Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
COMPILADORES
Professora: Mariza Andrade da Silva Bigonha
TRABALHO PRÁTICO - Valor: 15 PONTOS
Disponível em: 05/08/2015 - Devido em: 23/11/2013

1 Informações Gerais

A sua tarefa é construir um compilador para a mini-linguagem definida a seguir. A implementação deve ser obrigatoriamente em Java ou C++ e está dividida em 3 partes: *front-end*, tradução para bytecode, e, integração, execução e testes, descritas nas Seções 2, 3 e 4 respectivamente.

2 Front-End

O *front-end* deve gerar código para a linguagem intermediária de quádruplas, cujas instruções disponíveis devem ser definidas conforme a necessidade.

A linguagem dada **não** pode ser alterada.

Não haverá entrega de resultados intermediários e o prazo de entrega não é negociável.

O trabalho deve ser feito individualmente.

2.1 A Linguagem

Um programa na linguagem consiste em um bloco com declarações e comandos opcionais. O token **basic** representa os tipos básicos, **int, char, float** e **bool**.

$$
\begin{array}{rcl}
\textit{program} & \rightarrow & \textit{block} \\
\textit{block} & \rightarrow & \texttt{\{}\ \textit{decls stmts}\ \texttt{\}} \\
\textit{decls} & \rightarrow & \textit{decls decl} \mid \epsilon \\
\textit{decl} & \rightarrow & \textit{type}\ \textbf{id}\ \texttt{;} \\
\textit{type} & \rightarrow & \textit{type}\ \texttt{[}\ \textbf{num}\ \texttt{]} \mid \textbf{basic} \\
\textit{stmts} & \rightarrow & \textit{stmts stmt} \mid \epsilon
\end{array}
$$

$$
\begin{array}{rcl}
\textit{stmt} & \rightarrow & \textit{loc}\ \texttt{=}\ \textit{bool}\ \texttt{;} \\
 & \mid & \textbf{if}\ \texttt{(}\ \textit{bool}\ \texttt{)}\ \textit{stmt} \\
 & \mid & \textbf{if}\ \texttt{(}\ \textit{bool}\ \texttt{)}\ \textit{stmt}\ \textbf{else}\ \textit{stmt} \\
 & \mid & \textbf{while}\ \texttt{(}\ \textit{bool}\ \texttt{)}\ \textit{stmt} \\
 & \mid & \textbf{do}\ \textit{stmt}\ \textbf{while}\ \texttt{(}\ \textit{bool}\ \texttt{)}\ \texttt{;} \\
 & \mid & \textbf{break}\ \texttt{;} \\
 & \mid & \textit{block} \\
\textit{loc} & \rightarrow & \textit{loc}\ \texttt{[}\ \textit{bool}\ \texttt{]} \mid \textbf{id}
\end{array}
$$

$$
\begin{array}{rcl}
bool & \rightarrow & bool \texttt{ || } join \mid join \\
join & \rightarrow & join \texttt{ \&\& } equality \mid equality \\
equality & \rightarrow & equality \texttt{ == } rel \mid equality \texttt{ != } rel \mid rel \\
rel & \rightarrow & expr \texttt{ < } expr \mid expr \texttt{ <= } expr \mid expr \texttt{ >= } expr \mid \\
 & & \quad expr \texttt{ > } expr \mid expr \\
expr & \rightarrow & expr \texttt{ + } term \mid expr \texttt{ - } term \mid term \\
term & \rightarrow & term \texttt{ * } unary \mid term \texttt{ / } unary \mid unary \\
unary & \rightarrow & \texttt{ ! } unary \mid \texttt{ - } unary \mid factor \\
factor & \rightarrow & \texttt{ (} bool \texttt{) } \mid loc \mid \textbf{num} \mid \textbf{real} \mid \textbf{true} \mid \textbf{false}
\end{array}
$$

2.2 Criando o *Front-END*

Você deve implementar o *front-end* em partes, cada uma delas representando uma fase do compilador. Por exemplo, o código para os *pacote*s podem aparecer em cinco diretórios: **main, lexer, symbols, parser** e **inter**. Os comandos para criar o compilador variam de um sistema para outro. Os seguintes são de uma implementação do UNIX:

```
javac lexer/*.java
javac symbols/*.java
javac inter/*.java
javac parser/*.java
javac {\bf Main}/*.java
```

SUGESTÃO: para facilitar, siga a sugestão dada. Sugiro também que fiquem atentos no conteúdo da matéria em sala de aula para o desenvolvimento das fases citadas acima.

