第3章
プログラムの構造

構造化プログラミング／基本制御構造／多方向への分岐／
ソフトウェアの開発モデル

C言語は構造化しやすい言語といわれています．構造化とは何でしょうか．また，なぜ必要なのでしょうか．この章ではC言語でのプログラミングの前提条件，基礎知識について解説します．基本理念として，プログラム作成の前提となるものです．理論としては他の言語によるプログラミングにも適用できるものです．しかし一部には言語の構造・特性上，適用できないものもあります．

3.1　構造化プログラミング

3.1.1　構造化の定理

1960年代後半から，コンピュータを利用する人の数が増えてきました．当然プログラミングに携わる人口も増えてくるなか，プログラムの作り方に対する論議も多くなりました．1966年には，ベーム（C.Bohem）とヤコピーニ（G.Jacopini）が，「プログラムが一つの入口と，一つの出口でできていれば，どのようなプログラムも順次，選択，繰り返しの三つの基本制御構造を組み合わせて作ることができる」という，いわゆる構造化（structured）の定理と呼ばれる理論を提唱しています．

また，1968年にはオランダのダイクストラ（E.W. Dijkstra）が提唱した「goto文有害説」によって，ますます構造化熱は高まりました．goto論争なるものも引き起こしました．`go to`～という構文によって，プログラム中のどこにでもジャンプできる構造は，プログラム全体を分かりにくくしている元凶であるというものです．つまり構造化を阻害する要因であるという理論です．

構造化プログラミングについては，その他にも多数の提唱がありますが，基本的には今日我々が日常行っている手法に継承されています．とくに近年はプログラムの容量が増える一方で，多数の人間の分業や協業によりプログラムを作成せざるをえません．その作業を効率的に進めるためにも，記述されたプログラムが明瞭でわかりやすく，しかも統一されているということが必要不可欠な条件になります．

もちろんプログラムが正しく動作することが必須条件ですから，事前に正しく動作することを確かめておくことが必要になります．プログラムの構造を担当する人，実際に作る人，そしてそれを検査・実証する人，というぐあいに複数の人間によって構築されるものです．設計の意図が正しく伝わりにくい構造のプログラムは，多くの場合，なんらかの問題を含んでいることが多いのです．そのためにも，構造化というテーマは避けては通れません．

3.1.2　アルゴリズムとは

問題が与えられたとき，その解法・手順の論理・理論をアルゴリズム（algorithm）といいます．JISでは，「明確に定

義された有限個の規則の集まりであって，有限回適用することにより問題を解くもの．たとえばSin xを決められた精度で求める算術的な手順をもれなく記述した文」と定義しています．明確に定義されたということは解釈が多岐に渡らないことと解釈できます．言語の文法に定められた通りにとも解釈できます．そして規則の集まりというのは，言語文法に則った命令語あるいはステートメントを使うということです．また有限回適用というキーワードについては，どんなアルゴリズムも必ず停止する，終了するということが条件ということです．あえて日本語の簡単な用語にすれば算法ということになります．また簡単にいえば「問題を解く手順」でよいでしょう．

数学の問題を解くときにも，その手法や算法があります．数学の分野は，プログラミングより歴史がありますから，定理や公理がはっきりしています．また解き方に関しては定石化している手法があります．しかし新しい数学の理論も今なお発表されています．そういう意味では，プログラミングに関する技法はまだまだ発展途中かもしれません．

プログラミング技法の多くは，広い意味での数学に裏打ちされているものが多く，そのアルゴリズムの正当性を証明するのは，数学であり論理学です．プログラムの世界においても，アルゴリズムの定石（みたいなもの）は定型化されています．しかしこうでなければならないというものは少なく，こうもできる，こういうやりかたもある，というのが実情です．しかし一定の方向性はあり，それらを早くマスタすることがプログラミング上達の鍵ともなっています．

3.1.3　構造化の目的

コンピュータが広く利用されるようになったこと，相乗的に利用のニーズが高まっていくこと，そしてハードウェアの価格，とくにメモリの価格が低下したことによって，実装できるプログラムも大きくなってきました．プログラムが大規模になればなるほど，構造化プログラミングの要求は高まってきます．

■ 良いプログラムとは

プログラムの容量が増えれば増えるほど，一つのステートメントが関連する要素は相乗的に増えてきます．プログラムの作成過程は，よく建築の過程に例えられます．しっかりした設計のもと，基礎工事が終わったうえに土台を置き躯体を組み，それをしっかりした壁で補強していきます．建具や内装，家具はその後です．まして外溝工事や装飾などは最後の最後です．

建築に比べ歴史の浅いプログラム設計は，このプロセスが無視されがちです．プログラムを作ろうとするとき，画面の配置は，色は，と細部の部品から作成を始め，積み重ねで完成していくケースも見受けます．細部から作り出したということは，建具や家具から作り出し，その隙間に柱や土台を置き，最後に基礎工事をするようなものです．当然建物自体も歪んでしまうでしょうし，強度も不十分なものになるでしょう．

骨組みのしっかりしていないプログラムは，基礎工事や躯体に欠陥のある建物みたいなもので，後々の改変や改造に耐えられません．構造化とは，この骨組みの設計をしっかりやるというプロセスのことといっても過言ではありません．そしてよいプログラムとは，所期の機能を満足することは当然として，骨組みのしっかりしたプログラムということにもなります．

