Processamento de Linguagens e Compiladores
(3º ano de LCC)

Trabalho Prático Nº3 (YACC)

Relatório de Desenvolvimento

João Ferreira (A76628)

Luís Daniel Félix (A74246)

Maria Manuela Silva (A74408)

6 de Janeiro de 2019

Conteúdo

Capítulo 1

Introdução

No trabalho prático nº3, é-nos pedida uma linguagem de programação imperativa simples implementando tanto a sua gramática como o seu correspondete analisador lexico. Surge inserido no âmbito da disciplina de Processamento de Linguagens e Compiladores.O objetivo principal é o desenvolvimento de um compilador gerando código para uma máquina de stack virtual e aumentar a experiência em engenharia de linguagens.

Com o desenvolvimento de um processador de linguagens segundo o método de tradução dirigida pela sintaxe, no final deste projeto, devemos ter uma linguagem totalmente definida com base numa GIC e o seu respetivo compilador que deve gerar código assembly.

Ao longo deste documento podemos encontrar a exposição do problema, a sua análise, a forma como o interpretamos e resolvemos e as dificuldades que surgiram ao longo da sua realização.

Capítulo 2

Análise e Especificação da Linguagem Desenvolvida

2.1 Descrição informal do problema

Com a realização deste trabalho pretende-se definir uma Linguagem de Programação Imperativa que permita declarar variáveis atómicas dos tipos Inteiro, Real e Booleano e fazer as Operações de Atribuição de Expressões a Variáveis declaradas, Leitura (de inteiros ou reais), Escrita (de inteiros, Reais, Booleanos ou Strings), Condições e Ciclos Repetir e gerar código Assembly para a VM.

2.2 Especificação do Requisitos

2.2.1 Dados

De forma a guardar as variáveis declaradas e para que se possa ter acesso posteriormente, assim como para verificar a não declaração ou a não inicialização destas, recorremos a uma tabela de hash com a seguinte representação:

```
struct node{
char *key;
int type;
int address;
int inicializada;
};
```

Em que:

- char *key - corresponde à chave da tabela, sendo esta o nome da variável;
- int type - corresponde ao tipo da variável, assumindo o valor '0' no caso de uma variável do tipo INTEIRO, '1' no caso de uma variável do tipo REAL e '2' no caso de uma variável do tipo BOOL;
- int address - corresponde ao endereço da variável na stack;
- int inicializada - verifica se a variável foi inicializada ou não, tomando o valor de 0 no caso de nao estar.

Seguidamente, foram definidas as funções de hash apresentadas a baixo na imagem
Recorremos ainda a uma extutura adicional, uma stack, para poder guardar os numeros das labels dos ciclos e das condições, para que seja possivel aninhar os mesmos.

```
HashTable createHashTable(const unsigned int capacity);

void destroyHashTable(HashTable h);

HashTable addHashTable(HashTable h, char *key, int address, int type);

table get_HashTable(HashTable hashtbl, char *key);

int getEndereco(const HashTable h, char *key);

int getInicializacao(const HashTable h, char *key);
```

Figura 2.1: Funções de Hash

```
    struct STACK
{
    int stk[MAXSIZE];
    int sp;
};
typedef struct STACK* stack;
```

Posteriormente foram definidas as funções para atuar na stack, como se pode ver na seguinte imagem.

```
typedef struct STACK* stack;

int top(stack s);
void push(stack s, int elem);
int pop();
stack initStack();
```

