Nombre:

1.- (2 ptos) Sea el siguiente programa en C++:

```
    #include <iostream.h>
    #include <stdlib.h>

    int const BASE = 5;

    int main (void);
    int Examen (int);

    int main(void)
    {
       int x, y;

(1)    x = 10;

(2)    y = Examen (x);

       cout << "y = " << y << endl;

          system("PAUSE");
          return 0;
    }
```

```
    int Examen (int y)
    {
       int x;

       if (y == 0)
(3)       x = 0;
       else
       {
(4)       x = Examen (y / BASE);
(5)       x = x * 10 + y % BASE;
       }
       return x;
    }
```

Realiza la traza del programa y di cuál será el valor final de y.

	x	y						
1	**10**	**?**	**y**	**x**				
2	**10**	**?**	**10**	**?**	**y**	**x**		
4₁	**10**	**?**	**10**	**?**	**2**	**?**	**y**	**x**
4₂	**10**	**?**	**10**	**?**	**2**	**?**	**0**	**?**
3	**10**	**?**	**10**	**?**	**2**	**?**	**0**	**0**
4₂′	**10**	**?**	**10**	**?**	**2**	**0**		
5₁	**10**	**?**	**10**	**?**	**2**	**2**		
4₁′	**10**	**?**	**10**	**2**				
5₂	**10**	**?**	**10**	**20**				
2′	**10**	**20**						

2.- (1.5 pto) Sea la siguiente declaración de tipos

```
typedef int Tipo;
typedef string Vector[10];
struct Reg {
         Vector nombre;
         Tipo dato;
};
typedef Reg * PReg;
```

Si x es una variable de tipo PReg, y sabemos que al inicio del programa se ha ejecutado la instrucción: `x = new Reg[10]`, indicar el tipo resultante de las siguientes expresiones de acceso o si son incorrectas y por qué:

`x[1]` **REG**

`*x.nombre` **MAL. x es un vector y se accede a través de un índice o es un puntero y se accede a través de ->**

`x[2].tipo[2]` **MAL. tipo no es un miembro de la estrucutra**

`x->dato` **Tipo / int**

`x[0].nombre[3]` **string**

`x[3].nombre[2][2]` **char**

Nombre:

3.- (1 pto) Evalúa la siguiente expresión paso a paso:

```
(3 - int ( (12 < 4 * 3 - 1) || !(12 > 4 * 3 + 1) && (12 > 8 / 10 * 100) ) )
```

= (3 – int ((12 < 12 – 1) || !(12 > 12 + 1) && (12 > 0 * 100))) =
= (3 – int ((12 < 11) || !(12 > 13) && (12 > 0))) =
= (3 – int (FALSE || !FALSE && TRUE)) =
= (3 – int (FALSE || TRUE && TRUE)) =
= (3 – int (FALSE || TRUE)) =
= (3 – int (TRUE)) =
= (3 – 1) =
= 2

4.- (1 pto) Dado el siguiente programa:

```
#include <iostream.h>
#include <stdlib.h>

int main (void);
int Llamar (int, int &);

int main(void)
{
   int x, y;

   x = 3;
   y = 5;

   x = Llamar (x, y);
   y = Llamar (y, x);

   cout << "x = " << x << endl;
   cout << "y = " << y << endl;
}
```

```
int Llamar (int x, int & y)
{
   y = x + y;
   x = y + 3;

   return x;
}
```

Di qué mostrará por pantalla:

Mostrará: **x = 19**
y = 22

5.- (1.5 pto) Escribe el bucle para leer un fichero de texto carácter a carácter mediante la función get y mostrarlo por pantalla. Se deben escribir DOS versiones del bucle: una con while y otra con do ... while y hay que tener en cuento que el fichero puede estar vacío

```
int main(void)
{
    ifstream f;
    char c;

    f.open("f.txt");
    // Aquí va el bucle
    f.close();
}
```

Versión con **while**:

```
    c = f.get ();
    while (!f.eof () )
    {
        cout << c;
        c = f.get ();
    }
```

Versión con **do ... while**:

```
    do
    {
        c = f.get ();
        if (!f.eof () )
            cout << c;
    }
    while (!f.eof () );
```

Nombre:

6.- (1 pto) Dada la siguiente definición de tipos:

```
const int MAX = 10;
const int DIM = 4;

typedef char Tab[DIM][DIM];
struct Tablero
{
      int tam;
      Tab t;
}

typedef Tablero Partida[MAX];
Partida x;
```

Y sabiendo que el tamaño de un char es 1 byte, el tamaño de un entero son 4 bytes y que la variable dato comienza en la posición de memoria 1000, calcular en que posición de memoria está el carácter `x[2].t[1][3]`. Escribir el cálculo completo y no sólo el resultado.

p = p0 + 2 * sizeof (Tablero) + 1 * sizeof (int) + 1 * sizeof (char[DIM]) + 3 * sizeof (char) =

= 1000 + 2 *20 + 1 * 4 + 1 * 4 + 3 * 1 =

= 1000 + 40 + 4 + 4 + 3 = 1051

sizeof (Tablero) = sizeof (int) + sizeof(Tab) = 4 + 16 = 20

sizeof (Tab) = sizeof (char[DIM][DIM]) = sizeof (char[4][4]) = sizeof (char) * 4 * 4 = 1 * 4 * 4 = 16

sizeof (char[DIM]) = sizeof (char) * 4 = 1 * 4 = 4

7.-(1 pto) Dada la siguiente gramática:

```
<NN> := + <N> | - <N> | <N>
<N> := <DD> | <DD>.<D>
<DD> := <D><DD> | <D>
<D> := 0 | 1 | 2 | 3 | 4 | 5
```

Di qué frases son correctas y cuales son incorrectas según esta gramática:

	V / F
a) `-23.5`	V
b) `+0.29`	F
c) `-(+0.3)`	F
d) `-.3`	F
e) `-23.54`	F

Nota: Cada apartado correcto puntúa 0.2. Cada apartado incorrecto resta –0.1.

8.-(1 pto) Explica las diferencias entre lenguaje ensamblador y lenguaje máquina.

Mientras que las instrucciones en lenguaje máquina son directamente reconocidas por los circuitos del procesador (están escritas con '0' y '1'), las instrucciones en ensamblador son mnemotécnicos que deben ser traducidos por un traductor. Además en lenguaje máquina los datos se refieren por su dirección de memoria mientras que en ensamblador se pueden utilizar etiquetas para referir los datos.