SCIENCES SORBONNE UNIVERSITÉ

**UFR d'Ingénierie**

10 janvier 2023

# L1 (2022-2023)

# *Éléments de programmation en C (LU1IN002)*

**TD**

**Semaines 1 à 11**

# Semaine 1 - TD

## Objectifs

— Fonctions
— Fonction d'affichage (fonction `printf`)
— Alternatives
— Directive **`#define`**
— Assert

## Exercices

### Exercice 1 – Premier programme C

Soient la fonction et le jeu de tests suivants écrits en Python.

```
def surface(l,L):
    """int * int -> int
    hypothèse : (L >= l) et (l >= 0)

    renvoie la surface du rectangle défini par sa largeur l et sa longueur L."""

    return (l*L)

# jeu de tests :
assert surface(0,0) == 0
assert surface(0,20) == 0
assert surface(1,1) == 1
assert surface(5,10) == 50
```

#### Question 1

Écrivez la fonction `surface` en C.

En C, la fonction `assert` prend en paramètre une expression, si cette dernière est vraie, le programme poursuit son exécution, sinon il s'arrête en affichant un message indiquant l'expression qui n'est pas vérifiée. Pour utiliser cette fonction, il est nécessaire d'utiliser la directive **`#include`** `<assert.h>`.

Considérons le programme suivant (écrit dans le fichier `exemple_assert.c`) :

```
#include <assert.h>

int addition(int a, int b){
    /* renvoie la somme a+b */
    return a + b;
}

int main() {
    assert(addition(1,1) == 2);
    assert(addition(-1,1) == 1);
    assert(addition(20,0) == 20);
    return 0;
}
```

Lors de l'exécution, le programme s'arrête sur le deuxième `assert` car l'expression passée en paramètre est fausse. Les instructions qui suivent un `assert` non vérifié ne sont pas exécutées. Le message suivant est affiché :

```
Assertion failed: (addition(-1,1)== 1), function main, file exemple_assert.c, line 12.
Abort
```

### Question 2

Écrivez le programme C complet permettant de tester la fonction `surface` de la question précédente en utilisant la fonction `assert`.

## Exercice 2 – Fonction d'affichage

### Question 1

Écrivez une fonction `multi` qui prend en paramètres un entier et un flottant et qui renvoie le résultat de la multiplication des deux valeurs.

La fonction `printf` permet d'afficher du texte à l'écran. Pour pouvoir l'utiliser il faut inclure la directive **`#include`** `<stdio.h>` en début de programme.

La fonction `printf` prend en paramètre une chaîne de caractères. L'exécution de la commande `printf("Ceci est du texte.\n");` affichera le texte `Ceci est du texte.` suivi d'un passage à la ligne représenté par le caractère `\n`.

Pour afficher la valeur de variables (ou la valeur de retour d'une fonction), il faut préciser dans le texte le format de la valeur que l'on veut afficher et faire suivre le chaîne de caractères de la liste des variables (ou appels de fonctions) dont on veut afficher la valeur (dans l'ordre d'apparition des formats).

- `%d` correspont à une valeur de type entier.
- `%f` correspont à une valeur de type flottant (affiche 6 chiffres après la virgule). La valeur affichée peut-être une valeur arrondie.
- `%c` correspont à une valeur de type caractère.
- `%s` correspont à une valeur de type chaîne de caractères.

Considérons le programme suivant :

```
#include <stdio.h>

int main() {
int i=2;
float a=1.27;
char k='h';

printf("entier : %d, flottant : %.10f, caractere : %c\n",i,a,k);
return 0;
}
```

Son exécution donnera l'affichage de la ligne :

```
entier : 2, flottant : 1.270000, caractere : h
```

Pour contrôler le nombre de chiffres affichés après la virgule lors de l'affichage d'un flottant, il suffit d'utiliser le format `%.xf` où `x` correspond au nombre de chiffres après la virgule souhaité.

Dans le programme précédent si on remplace le `%f` par `%.2f` on obtient l'affichage suivant :

```
entier : 2, flottant : 1.27, caractere : h
```

Si, dans le même programme, on remplace le `%f` par `%.1f` on obtient l'affichage suivant :

```
entier : 2, flottant : 1.3, caractere : h
```

### Question 2
Nous souhaitons maintennant compléter la fonction `main` suivante pour qu'elle affiche le résultat de la multiplication de `op1` et `op2` sous la forme `3 * 2.70 = 8.10`. Votre code doit bien sûr être indépendant des valeurs des variables `op1` et `op2` et doit faire appel à la fonction `multi`.

```
int main() {
  int op1;
  float op2;

  op1=3;
  op2=2.7;

  /* instruction d'affichage a ajouter */

  return 0;
}
```

## Exercice 3 – Le plus grand des trois

### Question 1
Écrivez une fonction `plusGrand` qui prend en paramètre trois entiers et qui renvoie le plus grand des trois.

### Question 2
Écrivez la fonction `main` permettant de tester le fonction `plusGrand`.

### Question 3
Donnez une instruction, faisant appel à la fonction `plusGrand`, permettant d'afficher l'entier le plus grand parmi cinq.

## Exercice 4 – Signe d'une somme

### Question 1
Écrivez la fonction `signeSomme` qui prend en paramètre deux entiers et qui, sans calculer la somme, renvoie 0 si la somme est nulle, -1 si elle est négative et 1 sinon.

### Question 2

Écrivez la fonction `main` qui permet de tester la fonction `signeSomme` en utilisant la fonction `assert`.

## Exercice 5 – Prix d'une place de cinéma

Un cinéma pratique les tarifs suivants (tarifs en juin 2018) :
- tarif normal : 11,40 €
- séance débutant avant 11 heures (et après 8 heures) : 7,10 €
- moins de 14 ans : 4,5 €
- moins de 26 ans du lundi au vendredi : 4,90 €
- moins de 26 ans samedi et dimanche : 7,90 €.

## Question 1

La directive **`#define`** permet d'associer une valeur à une chaîne de caractères. Lors de la compilation, chaque occurrence de la chaîne sera remplacée par la valeur.

La directive **`#define`** `TAILLE 3` associe la valeur 3 à la chaîne de de caractères `TAILLE`. Chaque apparition du mot `TAILLE` sera remplacée par la valeur 3. L'instruction `x=TAILLE;` revient donc à donner la valeur 3 à la variable `x`.

De cette façon, il est possible de nommer des « constantes » dont la valeur peut être modifiée en une seule écriture (dans la directive **`#define`**). Le code des fonctions sera donc indépendant de la valeur des constantes.

Nous utilisons cette directive pour définir les tarifs :

```
#define TNORMAL 11.4
#define TMOINS14 4.5
#define TMOINS26S 4.9
#define TMOINS26WE 7.90
#define TMATIN 7.10
```

Écrivez une fonction `prixPlace` qui :
- prend trois paramètres, deux entiers représentant un age et un jour (lundi=1, ..., dimanche=7) et un flottant correspondant à l'heure de début de la séance,
- renvoie le tarif appliqué.

