Compilateur StenC vers MIPS32
Rapport de projet

Table des matières

I/ Rappels des objectifs du projet

II/ Développement du compilateur

III/ Spécification complète du compilateur

Précisions supplémentaires :

IV/ Critiques et améliorations

V/ Manuel d'utilisation

BALZAN-WONG Juan Andres (M1 ISI)
PERRIN Romain (M1 ISI)

I/ Rappels des objectifs du projet

L'objectif de ce projet était de développer un compilateur capable de réaliser une traduction depuis un code écrit dans un langage appelé **StenC** vers un code écrit dans le langage assembleur **MIPS32**.
Le StenC est en réalité un sous-ensemble du langage C auquel on a ajouté un nouveau type ***stencil*** et un nouvel opérateur d'application de stencil ***$***.

II/ Développement du compilateur

Le développement du compilateur s'est fait de manière incrémentale en partant d'un petite grammaire sans action et en y ajoutant au fur et à mesure des règles de plus en plus complexes et des actions permettant de générer du code assembleur.

La première étape peut être considérée comme une étape d'exploration des outils d'analyse lexicale/syntaxique. Elle consistait à écrire un lexeur définissant tous les tokens nécessaires à la compilation du StenC ainsi que d'écrire une grammaire capable de reconnaître le langage StenC sans pour autant produire le moindre code.

La deuxième étape est l'étape de génération de code intermédiaire (sous forme de code à 3 adresses). En effet, l'objectif de cette seconde étape était d'écrire la grammaire des expressions arithmétiques et de produire une liste de symboles et une liste de quads servant à représenter les codes 3 adresses. Aucun code assembleur n'a été produit durant cette étape néanmoins toutes les structures permettant la génération de code étaient prêtes. La table des symboles et la liste de quads ont été mises en place.

La troisième étape est l'étape de génération de code assembleur. Durant cette étape, nous avons écrit la grammaire des expressions booléennes ainsi que la grammaire des structures de contrôles (les deux fonctionnant ensembles).
Une structure mips_generator permettant de traduire la liste des quads en du code assembleur MIPS32. Le programme est devenu un compilateur à proprement parler.

La dernière étape a vu l'introduction des tableaux d'entiers multidimensionnels ainsi que des références de tableaux.
Nous trouvions plus simple de prendre toute la chaîne de caractères contenant la référence de tableau tab[indice 1]...[indice n] et de réaliser nos propres structures et fonctions de parsing afin d'extraire diverses informations de a

chaîne (nom de l'identifiant de tableau, nombre de dimensions, taille de chaque dimension, indice de chaque dimension, taille des données, données…).

La dernière étape aurait été l'ajout des stencils et de l'opérateur d'application des stencils mais le temps restant était insuffisant.

III/ Spécification complète du compilateur

Cette section décrit ce que notre compilateur est capable de traiter.

<table>
<tr><th colspan="3">Expression artithmétiques</th></tr>
<tr><th>Opérateur</th><th>Support</th><th>Commentaire</th></tr>
<tr><td>+</td><td>x</td><td rowspan="8">Support de ces opérateurs sur les entiers ainsi que sur les variables d'entiers.</td></tr>
<tr><td>- (unaire)</td><td>x</td></tr>
<tr><td>- (binaire)</td><td>x</td></tr>
<tr><td>*</td><td>x</td></tr>
<tr><td>/</td><td>x</td></tr>
<tr><td>++</td><td>x</td></tr>
<tr><td>--</td><td>x</td></tr>
<tr><td colspan="3">Conditions booléennes</td></tr>
<tr><th>Opérateur</th><th>Support</th><th>Commentaire</th></tr>
<tr><td>&&</td><td>x</td><td rowspan="9">Support de ces opérateurs seulement sur les identificateurs

a < b && !(c >= d) est accepté
a < 12 && !(c >= 42) ne l'est pas

Il faut donc déclarer et affecter toutes les variables formant une condition avant la condition elle-même</td></tr>
<tr><td>||</td><td>x</td></tr>
<tr><td>!</td><td>x</td></tr>
<tr><td>==</td><td>x</td></tr>
<tr><td>!=</td><td>x</td></tr>
<tr><td>!</td><td>x</td></tr>
<tr><td><</td><td>x</td></tr>
<tr><td>></td><td>x</td></tr>
</table>

<=	x	
>=	x	
Structures de contrôle		
Structure	**Support**	**Commentaire**

if (condition)
{
instructions
}
	x	condition représente une condition booléenne (section précédente)
		
		Les accolades sont nécessaires même s'il n'y a qu'une seule instruction.
		
