Pregunta 1

Parte a.- Programe la función *raiz* con el siguiente encabezado:

```
typedef double (*Funcion)(void *ptr, double x);
double raiz(Funcion f, void *ptr,
            double x0, double x1, double eps);
```

Esta función debe usar el *método de la bisección* para encontrar una raíz (o cero) de la función *f(ptr, x)*, es decir encontrar *x* tal que *f(ptr, x)=0*. La búsqueda de *x* se debe hacer entre x_0 y x_1 sabiendo que *f(ptr,* x_0*)* y *f(ptr,* x_1*)* son de signos opuestos. El máximo error tolerable para *x* es *eps*. El método de la bisección es un algoritmo simple para calcular una raíz de la función *f*. Este consiste en calcular $x_m=\frac{x_0+x_1}{2}$ y evaluar *f(ptr,* x_m*)*. Si *f(ptr,* x_m*)* es del mismo signo que *f(ptr,* x_0*)* se hace $x_0=x_m$, si no $x_1=x_m$. Se repite el procedimiento hasta que $|x_1-x_0|\leq eps$.

Por ejemplo si se desea calcular una raíz de la función e^{ax}*-10* entre 0 y 1 se debe definir la función:

```
double expax(void *ptr, double x) {
  double *pa= ptr, a= *pa;
  return exp(a*x)-10; // calcula e elevado a a*x-10
}
```

y luego calcular:

```
double a= …;
double cero= raiz(expax, &a, 0., 1., 0.00001);
```

Parte b.- Considere el polinomio: $a_n x^n+a_{n-1}x^{n-1}+\cdots+a_1 x+a_0$

Programe la función *raiz_poli* con el siguiente encabezado:

```
double raiz_poli(double a[], int n,
                 double x0, double x1, double eps);
```

Esta función debe calcular la raíz del polinomio de más arriba entre x_0 y x_1 sabiendo que la evaluación del polinomio en estos 2 puntos es de signos opuestos. Los coeficientes $a_n, a_{n-1}, \ldots, a_1, a_0$ corresponden a los elementos del arreglo *a[n]*, *a[n-1]*, …, *a[1]*, *a[0]*, en ese mismo orden. El error tolerado para la raíz es *eps*.

Restricción: Ud. debe usar la función *raiz* de la parte *a* para hacer la búsqueda de la raíz del polinomio.

Pregunta 2

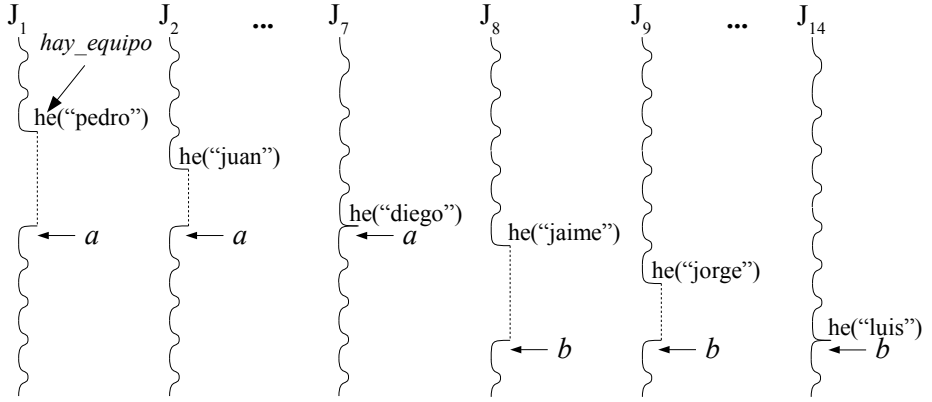
Se necesita formar equipos de 7 jugadores de futbolito. Los jugadores son representados por threads que invocan la función *hay_equipo* indicando como argumento su nombre. Esta función espera hasta que 7 jugadores hayan invocado la misma función retornando un arreglo de 7 strings con los nombres del equipo completo. Este es un ejemplo del código de un jugador:

```
void *jugador(void *ptr) {
  char *nombre= ptr;
  for (;;) {
    dormir();
    char **equipo= hay_equipo(nombre);
    jugar_futbolito(equipo);
    beber_cerveza();
} }
```

Programe la función *hay_equipo*. El encabezado es el siguiente:

```
char **hay_equipo(char *nombre);
```

Las funciones *dormir*, *jugar_futbolito* y *beber_cerveza* son dadas. El siguiente diagrama muestra un ejemplo de ejecución:



Observe que la llamada a *hay_equipo* espera hasta que se haya formado un equipo con 7 jugadores. Los primeros 7 jugadores (J_1 a J_7) forman el equipo *a* y por lo tanto sus llamadas a *hay_equipo* retornan el mismo arreglo *a* con los 7 nombres del equipo: "pedro", "juan", …, "diego". Los siguientes 7 jugadores (J_8 a J_{14}) forman el equipo *b* y sus llamadas a *hay_equipo* retornan el arreglo *b*, distinto de *a*, con los nombres "jaime", "jorge", …, "luis". Resuelva el problema usando variables globales como un mutex, una condición, etc.