

```
/**
** Hanoi1 - programme de demonstration de la recursivite pour l'initiation
** au C 5 de GS Infos 21 : le celebre probleme des tours de hanoi.
**
** Version recursive.
**
** Philippe Manet      12 Avril 1992
**
**/
```

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

```
#pragma optimize -1
```

```
void hanoi ( short n, char a, char b, char c )
```

```
{
```

```
/*
```

```
* Deplacement de N disques de A a C avec B comme intermediaire.
```

```
*/
```

```
if ( n > 0 ) {
```

```
    hanoi ( n - 1, a, c, b );
```

```
    printf ( "Deplacement d'un disque de \"%c\" vers \"%c\"\n", a, c );
```

```
    hanoi ( n - 1, b, a, c );
```

```
}
```

```
} /* hanoi () */
```

```
void main ( void )
```

```
{
```

```
    char buf[4];
```

```
    short n;
```

```
/*
```

```
* Scanf est bugge : on lit donc une chaine de caracteres que l'on decode
* ensuite.
```

```
*/
```

```
printf ( "Nombre de disques ? ");
```

```
gets ( buf );
```

```
n = atoi ( buf );
```

```
hanoi ( n, 'A', 'B', 'C' );
```

```
} /* main () */
```

```
/**
** Hanoi2 - programme de demonstration de la recursivite pour l'initiation
** au C 5 de GS Infos 21 : le celebre probleme des tours de hanoi.
**
** Version non recursive.
**
** Philippe Manet      12 Avril 1992
**
**/
```

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

```
#pragma optimize -1
```

```
/*
* Pile utilisee pendant les deplacements des disques.
*/
```

```
struct {
    short count;
    struct {
        short n;
        char a;
        char b;
        char c;
    } entry[100];
} stack;
```

```
void push ( short n, char a, char b, char c )
```

```
{
    short i;

    i = stack.count++;
    stack.entry[i].n = n;
    stack.entry[i].a = a;
    stack.entry[i].b = b;
    stack.entry[i].c = c;
} /* push () */
```

```
void pop ( short *n, char *a, char *b, char *c )
```

```
{
    short i;

    i = --stack.count;
    *n = stack.entry[i].n;
    *a = stack.entry[i].a;
    *b = stack.entry[i].b;
    *c = stack.entry[i].c;
```

```
} /* pop () */
```

```
void swap ( char *x, char *y )
```

```
{  
    char t;
```

```
    t = *x;  
    *x = *y;  
    *y = t;
```

```
} /* swap () */
```

```
void hanoi ( short n, char a, char b, char c )
```

```
{  
  
    /*  
    * Deplacement de N disques de A a C avec B comme intermediaire.  
    */
```

```
    stack.count = 0;
```

```
    do {
```

```
        if ( stack.count != 0 ) {
```

```
            pop ( &n, &a, &b, &c );  
            printf ( "Deplacement d'un disque de \"%c\" vers \"%c\"\n", a, c );  
            n--;  
            swap ( &a, &b );
```

```
        }
```

```
        while ( n > 0 ) {
```

```
            push ( n, a, b, c );  
            n--;  
            swap ( &c, &b );
```

```
        }
```

```
    } while ( stack.count > 0 );
```

```
} /* hanoi () */
```

```
void main ( void )
```

```
{
```

```
char buf[4];
short n;

/*
 * Scanf est bugge : on lit donc une chaine de caracteres que l'on decode
 * ensuite.
 */

printf ( "Nombre de disques ? ");
gets ( buf );
n = atoi ( buf );

hanoi ( n, 'A', 'B', 'C' );

} /* main () */
```

```

/**
** Hanoi3 - programme de demonstration de la recursivite pour l'initiation
** au C 5 de GS Infos 21 : le celebre probleme des tours de hanoi.
**
** Version graphique.
**
** Philippe Manet      12 Avril 1992
**
**/

```

```

#include <stdio.h>
#include <orca.h>

```

```

#include <Types.h>
#include <QuickDraw.h>

```

```

#pragma optimize -1

```

```

#define MAX_DISKS  14
#define BASE      180      /* base des tours */
#define ITOWER    100
#define TOWER1    60      /* position X des tours */
#define TOWER2    ( TOWER1 + ITOWER )
#define TOWER3    ( TOWER2 + ITOWER )
#define HDISK     10      /* hauteur d'un disque */
#define LDISK     10      /* moitie longueur du plus petit disque */
#define IDISK     2       /* moitie diff longueur entre 2 disques */

