FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/FJP

Semestrální práce

Adam Mištera, Filip Hácha / 16. 12. 2019

Vybrané technologie

- Lexikální analýza
  - FLEX
- Syntaktická analýza
  - Vlastní zásobníkový automat
  - Implementace v jazyce C#
- Intepret
  - Vlastní
  - Implementace v jazyce C#

Rozsah – volitelné konstrukce (beze změn)

- Cykly – for, while, do .. while
- Větvení – if, if else
- Datové typy – int, boolean, char
- Switch
- Násobné přiřazení
- Funkce pro vstup a výstup do konzole
- Pole (int[], boolean[], char[])
- Funkce s parametry předávanými hodnotou
- Alokace a uvolňování paměti

Lexikální analýza

```
 7  function    printf("<%s>", "FUNCTION");
 8  return      printf("<%s>", "RETURN");
 9
10  if          printf("<%s>", "IF");
11  else        printf("<%s>", "ELSE");
12  switch      printf("<%s>", "SWITCH");
13  case        printf("<%s>", "CASE");
14  do          printf("<%s>", "DO");
15  while       printf("<%s>", "WHILE");
16  for         printf("<%s>", "FOR");
17
18  int         printf("<%s>", "INT");
19  char        printf("<%s>", "CHAR");
20  bool        printf("<%s>", "BOOL");
21
22  [+]         printf("<%s>", "OPERATOR_PLUS");
23  [-]         printf("<%s>", "OPERATOR_MINUS");
24  [*]         printf("<%s>", "OPERATOR_MULT");
25  [/]         printf("<%s>", "OPERATOR_DIV");
26  [%]         printf("<%s>", "OPERATOR_MOD");
27  [=][=]      printf("<%s>", "OPERATOR_EQ");
28  [!][=]      printf("<%s>", "OPERATOR_NEQ");
29  [<][=]      printf("<%s>", "OPERATOR_LQ");
30  [>][=]      printf("<%s>", "OPERATOR_GQ");
31  [<]         printf("<%s>", "OPERATOR_LESS");
32  [>]         printf("<%s>", "OPERATOR_GREAT");
33  [:=]        printf("<%s>", "OPERATOR_ASSIGN");
34  [&][&]      printf("<%s>", "OPERATOR_AND");
35  [|][|]      printf("<%s>", "OPERATOR_OR");
36  [!]         printf("<%s>", "OPERATOR_NOT");
```

```
38  [(]         printf("<%s>", "BRACKET_ROUND_LEFT");
39  [)]         printf("<%s>", "BRACKET_ROUND_RIGHT");
40  [\[]        printf("<%s>", "BRACKET_SQUARE_LEFT");
41  [\]]        printf("<%s>", "BRACKET_SQUARE_RIGHT");
42  [{]         printf("<%s>", "BRACKET_CURLY_LEFT");
43  []}]        printf("<%s>", "BRACKET_CURLY_RIGHT");
44
45
46  [0-9]       printf("<%s[%s]>", "BRACKET_DIGIT", yytext);
47  [.]         printf("<%s>", "BRACKET_DOT");
48  [']         printf("<%s>", "QUOT_SINGLE");
49  ["]         printf("<%s>", "QUOT_DOUBLE");
50  [a-zA-Z_]   printf("<%s[%s]>", "IDENT", yytext);
51  [;]         printf("<%s>", "SEMICOLON");
```

Typový systém

- Int, bool, char,
- Pole (ukazatel na první prvek)
  - Na konci každého pole ukončovací znak
- Řetězec implementován jako pole znaků

- Všechny datové typy jsou v podstatě int ☺

Překlad

- Rozkladová tabulka implementována jako slovník pravidel
- Levá strana – token
- Pravá strana – pole tokenů
- Delegát, který bude vykonán při aplikaci pravidla
- Terminální symbol – speciální token

```
public class RewriteRule
{
    public delegate void OnRewrite();


    public Token LeftSide { get; set; }

    public Token[] RightSide { get; set; }

    public OnRewrite RewriteHandler { get; set; }
}
```

Překlad

```
while (stack.Peek() != null)
{
    if (stack.Peek().IsTerminal && stack.Peek().Equals(tokenReader.CurrentToken()))
    {
        stack.Pop();
        tokenReader.Read();
    }

    var rule = DecompositionTable.Get(stack.Peek(), tokenReader.CurrentToken());

    if (rule != null)
    {
        rule.RewriteHandler.Invoke();

        foreach (var token in rule.RightSide.Reverse())
        {
            stack.Push(token);
        }
    }
}
```

Interpret

- Soubor s instrukcemi ve formátu:
  - <Název instrukce> <Argument 1> <Argument 2>
- Správa paměti
  - Realizována obalením funkčností CoreCLR runtime

Git

- Git repozitář
  - https://gitlab.com/hachaf/fjp.git

FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Děkuji za pozornost