I.U.T. INFORMATIQUE DE LYON

D.S. ALGORITHMIQUE - LANGAGE C

DURÉE 2 heures

(SEULS documents autorisés:photocopies intro. au langage C)

I. INTERPRÉTATION D'UN SWITCH (3 points)

Soit la portion de programme suivante:a et b sont des entiers

```
a=9;
switch (a-b)
 { case 1: a=a+1;
           b=b-1;
           break;
   case 2: do
            {a=a+1;
            }while(a!=b);
           break;
   case 3: while(a>=b)a=a-b;
           break;
   case 4: for(i=1;i<=a;i=i+1)b=a+b;
 }
 printf("a vaut %d\n",a);
```

Sachant qu'on a eu le message :

a vaut 3

que valait b AVANT l'exécution du switch ? Justifier votre réponse.

II. INVERSION DE L'ORDRE DES ÉLÉMENTS D'UN VECTEUR v(5 points)

v est supposé initialisé et contient **n** éléments actifs.

Ecrire une fonction

inversion(. . .)

permettant de ranger dans le **MÊME** vecteur et **SANS** utiliser un autre vecteur, les éléments de **v** dans l'**ORDRE INVERSE** de l'ordre initial (le premier en dernier, le deuxième en avant-dernier, etc..).

Cette fonction devra, de plus, comptabiliser, dans la variable locale **nbech** le nombre effectif d'échanges effectués(pas d'échange en cas d'égalité). **nbech** est, évidemment, à renvoyer en fin de traitement.

AVANT

APRÈS

III. INSERTION D'UNE VALEUR val AU BON ENDROIT DANS UN VECTEUR TRIÉ v (6 points)

v est supposé initialisé et contient **n** entiers rangés par ordre croissant. Il s'agit d'entiers. NMAX est défini en tête de programme.

Ecrire une procédure

insertion(. . .)

permettant d'insérer la valeur entière **val** à la place qui lui revient dans le vecteur **v** à **n** éléments actifs déjà triés avec décalage, vers la droite, des éléments concernés pour permettre l'insertion. Refuser l'insertion si **NMAX**, le nombre maximum de postes, est atteint.

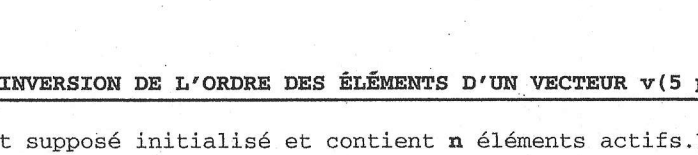
IV. TRI INDIRECT D'UN VECTEUR v À L'AIDE D'UN VECTEUR D'INDICES vind(6 points)

Il s'agit d'effectuer un tri **VIRTUEL** des éléments du vecteur v par **la méthode du minimum** MAIS sans modifier les éléments de v lui-même.

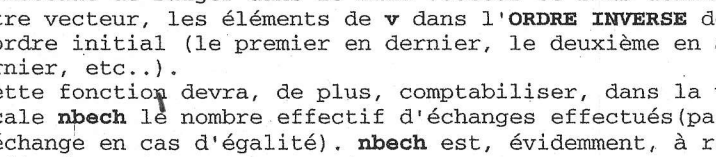
On utilisera un autre vecteur **vind** (vecteur d'indices) qui, lui, subira les échanges afin de conserver la nouvelle disposition virtuelle des postes du vecteur **v** après tri.

Ainsi, pour trier, **indirectement**, le vecteur **v** avec 4 postes actifs,on a :

AVANT LE TRI



APRES LE TRI



IV.1. Ecrire une procédure

tri_indirect(. . .)

pour réaliser le tri indirect du vecteur **v** via le vecteur d'indices **vind**.

IV.2. Ecrire une procédure

aff_resul_tri(. . .)

pour afficher le résultat du tri, par ordre croissant, à raison de 10 éléments par ligne via le vecteur d'indices **vind**.

