コンピュータ工学 講義プリント(7 月 17 日)

今回の講義では、フローチャートについて学ぶ。

・フローチャートとは

フローチャートは、コンピュータプログラムの処理の流れを視覚的に表し、処理の全体像を把握しやすくするために書く図である。日本語では流れ図という。

図 1 は、ユーザーに 0 以上の整数 n を入力してもらい、その後 1 から n までの全ての整数の合計 sum を計算し、最後にその sum を表示するプログラムのフローチャートの例である。

図 1 の様に、図で計算手順を表せば、プログラムの流れが視覚的に理解できるのが理解できるだろう。

また、フローチャートを使えば、特定のコンピュータ言語の知識がなくても、プログラムの処理手順が理解できるため、プログラムの仕様書などに良く使われる。

さらに、アルゴリズム(コンピュータの処理手順を、特定のコンピュータ言語によらずに表現した物)の視覚的な表現にもフローチャートが良く使われる。

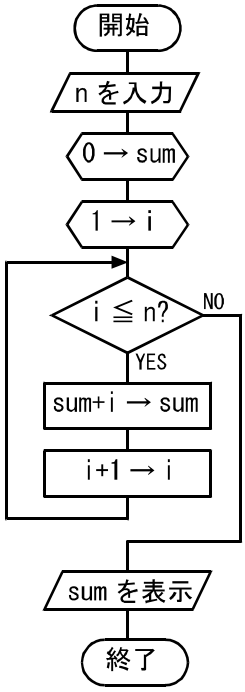


図 1、1～n の整数の合計を求めるフローチャートの例（その 1）

なお、図 1 のフローチャートから、Pascal のプログラムを作成するとリスト 1 の様になる。また C 言語のプログラムを作成するとリスト 2 の様になる。これらの例からも、フローチャートが、特定のコンピュータ言語によらずに、処理手順を図示できている事が理解できる。

なお、後の講義でアセンブリ言語(マイコンの CPU が理解できる機械語を記号化した言語)を学ぶが、変数 sum に変数 i を足して、その結果を sum に代入する(sum + i → sum)というような、Pascal や C 言語などの高級言語では 1 行で記述できる処理も、アセンブリ言語では数行に渡って記述する必要がある。そのため、アセンブリ言語ではリストを見ても処理の概要が理解しにくい事が多く、プログラムを書く前にフローチャートを描いて、処理の流れを把握しておく意義が大きい。

リスト 1、図 1 を Pascal でプログラム化したもの

```
program SumUp1(input,output);
var
    n,sum,i : integer;

begin
    readln(n);
    sum:=0;
    for i:=1 to n do
    begin
        sum:=sum+i;
```

```
    end;
    writeln(sum);
end.
```

リスト 2、図 1 を C 言語でプログラム化したもの

```
#include <stdio.h>

int main(void)
{
    int n,sum,i;

    scanf("%d",&n);
    sum=0;
    for(i=1; i<=n; i++) {
        sum+=i;
    }
    printf("%d¥n",sum);
    return 0;
}
```

・フローチャートの描き方

フローチャートの描き方には、色々な流儀がある。日本では JIS 規格でフローチャートの描き方が規格化されているものの、必ずしも JIS 規格の通りのフローチャートが一般に使われているとは限らない。以下にフローチャートの描き方の説明をするが、世間一般に使われているフローチャートが必ずしもこのルールに従っているわけではない事に注意が必要である。

フローチャートに使われる主な記号の一覧を、次のページの表 1 に示す。

フローチャートの最初と終わりには、端子記号を書く。端子記号の中には、図 1 の様に、「開始」あるいは「終了」などと書く。

計算や代入などの処理は、長方形の処理記号を使って表現する。処理記号の中には、「a に 1 を代入する」などの言葉の表記で処理内容を記述するか、「1→a」などの様に矢印で代入を表して、処理内容を記述する。

サブルーチン(Pascal の関数や手続き、C 言語の関数)の呼び出しには、定義済み処理記号を使う。記号の中には、呼び出す処理の名称を書く。

外部の装置(キーボード、ディスプレイ、プリンタ、ディスク)などと情報の入出力をするには、平行四辺形の入出力記号を使う。入出力記号の中には「x を入力」、「x を表示」など、入出力処理の内容を言葉で書く。

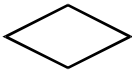
条件判断により、処理を分岐させる場合(if 文に相当)には、判断記号を使う。記号の中には「i>5?」などと、条件をクエスチョンマーク付きで書く。分岐先の流れ線や矢印には、図 1 の様に「YES」や「NO」と書いて、条件が成立した場合の流れと、条件が成立しなかったときの流れを区別する。

