Relatório Final - INF1022 - 2021.2

Nicolas Paes Leme - 2011050

Objetivos

Construir usando Bison/Flex um analisador sintático para a linguagem Provol-One, que é definida pela gramática disponível no enunciado. Também é necessário que esse analisador desenvolvido tenha mensagens de erros úteis para o usuário, indicando **onde** ocorreu o erro, além de indicar o contexto do erro.

Foi escolhido a linguagem C para a linguagem final da compilação do arquivo de entrada em Provol-One.

A linguagem Provol-One sofreu pequenos ajustes para esse trabalho, mas foi necessário manter a linguagem compacta (não adição de funções extras) para garantir a robustez das mensagens de erro em todas as etapas da compilação.

Linguagem Provol-One

A linguagem tem a seguinte Gramática:

```
program -> ENTRADA varlist SAIDA varlist cmds FIM
varlist -> id varlist | id
cmds -> cmd cmds | cmd
cmd -> ENQUANTO id FACA cmds END
cmd -> id = id | INC(id) | ZERA(id)
```

A única pequena alteração feita na gramática foi a criação de um novo Token `END` para deixar a mensagem de erro mais clara no caso de uma falta do comando para terminar o *Loop*;

Flex prov.l

No arquivo `prov.l` temos a definição do nosso analisador léxico, com as regras de *regex* que retornam os tokens que vamos usar posteriormente para compilar o nosso programa.

```
%option yylineno;

[ \t]       ;
[ \n]   { lineno = lineno + 1;}
ENTRADA { return ENTRADA; }
SAIDA { return SAIDA; }
FIM { return FIM; }
END { return END; }
FACA {return FACA; }
INC { return INC; }
ZERA { return ZERA; }
ENQUANTO {return ENQUANTO; }
, {return COMMA; }
```

```
"(" {return LPAR; }
")" {return RPAR;}
"=" {return EQUAL;}
[a-zA-Z]([a-zA-Z]|[0-9])* {
    yylval.name = strdup(yytext);
    insert(yytext,strlen(yytext),0,lineno);
    return id; }
\/\/.* ;
\/\*(.*\n)*.*\*\/ ;
.           printf("unexpected %s \n", yytext);
```

- São aceitos nesse linguagem somente nome (identificadores)váriaveis com combinação de letras e número, sendo iniciados por letras
- A variável `lineno` é usada em durante toda a compilação para mostrar o número de linhas, é somada todas as vezes que se encontra um *\n*
- Não existem tipos de variáveis nessa linguagem, todas elas são consideradas **INT**
- As funções e a main definidas nesse arquivo são somente funções de teste

Bison prov.y

Aqui temos a grande parte do nosso compilador, ele lê os tokens que o programa interior gerou e faz as análises necessárias de sintaxe e semântica.

A definição dos `%token` é interessante por, usando a comando `#define YYERROR_VERBOSE 1` o *flex* além de gerar mensangens de erros de compilação muito mais claras, usa-se a string posterior à declaração do token para mostrar os tokens faltantes e/ou errados.

Mensagens de erro na análise sintática aparecem assim:

```
ERROR: syntax error, unexpected FIM, expecting Variable - Line 5
Parsing error
```

Definição dos Tokens

```
%start Program
%token ENTRADA "ENTRADA"
%token id "Variable"
%token SAIDA "SAIDA"
%token FIM "FIM"
%token FACA "FACA"
%token INC "INC"
%token ZERA "ZERA"
%token ENQUANTO "ENQUANTO"
%token COMMA ","
%token LPAR "("
%token RPAR ")"
%token EQUAL "="
%token END "End of Program"
```

Seguindo temos a definição da gramática, existem diversas funções após a leitura de tokens. A maioria tem a denominação ***push***_*, que mostra que ali é possível colocar uma linha nova no arquivo de saída em C usando as variáveis necessárias.

As vezes é necessário mudas a maneira que uma função **push** constrói a linha, se for o caso existem variáveis como **isEntrada**, que se tem um valor 1 modifica o uso da função **push_var** para uma declaração de variável de entrada, e caso seja 0, **push_var** apensar coloca o uso da variável normalmente.

