Le langage C

Partie du cours d'OS sur le langage C ATTENTION, ce chapitre n'est pas complet

Introduction aux laboratoires

Outils

- Linux (une machine virtuelle est disponible sur l'espace de cours)
- Clion (Jetbrains) est l'IDE recommandé pour le C et C++ (mais bon vim, emacs, helix et tout sont bien aussi hein)

Exam

L'examen de janvier se fait par deux interogation :

- Interogation C le 25/10
- Interogation OS le 20/12

La seconde session se fait en examen en session à l'interrogation OS. A savoir que les intérogations se font normalement à cours ouvert.

Ressourcess

- Vidéos de Swinnen sur le Swilabus
- Les PDF de théorie de chaque chapitre sur l'espace de cours
- Les séances de laboratoires pour avoir des informations complémentaires

Conseils

- Lire le document du chapitre avant le cours
- Regarder la vidéo de Swinnen qui corresponds
- Assister aux séances de labo pour avoir des informations complémentaires et faire les exercices en présence du prof et des autres élèves.
- Bien suivre le calendrier de cours disponible sur l'espace du cours

> Note : Le labo 2 a été annulé

Setup

Virtual machine

Il est recommandé d'utiliser la machine virtuelle Fedora disponible sur l'espace de cours qui vient préinstallée avec l'IDE "Clion" de Jetbrains.

Pour l'ajouter vous devrez installer le logiciel gratuit (et open source) Virtual Box dans laquelle vous pouvrez ensuite importer la VM et la lancer.

Avec un IDE (clion)

Si vous êtes sur macOS ou Linux cependant, vous pouvez aussi simplement installer Clion car macOS et Linux sont des systèmes UNIX-like, ce qui signifie qu'ils sont compatibles.

Ensuite il ne faut pas oublier de suivre les étapes données sur l'espace de cours pour y inclure les flags de compilations (qui vont donner des règles supplémentaires et afficher des warnings lors de la compilation de vos programmes).

Manuellement

Si vous êtes sur Linux, WSL (Windows Subsystem for Linux) ou macOS, vous pouvez aussi ne pas utiliser d'IDE et installer `gcc` (version 13.2) et y inclure les flags de compilation.

Par exemple vous pouvez faire un alias de la commande gcc comme suit :

```
echo 'alias gcc="gcc -std=iso9899:1990 -Wpedantic -Wall -Werror"' >> ~/.bashrc
source ~/.bashrc
```

Vous pouvez avoir plus d'information sur l'utilisation de gcc en consultant son manuel d'utilisation (`man gcc`) ou en allant voir sur internet comment l'utiliser (vous pouvez aussi aller voir une cheatsheet ici).

Introduction au C

Qu’est ce que le C

Le langage C est un langage de bas niveau (contrairement à Java qui est plus un langage de haut niveau). Le langage C est de moins en moins utilisé directment mais de nombreux langages ont été fait à partir de C tel que C++, Java, PHP, Python ou PERL.

Paradigme impératif

Contrairement à Java qui fonctionne dans un paradigme orienté objet, C est un paradigme *impératif*, comme dans Java on manipule des données en indiquant instruction par instruction comment construire un résultat. Cependant à la différence de Java, il n’y a pas de notion d’objet, de polymorphisme, ou d’héritage (car cela est propre au paradigme *orienté objet*).

En C on manipule uniquement des types structurés (types très basique, comme des classes qui n’aurait aucune méthode), des fonctions, des tableaux, des chaines de caractères ou des pointeurs.

Histoire du C

Le C a été développé dans les années 70 dans le but de créer un langage plus adapté pour écrire la nouvelle version d’un système d’exploitation nomé Unix, qui sera le parent (indirect) des systèmes Linux, BSD ou encore macOS. Même si C a été développé il y a pas mal de temps, il continue toujours à évoluer aujourd’hui.

Utilisation du C

Le langage C est beaucoup utilisé pour tout ce qui touche au système d’exploitation, et également dans des ordinateurs qui n’ont que très peu de ressources tel que les Arduino ou les Raspberry Pi.

