

Projet de Compilation

Réalisation d'un compilateur

Le but du projet est de construire un compilateur d'un langage de programmation évolué dans un langage intermédiaire, style assembleur à trois adresses. On demande aussi d'écrire un interpréteur de cet assembleur qui donnera un résultat exécutable.

1 Le langage intermédiaire

Les registres

On pourra utiliser les registres spécialisés suivants :

- un compteur de programme `InstructionCourante`, pointant sur l'instruction en cours,
- un compteur `InstructionSuivante` pointant sur la prochaine instruction à effectuer,
- un registre `SommetPile` pointant sur le sommet de la pile,
- un registre `Base` qui contient l'adresse de la base de l'enregistrement d'activation courant,
- un registre d'index `Index`.

Les instructions et les modes d'adressage

Les modes d'adressage sont les suivants :

Nom	Syntaxe	Signification
Immédiat	<code>#A</code>	A est interprété littéralement.
Direct	<code>A</code>	A est l'adresse de l'opérande.
Indirect	<code>@A</code>	A contient l'adresse de l'opérande.
Indexé	<code>Index(A)</code>	l'adresse de l'opérande s'obtient en ajoutant le déplacement A au contenu du registre d'index.

Les instructions sont données dans le tableau qui suit. Ce sont des instructions à une, deux, ou trois adresses. La signification est donnée de façon informelle : on ne tient pas compte des modes d'adressage employés dans la colonne ; l'Opération réalisée ; A, B et C sont le plus souvent interprétés comme s'ils étaient donnés en mode direct. La signification réelle doit bien entendu tenir compte des modes d'adressage. Certains modes sont évidemment interdits ; par exemple, une destination n'est jamais donnée en adressage immédiat.

Les accents ne sont mis ici que pour la lisibilité ; on pourra ne pas s'en préoccuper.

Nom	Arguments	Opération réalisée
LIRE	A	Lit une valeur sur l'entrée standard et la charge dans A
ÉCRIRE	A	Écrit A sur la sortie standard
TRANSFÉRER	A B	Transfère A dans B
EMPILER	A	Empile A
DÉPILER	A	Dépile A
ÉTIQUETTE	A	Positionne l'étiquette A
INCRÉMENTER	A	Incrémente A
DECRÉMENTER	A	Décrémente A
ADDITIONNER	ABC	C reçoit $A+B$
SOUSTRAIRE	ABC	C reçoit $A-B$
MULTIPLIER	ABC	C reçoit $A*B$
DIVISER	ABC	C reçoit le quotient de la division entière de A par B
RESTE	ABC	C reçoit le reste de la division entière de A par B
CHANGT_SIGNE	AB	B reçoit l'opposé de A
ET	ABC	C reçoit le ii et ll logique, bit à bit, de A et B
OU	ABC	C reçoit le ii ou ll logique, bit à bit, de A et B
NON	AB	B reçoit le ii non ll logique, bit à bit, de A
TEST_SL_INFÉRIEUR	ABC	C reçoit 1 ou 0 suivant que A est inférieur ou non à B
TEST_SL_NUL	AB	B reçoit 1 ou 0 suivant que A est nul ou non
BRANCHEMENT	A	Branchement à l'adresse (étiquette) spécifiée par A
BRANCHEMENT_SL_VRAI	AB	Branchement à l'adresse spécifiée par A si le booléen désigné par B est vrai
APPELER	A	Appelle la fonction désignée par A
RETOURNER		Retour d'appel
ARRÊTER		Arrêt
PAS_D_OPÉRATION		Ne fait rien

2 Le langage à compiler

Éléments lexicaux

Les commentaires sont compris entre `/*` et `*/` et ne s'imbriquent pas. Les blancs sont interdits au milieu d'un mot clé, ignorés ailleurs. Un `identificateur` est une suite de lettres, et une constante entière `nb_entier` est une suite de chiffres. Le lexème virgule `,` joue le rôle de séparateur d'identificateurs. Le point-virgule `;` joue le rôle de terminateur d'instruction. Les opérateurs relationnels sont `==`, `!=`, `<=`, `>=`, `<`, `>`. Les opérateurs logiques sont `||` (ou), `&&` (et), et `!` (non). L'affectation est notée `=`. Les opérateurs arithmétiques sont `+`, `-`, `*`, `%` et `/`. Le moins unaire est aussi noté `-`. Parenthèses `()` et crochets `[]` sont des lexèmes utilisés comme en C.

