Langage C avancé : Séance 2

Vecteurs

Chaines de caractères

Entrées/sorties sur clavier/écran

Marc TAJCHMAN
@-mail : marc.tajchman@cea.fr

CEA - DES/ISAS/DM2S/STMF/LDEI

Remarques générales que j’ai oublié de mentionner dans la première séance:

Dans le code source C:

- toutes les instructions et les définitions de variables se terminent par un séparateur « ; »
- La partie de texte qui commence par /* et qui se termine par */ est ignorée par le compilateur (c’est un « commentaire »)

- Certaines notions sont utilisées et présentées dans cette séance (mémoire dynamique, représentation en mémoire, …), ce sera développé plus tard.

Code source minimal

Code C minimal

- Ouvrir un éditeur de texte
- Entrer les 4 lignes ci-dessous dans l'éditeur de texte et sauver dans un fichier ex1.c

```
int main()
{
   return 0;
}
```

- Enregistrer dans un fichier ex1.c (retenir dans quel répertoire se trouve ex1.c)

Commentaires sur le code source minimal

- Le code source minimal contient une fonction « main » qui sera utilisée comme point de départ de l'exécution du code binaire.
- Cette fonction n'utilise pas de données, ne contient aucune instruction et génère un seul résultat entier (égal à 0)
- Cet entier est en général utilisé comme indication si l'exécution s'est bien passée (0 si exécution réussie, autre valeur en cas de problème)

Chaque fois que vous commencerez un nouveau code C, il faudra taper ce code source minimal et le compléter

Compilation et exécution du code source minimal

- Ouvrir un terminal de commandes
- Se placer dans le répertoire qui contient le fichier « ex1.c » (dans lequel se trouve le code source minimal).
- Compiler en tapant la commande :

```
gcc ex1.c –o ex1
```

  qui génère le fichier binaire « ex1 »

- Exécuter le fichier binaire « ex1 » en tapant la commande :

```
./ex1
```

(rien n’est affiché à l’écran, c’est normal, le code minimal ne contient aucune instruction)

Les vecteurs

Qu’est ce qu’un vecteur ?

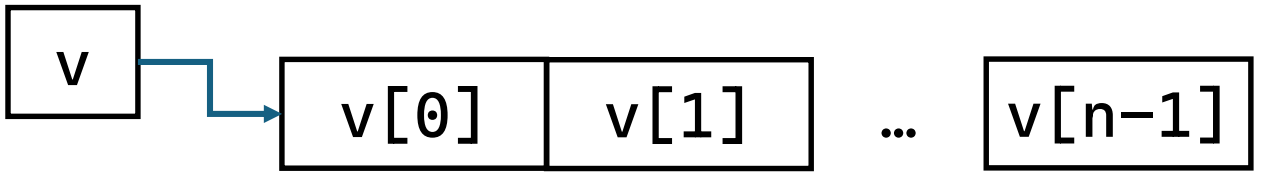
- Un vecteur est un ensemble de n éléments de même type (int, double ou autre type C)

- Pour utiliser un vecteur de type T (T = int, double ou autre type C), on passe par un pointeur :

```
T * v;
```

(explicitement ou implicitement)

- En mémoire:



Composantes d’un vecteur

Les composantes d’un vecteur v de taille n sont accessibles sous la forme:

```
v[i] = …      /* écriture (modification) */
x = … v[j] … /* lecture  (utilisation) */
```

où i et j sont des entiers entre 0 et n-1

Toute utilisation de v[i] où

- i est négatif ou
- i est égal ou supérieur à n

est une erreur (on verra dans une autre séance des outils pour détecter ce type d’erreur)

Plusieurs façons de définir un vecteur (1)

Vecteur de taille définie dans le code source

```
double v[3];
```

La taille de v est constante (ici 3), mais chaque composante peut être modifiée:

```
v[2] = v[1] + 1.4;
```

Attention, à la déclaration, les composantes d'un vecteur n'ont pas de valeur prédéfinie (en particulier, pas 0)

Il y a un pointeur (fixe) invisible pour l'utilisateur vers le début de la zone mémoire occupée par le vecteur.

