WYŻSZA SZKOŁA
INFORMATYKI I ZARZĄDZANIA
z siedzibą w Rzeszowie

WYDZIAŁ INFORMATYKA STOSOWANA

Kierunek: INFORMATYKA

Michał Bąk
Nr albumu studenta 32366

Analizator PHP

PROJEKT

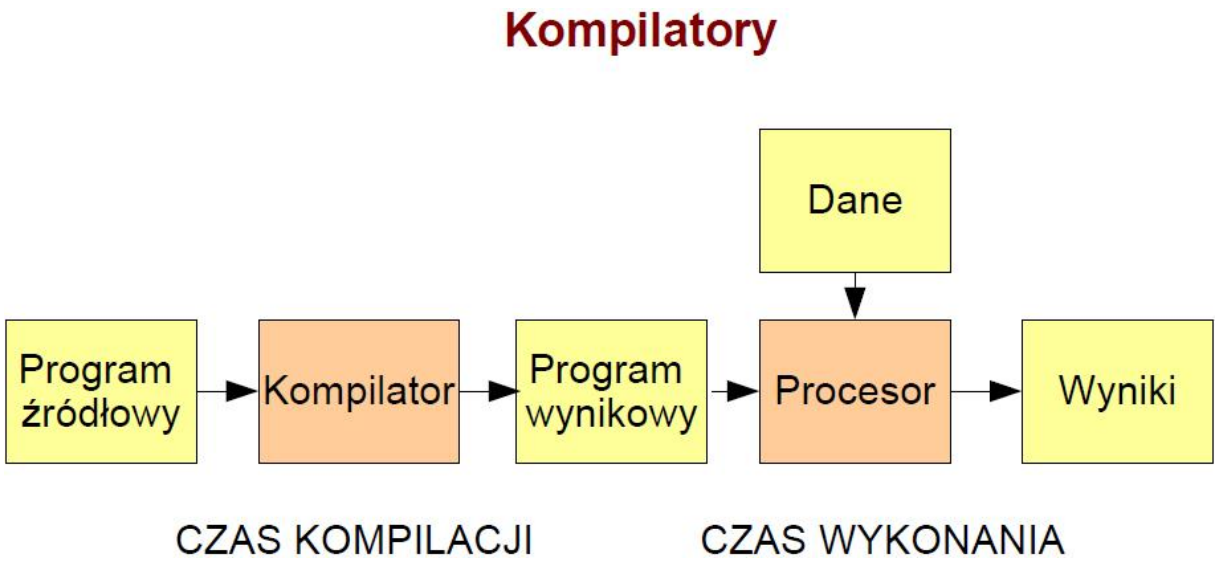
Rzeszów 2010

Wstęp

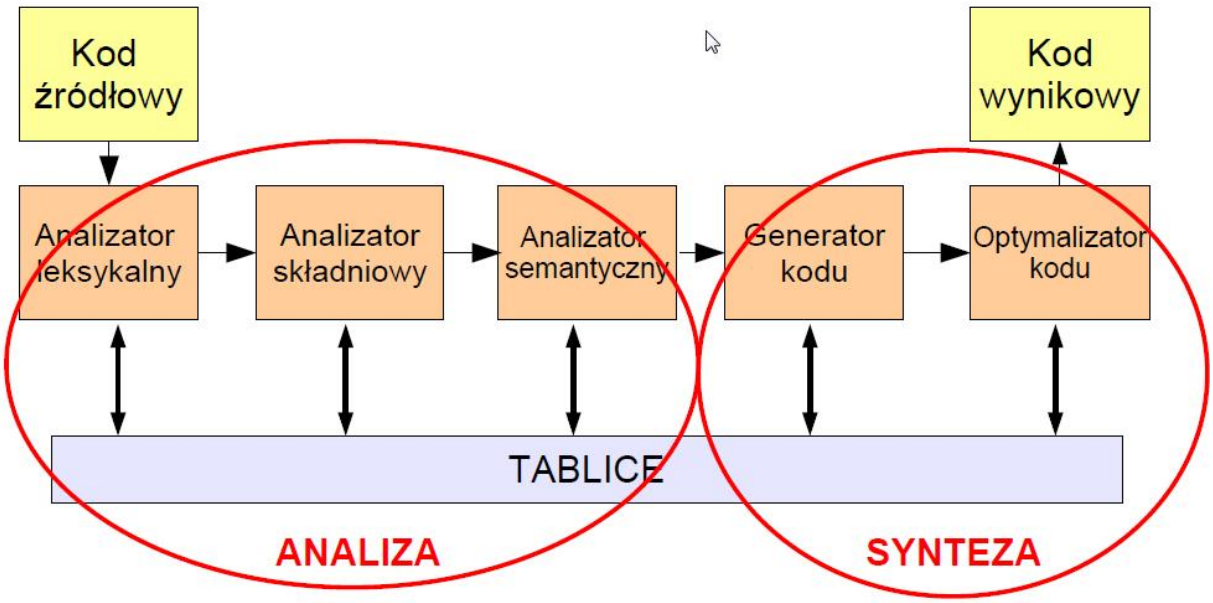
Projekt został stworzony na zaliczenie przedmiotu „Budowa Kompilatorów”. Analizator pozwala na proste przeanalizowanie składni języka programowania PHP i stwierdzeniu czy wprowadzony kod do terminala został poprawnie napisany. JavaCC jest generatorem parserów dla języka Java, operuje na gramatykach typu LL(k).

