

## Structures de données : les bases (corrigé)

### Un premier programme C sous LINUX

Dans un terminal LINUX, créez un répertoire de travail (in3i21, par exemple) et placez-y vous.

```
acme1:~> mkdir in3i21
acme1:~> cd in3i21
acme1:~/in3i21>
```

Avec un éditeur de texte (par exemple, nedit),

```
acme1:~/in3i21> nedit &
```

saisissez le programme source C suivant :

```
1  /*
2  pgm01.c Un premier exemple de programme C
3  */
4  #include<stdio.h>
5  int main() {
6      int n;
7      printf("Entrez un nombre entier : ");
8      scanf("%d", &n);
9      printf("%d x %d = %d\n", n, n, n*n);
10     return 0;
11 }
```

Sauvegardez-le sous le nom pgm01.c, puis compilez-le :

```
acme1:~/in3i21> gcc pgm01.c
```

Si la compilation se passe bien, un fichier exécutable de nom a.out est créé et vous pouvez l'exécuter en lançant la commande :

```
acme1:~/in3i21> ./a.out
```

Un message apparaît, correspondant à la ligne 7 du listing.

```
Entrez un nombre entier :
```

Le programme s'interrompt alors, attendant l'intervention de l'utilisateur pour la saisie du nombre (ligne 8 du listing).

Si on entre 12, par exemple, le programme affiche le résultat (ligne 9 du listing), puis se termine :

```
Entrez un nombre entier : 12
12 x 12 = 144
```

### Un premier programme C avec VisualStudio

Lancez VisualStudio.

Dans le menu Fichier, choisissez Nouveau, puis Projet.

Dans la fenêtre qui s'ouvre, choisissez VisualC++, Win32, Application console Win32, donnez lui un nom (par exemple essai) et un emplacement (par exemple in3i21).

Cliquez sur **Ok**.

Dans l'assistant, cliquez sur **Suivant**, et cochez la case Projet vide dans Options supplémentaires, puis

**Terminer**.

Dans le menu Projet, choisissez Ajouter un nouvel élément, choisissez Fichier C++ et entrez son nom essai.c. Précisez bien le suffixe .c

Saisissez le programme suivant :

```
1  /*
2  pgm01.c Un premier exemple de programme C
3  */
4  /* VisualStudio est tres paranoiaque et par default ne permet pas
5  l'utilisation de scanf. Pour l'autoriser, entrez la ligne suivante:
6  */
7  #define _CRT_SECURE_NO_WARNINGS
8
9  #include<stdio.h>
10 int main() {
11     int n;
12     printf("Entrez un nombre entier : ");
13     scanf("%d", &n);
14     printf("%d x %d = %d\n", n, n, n*n);
15     return 0;
16 }
```

Dans le menu Générer, choisissez Générer la solution.


Si tout se passe bien, vous pouvez exécuter le programme (menu Déboguer, Exécuter sans débogage).

Un message apparaît, correspondant à la ligne 12 du listing.

```
Entrez un nombre entier :
```

Le programme s'interrompt alors, attendant l'intervention de l'utilisateur pour la saisie du nombre (ligne 13 du listing).

Si on entre 12, par exemple, le programme affiche le résultat (ligne 14 du listing), puis se termine :

```
Entrez un nombre entier : 12 
12 x 12 = 144
```

VisualStudio vous envoie le message Appuyez sur une touche pour continuer... pour vous signaler la fin de l'exécution de votre programme.

## Exemple d'écriture d'une fonction

Écrivez deux fonctions calculant la factorielle de  $n$ , la première à l'aide d'une boucle **for** et la deuxième en utilisant la récursivité.

