

流れ図ソフトのシステム構造

太田 幸一

要旨

前回、論文「アルゴリズム学習のためのアプリケーションソフト」において、本学の講義「アルゴリズム論」で活用している流れ図のアプリケーションソフトを紹介した。

今回は、その独自に考案した流れ図ソフトのシステム構造を解説する。この流れ図ソフトのシステムは、流れ図によってアルゴリズムを表現・実行するもので、本論文では、その構造を利用することで、アルゴリズム学習の容易さを提示するものである。

本システムを広く利用することで、分かりやすく使用しやすい流れ図のプログラムを実現し、視覚的なコントロール手段と成り得るのである。

キーワード：アルゴリズム、流れ図、プログラム言語、情報処理教育

1. はじめに

アルゴリズムの授業では、流れ図によるアルゴリズム学習のためのアプリケーションソフトを使用している。そして、そのソフトに加えて、そのソフトのシステムを利用して考案・作成したパズルゲームを、授業の初期段階でアルゴリズム学習者に提供し、好評を得ている。

パズルゲームに関しては次回に譲り、本論文では、流れ図によるアルゴリズム学習のためのアプリケーションソフトの構造を解説する。

2. 流れ図の基本構造

図1の $3n+1$ 問題の流れ図を例に、流れ図のアプリケーションソフトでの流れ図の構造について述べていく。

$3n+1$ 問題は、

「任意の0でない自然数の値を n とし、

・ n の値が偶数の場合 ($n\%2=0$)、 n を2で割った値を、新たな n の値にする ($n/=2$)

・ n の値が奇数の場合 ($n\%2!=0$)、 n に3をかけて1を足した値を、新たな n の値にする ($n=3*n+1$)

という操作を繰り返していくと、有限回で n の値が1となる」

という有名な問題である。

図1 3n+1 問題

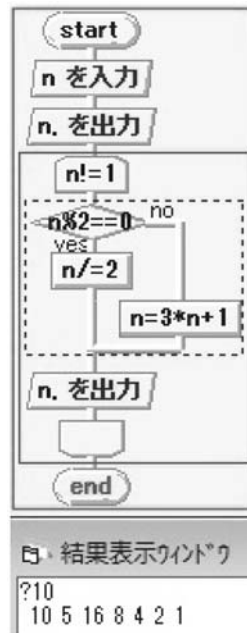


図1の流れ図の実行では、まず0でない自然数の値を変数nに入力し、確認のためその値を出力する。

そして、偶数なら2で割り、奇数なら3倍して1を足して、そのnの値を表示する処理を、nの値が1になるまで続けていく。

図1は、変数nの値を10として実行した場合、変数nの値が10、5、16、8、4、2、1と順に変化していき、6回目の変化で1となり、そこで終了する様子を表している。

この流れ図は、構造化プログラミングのプログラムの基本となる接続・判断・繰り返しの3種類の構造が、すべて組み合わせられて成り立っている。

ここで、流れ図を構成している流れ図記号の組み合わせの構造は、流れ図のアプリケーションソフトを記述したプログラミング言語 Visual Basic 内では、整数配列変数 iM を、各流れ図記号についてそれぞれ3～5個使用したポインタによるリンクのつながり方で表現する。

図1の流れ図内の流れ図記号は、上から順に「start」の文字列からなる端子記号、「n を入力」の入力記号、「n. を出力」の出力記号、「n!=1」の前判定繰り返し記号と、繰り返しの範囲を示す、前判定繰り返し記号の上部から四角の実線で囲んだ中の、一番下にある空白文字の折り返し記号、その繰り返しの範囲の中で最初に実行される「n%2==0」の判断記号、判断の内容が真の場合に続く「n/=2」の処理記号、判断の内容が偽の場合に続く「n=3*n+1」の処理記号、判断の範囲を示す四角の点線の囲み終了の後の「n. を出力」の出力記号と、流れ図の一番下にあつて最後の処理となる「end」の端子記号からな

る。

以下で図 1 を例として、整数配列変数 iM と整数配列変数 $cTNo$ と文字列配列変数 $cTbl\$$ による接続・判断・繰り返しの表現と、それらのつながり方を説明していく。