2.3 Teste

Exemplo de código fonte para ser usado como teste de seu *front-end*:

```
1)  {                // Arquivo test
2)    int i; int j; float v; float x; float[100] a;
3)    while( true ) {
4)       do i = i+1; while( a[i] < v);
5)       do j = j-1; while( a[j] > v);
6)       if( i >= j ) break;
7)       x = a[i]; a[i] = a[j]; a[j] = x;
8)    }
9)  }
```

2.3.1 Resultado

Para esta entrada, o *front-end* produz

```
1) L1:L3:  i = i + 1
2) L5:     t1 = i * 8
3)         t2 = a [ t1 ]
```

```
4)          if t2 < v goto L3
5) L4:      j = j - 1
6) L7:      t3 = j * 8
7)          t4 = a [ t3 ]
8)          if t4 > v goto L4
9) L6:      iffalse i >= j goto L8
10) L9:     goto L2
11) L8:     t5 = i * 8
12)         x = a [ t5 ]
13) L10:    t6 = i * 8
14)         t7 = j * 8
15)         t8 = a [ t7 ]
16)         a [ t6 ] = t8
17) L11:    t9 = j * 8
18)         a [ t9 ] = x
19)         goto L1
20) L2:
```

3 Tradutor para ByteCode

A sua tarefa é construir um **Tradutor** da linguagem intermediária, quádruplas, gerada pelo *front-end*, Seção 2, para o código da máquina virtual de Java **(ByteCode)**, de forma a ser possível compilar e executar programas na linguagem dada.

A linguagem dada **não** pode ser alterada.

3.1 Testes

Para testar seu tradutor use como entrada o código a seguir. É importante acrescentar outros testes.

```
1) L1:L3:   i = i + 1
2) L5:      t1 = i * 8
3)          t2 = a [ t1 ]
4)          if t2 < v goto L3
5) L4:      j = j - 1
6) L7:      t3 = j * 8
7)          t4 = a [ t3 ]
8)          if t4 > v goto L4
9) L6:      iffalse i >= j goto L8
10) L9:     goto L2
11) L8:     t5 = i * 8
12)         x = a [ t5 ]
13) L10:    t6 = i * 8
14)         t7 = j * 8
15)         t8 = a [ t7 ]
16)         a [ t6 ] = t8
17) L11:    t9 = j * 8
18)         a [ t9 ] = x
19)         goto L1
20) L2:
```

que representa o código intermediário produzido pelo *front-end* para o seguinte programa fonte:

```
1)  {                // Arquivo test
2)    int i; int j; float v; float x; float[100] a;
3)    while( true ) {
4)       do i = i+1; while( a[i] < v);
5)       do j = j-1; while( a[j] > v);
6)       if( i >= j ) break;
7)       x = a[i]; a[i] = a[j]; a[j] = x;
8)    }
9)  }
```

4 Integralização do Compilador

A sua tarefa é apresentar o compilador integrado, os testes, e os resultados obtidos de sua execução. O compilador integrado corresponde as partes descritas nas Seções 2 e 3. Para receber os 15 pontos no trabalho é fundamental executar o compilador implementado, bem como apresentar os resultados da execução para pelo menos 2 exemplos, aquele apresentado na Seção 4.1.1 e mais outro.

4.1 Testes

4.1.1 Programa Fonte

```
1)  {                // Arquivo test
2)    int i; int j; float v; float x; float[100] a;
3)    while( true ) {
4)       do i = i+1; while( a[i] < v);
5)       do j = j-1; while( a[j] > v);
6)       if( i >= j ) break;
7)       x = a[i]; a[i] = a[j]; a[j] = x;
8)    }
9)  }
```

5 Roteiro para a Entrega do Compilador Integrado

1. Salve todos os arquivos necessários para execução do trabalho em uma pasta chamada tpfinal.

2. Indique no trabalho impresso quais são os passos necessários para executar os testes - esses passos serão rigorosamente seguidos e deverão ser bem claros e objetivos.

3. Forneça dois arquivos de teste para seu trabalho. Esses testes deverão ser bem simples e vir acompanhados de instruções de como compilá-los e executá-los. Sugestão, use o teste da Seção 4.1 como primeiro teste. Dê os seguintes nomes para os arquivos: exemplo.txt e exemplo1.txt, respectivamente. Chame as instruções que os acompanham de instrucoes.txt e instrucoes.txt.

4. Coloque seu NOME no CD explicitando as partes *front-end*, *tradutor para bytecode* e compilador integrado.

5. Os testes serão efetuados:

   (a) Para Java: em uma máquina com Windows, dotada de CUP, JFlex e Java 6. Serão utilizadas as últimas versões disponíveis do JFlex e do CUP em seus sites oficiais.

   (b) Para C: em uma máquina com Linux via SSH ou Windows, dotadas de Flex, Bison e GCC 3.4.

   (c) Para outras linguagens e ambientes **forneça instruções COMPLETAS** de como compilar o código e executá-lo em uma das máquinas acessíveis do DCC utilizando SSH.

6. As instruções fornecidas para execução dos testes serão seguidas a risca, portanto, caso elas não funcionem os testes poderão **NÃO** ser executados.