Processamento de Linguagens (3º ano de Curso)

Trabalho Prático Nº2

Relatório de Desenvolvimento

Tiago Conceição (55171) Ricardo Costa (61012)
Tiago Oliveira (61083)

5 de Junho de 2015

Resumo

Este trabalho tem como objectivo a definição de uma linguagem iterativa simples e a criação do seu compilador - com recurso às ferramentas *Flex* e *Yacc* - que gera código para uma máquina de stack virtual.

Os objectivos de aprendizagem foram alcançados, havendo agora uma muita maior compreensão das gramáticas independentes de contexto com condição LR().

Dos requisitos propostos no enunciado, todos foram alcançados com bastante precisão.

Conteúdo

Capítulo 1

Introdução

Neste trabalho, foram resolvidos todos os problemas propostos com ainda bastante tempo de sobra até ao dia da entrega. Naturalmente, pretendemos incluir algumas funcionalidades adicionais, como já havíamos feito no trabalho anterior.

Após uma conversa entre o grupo e o docente Pedro Henriques e este ter visto que o trabalho estava já bastante adiantado, sugeriu que implementássemos funções na nossa linguagem.

Nós revelamos também intenção de incluir números de vírgula flutuante. No entanto, apontamos que a sobrecarga de trabalhos nesta altura do semestre é grande e que provavelmente não iríamos ter tempo para realizar estas tarefas. E foi o que aconteceu.

Nota: pessoalmente (Tiago Conceição), pondero propôr uma discussão com a direcção de curso sobre o estado actual do curso, em particular, no 3º ano e no que se refere á quantidade e tamanho dos trabalhos e também á pertinência de alguns deles.

Estrutura do Relatório

O relatório encontra-se dividido em 3 capítulos:

> Análise e Especificação onde é feita uma descrição informal do problema seguida da especificação dos requisitos necessários á sua resolução;
>
> Concepção/desenho da Resolução onde se apresentam as estruturas de dados utilizadas e ainda a estrutura dos algoritmos a conceber;
>
> Testes realizados e Resultados onde se incluem alguns exemplos de input e respectivo output.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Pretende-se que o compilador seja capaz de construir ficheiros de texto com pseudo-código *assembly* para a uma máquina de stack virtual. Os inputs do compilador são programas escritos numa linguagem imperativa simples, também esta a definir.

Então, a linguagem deve suportar:

- declaração de variáveis inteiras, escalares ou vectoriais;
- manuseamento destas variáveis, com operação de indexação, no caso dos arrays;
- expressões lógico-aritméticas;
- atribuição de expressões a variáveis;
- leitura do *standard input*;
- escrita para o *standard output;*
- estruturas de controlo de fluxo condicionais;
- estruturas de controlo de fluxo cíclicas;

Após a definição da linguagem é necessário realizar as análises léxica e sintática, seguidas das acções semânticas, que neste caso resultarão num conjunto de instruções *assembly* ordenadas correctamente.

2.2 Especificação do Requisitos

O primeiro passo deve ser a definição dos símbolos terminais e não-terminais, para que se possa construir a gramática independente de contexto, adicionando as regras de produção necessárias.

Então, seja G a gramática para a linguagem inperativa simples: G = <T,N,S,P>, onde T é o conjunto dos símbolos terminais, N o conjunto dos símbolos não terminais, S o axioma da gramática e P o conjunto de regras de produção.

Seja Alfa o conjunto dos caracteres do alfabeto, ou seja, de a até z e A a Z. Seja Num o conjunto dos algarismos da base decimal, isto é, de 0 a 9.

Seja *Int* a expressão regular que representa um número inteiro: $INT = (n \in Num)^+$.

Seja *VAR* a expressão regular que representa o formato de nome de variável aceite em C: $VAR = (a \in Alfa + \_)(a \in Alfa + \_ + i \in Int)^*$.

Então temos:

T = { 'VARS', 'START', 'END', 'SCAN', 'PRINT', 'IF', 'ELSE', 'WHILE', STRING, 'ERRO', '(', ')', '{', '}', '[', ']', ';', '"', '"', '+', '-', '*', '/', '%', '<', '>', '=', '!', '&', '|', '\n', '$', $v \in VAR$, $i \in INT$ }

N = { Fonte, Cabeca, Corpo, Declaracoes, Declaracao, Instrucoes, instrucao, Atribuicao, Escrita, Leitura, Condicional, Else, Ciclo, Exp }

S = { Fonte }

P = { Cabeca → VARS Declaracoes

Declaracoes → Declaracoes Declaração

Declaracoes → Declaracao

Declaracao → VAR ';'

Declaracao → VAR '[' INT ']' ';'

Corpo → Start Instrucoes END

Instrucoes → Instrucoes Instrucao

Instrucoes → Instrucao

Instrucao → Atribuicao

Instrucao → Escrita

Instrucao → Leitura

Instrucao → Condicional

Instrucao → Ciclo

Instrucao → ';'

Atribuicao → VAR '[' Exp ']' '=' Exp ';'

Atribuicao → VAR '=' Exp ';'

Atribuicao → VAR OPINCDEC ';'

Atribuicao → VAR OPASS Exp ';'

Leitura → SCAN VAR ';' INPUT

Leitura → SCAN VAR '[' Exp ']' ';' INPUT

Escrita → PRINT Exp ';'

Escrita → PRINT STRING ';'

Condicional → IF '(' Exp ')' '{' Instrucoes '}' Else

Else → ELSE '{' Instrucoes '}'

Else → ϵ

Ciclo → WHILE '(' Exp ')' '{' Instrucoes '}'