Figura 2.2: Funções da Stack

Capítulo 3

Conceção/desenho da Resolução e Codificação

3.1 Gramática

Sguindo como base uma gramática do tipo **GIC** começamos por definir os símbolos terminais e não terminais e só depois criar as produções para a formação da linguagem e o seu respetivo símbolo inicial.
Considerando G como a gramática a desenvolver, seja **G = (T,NT,S,P)** onde **T** é o conjunto dos símbolos terminais, **NT** é o conjunto dos símbolos não terminais, **S** é o simbolo inicial e **P** é o conjunto das regras de produção.

```
    G = (
        T = {':', ';',',','<','<','=','!','{','}',')','(','*','+','/','-',
        NUMF, NUMI, NBOOL, INTEIRO, REAL, BOOL,STR, SE,
        SENAO, ENQUANTO, INICIO, FIM, ENVIA, RECEBE, ID},

        NT = {Programa, Topo, Corpo, Declaracao, Tipo,Variaveis, Variavel,Instrucoes, Instrucao,
        Condicao, Ciclo, IO, Atribuicao, Operacao, Exp, Termo, Fator,Output, Input},


        S = Programa,


        P = {

Programa : Topo Corpo
         ;

Topo   : Declaracao
        | Topo Declaracao
        ;

Declaracao : Tipo Variaveis ';'
           ;

Tipo : INTEIRO
     | REAL
     | BOOL
     ;
```

```
Variaveis : Variaveis ',' Variavel
          | Variavel
          ;

Variavel : ID
         ;

Corpo : INICIO Instrucoes FIM
      ;

Instrucoes : Instrucoes  Instrucao
           | Instrucao
           ;

Instrucao : Condicao
          | Ciclo
          | IO ';'
          | Atribuicao ';'
          ;

Atribuicao : Variavel '=' Operacao
           ;


Condicao : SE '(' Operacao')' ':' '{' Instrucoes '}'  SENAO '{' Instrucoes '}'
         | SE '(' Operacao')' ':' '{' Instrucoes '}'
         ;

Ciclo : ENQUANTO '(' Operacao')' ':' '{' Instrucoes '}'
      ;

Operacao : Exp '!' '=' Exp
         | Exp '=' '=' Exp
         | Exp '<' '='  Exp
         | Exp '>' '='  Exp
         | Exp '<' Exp
         | Exp '>' Exp
         | Exp
         ;

Exp   : Termo
      | Exp '+' Termo
      | Exp '-' Termo
      ;

Termo : Fator
      | Termo '*' Fator
      | Termo '/' Fator
      ;


Fator : NUMI
      | NUMF
```

```
      | ID
      | NBOOL
      | '(' Exp ')'
      ;



IO : Output
   | Input
   ;

Output : ENVIA Exp
       | ENVIA STR
       ;

Input : RECEBE ID
      ;

        }
        )
```

3.2 Analisador Léxico

3.2.1 Simbolos

Usamos apenas uma expressão regular para obter o valor ASCII correspondente a cada símbolo.

```
    [=+\-*/()><!,:;{}]            { return yytext[0]; }
```

3.2.2 Palavras Reservadas

No caso das palavras reservadas, resolvemos do seguinte modo:

```
INTEIRO              {return INTEIRO;}
REAL                 {return REAL;}
BOOL                 {return BOOL;}
SE                   {return SE;}
SENAO                {return SENAO;}
ENQUANTO             {return ENQUANTO;}
INICIO               {return INICIO;}
FIM                  {return FIM;}
ENVIA                {return ENVIA;}
RECEBE               {return RECEBE;}
```

3.2.3 Expressões

As expressões são constituídas pelas expressõess regulares que geram strings, inteiros, reais,boleanos e id's.
Uma String (STR) é uma série de caracteres que começa e termina em aspas.
Um inteiro (NUMI) é um ou mais números.
um real (NUMF) é um ou mais numeros com um "."e novamente um ou mais números.
Um boleano (NBOOL) é um caracter F, f, T ou t.
Um ID é um identificador de uma variável, sendo inicializado com uma letra, obrigatoriamente, e posteriormente, pode conter ou não letras e/ou dígitos.

```
[TtFf]                {yylval.valb=strdup(yytext); return NBOOL; }
[a-zA-Z][a-zA-Z0-9]*  { yylval.vals=strdup(yytext); return ID; }
[0-9]+                { yylval.vali=atoi(yytext); return NUMI; }
[0-9]+'.'[0-9]+       { yylval.valf=atof(yytext);  return NUMF; }
\"[^"]+\"              { yylval.vals = strdup(yytext);return STR;}
```

3.3 Compilador

O compilador é constituido pela analise lexica, sintática e semântica. As análises sintática e e semantica são responsaveis por analisar uma sequência de entrada para determinar a sua estrutura gramatical segundo uma gramática e verificar os erros semânticos, respetivamente.

