Clase 2. Funciones y arreglos.

Funciones

Funciones

Una función es un bloque de código que realiza una tarea. Usualmente las funciones:

- Se declaran. Para que el compilador sepa que existen.
- Se definen. Se especifica la tarea que realiza la función, (se códifica el cuerpo de la función).
- Se llama. Se ejecuta el bloque de código de la función desde otra función (cualquiera, o al menos desde la función main).

Todo programa en C tiene al menos una función (la función main).

Declaración y Definción de una función

- La declaración de una función le indica al compilador que la función existe (nombre), el tipo de dato que regresa, y los argumentos que recibe.

Declaración y Definción de una función

- La declaración de una función le indica al compilador que la función existe (nombre), el tipo de dato que regresa, y los argumentos que recibe.
- Una vez declarada la función esta puede ser llamada dentro de otra función.

Declaración y Defincíon de una función

- La declaración de una función le indica al compilador que la función existe (nombre), el tipo de dato que regresa, y los argumentos que recibe.
- Una vez declarada la función esta puede ser llamada dentro de otra función.
- La forma genérica de definir una función en C es la siguiente:
  tipo_que_regresa nombre(lista, de, parámetros).

Declaración y Definción de una función

- La declaración de una función le indica al compilador que la función existe (nombre), el tipo de dato que regresa, y los argumentos que recibe.
- Una vez declarada la función esta puede ser llamada dentro de otra función.
- La forma genérica de definir una función en C es la siguiente:
  tipo_que_regresa nombre(lista, de, parámetros).
- La definición de la función es requerida por el compilador en tiempo de ligado (linking).

la defindción de la función especifica (en código) lo que hace la función y consiste de dos partes:
La cabecera de la función (que debe coincidir con al declaración).
Y el cuerpo de la función.

Partes de una función


```
1  #include <stdio.h>
   /*Declaracion*/
3   int suma5(int);

5  int main()
   {
7        int x=4;
   /*Llamada*/
9      suma5(x);
     return 0;
11 }
   /*Definicion*/
13 int suma5(int a)
   {
15   return a+5;
   }
17
```

En este código se declara la función **suma5**:

- **Tipo de retorno**. La declaración y def. indica que regresa un entero. Este valor de retorno se escribe inmediatamente después de la palabra reservada **return**.

Partes de una función


```
#include <stdio.h>
/*Declaracion*/
 int suma5(int);

int main()
{
      int x=4;
/*Llamada*/
    suma5(x);
  return 0;
}
/*Definicion*/
int suma5(int a)
{
  return a+5;
}
```

En este código se declara la función **suma5**:

- **Tipo de retorno**. La declaración y def. indica que regresa un entero. Este valor de retorno se escribe inmediatamente después de la palabra reservada **return**.
- **Nombre de la función**. La declaración/def. indica que la función se llama **suma5**.

Partes de una función


```
#include <stdio.h>
/*Declaracion*/
 int suma5(int);

int main()
{
      int x=4;
/*Llamada*/
    suma5(x);
  return 0;
}
/*Definicion*/
int suma5(int a)
{
  return a+5;
}
```

En este código se declara la función **suma5**:

- **Tipo de retorno**. La declaración y def. indica que regresa un entero. Este valor de retorno se escribe inmediatamente después de la palabra reservada **return**.
- **Nombre de la función**. La declaración/def. indica que la función se llama **suma5**.
- **Parámetros** La declaración y definición indican que recibe un entero, aunque solo la defínción especifica el nombre del parámetro.

Partes de una función


```
#include <stdio.h>
/*Declaracion*/
 int suma5(int);

int main()
{
      int x=4;
/*Llamada*/
    suma5(x);
  return 0;
}
/*Definicion*/
int suma5(int a)
{
  return a+5;
}
```

En este código se declara la función **suma5**:

- **Tipo de retorno**. La declaración y def. indica que regresa un entero. Este valor de retorno se escribe inmediatamente después de la palabra reservada **return**.
- **Nombre de la función**. La declaración/def. indica que la función se llama **suma5**.
- **Parámetros** La declaración y definición indican que recibe un entero, aunque solo la defínción especifica el nombre del parámetro.
- **Cuerpo de la función**. El cuerpo de la función solo está en la definición y se encuentra acotado por llaves inmediatamente después de la cabecera.