Exp → NOT Exp

Exp → Exp OPREL Exp

Exp → Exp OPRELEQ Exp

Exp → Exp OPLOG Exp

Exp → Exp OPADD Exp

Exp → Exp OPMUL Exp

Exp → VAR

Exp → VAR '[' Exp ']'

Exp → INT

Exp → '(' Exp ')'

}

Finalizada a especificação da gramática independente de contexto necessária á resolução do problema, passa-se ao desenho da sua solução.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

Após ter ficado esclarecido, numa aula TP, que não é necessário ter em atenção a eficiência das estruturas de dados, apenas são utilizadas listas ligadas. Então, as estruturas de dados a utilizar são as seguintes:

É necessário guardar as variáveis declaradas para que se possa aceder aos seus endereços de memória e para testar se há re-declarações, variáveis não declaradas, utilização de variáveis não inicializadas ou utilização de variáveis de tipo errado.

```
struct variavel
{
  char *nome;
  char *tipo;
  int endereco;
  int inicializada;
  struct variavel *next;
};
```

No caso das instruções condicionais e cíclicas, só se sabem os endereços de salto após a leitura de todas as instruções contidas nestas estruturas, pelo que é necessário voltar atrás e indicar os endereços de salto correctamente. Assim, são guardadas todas as instruções numa lista que, no final do parsing, são imprimidas para o ficheiro de *output* final.

```
struct instrucao
{
  int endereco;
  char *instrucao;
  struct instrucao *next;
};
```

Para resolver o problema de aninhamento de condicionais e de ciclo, é necessário guardar o endereço para a instrução *jump if zero*, que sai da estrutura e no caso dos ciclos, é também guardado o endereço para a instrução *jump* incondicional, que salta para o início do ciclo.

Neste caso, as listas ligadas funcionam em pilha, ou seja, LIFO, para resolver correctamente os aninhamentos.

```
struct ifAddr
{
  int jz;
  struct ifAddr *next;
};

struct whileAddr
{
  int jump;
  int jz;
  struct whileAddr *next;
};
```

Com o auxílio destas estruturas e algoritmos sobre elas, segue-se o funcionamento da aplicação.

3.2 Algoritmos

Após ter ficado esclarecida a utilização da ferramenta *Yacc*, o grupo revelou-se algo surpreendido com a facilidade com esta gera o resultado, desde que as acções semânticas sejam colocadas no sítio correcto. Assim, não é necessário ter muitas estruturas e algoritmos intermédios para chegar ao *output* final.

Os algorimos que funcionam sobre as estruturas encontram-se no ficheiro "estrutura.c"e são as seguintes:

- *insertVariavel*, para inserir uma variável na lista de variáveis;
- *existeVariavel*, para testar se uma dada variável já se encontra declarada;
- *tipoVariavel*, para testar o tipo de uma variável que está a ser invocada;
- *enderecoVariavel*, para aceder ás posições de memória das variáveis;
- *inicializaVariavel*, para inicializar uma variável;
- *variavelInicializada*, para testar se uma variável está inicializada;
- *insertInstrucao*, para preencher a lista de instruções a ser imprimida em ficheiro;
- *pushIfAddr* e *popIfAddr*, para (des)empilhar endereço inicial de um condicional;
- *pushWhileAddr* e *popWhileAddr()*, para (des)empilhar endereço inicial de um ciclo;
- *ifJump*, que procura na lista de instruções por um endereço que retirou da pilha e lhe coloca o endereço onde se encontra actualmente, mais uma unidade, que é a instrução no fim do *if*
- *elseJump*, é análoga à função anterior mas em vez de imprimir 'JZ <label>', imprime 'JUMP <label>'. Isto para o caso de um *if* com *else* não execute o *else* caso entre no *if*;
- *whileJump*, análoga ás funções anteriores mas retorna o outro endereço que se encontra na pilha, de *jump* incondicional, para inserir na lista de instruções a instrução correspondente.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Operadores

Um dos principais problemas de uma linguagem encontra-se no tratamento de expressões lógico-aritméticas. As operações têm prioridades diferentes e se forem executadas em sequência, o resultado final é errado.

No *Yacc*, a resolução deste problema é extremamente simples, para definir as prioridades basta indicar a associatividade dos operadores e escreve-los na ordem desejada. A ordem das prioridades é ascendente, isto é, o último operador a ser indicado é o de maior prioridade. Então, temos:

```
%left OPLOG
%left NOT OPREL OPRELEQ
%left OPADD
%left OPMUL
```

Assim, os operadores têm as seguintes prioridades, por ordem ascendente: '&&', '||', '!', '>', '<', '>=', '<=', '==', '!=', '+', '-', '*', '/', '%'.

Aninhamento

O problema mais difícil de resolver foi o do aninhamento de condicionais e ciclos, pois é necessário voltar atrás nas instruções já lidas. A solução passou pela criação e manipulação de pilhas contendo os endereços necessários para determinar os saltos.

Assim, sempre que o *parser* lê um condicional ou ciclo, empilha o seu endereço na pilha correspondente e, quando lê o fim da estrutura, remove da pilha o endereço e actualiza as instruções.

Nota: é necessário inserir na lista de instruções uma instrução 'if' ou 'while' quando estas são detectadas, que depois são alteradas para instruções de salto, caso contrário, a ordenação dos endereços final estaria errada.

