9. Quelques exercices d'examen

I. Elagage

On dispose d'un tableau a[N] dont chaque case contient un nombre. Certains de ces nombres sont des 0. Sans utiliser d'autre tableau (mais en désespoir de cause prenez un autre tableau), faire le programme qui permet d'avoir dans a[N] tous les nombres du tableau a[N] initial sauf ses 0. Par exemple, on veut passer du tableau a[N] = 0 0 1 2 0 3 0 0 0 4 4 0 0 5 , avec N=14, au même tableau a[N] devenu 1 2 3 4 4 5. Le programme devra aussi donner la longueur occupée par les nombres (autres que 0) dans le tableau final.

On se donne N *et l'on remplit le tableau* a[N]
k=0 ; /* k *est une variable qui court, et qui dans ce qui suit sera toujours inférieure ou égale à* i */
for(i=0 ;i<N ;i++) if (a[i] !=0) a[k++]=a[i] ;
longueuroccupee=k ;

II. Tri par comptage

N nombres tous différents sont placés dans un tableau a[N] (cases numérotées de 0 à N-1). Pour chaque élément a[i] de ce tableau, on compte le nombre d'éléments du tableau qui lui sont inférieurs, et l'on place le résultat dans la case numéro i d'un tableau b[N]. Par exemple pour N=6, avec le tableau a[6] : 7 3 9 2 5 4, on trouve b[6] : 4 1 5 0 3 2.

1) Faire le programme qui fabrique le tableau b[N] à partir du tableau a[N]. On ne pourra pas échapper à une double boucle.

2) On va maintenant utiliser le tableau b[N] pour trier les nombres du tableau a[N]. Par un seul parcours du tableau b[N], et en utilisant aussi le tableau a[N], on remplit un nouveau tableau c[N] où les nombres de a[N] vont se trouver dans l'ordre croissant. Programmer.
Par exemple *b : 4 1 5 0 3 2*
a : 7 3 9 2 5 4 $\rightarrow$ *c : 2 3 4 5 7 9*

1) *Se donner* N *et remplir le tableau* a[N]
mettre à 0 *le tableau* b[N]
for(i=0 ;i<N ; i++) for(j=0 ; j<N ; j++) if (i !=j)
if (a[j] < a[i]) b[i]++ ;

Ou bien pour gagner un peu de temps, on compare les paires au lieu de comparer les couples comme précédemment :

for(i=0 ; i<N ; i++) for(j=i+1 ; j<N ; j++)
if (a[j]<a[i]) b[i]++ ; else b[j]++ ;

2) for(i=0 ; i<N ; i++) c[b[i]]=a[i] ;

III. Généralisation du problème de Josephus Flavius

N nombres 1, 2, 3,…, N sont disposés en cercle. On les enlève un par un suivant le procédé suivant : On part de 1, qu'on enlève, et on avance d'un cran. On tombe sur 2 que l'on enlève, et l'on avance de deux crans (correspondant au nombre que l'on vient de supprimer). On tombe sur 4, etc. A

chaque étape, après avoir enlevé le nombre k, on avance de k crans sur le cercle. A la fin, il ne reste qu'un seul nombre et l'on s'arrête. Par exemple pour N=9, les éléments supprimés sont 1,2,4,8,5,6,3,7, et il reste le 9. On veut programmer cela pour afficher le nombre final, et l'on utilise pour cela une liste chaînée doublement circulaire.

1) Définir la cellule de base.

2) Fabriquer l'embryon de la liste avec un élément pointé par debut, puis faire des insertions successives pour fabriquer la liste complète des N cellules.

3) Faire la boucle du programme où à chaque fois est enlevée une cellule. On pourra appeler avant et apres les pointeurs placés de part et d'autre de la cellule aenlever.

