QROC RN Informatique – Avril 2013

Répondre sur la feuille impérativement. Notes personnelles et polycopiés distribués seuls autorisés. Calculatrice autorisée.

NOM : *Correction*
Prénom :

Chaînes de caractères

On considère le programme complet suivant [1]:

```
#include <stdio.h>
#include <string.h>
FILE *file;
const int WORD_MAX_LENGTH=26;

int isPalindrome(char *word) {
    int len=strlen(word);
    for(int i=0;i<len/2;i++)
        if (word[i]!=word[len-i-1]) return 0;
    return 1;
}

int main() {
    char word[WORD_MAX_LENGTH];
    file=fopen("/home/colin/module/a43/dico_no_acccent.txt","ro");
    while (!feof(file)){
        fscanf(file, "%s",word);
        if (isPalindrome(word))
            printf("%s\t",word);
    }
    return 0;
}
```

- Que fait le programme ci-dessus, sachant que *dico_no_accent.txt* est un fichier texte contenant environ 330000 mots de la langue française (sans accents ni majuscules) et que l'instruction *fscanf(file, "%s",word)* effectue une lecture mot par mot de ce fichier.

Ce programme affiche les palindromes de la langue française en les séparant par une tabulation. *Extrait :* *(...) reifier ressasser retater rever rotor sagas salas (...)*

- en vous inspirant de la fonction *isPalyndrome* écrire la fonction *isWithoutE* qui retourne vraie si le mot passé ne contient aucune lettre *e*, et faux dans le cas contraire.

[1] Rappels :
*int strlen(char *s)* retourne le nombre de caractère de s
void strcpy(char *t, const char *s) recopie s dans t
Ex:
char buf[100);
strcpy(buf,"chien"); // chien est recopié dans buf

```
int isWithoutE(char *word) {
    // version avec double parcours du mot (strlen, puis la boucle)
    int len=strlen(word);
    for(int i=0;i<len;i++)
        if (word[i]=='e') return 0;
    return 1;
}

int isWithoutE(char *word) {
    // version avec simple parcours du mot par pointeur
    while(*word++)
        if (*word=='e') return 0;
    return 1;
}
```

- Modifier le programme pour qu'il affiche le plus long mot de la langue française sans *e* (la réponse est *constitutionnalisassions*...) en complétant les zones ci-dessous (vous supposerez que la question précédente a été correctement résolue)

```
... programme comme ci-dessus
// à compléter ci-dessous (pour les variables globales éventuelles)
int maxlen=0;
char wordMax[WORD_MAX_LENGTH]


int main() {
    char word[WORD_MAX_LENGTH];
    file=fopen("/home/colin/module/a43/dico_no_acccent.txt","ro");
    while (!feof(file)){
        fscanf(file, "%s",word);
        // à compléter ici (pour le code)
        if (isWithoutE(word)) {
            if (strlen(word)>=maxlen) {
                maxlen=strlen(word);
                strcpy(wordMax,word);
            }
        }
    }
    printf("%d : :%s\n",maxlen,wordMax);
    return 0;
}
```

- De même modifier le programme ci-dessus pour obtenir le plus long mot de la langue française comportant le plus de e (ex. de réponse : *degenerescence*...)

```
... programme comme ci-dessus
// à compléter ci-dessous (pour les variables globales éventuelles)
int ce=0,maxe=0
char wordMaxe[WORD_MAX_LENGTH]
int main() {
    char word[WORD_MAX_LENGTH];
    file=fopen("/home/colin/module/a43/dico_no_acccent.txt","ro");
    while (!feof(file)){
        fscanf(file, "%s",word);
        // à compléter ici (pour le code)
        for(i=0,ce=0;i<strlen(word);i++)
            if (word[i]=='e') ce++;
        if (ce>=maxe) {
             maxe=ce;
             strcpy(wordMaxe,word);
        }
    }
    printf("%d : :%s\n",maxe,wordMaxe);
    return 0;
}
```