Exemple de solution :

```
/* int factorielle_iterative(int n)
 * calcule la factorielle de n
 * Principe : accumulation des produits des n premiers nombres
 * arguments : n entier positif
 * retour : factorielle de n si n>=0
              1 si n est negatif
 * JCG ecrit le 22/03/2001 modif le 14/04/2013
 */
int factorielle_iterative(int n) {
    int i, p = 1;
    for (i = 1; i <= n; ++i) p *= i;
    return p;
}
/*****
 * int factorielle_recursive(int n)
 * calcule la factorielle de n
 * Principe : F(n)=1 si n=0, F(n)=n*F(n-1) si n>0
 * arguments : n entier positif
 * retour : factorielle de n si n>=0
              1 si n est negatif
 * JCG ecrit le 22/03/2001 modif le 14/04/2013
 */
int factorielle_recursive(int n) {
    if (n <= 0)
        return 1;
    else
        return n * factorielle_recursive(n-1);
}
/*****/
#include <stdio.h>
int main(void) {
    int n;
    do {
        printf("Entrez un nombre entier (0 pour terminer) : ");
        scanf("%d", &n);
        printf("factorielle_iterative(%2d) = %15d\n", n, factorielle_iterative(n));
        printf("factorielle_recursive(%2d) = %15d\n", n, factorielle_recursive(n));
    } while (n != 0);
    return 0;
}
```

Compilez et exécutez ce programme. Entrez les valeurs 1, 2, 5, 10, 13. Comment peut-on voir que le résultat pour 13 est faux? Pourquoi est-il faux?

Comment modifier les fonctions pour que le nombre retourné soit un nombre impossible (-1 par exemple) lorsque le résultat ne peut pas être calculé?

### Corrigé

\*\*\*\*\*

Si l'on est pas sûr que le résultat d'une opération restera dans les limites représentables, il faut effectuer l'opération inverse entre le plus grand nombre représentable et un des opérandes et comparer le résultat au deuxième opérande pour "voir" si l'opération peut se faire. Par exemple, pour la factorielle :

```
#include <limits.h>
/*
limits.h définit entre autres :
CHAR_BIT      : taille d'un char en bits

SCHAR_MAX     : plus grand signed char
SCHAR_MIN     : plus petit signed char
UCHAR_MAX     : plus grand unsigned char

INT_MAX       : plus grand int
```

```

INT_MIN      : plus petit int
UINT_MAX     : plus grand unsigned int
SHRT_MAX     : plus grand short
SHRT_MIN     : plus petit short
USHRT_MAX    : plus grand unsigned short
LONG_MAX     : plus grand long
LONG_MIN     : plus petit long
ULONG_MAX    : plus grand unsigned long
*/

int factorielle_iterative2(int n) {
    int i, p = 1;
    if (n < 0) return -1;
    for (i = 1; i <= n; ++i) {
        if (INT_MAX/i < p) /* <=> si p*i>INT_MAX */
            return -1;
        else
            p *= i;
    }
    return p;
}

/*****
for (i=10;i<15;++i)
    printf("%d->%d, ",i,factorielle_iterative2(i));
donne : 10->3628800, 11->39916800, 12->479001600, 13->-1, 14->-1,
*****/

*****

```

# 1 Les types du langage C

## 1.1 Types de base

### Exercice 1. Détermination de type

Choisissez parmi les types de base (`char`, `int`, `double`, `char[]`) le type vous semblant le plus adapté pour représenter les données suivantes :

- une année de naissance ;
- un numéro de téléphone ;
- un numéro de département français ;
- un taux de TVA ;
- une taille de fichier.

#### Corrigé

```

*****
- année de naissance : int (on peut faire des calculs)
- numéro de téléphone : char[] (pas de calcul, 0 initial signifiant)
- numéro de département : char[] (pas de calcul, 0 initial signifiant, et la Corse !)
- taux de TVA : double (des calculs, évidemment)
- taille de fichier : int (des calculs, évidemment)
*****

```

### Exercice 2. Proximité de deux double

Pour "visualiser" le codage binaire d'un `double`, voir le site : <http://babbage.cs.qc.cuny.edu/IEEE-754/index.xhtml>

Avec les définitions suivantes,

```

#define ZERO 1e-100
#define EPSILON 1e-10

```

écrivez la fonction

```
int proche(double a, double b);
```

qui renvoie un entier non nul si *a* et *b* sont proches l'un de l'autre (tous deux inférieurs à *ZERO* en valeur absolue, ou égaux à *EPSILON* près), et 0 sinon.

