Compiladores

Trabalho 2015/2016

18/06/16

UNIVERSIDADE DE ÉVORA

YA!

Language

Docente:
Pedro Patinho

Realizado por:
João Calhau - 31621
José Pimenta - 31677

Índice

Introdução

No âmbito da unidade curricular de Compiladores pretende-se, através das ferramentas e métodos abordados ao longo das aulas e Compiladores implementar um compilador de YA! composto pelo analisador lexical (flex), sintático (bison) e semanântico e geradore de código MIPS. O programa irá então receber 1 ficheiro com código em YA! e gerar o código correspondente em MIPS.
Iremos assim tentar dar o nosso melhor, e iremos tentar utilizar os métodos que achemos mais correctos ou propícios à boa evolução do trabalho e que no final se concretize o que nos é pedido.

Análise Lexical

Na análise lexical estão definidos todos os tipos que se podem utilizar no trababalho (int, float, id, string, bool), e palavras reservadas (ex: mod, return, void, if, else,...), e ainda os literais que se podem utilizar (BOOL_LIT, INT_LIT, FLOAT_LIT, STRING_LIT).
Na análise lexical também são ignorados todos os espaços, tabs e new lines, isto porque por exemplo, um if pode ter o corpo definido na mesma linha, ou ter por várias, tal como se verifica num dos exemplos de código YA dado pelo professor.

Análise Sintática e Semântica

No inicio do ficheiro são incluidos “includes” onde também se encontra um include para a apt. Seguidamente são declarados métodos que se usam para a symbol table (que se encontram definidos no final do ficheiro).
Depois encontra-se uma union que contém todos os tipos utilizados na analise sintática e para a criação da APT, que se encontram definidos no ficheiro apt.h, e depois definidos os tokens e types também para realização da analise sintática.

A APT é inicializada em “input:”, sendo que 1º têm de se definir declarações (ids ou funções ou defines), e nas definições de funções, no corpo, contem statements, que podem ser declarações, expressões, return, if then else, break, while, next, e ainda utilizar outras funções.

Depois de tudo encontra-se definida a symbol table, que é um array com 256 posições com structs do tipo “valor”, definidas também aqui.
As funções aqui definidas deixa inserir char*, Ids, Id e fazer look para verificar se já encontram-se na symbol table.
Esta versão da symbol table não é tão boa porque não contém scopes, apesar de ainda termos tentado criá-las. A versão que tentámos fazer para o trabalho com scopes e hashtable está incluida no ficheiro “symbolOLD.c”, que decidimos incluir no trabalho, pois achamos que esteja mais completa que a utilizada propriamente no trabalho, mas para termos 1 trabalho funcional que compila usámos a versão simplificada.

Em termos do ficheiro de “apt.h”, criamos vários enum para definir os tipos possiveis para os vários tipos de structs, definimos as ditas structs também e temos as funções que criam nós para a APT do tipo das structs que definimos anteriormente, levando como argumentos outros tipos definidos aqui neste ficheiro, ou tipos pre existentes (char, int, float,..).

A symbol table é utilizada ao longo da criação da APT, tendo os métodos nas declarações e nas chamadas de funções, assigns, entre outros. Devido à apt ser criada BOTTOM-UP, há casos em que as declarações dentro de argumentos de função aparecerem antes da função na symbol table, que pelo tempo apertado e talvez falta de conhecimento não conseguimos corrigir, mas achamos que está minimamente aceitável.

Como correr o trabalho?

Para correr o trabalho basta utilizar o make, com:

make all

e assim compila o trabalho, para fazer a árvore e ter 1 representação textual, eainda uma representação sa symbol table basta executar do tipo:

./ya < example1.ya

sendo **ya** o ficheiro compilado, e **example1.ya** 1 ficheiro com código na linguagem ya.

Exemplo com example1.ya

```
pimenta@pimenta-VirtualBox:~/Desktop/6SEM-UE/COMPILADORES/TRABALHO/trabalho/FINA
L3$ make all
bison -y -d -v ya.y -o parser.c
gcc -g parser.c lexer.c -o ya -lm
pimenta@pimenta-VirtualBox:~/Desktop/6SEM-UE/COMPILADORES/TRABALHO/trabalho/FINA
L3$ ./ya < example1.ya
Producao:ids ->id
Producao:type  int
Producao:exp -> intliteral
Producao:decl1 -> i,j:tipo = exp
Producao: decl-> decl1
Producao: decls-> decl
Producao:ids ->id
Producao:type  string
Producao:exp -> stringliteral
Producao:decl1 -> i,j:tipo = exp
Producao: decl-> decl1
Producao: decls-> decls decl
Producao:type  int
Producao: idstype -> id:tipo
Producao:type  void
Producao:ids ->id
Producao:mix  ids
Producao:funcao  id(mix)
Producao:exp -> funcao
Producao:stat-> exp
Producao:stats-> stat
Producao:corpo->  stats
Producao:decl1 i(i:tipo): tipo { corpo }
Producao: decl-> decl1
Producao: decls-> decls decl
Producao:type  void
Producao:ids ->id
Producao:type  bool
Producao:exp -> boolliteral
Producao:decl1 -> i,j:tipo = exp
Producao: decl-> decl1
Producao:stat-> decl
Producao:id   id
Producao:exp -> id
Producao:id   id
Producao:exp -> stringliteral
Producao:exp -> id=exp
Producao:stat-> exp
Producao:ids ->id
Producao:mix  ids
Producao:funcao  id(mix)
Producao:exp -> funcao
Producao:stat-> exp
Producao:stats-> stat
Producao:stats-> stat stats
Producao:corpo->  stats
Producao:stat-> if expbool then corpo
```


Producao:stat-> if expbool then corpo
Producao:stats-> stat
Producao:stats-> stat stats
Producao:corpo-> stats
Producao:decl1 i(): tipo { corpo }
Producao: decl-> decl1
Producao: decls-> decls decl
SYMBOL TABLE:
a INT
b STRING
b INT
f VOID
a BOOL
main VOID
Producao: program
pimenta@pimenta-VirtualBox:~/Desktop/6SEM-UE/COMPILADORES/TRABALHO/trabalho/FINA
L3$

Conclusão

Após a realização deste trabalho, ficámos a conhecer melhor como funcionam as construções de um compilador. Aplicámos conhecimentos adquiridos de modo a tentar resolver o problema proposto pelo professor, e conseguimos resolver parcialmente o problema proposto.
Conseguimos assim neste trabalho realizar a análise lexical e sintática correctamente, sendo que na análise semântica contemos a APT e a symbol table a funcionar minimamente, não tendo o RA para ver alguns conflitos de tipos, e sendo que a symbol table que usamos no trabalho é um pouco inferior à que tentámos realizar, sendo que por não conseguirmos a utilizar que usámos 1 versão anterior mais simples mas que funciona (sendo que incluimos a symbol table idealizada num ficheiro em separado).
Neste trabalho prático não foram assim completados na totalidade os objetivos do enunciado, devido a termos vários trabalhos de outras disciplinas para fazer, testes, exames e entre outros, mas ainda assim tentámos fazer o nosso melhor possível.