TP 4
COMPILADORES

Dupla: **Roger Filipe Sales Gusmão**
Eduardo Caetano

1 . Introdução

O compilador è um tradutor que faz um mapeamento de programas, sendo dividido em duas interfaces: Front-end e back-end são termos generalizados que referem-se às etapas inicial e final de um processo. O front-end é responsável por coletar a entrada em várias formas do usuário e processá-la para adequá-la a uma especificação útil para o back-end. O front-end é uma espécie de interface entre o usuário e o back-end.

Com a implementação de Front-End, temos como resultado o código intermediário da gramatica SmallL listada abaixo:

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

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

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

O software desenvolvido é o compilador completo tendo a capacidade de gerar a partir do codigo fonte o codigo intermediario e a partir deste gera o codigo para a maquina virtual TAM.

2. Maquina Tam

A maquina TAM possui o seguinte conjunto de instruçoes:

Código Operação	Menmonico	Semântica
0	LOAD(n) d[r]	Carrega um objeto, n-word, do endereço de dados (d + registrador r) e o coloca na pilha.
1	LOADA d[r]	Empilha o endereço de dados (d + registrador r) no topo da pilha.
2	LOADI(n)	Desempilha o endereço de dados da pilha, carrega o objeto, n-word, daquele endereço e o empilha no topo da pilha.
3	LOADL d	Empilha o literal de valor d, 1-word, na pilha.
4	STORE(n) d[r]	Desempilha um objeto, n-word, da pilha e o armazena no endereço de dados (d + registrador r).
5	STOREI (n)	Desempilha um endereço da pilha, então desempilha um objeto, n-word, da pilha e o armazena naquele endereço.
6	CALL(n) d[r]	Ative a rotina no endereço de código (d + registrador r) usando o endereço no registrador n como *static link*.
7	CALLI	Desempilhe o *static link* e o endereço de código da pilha e então ative a rotina naquele código de endereço.
8	RETURN (n) d	Retorna da rotina corrente; desempilha o resultado, n-word, da pilha; e, então, desempilha o *frame* mais ao topo; desempilhe os argumentos, d words, e finalmente empilhe o resultado de volta no topo da pilha.
9	–	Não usado.
10	PUSH d	Empilha d palavras não inicializadas na pilha.
11	POP(n) d	Desempilha o resultado, n-word, da pilha, e então, desempilha mais d palavras, após, empilha o resultado de volta na pilha.
12	JUMP d[r]	Desvie para o endereço de código (d + registrador r).
13	JUMPI	Desempilha o endereço de código da pilha, e então, desvia para aquele endereço.
14	JUMPIF(n) d[r]	Desempilha um valor, 1-word, da pilha e então desvia para o endereço de código (d + registrador r) **se e somente se** aquele valor for igual a n.
15	HALT	Pare a execução do programa.

Figure 1: Instruções de TAM

Contem o seguinte conjunto de registradores:

Número do Registrador	Mnemonico Registrador	Nome do Registrador	Semântica
0	CB	Base do Código	constante
1	CT	Topo do Código	constante
2	PB	Base das Primitivas	constante
3	PT	Topo das Primitivas	constante
4	SB	Base da Pilha	constante
5	ST	Topo da Pilha	é mudado para a maior parte das instruções
6	HB	Base do Heap	constante
7	HT	Topo do Heap	é mudado pelas rotinas no heap
8	LB	Base Local	é mudado com a ativação e retorno das instruções
9	L1	Base Local 1	L1 = conteúdo(LB)
10	L2	Base Local 2	L2 = conteúdo(conteúdo(LB))
11	L3	Base Local 3	L3 = conteúdo(conteúdo(conteúdo((LB)))
12	L4	Base Local 4	L4 = conteúdo(conteúdo(conteúdo(conteúdo((LB))))
13	L5	Base Local 5	L5 = conteúdo(conteúdo(conteúdo(conteúdo(conteúdo((LB)))))
14	L6	Base Local 6	L6 = conteúdo(conteúdo(conteúdo(conteúdo(conteúdo(conteúdo((LB))))))
15	CP	Apontador do Código	é mudado para todas as instruções

Figure 5: Registradores de TAM

E sua pilha possui a seguinte disposição:

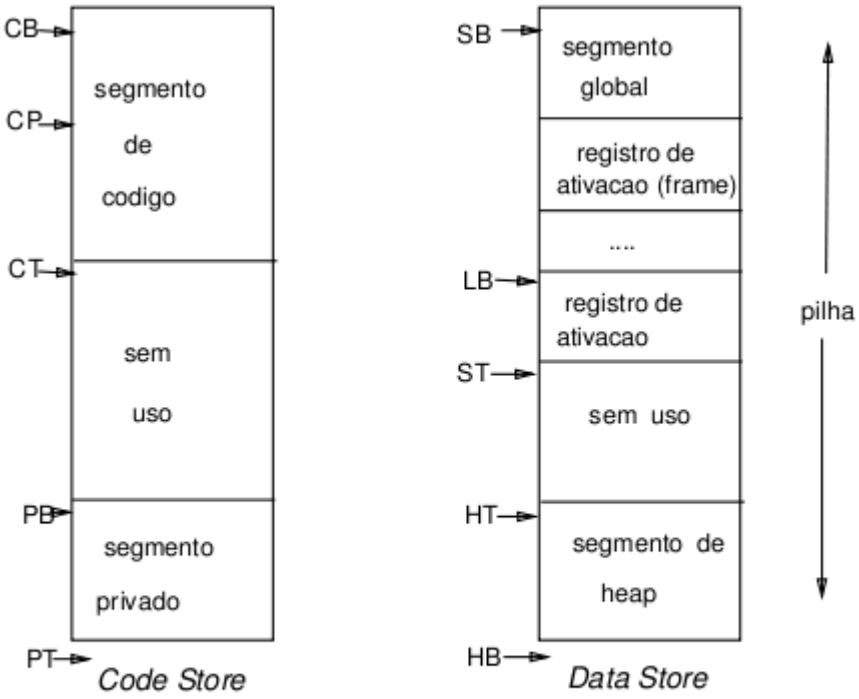

Figure 1: Arquitetura das memórias de instruções e dados de TAM

Na geração das instruçoes TAM utilizamos os registradores CB e CT para controle do fluxo das instruçoes e os registradores SB e ST para armazenamento das variaveis ao longo do codigo.

3 . O Software

O software é constituido dos seguintes pelos softwares desenvolvidos no Trabalho Pratico 2 (Gerador de Codigo Intermediario) desenvolvido em Java e compilado pelo compilador Java 1.5. e no Trabalho Pratico 3 (Gerador de codigo para a maquina TAM) desenvolvido em C++ e compilado no g++ 4.3.

Para a integracao dos dois softwares fora implementado o integrador CompilaJava.c, que é chamado no makefile, para a compilacao do software TP2. Quando executado a funcao main do software TP3 se encarrega de chamar o gerador de codigo intermediario, colocando a saida no arquivo 'CodigoIntermediario', sendo este direcionado como entrada para o gerador de codigo TAM.

4 . Execução

Esta ferramenta possui o seguinte makefile integrado:

```
all: run

run: TP3 COMPILA_TP2
	./COMPILA_TP2
----->	./TP3 entrada saida1

TP3: main.o tabela.o quadruplas.o tradutor.o
	g++ -g main.o tabela.o quadruplas.o tradutor.o -lm -o TP3

COMPILA_TP2: CompilaJava.o
	g++ -g CompilaJava.o -o COMPILA_TP2

main.o: main.cpp
	g++ -g -c -Wall -Werror main.cpp -o main.o

tabela.o: tabela.cpp tabela.hpp
	g++ -g -c -Wall -Werror tabela.cpp -o tabela.o
```

quadruplas.o: quadruplas.cpp quadruplas.hpp
	g++ -g -c -Wall -Werror quadruplas.cpp -o quadruplas.o

tradutor.o: tradutor.cpp tradutor.hpp
	g++ -g -c -Wall -Werror tradutor.cpp -o tradutor.o

CompilaJava.o: CompilaJava.cpp
	g++ -g -c -Wall -Werror CompilaJava.cpp -o CompilaJava.o

clean:
	-rm -rf main.o tabela.o quadruplas.o tradutor.o CompilaJava.o

Em que na linha 5 deve conter em:

	entrada = nome arquivo entrada
	saida = nome arquivo saida

A ferramenta deve ser compilada e executada utilizado no compilador g++ 4.1, através das seguintes instrução :

	make

5 . Testes

Foram utilizados os testes abaixo:

Teste 1

{

```
 int i; int j; float v; float x; float[100] a;
 while( true ) {
   do i = i+1; while( a[i] < v);
   do j = j-1; while( a[j] > v);
   if( i >= j ) break;
   x = a[i]; a[i] = a[j]; a[j] = x;
 }
}
```

Codigo Intermediario Gerado:

```
L1:L3:  i = i + 1
L5:     t1 = i * 8
        t2 = a [ t1 ]
        if t2 < v goto L3
L4:     j = j - 1
L7:     t3 = j * 8
        t4 = a [ t3 ]
        if t4 > v goto L4
L6:     iffalse i >= j goto L8
L9:     goto L2
L8:     t5 = i * 8
        x = a [ t5 ]
L10:    t6 = i * 8
        t7 = j * 8
        t8 = a [ t7 ]
        a [ t6 ] = t8
L11:    t9 = j * 8
        a [ t9 ] = x
        goto L1
L2:
```

