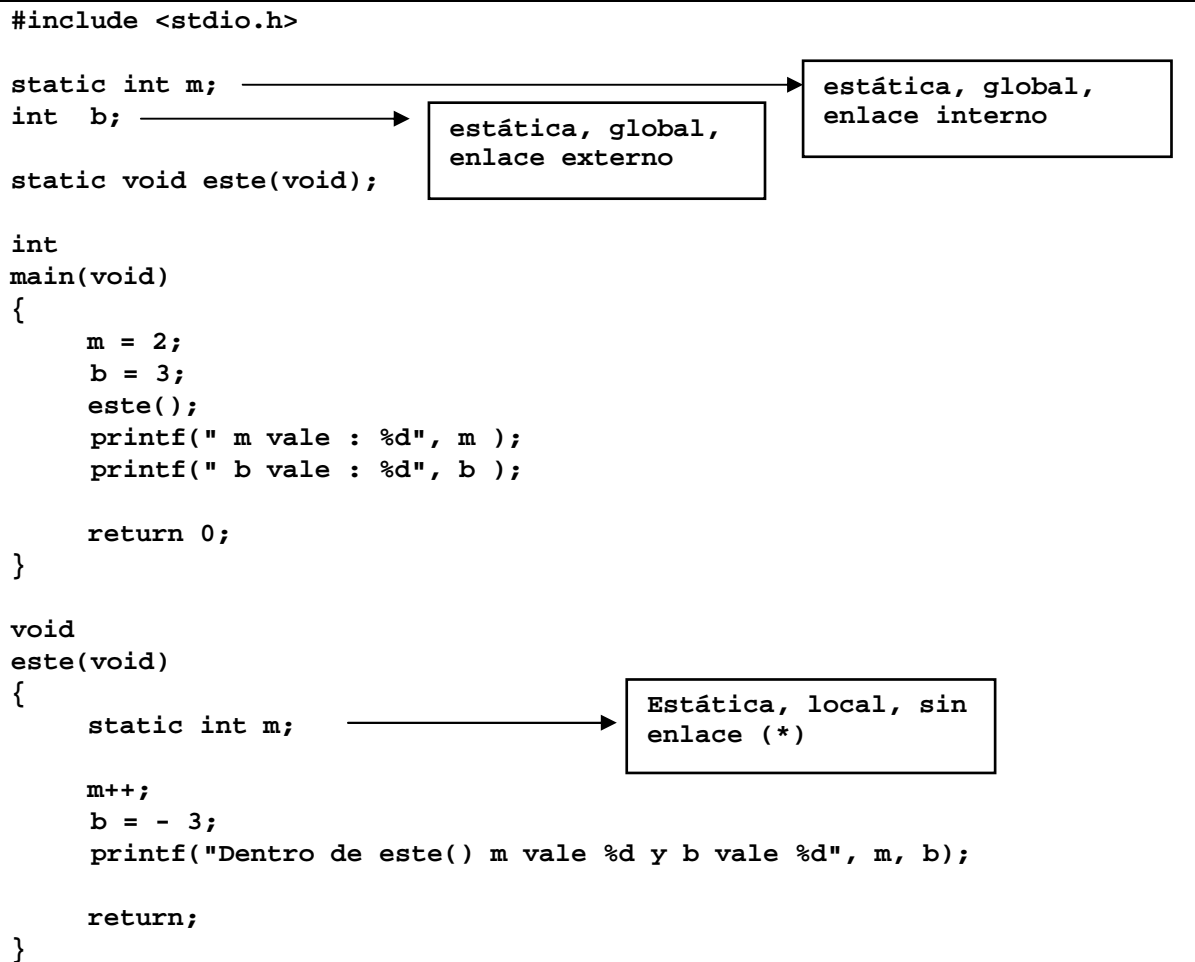
Automática, local,  
sin enlace

La salida del programa es:

**Ingrese un carácter:**

**letra es <valor devuelto por getchar>**

d)