Definición y declaración de una función

Declaración y definición separada

```
#include <stdio.h>
/*Declaracion de funcion*/
int suma5(int);

int main()
{
    int x=4;
/*Llamada*/
   suma5(x);
  return 0;
}
/*Definicion de funcion*/
int suma5(int a)
{
  return a+5;
}
```

La función se declara y define al mismo tiempo.

```
#include <stdio.h>
/*Declaracion de funcion y
Definicion de funcion*/
int suma5(int a)
{
  return a+5;
}

int main()
{
    int x=4;
/*Llamada*/
   suma5(x);
  return 0;
}
```

Argumentos o parámetros de la función

Una función en C recibe cualquier número de argumentos y devuelve solo UN valor. Ejemplo de una función que recibe 3 argumentos de flotantes de doble precisión, 2 argumentos enteros,y 1 argumento caracter, y regresa un valor entero

```
1  int funcion( double x, double y, double z, int a, int
       b, char c)
   {
3      x=3;
   return 0;
5  }

7  int main()
   {
9    int res;
     res=funcion(res,2.3,3.1,8,-9,'x');
11 return 0;
   }
13
```

Paso de argumentos por copia

Al momento de llamar la función el valor de *res*, se copia al de *x* dentro de la función, al igual que los demás valores se copian al argumento correspondiente en la función. Esto quiere decir que *res* se lee y copia en la llamada, aunque *x* se modifica en la función, la memoria de res nunca se toca dentro de la función para escritura. Se escribe en *res* hasta que la función devuelve 0.

Por el momento consideraremos solo el pase de argumentos por copia.

```
int funcion( double x, double y, double z, int a, int
    b, char c)
{
    x=3;
return 0;
}

int main()
{
  int res;
  res=funcion(res,2.3,3.1,8,-9,'x');
return 0;
}
```

Archivos cabecera propios

misfunciones.h

```
#ifndef MISFUNCIONES_H
#define MISFUNCIONES_H
  int funcion1(int n);
#endif
```

misfunciones.c

```
#include "misfunciones.h"
  int funcion1(int n)
 {
    return 10+n;
 }
```

main.c

```
#include "misfunciones.h"
/*Si no pusiera la
    compilacion
    condicional esta
    segunda inclusion me
    daria un error*/
#include "misfunciones.h"

int main()
{
  funcion1(10);
return 0;
}
```

Archivos de cabecera propios y compilación condicional

Suponga que declara en un archivo con extensión .h sus propias funciones. Ej. En el archivo misfunciones.h

```
1 #ifndef MISFUNCIONES_H
  #define MISFUNCIONES_H
3    int funcion1(int n);
  #endif
```

Las directivas condicionales de precompilador (para compilación condicional):

- # ifndef. Si no está definida la macro MISFUNCIONES_H
- # define. Define MISFUNCIONES_H como todo el código siguiente.
- # endif Termina el condicional de compilación.

Esto impide que la función sea declarada tantas veces como se incluye el archivo.

Compilación con varios archivos de declaración y definición de funciones

Supongamos que definimos 3 archivos cabecera con funciones que realizan tareas de diferente tipo:

funciones1.h, funciones2.h y funciones3.h

Estos archivos contienen *declaraciones de las funciones*, y sus correspondientes definiciones se encuentran en:

funciones1.c, funciones2.c y funciones3.c
La forma de compilar (desde consola de windows/Linux):
gcc -c funciones1.c
gcc -c funciones2.c
gcc -c funciones3.c
gcc -c main.c
gcc -o ejecutable main.o funciones3.o funciones1.o funciones2.o

Comentarios sobre funciones

- Al declarar la función no es necesario poner el nombre de los parámetros solo el tipo. Y este debe de coincidir en la definición donde si es necesario dar nombre a los parámetros, o marcará un error **type mismatch**.

