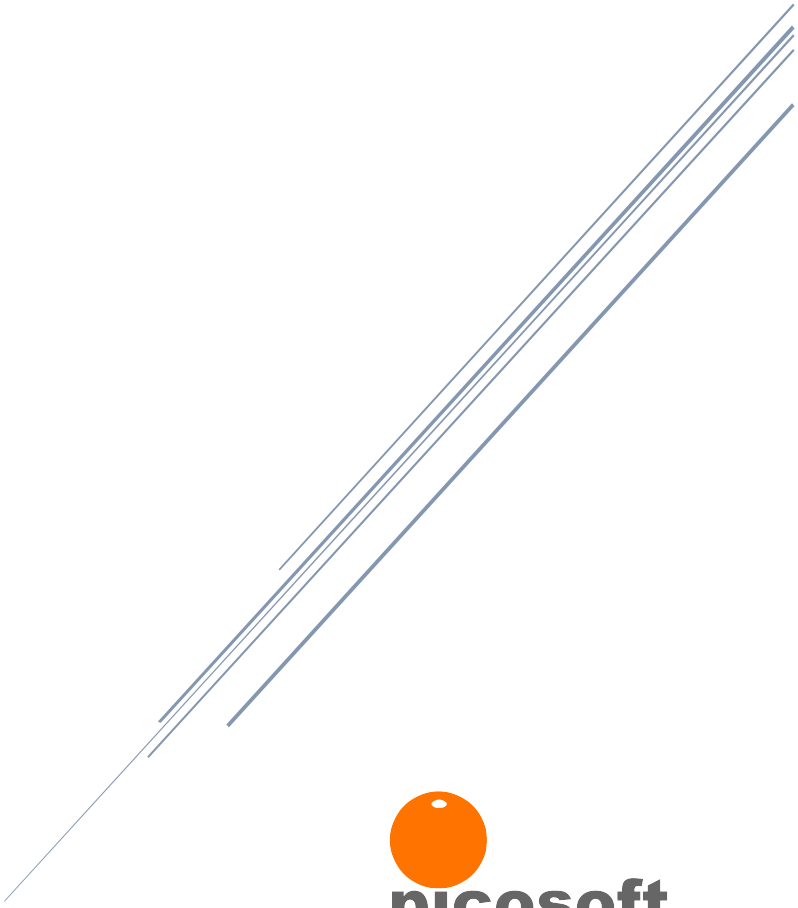
ORANGE-BASIC II

ユーザーズマニュアル



picosoft

はじめに

ORANGE-BASIC II は、BASIC 処理系を理解するためのシンプルなインタープリターで、すべて C 言語で記述してあります。(Arduino 関数を呼び出す部分のみ拡張子が cpp のファイルになっていますが、C 言語の範囲で記述しています。)

開発の目的の一つとして、古い CPU のベンチマークで有名な ASCIIART.BAS を無変更で動作させ、最新の CPU と比較したいという思いがありました。そのため、いわゆる Tiny BASIC と呼ばれる最小限の機能しかない BASIC ではなく、Microsoft 系 BASIC の機能を取り込んでいます。具体的には実数演算、多次元配列などです。

また、独自の機能として、Arduino 関数の一部をコマンドとしても使用できるようにしました。お手軽にハードのテストを行うのに便利かもしれません。

機能を充実させることよりもシンプルで理解しやすいことを目指しました。ソースコードは丁寧にモジュール分割してあり、機能追加は容易のはずです。(とはいえ、プログラム言語処理系の基礎知識は必要です。完全理解するための解説本は別途出版予定です。)

このたびは ORANGE-console-Shield という製品の付属品として ORANGE-BASIC II Ver 1.0 全ソースコードをリリースします。本書は ORANGE-BASIC II の言語仕様、そのビルド方法、機能追加の方法について説明しています。

目次

1.　ビルドと動作確認

1.1.　配布ファイル

配布ファイルは以下のようなフォルダー構成になっています。

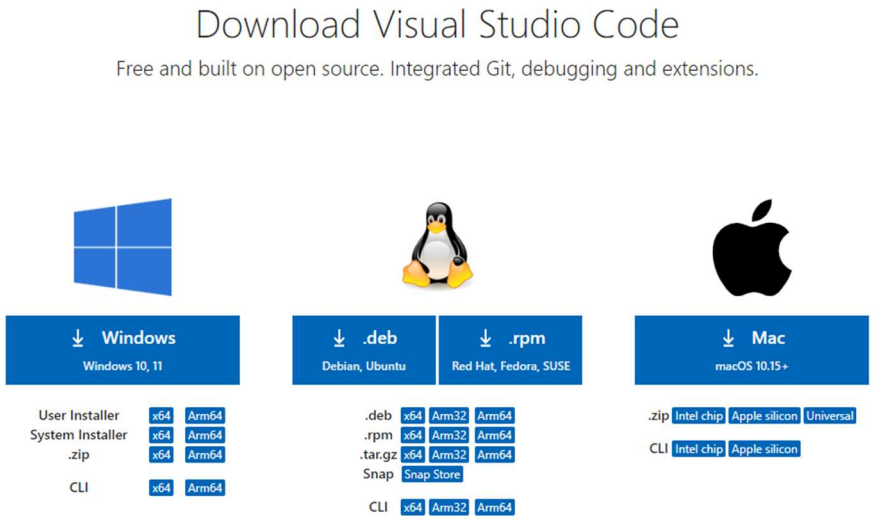
<Arduino>	<ORANGE-BASIC-II>	ソースコード一式(Arduino IDE 用)
<BAS>	サンプルプログラム	
<ORANGE-BASIC-II>	ソースコード一式(VSCode 用)	
ORANGE-BASIC II ユーザーズマニュアル.pdf	ORANGE-BASIC II ユーザーズマニュアル(本書)	
ORANGE-console-Shield組立キット取扱説明書.pdf	ORANGE-console-Shield 組立キット取扱説明書	
ORANGE-console-Shield-sch.pdf	ORANGE-console-Shield 回路図	

