Projet de traduction
L3 Informatique – 2008–09
Typage des expressions en C ANSI

19 février 2009

1 Sujet

Le but du projet est d'étudier les expressions et les types en C.

Voici un premier exemple d'entrée du projet :

```
char c;
type c*2+2.0
```

Le projet affiche le type du résultat de l'expression (double) et affiche chaque promotion entière et conversion implicite nécessaire au calcul de l'expression : c doit être promu en int, puis le résultat de $c * 2$, qui est typé int, doit être converti en double avant d'en faire la somme avec 2.0. En plus d'exhiber ces conversions et promotions, le programme devra indiquer les pertes possibles dans l'évaluation de l'expression, et les erreurs de typage s'il y en a. Tout résultat machine dépendant, que ce soit au niveau du typage ou de la valeur de l'expression, devra être signalé. En plus du signalement, on demande que le résultat soit donné par le programme en fonction des caractéristiques locales de la machine sur laquelle il est exécuté. Par exemple, dans l'expression $(-1)\%4$, le résultat est de type int, car les deux opérandes de % sont typés int, mais la valeur du résultat est machine dépendant, car un des deux opérandes est négatif. Comme autre exemple, supposons qu'on ait une expression qui additionne un int a et un unsigned int b. Le type et la valeur du résultat sont tous les deux machine-dépendants, la norme ANSI spécifiant le comportement suivant :

- si le type long int contient toutes les valeurs typées unsigned int, alors le résultat est de type long int : a est converti en long int, b aussi, on fait la somme qui est un long int. On remarque que dans ce cas il n'y a pas d'erreur possible due aux convertions appliquées ;
- sinon, a et b sont convertis en unsigned long int : cette convertion n'a pas d'incidence sur la valeur de b, mais elle en a une très importante sur la valeur de a si elle est négative (aucune si la valeur de a est positive ou nulle). En effet, cela revient à convertir une valeur négative en valeur positive, ce qui va rendre le résultat faux ! La somme de deux unsigned long int étant aussi un unsigned long int, le résultat (potentiellement faux si la valeur de a est négative) est de type unsigned long int.

Le projet devra respecter la syntaxe des types et expressions du C ANSI, tels que définis dans le livre "Le langage C (norme ANSI)" de Kernighan et Ritchie. Il doit en particulier fonctionner pour tout types et expressions correctes en C ANSI. Pour simplifier on ne tiendra pas compte des modificateurs const, static, volatile, etc. On supposera que typedef ne fait pas partie du langage. Le programme lira son entrée sur l'entrée standard, et produira sa sortie sur la sortie standard.

Voici un dernier exemple d'entrée :

```
double a;
int i;
struct nom {int a; char b;}
```

```
struct nom (*f)(void);
short *p;

type a=i+5.0-(unsigned long)3+(*f)().b++*p
```

Attention : le projet ne doit pas évaluer les expressions, mais juste leurs types et indiquer les opérations de promotion et conversion, ainsi que les problèmes qui peuvent apparaitre au moment de l'évaluation des expressions.

2 Ce qu'il vous est demandé

Le projet devra être écrit en C ANSI, compilé avec -ansi -Wall –pedantic sans aucun warning, avec un Makefile. Les parties d'analyses lexicales et syntaxiques devront être écrites avec respectivement flex et bison.

Il devra être développé par binômes (2 personnes) au plus. Il devra être rendu au plus tard A FIXER, sous la forme d'un mail envoyé au chargé de TD, avec en pièce jointe une archive du projet au format tar.gz contenant, dans des repertoires séparés, les sources, les exemples, les documentations, etc. Le sujet du mail devra être "Projet traduction L3", le nom du fichier d'archive sera Nom1-Nom2.tar.gz, où Nom1 et Nom2 sont les noms des auteurs. L'extraction de l'archive devra produire un répertoire Nom1-Nom2 contenant le projet.

Un soin tout particulier sera apporté à la lisibilité du code (découpage en fonctions, programmation modulaire, ...) qui devra être judicieusement commenté. Le projet sera rendu avec un rapport, dans lequel seront en particulier exposés les problèmes, leurs solutions, les algorithmes utilisés, et l'architecture logicielle choisie. Le rapport contiendra également quelques exemples détaillés.

Il est particulièrement recommandé d'apporter beaucoup de soin à la conception du projet : une architecture bien faite et un découpage clair en fonctions, avec une utilisation judicieuse des types et des commentaires rendent les programmes plus faciles à relire et donc les erreurs plus faciles à repérer, donc à éviter.

Vous pourrez, si vous le souhaitez, implanter des options qui sont laissées à votre choix. Attention, il est préférable de faire le minimum de ce qui est demandé, et de bien le faire, plutôt que de réaliser un projet plein d'options marchant mal et mal conçu. Si vous vous sentez fort, et avez confiance en vous, vous pourrez par exemple réflechir à l'ajout de la prise en compte des typedef. Quelles sont les difficultés rencontrées alors dans l'analyse syntaxique de l'entrée ? Comme base de reflexion, dans l'exemple suivant

```
...
int main(void) { t (a); }
```

Qu'est-ce que `t (a);` ?