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes realizados (valores introduzidos) respectivos resultados obtidos: cálculo do expoente, do factorial, do máximo divisor comum e do maior elemento de um array.

```
//--- par.c ---//

VARS

a;

START

PRINT "Introduza um inteiro ";
SCAN a;
>17

if(a%2 == 0)
{
  PRINT "O número ";
  PRINT a;
  PRINT " é par";
}
else
{
  PRINT "O número ";
  PRINT a;
```

```
  PRINT " é ímpar";
}

END


//--- par ---//

_000000: PUSHI 0
_000001: START
_000002: PUSHS "Introduza um inteiro "
_000003: WRITES
_000004: READ "17 "
_000005: ATOI
_000006: STOREG 0
_000007: PUSHG 0
_000008: PUSHI 2
_000009: MOD
_000010: PUSHI 0
_000011: EQUAL
_000012: JZ _000020
_000013: PUSHS "O número "
_000014: WRITES
_000015: PUSHG 0
_000016: WRITEI
_000017: PUSHS " é par"
_000018: WRITES
_000019: JUMP _000026
_000020: PUSHS "O número "
_000021: WRITES
_000022: PUSHG 0
_000023: WRITEI
_000024: PUSHS " é ímpar"
_000025: WRITES
_000026: STOP


//--- expoente.c ---//

VARS

a;
b;
exp;

START
```

```
PRINT "Introduza base e expoente: ";
SCAN a;
>2
SCAN b;
>3

exp = 1;
while(b > 0)
{
  exp = exp * a;
  b = b - 1;
}

PRINT "Resultado é ";
PRINT exp;

END


//--- expoente ---//

_000000: PUSHI 0
_000001: PUSHI 0
_000002: PUSHI 0
_000003: START
_000004: PUSHS "Introduza base e expoente: "
_000005: WRITES
_000006: READ "2"
_000007: ATOI
_000008: STOREG 0
_000009: READ "3"
_000010: ATOI
_000011: STOREG 1
_000012: PUSHI 1
_000013: STOREG 2
_000014: PUSHG 1
_000015: PUSHI 0
_000016: SUP
_000017: JZ _000027
_000018: PUSHG 2
_000019: PUSHG 0
_000020: MUL
_000021: STOREG 2
_000022: PUSHG 1
_000023: PUSHI 1
```

```
_000024: SUB
_000025: STOREG 1
_000026: JUMP 14
_000027: PUSHS "Resultado é "
_000028: WRITES
_000029: PUSHG 2
_000030: WRITEI
_000031: STOP


//--- factorial.c ---//

VARS

n;
q;

START

PRINT "Introduza inteiro ";
SCAN n;
>5

q = 1;
while(n > 0)
{
  q *= n;
  n--;
}

PRINT "Factorial de ";
PRINT n;
PRINT " é ";
PRINT q;

END


//--- factorial ---//

_000000: PUSHI 0
_000001: PUSHI 0
_000002: START
_000003: PUSHS "Introduza inteiro "
_000004: WRITES
_000005: READ "5"
```

```
_000006: ATOI
_000007: STOREG 0
_000008: PUSHI 1
_000009: STOREG 1
_000010: PUSHG 0
_000011: PUSHI 0
_000012: SUP
_000013: JZ _000023
_000014: PUSHG 1
_000015: PUSHG 0
_000016: MUL
_000017: STOREG 1
_000018: PUSHG 0
_000019: PUSHI 1
_000020: SUB
_000021: STOREG 0
_000022: JUMP 10
_000023: PUSHS "Factorial de "
_000024: WRITES
_000025: PUSHG 0
_000026: WRITEI
_000027: PUSHS " é "
_000028: WRITES
_000029: PUSHG 1
_000030: WRITEI
_000031: STOP
```

```
//--- mdc.c ---//

VARS

a;
b;
res;

START

PRINT "Introduza dois inteiros a e b ";
SCAN a;
>27
SCAN b;
>9

while(a%b != 0)
```

```
{
  res = a;
  a = b;
  b = res%b;
}

PRINT "O máximo divisor comum de a e b é ";
PRINT res;

END


//--- mdc ---//

_000000: PUSHI 0
_000001: PUSHI 0
_000002: PUSHI 0
_000003: START
_000004: PUSHS "Introduza dois inteiros a e b "
_000005: WRITES
_000006: READ "27"
_000007: ATOI
_000008: STOREG 0
_000009: READ "9"
_000010: ATOI
_000011: STOREG 1
_000012: PUSHG 0
_000013: PUSHG 1
_000014: MOD
_000015: PUSHI 0
_000016: EQUAL
_000017: NOT
_000018: JZ _000028
_000019: PUSHG 0
_000020: STOREG 2
_000021: PUSHG 1
_000022: STOREG 0
_000023: PUSHG 2
_000024: PUSHG 1
_000025: MOD
_000026: STOREG 1
_000027: JUMP 12
_000028: PUSHS "O máximo divisor comum de a e b é "
_000029: WRITES
_000030: PUSHG 2
_000031: WRITEI
```

```
_000032: STOP


//--- maxArray.c ---//

VARS

a[10];
i;
max;

START

PRINT "Preencher array com 10 elementos ";
i = 0;
while(i < 10)
{
  PRINT "Introduza inteiro ";
  SCAN a[i];
  >X  //Introduzido pelo utilizador
}

max = a[0];
i = 1;
while(i < 10)
{
  if(a[i] > max)
  {
    max = a[i];
    i++;
  }
}

PRINT "Elemento máximo do array é ";
PRINT max;

END


//--- maxArray ---//

_000000: PUSHN 10
_000001: PUSHI 0
_000002: PUSHI 0
_000003: START
_000004: PUSHS "Preencher array com 10 elementos "
```

```
_000005: WRITES
_000006: PUSHI 0
_000007: STOREG 10
_000008: PUSHG 10
_000009: PUSHI 10
_000010: INF
_000011: JZ _000020
_000012: PUSHS "Introduza inteiro "
_000013: WRITES
_000014: PUSHI 0
_000015: PUSHG 10
_000016: READ "X"
_000017: ATOI
_000018: STOREN
_000019: JUMP 8
_000020: PUSHI 0
_000021: PUSHI 0
_000022: LOADN
_000023: STOREG 11
_000024: PUSHI 1
_000025: STOREG 10
_000026: PUSHG 10
_000027: PUSHI 10
_000028: INF
_000029: JZ _000045
_000030: PUSHI 0
_000031: PUSHG 10
_000032: LOADN
_000033: PUSHG 11
_000034: SUP
_000035: JZ _000044
_000036: PUSHI 0
_000037: PUSHG 10
_000038: LOADN
_000039: STOREG 11
_000040: PUSHG 10
_000041: PUSHI 1
_000042: ADD
_000043: STOREG 10
_000044: JUMP 26
_000045: PUSHS "Elemento máximo do array é "
_000046: WRITES
_000047: PUSHG 11
_000048: WRITEI
_000049: STOP
```

