Úvod

Když jsem po loňském Invexu začal přemýšlet o další soutěži, řekl jsem si, že by bylo zajímavé udělat prostředí pro nějaký programovací jazyk. Tatínek mi poradil, že nejjednodušší jazyk pro naprogramování je FORTH. I když o něm řada lidí neví, vyskytuje se kolem nás na každém kroku. Ve FORTHu např. pracují všechny postscriptové tiskárny, protože tiskárenský jazyk PostScript je jenom zvláštní druh Forthu. Na stejném principu jako FORTH pracuje i jádro virtuálního stroje Javy.

Celé jaro jsem se FORTH učil z jedné knížky, kterou kdysi dávno vydal Software Slušovice, ze seriálu v Amatérském rádiu a z nějakých anglických textů, které jsem si stáhl z internetu. Na tomto jazyku se mi líbilo, že se dá neustále rozvíjet a že se na jeho programování hodně věcí naučím.

Jazyk FORTH má vlastní normu, jejíž poslední verze je z roku 1994 a kterou si můžete stáhnout z adresy ftp://ftp.uu.net/vendor/minerva/x3j14/dpans94.zip.

Ve svém interpretu jsem se snažil držet této normy, i když jsem definoval pouze malou část slov, o kterých se v ní píše.

Můj program se jmenuje M4th, protože se název FORTH občas zkracuje na 4th, protože se vyslovuje stejně, jako anglická číslovka „fourth“, a ta se právě zapisuje 4th. Jeho název vznikl tak, že jeho autor Charles Moore jej považoval za jazyk čtvrté generace. Chtěl jej proto nazvat FOURTH, ale jeho počítač dovoloval pouze 5písmenné názvy, tak musel jedno písmeno z názvu vyhodit.

Jak program spustíte a ukončíte

Program nepotřebuje žádné parametry. Stačí spustit soubor M4th.exe nebo přeložit a spustit soubor M4th.SGP. Pak už můžete začít pracovat podle následující učebnice.

Program ukončíte slovem **BYE**. Pozor, musí se napsat velkými písmeny. Nelekněte se, že po jeho zadání prostředí Baltazara ohlásí chybu dělení nulou. Protože Baltazarovo C neumí dlouhé skoky (skoky z jedné funkce do jiné , která ji volala), nepodařilo se mi najít jiný způsob, jak program ukončit. Spustíte-li soubor rovnou z příkazového řádku, chybu nehlásí.

Začínáme

První, co začátečníka na jazyce FORTH překvapí, je jeho naprostá odlišnost od jazyků jiných. Práce v jazyku je postavena na dvou zásobnících. První je systémový a jsou na něj ukládány návratové adresy a některé další údaje. Uživateli se nedoporučuje tento zásobník používat. Jelikož je tento zásobník označován „zásobník návratových adres“, budeme ho označovat ZNA.

Pro uživatele je určen zásobník druhý, na kterém se provádí veškeré operace. Budeme ho označovat UZ (uživatelský).

Program jazyku FORTH je tvořen posloupností „slov“, oddělených mezerami nebo znaky „Nová řádka“. Slovo může být označeno libovolnou posloupností znaků bez mezer a řídících znaků. Vlastností jazyka FORTH je, že můžeme vytvářet vlastní slova poskládaná z jiných slov. Každému slovu je přiřazena nějaká činnost a slovo tedy představuje obdobu procedury. Tato procedura očekává všechny parametry na zásobníku, odkud si je vyzvedne a ze zásobníku je smaže. Případně uloží na zásobník výsledky.

Definice slov jsou uloženy v tzv. slovníku, který bývá někdy označován třetí zásobník. Slouží k uchování definic nových slov, proměnných, polí, atd. Slova do něj můžeme přidávat nebo je z něj mazat podobně, jako u zásobníku.

Interpret čte program slovo za slovem, každé slovo nejdříve najde ve slovníku a poté vykoná patřičnou činnost. Pokud slovo ve slovníku nenajde, pokusí se ho interpretovat jako číslo, jehož hodnotu uloží na TOS. Pokud se ho nepodaří interpretovat, ohlásí chybu.

Základní interpret jazyka FORTH obsahuje kolem 200 slov. Tato skutečnost vyžaduje oproti jiným jazykům odlišný styl výuky. Každá kapitola se bude zabývat některou z vlastností jazyka FORTH. Na jejím začátku budou uvedeny definice nových slov, která se budou dále používat.

V popisech o stavu zásobníku budeme používat následující označení:

a	adresa
(a)	obsah buňky na adrese a
(a)B	obsah bajtů na adrese A

Zásobník

Než se začneme bavit o programování, vysvětlíme si zásobník, s ním spojené operace a věci kolem něj. Zásobník bude v celé učebnici důležitý a bez něj by FORTH nebyl FORTH . Zásobník je datová struktura, kterou si můžete představit jako šuplík. Do šuplíku dáváme papíry na sebe a vždy máme přístup jen k tomu hornímu, neboli k tomu, který jsme do šuplíku vložili jako poslední. Abychom se dostali k dříve položenému papíru, musíme odendat všechny papíry, které jsme uložili později a jsou nad ním. Takto pracuje zásobník.

