

**TECHNICKÁ UNIVERZITA V KOŠICIACH**  
**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

**Zadanie (prekladač)**  
**Dokumentácia zadania**

## Zadanie práce

Zadanie:

- Jazyk má podporu pre zátvorkové výrazy aritmetických operácií a operácií porovnávania (to už máte).
- Na začiatku programu sa nachádza deklarácia premenných, používaných v programe.
- Jazyk pozostáva zo sekvencie príkazov. Môžete teda do premenných priradzovať hodnoty alebo výsledky výrazov.
- Jazyk umožňuje vypísať vypočítaný výraz na obrazovku a načítať premenné od používateľa.
- Jazyk obsahuje príkazy pre cyklus a vetvenie.

Štruktúra:

- Makefile (aby sa dal projekt skompilovať príkazom make).
- Súbory so zdrojovým kódom (\*.c, \*.h).
- Príklady vstupných súborov vo vytvorenom jazyku (\*.input).
- Dokument vo formáte PDF popisujúci dané zadanie - nahrádza verbálnu obhajobu zadania. Mal by obsahovať napr. tieto informácie:
  - gramatika jazyka,
  - ukážka vstupného súboru v tomto jazyku, binárneho výstupného súboru (disassemblovaného), výsledok vykonania v Computrone,
  - aké vývojové prostredie ste použili,
  - ako ste postupovali - či ste vytvorili novú kostru projektu alebo ste upravovali projekt z cvičení, v akom poradí ste implementovali časti prekladača a pod.,
  - s čím boli problémy, ako ste ich vyriešili,
  - ktoré doplňujúce úlohy ste urobili a ktoré nie,
  - čo sa Vám na zadaní páčilo alebo nepáčilo,
  - čo sa Vám nepodarilo urobiť, čo je urobené navyše a chceli by na to upozorniť, atď.

# Obsah

Gramatika Jazyka .....	4
Zadanie .....	5
1.1. Ukážka vstupného súboru v tomto jazyku .....	5
1.2. Binárny výstup súboru (disassembler) .....	6
1.3. Výsledok vykonania v Computrone .....	8
1.4. Spustenie a beh programu .....	9
1.5. Návrh a implementácia riešenia .....	10
1.5.1. Vývojové prostredie .....	10
1.5.2. Postup riešenia .....	10
1.5.3. Problémy .....	10
1.5.4. Doplnujúce úlohy .....	10
1.5.5. Implementácie navyše .....	11
Záver .....	12

## Gramatika Jazyka

Op -> Expr [ (= | < | > | != | <= | >= ) Expr]

Expr -> Mul { ("+" | "-") Mul }

Mul -> Power { ("\*" | "/" ) Power }

Power -> POW\_ASOCIATIVE == LEFT Power -> Integer [ "^" Power ]

-> POW\_ASOCIATIVE == RIGHT Power -> Integer { "^" Integer }

-> POW\_ASOCIATIVE == NO Power -> Integer [ "^" Integer ]

Integer -> [-] Number

Number -> VALUE | "(" Term

Term -> VALUE | "(" Expr ")"

Stm -> "Sequence" | "Read" | "If" | "While" | "Print" | "Id"

BoolExpr -> (Op [ ( "&&" | "||" ) Op ] ) | ( BoolExpr [ ( "!" | "(" ) Op ] )

BoolTerm -> ("true" | "false") | ("!") BoolExpr | Expr

Sequence -> ";" Stm

## Zadanie

### 1.1. Ukážka vstupného súboru v tomto jazyku

#### Program č. 1 :

Program pre nájdenie maxima hodnôt uložených v troch premenných (x, y, z):

```
1  var x, y, z, max;
2  read x,y,z;
3
4  if(y<= x && z <= x) then {
5      max := x;
6  }else{
7      if(x<=y && z<=y)then{
8          max:=y;
9      }else{
10         max:=z;
11     }
12 };
13
14 print max;
```

1.obr. program č.1

#### Program č. 2 :

Program pre výpočet najväčšieho spoločného deliteľa pomocou Euklidovho algoritmu nerekurzívne. Vstupné hodnoty sú v premenných x, y, na konci výpočtu sa najväčší spoločný deliteľ nachádza v oboch premenných.

```
1  var x,y;
2  read x,y;
3
4  while !(x = y) do {
5      if(y<=x)then{
6          x:=x-y;
7      }else{
8          y:=y-x;
9      };
10 };
11
12 print x;
```