BASIC 処理系はソースコードで配布しますので、ご自身でビルドしてからターゲットのマイコンボードに書き込む必要があります。まずは、<ORANGE-BASIC-II>フォルダーを適当な場所に丸ごとコピーしておきます。

ORANGE-BASIC II は Arduino IDE ではなく、Visual Studio Code(以降 VSCode)とその拡張機能の PlatformIO を用いて開発しています。配布ディスクのルート直下の<ORANGE-BASIC-II>にソースコード一式があります。

1.2.　VSCode のインストール

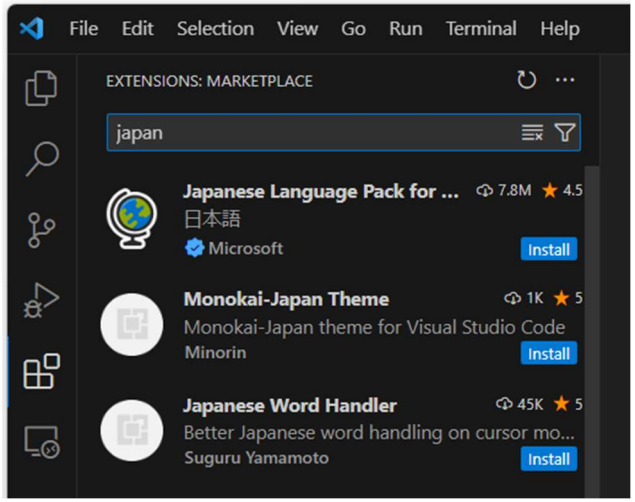
VSCode ダウンロードページ(https://code.visualstudio.com/download)で、OS を指定して VSCocde のインストーラーをダウンロードします。



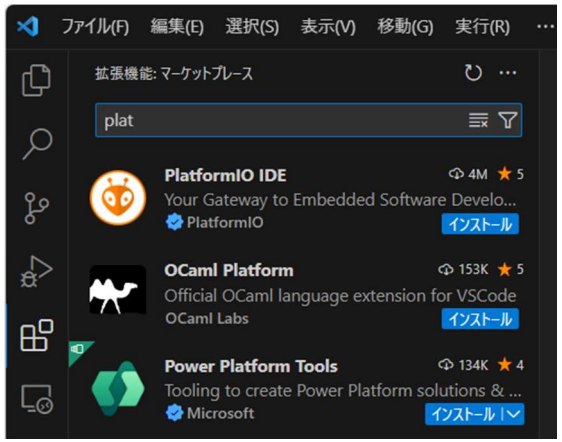
ダウンロードが完了したら、インストーラーを実行して VSCode をインストールしてください。

1.3. PlatformIO のインストール

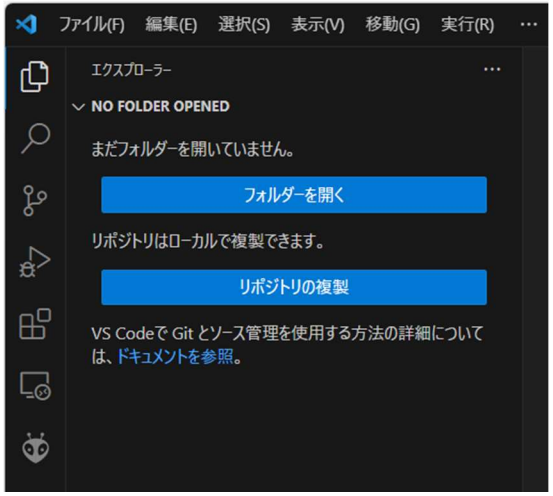
VSCode を起動したら、Japanese Language Pack for VS Code をインストールしましょう。Ctrl+Shift+X キーを押して検索用入力フィールドに japan 入力し、Japanese Language Pack for VS Code の下の Install ボタンをクリックします。インストール後、Restart ボタンが現れたらクリックし再起動します。