なお、 $cTNo$ の値は流れ図記号の種類番号で、図 2 で示すように、入力記号は 3，出力記号は 4，前判定繰り返し記号は 6，判断記号は 5，処理記号は 1 である。

端子記号の「start」と「end」は、 iM を作る前段階で利用するだけで、流れ図を表現する iM の中には存在しない。よって、 iM の中に現れない端子記号には種類番号はない。

$cTbl\$$ の値は流れ図記号内の文字列で、図 1 の流れ図の場合、前判定繰り返し記号内の「 $n!=1$ 」，判断記号内の「 $n\%2==0$ 」，処理記号内の「 $n/=2$ 」や「 $n=3*n+1$ 」などである。

○接続の構造

接続の流れ図記号は、 $iM(1) \sim iM(3)$ や $iM(4) \sim iM(6)$ のように、連続する整数配列変数 iM を 3 個使用する。

図 2 接続の例

流れ図の先頭		0	1	2	3	4	5	6	7
番地		0	1	2	3	4	5	6	7
iM	1	0	0	4	0	1	7	1	
$cTNo$	3	4	6	5	1	1	4		
$cTbl\$$	n	n,	$n!=1$	$n\%2==0$	$n/=2$	$n=3*n+1$	n,		

配列変数 iM の先頭、すなわち $iM(0)$ の値は特別の役割を持っている。

流れ図は、ひとつのメインの流れ図と、他に定義済み流れ図記号で示される 0 個以上の関数流れ図の組み合わせで成り立っている。そこで $iM(0)$ の値は、場合に応じて、メインあるいは関数の流れ図の先頭 iM の番地を示す。

図 1 の流れ図は、メインのみからなる流れ図で、メインの流れ図が $iM(1)$ から始まるため、 $iM(0)$ の値は 1 である。

流れ図記号を表現する iM の最初の値は、接続・判断・繰り返しのいずれかを示す区分番号である。接続か判断か繰り返しかの区別番号は、接続が 0，判断が 2，繰り返しが 1 である。

図 1 の流れ図の最初の流れ図記号は、接続の「 n を入力」の入力記号である。よって、この流れ図記号の $iM(1)$ の値は区別番号 0 である。そして、この入力記号の種類番号は 3 で、文字列は「 n を入力」であるので、それらの値を $cTNo$ と $cTbl\$$ に順に記憶していくことから、 $cTNo(0)$ の値は 3， $cTbl\$(0)$ の値は「 n を入力」となる。さらに $iM(2)$ は、流れ図記号の種類番号とその文字列を記憶している $cTNo$ と $cTbl\$$ の引数の値 0 となる。なお、「 n を入力」の文字列は、入力記号の特質により自動的に画面上に表示するので、 $cTbl\$$ には、「 n を入力」の文字列の部分を省略した「 n 」が記憶される。

次の流れ図記号の iM が始まる番号は、最初の流れ図記号が入力記号の接続であり、 iM

が3つ使われているので、 $iM(1)$ から3つ後の $iM(4)$ となる。よってこの流れ図記号の $iM(3)$ の値は、次に実行する流れ図記号の先頭 iM の番地である4となる。

「 n を入力」の入力記号に続く2番目の流れ図記号は、「 n ,を出力」の出力記号である。これも接続であるから、 $iM(4)$ の値は区分番号の0である。そして、この出力記号の種類番号は4で、文字列は「 n ,を出力」であるので、それらの値は次の $cTNo$ と $cTbl\$$ に記憶していくことから、 $cTNo(1)$ の値は種類番号の4、 $cTbl\$(1)$ の値は「 n ,を出力」となる。さらに $iM(5)$ は、流れ図記号の種類番号とその文字列を記憶している $cTNo$ と $cTbl\$$ の引数の値1となる。なお、「を出力」の文字列は「を入力」と同様、出力記号の特質により自動的に画面に表示するので、 $cTbl\$$ には、「を出力」の文字列の部分を省略した「 $n,$ 」が記憶される。