		Les instructions peuvent elles-mêmes être des structures de contrôle.

if (condition)
{
instructions
}
	x	

while (condition)
{
instructions
}
	x	

for (init; cond; iterator)
{
condition
}
	x	init représente une affectation (l'identificateur ne peut être déclaré dans le for)
		condition est une condition booléenne
		iterator est soit i++, i--, ++i ou --i
		++i ou --i sera incrémenté avant le bloc et i++ et i-- le sera après

Tableaux multidimensionnels et références		
Instruction	**Support**	**Commentaire**
int tab[] = {...};	x	Supporte à la fois des indices sous forme d'entiers et sous forme d'identificateurs
tab[i1]...[in] = id;	x	
tab[i1]...[in] = 42;	x	
id = tab[i1]...[in];	x	
tab[i1]...[in] = tab [i1]...[im];	x	

Stencils et opérateurs		
Instruction	**Support**	**Commentaire**
stencil s{1,2} = {...};		non implémenté faute de temps
id $ s;		
Fonctions et fonctions récursives		
Instruction	**Support**	**Commentaire**
printf(“chaine”);	x	la chaîne doit être donnée directement entre les parenthèses (par de de string)
printi(nb);	x	l’argument peut être soit un identificateur soit un entier
printi(42);	x	

int f(arg1, …, argn)
{
instructions
return val;
}
		non implémenté faute de temps

int a = f(arg1,..., argn);		

Précisions supplémentaires

Notre compilateur est capable de réaliser un certain degré d’analyse sémantique. Il est capable de détecter qu’une variable est utilisée sans avoir été déclarée (niveau 1) ou initialisée (niveau 2). Il est également capable de distinguer les différents types représentés.

Si l’on tente d’affecter un tableau à un entier il le détectera. Enfin il est capable d’analyser les dimensions des tableaux. Autrement dit si l’on tente d'accéder à un tableau tridimensionnel en indiquant que deux dimensions il détectera que le tableau en question est tridimensionnels et que l’on ne peut accéder qu’à une seule case contenant un entier et non à des pointeurs.

ex :
int tab[] = {{{1,2},{3,4}},{{5,6},{7,8}}};
int a = tab[0][1];
Provoquera le message suivant :

“semantic error : cannot access a 2-dimensionnal reference when the array is 3-dimensionnal”

Le comportement par défaut lorsqu’une erreur est détectée (qu’elle soit lexicale, syntaxique ou bien sémantique) est de stopper l’exécution du compilateur et d'informer l’utilisateur sur la nature de l’erreur.

Il n’est pas possible de définir des variables locales à une structure de contrôle. Par exemple for (int i = 0…) ne fonctionnera pas alors que int i; for (i = 0…) est accepté.

IV/ Critiques et améliorations

Evidemment nous ne sommes pas arrivés au bout du projet.

Néanmoins dans la mesure où le support des tableaux multidimensionnels et des références de tableaux est assuré, le travail restant pour intégrer les stencil n’est plus très conséquent. Les stencils n’étant finalement que des tableau auxquels on ajoute deux informations supplémentaires, ils sont simples à conserver dans une structure (tout comme nous avons fait pour les tableaux et références mais en plus simple).

Les fonctions ne sont pas non plus supportées mais là il reste beaucoup de travail afin de gérer la portée des variables. Dans notre compilateur toutes les variables sont statiques mais globales (au main).
Il faudrait définir des piles de tables des symboles pour pouvoir gérer correctement la portée de chaque variable.

La manière dont a été définit le return du main pose quelques problèmes. En effet, puisque le return est reconnu comme étant
return_statement -> RETURN expression
théoriquement rien n’empêche d’écrire
return if (a < b) { b = c; } else { a++; }.

Il y a beaucoup d’allocations de mémoire (variables temporaires, structures de gestion interne pour les tableaux et références) mais pas autant de libérations donc des fuites mémoires non résolues résident encore dans le programme. Les éventuels segfaults qui restaient ont normalement été corrigés.

V/ Manuel d'utilisation

Le code est disponible à l'adresse suivante :
https://github.com/Romain96/M1S1_Compilation
(le dépôt est privé mais sera passé public lors du rendu du projet)

Le compilateur est disponible avec plusieurs options.
Le minimum attendu est *./stenc -i fichier.c -o fichier.s*
fichier.c doit contenir a minima une fonction

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

Il y a 4 options :

- -s : permet d'afficher la table des symboles en fin d'analyse
- -q : permet d'afficher la liste de quads en fin d'analyse
- -v : active le mode verbeux et affiche les règles de grammaire traversées
- -m : active le mode verbeux lors de la génération de l'assembleur

Par défaut rien n'est affiché (sauf en cas d'erreur).