1) struct cell { int n ; struct cell * s ; struct cell * p ;} ;

2) *se donner N*
```
   debut= (struct cell *) malloc (sizeof(struct cell)) ;
   debut->n=1 ; debut->s=debut ; debut->p= debut ;
```

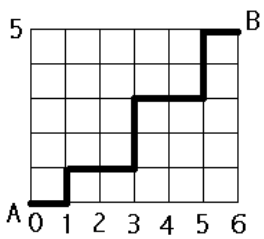
3) oldcell=debut ;
```
  for(i=2 ;i<=N ; i++)    /* on insère la nouvelle cellule entre oldcell et debut */
   { newcell= (struct cell *) malloc(sizeof(struct cell)) ;
    newcell->n=i ; newcell->s=debut ; newcell->p=oldcell ;
    oldcell->s=newcell ; debut->p=newcell ;
    oldcell=newcell ;
   }
```
Affichage pour vérifier que tout va bien :
```
ptr=debut ; do { printf(«%d »,ptr->n) ; ptr=ptr->s ;} while (ptr !=debut) ;
```

4) On va placer un pointeur courant sur *debut*, et celui-ci ne va cesser de tourner dans la liste jusqu'à ce qu'il ne reste qu'un nombre. Si en cours de route, la cellule *debut* est supprimée (elle est d'ailleurs supprimée tout de suite !), cela n'a pas d'importance, car on ne sort jamais de la liste.

```
ptr=debut ;  /* le pointeur qui court, pour se placer sur l'élément à éliminer */
avant=debut->p ; aenlever=debut ; apres=debut->s ;  /* aenlever entre avant et apres */
while(aenlever !=apres)
  { avant->s=apres ; apres->p=avant ;  /* on enlève */
    nombrepas=ptr->n ; ptr->s=NULL ; ptr->p= NULL ;
    ptr=apres ; for(i=1 ;i<nombrepas ; i++) ptr=ptr->s,
   aenlever=ptr ; avant=ptr->p ; apres=ptr->s ;  /* on prépare la prochaine suppression */
  }
afficher le survivant aenlever->n ;
```

Remarque : si l'on ajoute une grande boucle permettant d'afficher le survivant *S*(*N*) pour les valeurs successives de *N*, des milliers par exemple, il risque d'y avoir une saturation en mémoire à cause des *malloc*. Il convient d'ajouter dans le programme *free*(*ptr*) après avoir effectué une suppression (juste après *ptr->p=NULL*).

IV. Achats de produits parmi plusieurs types



1) *Dans un quadrillage, on va d'un point A à un point B, ces deux points étant séparés par n=6 pas horizontaux et p=5 pas verticaux. Les plus courts chemins menant de A à B sont tous à base de pas → ou ↑. On veut savoir combien il y a de plus courts chemins de A à B. Pour trouver ce résultat on pourra coder chaque chemin par un mot en binaire avec 0 pour indiquer un pas → , et 1 pour indiquer un pas ↑. Précisez les caractéristiques de ces mots. Finalement il y a autant de chemins que de tels mots en binaire. En déduire leur nombre.*

2) Un magasin dispose de 7 types de produits, disponibles chacun en grandes quantités (par exemple, des chemises, des manteaux, des chaussures, …). Pour simplifier on note ces catégories de produits de 0 à 6. Une personne achète 5 produits parmi ces 7 types (évidemment elle peut prendre plusieurs produits du même type, ou aucun). En faisant le lien avec la question précédente, que signifie le chemin indiqué sur le dessin ci-dessus, autrement dit qu'est-ce que la personne va acheter dans ce cas? En déduire combien il y a de façons de prendre 5 produits parmi les 7 types.

3) Généraliser : combien y a-t-il de façons de prendre P produits parmi N catégories d'objets ? Cela s'appelle des combinaisons avec répétition, à la différence des combinaisons vues en cours, où les répétitions ne sont pas permises.

4) Programmation (indépendant de ce qui précède)

4a) Prenons un exemple avec P=5 et N=3 : la personne prend P=5 objets parmi N=3 types d'objets (ces types étant notés 0, 1, 2). On veut énumérer toutes les façons de faire cela. Pour cela on utilise des mots à base de N=3 lettres, de longueur P=5. Les mots vont être classés dans l'ordre alphabétique, et à l'intérieur de chaque mot, on met les 0 avant les 1 et les 1 avant les 2 (ordre alphabétique interne). Cela donne dans le cas présent : 00000, 00001, 00002, 00011, 00012, 00022, 00111, 00112, 00122, 00222, 01111, 01112, 01122, 01222, 02222, 11111, 11112, 11122, 11222, 12222, 22222, soit 21 mots. Par exemple le mot 01112 signifie qu'on a pris un objet du type 0, trois objets du type 1 et un objet du type 2. Comment fait-on pour passer d'un mot au suivant ? Déterminer avant tout le pivot, c'est-à-dire la première lettre qui change derrière un bloc fixe le plus long possible. Par exemple 02222 a pour pivot 0, et son successeur est 11111. Faire le programme qui énumère tous ces mots dans le cas général (N et P étant donnés quelconques).