M.RAMBURRUN

/***** ÉLÉMENTS DE SOLUTION DU DEVOIR SURVEILLÉ *****/

INTERPRÉTATION D'UN SWITCH

On a $1 \leq a-b \leq 4$ d'où $1 \leq 9-b \leq 4$ donc $5 \leq b \leq 8$

b valait 6 en fait.

En effet, a est diminué de 9 à 3 et la seule éventualité où a est diminué correspond à case 3. D'où 9 - b = 3 ce qui donne b = 6.
Pour case 1 et case 2, a est incrémenté et pour case 4 a n'est pas modifié.

Il existe deux solutions pour L'INVERSION

LUTION 1:

```
int inversion1(int v[], int n)
int i;      /* indice de parcours du vecteur v          */
int nbech;  /* compteur du nombre effectif d'échanges   */
int t;      /* variable de travail pour les échanges    */

 nbech=0;
 for(i=0; i< n /2; i++ )
 {  if (v[i] != v[n-i-1])
    {
      t=v[i];
      v[i]=v[n-i-1];
      v[n-i-1]=t;
      nbech++;
    }
 }
 return(nbech);
}
```

LUTION 2:

```
int inversion2(int v[], int n)
int i;      /* indice de parcours du vecteur v de gauche vers la droite */
int j;      /* indice de parcours du vecteur v de droite vers la gauche */
int nbech;  /* compteur du nombre effectif d'échanges */
int t;      /* variable de travail pour les échanges */

 nbech=0;
 for(i=0, j=n-1; i<j; i++, j--)
  { if (v[i] != v[j])
     {
      t=v[i];
      v[i]=v[j];
      v[j]=t;
      nbech++;
     }
  }
 return(nbech);
}
```

III. INSERTION D'UNE VALEUR val AU BON ENDROIT DANS UN VECTEUR TRIÉ v

```
void insertion(int v[], int *n, int val)
{ int i;
 if(*n)=NMAX) printf("Plus de place dans le vecteur\n");
 else
 {i=*n;
  while ( i>0 && v[i-1] > val ) /* ATTENTION A L'ORDRE DANS LE TEST */
   {
    v[i]=v[i-1];
    i=i-1;
   }
  v[i]=val;
  (*n)++;   /* À NE PAS OUBLIER SURTOUT */
 }
}
```

IV. TRI INDIRECT D'UN VECTEUR v À L'AIDE D'UN VECTEUR D'INDICES vind

v reste inchangé; le tri est répertorié sur le vecteur d'indices vind qui, à l'issue du traitement, donne les numéros des postes du vecteur v tels qu'ils devraient être après le tri.

```
void tri_indirect(int v[], int n, int vind[])
{
 int i,j,t;

 for(i=0;i<n; i++)  vind[i]=i; /* initialisation du vecteur d'indices */

  /* on intervertit les INDICES pointant vers v(grâce au vecteur vind) */
  /* à chaque fois qu'on trouve un meilleur minimum.
 for(i=0;i<n-1; i++ )  /* on recommence le processus n-1 fois */
  for(j=i+1;j<n; j++ )if( v[ vind[j] ] < v[ vind[i] ] )
                      {
                       t=vind[i];
                       vind[i]=vind[j];
                       vind[j]=t;
                      }
}

void aff_resul_tri(int v[], int n, int vind[])
{
 int i;

 printf("Voici les elements apres tri :\n");
 for(i=0;i<n;i++)
  {
   printf("%7d ", v[ vind[i] ] );  /* if( (i+1)%10==0) printf("\n"); éventuellement */
   if( (i+1)%200 ==0 )  /* 20 lignes de 10 éléments, au plus, par écran */
    {
     printf("Appuyer sur une touche pour continuer\n");

     while(getchar()!='\n');/*ou fgets(tab_vid,80,stdin); avec char tab_vid[81]; */
    }
  }
 printf("\n");
}
```

autre solution: avec définition de posmin en plus

```
for (i=0 ; i< n-1; i++)
{ posmin=i;
 for(j=i+1; j<n; j++)
   if(v[vind[j]] < v[vind[posmin]]) posmin=j
 if(posmin != i)
 {t=vind[i];
  vind[i] = vind[posmin];
  vind[posmin]=t;
 }}
```