```

```

Rect pos_disk[3][MAX_DISKS];
short num_disk[3][MAX_DISKS];
short height[3];
short num_disks;

```

```

void draw_disk ( short tower, short disk, short flag )

```

```

{
    short x;

    if ( flag ) {
        SetSolidPenPat ( disk );
        PaintRect ( &pos_disk[tower][disk] );
    } else {
        EraseRect ( &pos_disk[tower][disk] );
    }
    /*
    * Redessine la portion de tour effacee.
    */
}

```

```

    if ( pos_disk[tower][disk].v1 >= BASE - ( num_disks + 1 ) * HDISK ) {

        SetSolidPenPat ( 15 );
        SetPenSize ( 4, 1 );
        x = tower == 0 ? TOWER1 - 2 : tower == 1 ? TOWER2 - 2 : TOWER3 - 2;
        MoveTo ( x, pos_disk[tower][disk].v1 );
        LineTo ( x, pos_disk[tower][disk].v2 - 1 );
        SetPenSize ( 1, 1 );

    }

}

} /* draw_disk () */

void pause ( void )

{

    short w;

    for ( w = 0; w < 22222; w++ );

} /* pause () */

void move_disk ( char from, char to )

{

    short tower, disk, dest_x, dest_y, move_i;

    /*
    * Deplacement graphique d'un disque.
    */

    tower = from - 'A';
    disk = num_disk[tower][height[tower]--];

    /*
    * Deplacement vers le haut sur la tour de depart.
    */

    do {

        draw_disk ( tower, disk, false );
        pos_disk[tower][disk].v1 -= HDISK;
        pos_disk[tower][disk].v2 -= HDISK;
        draw_disk ( tower, disk, true );
        pause ();

    } while ( pos_disk[tower][disk].v1 >= BASE - ( num_disks + 3 ) * HDISK );

```

```

/*
 * Deplacement horizontal.
 */

dest_x = ( to - from ) * ITOWER + pos_disk[tower][disk].h1;
move_i = LDISK * ( to > from ? 1 : -1 );

do {

    draw_disk ( tower, disk, false );
    pos_disk[tower][disk].h1 += move_i;
    pos_disk[tower][disk].h2 += move_i;
    draw_disk ( tower, disk, true );
    pause ();

} while ( pos_disk[tower][disk].h1 != dest_x );

/*
 * Deplacement vers le bas sur la tour d'arrivee.
 */

tower = to - 'A';
num_disk[tower][++height[tower]] = disk;
pos_disk[tower][disk] = pos_disk[from - 'A'][disk];
dest_y = BASE - ( height[tower] + 1 ) * HDISK;

do {

    draw_disk ( tower, disk, false );
    pos_disk[tower][disk].v1 += HDISK;
    pos_disk[tower][disk].v2 += HDISK;
    draw_disk ( tower, disk, true );
    pause ();

} while ( pos_disk[tower][disk].v1 < dest_y );

} /* move_disk () */

void hanoi ( short n, char a, char b, char c )

{

/*
 * Deplacement de N disques de A a C avec B comme intermediaire.
 */

if ( n > 0 ) {

    hanoi ( n - 1, a, c, b );
    move_disk ( a, c );
    hanoi ( n - 1, b, a, c );