$iM(6)$ の値は、次に実行する流れ図記号の先頭 iM の番地である7となる。

○繰り返しの構造

繰り返しの流れ図記号は、 $iM(7) \sim iM(10)$ のように、連続する整数配列変数 iM を4個使用する。

図3 繰り返しの例

番地	2	3	4	5	6	7	8	9	10	11
iM	0	4	0	1	7	1	2	11	0	2
$cTNo$	6	5	1	1	4					
$cTbl\$$	$n!=1$	$n\%2==0$	$n/=2$	$n=3*n+1$	$n,$					

$iM(7)$ の値は、「 $n!=1$ 」の前判定繰り返し記号の繰り返しとなるので、区別番号1である。

そして $cTNo(2)$ の値は前判定繰り返し記号の種類番号の6で、 $cTbl\$(2)$ の値は「 $n!=1$ 」となることにより、 $iM(8)$ の値は $cTNo$ と $cTbl\$$ の引数の2となる。繰り返しには、前判定繰り返し記号の他に、後判定繰り返し記号があり、その場合は種類番号7となる。

繰り返しの3番目の iM すなわち $iM(9)$ は、繰り返しの中で最初に実行する流れ図記号の先頭 iM の番地である11となる。

図1の流れ図を見ると、この前判定繰り返し記号の実行が終わると「end」の端子記号に実行が移り、流れ図の実行はすべて終了する。

繰り返しの4番目である最後の iM 、すなわち $iM(10)$ の値は、繰り返しが終了した後に、次に実行する流れ図記号の先頭 iM の番地であるが、この流れ図記号の実行で流れ図のすべてが終了するので、この場合は特別の値の0となる。

○判断の構造

判断の流れ図記号は、 $iM(11) \sim iM(15)$ のように、連続する整数配列変数 iM を5個使

用する。

図4 判断の例

番地	11	12	13	14	15	16	17	18	19	20	21	22	23	24
iM	2	3	16	19	22	0	4	11	0	5	11	0	6	7
cTNo														
cTbl\$														

番地	3	4	5	6
iM	4	0	1	7
cTNo	5	1	1	4
cTbl\$	$n\%2==0$	$n/=2$	$n=3*n+1$	$n,$

iM(11) の値は、「 $n\%2==0$ 」の判断記号の判断となるので、区別番号 2 である。

そして cTNo(3) の値は判断記号の種類番号の 5 で、cTbl\$(3) の値は「 $n\%2==0$ 」となることにより、iM(12) の値は cTNo と cTbl\$ の引数の 3 となる。

判断の 3 番目の iM すなわち iM(13) の値は、判断の内容が真である場合に最初に実行する流れ図記号の先頭 iM の番地である 16 となる。

また判断の 4 番目の iM すなわち iM(14) の値は、判断の内容が偽である場合に最初に実行する流れ図記号の先頭 iM の番地である 19 となる。

そして判断の 5 番目の iM すなわち iM(15) の値は、判断の真と偽の両方の実行がすべて終了し、真と偽の流れ線が合流した後に、次に実行する流れ図記号の先頭 iM の番地である 22 となる。

判断の内容が真の場合に、最初に実行する接続の流れ図記号である処理記号は、cTNo(4) の値が種類番号の 1、cTbl\$(4) の値が「 $n/=2$ 」となるので、iM(16) と iM(17) の値は、接続の区分番号の 0 と cTNo と cTbl\$ の引数の 4 である。そして iM(18) の値は、次に実行する流れ図記号の先頭 iM の番地の値が入るのであるが、この流れ図記号が、判断内容が真の場合の最後の実行であり、実行が判断記号の最初に戻るため、判断記号の先頭 iM の番地である 11 となる。

同様に、判断の内容が偽の場合に、最初に実行する接続の流れ図記号である処理記号は、cTNo(5) の値が種類番号の 1、cTbl\$(5) の値が「 $n=3*n+1$ 」となるので、iM(19) と iM(20) の値は、接続の区分番号の 0 と cTNo と cTbl\$ の引数の 5 である。そして iM(21) の値は、次に実行する流れ図記号の先頭 iM の番地の値が入るのであるが、これもこの流れ図記号が、判断内容が偽の場合の最後の実行であり、実行が判断記号の最初に戻るため、判断記号の先頭 iM の番地である 11 となる。