Položky na zásobníku budeme označovat:

TOS – vršek zásobníku (Top of stack) – položka kterou jsme na zásobní dali naposledy, nebo kterou se tam chystáme dát

NOS – položka pod TOS (Next on stack)

NNOS – položka pod NOS

V dalším textu budu zásobník zobrazovat, takže TOS bude vždy dole.

1 TOS	**1 NOS** **2 TOS**	**1 NNOS** **2 NOS** **3 TOS**	**1** **2 NNOS** **3 NOS** **4 TOS**	**1** **2** **3 NNOS** **4 NNOS** **5 TOS**

Aritmetické operace

Nová slova:

+ (n1 n2 → (n1+n2))
Sečte (TOS) a (NOS), výsledek uloží na TOS.

– (n1 n2 → (n1-n2))
Odečte (TOS) od (NOS), výsledek uloží na TOS.

***** (n1 n2 → (n1*n2))
Vynásobí (TOS) a (NOS), výsledek uloží na TOS.

/ (n1 n2 → (n1/n2))
Vydělí (NOS) (TOS), výsledek uloží na TOS.

DUP (n → n n)
Zduplikuje vrchní položku uz.

. (n →)
Vytiskne (TOS) na obrazovku a smaže ho.

Ve FORTHU se počítá podle postfixové notace, nazývané také obrácená polská notace (RPN – Reverse Polish Notation). V tomto způsobu zápisu se nejdříve uvádějí operandy a až pak operátory. Chceme-li spočítat např. „3*7“, napíšeme to takto:

3	**7**	*****
3	3	21
	7	

Nebo složitější příklad „3 * (6 + 9) + (5 + 3) * 6“ vypočítáme následovně:

3	**6**	**9**	**+**	*****	**5**	**3**	**+**	**6**	*****	**+**
3	3	3	3	45	45	45	45	45	45	93
	6	6	15		5	5	8	8	48	
		9				3		6		

Nyní zkuste sami vypočítat tyto příklady:

1. 2 * ((6 / 2) * (3 + 8))
2. (4 + (2 * 7)) / 2 * 3
3. 9 * (49 / 7) + (2 + 8)

Kontrolní řešení:

1. 2 6 2 / 3 8 + *
2. 4 2 7 * + 2 / 3 *
3. 9 49 7 / * 2 8 + +

Definice nových slov

Nová slova:

: xxx (→)
Začátek dvojtečkové definice slova xxx. Nastavení režimu COMPILE.

; (→)
Konec dvojtečkové definice. Nastavení režimu EXECUTE.

SWAP (x1 x2 → x2 x1)
Prohodí (TOS) s (NOS).

NOT (f → -f)
Neguje logickou hodnotu TOS.

AND (f1 f2 → (f1&f2)b)
Logický součin (TOS) a (NOS) bit po bitu.

OR (f1 f2 → (f1|f2)b)
Logický součet (TOS) a (NOS) bit po bitu.

XOR (f1 f2 → (f1|f2)b)
Exkluzivní OR (TOS) a (NOS) bit po bitu.

0< (n → f(n<0))
Uloží na TOS logickou hodnotu výrazu (TOS) < 0.

Slova v lekci nadefinovaná:

2NA (n → (n*n))
Umocní (TOS) na druhou. Na TOS uloží výsledek.

1+ (n → (n+1))
Přičte k (TOS) jedničku.

1- (n → (n-1))
Odečte od (TOS) jedničku.

Možnost na definování vlastních slov je výhodou jazyka FORTH.

Slova můžeme nadefinovat pomocí dvojtečkové definice. Např. slovo 2NA, které umocní TOS uživatelského zásobníku na druhou, bychom mohli nadefinovat takto:

: 2NA DUP * ;

Definice začíná dvojtečkou, která přepne režim EXECUTE, ve kterém se slova provádí, na režim COMPILE, ve kterém se slovo bude zapisovat do slovníku.

Po dvojtečce následuje mezerou oddělené jméno slova, které může být dlouhé maximálně 31 znaků a nesmí obsahovat mezery ani řídící znaky.

Nakonec se mezerou odděleně napíše posloupnost slov, která bude nové slovo provádět.
V našem případě slovo zduplikuje TOS a vynásobí dvě stejné položky TOS a NOS, na zásobníku bude výsledek operace.

Celou definici ukončíme opět mezerou odděleným středníkem, který nastaví opět režim EXECUTE.

Nyní zkuste sami nadefinovat slovo **1+**, které k TOSu přičte 1, nebo **1-**, které od TOSu odečte 1.

Kontrolní řešení:

1+ **: 1+ 1 + ;** Dá na zásobník jedničku a pak ji přičte k NOSu

1- **: 1- 1 - ;** Dá na zásobník jedničku a pak ji odečte od NOSu

Vždy, když teď napíšete 1+, přičte k vršku uz jedničku.

