6. Geração de código e gestão de erros #teórico-prático

geração de código

um compilador pode traduzir o código fonte de uma linguagem noutra

estratégia identificar padrões de geração de código e após a análise semântica, percorre novamente a árvore sintática, gerando o código nos pontos apropriados

é necessário definir o contexto de geração de código para cada elemento e depois garantir que o mesmo é compatível com todas as utilizações do mesmo

sendo a maioria das linguagens textuais, os padrões de geração de código consistem em padrões de geração de texto

String Template

biblioteca que fornece solução estruturada para a geração de código textual

www.stringtemplate.org

```
import org.stringtemplate.v4.*;
```

padrão de texto permite o preenchimento de buracos delimitados por <expr> posteriormente à definição do padrão do texto, delegados em diferentes partes do gerador de código

ST

exemplo

```
ST hello = new ST("Hello, <name>");
hello.add("name", "World");
S.out.println(hello.render());
```

string template group permite agrupar os padrões numa espécie de funções

ST Group
STGroupString

exemplo

```
STGroup group = new STGroupString(
    "assign(var, expr) ::= \"<var> = <expr>;\"" );
ST assign = group.getInstanceOf("assign");
assign.add(—);
assign.render();
```

cada função pode ser colocada num ficheiro .st

assign.st assign(var, expr) ::= "<var> = <expr>;"

STGroupDir usage STGroup grp = new STGroupDir(".");
ST assign = grp.getInstanceOf("assign");

ou criados ficheiros modulares com várias funções .stg

STGroupFile usage STGroup grp = new STGroupFile("templates.stg");
ST assign = grp.getInstanceOf("assign");

VIP tipos de template
" — " única linha
<< ≡ >> multiplas linhas, preservando \t e \n
<% ≡ %> " " , ignorando \t e \n

String template estruturas

dicionários

typeValue ::= [
"integer" : "int",
default : "null"
]

instruções condicionais

stats(stat) ::= <<
<if(stat)><stat; separator="\n"><endif>
>>

funções definição de padrões com recurso a outros (como se fossem funções)

module(name, stat) ::= <<
public class <name> {
public static void main(String[] args) {
<stats(stat)>
}
}
>>

listas para concatenar texto e argumentos de padrões

binaryExpression(type,var,e1,op,e2) ::=
"<decl(type,var,[e1," \" ",op," \",e2])>"

GCGE

String Template geração de código não expressões

exemplo

```
typeValue ::= [ ... ]
init(value) ::= "<if(value)> = <value> <endif>"
decl(type, var, value) ::=
  "<typeValue.(type)> <var> <init(value)>;"
binaryExpression(type,var,e1,op,e2) ::= ....
```

geração de código intermédio

TAC codificação de triplo endereço

Padrão para expressões muito utilizado na geração de código de baixo nível

instruções na forma $x = y\ op\ z$ são as mais comuns

existem no entanto chamadas e de controlo de fluxo

no máximo têm três operandos

cada uma realiza uma operação elementar

exemplo a+b*c

```
t1 = c;        t4 = a;
t2 = b;        t5 = t4 + t3;
t3 = t1 * t2;
```

obs embora use muitos registos, é codificável em linguagens de baixo nível (pode ser corrigido numa fase posterior de otimização)

endereços nome do código fonte (end. memória)
constante (valor literal)
nome temporário criado na decomposição TAC

instruções		
	x = y op z	para x
	x = op y	call p
	x = y	y = call p
	goto L	return y
	label L	[arrays]
	if x goto L (iffalse)	[ponteiros]
	if x <op> y goto L	

controlo de fluxo	if(cond){ A }	iffalse cond goto 11
	else { B }	A
		goto 12
		label 11
		B
		label 12

while(cond) { — 4

label 11
IFFalse cond goto 12
—
goto 11
label 12

Funções usar stack
push / pop

gestão de erros

no ANTLR4 a gestão de erros é definida na interface ANTLRErrorListener e implementado na classe base BaseErrorListener

na presença de erros sintáticos é invocado o método syntaxError que podemos reescrever (override)

obs: Para passarmos todos os erros de reconhecimento de tokens para a análise sintática, podemos acrescentar como última regra da gramática
ERROR: . ;

recuperar de erros
o ANTLR4 implementa, o que permite ao analisador sintático continuar a processar depois de detetado um erro, de forma a identificar o maior nr. de erros em cada compilação

alterar estratégia
as estratégias são definidas pela interface ANTLRErrorStrategy, existindo duas implementações na biblioteca de suporte
DefaultErrorStrategy por defeito ativa, implenta recuperação de erros
BailErrorStrategy terminação imediata qnd surge primeiro erro
esta pode ser alterada no Parser, concretamente no método .setErrorHandler