

Neknaj Language Processing System

Bem130

2023/12/24

目次

第 I 部	概要	1
1	特徴	1
第 II 部	Nekanj Language for Programming	1
2	サンプルコード	1
3	ツールキット	1
4	記法	2
4.1	コメント	2
4.2	型	2
4.3	式	2
4.4	文	2
4.5	ブロック	3
4.6	単ブロック	3
4.7	制御構造	3
4.8	関数	4
第 III 部	Nekanj Virtual Machine	4
5	概要	4
6	命令セット	4

第 I 部

概要

スタックマシンを基本にした Bem130 の自作プログラミング言語とその処理システム

1 特徴

特徴	理由	主な対象
逆ポーランド記法	中置演算子や括弧を含む式の解析が難しかった為 引数の式を先に書くことで処理の順番が明確になる為	NLP
代入を表す:>	等号として用いられる=との違いを明確にするため 代入の方向を明確にする為 顔文字のようで可愛い為	NLP
右に記述する代入先の変数	式を先に書くことで代入の処理の順番が明確になる為	NLP
コメントアウトとノート	不要なコードと、必要なメモを区別するため	NLP
関数の定義の巻き上げ	定義文の前でも使用できるのが便利で気に入った為	NLP
変数の定義の巻き上げ	同じスコープの名前が指すものを統一する為	NLP
浮動小数点数は基数 10 が基本	2 進化による丸め誤差が気に入らなかった為	BemLib for NVM
コンパイル結果を include する	ソースコードの include が面倒そうに感じた為	NLPS
むやみにハンドリングする例外	例外の為に特別な処理を作るのが気に入らなかった為	NLPS

第 II 部

Nekanj Language for Programming

逆ポーランド記法を基本とするプログラミング言語

2 サンプルコード

```
!include: stdcalc;  
!using: stdcalc;  
!replace: pi: 3.1415;  
## block comment *##  
  
!fn: 4.int(4.int: max): main {  
  !local: 4.int: z; # this is a line comment  
  0 0 add :> !local: 4.int: y;  
  0 :> return;  
}
```

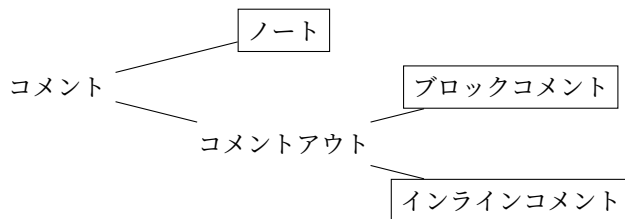
3 ツールキット

JavaScript 版と C++ 版があるが、どちらも完成していない

種類	ファイル名	説明
仮想マシン	nve.worker.js	
仮想マシン	nve.worker.cpp	
コンパイラ類	nlp.ts	
コンパイラ類	nlp.js	nlp.ts をコンパイルしたもの
エディタ	editor.html	nlp.ts 向けの GUI
エディタ	debugger.html	nlp.ts 向けの GUI, editor.html よりも多くの情報を表示

4 記法

4.1 コメント



```
#: ノート
```

```
# インラインコメント
```

```
## ブロックコメント *#
```

4.2 型

```
サイズ.種類
```

4.3 式

トークンをスペースでつないだもの

```
トークン トークン トークン ...
```

4.4 文

4.4.1 トップレベルの文

■4.4.1.1 include

```
!include: 名前;
```

■4.4.1.2 using

```
!using: 名前;
```

■4.4.1.3 global

```
!global: 型: 名前;
```

■4.4.1.4 replace

```
!replace: 置き換え前: 置き換え後;
```

4.4.2 ブロック内の文

■4.4.2.1 式文

```
式;
```

■4.4.2.2 local 宣言

```
!local: 型: 名前;
```

■4.4.2.3 代入

```
式 :> 名前;
```

■4.4.2.4 local 宣言付代入

```
式 :> !local: 型: 名前;
```

■4.4.2.5 return

```
式 :> return;
```

4.5 ブロック

文, 構造を複数書いたもの

構造には、単ブロックと制御構造が含まれる

ブロック要素には、文と構造が含まれる

ブロックは入れ子にすることができる

```
{  
  ブロック要素  
  ブロック要素  
  ブロック要素  
  ...  
}
```

4.6 単ブロック

```
ブロック;
```

4.7 制御構造

4.7.1 基本形

条件式, 種類, ブロック を組にしたもの

種類によって細かい記法は異なる

```
!ctrl:(式) 種類 ブロック;
```

4.7.2 while

```
!ctrl:(式) while ブロック;
```

4.7.3 if

基本となる if の後ろに、任意個の elseif と一つの else を付けることができる

```
!ctrl:(式) if ブロック;
```

```
!ctrl:(式) if ブロック else ブロック;
```

```
!ctrl:(式) if ブロック (式) elseif ブロック;
```

```
!ctrl:(式) if ブロック (式) elseif ブロック else ブロック;
```

4.8 関数

```
!fn:戻り値型(引数): 名前 ブロック;
```

4.8.1 引数

```
型: 名前, 型: 名前, 型: 名前, ...;
```

第 III 部

Nekanj Virtual Machine

5 概要

1 ワード 32bits(4Bytes) のスタックマシン

6 命令セット

命令	引数	消費	追加	スタック長	処理
00	push	v	-	v	+1 スタックに値 v を入れる
01	fram	n	-	0($\times n$)	+n スタックに n 回 0 を入れる
02	pop	-	v	-	-1 スタックトップの値 v を 1 つ消す
03	popn	n	v($\times n$)	-	-n スタックトップの値 v を n 個消す
04	setv	l	v	-	-1 l 個目のローカル変数に値 v を入れる
05	getv	l	-	v	+1 l 個目のローカル変数から値 v を複製する
06	setgv	g	v	-	-1 g 個目のグローバル変数に値 v を入れる
07	getgv	g	-	v	+1 g 個目のグローバル変数から値 v を複製する
08	seth	-	h v	-	-2 ヒープ領域の h 番目に値 v を入れる
09	getv	-	h	v	+1 ヒープ領域の h 番目から値 v を複製する
0a	jmp	p	-	-	± 0 アドレス p までジャンプする
0b	ifjmp	p	cn	-	-1 cn が true ならば、アドレス p までジャンプする
0c	call	p	-	fp pc	+2 関数をの呼ぶ処理をし、アドレス p までジャンプする
0d	ret	n	fp pc v($\times n$)	-	-2-n 関数を呼ぶ前に戻って、引数分 n 回 pop する
10	equ	-	a b	v	-1 a == b
11	les	-	a b	v	-1 a < b
12	grt	-	a b	v	-1 a > b
13	not	-	a	v	± 0 not a
14	and	-	a b	v	-1 a and b
15	or	-	a b	v	-1 a or b
16	xor	-	a b	v	-1 a xor b
17	notb	-	a	v	± 0 not a
18	andb	-	a b	v	-1 a and b
19	orb	-	a b	v	-1 a or b
1a	xorb	-	a b	v	-1 a xor b
1b	lsft	-	a b	v	-1 a « b
1b	rsft	-	a b	v	-1 a » b
20	add	-	a b	v	-1 a + b
21	addc	-	a b x	c s	-1 a + b 繰り上がりは c

表 1 略語

NLPS	Neknaj Language Processing System
NLP	Neknaj Language for Programming
NLPO	Neknaj Language for Programming - Object file
NVA, NVASM	Neknaj Virtual machine - Assembly language
NVMC	Neknaj Virtual machine - Machine Code
NVM	Neknaj Virtual Machine