Programmable Calculator　プログラム検査書

・言語：Javascript

・動作環境：Internet Explorer 10 以降、Firefox 24 以降、Chrome 30 以降

・実行：index.html をブラウザーで開く動作確認

0. はじめに

その章の全ての演算及びソースが 1 行でかつ単純な場合、断りなしに Source を左側、Results を右側に表示している。

実際の Results には計算を実行した時刻が表示されるが、全ての実行結果において省略する。

また、断りのない場合、変数および関数の宣言は一切行っていないものとする。

同一の行であってもセミコロン';'を入力することでそれ以降の入力を次のコマンドとして扱う(これは関数内でも同様)。場合により、セミコロンを使用して 1 行に収めている部分もある。

1. 数値のみの基本的な演算

この電卓では、数値同士で四則演算を行う演算子として

+(足し算)　　-(引き算)　　*(掛け算)　　/(割り算)

を使用することができる。また、四則演算以外にも

%(剰余)　　^(累乗)　　!(階乗)

を使用しての計算、および先頭に負の数を用いた計算と、比較のための演算子として

==　　!=　　>=　　<=　　>　　<

の 6 つおよび条件のための演算子として && と || の 2 つの演算子を使用した true または false を返す演算が可能である。

＜実行結果＞

Source	Results
10+20	30
10-20	-10
10*20	200
10/20	0.5
10%20	10
10!	3628800
10^2	100
-10+20	10
-10*20	400
1==1	true
1==2	false
1!=2	true
1!=1	false
2>=1	true

1>=1	true
0>=1	false
0<=1	true
1<=1	true
2<=1	false
2>1	true
1>1	false
0<1	true
1<1	false
1==1 && 3>2	true
1==1 && 2>2	false
1==2 && 3>2	false
1==2 && 2>2	false
1==1 || 3>2	true
1==1 || 2>2	true
1==2 || 3>2	true
1==2 || 2>2	false

2. 計算の順序と括弧を用いた演算

式中に一切括弧を用いない場合、以下の順に優先してその部分の計算が行われる。

同系列のものは左から順に計算される。

① %(剰余)、^(累乗)、!(階乗)

② *(掛け算)、/(割り算)

③ +(足し算)、-(引き算)

また、計算中の括弧は小括弧'('')'のみ使用が許されている(無限に使用できるが、当然括弧'('と括弧閉じ')'の個数は同数であることが条件である)。

括弧を用いた場合、剰余・累乗・階乗の計算に先立って計算が行われる。

＜実行結果＞

Source	Results
1+3*4%5^2	49
(1+3)*4%5^2	64
1+(3*4%5)^2	145
1+3*4%(5^2)	13

3. 定義済み定数の使用

計算時に E, LN2, LN10, LOG2E, LOG10E, PI, SQRT2, SQRT1_2, EPSILON, MAX_VALUE, MIN_VALUE, NEGATIVE_INFINITY, POSITIVE_INFINITY を定数として使用できる。各定数の説明は省略する。

＜実行結果＞

Source	Results
E	2.718281828459045
LN2	0.6931471805599453
LN10	2.302585092994046
LOG2E	1.4426950408889634
LOG10E	0.4342944819032518
PI	3.141592653589793
SQRT2	1.4142135623730951
SQRT1_2	0.7071067811865476
EPSILON	2.220446049250313e-16
MAX_VALUE	1.7976931348623157e+308
MIN_VALUE	5e-324
NEGATIVE_INFINITY	-Infinity
POSITIVE_INFINITY	Infinity

4. 変数の定義・代入・参照

”アルファベットで始まる１文字以上の文字列”=式

という形で入力することでその文字列を変数として宣言できる。変数は式中で通常の数値と同様に扱うことができる。また、変数は何度でも再宣言が可能なほか、Variable Table 欄にて値の編集が可能である。

＜実行結果＞

Source	Results
a=10+20; a;	30
a=10+20; b=30; a+b;	60
a=10+20; b=30; a=a+b; a+b;	90

5. 関数の定義・呼び出し

function “アルファベットで始まる１文字以上の文字列” (引数) { 処理 }

といった形で記述することで関数を定義できる(引数はなくてもよい)。関数を式中で呼び出すには

call “関数名”(与える引数)

と記述する。もし関数の結果を計算式中で使用したいのであれば呼び出し式全体を小括弧'('')'でくくればよい。

関数は何度でも再宣言が可能なほか、Function Table 欄にて処理の編集が可能である。

＜実行結果＞

Source

```
function add(a,b){ return a+b; }
call add 5 3;
```

Results

```
    8

Source
    function add(){ return a+b; }
    a=5; b=3; call add;
Results
    8

Source
    function add(){ return a+b; }
    a=5; b=3; (call add)*3;
Results
    24
```