Objectifs du cours de C

Le but du cours sur le C n’est pas de devenir programmeur C mais bien de pouvoir créer de petites applications systèmes en C. Il est égalment important de faire un certain travail à domicile car les 14 heures de laboratoires ne seront pas suffisantes pour bien comprendre les concepts.

Environement de développement et configuration

Il existe de très nombreux environements de développement permettant de coder en C. Notament VS Code, vim, emacs, helix, clion ou encore Code Blocks. Les éditeurs recommandés pour le cours sont Code Blocks (gratuit et open source) et/ou clion (un éditeur de JetBrains propriétaire et payant, cependant des clés d'accès sont fournie par HELMo).

Quoi qu'il arrive, votre environement de développement doit être configuré pour :

- Utiliser `gcc 13.2`
- Utiliser les flags de compilation suivants : `-std=iso9899:1990 -Wpedantic -Wall -Werror` (plus d'information sur comment les configurer pour clion sont disponible sur la page du cours)

Hello World

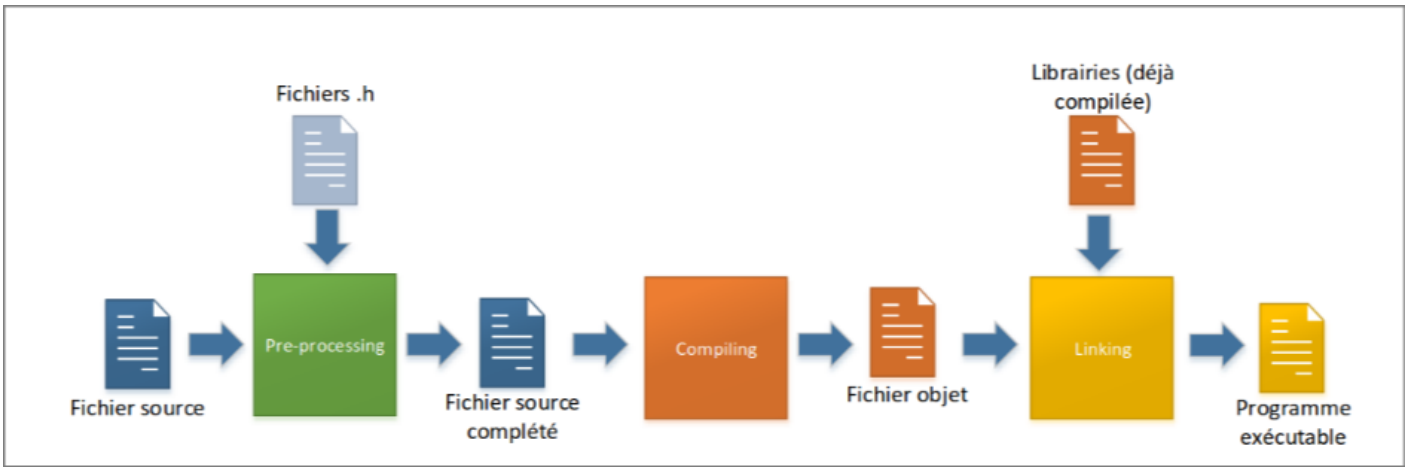
```
/* Les lignes commençant par # sont des directives au préprocesseur C
   Dans ce cas avec #include c'est une sorte d'import qui dit qu'il fait inclure une librairie. Dans ce cas on importe les librairies standard stdio et stdlib */
#include<stdio.h>
#include<stdlib.h>

/* Le programme principale exécuté se trouve dans la fonction main */
int main(void) {
    /* Printf vient de la librarie stdio et permet d'afficher du texte dans la console */
    printf("Hello World !\n");

    /* A la fin de tous les programmes en C, il retourne un entier. 0 dans le cas d'un succès (qui est déjà présent dans la constante de stdlib EXIT_SUCCESS); ou 1 dans le cas d'un échec (constante EXIT_FAILURE de stdlib).
    Cela permet d'indiquer à des programmes qui utiliserait celui-ci, si l'exécution s'est bien passée ou non */
    return EXIT_SUCCESS;
}
```

Libraries standard en C

Le langage C définit un certain nombre de librairies standard. Parmis celles ci, en voici 5 qui seront beaucoup utilisée dans ce cours d'introduction au C :

