Hacettepe University
Computer Science and Engineering Department

Akıllı Evler İçin
Yeni Bir Programlama Dili Tasarımı
2. Kısım

İlknur Kabadayı
20724821

e-mail : ikabadayii@gmail.com

SweetHome Dilinin BNF Gösterimi

<program> := <function> | <program> <function>
<function> := func <identifier> <left-bracket> <parameter-list> <right-bracket> <colon> <return-type> <block>
<block> := <block-start> <statement-list> <block-end>
<return-type> := <data-type>
<parameter-list> := <empty>
|<data-type> <identifier>
| <parameter-list> <data-type> <comma> <identifier>
<identifier> := <letter>
| <identifier> <letter>
| <identifier> <digit>
<statement-list> := <statement> ; | <statement-list > <statement> ;
<statement> := <declaration-statement>
| <assign-statement>
| <conditional-statement>
| <loop-statement>
| <function-calling>
<declaration-statement> := <basic-data-type> <identifier>
| <declaration-statement> <assign-operator> <rvalue>
| <declaration-statement>, <identifier>
| <sensor-type> <identifier> <assign-operator> sensor <sensor-code>
| <matrix-type> <identifier> <l-bracket> <dimension-format> <r-bracket>
| <matrix-type> <identifier> <l-bracket> <dimension-format> <r-bracket> <assign-operator> <block-start> <identifier-list> <block-end>
<dimension-format> := <integer-liretal> | <dimension-format> , <integer-literal>
<assign-statement> := <lvalue> <assign-operator> <rvalue>
| <lvalue> ++
| <lvalue> --
| <lvalue> **
|<identifier> <l-bracket> <dimension-format> <r-bracket> <assign-operator> <rvalue>
<lvalue> := <identifier>
<assign-operator> := << | +<< | -<< | *<< | /<< | %<<
<rvalue> := <arithmetic expression> | <function-calling>
<arithmetic-expression> := <term>
| <arithmetic expression> + <term>
| <arithmetic expression> - <term>
<term> := <primary>
| <term> * <primary>
| <term> / <primary>
| <term> % <primary>

<primary> := <constant> | <identifier>
<constant> := " <string-literal> " | <number-literal> | <float-literal> | '<char-literal>'
<operator> := + | - | * | / | %
<conditional-statement> := <if-statement> | <select-statement> | <interval-statement>
<if-statement> := if <boolean-expression> <block>
| <if-statement> else if <block>
| <if-statement> else <block>
<select-statement> := select <block-start> <select-case-statement> <block-end>
| select <block-start> <select-case-statement> <default-conditional-statement> <block-end>

<select-case-statement> := case <boolean-expression> <colon> <statement-list>
| case <boolean-expression> <colon> <statement-list> break
| case boolean_expression <colon> <statement_list> <select_case_statement>
| case boolean_expression <colon> <statement_list> break <select_case_statement>
<interval-statement> := interval <left-bracket> <number-type> <right-bracket> <block-start> <interval-case-statement>
| interval <left-bracket> <number-type> <right-bracket> <block-start> <interval-case-statement><default-conditional-statement> <block-end>

<interval-case-statement> := case <number-type> to <number-type> <colon> <statement-list>
|case <number-type> to <number-type> <colon> <statement-list> break
| case <number_type> to <number_type><colon><statement_list> break <interval_case_statement>
| case <number_type> to <number_type> <colon> <statement_list> <interval_case_statement>
<default-conditional-statement> := default <colon> <statement-list> break
| default <colon> <statement-list>
<loop-statement> := <while-loop> | <until-loop> | <do-while-loop> | <do-until-loop> | <for-loop> | <foreach-loop>
<while-loop> := while <left-bracket> <boolean-expression> <right-bracket> <block>
<until-loop> := until <left-bracket> <boolean-expression> <right-bracket> <block>
<do-while-loop> := do <block> while <left-bracket> <boolean-expression> <right-bracket> <block>
<do-until-loop> := do <block> until <left-bracket> <boolean-expression> <right-bracket> <block>
<for-loop> := for <left-bracket> <for-inititation> ; <boolean-expression> ; <assign-statement> <right-bracket> <block>
<for-inititation> := <declaration-statement> | <assign-statement> |<empty>
<foreach-loop> := foreach <left-bracket> <foreach-loop-type> <identifier> in <matrix-type-var> <right-bracket> <block>
<foreach-loop-type> := <basic-data-type> | <sensor-type>
<function-calling> := <identifier> <left-bracket> <identifier-list> <right-bracket> | <identifier> . <function-calling>
<identifier-list> := <empty>
| <call-parameter>
| <identifier-list> , <call-parameter>
<call-parameter> := <identifier>
| constant
| <identifier> <l-bracket> <integer-literal> <r-bracket>
| <identifier> <l-bracket> <identifier> <r-bracket>
<boolean-expression> := true | false | <identifier> | <logical-expression>
<logical-expression> := <boolean-expression> <boolean_op> <boolean-expression-sub>
| <boolean-expression> <relation_op> <boolean-expression-sub>
<boolean_op> := and | or
<boolean-expression-sub> := true | false | <identifier> | <constant>
<relation-operation> := < | <= | > | >= | = | < >
<matrix-type-var> := <identifier>
<data-type> := <basic-data-type> | <complex-data-type>
<basic-data-type> := <void-type> | <bool-type> | <number-types> | <string-type>
<void-type> := void
<bool-type> := bool
<number-types> := <int-type> | <float-type>
<int-type> := int
<float-type> := float
<string-type> := string
<complex-data-type> := <sensor-type> | <matrix-type>
<sensor-type> := sensor
<matrix-type> := matrix
<equal-sign> := =
<right-bracket> :=)
<left-bracket> := (