Číselné soustavy

Nová slova:

BASE (□ .(BASE).)
Proměnná, ve které je uložen základ soustavy, v níž jsou čtena vstupující(vystupující) čísla. Slovo BASE uloží na TOS adresu, na níž je uložena hodnota této proměnné.

DECIMAL (→)
Nastaví čtení a vyjadřování čísel v desítkové soustavě.

@ (a → (a))
Uloží na TOS obsah na adrese a.

! (n1 a →)
Uloží na adresu a položku z NOS.

( ) Komentářové závorky. Text mezi nimi se neprovádí. M4th závorky nepodporuje.

." (→)
Na obrazovku vytiskne následující text až po znak "(uvozovky). Mezi slovem ." textem musí být mezera. **Omlouvám se, ale toto slovo mi těsně před odevzdáním přestalo chodit.!**

CR (→)
Na obrazovce provede přechod na začátek dalšího řádku.

Slova v lekci nadefinovaná:

BIN (→)
Nastaví čtení a vyjadřování čísel v dvojkové soustavě.

HEX (→)
Nastaví čtení a vyjadřování čísel v šestnáctkové soustavě.

OCT (→)
Nastaví čtení a vyjadřování čísel v osmičkové soustavě.

TISK (n →)

Vytiskne TOS v BIN , OCT, DEC, HEX soustavě.

Na chvíli se vrátíme k číslům. V jazyku FORTH si může zvolit číselnou soustavu, ve které budou tištěna a čtena čísla. Základ číselné soustavy je uložen v proměnné BASE. Verze M4th zná slovo DECIMAL, které zavádí zápis a čtení v desítkové soustavě.

Můžeme si nadefinovat kratší verzi: `: DEC DECIMAL ;`

Teď si ukážeme postup nadefinování jiných základů. Abychom mohli nastavit čtení v dvojkové soustavě nadefinujeme si slovo:

:	BIN	2	BASE	!	;
		2	2 .(BASE).		

Jistě jste si všimli, že pokud vykonáte slovo, jež je názvem proměnné, uloží se na TOS adresa, na které je obsah proměnné. Slovo vykřičník uložilo na adresu z TOS obsah NOS. V našem případě tedy uložilo do proměnné BASE dvojku. Dvojková soustava se nastaví při použití slova BIN, nikoli při jeho definici. Změna proměnné BASE má vliv jen na vstupy a výstupy. Ve vnitř se pořád pracuje se soustavou binární. Dále si můžeme ukázat definici slova OCT a HEX:

:	OCT	8	BASE	!	;
		8	8 .(BASE).		

Takto vypadá definice slova OCT, pomocí kterého můžete nastavit výstupy a vstupy v osmičkové soustavě.

:	HEX	16	BASE	!	;
		16	16 .(BASE).		

Definice slova HEX, pomocí kterého můžete nastavit výstupy a vstupy v šestnáctkové soustavě.

Nyní si zkuste nadefinovat slovo TISK, které vytiskne TOS ve všech zatím nadefinovaných soustavách.

Program by mohl vypadat takto:

```
: TISK  ( název slova )
  CR BIN DUP .
( Odřádkuje, udělá kopii čísla, protože bude ještě potřeba a ho v dvojkové
soustavě )
   CR OCT DUP .
( Odřádkuje, udělá kopii čísla, protože bude ještě potřeba a ho v osmičkové
soustavě )
   CR DEC DUP .
( Odřádkuje, udělá kopii čísla, protože bude ještě potřeba a ho v desítkové
soustavě )
   CR HEX .        ;
( Odřádkuje, vytiskne číslo v osmičkové soustavě, číslo již nebude potřeba a
skončí )
```

„(“ a „)“ jsou slova a tak musejí být odděleny o ostatního textu.

Slovo TISK si můžete vyzkoušet. Nejdříve se přepneme do desítkové soustavy a na TOS dáme číslo. Po té můžeme slovo zavolat.

DEC 12 TISK

Správně by se mělo na obrazovce objevit:

1100

14

12

C ok

Zásobník návratových adres

Nová slova:

>R (x →) ; zna (→ x)
Přesune TOS uz na TOS zna.

R> (→ x) ; zna (x →)
Přesune TOS zna na TOS uz.

R@ (→ x) ; zna (x → x)
Zkopíruje TOS zna na TOS uz

EXIT (→)
Předčasné ukončení slova – RETURN uprostřed definice.

OVER (n1 n2 → n1 n2 n1)
Zkopíruje NOS na TOS.

DROP (n →)
Smaže TOS.

2DROP (n1 n2 →)
Smaže dvě vrchní položky uz.

2DUP (n1 n2 → n1 n2 n1 n2)
Zkopíruje dvě vrchní položky uz.

ROT (n1 n2 n3 → n2 n3 n1)
Rotuje tři vrchní položky uz doleva.

2ROT (n1 n2 n3 → n3 n1 n2)
Rotuje tři vrchní položky uz doprava.

Slova v lekci nadefinovaná:

3DUP (n1 n2 n3 → n1 n2 n3 n1 n2 n3)
Zduplikuje tři vrchní položky uz.

