Bütünleme Projesi - Programlama Dili Dizaynı

Eren Neyiş
185541026

1 Amaç

Projenin amacı derste öğrenilen temel kavramları kullanarak basit bir programlama dili oluşturmaktır. Derste işlenilen konularla paralel olarak dilin nasıl dizayn edildiği anlatılacaktır.

Figürdeki, Compilers: Principles, Techniques and Tools kitabından yardım alınmıştır.

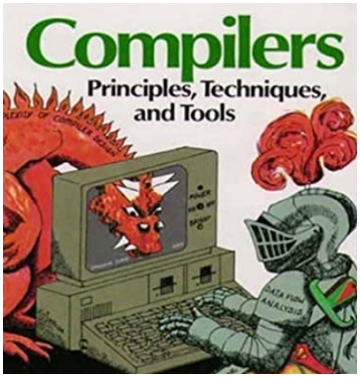

Figure 1: Compilers, principles, techniques, and tools / Alfred V. Aho

2 Sentaks ve Semantiği Tanımlama

2.1 Sentaks

Dizayn edilen dilde basit bir sentaks özelliği olarak ifadelerin veya deyimlerin sonunda noktalı virgül bulunur. Değişkenler sadece tam sayılar üzerinde 4 işlem olduğundan explicit olarak değişkenlerin başına data tipi belirtmek zorunlu değildir. Değişkenlerlerin hepsi aslında integer tipindedir ancak dilin yapısına bu integer keywordü aktarılmayarak dil sadece tutulmuştur. Her deyimin veya ifadenin sonuna noktalı virgül konulması gerekmektedir.

2.2 Semantik

Değişkenler anlık olarak tanımlanabilir. Bir değişkeni kullanmak için daha önceden tanımlamaya gerek yoktur.

3 Metinsel Sözdizim

3.1 Lexeme ve Tokenlar

Sözdizimler tokenlar olarak tanımlanmıştır. Bildiğimiz üzere lexeme'ler dilin en küçük parçalarıdır. Tokenlar ise lexemelerin tipini belirtir. Dizayn edilen dile toplama (+), çıkarma (-), bölme (/) ve çarpma (*) tokenları sırasıyla toplama işlemcisi, çıkarma işlemcisi, bölme işlemcisi ve çarpma işlemcisi olarak tanımlanmıştır.

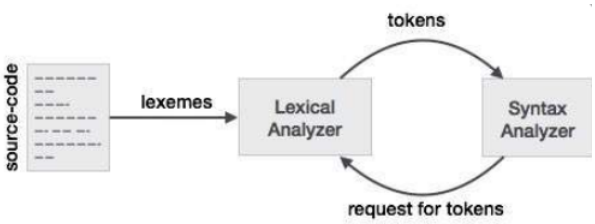

Figure 2: Lexemes and tokens in process

3.2 Değişkenler

Dil değişkenleri tek ingilizce harf olarak dizayn edilmiştir. Değişkenler yalnızca tek harf ingilizce karakter içeren harften oluşabilir. Örneğin "a" sentaktik olarak doğruyken "eren" lexer aşamasından geçemeyerek sentaks hatası ile sonuçlanacaktır.

4 Lexer

Bu bölümde lexer'dan ve lexer'ı oluşturan regular expression kurallarından bahsedilecektir. Bildiğimiz üzere programlama dili yazmanın ilk aşaması Lexer'dır. Bu aşamada program anlam ifade eden küçük parçalara ayrılır ve parsing için geçerli bir girdi olur. Lexer ı implement ederken eski bir UNIX tool u olan lex/flex'ten yararlanılmıştır.

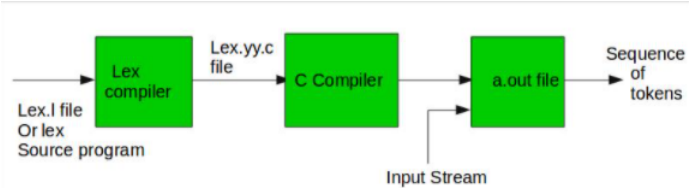

Figure 3: Flow of lexer tool lex/flex

4.1 Regular Expressions

Dizayn edilen dil değişken ismi olarak sadece küçük harf kabul edildiğinden regular expression formu yalnızca küçük harfleri kapsamaktadır. Dilde sayılar bir veya birden fazla basamak olacak şekilde tanımlanmıştır. Bu tanım aslında bir finite state automata tanımlar. Bunu implement etmek yerine basitçe regular expression olarak tanımlanmıştır.Özel operasyonlar (toplama gibi) ayrı olarak tanımlanmıştır. Boşluk ve tab karakterleri de göz ardı edilmiştir.

```
%{
#include "y.tab.h"
void yyerror (char *s);
int yylex();
%}
%%
"yazdir"           {return dump;}
"cik"            {return exit_shell;}
[a-z]         {yylval.id = yytext[0];
               return identifier;}
[0-9]+              {yylval.num = atoi(yytext);
               return number;}
[ \t\n]              ;
[-+*/=;]              {return yytext[0];}
.                    {ECHO; yyerror ("unexpected character");}

%%
int yywrap (void) {return 1;}
```