Librarie	Usage
stdio.h	Nécessaire pour les entrées sorties standard (gestion clavier et écran). **A inclure dans tous les programmes**
stdlib.h	Reprends les constantes et les fonctions importantes. **A inclure dans tous les programmes également**
string.h	Reprends les fonctions de manipulation de chaine de caractères (comparaison, copie, recherche, concaténation, etc)
math.h	Reprends les fonctions mathématiques (puissances, trigonométrie, etc)
time.h	Manipulation de la date et de l'heure

Processus de compilation



- Tout commence avec les fichiers source, c'est à dire les fichiers d'extension `.c`.
- Ensuite la phase de **pre-processing** va inclure les fichiers en-tête (sur lesquels on va revenir plus tard, mais ceux-ci contiennent les signatures des fonctions des libraries à importer). De cet étape de pre-processing, va résulter un fichier contenant tout le code source du projet + le contenu des fichiers en-têtes.
- Une fois la pre-processing finie, la **compilation** du fichier commence, ce qui résulte avec un fichier objet `.o`.
- Ensuite la partie suivante est le **linking** qui va lier les librairies au fichier `.o` pour donner le fichier exécutable final.

Déconstruire le processus de compilation en ligne de commande

Si vous voulez essayer (de le faire manuellement) par vous même, vous pouvez faire les commandes suivantes :

- Fichier .c : créer simplement un hello world comme montré plus haut dans un fichier `hello.c`
- Précompilation : `gcc -E hello.c`
- Compilation : `gcc -c hello.c`
- Linking : `gcc hello.o -o hello`

Compilation manuelle et console

Si vous voulez compiler le code par ligne de commande vous n'avez pas besoin de taper plein de commandes. Il suffit juste de faire `gcc \*.c` cependant il faut faire attention à plusieurs choses :

- Toujours inclure tous les fichiers .c dans la commande gcc
- Faire attention à la version de gcc
- Ne pas oublier d’ajouter les flags de compilations (qui donne des instructions suplémentaires au compilateur) à la commande gcc pour les projets de l’école

Notions fondamentales

```
/* Tout d'abord on doit inclure les librairies stdio et stdlib dans tous les projets. On a déjà parlé de ce que fait le #include dans le bloc de code Hello World */
#include<stdio.h>
#include<stdlib.h>

/* On donne les signatures des méthodes présentes dans le fichier dès le début car le compilateur C va lire le fichier de haut en bas et doit pouvoir directement savoir quelles fonctions existent dans le fichier */
float ajoute(float, float);
float soustrait(float, float);
float multiplie(float, float);
float divise(float, float);

/* La fonction main est le programme principale, ce qui va être exécuté lorsque l'on lance l'exécutable compilé du code */
int main(void) {
    /* Dès le début de la fonction on est obligé de déclarer nos variables */
    float n1, n2, resultat;
    char operation;

    /* Printf vient de stdio et permet d'afficher du code dans la console. Le caractère \n sert à retourner à la ligne */
    printf("Calculatrice simple\n");
    printf("Entrez l'opération à réaliser :");

    /* Scanf permet de récupérer un input d'un utilisateur dans la console. %f définissant un nombre flotant, %c un caractère et %*c servant à éliminer le denier caractère (le \n, soit le retour à la ligne) */
    /* Ces 3 valeurs (2 nombre flotatnts et un caractère) seront donc stoqué dans 3 variables (on passe donc les ADDRESSES de n1, opération et n2 en préfixant les variables d'un &) */
    scanf("%f %c %f%*c", &n1, &operation, &n2);

    /* Le  switch en C ne fonctionne qu'avec des valeurs entières. Par exemple ici '+' correspond à la valeur entière 43 dans la table ASCII. */
```

```
    switch(operation) {
        case '+':
            resultat = ajoute(n1, n2);
            break;
        case '-':
            resultat = soustrait(n1, n2);
            break;
        case '*':
            resultat = multiplie(n1, n2);
            break;
        case '/':
            resultat = divise(n1, n2);
            break;
    }

    /* Le printf ici fonctionne avec le même type de syntaxe que le scanf vu plus tot */
    printf("==> %f %c %f = %f\n", n1, operation, n2, resultat);

    /*  Enfin on retourne l'exit code du programme, ici un succès */
    return EXIT_SUCCESS;
}

/* Les méthodes annoncées dans l'en-tête plus haut sont définie ici */
float ajoute(float nombre1, float nombre2) {
    return nombre1 + nombre2;
}

float soustrait(float nombre1, float nombre2) {
    return nombre1 - nombre2;
}

float multiplie(float nombre1, float nombre2) {
    return nombre1 * nombre2;
}

float divise(float nombre1, float nombre2) {
    return nombre1 / nombre2;
}
```