同様に Ctrl+Shift+X キーを押して検索用入力フィールドに plat と入力し、拡張機能から PlatformIO IDE を インストールします。



VSCode を終了し、再起動すると左側にアイコン  が現れます。

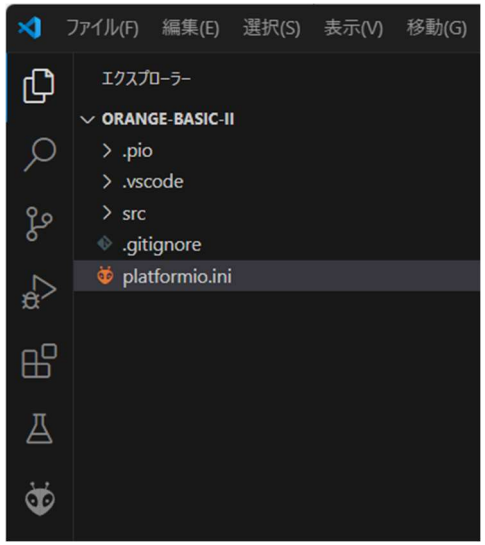


PlatformIO のインストールが進まない場合は、裏でエラーが出ている可能性があります。Python3 のインストールや path を確認してみましょう。

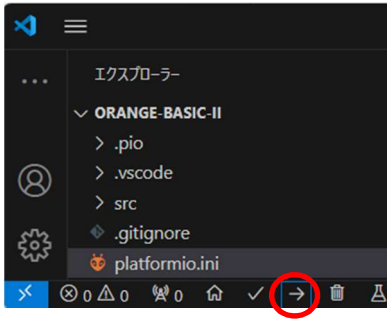
1.4.　ビルド＆書込み

VSCode を起動したら、ファイル(F)→フォルダーを開くを選択してください。USB メモリーのルート直下の ORANGE-BASIC-II フォルダーを指定します。(実際には USB メモリーの内容をハードディスクにコピーしていると思いますので、ハードディスク上の該当フォルダーを指定してください。)

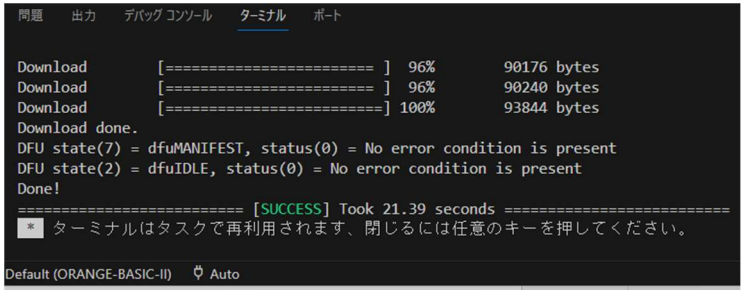
しばらくすると、次のような画面になります。



この状態で、画面下の  をクリックします。



SUCCESS と表示されたら書込み完了です。



失敗した場合は、USB ケーブルを抜き差ししてから（あるいは、Arduino Uno 上のリセットボタンを押してから）再度実行してみてください。それでも、うまく行かない場合は一度だけ Arduino IDE でビルドしてみてください。次からは成功すると思います。

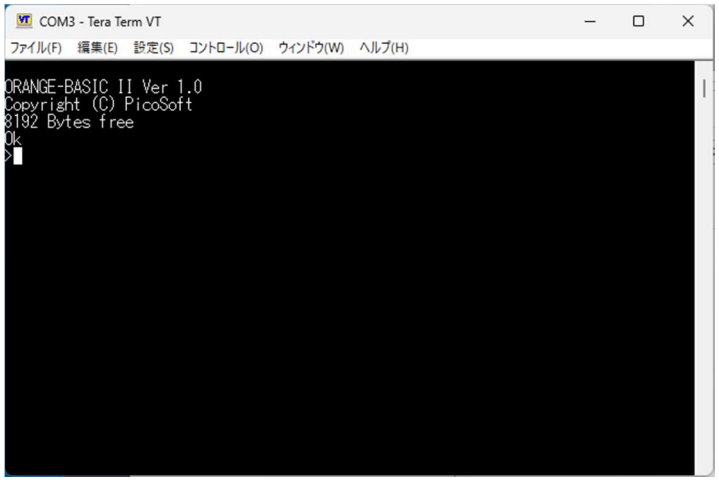
Arduino Uno R4 でのビルド方法の概略は以下の通りです。

① Arduino と PC を USB ケーブルで接続します。
② ツール→ボード→Arduino UNO R4 Boards を選びます。
③ ツール→ポート→COMxx を選びます。(xx はあらかじめデバイスマネージャーなどで調べておきます。)
④ ファイル→開くで、/Arduino/ORANGE-BASIC-II/ORANGE-BASIC-II.ino を開きます。
⑤ Arduino のリセットボタンを押してから、Arduino IDE の書き込みボタンをクリックします。

1.5. 動作確認

最初の動作確認はPCと接続してください。ソースコードを無変更でビルドした場合は、ORANGE-console-Shieldとつないでも何も出力されません。

Arduinoにファームを書き込んだら、PCとUSBケーブルで接続しTera Termなどを起動します。シリアルポートを設定し、ボーレートを115200bpsで接続します。Arduinoのリセットボタンを押すと起動メッセージが表示されます。



Arduino Uno R4にORANGE-console-Shieldを使用する場合はソースコードを変更しないと動作しません。ORANGE-BASIC-II/src/iocs/Iocs.c中の#define SERIAL_CH 0を#define SERIAL_CH 1に変更してからビルドしてください。ORANGE-console-ShieldのVGAモニターとUSBキーボードが使用できるようになります。

1.6. MicroSD Cardモジュール

Arduino Uno R4でMicroSDカードを使用するには、下図のようなMicroSD Cardモジュール(*)を接続しておく必要があります。files、load、saveなどが使用できるようになります。

MicroSD Card側	Arduino側
3.3V	3.3V
CS	D10
MOSI	D11
CLK	D13
MISO	D12
GND	GND

* https://store.shopping.yahoo.co.jp/orangepicoshop/pico-m-03l.html

2. ORANGE-BASIC II の言語仕様

2.1. ダイレクトモード

ORANGE-BASIC II が起動すると、画面にプロンプト(prompt)が「>」が出力されます。この状態をダイレクトモード(direct mode)といいます。この状態でコマンド(command、命令)を入力して Enter キーを押下すると、すぐに実行されます。