Capítulo 5

Conclusão

O grupo conclui que, dados os objectivos do trabalho prático e ainda as expectativas para este, o resultado final é bastante bom, tendo os problemas sido resolvidos na sua totalidade e com alguns extras, em alguns casos.

No assunto das gramáticas independentes de contexto e da geração de compiladores, os conhecimentos ficaram bastante consolidados, tendo o trabalho prático muito contribuido para isso.

É pena não termos conseguido implementar as funções pois seria um passo bastante grande para além dos objectivos propostos. De facto, é a crítica que todos os alunos estão a fazer e, na minha opinião, com razão pois o número de trabalhos ultrapassa o razoável.

Eu, Tiago Conceição, volto a "reclamar" sobre os trabalhos práticos neste relatório como tenho vindo a fazer noutros trabalhos e continuarei a fazê-lo nos que ainda me restam (que são 2 e tenho 4 dias para os terminar). Faço-o porque estou convicto de que esta situação está errada e necessita de urgente solução.

Termino dizendo que todas estas semanas passadas em frente ao computador resultaram em vários problemas de saúde, no que toca á minha visão, alimentação, postura corporal, condição muscular, insónias, *stress*, entre outros factores de desgaste psicológico. E os alunos não merecem nem precisam de passar por isto, sobretudo num país onde a profissão de programador é paga com literais trocos.