Types en C

Les types de C sont très basiques contrairement à ceux d'autres langages (de plus haut-niveau) tel que Java.

Type	Explication	Codé sur	Représentation dans printf/scanf	Valeurs admissibles
`char`	Destiné à contenir un seul caractère. Il y a une conversion automatique char en type entier, ainsi 'c' en char deviendra 99 (sa valeur ASCII) en entier	8 bits	`%c`	Tous les caractères codés sur 8 bits
`short`	Destiné à contenir des valeurs entières petites	16 bits	`%hi`	De -2^{15} à $+2^{15} - 1$
`int`	Destiné à contenir des valeurs entières	32 bits	`%i` ou `%d`	De -2^{31} à $+2^{31} - 1$
`unsigned int`	Destiné à contenir des valeurs entières non signées (strictement positives)	32 bits (mais 16 bits minimum)	`%u`	De 0 à $+2^{32} - 1$
`long int`	Destiné à contenir de grandes valeurs entières (cependant sous Unix, il est la même que `int`)	32 bits minimum	`%li`	De -2^{31} à $+2^{31} - 1$
`long long int`	Destiné à contenir des plus grandes valeurs entières	64 bits	`%lli`	De -2^{63} à $+2^{63} - 1$
`float`	Destiné à contenir des valeurs avec fraction décimale (précision simple)	32 bits	`%f`	
`double`	Destiné à contenir des valeurs avec fraction décimale (plus précis)	64 bits	`%lf`	

C **ne dispose pas de type booléen**, cependant la valeur entière `0` est toujours considérée comme FAUX et tout autre valeur est considérée comme VRAI.

Plus de représentation printf et scanf

Caractères spéciaux

Symbole	Signification
`\n`	Caractère de controle `LF` qui fait un retour à la ligne sous Linux
`\r`	Caractère de controle `CR`. `\r\n` provoque un retour à la ligne sous Windows
`\t`	Tabluatino vers la droite
`\\`	Caractère `\`
`%%`	Caractère `%`

Autres types non élémentaires

Symbole	Signification
`%s`	Chaine de caractère
`x`	Donnée `unsigned int` au format hexadécimal

Précision de l'affichage

Symbole	Signification	Valeur	Affichage
`%3d`	Donnée formattée sur 3 chiffres, les absences de chiffres sont remplacées par des espaces	`9`	`  9`
`%03d`	Même chose mais les espaces sont remplacés par des 0	`9`	`009`
`%.2f`	Permet de préciser le nombre de chiffres derrnière la virgule d'un valeur fractionnelle	`9.191`	`9.19`

Fonctions et protoypes

Les signatures des fonctions comme mises au début du fichier de la calculatrice sont appellé des prototypes ou des signatures de fonction. Elles annoncent les fonctions qui vont être présentes. Sauf qu'en vérité, ces signatures sont dans des fichiers séparés appellée *en-têtes* dans des fichiers `.h`. Ces fichiers sont ensuite inclus dans le programme en utilisant `#include "file.h"`.