コマンドにはステートメント(statement、文)とファンクション(function、関数)があります。ステートメントやファンクションで使用するキーワードの大文字/小文字を区別しません。

[例]

```
>print 1 + 2
3
Ok
```

他の BASIC 処理系にない特徴として、ファンクションもコマンドとして実行できます。

[例] Arduino ボード上の LED を点灯します。

```
>pinmode(13, OUTPUT)
Ok
>digitalWrite(13, HIGH)
Ok
>
```

2.2. プログラムモード

ダイレクトモードで先頭の行番号に続けてステートメントを入力すると、プログラムとして格納されます。行番号は 1～65535 まで使用できます。格納されたプログラムは list コマンドで確認できます。また、run コマンドで行番号の小さい順に実行されます。

[例]

```
>10 a = 1
>20 b = 2
>30 print a + b
>list
10 a = 1
20 b = 2
30 print a + b
Ok
>run
3
Ok
>
```

プログラムが正常実行されると、「Ok」が表示されます。エラーがあったときは、エラーメッセージを表示してからプロンプトが表示されます。

2.3. ステートメント

一つの行番号に対して、複数のステートメントを記述できます。その場合は「:」で区切ります。ただし、1 行は 255 文字以内という制限があります。

[例]

```
>for i = 0 to 2: print i: next
0
1
2
Ok
>
```

2.4. ファンクション

ファンクションはキーワードに続き「(」と「)」でパラメーターを囲みます。パラメーターがないときもカッコは必要です。また、パラメーターが複数あるときは「,」で区切ります。ファンクションは通常は式の中で使用しますが、前述の通りコマンドとしても使用できます。

2.5. 変数

単純変数はすべて単精度実数で、起動時に0に初期化されます。変数名は、英文字で始める必要があります。途中の文字には英文字の他に「_」、「.」、「$」、「!」、「#」「%」が使用できます。変数名の長さは無制限で全文字比較しますが、大文字/小文字を区別しません。

2.6. 配列変数

配列変数は3次元まで使用できます。配列は使用前にDIMステートメントで次元と要素数を宣言しておく必要があります。配列変数は単純変数と同様ですべての要素は0で初期化され、変数名の規則も同じです。

[例]

```
>list
10 dim point(10, 10)
20 point(1, 1) = 100
30 print point(1, 1)
Ok
>run
100
Ok
>
```

2.7. 定数

文字列定数は、「”」と「”」で囲んだものです。文字列定数は大文字と小文字が区別されます。

[例]

```
>print ”Hello, World!”
Hello, World!
Ok
>
```


数値定数は単精度実数で保持されますが、printステートメントで表示するときには、整数として表示できるときは整数で表示します。

[例]

```
>print 1
1
Ok
>print 1.0
1
Ok
>print 2e3
2000
Ok
>print 3e-3
0.003000
Ok
>
```

数値が整数として表現できる場合は先頭に「&h」を付加すると16進表現が使用できます。

[例]

```
>print &h7fff
32767
Ok
>
```

システム定数として、Arduinoで使われている定数が使用できます。使用できる定数は通りです。(Custom.cppの実装によります。)

システム定数	値
false	0
true	1
LOW	0
HIGH	1
CHANGE	2
FALLING	3
RISING	4
INPUT	1
OUTPUT	8
INPUT_PULLUP	2
INPUT_PULLDOWN	4
LSBFIRST	0
MSBFIRST	1

2.8. 式と演算子

式とは、定数や変数、ファンクションを演算子で結んだものです。演算は次の順位によって行われます。

順位	演算	意味
1	カッコで囲まれたもの	カッコで囲った式の演算を優先します。
2	ファンクション	指定ファンクションを実行した結果を返します。
3	^	べき乗を計算します。
4	-	符号を反転します。
5	*、/	乗算または除算を実行します。
6	+、-	加算または減算を実行します。
7	<、<=、>、>=	大小比較をし、結果が真ならば-1、偽ならば0を返します。
8	=、<>	等値比較をし、結果が真ならば-1、偽ならば0を返します。
9	NOT	ビットごとのNOT演算をした結果を返します。
10	AND	ビットごとのAND演算をした結果を返します。
11	OR	ビットごとのOR演算をした結果を返します。
12	XOR	ビットごとのXOR演算をした結果を返します。

2.9. ステートメント一覧

DELETE

機能

指定されたプログラム行を削除します。

書式

DELETE [<開始行番号>][-[<終了行番号>]]

例

```
>delete 1000
Ok
>delete 10
Ok
>delete 20-30
Ok
>delete -100
Ok
>delete 200-
Ok
>
```

DIM

機能

使用する配列変数の添字の最大値を設定し、メモリー上にその配列の領域を確保します。添字の最小値は0、最大値はシステムのメモリー容量に依存します。

書式

DIM　変数名(<最大値> [,<最大値>] [,<最大値>])

例

```
>list
10 dim a(10)
20 for i = 0 to 10
30      a(i) = i
40 next
50 for i = 0 to 10
60      print a(i);" ";
70 next
Ok
>run
0 1 2 3 4 5 6 7 8 9 10
Ok
>
```


