Pregunta 1

a.- El siguiente es un ejemplo de uso del tipo de datos *Cola*:

```
Cola *c= nuevaCola();
int p1= estaVacia(c); // verdad
agregar(c, "pedro");
int p2= estaVacia(c); // falso
agregar(c, "juan");
agregar(c, "diego");
char *s1= extraer(c); // "pedro"
char *s2= extraer(c); // "juan"
char *s3= extraer(c); // "diego"
int p3= estaVacia(c); // verdad
```

Defina la estructura de datos de *Cola* y programe las funciones *nuevaCola*, *agregar*, *extraer* y *estaVacía*.

Restricciones: Para representar la cola Ud. debe usar una lista simplemente enlazada. Además el tiempo para agregar y extraer strings debe ser constante, es decir *O(1)*. Esto significa que en ninguna de las operaciones Ud. puede recorrer la lista completa.

Ayuda: En la estructura *Cola* agregue un puntero para direccionar el último nodo de la lista enlazada.

b.- El tipo *C_Cola* es similar a *Cola*, pero admite que sus operaciones sean invocadas desde múltiples threads. Sus operaciones son:

- `C_Cola *nuevaC_Cola();`
- `void c_agregar(C_Cola *c, char *s);`
- `char *c_extraer(C_Cola *c);`

No existe la operación *c_estaVacia* para *C_Cola*. Cuando se llama a *c_extraer* y la cola está vacía, *c_extraer* espera hasta que otro thread agregue un string y retorna ese mismo string.

Defina la estructura de datos de *C_Cola* y programe las 3 operaciones. Asegúrese de evitar los dataraces. Resuelva esta pregunta usando la cola de la *parte a*.

c.- La función de abajo se llama *peligrosa* porque durante la ejecución de *f* podría ocurrir una división por 0, terminando el programa.

```
typedef void (*Funcion)(void *ptr);
int peligrosa(Funcion f, Funcion g, void *ptr) {
  (*f)(ptr);
  return 0;
}
```

Reprograme *peligrosa* de modo que si durante la ejecución de *f* ocurre una división por 0 entonces se invoque *(*g)(ptr)* y se retorne -1.

Ayuda: cuando ocurre una división por cero, se gatilla la señal SIGFPE. ¡Captúrela! La función de atención de SIGFPE no puede retornar, porque si lo hace volverá a ocurrir la división por cero.

Pregunta 2

I.- La función *f* de abajo invoca las funciones *p*, *g* y *h*. Estas 3 funciones reciben un parámetro de tipo *double* y retornan un *double*. Las 3 toman mucho tiempo en calcularse. Además se sabe que el resultado de *p(x)* es verdadero el 90 % de las veces, en cuyo caso el tiempo de cálculo de *f(x)* es el tiempo de cálculo de *p(x) más* el tiempo de cálculo de *g(x)*. Se necesita paralelizar la función *f* considerando una máquina dual core.

```
double f(double x) {
  return p(x) ? g(x) : h(x);
}
```

Reprograme la función *f* lanzando un thread que calcule *g(x)* al mismo tiempo que el thread principal calcula *p(x)*. Así en el 90% de las veces el cálculo de *f(x)* tomará el mayor de los tiempos de cálculo de *p(x)* y *g(x),* en vez de la suma de ambos tiempos. Cuando *p(x)* sea falso descarte el resultado de *g(x)* y calcule *h(x)* en el thread principal. En tal caso no habrá mejora en el tiempo de cálculo de *f(x)*.

II.- Reprograme nuevamente la misma función *f* de la *parte I* (la versión original) usando ahora *fork* para crear un proceso pesado que calcule *g(x)*. Necesitará crear un pipe para que el hijo le envíe el resultado al padre. En el caso en que *p(x)* sea falso calcule *h(x)* en el proceso padre.

III.- Programe la función *palabras(str)* que transforma el string *str* dejando solo las palabras contenidas en *str* que estén formadas por caracteres alfabéticos. Además retorna el número de palabras encontradas. Las palabras quedarán en el mismo string *str* separadas por un espacio en blanco. Este es un ejemplo de uso:

```
char str[]=
  "  return ('a'<=ch && ch<='z') || ('A'<=ch && ch<='Z');";
int n= palabras(str);
// n=9 y str es "return a ch ch z A ch ch Z";
```

Restricciones: No use el operador de subindicación de arreglos [ ] ni su equivalente *(p+i), use aritmética de punteros. No puede pedir memoria con *malloc* ni declarar arreglos.

Ayuda: Declare 2 punteros. Use uno para recorrer los caracteres del string y el otro para ir almacenando los caracteres que permanecen en el string.