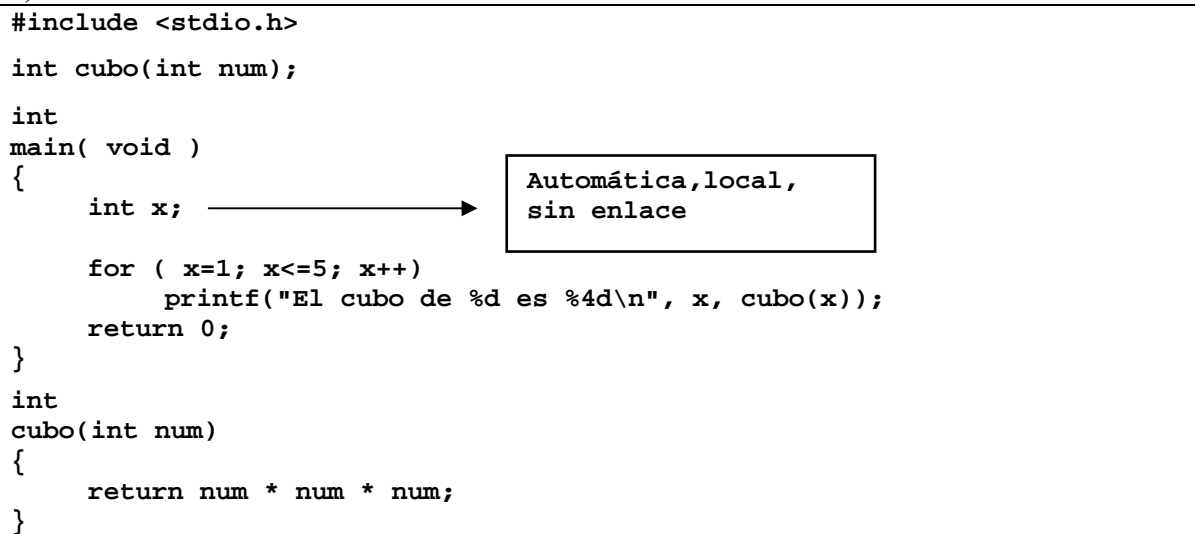


(\*) No todos los compiladores aceptan la repetición de nombres en variables estáticas.

La salida del programa es:

**Dentro de este() m vale 1 y b vale -3 m vale : 2 b vale : -3**

e)



La salida del programa es:

El cubo de 1 es 1  
 El cubo de 2 es 8  
 El cubo de 3 es 27  
 El cubo de 4 es 64  
 El cubo de 5 es 125

## Ejercicio 2

a) La salida que se obtiene es la siguiente

```

46 . .      no no no no no no no si
47 / /      no no no no no no no si
48 0 0      si no si no no si no no
49 1 1      si no si no no si no no
50 2 2      si no si no no si no no
51 3 3      si no si no no si no no
52 4 4      si no si no no si no no
53 5 5      si no si no no si no no
54 6 6      si no si no no si no no
55 7 7      si no si no no si no no
56 8 8      si no si no no si no no
57 9 9      si no si no no si no no
58 : :      no no no no no no no si
59 ; ;      no no no no no no no si
60 < <      no no no no no no no si
61 = =      no no no no no no no si
62 > >      no no no no no no no si
63 ? ?      no no no no no no no si
64 @ @      no no no no no no no si
65 A a      si si no no si si no no
66 B b      si si no no si si no no
67 C c      si si no no si si no no
68 D d      si si no no si si no no
69 E e      si si no no si si no no
70 F f      si si no no si si no no
71 G g      si si no no si no no no
72 H h      si si no no si no no no
  
```

b) La salida que se obtiene es la siguiente

```

-1.000000  1.000000 -1.000000 -1.000000
1.000000   nan      ←
-0.500000  0.500000 -0.000000 -1.000000
0.250000   nan      ←
0.000000  0.000000  0.000000  0.000000
0.000000  0.000000
0.500000  0.500000  1.000000  0.000000
0.250000  0.707107
1.000000  1.000000  1.000000  1.000000
1.000000  1.000000
1.500000  1.500000  2.000000  1.000000
2.250000  1.224745
  
```

El valor de **errno** en  
 estos casos será **EDOM**.

