Escrita de compiladores em Prolog

- Bom exemplo de vantagens de programação declarativa
- Exemplo de área de aplicação para motivar restante do curso

Escrita de compiladores em Prolog: exemplo de linguagem fonte

- Comandos:

```
<name> := <expression>
if <test> then <statement> else <statement>
while <test> do <statement>
begin <statement>; <statements> end
```

- Expressões:

```
<constant>
<name>
<expression> + <expression>
etc.
```

- Testes (operações relacionais)

```
<expression> = <expression>
etc.
```

Escrita de compiladores em Prolog: linguagem alvo

- Transferência de controle

```
JUMP <address>
JUMPEQ <address>
JUMPLT <address>
etc.
```

- Aritmética e operações de memória:

```
LOADC <integer>
ADDC <integer>
SUBC <integer>
etc.
LOAD <address>
STORE <address>
ADD <address>
SUB <address>
```

`etc.`

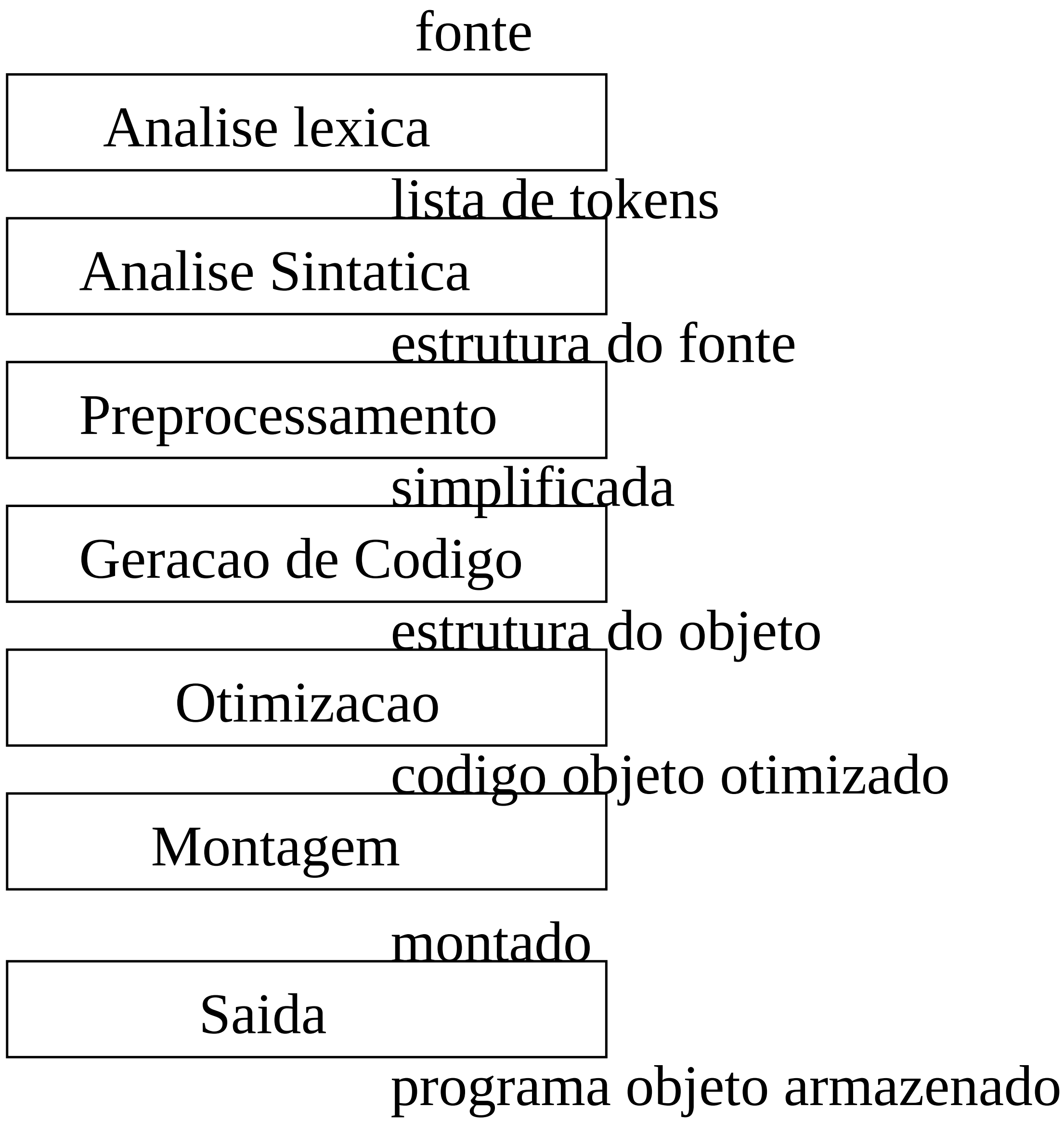
- Estas instruções assumem que a máquina de execução tem um único acumulador que um operando implícito da maioria das intruções.

