Cours 1. Les éléments du langage C.

Dimitri Galayko

1 La compilation

La compilation d'un programme en langage C se fait en mode "terminal", à l'aide de la commande *gcc*:

```
gcc <code_source1.c> <code_source2.c> -o <nom_du_fichier_executable>
```

Ainsi, on passe deux arguments au programme *gcc*: le nom du ou des fichiers contenant le code source, et après l'option "-o" on écrit le nom qu'aura le fichier exécutable.

Si la compilation a réussi, aucun message n'est généré. Il y a 2 niveaux d'erreurs: "error" et "warning".

"error": la compilation a échoué, il faut modifier le code.

"warning": la compilation a réussi, mais le compilateur a trouvé des bouts de code "suspects" susceptible de générer une execution incorrecte.

Le nom du ficher source et le numéro de la ligne où l'erreur se produit sont donnés.

L'option *-wall* active les avertissements supplémentaires qui peuvent permettre de détecter des erreurs d'inattention, de mauvaises pratiques, etc.

2 Les mots clés du langage C

Un mot clé du langage C est un mot réservé. On ne peut pas appeler une fonction ou une variable par un mot clé. En voici la liste exhaustive, en ordre alphabetique:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Pourquoi connaître les mots clés ?

1) Chaque mot clé correspond à un outil du langage,

2) Ce sont les noms à éviter pour les variables.

3 Les commentaires

En langage C, 2 manière de faire des commentaires:

– à l'aide de deux slashes //, ce qui signifie "tout ce qui suit est un commentaire jusqu'à la fin de la ligne.

– par délimitation avec les symboles /* et */ .

```
// Ceci est un commentaire

/* Tout ceci est
   un commentaire
   jusqu'au symbole
   signifiant
   la fin des commentaires
**********************
*/
```

Les commentaires sont ignorés par le compilateurs. Ils servent à décrire le code, afin que le programmeur puisse se souvenir plus tard des ses propres intentions, mais aussi pour permettre à d'autres personnes utiliser le code.

Attention! Les commentaires inclus par le mécanisme /* ... */ sont interdits (pourquoi?).

Il existent des outils générant une documentation sur un projet informatique à partir des commentaires (par ex., *doxygen*). Les commentaires doivent alors être d'un certain format.

4 Notion d'identificateur

Un identificateur est tout mot rencontré dans un programme C (hors commentaires), qui ne fait pas partie des mots clés. On peut dire également, que les identificateurs sont les noms inventés par le concepteur, et donnés aux entités du programme (variable, fonction, type...). Attention, tous les identificateurs que vous utilisez ne sont pas inventés par vous, mais éventuellement par les personnes ayant créé les bibliothèques.

Les règles: – Les identificatueurs sont composés des lettres (de 'a' à 'z' et de 'A' à 'Z'), de chiffres et du signe '_' (*underscore* ou soulignement). Pas de lettres à diacritiques.

– Le premier caractère ne peut pas être un chiffre

– Les identificateurs sont sensibles au registre (majuscule, minuscule).

5 Les variables

5.1 Généralité - introduction

Une variable est une entité de programme (un objet) utilisé pour stoquer une valeur. On fait l'analogie avec la notion d'une variable utilisée dans l'algèbre.

En langage C, une variable a:

– le type, c.a.d., quel est la nature de la valeur qui peut y être stoquée (entière, à virgule flottante, le nombre de bits, etc.) ,

– La durée de vie (variable locale/globale/statique)

– La portée de la variable (la zone de visibilité).

– Le support physique pour le stoquage d'information (variable stoquée dans la mémoire ou dans un régistre d'ordinateur).

Chaque variable utilisée dans le programme doit être déclarée avant d'être utilisée. La declaration d'une variable se fait de la manière suivante:

<nom_du_type> <nom_de_la_variable>

Le nom de la variable est un identificateur, on le choisit comme on souhaite (avec l'objectif d'améliorer la lisibilité du programme). Le type de la variable doit être connu au compilateur. Le langage C prévoit les types élémentaires tels que "valeurs entières", "valeurs à virgule flottante", "pointeurs", "type vide". Les autres types sont dite composés (structures, unions), on les étudiera plus tard.