Lorsque l'on inclu un code on inclu toujours le fichier en-tête et jamais le fichier .c. A noter que si on veut importer un fichier en-tête bien précis in peut specifier le chemin d'accès entre guillemets (exemple `#include "file.h"`) mais lorsque l'on veut ajouter une librarie standard, on va la mettre

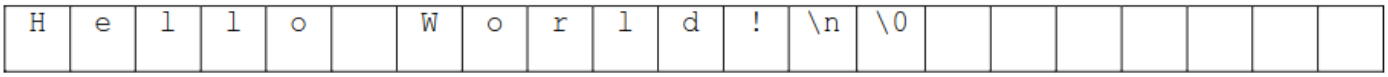
entre chevrons (exemple `#include <stdio.h>`)

Lorsque l’on a plusieurs fichiers dans un projet C, il est important de bien garder la règle d’un seul dossier par projet, sinon ça risque fort de foutre la merde. Lorsuqe

Chaines de caractères (et tableaux)

En C il n'y a pas de type String, les chaines de caractères sont simplement des tableaux de caractères. Sauf que puis ce que l'on ne sait pas combien de la longueur du tableau a été replis, donc on met un caractère de fin de chaine à la fin du tableau `\0`.

```
char ma_chaine[21] = "Hello World!\n";
```



Il est très important de toujours vérifier l'input des utilisateur·ice·s car si la personne entre quelque chose de plus long que la taille du tableau cela peut être une faille de vulnérabilité (car cela peut mener à un BufferOverflow). C'est notament arrivé au programme `sudo` sous linux. Si vous voulez en apprendre plus vous pouvez regarder cette vidéo.

Lecture des chaines de caractères

Il existe par exemple `gets()`, `scanf()` ou encore `fgets()` pour prendre un input de l'utilisateur·ice.

Cependant il ne faut **pas** utiliser `gets()` car il ne vérifie pas la taille des données (ce qui peut donc mener à un BufferOverflow). Il faut donc toujours utiliser `scanf` ou `getf`.

scanf

Voici par exemple comment récupérer les max 20 premiers caractères d'un input (le reste sera ignoré).

```
scanf("%20[^\n]%*c", ma_chaine);
```

Pour déconstruire un peu ce qu'il se passe ici :

- `%20` signifique que l'on prends les 20 premiers caractères
- `[^\n]` signifie que l'on arrete de prendre des caractères quand l'utilisateur·ice fait ENTER
- `%*c` signifie que l'on ignore le dernier caractère (le retour à la ligne `\n`)
  Autre chose intéressante à noter ici, il n'y a pas de `&` devant le nom de la fonction contrairement à avant quand on récupérait des caractère ou nombres uniques. Cela est dû au fait que `&` sert à passer l'addresse d'une variable (le pointeur) et qu'un tableau

(comme une chaine de caractère) *est* déjà une addresse (pointeur).

fgets

`fgets` fonctionne assez différemment de `scanf`, voici comment on peut faire quelque chose de similaire à l'exemple précédent en utilisant fgets :

```
fgets(ma_chaine, 20, stdin);
```

Attention cependant que fgets compte `\n` comme un caractère et l'inclus dans le résultat. Donc bien que la syntaxe de fgets soit plus simple, il faut mieux utiliser `scanf` car elle s'occupe du caractère `\n` toute seule.

Affichage des chaines de caractères

puts

L'exemple suivant va afficher la chaine de caractère en y ajoutant un retour à la ligne automatiquement à la fin (c'est tout l'interet du puts), c'est un peu comme le `System.out.println` en Java :

```
puts(ma_chaine);
```

fputs

Cet exemple fonctionne de manière similaire du `puts` sauf qu'il n'ajoute pas de retour à la ligne. C'est un peu comme le `System.out.print` en Java.

```
fputs(ma_chaine, stdout);
```

printf

Printf est surtout intéressant pour formatter l'affichage (c'est l'équivalent du `System.out.printf` en Java).

```
printf("%s\n", ma_chaine);
```

Autres fonctions

Il existe une librarie `string` en C permettant d'intéragir plus facilement avec les chaines de caractères. Attention cependant, il ne faut pas la confondre avec le type String en Java, car en C "string" n'est pas un type les chaines de caractères sont simplement des tableaux de `char`

Pour importer la librarie string, il suffit d'ajouter la ligne suivante au dessus du fichier :

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

Maintenant voici une petite listes des fonctions les plus utiles de string :