Vous devez bien sûr calculer le tarif le plus avantageux pour le spectateur.

## Question 2

Écrivez la fonction `main` permettant de tester la fonction `prixPlace`. Étant donné que cette fonction renvoie un flottant, il n'est pas possible de tester directement cette valeur. Nous souhaitons que la fonction `main` affiche les valeurs calculées lors du jeu de tests.

# Semaine 2 - TD

## Objectifs

— Boucles
— Pile
— Comparaison de réels

## Exercices

### Exercice 6 – Boucles en C

Soit la fonction et le jeu de tests suivants écrits en Python.

```
def somme_carres(m,n):
    """int * int -> int
    hypothèse : m <= n

    renvoie la somme des carrés des entiers dans l'intervalle [m;n]."""

    # s : int
    s = 0   # la somme des carrés

    # i : int
    i = m   # entier courant de l'intervalle

    while i <= n:
        s = s + i * i
        i = i + 1
    return s

# jeu de tests :
assert somme_carres(1,5) == 55
assert somme_carres(2,5) == 54
assert somme_carres(3,5) == 50
assert somme_carres(4,5) == 41
assert somme_carres(5,5) == 25
assert somme_carres(3,6) == 86
assert somme_carres(-4,0) == 30
assert somme_carres(0,4) == 30
```

#### Question 1
Écrivez, en C, une fonction `somme_carres` qui a le même comportement que la fonction Python et qui est écrite avec une boucle **`while`**.

#### Question 2
Écrivez la fonction `main` permettant de tester la fonction de la question précédente en utilisant la fonction `assert`.

#### Question 3
Nous considérons maintenant la même fonction mais écrite avec une boucle **`for`**.

```
def somme_carres(m,n):
    """int * int -> int
    hypothèse : m <= n

    renvoie la somme des carrés des entiers dans l'intervalle [m;n]."""

    # s : int
    s = 0   # la somme des carrés

    # i : int
    # entier courant de l'intervalle

    for i in range(m,n+1) :
        s = s + i * i
    return s
```

De la même façon, écrivez une fonction C équivalente en utilisant une boucle **for**.

## Exercice 7 – Nombres premiers

Nous souhaitons afficher les nombres premiers inférieurs ou égaux à un entier donné (MAX).

Nous considérons le programme ayant la structure suivante dans lequel la primitive **#define** définit l'entier maximum :

```
#include <stdio.h>

#define MAX 5

... premier(...) {
  ...
}

... afficheNombresPremiers(...) {
  ...
}

int main(){

  printf("liste des nombres premiers <= %d\n",...);
  ...
  return 0;
}
```

### Question 1

La fonction `premier` prend en paramètre un entier (entier naturel par hypothèse) et renvoie 1 s'il est premier, 0 sinon. La fonction `afficheNombresPremiers` prend en paramètre un entier et affiche les nombres premiers inférieurs ou égaux au paramètre.

Donnez la signature des fonctions `premier` et `afficheNombresPremiers` et complétez la fonction `main`. Dans cette question il ne vous est pas demandé d'écrire les fonctions `premier` et `listeNombresPremiers`

### Question 2

Comment faire pour tester le programme sur une autre valeur que 5 ?

### Question 3

Écrivez une fonction `premier` qui prend en paramètre un entier (entier naturel par hypothèse) et qui renvoie 1 s'il

est premier, 0 sinon. Nous vous rappelons qu'un nombre premier est un entier naturel qui admet exactement deux diviseurs positifs, 1 et lui-même, 1 n'est donc pas premier. Nous pouvons déterminer qu'un nombre n'est pas premier dès qu'on a trouvé un de ses diviseurs (autre que 1 et lui-même).

### Question 4
Écrivez une fonction `afficheNombresPremiers` qui prend en paramètre un entier `n_max` et qui affiche les nombres premiers inférieurs ou égaux à `n_max`.

### Question 5
Donnez les évolutions de la pile d'exécution lors de l'exécution du programme lorsque la valeur associée à `MAX` est 3.

## Exercice 8 – Décomposition d'une somme en euros

### Question 1
Écrivez une fonction `afficheBilletsPieces` qui prend une somme entière en paramètre et qui affiche la décomposition de cette somme en un nombre minimal de billets-pièces de 5, 2 et 1 €.

**Indice :** il faut considérer le plus possible de billets de 5 €, puis le plus de pièces de 2 € et compléter par les pièces de 1 €. N'oubliez pas que la division entre deux entiers donne un résultat entier et que l'opérateur `%` permet d'obtenir le reste de cette division entière (`13/5 = 2` et `13%5 = 3`).

### Question 2
Écrivez une fonction `afficheBilletsPiecesMultiple` qui prend une somme entière en paramètre et qui affiche toutes les décompositions possibles de cette somme en billets-pièces de 5, 2 et 1 €.

## Exercice 9 – Nombres parfaits

### Question 1
Ecrivez une fonction `sommeDiviseurs` qui prend en paramètre un entier naturel et qui renvoie la somme de ses diviseurs stricts.

### Question 2
Un nombre est k-parfait si la somme de ses diviseurs est égale à k fois le nombre (vous remarquerez que cette fois on ne se limite pas aux diviseurs stricts). Ecrivez une fonction `k_parfait` qui prend deux entiers naturels en paramètre, `n` et `k`, et qui renvoie 1 si `n` est k-parfait, 0 sinon. Vous devez bien-sûr utiliser la fonction `sommeDiviseurs`.

### Question 3
Nous souhaitons maintenant trouver, pour un entier donné, la plus petite valeur de k ($k \in [k_{min}, k_{max}]$) pour laquelle l'entier est k parfait. Les valeurs de $k_{min}$ et $k_{max}$ seront définies par des primitives **`#define`**. Ecrivez la fonction `trouver_k_parfait` qui prend en paramètre un entier naturel `n` et qui renvoie la plus petite valeur de k telle que `n` est k-parfait. Si on ne trouve pas de valeur, la fonction renvoie -1.

## Exercice 10 – Comparaison de valeurs réelles

### Question 1
Nous considérons une fonction `surface_float`, similaire à la fonction `surface` écrite la semaine dernière, dont les paramètres et la valeur de retour sont de type **`float`**. Pourquoi est-il fortement déconseillé d'utiliser la fonction `assert` pour tester directement la fonction `surface_float` ?

Pour tester l'égalité entre flottants, nous devons déterminer si les deux valeurs son égales à *epsilon près* de façon similaire à ce que vous avez fait en Python au premier semestre.

### Question 2

Écrivez une fonction `valeur_absolue` qui prend un paramètre de type **`float`** et renvoie sa valeur absolue.

### Question 3

Écrivez une fonction `egal_eps` qui prend en parmètres trois valeurs de type **`float`**, deux valeurs à comparer et *epsilon* et qui renvoie 1 si les deux valeurs sont égales à *epsilon près*, 0 sinon.