Plusieurs façons de définir un vecteur (2)

Vecteur de taille fixe et initialisé à la définition (la taille du vecteur est celle du nombre de valeurs initiales)

```
double v[] = {1.2, 3.4};
```

La taille de v est constante (ici 2), mais chaque composante peut être modifiée:

```
v[0] = v[1] + 1.4;
```

Plusieurs façons de définir un vecteur (3)

Vecteur de taille fixe mais dont la taille n'est pas connue dans le code source (seulement à l'exécution)

```
int n = …
…
double v[n];
```

Cette façon de définir un vecteur (vla : variable length array) **est déconseillée** : cela ne marche pas avec tous les compilateurs et/ou pour des tailles de vecteur (n) assez grandes.
(voir la séance où on parlera des détails de la gestion mémoire)

Utiliser plutôt l'allocation dynamique de la mémoire (page suivante).

Plusieurs façons de définir un vecteur (4)

Vecteur dont la taille peut être spécifiée à l'exécution et dont la mémoire est gérée dynamiquement.

Pour utiliser un vecteur de taille n et de type entier (par exemple):

- On déclare d'abord un pointeur sur entier:

```
int * v;
```

- On réserve (alloue) la mémoire suffisante pour n entiers (v désigne le début de la mémoire réservé par le système):

```
v = (int *) malloc(n * sizeof(int));
```

- Le vecteur peut être utilisé (n'oubliez pas de l'initialiser)

```
v[0] = 1;
V[1] = v[0] + 3;
```

Plusieurs façons de définir un vecteur (4)

- Quand le vecteur n'est plus utilisé, on signale au système que sa mémoire est disponible (on libère la mémoire)

```
free(v);
```

- Le pointeur est à nouveau disponible pour un autre vecteur

```
v = (int *) malloc(m * sizeof(int));
```

- …

Il n'y a pas de restriction sur la taille du vecteur (sauf la taille totale disponible dans le système).
Si malloc ne parvient pas à réserver la mémoire, il a comme résultat un pointeur null (NULL), pour plus de sécurité, ajouter un test sur le pointeur:

```
if (v == NULL)
   exit(-1); /* ou autre traitement d'erreur */
```

Exercice 1 : Echange de vecteurs

Echange du contenu de 2 vecteurs

- Définir et initialiser 2 vecteurs d'entiers de taille n=100 :

  $v = (v_i = i,\ i=0,\ldots,n)$ et $w = (w_i = i*i,\ i=0,\ldots,n)$

- Echanger le contenu de v et de w
  - en utilisant des vecteurs définis comme à la page 10
  - en utilisant des vecteurs définis avec de la mémoire dynamique (pages 13 et 14), proposer 2 façons de faire

Rappel: pour échanger deux valeurs de type entier (ou tout autre type C), on utilise généralement une variable supplémentaire du même type (temp dans l'exemple):

```
temp = a;
a = b;
b = temp;
```

Pour afficher le contenu d’un vecteur d’entiers

Si v est un vecteur d’entiers de taille n, les lignes de code source suivantes affichent à l’écran le vecteur:

```
printf("v = \n");
for (i=0; i<n; i++)
  printf("v[%3d]: %5d\n", i, v[i]);
printf("\n");
```

Il faut ajouter la ligne #include <stdio.h> au début du fichier et ne pas oublier de déclarer l’entier i.

Les chaines de caractères

Les chaines de caractères

- Ce sont des vecteurs de caractères
- Comme pour les nombres (entiers, décimaux), on peut définir des variables/constantes de type caractère ou chaines de caractères
- On utilise des pointeurs sur caractère pour les utiliser

Exemples:

```
int a = 1;
double x = 1.2e3;

char c1 = 'z';
char c2 = '1';

char * ch = "azerty";
```

a, x, c1, c2, ch sont des variables

1, 1.2e3 (=1200.0), 'z', '1' et "azerty" sont des constantes

(remarquer les guillemets simples pour les constantes caractères et les guillemets doubles pour les chaines)

Taille (ou longueur) des chaines de caractères

Les chaines de caractères sont des vecteurs de caractères, pour les utiliser, il faut donc connaitre leur taille.