**Ejercicio 3**

a)

```

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

int lugar(int num, int pos);

int
main(void)
{
    int j, boleto, esCapicua=1;
    boleto=getint("Ingrese un número de 5 dígitos:");
    if (boleto >= 10000 && boleto <= 99999) → Línea agregada
    {
        for(j=1; j<=2; j++)
            if (lugar(boleto,j) != lugar(boleto, 5-j+1))
            {
                esCapicua=0;
                break;
            }
        printf("El boleto %ses capicúa\n", (esCapicua) ? "" : "no ");
    }
    return 0;
}

int
lugar(int num, int pos)
{
    return (num / (int)pow(10,pos-1)) % 10;
}

```

b)

```

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

int lugar(int num, int pos);

int
main(void)
{
    int j, boleto, esCapicua=1;
    boleto=getint("Ingrese un número de 5 dígitos:");
    if ( boleto >= 10000 && boleto <= 99999)
    {
        for(j=1; j<=2; j++)
            if (lugar(boleto,j) != lugar(boleto, 5-j+1))
            {
                esCapicua=0;
                break;
            }
        printf("El boleto %ses capicúa\n", (esCapicua) ? "" : "no ");
    }
    else
        printf("Ingreso incorrecto de datos.\n");
    return 0;
}
/* Omitimos el código de la función lugar */

```

c)

```

#include <stdio.h>
#include <math.h>
#include "getnum.h"
int lugar(int num, int pos);
int
main(void)
{
    int j, flag, boleto, esCapicua=1;
    /***** LINEAS AGREGADAS *****/
    do
    {
        boleto=getint("Ingrese un número de 5 dígitos:");
        if (flag = (boleto < 10000 || boleto > 99999))
            printf("Ingreso incorrecto de datos.\n");
    }
    while (flag);
    /*****/
    for(j=1; j<=2; j++)
        if (lugar(boleto,j) != lugar(boleto, 5-j+1))
        {
            esCapicua=0;
            break;
        }
    printf("El boleto %ses capicúa\n", (esCapicua) ? "" : "no ");
    return 0;
}

```

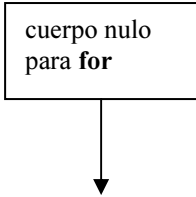
d)

```

#include <stdio.h>
#include <math.h>
#include "getnum.h"
int lugar(int num, int pos);
int
main(void)
{
    int j, cifras, boleto, auxB, esCapicua=1;
    boleto=getint("Ingrese un número:");
    /* Calcular la cantidad de cifras del número */
    for(cifras =0, auxB = boleto; auxB ; cifras++, auxB/=10) ;
    /* Se recorre el número comparando los extremos, la cantidad de comparaciones
    ** corresponden a la mitad de la cantidad de cifras, sea par o impar
    ** Línea anterior: for(j=1; j<=2; j++)
    */
    for(j=1; j<=cifras / 2; j++)
        if (lugar(boleto,j) != lugar(boleto, cifras-j+1))
        {
            esCapicua=0;
            break;
        }
    printf("El boleto %ses capicúa\n", (esCapicua) ? "" : "no ");
    return 0;
}

```

cuerpo nulo  
para for



e)

```

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

int
main(void)
{
    int j, cifras, boleto, auxB, esCapicua=1;
    boleto=getint("Ingrese un número:");
    for(cifras =0, auxB = boleto; auxB; cifras++, auxB/=10) ;
    /*****
    ** Se recorre el número comparando los extremos, la cantidad de comparaciones
    ** corresponden a la mitad de la cantidad de cifras, sea par o impar
    *****/
    for(j=1; j<=cifras / 2; j++)
        if (lugar(boleto,j) != lugar(boleto, cifras-j+1))
        {
            esCapicua=0;
            break;
        }
    return esCapicua; /* Devuelve 1 si es capicúa y 0 si no lo es */
}

```

### Ejercicio 4

```

#include <stdlib.h>

/* Devuelve un número entero aleatorio dentro del intervalo [izq, der] */
int
aleatorio(int izq, int der)
{
    return izq + (der - izq + 1) * (rand() / ((double)RAND_MAX+1));
}

