Pregunta 1

En un edificio en construcción de 10 pisos, múltiples trabajadores (representados por threads) requieren usar un taladro de precisión del cual hay una sola unidad. Para usarlo en el piso 10 se le debe solicitar llamando a *pedirTaladro(10)*. Esta función espera a que el taladro se desocupe y luego lo sube al piso 10. Después de usarlo se debe liberar llamando a *devolverTaladro( )*. Esta es una implementación correcta de *pedirTaladro* y *devolverTaladro*:

```
pthread_mutex_t m= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c= PTHREAD_COND_INITIALIZER;
int piso_actual= 1, ocup= 0;
```

```
void pedirTaladro(int piso){
  lock(&m);
  while (ocup)
    wait(&c, &m);
  ocup= 1;
  unlock(&m);
  moverTaladro(piso_actual,
               piso); // ¡lento!
  piso_actual= piso;
}
void devolverTaladro() {
  lock(&m);
  ocup= 0;
  broadcast(&c);
  unlock(&m);
}
```

La función *moverTaladro* es dada. Es lenta y toma tiempo proporcional a la cantidad de pisos que mueve el taladro. Con el fin de reducir el tiempo de traslado del taladro se pide reprogramar la función *pedirTaladro* de modo que se cumpla la siguiente estrategia de asignación: Cuando hay varias peticiones pendientes se atiende primero una petición que requiere el taladro en el mismo piso en donde se encuentra actualmente o la que lo requiera en el piso superior más cercano. Si no hay ninguna petición con esas características, se atiende primero la petición que lleve el taladro al piso de más abajo. Después se continúa atendiendo las peticiones de abajo hacia arriba. Por ejemplo si el taladro está en el piso 4 y hay peticiones para los pisos 2, 2, 3, 4, 6 y 10, el orden de atención sería 4, 6, 10, 2, 2 y 3, a menos que en el intertanto lleguen nuevas peticiones para otros pisos.

*Ayud*a: defina un arreglo *pend* de tamaño 11 de manera que *pend[i]* es la cantidad de peticiones pendientes para el piso *i*. Programe la función *prox(pend, p)* que retorne el siguiente piso a donde se debe mover el taladro dado que se encuentra en el piso *p*. Reprograme *pedirTaladro* usando *pend* y *prox*.

Pregunta 2

Programe la función *rotar* con el siguiente encabezado:

```
void rotar(char *s, int n);
```

Esta función debe rotar el string *s* en *n* caracteres, lo que significa que *s[0]* debe pasar a *s[n]*, *s[1]* a *s[n+1]*, …, *s[i]* a *s[(i+n)%strlen(s)]*, …, etc. El siguiente es un ejemplo de uso:

```
char s[ ] = "abcdefghijk";
rotar(s, 3); // s es el string "ijkabcdefgh"
```

Ud. sí puede usar funciones como *strlen*, *strcpy*, *strncpy*, etc.

Restricciones: No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. Ud. puede declarar un arreglo de a los más *n* caracteres, no más. No puede crear un copia de todo *s*.

Pregunta 3

Programe eficientemente la función *eliminarRango* que elimina de un conjunto de enteros todos los elementos que estén en el rango [y,z]. El conjunto se representa mediante una lista enlazada *ordenada ascendentemente*. Esta es la estructura de la lista enlazada y el encabezado de *eliminarRango*:

```
typedef struct nodo {
  int x;              // un elemento del conjunto
  struct nodo *prox;  // próximo nodo de la lista enlazada
} Nodo;
void eliminarRango(Nodo **phead, int y, int z);
```

Por ejemplo suponga que *head* es de tipo *Nodo** y corresponde a la lista enlazada que contiene 2, 4, 7, 8 y 9, entonces:

- *eliminarRango(&head, 4, 8)* dejaría a *head* con 2 y 9
- *eliminarRango(&head, 0, 5)* dejaría a *head* con 7, 8 y 9
- *eliminarRango(&head, 5, 10)* dejaría a *head* con 2 y 4.

Ud. debe liberar con *free* la memoria de todos los nodos eliminados de la lista. El requisito de eficiencia significa que Ud. debe eliminar todos los nodos con un sólo recorrido de la lista enlazada.

Pregunta 4

En el cuadro de la derecha se define la función *f*. Considere que las funciones *p*, *g* y *h* pueden tomar mucho tiempo de ejecución. El caso peor para el tiempo de cálculo de *f* es el tiempo que toma calcular *p* más el máximo que toma calcular *g* o *h*.

```
int p(double x);
double g(double x);
double h(double x);
double f(double x) {
  if (p(x))
    return g(x);
  else
    return h(x);
}
```

Reprograme *f* de manera que se calcule en paralelo *p*, *g* y *h* al ejecutar en un computador con 3 cores. De esta forma el caso peor mejora, puesto que será el máximo tiempo que toma calcular *p*, *g* o *h*. Para lograrlo Ud. debe crear 2 procesos hijos usando *fork*. Los hijos deben entregar sus resultados al padre por medio de pipes. El padre calcula *p*.