DEUG MIAS

Module S2
juin 2002

Examen d'Informatique

Dans ce problème, `N` désigne une constante fixée.

Question 1 (2 points)

Ecrire une fonction `void lireEntier(int *n)` qui place à l'adresse `n` une valeur entière positive ou nulle `< N`. La valeur sera lue au clavier en recommençant jusquà ce qu'elle soit correcte.

```
void lireEntier(int *n){
 do{
  printf("entrez un entier entre 0 et %d :",N-1);
  scanf("%d",n);
 }while(*n<0 || *n>=N);
}
```

Question 2 (2 points)

Ecrire une fonction `void lireTableau(int a[])` qui lit au clavier les valeurs des éléments d'un tableau `a` de taille `N`. Le tableau ne doit contenir que des valeurs comprises entre 0 et $N-1$.

```
void lireTableau(int a[]){
 int i;
 for(i=0;i<N;i++)
  lireEntier(&a[i]);
}
```

Question 3 (2 points)

Ecrire une fonction `void contenu(int a[], int temoin[])` qui prend en argument deux tableaux `a` et `temoin` de taille `N`. Les valeurs de `a` sont comprises entre 0 et $N-1$. Elle remplit le tableau `temoin` de façon que `temoin[j]` est égal au nombre de fois que la valeur `j` aparaît dans le tableau `a`.

```
int contenu(int a[], int temoin[]){
  int i;
  for(i=0;i<N;i++) temoin[i]=0;
  for(i=0;i<N;i++) temoin[a[i]]++;
}
```

On appelle *permutation* un tableau de taille `N` contenant chaque entier i avec $0 \leq i <$`N` (chacun de ces entiers aparaîtra alors une seule fois).

Question 4 (2 points)

Ecrire une fonction `int estUnePermutation(int a[])` qui renvoie 1 si le tableau `a` de taille `N` est une permutation et 0 sinon.

```
int estUnePermutation(int a[]){
  int i;
  int b[N];
  contenu(a,b);
  for(i=0;i<N;i++){
    if(b[i]==0) return 0;
  }
  return 1;
}
```

Question 5 (2 points)

Si `a` et `b` sont deux permutations, leur produit est la permutation `c` définie par `c[i]=a[b[i]]`. Ecrire une fonction
`void produit(int a[], int b[], int c[])` qui calcule dans `c` le produit de `a` et `b`.

```
void produit(int a[], int b[], int c[]){
  int i;
  for(i=0;i<N;i++)
    c[i]=a[b[i]];
}
```

Question 6 (2 points)

Si `a` est une permutation et `k` un entier, la puissance `k`-ième de `a` est définie comme le produit `a...a` (`k` fois). Ecrire une fonction
`void puissance(int a[], int k, int b[])` qui calcule la puissance `k`-ième de `a` dans `b`.

```
void puissance(int a[], int k, int b[]){
  int i;
  for(i=0;i<N;i++)
    b[i]=i;
  for(i=0;i<k;i++)
    produit(a,b,b);
}
```

Question 7 (2 points)

L'inverse d'une permutation `a` est la permutation `b` telle que `b[a[i]]=i` pour tout indice `i`. Ecrire une fonction `int estInverse(int a[], int b[])` qui renvoie 1 si `b` est l'inverse de la permutation `a`.

```
int estInverse(int a[], int b[]){
  int i;
  for(i=0;i<N;i++)
    if(b[a[i]]!=i) return 0;
  return 1;
}
```

Une *inversion* d'une permutation `a` est une paire (i, j) d'indices telle que $i < j$ mais $a[i] > a[j]$.

Question 8 (2 points)

Ecrire une fonction `int NbInversions(int * a)` qui rend le nombre d'inversions de la permutation `a`.

```
int NbInversions(int a[]){
  int inv=0,i,j;
  for(i=0;i<N;i++)
    for(j=i+1;j<N;j++)
      if(a[i]>a[j]) inv++;
  return inv;
}
```

La *table des inversions* d'une permutation `a` est le tableau de même taille `Inv` défini par `Inv[x]=y` s'il existe `y` paires `(i,j)` avec `i<j` telles que `a[i]=x` et `x>a[j]`.

Question 9 (2 points)

Ecrire une fonction `void TableInversions(int a[], int b[])` qui calcule dans `b` la table des inversions de la permutation `a`.

```
void TableInversions(int a[], int b[]){
  int i,j;
  for(i=0;i<N;i++)
    b[i]=0;
  for(i=0;i<n;i++)
    for(j=i+1;j<N;j++)
      if(a[i]>a[j]) b[a[i]]++;
}
```

Question 10 (2 points)

Ecrire une fonction `void retrouver(int a[], int b[])` qui place dans `b` la permutation dont `a` est la table des inversions.

```
void retrouver(int a[], int b[]){
  int x,i,j;
  for(x=0;x<N;x++){  //placer x
    i=x-a[x];      //i=place de x
    for(j=x;j>i;j--)
      b[j]=b[j-1];   //decaler
    b[i]=x;
  }
}
```