5.2 Les types entiers

Il existe 4 types de variables entières:

char	short	int	long	long long
1 octet	2 octet	2 ou 4 octets	4 ou 8 octets	8 octets
8 bits	16 bits	16 ou 32 bits	32 bits	64 bits

La seule différence entre ces types est le nombre de bits sur lesquels les valeurs sont codées. Pour rappel, le nombre de valeurs différentes codées par un type entier représenté sur n bits est exactement 2^n. Ainsi, le type *char* ne peut coder que 256 valeurs (2^8), le type *long* code 2^{32} valeurs, etc.

Un type entier peut être signé ou non-signé. Un type signé code un nombre entier signé, codé en "complément à 2". Un type non-signé code uniquement des nombres non-négatives. Le mot clé *unsigned*

devant le nom du type permet de spécifier qu'une variable est de type non-signé; par défaut les variables entières sont signées.

Un type entier donné code le même nombre de valeurs qu'il soit signé ou pas. En revanche, la plage des valeurs est différente:

$0...2^n$ pour la version non-signée d'un type entier sur n bits,

$-2^{n-1}...2^{n-1}-1$ pour la version signée d'un type entier signé sur n bits.

```
char i; // i est une variable de type char, signée. Elle code les valeurs
        // de -128 à 127, ça fait 256 valeurs au total.

unsigned char b; // b est une variable non-signée, elle code les valeur
                 // de 0 jusqu'à 255.
```

Deux remarques: longueur des types entiers et le type *char*.

- Longueur des types entiers. La taille des types entiers n'est pas garantie par le standard. On sait seulement que la tailles des types va croissant du type *short* au type *long long*. Par exemple, le nombre d'octet occupé dans la mémoire par une variable de type *int* dépend de l'architecture d'ordinateur: 2 octets pour les processeurs jusqu'à 16 bits, 4 octets pour les systèmes 32 et 64 bits. Sur les PC, ce sera toujours 4 octets, mais ça peut être 2 si on programme un microcontrolleur. De même, le type *long* peut être sur 4 ou 8 octets. Ainsi, faire attention veut créer un code indépendant de la plate-forme, et si l'execution de notre programme peut être affectée par la capacité de la variable entière.

- Le type *char*. On dit souvent qu'il est destiné pour coder les caractères. Ceci n'est pas correcte: c'est un type entier comme les autres, tout simplement, de petite capacité (8 bits). Hisotiquement, les caractères ont été codé sur 8 bits, donc ce type a effectivement été créé pour coder les caractères. Cependant:

  – de nos jours, le codage des caractères sur 8 bits est obsolète (on doit utiliser les systèmes basés sur Unicode comme UTF8), et on code un caractère sur un nombre de bits variable,

  – rien n'empêche d'utiliser une variable déclarée comme *char* pour stoquer les valeurs entières, l'utiliser dans les opérations arithmétiques, etc.

Pour connaître la taille des types, on utilise la fonction *sizeof* qui s'emploie ainsi:

```
a=sizeof(<nom du type>);
```

la variable a contiendra le nombre d'octets correspondant au type. Par exemple :

```
a=sizeof(int);
```

etc.

5.3 Les types flottants

Une variable de type "à virgule flottante" codent les valeurs réelles de type $1.2254 \cdot 10^5$. On parle de la virgule flottant car, quelque soit la valeur du nombre, il y a toujours exactement 1 chiffre avant la virgule. Par exemple, 192345.55 est représenté comme $1.9234555 \cdot 10^5$.

Il y a 3 types à virgule flottante. Ils se distinguent de la taille : *float*, *double* et long double. Ils sont toujours signés. La table ci-dessous en donne les paramètres.

	float	double	long double
	4 octets	8 octets	16 octets
Valeurs minimales	$1.2 \cdot 10^{-38}$	$2.3 \cdot 10^{-308}$	$3.4 \cdot 10^{-4932}$
Valeurs maximales	$3.4 \cdot 10^{38}$	$1.7 \cdot 10^{308}$	$1.1 \cdot 10^{4932}$

5.4 Les constantes

Une constante est une valeur numérique écrit dans un code. Il ne peut pas être modifié lors de la compliation.

Exemple:

```
int i;

// un code quelconque
//.....
//....

if (i<234)
        {
// ...
a=25;
```

Ici 234 et 25 sont des constantes: la variable *i* sera toujours comparée avec 234, et la variable *a* sera toujours affectée de 25.