4b) Maintenant un mot comme 01112 est remplacé par 131 pour indiquer que l'on a pris un objet du type 0, trois objets du type 1 et un objet du type 2. Les mots obtenus par le programme précédent pour P=5 et N=3 deviennent 500, 410,401, 320, 311, … 005. Faire le programme qui effectue cette conversion.

1) Le codage des chemins donne des mots de longueur 11 formés de six 0 et cinq 1. Il s'agit de remplir un casier de 11 cases. On commence par placer les six 0. Le nombre de façons de le faire est de choisir six cases parmi onze, soit C_{11}^{6} cas. Les 1 prennent les cases restantes. D'où le nombre de chemins : $C_{11}^{6} = C_{11}^{5} = 11.10.9.8.7/(5.4.3.2) = 462$.

2) Le chemin indique que l'on a pris un objet du type 1, deux du type 3, deux du type 5, et aucun des types 0, 2, 4 et 6. Il existe autant de chemins qu'il y a de façons de prendre 5 objets parmi 7 types d'objets, soit 462.

3) Ce nombre est C_{N+P-1}^{P} (ou C_{N+P-1}^{N-1}).

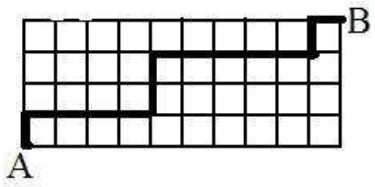
4) Le pivot est la première lettre obtenue en partant de la fin et en allant vers la gauche, qui n'est pas un 2 lorsque N=3, et qui n'est pas égale à N-1 dans le cas général. Une fois obtenu, on l'augmente de 1 et on met derrière cette même lettre jusqu'à la fin. D'où le programme :

```
on se donne N et P
for(i=0 ;i<P ; i++) a[i]=0 ;   /* le premier mot */
for( ;;)
  { afficher le tableau a[]
    i=P-1 ;    while (i>=0 && a[i]==P-1) i-- ;
    pospivot=i ; if (pospivot==-1) break ;
    pivot=a[pospivot]++ ;
   for(i=pospivot+1 ; i<P ; i++) a[i]=pivot ;
  }
```

4bis) A partir des mots précédents placés dans le tableau $a[P]$, on effectue leur conversion en mettant les nouveaux mots dans le tableau $b[N]$:

```
mettre le tableau b[N] à 0
for(i=0 ;i<P ;i++)  b[a[i]]++ ;
```

IVbis. Distribution de prospectus dans des boîtes aux lettres



1) *Dans un quadrillage, on va d'un point A à un point B, ces deux points étant séparés par n=10 pas horizontaux et p=4 pas verticaux. Les plus courts chemins menant de A à B sont tous à base de pas → ou ↑. On veut savoir combien il y a de plus courts chemins de A à B. Pour trouver ce résultat on pourra coder chaque chemin par un mot en binaire avec 0 pour indiquer un pas → , et 1 pour indiquer un pas ↑. Précisez les caractéristiques de ces mots. Finalement il y a autant de chemins que de tels mots en binaire. En déduire leur nombre.*

Cela fait : $C_{14}{}^{4}$ =14.13.12.11/(4.3.2) = 1001.

2) *Une personne distribue quatre prospectus tous identiques dans 11 boîtes aux lettres numérotées de 0 à 10. Une boîte peut contenir soit aucun prospectus, soit un nombre quelconque de prospectus (évidemment inférieur ou égal à 4). En faisant le lien avec la question précédente, que signifie le chemin indiqué sur le dessin ci-dessus, autrement dit quelle est la distribution de prospectus correspondante ? En déduire le nombre de façons de distribuer les 4 prospectus dans les 11 boîtes aux lettres.*

La personne place un prospectus dans la boîte 0, 2 dans la boîte 4, 1 dans la boîte 9, et aucun ailleurs. Le nombre de distributions possibles de prospectus est égale au nombre de chemins, soit 1001.