## 1.2 Les tableaux

En C, l'accès à un élément de tableau n'est pas sécurisé. Pour éviter de *sortir* d'un tableau, on utilise principalement deux solutions :

- on fournit à la fonction travaillant sur le tableau la longueur exploitable du tableau : tout accès peut alors être testé;
- on marque la *fin* du tableau par une sentinelle (un élément impossible) : utilisable principalement pour une exploitation séquentielle.

### Exercice 3. Parcours de tableau

Complétez les fonctions manquantes du programme suivant. Vérifiez que vos fonctions passent les tests.

```
#include <stdio.h>
/* la reponse a la question precedente */
#define ZERO 1e-100
#define EPSILON 1e-10
/* inclure <math.h> et compiler avec gcc -lm */
int proche(double a, double b) {
    return fabs(a) < ZERO ? fabs(b) < ZERO : fabs(a-b) < EPSILON * fabs(a);
}

double moyenne(double t[], int n) {
    /* calcule la moyenne des n premiers elements du tableau t */
    /* -----
    * a faire
    * -----
    */
    return 0.0;
}

double moyenne_positifs(double t[]) {
    /* calcule la moyenne des elements du tableau t jusqu'a rencontrer un
    element negatif et -1.0 si le premier element est deja negatif */
    /* -----
    * a faire
    * -----
    */
    return 0.0;
}

double test_moyenne() {
    double v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, -1.0};
    double d, attendu;
    attendu = 1;
    /* test moyenne */
    if (! proche((d = moyenne(v, 1)), attendu)) {
        printf("Pb moyenne. Attendu : %f Obtenu : %f\n", attendu, d);
    }
    attendu = 2.0;
    if (! proche((d = moyenne(v, 3)), attendu)) {
        printf("Pb moyenne. Attendu : %f Obtenu : %f\n", attendu, d);
    }
    attendu = 3.5;
    if (! proche((d = moyenne(v, 6)), attendu)) {
        printf("Pb moyenne. Attendu : %f Obtenu : %f\n", attendu, d);
    }
    /* test moyenne positifs */
    attendu = 3.5;
```

```

    if (! proche((d = moyenne_positifs(v)), attendu)) {
        printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
    }
    attendu = 5.0;
    if (! proche((d = moyenne_positifs(v + 3)), attendu)) {
        printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
    }
    attendu = -1.0;
    if (! proche(d = moyenne_positifs(v + 6), -1.0)) {
        printf("Pb moyenne_positifs. Attendu : %f Obtenu :%f\n", attendu, d);
    }
}
int main() {
    test_moyenne();
    return 0;
}

```

## Corrigé

```

*****
double moyenne(double t[], int n) {
    /* calcule la moyenne des n premiers elements du tableau t */
    int i;
    double s = 0.0;
    for (i = 0; i < n; ++i)
        s += t[i];
    return s / n;
}

double moyenne_positifs(double t[]) {
    /* calcule la moyenne des elements du tableau t jusqu'a rencontrer un element
negatif
retourne -1.0 si le premier element est negatif */
    int i;
    double s = 0.0;
    for (i = 0; t[i] >= 0 ; ++i)
        s += t[i];
    return i == 0 ? -1.0 : s / i;
}
*****

```

## Exercice 4. Parcours récursifs de tableau

Écrivez la fonction :

```
int chaine_longueur_rec(char* s);
```

qui calcule **récursivement** la longueur d'une chaîne de caractères terminée par '\0'.