判断内容の終了後、次に実行する流れ図記号は接続の出力記号である。出力記号は、cTNo(6) の値が種類番号の 6、cTbl\$(6) の値が「 $n,$ 」となるので、iM(22) と iM(23) の値は、接続の区分番号の 0 と cTNo と cTbl\$ の引数の 6 である。そして iM(24) の値は、次に実行する流れ図記号の先頭 iM の番地となるのであるが、この流れ図記号が繰り返し

の最後の実行であるので、実行が前判定繰り返し記号の最初に戻り、前判定繰り返し記号の先頭 iM の番地である 7 となる。

流れ図内の繰り返しと判断は、各ブロックとしての入れ子構造となっており、各ブロックの最後から先頭への巡回リンクを形成している。そして、その様子は図 1 のように、入れ子状態の実線あるいは点線の長方形の囲みで表示される。また、流れ図の実行の流れを処理するのに、繰り返しや判断の先頭 iM の番地をプッシュダウンで記憶していく。

すなわち図 1 の場合、まず前判定繰り返し記号の先頭 iM の番地である 7 をプッシュダウンで記憶し、続いて、その繰り返しの中に現れる判断記号の先頭 iM の番地である 11 をプッシュダウンで記憶する。

判断の場合は、判断内容が真の実行が終了し、続いて偽の実行が終了して、判断の実行が終了となる。判断内容が真の場合の中の、最後尾 iM の値が 11 と等しい流れ図記号の実行が終了すれば、真の実行が終了である。最後尾 iM の値が 11 と等しい流れ図記号は、iM(16)～iM(18) からなる接続の「 $n/=2$ 」の処理記号である。

そして判断内容が偽の場合の実行に移る。判断内容が偽の場合の中の、最後尾 iM の値が 11 と等しい流れ図記号の実行が終了すれば、偽の実行が終了である。最後尾 iM の値が 11 と等しい流れ図記号は、iM(19)～iM(21) からなる接続の「 $n=3*n+1$ 」の処理記号である。

判断の実行が終了すれば、11 をポップアップしてプッシュダウンの記憶から消し、チェックの対象から外す。そして、次にプッシュダウンで記憶している 7 をチェックの対象とする。

繰り返しの中の、最後尾 iM の値が 7 と等しい流れ図記号の実行が終了すれば、繰り返しが終了である。最後尾 iM の値が 7 と等しい流れ図記号は、図 1 の流れ図の下の方の、iM(22)～iM(24) からなる接続の「 n , を出力」の出力記号である。

繰り返しが終了すれば、7 をポップアップしてプッシュダウンの記憶から消す。これで、プッシュダウンの記憶が空となって、判断や繰り返しの範囲がすべて終了する。

3. 演算の処理

次に、文字列配列変数 cTbl\$ の値の、「 $n!=1$ 」・「 $n\%2=0$ 」・「 $n/=2$ 」・「 $n=3*n+1$ 」などの、演算に関する文字列の処理を解説する。

この流れ図のアプリケーションソフトを記述したプログラミング言語 Visual Basic において、演算に関しては図 5 のような処理を行っている。

文字列変数 X\$ は、処理対象となる流れ図記号内の文字列で、文字列配列変数 cTbl\$ 中のひとつの文字列である。

整数変数 C% は、X\$ に含まれている文字列から、演算子などの文字や文字列を取り出すための先頭から何文字目かを表わす文字位置の値である。

図 5 の Visual Basic の関数 Mid\$ (引数 1, 引数 2, 引数 3) は、引数 1 が文字や文字列が取り出される元となる文字列、引数 2 は引数 1 の先頭からの文字位置、引数 3 は、引