Fonction	Explication
`strlen(ma_chaine)`	Compte le nombre de caractères de la chaine jusqu'au `\0`
`strncmp(chaine1, chaine2, n)`	Compare les n premiers caractères des chaines. Si les deux sont les même cela signifie que les deux sont identiques
`strncpy(dest, source, n)`	Copie les n premiers caractères de la source vers la destination (le `\0` n'est pas ajouté)
`sprintf(dest, "%d + %d", 4, 5)`	Fait comme `printf` sauf qu'à la place de l'afficher, il le stoque dans une variable. C'est comme le `String.format` en Java
`sscanf(src, "%d + %d", &a, &b)`	Fait comme le `scanf` sauf qu'a la place de le demander depuis le stdin (standard input), il va le prendre depuis une chaine de caractère source
`memset(src, n, 0)`	Initialise les n premiers caractères de la chaine src avec le caractère mentioné (ici on remplace tout par `\0`)
`strchr(chaine, car)`	Recherche la première occurence d'un caractère dans une chaine et retourn eun pointeur vers celle-ci
`strstr(chaine, sous-chaine)`	Recherche la première occurence d'une sous-chaine donnée dans une chaine et retourne un pointeur vers celle-ci.

Exemple de manipulation de tableau/chaines

```
/* Stoquer une chaine dans un tableau */
char ma_chaine[20+1] = "Hello, World!";

/* Accéder au 5e caractère de la chaine */
printf("Le 5e caractère est %c\n", ma_chaine[4]);

/* Modifier un caractère */
ma_chaine[4] = ' ';

/* On peut aussi mettre le \0 n'importe où pour couper une chaine */
ma_chaine[4] = '\0';
printf("La chaine est maintenant : %s\n", ma_chaine);
```

Génération d'aléatoire

La généréation d'aléatoire se fait via la fonction `rand`, cependant il est important de se rappeller que l'aléatoire en informatique n'existe pas, on parle ici de **pseudo**-aléatoire.

Le fonctionnement de la fonction c'est que rand va prendre un nombre de départ (appelée `seed` qui par défault est 1) et va y faire des opérations pour que le nombre ai l'air aléatoire.

Ensuite le nombre produit va être utilisée comme seed pour générer d'autres nombres aléatoires par la suite.

Ce qui veut dire que par défault, ce programme donnera toujours les même valeurs :

```
printf("%d\n", rand()); /* 1804289383 */
printf("%d\n", rand()); /* 846930886 */
printf("%d\n", rand()); /* 1681692777 */
```

Pour avoir quelque chose qui se rapproche un peu plus de l'aléatoire, on peut fixer le seed pour être autre chose au début du programme, typiquement, le temps (UNIX time, qui est le nombre de secondes depuis le 1/1/1970 00:00 UTC).

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

int main(void) {
  srand(time(NULL));

  printf("%d\n", rand());
  printf("%d\n", rand());
  printf("%d\n", rand());

  return EXIT_SUCCESS;
}
```

Dans ce nouvel exemple on a fixé le temps comme étant le seed du rand, ainsi deux programmes ne s'exécutant pas dans la même seconde auront des résultats différents qui auront l'air aléatoire.

Les structures

Les structures en C permettent de créer des types personalisés, un peu comme les classes en Java mais sans méthodes (askip c'est possible de faire des méthodes mais c'est très peu commun et donc pas expliqué dans ce cours).

Il existe deux manière de faire une structure en C mais il vaut mieux toujours rester consistent sur la manière utilisée.

Voici une première manière de faire une structure en C :

```
struct etudiant {
    int matricule;
    char nom[50+1];
    char prenom[50+1];
    char adresse[200+1];
    char telephone[15+1];
    float moyenne;
};
```

Ensuite pour déclarer une variable :

```
struct etudiant un_etudiant;
```

Et voici la seconde manière de faire :

```
typedef struct {
    int matricule;
    char nom[50+1];
    char prenom[50+1];
    char adresse[200+1];
    char telephone[15+1];
    float moyenne;
} Etudiant;
```

Et ensuite pour déclarer une variable :

```
Etudiant un_etudiant;
```

Lecture et écriture des données

Etant donné qu'il n'y a pas de méthodes aux structures, il n'y a pas de modificateur `private` sur les attributs, les attributs peuvent donc être accédé et modifié sans limite.