Exemplo de Compilação

```
Programa Fonte                  Codigo Fonte


count := 1;                              loadc 1
                                         store count


result := 1;                             loadc 1
                                         store result


while count < value do          label1:  load count
                                         sub value
begin                                    jumpge label2
   count := count + 1;                   load count
                                         addc 1
                                         store count
   result := result * count;             load result
```

```
                mul count
                store result
                jump label1
        label2: .....
end
         count: integer
        result: integer
         value: integer
```

Estágios da Compilação



Representação da linguagem fonte (sintaxe abstrata)

```
count := count + 1          while x > 0 do...
```

Representação Prolog:

```
assign(name(count),            while(test(>,
       expr(+,name(count),                 name(x),
              const(1)))                   const(0)),
```

....)

Representação da linguagem objeto

```
LOAD count
ADDC 1
STORE count
```

Representação Prolog

```
(instr(load,count); instr(addc,1); instr(store,count))
```

Compilação de um comando de atribuição

Fonte:

```
<name> := <expression>
```

Objeto:

```
<expression code>
STORE <address>
```

Regra Prolog:

```
encodestatement(assign(name(X),Expr),D,(Exprcode;instr(store,Addr)) :-
  lookup(X,D,Addr),
  encodeexpr(Expr,D,Exprcode).
```

Compilação de expressões

Fonte	Objeto
<constant>	LOADC <constant>
<name>	LOAD <address>
<expression>+<constant>	<expression code> ADDC <constant>
<expression>+<name>	<expression code> ADD <address>

Compilação de Expressões

```
encode_expr(const(C),_,instr(loadc,C)).
encode_expr(name(X),D,instr(loadc,Addr) :-
   lookup(X,D,Addr).
encode_expr(expr(Op,Expr1,Expr2),D,(Expr1Code;Insts)) :-
   apply(Op,Expr2,D,Insts),
   encode_expr(Expr1,D,Expr1Code).

apply(Op,const(C),_,instr(Opcode,C)) :-
   literalop(Op,Opcode).
apply(Op,name(X),D,instr(Opcode,Addr)) :-
   memoryop(Op,Opcode),
   lookup)X,D,Addr).

literalop(+,addc).   memoryop(+,add).
....                 ....
```

Compilação do IF

Codigo Fonte:

```
IF <test> THEN <then> ELSE <else>
```

Codigo Objeto:

```
           <test code>
           <then code>
           JUMP <label2>
<label1>: <else code>
<label2>: ....
```

Compilação do IF

Fonte:

```
IF <test> THEN <then> ELSE <else>
```

Objeto:

```
     <test code>
     <then code>
<label1>:
     <else code>
<label2>:
```

Compilação do IF

Prolog:

```
encodestatement(if(Test,Then,Else),D,(Testcode;
                                      Thencode;
                                      instr(jump,L2));
                                      label(L1);
                                      Elsecode;
                                      label(L2))
               ) :-
    encodetest(Test,D,L1,Tescode),
    encodestatement(Then,D,Thencode),
    encodestatement(Else,D,Elsecode).
```

Compilação de operações relacionais

Fonte:

```
<expression> = <operand>
```

Objeto:

```
<evaluate expression-operand>
JUMPNE <label>
```

Prolog:

```
encodetest(test(Op,Expr,Operand),D,Label,(Evaluation;instr(Jump,Label)))
:-
    encode_expr(expr(-,Expr,Operand),D,Evaluation),
    jump_otherwise_op(Op,Jump).

jump_otherwise_op(=,jumpne).
jump_otherwise_op(<,jumpgt).
```

...