### Question 4

Écrivez maintenant une fonction `main` permettant de tester la fonction `surface_float` en utilisant la fonction `assert`.

# Semaine 3 - TD

## Objectifs

— Pile
— Adresse
— Pointeur

## Exercices

### Exercice 11 – Appels de fonctions et pile

**Question 1**
Dites ce qu'affiche le programme suivant et donnez l'évolution de la pile d'exécution.

```
#include <stdio.h>

int diff(int a, int b) {
  int x = a -b;
  return x;
}

int calcul(int a, int b) {
   int x;

   if (a > b) {
      x = diff(a,b);
   }
   else {
      x = diff(b,a);
   }
   return x;
}

int main() {
   int res;

   res = calcul(7,2);
   printf("Le premier resultat du calcul est %d.\n",res);
   res = calcul(-15,3);
   printf("Le deuxieme resultat du calcul est %d.\n",res);
   return 0;
}
```

### Exercice 12 – Qu'est-ce qu'un pointeur ?

Soit le programme suivant.

**Question 1**
Donnez l'évolution de la pile d'exécution lors de l'exécution du programme suivant. Déduisez-en l'affichage obtenu.

```c
#include <stdio.h>

void ma_fonction (int a, int b) {
  int c1,c2;
  int *d;

  c1 = a + b;
  d = &a;
  *d = *d + 2;
  c2 = a + b;

  printf("a = %d, b = %d, c1 = %d, c2= %d, *d = %d\n",a,b,c1,c2,*d);
}

int main() {
  int a=7, b =10;

  printf("Avant appel : a=%d, b= %d\n",a,b);
  ma_fonction(a,b);
  printf("Apres appel : a=%d, b= %d\n",a,b);
  return 0;
}
```

## Exercice 13 – Pointeur et paramètres

Soit le programme suivant :

```c
#include <stdio.h>

void ma_fonction(int *param1, int param2) {
   int var_loc = 3;

   *param1 = var_loc * param2;
   param2 = var_loc + 1;
}

int main() {
   int v1, v2;

   v1 = 10;
   v2 = 3;
   ma_fonction(&v2,v1);
   return 0;
}
```

### Question 1

Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme précédent.

## Exercice 14 – Fonction mettant à jour plusieurs informations

Dans cet exercice nous supposons que nous avons à notre disposition la fonction `int valeur_aleatoire(int min, int max)` qui retourne un entier choisi aléatoirement entre les valeurs `min` et `max` comprises. Vous verrez en TP comment écrire cette fonction.

### Question 1

Dites ce que fait le programme suivant :

```c
#include <stdio.h>

#define NB_VALEURS 1000
#define VMIN 0
#define VMAX 32000

int valeur_aleatoire(int min, int max) {
  /* renvoie une valeur choisie aleatoirement entre min et max */
  ....
}

int minimum(int val, int valMin) {
  /* renvoie la plus petite valeur entre val et valMin */

  if (val < valMin) {
    return val;
  }
  return valMin;
}

int maximum(int val, int valMax) {
  /* renvoie la plus grande valeur entre val et valMax */

  if (val > valMax) {
    return val;
  }
  return valMax;
}

int main(){
  int i,val;
  int min=VMAX, max=VMIN;

  for (i=0; i < NB_VALEURS; i++) {
    val=valeur_aleatoire(VMIN,VMAX);
    min=minimum(val,min);
    max=maximum(val,max);
  }

  printf("MIN = %d, MAX = %d\n", min,max);
  return 0;
}
```

## Question 2

Écrivez une fonction `minimum_maximum` qui permet en un seul appel de mettre à jour le minimum et le maximum. Voici la fonction `main` dans laquelle vous devez aussi compléter l'appel à `minimum_maximum` :

```c
int main(){
  int i,val;
  int min=VMAX, max=VMIN;

  for (i=0; i < NB_VALEURS; i++) {
    val=valeur_aleatoire(VMIN,VMAX);
    minimum_maximum(........);
  }

  printf("MIN = %d, MAX = %d\n", min,max);
```

```
  return 0;
}
```

## Exercice 15 – Permutation circulaire

### Question 1

Ecrivez la fonction `permute` qui permet de permuter la valeur de deux variables. Si initialemenent `a=10` et `b=-20`, nous souhaitons que la valeur de ces variables après appel à la fonction `permute` soit `a=-20` et `b=10`.

### Question 2

Écrivez la fonction `permute_circulaire` qui permet de faire une permutation circulaire entre les valeurs de trois variables. Votre fonction doit faire appel à la fonction `permute`.

Si initialemenent `a=10`, `b=20` et `c=30`, nous souhaitons que la valeur de ces variables après appel à la fonction `permute_circulaire` soit `a=20`, `b=30` et `c=10`.

### Question 3

Soit la fonction `main` suivante. Complétez l'appel à la fonction `permute_circulaire` et donnez l'évolution de la pile d'exécution lors de l'exécution du programme.

```
int main(){
  int a, b, c;

  a=10; b=20; c=30;
  permute_circulaire(.....);
  return 0;
}
```

# Semaine 4 - TD

## Objectifs

— Tableaux : opérations de base
— Tableau en valeur de retour

## Exercices

### Exercice 16 – Affichage d'un tableau, échange du contenu de deux cases

#### Question 1
Écrivez une fonction qui affiche le contenu d'un tableau de flottants.

#### Question 2
Modifiez la fonction pour qu'elle affiche le contenu du tableau à raison de $p$ éléments par ligne.

#### Question 3
Écrivez une fonction qui échange le contenu de la case d'indice $i$ d'un tableau de flottants avec le contenu de la case d'indice $j$. Les valeurs de $i$ et $j$ sont supposées correctes (elles correspondent bien à des indices du tableau).

Écrivez une fonction `main` permettant de tester votre fonction.

### Exercice 17 – Un tableau est-il un pointeur ?

#### Question 1
Représentez l'état de la mémoire à la fin de l'exécution du programme ci-dessous :

```
#include <stdio.h>
#include <stdlib.h>
#define N 4

int main() {
    int i;
    int tab[N];
    for (i = 0; i < N; i++) {
        tab[i] = 2 * i + 1;
        printf("%d\t", tab[i]);
    }
    printf("\n");
    return 0;
}
```

#### Question 2
On modifie maintenant la déclaration du tableau : `int tab[N];` est remplacé par :

```
    int *tab;
    tab = malloc(N * sizeof(int));
```

Le programme est-il toujours correct ? Si oui, représentez l'état de la mémoire à la fin de l'exécution du programme, sinon expliquez pourquoi.