M. RAMBURRUN

IUT Lyon 1
l'excellence technologique

D.S. ALGO-LANGAGE C

DURÉE : 1h50

SEUL DOCUMENT AUTORISÉ : MÉMENTO SYNTAXIQUE DU LANGAGE C

I. EXERCICE SUR UN VECTEUR (6 points)

```
NMAX=50
#définition de la fonction f admettant comme arguments le vecteur v et n et retournant un entier
def f(v,n):
#entrées :
#                le vecteur v est constitué d'entiers et est de taille NMAX
# n : int              n est le nombre actif de postes du vecteur v
#sortie :
 #la fonction f retourne un entier
#définitions de variables de travail :
  #i : int
  #k : int
#début
    i=0
    k=1
    while ( k < n ):
       if ( v[k] <= v[i] ):
          i=k
       #endif
       k=k+1
   #endwhile
    return(i)
#fin
```

I.1. Proposer des jeux de données différents en signalant le contenu de chaque zone mémoire afin de tester cette fonction.
I.2. Que fait cette fonction?
I.3. Réécrire la fonction en langage C en utilisant une boucle for.

II. TASSEMENT, SUR SA PARTIE GAUCHE, DES VALEURS NON NULS D'UN VECTEUR (3 points)

Écrire une procédure tasse(……) permettant le tassement des éléments utiles(ici les éléments non nuls) du vecteur v sur sa partie gauche. On élimine les zéros tout en mettant à jour le nombre actif d'éléments utiles.

III. INTERPRÉTATION D'UNE FONCTION (11 points)

```
int fonction( int x, int y )
{
  if ( abs(x) < abs(y) ) return (     fonction ( y,   x ) ) ;
  if (     y <     0  ) return (     fonction ( -y, -x ) ) ;
  if (     y !=    0  ) return ( x + fonction ( x, y-1 ) ) ;
   else                  return ( 0 ) ;
}
```

III.1. Que retourne cette fonction pour x =15 et y = 3 ?
III.2. Combien d'appels y aura-t-il dans les 5 cas suivants :
(l'appel initial dans la fonction appelante devra être pris en compte)

(a) x = 15 et y = 3
(b) x = 3 et y = 15
(c) x = 15 et y = -3
(d) x = -3 et y = 15
(e) x = -3 et y =-15

III.3. Quel est le traitement arithmétique effectué par cette fonction ?
III.4. Cette fonction comporte un "BUG" ? Lequel ? Proposer une solution.
III.5. Modifier cette fonction afin d'y intégrer un moyen de comptage du nombre d'appels qu'on afficherait en même temps que le résultat.

SOLUTION D.S. ALGO-LANGAGE C

I. EXERCICE SUR UN VECTEUR (6 points)

I.1. Test sur différents jeux d'essai

Pour
int v[]={4,3,1,2}; avec n valant 4,
i , qu'on retourne, vaudra 2.

Pour
int v[]={4,3,1,1}; avec n valant 4,
i vaudra 3.

Pour
int v[]= {1,1,1,1}; avec n valant 4,
i vaudra 3.

I.2. Que fait cette fonction ?

Cette fonction retourne le numéro de poste de la DERNIÈRE occurrence du minimum du vecteur v.