1ère possibilité : définir aussi un entier qui contient la longueur de la chaines

Pas pratique : on utilise souvent beaucoup de chaines de caractères, et il est facile et dangereux de se tromper en associant à une chaine, un entier qui contient la longueur d'une autre chaine : **non retenu dans le langage C**

2ème possibilité : ajouter, en fin de chaine, un caractère spécial '\0' qui permet de savoir quand la chaine se termine

Taille (ou longueur) des chaines de caractères (2)

Pour connaitre la longueur d'une chaine, C propose une fonction strlen qui fournit cette information en cherchant la position de \0

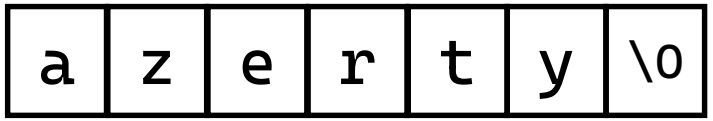
```
char * s = "azertyuiop";
int n = strlen(s);

/* n contient la valeur 10 */
```

Le résultat de strlen est un entier positif ou nul (cas d'une « chaine vide » char *s = "")

Représentation mémoire d'une chaine de caractères

Une chaine de n caractères est donc représentée en mémoire par une zone mémoire de n+1 emplacements

Par exemple:
"azerty" (chaine de 6 caractères) est représentée en mémoire par une zone mémoire de taille 7 x (taille de 1 caractère)



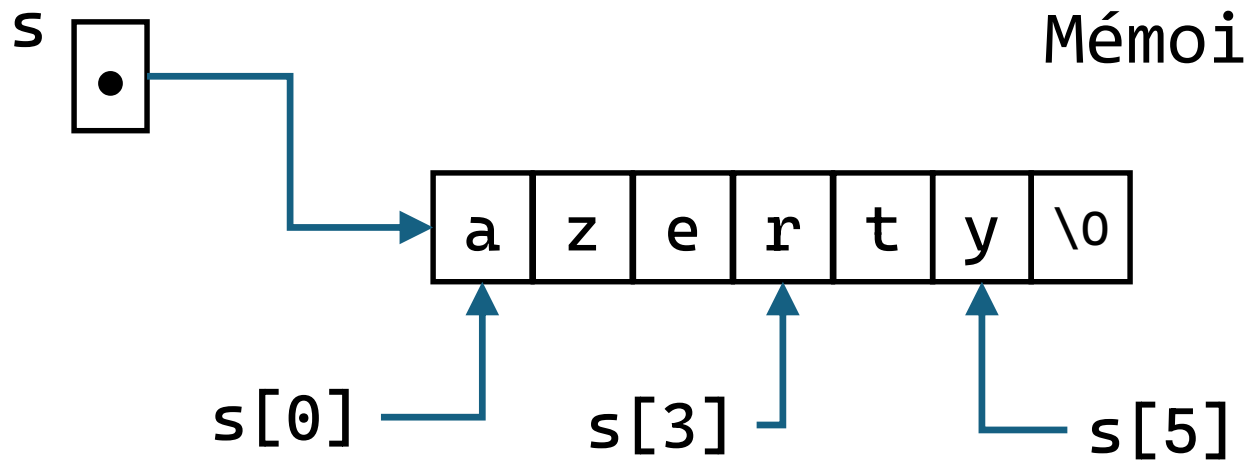
Le caractère en 7ème position à un code spécial (0), il n'est pas possible de l'afficher à l'écran ou de le rentrer au clavier
On appelle parfois ce type de valeur, une sentinelle.