### Question 3

Le programme suivant est-il correct ? Si oui, qu'affiche-t-il ? Sinon, pourquoi ?

```
#include <stdio.h>
#define N 4

int main() {
    int i;
    int tab[N] = {1, 3, 5, 7};
    int *tab2;

    tab2 = tab;
    tab2[1] = 2;
    for (i = 0; i < N; i++) {
        printf("%d\t", tab[i]);
    }
    printf("\n");
    return 0;
}
```

## Exercice 18 – Initialisation d'un tableau

### Question 1

Écrivez un programme qui déclare un tableau statique de $N$ entiers, l'initialise avec les valeurs de 1 à $N$ et l'affiche.

Nous souhaitons maintenant pouvoir initialiser le tableau avec des valeurs tirées aléatoirement.

### Question 2

Écrivez une fonction `init_alea` qui initialise un tableau d'entiers avec des valeurs tirées aléatoirement entre deux entiers `MIN` et `MAX`. Complétez le programme précédent pour tester la fonction.

## Exercice 19 – Tableau en valeur de retour

### Question 1

Quel est le résultat de l'exécution de ce programme ?

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

int *tirage(int max, int chance_max) {
/* renvoie le tirage de 5 numeros + numero chance */
   int i;
   int res[6];

   for (i = 0; i < 5; i++) {
      res[i] = 1 + (rand() % max);
   }
   res[5] = 1 + (rand() % chance_max);
   return res;
```

```
}

int main() {
   int i;
   int *loto;

   srand(time(NULL));
   loto = tirage(49, 10);
   printf("Tirage du jour : ");
   for (i = 0; i < 5; i++) {
      printf("%d\t", loto[i]);
   }
   printf("\nNumero chance : %d\n", loto[5]);
   return 0;
}
```

# Semaine 5 - TD

## Objectifs

— Chaînes de caractères
— Tableau à deux dimensions

## Exercices

### Exercice 20 – Comptage du nombre d'occurrences d'une valeur dans une chaîne

Nous considérons une chaîne de caractères dans laquelle certains caractères peuvent apparaître plusieurs fois. Nous vous rappelons qu'une chaîne de caractères est un tableau de caractères dont le dernier caractère a la valeur '`\0`'.

**Question 1**
Écrivez une fonction `compte` qui compte le nombre d'occurrences d'un caractère `cr` dans une chaîne. Écrivez une fonction `affiche_occur` qui utilise la fonction `compte` pour afficher le nombre d'occurrences de chaque caractère d'une chaîne.
Écrivez une fonction `main` permettant de tester vos fonctions.

À moins d'avoir été particulièrment astucieux dans l'écriture de la fonction `affiche_occur`, lorsqu'un caractère est présent plusieurs fois dans le tableau, son nombre d'occurrences est affiché à chaque fois.

**Question 2**
Modifiez la fonction `affiche_occur` pour n'afficher qu'une fois le nombre d'occurrences de chaque caractère.

**Question 3**
Écrivez une fonction qui renvoie la position de la valeur la plus fréquente dans le tableau. Complétez la fonction `main` pour tester votre fonction.

**Question 4**
Proposez une version optimisée de la fonction en utilisant le même mécanisme que pour afficher une seule fois chaque valeur.

### Exercice 21 – Distances entre villes

Nous considérons un ensemble de `NB_VILLES` préfectures représentées par le code postal de leur département. Par exemple, le nombre `29`, code postal du Finistère, représente la ville de Brest. Les villes sont stockées dans un tableau à une dimension. Les distances séparant ces villes sont regroupées dans un tableau à deux dimensions. L'ordre dans lequel les distances sont rangées correspond à celui des villes.

Dans la suite de l'exercice, nous supposons que la directive `#define NB_VILLES 5` fixe le nombre de villes et que la fonction `main` dispose de deux tableaux déclarés ainsi :

```
int villes[NB_VILLES] = {29, 59, 67, 75, 83};

int distances[NB_VILLES] [NB_VILLES] = {
                 {0, 598, 900, 504, 995},
                 {598, 0, 407, 203, 861},
                 {900, 407, 0, 397, 621},
                 {504, 203, 397, 0, 694},
                 {995, 861, 621, 694, 0}
};
```

Le tableau `villes` contient les villes considées, à savoir dans l'ordre : Brest (`29`), Lille (`59`), Strasbourg (`67`), Paris (`75`) et Toulon (`83`). Les distances entre ces villes sont stockées dans le tableau `distances`, dans lequel l'ordre des colonnes est identique à celui des lignes. On voit par exemple que la distance entre Brest et Lille est de `598` kilomètres.

### Question 1

Écrivez une fonction **`void`** `affiche_distances(`**`int`** `villes, `**`int`** `dist[NB_VILLES][NB_VILLES])` qui affiche les distances entre les villes sous la forme suivante (vous noterez l'ajout de la première ligne de l'affichage et la représentation de la valeur non significative `0`) :

```
km      29      59      67      75      83
29      -       598     900     504     995
59      598     -       407     203     861
67      900     407     -       397     621
75      504     203     397     -       694
83      995     861     621     694     -
```

### Question 2

Écrivez une fonction **`int`** `plus_proche (`**`int`** `laville, `**`int`** `villes[], `**`int`** `dist[NB_VILLES][NB_VILLES])` qui utilise les deux tableaux `villes` et `distances` pour calculer la ville la plus proche de celle dont le code correspond au paramètre `laville`.

La fonction renvoie l'indice du tableau `villes` correspondant à la ville recherchée. Par exemple, si `laville` vaut `29`, la fonction renvoie `3` qui est l'indice dans `villes` (également indice de ligne dans le tableau des distances) correspondant à la ville la plus proche de Brest, à savoir Paris (distance : `504`). Si `laville` ne correspond pas à une ville du tableau, la fonction retourne `-1`.

## Exercice 22 – Chaîne en valeur de retour

Nous voulons écrire une fonction `int_to_str` qui prend en paramètre un entier et qui construit et renvoie la chaîne de caractères représentant cet entier.

### Question 1

Donnez le prototype de la fonction.

Puisque le contenu de la pile est perdu à la fin de la fonction, nous allons allouer dynamiquement (donc dans le tas) la zone mémoire destinée à stocker la chaîne de caractères.

La fonction doit donc commencer par compter le nombre de chiffres qui composent le nombre à transformer pour connaître la taille de la zone à allouer. Puis, en commençant par les unités, transformer chaque chiffre en caractère avant de le copier dans la chaîne. La règle qui permet de transformer un chiffre en caractère est simple : pour obtenir le code ASCII du caractère représentant un chiffre, il suffit d'ajouter `48` (ou `0x30`) à la valeur du chiffre. Par exemple, le code ASCII de `'3'` est `51` (ou `0x33`).

### Question 2

Écrivez le corps de la fonction, ainsi qu'un programme permettant de la tester.

## Exercice 23 – Minimum conditionnel d'un tableau

### Question 1

Écrivez une fonction qui calcule et renvoie la position du minimum dans un tableau d'entiers `tab` contenant `taille` éléments. La fonction respectera le prototype suivant :