4.2 Özel Operasyonlar

Dilde yapılan herhangi bir dört işlemden sonra herhangi bir değişkenin içeriğini görmek için konsola yazmak için bir keyword tanımlamak gerekir. Bu keyword ün ismi "yazdir" olarak belirlenmiştir. Ayrıca programı sonlandırmak için bir başka keyword olan "cik" komutu kullanılabilir. Bunlar lexer'da özel karakterler olarak tanımlanmıştır.

5 Parser

Lexing işlemi lexer aracı yardımıyla yapılmıştı. Dil Lexer'dan başarılı olarak geçtikten sonra parser aşamasına gelir. Parser dizayn ettiğimiz kuralların yani dilin kendisinin doğru formda olup olmadığını kontrol eder ve form doğru ise sembol tablosunu yükler ve günceller. Gene bir UNIX aracı olan yacc/bison parsing işlemi olarak kullanılacaktır.

5.1 BNF Rules

Yacc dili Context Free Grammar olarak tanımlamamıza olanak sağlar. Kurallar BNF formuna yakın bir şekilde tanımlanır ve aşağıdaki gibidir.

```
%%
#include "y.tab.h"
start_sym   : assignment ';'  {;}
     | exit_shell ';'   {exit_program();}
     | dump expression ';'      {dump_screen($2);}
     | start_sym assignment ';' {;}
     | start_sym dump expression ';' {dump_screen($3);}
     | start_sym exit_shell ';' {exit_program();}
       ;

assignment : identifier '=' expression { update_symbol_table($1,$3); }
        ;
expression    : term               {$$ = $1;}
        | expression '+' term       {$$ = addition($1, $3);}
        | expression '-' term       {$$ = substraction($1, $3);}
        | expression '*' term       {$$ = mult($1, $3);}
        | expression '/' term       {$$ = divide_check($1, $3);}
        ;
term    : number             {$$ = $1;}
        | identifier       {$$ = symbol_index($1);}
       ;

%%
```

5.2 Bottom-Up/Shift Reduce Parsing

Bison/Yacc aracı Bottom-Up veya bir başka adıyla Shift Reduce Parsing tekniğini kullanılır. Bu teknikteki temel operasyonlar shift, reduce ve accept'tir. Shift adımı, giriş akışında bir sembol ilerler. Bu kaydırılan sembol, yeni bir tek düğümlü ayrıştırma ağacına dönüşür. Reduce adımı, son ayrıştırma ağaçlarından bazılarına tamamlanmış bir dilbilgisi kuralı uygular ve bunları yeni bir kök sembolüyle tek bir ağaç olarak birleştirir. Dilde oluşturulan ağacın görselleştirilmiş şekli aşağıdaki gibidir.

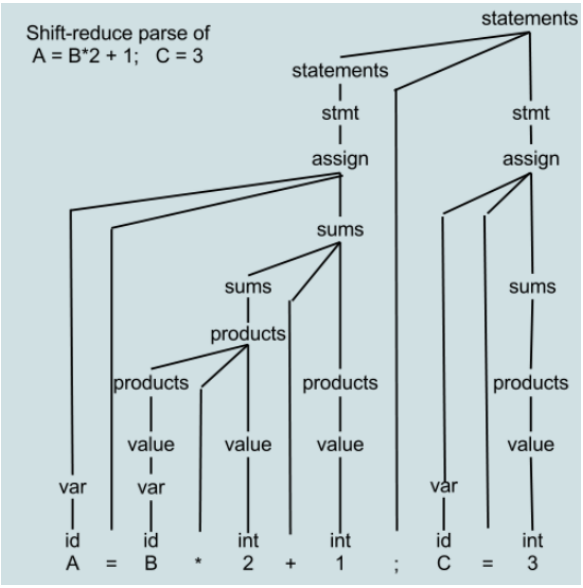

Figure 4: Shift Reduce Ayrıştırma Ağacı

5.3 Sembol Tablosu

Oluşturulan dilde sembol tablosu sınırlıdır çünkü değişkenler sadece ingilizce alfabesinde olan harflerden oluşmaktadır. Bunun için 26 elemanlık bir integer dizisi tanımlanmıştır.

References

[1] https://tldp.org/LDP/LG/issue87/ramankutt

[2] https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/

[3] https://stackoverflow.com/questions/14954721/what-is-the-difference-between-a-token-and-a-lexeme

[4] https://www.amazon.com/Compilers-Principles-Techniques-Alfred-Aho/dp/0201100886