Cada função **push**_* também tem as verificações necessárias como:

```
check_wasDclr(char* varName)
{
    list_t* l = lookup(varName);
    if(l == NULL || !l->st_dclr)
    {
        const char* errorStr = malloc(50*sizeof(char));
        sprintf(errorStr,"Variable %s not declared on line %d",varName, lineno);
        yyerror(errorStr);
    }
}
```

Função que verifica na tabela de símbolos se aquela variável já foi declarada. Todos as variáveis entram na tabela, mas somente aquelas que estão na **ENTRADA | SAIDA** podem ser consideradas declaradas.

Essa função é usada sempre que existe um símbolo como Token que está sendo lido.

Usando todas as mensagens com `#define BISON_VERBOSE 1` (Variável própria e não do bison), uma compilação de um arquivo tem mais ou menos essas mensagens após a sua compilação.

```
VAR NAME : X
Variable declared
VAR NAME : Z
Variable declared
VAR NAME : X
Variable declared
INC: X++
INC: X++
INC: X++
INC: X++
INC: X++
ZERA: X
ATTRIB: Z = X
Parsing done
Compilation done, no errors
```

Arquivo .prov (na Linguagem Provol-One)

```
ENTRADA X, Z
SAIDA X
INC(X)
INC(X)
ENQUANTO X FACA
INC(X)
INC(X)
INC(X)
END
ZERA(X)
Z=X
FIM
```

Arquivo .c criado no fim da compilação

PS.: O programa sempre sobreescreve no arquivo **output.c** o resultado da compilação, mesmo caso ela não seja bem sucedida.

PS2.: A compilação **NÃO** é interrompida no caso de um erro, pois assim todos os erros podem ser listados. De acordo com as minhas pesquisas uma função essencial de um compilador.

```c
#include <stdio.h>

int main()
{
    int X;
    printf("Entre com o valor de X \n");
    scanf("%d",&X);
    printf("Valor de X lido %d\n",X);
    int Z;
    printf("Entre com o valor de Z \n");
    scanf("%d",&Z);
    printf("Valor de Z lido %d\n",Z);
    X++;
    X++;
    int i = X;
    while(i != 0)
    {

    X++;
    X++;
    X++;
    i--;
    }
    X=0;
    Z = X;
    printf("Resultado final %d\n",X);
    return X;
}
```

Exemplos de Erros

Essa parte seguirá o formato

- Provol-One
- Mensagens de **Erro**

> É válido pontuar que existem outras mensagens impressas ao compilar com `#define BISON_VERBOSE 1` mas estamos focando aqui somente nas mensagens de erro impressas.

Unexpected Char

```
ENTRADA X, Y
SAIDA Z
INC(Y)
Z=&
FIM
```

```
ERROR: Unexpected char & - Line 4
ERROR: syntax error, unexpected FIM, expecting Variable - Line 5
Parsing error
```

Variable %% not Declared

```
ENTRADA X, Y
SAIDA Z
INC(Y)
Z=C
FIM
```

```
ERROR: Variable C not declared on line 5
Parsing done
Compilation errors
Error in line 5
Symbol Table
------------ ----- ----------  ----------
Name         Type  WasDclr      Lines
------------ ----- ----------  ----------
C            INT    FALSE      5
X            INT    TRUE       1    4
Y            INT    TRUE       1    3
Z            INT    TRUE       2    5
```

Incorrect comma

```
ENTRADA X,
SAIDA Z
Z=X
INC(X)
FIM
```

```
VAR NAME : X
Variable declared
ERROR: syntax error, unexpected SAIDA, expecting Variable - Line 2
Parsing error
```

Missing entrada

```
X
SAIDA Z
Z=X
INC(X)
FIM
```

```
ERROR: syntax error, unexpected Variable, expecting ENTRADA - Line 1
```

Missing FIM

```
ENTRADA X
SAIDA Z
Z=X
INC(X)
```

```
ERROR: syntax error, unexpected $end, expecting FIM - Line 4
Parsing error
```

Missing Return Variable

```
ENTRADA X
SAIDA
Z=X
INC(X)
```

```
ERROR: syntax error, unexpected =, expecting Variable or INC or ZERA or ENQUANTO -
Line 3
Parsing error
```

Warnings

Existem também alguns Warning, não interrompem a compilação e o arquivo C é válido.