<r-bracket> :=]
<l-bracket> := [
<block-start> := {
<block-end> := }
<comma> := ,
<colon> := :
<number-literal> := <integer-literal> | <float-literal>
<integer-literal> := <digit> | <integer-literal> <digit>
<float-literal> := <integer-literal>. <integer-literal>
<string-literal> := <letter> | <string-literal> <letter>
<char-literal> := <letter>
<letter> := <big-letter> | <small-letter>
<small-letter> := a | b | c | ç | d | e | f | g | ğ | h | ı | i | j | k | l | m | n | o | ö | p | q | r | s | ş | t | u | ü | v | w | x | y | z
<big-letter> := A | B | C | Ç | D | E | F | G | Ğ | H | I | İ | J | K | L | M | N | O | Ö | P | Q | R | S | T | U | Ü | V | W | X | Y | Z
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<empty> :=

Proje1'den Farklılık Gösteren Kısımlar

Genel olarak ilk aşamada tasarladığım dil yapısına sadık kaldım, ancak yacc tarafını gerçekleştirirken birkaç durumda düzeltemediğim çakışmalarla karşılaştığım için küçük değişiklikler yaptım.

İlk kısımda matris veri türünü `matrix dizi_adı (3,5);` şeklinde tanımlamıştım. Ancak kimi durumlarda fonksiyon çağırma işleci olan `()` 'la karıştığı için matris tanımımı şu şekilde değiştirdim:

```
matrix dizi_adı [3,5];
```

Matris tipindeki bu değişiklik matrislere atama yaparken de geçerlidir. Ayrıca koşullu ifadelerin ve döngülerin bitimi için de noktalı virgül (;) tanımladım. Bu şekilde yapmamın sebebi koşullu ifadeleri ve döngüleri de statement olarak tanımlamam ve her statement'ın da noktalı virgülle bitmesinin zorunlu olmasıdır.

Ayrıca BNF tanımımı da YACC'da karşılaştığım conflictleri düzeltecek şekilde değiştirdim ve son halini yukarıda tekrar tanımladım. Tasarım olarak çok değişiklik yapmasam da bazı mantıksal hataları düzeltmeye özen gösterdim.

YACC

YACC, bilgisayar bilimlerinin önemli dallarından birisi olan dil tasarımı ve dil geliştirilmesi sırasında (compiler theory) sıkça kullanılan bir kod üretici programdır. YACC basitçe dildeki sözdizim (syntax) tasarımı için kullanılır ve tasarladığımız dildeki kelimelerin sıralamasının istediğimiz şekilde girilip girilmediğini kontrol eder. Aynı zamanda sıralamadaki her kelimenin anlamını da yacc marifetiyle belirleyebiliriz.

YACC temel olarak BNF (Backus Normal Form) kullanarak cümle dizimini belirtmektedir. LEX ile birlikte kullanıldığında bir dil tasarımının neredeyse yarısı olan lexical (kelime) ve syntax (cümle) analizi tamamlanmış olur. Bundan sonra dildeki her kelime ve cümle diziliminin anlamını (semantic) kodlamak kalır. Biz bu ödevle birlikte kelime ve cümle analizini tamamladık.

- Yacc da önemli olan girdi kütüğümüzün tasarladığımız dile uygun olup olmamasını kontrol etmek ve conflict durumlarını ortadan kaldırmaktı.
- İşlem önceliğini klasik yöntemle çözdüm:

<arithmetic-expression> := <term>
| <arithmetic expression> + <term>
| <arithmetic expression> - <term>
<term> := <primary>
| <term> * <primary>
| <term> / <primary>
| <term> % <primary>

<primary> := <constant> | <identifier>

- Genel olarak karşılaştığım conflictler <non-terminal> := <terminal > | <nonterminal > <terminal> <nonterminal> şeklinde yazdığım için meydana geldi bu kısımları düzenlemek zor olmadı örneğin ;

  <statement-list> := <statement> | <statement-list > ; <statement-list>

  yerine

  <statement-list> := <statement> ; | <statement-list > <statement> ;

- Örnek deneme kütüğünde yazılmış olan yanlış ifadeler olması durumda hatalı satırın satır numarasını ve ayrıca kolon numarasını gösterdim. Yani satırın neresinde hata olduğunu söyleyecek şekilde programladım. Bir de eğer programcı block comment karakterleriyle tek satırlık bir yorum tanımlarsa ve daha sonra da anlamsız seyler yazmaya devam ederse bile programım tam olarak nerede hata olduğunu söyleyebiliyor.

- Eğer main yazmasaydım lex ve yacc kendi mainlerini üretecekti ama main'i kendim yazdım ve yyparse fonksiyonunun dönüş değerini kontrol ettim hatalı durumda hata var, yoksa parse başarılı mesajını gösterdim.

References

- http://www.bilgisayarkavramlari.com/2008/12/12/yacc/
- O'Reilly - Lex and Yacc
- http://www.cs.utk.edu/~eijkhout/594-LaTeX/handouts/parsing/yacc-tutorial.pdf