Langage C

Olivier Roques

2016-2017

1 Syntaxe

1.1 Les variables

Déclaration de variable :

```
extern <type> <identifiant>;
```

Définition d'une variable :

```
<type> <identifiant> = <valeurInitiale>;
```

Le type définit :

- L'encombrement dans la mémoire de la variable (sa taille, en octets) ;
- Les opérateurs associés ;
- La zone mémoire où est placée la variable.

L'emplacement détermine :

- Sa durée de vie (qui peut être celle du programme ou celle de la fonction) ;
- Sa visibilité (variable locale ou globale).

1.2 Les entiers

Plusieurs types d'entiers : `char` (sur 1 octet), `short` (sur 2 octets), `int` et `long` (souvent sur 4 et 8 octets respectivement mais dépend de la machine). Le type `boolean` n'existe pas.

Qualificatif `unsigned` : On n'utilise pas le bit de signe, ce qui double la dynamique des positifs.

1.3 Les types composés et le type enum

On peut composer les types de bases pour créer un type composé avec le mot-clé `struct` :

```
struct complexe{
    float reel;
    float imagine;
};
struct complexe c1 = {2.0, 3.0};
```

L'opérateur = est le seul qui s'applique à `struct`.

L'instruction `typedef` permet d'associer un raccourci à un nom de type :

```
typedef <type> <nom_nouveau_type>;
```

On peut définir un type comme une énumération de valeurs constantes possibles avec le mot-clé `enum`. Par défaut la première valeur est considérée comme un entier constant égal à 0.

```
enum bool{
    false, // ou bien true = 1, false = 0
    true
};
```

1.4 Les pointeurs

Un pointeur est une variable contenant l'adresse (la référence) d'une autre variable. `NULL` indique que le pointeur n'a pas été initialisé.

```
<type_objet_pointeur> *ptr;
ptr = &variable;      //& renvoie l'adresse (la référence) de 'variable'
variable = *ptr;      //* renvoie la valeur contenue à l'adresse 'ptr'
*ptr = adresse.
/* à gauche de '=', * permet de changer la valeur de la variable
contenue à l'adresse 'ptr' **/
```

1.5 Les tableaux et chaînes de caractères

La taille des tableaux doit être connue à la compilation. Le nom d'un tableau est en fait un pointeur constant sur l'adresse de début d'un tableau.

```
<type> nom_tableau[taille_tableau] = {value1, ...};
```

Une chaîne de caractère est un tableau de `char`, et le caractère `NULL` indique la fin de la chaîne.

1.6 L'opérateur sizeof()

L'opérateur `sizeof()` renvoie la taille d'une variable ou d'un type, en octets. La taille d'un type composé est supérieure à la somme de la taille de chacun de ses éléments (c'est la cas en particulier pour les tableaux).

1.7 Fonctions : déclaration et implémentation

La déclaration d'une fonction, appelée signature ou prototype, doit être placée en début du fichier source. Elle suit le schéma suivant :

```
<type_de_retour> <nom_de_la_fonction>(<arguments>);
```

L'implémentation d'une fonction suit le schéma suivant :

```
<type_de_retour> <nom_de_la_fonction>(<arguments>){
    // Déclaration des variables locales
    // Instructions
    return <valeur>;
}
```

`return` interrompt l'exécution de la fonction en cours d'exécution (et donc du programme lorsque `return` est invoqué dans la fonction `main`).

La fonction `main` est le point d'entrée du programme. Son implémentation suit le schéma suivant :

```
int maint(int argc, char* argv[]){
    // début du programme
    // ...
    // fin du programme
}
```

Ici, `argc` est nombre d'arguments passés sur la ligne de commande au lancement du programme et `argv` est le tableau de chaînes de caractères contenant les arguments passés sur la ligne de commande (séparés par un espace).

1.8 Fonction d'entrée, de sortie

Les fonctions d'entrée sortie s'appuient sur la notion de chaîne de caractères formatée : une chaîne de caractères à trous, dont les trous sont positionnés par l'identifiant de format. Ces formats identifient le type de valeur qui viendra remplacer le texte manquant :

- `%s` : une chaîne de caractères ;
- `%c` : un caractère ;
- `%d` : un entier (de type `int`, `char` ou `short`) ;
- `%ld` : un entier (de type `long`) ;
- `%f` : un nombre réel ;
- `%x` : un entier en hexadéciamal.