Accès aux caractères dans une chaine de caractères

Exemple:

```
char * s = "azerty";
```

Code Source



Mémoire

s[i] (i=0, …, strlen(s) −1) désigne le $i^{ème}$ caractère de la chaine (si i prend une valeur en dehors de l'intervalle [0, strlen(s) −1], c'est une erreur)

Plusieurs façons de définir une chaine de caractères (1)

Chaine de caractères **constante** (la valeur est fixée à la définition et ne change pas dans le code)

```
char * s = "azertyuiop";
```

Une tentative de modifier le contenu de s est acceptée par le compilateur, mais l'exécution s'arrêtera sur une erreur:

```
s[3] = 'X';
```

Donc, on conseille (fortement) de définir ce type de chaine comme suit:

```
const char * s = "azertyuiop";
```

Le compilateur refusera toute modification de s.

Plusieurs façons de définir une chaine de caractères (2)

Seconde façon de définir une chaine avec une valeur connue

```
char s[] = "azerty";
```

Dans ce cas, le système

- calcule la longueur (6) de la valeur de type chaine (azerty)
- réserve un espace mémoire de longueur 7 caractères
- recopie la valeur dans la nouvelle zone mémoire

Dans ce cas, une instruction telle que:

```
char * s[3] = 'X';
```

ne pose pas de problème (s n’est pas constante).

Plusieurs façons de définir une chaine de caractères (3)

Troisième façon de définir une chaine avec une **valeur non connue mais dont la taille maximale est connue**:

```
char * s;
int n;
n = ... /* par ex. valeur entree au clavier */

/* réservation de la mémoire de s */
s = (char *) malloc((n+1) * sizeof(char));

/* utilisation de s */

/* libération de la mémoire de s */
free(s)
```

Plus souple, et permet de définir de très grandes chaine de caractères, mais il faut gérer soi-même la mémoire utilisée par s (à l'aide des fonctions malloc et free)

Plusieurs façons de définir une chaine de caractères (4)

Variante si une chaine est initialisée avec une **valeur initiale connue** et la chaine est **modifiable dans la suite** :

```c
#include <string.h>
#include <stdlib.h>
#define CH "azerty"

int main()
{
    char *s = (char *)
         malloc((strlen(CH)+1)*sizeof(char));
    strcpy(s, CH); /* copie dans s */

    free(s);
    return 0;
}
```

La fonction malloc reserve une zone mémoire suffisante pour s (7 caractères)
La fonction strcpy est utilisée pour initialiser la chaine s.

Plusieurs façons de définir une chaine de caractères (5)

L'exemple précédent peut-être simplifié en:

```c
#include <string.h>
#include <stdlib.h>
#define CH "azerty"

int main()
{
    char *s = strdup(CH);

    free(s);
    return 0;
}
```

Plus simple, mais la fonction strdup est une fonction qui n'est pas disponible avec tous les compilateurs et il faut tout de même penser à libérer la mémoire avec free

Exercice 2: Affichage à l’écran

Le but est d’afficher à l’écran un texte suivi d’un entier

Il faudra :

1. Ajouter au code source minimal une variable de type chaine de caractères et lui donner une valeur
2. Ajouter au code source minimal une variable de type réel simple précision (float) et lui donner une valeur
3. Appeler la fonction C standard qui affiche à l’écran ces informations

Chaine de caractères

- Pour utiliser un texte dans le code source, on utilise une chaine de caractères, en tapant ces 2 lignes dans le code source :
  - une ligne pour définir le pointeur « message » vers un (des) caractère(s) et
  - une ligne pour connecter le pointeur à la chaine fixe de caractères "bonjour")

```
char * message;
message = "bonjour";
```

- On peut aussi écrire la même chose en une seule ligne :

```
char * message = "bonjour";
```

Réel de type float

- Pour utiliser un float dans le code source, on utilise une variable (de type float), en tapant ces 2 lignes dans le code source :
  - une ligne pour définir une variable de type float et
  - une ligne pour ranger une valeur dans cette variable

```
float x;
x = 34.4;
```

- On peut aussi écrire la même chose en une seule ligne :

```
float x = 34.4;
```

