ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΙΑ 6

Όνομα Μαθητή : Άγγελος Κωνσταντίνος
Επίθετο Μαθητή : Μεντζέλος

Αριθμός Μητρώου : 21390132

Ημερομηνία ολοκλήρωσης :
18/1/2023

ΑΡΧΗ ΑΣΚΗΣΗΣ 6

ΚΡΥΠΤΟΛΕΞΟ

Να γραφεί πρόγραμμα το οποίο, αρχικά, θα διαβάζει από ένα αρχείο κειμένου (το όνομα του οποίου δίνεται ως όρισμα στην γραμμή εντολών) τα παρακάτω δεδομένα:

- Το πλήθος των γραμμών (M) και το πλήθος των στηλών (N) ενός πίνακα γραμμάτων. Οι δύο αριθμοί βρίσκονται στην ίδια γραμμή και διαχωρίζονται με τον χαρακτήρα κόμμα “,”
- Ένα πίνακα κεφαλαίων γραμμάτων M X N θέσεων. Κάθε γραμμή του πίνακα βρίσκεται σε ξεχωριστή γραμμή του αρχείου.
- Το πλήθος των λέξεων που βρίσκονται κρυμμένες στον πίνακα (K).
- Τις K λέξεις που βρίσκονται κρυμμένες στον πίνακα. Κάθε λέξη είναι γραμμένη σε ξεχωριστή γραμμή του αρχείου.

Στην συνέχεια θα εντοπίζει τις κρυμμένες λέξεις, με αλφαβητική σειρά, στον πίνακα γραμμάτων. Οι λέξεις μπορεί να είναι γραμμένες στον πίνακα οριζόντια, κάθετα ή διαγώνια και προς τις δύο κατευθύνσεις. Κάποια γράμματα στον πίνακα μπορεί να ανήκουν σε περισσότερες από μία λέξεις. Για κάθε μία από τις λέξεις που κρύβονται στον πίνακα το πρόγραμμα θα τυπώνει την λέξη, την γραμμή και τη στήλη του πίνακα στην οποία βρίσκεται το πρώτο γράμμα της λέξης και την γραμμή και τη στήλη του πίνακα που βρίσκεται το τελευταίο γράμμα της λέξης.

Αφού βρει όλες τις κρυμμένες λέξεις, το πρόγραμμα θα αναφέρει τα γράμματα του πίνακα που δεν ανήκουν σε κάποια από αυτές. Η παρουσίαση των γραμμάτων θα γίνεται με τη σειρά που βρίσκονται στον πίνακα (από αριστερά προς τα δεξιά και από πάνω προς τα κάτω).

Το πρόγραμμα, μεταξύ άλλων, θα πρέπει να κάνει έλεγχο ορθότητας των δεδομένων του αρχείου. Σε περίπτωση λάθους θα τυπώνει κατάλληλο μήνυμα και θα τερματίζεται. Η υλοποίηση θα πρέπει να πληροί όλες τις ακόλουθες προδιαγραφές:

- Το πρόγραμμα δεν θα περιέχει ολικές (global) μεταβλητές.
- Το πρόγραμμα δεν θα περιέχει στατικά δεσμευμένους πίνακες. Όλοι οι πίνακες που θα χρησιμοποιηθούν θα πρέπει να δεσμευτούν δυναμικά.
- Δεν επιτρέπεται η χρήση της εντολής “goto” και δεν επιτρέπεται ο τερματισμός βρόχων με την χρήση της “break”.

Υλοποιήσεις οι οποίες παραβιάζουν έστω και μία από τις προηγούμενες προδιαγραφές δεν θα γίνουν δεκτές και οι εργασίες θα μηδενιστούν. Το παραδοτέο, εκτός από τον πηγαίο κώδικα και την τεκμηρίωση, θα περιέχει και τουλάχιστον δύο αρχεία εισόδου.

