POLYTECH LILLE

Université Lille1 Sciences et Technologies

Département IMA / 3A (S5) **Programmation Structurée 2012/2013**

http://laure.gonnord.org/pro/teaching/

Examen - 2h

éléments de correction

A – Exercice : Programme Mystère

Question 1 L'appel dans le main est équivalent à `f(t,2,4)`. La fonction récursive `f` modifie le tableau t en remplaçant chacune des cases (indices 0 à $n-1$) par leur carré (de droite à gauche, par le jeu des appels récursifs). Les modifications sur t sont enregistrées, donc l'impression donne ensuite :

```
2
1
4
9
4
```

Attention à bien imprimer la valeur de n

B – Problème (Compression de fichiers BZIP)

Partie I - BZip (compression par redondance)

Question 1 Pour calculer le nombre d'occurrences on parcourt le tableau entièrement en incrémentant un compteur qu'on a pris soin d'initialiser à 0.

```
int nb_occ(int t[N], int el){
  int i;
  int cpt;
  for (i=0;i<N;i++)
    if (t[i] == el) cpt=cpt+1;

  return cpt;
}
```

La complexité est de N tests.

Sans problème pour la majorité. Attention à la syntaxe du for et du test en C

Question 2 Pour cette question, on commence par remplir r avec des 0, ensuite on parcourt le tableau t en incrémentant à chaque fois la case d'indice $t[i]$ ·

```
void occurrences(int t[N], int r[256]){
  int i;

  for (i=0;i<256;i++) r[i]=0;//init
```

```
  for (i=0;i<N;i++) { // parcourt de t.
    r[t[i]]=r[t[i]]+1;//incrément
  }
}
```

La complexité de cet algorithme est N affectations, et 0 tests.

Si on utilise la question précédente, il faut faire attention aux indices. Le tableau r à remplir est de taille 256, et il se parcourt avec une boucle "for". C'est bien r qu'il s'agit de parcourir, pas t dans ce cas

Beaucoup de solutions ne fonctionnent pas à cause d'un algorithme qui est un mélange savant des deux précédents. Expliquer son algorithme est le meilleur moyen d'avoir un algorithme juste.

Question 3 On va retourner l'entier à calculer (fonction). Ensuite il suffit de remplir r, puis de parcourir le tableau r et de s'arrêter lorsque $r[i] = 0$.

```
int minichar(int t[N]){
  int i;
  int r[256];
  occurrences(t,r);// appel à la fonction précédente

  // le tableau r est rempli, on veut i tq r[i]=0

  for (i=0;i<256;i++){
    if (r[i]==0)
      return i;
  }
  return -42;// si non trouvé
}
```

On peut aussi utiliser un appel à `nb_occ(t,i)` Faire attention aux indices. Le tableau r est de taille 256.

Question 4 Sans surprise, on parcourt les deux tableaux $t1$ et $t2$ et on retourne dès qu'on trouve $t1[i]$ différent de $t2[i]$:

```
bool sont_egaux(int t1[N], int t2[N]){
  int i;
  for (i =0; i<N ; i=i+1)
    if (t1[i] != t2[i]) return false;
  return true;
}
```

Question sans problème pour la plupart. Néanmoins certains oublient le cas général `return true`

Question 5 Le marqueur est 0. Le texte après compression est

$$\underbrace{0}_{marqueur},\underbrace{0,2,1}_{3fois1},\underbrace{2}_{1fois2},\underbrace{1}_{1fois1},\underbrace{5}_{1fois5},\underbrace{0,2,3}_{3fois3},\underbrace{0,2,2}_{3fois2}$$

Question 6 Pour compresser, le tableau initial est donné, mais il faut passer en paramètre le tableau *resu* : si il est local, on va avoir du mal à récupérer le texte compressé ! On n'oublie pas de retourner la taille, que l'on va construire au fur et à mesure en augmentant un compteur. En ce qui concerne l'algorithmique, voici les étapes :

– récupérer le marqueur en appelant la fonction `minichar`
– repérer les suites de nombres identiques et compter la taille de ces suites. Une fois qu'on a compté, on fait l'encodage en remplissant 1 ou 3 cases du tableau *resu*.
– On s'arrête lorsqu'on a parcouru le tableau t complétement.

Malgré les consignes de l'énoncé, beaucoup TROP d'algorithmes ne sont pas justifiés. Ils n'ont pas eu tous les points. De plus, le parcours du tableau à compresser n'est pas à incrément constant : i augmente quelquefois de 1, quelquefois de 3, on n'utilise donc pas de boucle for, mais une boucle while [1]

```
int compresse (int t[N], int resu[M]){
  int marqueur;
  int icourant;// indice courant dans resu.
  int i=0; // pour le parcours du tableau t.
  int cpt,nb;
  marqueur = minichar(t); // on a  le marqueur

  resu[0] = marqueur;
  icourant =  1;

  while (i < N){ // parcours de t : fini lorsque i=N-1
    nb = t[i];// nouveau nb
    cpt = 1; // compteur d'occurrences de nb
    i = i+1;
    while (i < N && t[i]==nb){
      cpt = cpt+1;
      i = i+1;
    }

    // a ce stade je sais que cpt contient le nombre d'occurrences de nb
    if (cpt == 1){
      resu[icourant]= nb;
      icourant = icourant +1;
    } else {
      resu[icourant] = marqueur;
      resu[icourant+1] = cpt -1;
      resu[icourant+2] = nb;
      icourant = icourant + 3;
    }
  }
  return icourant; // nb de caracteres apres compression
}
```

Question 7 Pour `décompresse`, l'algorithme est (un peu) plus simple. On récupère le marqueur. Ensuite, on parcourt le tableau compressé *comp* (avec i), en considérant 1 ou 3 cases à la fois, suivant la valeur de $comp[i]$:
– si $comp[i]$ n'est pas le marqueur cela signifie que $comp[i]$ apparaît dans *decomp* une seule fois.
– sinon, cela signifie que $comp[i+2]$ apparaît $comp[i+1]+1$ fois dans *decomp*.

