Clase 15. Librerias estándar y operaciones a nivel bit.

stdarg.g

stdarg.h, variables,macros

Variable	Description
va_list	Es un tipo adecuado para almacenar la información necesaria por va_start, va_arg y va_end.

Macro	Description
void va_start(va_list ap, last_arg)	Inicializa la variable **ap** para ser usada con el con **va_arg** y **va_end**. **last_arg** es el último argumento con nombre (o conocido) que se le ha pasado a la función.
type va_arg(va_list ap, type)	Esta macro adquiere (y regresa) el siguiente argumento en la lista de parametros, considerandolo de tipo **type**
void va_end (va_list ap)	La macro permite regresar (return) de forma segura a la función con argumentos variables desde donde se llamó **va_arg**. Si **va_end** no se llama, el resultado de la función no está definido.

stddef.h

stddef.h, variables,macros

Variable	Description
ptrdiff_t	Entero con signo para almacenar la diferencia entre dos apuntadores.
size_t	Entero sin signo para almacenar el resultado que regresa **sizeof**.
wchar_t	Entero sin signo que indica el tamaño del widechar.

Macro	Description
NULL	apuntados a NULL
offsetof(type, miembro)	Da como resultado un entero de tipo size_t que es el offset en bytes de un miembro de una estructura con respecto del inicio.

stddef.h


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

typedef struct{
  int x;
  double y,z;
}E;
int main(int argc, char *argv[])
{
  printf("Distancia a y=%p\n",offsetof(E,y));

  return 0;
}
```

Salida:

```
Distancia  a y=0x8
```

signal.h

Señales

- Una **señal** es una interrupción que es entregada a un programa por el sistema operativo.

Señales

- Una **señal** es una interrupción que es entregada a un programa por el sistema operativo.
- Existe un comportamiento por default para ciertas señales. Sin embargo, es posible, desde un programa en C definir el funcionamiento del programa cuando recibe una señal.

Señales

- Una **señal** es una interrupción que es entregada a un programa por el sistema operativo.
- Existe un comportamiento por default para ciertas señales. Sin embargo, es posible, desde un programa en C definir el funcionamiento del programa cuando recibe una señal.
- Existen un conjunto de señales definidas en el estándar ANSI y otras mas para sistemas POSIX, para conocerlas/verlas todas hay que revisar la documentación (incluso cada sistema en particular puede defirnir sus señales).

Señales, ejemplos de uso

- Una señal puede ser, por ejemplo: la terminación del programa, una división por cero, un acceso incorrecto a la memoria, etc.

Señales, ejemplos de uso

- Una señal puede ser, por ejemplo: la terminación del programa, una división por cero, un acceso incorrecto a la memoria, etc.
- Cuando el sistema operativo detecta este error o comportamiento, envia una señal al proceso que lo genero.

Señales, ejemplos de uso

- Una señal puede ser, por ejemplo: la terminación del programa, una división por cero, un acceso incorrecto a la memoria, etc.
- Cuando el sistema operativo detecta este error o comportamiento, envia una señal al proceso que lo genero.
- Esta señal puede ser capturada por el proceso, básicamente como un número entero.

Señales, ejemplos de uso

- Una señal puede ser, por ejemplo: la terminación del programa, una división por cero, un acceso incorrecto a la memoria, etc.
- Cuando el sistema operativo detecta este error o comportamiento, envia una señal al proceso que lo genero.
- Esta señal puede ser capturada por el proceso, básicamente como un número entero.
- El proceso puede decidir que hacer cuando recibe la señal.

Señales, ejemplos de uso

- Una señal puede ser, por ejemplo: la terminación del programa, una división por cero, un acceso incorrecto a la memoria, etc.
- Cuando el sistema operativo detecta este error o comportamiento, envia una señal al proceso que lo genero.
- Esta señal puede ser capturada por el proceso, básicamente como un número entero.
- El proceso puede decidir que hacer cuando recibe la señal.
- Existen comportamientos ya definidos por default (formas en que el proceso maneja las señales) que son los que usamos normalmente, los ejemplos a continuación son para *nosotros* definir que debe hacer el proceso.

signal.h, variables,macros

Variable	Descripción
sig_atomic_t	Es un tipo de variable entera que **se supone** atomic. Es decir, solo es accesada por un hilo o un proceso a la vez, y no existen accesos concurrentes. Se utiliza para manejar las señales.

Macro	Descripción
SIG_DFL	Es un indicador para la función que maneja las señales que indica que se use el comportamiento por default.
SIGABRT	Terminación anormal del programa.
SIGFPE	Error de punto flotante.
SIGILL	Operación ilegal.
SIGINT	Señal de interrupción tal como Ctrl-C.
SIGSEGV	Acceso invalido de memoria.
SIGTERM	Señal de requerimiento de terminación.

stddef.h, funciones

- **void (*signal(int sig, void (*func)(int)))(int)**. Fija la función que manejará la señal.

Ejemplo 1, signal.h


```c
#include <stdio.h>
#include <signal.h>
int main(int argc, char *argv[])
{
  //Indico como se debe de manejar la interrupcion
  signal(SIGINT, SIG_DFL);

  sleep(10);

  return 0;
}
```

Salida:

Ejemplo 2, signal.h, signal


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

void manejador(int senial)
{
  printf("Recibi la senial %d\n",senial);
}


int main(int argc, char *argv[])
{
  //Indico como se debe de manejar la interrupcion
  signal(SIGINT, manejador);

  sleep(10);

  return 0;
}
```