```

```

    }

} /* hanoi () */

void main ( void )

{

    char buf[4];
    short d;

    /*
    * Scanf est bugge : on lit donc une chaine de caracteres que l'on decode
    * ensuite.
    */

    printf ( "Nombre de disques ? ");
    gets ( buf );
    num_disks = atoi ( buf );
    if ( num_disks > MAX_DISKS )
        exit ();

    /*
    * Initialisation du mode graphique et dessin des tours et de la base.
    */

    startgraph ( 320 );

    SetPenMode ( modeCopy );
    SetForeColor ( 15 );

    SetPenSize ( 1, 10 );
    MoveTo ( 0, BASE );
    LineTo ( 320, BASE );

    SetPenSize ( 4, 1 );
    MoveTo ( TOWER1 - 2, BASE );
    LineTo ( TOWER1 - 2, BASE - ( num_disks + 1 ) * HDISK );
    MoveTo ( TOWER2 - 2, BASE );
    LineTo ( TOWER2 - 2, BASE - ( num_disks + 1 ) * HDISK );
    MoveTo ( TOWER3 - 2, BASE );
    LineTo ( TOWER3 - 2, BASE - ( num_disks + 1 ) * HDISK );

    /*
    * Calcul des rectangles des disques et dessin initial.
    */

    for ( d = 0; d < num_disks; d++ ) {

        pos_disk[0][d].h1 = TOWER1 - ( LDISK + IDISK * ( num_disks - 1 - d ) );
        pos_disk[0][d].h2 = TOWER1 + LDISK + IDISK * ( num_disks - 1 - d );
    }
}

```

```

    pos_disk[0][d].v1 = BASE - HDISK * ( d + 1 );
    pos_disk[0][d].v2 = BASE - HDISK * d;
    num_disk[0][d] = d;
    num_disk[1][d] = num_disk[2][d] = -1;
}

height[0] = num_disks - 1;
height[1] = height[2] = -1;

SetPenSize ( 1, 1 );
SetSolidBackPat ( 0 );

for ( d = 0; d < num_disks; d++ )
    draw_disk ( 0, d, true );

hanoi ( num_disks, 'A', 'B', 'C' );

pause ();

endgraph ();
} /* main () */

```

Programmation n°2 : Structure de données "queue"

=====

Dans le précédent numéro de GS Infos, nous avons abordé dans la dernière partie de l'article sur la programmation de la calculatrice, la structure de données "pile". Dans cet article, nous allons traiter d'une autre structure de données, qui est en bien des points similaire. En fait, elle est même le pendant de la "pile" : il s'agit de la "queue". Dans la littérature, vous trouverez plus souvent employé le terme de "file" qui est une abréviation de "file d'attente", sans doute pour encore mieux montrer l'analogie entre les 2 structures de données. Personnellement, je préfère le terme de "queue" que je vais donc employer tout au long de cet article.

Concepts de Queue

=====

La structure de données "queue" est la réalisation informatique du concept de queue tel qu'on le connaît dans la vie courante, par exemple lorsque vous faites la queue devant une salle de cinéma.

D'un point de vue informatique, on considère la queue comme étant une liste, telle que ce concept a été introduit dans le précédent article, et qui sera détaillé dans le prochain numéro de GS Infos.

La propriété fondamentale des queues est que tous les éléments sont insérés à une extrémité et retirés de l'autre. Le côté où l'on insère les éléments est appelé la "queue" (c'est ce qui a donné son nom à la structure de données), tandis que celui duquel on retire les éléments est appelé la "tête". Les termes anglais correspondants sont "head" ou "front" pour la tête et "tail" ou "rear" pour la queue.

L'expression employée pour désigner le mouvement des éléments dans la queue est "premier arrivé, premier sorti", ce qui correspond bien à la notion de file d'attente. Le terme anglais correspondant est "first in, first out" que l'on abrège par "FIFO".

Si vous relisez l'article sur les piles du précédent numéro, vous constaterez que ces 2 structures sont quasiment identiques; en fait, la seule différence est que dans le cas d'une pile, les insertions et les suppressions se font du même côté, tandis que, pour une queue, les suppressions se font du côté opposé aux insertions.

Par conséquent les opérations sont identiques entre les 2 structures, seuls leurs effets changent. Donc, comme pour les piles, il n'y a que 5 opérations possibles pour les queues :

- Initialisation d'une queue;
- Test si la queue est vide;
- Ajout d'un élément à la fin de la queue; la littérature, voulant conserver la similitude avec les piles, utilise le terme d'"enfiler", que je trouve personnellement assez malheureux; n'ayant pas de meilleur terme à vous proposer, je me contenterai du mot anglais "enqueue";
- Suppression d'un élément en tête de la queue; le terme employé est "défiler", ce qui est tout aussi mauvais que "enfiler"; le mot anglais "dequeue" est beaucoup plus précis.
- Accès au premier élément de la queue sans le retirer.