Saida:

Impressão das quadruplas

OP	ARG_1	ARG_2	RESULT
L1			
L3			
+	i	1	i
L5			
*	i	8	t1
LOAD	a	t1	t2
<	t2	v	L3
L4			
-	j	1	j
L7			
*	j	8	t3
LOAD	a	t3	t4
>	t4	v	L4
L6			
>=	i	j	L8
L9			
goto	L2		
L8			
*	i	8	t5
LOAD	a	t5	x
L10			
*	i	8	t6
*	j	8	t7
LOAD	a	t7	t8
STORE	t6	t8	a
L11			
*	j	8	t9
STORE	t9	x	a
goto	L1		

CT	Impressão do Programa TAM
0	LOADL 0
1	LOADL 1
2	CALL succ
3	STORE (1) 0 [SB]
4	LOAD (1) 0
5	LOADL 8
6	CALL mult
7	STORE (1) 1 [SB]
8	PUSH 100
9	STORE (100) 2 [SB]
10	LOAD (1) 1 [SB]
11	STORE (1) 102 [SB]
12	LOAD (1) 102 [SB]
13	LOADL 0
14	CALL lt
15	JUMPIF (1) 0 [CB]
16	LOADL 0
17	LOADL 1
18	CALL sub
19	STORE (1) 103 [SB]
20	LOAD (1) 103
21	LOADL 8
22	CALL mult
23	STORE (1) 104 [SB]

```
24              LOAD ( 1 ) 2 [SB]
25              STORE (1) 105 [SB]
26              LOAD (1) 105 [SB]
27              LOADL 0
28              CALL gt
29              JUMPIF (1) 16 [CB]
30              LOAD (1) 0
31              LOAD (1) 103 [SB]
32              CALL ge
33              STORE (1) 106 [SB]
34              B2
35              LOAD (1) 0
36              LOADL 8
37              CALL mult
38              STORE (1) 107 [SB]
39              LOAD ( 1 ) 2 [SB]
40              STORE (1) 108 [SB]
41              LOAD (1) 0
42              LOADL 8
43              CALL mult
44              STORE (1) 109 [SB]
45              LOAD (1) 103
46              LOADL 8
47              CALL mult
48              STORE (1) 110 [SB]
49              LOAD ( 1 ) 2 [SB]
50              STORE (1) 111 [SB]
51              LOAD (1)111 [SB]
52              STORE (1) 111 [SB]
53              LOAD (1) 103
54              LOADL 8
55              CALL mult
56              STORE (1) 112 [SB]
57              LOAD (1)108 [SB]
58              PUSH 1
59              STORE (1) 113 [SB]
60              STORE (1) 2 [SB]
61              LOADL 0
62              JUMPI
63              HALT
------------------------
```

Teste 2

```
{
  bool a;
  int i;
  i = 0;
  while( true ) {
     if( i < 10 ){
      a = false;
      break;
     }
  }
}
```

Codigo Intermediario Gerado:

```
L1:       i = 0
L3:L4:    iffalse i < 10 goto L3
L5:       a = false
L6:       goto L2
```

```
        goto L3
L2:
```

Saida 2

```
Impressão das quadruplas
----------------------
OP                  ARG_1               ARG_2               RESULT
L1
=                   0                                       i
L3
L4
<                   i                   10                  L3
L5
=                   false                                   a
L6
goto                L2
goto                L3
----------------------

CT          Impressão do Programa TAM
-------------------------
0           LOADL 0
1           STORE (1) 0 [SB]
2           LOAD (1) 0 [SB]
3           LOADL 10
4           CALL lt
5           JUMPIF (1) 2 [CB]
6           LOADL 0
7           STORE (1) 1 [SB]
8           B2
9           LOADL 2
10          JUMPI
11          HALT
-------------------------
```

Teste 3

```
{
  int i;
  bool b;
  float[100] t;
  b = false;
  i = 0;
  do{
        if(b == true){
              t[i] = 1;
        }
        else{
               t[i] = 200;
        }
        if(i > 10){
              b = true;
        }
        i = i + 1;
  } while(true);
}
```

Codigo Intermediario Gerado:

```
L1:     b = false
L3:     i = 0
```

```
L4:     iffalse b == true goto L8
L7:     t1 = i * 8
        t [ t1 ] = 1
        goto L6
L8:     t2 = i * 8
        t [ t2 ] = 200
L6:     iffalse i > 10 goto L9
L10:    b = true
L9:     i = i + 1
L5:     goto L4
L2:
```

Saida 3

Impressão das quadruplas

OP	ARG_1	ARG_2	RESULT
L1			
==	b	true	L6
L5			
*	i	8	t1
STORE	t1	1	t
goto	L4		
L6			
*	i	8	t2
STORE	t2	200	t
L4			
>	i	10	L7
L8			
=	true		b
L7			
+	i	1	i
L3			
goto	L1		

CT	Impressão do Programa TAM
0	LOADL 0
1	LOADL 0
2	CALL eq
3	STORE (1) 0 [SB]
4	LOADL 0
5	LOADL 8
6	CALL mult
7	STORE (1) 1 [SB]
8	LOADL 0
9	PUSH 100
10	STORE (100) 2 [SB]
11	STORE (1) 1 [SB]
12	goto 19
13	LOADL 0
14	LOADL 8
15	CALL mult
16	STORE (1) 102 [SB]
17	LOADL 0
18	STORE (1) 104 [SB]
19	LOADL 0
20	LOADL 10
21	CALL gt
22	JUMPIF (1) 25 [CB]
23	LOADL 0

```
24              STORE (1) 103 [SB]
25              LOADL 0
26              LOADL 1
27              CALL succ
28              STORE (1) 104 [SB]
29              LOADL 0
30              JUMPI
31              HALT
-------------------------
```

Teste 4

```
{
 int i;
 int x;
 x = 100;
 i = 1;
 do{
        if(x < 20){
                do{
                        x = x - 1;
                } while( x > 0);
                i = i * 2;
        }
        else{
                 x = x - 1;
                 i = i / 2;
        }

        if(x == 10000){
                x = i;
                break;
        }

 } while(true);
}
```

Codigo Intermediario

```
L1:     x = 100
L3:     i = 1
L4:     iffalse x < 20 goto L8
L7:     x = x - 1
L10:    if x > 0 goto L7
L9:     i = i * 2
        goto L6
L8:     x = x - 1
L11:    i = i / 2
L6:     iffalse x == 10000 goto L5
L12:    x = i
L13:    goto L2
L5:     goto L4
L2:
```

Saida 4

```
Impressão das quadruplas
---------------------
OP                      ARG_1                   ARG_2                   RESULT
L1
=                       100                                             x
L3
```

```
=                   1                                   i
L4
<                   x                   20              L8
L7
-                   x                   1               x
L10
>                   x                   0               L7
L9
*                   i                   2               i
goto                L6
L8
-                   x                   1               x
L11
/                   i                   2               i
L6
==                  x                   10000           L5
L12
=                   i                                   x
L13
goto                L2
L5
goto                L4
----------------------

CT                  Impressão do Programa TAM
-------------------------
0                   LOADL 100
1                   STORE (1) 0 [SB]
2                   LOADL 1
3                   STORE (1) 1 [SB]
4                   LOAD (1) 0 [SB]
5                   LOADL 20
6                   CALL lt
7                   JUMPIF (1) 21 [CB]
8                   LOAD (1) 0
9                   LOADL 1
10                  CALL sub
11                  STORE (1) 0 [SB]
12                  LOAD (1) 0 [SB]
13                  LOADL 0
14                  CALL gt
15                  JUMPIF (1) 8 [CB]
16                  LOAD (1) 1
17                  LOADL 2
18                  CALL mult
19                  STORE (1) 1 [SB]
20                  goto 29
21                  LOAD (1) 0
22                  LOADL 1
23                  CALL sub
24                  STORE (1) 0 [SB]
25                  LOAD (1) 1
26                  LOADL 2
27                  CALL div
28                  STORE (1) 1 [SB]
29                  LOAD (1) 0
30                  LOADL 10000
31                  CALL eq
32                  STORE (1) 2 [SB]
33                  LOAD (1) 1
34                  STORE (1) 0 [SB]
35                  B2
36                  LOADL 4
```

```
37              JUMPI
38              HALT
------------------------
```

6 . Código Fonte

O progama é constituido do software TP2, TP3 e pelo integrador “CompilaJava.cpp” , sendo o controlador implementado na funcao main do software TP3.