図5 演算の優先度と実行コード

If Mid\$(X\$, C%, 3) = " (" Then x1% = 3: x2 = 2.02: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 3) = "&&(" Then x1% = 3: x2 = 2.03: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = "!(" Then x1% = 3: x2 = 2.04: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 2) = "++" Then x1% = 9: x2 = 9999.48: Exit Sub
If Mid\$(X\$, C%, 2) = "--" Then x1% = 9: x2 = 9999.49: Exit Sub
If Mid\$(X\$, C%, 2) = "+=" Then x1% = 8: x2 = 1.41: Exit Sub
If Mid\$(X\$, C%, 2) = "-=" Then x1% = 8: x2 = 1.42: Exit Sub
If Mid\$(X\$, C%, 2) = "*=" Then x1% = 8: x2 = 1.43: Exit Sub
If Mid\$(X\$, C%, 2) = "/=" Then x1% = 8: x2 = 1.44: Exit Sub
If Mid\$(X\$, C%, 2) = "%=" Then x1% = 8: x2 = 1.45: Exit Sub
If Mid\$(X\$, C%, 2) = "<=" Then x1% = 3: x2 = 3.09: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = "<" Then x1% = 3: x2 = 3.09: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = ">=" Then x1% = 3: x2 = 3.1: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = ">" Then x1% = 3: x2 = 3.1: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = "= " Then x1% = 3: x2 = 3.07: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 2) = "==" Then x1% = 3: x2 = 3.08: C% = C% + 2: Exit Sub
If Mid\$(X\$, C%, 1) = "=" Then x1% = 8: x2 = 1.31: Exit Sub
If Mid\$(X\$, C%, 1) = "<" Then x1% = 3: x2 = 3.05: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = ">" Then x1% = 3: x2 = 3.06: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "+" Then x1% = 3: x2 = 4.11: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "-" Then x1% = 3: x2 = 4.12: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "*" Then x1% = 3: x2 = 5.13: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "/" Then x1% = 3: x2 = 5.14: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "%" Then x1% = 3: x2 = 5.15: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "(" Then x1% = 4: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = ")" Then x1% = 6: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "[" Then x1% = 5: C% = C% + 1: Exit Sub
If Mid\$(X\$, C%, 1) = "]" Then x1% = 7: C% = C% + 1: Exit Sub

数2の文字位置から取り出される文字数である。関数 Mid\$ の値は、引数1の先頭から引数2文字目を最初の文字として、引数3の文字数だけの文字列である。

整数変数 x1% は、演算子のグループ番号の値である。ここでは、「++」のインクリメントと「--」のデクリメントがグループ番号9、「=」の代入と「+=」・「-=」・「*=」・「/=」・「%=」の代入演算がグループ番号8、論理演算・比較演算・四則演算と剰余演算がグループ番号3と、それぞれの演算をグループとして番号分けする。またグループ番号4~7は、演算の優先度を上げる「(」と「)」の丸カッコと、配列変数の引数を囲む「[」と「]」の角カッコである。

実数変数 x2 は、整数部と2桁の小数部からなる実数の値である。この値の整数部の値は演算処理の優先度で、一番高いのが「++」のインクリメントと「--」のデクリメントの9999、一番低いのが「=」の代入と「+=」・「-=」・「*=」・「/=」・「%=」の代入演算の1である。2桁の小数部の値は、それぞれの演算実行を実現させる実行コードの値である。

演算処理の優先度と先ほどのグループ番号を組み合わせた値を使い、逆ポーランド記法により実行コードを発生させていく。なお実行コードは、整数配列変数 pp% で記憶していく。

実数変数 x2 の情報を，整数変数 x2% と整数変数 x3% の二つの変数の情報に分けることをしなかったのは，図5を見れば分かるように，演算処理の優先度と実行コードの対が実数値として一度に表示されるからである。この実数値の値を2つの整数値に分解して使用するには，小数点以下を切り捨てたり，小数部2桁を2桁の整数に直したりする必要がある。しかし，ここではこのような余分の手間や処理時間を犠牲にしてまでも，視覚的に見易く分かり易い表現を選んだ。