3) *Généraliser : combien y a-t-il de façons de distribuer P prospectus dans N boîtes aux lettres ?*

$C_{N+P-1}{}^{P}$ (ou $C_{N+P-1}{}^{N-1}$).

4) *Programmation (voir exercice précédent*)

V. Mélange de cartes par la méthode de Gaspard Monge

On a en main N cartes, avec N supposé pair. Pour les mélanger, on fabrique un nouveau paquet de la façon suivante : on prend la première carte du paquet initial, puis on met la deuxième au-dessus, puis on met la troisième en-dessous, puis la quatrième au-dessus, et ainsi de suite en alternance dessus-dessous dans le paquet en cours de fabrication. Par exemple, le paquet 3 7 9 5 4 6 (avec N=6) devient 6 5 7 3 9 4 : on a pris le 3 puis on a mis le 7 au-dessus (ici à gauche), puis le 9 en-dessous (ici à droite).

a) En mettant le paquet initial dans un tableau a[N], avec ses cases indexées de 0 à N-1, et en utilisant un autre tableau b[N] pour mettre le paquet après mélange, programmer cette opération de mélange. Commencer par indiquer en quelle position (en fonction de N) va la première carte du paquet initial dans le paquet après mélange.

b) On veut savoir, pour N donné, combien de fois il faut répéter cette opération de mélange pour retrouver l'ordre initial (on aura intérêt à partir du paquet 0 1 2 … N-1). Cela s'appelle la période T de la permutation. Programmer.

a) *Remplir le tableau* a[N]

```
b[N/2]=a[0];     /* la première carte va en position N/2, et même si N est impair */
g=N/2-1;  d=N/2+1;
for(i=1; i<N; i+=2)   {b[g--]=a[i]; b[d++]=a[i+1]; }
```

b)

```
for(i=0;i<N;i++) a[i]=i;    /* remplissage de a[] dans l'ordre naturel */
```

```
T=0;
do {  T++;
      remettre le programme précédent
      for(i=0; i<N; i++)  a[i]=b[i];   /* on met b[] dans a[]  */
      fini=OUI;   for(i=0;i<N;i++) if (a[i] !=i)  {fini=NON;: break;}
    }
while (fini==NON);
afficher T
```

Questions subsidiaires :

c) Montrer que la période T obtenue pour N pair ne change pas quand on prend N+1 cartes (nombre impair).

On constate que pour N+1 impair, la dernière carte en position N reste toujours en position N.

d) Pour N pair, indiquer les formules qui permettent à une carte de passer de sa position à sa nouvelle position.

Une carte en position paire p va en position $(N+p)/2$. Une carte en position impaire i va en position $(N-i-1)/2$.

e) Combien vaut la période T pour $N=2^P$? On admettra que cette période est aussi celle de la carte en position 0.

Voici le mouvement de la carte en position 0:
$0 \rightarrow N/2 = 2^{P-1} \rightarrow N/2+N/4 = 2^{P-1}+2^{P-2} = 3.2^{P-2} \rightarrow 2^{P-1} + 3.2^{P-3} = 7.2^{P-3} \rightarrow \ldots \rightarrow 2^{P-1}-1 \rightarrow 0$, d'où $T = P+1$.

VI. Evolution d'un mot

Un mot à base des trois lettres 0, 1 et 2 subit l'évolution suivante. Au début (à l'étape 0) le mot est réduit à 0. Puis à chaque étape de l'évolution, 0 est transformé en 012 , 1 est transformé en 2, et 2 est transformé en 0. Cela donne :