6.1 CompilaJava.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string>
using namespace std;


int main( int argc, char *argv[] ){
        string compila = "javac lexer/*.java symbols/*.java inter/*.java parser/*.java teste/*.java";

        system (compila.c_str());
}
```

6.2 TP2

Segue abaixo o código do programa:

1 . Pacote Main

Main.java

```
package main;               // Arquivo Main.java

import java.io.*;
import lexer.*;
import parser.*;

public class TP2 {

        public static void main(String[] args) throws IOException {
                Lexer lex = new Lexer();
                Parser parse = new Parser(lex);
                parse.program();
                System.out.write('\n');
        }
}
```

2. Pacote Parse

Parser.java

```
package parser;             // Arquivo Parser.java
import java.io.*;


import lexer.*; import symbols.*; import inter.*;
public class Parser {
        private Lexer lex;

        private Token look;   // lookahead token
        Env top = null;
```

```
int used = 0;

public Parser(Lexer l) throws IOException { lex = l; move(); }

void move() throws IOException { look = lex.scan(); }

void error(String s) { throw new Error("near line "+Lexer.line +": "+s); }

void match(int t) throws IOException {

        if( look.tag == t ) move();
        else error("\n\nsyntax error");
}

public void program() throws IOException { // program -> block
        Stmt s = block();
        int begin = s.newlabel(); int after = s.newlabel();
        s.emitlabel(begin); s.gen(begin, after); s.emitlabel(after);
}

Stmt block() throws IOException { // block -> { decls stmts }

        match('{');
        Env savedEnv = top;
        top = new Env(top);

        decls();
        Stmt s = stmts();
        match('}');
        top = savedEnv;

        return s;
}

void decls() throws IOException {
        while( look.tag == Tag.BASIC ) {    // D -> type ID ;
                Type p = type(); Token tok = look; match(Tag.ID); match(';');
                Id id = new Id((Word)tok, p, used);
                top.put( tok, id );
                used = used + p.width;
        }
}

Type type() throws IOException {
        Type p = (Type)look;            // espera look.tag == Tag.BASIC
        match(Tag.BASIC);
        if( look.tag != '[' ) return p; // T -> basic
        else return dims(p);            // retorna tipo do arranjo
}

Type dims(Type p) throws IOException {
        match('['); Token tok = look; match(Tag.NUM); match(']');
        if( look.tag == '[' )
                p = dims(p);
        return new Array(((Num)tok).value, p);
}


Stmt stmts() throws IOException {
        if ( look.tag == '}' )          return Stmt.Null;
        else return new Seq(stmt(), stmts());
}
```

```
Stmt stmt() throws IOException {
        Expr x;
        Stmt  s1, s2;
        Stmt savedStmt;

        switch( look.tag ) {
        case ';':
                move();
                return Stmt.Null;
        case Tag.IF:
                match(Tag.IF); match('('); x = bool(); match(')');
                s1 = stmt();
                if( look.tag != Tag.ELSE ) return new If(x, s1);
                match(Tag.ELSE);
                s2 = stmt();
                return new Else(x, s1, s2);
        case Tag.WHILE:
                While whilenode = new While();
                savedStmt = Stmt.Enclosing; Stmt.Enclosing = whilenode;
                match(Tag.WHILE); match('('); x = bool(); match(')');
                s1 = stmt();
                whilenode.init(x, s1);
                Stmt.Enclosing = savedStmt; // reinicia Stmt.Enclosing
                return whilenode;
        case Tag.DO:
                Do donode = new Do();
                savedStmt = Stmt.Enclosing; Stmt.Enclosing = donode;
                match(Tag.DO);
                s1 = stmt();
                match(Tag.WHILE); match('('); x = bool(); match(')'); match(';');
                donode.init(s1, x);
                Stmt.Enclosing = savedStmt; // reinicia Stmt.Enclosing
                return donode;
        case Tag.BREAK:
                match(Tag.BREAK); match(';');
                return new Break();
        case '{':
                return block();
        default:
                return assign();
        }
}

Stmt assign() throws IOException {
        Stmt stmt; Token t = look;
        match(Tag.ID);
        Id id = top.get(t);
        if( id == null ) error(t.toString() + " undeclared");
        if( look.tag == '=' ) {       // S -> id = E ;
                move(); stmt = new Set(id, bool());
        }
        else {                        // S -> L = E ;
                Access x = offset(id);
                match('='); stmt = new SetElem(x, bool());
        }
        match(';');
        return stmt;
}



Expr bool() throws IOException {
```

```
        Expr x = join();
        while( look.tag == Tag.OR ) {
                Token tok = look; move(); x = new Or(tok, x, join());
        }
        return x;
}
Expr join() throws IOException {
        Expr x = equality();
        while( look.tag == Tag.AND ) {
                Token tok = look; move(); x = new And(tok, x, equality());
        }
        return x;
}
Expr equality() throws IOException {
        Expr x = rel();
        while( look.tag == Tag.EQ || look.tag == Tag.NE ) {
                Token tok = look; move(); x = new Rel(tok, x, rel());
        }
        return x;
}
Expr rel() throws IOException {
        Expr x = expr();
        switch( look.tag ) {
        case '<': case Tag.LE: case Tag.GE: case '>':
                Token tok = look; move(); return new Rel(tok, x, expr());
        default:
                return x;
        }
}
Expr expr() throws IOException {
        Expr x = term();
        while( look.tag == '+' || look.tag == '-' ) {
                Token tok = look; move(); x = new Arith(tok, x, term());
        }
        return x;
}
Expr term() throws IOException {
        Expr x = unary();
        while(look.tag == '*' || look.tag == '/' ) {
                Token tok = look; move();    x = new Arith(tok, x, unary());
        }
        return x;
}
Expr unary() throws IOException {
        if( look.tag == '-' ) {
                move(); return new Unary(Word.minus, unary());
        }
        else if( look.tag == '!' ) {
                Token tok = look; move(); return new Not(tok, unary());
        }
        else return factor();
}


Expr factor() throws IOException {
        Expr x = null;
        switch( look.tag ) {
        case '(':
                move(); x = bool(); match(')');
                return x;
        case Tag.NUM:
                x = new Constant(look, Type.Int);     move(); return x;
        case Tag.REAL:
```

```
                x = new Constant(look, Type.Float); move(); return x;
            case Tag.TRUE:
                x = Constant.True;              move(); return x;
            case Tag.FALSE:
                x = Constant.False;             move(); return x;
            default:
                error("syntax error");
                return x;
            case Tag.ID:
                String s = look.toString();
                Id id = top.get(look);
                if( id == null ) error(look.toString() + " undeclared");
                move();
                if( look.tag != '[' ) return id;
                else return offset(id);
            }
        }
        Access offset(Id a) throws IOException {    // I -> [E] | [E] I
            Expr i; Expr w; Expr t1, t2; Expr loc; // herda id
            Type type = a.type;
            match('['); i = bool(); match(']');

            type = ((Array)type).of;
            w = new Constant(type.width);
            t1 = new Arith(new Token('*'), i, w);
            loc = t1;
            while( look.tag == '[' ) {      // I multidimensional -> [ E ] I
                match('['); i = bool(); match(']');
                type = ((Array)type).of;
                w = new Constant(type.width);
                t1 = new Arith(new Token('*'), i, w);
                t2 = new Arith(new Token('+'), loc, t1);
                loc = t2;
            }
            return new Access(a, loc, type);
        }
}
```

3. Pacote Symbols

Array.java

```
package symbols;            // Arquivo Array.java
import lexer.*;
public class Array extends Type {
        public Type of;             // arranjo *of* type
        public int size = 1;

        public Array(int sz, Type p) {
            super("[]", Tag.INDEX, sz*p.width); size = sz; of = p;
        }
        public String toString() { return "[" + size + "] " + of.toString(); }
}
```

Env.java

```
package symbols;            // Arquivo Env.java

import java.util.*; import lexer.*; import inter.*;

public class Env {
        private Hashtable<Token, Id> table;
```

```
        protected Env prev;
        public Env(Env n) { table = new Hashtable<Token, Id>(); prev = n; }
        public void put(Token w, Id i) { table.put(w, i); }
        public Id get(Token w) {
                for( Env e = this; e != null; e = e.prev ) {
                        Id found = (Id)(e.table.get(w));
                        if( found != null ) return found;
                }
                return null;
        }
}
```

Type.java

```
package symbols;                // Arquivo Type.java
import lexer.*;
public class Type extends Word {
        public int width = 0;

        public Type(String s, int tag, int w) { super(s, tag); width = w; }
        public static final Type
        Int   = new Type( "int",   Tag.BASIC, 4 ),
        Float = new Type( "float", Tag.BASIC, 8 ),
        Char = new Type( "char", Tag.BASIC, 1 ),
        Bool = new Type( "bool", Tag.BASIC, 1 );
        public static boolean numeric(Type p) {
                if (p == Type.Char || p == Type.Int || p == Type.Float) return true;
                else return false;
        }
        public static Type max(Type p1, Type p2 ) {
                if ( ! numeric(p1) || ! numeric(p2) ) return null;
                else if ( p1 == Type.Float || p2 == Type.Float ) return Type.Float;
                else if ( p1 == Type.Int   || p2 == Type.Int   ) return Type.Int;
                else return Type.Char;
        }
}
```

4. Pacote Lexer

Lexer.java

```
package lexer;                // Arquivo Lexer.java
import java.io.*;
import symbols.*;
import java.util.Hashtable;

public class Lexer {
        public static int line = 1;
        public char peek = '\0';

        Hashtable<String, Word> words = new Hashtable<String, Word>();

        void reserve(Word w) { words.put(w.lexeme, w); }

        public Lexer() {
                reserve( new Word("if",    Tag.IF)    );
                reserve( new Word("else", Tag.ELSE) );
                reserve( new Word("while", Tag.WHILE) );
                reserve( new Word("do",    Tag.DO)    );
                reserve( new Word("break", Tag.BREAK) );
                reserve( Word.True ); reserve( Word.False );
                reserve( Type.Int ); reserve( Type.Char );
```