Variable %% was declared but not used

```
Parsing done
Compilation done, no errors
WARNING: Variable X was declared but not used
```

```c
#include <stdio.h>

int main()
{
    int X;
    printf("Entre com o valor de X \n");
    scanf("%d",&X);
    printf("Valor de X lido %d\n",X);
    int Y;
    printf("Entre com o valor de Y \n");
    scanf("%d",&Y);
    printf("Valor de Y lido %d\n",Y);
    int C;
    printf("Entre com o valor de C \n");
    scanf("%d",&C);
    printf("Valor de C lido %d\n",C);
    int Z = 0;
    Y++;
    Z = C;
    printf("Resultado final %d\n",Z);
    return Z;
}
```

Attribution to itself

```
ATTRIB: Z = Z
WARNING: Attributing the variable Z to itself at line 4
Parsing done
Compilation done, no errors
```

```c
#include <stdio.h>

int main()
{
    int X;
    printf("Entre com o valor de X \n");
    scanf("%d",&X);
    printf("Valor de X lido %d\n",X);
    int Z = 0;
    X++;
    Z = Z;
    printf("Resultado final %d\n",Z);
    return Z;
}
```

Uso, Testes e Detalhes Técnicos

A pasta final contém os seguintes arquivos:

```
hash.c    output.c   prov.tab.c  prov.y        y.tab.c
hash.h    prov.l     prov.tab.h  teste_f.prov  y.tab.h
lex.yy.c  provolone  provtoc     teste.prov
```

O programa foi compilado usando o comando:

> bison prov.y && flex prov.l && gcc -w -o provolone prov.tab.c lex.yy.c && ./provolone teste_f.prov

Foi usado durante o desenvolvimento as funções Debug do Flex e Bison

```c
#ifdef YYDEBUG
    yydebug = 1;
#endif
```

E também as funções verbosas que imprimem o caminho todo de compilação:

> bison -t -d prov.y && flex -d prov.l && gcc -w -o provolone prov.tab.c lex.yy.c && ./provolone teste_f.prov

```
ENTRADA X
SAIDA Z
Z=X
INC(X)
FIM
```

```
--(end of buffer or a NUL)
--accepting rule at line 20 ("ENTRADA")
--accepting rule at line 18 (" ")
--accepting rule at line 32 ("X")
--accepting rule at line 19 ("
")
--accepting rule at line 21 ("SAIDA")
VAR NAME : X
Variable declared
--accepting rule at line 18 (" ")
--accepting rule at line 32 ("Z")
--accepting rule at line 19 ("
")
--accepting rule at line 32 ("Z")
VAR NAME : Z
Variable declared
--accepting rule at line 31 ("=")
--accepting rule at line 32 ("X")
ATTRIB: Z = X
--accepting rule at line 19 ("
")
--accepting rule at line 25 ("INC")
--accepting rule at line 29 ("(")
--accepting rule at line 32 ("X")
--accepting rule at line 30 (")")
INC: X++
--accepting rule at line 19 ("
")
--(end of buffer or a NUL)
--accepting rule at line 22 ("FIM")
--(end of buffer or a NUL)
--EOF (start condition 0)
Parsing done
Compilation done, no errors
```

Foram usados no trabalho as seguintes versões dos programas e sistema operacional:

> Fedora release 32 (Thirty Two)

> gcc (GCC) 10.2.1 20201016 (Red Hat 10.2.1-6)

> flex 2.6.4

> bison (GNU Bison) 3.5

Referências

O'Reilly - Flex and Bison

kranthikiran01 - subc-compiler

ANSI C Yacc grammar

Bison Docs

Developer IMB - Better error handling using Flex and Bison

IRDE France - Advanced Use of Bison