## Corrigé

```

*****
int chaine_longueur_rec(char* s) {
    if (*s == 0)
        return 0;
    else
        return 1 + chaine_longueur_rec(s + 1);
}

ou, en oneliner :
int chaine_longueur_rec(char* s) {
    return *s == 0 ? 0 : 1 + chaine_longueur_rec(s + 1);
}
*****

```

Écrivez la fonction **récursive** :

```
int chaine_debute_par(char* s1, char* s2);
```

qui retourne un entier non nul si s1 commence par s2, et 0 sinon.

## Corrigé

```

*****

int chaine_commence_par(char* s1, char* s2) {
/* retourne un entier non nul si s1 commence par s2, 0 sinon */
    if (*s2 == 0) {
        return 1;
    }
    else {
        if (*s1 != *s2) {
            return 0;
        }
        else {
            return chaine_commence_par(s1 + 1, s2 + 1);
        }
    }
}

```

ou, en *oneliner* :

```

int chaine_commence_par(char* s1, char* s2) {
/* retourne un entier non nul si s1 commence par s2, 0 sinon */
    return (*s2 == 0) || (*s1 == *s2) && chaine_commence_par(s1 + 1, s2 + 1);
}

```

```

*****

```

### Exercice 5. Recherche de motif

Écrivez la fonction ;

```
int chaine_index(char* s1, char* s2);
```

qui retourne la position de s2 dans s1 si s1 contient s2, -1 sinon.

## Corrigé

```

*****

int chaine_index(char* s1, char* s2) {
/* retourne la position de s2 dans s1 si s1 contient s2, -1 sinon */
    int i1 = 0; /* position dans s1 */
    int i2 = 0; /* position dans s2 */
    while (s2[i2] != '\0' && s1[i1] != '\0') { /* tq 2 ch. en cours */
        if (s2[i2] == s1[i1]) { /* si egalite des caracteres */
            i2++;
            i1++; /* on avance de une position */
        } else { /* si inegalite */
            i1 -= i2-1; /* on recule dans s1 */
            i2 = 0; /* on repart a 0 dans s2 */
        }
    }
    if (s2[i2] == '\0') /* si tout s2 est consommé */
        return i1-i2; /* pos. debut de s2 ds s1 = pos s1 - lgr s2 */
    else /* si s1 est consommé sans que s2 soit terminée */
        return -1; /* s2 n'est pas dans s1 */
}

```

ou en réutilisant chaine\_commence par

```

int chaine_index(char* s1, char* s2) {
/* retourne la position de s2 dans s1 si s1 contient s2, -1 sinon */
/* la chaine s2 est dans la chaine s1 si :
- elle commence la chaine s1
- elle est dans la chaine s1+1
la chaine s2 n'est pas dans la chaine s1 si :
- la chaine s1 est nulle et la chaine s2 n'est pas nulle
*/
    int i;
    for (i = 0; s1[i] != '\0'; ++i)
        if (chaine_commence_par(s1 + i, s2))

```

```

    return i;
    return -1;
}

```

\*\*\*\*\*

### 1.3 Les pointeurs

Un pointeur est une variable destinée à contenir l'adresse d'une zone mémoire typée. Le rôle principal d'un pointeur est de permettre d'accéder à une zone mémoire de façon indirecte, pour pouvoir la modifier.

#### Exercice 6. Modification d'une variable par pointeur

Écrivez la fonction suivante

```

void carre(int* p) {
    /* modifie en l'elevant au carre l'entier dont l'adresse est dans p */
    // a completer
}

```

Testez-la avec :

```

int main() {
    int n = 5;
    carre(&n);
    printf("%d\n", n); /* => 25 */
    return 0;
}

```

Corrigé

\*\*\*\*\*

```

void carre(int* p) {
    /* modifie en l'elevant au carre l'entier dont l'adresse est dans p */
    *p *= *p; /* Combien le caractere '*' joue-t-il de roles ds la fonction ? */
}