Syntaxe

La syntaxe du langage est décrite par la grammaire suivante, où les terminaux sont indiqués en *fonte courrier* et les non terminaux *<entre crochets>*.

<code><programme></code>	→	<code>début <corps> fin</code>
<code><corps></code>	→	<code><liste de déclarations> <liste de fonctions> <liste d'instructions></code>
<code><liste de déclarations></code>	→	<code>ε <déclaration> ; <liste de déclarations></code>
<code><déclaration></code>	→	<code>entier identificateur entier statique identificateur tableau entier identificateur[nb_entier] tableau statique entier identificateur[nb_entier]</code>
<code><liste de fonctions></code>	→	<code>ε <fonction> ; <liste de fonctions></code>
<code><fonction></code>	→	<code><en-tête> début <corps> fin</code>
<code><en-tête></code>	→	<code>fonction <type> identificateur(<suite de paramètres>)</code>
<code><type></code>	→	<code>vide entier</code>
<code><suite de paramètres></code>	→	<code>ε <liste de paramètres></code>
<code><liste de paramètres></code>	→	<code><paramètre> <paramètre> , <liste de paramètres></code>
<code><paramètre></code>	→	<code>entier identificateur tableau entier identificateur[nb_entier]</code>
<code><liste d'instructions></code>	→	<code>ε <instruction> ; <liste d'instructions></code>
<code><instruction></code>	→	<code>identificateur = <expression> identificateur[<expression simple>] = <expression> retourner <expression> arrêt retour début <liste d'instructions> fin si <expression> alors <instruction> sinon <instruction> si <expression> alors <instruction> tant que <expression> faire <instruction> écrire <liste d'arguments> lire identificateur</code>
<code><expression></code>	→	<code><expression> <comparaison> <expression simple> <expression simple></code>
<code><expression simple></code>	→	<code><expression simple> + <terme> <expression simple> - <terme> <expression simple> <terme> <terme> - <terme></code>
<code><terme></code>	→	<code><terme> * <facteur> <terme> / <facteur> <terme> % <facteur> <terme> && <facteur> <facteur></code>
<code><facteur></code>	→	<code>identificateur nb_entier identificateur(<suite d'arguments>) (<expression>) identificateur[<expression simple>] ! <facteur></code>
<code><suite d'arguments></code>	→	<code>ε <liste d'arguments></code>
<code><liste d'arguments></code>	→	<code><expression> <expression> , <liste d'arguments></code>
<code><comparaison></code>	→	<code>< > == <= >= !=</code>

Sémantique

Le seul type de base est le type entier, et le seul type nouveau que l'on peut construire sont les tableaux d'entiers. Il n'y a pas de procédures, mais il y a des fonctions, qui peuvent être appelées récursivement, les paramètres étant passés par valeur. On remarquera par ailleurs que les tableaux peuvent être passés en argument.

La sémantique des différents opérateurs, ainsi que leur associativité et leur priorité est la même qu'en C. Par contre, il y a comme en PASCAL une notion de fonction imbriquées. Les variables sont allouées dynamiquement sauf si elles sont déclarées statiques.

3 Travail demandé

Le travail demandé peut être effectué par groupes de trois personnes. Il consiste en la réalisation d'un compilateur du langage source dans le langage intermédiaire. Vous devez également écrire un interpréteur du langage intermédiaire. Il n'est pas demandé que l'interpréteur reconnaisse tous les modes d'adressage ni toutes les instructions, mais il doit évidemment être compatible avec le code produit par le compilateur.

L'utilisation de `lex` et `yacc` est permise (mais non imposée).

Des jeux d'essais convaincants devront être joints à un rapport. Ce rapport doit contenir une présentation générale du compilateur et de l'interpréteur. Il doit exposer de façon synthétique les choix faits au cours de la réalisation en s'appuyant sur des exemples simples mais significatifs.

Il est conseillé de commencer par un compilateur traduisant un sous-langage du langage source. Néanmoins, il est aussi conseillé de réfléchir dès le départ à l'ensemble des tâches qu'il faudra effectuer, et de prévoir en conséquence les possibilités d'extensions qui seront nécessaires. Tout n'est pas entièrement dit dans le sujet, et il n'est pas interdit de faire preuve d'imagination.

On pourra enfin penser à faire quelques extensions, comme par exemple introduire un type booléen afin de faire des vérifications de type. On pourra aussi implémenter le passage de paramètres par adresse. Une autre extension possible est l'introduction de la récursivité croisée. Ces extensions ne doivent être réalisées qu'après la réalisation du compilateur demandé.