$0 \rightarrow 012 \rightarrow 01220 \rightarrow 012200012 \rightarrow \ldots$ On veut connaître le mot à l'étape N, N étant donné. Pour la programmation, on utilise une liste doublement chaînée circulaire, dont chaque cellule contient une lettre (0 ou 1ou 2). Faire le programme pour voir l'évolution du mot à chaque étape.

```
Déclaration de la cellule : struct cell {int n ; struct cell* s ; struct cell* p ;} ;
On se donne le nombre d'étapes N
debut= (struct cell *) malloc (sizeof(struct cell)) ; /* embryon de la liste*/
debut->n=1 ; debut->s=debut ; debut->p= debut ;
afficher : étape 0 :  0
for(etape=1 ; etape<=N ; etape++)  /* boucle des étapes */
  { ptr=debut ;  /* le pointeur courant ptr va faire  un tour complet à chaque étape */
    do
      { if (ptr->n==1)  {ptr->n=2 ; ptr=ptr->s ;}
        else if (ptr->n==2) {ptr->n=0 ; ptr=ptr->s ;}
        else {  apres=ptr->s ;  /* cas où ptr->n=0. On rajoute deux cellules entre ptr
                                   et apres */
             cell1=(struct cell *) malloc (sizeof(struct cell)) ;
             cell2=(struct cell *) malloc (sizeof(struct cell)) ;
             cell1->n=1 ; cell1->s=cell2 ; cell2->p=ptr ;
             cell2->n=2 ; cell2->s=apres ; cell2->p=cell1;
             ptr->s=cell1 ; apres->p=cell2 ;
             ptr=apres ;
            }
      }
     while(ptr !=debut) ;
```

```
  faire le tour de la liste pour afficher le mot
}
```

VII. Enumération de mots

On définit des mots de longueur N selon les règles suivantes : la lettre (le nombre) en position 0 est toujours 0, la lettre en position 1 est 0 ou 1, la lettre en position 2 est 0, ou 1 ou 2,…., la lettre en position N-1 est 0, ou 1, ou 2, ou 3, …, ou N-1. Autrement dit, la lettre d'indice i est un nombre inférieur ou égal à i. On veut programmer l'écriture de tous ces mots pour N donné, dans l'ordre alphabétique. Par exemple, pour N=3, on obtient 000, 001, 002, 010, 011, 012.

1) Quel est le nombre de ces mots pour N donné quelconque ?

2) Pour N=5, quel est le mot qui suit 01134 ?

3) Pour le programme final, on met en conditions initiales le mot 000…0. Puis on fera la boucle des étapes, permettant de passer d'un mot au suivant. A chaque passage, il s'agit de déterminer le pivot, c'est-à-dire la première lettre qu'il faut modifier, avec tout ce qui est à sa gauche qui reste fixe. Par exemple 0 1 1 2 4 5 6 admet pour pivot 2. Comment trouver le pivot ? En quoi celui-ci est-il changé ? Que met-on derrière lui ? Faire le programme, avec son test d'arrêt.

1) Une seule possibilité pour la lettre en position 0. A chaque fois deux possibilités pour la lettre en position 1. A chaque fois trois cas pour la carte en position 2, etc., et à chaque fois *N* cas pour la carte en position *N*-1. Le nombre de mots est égal à *N*!.

Remarque : à cause de *N*! il y a un lien étroit entre ces mots et les permutations de *N* objets.

2) Le mot qui suit 01134 est 01200.

3) Le pivot est la lettre la plus loin possible telle que $a[i]<i$, car celles d'après, s'il y en a, vérifient $a[i]=i$, ce qui les empêche d'augmenter. Pour avoir le pivot on part de la droite et on recule tant qu'on tombe sur $a[i]=i$. Une fois la lettre pivot trouvée, on l'augmente de 1, puis on met derrière le mot le plus petit possible, c'est-à-dire des 0. D'où le programme.

```
for(i=0 ;i<N ;i++) a[i]=0 ;  /* le premier mot */
for(;;)
  { afficher a[N]
    i=N-1 ;   while (i>=0 && a[i]==i) i-- ;
    pospivot=i ; if (pospivot==-1) break ;
    a[pospivot]++ ;
    for(i=pospivot+1 ; i<N ;i++) a[i]=0 ;
  }
```

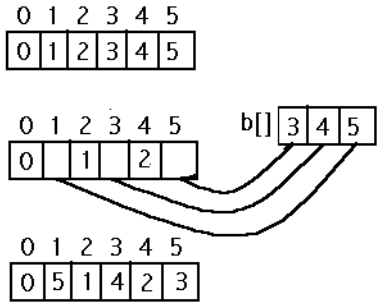
VIII. Mélange de cartes en peigne

On a en main un paquet de N cartes, avec N supposé pair. On le coupe en deux, puis on insère le deuxième morceau dans le premier avec avoir retourné ce deuxième morceau, le première carte du deuxième morceau devenant la dernière carte du paquet, comme dans l'exemple suivant :

