smaky info

Le langage C

Introduction

C est un langage universel de bas niveau. Cela ne veut pas dire que c'est un mauvais langage, mais plutôt qu'il permet de réaliser pratiquement tout comme en assembleur. En programmant en C on est très proche de la machine sans pour autant avoir à s'occuper des ses particularités. L'autre avantage du C par rapport à l'assembleur réside dans le fait que des programmes écrits en C sont relativement facilement portables d'un système d'exploitation à un autre, même si les deux systèmes sont basés sur des processeurs différents.

Types de données

C dispose de tous les types habituels de données:

- entiers 8,16,32 et 64 bits
- caractères
- nombres réels 32 et 64 bits
- chaînes de caractères
- tableaux
- pointeurs

Le type booléen n'existe pas, mais il est remplacé par le type entier.

Il est également possible de déclarer ses propres structures de données.

Premier exemple

Le petit programme qui suit effectue un calcul élémentaire sans rien afficher:

```
main()
{
  int a,b,c;
  a=1; b=2;
  c=a+b;
}
```

`main()` est le nom de la fonction principale constituant le programme; `int a,b,c` déclare les trois variables a, b et c comme variables entières 32 bits; le reste du programme se passe d'explications.

Le programme équivalent en PASCAL est:

```
PROGRAM main;
VAR a,b,c : Integer;

BEGIN
  a:=1; b:=2;
  c:=a+b;
END.
```

On remarque tout de suite qu'il y a une certaine ressemblance, mais également que l'écriture d'un programme C est plus condensée (le signe { équivaut à BEGIN, le signe } équivaut à END).

Initialisation automatique

En C, contrairement au PASCAL, une variable peut être déclarée tout en lui attribuant une valeur initiale. Ceci permet d'économiser le code nécessaire à l'initialisation dans le programme. Donc au lieu d'écrire:

```
int a ;
a=5 ;
```

on peut écrire simplement

```
int a=5;
```

Cette possibilité est particulièrement utile dans le cas d'un tableau qui doit être initialisé avec des valeurs arbitraires.

Eléments du langage

En C il existe des instructions de test, de bouclage et de discrimination comme en PASCAL. La boucle `for` est cependant beaucoup plus riche qu'en PASCAL et vaut la peine d'être mentionné. La syntaxe est la suivante:

```
for (init;test;pas) instruction
```

`init`, `test` et `pas` sont des expressions. `init` est effectué une fois au début, `pas` est effectué après `instruction` et `instruction` est effectué tant que l'expression `test` est vraie. Par exemple la boucle:

```
for (i=1;i<=10;i++) instruction
```

est équivalent à

```
FOR i:=1 TO 10 DO instruction
```

en PASCAL. Remarquez que `i++` signifie "incrémente la valeur de i de 1".

Deux instructions intéressantes qui n'existent pas en PASCAL sont `break` et `continue`. `break` termine la boucle prématurément et `continue` passe prématurément au tour de boucle suivant. En C il existe même le fameux `goto` qui est à éviter dans la mesure du possible.

Expressions

En C il n'y pas d'énoncé d'assignation proprement dit comme en PASCAL. Une assignation en C est en fait une expression qui a la valeur de la variable assignée. Puisque les variables booléennes en C sont représentés par le type `int` (0 = faux, autre chose que zéro = vrai): il est alors possible d'écrire des instructiuons telles que:

```
if (a=b) {
   ...
};
```

Si `b` vaut 0 l'expression `a=b` vaut également zéro donc faux et les instructions entre les crochets ne seront pas exécutées.