```
int indice_min(int tab[], int taille);
```

Le premier paramètre de la fonction est le tableau et le second la taille de celui-ci. La valeur renvoyée est la position dans le tableau (indice) du minimum trouvé par la fonction.

### Question 2

Peut-on, en utilisant la fonction précédente, afficher la plus petite valeur contenue dans le tableau ?

### Question 3

Écrivez une fonction qui renvoie la position de la plus petite valeur *positive ou nulle* contenue dans un tableau d'entiers quelconques. La fonction renvoie -1 si aucune valeur satisfaisant la condition n'est trouvée.

Écrivez une fonction `main` permettant de tester votre fonction.

# Semaine 6 - TD

## Objectifs

— Arithmétique des pointeurs
— Récursivité sur les tableaux

## Exercices

### Exercice 24 – Comparaison de chaînes de caractères

**Question 1**
Écrivez une fonction *itérative* qui prend deux chaînes de caractères en paramètre. La fonction renvoie 1 si les chaînes sont identiques, 0 sinon.

Écrivez une fonction `main` permettant de tester votre fonction.

Nous souhaitons maintenant effectuer la comparaison au moyen d'une fonction récursive.

**Question 2**
Quels sont les cas d'arrêt ?

**Question 3**
Quelle expression représente l'adresse en mémoire de la case d'indice `i` du tableau ? Comment peut-on accéder au contenu de cette case sans utiliser la notation `tab[i]` ?

**Question 4**
Écrivez une version *récursive* de la fonction de comparaison.

### Exercice 25 – Recherche dichotomique d'un élément dans un tableau trié

La recherche dichotomique consiste à déterminer si un élément appartient à un tableau trié en divisant par deux l'intervalle de recherche à chaque itération. Les opérations à effectuer lors d'une itération sont donc :

- déterminer l'indice représentant le milieu de l'intervalle courant ; si élément à cette position est l'élément recherché, l'algorithme se termine avec une réponse positive ;
- sinon
  - si l'élément du milieu est supérieur à l'élément recherché, poursuivre la recherche dans l'intervalle à gauche de l'élément du milieu ;
  - si l'élément du milieu est inférieur à l'élément recherché, poursuivre la recherche dans l'intervalle à droite de l'élément du milieu ;

Nous voulons dans un premier temps réaliser cette recherche en utilisant une fonction *itérative*. Soient `g` et `d` les variables représentant les bornes gauche et droite respectivement de l'intervalle de recherche.

**Question 1**
Que valent :

- l'indice de l'élément au milieu de l'intervalle ?

- les bornes du nouvel intervalle de recherche si l'élément du milieu est supérieur à l'élément recherché ?
- les bornes du nouvel intervalle de recherche si l'élément du milieu est inférieur à l'élément recherché ?

### Question 2
À quel moment peut-on déterminer que l'élément recherché ne figure pas dans le tableau ?

### Question 3
Comment se traduit la condition précédente ?

### Question 4
Écrivez une fonction *itérative* qui utilise le principe de la dichotomie pour tester la présence d'un élément dans un tableau. La fonction renvoie 1 si l'élément est présent, 0 sinon.

Écrivez une fonction `main` permettant de tester votre fonction.

### Question 5
Écrivez une fonction *récursive* qui utilise le principe de la dichotomie pour tester la présence d'un élément dans un tableau. La fonction renvoie 1 si l'élément est présent, 0 sinon.

### Question 6
Quelles sont les valeurs que vous choisiriez pour tester votre programme ?

### Question 7
Lorsqu'on recherche un élément qui n'est pas présent dans le tableau, est-il plus intéressant d'utiliser une recherche dichotomique ou une recherche linéaire ?

# Semaine 7 - TD

## Objectifs

— Structures
— Tableaux de structures

## Exercices

### Exercice 26 – Structures et fonctions

L'objet de base que nous considérons dans cet exercice est le point. Il est défini par ses coordonnées x et y (de type entier) et une couleur d'affichage (de type chaîne de caractères).

**Question 1**
Expliquez pourquoi un point ne peut pas être représenté par un tableau à trois éléments. Quel type est alors le plus adapté ?

**Question 2**
Définissez le type `point` permettant de représenter un point. La taille de la chaîne de caractères représentant la couleur sera limitée à 10 caractères (n'oubliez pas de réserver la place pour le caractère '`\0`' qui signale la fin de chaîne). Déclarez et initialisez ensuite un point nommé `mon_point`, situé aux coordonnées `x=3`, `y=4` et de couleur rouge. Pour recopier une chaîne de caractères, vous devrez utiliser la fonction `strcpy` qui prend en paramètres deux chaînes de caractères et qui recopie la valeur de la deuxième chaîne dans la première.

**Question 3**
Nous voulons maintenant pouvoir décrire des rectangles dont les côtés sont horizontaux et verticaux. Un rectangle est alors déterminé par les coordonnées de deux coins opposés (en bas à gauche et en haut à droite), la couleur d'affichage des traits et celle du fond. Définissez une structure `rectangle` qui utilise la structure `point`.

**Question 4**
Déclarez et initialisez une variable `mon_rectangle` de type `rectangle` dont les deux sommets opposés sont respectivement aux coordonnées (100, 200) et (300, 2), la couleur d'affichage des traits et des points est le rouge et la couleur du fond est le blanc. Nous vous rappelons que le repère associé à une fenêtre a son origine dans le coin en haut à gauche (voir figure 1).

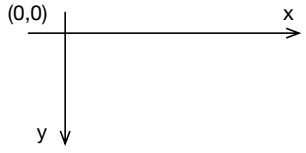


FIGURE 1 – Repère associé à une fenêtre

**Question 5**
Écrivez la fonction `point_dans_rectangle` qui prend un point et un rectangle en paramètres et qui renvoie 1 (vrai) si le point est dans le rectangle (les bords du rectangle sont dans le rectangle) et 0 (faux) sinon.

## Question 6

Écrivez la fonction `intersection_rectangles` qui prend en paramètres deux rectangles et qui renvoie le rectangle se trouvant à l'intersection des deux. Nous ferons l'hypothèse que l'intersection des deux rectangles passés en paramètre n'est pas vide. La couleur associée à tous les éléments de ce nouveau rectangle sera le noir (couleur des points, des traits et du fond).

Si les deux rectangles `r1` et `r2` de la figure 2 sont passés en paramètre, la fonction `intersection_rectangles` doit renvoyer le rectangle gris.

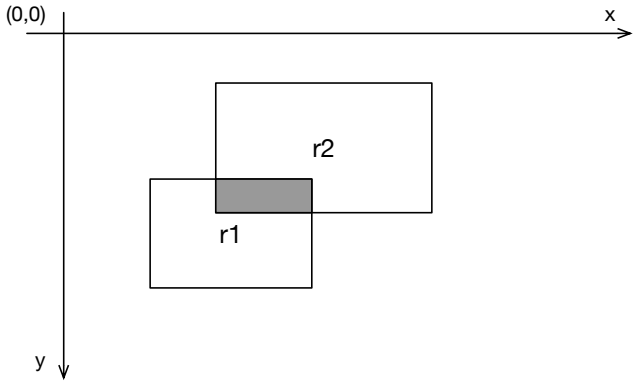


FIGURE 2 – Intersection de deux rectangles

Pour écrire la fonction `intersection_rectangles`, vous supposerez que vous avez à votre disposition la fonction `minimum` (resp. `maximum`) qui prend deux entiers en paramètres et qui renvoie le plus petit (resp. grand) des deux.

## Question 7

Donnez l'évolution de la pile d'exécution lors de l'exécution du `programme` suivant.