Les constantes sont, en général les nombres entiers ou à virgules. Les constantes ont les mêmes types que les variables. Les constantes entières peuvent être écrites en différentes systèmes de numération.

5.5 Constantes entières numériques

Il s'agit des nombres entiers exprimés avec les chiffres habituelles. Le préfixe désigne le système numération: *0x* pour l'hexadécimal, *0* pour les octaux. Sans préfixe sont notées les constantes codées en décimal (par défaut).

Exemple: *0x25fb3, 0x2200, -0124, 23*. Ici, les deux premières constantes sont écrites en système hexadécimal, la troisème en octal, et la deuxième en décimal.

Le suffixe précise le type des constantes:

- l ou L pour les constantes de type *long*,
- ll ou LL pour les constantes de type *long long*,
- u ou U pour les constatnes non-signées.

Par défaut, les constantes sont de type *int* signé.

Exemples :

245U - *int*, non-signée, codée en décimal *0x23l* - *long*, signée, codée en héxadécimal

5.6 Constantes entières sous forme de caractères

Le compilateur C sait associer un caractère avec son code ASCII. Il est alors possible d'employer les caractères comme valeurs pour les constantes entières. Puisque la table ASCII a 265 valeurs, il s'agit d'un type sur un octet: le type *char*. Une constante entière de type "caractère" est alors écrite comme un caractère entre apostrophes. Le code suivant illustre quelques exemples d'utilisation de constantes de type *char*.

```
    char a, b, c;
    a='3'; // le code ASCII de '3' est 51 en décimal
    a=51; // Cette ligne est presque équivalente à la ligne précédente
    b='#'; // le code ASCII de '#' est 35 en décimal
    c=a+b; // on peut juste additionner a et b comme n'importe quelles variables enti
        ères
    printf("%i\n", c); // Cette commande affiche 86  (51+35).
    printf("%c\n", c); // Cette commande affiche V dont le code ASCII est 86.
                                        // %c signifie que la variable doit être
                                            affiché par son code ASCII
                                        // (cf. la suite de ce document.
```

Il est important de comprendre, que la constante '#' est presque équivelente à 35. La différence est que la constante 35 est (par défaut) de type *int* et occupe 2 ou 4 octets, alors que '#' est de type *char* et donc occupe 1 octet. Cependant, grâce au mécanisme de conversion de types (cf. les cours suivants), cette différence n'est pas visible dans le code ci-haut.

Il existe un certain nombre de caractères spéciaux (codes) qui se définissent par une séquence de caractères:

'\n'	"\t'	'\b'	'\"	'\"'	'\?'
retour à la ligne	tabulation	backspace	'	"	?

On utilise également les chaînes de caractères: les séquences de plusieurs caractères enfermées entre guillemets. En particulier, les fonctions *printf* et *scanf* utilisent les chaînes de charactères en argument:

```
printf("Ceci est une chaine de caracteres\n");
```

On apprendra davantage sur les chaînes de caractères lorsque l'on étudiera les pointeurs.

5.7 Constantes à virgule flottante

Les constantes à virgule flottante s'écrivent ainsi:

```
float a;
a=1.256e-5; // e-5 signifie "fois 10 puissance -5"
a=1.256E-5; // la lettre e peut être majuscule
a=.23; // absence de chiffre avant la virgule signifie 0 avant la virgule
a=234.34e-5// on peut avoir la partie supérieure à 1 composée de plusieurs chiffres
```

6 Affichage en langage C

Le langage C n'est pas doté des moyens d'affichage d'information à l'écran. Les outils le permettant font partie des bibliothèques standards, et sont réalisés par des fonctions. Avant de présenter ces outils, nous introduisont la notion de la fonction en langage C.

7 Utilisation des fonctions

Dès les premiers pas dans le langage C, on est obligé d'utiliser le mécanisme appelé "fonction". Les fonctions sont des programmes accomplissant une action bien déterminée. Les fonctions prennent en entrée des paramètres, appelés *arguments*, qui spécifient l'action de la fonction. Par exemple, la fonction peut calculer le cosinus de la valeur passée en argument, ou afficher la valeur de la variable passée en argument d'une certaine manière, ou dessiner un cercle de diamètre donné.

