pascal で作ったサイクロイド曲線やトロコイド曲線や包絡線計算は回転変換が多数入るのでjavaScriptで直接作るのは大変です。そこで慣れたpascal上で描画を見ながら符号の間違い等を修正しつつ作りました。しかし､問題はpascalからJavaScriptに変換する作業。サイクロイド曲線の場合はそう大きくもなかったのでなんとか変換したのですが、トロコイド曲線の場合は行数も多く作業量は大きい。どうせ作業するなら、その作業をするプログラムを作ってしまえと作ったものです。

pascal	javaScript
begin	{
end	}
:=	=
=	==
<>	!=

こういう単純置換といっても = とか置換の順番を考えええると厄介でしょ？他にも変数の型定義を外すとかいろいろと手間な事があるのです。あと私の場合、Pascal を使っていると大文字小文字を区別しないので定義時の大文字小文字に使ってる変数名を修正する機能も必要でした

■変換

pascal	JavaScript	説明
= <>	== !=	比較
:=	=	代入
and or xor	& && | || ^	ビットまたは論理
not	~ !	否定 適当に処理されているので確認は必要
SHL	<<	シフト
DIV MOD	/ %	DIV 利用時は注意の事
{} (**)	/**/	コメント
$ & %	0x 0o 0b	16 進数等
#xnn ""	\xnn \'	文字エスケープ
begin end	{ }	ブロック
if then else	if() else	条件
for to do	for(;;)	繰り返し
while do	while()	繰り返し
repeat until	do{}while(!)	until 条件に！を付ける
case of end	switch(){}	複数分岐 continue/break に注意
exit	return Result	Result を無理やり付ける

with do	with()	JavaScript 側で非推奨なので使わない事
sin pi	Math.sin Math.PI	Math にありそうな関数は置換される
self	this	単なる置き換えで機能が違うので注意
inc(a)inc(a,n)	++a a+=n	inc/dec
length(a)	a.length	SetLength(a.b) も a.length=a
raise	throw	単なる置換 自前で処理する事
try except on	try catch(n){if	十分ではないので自前で処理
try finally	try finally	--
type a=()	const a={}	オブジェクト順番に数値を割り当てる
type a=set of()	const a={}	全部に false を割り当てる
a=record	function a(){return this;}	コンストラクタに置換
メソッド	T.prototype.a	メソッド実装部だけ変換
クラス変数 a	this.a	record/class 内で定義しておけば認識する
関数内変数	var 部は消去 let 付	関数内関数の場合 var にする場合あり
{ } (* *)	/* */ /* */	コメント内に /* */ があれば意図しない結果になるので事前に検査の事
{.?} (*.?*)	置換タグ	4 文字または 6 文字のこのコメントは特殊な動作をするので注意

■対応していない事

1.Type 変換先が判らない record 型のみ対応してるので type は必要 (コメントアウトされる)
2.@ ポインタや参照演算子 ^ の類 自前で置換タグで処理しておく事
3. 関数内関数 安定動作の為にそれらしくは処理はしてるが
4.forward JavaScript は巻上げがされるので不要なので {.J} 〜 {} で削除すればよいから
5.for 文の変数には毎回 let を付けてしまう 等 let 付けの怪しさ
6. 型定義を外すだけなので　配列なら var arry; のままですから var arry=[]; のように修正が必要
7.strToFloat 等 {.J}strToFloat{Number} と整数と少数点のどちらかにしたい場合もあるので
8.in 演算子　ちょっと JavaScript の in 演算子は使いにくい しそもそも列挙型がない
9.class 定義 コンストラクタに置換するが十分ではない
10.with do　JavaScript 側が推奨していないため
11.case 文内の break(switch 文に変換すると break は switch から抜ける意味に変わる)