Apêndice A

Código do Programa

```
//--- virtual.y ---//


%{
  #include "estrutura.c"

  extern int yylex();
  extern int yylineno;
  extern FILE *yyin;
  extern char *yytext;
  extern void yyerror(char *s);

  int gp = 0;
  int iAddr = 0;
  int whileAddr = 0;
  int erros = 0;
  char *filename;

  struct variavel *listaVariaveis = NULL, *auxVariavel = NULL;
  struct instrucao *listaInstrucoes = NULL;
  struct ifAddr *pilhaIfAddr = NULL;
  struct whileAddr *pilhaWhileAddr = NULL;
%}

%union
{
  int inteiro;
  char op;
  char *var;
  char *string;
```

```
}

%token <inteiro> INT
%token <var> VAR
%token <op> OPINCDEC OPASS OPLOG OPREL OPRELEQ OPADD OPMUL
%token <string> STRING INPUT
%token VARS START END SCAN PRINT IF ELSE WHILE NOT ERRO

%left OPLOG
%left NOT OPREL OPRELEQ
%left OPADD
%left OPMUL

%start Fonte

%%

Fonte : Cabeca Corpo
    ;

Cabeca :
     | VARS Declaracoes { listaInstrucoes =
        insertInstrucao(iAddr++,"START",listaInstrucoes); }
     ;

Declaracoes : Declaracoes Declaracao
          | Declaracao
          ;

Declaracao : VAR ';'
        {
          auxVariavel = malloc(sizeof(struct variavel));
            auxVariavel->nome = strdup($1);
            auxVariavel->tipo = strdup("inteiro");
            auxVariavel->endereco = gp;
            auxVariavel->inicializado = 0;

          if(!existeVariavel(auxVariavel->nome,listaVariaveis))
          {
            listaVariaveis = insertVariavel(auxVariavel,listaVariaveis);
            listaInstrucoes = insertInstrucao(iAddr++,"PUSHI 0",listaInstrucoes);
            gp++;
          }
          else
          {
            erros = 1;
```

```
            fprintf(stderr,"l.%d erro: variável '%s' já declarada\n",yylineno,$1);
          }
        }
       | VAR '[' INT ']' ';'
          {
            auxVariavel = malloc(sizeof(struct variavel));
            auxVariavel->nome = strdup($1);
            auxVariavel->tipo = strdup("array");
            auxVariavel->endereco = gp;
            auxVariavel->inicializado = 0;

          if(!existeVariavel(auxVariavel->nome,listaVariaveis))
          {
            listaVariaveis = insertVariavel(auxVariavel,listaVariaveis);
            char buf[20];
            sprintf(buf,"PUSHN %d",$3);
            listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
            gp += $3;
          }
          else
          {
            erros = 1;
            fprintf(stderr,"l.%d erro: variável '%s' já declarada\n",yylineno,$1);
          }
          }
       ;

Corpo :
    | START Instrucoes END { listaInstrucoes =
       insertInstrucao(iAddr++,"STOP",listaInstrucoes); }
      ;

Instrucoes : Instrucoes Instrucao
       | Instrucao
       ;

Instrucao : Atribuicao
      | Escrita
      | Leitura
      | Condicional
      | Ciclo
      | ';' { listaInstrucoes = insertInstrucao(iAddr++,"NOP",listaInstrucoes);  }
      ;

Atribuicao : VAR '=' Exp ';'
        {
```

```
    if(!existeVariavel($1,listaVariaveis))
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
    }
    else if(strcmp(tipoVariavel($1,listaVariaveis),"inteiro") != 0)
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não é um escalar\n",yylineno,$1);
    }
    else
    {
      inicializaVariavel($1,listaVariaveis);

      char buf[20];
      sprintf(buf,"STOREG %d",enderecoVariavel($1,listaVariaveis));
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    }

  }

 | VAR
    {
      if(!existeVariavel($1,listaVariaveis))
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
    }
    else if(strcmp(tipoVariavel($1,listaVariaveis),"array") != 0)
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não é um vector\n",yylineno,$1);
    }
    else
    {
      inicializaVariavel($1,listaVariaveis);

        char buf[20];
      sprintf(buf,"PUSHI %d",enderecoVariavel($1,listaVariaveis));
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    }
  }
'[' Exp ']' '=' Exp ';'
    {
    listaInstrucoes = insertInstrucao(iAddr++,"STOREN",listaInstrucoes);
    }
```

```
| VAR OPINCDEC ';'
   {
     if(!existeVariavel($1,listaVariaveis))
   {
     erros = 1;
     fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
   }
   else if(strcmp(tipoVariavel($1,listaVariaveis),"inteiro") != 0)
   {
     erros = 1;
     fprintf(stderr,"l.%d erro: variável '%s' não é um escalar\n",yylineno,$1);
   }
   else if(!variavelInicializada($1,listaVariaveis))
   {
     erros = 1;
     fprintf(stderr,"l.%d erro: variável '%s' não está
       inicializada\n",yylineno,$1);
   }
   else
   {
       char buf[20];
     sprintf(buf,"PUSHG %d",enderecoVariavel($1,listaVariaveis));
     listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
     listaInstrucoes = insertInstrucao(iAddr++,"PUSHI 1",listaInstrucoes);

     if($2 == '+')
       listaInstrucoes = insertInstrucao(iAddr++,"ADD",listaInstrucoes);
     else if($2 == '-')
         listaInstrucoes = insertInstrucao(iAddr++,"SUB",listaInstrucoes);

     sprintf(buf,"STOREG %d",enderecoVariavel($1,listaVariaveis));
     listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
   }
   }

| VAR
   {
     if(!existeVariavel($1,listaVariaveis))
   {
     erros = 1;
     fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
   }
   else if(strcmp(tipoVariavel($1,listaVariaveis),"inteiro") != 0)
   {
     erros = 1;
```

```
            fprintf(stderr,"l.%d erro: variável '%s' não é um escalar\n",yylineno,$1);
          }