Podstawowe pojęcia

Syntaktyka (składnia) języka, część gramatyki definiująca reguły budowania poprawnych wyrażeń języka.
Semantyka języka, przyporządkowanie znaczeń poprawnym składniowo wyrażeniom języka.
Alfabetem *V* nazywamy każdy skończony niepusty zbiór symboli.
Słowem (zdaniem) *σ* nad alfabetem *V* nazywamy dowolny skończony ciąg symboli tego alfabetu.
-*V** - zbior wszystkich słów nad alfabetem *V*, gdzie:
V+ - zbior wszystkich niepustych słów nad
alfabetem *V*, ε - słowo puste.
Językiem *L* nad alfabetem *V* nazywamy dowolny, niepusty podzbiór zbioru *V**
$L \subset V^*$, $L \neq \emptyset$
Gramatyka formalna służy do formalnego określenia reguł generowania zdań danego języka z symboli należących do określonego alfabetu.
Kompilator translator transformujący kod w języku wysokiego poziomu do kodu maszynowego, asemblerowego lub pośredniego.
Czas, w którym wykonywana jest transformacja nazywany jest czasem kompilacji.
Program wynikowy wykonywany jest w tzw. czasie wykonania.



Ogólna budowa kompilatora



Interpreter translator, w którym interpretacja kodu źródłowego odbywa się w czasie wykonania. Interpreter nie generuje programu wykonywalnego.
Analizator leksykalny pośrednik pomiędzy programem źródłowym i analizatorem Składniowym. Analizator leksykalny przez analizę znak po znaku kodu źródłowego dzieli go na podstawowe części (tokeny, elementy leksykalne, leksemy), np. nazwy zmiennych, stałych, etykiet, słowa kluczowe, operatory, itd. oraz na białe znaki i komentarze. Białe znaki i komentarze są ignorowane, zaś wydzielone leksemy przekazywane są do dalszych etapów analizy.
Analizator składniowy określa w jaki sposób program źródłowy (w postaci tokenow) ma zostać zdekomponowany na swoje części składowe.
W analizie składniowej następuje proces grupowania tokenów w bardziej ogólne klasy składniowe (np. wyrażenia, zdania, funkcje).
Analizator składniowy tworzy tzw. drzewo składniowe (liśćmi są tokeny, pozostałe węzły reprezentują typy klas składniowych).
Analizator semantyczny (znaczeniowy) ma za zadanie określić znaczenie programu źródłowego.
Danymi dla analizatora semantycznego są drzewa składniowe wygenerowane przez analizator składniowy.
Skaner jest analizatorem leksykalny.
Parser jest analizatorem składniowy.
Gramatyka regularna w regułach produkcji po prawej stronie występuje zawsze jeden symbol terminalny, a ewentualny symbol nieterminalny występuje po jego prawej (lub lewej) stronie . Wyrażenia regularne używa się przy skanowaniu (analizie leksykalnej).
Gramatyka bezkontekstowa w regułach produkcji lewa strona jest zawsze pojedynczym symbolem nieterminalnym. Gramatyka bezkontekstowa używa się przy parsowaniu (analizie składniowej).