2.obr. program č.2

## 1.2. Binárny výstup súboru (disassembler)

### Program č. 1 :

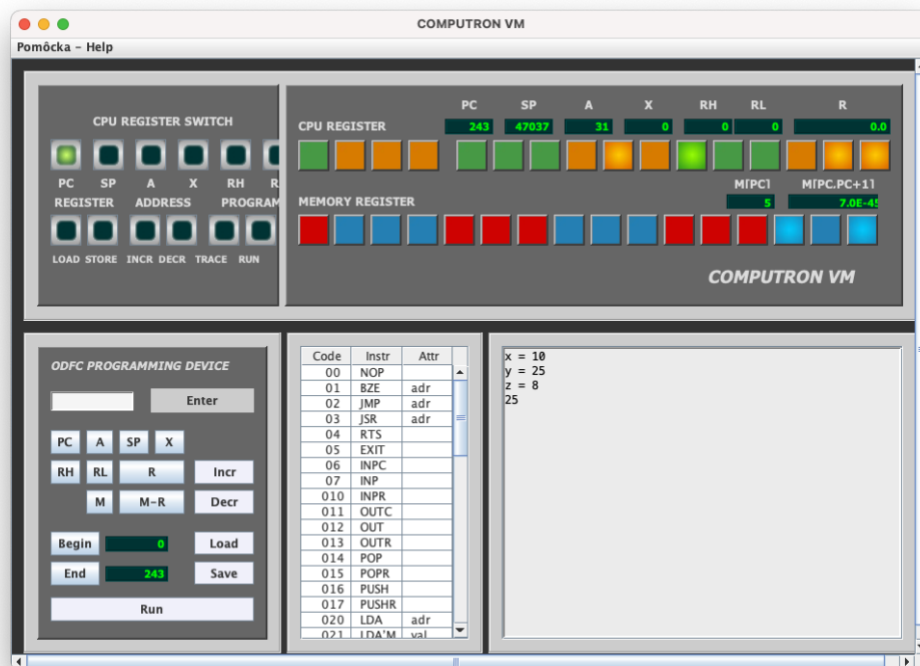
[00] JMP 07	[065] LDA 04	[0155] STA 02
// nerozpoznaná instrukcia	[067] PUSH	[0157] POP
[02] NOP	[070] LDA 03	[0160] LE 02
[03] NOP	[072] PUSH	[0162] PUSH
[04] NOP	[073] POP	[0163] LDA 05
[05] NOP	[074] STA 02	[0165] PUSH
[06] LDS 02	[076] POP	[0166] LDA 04
[010] LDAM 0170	[077] LE 02	[0170] PUSH
[012] OUTC	[0101] PUSH	[0171] POP
[013] LDAM 040	[0102] LDA 05	[0172] STA 02
[015] OUTC	[0104] PUSH	[0174] POP
[016] LDAM 075	[0105] LDA 03	[0175] LE 02
[020] OUTC	[0107] PUSH	[0177] PUSH
[021] LDAM 040	[0110] POP	[0200] POP
[023] OUTC	[0111] STA 02	[0201] STA 02
[024] INP	[0113] POP	[0203] POP
[025] STA 03	[0114] LE 02	[0204] AND 02
[027] LDAM 0171	[0116] PUSH	[0206] PUSH
[031] OUTC	[0117] POP	[0207] POP
[032] LDAM 040	[0120] STA 02	[0210] STA 02
[034] OUTC	[0122] POP	[0212] BZE 0230
[035] LDAM 075	[0123] AND 02	[0214] LDA 04
[037] OUTC	[0125] PUSH	[0216] PUSH
[040] LDAM 040	[0126] POP	[0217] POP
[042] OUTC	[0127] STA 02	[0220] STA 06
[043] INP	[0131] BZE 0147	[0222] POP
[044] STA 04	[0133] LDA 03	[0223] STA 02
[046] LDAM 0172	[0135] PUSH	[0225] JMP 0236
[050] OUTC	[0136] POP	[0227] LDA 05
[051] LDAM 040	[0137] STA 06	[0231] PUSH
[053] OUTC	[0141] POP	[0232] POP
[054] LDAM 075	[0142] STA 02	[0233] STA 06
[056] OUTC	[0144] JMP 0236	[0235] LDA 06
[057] LDAM 040	[0146] LDA 03	[0237] PUSH
[061] OUTC	[0150] PUSH	[0240] POP
[062] INP	[0151] LDA 04	[0241] OUT
[063] STA 05	[0153] PUSH	[0242] EXIT
	[0154] POP	