Comentarios sobre funciones

- Al declarar la función no es necesario poner el nombre de los parámetros solo el tipo. Y este debe de coincidir en la definición donde si es necesario dar nombre a los parámetros, o marcará un error **type mismatch**.
- Como se vió, se pueden compilar funciones sin que se usen y se pueden mandar llamar sin estar aun definidas (solo declaradas), en momento de compilación. La definición y declaración debe estar completa al momento del ligado (linking).

Comentarios sobre funciones

- Al declarar la función no es necesario poner el nombre de los parámetros solo el tipo. Y este debe de coincidir en la definición donde si es necesario dar nombre a los parámetros, o marcará un error **type mismatch**.
- Como se vió, se pueden compilar funciones sin que se usen y se pueden mandar llamar sin estar aun definidas (solo declaradas), en momento de compilación. La definición y declaración debe estar completa al momento del ligado (linking).
- Para que el programa funcione no es necesario: ni poner las funciones en archivos .h y .c por separado, ni compilar primero los .c a .o (con gcc -c), pero es muy deseable que se haga así parar reutilizanción de código, y facilidad para la programación y entendimiento del programa, además de acelerar la velocidad de compilación.

Comentarios sobre funciones

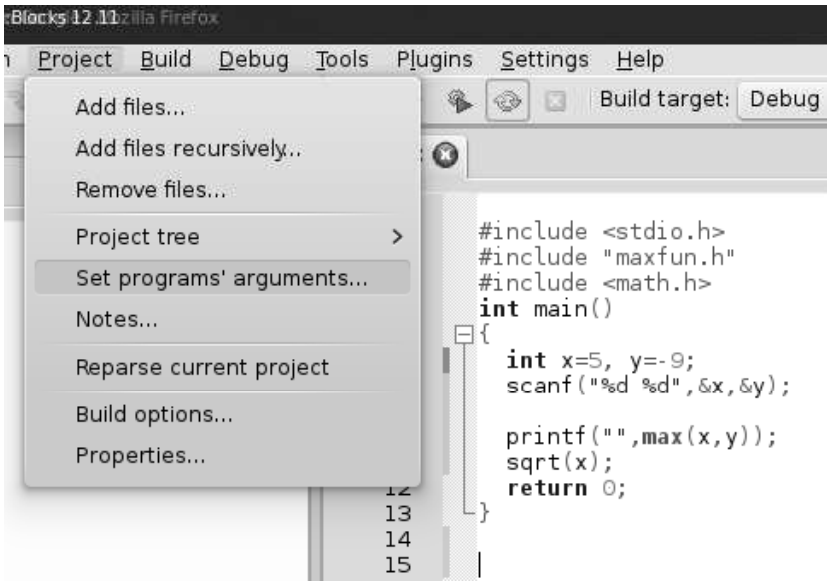
- Al declarar la función no es necesario poner el nombre de los parámetros solo el tipo. Y este debe de coincidir en la definición donde si es necesario dar nombre a los parámetros, o marcará un error **type mismatch**.
- Como se vió, se pueden compilar funciones sin que se usen y se pueden mandar llamar sin estar aun definidas (solo declaradas), en momento de compilación. La definición y declaración debe estar completa al momento del ligado (linking).
- Para que el programa funcione no es necesario: ni poner las funciones en archivos .h y .c por separado, ni compilar primero los .c a .o (con gcc -c), pero es muy deseable que se haga así parar reutilizanción de código, y facilidad para la programación y entendimiento del programa, además de acelerar la velocidad de compilación.
- Desde un IDE, usualmente, solo se necesita agregar al proyecto los archivos .c y .h para que la compilación y ligado lo haga de forma automática el IDE.