```
        reserve( Type.Bool ); reserve( Type.Float );
}

void readch() throws IOException { peek = (char)System.in.read(); }

boolean readch(char c) throws IOException {
        readch();
        if( peek != c ) return false;
        peek = '\0';
        return true;
}

public Token scan() throws IOException {

        for( ; ; readch() ) {

                while(peek == '\0'){
                        readch();

                }
                if( peek == 13)
                        continue;
                if( peek == ' ' || peek == '\t' )
                        continue;

                else if( peek == '\n' ) line = line + 1;
                else break;
        }
        switch( peek ) {
        case '&' :
                if( readch('&') ) return Word.and; else return new Token('&');
        case '|' :
                if( readch('|') ) return Word.or;    else return new Token('|');
        case '=' :
                if( readch('=') ) return Word.eq;    else return new Token('=');
        case '!':
                if( readch('=') ) return Word.ne;    else return new Token('!');
        case '<':
                if( readch('=') ) return Word.le;    else return new Token('<');
        case '>':
                if( readch('=') ) return Word.ge;    else return new Token('>');
        }
        if( Character.isDigit(peek) ) {
                int v = 0;
                do {
                        v = 10*v + Character.digit(peek, 10); readch();
                } while( Character.isDigit(peek) );
                if( peek != '.' ) return new Num(v);
                float x = v; float d = 10;
                for(;;) {
                        readch();
                        if( ! Character.isDigit(peek) ) break;
                        x = x + Character.digit(peek, 10) / d; d = d*10;
                }
                return new Real(x);
        }
        if( Character.isLetter(peek) ) {

                StringBuffer b = new StringBuffer();
                do {
                        b.append(peek); readch();
                } while( Character.isLetterOrDigit(peek) );
                String s = b.toString();
```

```
                    Word w = (Word)words.get(s);
                    if( w != null ) return w;
                    w = new Word(s, Tag.ID);
                    words.put(s, w);
                    return w;
                }

                Token tok = new Token(peek); peek = '\0';
                return tok;
        }
}
```

Num.java

```
package lexer;          // Arquivo Num.java
public class Num extends Token {
        public final int value;
        public Num(int v) { super(Tag.NUM); value = v; }
        public String toString() { return "" + value; }
}
```

Real.java

```
package lexer;          // Arquivo Real.java
public class Real extends Token {
        public final float value;
        public Real(float v) { super(Tag.REAL); value = v; }
        public String toString(){ return "" + value; }
}
```

Tag.java

```
package lexer; // Arquivo Tag.java

public class Tag {
        public final static int
        AND   = 256, BASIC = 257, BREAK   = 258, DO   = 259, ELSE  = 260,
        EQ    = 261, FALSE = 262, GE      = 263, ID   = 264, IF    = 265,
        INDEX = 266, LE    = 267, MINUS = 268, NE   = 269, NUM   = 270,
        OR    = 271, REAL = 272, TEMP     = 273, TRUE = 274, WHILE = 275;
}
```

Token.java

```
package lexer;
public class Token {
        public final int tag;
        public Token(int t) { tag = t; }
        public String toString() {return "" + (char)tag;}
}
```

Word.java

```
package lexer;          // Arquivo Word.java
public class Word extends Token {
        public String lexeme = "";
        public Word(String s, int tag) {super(tag); lexeme = s; }
        public String toString() { return lexeme; }
        public static final Word
        and = new Word( "&&", Tag.AND ), or = new Word( "||", Tag.OR ),
        eq = new Word( "==", Tag.EQ ), ne = new Word(   "!=", Tag.NE ),
        le = new Word( "<=", Tag.LE ), ge = new Word(   ">=", Tag.GE ),
```

```
        minus = new Word( "minus", Tag.MINUS ),
        True  = new Word( "true", Tag.TRUE ),
        False = new Word( "false", Tag.FALSE ),
        temp  = new Word( "t",     Tag.TEMP );
}
```

5. Pacote Inter

Access.java

```
package inter;              // Arquivo Access.java
import lexer.*; import symbols.*;
public class Access extends Op {
        public Id array;
        public Expr index;
        public Access(Id a, Expr i, Type p) {

                super(new Word("[]", Tag.INDEX), p); // achatar o arranjo
                array = a; index = i;
        }
        public Expr gen() { return new Access(array, index.reduce(), type); }
        public void jumping(int t,int f) { emitjumps(reduce().toString(),t,f); }
        public String toString() {
                return array.toString() + " [ " + index.toString() + " ]";
        }
}
```

And.java

```
package inter;              // Arquivo And.java

import lexer.*;

public class And extends Logical {
        public And(Token tok, Expr x1, Expr x2) { super(tok, x1, x2); }
        public void jumping(int t, int f) {
                int label = f != 0 ? f : newlabel();
                expr1.jumping(0, label);
                expr2.jumping(t,f);
                if( f == 0 ) emitlabel(label);
        }
}
```

Arith.java

```
package inter;              // Arquivo Arith.java
import lexer.*; import symbols.*;
public class Arith extends Op {
        public Expr expr1, expr2;
        public Arith(Token tok, Expr x1, Expr x2) {
                super(tok, null); expr1 = x1; expr2 = x2;
                type = Type.max(expr1.type, expr2.type);
                if (type == null ) error("type error");
        }
        public Expr gen() {
                return new Arith(op, expr1.reduce(), expr2.reduce());
        }
        public String toString() {
                return expr1.toString()+" "+op.toString()+" "+expr2.toString();
        }
}
```

Break.java

```
package inter;              // Arquivo Break.java


public class Break extends Stmt {
        Stmt stmt;
        public Break() {
                if( Stmt.Enclosing == Stmt.Null ) error("unenclosed break");
                stmt = Stmt.Enclosing;
        }
        public void gen(int b, int a) {
                emit( "goto L" + stmt.after);
        }
}
```

Constant.java

```
package inter;              // Arquivo Constant.java
import lexer.*; import symbols.*;
public class Constant extends Expr {
        public Constant(Token tok, Type p) { super(tok, p); }
        public Constant(int i) { super(new Num(i), Type.Int); }
        public static final Constant
        True = new Constant(Word.True, Type.Bool),
        False = new Constant(Word.False, Type.Bool);
        public void jumping(int t, int f) {
                if ( this == True && t != 0 ) emit("goto L" + t);
                else if ( this == False && f != 0) emit("goto L" + f);
        }
}
```

Do.java

```
package inter;              // Arquivo Do.java
import symbols.*;
public class Do extends Stmt {
        Expr expr; Stmt stmt;
        public Do() { expr = null; stmt = null; }
        public void init(Stmt s, Expr x) {
                expr = x; stmt = s;
                if( expr.type != Type.Bool ) expr.error("boolean required in do");
        }
        public void gen(int b, int a) {
                after = a;
                int label = newlabel();   // rotulo para expr

                stmt.gen(b,label);
                emitlabel(label);
                expr.jumping(b,0);
        }
}
```

Else.java

```
package inter;              // Arquivo Else.java
import symbols.*;
public class Else extends Stmt {
        Expr expr; Stmt stmt1, stmt2;
        public Else(Expr x, Stmt s1, Stmt s2) {
                expr = x; stmt1 = s1; stmt2 = s2;
                if( expr.type != Type.Bool ) expr.error("boolean required in if");
```

```
        }
        public void gen(int b, int a) {
                int label1 = newlabel();  // label1 para stmt1
                int label2 = newlabel();  // label2 para stmt2
                expr.jumping(0,label2);   // segue para stmt1 se expr for true
                emitlabel(label1); stmt1.gen(label1, a); emit("goto L" + a);
                emitlabel(label2); stmt2.gen(label2, a);
        }
}
```

Expr.java

```
package inter;              // Arquivo Expr.java
import lexer.*;
import symbols.*;

public class Expr extends Node {
        public Token op;
        public Type type;
        Expr(Token tok, Type p) { op = tok; type = p; }
        public Expr gen() { return this; }
        public Expr reduce() { return this; }
        public void jumping(int t, int f) { emitjumps(toString(), t, f); }
        public void emitjumps(String test, int t, int f) {
                if( t != 0 && f != 0 ) {
                        emit("if " + test + " goto L" + t);
                        emit("goto L" + f);
                }
                else if( t != 0 ) emit("if " + test + " goto L" + t);
                else if( f != 0 ) emit("iffalse " + test + " goto L" + f);
                else ;// nada, porque ambos t e f fall through

        }
        public String toString() {
                return op.toString();
        }
}
```

Id.java

```
package inter;              // Arquivo Id.java
import lexer.*; import symbols.*;
public class Id extends Expr {
        public int offset;    // endere¸o relativo

        public Id(Word id, Type p, int b) { super(id, p); offset = b; }
}
```

If.java

```
package inter;               // Arquivo If.java
import symbols.*;
public class If extends Stmt {
        Expr expr; Stmt stmt;
        public If(Expr x, Stmt s) {
                expr = x; stmt = s;
                if( expr.type != Type.Bool ) expr.error("boolean required in if");
        }
        public void gen(int b, int a) {
                int label = newlabel();

                expr.jumping(0, a);    // segue se for true, vai para a se for false
                emitlabel(label); stmt.gen(label, a);
```

