Cours “Algorithmique et Structure des Données”

Licence L2 Informatique, semestre S3

Année 2019-2022

Les types de variables, en C

Types de base :

mot-clé	type	bornes
int	Entier	-32768 à 32767, si processeur 16 bits
		-2147483648 à 2147483647, si 32 bits
float	flottant (Réel)	$3,4 \times 10^{-38}$ à $3,4 \times 10^{38}$
double	flottant double	$1,7 \times 10^{-308}$ à $1,7 \times 10^{308}$
long double	double long	$3,4 \times 10^{-4932}$ à $3,4 \times 10^{4932}$
char	Caractère (1 seul)	-128 à 127 (codage ASCII)

Il existe aussi des long int, short int, unsigned int, unsigned char, etc ...
voir par exemple : https ://fr.wikibooks.org/wiki/Programmation_C/Types_de_base

Codage des nombres

WARNING :

sur un ordinateur, quels que soient la précision du processeur, le langage utilisé, etc. . . il existe des limitations sur les nombres que la machine peut coder :

- il existe un plus grand entier positif et un plus petit entier négatif (en général son opposé, à 1 près),
- idem pour les nombres réels,
- il existe un plus petit réel strictement positif, i.e. il y a un intervalle entre 0 et ce nombre $\Rightarrow$ risques d'erreurs d'arrondi.

Expressions arithmétiques

Opérations :

symbole	opération
+	addition
−	soustraction
*	multiplication
/	division (réelle ou euclidienne)
%	reste de la division euclidienne

Exemple :

`9 / 4` vaut 2 `9 % 4` vaut 1 `9. / 4.` vaut 2.25

Expressions arithmétiques

Fonctions mathématiques : définies dans la "library" math.h

mot-clé	fonction	mot-clé	fonction
sqrt	racine carrée	sin	sinus
pow	puissance	cos	cosinus
	pow(x,2) $\leftarrow$ carré	atan	arc tangente
log	logarithme népérien	floor	troncature
exp	exponentielle		(réel $\rightarrow$ entier)
abs	valeur absolue	ceil	entier supérieur
fabs	valeur absolue (réel)		(réel $\rightarrow$ entier)

Expressions logiques

Opérateurs relationnels :

symbole	opérations
==	égal à
!=	différent de
<	inférieur à (strictement)
>	supérieur à (strictement)
<=	inférieur ou égal à
>=	supérieur ou égal à

Expressions logiques

Opérateurs logiques :

symbole	opérations
&&	et (conjonction)
||	ou (disjonction)
!	non (négation)

Exemple :

```
(x<3) && ((y!=(x-1)) || (y>2))
```

est vrai quand x vaut 1 et y vaut 7 ; c'est faux quand x vaut 2 et y vaut 1

Attention :

Ne pas confondre les opérateurs logiques et les opérateurs sur la représentation des nombres

`x && y` est différent de `x & y`

Fonction d'affichage

Programmer un affichage à l'écran :

printf("chaîne-format" ,liste d'arguments) ;

La chaîne-format est constituée de texte, de caractères de contrôle et de spécifications de formats pour l'affichage des variables.
La liste d'arguments, éventuellement vide, contient les noms des variables dont les valeurs doivent être affichées, en respectant le nombre, l'ordre et les types des spécifications contenues dans la chaîne-format.

Exemple :

```
printf("L'aire mesure %f centimètres carrés.\n",aire);
```

affiche du texte et se termine par un passage à la ligne ; la *valeur* de la variable **aire** (qui doit être de type float) sera écrite au milieu du texte.

Attention :
Les fonctions d'entrée/sortie ne sont pas dans le langage de base. Il faut inclure une bibliothèque :

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

Caractères de contrôle

Déplacement de la position d'écriture :

caractère	effet produit
\n	passe au début de la ligne suivante
\r	retourne au début de la ligne courante
\t	tabulation (remplie par des espaces)
\b	retourne à la position précédente

Ecriture de caractères réservés :

ce qu'il faut taper	caractère obtenu
\"	pour écrire des guillemets dans le texte
\\	pour écrire un antislash (backslash)
%%	pour écrire le symbole % dans le texte

Spécifications de formats

Pour annoncer le type de la variable :

spécification	type de la variable
%d	entier décimal (en base 10)
%x	entier hexadécimal (en base 16)
%f	flottant (réel)
%lf	flottant double
%c	caractère (1 seul)
%s	chaîne de caractères

Pour préciser la place à réserver :

On peut insérer un nombre entier entre le % et la lettre pour dire quel nombre minimum de positions (places d'écriture) il faudra réserver pour l'affichage de la variable.