変数の初期化など、各種初期設定の作業は、六角形の準備記号で表す。ただし、初期設定の作業も処理

記号で表す流儀もある。

端子記号、処理記号、定義済み処理記号、入出力記号、判断記号、準備記号は、流れ線と呼ばれる実線か、矢印付きの線で結び、処理がどの順番で流れるかを示す。原則的には処理は上から下に流れるようにフローチャートを描き、各記号は流れ線で結ぶ。しかし、分岐処理や反復処理などの場合は、処理を上から下へ流す書き方ができないこともある。その場合は矢印付きの線を使って、処理の流れの方向を示す。

表 1、フローチャートで使われる主な記号

記号	名称	意味
	端子	フローチャートの開始と終了を表す。
	処理	計算や代入などの処理を表す。
	定義済み処理	プログラムの別の部分で既に定義された処理(サブルーチン)の呼び出しを表す。
	入出力	外部の装置とのデータの入出力処理を表す。
	判断	条件による分岐を表す。
	準備	変数の初期化など、初期設定の作業を表す。
	流れ線と矢印	複数の記号をつなぎ、処理の流れを示す。通常は流れ線(普通の実線)を使い、流れ線の上側の記号から下側の記号に処理が流れる事を表すが、それ以外の方向に処理が流れるときに矢印を使う。

・順次処理の例

条件判断による分岐や、反復(ループ)を含まず、各処理を順次に実行する場合の例として、ユーザーが変数 x の値を入力し、次に y に x の 2 倍を代入して、最後に y をディスプレイに表示するプログラムのフローチャートを図 2 に示す。

この例では、開始と終了の端子記号の間に、x の入力、y の計算、y の表示の処理が、上から下に一直線に並んでおり、理解しやすいであろう。

図 2 のフローチャートから Pascal のプログラムを作成した例をリスト 3 に示す。

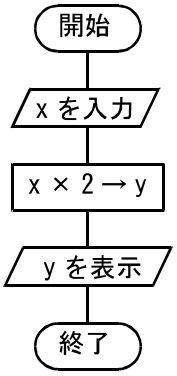


図 2、順次処理のフローチャートの例

リスト 3、図 2 を Pascal でプログラム化したもの

```
program double(input,output);
var
    x,y : integer;

begin
    readln(x);
```

```
    y:=x*2;
    writeln(y);
end.
```

図 2 とリスト 3 とを見比べると分かるように、プログラム名の宣言(program double(input,output);)や変数の宣言(var x,y:integer;)など、プログラム中の直接何らかの処理をしない行は、それらに相当する記述がフローチャート内には現れない。よって、フローチャートからプログラムを作成する際には、これらの宣言文を適時補う必要がある。

・条件分岐の例

条件判断をして処理を分岐させる例として、ユーザーが変数 x の値を入力し、x が負の値なら「マイナス」、x が 0 なら「ゼロ」、x が正の値なら「プラス」と画面に表示するプログラムのフローチャートを図 3 に示す。

判断記号により、処理が分岐すると、分岐した処理は、必ずどこかで合流する。図 3 の場合、終了の端子記号の直前で、3 つの処理の流れが 1 つに合流している様子が分かる。

条件分岐は Pascal の場合、通常 if 文によって実現される。(場合によっては case 文を使う事もある) 図 3 のフローチャートから Pascal でプログラムを生成した例をリスト 4 に示す。

