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

Projeto de Compiladores

2014/15– 2º semestre

Licenciatura em Engenharia Informática

Data de Entrega: 1 de Junho 2015

v1.0.2

Nota Importante: Qualquer tentativa de fraude leva à reprovação à disciplina tanto do facilitador como do prevaricador.

Compilador para a linguagem mili-Pascal (mPa)

Este projeto consiste no desenvolvimento de um compilador para a linguagem "mili-Pascal," que é um pequeno subconjunto da linguagem Pascal Standard ISO 7185:1990 com extensões relativas à passagem de parâmetros através da linha de comandos.

Nesta linguagem procedimental, os programas podem incluir dados e operações sobre esses dados. É possível utilizar variáveis e literais dos tipos lógico, inteiro e real. Também é possível usar literais do tipo cadeia de caracteres (string), mas apenas para efeitos de impressão no ecrã. A linguagem implementa expressões aritméticas e lógicas e operações relacionais simples, instruções de atribuição, de controlo (`if-then-else`, `while-do` e `repeat-until`) e de saída (`writeln`).

É possível passar parâmetros, que deverão ser literais inteiros, a um programa em mili-Pascal através da linha de comandos. Os seus valores podem ser recuperados através da construção `val(paramstr(`*i*`),`*v*`)`, que atribui o valor do *i*-ésimo parâmetro à variável *v*. O número de parâmetros pode ser obtido através da função pré-definida `paramcount`. A construção `writeln(...)` permite imprimir valores inteiros, reais (por exemplo, `1.000000000E+00`), lógicos (`TRUE` e `FALSE`), e cadeias de caracteres (por exemplo, `'Bom dia!'`). Finalmente, é possível definir funções, mas não procedimentos. São aceites (e ignorados) comentários dos tipos `(*...*)` e `{...}` (e ainda `{...*)` e `(*...}` !).

O significado de um programa em mili-Pascal será o mesmo que o seu significado em Pascal ISO 7185:1990 com a pré-definição das funções `paramcount`, `paramstr` e do procedimento `val`. Por exemplo, o seguinte programa deverá imprimir o valor do primeiro argumento passado na linha de comandos:

```
program echo(output);
var x: integer;
begin
    val(paramstr(1), x);
    writeln(x)
end.
```

Fases

O projeto será estruturado como uma sequência de quatro metas com ponderação e datas de entrega próprias, a saber:

1. Análise lexical (10%) – até 23 de março de 2015
2. Análise sintática (30%) – até 13 de abril de 2015
3. Análise semântica (25%) – até 4 de maio de 2015
4. Geração de código (20%) + relatório (15%) – até 1 de junho de 2015

Em cada uma das metas, o trabalho será obrigatoriamente 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 **1 de junho**, e incluir o relatório e todo o software criado, tal e qual foi submetido ao mooshak.

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 será **individual** e terá lugar entre os dias **8 e 12 de junho de 2015**. A nota da defesa (entre 0 e 100%) multiplica pela média ponderada das pontuações obtidas no mooshak e no relatório à data de entrega 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.

Fase I – Analisador lexical

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

NOTA: Em Pascal, não é feita qualquer distinção entre letras maiúsculas e minúsculas, salvo quando estas ocorrem no interior de cadeias de caracteres. Onde a seguir se usam letras minúsculas, também se devem considerar as correspondentes maiúsculas.

Tokens da linguagem mili-Pascal

ID : sequências alfanuméricas começadas por uma letra.

INTLIT : sequências de dígitos decimais.

REALLIT : sequências de dígitos decimais interrompidas por um único ponto e opcionalmente seguidas de um expoente, *ou* sequências de dígitos decimais seguidas de um expoente. O expoente consiste na letra "e", opcionalmente seguida de um sinal de + ou de - , seguida de uma sequência de dígitos decimais.

STRING : Sequências de caracteres (excluindo mudanças de linha) iniciadas por uma aspa simples (`'`) e terminadas pela primeira ocorrência de uma aspa simples que não seja seguida imediatamente por outra aspa simples. Por exemplo, "`'abc'`" e "`'texto entre ''aspas'''`".

ASSIGN = ":="
BEGIN = "begin"
COLON = ":"
COMMA = ","
DO = "do"
DOT = "."
ELSE = "else"
END = "end"
FORWARD = "forward"
FUNCTION = "function"
IF = "if"
LBRAC = "("
NOT = "not"
OUTPUT = "output"
PARAMSTR = "paramstr"
PROGRAM = "program"
RBRAC = ")"
REPEAT = "repeat"
SEMIC = ";"
THEN = "then"
UNTIL = "until"
VAL = "val"
VAR = "var"
WHILE = "while"
WRITELN = "writeln"
OP1 = "and" | "or"
OP2 = "<" | ">" | "=" | "<>" | "<=" | ">="

OP3 = "+" | "-"
OP4 = "*" | "/" | "mod" | "div"

RESERVED : palavras reservadas e identificadores requeridos do Pascal standard não usados. NOTA: os identificadores requeridos `boolean`, `false`, `integer`, `real` e `true` serão usados em fases posteriores do projeto, e não deverão ser RESERVED.

Implementação

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

```
./mpascanner < echo.mpa
```

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

```
PROGRAM
ID(echo)
LBRAC
OUTPUT
RBRAC
SEMIC
VAR
ID(x)
COLON
ID(integer)
SEMIC
BEGIN
VAL
LBRAC
PARAMSTR
LBRAC
INTLIT(1)
RBRAC
COMMA
ID(x)
RBRAC
SEMIC
WRITELN
LBRAC
ID(x)
RBRAC
END
DOT
```

O analisador deve aceitar (e ignorar) como separador de tokens espaço em branco (espaços, tabs e mudanças de linha), bem como comentários iniciados por `(*` ou `{` e terminados pela primeira ocorrência de `*)` ou `}`. Deve ainda 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 string\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://mooshak2.dei.uc.pt/~comp2015`. 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 `mpascanner.l` e ser colocado num ficheiro zip com o nome `mpascanner.zip`. O ficheiro zip não deve conter quaisquer diretórios.