UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Informática

Projeto de Compiladores

2013/14 – 2º semestre

Licenciatura em Engenharia Informática

Entrega final: 30 de Maio de 2014

v1.0.1

Nota Importante: A fraude denota uma grave falta de ética e constitui um comportamento inadmissível num estudante do ensino superior e futuro profissional licenciado. Qualquer tentativa de fraude leva à reprovação à disciplina tanto do facilitador como do prevaricador.

Compiladores

Compilador para a linguagem iJava

Este projeto consiste no desenvolvimento de um compilador para a linguagem iJava (imperative Java), que consiste num pequeno subconjunto da linguagem Java (versão 5.0). Os programas da linguagem iJava são constituídos por uma única classe (a classe principal), contendo necessariamente um método `main`, e podendo conter outros métodos e atributos, todos eles estáticos e (possivelmente) públicos.

Na linguagem iJava é possível utilizar variáveis e literais dos tipos inteiro (de 32 bits) com sinal e booleano, e variáveis dos tipos array de inteiros e array de valores booleanos (com uma única dimensão). A linguagem implementa expressões aritméticas e lógicas e operações relacionais simples, bem como instruções de atribuição e de controlo (`if-else` e `while`). Implementa ainda métodos (estáticos) envolvendo quaisquer dos tipos de dados referidos acima e/ou o tipo de retorno `void`.

É possível passar parâmetros, que deverão ser literais inteiros, a um programa iJava através da linha de comandos. Supondo que o nome dado ao argumento do método `main()` é `args`, os seus valores podem ser recuperados através da construção `Integer.parseInt(args[...])`, e o número de parâmetros pode ser obtido através da expressão `args.length`. A construção `System.out.println(...)` permite imprimir valores inteiros ou lógicos.

Finalmente, são aceites (e ignorados) comentários dos tipos `/* ... */` e `//...`.

O significado de um programa em iJava será o mesmo que o seu significado em Java. Por exemplo, o seguinte programa imprime o primeiro argumento passado na linha de comandos:

```
class echo {
     public static void main(String[] args) {
          int x;
          x = Integer.parseInt(args[0]);
          System.out.println(x);
     }
}
```

Fases

O projeto será estruturado como uma sequência de três metas, a saber:

1. Análise lexical (10%) – 21 de março de 2014
2. Análise sintática, construção da árvore de sintaxe abstrata, análise semântica (tabelas de símbolos, deteção de erros semânticos) (55%) – 28 de abril de 2014
3. Geração de código (20%) + relatório (15%) – 30 de maio de 2014

Em cada uma das metas, o trabalho deverá ser validado no mooshak, usando um concurso criado especificamente para o efeito. Para além disso, a entrega final do projeto deverá ser feita no inforestudante até às **23h59** do dia **30 de Maio**, e incluir o relatório e todo o software criado.

Defesa e grupos

O trabalho será normalmente realizado por grupos de dois alunos, admitindo-se também que o seja a título individual. A defesa oral do trabalho terá lugar na primeira semana do mês de junho. A nota da defesa (entre 0 e 100%) multiplica pela média ponderada das pontuações obtidas no mooshak e no relatório à data de cada uma das metas. *Excecionalmente*, e por motivos justificados (como, por exemplo, falha técnica), poderão ser atribuídas notas superiores a 100% na defesa, mas a classificação final nunca poderá exceder a pontuação obtida no mooshak para as diversas fases à data da última entrega.

Aplicam-se mínimos de 47,5% à nota final após a defesa.

Bibliografia

. Rui Gustavo Crespo, *Processadores de Linguagensm* IST Press, 1998

. A. Appel, *Modern compiler implementation in C*. Cambridge Press, 1998.

. T. Niemann, *A Compact Guide to Lex & Yacc,*
http://epaperpress.com/lexandyacc/epaperpress.

. Manual do yacc em Unix (comando “man yacc” na shell)

. John R. Levine, Tony Mason and Doug Brown, *Lex & Yacc,* O’Reilly. 2004

. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd edition, 1988.