```
        }
}
```

Logical.java

```
package inter;              // Arquivo Logical.java
import lexer.*; import symbols.*;
public class Logical extends Expr {
        public Expr expr1, expr2;
        Logical(Token tok, Expr x1, Expr x2) {
                super(tok, null);               // tipo nulo para começar

                expr1 = x1; expr2 = x2;
                type = check(expr1.type, expr2.type);
                if (type == null ) error("type error");
        }
        public Type check(Type p1, Type p2) {
                if ( p1 == Type.Bool && p2 == Type.Bool ) return Type.Bool;
                else return null;
        }
        public Expr gen() {
                int f = newlabel(); int a = newlabel();
                Temp temp = new Temp(type);
                this.jumping(0,f);
                emit(temp.toString() + " = true");
                emit("goto L" + a);
                emitlabel(f); emit(temp.toString() + " = false");
                emitlabel(a);
                return temp;
        }
        public String toString() {
                return expr1.toString()+" "+op.toString()+" "+expr2.toString();
        }
}
```

Node.java

```
package inter;              // Arquivo Node.java
import lexer.*;
public class Node {
        int lexline = 0;
        Node() { lexline = Lexer.line; }
        void error(String s) { throw new Error("near line "+lexline+": "+s); }
        static int labels = 0;
        public int newlabel() { return ++labels; }
        public void emitlabel(int i) { System.out.print("L" + i + ":"); }
        public void emit(String s) { System.out.println("\t" + s); }
}
```

Not.java

```
package inter;              // Arquivo Not.java
import lexer.*;


public class Not extends Logical {
        public Not(Token tok, Expr x2) { super(tok, x2, x2); }
        public void jumping(int t, int f) { expr2.jumping(f, t); }
        public String toString() { return op.toString()+" "+expr2.toString(); }
}
```

Op.java

```
package inter;              // Arquivo Op.java
import lexer.*; import symbols.*;
public class Op extends Expr {
        public Op(Token tok, Type p) { super(tok, p); }
        public Expr reduce() {
                Expr x = gen();
                Temp t = new Temp(type);
                emit( t.toString() + " = " + x.toString() );
                return t;
        }
}
```

Or.java

```
package inter;              // Arquivo Or.java
import lexer.*;

public class Or extends Logical {
        public Or(Token tok, Expr x1, Expr x2) { super(tok, x1, x2); }
        public void jumping(int t, int f) {
                int label = t != 0 ? t : newlabel();
                expr1.jumping(label, 0);
                expr2.jumping(t,f);
                if( t == 0 ) emitlabel(label);
        }
}
```

Rel.java

```
package inter;              // Arquivo Rel.java
import lexer.*; import symbols.*;
public class Rel extends Logical {
        public Rel(Token tok, Expr x1, Expr x2) { super(tok, x1, x2); }
        public Type check(Type p1, Type p2) {
                if ( p1 instanceof Array || p2 instanceof Array ) return null;
                else if( p1 == p2 ) return Type.Bool;
                else return null;
        }
        public void jumping(int t, int f) {
                Expr a = expr1.reduce();
                Expr b = expr2.reduce();
                String test = a.toString() + " " + op.toString() + " " + b.toString();
                emitjumps(test, t, f);
        }
}
```

Seq.java

```
package inter;              // Arquivo Seq.java
public class Seq extends Stmt {
        Stmt stmt1; Stmt stmt2;
        public Seq(Stmt s1, Stmt s2) { stmt1 = s1; stmt2 = s2; }
        public void gen(int b, int a) {
                if ( stmt1 == Stmt.Null ) stmt2.gen(b, a);
                else if ( stmt2 == Stmt.Null) stmt1.gen(b, a);
                else {
                        int label = newlabel();
                        stmt1.gen(b,label);
                        emitlabel(label);
                        stmt2.gen(label,a);
                }
        }
}
```

Set.java

```
package inter;              // Arquivo Set.java
import symbols.Type;
public class Set extends Stmt {
        public Id id; public Expr expr;
        public Set(Id i, Expr x) {
                id = i; expr = x;
                if ( check(id.type, expr.type) == null ) error("type error");
        }
        public Type check(Type p1, Type p2) {
                if ( Type.numeric(p1) && Type.numeric(p2) ) return p2;
                else if ( p1 == Type.Bool && p2 == Type.Bool ) return p2;
                else return null;
        }
        public void gen(int b, int a) {
                emit( id.toString() + " = " + expr.gen().toString() );
        }
}
```

SetElem.java

```
package inter;              // Arquivo SetElem.java
import symbols.Array;
import symbols.Type;
public class SetElem extends Stmt {
        public Id array; public Expr index; public Expr expr;
        public SetElem(Access x, Expr y) {
                array = x.array; index = x.index; expr = y;
                if ( check(x.type, expr.type) == null ) error("type error");
        }
        public Type check(Type p1, Type p2) {
                if ( p1 instanceof Array || p2 instanceof Array ) return null;
                else if ( p1 == p2 ) return p2;
                else if ( Type.numeric(p1) && Type.numeric(p2) ) return p2;
                else return null;
        }
        public void gen(int b, int a) {
                String s1 = index.reduce().toString();
                String s2 = expr.reduce().toString();
                emit(array.toString() + " [ " + s1 + " ] = " + s2);
        }
}
```

Stmt.java

```
package inter;              // Arquivo Stmt.java
public class Stmt extends Node {

        public Stmt() { }

        public static Stmt Null = new Stmt();
        public void gen(int b, int a) {}

        int after = 0;

        public static Stmt Enclosing = Stmt.Null; // usado para comandos break
}
```

Temp.java

```
package inter;              // Arquivo Temp.java
```

```
import lexer.*; import symbols.*;
public class Temp extends Expr {
        static int count = 0;
        int number = 0;
        public Temp(Type p) { super(Word.temp, p); number = ++count; }
        public String toString() { return "t" + number; }
}
```

Unary.java

```
package inter;              // Arquivo Unary.java
import lexer.*; import symbols.*;
public class Unary extends Op {
        public Expr expr;
        public Unary(Token tok, Expr x) {    // trata operador menos, para !, ver Not
                super(tok, null); expr = x;
                type = Type.max(Type.Int, expr.type);
                if (type == null ) error("type error");
        }
        public Expr gen() { return new Unary(op, expr.reduce()); }
        public String toString() { return op.toString()+" "+expr.toString(); }
}
```

While.java

```
package inter;              // Arquivo While.java
import symbols.*;
public class While extends Stmt {
        Expr expr; Stmt stmt;
        public While() { expr = null; stmt = null; }
        public void init(Expr x, Stmt s) {
                expr = x; stmt = s;
                if( expr.type != Type.Bool ) expr.error("boolean required in while");
        }
        public void gen(int b, int a) {
                after = a;

                expr.jumping(0, a);
                int label = newlabel();

                emitlabel(label); stmt.gen(label, b);
                emit("goto L" + b);
        }
}
```

6.3 TP3

Quadruplas.hpp

```
/*
 * Quadruplas.hpp
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */


#ifndef QUADRUPLA_H_
#define QUADRUPLA_H_


#include <iostream>
```

```
#include <string.h>
#include <stdlib.h>
#include <vector>

using namespace std;

#define NUM_OPERANDOS        3
#define VAZIO                "\0"

class Quadrupla
{
public:
        string op;                      /* operador */

//      op, r, n, d
//O numero op (0 a 15) define o
//codigo da operacao. O numeror (0 a 15) de ne o registrador a ser usado, enquanto n frequentemente (0 a
//255) de ne o tamanho do operando. Por ultimo, d (-32768 a 32767) usualmente de ne um deslocamento
//de um endereco (possivelmente negativo) d e r de nem em conjunto o endereco do operando (d +
//registrador r ). Alem disso, comentarios sao iniciados com caractere ' ' e terminam no nal da linha. Um
//exemplo de um arquivo de instrucoes e mostrado abaixo.

        string ops[NUM_OPERANDOS];          /* operandos 0, 1 e 2 */
public:
        Quadrupla();                    /* construtor */
        Quadrupla( string op_enter, string a, string b, string c); /* construtor */

        string get_operador( void );        /* retorna operador */

        void set_operador( string novo );       /* muda operador para novo */

        string get_operado_pos( int pos );       /* retorna operando pos */

        void set_ops( int pos, string v );      /* muda valor do operando pos para v */

        Quadrupla stringToQuadrupla( string s );    /* retorna a quadrupla representada por s */

        void imprimeQuadrupla();
};

class Quadruplas{
public:
        vector<Quadrupla> quadruplas;

        Quadruplas();

        Quadrupla getQuadrupla( int pos );
        int getNumQuadruplas();
        void imprimeQuadruplas();

        void addQuadrupla(Quadrupla q);
};



#endif
```

Quadruplas.cpp

```
/*
 * Quadruplas.cpp
 *
```

```
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */

#include "quadruplas.hpp"

/* construtor */
Quadrupla::Quadrupla(){

}

/* construtor */
Quadrupla::Quadrupla( string op_enter, string a, string b, string c)
: op(op_enter){
	cout <<"Quadrupla " << op_enter << "  " << a << "  " << b << "  " << c << endl ;
	ops[0] = a;
	ops[1] = b;
	ops[2] = c;
}

/* retorna operador */
string Quadrupla::get_operador( void ){
	return op;
}

/* muda operador para novo */
void Quadrupla::set_operador( string novo ){
	op = novo;
}

/* retorna operando pos */
string Quadrupla::get_operado_pos( int pos ){
	return ops[pos];
}