Une fonction retourne une valeur qui peut être enregistrée dans une variable. En langage C, les fonctions ne peuvent pas retourner des tableaux (un ensemble de plusieurs valeurs).

Pour utiliser une fonction, on effectue un *appel*. Un appel de fonciton se fait par une ligne de code de format suitant:

nom_de_la_fonction(argument1, argument2, ...);

Le nom de la fonction est un identificateur (cf. plus haut). Il est choisi par la personne qui a conçu la fonction. Les arguments sont les valeurs des paramètres qui doivent être transmises à la fonction pour spécifier son exécution.

Par exemple, la fonction de nom *pow* calcule la valeur de l'expression "x à la puissance y". Elle prend deux arguments: le premier c'est donc x (le nombre qu'il faut élever à une puissance), le deuxième est y (la puissance). La fonction retourne le résultat de l'opération.

La fonction *printf* est un autre exemple d'une fonciton. Celle-ci reçoit un nombre d'argument variable, selon le nombre des valeurs que l'on souhaite visualiser (cf. plus loin sur la fonction *printf*).

Les arguments de la fonction sont de types pré-définis. Par exemple, une fonction peut exiger que le premier argument soit de type *int*, le deuxième de type *float*, etc. Dans la fonction *pow*, les deux arguments sont de type *double*.

Une utilisation minimale d'une fonction consiste à *appeller* la fonction: demander au programme de l'executer en spécificant la (les) valeur(s) de ses arguments. Cela peut se faire sur une ligne:

```
printf("Affichez moi cela!\n"); // appel de la fonction prinf
pow(2,3); // calculer 2 à la puissance 3. Cette appel est inutile, car le résultat n'
    est pas utilisé (cf. l'exemple suivant).
```

Voici un exemple de code montrant l'utilisation de la fonction *pow*. Il montre que les appel des fonction

```
// on déclare les variables:
double a, b, c, d;

// on initialise a et b
a=3;
b=0.5;
c=pow(a,b); // cette ligne calcule la puissance 1/2 de 3, autrement dit,
                    // la racine carrée de 3
                        // le résultat se retrouve dans la variable c

c=pow(-3,-4); // ici la variable c contiendra -3 élevé à la puissance -4.

d=1+pow(a,b)*2; // on utilise la valeur retournée dans une expression arithmétique
```

Il est important de comprendre que le langage C fournit uniquement le mécanisme permettant de créer des fonctions. Toutes les fonctions que l'on utilise sont définies par quelqu'un. Les fonctions ajoutées forment un complément aux outils offerts par le langage.

7.1 Affichage formaté du contenu des variables

La fonction *printf* est essentiellement utilisée pour afficher le contenu des variables en même temps que du texte, sur le terminal.

Contrairement à la plupart des fonctions *printf* a un nombre variable d'argument.

Le premier argument de la fonction est toujours une chaîne de caractères (pour rappel, une séquence de caractères enfermée entre guillemets). C'est cette chaîne qui sera affichée. La plupart des caractères sont affichés tels quels. Par exemple, la ligne de code

```
printf("Hello, world!");
```

affichera tout simplement *Hello, world!*. Le curseur sera positioné juste après le caractère d'exclamation. Ainsi, si plus tard on veut reafficher la même chose, on aura sur le terminal :

```
%./hello_world
Hello, world!Hello, world!%
```

Ce n'est évidemment pas le résultat que l'on souhaite. L'insertion du symbole "retour à la ligne résoud le problème:

```
printf("Hello, world!\n");
```

On aura alors :

```
%./hello_world
Hello, world!
Hello, world!
%
```

Si on souhaite afficher des valeurs des variables, il faut insérer dans le texte à afficher une séquence de format suivant:

`%[flag][largeur][.précision][modificateur]type`

et après la fin de la chaîne de caractères, donner des valeurs. Par exemple,

```
int a;
a=3;
printf("La valeur de la variable a est %d, \n", a);
```

Ce code affichera "La valeur de la variable a est 3".

On passe maintenant en révu toutes les composantes de cette séquence.

1. % est le symbole qui signifie le début de la séquence. Si ce symbole doit être affiché tel quel, il faut utiliser le caractère d'échappement "
%".