```

\*\*\*\*\*

### 1.4 Les unions

#### Exercice 7. Affichage du codage binaire d'un double (facultatif)

Écrire la fonction

```

void aff_bin(double d);

```

qui affiche le codage binaire d'un double

Corrigé

\*\*\*\*\*

```

#include <limits.h>

typedef union {
    int n;
    double d;
    unsigned char c[sizeof(double)];
} union_type_simple;

char *char_to_codage_binaire(unsigned char c, char* t) {
    int i;
    for (i = 0; i < CHAR_BIT; ++i) {

```



```

        t[CHAR_BIT - 1 - i] = ((c & (1 << i)) != 0) + '0';
    }
    t[CHAR_BIT] = '\0';
    return t;
}

char *int_to_codage_binaire(int n, char* t) {
    int i;
    union_type_simple x;
    x.n = n;
    char_to_codage_binaire(x.c[0], t);
    for (i = 0; i < sizeof(int); ++i) {
        t[9 * i - 1] = ' ';
        char_to_codage_binaire(x.c[sizeof(int) - 1 - i], t + 9 * i);
    }
    return t;
}

char *double_to_codage_binaire(double d, char* t) {
    int i;
    union_type_simple x;
    x.d = d;
    char_to_codage_binaire(x.c[0], t);
    for (i = 0; i < sizeof(double); ++i) {
        t[(CHAR_BIT + 1) * i - 1] = ' ';
        char_to_codage_binaire(
            x.c[sizeof(double) - 1 - i],
            t + (CHAR_BIT + 1) * i
        );
    }
    return t;
}

void aff_bin(double d) {
    char t[(CHAR_BIT + 1) * sizeof(double)];
    double_to_codage_binaire(d, t);
    printf("%e = %s\n", d, t);
}

```

\*\*\*\*\*

## 2 Les piles

Une pile est une structure de donnée abstraite permettant de stocker et de récupérer des données selon un schéma LIFO (last in, first out) : seul l'accès au dernier élément de la pile est permis. L'ajout d'un nouvel élément (empiler) se fait au sommet de la pile, la consultation ou la suppression d'un élément n'est possible que sur le dernier élément ajouté.

Les opérations sur pile sont les suivantes :

- **empile** : ajoute dans la pile un élément
- **depile** : supprime de la pile le dernier élément empilé et retourne sa valeur
- **lit\_sommet** retourne la valeur du dernier élément empilé (la pile est inchangée)
- **est\_vide** : retourne Vrai si la pile est vide, Faux sinon
- **init** : vide la pile

### Exercice 8. Implémentation de pile par tableau

Une implémentation de pile simple (pile d'entiers par exemple) peut se faire en utilisant un tableau : la pile est constituée d'un tableau d'entiers, et d'un indice spécial indiquant la position dans le tableau du dernier élément empilé.

Par exemple :

```

#define TMAX 10
typedef struct{
    int donnees[TMAX];

```

```

    int sommet;
} t_pile;

```

La **création** d'une pile se fera simplement par la déclaration d'une variable de ce type :

```
t_pile p;
```

L'**initialisation** se fera par la mise à 0 du champ sommet de la structure :

```
p.sommet = 0;
```

ou mieux en écrivant et en utilisant la fonction qui le fera :

```

void init_pile(t_pile* pp) {
    pp->sommet = 0;
}
...
init_pile(&p);

```

L'**utilisation** de la pile créée et initialisée se fera également par l'intermédiaire de fonctions plutôt que par des instructions en ligne :

```

int est_vide_pile(t_pile* pp) {
    return pp->sommet == 0;
}

void empile(t_pile* pp, int x) {
    pp->contenu[pp->sommet] = x;
    ++pp->sommet;
    /* pour les voltigeurs :
    pp->contenu[pp->sommet++] = x;
    */
}

int depile(t_pile* pp) {
    --pp->sommet;
    return pp->contenu[pp->sommet];
    /* pour les voltigeurs :
    return pp->contenu[--pp->sommet];
    */
}

int lit_sommet_pile(t_pile* pp) {
    return pp->contenu[pp->sommet - 1];
}

```

Écrivez une fonction **main** tester la pile ainsi créée.

