Universidade do Minho

Transcrição de aula

Disciplina	Processamento de Linguagens – 3º ano - LEI		
Secretário	Número: 47089	Nome: Ricardo Ferreira	
Data: 2011-04-11		Nº Página 5	
Turno: TP1		Nº Alunos 27	

SUMÁRIO

Construção de Analisadores Léxicos com o flex.
Introdução à construção de parsers com o Jacc.

CALCULADORA ARITMÉTICA

a) GIC

INSTRUÇÕES → 1| INSTRUÇÃO
2| INSTRUÇÃO '\n' INSTRUÇÕES

INSTRUÇÃO → 3| IMPRESSÃO
4| ATRIBUIÇÃO
5| LEITURA

IMPRESSÃO → 6| EXPRESSÃO

EXPRESSÃO → id
8| id OP EXPRESSÃO

OP → 9| '+'
10| '-'

ATRIBUIÇÃO → 11 id '=' EXPRESSÃO
LEITURA → 12 '?' id

Transcrição (folha de continuação)

! a+(2-3)

INSTRUÇÕES $\xrightarrow{1}$ INSTRUÇÃO $\xrightarrow{3}$ IMPRESSÃO $\xrightarrow{6}$ '!' EXPRESSÃO $\xrightarrow{8}$

'!' id OP EXPRESSÃO $\xrightarrow{9}$ '!' id 't' EXPRESSÃO $\xrightarrow{9,a}$ '!' 't' id OP EXPRESSÃO

~~INSTRUÇÕES $\xrightarrow{10}$ '!' id 't' id EXPRESSÃO ?~~

Tá mal aqui
O 2º id deveria ser um número.

Tudo a expressão:

EXPRESSÃO $\xrightarrow{7}$ TERMO

| TERMO OP EXPRESSÃO

TERMO $\xrightarrow{13}$ id

| 14 | NUM

| 15 | '(' EXPRESSÃO ')'

$\xrightarrow{8}$ TERMO OP EXPRESSÃO $\xrightarrow{13,9}$ id 't' EXPRESSÃO $\xrightarrow{7}$

id 't' TERMO $\xrightarrow{15}$ id 't' '(' EXPRESSÃO ')' $\xrightarrow{8}$ id 't' '(' TERMO OP EXPRESSÃO ')'

$\xrightarrow{13,9,10}$ id 't' '(' NUM '-' EXPRESSÃO ')'

Analisador Lexico

```
/* ---- Tokens.h ---- */

#define NUM 1000
#define id 1001
#define SOMA 2000
#define SUB 2001
#define SEP 2002
#define APAR 2003
#define FPAR 2004
#define IGUAL 2005
#define IN 2006
#define OUT 2007
#define FAZO -1
```

Transcrição (folha de continuação)

```
%{
#include "tokens.h"
%}
%%
[0-9]+      return NUM;
[a-z]       return ID;
\+          return SOMA;
\-          return SUB;
\=          return IGUAL;
\(          return APAR;
\)          return FPAR;
\?          return IN;
\!          return OUT;
\;          return SEP;
[ \t]       ;  //espaços
.           return ERRO;
<<EOF>>     return 0;
```

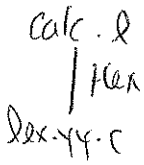
YACC (calc.y)

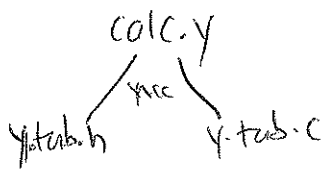
```
%token NUM ID SOMA SUB SEP APAR FPAR IGUAL IN OUT ERRO

%%
INSTRUCOES : INSTRUCAO SEP INSTRUCOES
           | INSTRUCAO
           ;

INSTRUCAO : IMPRESSAO | ATRIBUICAO | LEITURA ;
IMPRESSAO : OUT EXPRESSAO ;
ATRIBUICAO : ID IGUAL EXPRESSAO ;
LEITURA : IN ID ;
EXPRESSAO : TERMO | TERMO OP EXPRESSAO ;
TERMO : ID | NUM | APAR EXPRESSAO FPAR ;
OP : SOMA | SUB ;
%%
```

Transcrição (folha de continuação)

```
void yyerror (char *s) {
  fprintf(stderr, "Erro: %s %s", s, yytext);
}
int main () {
  if (yyparse() == 0) {
    printf ("ok\n");
  } else {
    printf ("Ocorreram erros\n");
  }
}
```

MAKEFILE

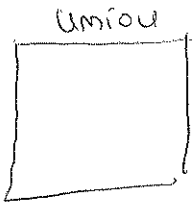





```
calc : y.tab.o lex.yy.o
       gcc -o calc y.tab.o lex.yy.o -lfl

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

y.tab.c y.tab.h : calc.y
       yacc -d calc.y

lex.yy.o : lex.yy.c y.tab.h
       gcc -c lex.yy.c

lex.yy.c : calc.l
       flex calc.l
```

Transcrição (folha de continuação)

flex



yacc

```
union A {
  int a;
  char *b;
};
```

O valor ou é do tipo a ou do tipo b.

```
struct x {
  union A {
    int a;
    char *b;
  } info;
  int tipo;
} X1;

if (X1.tipo == num)
  ... X1.info.a ...
```