END

機能

プログラムを終了します。

書式

END

FILES

機能

SD カード上のファイル一覧を表示します。

書式

files

例

```
>files
ASCIIART.BAS
DIM.BAS
SIEVE.BAS
TEK.BAS
Ok
>
```


FOR

機能

FOR から NEXT の間にあるステートメントを繰り返し実行します。

FOR に続く<変数名>=<開始値>で変数を初期化します。変数の値が<終値>に達した場合は、NEXT の次に制御が移ります。繰り返しごとに変数の内容は<増分>の値を足した値に更新されます。<増分>を省力すると増分の値は 1 になります。

FOR～NEXT は入れ子にすることができます。その場合は、NEXT の次に FOR で指定した変数名を記述して、どのループかを明示します。NEXT の次の変数名を省略した場合は、直近の FOR が対象となります。

書式

FOR <変数名> = <開始値> TO <終了値> [STEP <増分>]

NEXT <変数名>

例

```
>list
10 for i = 1 to 5 step 2
20      for j = 1 to 2 step 1
30              print i;" ";j
40      next j
50 next i
Ok
>run
1 1
1 2
3 1
3 2
5 1
5 2

Ok
>
```

GOSUB

機能

指定した行番号に制御を移します。（サブルーチンを呼び出します。）

移動した先の延長で RETURN 文があると、GOSUB 文の直後に制御が戻ります。（サブルーチンから戻ります。）

GOSUB～RETURN は入れ子にすることができます。

書式

GOSUB <行番号>

例

```
>list
10 a = 1: gosub 30: print a: end
20 end
30 a = a * 1000
40 return

Ok
>run
1000
Ok
>
```


GOTO

機能

指定した行に制御を移します。

書式

GOTO <行番号>

例

```
>list
10 i = 1
20 if i = 10 then print: end
30 print i;" ";
40 i = i + 1
50 goto 20
Ok
>run
1 2 3 4 5 6 7 8 9
Ok
>
```

INPUT

機能

キーボードから入力されたデータを変数に代入します。キー入力前に、指定したプロンプト(任意の文字列)を表示します。プロンプトを省略した場合は、"?"のみを表示します。

書式

INPUT [<プロンプト>;]<変数名>

例

```
>list
10 input a
20 input "b = "; b
30 print "a + b = "; a + b
Ok
>run
?1
b = 2
a + b = 3
Ok
>
```


IF

機能

条件判断を行い、その結果に応じた処理を行います。

条件式が真(0以外)の場合は文1を実行し、偽(0)の場合は文2を実行します。ただし、条件式が偽でELSEが省略された場合は次の行番号の文を実行します。

書式

IF <条件式> THEN <文1> [ELSE <文2>]

例

```
>list
10 i = 1
20 if i = 10 then print: end else print i;" ";
30 i = i + 1
40 goto 20
Ok
>run
1 2 3 4 5 6 7 8 9
Ok
>
```

KILL

機能

SD カード上の指定したファイルを削除します。ファイル名は大文字/小文字を区別しません。

書式

Kill <ファイル名>

例

```
>files
ASCIIART.BAS
DIM.BAS
SIEVE.BAS
TEK.BAS
Ok
>kill "tek.bas"
Ok
>files
ASCIIART.BAS
DIM.BAS
SIEVE.BAS
Ok
>
```


LET

機能

変数に値を代入します。LET は省略できます。通常は LET を省力して記述します。

書式

[LET] <変数名> = <式>

例

```
>list
10 let a = 1
20 b = 2
30 print a;" ";b
Ok
>run
1 2
Ok
>
```

LIST

機能

メモリーにあるプログラムを出力します。

書式

LIST [<開始行番号>][-[<終了行番号>]]

例

```
>list
10 a = 1
20 b = 1
30 print a + b
40 end
Ok
>list 20-
20 b = 1
30 print a + b
40 end
Ok
>list -20
10 a = 1
20 b = 1
Ok
>list 20-30
20 b = 1
30 print a + b
Ok
>
```


LOAD

機能

指定したファイル名のプログラムを SD カードから読み込みます。ファイル名は大文字/小文字を区別しません。LOAD 実行前にメモリー上にあったプログラムは消去されます。

書式

LOAD <ファイル名>

例

```
>load "dim.bas"
Ok
>list
1000 X_MAX = 9
1010 Y_MAX = 9
1020 dim table(X_MAX, Y_MAX)
1030 for x = 1 to X_MAX
1040     for y = 1 to Y_MAX
1050             table(x, y) = x * y
1060     next y
1070 next x
1080 for x = 1 to X_MAX
1090     for y = 1 to Y_MAX
1100             print table(x, y); " ";
1110     next y
1120     print
1130 next x
Ok
```

NAME

機能

ファイルのファイル名を変更します。

書式

NAME <旧ファイル名> AS <新ファイル名>

例

未実装です。

NEW

機能

メモリーにあるプログラムをすべて削除し、変数を 0 で初期化します。

書式

NEW

例

```
>new
Ok
>
```

PRINT

機能

画面に指定した文字列や数値を出力し、改行します。PRINT だけを指定すると改行されます。

複数のデータを表示する場合はセミコロン(;)で区切ります。最後のデータの後にセミコロン(;)を置くと改行し、そうでないときは改行します。

PRINT の省略形としてクエスチョンマーク(?)を使用することができます。

書式

PRINT [<式>・・・]

例

```
>print "ABC"
ABC
Ok
>print 123
123
Ok
>print 123;"ABC";
123ABCOk
>print 123;"ABC"
123ABC
Ok
>
```