          else if(!variavelInicializada($1,listaVariaveis))
          {
            erros = 1;
            fprintf(stderr,"l.%d erro: variável '%s' não está
               inicializada\n",yylineno,$1);
          }
          else
          {
            inicializaVariavel($1,listaVariaveis);

              char buf[20];
              sprintf(buf,"PUSHG %d",enderecoVariavel($1,listaVariaveis));
            listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
          }
        }
        OPASS Exp ';'
          {
            if($3 == '+')
              listaInstrucoes = insertInstrucao(iAddr++,"ADD",listaInstrucoes);
            else if($3 == '-')
              listaInstrucoes = insertInstrucao(iAddr++,"SUB",listaInstrucoes);
            else if($3 == '*')
              listaInstrucoes = insertInstrucao(iAddr++,"MUL",listaInstrucoes);
            else if($3 == '/')
              listaInstrucoes = insertInstrucao(iAddr++,"DIV",listaInstrucoes);
            else if($3 == '%')
              listaInstrucoes = insertInstrucao(iAddr++,"MOD",listaInstrucoes);

            char buf[20];
            sprintf(buf,"STOREG %d",enderecoVariavel($1,listaVariaveis));
          listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
          }
       ;

Leitura : SCAN VAR ';' INPUT
      {
        if(!existeVariavel($2,listaVariaveis))
        {
          erros = 1;
          fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$2);
        }
        else if(strcmp(tipoVariavel($2,listaVariaveis),"inteiro") != 0)
        {
          erros = 1;
```

```
      fprintf(stderr,"l.%d erro: variável '%s' não é um escalar\n",yylineno,$2);
    }
    else
    {
      inicializaVariavel($2,listaVariaveis);

      char buf[1024];
      sprintf(buf,"READ \"%s\"",$4);
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
      listaInstrucoes = insertInstrucao(iAddr++,"ATOI",listaInstrucoes);

      sprintf(buf,"STOREG %d",enderecoVariavel($2,listaVariaveis));
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    }
  }

| SCAN VAR
  {
    if(!existeVariavel($2,listaVariaveis))
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$2);
    }
    else if(strcmp(tipoVariavel($2,listaVariaveis),"array") != 0)
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não é um vector\n",yylineno,$2);
    }
    else
    {
      inicializaVariavel($2,listaVariaveis);

        char buf[1024];
      sprintf(buf,"PUSHI %d",enderecoVariavel($2,listaVariaveis));
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    }
  }
'[' Exp ']' ';' INPUT
  {
    char buf[20];
    sprintf(buf,"READ \"%s\"",$8);
    listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    listaInstrucoes = insertInstrucao(iAddr++,"ATOI",listaInstrucoes);
    listaInstrucoes = insertInstrucao(iAddr++,"STOREN",listaInstrucoes);
  }
```

```
    ;

Escrita : PRINT Exp ';'
      {
        listaInstrucoes = insertInstrucao(iAddr++,"WRITEI",listaInstrucoes);
      }

    | PRINT STRING ';'
      {
        char buf[1024];
        sprintf(buf,"PUSHS \"%s\"",$2);
        listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
        listaInstrucoes = insertInstrucao(iAddr++,"WRITES",listaInstrucoes);
      }
    ;


Condicional : IF '(' Exp ')'
        {
          pilhaIfAddr = pushIfAddr(iAddr,pilhaIfAddr);
          listaInstrucoes = insertInstrucao(iAddr++,"if",listaInstrucoes);
        }
      '{' Instrucoes '}'
        {
          ifJump(iAddr,pilhaIfAddr,listaInstrucoes);
          pilhaIfAddr = popIfAddr(pilhaIfAddr);
        }
      Else
      ;

Else : ELSE
    {
      pilhaIfAddr = pushIfAddr(iAddr,pilhaIfAddr);
      listaInstrucoes = insertInstrucao(iAddr++,"else",listaInstrucoes);
    }
    '{' Instrucoes '}'
    {
      elseJump(iAddr,pilhaIfAddr,listaInstrucoes);
      pilhaIfAddr = popIfAddr(pilhaIfAddr);
    }
   |
   ;

Ciclo : WHILE '('
      {
        whileAddr = iAddr;
```

```
      }
    Exp ')'
      {
        pilhaWhileAddr = pushWhileAddr(iAddr,whileAddr,pilhaWhileAddr);
        listaInstrucoes = insertInstrucao(iAddr++,"while",listaInstrucoes);
      }
    '{' Instrucoes
      {
        int jump = whileJump(iAddr+1,pilhaWhileAddr,listaInstrucoes);
        pilhaWhileAddr = popWhileAddr(pilhaWhileAddr);

        char buf[20];
        sprintf(buf,"JUMP %d",jump);
        listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
      }
    '}'
    ;

Exp : NOT Exp
    {
      listaInstrucoes = insertInstrucao(iAddr++,"NOT",listaInstrucoes);
    }

  | Exp OPREL Exp
    {
      if($2 == '>')
        listaInstrucoes = insertInstrucao(iAddr++,"SUP",listaInstrucoes);
      else if($2 == '<')
        listaInstrucoes = insertInstrucao(iAddr++,"INF",listaInstrucoes);
    }

  | Exp OPRELEQ Exp
    {
      if($2 == '>')
        listaInstrucoes = insertInstrucao(iAddr++,"SUPEQ",listaInstrucoes);
      if($2 == '<')
        listaInstrucoes = insertInstrucao(iAddr++,"INFEQ",listaInstrucoes);
      else if($2 == '=')
        listaInstrucoes = insertInstrucao(iAddr++,"EQUAL",listaInstrucoes);
      else if($2 == '!')
      {
        listaInstrucoes = insertInstrucao(iAddr++,"EQUAL",listaInstrucoes);
        listaInstrucoes = insertInstrucao(iAddr++,"NOT",listaInstrucoes);
      }
    }
```

```
| Exp OPLOG Exp
  {
    if($2 == '&')
      listaInstrucoes = insertInstrucao(iAddr++,"MUL",listaInstrucoes);
    if($2 == '|')
      listaInstrucoes = insertInstrucao(iAddr++,"ADD",listaInstrucoes);
  }

| Exp OPADD Exp