```
#include <stdio.h>

/* déclaration des types point et rectangle */

/* définitions des fonctions minimum, maximum
   et intersection_rectangles */

int main(){
  rectangle rectangle1;
  rectangle rectangle2;
  rectangle rectangle3;

  /* initialisation rectangle1
  point_bas_gauche = (100,300,"noir")
  point_haut_droite = (300,200,"noir")
  couleur_fond = "bleu"
  couleur_trait = "jaune" */

  /* initialisation restangle2
  point_bas_gauche = (200,250,"noir")
  point_haut_droite = (350,100,"noir")
  couleur_fond = "rouge"
  couleur_trait = "vert" */

  rectangle3=intersection_rectangles(rectangle1,rectangle2);

  return 0;
```

```
}
```

## Exercice 27 – Gestion de stock

Nous considérons la gestion du stock d'un commerce. Chaque article du stock est représenté par une référence (un entier), un prix (un flottant) et une quantité (un entier correspondant au nombre d'exemplaires en stock). Le stock est représenté par un tableau d'articles. Nous considérons que la taille du tableau correspond exactement au nombre d'articles référencés.

### Question 1
Définissez le type `article` qui permet de définir un article du stock.

### Question 2
Soit la déclaration **`#define`** `TAILLE_STOCK 40` qui permet de définir le nombre d'articles référencés (40 dans ce cas). Déclarez le tableau `stock` qui permettra de contenir tout le stock.

### Question 3
Écrivez la fonction `augmentationPrix` qui permet d'augmenter le prix d'un article en appliquant le taux d'augmentation (pourcentage) passé en paramètre. La fonction ne renvoie rien, elle doit donc avoir comme deuxième paramètre une information permettant d'accéder à l'article dont on veut modifier le prix.

### Question 4
Écrivez la fonction `augmentationGenerale` qui permet d'augmenter le prix de tous les articles en appliquant le taux d'augmentation passé en paramètre. Cette fonction doit faire appel à la fonction `augmentationPrix`.

### Question 5
Donnez l'évolution de la pile d'exécution lors de l'exécution du `programme` suivant.

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

#define TAILLE_STOCK 2

/* déclaration du type article */

/* définition des fonctions augmentationPrix et
   augmentationGenerale */



int main() {
  article stock[TAILLE_STOCK]={{152,10.5,10},{432,24.3,50}};


  augmentationGenerale(stock,TAILLE_STOCK,10.0);

  return 0;
}
```

### Question 6
Écrivez la fonction `validationReferences` qui prend en paramètre un stock et qui renvoie 1 si toutes les références sont différentes et 0 sinon.

# Semaine 8 - TD

## Objectifs

— Introduction aux listes chaînées
— Représentation de listes d'entiers
— Les différents parcours de liste

## Exercices

Dans les exercices de cette semaine, nous allons implémenter en langage C les opérations et fonctions que vous réalisiez en Python sur les listes. Dans un premier temps nous nous intéresserons uniquement aux listes d'entiers.

Les exercices s'appuieront sur la structure de données suivante :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
  int donnee;
  cellule_t *suivant;
};
```

## Exercice 28 – Représentation et affichage d'une liste

L'objectif de cet exercice est de manipuler une structure simple de liste, de comprendre sa représentation en mémoire et de voir un premier parcours de liste.

### Question 1

Écrivez une fonction `Creer_cellule` qui alloue et renvoie un pointeur vers une `cellule_t` dont le champ `donnee` sera initialisé avec un entier `d` donné en argument et le champ `suivant` à `NULL`.

Soit le programme suivant :

```
void Afficher_liste_int(cellule_t *liste){
/* Affiche les champs donnee des elements de la liste */
   cellule_t *cell = liste;
   while (cell != NULL) {
      printf("%d ", cell->donnee);
      cell = cell->suivant;
   }
   printf("\n");
}

int main() {
   cellule_t *nCell1=NULL, *nCell2=NULL, *nCell3=NULL;

   nCell1 = Creer_cellule(1);
   nCell2 = Creer_cellule(2);
   nCell3 = Creer_cellule(3);
   /* Breakpoint 1 ICI, Dessiner la memoire, tas et pile*/

   nCell1->suivant = nCell2;
   nCell2->suivant = nCell3;
```

```
    /* Breakpoint 2 ICI, Dessiner la memoire, tas et pile*/
    /* Afficher la liste en partant de chaque variable, definir la tete et la queue */
    Afficher_liste_int(nCell1);
    return 0;
}
```

**Question 2**
Représentez l'état de la mémoire (pile et tas) avant l'exécution de l'instruction **`return`** de la fonction `Creer_cellule` lors de l'appel `Creer_cellule(1)`, puis après l'exécution de l'appel.

**Question 3**
Représentez l'état de la mémoire (pile et tas) au *breakpoint 1* puis au *breakpoint 2*. Que fait ce programme ?

**Question 4**
Qu'affiche ce programme ?
Qu'affiche-t-il si l'appel à `Afficher_liste_int(nCell1)` devient `Afficher_liste_int(nCell2)` ?

**Question 5**
Que se passe-t-il si on ajoute les lignes suivantes à la fin du programme ?

```
nCell1->suivant = nCell3;
Afficher_liste_int(nCell1);
```

**Question 6**
Que se passe-t-il si on ajoute ensuite les lignes suivantes à la fin du programme :

