Le langage C, et sa sémantique

“C is quirky, flawed and an enormous success.”

Dennis Ritchie, *The Development of the C Language*, ACM SIGPLAN Notices 28(3):201–208, mars 1993.

Une vision simplifiée de C: 1. les expressions

Les expressions calculent des valeurs:

e	$::=$	x	variables
	$\mid$	`1` $\mid$ `2` $\mid$ `...` $\mid$ `-1` $\mid$ `-2` $\mid$ `...` $\mid$ `'a'` $\mid$ `'b'` $\mid$ `...`	constantes
	$\mid$	e`+`e $\mid$ e`-`e $\mid$ e`*`e $\mid$ e`/`e $\mid$ e`%`e $\mid$ `-`e	arithmétique
	$\mid$	e`&`e $\mid$ e `|` e $\mid$ e`^`e $\mid$ e`>>`e $\mid$ e`<<`e $\mid$ `~`e	opérations bit à bit
	$\mid$	e`==`e $\mid$ e`<`e $\mid$ e`<=`e $\mid$ e`>`e $\mid$ e`>=`e	comparaisons
	$\mid$	e`&&`e $\mid$ e `||` e $\mid$ `!`e	logique
	$\mid$	`*`e $\mid$ e`[`e`]` $\mid$ `&`e	pointeurs, tableaux
	$\mid$	e`(`e`, ..., `e`)`	appel de fonction

Une vision simplifiée de C: 2. les commandes ("statement")

Les commandes modifient la mémoire et forment une relation de transition:

c	$::=$	e`=`e`;`	affectation (attn! le test c'est ==!)
	$\mid$	`;`	ne fait rien
	$\mid$	$c\ c \ldots c$	séquence
	$\mid$	`{` ⟨déclations de variables⟩ c `}`	bloc
	$\mid$	`if (`e`)` c [`else` c]	conditionnelle (test)
	$\mid$	`while (`e`)` c	boucle while
	$\mid$	`do` c `while (`e`)`	boucle repeat
	$\mid$	`for (`[e]`;`[e]`;`[e]`)` c	boucle for
	$\mid$	`switch (`e`) {`	analyse de cas
		(`case` e`:` c `break;`)*	
		[`default:` c `break;`] `}`	
	$\mid$	`return` [e]`;`	retour de fonction

Une vision simplifiée de C: 3. les fonctions

Un programme C est une collection de déclarations de variables globales et de fonctions ($\sim$ procédures).

Ces déclarations sont regroupées en fichiers ("modules") $\langle$nom-de-fichier$\rangle$`.c`, qui seront compilés sous forme de fichiers objet $\langle$nom-de-fichier$\rangle$`.o`.

Chaque module peut référencer une variable ou une fonction d'un autre fichier en utilisant le mot-clé `extern`. Le programme démarre toujours en appelant la fonction `main`.

Sémantique simplifiée de C: 1. expressions

On va décrire la sémantique $[\![e]\!]\,\rho$ des expressions e dans un environnement ρ; ρ est une fonction qui à chaque variable x associe sa valeur.

$$
\begin{array}{rcll}
[\![x]\!]\,\rho & = & \rho(x) & \text{variables} \\
[\![c]\!]\,\rho & = & c & \text{constantes}
\end{array}
$$

Note: les constantes caractères (ex: $\mathtt{'a'}$) représentent leur code ASCII (ex: $\mathtt{'a'} = 97 = \mathtt{0x61}$).

Sémantique simplifiée de C: 1.1. arithmétique

$$
\begin{aligned}
[\![e_1{+}e_2]\!]\,\rho &= [\![e_1]\!]\,\rho + [\![e_2]\!]\,\rho \bmod (maxint+1)\\
[\![e_1{-}e_2]\!]\,\rho &= [\![e_1]\!]\,\rho - [\![e_2]\!]\,\rho \bmod (maxint+1)\\
[\![e_1{*}e_2]\!]\,\rho &= [\![e_1]\!]\,\rho \times [\![e_2]\!]\,\rho \bmod (maxint+1)\\
[\![e_1/e_2]\!]\,\rho &= \lfloor [\![e_1]\!]\,\rho / [\![e_2]\!]\,\rho \rfloor \qquad \text{(quotient)}\\
[\![e_1\%e_2]\!]\,\rho &= \text{reste de la division entière ci-dessus}\\
[\![-e]\!]\,\rho &= -[\![e]\!]\,\rho \bmod (maxint+1)
\end{aligned}
$$