  {
    if($2 == '+')
      listaInstrucoes = insertInstrucao(iAddr++,"ADD",listaInstrucoes);
    if($2 == '-')
      listaInstrucoes = insertInstrucao(iAddr++,"SUB",listaInstrucoes);
  }

| Exp OPMUL Exp
  {
    if($2 == '*')
      listaInstrucoes = insertInstrucao(iAddr++,"MUL",listaInstrucoes);
    if($2 == '/')
      listaInstrucoes = insertInstrucao(iAddr++,"DIV",listaInstrucoes);
    if($2 == '%')
      listaInstrucoes = insertInstrucao(iAddr++,"MOD",listaInstrucoes);
  }

| VAR
  {
    if(!existeVariavel($1,listaVariaveis))
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
    }
    else if(strcmp(tipoVariavel($1,listaVariaveis),"inteiro") != 0)
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não é um escalar\n",yylineno,$1);
    }
    else if(!variavelInicializada($1,listaVariaveis))
    {
      erros = 1;
      fprintf(stderr,"l.%d erro: variável '%s' não está inicializada\n",yylineno,$1);
    }
    else
    {
      char buf[20];
```

```
        sprintf(buf,"PUSHG %d",enderecoVariavel($1,listaVariaveis));
        listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
      }
    }

  | VAR
    {
      if(!existeVariavel($1,listaVariaveis))
      {
        erros = 1;
        fprintf(stderr,"l.%d erro: variável '%s' não declarada\n",yylineno,$1);
      }
      else if(strcmp(tipoVariavel($1,listaVariaveis),"array") != 0)
      {
        erros = 1;
        fprintf(stderr,"l.%d erro: variável '%s' não é um vector\n",yylineno,$1);
      }
      else if(!variavelInicializada($1,listaVariaveis))
      {
        erros = 1;
        fprintf(stderr,"l.%d erro: variável '%s' não está inicializada\n",yylineno,$1);
      }
      else
      {
          char buf[20];
        sprintf(buf,"PUSHI %d",enderecoVariavel($1,listaVariaveis));
        listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
      }
    }
  '[' Exp ']'
    {
      listaInstrucoes = insertInstrucao(iAddr++,"LOADN",listaInstrucoes);
    }

  | INT
    {
      char buf[20];
      sprintf(buf,"PUSHI %d",$1);
      listaInstrucoes = insertInstrucao(iAddr++,buf,listaInstrucoes);
    }

  | '(' Exp ')'
  ;


%%
```

```
void yyerror(char* s)
{
   fprintf(stderr,"erro: %d: %s: %s\n",yylineno,s,yytext);
   erros = 1;
}

int main(int argc, char *argv[])
{
  if(argc < 2)
  {
    printf("Introduzir ficheiro para compilação como argumento!\n");
    exit(0);
  }

  yyin = fopen(argv[1],"r");
    yyparse();

    filename = strdup(argv[1]);
  filename[strrchr(filename,'.')-filename] = '\0';
  FILE* f = fopen(filename,"w");

  if(!erros)
  {
    while(listaInstrucoes)
    {
      if(listaInstrucoes->instrucao)
      {
        fprintf(f,"_%06d: %s\n",listaInstrucoes->endereco,listaInstrucoes->instrucao);
        //printf("_%06d: %s\n",listaInstrucoes->endereco,listaInstrucoes->instrucao);
      }
      listaInstrucoes = listaInstrucoes->next;
    }
  }
  else
  {
    fprintf(f,"Houve erros durante a compilação\n");
  }

    return 0;
}


//--- virtual.l ---//

%{
```

```
  #include "y.tab.h"
%}

%option yylineno

espaco    [ \t]
inteiro   [0-9]+
var       [a-zA-Z_][a-zA-Z0-9_]*

vars    (?i:VARS)
start   (?i:START)
end     (?i:END)
inc     \+\+
dec     \-\-
addass  \+\=
subass  \-\=
mulass  \*\=
divass  \/\=
modass  \%\=
scan    (?i:SCAN)
print   (?i:PRINT)
string  \"([^"]|\\\")*\"
if      (?i:IF)
else    (?i:ELSE)
while   (?i:WHILE)

%%

[ \t\n] ;

[\(\)\[\]\{\}\;\=\"]  { return yytext[0]; }

{vars}  { return VARS; }
{start} { return START; }
{end}   { return END; }

{inc}     { yylval.op = yytext[0]; return OPINCDEC; }
{dec}     { yylval.op = yytext[0]; return OPINCDEC; }
{addass}  { yylval.op = yytext[0]; return OPASS; }
{subass}  { yylval.op = yytext[0]; return OPASS; }
{mulass}  { yylval.op = yytext[0]; return OPASS; }
{divass}  { yylval.op = yytext[0]; return OPASS; }
{modass}  { yylval.op = yytext[0]; return OPASS; }

{scan}  { return SCAN; }
\n{espaco}*\>[^\n]*\n   {
```

```
          yylval.string = strdup(strchr(yytext,'>')+1);
          yylval.string[strchr(yylval.string,'\n')-yylval.string] = '\0';
          return INPUT;
        }

{print}   { return PRINT; }
{string}  { yylval.string = strdup(yytext+1);
            yylval.string[strchr(yylval.string,'"')-yylval.string] = '\0';
            return STRING;
          }

{if}    { return IF; }
{else}  { return ELSE; }
{while} { return WHILE; }

\!    { return NOT; }
\>    { yylval.op = yytext[0]; return OPREL; }
\<    { yylval.op = yytext[0]; return OPREL; }

\>\=  { yylval.op = yytext[0]; return OPRELEQ; }
\<\=  { yylval.op = yytext[0]; return OPRELEQ; }
\=\=  { yylval.op = yytext[0]; return OPRELEQ; }
\!\=  { yylval.op = yytext[0]; return OPRELEQ; }

\&\&  { yylval.op = yytext[0]; return OPLOG; }
\|\|  { yylval.op = yytext[0]; return OPLOG; }

[\-\+]    { yylval.op = yytext[0]; return OPADD; }
[\*\/\%]  { yylval.op = yytext[0]; return OPMUL; }

{inteiro} { yylval.inteiro = atoi(yytext); return(INT); }
{var}     { yylval.var = strdup(yytext); return VAR; }

. return ERRO;


//--- estrutura.c ---//

#include "estrutura.h"

struct variavel* insertVariavel(struct variavel *variavel,
  struct variavel *listaVariaveis)
{
  struct variavel *ant=NULL,*pt=listaVariaveis,*aux=variavel;

  while(pt && pt->nome && strcmp(variavel->nome,pt->nome)>=0)
```

```
  {
    ant = pt;
    pt = pt->next;
  }

  if(!ant)
  {
    aux->next = listaVariaveis;
    return aux;
  }
  else
  {
    aux->next = pt;
    ant->next = aux;