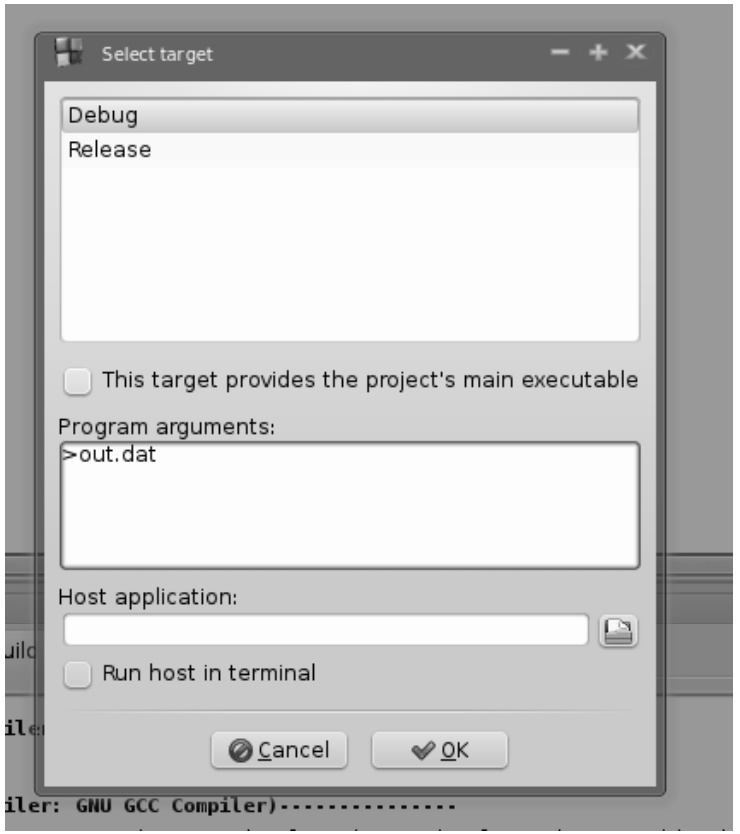
¿ Qué deberiamos saber hasta aquí?

- Declarar, definir y llamar funciones que reciben diferente cantidad de parámetros y regresan un valor.
- Usar archivos .h y .c.
- Usar los tipos de dato básicos.
- Compilar y ejecutar.

Redireccionar la salida desde codeblocks

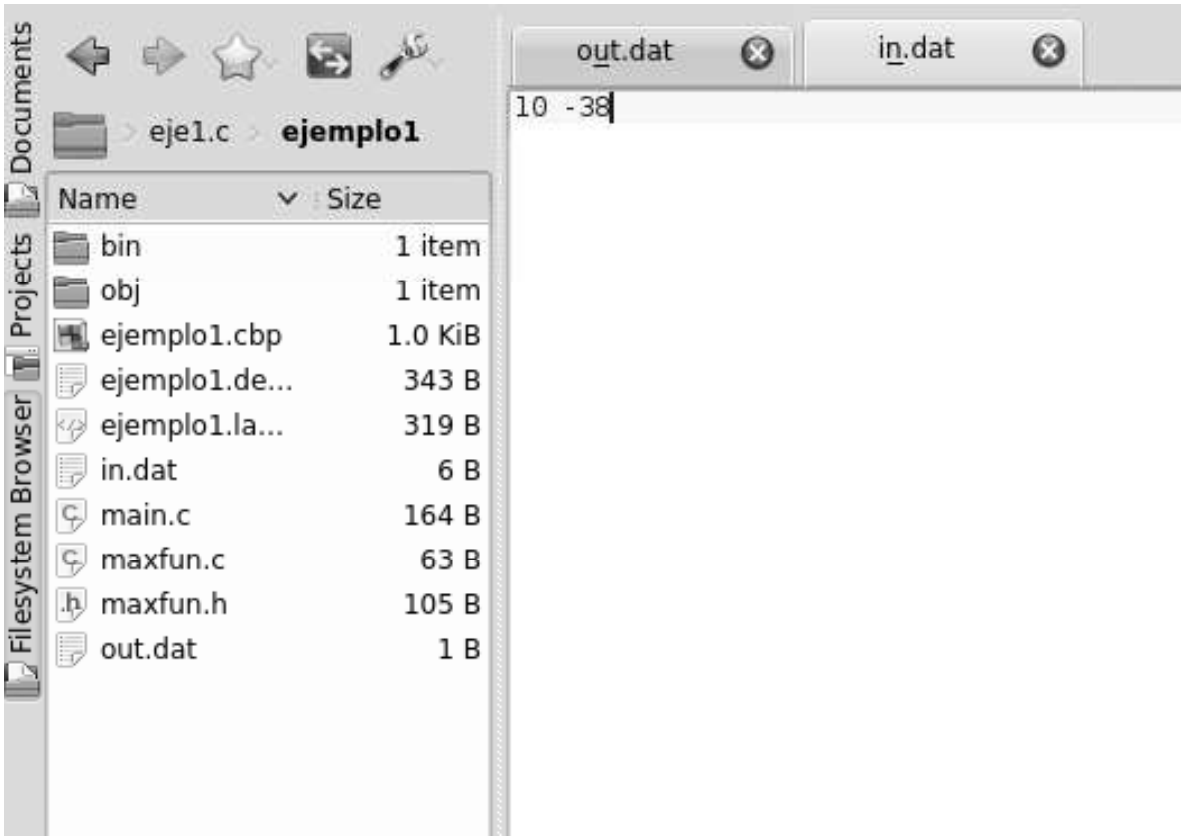
Poner como argumento del programa la redirección y puedo ver donde guarda el archivo de salida.