En PASCAL on est obligé d'écrire:

```
a:=b;
IF a=b THEN BEGIN
   ...
END;
```

On a vu un peu plus haut la construction `i++`. En fait, c'est également une expression. Pour parcourir un tableau jusqu'à ce que la valeur zéro soit atteinte peut donc s'écrire:

```
i=0;
while (tabl[i++]) ;
```

L'équivalent en PASCAL:

```
i:=0 ;
WHILE (tabl[i]<>0) DO i:=i+1 ;
```

Opérateurs

En C il existe un grand nombre d'opérateurs. Les opérateurs relationnels sont les mêmes qu'en PASCAL sauf que '`<>`' s'écrit '`!=`' en C et '`=`' s'écrit '`==`' ('`=`' est l'opérateur d'assignation).

Les opérateurs logiques AND, OR et NOT s'écrivent en C respectivement '`&&`', '`||`' et '`!`'. `&&` et `||` jouissent d'une propriété intéressante: la deuxième opérande n'est évaluée que si la connaissance de sa valeur est indispensable pour décider si l'expression correspondante est vraie ou fausse. Cette propriété est particulièrement intéressante si la deuxième opérande fait appel a une fonction qui effectue une certaine action. Par exemple:

```
if  (i<10 && getval(i)==999) ...
```

la fonction `getval` rend une valeur en fonction de i où il doit être inférieur à 10 sinon le programme plante. Si i est plus grand que 10 l'expression `i<10` est fausse et la deuxième opérande (getval(i)==999) n'est plus effectuée.

Les opérateurs d'incrémentation et de décrémentation `++` et `--` permettent d'incrémenter et de décrémenter de 1 la valeur d'une variable. On peut mettre ces opérateur avant ou après la variable.

L'expression `++i` a pour effet d'incrémenter la variable i et sa valeur est celle de i après l'incrémentation. L'expression `i++` a le même effet mais sa valeur est celle de i avant l'incrémentation.

Les opérateurs d'affectation élargies permettent

de remplacer des expressions telles que:

```
i = i + k ;
i = i * k ;
etc.
```

respectivement par:

```
i += k ;
i *= k ;
etc.
```

L'opérateur le plus curieux est certainement l'opérateur conditionnel. Il permet de remplacer:

```
if (a>b) max=a; else max=b;
```

par

```
max = a>b ? a : b ;
```

La syntaxe générale de cet opérateur est:

```
expr1 ? expr2 : expr3
```

la valeur de cette expression est celle de expr2 si expr1 est vraie sinon c'est celle de expr3.

Fonctions

Le C ne connaît pas de procédures comme le PASCAL mais uniquement des fonctions. La fonction suivante rend la somme de ses deux arguments:

```
int somme(int a, int b)
{
  return a+b ;
};
```

L'appel de la fonction s'effectue de la même manière qu'en PASCAL par exemple:

```
c=somme(3,4) ;
```

Le mot réservé `return` quitte immédiatement la fonction qui prendra comme valeur la valeur de l'expression suivant le mot `return`. La valeur de retour de la fonction peut cependant être ignorée en écrivant simplement le nom de la fonction par exemple:

```
somme(5,6) ;
```

ce qui n'a évidemment pas beaucoup de sens ici.

Le préprocesseur C

Le langage C est un langage compilé, mais avant la compilation proprement dite, le programme source est passé à travers un préprocesseur qui permet les facilités suivantes:

- inclusion de fichiers d'en-tête
- macros avec paramètres
- compilation conditionnelle

C'est similaire aux directives de compilation du PASCAL $I, $DEFINE, $IFDEF etc. mais bien plus souple.

Compilation séparée

Le C supporte la compilation séparée, c-à-d. qu'un un programme peut être découpé en plusieurs modules de taille raisonnable, puis compilé séparément. Le tout est ensuite lié ensemble pour former un fichier .CODE exécutable. Le C convient donc bien à l'écriture d'applications importantes.

Utilisation du C

Pour compiler un programme C il faut lancer le compilateur à l'aide du programme CC. Ce programme doit être lancé depuis le FILER. Pour compiler le programme monprog.c en monprog.code il faut taper la commande

```
cc monprog.c -o monprog.code
```

en tapant le tout en minuscules.