Podstawowa notacja do budowy wyrażeń regularnych

- symbole z alfabetu danego języka
- operator alternatywy I
- operator konkatenacji .(kropka)
- operator powtórzenia *
- [abc] jest równoważne a | b | c
- [a-k] jest równoważne a | b | ... | k
- MN jest równoważne M.N
- M+ jest równoważne M.M*, co najmniej jednokrotne wystąpienie wyrażenia M
- M? co najwyżej jednokrotne wystąpienie wyrażenia M

Podstawowa struktura pliku *.jj

Jednostka kompilacji

```
PARSER_BEGIN(KlasaParsera)
public class KlasaParsera
{
public static void main(String args[]) throws ParseException
{
KlasaParsera parser = new KlasaParsera(System.in);
parser.Input();
}
}
PARSER_END(KlasaParsera)
```

Reguły produkcji

```
void Input() :
{ }
{ ... }
PARSER_BEGIN(KlasaParsera)
public class KlasaParsera
{
//...
}
PARSER_END(KlasaParsera)
```

Specyfikacja ignorowanych ciągów znaków

```
SKIP:
{ ... }
```

Specyfikacja leksemów (tokenów)

```
TOKEN:
{ ... }

void Input() :
{ }
{ ... }
}
```

< nazwa : wyrażenie_regularne >

Zakres instrukcji, które obsługuje Analizator PHP

Podstawowy blok
<?php ?>

Deklaracja zmiennej
<?php $_liczba; ?>
<?php $liczba; ?>
<?php $liczba = 1; ?>
<?php $tekst = „tekst”; ?>

Wypisanie tekstu na ekran monitora
<?php echo $zmienna; ?>
<?php print $zmienna; ?>

Działania na liczbach
<?php $zmienna = 9 + 8 – 1 / 3 * 1; ?>

Funkcja

Deklaracja funkcji
<?php function nazwa($zmienna, $zmienna2, $zmienna = 3){} ?>

Wypisanie funkcji
<?php nawias(); ?>
<?php nawias(5, 8); ?>

Instrukcja warunkowa if else

```
<?php
if($ liczba = $liczba2)
{
if($liczba){}
else{}
 }
else
{
}

if($ liczba <= $liczba2)
{
if($liczba3 >= $liczba4){}
else{}
 }
else
{
}

if($ liczba > $liczba2)
{
if($liczba3 <  $liczba4){}
```

```
else{}
 }
else
{
}

if($ liczba == $liczba2)
{
else
{
}

?>
```

Switch case

```
 <?php
        switch($a)
        {
                case 7:  $a = 9; break;
                default 9: $a;
         }

?>
```