RANDOMIZE

機能

rand 関数によって使用される擬似乱数ジェネレーターの開始シード値を設定します。

書式

RANDOMIZE <シード値>

例

```
>list
10 randomize millis()
20 for i = 1 to 10
30   print rnd(100)
40 next
Ok
>run
72
98
82
23
54
86
1
28
44
55
Ok
>
```

REM

機能

REM に続く文字列をプログラムとして無視します。REM の代わりに「'」を用いることもできます。

書式

REM［<任意の文字列>］

例

```
>list
10 rem
20 rem print 1
30 print 2
40 print 3 ' print 4
Ok
>run
2
3
Ok
>
```


RENUM

機能

プログラムの行番号を付け直します。パラメーターを省略すると、新しい行番号は 1000 から始まり、1010、1020・・・と付けられます。GOTO や GOSUB 使われている行番号も更新されます。

書式

RENUM［<新行番号>][,<増分>］

例

```
>renum
Ok
>renum 1
Ok
>renum 10, 10
Ok
>
```

RUN

機能

メモリーにあるプログラムを実行します。プログラムが正常実行されると、「Ok」が表示されます。オプション(,t)を指定した場合、実行時間が 1 ms 以上のときには、Ok の後に実行時間が表示されます。エラーがあったときは、エラーメッセージを表示してからプロンプトが表示されます。

プログラムを強制終了させるときは、ESC キーを押します。

書式

RUN [,t]

例

```
>list
10 for i = 1 to 10000: next
20 print i
Ok
>run
10001
Ok
>run, t
10001
Ok (179ms)
>
```


SAVE

機能

プログラムを指定したファイル名で SD カード上のファイルに保存します。ファイル名は大文字/小文字を区別しません。

書式

SAVE <ファイル名>

例

```
>save "test.bas"
Ok
```

TROFF

機能

プログラムのトレース機能を OFF にします。

書式

TROFF

TRON

機能

プログラムのトレース機能を ON にします。トレース機能が ON なると実行している行の行番号を表示するようになります。

書式

TRON

例

```
>list
10 tron
20 a = 1
30 if a = 1 then goto 50
40 a = a + 1
50 print "a = "; a
Ok
>run
[20][30][50]a = 1
Ok
>
```

2.10.　ファンクション一覧

ABS

機能

指定した式の絶対値を返します。

書式

ABS(<式>)

例

```
>print abs(-1)
 1
Ok
>
```


CHR$

機能

指定したキャラクターコードに対応する文字列を返します。

書式

CHR$(<式>)

例

```
>list
10 for c = &h41 to &h45
20 print chr$(c)
30 next
Ok
>run
A
B
C
D
E
Ok
>
```

COS

機能

指定した式の余弦(cos)を返します。

書式

cos(<式>)

例

```
>print cos(2 * 3.14159)
1
Ok
>
```


FRE

機能

テキストエリアの未使用領域の大きさを返します。

書式

FRE()

例

```
>print fre()
8192
Ok
>
```

INT

機能

指定した式の値を超えない最大の整数を返します。

書式

INT(<式>)

例

```
>print int(-2.3)
-3
Ok
>print int(2.3)
2
Ok
>
```


RND

機能

0 から 指定した式-1 までの範囲の擬似乱数を返します。

書式

RND(<式>)

例

```
>list
10 for i = 1 to 10
20   print rnd(100)
30 next
Ok
>run
33
43
62
29
0
8
52
56
56
19
Ok
>
```

SIN

機能

指定した式の正弦(sin)を返します。

書式

sin(<式>)

例

```
>print sin(3.14159 / 2)
1
Ok
>
```


SQR

機能

指定した式の平方根を返します。

書式

SQR(<式>)

例

```
>print sqr(2)
1.414213
Ok
>
```

TAN

機能

指定した式の正接(tan)を返します。

式はラジアン単位です。

書式

tan(<式>)

例

```
>print tan(0)
0
Ok
>
```


2.11. Arduino 関数

Arduino 関数のうちで以下のものは BASIC のコマンドとして使用できます。ただし、実装は最低限の参考用ですので、機器に合わせて追加実装が必要です。(3.4.参照)

デジタル I/O			
書式	戻り値	パラメーター	機能
digitalRead(pin)	HIGH、LOW	pin: 読み取るピンの番号	指定したデジタルピンの状態を読み取ります。
digitalWrite(pin, value)	なし	pin: 設定するピンの番号 value: HIGH、LOW	デジタルピンに値を出力します。
pinMode(pin, mode)	なし	pin: 設定するピンの番号 mode: INPUT 、 OUTPUT 、 INPUT_PULLUP	ピンの動作モード設定します。

アナログ I/O			
書式	戻り値	パラメーター	機能
analogRead(pin)	0-1023	pin: 読み取るピンの番号	指定したアナログピンの値を読み取ります。
analogWrite(pin, value)	なし	pin: 設定するピンの番号 value: デューティ比	指定したピンにアナログ値(PWM波)を書き込みます。

時間			
書式	戻り値	パラメーター	機能
delay(ms)	なし	ms: 一時停止する時間(ミリ秒単位)	指定した時間だけプログラムを一時停止します。
micros()	経過時間	なし	プログラムを起動してからの経過時間をマイクロ秒単位で返却します。
millis()	経過時間	なし	プログラムを起動してからの経過時間をミリ秒単位で返却します。