Que se passe-t-il lorsqu'on empile sur une pile pleine ?

Écrivez une fonction **empile\_securit** qui empile une valeur sur la pile si elle n'est pas pleine, mais ne fait rien sur une pile pleine (l'écriture d'une fonction annexe **est\_pleine\_pile** sera peut-être utile).

Peut-on imaginer une valeur de retour pour cette fonction indiquant si l'empilement a bien pu se faire ?

De même, écrivez une fonction **depile\_securit** qui dépile sur une pile non vide, mais ne fait rien sur une pile vide.

Peut-on imaginer une valeur de retour pour cette fonction indiquant si le dépilement a bien pu se faire ?

#### Corrigé

\*\*\*\*\*

```

#include <stdio.h>

#define TMAX 10

typedef struct {
    int tab[TMAX]; /* le tableau contient les donnees */
}

```

```
int curseur; /* le curseur contiendra l'indice du tableau
              auquel le prochain empilement pourra se faire
*/
} pile;

int retire_pile(pile* p);
void ajoute_pile (pile* p, int n);
int sommet_pile(pile* p);
void affiche_pile(pile* p);
int pile_vide(pile* p);
int pile_pleine(pile* p);

void ajoute_pile (pile* p, int n) {
    /*
    l'empilement consiste a inscrire dans le tableau a l'emplacement du curseur
    la valeur a empiler puis a incrementer le curseur
    (postincrementation : p->curseur++)
    Pas de test de pile pleine (a faire avant si doute).
    */
    p->tab[p->curseur++] = n;
}

int retire_pile(pile* p) {
    /*
    le depilement consiste a decrementer le curseur
    (predecrementation : --p->curseur)
    puis a retourner la valeur du tableau situee a cet emplacement.
    Pas de test de pile vide (a faire avant si doute).
    */
    return p->tab[--p->curseur];
}

int sommet_pile(pile* p) {
    /*
    comme le curseur est situe une position apres le dernier empilement,
    il faut retourner l'emplacement precedent
    Pas de test de pile vide (a faire avant si doute).
    */
    return p->tab[p->curseur - 1];
}

void affiche_pile(pile* p) {
    /*
    Exemples :
        (1, 2, 3, 4) // pile normale
        (1) // un seul element
        () // pile vide

    afficher parenthese ouvrante
    afficher le premier,
    puis afficher les suivants precedes d'une virgule
    afficher parenthese fermante
    */
    int i;
    printf("(");
    if (! pile_vide(p)) {
        printf("%d", p->tab[0]);
        for (i = 1; i < p->curseur; ++i) {
            printf(", %d", p->tab[i]);
        }
    }
    printf(")");
}

int pile_vide(pile* p) {
    /*
    si curseur a 0, la pile est vide
    */
}
```