0 1 2 3 4 5 $\rightarrow$ 0 5 1 4 2 3 ou encore 0 5 1 4 2 3 $\rightarrow$ 0 3 5 2 1 4

1) On place les cartes, notées par exemple 0, 1, 2, 3, …, dans les cases numérotées 0, 1, 2, 3,… d'un tableau a[N]. Programmer de façon à obtenir le tableau a après l'opération de mélange. Pour cela :

- *commencer par couper en deux le paquet, en mettant le deuxième morceau dans un tableau auxiliaire b.*
- *puis écarter les cartes du premier morceau.*
- *enfin insérer dans les places vides les cartes du tableau b à l'envers*



2) *On répète cette opération de mélange jusqu'à retrouver l'ordre initial des cartes. Programmer pour obtenir le nombre de coups (la période T du mélange) qu'il faut faire, cette période étant fonction de N.*

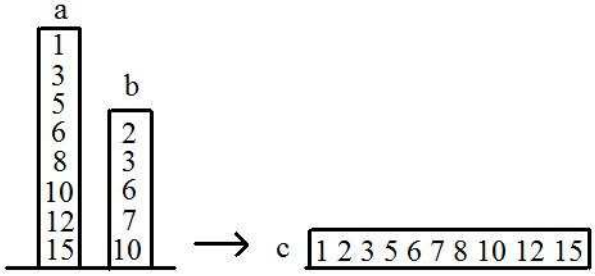
```
1)  mettre i dans a[i] de 0 à N-1
k=0 ; for(i=N/2 ;i<N ;i++) b[k++]=a[i] ;
for(i=N/2-1 ;i>=0 ;i--) a[2*i]=a[i] ;
k=N-1 ; for(i=0 ;i<N/2 ;i++)  {a[k]=b[i] ; k -=2 ;}

2)  T=0 ;
do  { T++ ;
       placer ici le programme précédent
       fini=0UI,
       for(i=0 ; i<N ; i++)  if (a[i] !=i )  { fini=NON ; break ;}
    }
while (fini==NON) ;
afficher T
```

ou bien en utilisant une pseudo-fonction *fini*() ramenant OUI ou NON (1 ou 0):

```
T=0,  do { T++ ; programme précédent }
        while (fini() == NON) ;
int fini(void)
  { for(i=0 ;i<N ;i++) if (a[i] !=i) return NON ;
    return OUI ;
  }
```

IX. Union et intersection de deux ensembles de nombres

1) *On a deux ensembles formés de nombres entiers naturels. Le premier ensemble a N1 nombres tous différents placés dans le tableau a[N1], le deuxième possède N2 nombres placés dans le tableau b[N2]. On suppose dans cette question que les nombres sont triés (en ordre croissant) dans chacun des deux ensembles. Il s'agit de déterminer l'union de ces deux ensembles qui ont été ordonnés, formée de tous les nombres qui sont dans a[] OU dans b[], c'est-à-dire les nombres présents dans chacun des deux ensembles (en évitant les répétitions de nombres identiques). L'union de ces deux ensembles sera placée dans un tableau c[], où les nombres sont aussi triés. Par exemple, avec a[8] et b[5] ci-dessous, on obtient le tableau c[] :*



Faire le programme, en utilisant l'algorithme des têtes coupées (comme on l'a vu en cours pour la fusion de deux listes triées, et en exercice pour l'intersection de deux ensembles, cf. examen 2008 dans le chapitre 9-b) suivant).

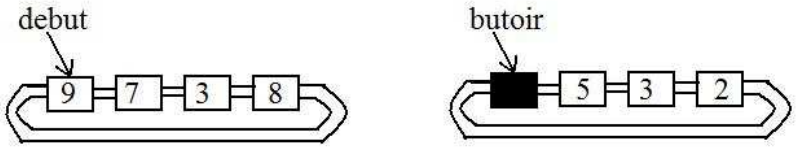
On suppose remplis les tableaux *a*[*N*1] et *b*[*N*2] qui sont triés. On appelle la fonction *union2*() pour avoir leur union dans le tableau *c*[].

```
void union2(void)   /* ne pas l'appeler union qui est un mot réservé du langage C */
{  int c[1000]; int i,j,k,longueurc;
   i=0; j=0; k=0;
   while(!(i==N1 || j==N2))
    if (a[i]==b[j]) { c[k++]=a[i]; i++; j++;}
    else if (a[i]<b[j]) {c[k++]=a[i++]; }
    else c[k++]=b[j++];

    if (i==N1) while(j<N2) c[k++]=b[j++];
    else while(i<N1) c[k++]=a[i++];

    longueurc=k;     for(k=0;k<longueurc; k++) printf("%d ",c[k]);
}
```