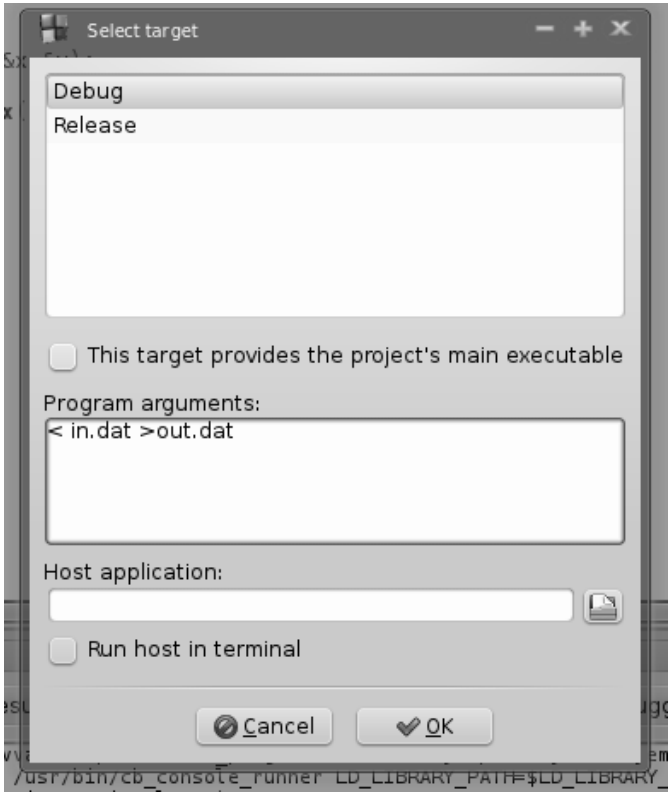
Redireccionar entrada desde codeblocks

En la ruta donde guardó el archivo de salida, creo un archivo de entrada con los datos que lee mi programa.



Redireccionar entrada y salida desde codeblocks

Finalmente, redirecciono la entrada y salida (o alguna de ellas, normalmente la entrada) y ya no necesito estar tecleando cada que ejecuto mi programa.