Fonction de lecture

Lecture d'une donnée qui sera tapée au clavier :

scanf("spécification du format" ,&nom_de_variable) ;

Exemple :

```
scanf("%d",&n);
```

attend que l'utilisateur tape un entier au clavier et place sa valeur dans la variable n (qui doit être de type int).

N.B. La présence du & devant le nom de la variable est obligatoire car il fait référence à l'emplacement mémoire à laquelle se trouve cette variable.

Types de base

On considérera comme types de base au moins ceux qui existent en C et dans la plupart des autres langages :

Entier Réel Caractère

Il est utile de considérer comme type de base le type Booleen, avec les deux valeurs possibles VRAI et FAUX (qui seront donc des *constantes*).

Il est aussi pratique d'inclure le type Chaine dans les types de base, pour les variables qui contiendront des chaînes de caractères, même si, a priori, une chaîne n'est autre qu'un *tableau* de caractères.

Autres types

Les types structurés, tels que les tableaux (vecteurs, matrices) ou autres objets plus exotiques [cf. suite du cours], sont composés à partir de ces types de base.

Plus généralement, on peut définir toutes sortes de nouveaux types et leur donner des noms, après avoir spécifié comment ils sont construits à partir des types de base.

Les pointeurs ont un rôle un peu à part [cf. suite du cours], mais on peut aussi définir des types de pointeurs.

Qu'est-ce qu'une "constante" ?

une constante est une variable dont la valeur ne peut pas être modifiée, par aucune instruction. Ceci mis à part, une constante peut être utilisée comme une variable. Elle a :

- un nom, qui est :
  - soit prédéfini (ex. VRAI, FAUX, PI, etc...)
  - soit que l'on choisit comme celui d'une variable ; on déclare, en même temps que ce nom, la valeur de ladite constante.
- un type, qui est implicitement défini par la valeur attribuée à la constante lors de sa déclaration.

N.B. Si l'on veut que la constante coef vaille 2 mais soit de type Réel, alors on déclarera : Constante coef = 2.0

Pourquoi un tableau ?

Lorsqu'on doit utiliser plusieurs variables ayant des rôles similaires, on peut leur donner un même nom suivi d'un numéro.

Exemple : Entrer une série de notes $\Leftarrow$ on peut les nommer Note1, Note2, Note3, etc...

C'est peu pratique, surtout que l'algorithme doit pouvoir convenir à un nombre quelconque de notes !

On utilisera alors **une seule** variable, de type tableau, que l'on nommera Note et qui contiendra plusieurs valeurs successives, rangées dans des cases repérées par un indice : Note[i]=val_i

Note	val_1	val_2	...	val_n
indice :	$i = 1$	$i = 2$	...	$i = n$

Taille d'un tableau : définition

Un tableau contient nécessairement des éléments **de même type**.
Le nombre d'éléments (i.e. le nombre de cases) est la taille du tableau.

Cette taille doit être connue au moment de la déclaration d'une variable de type tableau, puisque la déclaration détermine la place mémoire à allouer
$\Rightarrow$ on doit en général surévaluer la taille des tableaux pour qu'ils puissent servir à des jeux de données différents.

Exemple : pour permettre d'entrer une série de notes plus ou moins longue, on déclarera :

Réel **Note**[nmax]

où nmax est un entier positif exprimé en chiffres ou par une constante préalablement déclarée.

Taille d'un tableau : usage

On travaille en général sur un tableau à l'aide de boucles.

Exemple : on demande à l'étudiant combien il veut entrer de notes en lisant une variable **n**, puis on écrit une boucle "Pour" pour lire les valeurs val_i ($1 \leqslant i \leqslant n$).