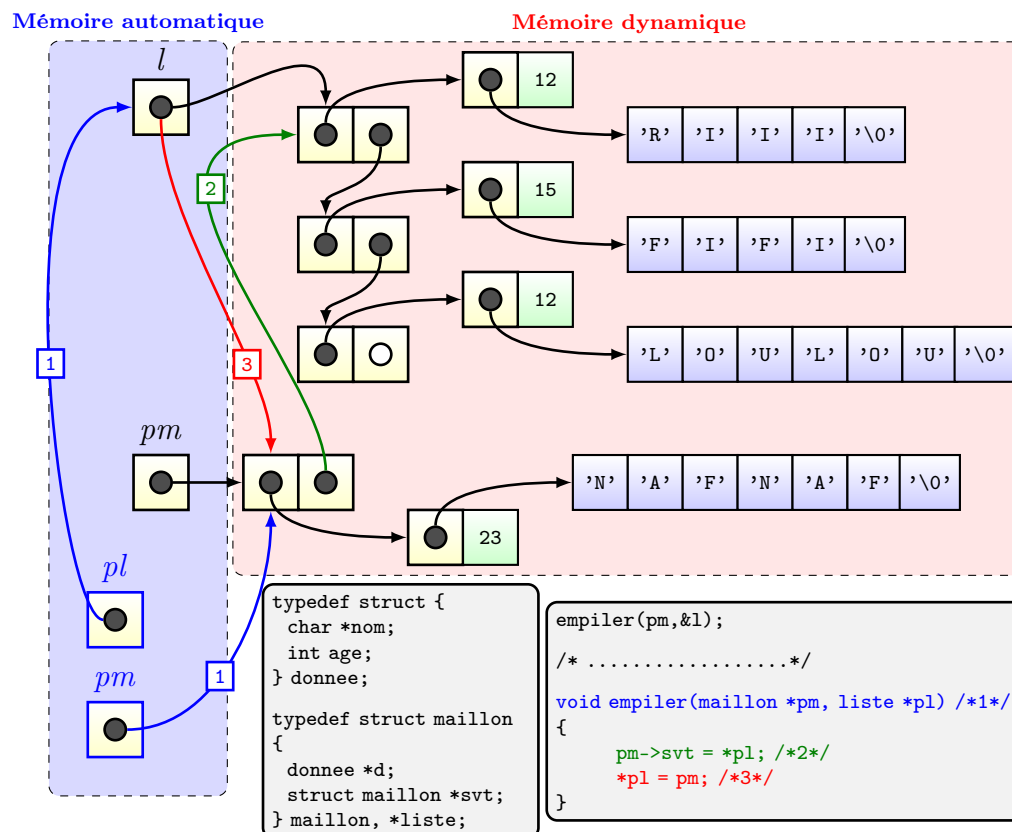
```
    */
    return p->curseur == 0;
}
int pile_pleine(pile* p) {
    /*
    si curseur a TMAX, impossible d'empiler
    */
    return p->curseur == TMAX;
}

int main() {
    /*
    notez qu'il n'ya pas besoin de connaitre la structure interne de la pile.
    Le meme programme pourra etre utilise avec une autre implementation.
    */
    int c, n;
    pile p;
    do {
        /*
        exemple de menu
        */
        printf("\n");
        printf("1. Empiler \n");
        printf("2. Depiler \n");
        printf("3. Sommet \n");
        printf("4. Affichage \n");
        printf("\n");
        printf("0. Quitter \n");
        printf("Votre choix : ");
        scanf("%d", &c);
        switch(c) {
            case 1 :
                if (pile_pleine(&p)) {
                    printf("Pile pleine. Empilement impossible\n");
                    break;
                }
                printf("Empilement. Entrez le nombre a empiler : ");
                scanf("%d",&n);
                ajoute_pile(&p, n);
                break;
            case 2 :
                if (pile_vide(&p)) {
                    printf("Pile vide. Depilement impossible\n");
                    break;
                }
                printf("Depilement de %d\n", retire_pile(&p));
                break;
            case 3 :
                if (pile_vide(&p)) {
                    printf("Pile vide. Pas de sommet ");
                    break;
                }
                printf("Sommet :  %d\n", sommet_pile(&p));
                break;
            case 4 :
                printf("Affichage :");
                affiche_pile(&p);
                printf("\n");
                break;
        }
    } while (c!=0);
    getchar(); /* pause */
    return 0;
}
```

\*\*\*\*\*

## Exercice 9. Implémentation par liste chaînée

## Pile : liste chaînée



Implémentez une gestion de pile selon le schéma ci-dessus.