Ciclos y condicionales

Condicion

En C se utilizan los siguientes operadores relacionales para las condiciones:

Operador	Nombre	Ejemplo para ser verdadero
$>$	Mayor que	$5 > 4$
$<$	Menor que	$4 < 5$
$>=$	Mayor o igual que	$5 >= 4$, $5 >= 5$
$<=$	Menor o igual que que	$4 <= 5$, $4 <= 4$
$==$	Igual	$5 == 5$, $4 == 4$
$!=$	Diferente	$5! = 4$, $4! = 5$

If, else


```
if(5>3)
  printf("5 esmayor que 3\n");

if (5<3){
    printf("Nunca entrara aqui\n");
}else{
   printf("5 no es menor que 3");
}
```

Ciclo For

Por el momento solo veremos dos ciclos en sus formas sencillas para que los puedan utilizar inmediatamente.
El ciclo for es de la forma siguiente:

for (inicialización de la variable; condición de paro; actualización de la variable)
{
código cíclico;
}

Ciclo for


```
int i;
for (i=0; i<10; i++)
{
    printf("i=%d\n",i);
}
```

En C anterior a 99, las declaración/definición de la variable solo se permite al inicio de la función, antes de cualquier operación. En el ciclo for:

- Se inicializa la variable i.
- Se escribe la condición de que el ciclo parará cuando i llegue a diez.
- Se escribe el incremento en i usando el operador ++.
- En caso de ser un conjunto de enunciados los que se repetiran se abre llave (para una sola instrucción no es necesario).

Ciclo for


```
1 int i;
  for (i=0; i<10; i++)
3 {
      printf("i=%d\n",i);
5 }
```

Se ejecuta el ciclo en el siguiente orden:

1. primera iteración se inicializa i=0.
2. Se verifica la condición.
3. Se ejecuta el código dentro de las llaves.
4. se actualiza el valor de las variables.
5. Vuelve a 2

Ciclo for

```
1  for (i=0; i<10; i++)
       printf("i=%d\n",i);
```


Note que: en los parámetros del for, no es necesario que sean exactamente los que se ejemplifican aunque son los más comunes:

- La primera expresión(es) son inicializaciones de variables, o cualesquiera expresiones que solo se ejecutarán UNA vez, antes de comenzar el ciclo.

Ciclo for

```
for (i=0; i<10; i++)
2       printf("i=%d\n",i);
```


Note que: en los parámetros del for, no es necesario que sean exactamente los que se ejemplifican aunque son los más comunes:

- La primera expresión(es) son inicializaciones de variables, o cualesquiera expresiones que solo se ejecutarán UNA vez, antes de comenzar el ciclo.
- Despues del primer punto y coma, la siguiente expresión(es) son condiciones, que deben de regresar un valor lógico (verdadero o falso, o cero y diferente de cero). Si es verdadero el ciclo continua, si es falso el ciclo termina.

Ciclo for

```
for (i=0; i<10; i++)
2       printf("i=%d\n",i);
```


Note que: en los parámetros del for, no es necesario que sean exactamente los que se ejemplifican aunque son los más comunes:

- La primera expresión(es) son inicializaciones de variables, o cualesquiera expresiones que solo se ejecutarán UNA vez, antes de comenzar el ciclo.
- Despues del primer punto y coma, la siguiente expresión(es) son condiciones, que deben de regresar un valor lógico (verdadero o falso, o cero y diferente de cero). Si es verdadero el ciclo continua, si es falso el ciclo termina.
- Después del tercer punto y coma, las expresiones se ejecutan después del cuerpo del ciclo.

Ciclo for

```
for (i=0; i<10; i++)
2       printf("i=%d\n",i);
```


Note que: en los parámetros del for, no es necesario que sean exactamente los que se ejemplifican aunque son los más comunes:

- La primera expresión(es) son inicializaciones de variables, o cualesquiera expresiones que solo se ejecutarán UNA vez, antes de comenzar el ciclo.
- Despues del primer punto y coma, la siguiente expresión(es) son condiciones, que deben de regresar un valor lógico (verdadero o falso, o cero y diferente de cero). Si es verdadero el ciclo continua, si es falso el ciclo termina.
- Después del tercer punto y coma, las expresiones se ejecutan después del cuerpo del ciclo.
- En cada sección de enunciados (entre los puntos y comas), el conjunto de expresiones o instrucciones deben de ir separadas por comas.