3PICK (n1 n2 n3 → n1 n2 n3 n1)
Zkopíruje NNOS na TOS.

NOSZNA (→ n1) ; (n1 n2 → n1 n2)
Zkopíruje NOS zna na TOS uz.

Již na začátku jsem říkal, že FORTH má dva zásobníky. Doposud jsme používali jen uživatelský. V této lekci se naučíme používat zásobník návratových adres (zna). Zna se používá pro uschování návratové adresy z procedury – slova, které je právě vykonáváno tedy pro uschování adresy slova, které se bude provádět po slově vykonávaném. Tato adresa se během slova nemění, zásobník se skoro vůbec nepoužívá. Můžeme ho použit my. Před ukončením slova musíme všechny naše položky odebrat, aby byl zna ve stejném stavu jako před slovem. Druhým omezením, které musíme mít na paměti, je že některá slova již zna používají a jeho neodborným použitím bychom mohli jejich činnost ohrozit!!!

Dále budu v popisu zásobníků oddělovat uz a zna středníkem;

Slova pomocí nichž na zna ukládáme nebo z něj čteme jsou: **>R** a **R>.**

Pomocí těchto slov si ukážeme jak by mohlo být nadefinované slovo **OVER**.

:	**OVER**	**>R**	**DUP**	**R>**	**SWAP**	**;**
n1 ;	**n1 ;**	**n1 ; n2**	**n1 ; n2**	**n1 ;**	**n1 ;**	**n1 ;**
n2 ;	**n2 ;**	**;**	**n1 ;**	**n1 ;**	**n2 ;**	**n2 ;**
;	**;**	**;**	**;**	**n2 ;**	**n1 ;**	**n1 ;**

Na zásobníku jsme dostali dvě čísla. Číslo na TOS si uschováme na zna. Na uz je již jen jedno číslo, které duplikujeme. Vrátíme druhé číslo ze zna na uz. Na zásobníku máme **n1 n1 n2**. Nakonec provedeme slovo SWAP, které prohodí TOS a NOS. Výsledkem je **n1 n2 n1**.

To, že používání zna není zcela bezproblémové, si ukážeme na dalším příkladu.

Zkuste naprogramovat slovo NOSZNA, které zkopíruje NOS zna na TOS uz.

Výsledek mohl vypadat takto:

:	**NOSZNA**	**R>**	**R>**	**DUP**	**>R**	**SWAP**	**>R**	**;**
; n1	**; n1**	**n2 ; n1**	**n2 ;**	**n2 ;**	**n2 ; n1**	**n1 ; n1**	**n1 ; n1**	**n1 ; n1**
; n2	**; n2**		**n1 ;**	**n1 ;**	**n1 ;**	**n2 ;**	**; n2**	**; n2**
				n1 ;				

Dále si zkuste nadefinovat slovo **3DUP**, které zduplikuje tři vrchní položky uz a slovo **3PICK**, které zkopíruje na TOS obsah NNOS.

Kontrolní řešení:

```
3DUP        : 3DUP   >R  2DUP  R@  2ROT  R> ;

3PICK       : 3PICK   >R  OVER  R>  SWAP  ;
```

Rozhodování

IF … THEN

Nová slova:

IF . . . (TRUE) . . . THEN

IF (f →)
Testuje TOS na jeho pravdivostní hodnotu. V případě „TRUE“ se provede posloupnost slov mezi **IF** a **THEN,** v případě „FALSE“ se přeskočí a pokračuje se rovnou prvním slovem za **THEN.**

THEN (→)
Ukončuje konstrukci.

NEGATE (n → -n)
Vynásobí TOS číslem –1.

QUIT (→)
Havarijní ukončení příkazu – obdoba příkazu STOP. Další příkazy čte z klávesnice.

ABS (n → abs(n))
Uloží na TOS absolutní hodnotu TOS.

?DUP (n → n (n>0))
Zduplikuje TOS, jen když je různý od nuly.

MAX (n1 n2 → MAX(n1 . n2))
Uloží na TOS hodnotu většího z argumentů.

MIN (n1 n2 → MIN(n1 . n2))
Uloží na TOS hodnotu menšího z argumentů.

Předpokládám, že konstrukci **IF….THEN** jste podle vysvětlení v seznamu slov pochopili, můžeme si ukázat rovnou její použití. Zkusíme si napsat slovo ABS, které FORTH již zná, ale to nám nevadí.

:	ABS	DUP	0<	IF	NEGATE	THEN	;
N	**n**	**n** **n**	**n** **(n<0)**	**n**	**-n**	**(-)n**	**(-)n**

Nejdříve zduplikujeme položku na zásobníku. Poté přemažeme TOS logickou hodnotou (TOS > 0). IF zjistí, zda je na zásobníku TRUE nebo FALSE. Pokud tam bude TRUE, provede se slovo NEGATE. V opačném případě se bude pokračovat za slovem THEN.

Nyní zkuste sami naprogramovat slovo MAX nebo MIN.