```
nCell3->suivant = nCell2;
Afficher_liste_int(nCell1);
```

**Question 7**
Ré-écrivez la fonction `main` telle qu'elle a été donnée en question 2 mais avec une seule variable de type `cellule_t*`, variable que nous nommerons `tete`.

## Exercice 29 – Parcours de liste

**Question 1**
Écrivez une fonction `len` qui prend en paramètre une liste de `cellule_t` et qui renvoie son nombre d'éléments. Ajoutez à la fonction `main`, du corrigé de la denière question de l'exercice précédent, une instruction permettant de tester votre fonction `len`.

**Question 2**
Écrivez une fonction `existe` qui prend en paramètre une liste de `cellule_t` et un entier `val`. La fonction renvoie `0` (faux) si la valeur `val` n'existe pas dans la liste et `1` sinon.

**Question 3**
Modifiez la fonction `existe`, pour qu'elle renvoie un pointeur vers le premier élément de valeur `val` dans la liste. La fonction renvoie `NULL` si aucun élément n'a cette valeur. Vous appellerez cette nouvelle fonction `Renvoyer_element_debut`. Donnez des instructions permettant de vérifier que la fonction a bien renvoyé un pointeur sur un élément ayant la bonne valeur.

**Question 4**
Modifiez la fonction `Renvoyer_element_debut` pour qu'elle renvoie un pointeur vers le **dernier** élément de valeur

`val` dans la liste. La fonction renvoie `NULL` si aucun élément n'a cette valeur. Vous appellerez cette nouvelle fonction `Renvoyer_element_fin`.

Comment faire pour tester que la fonction `Renvoyer_element_debut` renvoie bien le premier élément rencontré et la fonction `Renvoyer_element_fin` le dernier ?

### Question 5

Dans le cas d'une liste non vide, quelle information vous permet de savoir qu'un élément est le dernier de la liste ?

Écrivez une fonction `Renvoyer_dernier_element` qui renvoie un pointeur vers le dernier élément de la liste. La fonction renvoie `NULL` si la liste est vide. Complétez la fonction `main` pour tester votre fonction.

### Question 6

Écrivez une fonction `Modifier_element(int val, int pos, cellule_t* liste)` qui affecte la valeur `val` à l'élément en position `pos` de la liste donnée en argument. Si la liste comporte moins de `pos` éléments (par analogie avec les tableaux, on suppose que le premier élément correspond à la position `0`), elle n'est pas modifiée. Nous ferons l'hypothèse que `pos` est supérieur ou égal à 0.

# Semaine 9 - TD

## Objectifs

— Ajout d'un élément dans une liste
— Suppression d'un élément d'une liste
— Parcours simultané de plusieurs listes

## Exercices

Les fonctions que nous allons écrire cette semaine modifient la liste sur laquelle elles s'appliquent soit en y ajoutant un élément soit en en supprimant un. Si l'ajout se fait en tête de liste ou si le premier élément de la liste est supprimé, la tête de liste est modifiée, il faut donc récupérer sa nouvelle valeur. Ces fonctions vont donc prendre en paramètre un pointeur sur un élément de liste (le premier élément de la liste avant modification) et retourner un pointeur sur un élément de liste (le nouveau premier élément de la liste modifiée).
**Attention**, ces fonctions modifient la liste initiale et renvoient le pointeur sur le premier élément de la liste après modification. La liste initiale n'existe plus après appel à l'une de ces fonctions, elle a été modifiée.

## Exercice 30 – Insertions

Dans cet exercice, nous allons commencer à construire des listes. Une liste se construit en ajoutant des éléments à une liste existante, éventuellement vide.

Il existe 3 types d'insertion dans les listes : au début (en tête), à la fin (en queue) ou à une position spécifique donnée. Nous allons implémenter chacun des 3 cas.

Nous ferons appel à la fonction `Creer_cellule` qui alloue et renvoie un pointeur vers une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` donné en argument et le champ `suivant` à `NULL`.

### Question 1

Écrivez une fonction `Inserer_tete` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en tête de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

### Question 2

Écrivez une fonction `Inserer_fin_it` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en queue de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

### Question 3

Qu'affiche la fonction `main` suivante ?

```
int main() {
   cellule_t *tete = NULL;
   int i;

   for(i = 0; i < 5; i++) {
      tete = Inserer_tete(i,tete);
   }
   Afficher_liste_int(tete);
   for(i = 5; i < 10; i++) {
      tete = Inserer_fin_it(i,tete);
   }
   Afficher_liste_int(tete);
```

```
    return 0;
}
```

### Question 4

Écrivez une fonction `Inserer_en_pos(int d, int pos, cellule_t *liste)` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec l'entier `d` passé en argument. La fonction insère la nouvelle cellule à la position `pos` de `liste` et renvoie un pointeur vers la nouvelle liste ainsi formée. Nous ferons l'hypothèse que `pos` est supérieur ou égal à 0.

Par analogie avec les indices de tableaux, on notera `0` la position du premier élément. Si la valeur `pos` est supérieure au nombre d'éléments dans la liste, l'élément sera ajouté à la fin.

Par exemple, si on insère un élément de valeur 15 en position 5 de la liste `1 2 3 4 5 6 7 8 9 10`, la liste résultante contiendra les valeurs `1 2 3 4 5 15 6 7 8 9 10`.

La Figure 3 donne un état de la mémoire au cours de l'exécution d'un programme. *@ti* représente l'adresse dans le tas d'une zone allouée pour contenir une `cellule_t)`.

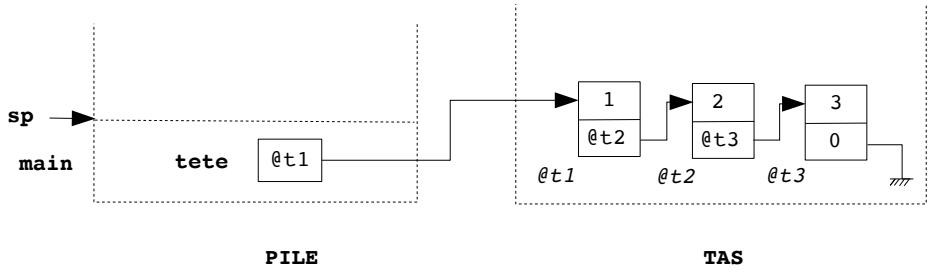


FIGURE 3 – état mémoire à l'appel de `Inserer_en_pos`

### Question 5

On suppose qu'à ce moment, on exécute l'instruction `tete = Inserer_en_pos(-1, 2, tete)`. Représentez l'état de la mémoire à l'entrée de la fonction, puis juste après sa terminaison.

### Question 6

Ajoutez dans la fonction `main` les instructions permettant de vérifier que votre fonction se comporte correctement dans tous les cas.

## Exercice 31 – Suppression

L'objectif de cet exercice est de savoir détruire (désallouer) des éléments de listes ou des listes entières.

### Question 1

Écrivez une fonction `cellule_t* Liberer_liste(cellule_t* liste)` qui libère (désalloue) tous les éléments de la liste `liste`. Cette fonction renverra `NULL` pour garantir que la variable de la fonction appelante qui récupère le résultat ne pointe pas sur une zone mémoire désallouée.

On considère l'état mémoire de la Figure 4, obtenu après l'appel de fonction `tete = Liberer_liste(tete)` (`tmp` est la variable locale qui mémorise l'adresse de l'élément à désallouer).

### Question 2

Représentez l'état de la mémoire à la fin de la première itération (désallocation du premier élément).

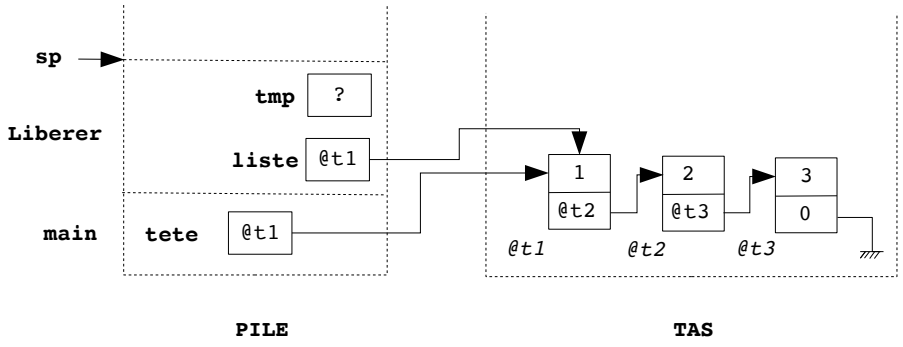


FIGURE 4 – état mémoire à l'entrée de la fonction `Liberer_liste`

### Question 3

Écrivez une fonction `cellule_t* Supprimer_en_pos(int pos, cellule_t *liste)` qui supprime (désalloue) l'élément de la liste en position `pos`. Si l'indice est trop grand, la fonction ne modifie pas la liste. Nous faisons l'hypothèse que `pos` est supérieur ou égal à 0.

## Exercice 32 – Parcours de plusieurs listes

Dans cet exercice, nous allons allons écrire des fonctions nécessitant de parcourir plusieurs listes simultanément.

### Question 1

Écrivez une fonction `listes_identiques` qui prend en paramètre deux listes d'entiers et qui renvoie 1 si les listes sont identiques, 0 sinon. Deux listes son identiques si elles contiennent exactement les mêmes éléments et dans le même ordre.

### Question 2

Écrivez une fonction `listes_incluses` qui prend en paramètre deux listes d'entiers et qui renvoie 1 si la première liste est incluse dans le seconde (ou égale), 0 sinon. Nous ferons l'hypothèse que les listes sont triées en ordre croissant.

# Semaine 10 - TD

## Objectifs

— Récursivité
— Extraction d'une sous-liste

## Exercices

### Exercice 33 – Manipulations récursives sur les listes

Une liste est une structure intrinsèquement récursive puisqu'elle est composée d'un élément suivi d'une liste (ou l'inverse...). Beaucoup de fonctions s'écrivent donc assez naturellement sous forme récursive. Nous allons appliquer cette approche à quelques unes des opérations que nous avons déjà programmées.

#### Question 1

Écrivez une fonction **récursive** `cellule_t* Renvoyer_element_debut_rec(int val, cellule_t* liste)` qui renvoie l'adresse de la première occurrence de `val` dans la liste. La fonction renvoie `NULL` si la valeur recherchée n'est pas dans la liste.

#### Question 2

Écrivez une fonction **récursive** `cellule_t *Inserer_fin_rec(int d, cellule_t *liste)` qui crée une `cellule_t` dont le champ `donnee` est initialisé avec un entier `d` passé en argument. La fonction insère la nouvelle cellule en fin de la liste passée en argument et renvoie un pointeur vers la nouvelle liste ainsi formée.

On considère la fonction `main` suivante, l'état de la mémoire avant l'appel à `Inserer_fin_rec` est donné en Figure 5.