Le programme MAKE

Le but de ce logiciel est de déterminer automatiquement quelles parties d'un gros projet doivent être recompilées et de lancer les commandes appropriées.

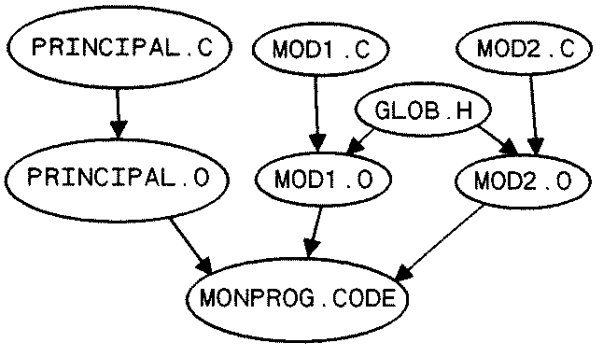
Un fichier nommé MAKEFILE contient la description des relations existant entre les différents fichiers d'un projet. Il suffit alors de lancer la commande make depuis le FILER et toutes les commandes pour compiler le projet seront lancées automatiquement.

La structure d'un projet C est généralement la suivante: il y a plusieurs modules dont les fichiers source insèrent un ou plusieurs fichiers d'en-tête contenant des définitions communes aux différents modules. Chacun des fichiers source aboutit à un fichier objet ayant l'extension ".o". Le fichier .CODE final résulte de la fusion des fichiers .o.

Exemple:

Le logiciel monprog.code est composé des trois fichiers principal.c, mod1.c et mod2.c.

mod1.c et mod2.c insèrent le fichier d'en-tête glob.h.



Donc monprog.code dépend de principal.o, mod1.o et mod2.o, principal.o dépend de principal.c etc.

Dans le fichier makefile on décrit les dépendences de chaque fichier ainsi que la commande permettant de générer le fichier. Le makefile pour le projet ci-dessus est alors le suivant:

```
objets = principal.o mod1.o mod2.o
monprog.code : $(objets)
        cc -o monprog.code $(objets)
mod1.o : mod1.c global.h
        cc -c mod1.c
mod2.o : mod2.c global.h
        cc -c mod2.c
principal.o: principal.c
        cc -c principal.c
```

Explication:

La première ligne assigne à la variable objets la ligne qui suit le signe '='. Par la suite $(objets) signifie le contenu de la variable objets. Pour fabriquer le fichier monprog.code il faut lancer la commande:

```
cc -o monprog.code principal.o mod1.o mod2.o
```

Pour fabriquer le fichier mod1.o il faut lancer la ligne de commande:

```
cc -c mod1.c
```

etc.

A l'aide de la description dans le fichier makefile et des dates de création des différents fichiers, make est capable de ne recompiler que les fichiers qui ont vraiment besoin d'être compilés parce que un ou plusieurs fichiers dont ils dépendent ont changés.

Le langage C++

En plus du langage C il existe également le langage C++, sur-ensemble de C permettant d'écrire des programmes orientés objet.s

Utilitaires supplémentaires

La distribution de l'environnement C comporte en outre un certain nombre d'utilitaires fort pratiques:

GDB

Un dévermineur symbolique très puissant (runtime et post-mortem).

BISON

Génère un programme C qui applique une grammaire à un fichier, à partir d'une déscription de cette grammaire.

GPERF

Génère une fonction de hachage parfaite à partir d'une liste de mots clés.

Remarque: le compilateur C a été écrit à l'aide de GPERF et BISON.

FLEX

Génère un analyseur lexical en C à partir d'une description de grammaire régulière.

DIFF, DIFF3

Montre les différences entre deux ou trois fichiers.

GREP, EGREP, FGREP

Recherche de chaînes de caractères dans un ensemble de fichiers.

EC

Editeur C facilitant la recherche des erreurs de compilation et l'écriture du fichier makefile.