Řešení:

MAX **: MAX 2DUP < IF SWAP THEN DROP ;**

MIN **: MIN 2DUP > IF SWAP THEN DROP ;**

IF ... ELSE ... THEN

Nová slova:

IF . . .(TRUE) . . . ELSE . . . (FALSE) . . . THEN

IF ELSE (f →)
IF testuje TOS na jeho pravdivostní hodnotu. V případě TRUE se vykoná posloupnost slov mezi IF a ELSE. V případě FALSE se vykoná posloupnost slov mezi ELSE a THEN.

THEN (→)
Ukončuje konstrukci.

*/ (n1 n2 n3 → ((n1 * n2) / n3))
Uloží na zásobník výsledek příkladu NNOS * NOS / TOS, přičemž mezivýsledek je počítán ve dvojnásobné přesnosti.

Slova v lekci nadefinovaná:

ROZHODNI (n → n)
Když je TOS menší než 20, vytiskne: „Maly“.
Jinak vytiskne: „Velky“.

Určitě jste konstrukci pochopili, a tak si rovnou ukážeme její použití. Zkusíme naprogramovat slovo ROZHODNI.

Řešení:

: **ROZHODNI DUP 20 < IF ." Maly" ELSE ." Velky" THEN ;**

Položku jsme zduplikovali, abychom jí neztratili, zjistili jsme, zda je menší než dvacet a podle toho jsme se vytiskli text. Na konci nesmí chybět slovo THEN. Dále si zkuste sami vymyslet pár příkladů, na kterých byste si mohli rozhodování vyzkoušet.

Cykly

Cyklus s parametrem

Nová slova:

DO . . . LOOP

DO (n1 n2 →)
Očekává v TOS počáteční hodnotu parametru cyklu a v NOS hodnotu ukončovací, s níž se cyklus již provádět nebude.

LOOP (→)
Zvětší parametr o jedničku a testuje, zda je menší než ukončovací hodnota. Pokud je, vykoná se znovu tělo cyklu, jinak se cyklus ukončí.

I (→ n)
Uloží na TOS hodnotu parametru nejvnitřnějšího cyklu.

Slova v lekci nadefinovaná:

NASOBILKA (n1 →)
Vytiskne násobky čísla, které najde na TOS.

Místo dlouhého úvodu přejdeme hnedka k příkladům. Nadefinujeme si slovo NASOBILKA, které vytiskne násobky čísla, které najde na zásobníku.

```
: NASOBILKA 10 1 DO (  Cyklus půjde od 1 do 10, násobky budou od 1 do 9  )
     CR  (  Násobek bude na novém řádku  )
     DUP (  Uschování násobeného čísla  )
     I 2DUP . ." * " . ." = " * . (  Vytiskne se příklad n * i = n*I  )
    LOOP  (  Konec cyklu  )
DROP      (  Smazání násobeného čísla  )
;
```

Jak jste si určitě všimli, pomocí cyklu se dá obyčejný program velmi zkrátit a zjednodušit.

Tento program byl zatím nejsložitější, který jsme v jazyku FORTH dělali. Vysvětlíme si na něm několik zásad, které bychom měli v jazyku FORTH dodržovat.

První zásadou je komentování programu, abyste se v něm vyznali i po delší době nebo aby mu rozuměl i někdo jiný.

Druhou zásadou je nedefinovat příliš dlouhá a složitá slova, nejen že jsou nepřehledná, ale mnohem snadněji se v nich udělají a hůře hledají chyby.

Třetí zásadou je zavést si nějakou grafickou úpravu, vyjadřující vnořenost jednotlivých programových struktur, a tu pak používat.

Tyto pravidla jsou tím víc důležitá, čím víc je složitější program.

Cyklus s parametrem 2

Nová slova:

DO . . . +LOOP

+LOOP (n →)
Zvětší parametr cyklu o TOS a testuje, zda je menší než ukončovací hodnota cyklu. Pokud je test TRUE, vykoná se další běh cyklu. Pokud je FALSE, cyklus se ukončí.

J (→ n)
Uloží na TOS hodnotu parametru druhého nejvnitřnějšího cyklu (V té samé definici!).

MOD (n1 n2 → (n1 % n2))
Uloží na TOS zbytek po dělení NOS / TOS (Dělení modulo).

LEAVE (→)
Nastaví ukončení cyklu při příštím vykonávání LOOP nebo +LOOP tím, že položí hodnotu ukončovací rovnou hodnotě cyklu.

Slova v lekci nadefinovaná:

PRVOC (n1 n2 →)
Vypočte všechna prvočísla z intervalu definovaného na uz.