図6 流れ図の実行処理

```

Select Case pp%(pc)
Case 1 '変数・定数フラッシュダウン
  pdst(stpt + 1) = pp%(pc + 1) '変数・定数なら真の-1
  stack(stpt + 1) = pp%(pc + 2) 'アドレスを入れる
  stpt = stpt + 1: pc = pc + 2
Case 2 'or( ||) Case 2~27まで演算ルーチン
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) Or stack(stpt + 1)
Case 3 'and(&&)
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) And stack(stpt + 1)
Case 4 'not(!)
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  stack(stpt) = Not stack(stpt)
Case 5 '<
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) < stack(stpt + 1)
Case 6 '>
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) > stack(stpt + 1)
Case 7 '<>
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) <> stack(stpt + 1)
Case 8 '=
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) = stack(stpt + 1)
Case 9 '<=
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) <= stack(stpt + 1)
Case 10 '>=
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) >= stack(stpt + 1)
Case 11 '+'
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) + stack(stpt + 1)
Case 12 '-'
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) - stack(stpt + 1)
Case 13 '*'
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  stack(stpt) = stack(stpt) * stack(stpt + 1)
Case 14 '/'
  stpt = stpt - 1
  If pdst(stpt) Then stack(stpt) = Nv(stack(stpt)): pdst(stpt) = False
  If pdst(stpt + 1) Then stack(stpt + 1) = Nv(stack(stpt + 1))
  If stack(stpt + 1) = 0 Then esw = 50: Err_ichi X$, X$: Exit Sub
  stack(stpt) = stack(stpt) / stack(stpt + 1)

```


4. 流れ図の実行

流れ図の実行は、図6のように、整数配列変数 pp% の値である実行コードによって処理される。ここで、整数変数 pc は pp% の引数である。

まず、pp%(pc) の値の実行コードが1の場合、変数または定数の処理を行う。変数または定数の値は、プッシュダウンの記憶用の実数配列変数 stack(stpt) に入力する。また、流れ図の中の流れ図記号の文字列には、a[x] や h[3] のような配列変数 a や h が使用できるので、変数 x や定数 3 が、配列変数 a や h の引数であるのか、あるいは引数でないのかにより、プッシュダウンの記憶で使用する配列変数 pdst(stpt) の値を真か偽として次に引き継ぐ。この場合、先程の stack(stpt) の値は、変数 x の値または定数 3 である。

pp% の値の実行コードが2~14の場合は、図5の x2 の小数部2桁の実行コードに対応している。

例えば、pp% の値の実行コードが2の場合は、論理和の演算 or (II) の演算処理である。演算式 a II b は逆ポーランド記法で a b II となっているので、a すなわち中間変数 stack(stpt) の値と b すなわち中間変数 stack(stpt+1) の値の論理和の値を、新たな中間変数 stack(stpt) としている。なお変数 pdst(stpt) が真の場合の処理は、中間変数 stack(stpt) の値が流れ図記号の配列変数の引数の値であるため、あらためて配列変数の値としているのが stack(stpt)=Nv(stack(stpt)) の式である。なお、流れ図のすべての変数の値と定数は配列変数 Nv で記憶する。

図6のように、pp% の値の実行コードが3~14の場合についても同様の処理となる。た

図7 エラーメッセージ

```

Case 50
  MsgBox "0で割り算をしています。", 48, "エラー"
Case 51
  MsgBox "平方根の引数が負です。" & CRLF & "√(" & Y$ & ")", 48, "エラー"
Case 52
  MsgBox "対数の引数が0か負です。" & CRLF & "log(" & Y$ & ")", 48, "エラー"
Case 53
  MsgBox "0で割り算の余りを計算しています。", 48, "エラー"
    
```

図8 エラーメッセージの例



だし、pp% の値が14の場合は割り算の処理であるので、割る値が0のであればエラーメッセージコードの変数 esw の値を50とし、図7のように“0で割り算をしています。”のエラーメッセージを図8のように表示する。

5. 流れ図実行の処理

流れ図ソフトの実行は、図9の実行メニューで示されるように、「通常実行」・「ステップ実行」・「シングルステップ」の3種類である。ここで、流れ図内の流れ図記号の文字列はすべて、流れ図の表示と同時に実行コードの pp% に翻訳されている。

「通常実行」では、実行コードの pp% を一括して実行する。実行処理の中ではこの形式が一番速いが、実行開始から終了まで、入出力の処理を除いて、実行の途中経過を表示せずに実行する。