```

### Ejercicio 5

```

#include <math.h>
#define EPSILON 0.00001

double
potencia(double base, int exponente)
{
    double pot;
    double auxBase;

    if ((fabs(base) < EPSILON && exponente <= 0 )
        return -1.0;

    if (exponente < 0)
    {
        auxBase = 1 / base;
        exponente *= -1;
    }
    else
        auxBase = base;

    for(pot = 1; exponente; pot *= auxBase, exponente--);
    return pot;
}

```

**Validar los parámetros**

**En caso de exponente negativo, modifica la base y el exponente para obtener el valor correcto.**



**Ejercicio 6**

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

/* Si el caracter es una letra minúscula devuelve la mayúscula,
** sino devuelve el mismo caracter */
int minToMay(int letra);

/* Si el caracter es una letra mayúscula, devuelve la minúscula
** sino devuelve el mismo caracter */
int mayToMin(int letra);

/* Devuelve la letra siguiente en forma circular. Si no es una letra retorna el
** caracter recibido
*/
int letraSig(int letra);

/* Devuelve el caracter siguiente */
int carSig(int letra);

/* Retorna la opción de menú elegida por el usuario */
int menu(void);

int
main(void)
{
    int c, resp;
    int opcion;

    do
    {
        printf("\nIngrese un caracter (EOF para terminar):");
        c=getchar();

        if (c != EOF)
        {
            opcion = menu();

            switch(opcion)
            {
                case 1: resp = minToMay(c); break;
                case 2: resp = mayToMin(c); break;
                case 3: resp = carSig(c); break;
                case 4: resp = letraSig(c); break;
                default:
                    resp = 0;
                    printf("Opción inválida\n");
            }
            if ( resp != 0 )
                printf ("Respuesta: %c (%3d)\n", resp, resp);
        }
    }
    while (c!= EOF);
    return 0;
}
```

```
int
menu(void)
{
    int opcion;

    printf("\n1 - Convertir a mayúsculas\n");
    printf("2 - Convertir a minúsculas\n");
    printf("3 - Caracter siguiente\n");
    printf("4 - Siguiente letra\n");

    opcion = getint("Ingrese una opcion:");
    return opcion;
}

int
minToMay(int letra)
{
    if (letra >= 'a' && letra <= 'z')
        letra -= 'a' - 'A';
    return letra;
}

int
mayToMin(int letra)
{
    if (letra >= 'A' && letra <= 'Z')
        letra += 'a' - 'A';
    return letra;
}

int
carSig(int letra)
{
    return letra + 1;
}

/* La imposición de 4 líneas nos obliga a esforzarnos y escribir en forma
** no muy clara una función de 3 líneas.
** Un código más claro, de 6 líneas, podría ser:

int
letraSig(int letra)
{
    if ( (letra>='A' && letra<='Z' || letra>='a' && letra <= 'z')
        if ( letra == 'z' || letra == 'Z')
            letra -= ( 'Z' - 'A');
        else
            letra = carSig(letra);
    return letra;
}
*/
int
letraSig(int letra)
{
    if ( ! (letra>='A' && letra<='Z' || letra>='a' && letra <= 'z'))
        return letra;
    return ( letra == 'z' || letra == 'Z') ?
        letra - ( 'Z' - 'A')
        : carSig(letra);
}
```

## Ejercicio 7

```
/* Este programa estima el valor de e a la X por medio de una serie, y luego muestra la
** comparación con el valor calculado usando la función provista por la biblioteca
** matemática. El valor de X es leído desde la entrada estándar
*/

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

#define EPSILON 0.0000001

/* Devuelve la estimación de e a la X con un error de EPSILON,
** recibe el valor de X
*/
double ex(double x);

int
main (void)
{
    double eEstimado, eReal, x;

    x = getfloat("Ingrese el valor de x para calcular exp(x): ");
    if ( x < 0 )
        printf("Solo para positivos\n");
    else
    {
        eEstimado = ex(x);
        eReal = exp(x);
        printf("\nValor calculado: %f\n",eEstimado);
        printf("Valor real: %f\n",eReal);
    }
    return 0;
}

double
ex(double x)
{
    double factorial=1;
    double valor, anterior, potX;
    int termino=1;