■やってくれる事

1.exit は java にはない pascal は関数の終了なので return か　return Result に置換
2. 関数の先頭で let Result; 最後に return Result 追加
3.' 文字列 '#13 のような文字列を " 文字列 \r" と置換
4.inc(a) を ++a に置換 inc(a,3) を a+=3 に置換
5.setlength(a,3) を a.length=3 に置換 length(a) を a.length に置換
6.pi を Math.PI 他 Math にありそうな関数は Math. を付け小文字にする
7.ln() を Math.log() に置換　(他にも名前が違う場合もあるだろうけど調査不足)
8.sqr(x) を 2 乗 sqr(x) に置換するので自分で 2 乗に修正すること例 (x)**2 のように
9. 変数名を最初に使った名前に大文字小文字を固定する
10.$abc のような数を 0xabc と置換
11.if while for 文を置換
12.Repeat 〜 Until 式 ; do{ 〜 }while(! 式);
13.case 式 of 〜 end;　　switch(){ case ラベル : 〜 default:break;};
14.procedure を function に置換
15.var const の型名を削除
16. 関数内の var 定義を消して 利用時に let を付けまくる（無駄な分は自分で修正して下さい)
17.AND OR XOR NOT 等の & | ^ 置換（ビット演算子としてるので論理型なら自分で修正)
18.:= を = = を == <> を != (＝＝＝等 使うべきなら自分で)
19. コメント {〜} を /* 〜 */ に置換
20.ord('+') を /*+*/0x2B のように数値に置換
21.ord(式) を定数でないなら 式 .codePointAt(0) に置換 (pascal は 1 から始まるのでややこしい場面では自分で {.J} タグで)
22.intToStr/floatToStr を string に置換 strToInt 等は {.J} タグで自前処理する事
23.record 型を function 型名 (){ this. メンバー ... return this;}; と置換
24. メソッド定義を クラス名 .prototype. 関数名 =function と置換
25. メソッド内の record 型メンバー変数に this. を付ける
26. 置換タグ {.J} {.o} {.r} {.E} の処理

■置換タグ

置換タグ	置換後	説明
{.J}hoge{bom}	bom	最初に見つけた {} コメント内と入れ変える .hoge も bom も何も解釈しない
(*.J*)hoge(*bom*)	bom	上と同じ 出力側に {} を含む時利用する
{.J}hoge{}	--	JavaScript には出力しない
{.J}{hoge}	hoge	JavaScript にだけ出力する
{.J} 〜 //{hoge}a	hogea	コメントアウト内の {} でもヒットするがコメントアウト機能が消える
{.a}lab	Object.assign({},lab)	lab は解釈されないので this. 等を付けるのは自分で
{.o}foo(hoge,	hoge.foo(	オブジェクト用で 第一引数のメソッドにする

{.r}'hoge'	hoge	正規表現用でブラケットを外す ("" には対応していない \x27 を使う事)
{.E} 〜 {.e}	--	出力禁止 {.e} が無ければ終端まで //{.e} のようにコメント内でもヒットする
{.V}	--	以後 let const を var として出力する
{.v}	--	直後のローカル変数定義を var で出力する
{.L}	--	直後の not and or を ! && ||
{.B}	--	直後の not and or を ^ & |
{a*/b/*c}	/*a*/b/*c*/	b が有効になる。/* */ がコメント内に含まれていないか確認しておく事

・置換タグは (**) スタイルでも有効になる
　・Delphi の (**) コメントは (* } *) のように {} を挟めるので {} を含む置換に有効
・置換タグはコメントアウトされていたら無効になる
　・{.J} 〜 //{hoge}a はコメント内でも hoge に置換されるがコメントアウト機能も無効になる
　・{.e} はコメント内でもヒットするがそのタグの右側から有効になるので注意
・{.a}{.o}{.r} は直後の文字列に有効になる。空白を挟むと失敗する

自前で事前置換設定

javaScript 側を手作業で修正してしまうと、元の pascal コードにミスがあれば両方を修正する事になる。だから pascal 側で修正可能なように考えておく。

pascal	第一候補	第 2 候補	
strToFloat(s)	Number(s)	parseFloat(s)	{.J}strToFloat{Number}(s)
strToInt(s)	parseInt(s,10)	Number(s)	
Exception.Create(s)	(s)	--	raise は throw に置換されるので場合にっては文字列型で

・置換タグで代入分全体を置換すると let 等の付けるタイミングをミスる右辺だけにする事
・置換タグの置き換え先に PI や sin 等が入っているなら Math. を自前で付ける必要がある

■正規表現

→ z 変換で周波数特性を得る javascript 作成奮戦記 に replace の類似品がある

```
{.o}replace(Zs, {.r}'/[  |\s]+/g', '')
 と pascal 側ですれば
Zs.replace( /[  |\s]+/g, "")
```

と置換される

■ a in s

・ 文字が含まれているかなら s を文字列にして {.J}a in s{s.includes(s)}
・ 数値が含まれているかなら s を配列にして {.J}a in s{s.includes(a)}
・ 条件設定なら s = set of(a,b,c...) を定義し {.J}a in s{s.a}

includes は古い JavaScript のバージョンでは使えない (WSH のように) ので自前で似た関数を作る必要があります
WSH の Jscript には include はないけど indexOf はあるので
if(!String.prototype.includes) String.prototype.includes=function(s){return this.indexOf(s) !==-1;};
配列の場合は WSH に indexOf も無いようなので
if(!Array.prototype.includes)Array.prototype.includes=function(s){
for(var c in this)if(c==s)return true;return false;};