流れ図ソフトは、図10のように4つのウィンドウから成る。

図9 実行のメニュー

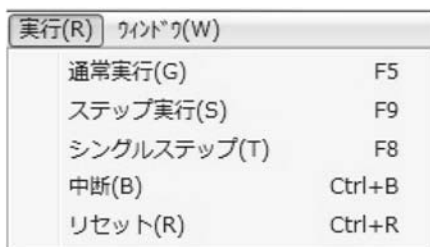


図10 流れ図のステップ実行



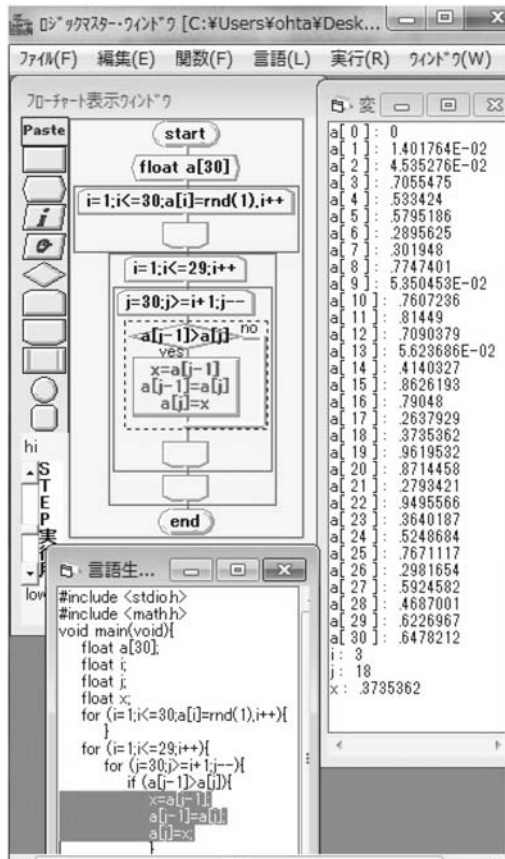
中心となるのが、左端の縦長ウインドウの「フローチャート表示ウインドウ」である。このウインドウの中に流れ図を表示する。

右の上から3つが順に、「変数値表示ウインドウ」・「言語生成ウインドウ」・「結果表示ウインドウ」である。

「ステップ実行」では、流れ図記号ごとに実行の途中経過を表示しながら実行する。実行の途中経過の切り替え速度は、「フローチャート表示ウインドウ」内左下の、「STEP 実行用」とある、上が「hi」、下が「low」の表示の上下スクロールバーをマウスでドラッグして変える。図10では、ステップ実行の速度は真ん中あたりである。流れ図記号内の文字列の色は通常は黒だが、現時点の実行途中の流れ図記号の文字列の色は、図10の「n/=2」のように赤く表示する。ここでは、流れ図記号の処理記号「n/=2」がその時点で実行中であることを示している。

また、流れ図は「言語生成ウインドウ」にc言語の様相でc言語プログラムを表示するので、ここでも実行中の流れ図記号に連動して、流れ図記号に対応するc言語の文である「n/=2;」を反転表示する。

図11 流れ図のステップ実行






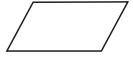
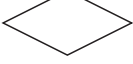
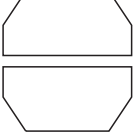


さらに「結果表示ウインドウ」では、入力記号の「n」実行時の「?」と、そこで10を入力したので「10」の文字列と、出力記号の「n,」により、変数 n の値が「10 5 16 8」と変化してきた状態を表示している。

最後のウインドウの「変数値表示ウインドウ」では、流れ図で使用させている変数すべての変数名と実行途中のその時点のおのおのの値を表示する。ここでは変数 n のひとつ前の値が 8 なので、8 は偶数であるから処理記号「n/=2」が実行され、その結果「n: 4」のように、変数 n の値が 4 となった状態での表示となっている。

図11はバブルソートの様子である。このように、配列の値がたえず入れ替えるような流れ図では、「変数値表示ウインドウ」内で、配列の値が入れ替わる様子をステップ実行で表示する。ここでは、配列の値の一番小さい値 1.401764E-02 と二番目に小さい値の 4.535276E-02 が、一番上の a[1] と二番目の a[2] の値になった時点の状態を表示している。