Au *scrabble* les mots sont affectés d'un score calculé en sommant le poids de chaque lettre du mot. Par exemple le mot *chien* possède un score de 10 points correspondant à la somme du poids des lettres qui le constituent (3+4+1+1+1). La fonction ci-dessus permet de calculer le poids d'une lettre.

```
int coeff(char c) {
    if (c=='k' || c=='w' || c== 'x' || c=='y' || c=='z') return 10;
    if (c=='j' || c=='q' ) return 8;
    if (c=='f' || c=='h' || c== 'v' ) return 4;
    if (c=='b' || c=='c' || c== 'p' ) return 3;
    if (c=='d' || c=='m' || c== 'g' ) return 2;
    return 1;
}
```

- écrire la fonction qui calcule le score d'un mot en utilisant la fonction ci-dessus.

```
int score(char * s) {
    // version tableau
    int sum=0;
    for(int i=0;i<strlen(word);i++) sum+=coeff(word[i]);
    return sum;
}

int score(char * s) {
    // version pointeur
    int sum=0;
    while(*s++) sum+=coeff(*s);
    return sum;
}
```

En utilisant cette fonction score il est possible de déterminer le mot de la langue française présentant le plus gros score : il s'agit du mot *psychophysiologistes* (score de 58)

Dans le véritable jeu de scrabble, néanmoins, ce mot ne put être utilisé en raison du fait qu'on ne dispose que d'un seul jeton correspondant à la lettre *y*. Pour un scrabble en langue française la distribution des jetons suit la règle suivante :

```
int freq(char c) {
    if (c=='e') return 15;
```

```
    if (c=='a') return 9;
    if (c=='i') return 8;
    if (c=='n'||c=='o'||c=='r'||c=='s'||c=='t'||c=='u') return 6;
    if (c=='l') return 5;
    if (c=='d'||c=='m') return 3;
    if (c=='q'||c=='k'||c=='j'||c=='w'||c=='x'||c=='y'||c=='z') return 1;
    return 2;
}
```

- Écrire la portion de code qui affiche le mot de la langue française possédant le score le plus élevé et qui soit compatible avec les règles du scrabble français (il s'agit du mot *deshypothequassiez,* dont le score est 50). Vous pouvez vous appuyer sur les fonctions *score*, *freq* évoquées précédemment.
  Attention : cette question est plus difficile. Si vous n'avez pas le temps d'écrire le code vous pouvez vous contenter de décrire la démarche générale.

// réponse niveau 1 :

je suppose que je dispose d'une fonction qui teste si un mot est compatible avec le nombre de lettres proposé par le jeu.

je parcours les mots du dictionnaire, pour chacun d'entre eux je mesure le score. Si le mot est compatible et que le score est plus grand que le plus grand score rencontré je mémorise le mot.

Après la boucle j'affiche le dernier mot mémorisé

// reponse niveau 2 (implémetation de ci-dessus) :

```
int maxScore=0 ;
char wordMaxScore[WORD_MAX_LENGTH];

// score
int main() {
    char word[WORD_MAX_LENGTH];
    file=fopen("/home/colin/module/a43/dico_no_acccent.txt","ro");
    while (!feof(file)){
        fscanf(file, "%s",word);
        // à compléter ici (pour le code)
        if (score(word)>=maxScore && isScrabbleComp(word)) {
            maxScore=score(word);
            strcpy(wordMaxScore,word);
        }
    }
    printf("%d : :%s\n",maxe,word);

    return 0;
}
```

// reponse niveau 3:idem+implémentation de isScrabbleComp

```
// version avec tableau alphabetique des occurrences
int isScrabbleComp(char *word) {
    // le mot est-il scrabble compatible?
    int freqTab['z'-'a'+1];
    for(int i=0;i<'z'-'a'+1;i++) freqTab[i]=0;
    for(int i=0;i<strlen(word);i++) freqTab[word[i]-'a']++;
    for(int i=0;i<'z'-'a'+1;i++)
        if (freqTab[i]>freq(i+'a')) return 0;
    return 1;
}
```

```
// version sans tableau alphabetique des occurrences
int isScrabbleComp(char *word) {
    // le mot est-il scrabble compatible?
    char *p=word, *pbegin=word;
    int sum;
    while(*word++) {
        sum=0;p=pbegin;
        while (*p++) if (*p==*word) sum++;
        if (sum>freq(*word)) return 0;
    }
    return 1;
}
```

