Implementace aritm. výrazu

jde o zásobníkový automat => bude využívat zásobník na rekurzivní volání podprogramů

uložení do zásob. = expanze = zavolání podprogramu

jde o metodu REKURZIVNÍ SESTUP

expanze - provádí se pro neterminální symboly

PŘ

int lexan(); - nutno definovat - viz slajdy
(lexikální analyzátor)

int lexan(); // vrací T_Lzávorka, T_číslo, ... T_EOF
char *jmeno; // jméno identifikátoru
double hodnota; // hodnota čísla

vyplní lexan

int _symb = NOSYMB;
# define = definice maker

define Symb \
((_symb == NOSYMB) ? \
_symb = lexan() : _symb)

```
#define Srovnani(x) \
    ((Symb == x) ? (_symb = NOSYMB, 1) : \
                                  pokud OK, vrátí 1
    (Chyba_srovnani(x, _symb), _symb = NOSYMB, 0))
```

rozklad

→ E → TE'

E → -T E'

E' → +T E'

E' → -T E'

E' → ε

T → FT'

T' → * F T'

T' → / F T'

T' → ε

F → c

F → (E)

implementace dle pravidel

pro terminální symbol zavolám

pro neterminální expanze

```
void E(){
  switch (Symb){
    case : T_cislo;
    case : T_l_zavorka:
            T();
            E_c();
            break;
    case T_minus:
            Vyzvani (T_minus);
            T();
            E_c();
            break;
    default : chyba ();
}
```

(PŘ) → E'

```
void E_e() {
    switch (Symb) {
        case T_plus:
            Srovnani (T_plus);
            T();
            E_e();
            break;
        case T_minus:
            Srovnani (T_minus);
            T();
            E_e();
            break
        case T_~~#~~pravy:      // ")"
        case T_EOF;             // ε
            break;
        default chyba();
    }
}
```

Lze zjednodušit - ~~lze~~ napsat
case pro T_plus a default pro T_minus

POZOR default musí být vždy !!!

```
void F() {
    switch Symb {
        case T_cislo:
            Generuj(T_cislo);
            E();
            break;
        default Generuj(T_var);
            E();
            Generuj(T_prav);
    }
}
```

Překlad:

~~[illegible]~~

používají se pravidla s (X)

terminální symbol = srovnání

neterminální

expanze

```
void E_c() {                                  pro E'
    switch (Symb) {
        case T_plus:
            Srovnani(T_plus);
            T();
            Vystup("+");
            E_c();
            break;
        case T_minus:
            Srovnani(T_minus);
            T();
            Vystup("-");
            E_c()
            break;
        default:          // E' -> ε
    }
}
```

pro F

```
void F(){
    if (symb == T_cislo){
        srovnani (T_cislo);
        vystup (hodnota);
    }
    else {
        srovnani (T_lzav);
        E();
        srovnani (T_pzav);
    }
}
```

Využití EBNF

- '-' je nepovinné; $\{(\text{'+'} \mid \text{'-'})\,T\}$ = E'

pravidla

$E ::= [\text{"-"}]\ T\ \{(\text{'+'} \mid \text{'-'})\ T\}$

$T ::= F\ \{(\text{'*'} \mid \text{'/'})\ F\}$

$F ::= \text{'('}\ E\ \text{')'} \mid c$

EBNF lze použít v rekurzivním sestupu ke snížení počtu rekurzí

```
void E(){
    if (Symb == T_minus) Scorrani(T_minus);
        T();
        while (Symb == T_plus || Symb == T_minus){
            if (Symb == T_plus) Scorrani(T_plus);
            else Scorrani(T_minus);
            T();
        }
}
```