■関数内関数

・メソッドにすると this が大量に使われるので見辛い
・そこで関数内関数を使う事になる。外側で定義した変数は内側の関数から参照出来る
・しかしメソッドの関数内関数には注意が必要
  ・Pascal と違い内側の関数から this(self) は参照出来ない。
  ・よって pascal 側で外側で self を別変数に保存し、関数内関数ではそちらを参照する必要がある
  ・クラスメンバの参照も同じで、関数内関数ではオブジェクトメンバーを直接参照は出来ない
  ・出来るだけローカル変数を使いトップのメソッドでローカル変数をメンバーに代入する

■ case **文中の** break

JavaScript には goto が無いので流れの制御には continue/break を使うしかないが､外のループを case 文中から break で抜けるコードを書いていると、switch 文に変換された時に望みの動作にならない。switch 文中では break は switch 文から抜ける意味になる。

if 文に変換しておくか 関数にして exit で抜けるか そのままのスタイルにしたいならある程度の変換はされる。

repeat 〜 case c of 1:break; 〜 であれば
L1:repeat 〜 case c of 1:break L1; 〜
のようにブレークにラベルが付く。 pascal 側と同一の動作になるかは正直分からない。

■オブジェクトの利用

type TA=record 〜 end コンストラクタ TA=function(){ 〜 return this;} に置換されているので
var rec:TA: の変数を使う時は
{.J}{rec = new TA();} を関数の begin 後に入れる必要がある
class 定義もある程度は処理するが難しい
pascal 側は record 型で

■ @ ^ ポインタや参照を利用した場合

・Delphi モードならメンバ参照 p^.hoge は p.hoge と書けるので、そのように置換する
・p:=@s は @ を消去しても JavaScript 側はそのように動く (すべてのオブジェクトは参照でしかない)
・p^:=s や s:=^p は ^ を消去してもほぼ問題はない　問題は大抵その前にある

```
save:=s; s:=data; 処理 ;s:=save; とある場合　オブジェクトの場合は s は元に戻らない
save:=s; s:={.a}data; 処理 ;s:=save; と {.a} を使えば
save=s;s=Object.assign({},data); のように置換される
```

■ AND OR NOT

・これらは & | ~ か && || NOT に変換される
・javaScript は　a & b と記述しても a と b が論理値なら 0 1 に変換されるので多くの場合問題ない
・逆に論理値が求められる時に 数値の 0 以外は true 0 は false に扱われる また null undefined 空文字列等も false になる
・よって a&b の a,b 双方が論理値で if 文のような論理値が必要な場合には問題なく動作する
・c 言語なら論理演算なら && にするべきなようだが JavaScript は特殊な動作をするので注意。

```
javaScript は　a && b　とした時 a が false に変換される時 false が帰るのではなく a が帰る
両方 ( 複数列挙していれば全部が )true に変換可能なら b が帰る
a||b も同じで a が false に変換可能なら無条件で b となり そうでなければ b は評価もされない
よって b が関数呼び出しの時に pascal で {$B -} のように b は呼び出されない
```

・つまり && || に変換すべきなのは　このショートカット効果を積極的に利用した場合となる
・問題は NOT で ~ の場合 論理値 true false は 0,1 と解釈され反転するので -1 と -2 になり論理値としては true となる
　・if while until で先頭に NOT があれば ! に置換するので大抵は大丈夫だろう

■ High

・High(a) は length(a)-1 と置換しておけばとりあえずは動く
・しかし for in 構文 に変換する事を考えた方がいい

```
for i:=0 to High(a) do a[i]:=0;
for(let i in a) a[i]=0;
注意 :for of の方が良いそうです（a には後から配列以外も追加可能な為 )
```

■ with

・一応変換はするが、JavaScript 側が推奨していないため pascal 側で with を使わないように修正したのでテストしていない。

■ format

・JavaScript に該当するものがない
・置換タグで自前で作成するしかない

形式	JavaScript	--
%d	N.toString()	10 進数
%x	N.toString(16)	16 進数 (桁数指定ではない)
%10.2f	N.toFixed(2).padStart(10	' ')) . 小数点以下 2 桁で 10 文字分