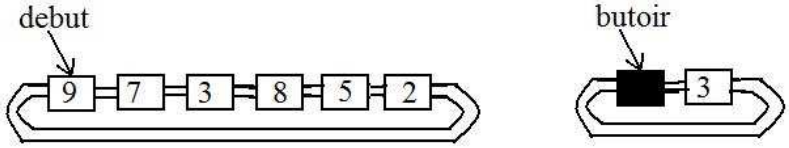
2) *On a toujours deux ensembles, avec les éléments du premier tous différents entre eux, et ceux du deuxième aussi tous différents entre eux, mais maintenant leurs éléments ne sont pas triés, un ensemble étant par définition non ordonné. On veut déterminer leur union ainsi que leur intersection. Pour cela, on va utiliser deux listes chaînées doublement circulaires, l'une où se trouvent les N1 nombres du premier ensemble, et l'autre où se trouvent les N2 nombres du deuxième. L'objectif du programme est de transformer ces deux ensembles de façon qu'à la fin le premier contienne leur union, et le deuxième leur intersection.*

*2-a) Construire les deux listes chaînées représentant les deux ensembles, avec des nombres au hasard (évidemment sans répétitions). On suppose définie la structure de la cellule, de la forme struct cell { int n ; struct cell * s ; struct cell * p ;} ;*

2-b) Voici un exemple, où la première liste a N1=4 éléments et la deuxième N2=3 éléments, en notant que le moyen d'accès à la première liste est la cellule debut, et que la deuxième liste a une case supplémentaire butoir factice, indispensable car c'est dans cette liste que vont être faites des suppressions de cellules, et que malgré ces suppressions, la cellule butoir, seul moyen d'accès à la deuxième liste, restera présente :



Il s'agit de faire le programme qui à partir de ces deux listes déjà construites donne à la fin les deux listes transformées, la première contenant l'union et la deuxième l'intersection :



Comment faire ? Un pointeur ptr2 va parcourir la deuxième liste à partir de butoir->s jusqu'au retour à butoir. A chaque fois que ptr2 pointe sur une cellule, on fait un tour complet dans la première liste avec un pointeur ptr1 de debut jusqu'au retour à debut.

Si pour un ptr2 donné, on tombe sur un ptr1 avec les mêmes nombres dans les cellules correspondantes, les deux cellules concernées restent chacune dans leur liste, il n'y a donc rien d'important à faire, sinon de mettre un drapeau flag à 1, on pourra même faire un break pour arrêter de tourner dans la boucle de la première liste, il restera seulement à avancer ptr2 d'un cran pour préparer l'étape suivante. Par contre, si lors du tour complet de la première boucle on ne tombe jamais avec un ptr2->n égal à ptr1->n, le flag étant donc resté à 0, on devra ajouter une cellule dans la première liste, contenant ptr2->n, en la plaçant par exemple à la fin de la première liste, puis on devra supprimer la cellule où pointe ptr2 dans la deuxième liste. Faire ce programme.