「シングルステップ」では、図 9 の実行メニューの「シングルステップ」の項目のマウスクリックか、ファンクションキー F8 を押すごとに、流れ図記号を 1 つずつ順に実行し

図12 流れ図記号

記号	記号名	記号の説明
	処理 (process)	任意の種類の処理を表します。よく、演算式で計算された結果の値を、変数の値として記憶するのに用いられます。
	定義済み処理 (predefined process)	関数やモジュールなど、他の場所で定義された 1 つ以上の演算または命令群からなる処理を表します。
	準備 (preparation)	変数の初期値設定や乱数発生などの準備など、その後の動作のための最初の準備処理を表します。
	データ (data)	データ（媒体を指定しないデータ）の、キーボードなどからの入力、あるいは画面などへの出力を表します。
	判断 (decision)	1 つの入口と複数の択一的な出口を持ち、記号の中の条件の評価に従って、ただ一の出口を選択することを表します。
	ループ端 (loop limit) 上図：ループ始端 下図：ループ終端	繰り返し実行の最初と最後を、ループ始端とループ終端で表します。ループ始端からループ終端に挟まれた区間の命令群は、ループ始端あるいはループ終端の中の条件の評価に従って繰り返し実行されます。ループ端の対の複合は入れ子構造になります。
	結合子 (connector)	対となる 2 つの同じ名前の結合子の、一方からもう一方に実行の流れが移ることで、流れ図の中の実行の流れの結合を表します。
	端子 (terminator)	流れ図の始まりと終わり、あるいは関数の入口と出口を表します。start, end, entry, return と、それぞれの記号内に表記します。

ながら、「ステップ実行」の実行と同じように、「フローチャート表示ウインドウ」・「変数値表示ウインドウ」・「言語生成ウインドウ」・「結果表示ウインドウ」の4つのウインドウ内に途中経過を表示する。

「ステップ実行」と「シングルステップ」は、流れ図のアルゴリズムの働きを、流れ図記号ごとに非常に明確に表示するので、アルゴリズム学習者に対して力強い助けとなる。

実行メニューの「中断」は実行中の流れ図の実行を途中で停止する。

また実行メニューの「リセット」は、実行中の流れ図の実行を途中で停止し、実行を初期の状態に戻す。

実行メニューの「通常実行」・「ステップ実行」・「シングルステップ」の操作は、「中断」で流れ図の実行が途中で停止しているのであれば、その途中から実行を再開する。

6. お わ り に

流れ図ソフトを作成する際、独自のアイデアを取り入れており、そのいくつかを紹介した。

アルゴリズムの流れ図表記は、古くからある最も定番の記法である。

流れ図を構成する流れ図記号は、図12のような形式で“JIS X 0121: 1986”「情報処理用流れ図・プログラム網図・システム資源図記号」により規格化され、形・流れる方向などが統一化されている。これら、古くから慣れ親しんでいる表記でアルゴリズムを学習することは有用と考え、バージョンアップを繰り返しながら、流れ図ソフトは現在に至っている。

この流れ図ソフトは、1995年に出版した「構造化プログラミングのアルゴリズム」の付録として、そのソフトやサンプルの流れ図が入った3.5インチのフロッピーディスクを付けての販売を起源としている。

参 考 文 献

- [1] 太田幸一：「アルゴリズム学習用ソフトの開発」, 教育システム情報学会研究報告 p. 158-161 (2007-3)
- [2] 太田幸一：「プログラミングゲーム」, ゲーム学会第5回合同研究会研究報告 p. 3-4 (2007-7)
- [3] 太田幸一：「アルゴリズム学習の教材」, 平成19年度大学教育・情報戦略大会 私立大学情報教育協会 p. 114-115 (2007-9)
- [4] 太田幸一：「プログラミング学習のためのパズルゲーム」, ゲーム学会「ゲームと教育」研究部会研究会報告 p. 5-7 (2013-3)
- [5] 太田幸一：「アルゴリズム学習のためのアプリケーションソフト」, 大阪経大論集 p. 53-66 (2014-9)