Réingénierie

Wikipedia donne cette définition d'un nombre parfait :
Un entier naturel est parfait s'il est la somme de ses diviseurs propres (c'est à dire tous ses diviseurs sauf lui-même).
Ainsi 6 est un nombre parfait car 6 = 1 + 2 + 3.
de même 28, 496 sont parfaits

- 28 = 1 + 2 + 4 + 7 + 14
- 496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248

24 n'est pas parfait car 24 != 1+2+3+4+6+8+12
Wikipédia donne sous la forme ci-dessous l'algorithme permettant de présenter les nombres parfaits inférieurs à 10000 :

```
Variable Somme, N, i entiers
Pour N dans 1..10000
  Somme ← 0
  Pour i dans 1..N-1
      Si mod(N;i)==0 alors
          Somme=Somme+i
      FinSi
  FinPour
  Si Somme==N alors
      Afficher N "est parfait"
  FinSi
FinPour
```

- Traduire ce pseudo-code en un programme C complet (l'opérateur modulo est *%*)

```
int main() {
    int sum,i,n;
    for (n=1; n<=10000; n++) {
        sum=0;
        for(i=1; i<n; i++)
            if (n%i==0) sum+=i;
        if (sum==n) printf("%d : parfait\n",n);
    }
    return 0;
}
```

- En déduire la fonction ***int isParfait (int)*** permettant de savoir si un nombre est parfait

```
int isParfait(int n) {
    int d,sum=0;
    for(d=1;d<n;d++)
        if (n%d==0) sum+=d;
    return sum==n;
}
```

- *Euclide* (3 siècles av. JC) et *Euler* (18 siècles ap. JC) ont montré qu'un nombre parfait peut toujours s'écrire sous la forme $2^{n-1}(2^n - 1)$. Par exemple $6=2^1(2^2 - 1)$. Écrire la fonction qui permet de déterminer la valeur de **n** correspondant à un nombre entier donné. Par exemple *getEuler(6)* vaut 2. Si cet entier n'existe pas la fonction retourne 0. (question difficile)

// idée générale : tester successivement les valeurs de $2^{i-1}(2^i - 1)$, en commençant à i=0, jusqu'à ce que la valeur obtenue soit supérieure ou égale à n.
Si elle est égale on retourne i, si elle est supérieure on retourne 0.
Le code ci-dessous implémente cela en utilisant la fonction *pow* pour l'élévation à la puissance.

```
int getEuler(int n) {
    int res=0,i=0;
    while (res<n) {
        res=pow(2,i-1) * (pow(2,i)-1);
        i++;
    }
    if (res==n) return 1;
    return 0;
}
```

Attention néanmoins : la signature de pow est double pow(double,double). Si on veut rester dans le domaine des entiers le code ci-dessous est plus prudent...

```
int getEuler(int n) {
    int u=1,v=1,i=0;
    while (u*v<=n) {
        if (u*v==n) return i+1;
        u=2*u;
        v=2*(v+1)-1;
        i++;
    }
    return 0;
}
```

Pour comprendre ce code on part de l'idée que $2^{i-1}(2^i - 1)$ peut être vue comme une suite : $w_i=u_i*v_i$ où $u_i= 2^{i-1}$ et $v_i= 2^i - 1$
En terme de suite u_i peut être définie ainsi :
$u_1= 1$ et
$u_i= 2*u_{i-1}$
De même en terme de suite v_i peut être définie ainsi :
$v_1= 1$ et
$v_i= 2*(v_{i-1} + 1)-1$
d'où l'algorithme ci-dessus
L'avantage de cette formulation est qu'on reste dans le domaine des entiers, et donc qu'on ne s'expose pas à des problèmes d'arrondis dans la comparaison u*v==n, et enfin que les seules opérations impliquées sont de simples multiplication ou addition (pour information : pow(x,y), c'est à dire x^y est calculée par $e^{y*\ln(x)}$)

- Question plus facile : utiliser la fonction précédente (qu'elle soit écrite ou non) pour afficher le résultat suivant (liste des entiers pour lesquels ce nombre ***n*** existe), c'est à dire

  1 -> 1
  6 -> 2
  28 -> 3
  120 -> 4
  496 -> 5
  2016 -> 6
  8128 -> 7

```
int main() {
   for(n=1;n<10000;n++)
        if (getEuler(n))
             printf("%d  -> %d\n",n,getEuler(n));
   return 0;
}
```