文字数指定するなら toString 等で文字列にしてから .padStart(桁数 ,' ') を追加

■ Canvas

差が大きすぎるので描画部は JavaScript 側で作成すべきかもしれない

VCL	Java2d	--
c.lineto moveto	lineTo moveTo (同名だが即座に描画はしない)	
c.FillRect	c.fillRect (3,4 番目の引数は差分に置換する必要がある)	
c.Rectangle	c.strokeRect(3,4 番目の引数は差分に置換する必要がある)	
c.roundRect	4 番目の引数は差分 5 番目で半径) 即座に描画しない "	
c.TextOut(x,y,s)	c.fillText(s,x,y)	
c.TextWidth(s)	c.measureText(s).width	
c.TextHeight(s)	with(c.measureText(s))actualBoundingBoxAscent +actualBoundingBoxDescen	
c.Pen.Color	c.strokeStyle (色の設定方法の違いに注意)	
c.Brush.Color	c.fillStyle //	

```
 roundRect 等は一部のブラウザで対応していない
 moveTo lineTo  等を使っている場合は
 {.J}{cv.beginPath();} // としてから 描画ルーチンを実行し
 {.J}{cv.stroke();}    // で描画する
色の指定は
 {.J}{
 const   clBlack   = "#000000";
 const   clMaroon  = "#000080";
 const   clGreen   = "#008000";
 const   clOlive   = "#008080";
 const   clNavy    = "#800000";
 const   clPurple  = "#800080";
 const   clTeal    = "#808000";
 const   clGray    = "#808080";
 const   clSilver  = "#C0C0C0";
 const   clRed     = "#0000FF";
 const   clLime    = "#00FF00";
 const   clYellow  = "#00FFFF";
 const   clBlue    = "#FF0000";
 const   clFuchsia = "#FF00FF";
 const   clAqua    = "#FFFF00";
 const   clWhite   = "#FFFFFF"; }
文字サイズの入手は
   {.J}{const ts=cv.measureText(s);}
   w := {.J}cv.TextWidth(s) {ts.width};
   h := {.J}cv.TextHeight(s){ts.actualBoundingBoxAscent +ts.actualBoundingBoxDescent};
```

pas2js.exe

・Lazarus に付いているので試してはみたのです。
・良く出来ていて Pascal コードがそのまま Web 上で JavaScript として動きます
・でも、いろんな問題を力技で解決してるからか追加コードが多いのと、一部の関数だけ使うには向かない感じ

■作成開始時の方針

・ローカル変数は var 行ごと t 定義部を消去する。代わりに変数名を覚えておいて大文字小文字のゆらぎを吸収する
・ローカル変数は最初に代入する時に let を付ける　ネストの深さが戻ったらまた let を付け

る
・このために、ループ内で代入した値を後で使うような場合は問題が起きます。pascal 側でループ外で 0 でも代入しておきましょう。
・グローバル変数には宣言時に var を付ける
・宣言してないラベルはグローバル変数と同様に登録して最初に使った大文字小文字に合わせます
・const 宣言は宣言時に const を付ける
・case 文とか自分が使っていない機能は後回し
・ポインタとか使ってるコードは考えない。Pascal 側で使わないように修正してから変換すればいいや
・クラスとかメソッドとかも　JavaScript にどう変換して良いかもわからないから Pascal 側で変更しよう
・文字列・数字・コメント類は追いかけて変換対象にならないようにしよう
・エディタの置換だと、このあたりの事が出来ないから

■手に入る場所

・[ClipBd 電卓] のページにある ClipBd 電卓 src.zip に Pas2JavaSc.pas があります。

ClipBd 電卓内 CalcAnsiScript で動くようにしてるのですが uAnsiCharTools.pas があれば CalcAnsiScript を uses から外し TPas2JavaScCalc を削除すれば単独で使えるでしょう。

・ClipBd 電卓 .exe はウイルスチェックとかでダウンロードも出来ないかもしれませんがダウンロードしたなら
・#P2Js を変換したいコードの行頭に挿入して選択してコピーし ClipBd 電卓 .exe を実行した後　ペーストすれば変換されています

```
{
#P2Js }
のように２行挿入しておけば挿入したままでも問題ないでしょう
```

・自分のツールに組み込みたいならどうぞご利用下さい。
・問題は Ansi 文字列のポインタ渡しである点でしょう。
・クリップボード経由なら文字列は適時変換されてくるので問題なさそうですが自前で Ansi だと面倒かもしれません。
・uAnsiCharTools の UTF8 版も作っているので必要な人はコメントにでも書いて下さい

record 型への対応 - 裏目小僧 (2023 年 03 月 12 日 07 時 57 分 16 秒)

```
type  TnumOP = record dt: double;fn: string;op: DWORD; end;
を
function TnumOP(){this.dt= 0 /*double*/;this.fn= "" /*string*/;this.op= DWORD;  return this;};
と置換
(a,b,c) は  {a:0,b:1,c:2} に置換
set of (a,b,c) は  {a:false,b:false,c:false} に置換
set of ラベルは対応しない
```

・置換用のタグを追加
・debug 出力を追加
・ #P2Js debug で有効になる
・ 有効時 /*< 処理名 , スタック深さ >*/ が出力に含まれるようになる

■ ClipBd 電卓 ファイルモード

・ファイル名を引数として ClipBd 電卓を起動した時
・ファイルモードでは #P2Js の 5 文字がファイル中にある場合に処理し、次の行より処理を

始める
・ファイルモードでは {#P2Js} や //#P2Js のように行頭から始まらなくてもよいが大文字小文字は区別する