Pętle

Pętla while

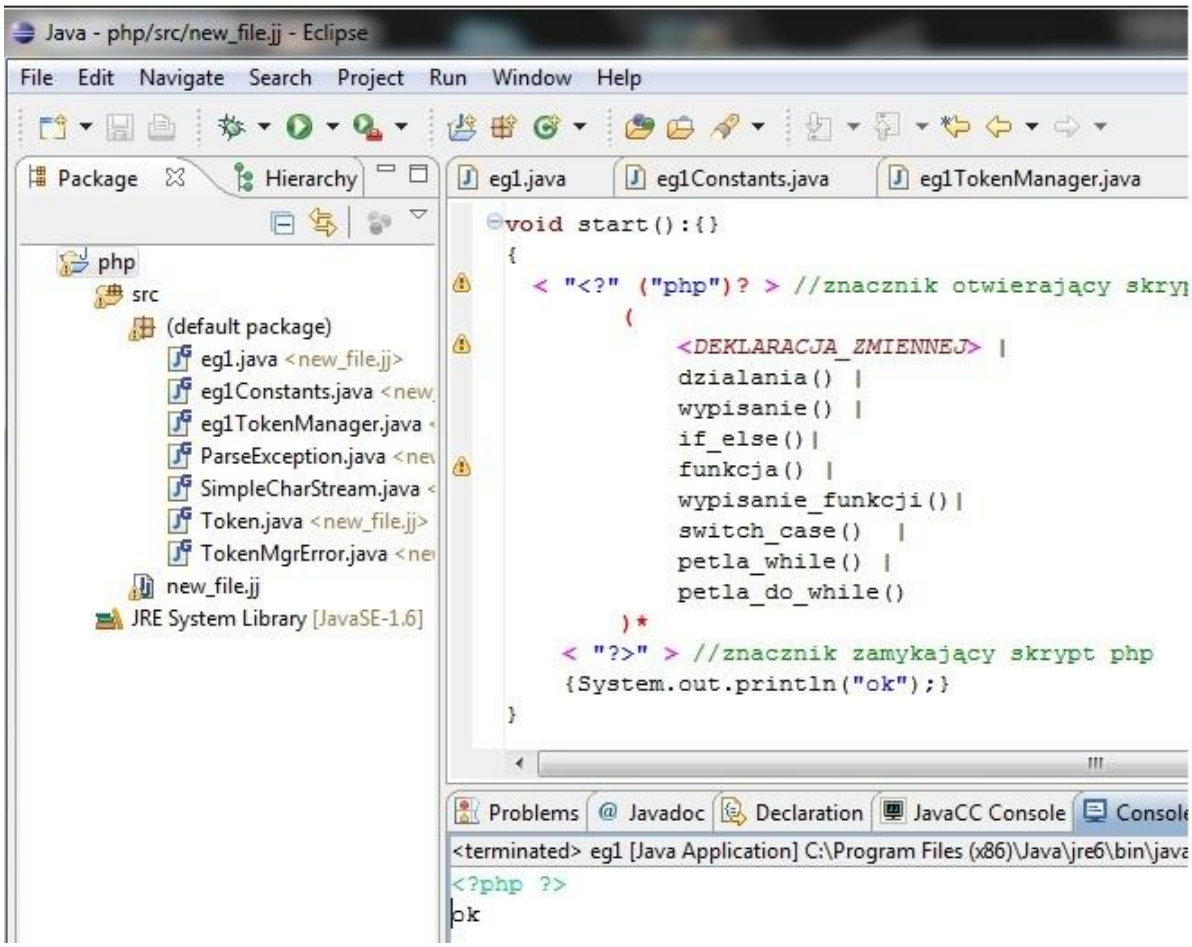
While

```
<?php
while($liczba)
{
        while($liczba == $liczba2){}
}

while($liczba < $liczba2)
{
        while($liczba3 > $liczba4){}
}

while($liczba <= $liczba2)
{
        while($liczba3 >= $liczba4){}
}



?>
```

Pętla do while
<?php do{}while($liczba); ?>

Pętla for
<?php for($a=8; "michal"=="michal"; $a=a+b){} ?>

Przykład działania parsera
Po wpisaniu podstawowego bloku <?php ?> analizator wypisuje słowo ok, patrz poniżej