Segue-se a contrução do compilador.

3.3.1 Iniciação e Terminação da geração do código

```
Programa : Topo Corpo   { fprintf(f,"stop\n"); }
         ;

Topo   : Declaracao
       | Topo Declaracao
      ;

Declaracao : Tipo Variaveis ';' { fprintf(f,"start\n");}
           ;
```

3.3.2 Declaração de Variáveis e inserção das mesmas na Tabela de Hash

Quando as variáveis são declaradas no inicio do programa, são guardadas numa tabela de Hash denominada de hashTableVar.

Dependendo do tipo de variável, como ja tinhamos referido anteriormente, as variáveis são distinguidas pelo tipo devido à forma como sao guardadas na tabela de Hash. O tipo 0 é para os inteiros, tipo 1 para os reais e tipo 2 para os boleanos.

Recorremos à função que definimos em C **insereVariavel()** para inserir a variavel na tabela de Hash, mais precisamente inserir o nome da variavel ($1).

```
Tipo : INTEIRO  {type = 0;}
     | REAL     {type = 1;}
     | BOOL     {type = 2;}
     ;


Variaveis : Variaveis ',' Variavel
           | Variavel
           ;

Variavel : ID { insereVariavel(hashTableVar,$1); }
         ;
```

3.3.3 Condições

A nossa estrutura de condições é constituídaa por duas condições diferentes. Temos "SE", seguido de um conjunto de operações e instruções seguindo-se um "SENAO"ou então temos "SE"e apenas um conjunto de operaões e instruções.

Criamos a stack denominada labels e ainda int labelcounterSE para contar o número de acessos if e else.
Não conseguimos colocar a parte relativa à stack pois estavamos com um erro que nao conseguimos resolver.

```
    Condicao : SE
              {
              labelcounterSE++;
               fprintf(f,"iflabel%d:\n",labelcounterSE);
               }
               '(' Operacao')' ':' '{' Instrucoes '}'  SENAO
                {
                fprintf(f,"elseLabel%d:\n",labelcounterSE);
                } '{' Instrucoes '}'
         | SE {
              labelcounterSE++;
               fprintf(f,"iflabel%d:\n",labelcounterSE);
               }
               '(' Operacao')' ':' '{' Instrucoes '}'
         ;
```

3.3.4 Ciclo

O ciclo funciona de forma idêntica às condições. Na estrutura deste temos "ENQUANTO"seguido de um conjunto de Operações e instruções.
Definimos o int labelcounterENQUANTO para contar o numero de acessos ao ciclo.

```
Ciclo : ENQUANTO {
                labelcounterENQUANTO++;
                fprintf(f,"Whilelabel%d:\n",labelcounterENQUANTO);
                }'(' Operacao')'':' '{' Instrucoes '}'
      ;
```

3.3.5 Atribuições

Os dois nao terminais Exp e Termo foram definidos para ganratir a prioridade das operações sendo que o Exp trata das operações com menos prioridade (somas e subtrações) e o Termo trata das operações com mais prioridade (multiplicações e divisões).
Segue-se o não temrinal Fator. Este é o primeiro a ser reconhecido pois contem basicamente as variáveis.

```
Atribuicao : Variavel '=' Operacao
           ;

Operacao : Exp '!' '=' Exp   { fprintf(f,"\t\tnot\n \t\tequal \n"); }
         | Exp '=' '=' Exp   { fprintf(f,"\t\tequal\n"); }
         | Exp '<' '='  Exp  { fprintf(f,"\t\tinfeq\n"); }
         | Exp '>' '='  Exp  { fprintf(f,"\t\tsupeq\n"); }
         | Exp '<' Exp       { fprintf(f,"\t\tinf\n"); }
         | Exp '>' Exp       { fprintf(f,"\t\tsup\n"); }
         | Exp
         ;

Exp    : Termo
```

```
      | Exp '+' Termo      { fprintf(f,"\t\tadd\n"); }
      | Exp '-' Termo      { fprintf(f,"\t\tsub\n"); }
      ;

Termo : Fator
      | Termo '*' Fator   { fprintf(f,"\t\tmul\n"); }
      | Termo '/' Fator   { fprintf(f,"\t\tdiv\n"); }
      ;


Fator : NUMI         {fprintf(f,"\t\tpushi %d\n",$1);}
      | NUMF         {fprintf(f,"\t\tpushf %f\n",$1);}
      | ID           {fprintf(f,"\t\tpushs %s\n",$1);}
      | NBOOL        {fprintf(f,"\t\tpushs %s\n",$1);}
      | '(' Exp ')'
      ;
```