Affichage de valeurs à l'écran

- Pour afficher des informations à l'écran, il faut
  - Inclure le fichier système stdio.h qui définit les fonctions nécessaires
  - Appeler la fonction printf (voir plus de précisions à la page suivante)
- Le code source devient:

```c
#include <stdio.h>
int main()
{
    char * message = "bonjour x = ";
    float x = 34.4;
    printf("%s %f", message, x);
    return 0;
}
```

La fonction printf

printf est une fonction système de C pour afficher des informations à l'écran, elle a un nombre variable de paramètres (au moins 1) :

- Le premier paramètre (appelé format) est une chaine de caractères qui indique le format d'affichage (nombre et type de valeurs à afficher)
- Dans le format, « % » suivi d'un entier éventuel et d'un caractère indique le type de valeur à afficher et la façon de traduire la valeur en texte (largeur, nombre de décimale, etc…)
- Le(s) paramètre(s) suivant(s) sont les valeurs à afficher, chaque paramètre doit correspondre à un des « % », en respectant l'ordre et les types

La fonction printf (2)

Exemple:

```
printf("%s n = %3d", message, N);
```

Valeur/variable de type entier, compatible avec %s

Valeur/variable de type « chaine de caractères », compatible avec %3d

Paramètre de type chaine appelé « format »
%s : affichage d'une chaine de caractères
%3d : affichage d'un entier sur 3 positions cadré à droite si l'entier est entre -99 et 999, sur >3 positions sinon
Autres caractères du format affichés tels quels

Référence sur printf

La spécification complète de printf et de son paramètre format est disponible dans la norme C (payante)
https://www.iso.org/fr/standard/82075.html

Mais on pourra trouver une information assez complète dans les manuels C ou dans la page
https://fr.cppreference.com/w/c/io/fprintf

Exercice

1. Créer un fichier avec le code source de la page 23
2. Le compiler et l’exécuter
3. Modifier la valeur de x à 34.5 dans le code source
4. Recompiler et exécuter à nouveau ?
5. Essayer d’expliquer ce qui est affiché dans les 2 cas.

Exercice 3: Lecture d’informations depuis le clavier

Le but est d’utiliser des valeurs entrées par l’utilisateur au clavier

Il faudra :

1. Définir des variables qui vont contenir les informations (pas nécessaire de les initialiser)
2. Appeler la fonction C standard `scanf` qui attend que l’utilisateur entre des informations et les range dans les variables ci-dessus

Variables

- Le code source doit définir des variables de types correspondant aux types des informations introduites par l'utilisateur au clavier et de **tailles suffisantes**

- Par exemple

```
int N;
char *prenom = (char *) malloc(10*sizeof(char));
```

Si l'utilisateur peut rentrer au clavier, un entier et une chaine de caractères de taille maximale 9 (penser au caractère « sentinelle »)

Lecture des valeurs depuis le clavier

- Pour lire des informations depuis le clavier, il faut
  - Inclure le fichier système stdio.h qui définit les fonctions nécessaires
  - Appeler la fonction scanf (voir plus de précisions à la page suivante)

Pour lire une variable, il faut mettre en paramètre un pointeur sur cette variable, sauf si la variable est déjà un pointeur (par exemple, chaine de caractères), voir exemple page suivante

Exemple de lecture avec la fonction scanf

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

int main()
{
  char *prenom = (char *) malloc(10*sizeof(char));
  float x;

  printf("Entrez un nombre décimal suivi de votre prénom > ");
  scanf("%f%9s", &x, prenom);

  printf("%s, vous avez entré %f\n", prenom, x);
  free(prenom);
  return 0;
}
```

Référence sur scanf

La spécification complète de scanf est disponible dans la norme C (payante)
https://www.iso.org/fr/standard/82075.html

Mais on pourra trouver une information assez complète dans les manuels C ou dans la page
https://fr.cppreference.com/w/c/io/fscanf

Exercice

1. Créer un fichier avec le code source de la page 32
2. Le compiler
3. Exécuter le code binaire, rentrer un nombre suivi d’un mot (un prénom), faire plusieurs essais avec

   12.3Jean

   12.3 Jean

   12.34 Jean