**Program č. 2 :**

[00] JMP 05	[075] STA 02
// nerozpoznana instrukcia	[077] POP
[02] NOP	[0100] LE 02
[03] NOP	[0102] PUSH
[04] LDS 02	[0103] POP
[06] LDAM 0170	[0104] STA 02
[010] OUTC	[0106] BZE 0136
[011] LDAM 040	[0110] LDA 03
[013] OUTC	[0112] PUSH
[014] LDAM 075	[0113] LDA 04
[016] OUTC	[0115] PUSH
[017] LDAM 040	[0116] POP
[021] OUTC	[0117] STA 02
[022] INP	[0121] POP
[023] STA 03	[0122] SUB 02
[025] LDAM 0171	[0124] PUSH
[027] OUTC	[0125] POP
[030] LDAM 040	[0126] STA 03
[032] OUTC	[0130] POP
[033] LDAM 075	[0131] STA 02
[035] OUTC	[0133] JMP 0156
[036] LDAM 040	[0135] LDA 04
[040] OUTC	[0137] PUSH
[041] INP	[0140] LDA 03
[042] STA 04	[0142] PUSH
[044] LDA 03	[0143] POP
[046] PUSH	[0144] STA 02
[047] LDA 04	[0146] POP
[051] PUSH	[0147] SUB 02
[052] POP	[0151] PUSH
[053] STA 02	[0152] POP
[055] POP	[0153] STA 04
[056] NE 02	[0155] POP
[060] PUSH	[0156] STA 02
[061] POP	[0160] JMP 045
[062] STA 02	[0162] LDA 03
[064] BZE 0163	[0164] PUSH
[066] LDA 04	[0165] POP
[070] PUSH	[0166] OUT
[071] LDA 03	[0167] EXIT
[073] PUSH	
[074] POP	

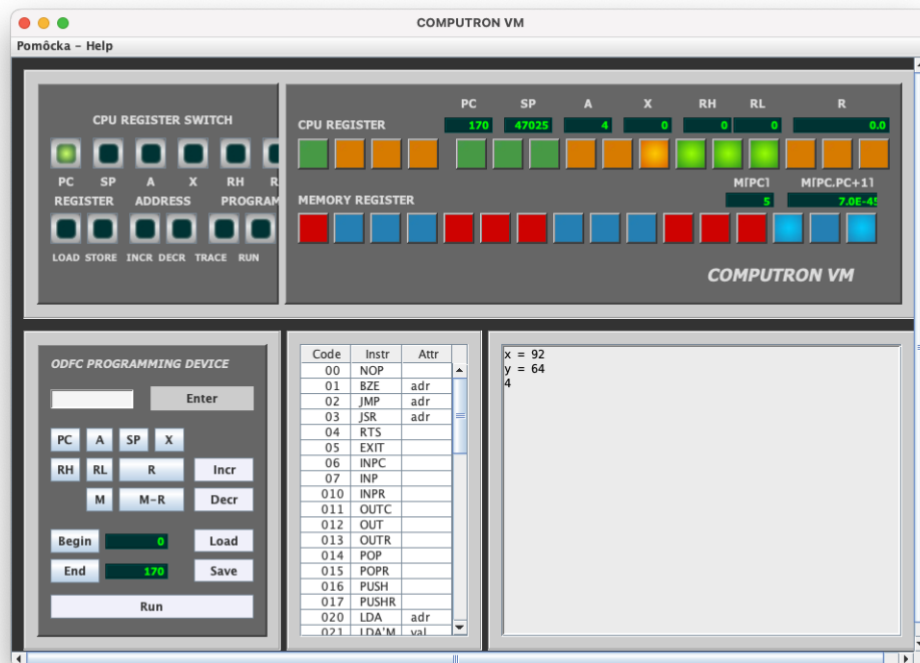
### 1.3. Výsledok vykonania v Computrone

#### Program č. 1:



3.obr. program č. 1 Computron VM

#### Program č. 2:



4.obr. program č. 2 Computron VM



## 1.4. Spustenie a beh programu

Zadanie je kompilovateľne v konzole pomocou nástroja gcc alebo pomocou príkazu **make** .

Po použití príkazu **make** sa vygenerujú súbory s koncovkou .o, a hlavný program **prekladac**, ktorý je spustiteľný cez pomocou príkazu **./prekladac**

Po spustení programu máme možnosť zadať program na preloženie ručne, alebo vybrať možnosť na spustenie programu s koncovkou .txt.

