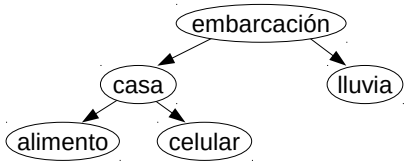
Pregunta 1

Un diccionario se implementa mediante un árbol de búsqueda binaria (ABB) almacenado en el archivo *dicc.txt* en el siguiente formato:

```
1     2     embarcacion  todo tipo de artilugio capaz de navegar en el agua
3     4     casa         edificacion construida para ser habitada
-1    -1    lluvia       condensacion del vapor de agua contenida en las nubes
-1    -1    alimento     sustancia ingerida por un ser vivo
-1    -1    celular      aparato portatil de un sistema de telefonia celular
```

Cada línea del archivo es de 100 caracteres y corresponde a un nodo del ABB. Los primeros 10 caracteres corresponden al número de la línea en donde se encuentra el hijo izquierdo o -1 si es nulo. Los 10 siguientes caracteres son el número de línea del hijo derecho (o -1). Luego vienen 20 caracteres para la llave y 59 caracteres para el valor asociado. La línea se termina con un *\n* para completar los 100 caracteres. La primera línea es la número 0 y corresponde a la raíz del ABB. Siempre existe un espacio en blanco que separa una columna de la otra. El archivo del ejemplo representa el siguiente ABB:



Programe el comando *consultar* que despliega el valor asociado a una llave. Ejemplos de invocaciones y sus resultados son:

```
$ ./consultar dicc.txt casa
edificacion construida para ser habitada
$ ./consultar dicc.txt embarcacion
todo tipo de artilugio capaz de navegar en el agua
$ ./consultar dicc.txt nadaquever
palabra no encontrada
$
```

Ud. debe buscar *eficientemente* la llave en el archivo, lo que quiere decir que no puede buscar secuencialmente. Tenga cuidado en considerar que *strcmp("casa", "casa ")* es distinto de 0.

Para obtener el número almacenado en un string use la función *atoi*. Por ejemplo *atoi(" -123 ")* entrega -123 y *atoi(" 8 10")* entrega 8, es decir que *atoi* solo considera el primer número del string.

A modo de recuerdo, esta es la API para manejar archivos:

```
FILE *fopen(char *nom_arch, char *modo);  // modo es "r" para lectura
size_t fread(char *buf, size_t ancho_item, int n_item, FILE *file);
int fseek(FILE *file, long despl, int w);  // w==SEEK_SET
int fclose(FILE *file);
```

Pregunta 2

Parte a.- El sabio chino, dueño del restaurant que visitan los 5 filósofos, ha accedido a que un filósofo se siente en cualquiera de las 5 sillas de la mesa. Ya no es necesario que el filósofo *i* se siente en la *i-ésima* silla. Con esto la función que representa a cada filósofo queda como sigue:

```
void filosofo(int i) {
  for (;;) {
    int k= buscarSilla(); // k entre 0 y 4
    comer(k, (k+1)%5);
    desocuparSilla(k);
    pensar();
} }
```

Programe las funciones *buscarSilla* y *desocuparSilla*. Recuerde que hay solo 5 palitos, que un filósofo que se sienta en la silla *k* (con *k* entre 0 y 4) necesita los palitos *k* y *(k+1)%5* para poder comer y que 2 filósofos no pueden comer con el mismo palito simultáneamente. Su solución debe usar eficientemente los palitos, lo que significa que cuando existen sillas con sus 2 palitos desocupados y hay un filósofo con la intención de comer, *buscarSilla* le debe asignar una silla de inmediato. No se requiere evitar hambruna.

Parte b.- La siguiente función busca un factor del número entero *x* en el rango *[i, j]*:

```
typedef unsigned long long ulonglong;
typedef unsigned int uint;
uint buscarFactor(ulonglong x, uint i, uint j) {
  for (uint k= i; k<=j; k++) {
    if (x % k == 0)
      return k; // x es divisible por k
  }
  return 0; // x no es divisible por ningún entero en el rango [i, j]
}
```

Reprograme la función *buscarFactor* de manera que la búsqueda se haga paralelamente en 8 cores.