INF-155: Algoritmos y complejidad
Tarea #3
Recursion is beautiful

Anghelo Carvajal
201473062-4

23 de octubre de 2018

1. Pregunta 1

Escriba una versión rudimentaria de `grep(1)`, al que se le llama como:

`tarea-3-1 <expresión><archivo>`

que escriba todas las líneas en que la ¡expresión¿calza. No escriba nada más.

1.1. Respuesta 1

Teniendo en consideración lo pedido, no es necesario modificar el código de `match.c` que se nos ha entregado. Tan solo debemos escribir funciones que usen las funciones de `match.c`. Un simple `main.c`.

El código es el siguiente:

Código 1: `main.c`

```c
#include "match.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

void checkCommandLine(int argc, char *argv[]){
    if(argc < 3){
        printf("Usage: %s <expression> <file>\n", argv[0]);
        exit(1);
    }
}

void testError(bool expression, const char *message){
    if(expression){
        fprintf(stderr, "%s\n", message);
        exit(2);
    }
}

char *readFullFile(const char *fileName){
    FILE *opened = fopen(fileName, "r");
    testError(opened == NULL, "Couldn't open file.");

    testError(fseek(opened, 0, SEEK_END) != 0, "fseek failed.");
    long size = ftell(opened);
    testError(fseek(opened, 0, SEEK_SET) != 0, "fseek failed.");

    char *fullFile = (char *)malloc(sizeof(char) * size);
    testError(fullFile == NULL, "malloc failed.");

    size_t readed = fread(fullFile, sizeof(char), size, opened);
    testError(readed != size, "fread failed.");
    fclose(opened);

    return fullFile;
}

int main(int argc, char *argv[]){
    checkCommandLine(argc, argv);
    char *fullFile = readFullFile(argv[2]);

    for(char *tok = strtok(fullFile, "\n"); tok != NULL; tok = strtok(NULL, "\n")){
        if(match(argv[1], tok)){
            printf("%s\n", tok);
        }
    }

    free(fullFile);
    return 0;
}
```

Este código consta de 4 funciones:

- `void checkCommandLine(int argc, char *argv[])`. Función que chequea que los parámetros con los que se invoco el programa sean los mínimos (2 parámetros).

  Se invoca con los mismos parámetros que recibe la función `main`. Si los parámetros no son los suficientes, entrega un mensaje a `stdout` y sale del programa.

- `void testError(bool expression, const char *message)`. Una suerte de `assert`.

  Si `expression` es `true` imprime al `stderr` el mensaje `message` y luego finaliza la ejecución del programa.

  En caso de que `expression` sea `false`, esta función hace nada.

- `char *readFullFile(const char * fileName)`. Función que recibe un nombre de archivo y entrega un arreglo de caracteres que contiene los contenidos del archivo.

  En caso de algún error, esta función finaliza la ejecución del programa mostrando un mensaje en `stderr` (invoca a `testError()`).

  Note que el arreglo de caracteres fue creado usando `malloc`, así que es responsabilidad del *caller* llamar al `free` correspondiente.

- `int main(int argc, char *argv[])`. Muestra las lineas del archivo que calzan con la expresión entregada como parámetro.

En el resto de preguntas lo único que cambia es el código de `match.c`, de modo que el archivo `main.c` sera el mismo en para todas las preguntas.

2. Pregunta 2

Modifique el código dado para agregar la operación '+' (una o más veces lo anterior).

2.1. Respuesta 2

Para poder implementar lo pedido, debemos agregar una función. Como somos sumamente creativos, la llamaremos `matchplus()`. Esta función es casi igual que la función `matchstar()`, con la diferencia que se cambia el `do-while` por un `while`, debido a que debemos calzar al menos 1 del carácter pedido.

Luego simplemente agregaremos esta posibilidad dentro de `matchhere()`.

El código resultante es:

Código 2: `2/match.c`

```c
#include "match.h"

/* match: search for regexp anywhere in text */
int match(char *regexp, char *text){
    if (regexp[0] == '^'){
        return matchhere(regexp + 1, text);
    }
    do {    /* must look even if string is empty */
        if (matchhere(regexp, text)) {
            return 1;
        }
    } while (*text++ != '\0');
    return 0;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text){
    if (regexp[0] == '\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[1] == '+')
        return matchplus(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}
```

```
30
31 /* matchstar: search for c*regexp at beginning of text */
32 int matchstar(int c, char *regexp, char *text){
33     do {    /* a * matches zero or more instances */
34         if (matchhere(regexp, text))
35             return 1;
36     } while (*text != '\0' && (*text++ == c || c == '.'));
37     return 0;
38 }
39
40 /* matchplus: search for c+ regexp at beginning of text */
41 int matchplus(int c, char *regexp, char *text){
42     while(*text != '\0' && (*text++ == c || c == '.')){
43         if (matchhere(regexp, text)) /* a + matches one or more instances */
44             return 1;
45     }
46     return 0;
47 }
```

3. Pregunta 3

Modifique el código dado para agregar la operación '?' (cero o una vez lo anterior).

3.1. Respuesta 3

Agregaremos una nueva función que se encargue de calzar el caso '?'. En base a nuestra creatividad, la llamamos `matchquestion()`. La implementación es sencilla, simplemente si es que el carácter calza, avanzamos 1 en la lectura del texto. Si no calza, no avanzamos.