On utilise la fonction `printf` pour afficher du texte sur la sortie standard (la console par défaut).

2 Les pointeurs

Tous les pointeurs sont implémentés sur le même nombre d'octets.

Les tableaux Il n'y a pas de tableaux en C : le nom d'un tableau est un pointeur constant sur l'adresse de début d'un tableau, il ne peut donc pas être modifié. Dans le code suivant :

```
int tab[100];
int *prt;
ptr = tab;
```

`ptr` et `tab` désigent le même tableau. Les notations suivantes sont alors équivalentes :

```
tab[i] = 0;
*(tab + i) = 0;
ptr[i] = 0;
*(ptr + i) = 0;
```

Si on a `TYPE *ptr;`, alors `ptr + k` désigne la case mémoire d'adresse (valeur de `ptr`) + k*sizeof(TYPE).

Les chaînes de caractère Il n'y pas de type `String`, les chaînes de caractère sont représentées par des tableaux de `char`. La fin d'un tableau est renseigné par la valeur '\0', appelé caractère `NULL`.

Si à travers une fonction, on veut modifier la valeur des variables passées en paramètres, ou si on veut passer en paramètres des structures lourdes, on passe en paramètre l'adresse de la variable.

Vocabulaire :

- Paramètre formel : paramètre donnée à la définition d'une fonction.
- Paramètre actuel : paramètre donnée lors de l'appel de la fonction.

3 Allocation dynamique de mémoire

```
malloc(sizeOf(...)); /** allocation, renvoie un void */
realloc(sizeOf(...); /** reallocation, renvoie un void */
free(adresse);       /** liberation, renvoie un void */
```

4 Chaîne de production

Espace d'adressage d'un programme On distingue deux zones :

- L'une dédiée aux informations allouées dynamiquement, pendant l'exécution du programme : on y trouve une pile (pour les variables locales) et un tas où se trouvent les variables allouées par malloc.
- L'autre, statique, allouée à la compilation : on y trouve notamment une zone accessible à la fois en lecture et en écriture (ou sont stockées les variables globales et static).

Compilateur Il traduit le fichier source (.c) en langage d'assemblage et affecte des adresse aux variables. Il construit un fichier objet (.o).

Éditeur de lien Rassemble les fichiers objets pour construire un fichier exécutable, il essaie de résoudre toutes les références non satisfaites en consultant des bibliothèques.

L'outil gcc Il crée un fichier exécutable à partir d'un fichier fic.c en plusieurs étapes :

1. Appel du préprocesseur C (cpp) qui crée un fichier fic.i
2. Appel du compilateur C (cc1) qui crée un fichier assembleur fic.s
3. Appel de l'assembleur (as) pour générer le fichier objet fic.o à partir de fic.s
4. Appel de l'éditeur de liens (ld), la bibliothèque C standard est incluse par défaut et un fichier exécutable a.out est produit, si possible. Les fichiers intermédiaires sont détruits.

Bibliothèques partagées Les fonctions de ces bibliothèques sont chargées en mémoire à l'exécution du programme. Si la fonction est déjà chargée, ce binaire sera partagé par tous les exécutables qui l'utilisent.

Les fichiers exécutables sont alors plus petits, en effet les fichiers objets manquants ne sont pas ajoutés lors de l'édition de liens, mais remplacés par des références vers des fichiers exécutables.

Directives au préprocesseur Les lignes commençant par #, appelées directives, sont traitées par le préprocesseur avant la compilation. Elles permettent d'insérer la déclaration des types et fonctions de la librairie cocernée. Lors de l'appel à gcc, cette directive est remplacée par le contenu du fichier.

- define : effectue des substitutions de chaînes de caractères.
- ifdef, ifndef, endif : si la variable suivant ifdef n'est pas définie, les instructions comprises entre ce ifdef et le endif suivant ne sont pas incluses dans le fichier à compiler. Même principe pour ifndef.
- include : demande au préprocesseur d'ajouter le contenu d'un fichier dans le fichier courant. Avec <> : bibliothèques partagées, avec "" : prototypes.

Cible et dépendance : fichier Makefile Un fichier Makefile contient la représentation du graphe de dépendance d'une application sous forme de texte. Il contient aussi, à la suite de chaque règle de dépendance, les actions à entreprendre pour passer des dépendances à la cible.