Taktiež máme možnosť zadať názov súboru už pri spúšťaní prekladača v príkazovom riadku:

**./prekladac ./programy/program1.txt** . Po takomto spustení sa nám priamo vygeneruje výstup pre program1.txt.

Výstup z prekladača má názov : **program.bin** , tento súbor je spustiteľný vo virtuálnom stroji Computron.

Po spustení prekladača je v programe implementovaná funkcionálna tzv. Simulatora, ktorá simuluje beh programu na Computrone (žiada vstupy, a vypíše výstupy) priamo v konzolovom prostredí.

## 1.5. Návrh a implementácia riešenia

### 1.5.1. Vývojové prostredie

Na vývoj prekladača som použil vývojové prostredie [Visual Studio Code](#) na operačnom systéme macOS.

Na kontrolu správnosti riešenia som používal aplikáciu [Computron](#).

### 1.5.2. Postup riešenia

Na začiatku som si stiahol **kostru projektu**

- [Zdrojový kód lexikálneho analyzátora](#)
- [Zdrojový kód interpretátora gramatiky jazyka](#)
- [Zdrojový kód generátora príkazov pre Computron](#)

Postupne som pokračoval upravovaním tohto kódu a pridávaním nových vlastných knižníc a súborov. Vytvoril som si Makefile na spojenie všetkých súborov a jednotne spustenie.

Na kontrolu zadania som používal príklady ktoré som si zapísal do adresára programy. Inspiráciu som bral z dokumentu ktorý nám bol poskytnutý na cvičení.

Pri implementovaní nových funkcií som sa držal plánu ktorý bol zverejnený na [stránke predmetu](#).

### 1.5.3. Problémy

Najväčší problém s ktorým som sa stretol bolo porozumenie príkazov pre Computron VM a následne implementovanie funkcií v súbore *generator.c*. Na riešenie tohto problému som hlavne používal [príručku pre computron](#), v ktorej boli podrobne vysvetlene príkazy.

### 1.5.4. Doplnujúce úlohy

V mojom riešení som implementoval / vyriešil tieto doplnujúce úlohy:

1. Cvičenie 2 : Úloha A.1, Úloha A.2
2. Cvičenie 5 : Úloha A.1, Úloha A.2
3. Cvičenie 7 : Úloha A.1

### 1.5.5. Implementácie navyše

V mojom zadaní som navyše implementoval možnosť negovania výrazu pred zátvorov:

Pr. **while ! ( a <= b ) do ....**

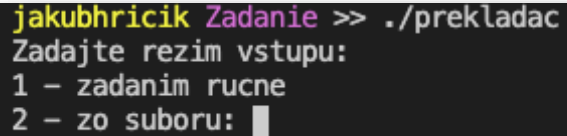
kde "!" neguje výraz v zátvorke a, „a“ a „b“ sú premenne

môžeme použiť aj takýto výraz :

**while ! ( ! ( a != b ) ) do ...**

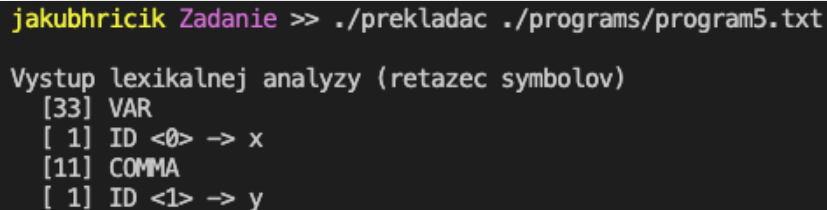
Taktiež som pridal možnosť zadania vstupu priamo v konzole alebo voľby pre výber z príkladov uložených v súbore.

**Možnosti spustenia:**



```
jakubhricik Zadanie >> ./prekladac
Zadajte rezim vstupu:
1 - zadanim rucne
2 - zo suboru: █
```

5.obr. možnosti spustenia programu 1



```
jakubhricik Zadanie >> ./prekladac ./programs/program5.txt
Vystup lexikalnej analyzy (retazec symbolov)
[33] VAR
[ 1] ID <0> -> x
[11] COMMA
[ 1] ID <1> -> y
```

6.obr. možnosti spustenia programu 2

## Záver

Zadanie sa mi páčilo pravé preto lebo bolo trochu náročnejšie ako zadania na iných predmetoch a ja mam rad výzvy. Zo začiatku to bolo možno ťažšie na pochopenie, ale po čase som sa do toho dostal a vedel som čo ako funguje a ako to spraviť. Možno celkovo by som prijal trochu názornejšej ukážky na cvičeniach, ale inak to bolo veľmi dobre zadanie.