Sémantique simplifiée de C: 1.2. opérations bit à bit

$$
\begin{array}{rcl}
[\![e_1 \& e_2]\!]\,\rho & = & \text{et bit à bit de } [\![e_1]\!]\,\rho \text{ et } [\![e_2]\!]\,\rho \\
[\![e_1 | e_2]\!]\,\rho & = & \text{ou —} \\
[\![e_1 \,\hat{}\, e_2]\!]\,\rho & = & \text{ou exclusif —} \\
[\![\tilde{}\, e]\!]\,\rho & = & \text{non —} \\
[\![e_1 >> e_2]\!]\,\rho & = & [\![e_1]\!]\,\rho / 2^{[\![e_2]\!]\rho} \\
[\![e_1 << e_2]\!]\,\rho & = & [\![e_1]\!]\,\rho * 2^{[\![e_2]\!]\rho} \bmod (maxint + 1)
\end{array}
$$

Sémantique simplifiée de C: 1.3. comparaisons

$$
\begin{aligned}
[\![e_1 \texttt{==} e_2]\!]\,\rho &= \delta([\![e_1]\!]\,\rho = [\![e_2]\!]\,\rho) \\
[\![e_1 \texttt{<} e_2]\!]\,\rho &= \delta([\![e_1]\!]\,\rho < [\![e_2]\!]\,\rho) \\
[\![e_1 \texttt{<=} e_2]\!]\,\rho &= \delta([\![e_1]\!]\,\rho \leq [\![e_2]\!]\,\rho) \\
[\![e_1 \texttt{>} e_2]\!]\,\rho &= \delta([\![e_1]\!]\,\rho > [\![e_2]\!]\,\rho) \\
[\![e_1 \texttt{>=} e_2]\!]\,\rho &= \delta([\![e_1]\!]\,\rho \geq [\![e_2]\!]\,\rho)
\end{aligned}
$$

où $\delta(\text{vrai}) = 1$, $\delta(\text{faux}) = 0$.

Sémantique simplifiée de C: 1.4. logique

$$
\begin{aligned}
[\![e_1 \&\& e_2]\!]\,\rho &= \begin{cases} 0 & \text{si } [\![e_1]\!]\,\rho = 0 \\ [\![e_2]\!]\,\rho & \text{sinon} \end{cases} \\
[\![e_1 || e_2]\!]\,\rho &= \begin{cases} [\![e_1]\!]\,\rho & \text{si non nul} \\ [\![e_2]\!]\,\rho & \text{sinon} \end{cases} \\
[\![!e]\!]\,\rho &= \begin{cases} 1 & \text{si } [\![e]\!]\,\rho = 0 \\ 0 & \text{sinon} \end{cases}
\end{aligned}
$$

Sémantique simplifiée de C: 1.5. pointeurs, tableaux

Idée: `*`e retourne la valeur stockée à l'adresse [qui est la valeur de] e.
Corriger la sémantique! Elle doit prendre un deuxième argument $\sigma : \mathbb{N} \to \mathbb{N}$ représentant la mémoire (comme en assembleur).

$$\begin{aligned} [\![*e]\!]\, \rho\sigma &= \sigma([\![e]\!]\, \rho\sigma) \\ [\![e_1[e_2]]\!]\, \rho\sigma &= \sigma([\![e_1]\!]\, \rho\sigma + N \times [\![e_2]\!]\, \rho\sigma) \end{aligned}$$

où N est la taille d'un élément quelconque du tableau e_1. Par exemple, si e_1 est un `int []` (tableau d'entiers), alors $N =$ `sizeof (int)` $= 4$ (typiquement).

`Note:` en C, une variable tableau (e.g. `int []`) est de type pointeur (e.g. `int *`); e`[0]` est synonyme de `*`e; e_1`[`e_2`]` est synonyme de `*(`e_1`+`e_2`)` [modifier la sémantique de + lorsqu'on additionne un pointeur et un entier!]

