Clase 6. Completando las sesiones pasadas: variables locales y globales. Ejercicio memoria dinamica, ciclos, funciones, arreglos, impresion hacia archivo.

Variables locales y globales

Variables globales

Las variables **globales** en C pueden ser modificadas, leidas y escritas desde cualquier función.
Las variables **locales** son variables que solo tienen alcance dentro del bloque o función donde son declaradas. Esto quiere decir, que son locaciones de memoria que son asignadas a ese nombre solo mientras la función o bloque está en ejecución, cuando la función termina esa memoria queda libre para su reuso.

Ejemplo, variables globales y locales


```c
#include <stdio.h>
int i;
int funcion(int i){
 i=0;
 printf("&i=%p en funcion, i=%d\n",&i, i);
 return i;
}
int funcion2(){
 i=8;
 printf("&i=%p en funcion2(global), i=%d\n",&i,i);
 return i;
}
```

```c
int main(){
 int i=1585;
 printf("&i=%p en main, i=%d\n",&i,i);
 funcion(i);
 funcion2();
 for (int i=0; i<1; i++){
  printf("&i=%p en for, i=%d\n",&i,i);
  if(i==0){
   int i=1964;
   printf("&i=%p en if, i=%d\n",&i,i);
  }
 }
return 0;
}
```

```
&i=0x7fff3a9a131c en main, i=1585
&i=0x7fff3a9a12fc en funcion, i=0
&i=0x601044 en funcion2(global), i=8
&i=0x7fff3a9a1318 en for, i=0
&i=0x7fff3a9a1314 en if, i=1964
```

do while

El ciclo do while, es similar al ciclo while pero la condición de paro se verifica después de cada ejecución (incluyendo la primera), esto quiere decir que primero ejecuta al menos una vez el bloque de código y después verifica la condición.

```
int condicion;
do
{
    printf(condicion?\n);
    scanf("%d",&condicion);
}while(condicion!=0)
```

break y continue

Las sentencias **break** y **continue** son utilizados para modificar el flujo de control de un programa.

- **break** En una estructura de control ocasiona la salida de la estructura.

```
int i=0;
while (i<=10)
{
   if (i==5)
      break;
   printf("i=%d ",i);
   i++;
}
```

Salida:
i=0 i=1 i=2 i=3 i=4

break y continue

Las sentencias **break** y **continue** son utilizados para modificar el flujo de control de un programa.

- **continue** En una estructura de repetición ocasiona el salto de las intrucciones remanentes en la estructura y pasa a la siguiente repetición.

Nota: **Note que el ejemplo es un loop infinito!**

```
1 int i=0;
  while (i<=10)
3 {
    if (i==5)
5       continue;
    printf("i=%d ",i);
7   fflush(stdout);
    i++;
9 }
```

Tipos básicos

char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 byte	1.2E-38 to 3.4E+38 6 decimal places
double	8 byte	2.3E-308 to 1.7E+308 15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932 19 decimal places

Tipos enum

El tipo **enum** es una forma de definir un dato categorico, aunque en realidad son enteros y pueden utilizarse como tales, sirven para facilidad de programación.


```
#include <stdio.h>
enum TIPO{A,B,C,D};
int main(int argc, char *argv[])
{
    enum TIPO x=A;
    printf("Tamanio en bytes=%d\n",sizeof(enum TIPO));
    for(int i=0; i<6; i++)
    {
        printf("Valor de x=%d valor de i=%d\n",x,i);
        if(x==A)
 printf("Valor de x=%d valor de A=%d\n",x,A);
        x++;
    }
    x=-10;
    printf("Valor negativo de x=%d \n",x);
return 0;
}
```

argc y argv

argc y argv son los argumentos que puede recibir el main cuando es llamada por el sistema,
argc contiene el número de argumentos provistos en la linea de comandos.
argv es la lista de argumentos, comenzando por el nombre del programa.

```
1 #include <stdio.h>
  int main(int argc, char *argv[])
3 {
    printf("argumento 0=%s argc=%d\n",argv[0],argc);
5   if (argc>1)
      printf("argumento 1=%s\n",argv[1]);
7   if (argc>2)
      printf("argumento 2=%s\n",argv[2]);
9 return 0;
  }
```

Ejecución:
./ejemplo
argc en el caso anterior vale 1, y argv[1]=./ejemplo
Ejecución:
./ejemplo -input archivo
argc en el caso anterior vale 3, y argv={./ejemplo, -input, archivo}

Ejercicio

- Abrir el archivo Tux en un editor de texto sencillo y ver lo que contiene: 2 define y un arreglo estático de chars.
- Incluir como cabecera este archivo.
- Los define son el número de renglones y columnas de los pixeles de una imagen, requerir una matriz entera de estas dimensiones de memoria dinámica.

- Los datos en el arreglo de chars representan valores de blanco o negro en una imagen. Por ejemplo, supongamos que la imagen tiene 2 de alto (renglones) por 15 de ancho (columnas). El arreglo de chars contiene algo como lo siguiente: {0xfa 0x12 0xa1 0x00}. Donde el primer renglón es {0xfa 0x12} y el segundo {0xa1 0x00}, esto se debe a que cada char codifica 8 bits y cada bit indica si la imagen es blanca o negra. Entonces, si tenemos una matriz de enteros de 2x15 para el ejemplo, los primeros 8 bits se toman del primer caracter, ej:
  0xfa=250={0, 1, 0, 1, 1, 1, 1, 1}...calculado como:
  (1*0+2*1+4*0+8*1+16*1+32*1+64*1+128*1)
  0x12=12={0, 0, 1, 1, 0, 0, 0, 0}...calculado como
  (1*0+2*0+4*1+8*1+16*0+32*0+64*0+128*0)

  Si la matriz de enteros se llama mat, entonces mat[0][i] para i desde 0 a 7 es el primer vector y desde el 8 al 14 es el segundo (el ultimo bit se deshecha.
  Pida memoria dinámica para la matriz, y llenela con los valores que le corresponden.
- Libere la memoria.

Continuación ejercicio, gnuplot

gnuplot es un programa de graficación, se puede llamar desde dentro de un programa en C, o puede ejecutarse externamente, una forma de ejecución es mediante un archivo.
Imprima un archivo con las siguientes lineas:

```
set style line 1 lc rgb 'black' pt 0 lw 1
set xrange [-1:220]
set yrange [-1:220]
plot '-'  w p ls 1 noti
2, 3
4, 5
e
```

Donde el 2,3 y 4,5 son las coordenadas de la matriz entera, que tienen valor de 1.
El archivo se puede cargar en gnuplot con la instrucción:

```
load 'archivo.gplot'
```