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

```
char **create_grid(int ,int ,  FILE * );
char **create_words(int , FILE *);
bool **create_unused(int , int );
void print_grid(char ** , int   , int );
void print_words(char ** , int  );
void print_word(char*,int,int , int , int);
void print_unused(char **,bool** , int , int);
void free_grid(char ** , int );
void free_unused(bool ** , int );
void sort(char ** , int , int);
int searchWord(char **,char*, int  , int  , bool **);

int main(int argc ,  char **argv)
{
       int m , n ,i  , find;
       char **pin;
    char **words;
    bool **visited;
    FILE *fp;
    if (argc!=2)
        perror("Error... Must put a file ");
    fp = fopen(argv[1],"r"); assert(fp!=NULL);
    fscanf(fp,"%d,%d\n",&m,&n);
       pin=create_grid(m,n,fp);
    fscanf(fp,"%d",&find);
       visited=create_unused(m,n);
    words=create_words(find,fp);
    sort(words,find,n);
    for( i =0 ; i < find;i++)
    {
        if(searchWord(pin,words[i],m,n,visited))
        {
            continue;
        }
        else
        {
            printf("The word %s not found\n",words[i]);
        }
    }
    print_unused(pin,visited,m,n);
    free_grid(pin,m);
    free_grid(words,find);
    free_unused(visited,m);
    fclose(fp);
       return 0;
}
char **create_grid(int m , int n ,  FILE *fp )
{
       char **array;
       int i,j;
    array=(char **)malloc(m*sizeof(char *)); assert(array!=NULL);
    for ( i=0;i<m;i++)
    {
        array[i]=(char*)malloc(n*sizeof(char)); assert(array[i]!=NULL);
    }
       for (i=0;i<m;i++)
    {
             for( j=0;j<n;j++)
```

```
        {
                    fscanf(fp,"%c ",&array[i][j]);
        }
    }
      print_grid(array,m,n);
      return array;
}
char **create_words(int rows , FILE *fp)
{
    char **words;
    int i , j ;
    char *buf;
    buf=(char*)malloc(201); assert(buf!=NULL);
    printf("The words I'm looking for are %d times\n",rows);
    words=(char ** )malloc((rows)*sizeof(char*)); assert(words!=NULL);
    for(i=0;i<rows;i++)
    {
        fscanf(fp,"%s ",buf);
        words[i]=(char*)malloc((strlen(buf)+1)*sizeof(char));
assert(words[i]!=NULL);
        strcpy(words[i],buf);
    }
    //print_words(words,rows);
    free(buf);
    return words;
}
bool **create_unused(int m , int n)
{
    int i,j;
    bool **visited;
    visited=(bool **)malloc(m*sizeof(bool *));
    for(i=0;i<m;i++)
    {
        visited[i]=(bool*)malloc(n*sizeof(bool));
    }
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            visited[i][j]=false;
        }
    }
    return visited;
}
void free_grid(char **grid , int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        free (*(grid + i));
    }
    free(grid);
}
void free_unused(bool **grid , int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        free (*(grid + i));
```

```
    }
    free(grid);
}
void print_word(char *word , int row1, int col1 , int row, int col)
{
    printf("The word %s found in direction first letter : %d,%d and last
letter : %d,%d \n",word,row1+1,col1+1 , row +1 , col+1);
}
void print_grid(char **grid , int m  , int n )
{
    int i , k ;
    printf("WordSearch %d,%d\n",m,n);
    for(i=0;i<n*3;i++)
    {
        printf("-");
    }
    printf("\n");
    for(k=0;k<m;k++)
    {
        for(i=0;i<n;i++)
        {
            printf("[%c]",grid[k][i]);
        }
        printf("\n");
    }
    for(i=0;i<n*3;i++)
    {
        printf("-");
    }
    printf("\n");
}
void print_words(char **words , int m  )
{
    int i , k ;
    printf("The words are :\n");
    for(k=0;k<m;k++)
    {
        printf("%s\n",words[k]);
    }
}
void sort(char **array , int m , int n)
{
    int swap,i;
    char *tmp;
    do
    {
        swap=0;
        for (i=0;i<m-1;i++)
            if (strcmp(array[i],array[i+1]) > 0)
            {
                tmp=array[i];
                array[i]=array[i+1];
                array[i+1]=tmp;
                swap=1;
            }
    }
    while(swap);
}
void print_unused(char **grid ,bool **visited , int m  , int n )
```

```
{
    int i , k ;
    printf("The unused letters are : ");
    for(k=0;k<m;k++)
    {
        for(i=0;i<n;i++)
        {
            if(!visited[k][i])
            {
                printf("%c",grid[k][i]);
            }
        }
    }
}
int searchWord(char **grid,char*word, int m  , int n , bool **visited)
{
    int i,j , k , p ,found  , count , last_i=0,last_j=0;
    int word_len=strlen(word);
    //printf("%d",word_len);
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            if(grid[i][j]==word[0])
            {
                //right
                if(j + word_len <= n && grid[i][j+1]==word[1])
                {
                    count = 0;
                    /*second(i,j,grid,word,)*/