/* muda valor do operando pos para v */
void Quadrupla::set_ops( int pos, string v ){

	if(pos < 0 || pos > 3)
		cout <<"ERRO : Quadrupla::set_ops() : indice invalido" << endl;
	else
		ops[pos] = v;
}

/* retorna a quadrupla representada por s */
Quadrupla Quadrupla::stringToQuadrupla( string s ){

	cout << "linha = " << s << endl;
	size_t found;
	found=s.find_first_of(":");

//	procuro por ":"
	if ( found != string::npos )
	{
		int first_letter = 0;

		while(s[first_letter] == '\t' || s[first_letter] == ' ' ) first_letter++;
		return Quadrupla( s.substr(first_letter, found ), VAZIO, VAZIO, VAZIO);
	}
	else{
		//			op, arg1, arg2, result
		//maximo de palavras possiveis numa instrucao
		string op[6];
```

```
for(int i = 0; i < 6; i++) op[i] = VAZIO;

int op_index = 0;


char empty = ' ';
unsigned int index = 0;

//vou ate o primeiro caracter da quadrupla
while( s[index] == empty || s[index] == '\t' || s[index] == 13) index++;

int first_index = index;
for (; index < s.length(); index++){

	if( s[index] == empty || index == s.length() - 1 || s[index] == '\t' ){

		if(s[index] == empty)
			op[op_index] = s.substr(first_index, index - first_index  );
		else if(index == s.length() - 1)
			op[op_index] = s.substr(first_index, index - first_index  + 1);

		cout << " OP " << op_index << " :: " << op[op_index] << ";" << endl;
		op_index ++;

		if(index == s.length() - 1|| s[index] == '\n' || s[index] == EOF)
			break;

		while(s[index] == empty || s[index] == '\t' || s[index] == 13){
			index++;
		}

		first_index = index;

		index --;
	}
}

//atualizo o marcador do inicio da proxima quadrupla
//iffalse x < y goto L5
if( op[0].compare("iffalse") == 0){
	if(op_index == 6)
		return Quadrupla( op[2], op[1], op[3], op[5]);
	//os dois abaixo NAO TENHO CERTEZA ERRO
	//iffalse a goto L6
	else if(op_index == 4)
		return Quadrupla( "!=", op[1], "0", op[3]);
	else{
		cout << "QUADRUPLA : " << s << "          NAO EXISTENTE" << endl;
		exit(0);
	}
}
else if(op[0].compare("if") == 0 ){
	if(op_index == 6) //iffalse i >= j goto L8
		return Quadrupla( op[2], op[1], op[3], op[5]);
	else if(op_index == 4)
		return Quadrupla( "==", op[1], "1", op[3]);
	else{
		cout << "QUADRUPLA : " << s << "          NAO EXISTENTE" << endl;
		exit(0);
	}
}
else if(op[0].compare("goto") == 0)
```

```
                return Quadrupla( op[0], op[1], VAZIO, VAZIO);
        //              x = t [ t1 ]
        else if(op[3].compare("[") == 0 && op[5].compare("]") == 0)
                return Quadrupla( "LOAD", op[2], op[4], op[0]);
        //              x [ t1 ] = t
        else if(op[1].compare("[") == 0 && op[3].compare("]") == 0)
                return Quadrupla( "STORE", op[2], op[5], op[0]);
        //a = a + 1
        else if(op_index == 5)
                return Quadrupla( op[3], op[2], op[4], op[0]);
        //a = minus 1
        else if(op_index == 4) // minus, 1, VAZIO, a
                return Quadrupla( op[2], op[3], VAZIO, op[0]);
        //a = t5
        else if(op_index == 3)
                return Quadrupla( op[1], op[2], VAZIO, op[0]);
        else{
                cout << "QUADRUPLA : " << s << "         NAO EXISTENTE" << endl;
                exit(0);
        }

    }
    return Quadrupla();

}

void Quadrupla::imprimeQuadrupla(){
//      cout <<"op                      arg1                    arg2                    result" << ENDL;
        cout << op << "                 " << ops[0] << "                " << ops[1] << "                " <<
ops[2] << endl;
}

/
*=========================================================================================
=============*/

Quadruplas::Quadruplas()
{
        quadruplas.resize(0);
}

Quadrupla Quadruplas::getQuadrupla( int pos )
{
        return quadruplas[pos];
}

void Quadruplas::addQuadrupla( Quadrupla q )
{
        quadruplas.push_back(q);
}

int Quadruplas::getNumQuadruplas()
{
        return quadruplas.size();
}

void Quadruplas::imprimeQuadruplas()
{
        Quadrupla q;
        cout << "Impressao das quadruplas" << endl;
        cout << "----------------------" << endl;
        cout <<"OP                      ARG_1                   ARG_2                   RESULT" << endl;
        for ( unsigned int i = 0; i < quadruplas.size(); i++ )
```

```
	{
		q = quadruplas[i];
		q.imprimeQuadrupla();
	}
	cout << "---------------------" << endl << endl;
}
```

Tabela.cpp

```
/*
 * tabela.cpp
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */

#include "tabela.hpp"

Tabela::Tabela()
{
	L = 0;	/* Considera-se que a primeira posicao da tabela é a de indice 0.
							L é o final da arvore binaria */
	Raiz = -1;

	nivel = 1;		/* O primeiro nivel  o 1 */

	escopo[nivel] = 0;	/* escopo[1] contem o indice do primeiro elemento */


}

/*****************************************************************************

Implementacoes

*****************************************************************************/

/************  Funcao que define os erros provaveis de ocorrer  **********/

void Tabela::Erro(int num)
{
	//		char opcao;

	switch (num) {
	case 1: printf("Tabela de Simbolos cheia\n");		break;

	case 2: //printf("Este item nao foi encontrado\n");
		break;

	case 3: printf("Este item ja foi inserido\n"); break;

	}

}

/*******************	Funcao de entrada num bloco	*********************/
void Tabela::Entrada_Bloco()
{
	// Limpa a tela
	//		clrscr();

	nivel++;
```

```
	if (nivel > NMax)
		Erro(1);

	else escopo[nivel] = L;

	printf("\nEntrada no nivel  %d\n\n",nivel);
}

/*********************  Funcao de saida de um bloco  ************************/
void Tabela::Saida_Bloco()
{

	int i;

	L= escopo[nivel];

	if (Raiz >= L)

		Raiz =0;

	else for (i=0;i<=(L-1); i++)

	{

		if (TabelaS[i].esq >= L)

			TabelaS[i].esq = -1;

		if (TabelaS[i].dir >= L)

			TabelaS[i].dir = -1;

	}

	printf("\nSaida do nivel %d",nivel);

	nivel--;
}

/*****************  Funcao que pesquisa item na tabela  *******************/
int Tabela::Get_Entry(const char * x)   /* Pesquisa o simbolo "x" e retorna o indice
                                                da Tabela de simbolos onde ele se encontra */
{
	int S, K, aux;
	S = Raiz;
	K = -1;

	while (S != -1)
	{
		aux = strcmp(TabelaS[S].nome,x);

		if (aux == 0)
		{
			K = S;
			S = TabelaS[S].dir;
		}

		else if (x < TabelaS[S].nome)	S = TabelaS[S].esq;

		else S = TabelaS[S].dir;

	}
```

```
	if (K != -1) /*Verificar se é -1 ou 0 */
	{
//		printf("O item esta no nivel  %d\n", TabelaS[K].nivel);
//		printf("		Indice %u\n",K);
	}

	else{
		Erro(2);
		return -1;
	}

	return (K);  /* Retorna o indice no vetor TabelaS do elemento procurado*/

}

char * Tabela::PegaAtributo(int indice){
	return TabelaS[indice].atributo;
}


/***************  Funcao que instala o item na tabela  *****************/
/* Instala o simbolo "X" com o atributo atribut na Tabela de Simbolos */
void Tabela::Instala(const char * X, const char * atribut)
{
	//cout << " Instala : " << X << "		: atrib :: " << atribut << endl;
//	getchar();
	int S, i, k, aux;

	//	clrscr();

	S = Raiz;

	i = Raiz;

	k = -1;

	while (S != -1)   /* Enquanto nao achar um nodo folha  */
	{

		i=S;

		aux = strcmp(TabelaS[S].nome,X);

		if (aux == 0)

		{
			k = S;

			S = TabelaS[S].dir;

			if (i >= escopo[nivel]) Erro(3);

		}

		else if ( X < TabelaS[S].nome )		S = TabelaS[S].esq;
		else 					S = TabelaS[S].dir;
	}

	if ( (k > escopo[nivel] ) || ( L >= NMax + 1 ) )	Erro(1);
	else
	{
		TabelaS[L].nivel = nivel;
```

```
                aux = strlen(atribut);

                for (k = 0; k<= aux-1; k++)
                        TabelaS[L].atributo[k] = atribut[k];
                for(; k < 10; k++ )
                        TabelaS[L].atributo[k] = '\0';

                TabelaS[L].esq = TabelaS[L].dir = -1;

                aux = strlen(X);

                for (k = 0; k<=(aux-1); k++)

                        TabelaS[L].nome[k] = X[k];

                if (Raiz == -1) Raiz= L;

                else if (X < TabelaS[i].nome)

                        TabelaS[i].esq = L;

                else TabelaS[i].dir = L;
                L++;
        }
}

void Tabela::imprimir()
{

        int i;

        //      clrscr();

        for (i=0; i<=L-1; i++)

