**Cvičení 8: Syntaktická analýza ve spojení se základními sémantickými akcemi**

(195.113.207.171:32)

**Cíl:** Pokračovat v procvičování základních možností nástrojů *lex* a *yacc*, tentokráte ve spojení s aplikací jednoduchých sémantických pravidel (akcí).

Minule syntaxe

Dnes sématická pravidla

**To znamená, že přidáme dolary $$ - levá strana**

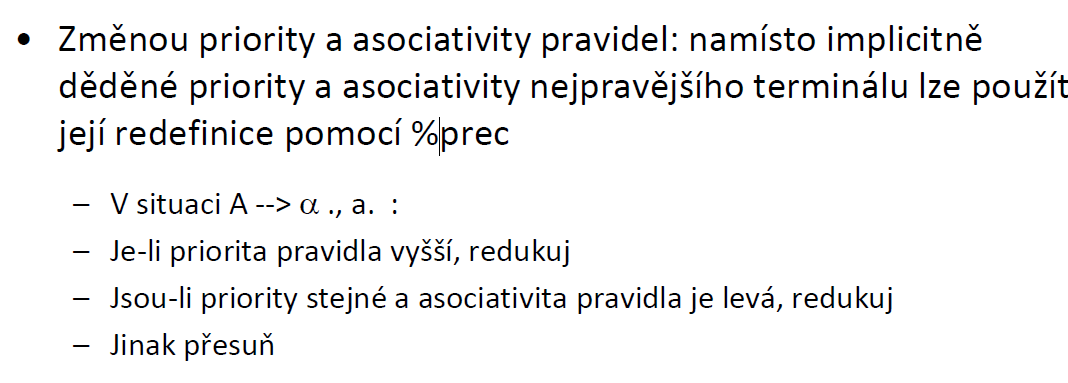
**$1,. $2, pořadí na pravé straně**

**další trik musí se** iniciace hodnot proměnné *yylval* v *lex, jinak by dolary nešly*

*Pak se vypíše výsledek statement: expression {printf("Value: %d\n",$1);}, protože implicitní akce je přiřzení*

*Pozor, jde jen po spuštění programu. Musí se spustit z přík. řádku, ne výstup ze souboru.*

***Ve zdrojáku***



**Nejvyšší priorita**

**unární mínus tím prec se vyšší priorita**

**Priorita nejnižší nahoře**

**%token NUMBER**

**%left '-' '+'**

**%left '\*' '/'**

**%nonassoc UMINUS UPLUS**

!! To pole je již tabulka symbolů

sématický zásobník jsou i ty dolary, to dělí yaccsá osobě

libovolně použít, musí být v tabulce

**Příklad 1: Jednoduchý kalkulátor**

Modifikujte gramatiku z posledního cvičení předchozí hodiny:

statement: --> expression

expression --> expression binary\_operator expression

| unary\_operator expression

| ( expression )

| NUMBER

binary\_operator -> + | - | \* | /

unary\_operator -> - | +

tak, aby výsledný program kromě syntaktické kontroly počítal i skutečné hodnoty zadaných aritmetických výrazů. Dosáhnete toho přidáním elementárních sémantických akcí ke stávajícím syntaktickým pravidlům (iniciace hodnot proměnné *yylval* v *lex*\_u a přístup k nim z *yacc\_u* pomocí proměnných *$<číslo>* ).

Příklad chování programu:

80/8+5\*4

Value: 30

8/80++5+-4

Value: 1

**Příklad 2: Rozšíření možností gramatiky z předchozího příkladu 9**

Musí se definovat union

%union { double dval; int var; }

Pak token je tozho typu

netyopové inkrement, dekrement

Ty které mají hodnotu, se musí deklarovat

Normálně jako integer, ale zde už je to double

správně k tokenům to napsat.

Kulišárna

promitivní tabulka symbolů - vbltable, on uloží

a pracuje s tou hodnotou

udělá se pole asci znaků a-z (a je první)

adresa je dána znakem

Další finta – cyklí se to. Seznam statement odělená ; proto můžeme vložti i soubor statement\_list: statement ';' '\n'

Ošetření dělení nulou , měli by si napsat sami

if ($3 == 0.0)

Vše totéž jako v př. 1 Doalr je zde adresové místzo v tabule, Incr a dct přičítá se číslo k obsauh adrese

Adresa název proměnné

hodnota je obsah

Další drobnost: používají se zde matem fce z knihovny #include <math.h> například pro

Předá

Když tam je číslo, nedělá seni, když tam je číslo

lexx

yylval.dval - převede do floatu a uloží sematic zásobníku

odečte a (0) ,

uloží do nteger a uloží tu adresu.

Tisken jako znak

yylval.dval -em, aby se dostal k těm dolarům zásobník

práce s hodnotami yylval

Vše znají z přednášek.

Rozšiřte gramatiku z příkladu 1 tak, aby:

1. Umožňovala řetězení základních aritmetických příkazů (*statements*) do formy „programu“, přičemž jednotlivé instrukce musí být vždy zakončeny středníkem a začínat na novém řádku. Jakákoli jiná syntaxe je nepřijatelná a vede na chybové hlášení,
2. Kromě čísel mohla pracovat i s jednoznakovými, malými písmeny reprezentovanými textovými proměnnými, na jejichž adresy se ve formátu *double* ukládají hodnoty aritmetických výrazů (*expressions*). Jako datovou strukturu pro tyto primitivní tabulku symbolů použijte statického pole. Obsah libovolné proměnné bude možno vytisknout příkazem

*print <jméno proměnné>* ;

1. Znala funkce odmocnina *(sqrt),* logaritmus *(log),* inkrement *(++)* a dekrement *(--)* hodnoty dané proměnné,
2. Dělení nulou bylo ošetřeno funkcí *yyerror().*

Pro práci s více základními numerickými typy (v tomto případě integer a double) je třeba použít deklaraci *%union*.

Příklad zdrojového kódu:

i=6\*7;

i--;

a=8;

i=i+a;

print i;

print (sqrt(i)+8);

<Ctrl-D>

jeho spuštění:

> ./cv2 <data.txt

a výstupu:

49.000000

15.000000

OK