2. *[flag]* Ce champ fournit des options de cardage. Les crochets signifient que l'utilisation de ce champ est optionnelle. Par défaut, la valeur affichée est justifiée à gauche (des espaces sont ajoutés si nécessaire). Les valeurs suivantes sont possilbes:

- justification à gauche de la valeur affichée

+ affiche le signe (+ ou -) avant la valeur numérique

espace : imprime un espace dvant un nombre positif, à la place du signe.

3. *[largeur]* donne le nombre minial de caractères à imprimer (des espaces sont ajoutés si nécessaire). Cependant, si le texte à écrire est plus long, il est écrit en totalité.

4. [.precision] définit, pour les réels, le nombre de chiffres après la virgule (ce nombre doit être inférieur à la largeur). Dans le cas de nombres entiers, ce champ indique le nombre minimal de chiffres désirés, avec l'ajout de 0 sinon. Pour une chaîne de caractères (%s), ce paramètre indique la longueur maximale imprimée (tronquée si trop longue).

5. [modificateur] modifie l'interprétation à donner à la valeur à afficher. Les valeurs possibles sont *h* pour *short*, *l* (*long* pour les entiers, *double* pour les réels), ou encore *L* (*long double* pour les réels).

6. *type* précise l'interprétation à donner à la valeur. Les valeurs possibles sont données dans la table ci-dessous.

d	int	décimale signée
u	unsigned int	décimale non signée
o	int	octale non signée
x	int	hexadecimale non signée
f	int	décimale virgule fixe
e	int	décimale notation exponentielle
g	int	décimale, représentation la plus courte parmi "virgule fixe" et "notation exponentielle"
c	unsigned char	caractère
s	char *	chaîne de caractères

La fonction *printf* retourne le nombre de caractères écrits, ou EOF (le caractère End Of File) si pour une raison quelconque l'affichage a échoué.

7.2 Affichage de caractères

La fonction *putchar* affiche un caractère à l'écran. Par exemple:

```
char ch;
ch='e';
putchar(ch);
```

Le résultat de se programme sera l'affichage du caractère 'e' à l'écran. Il n'y a pas de retour à la ligne du curseur.

8 Lecture depuis le clavier

8.1 La lecture formatée: scanf

La fonction scanf permet de lire les données saisies au clavier, en format spécifié.

Exemple:

```
int a;
scanf("%d", &a);
```

Ce bout de code attends à ce que l'utilisateur saisisse un nombre entier en format décimal. La valeur saisie est enregistrée dans la variable *a*.

L'utilisation de la fonction *scanf* est similaire à celle de *printf*. Le premier argument définit le format du ou des valeurs lues, avec un peu moins d'options. Le format s'écrit sous la forme:

`%[*][largeur][modificateur]type`

[*] : si étoile est écrite après %, la valeur sera lue mais ne sera pas stockée

[largeur] : le nombre maximal des caractères à lire

[modificateur] : permet de préciser la longeur (la capacité) pour la donnée: *h* pour *short*, *l* pour *long*, *L* pour *long double*.

[type]: spécifie le type de donnée lue et comment il faut le lire (cf. le tableau).

Type	description	Argument requis
c	caractère simple, espace inclu	char *
d	entier, signé ou pas	int *
e,E,f,g,G	nombre à virgule (éventuellement en format exponentiel)	float *
o	entier en notation octale	int *
x	entier en notation exadécimale	int *
u	entier non-signé	unsigned int *
s	chaîne de caractères (jusqu'à la lecture d'un espace)	char *

Après le format de lecture (délimité par les guillemets), on met les *adresses* des variables vers lesquels les résultats seront lus. Les adresses sont spécifiées à l'aide du symbole "et commerciale" placé devant le nom de la variable.

Voici trois exemples de lecture de données au clavier. Notez qu'avant chaque utilisation de *scanf*, il est de bon usage d'afficher un message invitant à saisir une valeur.

```
int a1;
double b2;
char ch;
printf("Saisissez une valeur entière en numération décimale: ");
scanf("%d", &a1);

printf("Saisissez une valeur réelle: ");
scanf("%e", &b2);

printf("Saisissez un caractère: ");
scanf("%c", &ch);
```

8.2 Lire juste un caractère

La fonction *getchar* lit exactement un caractère sur le clavier. En voici l'exemple d'utilisation.

```
char ch;
printf("Saisissez un caractère :");
ch=getchar();
printf("Le caractère saisi est %c\n", ch);
```