3.3.6 IO

No Output, imprime-se expressões e strings.
No Input,depois de receber um ID, verifica se o nó associado à chave (ID) não é nulo e a partir daí lê as variavies introduzidas. No caso de o nó associado à chave ser nulo envia um erro.

```
IO : Output
   | Input
   ;

Output : ENVIA Exp { fprintf(f,"\t\twritei\n");}
       | ENVIA STR { fprintf(f,"\t\tpushs \%s\n",$2); fprintf(f,"\t\twrites\n");}
       ;

Input : RECEBE ID {if(get_HashTable(hashTableVar,$2)!=NULL)
      {
        fprintf(f,"\t\tread\n");
        fprintf(f,"\t\tatoi\n");
        fprintf(f,"\t\tstoreg %d\n",getEndereco(hashTableVar,$2));
      } else
      {
        erroDeclaracaow(numeroLinha,$2);
        }
        }
      ;
```

Capítulo 4

Testes e Resultados

4.1 Maximo entre dois números

Programa

```
INTEIRO  e,r, max;

INICIO
e = 100;
RECEBE r;

SE (e>r): {max = e;} SENAO {max = r;}

ENVIA max;

FIM
```

Codigo Assembly gerado

```
pushi 0
pushi 0
pushi 0
start
pushi 100
read
atoi
storeg 1
iflabel0:
pushs e
pushs r
sup
pushs e
elseLabel0:
pushs r
pushs max
writei
stop
```

4.2 Soma de todos os números entre 2 e 15

Programa

```
INTEIRO count,b,r;

INICIO

count=1;
r = 0;

ENQUANTO(count < 100) : {
RECEBE b;
r= count/b;
count = count+1;

}

ENVIA a;

FIM
```

Codigo Assembly gerado

```
  pushi 0
pushi 0
pushi 0
start
pushi 2
pushi 0
Whilelabel0:
pushs count
pushi 15
inf
read
atoi
storeg 0
pushs s
pushs x
add
pushs count
pushi 1
add
pushs s
writei
stop
```

4.3 Soma notas de uma turma

Programa

```
REAL nota,soma;
INTEIRO total,count;
```

```
INICIO

count=1;
soma = 0;

ENQUANTO(count <= total) : {
RECEBE total;
RECEBE nota;

soma = soma+nota;
count = count+1;

}

ENVIA soma;

FIM
```

Codigo Assembly gerado

```
pushi 0
pushi 0
start
pushi 0
pushi 0
start
pushi 1
pushi 0
Whilelabel0:
pushs count
pushs total
infeq
read
atoi
storeg 2
read
atoi
storeg 0
pushs soma
pushs nota
add
pushs count
pushi 1
add
pushs soma
writei
stop
```

4.4 Tabuada do N ate 10

Programa

```
INTEIRO n,count,r;
```

```
INICIO

r=0;
count=1;
RECEBE n;

ENQUANTO(count <= 10) : {
r = n*count;
count = count+1;
ENVIA r;

}


FIM
```

Codigo Assembly gerado

```
    pushi 0
pushi 0
pushi 0
start
pushi 0
pushi 1
read
atoi
storeg 0
Whilelabel0:
pushs count
pushi 10
infeq
pushs n
pushs count
mul
pushs count
pushi 1
add
pushs r
writei
stop
```

Capítulo 5

Conclusão

Neste trabalho, conseguimos cobrir grande parte dos objetivos propostos. No entanto, não conseguimos acabar o ciclo nem as condições de forma a passar a informação para a stack.

Apesar disso, esta experiência foi extremamente valiosa e interessante uma vez que atravéss dela, os membros do grupo adquiriram capacidades para o desenho e implementação de compiladores.