I.3. Réecriture de la fonction à l'aide d'un for

```
int f( int v[], int n)
{
  int i; /* pour trouver le numéro de poste de la dern. occ. du min */
  int k; /* pour parcourir le vecteur à partir du 2eme élément      */

  i=0;

  for( k = 1; k < n; k++ ) if ( v[k] <= v[i] ) i = k;

  return i;
}
```

II. TASSEMENT, SUR SA PARTIE GAUCHE, DES VALEURS NON NULS D'UN VECTEUR (3 points)

```
void tasser( int v[], int *n )
{
 int i;          /* indice de parcours du vecteur */
 int nbrenonnul; /* compteur pour gérer les éléments non nuls de v */

 for( i = 0; i < *n; i++ )
   if ( v[i] )   /* équivaut à faire if ( v[i] != 0 ) */
   {
     v[nbrenonnul] = v[i];
     nbrenonnul++;
   }
 *n = nbrenonnul;   /* mise à jour de *n */
}
```

III. INTERPRÉTATION D'UNE FONCTION (11 points)

```
int fonction( int x, int y )
{
  if ( abs(x) < abs(y) ) return (     fonction ( y, x ) ) ;
  if (     y  <     0  ) return (     fonction (-y,-x ) ) ;
  if (     y !=     0  ) return ( x + fonction ( x,y-1) ) ;
   else                   return ( 0 ) ;
}
```

III.1. Que retourne cette fonction pour x =15 et y = 3; réponse : 45
III.2. Combien d'appels y aura-t-il dans les 5 cas suivants :
(l'appel initial dans la fonction appelante devra être pris en compte)

(a)	x = 15 et y = 3	réponse : résultat= 45 nombre d'appels=4
(b)	x = 3 et y = 15	réponse : résultat= 45 nombre d'appels=5
(c)	x = 15 et y = -3	réponse : résultat=-45 nombre d'appels=6
(d)	x = -3 et y = 15	réponse : résultat=-45 nombre d'appels=7
(e)	x = -3 et y =-15	réponse : résultat= 45 nombre d'appels=7

III.3. Quel est le traitement arithmétique effectué par cette fonction ?

Réponse : on ramène la multiplication de 2 entiers à de simples additions d'entiers : x * y = (x + x + … +x) (y fois)

III.4. Cette fonction comporte un "BUG" ? Lequel ? Proposer une solution.
Réponse : la fonction boucle indéfiniement sans atteindre la condition d'arrêt pour tout couple (x0, -x0) où x0 est un entier positif.
Par exemple (1,-1) ou (15,-15) entraîne un bouclage infini jusqu'à dépassement de la capacité de la pile du programme.
Pour éliminer ce "BUG", il suffit de remplacer

```
    if (    y  <    0  ) return ( fonction (-y,-x ) ) ;
```

par

```
    if (    y  <    0  ) return ( fonction (-x,-y ) ) ;
```

III.5. Solution intégrant un comptage explicite :

```
int fonction( int x, int y , int *ctr )
{
  (*ctr)++;
  if ( abs(x) < abs(y) ) return ( fonction ( y, x ,ctr ) ) ;
  if (    y  <    0  ) return ( fonction (-y,-x,ctr ) ) ;
  if (    y !=    0  ) return ( x + fonction ( x, y - 1, ctr ) ) ;
   else                  return ( 0 ) ;
}
int main()
{
    int ctr = 0;
    printf("fonction( 15,  3 ) = %3d; nombre d'appels = %d\n",
            fonction( 15, 3,  &ctr), ctr );
    ctr = 0;
    printf("fonction( 3,  15 ) = %3d; nombre d'appels = %d\n",
            fonction( 3, 15,  &ctr), ctr );
    ctr = 0;
    printf("fonction( 15 ,-3 ) = %3d; nombre d'appels = %d\n",
            fonction( 15,-3,  &ctr), ctr );
    ctr = 0;
    printf("fonction( -3, 15 ) = %3d; nombre d'appels = %d\n",
            fonction( -3,15,  &ctr), ctr );
    ctr = 0;
    printf("fonction( -3,-15 ) = %3d; nombre d'appels = %d\n",
            fonction( -3,-15, &ctr), ctr );
    return 0;
}
```