    if ( x < 0 )
        return -1;

    potX = x;
    valor = 1;
    anterior = 0;
    while (valor - anterior > EPSILON)
    {
        anterior = valor;
        valor += potX / factorial;
        potX *= x;
        termino++;
        factorial *= termino;
    }

    return valor;
}
```

**Ejercicio 8**

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

/* Devuelve un numero real redondeado a la cantidad de cifras
** decimales pedidas en el segundo parámetro
*/
double redondeo(double numero, int cifras);

int menu(void);

int
main(void)
{
    float num;
    int opcion;

    num=getfloat("\nIngrese un numero real:");
    opcion = menu();
    switch(opcion)
    {
        case 1: num= redondeo(num,0); break;
        case 2: num= redondeo(num,1); break;
        case 3: num= redondeo(num,2); break;
        case 4: num= redondeo(num,3); break;
        default: printf("Opción inválida\n"); break;
    }
    if (opcion >= 1 && opcion <= 4)
        printf("El numero es %g\n", num);
    return 0;
}

int
menu(void)
{
    int opcion;

    printf("\n1 - Redondeo al entero más cercano\n");
    printf("2 - Redondeo a la décima más cercana\n");
    printf("3 - Redondeo a la centésima más cercana\n");
    printf("4 - Redondeo a la milésima más cercana\n");

    opcion = getint("Ingrese una opción:");
    return opcion;
}

double
redondeo(double numero, int cifras)
{
    long factor;

    for(factor = 1; cifras ; factor *= 10, cifras--);
    return floor( numero * factor + 0.5) / factor;
}
```

Presentamos ahora una versión mejorada de la función *main* propuesta arriba.

```
int
main(void)
{
    float num;
    int opcion;

    num=getfloat("\nIngrese un numero real:");

    opcion = menu();
    if ( opcion >=1 && opcion <=4 )
        printf("El numero es %g\n", redondeo(num, opcion -1));
    else
        printf("Opción inválida\n");

    return 0;
}
```

### Ejercicio 9

```
#include <stdio.h>
#include <math.h>
#define INCREMENTO 0.001
#define EPSILON 0.0001

/* Devuelve la imagen de x */
double funcion(double x);

int
main(void)
{
    double izq = -4;
    double der = 4;

    int signoAnt;
    int signoFun;

    /* Se guarda el signo de la función en izq */
    signoAnt = (funcion(izq)>= 0) ? 1 : -1 ;

    for( ; izq <= der; izq += INCREMENTO)
    {
        signoFun = (funcion(izq)>= 0) ? 1 : -1 ;

        /* Se verifica si hay un cambio de signo */
        if (signoAnt != signoFun)
            printf("Raiz en %g\n",izq);
        else
            /* Se verifica si se acerca a cero en menos de EPSILON */
            if (fabs(funcion(izq)) < EPSILON)
                printf("Raiz en %g\n",izq);
        signoAnt = (funcion(izq)>= 0) ? 1 : -1 ;
    }

    return 0;
}

double
funcion(double x)
{
    return sin(x);
}
```

Extremos del intervalo

Cambiar por la función deseada

**Ejercicio 10**

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

/* Macro que pide un denominador distinto de cero */
#define LEER_DENOM_NO_CERO(denom) {do {denom=getint(""); \
    if (denom==0) printf("El denominador no puede ser cero.\n"); }\
    while(denom==0);}

/* Imprime la simplificación de la fracción representada por num y den */
void simplFrac(int num, int den);

/* Imprime la suma de dos fracciones representadas por num1 y den1, y num2 y den2 */
void sumarFrac(int num1, int num2, int den1, int den2);