Ne vždy se nám hodí cyklus, ve kterém se parametr mění jen o jedničku. Někdy je potřeba cyklu, ve kterém bychom mohli měnit parametr o kolik chceme. Takovýto cyklus můžeme v jazyku FORTH použít pomocí slov DO a +LOOP. Slovo DO pracuje stejně, jak jsme si ukázali v minulé lekci. Slovo +LOOP ale zvětší parametr cyklu o hodnotu, kterou najde na zásobníku. Test, který pak provádí, záleží na znaménku přírůstku cyklu. Pokud je přírůstek kladný, je test stejný jako u slova LOOP, tedy „I < IMAX“. Pokud je přírůstek záporný, je test negací původního, tedy „I >= IMAX“. Pokud je odpovídající podmínka splněna, nic nebrání tomu, aby cyklus pokračoval s novou hodnotou parametru. Pokud podmínka splněna není, cyklus skončí a pokračuje se prvním slovem za slovem +LOOP.

Použití tohoto cyklu si demonstrujeme na slově PRVOC. Slovo PRVOC spočte všechna prvočísla z intervalu definovaného na uz.

```
: PRVOC
     (  Na NOS máme horní mezi a na TOS spodní mezi.  )
     (  Předpokládáme, že spodní meze je vyšší než 10.  )

DUP   2   MOD   NOT   IF
      ( Testujeme zda je spodní meze lichá, )
      ( protože sudé číslo nemůže být prvočíslo.  )

1   +   THEN
      (  Teď už je určitě lichá.  )

DO    (  Testujeme čísla z daného intervalu.  )

-1
     (  Předpokládáme že je prvočíslem  )

I   3   /    3    DO
     (  Budeme dělit čísli od jedné do 1/3  )

J   I   MOD   IF
     (  Testujeme, zda je číslo dělitelné  )

NOT   LEAVE   THEN
     (  Číslo je dělitelné - není to prvočíslo )

2   +LOOP
     (  Dělme dalším lichým číslem, sudým to nejde.  )

IF (  Má TOS příznak prvočíselnosti  )

I   .   THEN
     (  Našli jsme prvočíslo, vytiskneme ho.  )

2   +LOOP
     (  Budeme testovat další liché číslo  )
```

;

Cyklus s podmínkou

BEGIN…UNTIL

BEGIN (→)
Slouží jako návěští označuje počátek cyklu.

UNTIL (→)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě FALSE opakuje znovu cyklus(pokračuje příkazem za BEGIN). V případě TRUE jej ukončí a pokračuje dál.

BEGIN…WHILE…REPEAT

BEGIN (→)
Slouží jako návěští označuje počátek cyklu.

WHILE (F →)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě TRUE se vykoná posloupnost mezi slovy WHILE a REPEAT. V případě FALSE se pokračuje slovem za REPEAT.

REPEAT (→)
Je posledním slovem cyklu BEGIN…WHILE…REPEAT. Vrací výpočet zpět za BEGIN.

BEGIN…AGAIN (nekonečný cyklus)

BEGIN (→)
Slouží jako návěští označuje počátek cyklu.

AGAIN (→)
Je posledním slovem cyklu BEGIN…AGAIN. Vždy vrací výpočet zpět za BEGIN.

Kromě cyklů s parametrem obsahuje jazyk FORTH cykly:

BEGIN…UNTIL

Klasický cyklus s testováním na konci.

BEGIN…WHILE…REPEAT

Nejdříve se provede tělo mezi BEGIN a WHILE. Poté WHILE testuje TOS na jeho pravdivostní hodnotu. V případě TRUE se vykoná tělo mezi WHILE a REPEAT. V případě FALSE skáče za REPEAT. REPEAT skáče na příkaz za BEGIN.

Příklad

```
:  ZMENSUJ  BEGIN  DUP   0   >   WHILE     1   -     REPEAT    ;
```

Tento cyklus zmenšuje TOS dokud je větší než nula.

BEGIN…AGAIN

Je nekonečný cyklus ze kterého se vyskakuje slovem EXIT.

Použití:

Potřebuji slovo, které bude stále provádět jednu posloupnost.

```
: Opakuj BEGIN  DUP  .   AGAIN ;
```

Pokud spustíte toto slovo, bude pořád tisknout TOS a nepůjde z něj vyskočit. NESPOUŠTĚT!

M4th a Baltazar

Jazyk M4th obsahuje kromě klasických FORTHovských slov, navíc slova, pomocí nichž můžeme Baltazarovi ukládat jednoduché příkazy, které jsou uvedeny vzadu v seznamu hlavních slov.

Pokud chceme ovládat Baltazara, použijeme slovo „>BALT“, kterým se přepneme do grafického režimu a od té doby se vše píše do dolního malého okénka, aby měl Baltazar dvorek pro sebe. Jelikož je okénko hodně malé, má vlastní procedury, které na něj tisknou úsporněji. Slova pro Baltazara a pro FORTH můžeme kombinovat, takže si např. vytvoříme cyklus.

```
:  Cyklus  BEGIN   1  -  DUP  krok  1  2  caruj  UNTIL  ;
```

TOSkrát provede cyklus, ve kterém Baltazar udělá krok a vyčaruje před sebe druhý předmět z první Banky.

POZOR! Pokud napíšete v malém okénku chybu, nenapíše vám to oznámení že jste napsali neznámé slovo, jako v normálním režimu, ale program zatuhne. Vzhledem k časové tísni jsem tuto chybu nestihl odstranit.