```
int main() {
   cellule_t *tete = NULL;

   tete = Creer_cellule(1);
   tete->suivant = Creer_cellule(2);

   tete = Inserer_fin_rec(3, tete);
   Afficher_liste_int(tete);
   return 0;
}
```

#### Question 3

Déroulez l'exécution de l'appel à `Inserer_fin_rec(3, tete)`

#### Question 4

Écrivez une fonction **récursive** `Supprimer_en_pos_rec(int pos, cellule_t *liste)` qui supprime (désalloue) l'élément de la liste en position `pos`. Si l'indice est trop grand, aucune désallocation n'est effectuée.

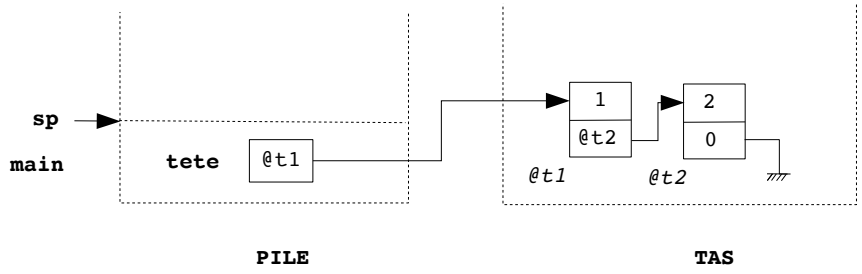


FIGURE 5 – état mémoire avant l'appel à `Inserer_fin_rec`

## Exercice 34 – Extraction d'une sous liste

Nous allons extraire une sous-liste d'une liste de deux façons :
- en construisant une nouvelle liste contenant les éléments que nous souhaitons garder
- en supprimant de la liste les éléments que l'on ne souhaite pas garder.

### Question 1
Écrivez une fonction **itérative** `Creer_liste_positifs` qui prend en paramètre une liste d'entiers, qui construit la liste contenant les éléments strictements positifs de la liste initiale et qui renvoie la tête de cette nouvelle liste (qui sera égale à `NULL` si aucun entier de la liste n'est strictement positif). La liste passée en paramètre n'est pas modifiée. Vous appellerez la fonction `Inserer_tete` pour ajouter un élément en tête de liste.

### Question 2
Donnez une version **récursive** de la fonction `Creer_liste_positifs`. Vous appellerez la fonction `Inserer_tete` pour ajouter un élément en tête de liste.

### Question 3
Quelle différence sur le résultat obtenu constatez-vous entre les deux fonctions précédentes ?

### Question 4
Écrivez une fonction `Garder_positifs` qui prend en paramètre une liste d'entiers et qui supprime de cette liste tous les éléments qui ne sont pas strictement positfs. La fonction renvoie la nouvelle tête de liste.

# Semaine 11 - TD

## Objectifs

— Listes
— Révisions

## Exercices

### Exercice 35 – Parcours de listes de types différents

Dans cet exercice nous allons construite un multi-ensemble trié par ordre croissant des valeurs à partir d'une liste d'entiers triée elle aussi par ordre croissant.

Les éléments d'une liste d'entiers sont du type suivant :

```
typedef struct _cellule_t cellule_t;
struct _cellule_t{
  int donnee;
  cellule_t *suivant;
};
```

Ceux d'un multi-ensemble sont du type suivant :

```
typedef struct _element_t element_t;
struct _element_t{
  int valeur;
  int frequence;
  element_t *suivant;
};
```

#### Question 1

Écrivez la fonction `Creer_multi_ensemble_liste` qui prend en paramètre une liste d'entiers triés en ordre croissant (un même entier peut apparaître plusieurs fois) et crée le multi-ensemble correspondant triée par ordre croissant des valeurs. La fonction renvoie un pointeur sur le premier élément du multi-ensemble.