Wire			
書式	戻り値	パラメーター	機能
Wire.begin()	なし	なし	Wireライブラリを初期化します。
Wire.requestFrom(address, count)	受信したバイト数	address: データを要求するデバイスのアドレス(7ビット) count: 要求するデータのバイト数	他のデバイスにデータを要求します。
Wire.beginTransmission(address)	なし	address: 送信対象のアドレス(7ビット)	指定したアドレスの I2C スレーブに対して送信処理を始めます。
Wire.endTransmission() Wire.endTransmission(stop)	0: 成功 1: 送ろうとしたデータが送信バッファのサイズを超えた 2: アドレスを送信し、NACK を受信した 3: データを送信し、NACK を受信した 4: その他のエラー 5: タイムアウト	stop(省略可): true に設定すると stop メッセージをリクエストのあと送信し、I2C バスを開放します(デフォルト)。false に設定すると restart メッセージをリクエストのあと送信し、コネクションを維持します。	スレーブデバイスに対する送信を完了します。
Wire.write(value)	送信したバイト数	value: 送信する 1 バイトのデータ	スレーブデバイスがマスタからのリクエストに応じてデータを送信するときと、マスタがスレーブに送信するデータをキューに入れるときに使用します
Wire.available()	読み取り可能なバイト数	なし	read()で読み取ることができるバイト数を返します。
Wire.read()	受信データ	なし	マスタデバイスでは、requestFrom()を実行したあと、スレーブから送られてきたデータを読み取るときに使用します。

3. ソースコード概説

3.1. ソースコード構成

ORANGE-BASIC II のソースコードは次のようなフォルダー構成になっています。

<ORANGE-BASIC-II>	<src>	<basic>	Ast.c	抽象構文木の管理
			Ast.h	
			Basic.c	BASIC メイン処理
			Error.c	エラーメッセージ
			Error.h	
			Expr.c	式の解析
			Expr.h	
			Func.c	ファンクションの解析
			Func.h	
			Lexer.c	字句解析
			Lexer.h	
			Stmt.c	ステートメントの解析
			Stmt.h	
			Text.c	BASIC テキストの管理
			Text.h	
			Vars.c	変数の管理
			Vars.h	
		<extern>	Extern.cpp	追加ファンクションの実装(Arduino インターフェース)
			Extern.h	
		<iocs>	Fio.cpp	ファイル I/O
			Fio.h	
			Iocs.c	基本 I/O
			Iocs.h	
			Uart.cpp	シリアル通信
			Uart.h	
		<lib>	Lib.c	標準ライブラリーの個別実装
			Lib.h	
			Map.c	ハッシュマップ
			Map.h	
			Stack.c	スタック
			Stack.h	
		Basic.h		BASIC 処理系各種定数
		main.cpp		メイン処理
	platformio.ini	プロジェクトファイル		

3.2.　　コーディング規約

読みやすさに重点を置き、いくつかのルールを元に記述しています。各機能は基本的に一つのソースファイル(*.c)とヘッダーファイル(*.h)のペアで実現します。ヘッダーファイルだけを見ればインターフェースがわかるようになっていて、ソースファイルはその実装です。

ファイル名は先頭を英大文字にし、ペアのソースファイルとヘッダーファイルは拡張子以外は同名になります。　ヘッダーファイルには外部関数のプロトタイプと関連する定義のみを記述します。外部関数はファイル名_で始まります。ソースコード中にXclass_method()という関数を呼び出している部分があれば、その実装はXclass.cというファイルの中にあります。

ソースファイル中の内部関数はstaticを付けます。前方参照にならないように外部関数の前に記述します。(内部関数のプロトタイプはなるべく記述しないようにします。)

3.3.　　機種依存部

提供するORANGE-BASIC IIのソースコードはArduino Uno R4用ですが、他のCPUを搭載したArduinoやM5Stack、micro:bit、Wio Terminal、Raspberry Pi Picoなどでもリビルドすれば、ソースは無修正（あるいは、ほんの少しの変更）で動作します。

機種依存部分は<iocs>配下のソースにまとめられています。この部分を書き直せばArduino系以外の機種でも動作可能ですし、入出力をシリアル通信以外にも変更できます。

例として、M5Stack Cardputerへの移植方法を説明します。Cardputer単体で動作するように、入出力もシリアル通信ではなくやCardputer本体のLCDやキーボードに変更します。

① platform.iniをCardputer用に変更します。(*)
② <iocs>配下のファイルを修正します。(*)
(*)「M5Stack Cardputerシリアルコンソール化キット」にはCardTerm用の全ソースコードが付属しています。そちらのplatform.ini、<iocs>配下のファイルがそのまま利用できます。

platform.iniと<iocs>配下のファイルを置き換えてビルドするとCardputerの画面に文字は表示されますが、カーソルが表示されません。初期状態ではカーソル表示が無効になっているからです。main.cppの先頭部で#include "./iocs/gtext/Gtext.h"を追加し、setup()の最後にGtext_cursor(true);を追加してからビルドすれば、Cardputer単体で動作するようになります。

機種によっては、メモリーが不足して動作しない場合があります。その場合は、Basic.hの各種定数を修正してください。逆にCardputerではメモリーが十分にありますので、テキストエリアのサイズを増やすことができます。