        {

                printf("\n\nNome : ");

                printf("%s", TabelaS[i].nome);

                printf("\n");

                printf("Atributo : ");

                printf("%s", TabelaS[i].atributo);

                printf("\n");

                printf("Nivel : ");

                printf("%i", TabelaS[i].nivel);

                printf("\n");

                printf("Esquerdo : ");

                if (TabelaS[i].esq == -1) printf("0");

                else printf("%i", TabelaS[i].esq);

                printf("\n");
```

```
                printf("Direito : ");
                if (TabelaS[i].dir == -1) printf("0");
                else printf("%i", TabelaS[i].dir);
                printf("\n");
                printf("\n"); }
}
```

Tabela.hpp

```
/*
 * tabela.h
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */


#ifndef TABELA_H_
#define TABELA_H_

#include <string.h>
#include <stdio.h>
#include <string.h>
#include <iostream>


using namespace std;

#define NMax 10000     /* Numero maximo de niveis possiveis */

struct Arvore_Tabela{
        char nome[10];     /* Contem o nome do Simbolo */
        int  nivel;        /* Contem o nivel do Simbolo relacionado */
        char atributo[10]; /* Contem o atributo do  relacionado */
        int  esq;          /* Filho da esquerda do simbolo relacionado */
        int  dir;      /* Filho da direita do simbolo relacionado */
};

class Tabela{

private:
        Arvore_Tabela TabelaS[100];     /* Vetor de struct que contem a tabela de simbolos */

public:
        int escopo[10];
        int nivel;    /* inteiro que contem o numero do nivel atual */
        int L;     /* inteiro que contem o indice do ultimo elemento da Tabela de Simbolos */
```

```
        int Raiz;   /* inteiro que contem o indice do primeiro elemento da Tabela de Simbolos */

        Tabela();

        void Entrada_Bloco(void);

        void Erro(int numero);

        void Saida_Bloco(void);

        int  Get_Entry(const char * x);

        void Instala(const char *name, const char * atributo);

        char * PegaAtributo(int indice);

        void imprimir(void);

};

#endif
```

Tradutor.hpp

```
/*
 * tradutor.hpp
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */


#ifndef TRADUTOR_H_
#define TRADUTOR_H_

#include <map>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <vector>
using namespace std;

#include "quadruplas.hpp"
#include "tabela.hpp"


#define MAX_DATA 10000
#define MAX_ARRAY_SIZE_INT 100
#define MAX_ARRAY_SIZE_CHAR "100"


class Tradutor {

private:

        Tabela tabela;
        string instrucao;
        vector<string> instrucoesTAM;

        int SB ;
        int ST, LB, LT;
```

```
//		Acompanha o codigo = numero de instrucoes ate o momento
		int CT ;

//		map<string, string (*)(Quadrupla &q, Tabela &t)> opcodes;

public:

		Tradutor();

		string getInstrucao();

		void setInstrucao( string s );

		void traduzQuadruplas( Quadruplas Q );

		void traduzQuadrupla( Quadrupla q );

		void adicionaInstrucao( string inst );

		void imprimeProgramaTAM( string file_name );

		string operacaoTAM(string op);

		string Int_to_String(int i);

		void adicionaVariavel(string operador, int tamanho_enter, bool aumenta_pilha);
};


#endif
```

Tradutor.cpp

```
/*
 * tradutor.cpp
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */

#include "tradutor.hpp"
#include <fstream>

void Tradutor::imprimeProgramaTAM( string file_name )
{
		ofstream output( file_name.c_str(), ios::out );
		cout << "\n\nCT		Impressao do Programa TAM                                                    " << endl;
		cout << "-------------------------" << endl;

		for ( unsigned int i = 0; i < instrucoesTAM.size(); i++ )
		{
			cout << i << "		"<< instrucoesTAM[i] << endl;
			output << instrucoesTAM[i] ;
			if( instrucoesTAM[i].compare("HALT") == 0) break;
		}

		cout << "-------------------------" << endl;
		output.close();
}

Tradutor::Tradutor()		:
			instrucoesTAM(MAX_DATA)
```

```
		{
	SB = 0;
	CT = 0; // numero de instrucoes ate o momento
	ST = 0;
		}

string Tradutor::getInstrucao(){
	string saida;
	return saida;
}

void Tradutor::setInstrucao( string s ){

}

void Tradutor::traduzQuadruplas( Quadruplas Q ){
	tabela.Entrada_Bloco();
	//		Para toda quadrupla
	//cout << "Traduz Quadruplas" << endl;
	for(unsigned int i = 0 ; i < Q.quadruplas.size(); i++){
		traduzQuadrupla(Q.quadruplas[i]);
	}

	Quadrupla finaliza(VAZIO, VAZIO, VAZIO, VAZIO);
	traduzQuadrupla(finaliza);
	//corrigindo labels pendentes
	int c; string cend;
	for ( unsigned int i = 0; i < instrucoesTAM.size(); i++ ){
		string inst = instrucoesTAM[i];

		if(inst[0]  == 'A' ){ //se for label pendente
			//se A pendentede em JUMPIF
			//se B pendente em GOTO
			inst[0] = 'L';
			if((c = tabela.Get_Entry(inst.c_str()) )!= -1){
				cend = tabela.PegaAtributo(c);

				instrucoesTAM[i] = "JUMPIF (1) " + cend + " [CB]";

				cout << "NOVA INST1: " << instrucoesTAM[i] << endl;

			}
		}
		else if(inst[0]  == 'B' ){ //se for label pendente
			inst[0] = 'L';
			//se A pendentede em JUMPIF
			//se B pendente em GOTO
			if((c = tabela.Get_Entry(inst.c_str()) )!= -1){
				cend = tabela.PegaAtributo(c);
				instrucoesTAM[i] = "goto " + cend ;
					cout << "NOVA INST2: " << instrucoesTAM[i] << endl;

			}
		}

	}

	tabela.Saida_Bloco();
}


void Tradutor::traduzQuadrupla( Quadrupla q ){
	int a, b, c;
```

```
	string aend, bend ,cend;

//	q.imprimeQuadrupla() ;

	if(q.op.find_first_of("+*-/=") != string::npos || q.op.compare("minus") == 0){
		//cout << " TraduzQuadrupla 1" << endl;
		//			b = 1 operador = ops[0]
		if((b = tabela.Get_Entry(q.ops[0].c_str() )) != -1){
			bend = tabela.PegaAtributo(b);
			adicionaInstrucao("LOAD (1) " + bend);
		}
		else{
			//coloco 0 na pilha
			b = atoi( q.ops[0].c_str() );
			if(b > 0)
				adicionaInstrucao("LOADL " + Int_to_String(b));
			else
				adicionaInstrucao("LOADL 0");
		}
		//			ops[1] = 2 operador
		if((c = tabela.Get_Entry(q.ops[1].c_str())) != -1){
			cend = tabela.PegaAtributo(c);
			adicionaInstrucao("LOAD (1) " + cend + " [SB]");
		}
		else{
			c = atoi(q.ops[1].c_str());
			if(c > 0)
				adicionaInstrucao("LOADL " + Int_to_String(c));
			else	//coloco 0 na pilha
				adicionaInstrucao("LOADL 0");
		}

		//Se nao se encontra na tabela entao adiciono-o, somente para o termo que recebe o Resultado
		adicionaInstrucao("CALL " + operacaoTAM(q.get_operador()) );

		//			Segundo operado
		if((a = tabela.Get_Entry(q.ops[2].c_str())) != -1){
			aend = tabela.PegaAtributo(a);
			adicionaInstrucao("STORE (1) " + aend + " [SB]");
		}
		else{
			adicionaVariavel(q.ops[2], 1, false);
		}
	}
	//if if i < j goto l , ou ifelse ..
	else if(q.op.compare("!=") == 0 || q.op.compare("==") == 0 ||
			q.op.compare("<=") == 0 ||q.op.compare("<") == 0 || q.op.compare(">") == 0 ||
q.op.compare(">=") == 0 ){
		//cout << " TraduzQuadrupla 2" << endl;
		//operador i < j
		//i
		if((a = tabela.Get_Entry(q.ops[0].c_str() )) != -1){
			aend = tabela.PegaAtributo(a);
			adicionaInstrucao("LOAD (1) " + aend + " [SB]");
		}
		else{
			//coloco 0 na pilha
			a = atoi(q.ops[0].c_str());
			if( a  > 0)
				adicionaInstrucao("LOADL " + Int_to_String(a));
			else
				adicionaInstrucao("LOADL 0");
```

```
		}

		//j
		if((b = tabela.Get_Entry(q.ops[1].c_str())) != -1){
			bend = tabela.PegaAtributo(b);
			adicionaInstrucao("LOAD (1) " + bend + " [SB]");
		}
		else{
			//coloco 0 na pilha
			b = atoi(q.ops[1].c_str());
			if(b > 0)
				adicionaInstrucao("LOADL " + Int_to_String(b));
			else
				adicionaInstrucao("LOADL 0");
		}

		//Se nao se encontra na tabela entao adiciono-o, somente para o termo que recebe o Resultado
		//operacao da condicao
		adicionaInstrucao("CALL " + operacaoTAM(q.get_operador()) );

		//ops[2] = endereco da label do desvio
		//endereco do goto esta em c
		if((c = tabela.Get_Entry(q.ops[2].c_str())) != -1){

			cend = tabela.PegaAtributo(c);
//			cout << "LABEL : " << q.ops[2] << "		c :"<< cend << endl;
			adicionaInstrucao("JUMPIF (1) " + cend + " [CB]");

