Compiladores - Analisador Sintático

Lucas Pandolfo Perin
Gabriel Gaspar Becker
Carlos Fernando Montibeller da Silva

Implementação do arquivo de extensão “jj” baseado no livro “Como Construir um Compilador”.

Foram feitas as alterações no arquivo de extensão “jj” para que, com o uso do parser, seja possível consumir tokens a partir de um erro sintático e dar continuidade na análise. Assim é possível que o parser faça a análise total do programa, detectando o maior número de erros, facilitando correções. Adicionalmente, foram incluidas as classes necessárias do pacote *recovery* para que o parser possa executar as novas funções.

O método utilizado é o método da ressincronização (conhecido também como método do pânico). Para fazer a ressincronização, uma produção que detecta um erro sintático deverá consumir todos os tokens até achar um token válido para continuar a análise. Isso é feito utilizando os tokens *follow* da produção em questão, por exemplo:
Caso exista mais de uma declaração de variável dentro de uma classe e uma delas possua um erro sintático, os tokens dessa declaração incorreta são ignorados até que comece a próxima declaração de variável.

Abaixo, segue as produções que diferem da implementação X++:

```
void atribstat(RecoverySet g) throws ParseEOFException : //comando_atribuicao
{
RecoverySet f1 = new RecoverySet(ASSIGN).union(g);
}
{
try {
        lvalue(f1) <ASSIGN> ( alocexpression(g) | expression(g) [conditional(g)] )
}
catch (ParseException e)
{
  consumeUntil(g, e, "atribstat");
}
}

void conditional(RecoverySet g) throws ParseEOFException :
{
RecoverySet f1 = new RecoverySet(DDOT).union(g);
}
```

```
{
try {
   <QUESTIONMARK> ( expression(f1) | alocexpression(f1) ) <DDOT> ( expression(f1) |
alocexpression(f1) )
}
```

```
void whilestat(RecoverySet g) throws ParseEOFException :
{
RecoverySet f1 = new RecoverySet(RPAREN).union(g);
}
{
  try
{
   < WHILE > <LPAREN> expression(f1)    < RPAREN > statement(g)
}
catch(ParseException e)
{
  consumeUntil(g, e, "whilestat");
}
}

void switchstat(RecoverySet g) throws ParseEOFException :
{
RecoverySet f1 = new RecoverySet(RPAREN).union(g),
            f2 = new RecoverySet(RBRACE).union(g);
}
{

try
{
   < SWITCH > < LPAREN > expression(f1) < RPAREN > < LBRACE > caselist(f2) < RBRACE >
}
catch(ParseException e)
{
   consumeUntil(g, e, "switchstat");
}
}


void caselist(RecoverySet g) throws ParseEOFException :
{
}
```

```
{
  try
  {

    casestat(g) [caselist(g)]
|
    <DEFAULT_CASE> <DDOT> statement(g)
}
catch(ParseException e)
{
    consumeUntil(g, e, "caselist");
}
}

void casestat(RecoverySet g) throws ParseEOFException :
{
}
{
  try
  {
    < CASE > factor() < DDOT > statement(g)
  }
catch(ParseException e)
{
    consumeUntil(g, e, "casestat");
}
}
```