Fase I – Analisador lexical

O analisador lexical deve ser implementado em C utilizando a ferramenta `lex`. Os tokens da linguagem são apresentados de seguida.

Tokens da linguagem iJava

ID: Sequências alfanuméricas começadas por uma letra, onde os símbolos "_" e "$" contam como letras. Maiúsculas e minúsculas são consideradas letras diferentes.

INTLIT: Sequências de dígitos decimais, e sequências de dígitos hexadecimais (incluindo a-f e A-F) precedidas de "0x".

BOOLLIT = "true" | "false"
INT = "int"
BOOL = "boolean"
NEW = "new"
IF = "if"
ELSE = "else"
WHILE = "while"
PRINT = "System.out.println"
PARSEINT = "Integer.parseInt"
CLASS = "class"
PUBLIC = "public"
STATIC = "static"
VOID = "void"
STRING = "String"
DOTLENGTH = ".length"
RETURN = "return"
OCURV = "("
CCURV = ")"
OBRACE = "{"
CBRACE = "}"
OSQUARE = "["
CSQUARE = "]"
OP1 = "&&" | "||"
OP2 = "<" | ">" | "==" | "!=" | "<=" | ">="
OP3 = "+" | "-"
OP4 = "*" | "/" | "%"
NOT = "!"
ASSIGN = "="
SEMIC = ";"
COMMA = ","
RESERVED = keywords do Java não utilizadas em iJava, bem como o literal "null".

Implementação

O analisador deverá chamar-se `ijscanner`, ler o ficheiro a processar através do `stdin`, e emitir o resultado da análise para o `stdout`. Caso o ficheiro `echo.ijava` contenha o programa de exemplo dado anteriormente, a invocação

```
./ijscanner < echo.ijava
```

deverá imprimir a correspondente sequência de tokens no ecrã. Neste caso:

```
CLASS
ID(echo)
OBRACE
PUBLIC
STATIC
VOID
ID(main)
OCURV
STRING
OSQUARE
CSQUARE
ID(args)
CCURV
OBRACE
INT
ID(x)
SEMIC
ID(x)
ASSIGN
PARSEINT
OCURV
ID(args)
OSQUARE
INTLIT(0)
CSQUARE
CCURV
SEMIC
PRINT
OCURV
ID(x)
CCURV
SEMIC
CBRACE
CBRACE
```

O analisador deve aceitar (e ignorar) comentários dos tipos `/* ... */` e `//...`, e detetar a existência de quaisquer erros lexicais no ficheiro de entrada. Sempre que um token possa admitir mais do que um valor semântico, o valor encontrado deve ser impresso entre parêntesis logo a seguir ao nome do token, como se exemplificou acima para `ID` e `INTLIT`.

Tratamento de erros

Caso o ficheiro de entrada contenha erros lexicais, o programa deverá imprimir uma das seguintes mensagens no `stdout`, conforme o caso:

- "Line <num linha>, col <num coluna>: illegal character ('<c>')\n"
- "Line <num linha>, col <num coluna>: unterminated comment\n"

onde <c>, <num linha> e <num coluna> devem ser substituídos pelos valores correspondentes ao (início do) token que originou o erro. O analisador deve recuperar da ocorrência de erros lexicais a partir do fim desse token.

Submissão

O trabalho deverá ser validado no mooshak, usando o concurso criado especificamente para o efeito em `http://mooshak.dei.uc.pt/~comp2014`. Será tida em conta apenas a última submissão ao problema A desse concurso. Os restantes problemas destinam-se a ajudar na validação do analisador. No entanto, o mooshak não deve ser utilizado como ferramenta de debug!

O ficheiro lex a submeter deve chamar-se `ijscanner.l` e ser colocado num ficheiro zip com o nome `ijscanner.zip`. O ficheiro zip não deve conter quaisquer diretórios.