C

L3IF - Projet 1

Bogdan Pasca <bogdan.pasca@ens-lyon.fr>
Marin Bougeret <marin.bougeret@imag.fr>
Kevin Perrot <kevin.perrot@ens-lyon.fr>
ENS Lyon 2010-2011

Cours inspiré de "Introduction au langage C" de Bernard Cassagne

Introduction

Introduction - Présentation

C est un langage de programmation...

- impératif : les instructions sont exécutées en séquence
- bas niveau : on est (relativement) proche du langage machine, ce qui nous donne une bonne vitesse d'exécution.
- conçu pour l'écriture système : on peut interagir avec le système par des appels de fonctions contenues dans des bibliothèques.

Introduction - Ecriture/Compilation/Exécution

Un programme en C...

1. s'écrit à l'aide d'un éditeur de texte dans un fichier avec l'extension `.c`
   `monFichier.c`
2. se compile avec `cc`, `gcc` ou votre compilateur favori, ce qui donne un exécutable (un "binaire")
   `monFichier`
3. s'exécute
   `./monFichier`

Introduction - Exemple

monFichier.c :

```
#include<stdio.h>
int main(){
  printf("hello world \n");
  return 0;
}
```

la fonction `printf` (de la bibliothèque `stdio.h`) permet d'afficher une chaîne de caractères sur la sortie standard (l'écran à priori) et `\n` désigne un retour à la ligne. La première ligne inclut la bibliothèque d'entrée sortie pour utiliser l'affichage.

compilation :

```
$ gcc monFichier.c -o monFichier
```

l'option `-o` permet de préciser le nom de l'exécutable produit à la compilation (par défaut son nom est "a.out").

exécution :

```
$ ./monFichier
hello world
```

Introduction - Remarques

- Le symbole `;` termine une instruction.
- Une séquence d'instructions entre accolades `{...}` forme un bloc.
- Les commentaires sont entre les marqueurs `/*` et `*/`
- Certains mots clés ne doivent pas être utilisés comme nom de variable, par exemple `auto double int struct break else long switch case enum register typedef char extern return union const float short unsigned continue for signed void default goto sizeof volatile do if static while`
- Au lancement du processus, la machine exécute la fonction `int main()`

Variables

Variables - Types de base

En C on peut déclarer et utiliser des variables, qui contiennent une valeur d'un certain type.

Les types de base comprennent :

- `int` : entier 32 bits
- `char` : caractère
- `float` : flottant simple précision
- `double` : flottant double précision

Variables - Déclaration

On déclare une variable en indiquant son type et en lui donnant un nom et éventuellement une valeur initiale.

```
int i; /* déclare un entier i */
int i,j; /* déclare deux entiers i et j */
int i=0; /* déclare un entier i valant 0 */
int i=0,j=32; /* déclare deux entiers i et j
                 valant (resp.) 0 et 32 */
float f=3.14;
double g=1.41421;
char c='a'; /* les caractères s'écrivent '.' */
```

Une variable est visible uniquement à l'intérieur du bloc dans lequel elle est définie.

Opérateurs

Opérateurs - Affectation

Une affectation est de la forme `nomVariable = expression`

- Elle affecte la valeur de `expression` à la variable `nomVariable`
- Une affectation vaut la valeur affectée

```
i=1
j=i+1
j=1+(i=2) /*j vaut 3 */
f=0.99
c='b'
i++ /* incrémente i */
i-- /* décrémente i */
```

On peut par exemple affecter un entier à une variable flottante, mais la conversion de type n'est pas toujours possible (attention !).

Opérateurs - Arithmétique

Additon : `expression + expression`
Soustraction : `expression - expression`
Multiplication : `expression * expression`
Division : `expression / expression`
Modulo : `expression % expression`

- Les deux membres de ces expressions sont évalués dans un ordre non-prédéfini (attention aux effets de bord).
- On peut effectuer des opérations avec des expressions de types différents mais compatibles (par exemple, un entier moins un flottant donne un flottant).
- `- expression` $\Leftrightarrow$ `0 - expression`
- Les divisions par zéro font planter l'exécution.

