Le langage C

8 janvier 2010

Le langage C

Présentation

- Créé en 1972 dans les laboratoires Bell (AT&T)
- Compromis entre un langage de bas niveau (Assembleur) et un langage de haut niveau (Pascal, Ada, Java, etc)
- Rapide : pas de vérification de gestion de la mémoire, elle est à la charge du développeur !
- Faiblement typé : chaque variable doit être typée, mais certaines opérations entre différents types sont possibles (par ex. on peut comparer un entier à un caractère)

Le langage C

Les bases

- Un **fichier source** C est un fichier texte qui a typiquement l'extension ".c".
- Sous Linux, le compilateur **gcc** s'occupe de traduire le langage C en langage machine. Il produit un fichier exécutable.
  Utilisation : **gcc -Wall source.c -o executable**
  Exécution : **./executable**
- Sous Windows, vous pouvez télécharger gratuitement l'environnement de développement **DevC++**

Les variables

Types de base

Type	Description	Valeur	Codage
char	Caractère	les caractères ASCII	8 bits
int	Entier	2147483648 à 2147483647	32 bits
float	Réel	$3.4 * 10^{-38}$ à $3.4 * 10^{38}$	32 bits

Remarques

- Le type **Booléen** n'existe pas, il est représenté par des entiers : 0 pour faux, un entier différent de zéro pour vrai
- Le type **Chaîne de caractère** n'existe pas, une chaîne est représentée par un tableau de caractères

Les variables

Déclaration d'un type de base

```
int n ;
float x = 5.7 ; // initialisé en même temps
char c ;
```

Déclaration d'un tableau

```
int t[354] ; // un tableau de 354 entiers
float v[3] ; // un vecteur 3D
char s[100] ; // une chaine !
```

Remarque

Toute instruction C se termine par ;

Les variables

Affectation

```
int n ;
float x ;
char c ;
float v[3] ;
char s[100] ;

n = 12 ;
x = 3.12 ;
c = 'a' ;
v[0] = 1.0 ; v[1] = 2.31 ; v[2] = 0.4 ;
strcpy(s,"bonjour") ; // voir section "chaines"
```

Opérateurs

Opérateurs arithmétiques

- Addition :

```
int x = 12 + 8;
```

- Multiplication :

```
int y = 3 * x;
```

- Division :

```
int n = 12 / 5; // Division entière (DIV) !
float x = 12 / 5.0;
```

- Modulo :

```
int p = 12 % 5;
```

Opérateurs et symboles

Tests

- Egalité :
  x == 12 ← Attention : x=12 est une affectation !
  'a' == 'b'
- Comparaison :
  x <= 3
  'b' > 'a'

Opérateurs et symboles

Opérateurs logiques

- et :
  cond1 && cond2
- ou :
  cond1 || cond2
- non :
  !cond

Structures conditionnelles, boucles

Si...Alors...Sinon

```
if (<expression booléenne>) {
  ...
}
else {
  ...
}
```

Exemple

```
if (n<0 || n>100) {
  n = n + 1;
}
else {
  n = n - 1;
}
```

Structures conditionnelles, boucles

Tant que

```
while (<expression booléenne>) {
  ...
}
```

Exemple

```
int i = 0;
float x = 1.0;
while (i<10) {
  x = x / 2;
  i++; // équivaut i = i + 1;
}
```

Structures conditionnelles, boucles

Répète... jusqu'à ce que

```
do {
  ...
} while (<expression booléenne>);
```

Exemple

```
int i = 0;
float x = 1.0;
do {
  x = x / 2;
  i++;
} while (i<10);
```

Structures conditionnelles, boucles

Pour i de 1 à n

```
for (<init> ;<condition> ;<incrémentation>) {
  ...
}
```

Exemple

```
int n = 1 ;
int i ;
for (i=1 ; i<=5 ; i++) {
 n = n * i ;
}
```

Commentaires

Deux types de commentaire

- // : commentaire sur une ligne
- /* : debut de bloc de commentaire
- */ : fin de bloc de commentaire

Exemple

```
int x ; // ceci est un commentaire
/* tout ceci
est aussi un
commentaire */
```

Include

Include

Le mot clé `#include` permet l'inclusion des librairies composées de fonctions utilisables dans votre programme. Les plus connues sont :

- `stdio.h` : fonctions de lecture/écriture
- `string.h` : fonctions de manipulation de chaînes
- `stdlib.h` : fonctions d'allocation mémoire

Lecture - écriture

Lecture

La commande `scanf` permet de lire la saisie d'un ulilisateur. Notez le & devant le nom de la variable. Son existence sera expliquée un peu plus tard. Il faut inclure stdio.h.