/* devuelve el divisor común máximo de dos números */
int dcm (int num1, int num2);

int menu(void);

int
main(void)
{
    int opcion;
    int num1, num2, den1, den2;

    do
    {
        opcion=menu();

        switch(opcion)
        {
            case 1:
                num1 = getint("\nIngrese el numerador:");
                printf("\nIngrese el denominador:");
                LEER_DENOM_NO_CERO(den1)
                simplFrac(num1,den1);
                break;
            case 2:
                num1 = getint("\nIngrese el numerador de la 1ra. fracción:");
                printf("\nIngrese el denominador de la 1ra. fracción:");
                LEER_DENOM_NO_CERO(den1)
                num2 = getint("\nIngrese el numerador de la 2da. fracción:");
                printf("\nIngrese el denominador de la 2da. fracción:");
                LEER_DENOM_NO_CERO(den2)
                sumarFrac(num1, num2, den1, den2);
                break;
            case 3: break;
            default: printf("\nOpción invalida"); break;
        }
    }
    while (opcion != 3);

    return 0;
}
```

```
int
menu(void)
{
    int opcion;

    printf("\n1 - Simplificar una fracción");
    printf("\n2 - Sumar dos fracciones");
    printf("\n3 - Terminar");

    opcion = getint("\nElija una opción:");

    return opcion;
}

void
simplFrac(int num, int den)
{
    int numAux, denAux;
    int valor;

    numAux=num;
    denAux=den;

    /* divide el numerador y el denominador por el dcm, hasta que este se haga 1 */
    if (abs(( valor = dcm(num, den))) != 1)
    {
        num /= valor;
        den /= valor;
    }

    printf("\n%d/%d simplificada es %s%d", numAux, denAux,
                                                (num*den >= 0)? "":"-", abs(num));
    if (abs(den)!=1)
        printf("/%d\n",abs(den));
    return;
}

void
sumarFrac(int num1, int num2, int den1, int den2)
{
    int numCom,denCom;
    int valor;

    /* Calcula la suma */
    denCom = den1 * den2;
    numCom = denCom / den1 * num1 + denCom / den2 * num2;

    /* simplifica la fracción de respuesta */
    if (abs(( valor = dcm(numCom, denCom))) != 1)
    {
        numCom /= valor;
        denCom /= valor;
    }

    printf("\n%d/%d + %d/%d = %s%d/%d",num1, den1, num2, den2,
                                                (numCom*denCom < 0)? "-":"",abs(numCom), abs(denCom));
    return;
}
```

```
int
dcm ( int num1, int num2)
{
    int auxi ;

    auxi = num1;
    while (auxi!=0)
    {
        num1 = num2;
        num2 = auxi;
        auxi = num1 % num2 ;
    }

    return num2;
}
```

Una de las críticas que se puede hacer al código anterior, es que sumarFrac repite el código de simplFrac, ya que también simplifica la suma de las fracciones. Podría cambiarse el código de sumaFrac como se muestra debajo, pero en este caso la impresión de los carteles no sería la correcta.

```
void
sumarFrac(int num1, int num2, int den1, int den2)
{
    int numCom,denCom;
    int valor;

    /* Calcula la suma */
    denCom = den1 * den2;
    numCom = denCom / den1 * num1 + denCom / den2 * num2;

    simplFrac(numCom, denCom);

    return;
}
```

La solución ideal sería que ni simplFrac ni sumarFrac impriman el resultado sino que lo devuelvan, pero aún no vimos cómo poder retornar dos valores. En la práctica 6 veremos cómo hacer para separar el front-end del back-end para estas funciones.



**Ejercicio 11**

```
int a, b, c;
a = getchar();
b = getchar();
c = getchar();

if ( isupper(a) && a!='A' && a!='E' && a!='I' && a!='O' && a!='U' )
    printf("El caracter %c es una consonante mayúscula \n", a);
else if ( islower(a) && a!='a' && a!='e' && a!='i' && a!='o' && a!='u' )
    printf("El caracter %c es una consonante minúscula \n", a);
else if ( isalpha(a) )
    printf("El caracter %c es una vocal \n", a);
else
    printf("El caracter %c no es una letra \n", a);

if ( b % 2 == 0 )
    printf("El valor ASCII de %c es par \n", b);

if ( isdigit(b) && (b-'0')%2 != 0 )
    printf("El caracter %c representa un dígito impar \n", b);

if ( isupper(c) || c=='a' || c=='e' || c=='i' || c=='o' || c=='u' )
    printf("El caracter %c es una vocal o es mayúscula \n", c);

if ( isalpha(c) && toupper(c)!='E' && toupper(c)!='I' &&
    toupper(c)!='O' && toupper(c)!='U' )
    printf("El caracter %c no es una vocal pero es letra \n", c);
```

### Ejercicio 12