```
/* Lecture d'un struct */
printf("%s %s\n", un_etudiant.nom, un_etudiant.prenom);


/* Ecriture d'un struct */
un_etudiant.matricule = 123456;
```

Partage des structures entre programmes

Pour partager les structures entre plusieurs programmes on peut simplement mettre la déclaration du struct dans un fichier en-tête (`.h`).

Les tableaux

Il est possible en C de déclarer un tableau contenant des données de types identiques qui sont ensuite rangées en mémoire dans des cases contigues.

Un tableau en C est une addresse mémoire (appelée pointeur) donc quand on demande le premier élément, on prends la première case au niveau du pointeur. Pour prendre le deuxième, on va une case plus loin et ainsi de suite.

Ce qui signifie que l'on peut aller plus loin que la taille réservée du tableau, lorsque l'on fait ça on dit que l'on "jardine en mémoire" ce qui est risqué car cela peut faire planter le programme et que les données en dehors du tableau peuvent être réécrites par d'autres variables dans le programme.

Définition d'un tableau

```
/* Définition d'un tableau de 10 entiers simple */
int suite[10];

/* Initialisation d'un tableau de 5 entiers */
int suite[10] = { 1,2,3,4,5 };

/* Lecture du troisième élément d'un tableau */
printf("%d\n", suite[2]);

/* Ecriture d'un élément du tableau */
suite[2] = 42;
```

Attention à l'initialisation des variables et tableaux

Cela veut aussi dire que si un tableau n'est pas initialisé, si on récupère une position qui n'a pas encore été définie on peut tomber sur d'anciennes valeurs encore dans la mémoire.

C'est pour cela qu'il faut généralement garder une variable supplémentaire pour garder le compte du nombre de cases écrites du tableau (cela n'est pas nécessaire pour les chaines de caractère car on sait que la chaine se finit au caractère `\0`). La longueur des données réelement dans un tableau est appellée "taille effective" tandis que la talile en mémoire du tableau est appellée taille physique.

Pour ce qui est des variables c'est pareil, par défault les variables ne sont pas initialisées (bien que cela peut varier des OS et des compilateurs). C'est pourquoi il vaut toujours mieux intialiser les

variables. Car les variables sont simplement des addresses mémoires, donc si on lit une variable non initialisée on va lire les données qui sont à cet endroit dans la mémoire (il peut donc y avoir un peu n'importe quoi).

Voici un exemple de code permettant de tester cela :

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

int main(void) {
  /* On crée une variable non initialisée */
  int ma_variable;

  /* On demande à sscanf d'initialiser la variable à partir de rien : spoiler il va pas l'initialiser */
  sscanf("", "%d", &ma_variable);

  /* On lit la variable non initialisée */
  printf("Ma variable non-initialisée vaut : %d\n", ma_variable);

  return EXIT_SUCCESS;
}
```

Tableaux et fonctions

On peut passer des tableaux en arguments de fonctions cependant étant donné que le tableau est un pointeur il ne sera pas copié car c'est bien sa référence qui sera passée à la fonction.

Une conséquence de ça c'est qu'une fonction en C ne peut pas retourner un tableau, pour traiter des tableaux il vaut mieux passer le tableau en argument, le modifier dans la fonction et retourner la taille effective du tableau sous forme de int.

Voici un exemple simple d'utilisation de tableaux dans des fonctions :

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

int add_42(int tableau[], int taille_tableau);

int main(void) {
  /* On crée un tableau avec une taille de 10 contenant 3 éléments */
  int tableau[10] = {1,2,3};

  /* On note la taille effective du tableau comme étant 3 (pour les 3 éléments) */
```

```
  int taille_tableau = 3;

  /* On passe le tableau et la taille dans la fonction qui va modifier le tableau et retourner la nouvelle taille */
  taille_tableau = add_42(tableau, taille_tableau);

  /* On affiche le nouvel élément de notre tableau */
  printf("Le nouvel élément du tableau est %d\n", tableau[taille_tableau - 1]);

  /* On ferme le programme */
  return EXIT_SUCCESS;
}

/* On défini une fonction retournant un entier (nouvelle taille), un tableau de taille indéfinie, et la taille effective du tableau */
int add_42(int tableau[], int taille_tableau) {
  tableau[taille_tableau] = 42;
  return taille_tableau + 1;
}
```