Opérateurs - Comparaison

`expression op expression`

`op` :=	`>`	strict. supérieur
	`<`	stricy. inférieur
	`>=`	supérieur ou égal
	`<=`	inférieur ou égal
	`==`	égal
	`!=`	différent
	`&&`	et logique
	`||`	ou logique
	`!`	non logique

Les deux expressions sont évaluées puis comparées. La comparaison donne une valeur de type `int` valant 1 si la condition est vraie, 0 sinon.

Attention aux parenthèses pour les expressions logiques.
Pour les expression logiques, les membres sont évalués de gauche à droite.

Instructions

Instructions - Expressions

`expression ;`

L'expression est évaluée et sa valeur est ignorée.

```
i=1;
j=i+1;
j+1; /* instruction inutile */
```

Instructions - Composées

```
{
  instructionsDeDéclarations
  instructions
}
```

- Avantage de grouper des instructions sous la forme d'une seule instruction (un "bloc").
- Une variable déclarée dans un bloc n'est accessible qu'à l'intérieur du bloc.

Instructions - Conditionnelles

`if (expression)` $\texttt{instruction}_1$
`if (expression)` $\texttt{instruction}_1$ `else` $\texttt{instruction}_2$

`expression` est évaluée. Si sa valeur est non nulle, $\texttt{instrcution}_1$ est exécutée, sinon $\texttt{instrcution}_2$ est exécutée si elle existe.

- `expression` doit être entre parenthèses.
- Les blocs sont utiles pour exécuter plusieurs instructions dans une branche d'une structure conditionnelle.
- Evitez les ambiguïtés dans les `if` imbriqués. Les ambiguïtés sont levées en faisant correspondre les `else` avec les `if` les plus proches.

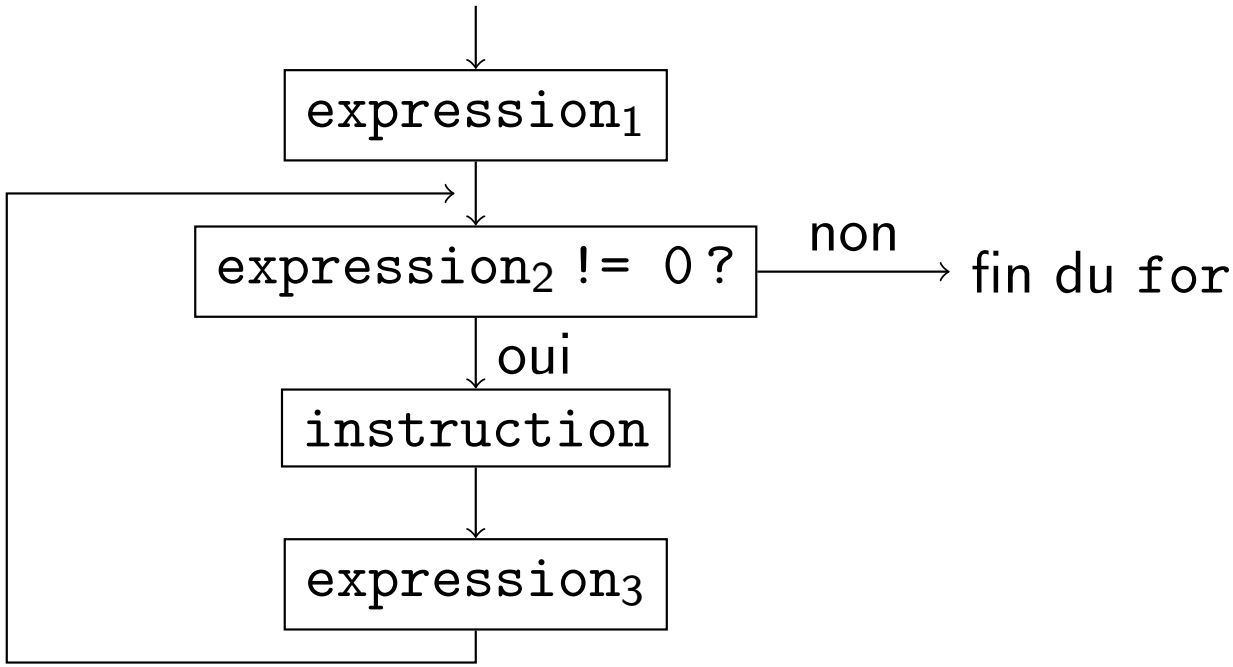
Instructions - Conditionnelles - Exemple

```
if (i >= j) {
  if (j != 1)
    i = 1;
}
else{
  i++;
  j--;
}
```

Instructions - Boucle `for`

`for (expression`$_1$ `;expression`$_2$ `;expression`$_3$`) instruction`

L'exécution correspond à l'organigramme :



- `expression`$_1$ et `expression`$_3$ ont pour rôle de réaliser des effets de bord (initialisation, itération).
- `expression`$_2$ est un test de bouclage.
- `instruction` est le travail de la boucle.