```
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 );
/* while(!(n > 0 && n % 2 == 0)); otra forma de hacerlo */

/* este ciclo debe incluir la lectura de caracteres y se debe ejecutar mientras no se
 * llegue a fin de archivo y los caracteres sean dígitos o minúsculas */
while ((c=getchar())!=EOF && (isdigit(c) || islower(c)))
    putchar(c);
```

### Ejercicio 13

- Sirve para que el compilador, al momento de compilar **main.c**, conozca los tipos de datos involucrados en las funciones de **func.c** (que son invocadas desde **main.c**).
- Serviría para incluir en **main.c** todo el código de **func.c** y así evitar la compilación separada de ambos módulos. **ESTA ES UNA PRACTICA INACEPTABLE** (No cumple con las Normas Indian Hill).
- La secuencia de comandos sería
  - gcc -c main.c
  - gcc -c func.c
  - gcc -opgm main.o func.o

o bien

- gcc -opgm main.c func.c

### Ejercicio 14

La función recibe:

- ip**: unsigned long con el número IP de un host
- bitsNet**: unsigned char que indica cuántos bits se utilizan para identificar a la red

Debe obtenerse primero la mascara de red correspondiente a la cantidad de bits indicada por bitsNet. Se puede hacer directamente a partir de **bitsNet**, haciendo un corrimiento de **bitsNet** bits y luego completar con 0 hasta llegar a 32 bits. O bien calcular los bits de Host y complementar:

Ejemplo:

**bitsNet** = 23 → **bitsHost** = 9 → **mask** = 11111111 11111111 11111110 00000000

Una vez obtenida la máscara de red, el cálculo de la dirección de red y de la dirección de host se obtienen de la manera conocida:

**ip** → 00010000 11111111 00010001 00010010 (10 FF 11 12)  
**mask** → 11111111 11111111 11111110 00000000  
**ip & mask = dir red** → 00010000 11111111 00010000 00000000  
**ip & ~mask = dir host** → 00000000 00000000 00000001 00010010

Por último, para mostrar las direcciones en la forma habitual de octetos, se debe subdividir el unsigned long en 4 bytes u octetos.

```

#include <stdio.h>
#include <assert.h>

unsigned long netMask(unsigned char bitsNet);
void ipWithFormat(unsigned long ip);
void printip(unsigned long ip, unsigned char bitsNet);

int
main(void) {
    unsigned long ip = 0x10FF1112;

    printip(ip, 16);
    printip(ip, 24);
    printip(ip, 23);
    printip(ip, 25);

    return 0;
}

void
printip(unsigned long ip, unsigned char bitsNet)
{
    unsigned long mask;
    unsigned long net;
    unsigned long host;

    /* Obtención de mascara de red */
    mask = netMask(bitsNet);
    /* Calculo de direcciones de red y host */
    net = ip & mask;
    host = ip & ~mask;

    /* Impresión en formato apropiado */
    printf("Red = ");
    ipWithFormat(net);
    printf("\nHost = ");
    ipWithFormat(host);
    putchar('\n');
}

unsigned long
netMask(unsigned char bitsNet)
{
    unsigned long aux = 1;
    unsigned char bitsHost = 32 - bitsNet;

    assert(bitsNet != 0);

    while ( --bitsHost > 0)
        aux = aux << 1 | 1; /* Todos 1 en la porcion de host */
    return ~aux; /* Se complementa */
}

void
ipWithFormat(unsigned long ip){
    /* Obtener los 4 octetos o bytes */

```

```
unsigned char b1, b2, b3, b4;
unsigned long mask = 0xFFFFFFFF;

b1 = (ip & mask) >> 24;
mask >>=8;
b2 = (ip & mask) >> 16;
mask >>=8;
b3 = (ip & mask) >> 8;
mask >>=8;
b4 = (ip & mask);
printf("%d.%d.%d.%d",b1,b2,b3,b4);
}
```