On trouvera ci-dessous le programme quasiment complet. Pour répondre à la question 2-a), on commence par faire une fonction *tableauhasard()* qui remplit un tableau *aux*[] de *N* nombres avec des nombres au hasard tous différents et compris entre 0 et $K - 1$ (voir à la fin du programme, évidemment il convient de prendre *K* au moins égal à *N*, et plutôt nettement plus grand). Cela permet de remplir le tableau *a*[] correspondant au premier ensemble, puis le tableau *b*[] du second. A partir de là con construit l'une après l'autre les deux listes chaînées associées aux deux ensembles. Cela constitue la première partie du programme ci-dessous. Puis on lance la grande boucle avec un pointeur *ptr2* qui va parcourir la deuxième liste. A chaque position de *ptr2*, on parcourt la première liste avec un pointeur *ptr1*. Si l'on tombe sur deux nombres égaux, ils restent chacun dans leur liste, on se contente de mettre *flag* à 1 (*flag* ayant été mis à 0 au début de chaque parcours de la première liste). Par contre si l'on n'a pas deux nombres égaux, on insère celui qui était dans la deuxième liste dans la première, celle de l'union, et on le supprime de la deuxième liste, celle de l'intersection.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define OUI 1
#define NON 0
void tableauhasard(int N, int K);
void union2(void); void intersection(void);
int aux[1000],a[1000],b[1000];
int N1, N2;
struct cell {int n; struct cell *s; struct cell* p;};
struct cell * debut, * newcell, * avant,* ptr,* butoir, *ptr1, *ptr2, *avant1, *apres2, *avant2;

int main()
{  int i, flag;
   srand(time(NULL));

/******* PREMIERE PARTIE DU PROGRAMME *******/
   N1=10;  tableauhasard(N1,2*N1); for(i=0;i<N1;i++) a[i]=aux[i];
   debut=(struct cell *)malloc(sizeof(struct cell));
   debut->n=a[0]; debut->s=debut; debut->p=debut;  /* embryon de la liste */
   avant=debut;
   for(i=1;i<N1;i++)  /* insertions successives */
     {  newcell=(struct cell *)malloc(sizeof(struct cell));
        newcell->n=a[i]; newcell->s=debut; newcell->p= avant;
        avant->s=newcell; debut->p= newcell;
        avant=newcell;
     }
   afficher la liste chaînée correspondant au premier ensemble

   N2=6;
   tableauhasard(N2,3*N2);   for(i=0;i<N2;i++) b[i]=aux[i];
   butoir=(struct cell *)malloc(sizeof(struct cell));
```

```
   butoir->n=1000; butoir->s=butoir; butoir->p=butoir;
   avant=butoir;
   for(i=0;i<N2;i++)
    {  newcell=(struct cell *)malloc(sizeof(struct cell));
       newcell->n=b[i]; newcell->s=butoir; newcell->p= avant;
       avant->s=newcell; butoir->p= newcell;
       avant=newcell;
    }
   afficher la liste chaînée (à partir de butoir->s) correspondant au deuxième ensemble

 /*********** DEUXIEME PARTIE ***********/
   ptr2=butoir->s;
   do       /* pour chaque position de ptr2, on va faire tourner ptr1 */
     { ptr1=debut;    flag=0;
       do
        { if (ptr1->n==ptr2->n)  { flag=1; ptr2=ptr2->s; break;} /* deux nombres égaux */
          else ptr1=ptr1->s;
        }
       while (ptr1!=debut);
       if (flag==0)
        { avant1=debut->p;  /*on insère entre avant1 et debut */
          newcell=(struct cell *)malloc(sizeof(struct cell));
          newcell->n=ptr2->n; newcell->s=debut; newcell->p= avant1;
          avant1->s=newcell; debut->p= newcell;

          avant2=ptr2->p;     apres2=ptr2->s;  /* on supprime entre avant2 et apres2 */
          ptr2->s=NULL;   ptr2->p=NULL;  avant2->s=apres2;    apres2->p=avant2;
          free(ptr2);   ptr2=apres2;
        }
     }
   while(ptr2!=butoir);
   printf("\n\n");   ptr=butoir->s;  /* affichage de l'intersection et de l'union */
   do { printf("%d ", ptr->n); ptr=ptr->s;}   while (ptr!=butoir);
   printf("\n\n"); ptr=debut;
   do { printf("%d ", ptr->n); ptr=ptr->s;}  while (ptr!=debut);
   getchar(); return 0;
}

void tableauhasard(int N, int K)   /* remplissage de aux[N] par des nombres différents */
{ int i,h,dejafait,j;
  for(i=0;i<N; i++)   {  do  { dejafait=NON;  h=rand()%K;
                               for(j=0;j<i;j++) if(aux[j]==h) {dejafait=OUI; break;}
                             }
                         while( dejafait==OUI);
                         aux[i]=h;
                      }
}
```