On remplit un tableau par valeurs d'indice croissantes, mais. . . toutes les cases d'un tableau ne seront pas remplies ! $\Rightarrow$ *WARNING :* les variables Note[i] qui n'auront pas reçu d'affectation peuvent valoir *n'importe quoi*.

Réciproquement, si l'on cherche à accéder à un élément de tableau d'indice supérieur à sa taille déclarée, il y a débordement. Ceci provoque une erreur (signalée ou non, selon le langage) lorsque l'indice **i** est supérieur à la taille nmax.

Dans l'exemple ci-dessus, il peut être utile de vérifier que **n**$\leqslant$nmax . . . !

Dimension d'un tableau

Les tableaux dont on vient de parler sont de dimension 1, c'est-à-dire qu'ils correspondent à des *vecteurs*. On peut aussi définir des variables correspondant à des *matrices* en utilisant des tableaux de dimension 2, voire même des tableaux multidimensionnels de dimension d.

La dimension d'un tableau est par définition le nombre d'indices nécessaire à décrire l'un de ses éléments ; par exemple : Matrix[i,j]=m_{ij}

Matrix

m_{11}	m_{12}	...	m_{1j}	...	m_{1n}
m_{21}	m_{22}	...	m_{2j}	...	m_{2n}
...	...	...	...	...	...
m_{i1}	m_{i2}	...	m_{ij}	...	m_{in}
...	...	...	...	...	...
m_{p1}	m_{p2}	...	m_{pj}	...	m_{pn}

où i est l'indice de ligne et j l'indice de colonne.

Déclaration et usage

Pour déclarer un tableau de dimension 2, on écrit (pour la variable **Matrix** de l'exemple ci-dessus) :

Entier **Matrix**[p][n]

où p et n sont des entiers positifs, notés en chiffres ou définis par des constantes.

Chaque élément Matrix[i][j], pour un $i \leqslant p$ et un $j \leqslant n$ donnés, est alors une variable de type Entier et s'utilise comme telle.

N.B. On peut initialiser à la déclaration la liste des valeurs d'un tableau :

Entier **Matrix**[2][3]={ {1,2,3} , {4,5,6} }

pour créer le tableau bidimensionnel **Matrix**

1	2	3
4	5	6

dont on pourra ultérieurement modifier la valeur d'un ou plusieurs éléments.

En C, les indices commencent à 0

WARNING :

En C, les indices de tableau commencent nécessairement à 0, ce qui implique que les valeurs autorisées vont de 0 à $nmax - 1$ (donc 99, pour une variable de type tableau nommée **tablo** et ainsi déclarée) :

```
#define nmax 100    // commentaire : déclaration d'une constante
float tablo[nmax];
```

On peut initialiser un tableau à la déclaration :

```
int feu[4] = {3,2,1,0};
```

N.B. Pour ce tableau, on aura : `feu[0]=3` *et* `feu[3]=0` *mais la taille du tableau vaut bien* 4 *(cf. sa déclaration).*

Tableaux de caractères

Un tableau de caractères peut être initialisé par :

- une liste de caractères (comme un tableau de nombres) :
  `char tab[n]={'e','x','e','m','p','l','e'};`
- une chaîne de caractères littérale :
  `char tab[n]=''exemple'';`

Le compilateur C complète toute chaîne de caractères avec un caractère nul '\0'. Il faut donc que le tableau ait au moins un élément de plus que le nombre de caractères de la chaîne littérale.
$\Rightarrow$ il faut **n=8** même si le mot "exemple" ne comporte que 7 caractères.

Tableaux et affectation

En C, on ne peut pas affecter directement un tableau à une autre variable de type tableau (même si la dimension et la taille sont identiques) :

~~tablo2 = tablo1~~

Il faut passer par une boucle for :

```
#define nmax 5
int tablo1[nmax]={12,-7,5,14,23};
int tablo2[nmax];
main()
{
  for (int i=0; i<nmax; i++)
    tablo2[i]=tablo1[i];
  for (int i=0; i<nmax; i++)    // plus clair que faire tout dans une seule boucle
    printf(‘‘tablo1[%d]=%d\t tablo2[%d]=%d\n",i,tablo1[i],i,tablo2[i]);
}
```

WARNING : ne pas confondre indice et valeur !