Renseignez-vous pour savoir comment utiliser l'allocateur de mémoire standard (malloc/free)

## Corrigé

\*\*\*\*\*

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

/* enveloppe malloc/free pour pouvoir afficher les valeurs si DEBUG */
#define DEBUG

void* allocation_memoire(size_t t) {
    void* ret = malloc(t);
#ifdef DEBUG
    printf("Allocation donnee : %8p (%4d octets)\n", ret, t);
    getchar();
#endif
    return ret;
}

void liberation_memoire(void* p) {
#ifdef DEBUG
    printf("Liberation donnee : %24p\n", p);
    getchar();
#endif
    free(p);
}
```

```
/* les donnees a empiler */
typedef struct {
    char* nom;
    int age;
} donnee;

void donnee_affiche (donnee* d) {
    printf("%s %d", d->nom, d->age);
}

donnee* donnee_cree(char* nom, int age) {
    donnee* ret;
    printf("%s\n", nom);
    ret = allocation_memoire(sizeof(donnee));
    ret->age = age;
    ret->nom = allocation_memoire(strlen(nom) + 1);
    strcpy(ret->nom, nom);
    return ret;
}

void donnee_libere(donnee* d) {
    liberation_memoire(d->nom);
    liberation_memoire(d);
}

/* les maillons de chaines et les listes chainees */
typedef struct maillon {
    donnee* pc;
    struct maillon* svt;
} maillon, * liste;

void maillon_affiche (maillon* m) {
    donnee_affiche (m->pc);
}

maillon* maillon_cree(donnee* pc) {
    maillon* ret;
    ret = allocation_memoire(sizeof(maillon));
    ret->pc = pc;
    ret->svt = NULL;
    return ret;
}

void maillon_libere(maillon* pc) {
    liberation_memoire(pc);
}

void liste_init(liste* pl) {
    *pl = NULL;
}

void liste_ajoute_en_tete(liste* pl, maillon* m) {
    m->svt = *pl;
    *pl = m;
}

maillon* liste_retire_tete(liste* pl) {
    maillon* ret = *pl;
    *pl = ret->svt;
    return ret;
}

void liste_affiche(liste* pl) {
    maillon* m = *pl;
    while (m != NULL) {
        maillon_affiche(m) ; printf("\n");
        m = m->svt;
    }
}

/* et enfin la pile */
typedef liste pile;
```

```

void pile_empile(pile* pl, char* nom, int age) {
    donnee* d;
    maillon* m;
    d = donnee_cree(nom, age);
    m = maillon_cree(d);
    liste_ajoute_en_tete(pl, m);
#ifdef DEBUG
    printf("--->Empilement de "); maillon_affiche(m); printf("\n");
#endif
}

void pile_depile(pile* pl, char* pnom, int* page) {
    maillon* m = liste_retire_tete(pl);
#ifdef DEBUG
    printf("--->Retrait de "); maillon_affiche(m); printf("\n");
#endif
    donnee* pc = m->pc;
    maillon_libere(m);
    strcpy(pnom, pc->nom);
    *page = pc->age;
    donnee_libere(pc);
}

void pile_detruit(pile p) {
    maillon* q;
    donnee* d;
    while (p != NULL) {
        q = p->svt;
        d = p->pc;
        maillon_libere(p);
        donnee_libere(d);
        p=q;
    }
}

void pile_init(pile *p) {
    liste_init(p);
}

void pile_affiche(pile *p) {
    liste_affiche(p);
}

/*****
int main() {
    pile p;
    pile_init(&p);
    char nom[10]; int age;
    pile_empile(&p, "fifi", 15);
    pile_empile(&p, "riri", 12);
    pile_empile(&p, "loulou", 12);

    printf("\n\n      Affichage pile\n");
    pile_affiche(&p);
    printf("\n\n");

    pile_depile(&p, nom, &age);

    pile_empile(&p, "nafnaf", 18);

    printf("\n\n      Affichage pile\n");
    pile_affiche(&p);
    printf("\n\n");

    printf("\n\n      Destruction pile\n");
    pile_detruit(p);

    getchar();
    return 0;
}

```

\*\*\*\*\*