Dodatek

Kod źródłowy

```
/**
 * JavaCC file
 */

options {
  JDK_VERSION = "1.5";
}
PARSER_BEGIN(eg1)

public class eg1 {
  public static void main(String args[]) throws ParseException {
    eg1 parser = new eg1(System.in);
       parser.start();
}

}
PARSER_END(eg1)

SKIP :
{
       " "
|      "\r"
|      "\t"
|      "\n"
|      "//"
|      "/*"
|      "*/"
}


TOKEN :
{
  <ECHO: "echo">
| <PRINT: "print">

//Zmienne
| <STALA: "define">
| <DOLLAR: "$">

//Nawiasy //Nie wiem czy potrzebne
| <P_KLAMRA: "{">
| <K_KlAMRA: "}">
| <P_NAWIASK: "[">
| <K_NAWIASK: "]">
| <P_NAWIAS: "(">
| <K_NAWIAS: ")">

//Znaki
| <SREDNIK: ";">
//| <KONKATENACJA: "\." >

| <PODKRESLENIE: "_">
| <DWUKROPEK: ":">
| <PRZECINEK: ",">
| <PYTAJNIK: "?">
| <CUDZYSLOW : "\"">
| <APOSTROFY : "'">

//Operatory matematyczne
| <PLUS: "+">
| <MINUS: "-">
| <MNOZENIE: "*">
| <UKOSNIK: "/">
| <MODULO: "%">
```

```
| <ROWNA: "=">
| <ROWNY: "==">
| <NIEROWNY: "!=">
| <MNIEJSZY: "<">
| <WIEKSZY: ">">
| <MNIEJSZY_ROWNY: "<=">
| <WIEKSZY_ROWNY: ">=">

//Operatory łączenia
| <PLUS_ROWNA: "+=">
| <MINUS_ROWNA: "-=">
| <MNOZENIE_ROWNA: "*=">
| <DZIELENIE_ROWNA: "/=">
| <INKREMENTACJA: "++">
| <DEKREMENTACJA: "--">

//Struktury kontrolne
| <IF: "if">
| <ELSE: "else">
| <SWITCH: "switch">
| <CASE: "case">
| <DEFAULT_: "default">
//Pętle
| <DO: "do">
| <FOR: "for">
| <WHILE: "while">
| <FOREACH: "foreach">
| <AS: "as">
| <BREAK: "break">
| <CONTINUE: "continue">

| <RETURN: "return">

//Operatory logiczne
| <OR: "||">
| <AND: "&&">
| <NOT: "!">

//Wyrazenia logiczne
| <TRUE: "true">
| <FALSE: "false">

//Funkcje
| <FUNCTION: "function">

| <LICZBA: ["0"-"9"]>
| <LITERA: ["A"-"Z","a"-"z"]>

//Na początku nazwy może wystąpić znak "_" i dowolne słowo przynajmniej jeden raz,
nie może wystąpić liczba
| <NAZWA: (<LITERA> | <PODKRESLENIE> ) + ((<LITERA> | <LICZBA>)  |  <PODKRESLENIE>)
*>

//Łańcuch, w zmiennej string, pomiędzy cudzysłowami można wstawić dowolną liczbę
znaków <[.] * >
|  <LANCUCH:(<CUDZYSLOW> (<LITERA>)+  <CUDZYSLOW>) | (<APOSTROFY> (<LITERA>)+
<APOSTROFY>)>

|  <ZMIENNA :<DOLLAR> <NAZWA>>

|  <DEKLARACJA_ZMIENNEJ: <ZMIENNA> (<ROWNA>(<LANCUCH> | <LICZBA>))? <SREDNIK> >
}

void start():{}
{
  <"<?" ("php")?> //znacznik otwierający skrypt php, napis "php" nie musi wystąpić
w znaczniku, zależy to od ustawienia serwera
```

```
                (
                        <DEKLARACJA_ZMIENNEJ> |
                        dzialania() |
                        wypisanie() |
                        if_else()|
                        funkcja() |
                        wypisanie_funkcji()|
                        switch_case()  |
                        petla_while() |
                        petla_do_while()|
                        petla_for()
                )*
        <"?>"> //znacznik zamykający skrypt php
        {System.out.println("ok");}
}

void wypisanie() : {}
{
        (<ECHO> | <PRINT>)(((<LANCUCH> | <LICZBA> ) <SREDNIK>) |
(<DEKLARACJA_ZMIENNEJ>))
}

void if_else() : {}
{
        <"if" > <P_NAWIAS> (<LANCUCH> | <LICZBA> | < ZMIENNA >) ((<ROWNY> | <ROWNA> |
<NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> | <WIEKSZY_ROWNY>)(<LANCUCH>
| <LICZBA> | < ZMIENNA >)) * <K_NAWIAS>
        <P_KLAMRA>
                //zawartosc
                (
                        <DEKLARACJA_ZMIENNEJ> |
                        dzialania() |
                        wypisanie() |
                        funkcja() |
                        wypisanie_funkcji()|
                        switch_case()  |
                        petla_while() |
                        petla_do_while() |
                        petla_for()
                )*

                //skrzynka warunkowa
                (
                        <"if"> <P_NAWIAS> (<LANCUCH> | <LICZBA> | <ZMIENNA>) *
((<ROWNY> | <ROWNA> | <NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> |
<WIEKSZY_ROWNY>)(<LANCUCH> | <LICZBA> | < ZMIENNA >)) * <K_NAWIAS>
                        <P_KLAMRA>
                        <K_KlAMRA>
                        <"else">
                        <P_KLAMRA>
                        <K_KlAMRA>
                )*


        <K_KlAMRA>
        < "else" >
        <P_KLAMRA>
                //zawartosc
                (
                        <DEKLARACJA_ZMIENNEJ> |
                        dzialania() |
                        wypisanie() |
                        funkcja() |
                        wypisanie_funkcji()|
                        switch_case()  |
                        petla_while() |
                        petla_do_while() |
                        petla_for()
```

```
            )*

            //skrzynka warunkowa
            (
                  <"if"> <P_NAWIAS> (<LANCUCH> | <LICZBA> | <ZMIENNA>) *
((<ROWNY> | <ROWNA> | <NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> |
<WIEKSZY_ROWNY>)(<LANCUCH> | <LICZBA> | < ZMIENNA >)) * <K_NAWIAS>
                  <P_KLAMRA>
                  <K_KlAMRA>
                  <"else">
                  <P_KLAMRA>
                  <K_KlAMRA>
            )*
      <K_KlAMRA>
}

void switch_case() : {}
{
      <SWITCH> <P_NAWIAS> <ZMIENNA> <K_NAWIAS>
      <P_KLAMRA>
      (
        <CASE> (<LICZBA> | <LANCUCH>) <DWUKROPEK>
            (
                  <DEKLARACJA_ZMIENNEJ> |
                  dzialania() |
                  wypisanie() |
                  funkcja() |
                  wypisanie_funkcji() |
                  petla_while() <SREDNIK> |
                  petla_do_while() <SREDNIK> |
                  petla_for() <SREDNIK>
            )
      <BREAK> <SREDNIK>
      )*

      <DEFAULT_> (<LICZBA> | <LANCUCH>) <DWUKROPEK> (<DEKLARACJA_ZMIENNEJ> |
dzialania()<SREDNIK> | wypisanie() <SREDNIK> | funkcja() <SREDNIK> |
wypisanie_funkcji() <SREDNIK> | petla_while() <SREDNIK> | petla_do_while()
<SREDNIK> )
      <K_KlAMRA>
}

void petla_while() : {}
{

      <WHILE><P_NAWIAS> (<LANCUCH> | <LICZBA> | < ZMIENNA >) ((<ROWNY> | <ROWNA> |
<NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> | <WIEKSZY_ROWNY>)(<LANCUCH>
| <LICZBA> | < ZMIENNA >)) * <K_NAWIAS>
      <P_KLAMRA>
            //zawartosc