Exemple

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

- `scanf("%d", &n); // d pour lire un entier`
- `scanf("%f", &x); // f pour lire un réel`
- `scanf("%c", &c); // c pour lire un caractère`

Lecture - écriture

Ecriture

La commande `printf` permet d'afficher à l'écran. Les éléments %d, %f et %c seront remplacés par les valeurs de variables. Il faut inclure stdio.h.

Exemple

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

- `printf("juste une chaine\n");`
- `printf("%d", n); // d pour un entier`
- `printf("Le carré de %f est : %f", x, x*x); // f pour un réel`
- `printf("%c %d %f", c, n, x); // un caractère, un entier, un réel`

Les chaînes de caractère

Concept

Il n'existe pas de type "string" en C. Une chaîne est représentée par un tableau de caractères (code ASCII). La valeur 0 indique la fin de la chaîne dans le tableau.

```
char str[100] ; // déclare une chaine de 100 caractères max.
```

Les chaînes de caractère

Manipulation de chaînes

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

- Saisie : `scanf("%s", str); // Notez : %s, et pas de & avant str !`
- Affichage : `printf("str = %s", str);`

Les fonctions principales pour manipuler les chaînes sont :
`#include <string.h>`

- Affectation : `strcpy(str, "voici une longue chaine");`
- Affectation d'un caractère : `str[4] = 'o';`
- Extraction d'un caractère : `char c = str[12];`
- Longueur : `int n = strlen(str);`
- Comparaison : `int equal = strcmp(str, "toto"); // == 0 si égalité`
- Concaténation : `strcat(str, "...et elle est encore plus longue");`

Les fonctions

Syntaxe

type retourné *nom de la fonction* (`liste de paramètres`) {
  ...
  return ...
}

Exemple

```
int fact (int n) {
 int f = 1;
 int i;
 for (i=1;i<=n;i++) {
  f = f * i;
 }
 return f;
}
```

N'oublions pas les fonctions récursives

Exemple

```
int fact (int n) {
 if (n == 0) {
  return 1;
 }
 else {
  return n * fact (n-1);
 }
}
```

Procédure

Syntaxe

En C, il n'y a pas de procédure en tant que telle. Pour une fonction qui ne retourne rien, le type est alors remplacé par le mot-clé **void**.

```
void nomFonction(type1 arg1, type2 arg2, ...) {
  ...
}
```

Exemple

```
void afficher (int n) {
 int i;
 for (i=1;i<=n;i++) {
  printf("%d\n", i);
 }
}
```

Procédure

Programme

La méthode initiale s'appelle "main". Comme en pseudo-code elle est (dans un premier temps) sans paramètre. Par contre, il ne s'agit pas d'une procédure mais d'une fonction qui renvoie au système d'exploitation un entier indiquant le type d'arrêt du programme. 0 signifie "terminaison correcte sans incident".

```
int main() {
  ...
  return 0;
}
```

Exemple

```
int main () {
 printf("Hello World");
 return 0;
}
```

Prototypage

Problème

Le compilateur C lit les fichiers source du haut en bas. Il constatera donc une erreur si une méthode est déclarée APRES sa première utilisation. L'exemple suivant provoque une erreur de compilation.

Exemple avec erreur

```
#include <stdio.h>

int main () {
 printf("%d",toto(3));
 return 0;
}

int toto (int n) {
 return 2 * n;
}
```

Prototypage

1e solution

La première solution (à déconseiller) consiste à définir toutes les méthodes dans l'ordre de leur utilisation.

Exemple

```
#include <stdio.h>

int toto (int n) {
 return 2 * n;
}

int main () {
 printf("%d",toto(3));
 return 0;
}
```

Prototypage

2e solution

La deuxième solution (à préférer) consiste à "prototyper" les méthodes au début du fichier source. Pour ce faire, on indique leurs en-têtes suivies d'un point-virgule.

Exemple

```
#include <stdio.h>

int toto (int n); // prototypage !

int main () {
 printf("%d",toto(3));
 return 0;
}

int toto (int n) {
 return 2 * n;
}
```