6. スコープ機能

関数内で使用している変数とコマンド全体内で使用している変数で競合が発生した場合でも使用者の想定通りに動作するよう、デフォルトでスコープ機能はオンになっている。もし使用者が敢えて変数を競合させたい場合は先頭行に

```
    /*! disable scope */
```

と入力することでスコープ機能をオフにすることができる。デフォルトでオンなので宣言の必要はないが、もしスコープ機能がオンであると明言する場合は先頭行に

```
    /*! enable scope */
```

と入力すればよい。

＜実行結果＞

```
Source
    /*! enable scope */
    function add(a,b){ return a+b; }
    a=0; (call add 5 3)+a;
Results
    8

Source
    /*! disable scope */
    function add(a,b){ return a+b; }
    a=0; (call add 5 3)+a;
Results
    13
```

7. If 文

関数および while 文(次章にて記述)の内外を問わず、

if (分岐条件) { 処理 } (使用するのであれば else{ 処理 })

と記述することで通常のプログラムと同様の条件分岐が可能である。分岐条件には比較演算子を利用した、true または false を返す記述式が必須である。

なお、else if は使用できないので

if (分岐条件) { 処理 } else{ if (分岐条件) { 処理 } }

などと記述する必要がある。

＜実行結果＞

```
Source
    a=5;
    if(a%2!=0){ a=a*2; }
    a;
Results
    10

Source
    a=6;
    if(a%2!=0){ a=a*2; }
    a;
Results
    6

Source
    a=6;
    if(a%2!=0){ a=a*2; }
    else{ a=a-1; }
    a;
Results
    5

Source
    function ex (a) {
      if(a%2!=0){ a=a*2; }
      else{ a=a-1; }
    }
    call ex 10;
Results
```

```
9
```

8. While 文

関数および if 文の内外を問わず

```
While ( ループの続行条件 ) { 処理 }
```

と記述することで通常のプログラムと同様のループが可能である。

ループの続行条件には if 文の分岐条件と同様比較演算子を利用した true または false を返す記述式が必須である。また、break は使用できない。

＜実行結果＞

Source

```
a=10 i=a;
while(i>0){
  i=i-1;
  a=a+i;
}
a;
```

Results

```
55
```

Source

```
a=-1 i=a;
while(i>0){
  i=i-1;
  a=a+i;
}
```

a;Results

```
-1
```

Source

```
function ex (a) {
  i=a;
  while(i>0) {
    i=i-1;
    a=a+i;
  }
}
call ex 10;
```

Results

9. 定義済み関数の使用

計算時に

abs ,acos , acosh ,asin ,asinh ,atan ,atanh ,atan2 ,cbrt ,ceil ,clz32
cos ,cosh ,exp , expm1 ,floor ,fround ,hypot ,imul ,log ,log1p ,log10 ,log2
min ,max ,pow ,random ,round ,sign ,sin ,sinh ,sqrt ,tan ,tanh ,trunc

を定数として使用できる。各定数の説明は省略する。

＜実行結果＞

Source	Results
call abs -1;	1
call acos 0;	1.5707963267948966
call acosh 2;	1.3169578969248166
call asin 1;	1.5707963267948966
call asinh 2;	1.4436354751788103
call atan 1;	0.7853981633974483
call atanh 0.5;	0.5493061443340549
call atan2 1 1;	0.7853981633974483
call cbrt -8;	-2
call ceil 0.5;	1
call clz32 128;	24
call cos PI/3;	0.5
call cosh 2;	3.762195691083631
call exp 2;	7.3890560989306495
call expm1 2;	6.3890560989306495
call floor 0.5;	0
call fround 0.4;	0.4000000059604645
call hypot 1 1 1 2 2;	3.3166247903554
call imul 2^30 1.5;	1073741824
call log E^2;	2
call log1p E^2-1;	2
call log10 100;	2
call log2 16;	4
call min 0 1 2;	0
call max 0 1 2;	2
call pow 2 1.5;	2.8284271247461903
call round 1.5;	2
call sign -5;	-1

call sin PI/6;	0.5
call sinh 2;	3.6268604078470186
call sqrt 4;	2
call tan PI/4;	1
call tanh 2;	0.964027580075817
call trunc 1.4;	1

10. コメント

関数・if 文・while 文の内外を問わず、コメントアウトしたい部分の両端を'/*''*/'でくくるとその部分をコメントアウトすることができる。

＜実行結果＞

Source	Results
a=1; /*a=2;*/ a;	1

11. 構文エラー

構文エラーは入力と同時に Parse Result 部に出る。ここでは構文解析時に予想外の表現が記述された場合、Parse Result 部に

Error: Parse error on line（間違いのある最初の行):
（間違っている部分の指摘）
Excepting（本来来るべき表現）, got（その部分で読んだ文字）