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i][j+p]==word[p] && j + word_len <= n)
                        {
                            found = 0;
                            visited[i][j+p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_j=j+count-1;
                            print_word(word,i,j , i , last_j);
                            return 1;
                        }
                    }
                }
                //down
                if(i + word_len <= m && grid[i+1][j]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i+p][j]==word[p] && i + word_len <= m)
                        {
                            found = 0;
                            visited[i+p][j]=true;
                            count++;
                        }
                        if(count==word_len)
```

```
                        {
                            last_i=i+count-1;
                            print_word(word,i,j , last_i , j);
                            return 1;
                        }
                    }
                }
                //down right
                if(i + word_len <= m && j + word_len <= n &&
grid[i+1][j+1]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i+p][j+p]==word[p] && i + word_len <= m && j
+ word_len <= n)
                        {
                            found = 0;
                            visited[i+p][j+p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_i=i+count-1;
                            last_j=j+count-1;
                            print_word(word,i,j , last_i , last_j);
                            return 1;
                        }
                    }
                }
                //down left
                if(i + word_len <= m && j - word_len >= -1 && grid[i+1][j-
1]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i+p][j-p]==word[p] && i + word_len <= m && j
- word_len >= -1)
                        {
                            found = 0;
                            visited[i+p][j-p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_i=i+count-1;
                            last_j=j-count+1;
                            print_word(word,i,j , last_i , last_j);
                            return 1;
                        }
                    }
                }
                //up
                if(i - word_len >= -1 && grid[i-1][j]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
```

```
                        if(grid[i-p][j]==word[p] && i - word_len >= -1)
                        {
                            found = 0;
                            visited[i-p][j]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_i=i-count+1;
                            print_word(word,i,j , last_i , j);
                            return 1;
                        }
                    }
                }
                //left
                if(j - word_len >= -1 && grid[i][j-1]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i][j-p]==word[p] && j - word_len >= -1)
                        {
                            found = 0;
                            visited[i][j-p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_j=j-count+1;
                            print_word(word,i,j , i , last_j);
                            return 1;
                        }
                    }
                }
                //up right
                if(i - word_len >= -1 && j + word_len <= n && grid[i-
1][j+1]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i-p][j+p]==word[p] && i - word_len >= -1 &&
j + word_len <= n)
                        {
                            found = 0;
                            visited[i-p][j+p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_i=i-count+1;
                            last_j=j+count-1;
                            print_word(word,i,j , last_i , last_j);
                            return 1;
                        }
                    }
                }
                //up left
```

```
                if(i - word_len >= -1 && j - word_len >= -1 && grid[i-1][j-
1]==word[1])
                {
                    count = 0;
                    for(p=0;p<word_len;p++)
                    {
                        if(grid[i-p][j-p]==word[p] && i - word_len >= -1 &&
j - word_len >= -1)
                        {
                            found = 0;
                            visited[i-p][j-p]=true;
                            count++;
                        }
                        if(count==word_len)
                        {
                            last_i=i-count+1;
                            last_j=j-count+1;
                            print_word(word,i,j , last_i , last_j);
                            return 1;
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

ΤΕΛΟΣ ΑΣΚΗΣΗΣ 6