		}
		else{ //para as labels ainda nao definidas, nao seto essa instrucao
			string inst = q.ops[2];
			inst[0] = 'A';
			adicionaInstrucao( inst);

//			cout << "ERRO LABEL : " << c << " INDEFINIDA " << endl;
			//			char * atributo ;
			//			sprintf (atributo, "%d", ST);
			//			ST++;
			//			//pega atributo
			//			tabela.Instala(q.ops[2].c_str(), atributo);
		}
	}
	else if(q.op.compare("goto") == 0){
		//cout << " TraduzQuadrupla 3" << endl;
		//pego o endereco do label do goto = ops[0]
		if((a = tabela.Get_Entry(q.ops[0].c_str())) != -1){
			aend = tabela.PegaAtributo(a);
			adicionaInstrucao("LOADL " + aend);
			adicionaInstrucao("JUMPI" );
		}
		else{
			string inst = q.ops[0];
			inst[0] = 'B';
			adicionaInstrucao( inst);

//			cout << "ERRO LABEL : " << inst << " INDEFINIDA" <<endl;
		}
	}
	//			x = t [ t1 ]
	//  else if(op[3].compare("[") == 0 && op[5].compare("]"))
	//		return Quadrupla( "LOAD", op[2], op[4], op[0]);
	//op[0] = t
	//op[1] = t1
```

```
	//op[2] = x
	else if(q.op.compare("LOAD") == 0){
		//cout << " TraduzQuadrupla 5" << endl;
		//cout << " TraduzQuadrupla 5.1" << endl;

		int endereco = 0 ;
		//t1
		if((b = tabela.Get_Entry(q.ops[1].c_str())) != -1){
			//cout << " TraduzQuadrupla 5.2" << endl;
			bend = tabela.PegaAtributo(b);
			//somo t + t1
			endereco += atoi(bend.c_str());
		}

		//t
		if((a = tabela.Get_Entry(q.ops[0].c_str())) != -1){
			aend = tabela.PegaAtributo(a);
			endereco = atoi(aend.c_str());
			//cout << "End.:" << endereco << endl;
		}
		else{ //adiciono a variavel na tabela
			adicionaVariavel(q.ops[0], MAX_ARRAY_SIZE_INT, true);
		}

		//cout << " TraduzQuadrupla 5.0" << endl;
		adicionaInstrucao("LOAD ( 1 ) " + Int_to_String(endereco) + " [SB]");
		//cout << " TraduzQuadrupla 5.1" << endl;

		//x
		//cout << " TraduzQuadrupla 5.3" << endl;
		if((c = tabela.Get_Entry(q.ops[2].c_str())) != -1){
			//cout << " TraduzQuadrupla 5.2 : " <<  c << endl;
			cend = tabela.PegaAtributo(c);
			adicionaInstrucao("STORE (1) " +  cend + " [SB]");
			//cout << " TraduzQuadrupla 5.4" << endl;
		}
		else{ //adiciono a variavel na tabela
			//cout <<"OP = " << q.ops[2] << endl;
			//cout << " TraduzQuadrupla 5.5" << endl;
			adicionaVariavel(q.ops[2], 1, false);
		}

		//cout << " TraduzQuadrupla 5 SAIU" << endl;
	}
	else if(q.op.find_first_of("L")  != string::npos){ //se for label

		string atributo ;
		atributo = Int_to_String(CT);

		//marco a localizacao como atributo da label
		tabela.Instala(q.op.c_str(), atributo.c_str());
	}
	//			x [ t1 ] = t
	// return Quadrupla( "STORE", op[2], op[5], op[0]);
	else if(q.op.compare("STORE") == 0){
		//ops[0] = t1
		//ops[1] = t
		//ops[2] = x
		//cout << " TraduzQuadrupla 6 " << endl;
		//ops[1] = t
		if((c = tabela.Get_Entry(q.ops[1].c_str())) != -1){
			cend = tabela.PegaAtributo(c);
//			cout << " CEND : " << cend << "			c : " << c<< endl;
```

```
                        adicionaInstrucao("LOAD (1)" + cend + " [SB]");
                }
                else adicionaInstrucao("LOADL 0" );
                //ops[2] = x
                int endereco = 0;
                if((a = tabela.Get_Entry(q.ops[2].c_str())) != -1){

                        aend = tabela.PegaAtributo(a);
                        endereco = atoi(aend.c_str());
                }
                else{ //adiciono a variavel na tabela
                        //GRAVO X
                        adicionaVariavel(q.ops[2], MAX_ARRAY_SIZE_INT, true);
                }

                //ops[0] = t1
                if((b = tabela.Get_Entry(q.ops[0].c_str())) != -1){
                        bend = tabela.PegaAtributo(b);
                        endereco += atoi(bend.c_str());
                }
                else{ //adiciono a variavel na tabela
                        adicionaVariavel(q.ops[0], 1, true);
                }
                adicionaInstrucao("STORE (1) " + Int_to_String(endereco) + " [SB]" );
        }
        else if((unsigned int )q.op.compare(VAZIO) == 0){ //se chegar uma quadrupla com op vazia significa que as
operacoes
                //acabaram
                adicionaInstrucao("HALT" );
        }
        else
                cout << "Instrucao nao encontradao" << endl;


}

void Tradutor::adicionaVariavel(string operador, int tamanho_enter, bool aumenta_pilha){
        //cout << "Adiciona Variavel " << endl;
        string tamanho = Int_to_String(tamanho_enter);

        if(aumenta_pilha == true)
                adicionaInstrucao("PUSH " + tamanho);
        string atributo = Int_to_String(ST);
        //cout << "Instala 2 : ST : " << ST << "                    : tamanho : " << tamanho_enter << endl;
        ST += tamanho_enter;
        //gravo t1

        tabela.Instala(operador.c_str(), atributo.c_str());

        adicionaInstrucao("STORE (" + tamanho +  ") " +  atributo + " [SB]");
};

string Tradutor::Int_to_String(int i){

        std::string s;
        std::stringstream out;
        out << i;
        s = out.str();

        return s;
}
```

```
string Tradutor::operacaoTAM(string op){
	string saida;
	if(op.compare(">=") == 0) return "ge";

	else if(op.compare("==") == 0) return "eq";

	else if(op.compare("!=") == 0) return "ne";

	else if(op.compare("<=") == 0) return "le";

	else if(op.compare("<") == 0) return "lt";

	else if(op.compare(">") == 0) return "gt";

	else if(op.compare("*") == 0) return "mult";

	else if(op.compare("/") == 0) return "div";

	else if(op.compare("+") == 0) return "succ";

	else if(op.compare("-") == 0) return "sub";

	else if(op.compare("minus") == 0) return "neg";

	else return "ERROR_OPERADOR";

}

void Tradutor::adicionaInstrucao( string inst ){
	//		conta a adicao da nova instrucao
	//cout << "Instrucao : "<< inst << endl;

	instrucoesTAM[CT] = inst ;

	CT++;


}
```

Main.cpp

```
/*
 * main.cpp
 *
 *  Created on: 30/11/2009
 *      Author: rogerfsg
 */

#include "quadruplas.hpp"
#include "tabela.hpp"
#include "tradutor.hpp"

#include <fstream>

#include <stdlib.h>
#include <stdio.h>
using namespace std;

void leArqEntrada( string file_name, Quadruplas &Q )
{
   string s;
   Quadrupla q;
```

```
   ifstream input( file_name.c_str(), ios::in );

   if ( !input )
   {
      cout << "O arquivo " << file_name << " nao existe. O programa terminara agora!" << endl;
      exit(1);
   }
   else
   {

      while ( getline( input, s ) )
      {

                              size_t found;

            found = s.find_first_of(":");
            int first_index = 0;
            bool validade = false;

            while (found < string::npos)
            {
              string palavra = s.substr(first_index, found - first_index + 1);

              q = q.stringToQuadrupla( palavra );
              Q.addQuadrupla( q );

              //atualizo o marcador do inicio da proxima quadrupla
              first_index = found + 1;
              found = s.find_first_of(":",found + 1);

              validade = true;
            }
            if(!validade){
                  if(s.length() > 0 ){
                                          //gera a quadrupla com a lnha inteira
                                          q = q.stringToQuadrupla( s );
                                          Q.addQuadrupla( q );
                  }
            }
      }
   }
   input.close();
}

int main( int argc, char *argv[] )
{
  string file_name_in;
  string file_name_out;
  Tradutor T;
  Quadruplas Q;
  if( argc < 3 )
  {
      cout << "Escreva o nome do arquivo de entrada: " << endl;
      cin >> file_name_in;
      cout << "Escreva o nome do arquivo de saida: " << endl;
      cin >> file_name_out;
  }
  else
  {
      file_name_in = argv[1];
      file_name_out = argv[2];
  }
```

```
  string interpretador = "java teste.Main < " + file_name_in + " > Codigo_Intermediario";
  cout << "Interpretador : " << interpretador << endl;
  system (interpretador.c_str());

  leArqEntrada( "Codigo_Intermediario", Q );
  cout << endl;
  Q.imprimeQuadruplas();

  T.traduzQuadruplas( Q );
  T.imprimeProgramaTAM( file_name_out );
  cout << endl;
  return 0;
}
```