Návrh gramatiky

KIV/FJP

Milan Janoch & Jakub Pavlíček

Plánované konstrukce – povinné

- definice celočíselných proměnných
- definice celočíselných konstant
- přiřazení
- základní aritmetiku a logiku (+, -, *, /, AND, OR, negace a závorky, operátory pro porovnání čísel)
- cyklus (libovolný – while)
- jednoduchou podmínku (if bez else)
- definice podprogramu (procedura, funkce, metoda) a jeho volání

Plánované konstrukce – rozšiřující – 1 bod

- každý další typ cyklu (for, do while)
- else větev
- datový typ boolean a logické operace s ním
- datový typ real (s celočíselnými instrukcemi)
- datový typ string (s operátory pro spojování řetězců)
- rozvětvená podmínka (switch, case)
- násobné přiřazení (a = b = c = d = 3;)
- paralelní přiřazení ({a, b, c, d} = {1, 2, 3, 4};)

Plánované konstrukce – rozšiřující – 2 body

- operátor pro porovnání řetězců
- návratová hodnota podprogramu
- parametry předávané hodnotou

Ukázkový zdrojový kód – deklarace + přiřazení

```
const int l = 5;                // konstanta

long x = 6;                     // dat. typ long

boolean m = true;               // dat. typ boolean

float p = 2.2;                  // dat. typ real

string s = "skibidi";           // dat. typ string

string k = "sig" + "ma";        // spojování řetězců
```

Ukázkový zdrojový kód – cykly + logické operace

```
for (int x = 5; x < 5; x = x+1;) {
   // příkazy
}

while (x > 5 && x < 10) {
   // příkazy
}

do {
   // příkazy
} while (x < 5 || !a);
```

Ukázkový zdrojový kód – rozvětvená podmínka

```
switch (x) {
    case 5:
        // příkazy
        break;
    case 10:
        // příkazy
        break;
}
```

```
switch (x) {
    case 5:
        // příkazy
        break;
    case 10:
        // příkazy
        break;
    default:
        // příkazy
        break;
}
```

Ukázkový zdrojový kód – násobné + paralelní přiřazení

- Násobné přiřazení

  a, b, c, d = 12;

- Paralelní přiřazení

  {h, i, j} = {3,6,9};

Ukázkový zdrojový kód – porovnání řetězců + návratová hodnota podprogramu

```
boolean foo (string a, string b) {
  return a === b;
}
....

string s1 = "mewing";
string s2 = "jawline";

boolean result = foo(s1,s2);
```

Návrh gramatiky – terminální symboly

- Písmena

  "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" |
  "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

- Čísla

  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Návrh gramatiky – neterminální symboly + počáteční stav