Sémantique simplifiée de C: 1.6. opérateur `&`

Slogan: `&`e est l'adresse de e.

On doit avoir: `*(&`e`)` $= e$, `&(*`e`)` $= e$.

Note: si `&x` a un sens, par contre `&3` n'en a pas. Les expressions pour lesquelles il est sensé de calculer leur adresse sont appelées les lvalues.

$$
\begin{array}{rcll}
lval & ::= & x & \text{variables} \\
 & | & \texttt{*}e \quad | \quad e\texttt{[}e\texttt{]} & \text{pointeurs, tableaux}
\end{array}
$$

Note: les affectations e_1=e_2 n'ont de sens que lorsque e_1 est une lvaleur; le résultat est alors de stocker la valeur de e_2 à l'adresse [valeur de] e_1.

Sémantique simplifiée de C: 1.7. appel de fonction

Idée: $e(e_1, \ldots, e_n)$ évalue e, e_1, …, e_n; e doit être un nom de fonction f (ou un pointeur $\star x$, où $x = \texttt{\&} f$), qui est appelée avec paramètres [les valeurs de] e_1, …, e_n; la valeur retournée (par `return`) est celle de $e(e_1, \ldots, e_n)$.

Pb.: la fonction f peut faire des effets de bord (modifications de la mémoire, entrées-sorties, etc.), qui ne sont pas prises en compte par la sémantique.

$\Rightarrow$ compliquer la sémantique des expressions. On utilise des jugements:

$$\sigma \vdash e \Rightarrow V, \sigma'$$

où V est une valeur, σ et σ' des mémoires.
Note: on n'a plus besoin de ρ: pourquoi?
Exercice: réécrire la sémantique des expressions...

Approchons-nous de la vraie sémantique de C...

Tant qu'à avoir des expressions qui ont des effets de bord, les affectations vont être considérées comme des expressions et non des commandes:

la valeur de $lval$=e est celle de e

... et on considère que toute expression peut servir de commande, dont la valeur est ignorée.

Nouvelles abréviations:

$lval$+=e	$\Leftrightarrow$	$lval$=$lval + e$	(sauf que $lval$ n'est évaluée qu'1 fois)
	(pareil pour les autres opérateurs, $-$, $\star$, ..., >>, <<, ...)		
++$lval$	$\Leftrightarrow$	$lval$+=1	
$--lval$	$\Leftrightarrow$	$lval-$=1	
$lval$++	$\Leftrightarrow$	$lval$+=1	(mais retourne l'ancienne val. de $lval$)
$lval--$	$\Leftrightarrow$	$lval-$=1	(mais retourne l'ancienne val. de $lval$)

La sémantique des commandes de C (1)

$$\frac{\sigma \vdash e \Rightarrow V, \sigma'}{\sigma \vdash e\texttt{;} \Rightarrow \sigma'} \qquad\qquad \frac{}{\sigma \vdash \texttt{;} \Rightarrow \sigma}$$

$$\frac{\sigma \vdash c_1 \Rightarrow \sigma_1 \ldots \sigma_{n-1} \vdash c_n \Rightarrow \sigma_n}{\sigma \vdash c_1 \ldots c_n \Rightarrow \sigma_n}$$

$$\frac{\sigma \vdash e \Rightarrow 0, \sigma' \quad \sigma' \vdash c_2 \Rightarrow \sigma''}{\sigma \vdash \texttt{if (}e\texttt{)}\ c_1\ \texttt{else}\ c_2 \Rightarrow \sigma''} \quad \frac{\sigma \vdash e \Rightarrow n, \sigma' \quad \sigma' \vdash c_1 \Rightarrow \sigma''}{\sigma \vdash \texttt{if (}e\texttt{)}\ c_1\ \texttt{else}\ c_2 \Rightarrow \sigma''}$$

où $n \neq 0$.

Note: $\texttt{if (}e\texttt{)}\ c_1 \Leftrightarrow \texttt{if (}e\texttt{)}\ c_1\ \texttt{else;}$.