            (
                  <DEKLARACJA_ZMIENNEJ> |
                  dzialania() |
                  wypisanie() |
                  funkcja() |
                  wypisanie_funkcji()|
                  switch_case()  |
                  petla_do_while() |
                  petla_for()
            )*
            //petla
            (
                  <WHILE> <P_NAWIAS> (<LANCUCH> | <LICZBA> | <ZMIENNA >) ((<ROWNY>
| <ROWNA >)(<LANCUCH> | <LICZBA> | < ZMIENNA >)) * <K_NAWIAS>
                  <P_KLAMRA> <K_KlAMRA>
            )*
      <K_KlAMRA>
```

```
}

void petla_do_while() : {}
{

	<DO>
	<P_KLAMRA>
		//zawartosc
		(
			<DEKLARACJA_ZMIENNEJ> |
			dzialania() |
			wypisanie() |
			funkcja() |
			wypisanie_funkcji()|
			switch_case()  |
			petla_while() |
			petla_for()
		)*
		//petla
		(
			<DO>
			<P_KLAMRA>
			<K_KlAMRA>
			<WHILE><P_NAWIAS> (<LANCUCH> | <LICZBA> | <ZMIENNA >) ((<ROWNY>
| <ROWNA >)(<LANCUCH> | <LICZBA> | <ZMIENNA >)) *  <K_NAWIAS> <SREDNIK> <K_KlAMRA>
		)*
	<K_KlAMRA>
	<WHILE><P_NAWIAS> (<LANCUCH> | <LICZBA> | < ZMIENNA >) ((<ROWNY> | <ROWNA> |
<NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> | <WIEKSZY_ROWNY>)(<LANCUCH>
| <LICZBA> | < ZMIENNA >)) * <K_NAWIAS><SREDNIK>
}

void petla_for() : {}
{

	<FOR> <P_NAWIAS>
		(<DEKLARACJA_ZMIENNEJ> | dzialania())
		(<LANCUCH> | <LICZBA> | <ZMIENNA>)((<ROWNY> | <ROWNA> | <NIEROWNY> |
<MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> | <WIEKSZY_ROWNY>)(<LANCUCH> | <LICZBA> |
< ZMIENNA>))* <SREDNIK>
		(<ZMIENNA> <ROWNA> (<LICZBA> | <LITERA>) ((<PLUS > | <MINUS> |
<MNOZENIE> | <UKOSNIK> | <MODULO>) (<LICZBA> | <LITERA>)) *)
	<K_NAWIAS>
	<P_KLAMRA>
		//zawartosc
		(
			<DEKLARACJA_ZMIENNEJ> |
			dzialania() |
			wypisanie() |
			funkcja() |
			wypisanie_funkcji()|
			switch_case()  |
			petla_while() |
			petla_do_while()
		)*
		//petla
		(
			<FOR> <P_NAWIAS>
				(<DEKLARACJA_ZMIENNEJ> | dzialania())
				(<LANCUCH> | <LICZBA> | <ZMIENNA>)((<ROWNY> | <ROWNA> |
<NIEROWNY> | <MNIEJSZY> | <WIEKSZY> | <MNIEJSZY_ROWNY> | <WIEKSZY_ROWNY>)(<LANCUCH>
| <LICZBA> | < ZMIENNA>))* <SREDNIK>
				(<ZMIENNA> <ROWNA> (<LICZBA> | <LITERA>) ((<PLUS > |
<MINUS> | <MNOZENIE> | <UKOSNIK> | <MODULO>) (<LICZBA> | <LITERA>)) *)
			<K_NAWIAS>
			<P_KLAMRA><K_KlAMRA>
		)*
	<K_KlAMRA>
```

```
}

void dzialania() : {}
{
	//w PHP można wykonywać działania na literach
	<ZMIENNA> <ROWNA> (<LICZBA> | <LITERA>) ((<PLUS > | <MINUS> | <MNOZENIE> |
<UKOSNIK> | <MODULO>) (<LICZBA> | <LITERA>)) * <SREDNIK>
}

void funkcja() : {}
{
	<FUNCTION> <NAZWA> <P_NAWIAS> ((<ZMIENNA>("," <ZMIENNA>)*)?) |
(<DEKLARACJA_ZMIENNEJ>("," <DEKLARACJA_ZMIENNEJ>)*)?  <K_NAWIAS>
	<P_KLAMRA>
		//zawartosc
		(
			<DEKLARACJA_ZMIENNEJ> |
			dzialania() |
			wypisanie() |
			if_else()|
			wypisanie_funkcji()|
			switch_case()  |
			petla_while() |
			petla_do_while()
		)*
		(<RETURN> (<DOLLAR> <NAZWA>) | <LICZBA> | <LANCUCH> | <SREDNIK>)?
	<K_KlAMRA>
}

void wypisanie_funkcji() :{}
{
	<NAZWA> <P_NAWIAS> (<LICZBA> | <LANCUCH>) * (<PRZECINEK> (<LICZBA> |
<LANCUCH>)) *  <K_NAWIAS> <SREDNIK>
}
```