Ciclo while

El ciclo while es mas simple:

```
while (condicion)
{
  codigo_que_se_repite;
}
```

- Mientras la condición sea verdadera el ciclo se repite.
- A fin de no hacer ciclos infinitos la variable o variables sujetas a condición debe de modificarse dentro del ciclo.

Ciclo while


```
int i=0;
while(i<10)
{
   printf("valor de i=%d\n",i);
   i++;
}
```

En el argumento del ciclo while, también se pueden escribir un conjunto de expresiones o llamdas a función, etc. Que se ejecutarán al comienzo del ciclo. Solo la última expresión se toma como condición.

Comentarios sobre ciclos y condicionales

- En los argumentos de los ciclos y condicionales se puede poner más de una expresión o llamada a función separadas por una coma. Ej.:
  if(x=y+1,x¡10)
  x=y+1 es una expresión que se evalua antes de la condición, la condición siempre es la última expresión.

Comentarios sobre ciclos y condicionales

- En los argumentos de los ciclos y condicionales se puede poner más de una expresión o llamada a función separadas por una coma. Ej.:
  if(x=y+1,x¡10)
  x=y+1 es una expresión que se evalua antes de la condición, la condición siempre es la última expresión.
- Si después del paréntesis del ciclo o condicional se pone un punto y coma, quiere decir que ahí se está dando por terminada la instrucción, ej“
  if(x==5);
  Este if en realidad no hace nada porque tiene un punto y coma que dice que ahí termina la ejecución condicional.

Resumen

Hasta aquí deberiamos de poder programar:

- Usar condiciones If y else, con expresiones compuestas (usando && y ||).
- Usar ciclos for con variables que se incrementan y usan como condición de paro.
- Usar ciclos while con condiciones compuestas.

Tarea 1

prog1.7 Realice un programa que a través de impresiones con *printf* ejemplifique el orden en que se evaluan las expresiones en los argumentos y cuerpo del ciclo for.

prog1.8 Escriba un programa que en una función (en su archivo .h y .c), lea un número n entero y después n números flotantes (sin utilizar arreglos), y calcule el promedio de los últimos, use el ciclo for.

prog1.9 Escriba un programa que en una función (en su archivo .h y .c), lea números flotantes positivos y calcule el promedio, la lectura y cálculo se detendrá cuando se inserte un número negativo, use el ciclo while.

prog1.10 Escriba un programa que en una función (en su archivo .h y .c), lea letras como caracteres, el programa se detendrá cuando se inserte un caracter que no sea una letra (use if y else para determinar si es una letra).

prog1.11 En una transferencia de archivos de tamaño similar, se intenta predecir: a) el tiempo de transferencia del siguiente archivo, b) el tiempo total de transferencia (predicho cada que se copia un nuevo archivo) de todos los archivos, c) El tiempo promedio de transferencia, y d) la mediana del tiempo de transferencia. Por redirección de la entrada estándar, lea los datos del archivo que se envió al correo. E intente predecir lo que se solicita. El error de cada inciso, se calculará como la suma de las diferencias absolutas del tiempo predicho contra el tiempo real, desde el archivo 2 hasta el 100000. Para el inciso a) se puede accumular cada que se lea un nuevo dato. Para el inciso b) el tiempo total real lo consideraremos conocido de 1166223). Para el inciso C y D, la media y mediana real son de 11.66223 y 10.519776 respectivamente. La forma de predecir los valores es libre con las restricciones de no usar arreglos y utilizar solo lo visto hasta ahora en clase, utilizando un máximo 10 variables flotantes y 5 enteras.

Referencias

Loops
http://www.cprogramming.com/tutorial/c/lesson3.html
Loops
http://www.tutorialspoint.com/cprogramming/c_loops.htm
http://www.cprogramming.com/tutorial/c/lesson3.html

Temas futuros no cubiertos en está clase:

- Macros en general.
- Forma de regresar mas de un valor (datos definido por el usuario, apuntadores).
- Formas de pasar argumentos.
- Funciones que reciben un número variable de argumentos.
- Ejemplos de ciclos en general.
- switch-case.