と表示される。

・行の先頭に数字・英字・負の符号'-'・if・while・function・call・return・コメント以外が記述されている場合

＜実行結果＞

```
Source
    %2
Parse Result
    Error: Parse error on line 1:
    %2
    ^
    Expecting 'EOF', 'COMMENT', 'FUNCTION', 'IDENTIFIER', '(', 'CALL', 'IF', 'WHILE',
    'RETURN', '-', 'NUMBER', got '%'
```

・行の先頭の'-'の後に数字・英字・負の符号'-'・括弧'('・call 以外が記述されている場合

＜実行結果＞

```
Source
    3+
Parse Result
```

```
    Error: Parse error on line 1:
    -/**/
    _^
    Expecting 'IDENTIFIER', '(', 'CALL', '-', 'NUMBER', got 'COMMENT'
```

・数字の後に数字・括弧'('')'・中括弧閉じ'}'・演算記号・if・while・function・call・return・コメント以外が記述されている場合

＜実行結果＞

```
Source
    3=4;
Parse Result
    Error: Parse error on line 1:
    3=4;
    _^
    Expecting 'EOF', 'COMMENT', ';', 'FUNCTION', 'IDENTIFIER', '(', ')', '}', 'CALL', 'IF',
    'WHILE','RETURN','RELATION','MAG_RELATION','-','+','*','/','%','^','!','NUMBER',
    got '='
```

・演算記号の後に数字・英字・負の符号'-'・括弧'('・call 以外が記述されている場合

＜実行結果＞

```
Source
    3+
Parse Result
    Error: Parse error on line 1:
    3+%
    __^
    Expecting 'IDENTIFIER', '(', 'CALL', '-', 'NUMBER', got '%'
```

・括弧'('を用いた式で括弧閉じ')'が不足している場合

＜実行結果＞

```
Source
    ((3+3)
Parse Result
    Error: Parse error on line 1:
    ((3+3)
    ______^
    Expecting ')', 'MAG_RELATION', '-', '+', '*', '/', '%', '^', '!', got 'EOF'
```

・完成している式の後に数字・英字・負の符号'-'・括弧'('・中括弧閉じ'}'・演算記号・if・while・function・return・コメント以外が記述されている場合

＜実行結果＞

```
Source
    3+2)
Parse Result
    Error: Parse error on line 1:
    3+2)
    ___^
    Expecting 'EOF', 'COMMENT', ';', 'FUNCTION', 'IDENTIFIER', '(', '}', 'CALL', 'IF',
    'WHILE', 'RETURN', '-', 'NUMBER', got ')'
```

・return の後に数字・英字・負の符号'-'・括弧'('・call・以外が記述されている場合

＜実行結果＞

```
Source
    return %
Parse Result
    Error: Parse error on line 1:
    return %
    _______^
    Expecting 'IDENTIFIER', '(', 'CALL', '-', 'NUMBER', got '%'
```

・function または call の後に英字以外が記述されている場合

＜実行結果＞

```
Source
    function +
Parse Result
    Error: Parse error on line 1:
    function +
    _________^
    Expecting 'IDENTIFIER', got '+'
```

・if または while の後に条件の記述を示すための括弧'('以外が記述されている場合

＜実行結果＞

```
Source
    if-
Parse Result
    Error: Parse error on line 1:
```

```
    if-
    __^
    Expecting '(', got '-'
```

・if または while の条件記述内に true または false 以外を返す式が記述されている場合

＜実行結果＞

```
Source
    if(1){ }
Parse Result
    Error: Parse error on line 1:
    if(1){ }
    ____^
    Expecting 'MAG_RELATION', '-', '+', '*', '/', '%', '^', '!', got ')'
```

・if または while の条件文の後に中括弧'{'以外が記述されている場合

＜実行結果＞

```
Source
    if(1>0)+
Parse Result
    Error: Parse error on line 1:
    if(1>0)+
    _______^
    Expecting '{', got '+'
```

・if または while の処理の後に中括弧閉じ'}'を記述しなかった場合

＜実行結果＞

```
Source
    if(1>0){ a=1
Parse Result
    Error: Parse error on line 1:
    if(1>0){ a=1
    ______________^
Expecting 'COMMENT', 'FUNCTION', 'IDENTIFIER', '(', '}', 'CALL', 'IF', 'WHILE', 'RETURN', '-',
'NUMBER', got 'EOF'
```