- Neterminální symboly
  - Celkový počet: 49
- Počáteční stav
  - <program>

```
<program> ::= <definitions> <statements>

<definitions> ::= <var-definition>
                | <const-definition>
                | <function-definition>
                | <definitions> <var-definition>
                | <definitions> <const-definition>
                | <definitions> <function-definition>

<const-definition> ::= "const" <data-type> <identifier> "=" <constant-value> ";"

<var-definition> ::= <data-type> <identifier> "=" <expression> ";"

<function-definition> ::= <data-type> <identifier> "(" <parameters> ")" "{" <statements> "return" <expression> ";" "}"

<data-type> ::= "short" | "int" | "long" | "float" | "boolean" | "string"

<statements> ::= <statement> | <statements> <statement>

<statement> ::= <assignment>
                | <parallel-assignment>
                | <if-statement>
                | <while-statement>
                | <do-while-statement>
                | <for-statement>
                | <switch-statement>
                | <function-call> ";"
                | <expression> ";"

<assignment> ::= <identifier-list> "=" <expression> ";"

<parallel-assignment> ::= "{" <identifier-list> "}" "=" "{" <expression-list> "}" ";"

<identifier-list> ::= <identifier> | <identifier-list> "," <identifier>

<expression-list> ::= <expression> | <expression-list> "," <expression>

<if-statement> ::= "if" "(" <logical-expression> ")" "{" <statements> "}" <optional-else>

<optional-else> ::= "else" "{" <statements> "}" | ε

<while-statement> ::= "while" "(" <logical-expression> ")" "{" <statements> "}"

<do-while-statement> ::= "do" "{" <statements> "}" "while" "(" <logical-expression> ")" ";"

<for-statement> ::= "for" "(" <var-definition> <logical-expression> ";" <assignment> ")" "{" <statements> "}"

<switch-statement> ::= "switch" "(" <expression> ")" "{" <case-block> <optional-default> "}"

<case-block> ::= <case-clause> | <case-block> <case-clause>

<case-clause> ::= "case" <constant-value> ":" <statements> "break;"

<optional-default> ::= "default:" <statements> "break;" | ε

<function-call> ::= <identifier> "(" <arguments> ")"

<logical-expression> ::= <logical-term> | <logical-expression> "||" <logical-term>

<logical-term> ::= <logical-factor> | <logical-term> "&&" <logical-factor>

<logical-factor> ::= <comparison-expression> | "!" <logical-factor> | <boolean-value>

<comparison-expression> ::= <expression> <comparison-operator> <expression>
<expression> ::= <arithmetic-expression>
                | <logical-factor>
                | <string-expression>
                | <function-call>

<string-expression> ::= <string-term> | <string-expression> "+" <string-term>

<string-term> ::= <string> | <identifier> | <function-call>


<arithmetic-expression> ::= <term> | <arithmetic-expression> <add-operator> <term>

<term> ::= <factor> | <term> <mul-operator> <factor>

<factor> ::= <integer>
            | <float>
            | <boolean-value>
            | <identifier>
            | "(" <expression> ")"

<parameters> ::= <data-type> <identifier> | <parameters> "," <data-type> <identifier>

<arguments> ::= <expression> | <arguments> "," <expression> | ε

<comparison-operator> ::= "==" | "===" | "!=" | "<" | "<=" | ">" | ">="

<add-operator> ::= "+" | "-"

<mul-operator> ::= "*" | "/"

<identifier> ::= <letter> <identifier-tail>

<identifier-tail> ::= <letter> <identifier-tail> | <digit> <identifier-tail> | ε

<integer> ::= <digit> <integer-tail>

<integer-tail> ::= <digit> <integer-tail> | ε

<float> ::= <integer> "." <integer>

<boolean-value> ::= "true" | "false"

<string> ::= "\"" <string-content> "\""

<string-content> ::= <letter> <string-content> | <digit> <string-content> | ε

<constant-value> ::= <integer> | <float> | <boolean-value> | <string>

<letter> ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
        | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Vybraná pravidla + konstrukce

- Každá case větev musí končit breakem

```
switch (x) {
    case 5:
        .....
        break;
}
```

<case-clause> ::= "case" <constant-value> ":" <statements> "break;"

<optional-default> ::= "default:" <statements> "break;" | ε

- Inkerement for cyklu končí středníkem

```
for (int x = 5; x < 5; x = x+1;)
```

<for-statement> ::= "for" "(" <var-definition> <logical-expression> ";" <assignment> ")" "{" <statements> "}„

<assignment> ::= <identifier-list> "=" <expression> ";"

- Reálné číslo musí vždy obsahovat alespoň jednou desetinnou cifru

```
float realNumber = 2.0;
```

<float> ::= <integer> "." <integer>

<integer> ::= <digit> <integer-tail>

<integer-tail> ::= <digit> <integer-tail> | ε

Vybraná pravidla + konstrukce - pokračování

- Identifikátor musí začínat písmenem

  int a123 = 321;

```
<identifier> ::= <letter> <identifier-tail>
<identifier-tail> ::= <letter> <identifier-tail> | <digit> <identifier-tail> | ε
```

- String musí být uzavřen v dvojitých uvozovkách, nikoliv šablonovými (`Template lit`)

  string name = "John Pork";

```
<string> ::= "\"" <string-content> "\""
<string-content> ::= <letter> <string-content> | <digit> <string-content> | ε
```

- Prioritizace vyhodnocování je dána gramatikou (negace před OR, / před +, apod.)

  boolean o = !a && b || (c);

```
<logical-expression> ::= <logical-term> | <logical-expression> "||" <logical-term>
<logical-term> ::= <logical-factor> | <logical-term> "&&" <logical-factor>
<logical-factor> ::= <comparison-expression> | "!" <logical-factor> | <boolean-value>
```

Děkujeme za pozornost

Dotazy?

GitHub repozitář

- https://github.com/M1LNES/FJP-semester-work