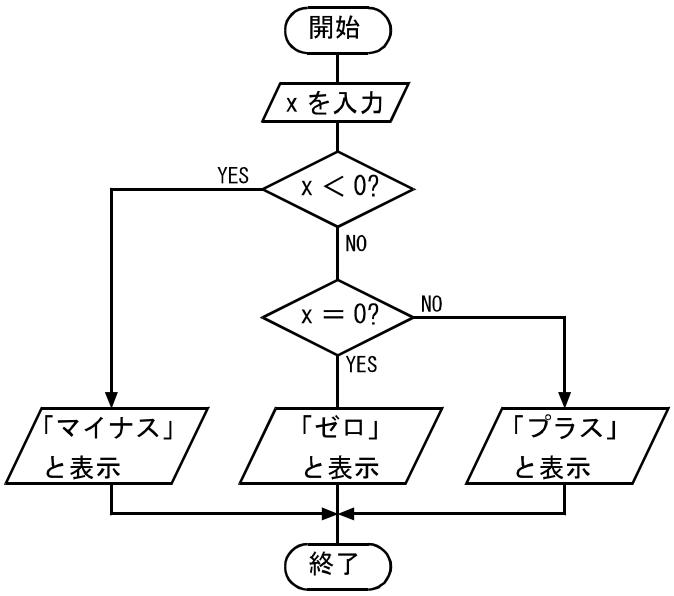


図 3、条件分岐のあるフローチャートの例

リスト 4、図 3 を Pascal でプログラム化したもの

```
program sign(input,output);
var
    x : integer;

begin
    readln(x);
    if x<0
    then
        writeln('マイナス')
    else
    begin
        if x=0
        then
            writeln('ゼロ')
        else
```

```
            writeln('プラス');
        end;
    end.
```

・永久ループの例

何らかの処理を永遠に繰り返す事を永久ループという。永久ループを含む例として、1、2、3、4、…と 1 ずつ大きな数を表示し続けるプログラムのフローチャートの例を図 4 に示す。(ただし、コンピュータで扱える数には上限があるため、このフローチャートに従って作ったプログラムは、いずれ変数 i がオーバーフローする)

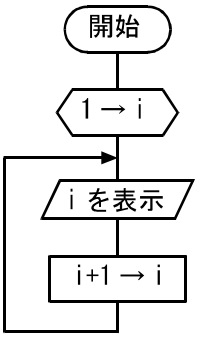


図 4、永久ループするフローチャートの例

図 4 を見ると、終了の端子記号がないことに気づく。永久ループするプログラムは、終了する事がないので、終了の端子記号がフローチャートに現れない。マイコンのプログラムは、電源を切るまで同じ処理を繰り返すように作ることが多いため、この様な終了の端子記号のないフローチャートになる場合が多い。

永久ループは、Pascal では while 文や repeat until 文により記述する事が多い。図 4 のフローチャートを、while 文を使って Pascal プログラムにした例をリスト 5 に示す。

リスト 5、図 4 を Pascal でプログラム化したもの

```
program loop(input,output);
var
    i : integer;

begin
    i:=1;
    while true do
    begin
        writeln(i);
        i:=i+1;
    end;
end.
```

・終了するループの例

図 4 の様に無限に繰り返すループもあるが、ループは通常、何らかの条件により終了する場合が多い。図 1 に示した、1 から n(入力した数)までの整数の合計を計算するプログラムは、終了するループを含む例である。

終了するループは、終了条件の判断をループの最初で行う場合と、ループの最後で行う場合とがある。(特殊な例として、ループの途中で終了条件の判断をする場合もある)

図 1 のフローチャートでは、ループの最初で終了条件を判断している。この様な場合、Pascal でプログラム化する際には、for 文や while 文を用いるのが普通である。リスト 1 では、for 文を用いている。

図 1 のフローチャートを、ループの最後で終了条件の判断をするように描きなおしたものが図 5 である。この様にループの最後で終了条件の判断を行う場合は、repeat until 文を使ってプログラム化するのが普通である。

リスト 6 に、図 5 のフローチャートからプログラムを作成した例を示す。

リスト 1 のプログラムでは、ループの最初で終了条件を判断しているため、n=0 の場合は 1 度もループが実行されず、sum=0 になる。一方でリスト 6 のプログラムでは、ループの最後で終了条件を判断しているため、n=0 であっても 1 度だけループが実行されてしまう。その結果、sum=1 となる。この様に、n=0 の時だけ実行結果が違うことに注意が必要であるが、それ以外の値を入力した場合は、どちらのプログラムも同じ結果となる。

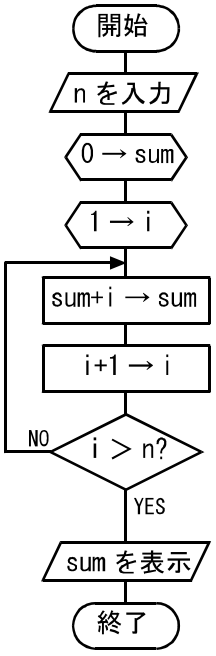


図 5、1～n の整数の合計を求めるフローチャートの例（その 2）

リスト 6、図 5 を Pascal でプログラム化したもの

```
program SumUp2(input,output);
var
    n,sum,i : integer;

begin
    readln(n);
    sum:=0;
    i:=1;
    repeat
        sum:=sum+i;
        i:=i+1;
    until i>n;
    writeln(sum);
end.
```