定数名	デフォルト値	意味
MAX_TEXT_AREA	8192	テキストエリアの最大サイズ
MAX_FLOAT_VARS	20	単純変数の最大個数
MAX_ARRAY_VARS	5	配列変数の最大個数
MAX_AST	30	構文抽象木の最大ノード数
MAX_STRING_BUF	255	文字列のサイズ
MAX_REFERENCE	50	リナンバー用の最大参照個数

3.4. カスタマイズ部

ORANGE-BASIC II の言語処理本体は、<basic>配下のソースにまとめられています。この部分を改造すれば、コマンド（ステートメント、ファンクション）や構文を追加変更できます。しかし、この部分の理解なしでも、新しいファンクションを追加する方法があります。

<extern>配下の extern.c を変更するだけでファンクションを追加することができます。元々、このソースは Arduino の関数を使えるように設計しました。サンプルしていくつかの関数を実装しています。

たとえば、digitalWrite()という関数（すでに実装済み）を追加する場合は、extern.c を以下の手順で修正します。

① 関数名の登録

関数名と列挙定数を追加します。追加する位置はどこでも構いませんが、関数名(digitalwrite)は必ずすべて小文字にします。小文字で定義しておけば、BASIC のコマンドとして digitalWrite()のように大文字と小文字が混在したものも認識します。

```
static Keyword keywords[] = {
    {"digitalread", TOKEN_DIGITAL_READ},
    {"digitalwrite", TOKEN_DIGITAL_WRITE},
    {"pinmode", TOKEN_PIN_MODE},
    {"analogread", TOKEN_ANALOG_READ},
    {"analogwrite", TOKEN_ANALOG_WRITE},
    {"millis", TOKEN_MILLIS},
    {"micros", TOKEN_MICROS},
    {"delay", TOKEN_DELAY},
    {"wire.begin", TOKEN_WIRE_BEGIN},
    {"wire.begintransmission", TOKEN_WIRE_BEGIN_TRANSMISSION},
    {"wire.write", TOKEN_WIRE_WRITE},
    {"wire.endtransmission", TOKEN_WIRE_ENDTRANSMISSION},
    {"wire.requestFrom", TOKEN_WIRE_REQUEST_FROM},
    {"wire.available", TOKEN_WIRE_AVAILABLE},
    {"wire.read", TOKEN_WIRE_READ}};
```

```
enum {
    TOKEN_DIGITAL_READ = TOKEN_LEXER_END,
    TOKEN_DIGITAL_WRITE,
    TOKEN_PIN_MODE,
    TOKEN_ANALOG_READ,
    TOKEN_ANALOG_REFERENCE,
    TOKEN_ANALOG_WRITE,
    TOKEN_MILLIS,
    TOKEN_MICROS,
    TOKEN_DELAY,
    TOKEN_WIRE_BEGIN,
    TOKEN_WIRE_BEGIN_TRANSMISSION,
    TOKEN_WIRE_WRITE,
    TOKEN_WIRE_ENDTRANSMISSION,
    TOKEN_WIRE_REQUEST_FROM,
    TOKEN_WIRE_AVAILABLE,
    TOKEN_WIRE_READ
};
```

② ラッパー関数の追加

呼び出し先の関数のパラメーターに合わせて、ラッパー関数を作成します。作成したラッパー関数の中で本物の関数を呼ぶようにします。ただし、現状の ORANGE-BASIC II での変数はすべて単精度実数ですので、ラッパーの関数のパラメーターの型や関数の戻り値は float 型しか使えません。必要に応じでラッパー関数の中で型変換してください。

```
static float digitalWriteF0(float pin, float value) {
    digitalWrite(pin, value);
    return 0;
}
```

③ 分岐関数への登録

Extern_func()という分岐関数に、作成したラッパー関数を追加します。このときに、作成したラッパー関数を Func_p2Function()という関数でさらにラッピングします。


```
Node* Extern_func() {
    Node* node = NULL;
    switch (TOKEN->tokenValue) {
        case TOKEN_DIGITAL_READ:
            node = Func_p1Function(digitalReadF0);
            break;
        case TOKEN_DIGITAL_WRITE:
            node = Func_p2Function(digitalWriteF0);
            break;
        case TOKEN_PIN_MODE:
            node = Func_p2Function(pinModeF0);
            break;
        case TOKEN_ANALOG_READ:
            node = Func_p1Function(analogReadF0);
            break;
        case TOKEN_ANALOG_WRITE:
```

digitalWriteF0()という関数はパラメーターが二つありますので、Func_p2Function()でラッピングしました。パラメータの数によってラッピングする関数を決める必要があります。これらの関数のプロトタイプ宣言は basic/Func.h(実装は basic/Func.c)の中にあります。

関数プロトタイプ宣言	説明
Node* Func_p0Function(float (*f)());	パラメーター0 個の関数をラッピングします。
Node* Func_p1Function(float (*f)(float));	パラメーター1 個の関数をラッピングします。
Node* Func_p2Function(float (*f)(float, float));	パラメーター2 個の関数をラッピングします。
Node* Func_p01Function(float (*f0)(), float (*f1)(float));	Wire.endTransmission()関数のような、呼び出したい関数がオーバーロードされている場合に使用します。(この場合は、パラメーターが 0 個または 1 個)

ORANGE-BASIC II ユーザーズマニュアル

2024 年 4 月 30 日　1.0 版発行

著者　宇喜多裕一
発行　ピコソフト株式会
〒370-1201
群馬県高崎市倉賀野町 1803-6-104
http://www.picosoft.co.jp/