Exercice :

CAHIER DES CHARGES :

Ecrire une boucle en C pour trouver le **maximum** d'un tableau de nombres réels ainsi que son indice.

CORRIGE : ..... À SUIVRE .....

Un type "structure"

Lorsqu'on veut définir une variable contenant plusieurs éléments qui ne sont pas tous du même type, on ne peut pas utiliser le type *tableau*. On définit alors un type structuré qu'on appellera **structure**.

Par définition, une **structure** est une suite finie d'objets de types différents.

Pour la déclaration, on procède en deux étapes :

1. on déclare un *modèle* de structure, en expliquant quels types déjà existants entrent dans sa composition, et on leur donne des noms,
2. puis on déclare une (ou des) variable(s) qui sont des objets de ce type

Notion de champ

Les différents éléments qui interviennent dans une structure sont appelés des champs. Chaque champ reçoit un nom.

Pour déclarer une variable bidule, du type structuré **Modele**, on adopte donc le schéma suivant :

1. ```
   Structure Modele
     type_1 champ_1
     type_2 champ_2
       ...
     type_n champ_n
   FinStructure
   ```
2. `Structure` **Modele** `bidule`
   ```

Usage d'une variable de type structuré

Les différents éléments sont repérés par le nom du champ (et non plus par un indice). On y accède en faisant suivre le nom de la variable par un point puis par le nom du champ : bidule.champ_i

Exemple : représentation d'un point du plan désigné par une lettre

```
// déclaration du modèle de type structuré
Structure Point
  Caractère nom
  Réel abs
  Réel ord
FinStructure
// déclaration d'une variable de type Point
Structure Point pnt=('P',3.2,-5.4) // initialisation à la déclaration
// manipulations sur cette variable
Début
  Réel dist_orig
  dist_orig = RacineCarree(pnt.abs * pnt.abs + pnt.ord * pnt.ord)
  Ecrire(''Distance du point ''+pnt.nom+'' à l'origine = ''+dist_orig)
Fin
```

Création d'un nouveau type

On peut aller plus loin en définissant un nouveau type ayant le nom du modèle de structure. *Exemple :* on pose **Point** comme étant un nouveau nom de type qu'on utilisera par la suite comme tous les autres types déjà connus.

Extension : ceci rejoint le principe de base des *langages orientés objets :* les classes ne sont autres que des types abstraits qui étendent la notion de type structuré. En POO[1], chaque classe est caractérisée par des **attributs**, qui correspondent aux champs de nos types structurés, et des **méthodes** qui définissent les codes des manipulations autorisées sur tous les objets de cette classe.

1. POO = Programmation Orientée Objet

Exemple de structure en langage C

N.B. Même si les champs sont, finalement, d'un même type, on peut avoir intérêt à déclarer une variable dans un type structuré plutôt que dans un tableau.

Exemple : représentation d'un nombre complexe

```
#include <math.h>
// déclaration du modèle de type structuré
struct Complexe
{
  double reel;   // partie réelle
  double imag;   // partie imaginaire
};
main()
{
// déclaration d'une variable de type Complexe
  struct Complexe z;
// manipulations sur cette variable
  double norme;
  norme = sqrt(z.reel * z.reel + z.imag * z.imag);
  printf(``norme de (%f + i %f) = %f \n'',z.reel,z.imag,norme);
}
```

Définition d'un nouveau type

Le mot-clé typedef permet de définir un nouveau nom de type. Reprenons l'exemple ci-dessus :

Même exemple : représentation d'un nombre complexe

```
#include <math.h>
// déclaration du modèle de type structuré
struct Complexe
{
  double reel;   // partie réelle
  double imag;   // partie imaginaire
};
typedef struct Complexe complexe;   // on définit ici le nouveau type
main()
{
// déclaration d'une variable de type complexe
  complexe z;   // ''complexe'' est utilisé comme un type usuel
                // on n'écrit plus le mot-clé ''struct'' à ce niveau-là
// manipulations sur cette variable
    ...  // [cf. page précédente]
}
```