Pokud se chceme přepnout z malého okénka zpět do textového režimu, použijeme slovo „>DOS“.

Přehled nejdůležitějších definovaných slov

Slova M4th

! (n1 a →)
Uloží na adresu a položku z NOS.

( ) Komentářové závorky. Text mezi nimi se neprovádí. M4th závorky nepodporuje.

– (n1 n2 → (n1-n2))
Odečte (TOS) od (NOS), výsledek uloží na TOS.

***** (n1 n2 → (n1*n2))
Vynásobí (TOS) a (NOS), výsledek uloží na TOS.

/ (n1 n2 → (n1/n2))
Vydělí (NOS) (TOS), výsledek uloží na TOS.

+ (n1 n2 → (n1+n2))
Sečte (TOS) a (NOS), výsledek uloží na TOS.

< (n1 n2 → f(n1<n2))
Uloží na TOS logickou hodnotu výrazu (NOS) < (TOS).

= (n1 n2 → f(n1=n2))
Uloží na TOS logickou hodnotu výrazu (NOS) = (TOS).

> (n1 n2 → f(n1>n2))
Uloží na TOS logickou hodnotu výrazu (NOS) > (TOS).

0< (n → f(n<0))
Uloží na TOS logickou hodnotu výrazu (TOS) < 0.

0= (n → f(n=0))
Uloží na TOS logickou hodnotu výrazu (TOS) = 0.

0> (n → f(n>0))
Uloží na TOS logickou hodnotu výrazu (TOS) > 0.

@ (a → (a))
Uloží na TOS obsah na adrese a.

AND (f1 f2 → (f1&f2)b)
Logický součin (TOS) a (NOS) bit po bitu.

BASE (→ .(BASE).)
Proměnná, ve které je uložen základ soustavy, v níž jsou čtena vstupující(vystupující) čísla. Slovo BASE uloží na TOS adresu, na níž je uložena hodnota této proměnné.

DECIMAL (→)
Nastaví čtení a vyjadřování čísel v desítkové soustavě.

DUP (n → n n)
Zduplikuje vrchní položku uz.

NOT (f → -f)
Neguje logickou hodnotu TOS.

OR (f1 f2 → (f1|f2)b)
Logický součet (TOS) a (NOS) bit po bitu.

SWAP (x1 x2 → x2 x1)
Prohodí (TOS) s (NOS).

XOR (f1 f2 → (f1|f2)b)
Exkluzivní OR (TOS) a (NOS) bit po bitu.

. (n →)
Vytiskne (TOS) na obrazovku a smaže ho.

: xxx (→)
Začátek dvojtečkové definice slova xxx. Nastavení režimu COMPILE.

; (→)
Konec dvojtečkové definice. Nastavení režimu EXECUTE.

.“ (→)
Na obrazovku vytiskne následující text až po znak “(uvozovky). Mezi slovem **.“** a textem musí být mezera. **Omlouvám se, ale toto slovo mi těsně před odevzdáním přestalo chodit.**!

CR (→)
Na obrazovce provede přechod na začátek dalšího řádku.

>R (x →) ; zna (→ x)
Přesune TOS uz na TOS zna.

R> (→ x) ; zna (x →)
Přesune TOS zna na TOS uz.

R@ (→ x) ; zna (x → x)
Zkopíruje TOS zna na TOS uz.

EXIT (→)
Předčasné ukončení slova – RETURN uprostřed definice.

OVER (n1 n2 → n1 n2 n1)
Zkopíruje NOS na TOS.

DROP (n →)
Smaže TOS.

2DUP (n1 n2 → n1 n2 n1 n2)
Zkopíruje dvě vrchní položky uz.

2DROP (n1 n2 →)
Smaže dvě vrchní položky uz.

ROT (n1 n2 n3 → n2 n3 n1)
Rotuje tři vrchní položky uz doleva.

2ROT (n1 n2 n3 → n3 n1 n2)
Rotuje tři vrchní položky uz doprava.

IF (f →)
Testuje TOS na jeho pravdivostní hodnotu. V případě „TRUE“ se provede posloupnost slov mezi **IF** a **THEN**, v případě „FALSE“ se přeskočí a pokračuje se rovnou prvním slovem za **THEN**.

THEN (→)
Ukončuje konstrukci.

NEGATE (n → -n)
Vynásobí TOS číslem –1.

QUIT (→)
Havarijní ukončení příkazu – obdoba příkazu STOP. Další příkazy čte z klávesnice.

ABS (n → abs(n))
Uloží na TOS absolutní hodnotu TOS.

?DUP (n → n (n>0))
Zduplikuje TOS, jen když je různý od nuly.

MAX (n1 n2 → MAX(n1 . n2))
Uloží na TOS hodnotu většího z argumentů.

MIN (n1 n2 → MIN(n1 . n2))
Uloží na TOS hodnotu menšího z argumentů.

IF ELSE (f →)
IF testuje TOS na jeho pravdivostní hodnotu. V případě TRUE se vykoná posloupnost slov mezi IF a ELSE. V případě FALSE se vykoná posloupnost slov mezi ELSE a THEN.