La sémantique des commandes de C (2)

$$\frac{\sigma \vdash e \Rightarrow 0, \sigma'}{\sigma \vdash \texttt{while (}e\texttt{)}\ \ c \Rightarrow \sigma'} \qquad \frac{\sigma \vdash e \Rightarrow n, \sigma' \quad \sigma' \vdash \texttt{while (}e\texttt{)}\ \ c \Rightarrow \sigma''}{\sigma \vdash \texttt{while (}e\texttt{)}\ \ c \Rightarrow \sigma''}$$

où $n \neq 0$.

Note: $\texttt{do}\ c\ \texttt{while (}e\texttt{)} \Leftrightarrow c\ \ \texttt{while (}e\texttt{)}\ c$

Note: $\texttt{for (}e_1; e_2; e_3\texttt{)}\ \ c \Leftrightarrow e_1; \texttt{while (}e_2\texttt{) \{}\ \ c\ \ e_3\texttt{; \}}$

Exercice: écrire la sémantique du `switch`, en se basant sur celle du `if`.

Retour sur les expressions; évaluation court-circuit

Opérateurs logiques évalués de gauche à droite.

$$\frac{\sigma \vdash e_1 \Rightarrow 0, \sigma'}{\sigma \vdash e_1 \texttt{\&\&} e_2 \Rightarrow 0, \sigma'} \qquad \frac{\sigma \vdash e_1 \Rightarrow n, \sigma' \quad \sigma' \vdash e_2 \Rightarrow n', \sigma''}{\sigma \vdash e_1 \texttt{\&\&} e_2 \Rightarrow n', \sigma''}$$

Note: e_2 n'est même pas évalué si e_1 vaut 0 (règle de gauche).

$\rightarrow$ important notamment lorsque e_2 ne termine pas!

Exercice: écrire la sémantique de `||`, `!`.

Retour sur les expressions: l'indéterminisme

Pour les autres opérations, par ex. +, l'ordre d'évaluation est non spécifié.
En première approximation:

$$\frac{\sigma \vdash e_1 \Rightarrow V_1, \sigma' \quad \sigma' \vdash e_2 \Rightarrow V_2, \sigma''}{\sigma \vdash e_1 \texttt{+} e_2 \Rightarrow V_1 + V_2 \bmod (maxint + 1), \sigma''}$$

$$\frac{\sigma \vdash e_2 \Rightarrow V_2, \sigma' \quad \sigma' \vdash e_1 \Rightarrow V_1, \sigma''}{\sigma \vdash e_1 \texttt{+} e_2 \Rightarrow V_1 + V_2 \bmod (maxint + 1), \sigma''}$$

Exercice: comment s'évalue `i = 3 + (i=4);`? `i = i++`?

Note: en réalité, c'est plus compliqué: C a le droit d'entrelacer les lectures et écritures mémoire de e_1 et e_2 dans un ordre quelconque!

C, informellement

C’est un “assembleur portable”.

Assembleur . . . . . . car il manipule des objets au même niveau d’abstraction.

Portable . . . . . car le même source fonctionne (grosso modo) sur différentes machines.

C ≈ assembleur

`x = y;`	`mov y, x`
`x = y+z;`	`mov z, x; add y, x`
`if (x==y) ...`	`cmp x, y; je ...`
`goto toto;`	`jmp toto`
`x = f ();`	`call f; mov %eax, x`
`return 42;`	`mov $42, %eax; ret`
`x = f (y);`	`push y; call f; add $4,%esp; mov %eax, x`
`x = *p;`	`mov p, %reg; mov (%reg), x`
`*p = x;`	`mov p, %reg; mov x, (%reg)`

C contient aussi des constructions d'un peu plus haut niveau...

La sémantique formelle (?) de C

- Sémantique définie en langage naturel: norme ISO/IEC 9899-1999 ("ANSI C").
- Un gros effort:
  Nikolaos S. Papaspyrou, *Denotational Semantics of ANSI C*, to appear in *Computer Standards and Interface*, 2001. (Regarder particulièrement la section 3.4.) `http://www.cs.yale.edu/homes/nickie/`.
  Version complète: `ftp://ftp.softlab.ntua.gr/pub/users/nickie/papers/thesis.ps.gz`
- Autres références: tapez `"langage C"` sous Google, vous n'aurez que l'embarras du choix! Une accessible est:
  `http://diwww.epfl.ch/w3lsp/teaching/coursC/`