Salida (si se presion Ctrl-C):

```
^CRecibi la senial 2
```

Ejemplo 3, signal.h, raise


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

void manejador(int senial)
{
  printf("Recibi la senial %d\n",senial);
}


int main(int argc, char *argv[])
{
  //Indico como se debe de manejar la interrupcion
  signal(SIGINT, manejador);

  sleep(1);
  raise(SIGINT);
  return 0;
}
```

Salida (si se presion Ctrl-C):

```
Recibi la senial 2
```

Ejemplo 4, signal.h, psignal


```c
#include <stdio.h>
#include <signal.h>
void manejador(int senial){
   printf("Recibi la senial %d\n",senial);
}
void fija_seniales(){
  //Indico como se debe de manejar la interrupcion
  signal(SIGINT, manejador);
  //Para posix
  psignal(SIGFPE,"Error de punto flotante!\n");
}
int main(int argc, char *argv[]){
  fija_seniales();
  double x=1.0/0.0;
  raise(2);
  return 0;
}
```

Salida (si se presion Ctrl-C):

```
Error de punto flotante!
: Floating point exception
Recibi la senial 2
```

Ejemplo 4, signal.h, psignal


```
1  #include  <stdio.h>
   #include  <signal.h>
3  #include  <unistd.h>
   void manejador(int senial){
5     printf("Recibi la senial %d\n",senial);
      if (senial==SIGFPE)
7       abort();
   }
9  void fija_seniales(){
     //Indico como se debe de manejar la interrupcion
11  signal(SIGTERM, manejador);
    signal(SIGFPE, manejador);
13
   }
15 int main(int argc, char *argv[]){

17   fija_seniales();
     double x=-1/0; //con 0.0 da -inf
19   return 0;
   }
```

Salida (si se presion Ctrl-C):

```
Recibi la senial 8
Aborted
```

Tarea 8

prog8.2 Hacer un programa que habra un archivo de texto name_date.log usando un apuntador global. Donde **name** es el mismo nombre del programa que se está ejecutando (se puede sacar de argv), y date es la fecha en algun formato (diamesaño por ejemplo).

1. Provocar (de forma real no con raise()) señales de: Interrupción, terminación (comandos kill y killall, señal SIGTERM), punto flotante y acceso indebido de memoria.
2. Las señales de punto flotante y acceso indebido a memoria suceden de manera aleatoria (hay que generarlos de manera aleatoria).
3. Si el programa no recibe ningún argumento genera variables y datos aleatorios, si recibe un argumento es el nombre de un .log anterior.
4. El .log almacena los datos actuales, y el tipo de señal, cuando esta sucede.
5. A partir del log anterior se puede saber que tipo de señal fue la que lo genero y los datos, y entonces: si la señal es de acceso mal a memoria o operacion indebida de punto flotante, se le pregunta al usuario si desea continuar y se recargan los datos. Si la señal es de interrupción o terminación se recargan los datos y se reinicia.
6. Después de reiniciar se genera un nuevo .log (con la fecha cambiada).

Operaciones con bits

Operadores a nivel bit

- **Los operadores a nivel bit** en realida NO existen, lo que se hace es trabajar con bytes completos.
- El operador << recorre un número de posiciones los bits de una variable a la izquierda y le anexa ceros al final.
- El operador >> recorre un número de posiciones los bits de una variable a la derecha y le anexa ceros al principio.


```
 #include  <stdio.h>
int main(int argc, char *argv[]){
  unsigned int x=1;
   printf("Tamanio x en bits=%d\n", 8*sizeof(x));
   printf("Valor x=%d\n",x);
  x=x<<1;
   printf("Recorro 1 bit, valor=%d\n",x);
  x=x<<1;
   printf("Recorro 1 bit, valor=%d\n",x);
  x=x<<1;
   printf("Recorro 1 bit, valor=%d\n",x);
  return 0;
}

```

Operadores a nivel bit

Operador **AND**:
00010000 & 11111111 = 00010000
Operador **OR**:
00010000 |11111111 = 11111111
Operador **NOT**:
$\sim$ 11111111 = 0000000
Operador **XOR**:
01010101 $\wedge$ 11111111 = 10101010

Operadores a nivel bit


```c
#include <stdio.h>
int main(int argc, char *argv[]){
  unsigned int x=1;
   printf("Tamanio x en bits=%d\n", 8*sizeof(x));
   printf("Valor x=%d\n",x);
   x=x<<1;
   printf("Recorro 1 bit, valor=%d\n",x);
   x=x<<1;
   printf("Recorro 1 bit, valor=%d\n",x);
   x=x>>1;
   printf("Recorro 1 bit, valor=%d\n",x);
   return 0;
}
```

```
Tamanio x en bits=32
Valor x=1
Recorro 1 bit, valor=2
Recorro 1 bit, valor=4
Recorro 1 bit, valor=2
```

Operadores a nivel bit

Operador **AND**:

00010000 & 11111111 = 00010000

Operador **OR**:

00010000 |11111111 = 11111111

Operador **NOT**:

$\sim$ 11111111 = 0000000

Operador **XOR**:

01010101 $\wedge$ 11111111 = 10101010

Tarea 8

prog8.3 Hacer un programa con una funcion que:

1. Tenga dos funciones que toman 2 apuntadores a void **x** y **y**, y un número entero n. n es el número de bytes que miden los datos apuntados por los apuntadores void. Las funciones tienen los mismos argumento.
2. La función 1 transforma los bits (con operadores binarios) en **x** de la siguiente forma: 00=10, 01=11, 10=00 y 11=01. Y escribe el resultado en **y**.
3. La función 2 transforma los bits en **x** de la siguiente forma: 10=00, 11=01, 00=10 y 01=11.
4. El programa lee de la entrada estándar (consola) una línea de texto, e imprime el resultado (como caracteres) de aplicar las dos funciones.