***/** (n1 n2 n3 → ((n1 * n2) / n3))
Uloží na zásobník výsledek příkladu NNOS * NOS / TOS, přičemž mezivýsledek je počítán ve dvojnásobné přesnosti.

DO (n1 n2 →)
Očekává v TOS počáteční hodnotu parametru cyklu a v NOS hodnotu ukončovací, s níž se cyklus již provádět nebude.

LOOP (→)
Zvětší parametr o jedničku a testuje, zda je menší než ukončovací hodnota. Pokud je, vykoná se znovu tělo cyklu, jinak se cyklus ukončí.

I (→ n)
Uloží na TOS hodnotu parametru nejvnitřnějšího cyklu.

+LOOP (n →)
Zvětší parametr cyklu o TOS a testuje, zda je menší než ukončovací hodnota cyklu. Pokud je test TRUE, vykoná se další běh cyklu. Pokud je FALSE, cyklus se ukončí.

J (→ n)
Uloží na TOS hodnotu parametru druhého nejvnitřnějšího cyklu (V té samé definici!).

MOD (n1 n2 → (n1 % n2))
Uloží na TOS zbytek po dělení NOS / TOS (Dělení modulo).

LEAVE (→)
Nastaví ukončení cyklu při příštím vykonávání LOOP nebo +LOOP tím, že položí hodnotu ukončovací rovnou hodnotě cyklu.

BEGIN (→)
Slouží jako návěští označuje počátek cyklu.

UNTIL (→)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě FALSE opakuje znovu cyklus(pokračuje příkazem za BEGIN). V případě TRUE jej ukončí a pokračuje dál.

WHILE (F →)
Testuje (TOS) na jeho pravdivostní hodnotu. V případě TRUE se vykoná posloupnost mezi slovy WHILE a REPEAT. V případě FALSE se pokračuje slovem za REPEAT.

REPEAT (→)
Je posledním slovem cyklu BEGIN…WHILE…REPEAT. Vrací výpočet zpět za BEGIN.

AGAIN (→)
Je posledním slovem cyklu BEGIN…AGAIN. Vždy vrací výpočet zpět za BEGIN.

Slova týkající se příkazů pro Baltazara

>BALT (→)
Přepne na grafický mód s tím, že se bude psát do dolního okénka, aby po horním mohl chodit Baltazar.

>DOS (→)
Přepne zpět na textový režim.

barvapred (→ N)
Na TOS dá číslo barvy před Baltazarem.

predmetpred (→ N)
Na TOS dá číslo předmětu před Baltazarem.

smer@ (→ N)
Na TOS dá číslo směru do kterého je Baltazar otočen.

vidit! (F →)
Na TOS čeká 1 nebo 0, podle toho nastaví viditelnost Baltazara. A na TOS dá původní viditelnost.

pozice! (X Y →)
Přesune Baltazara na souřadnice ze zásobníku.

pozice@ (→ X Y)
Na uz dá souřadnice Baltazara.

vyberpredmet (→ N N1)
Nechá uživatele vybrat předmět z bank. Na NOS dá číslo banky a na TOS číslo předmětu.

caruj (N N1 →)
Vyčaruje předmět s číslem s TOS z banky s číslem NOS.

ukazse (→)
Nastaví Baltazara viditelného.

zmiz (→)
Nastaví Baltazara neviditelného.

krok (→)
Baltazar udělá krok.

vpravo (→)
Baltazar se otočí doprava.

vlevo (→)
Baltazar se otočí doleva.

bezobl (→)
Nastaví čarování bez obláčků.

sobl (→)
Nastaví čarování s obláčky.

Slova z příkladů

2NA (n → (n*n))
Umocní (TOS) na druhou. Na TOS uloží výsledek.

1+ (n → (n+1))
Přičte k (TOS) jedničku.

1- (n → (n-1))
Odečte od (TOS) jedničku.

BIN (→)
Nastaví čtení a vyjadřování čísel v dvojkové soustavě.

HEX (→)
Nastaví čtení a vyjadřování čísel v šestnáctkové soustavě.

OCT (→)
Nastaví čtení a vyjadřování čísel v osmičkové soustavě.

TISK (n →)
Vytiskne TOS v BIN , OCT, DEC, HEX soustavě.

3DUP (n1 n2 n3 → n1 n2 n3 n1 n2 n3)
Zduplikuje tři vrchní položky uz.

3PICK (n1 n2 n3 → n1 n2 n3 n1)
Zkopíruje NNOS na TOS.

NOSZNA (→ n1) ; (n1 n2 → n1 n2)
Zkopíruje NOS zna na TOS uz.

ROZHODNI (n → n)
Když je TOS menší než 20, vytiskne: „Maly“.
Jinak vytiskne: „Velky“.

NASOBILKA (n1 →)
Vytiskne násobky čísla, které najde na TOS.

PRVOC (n1 n2 →)
Vypočte všechna prvočísla z intervalu definovaného na uz.