Instructions - Boucle `for` - Exemple

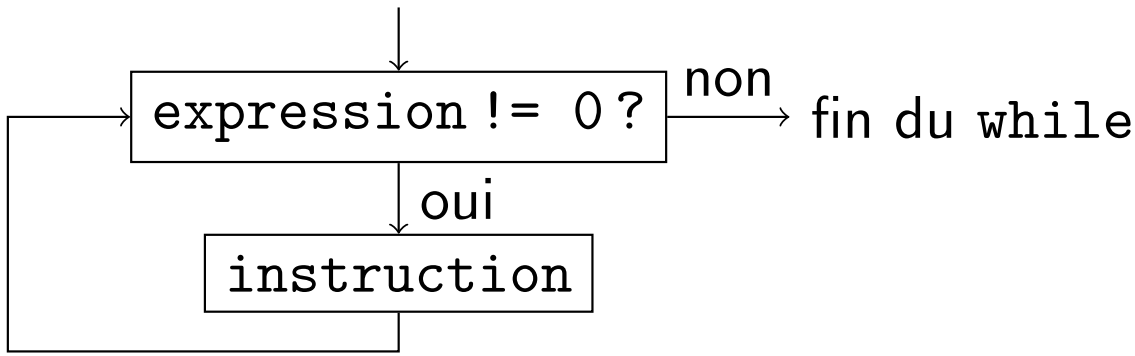
```
j=0;
for(i=1; i<5; i++){
  j = j + i;
}
```

A la fin, `j` vaut 10.

Instructions - Boucle `while`

`while (expression) instruction`

L'exécution correspond à l'organigramme :



Instructions - Boucle `while` - Exemple

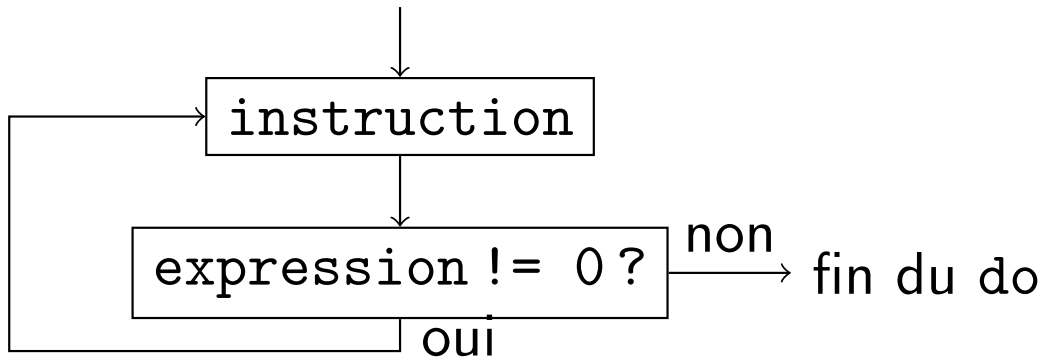
```
j=1;
i=2;
while(j<100){
  j = j * i;
}
```

A la fin, `j` vaut 128.

Instructions - Boucle do

```
do instruction while (expression) ;
```

L'exécution correspond à l'organigramme :



Instructions - Boucles - `break` et `continue`

- L'instruction `break ;` provoque l'arrêt de la première boucle `for`, `while`, `do` englobante.
- L'instruction `continue ;` provoque l'arrêt de l'itération courante et le passage à l'itération suivante de la première boucle `for`, `while`, `do` englobante.

Fonctions et Procédures

Fonctions et Procédures

Voir les chapitres

- 1.15 (sauf 1.15.8)
- 1.16
- 1.17
- 1.18
- 1.19

de "Introduction au langage C" de Bernard Cassagne.

Tableaux

Tableaux

Voir les chapitres

- 2.1
- 2.3

de "Introduction au langage C" de Bernard Cassagne.