・if または while の処理内に何も記述しなかった場合

＜実行結果＞

```
Source
```

```
    if(1>0){ }
Parse Result
    Error: Parse error on line 1:
    if(1>0){ }
    ---------^
    Expecting 'COMMENT', 'FUNCTION', 'IDENTIFIER', '(', 'CALL', 'IF', 'WHILE', 'RETURN', '-',
    'NUMBER', got '}'
```

・コメントアウト開始の記号'/*'の後にコメントアウト終了の記号'*/'を記述しなかった場合

＜実行結果＞

```
Source
    /* 3
Parse Result
    Error: Parse error on line 1:
    /* 3
    ^
    Expecting 'EOF', 'COMMENT', 'FUNCTION', 'IDENTIFIER', '(', 'CALL', 'IF', 'WHILE',
    'RETURN', '-', 'NUMBER', got '/'
```

12. 実行時エラー

・存在しない変数を呼び出した場合、その変数は定義されていないというエラーを返す。

＜実行結果＞

```
Source
    ex;
Result
    Line 1: Variable ex is undefined.

    Code
    > ex

    Stacktrace (Limited only top 5)
```

・存在しない関数を呼び出した場合、その関数は定義されていないというエラーを返す。

＜実行結果＞

```
Source
    call ex;
Result
    Line 1: Function ex is undefined.
```

```
    Code
    > call ex

    Stacktrace (Limited only top 5)
```

・予め定義されている定数に別の値を代入しようとした場合、その定数は変更できないというエラーを返す。
＜実行結果＞

```
Source
    PI=2;
Result
    Line 1: Const PI is read-only.

    Code
    > PI = 2;

    Stacktrace (Limited only top 5)
```

・ゼロで割ろうとした場合、ゼロで割っているというエラーを返す。
＜実行結果＞

```
Source
    1/0;
Result
    Line 1: Division by zero.

    Code
    > (1 / 0)

    Stacktrace (Limited only top 5)
```

・ゼロで割ろうとした場合、ゼロで割っているというエラーを返す。
＜実行結果＞

```
Source
    1/0;
Result
    Line 1: Division by zero.
```

```
    Code
    > (1 / 0)

    Stacktrace (Limited only top 5)
```

・関数呼び出し時、引数が指定された数より少ない場合は変数が少なすぎるというエラーを返す。
＜実行結果＞

```
Source
    function add(a,b){ return a+b; }
    call add 5;
Result
    Line 2: Not enough arguments to add.

    Code
    > call add 5

    Stacktrace (Limited only top 5)
```

・関数呼び出し時、引数が指定された数より多い場合は変数が多すぎるというエラーを返す。
＜実行結果＞

```
Source
    function add(a,b){ return a+b; }
    call add 5 3 1;
Result
    Line 2: Too many arguments to add.

    Code
    > call add 5 3 1

    Stacktrace (Limited only top 5)
```

・スコープ使用時、スコープ外からスコープ内の変数を呼び出した場合、その変数は定義されていないというエラーを返す。
＜実行結果＞

```
Source
    /*! enable scope */
    function add(a,b){ return a+b; }
    call add 5 3; a;
```

Result

Line 3: Variable a is undefined.

Code

> a

Stacktrace (Limited only top 5)

・整数が条件の階乗演算子'!'で整数以外を計算しようとすると(スタックが原因で)計算出来ないというエラーを返す。

＜実行結果＞

Source

1.5!

Result

Line 3: Maximum call stack size exceeded

Code

> (1.5!)

Stacktrace (Limited only top 5)

13. 仕様

以下の場合、エラーは出ないが計算とは無関係な数値を返す。

・ある数の0で割った余りを求めようとする - ある数そのものを返す

・0以下の階乗 - 1を返す

・整数以外で割った余りを求めようとする - 商が整数になるときの正の最小の(割られる数－割る数×商)が返される

＜実行結果＞

Source	Results
2%0	2
-1!	1
2%0.3	0.20000000000000007

予め定義されている関数と同じ名前の関数をユーザーが定義しても、すでに定義されている関数の方が呼び出される。

＜実行結果＞

Source

call abs;

```
Result
    NaN
```
（上の実行結果は 1 つの引数を渡され、引数の絶対値を戻り値として返す定義済関数 abs に引数を渡さなかった場合の挙動を確認するためのものである。）

```
Source
    function abs(){
     return 0;
    }
    call abs;
Result
    NaN

Source
    Source
    function abs(){
     return 0;
    }
    call abs 1;
Result
    1

Source
    function abs(x){
     return 0;
    }
    call abs 1;
Result
    1

Source
    function abs(x){
     return 0;
    }
    call abs;
Result
    NaN
```

以上をもって実装内容の検証とする。