Luego agregamos esta posibilidad al conjunto de *if*s de la función `matchhere()`.

El resultado es el siguiente:

Código 3: `3/match.c`

```
1  #include "match.h"
2
3  /* match: search for regexp anywhere in text */
4  int match(char *regexp, char *text){
5      if (regexp[0] == '^') {
6          return matchhere(regexp + 1, text);
7      }
8      do {    /* must look even if string is empty */
9          if (matchhere(regexp, text)) {
10             return 1;
11         }
12     } while (*text++ != '\0');
13     return 0;
14 }
15
16 /* matchhere: search for regexp at beginning of text */
17 int matchhere(char *regexp, char *text){
18     if (regexp[0] == '\0')
19         return 1;
20     if (regexp[1] == '*')
21         return matchstar(regexp[0], regexp+2, text);
22     if (regexp[1] == '+')
23         return matchplus(regexp[0], regexp+2, text);
24     if (regexp[1] == '?')
25         return matchqustion(regexp[0], regexp+2, text);
26     if (regexp[0] == '$' && regexp[1] == '\0')
27         return *text == '\0';
28     if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
29         return matchhere(regexp+1, text+1);
30     return 0;
31 }
32
33 /* matchstar: search for c*regexp at beginning of text */
```

```c
int matchstar(int c, char *regexp, char *text){
    do {    /* a * matches zero or more instances */
        if (matchhere(regexp, text))
            return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}

/* matchplus: search for c+ regexp at beginning of text */
int matchplus(int c, char *regexp, char *text){
    while(*text != '\0' && (*text++ == c || c == '.')){
        if (matchhere(regexp, text))  /* a + matches one or more instances */
            return 1;
    }
    return 0;
}

/* matchqustion: search for c? regexp at beginning of text */
int matchqustion(int c, char *regexp, char *text){
    /* a + matches zero or one instances */
    if(c == *text || c == '.'){
        return matchhere(regexp, text+1);
    }
    return  matchhere(regexp, text);
}
```

4. Pregunta 4

Al programa con ambas operaciones adicionales agregue la posibilidad de citar un carácter especial, vale decir, escribir por ejemplo '\?' para calzar un '?'.

4.1. Respuesta 4

Finalmente, para poder calzar un carácter especial que comienza con '\', creamos una función (`matchscape()`!), la cual calza el carácter que se quiere escapar. Si no calza, retorna 0. Si calza, sigue leyendo.

Esta función también considera la posibilidad de que el carácter escapado este continuado por los caracteres especiales '*', '+' o '?' y los calza de forma correspondiente.

Lo agregamos a `matchhere` y tenemos el problema solucionado.

Note que la forma en que se implemento, si se encuentra el carácter '\', el carácter siguiente debe calzar en el texto, o

Código 4: `4/match.c`

```c
#include "match.h"

/* match: search for regexp anywhere in text */
int match(char *regexp, char *text){
    if (regexp[0] == '^') {
        return matchhere(regexp + 1, text);
    }
    do {    /* must look even if string is empty */
        if (matchhere(regexp, text)) {
            return 1;
        }
    } while (*text++ != '\0');
    return 0;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text){
    if (regexp[0] == '\0')
        return 1;
    if(regexp[0] == '\\')
        return matchscape(regexp[1], regexp+2, text);
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
```

```
24      if (regexp[1] == '+')
25          return matchplus(regexp[0], regexp+2, text);
26      if (regexp[1] == '?')
27          return matchqustion(regexp[0], regexp+2, text);
28      if (regexp[0] == '$' && regexp[1] == '\0')
29          return *text == '\0';
30      if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
31          return matchhere(regexp+1, text+1);
32      return 0;
33  }
34
35  /* matchstar: search for c*regexp at beginning of text */
36  int matchstar(int c, char *regexp, char *text){
37      do {    /* a * matches zero or more instances */
38          if (matchhere(regexp, text))
39              return 1;
40      } while (*text != '\0' && (*text++ == c || c == '.'));
41      return 0;
42  }
43
44  /* matchplus: search for c+ regexp at beginning of text */
45  int matchplus(int c, char *regexp, char *text){
46      while(*text != '\0' && (*text++ == c || c == '.')){
47          if (matchhere(regexp, text)) /* a + matches one or more instances */
48              return 1;
49      }
50      return 0;
51  }
52
53  /* matchqustion: search for c? regexp at beginning of text */
54  int matchqustion(int c, char *regexp, char *text){
55      /* a + matches zero or one instances */
56      if(c == *text || c == '.'){
57          return matchhere(regexp, ++text);
58      }
59      return  matchhere(regexp, text);
60  }
61
62  /* matchscape: search for \c regexp at beginning of text */
63  int matchscape(int c, char *regexp, char *text){
64      if (regexp[0] == '*')
65          return matchstar(c, ++regexp, text);
66      if (regexp[0] == '+')
67          return matchplus(c, ++regexp, text);
68      if (regexp[0] == '?')
69          return matchqustion(c, ++regexp, text);
70
71      return c == *text ? matchhere(regexp, ++text) : 0;
72  }
```