    return listaVariaveis;
  }
}

int existeVariavel(char *variavel, struct variavel *listaVariaveis)
{
  struct variavel *pt=listaVariaveis;
  int res = 0;

  while(pt && pt->nome && strcmp(variavel,pt->nome)>=0)
  {
    if(strcmp(variavel,pt->nome)==0)
      res = 1;
    pt = pt->next;
  }

  return res;
}

int enderecoVariavel(char *variavel, struct variavel *listaVariaveis)
{
  struct variavel *pt=listaVariaveis;
  int res = -1;

  while(pt && pt->nome && strcmp(variavel,pt->nome)>=0)
  {
    if(strcmp(variavel,pt->nome)==0)
      res = pt->endereco;
    pt = pt->next;
  }

  return res;
```

```
}

char* tipoVariavel(char *variavel, struct variavel *listaVariaveis)
{
  struct variavel *pt=listaVariaveis;
  char* res = NULL;

  while(pt && pt->nome && strcmp(variavel,pt->nome)>=0)
  {
    if(strcmp(variavel,pt->nome)==0)
      res = pt->tipo;
    pt = pt->next;
  }

  return res;
}

void inicializaVariavel(char *variavel, struct variavel *listaVariaveis)
{
  struct variavel *pt=listaVariaveis;

  while(pt && pt->nome && strcmp(variavel,pt->nome)>=0)
  {
    if(strcmp(variavel,pt->nome)==0)
      pt->inicializado = 1;
    pt = pt->next;
  }
}

int variavelInicializada(char *variavel, struct variavel *listaVariaveis)
{
  struct variavel *pt=listaVariaveis;
  int res = 0;

  while(pt && pt->nome && strcmp(variavel,pt->nome)>=0)
  {
    if(strcmp(variavel,pt->nome)==0)
      res = pt->inicializado;
    pt = pt->next;
  }

  return res;
}

struct instrucao* insertInstrucao(int endereco, char *instrucao,
   struct instrucao *listaInstrucoes)
```

```
{
  struct instrucao *ant=NULL,*pt=listaInstrucoes,*aux=NULL;

  aux = malloc(sizeof(struct instrucao));
  aux->endereco = endereco;
  aux->instrucao = strdup(instrucao);
  aux->next = NULL;

  while(pt && aux->endereco >= pt->endereco)
  {
    ant = pt;
    pt = pt->next;
  }

  if(!ant)
  {
    aux->next = listaInstrucoes;
    return aux;
  }
  else
  {
    aux->next = pt;
    ant->next = aux;
    return listaInstrucoes;
  }
}

struct ifAddr* pushIfAddr(int endereco, struct ifAddr *pilhaIfAddr)
{
  struct ifAddr *aux = malloc(sizeof(struct ifAddr));
  aux->jz = endereco;
  aux->next = pilhaIfAddr;
  return aux;
}

struct ifAddr* popIfAddr(struct ifAddr *pilhaIfAddr)
{
  struct ifAddr *res = pilhaIfAddr->next;
  free(pilhaIfAddr);
  return res;
}

void ifJump(int endereco, struct ifAddr *pilhaIfAddr,
  struct instrucao *listaInstrucoes)
{
  struct instrucao *pt=listaInstrucoes, *aux=NULL;
```

```
  while(pt && pt->endereco != pilhaIfAddr->jz)
    pt = pt->next;

  char buf[20];
  sprintf(buf,"JZ _%06d",endereco);
  pt->instrucao = strdup(buf);
}

void elseJump(int endereco, struct ifAddr *pilhaIfAddr, struct instrucao *listaInstrucoes)
{
  struct instrucao *pt=listaInstrucoes, *aux=NULL;

  while(pt && pt->endereco != pilhaIfAddr->jz)
    pt = pt->next;

  char buf[20];
  sprintf(buf,"JUMP _%06d",endereco);
  pt->instrucao = strdup(buf);
}

struct whileAddr* pushWhileAddr(int endereco, int whileAddr,
   struct whileAddr *pilhaWhileAddr)
{
  struct whileAddr *aux = malloc(sizeof(struct whileAddr));
  aux->jump = whileAddr;
  aux->jz = endereco;
  aux->next = pilhaWhileAddr;
  return aux;
}

struct whileAddr* popWhileAddr(struct whileAddr *pilhaWhileAddr)
{
  struct whileAddr *res = pilhaWhileAddr->next;
  free(pilhaWhileAddr);
  return res;
}

int whileJump(int endereco, struct whileAddr *pilhaWhileAddr,
   struct instrucao *listaInstrucoes)
{
  struct instrucao *pt=listaInstrucoes, *aux=NULL;

  while(pt && pt->endereco != pilhaWhileAddr->jz)
    pt = pt->next;
```

```
  char buf[20];
  sprintf(buf,"JZ _%06d",endereco);
  pt->instrucao = strdup(buf);

  return pilhaWhileAddr->jump;
}


//--- estrutura.h ---//

#ifndef _ESTRUTURA
#define _ESTRUTURA

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

struct variavel
{
  char *nome;
  char *tipo;
  int endereco;
  int inicializado;
  struct variavel *next;
};

struct instrucao
{
  int endereco;
  char *instrucao;
  struct instrucao *next;
};

struct ifAddr
{
  int jz;
  struct ifAddr *next;
};

struct whileAddr
{
  int jump;
  int jz;
  struct whileAddr *next;
};
```

```
struct variavel* insertVariavel(struct variavel *variavel,
  struct variavel *listaVariaveis);
int existeVariavel(char *variavel, struct variavel *listaVariaveis);
int enderecoVariavel(char *variavel, struct variavel *listaVariaveis);
char* tipoVariavel(char *variavel, struct variavel *listaVariaveis);

struct instrucao* insertInstrucao(int endereco, char *instrucao,
  struct instrucao *listaInstrucoes);

struct ifAddr* pushIfAddr(int endereco, struct ifAddr *pilhaIfAddr);
struct ifAddr* popIfAddr(struct ifAddr *pilhaIfAddr);

struct whileAddr* pushWhileAddr(int endereco, int whileAddr,
  struct whileAddr *pilhaWhileAddr);
struct whileAddr* popWhileAddr(struct whileAddr *pilhaWhileAddr);

void ifJump(int endereco, struct ifAddr *pilhaIfAddr,
  struct instrucao *pilhaInstrucoes);
void elseJump(int endereco, struct ifAddr *pilhaIfAddr,
  struct instrucao *pilhaInstrucoes);

int whileJump(int endereco, struct whileAddr *pilhaWhileAddr,
  struct instrucao *pilhaInstrucoes);

#endif


//--- makefile ---//

virtual : lex.yy.o y.tab.o
  gcc -o virtual y.tab.o lex.yy.o -ll

y.tab.o : y.tab.c
  gcc -c y.tab.c

y.tab.c y.tab.h : virtual.y estrutura.c estrutura.h
  yacc -d virtual.y

lex.yy.o : lex.yy.c
  gcc -c lex.yy.c

lex.yy.c : virtual.l y.tab.h
  flex virtual.l
```