De la même manière, un algo non justifié n'a pas la totalité des points.

1. en C, incrémenter i à l'intérieur d'une boucle `for(i=...)` est autorisé, mais c'est une atrocité algorithmique.

On obtient donc l'action suivante :

```
void decompresse (int comp[M], int decomp[N], int taille){
  int marqueur = comp[0];
  int i=1; //pour parcourir le tableau compressé comp .
  int id; //indice courant dans le tableau decomp
  id = 0;

  int nb,j,nbocc; // variables locales

  while (i<taille){
    if (comp[i] == marqueur){// considérer donc comp[i+1] et comp[i+2];
      nb = comp[i+2];
      nbocc = comp[i+1]+1;
      //on sait qu'on a nbocc fois le nombre nb
      for (j=0;j<nbocc;j++){ // on remplit decomp
         decomp[id+j] = nb;
      }
      id = id+nbocc;
      i = i+3;
    } else { // sinon 1 seul caractere a considérer
      decomp[id] = comp[i];
      id = id+1;
      i = i+1;
    }
    //dans les deux cas i augmente
  }
}
```

Question 8 Programme complet :

```
#include <stdio.h>
#include <stdbool.h>

#define N 12
#define M 42

int nb_occ (int t[N], int el){
...
}
...autres fonctions ...

int main(){

  int r[256];
  int t[N]={0,0,3,2,3,3,3,3,3,3,5,5};
  int comp[M];
  int texte2[N];

  int l = compresse(texte,comp);
  decompresse(comp,texte2,l);
```

```
  if ( sont_egaux(texte,texte2) )
    printf("oui, ils sont égaux!\n");
  else
    printf("non, ils ne sont pas égaux!\n");

  return 0;
}
```

Question facile à traiter même si vous n'avez écrit que les signatures des fonctions de compression et décompression.

Partie II - Algorithme "Move To Front"

Question 9 En remarquant que le caractère 'a' va à la case $0 =' a' - 97$, il vient de suite (on écrit une action avec *assoc* paramètre-résultat) :

Action *remplitAssoc(assoc)*
- **R**: assoc : Vecteur[26] de caractères
- **L**: i : Entier
- **Pour** *i* **de** *0* **à** *25* **Faire**
  - t[i] ← i+97
- **Fpour**

FAction

Attention aux indices.

Question 10 Le décalage n'est pas très difficile à concevoir non plus, l'algorithme est décrit dans l'énoncé (recherche de l'indice, puis décalage, puis insertion puis retour). On écrit une fonction en passant *assoc* en donnée/résultat et en retournant *i*

Fonction *decaleAssoc(assoc,ch)*
- **D/R**: assoc : Vecteur[26] de caractères
- **D**: ch : caractère
- **L**: i,j : Entiers
- i← 0
- **Tq** *(t[i]* $\neq$ *ch)* **faire**
  - i← i+1
- **Ftq**
- {i est l'indice d'apparition de ch}
- **Pour** *j* **de** *i-1* **à** *0* **Par pas de** *-1* **Faire**
  - {décalages}
  - t[j+1] ← t[j]
- **Fpour**
- t[0] ← ch {première case}
- **Retourner** *i*

FFonction

C'est le passage des paramètres qui a posé le plus de problèmes ici. Il convient de passer le tableau *assoc* en D/R, car ce tableau est modifié à chaque étape du codage. Algorithmiquement, il convient de ne pas mélanger le calcul de l'indice d'apparition (un parcours s'impose, le caractère ch n'étant pas forcément à l'indice $ch - 97$), et le décalage (qui est un décalage simple, pas un échange).

Question 11 Encodage de la chaîne *cool*. En calculant *soigneusement*, on trouve :

$$2, 14, 0, 12$$

Il y a une erreur de calcul dans l'énoncé pour le codage de la chaîne ima, sans conséquence puisqu'elle a été corrigée pendant l'épreuve.

Question 12 Pour l'encodage, on écrit une action avec un tableau contenant le texte, et un tableau qui va récupérer l'encodage du texte. Le tableau d'association sera local.

```
Action encode(t,resu)
    D: t : Vecteur[N] de caractères
    R: resu : Vecteur[N] de caractères
    L: assoc : Vecteur[26] de caractères
    L: nb,i : Entiers
    remplitAssoc(assoc)
    Pour i de 0 à N-1 Faire
        ch ← t[i]
        nb ← decaleAssoc(t,ch)
        resu[i] ← nb
    Fpour
FAction
```

Pour une action d'encodage, seuls les tableaux d'entrée (texte à coder) et sortie (texte résultat) doivent être pasés en paramètre. Le tableau d'assoc est local.