プログラムの量が増えれば，テスト，検証の段階で全てを網羅できないものも出てきます．しかし構造がしっかりしているプログラムは，他人が見ても内容・趣旨の確認・検証が容易になり，検証の時間も短縮でき，目も届きます．そして全てのケースを検証しなくても，その動作の確実性が判明するものもあります．

囲碁などの世界では，「こういう場合にはこの手」という，いわゆる定石というものがあります．プログラムの世界にも，同様の手法があって当然です．すでに先人が経験した失敗の轍を踏まないためにも，定石を覚えることも上達の秘訣です．

3.1.4　トップダウン展開

プログラムを作ろうとする際に，まず基本的な機能の洗い出しをしなければなりません．機能の抽出が終われば，アルゴリズムの選択・決定ができます．良いプログラムの項でも述べましたが，骨組みのしっかりしたプログラムが

要件です．骨組みを作り，だんだんに細部に入り，後刻装飾していくのが，トップダウン展開です．まず機能としてのまとまりを把握し，大枠を固めます．その後に細分化を進め，肉付けをしていくという段階を踏んでいきます．この進め方を段階的詳細化などともいいます．

3.1.5　オブジェクト指向

近年オブジェクト指向(object oriented)プログラミングということもよくいわれます．オブジェクトは，ソース・プログラム(source program)に対するオブジェクト・プログラム(object program)，つまり目的物に対する用語ですが，ここではもう少し広い意味で，ひとつの部品という意味に使われています．

かつては一つのプロジェクトは，一人で始めて完成していることもありました．しかしコンピュータの応用範囲が広がり，アプリケーションの種類・規模も増えていくと，どうしても複数の人間によるコラボレーションにならざるをえません．しかも一部類似のアプリケーションも増えてくることになります．ましてソフトウェア開発作業の何分の1かは，仕様変更やバージョン・アップにともなう内部の改変です．

そこで前述のように，命題が既存の技術でカバーできるものはないか，新規に作成しなければならないのか，慎重な検討が必要です．また新規に作成する場合でも，その技法が将来流用される可能性があるかどうかを考えて設計しなければなりません．つまり新規に作成する場合でも，なるべく他への影響が少なく，独立性の高い構成を考えなくてはならないということです．一つの部品のような，この独立性と完成度の高いプログラム・モジュールをとくにオブジェクトと呼ぶようになっています．またデータと手続きを一体化した形態をカプセル化などとも呼んでいます．電気製品にたとえれば，モータという完成体，あるいはサブ・アセンブリのようなもので，モータそのものは扇風機にもHDドライブにもビデオ・デッキにも使えるような形態です．

コンピュータによる処理とは，入力となるデータがあり，それらに処理を加えて結果を出力する過程です．処理内容や方法の詳細がわからなくても，入力と出力の内容，仕様があっていれば，そのプログラムは利用できるということになります．

3.2　基本制御構造

前述のように順次，選択，反復の三つをプログラムの基本制御構造といいます．以下この三つの構造について説明します．

3.2.1　順次

順接，連接，順構造などともいいます．英語ではsequenceです．文字通り順番に次々と処理が続いている形態です．取り立てて制御構造というものではありませんが，プログラムの一番基本部分を示しています．**図3.2.1**にその流れ図表記を示します．

本来プログラムというのは，運動会や音楽会のプログラムがそうであるように，初めから順番に処理されていくものです．その手順を記したものです．C言語の表記においても，文1，文2･･･と，上から下へ連なっていく形になります．

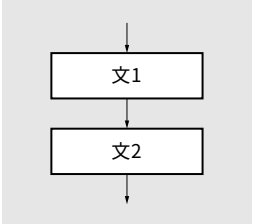

図3.2.1　順次の構造

3.2.2　選択

ある条件の下で，プログラムの流れが分かれること選択(selection)の構造といいます．**図3.2.2**にその流れを示します．

ある条件を判定し，この場合，結果は真理値ですが，真ならば処理の1，偽ならば処理の2という形です．それぞれの処理をした後は，また合流して一つの流れになります．どちらか一方しか処理されない形です．他の言語でもif～then～else～の構文などがありますが，C言語でも条件の判定にifというキーワードを使用します．「も

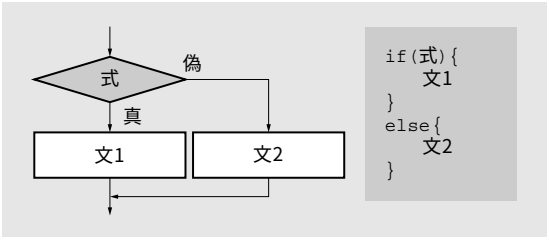

図3.2.2　選択の構造(1)

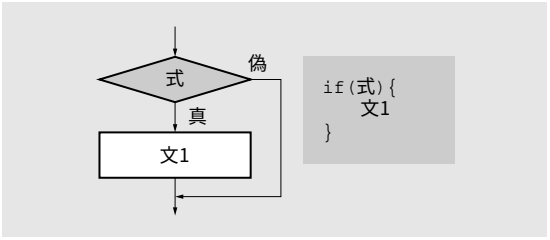

図3.2.3　選択の構造(2)

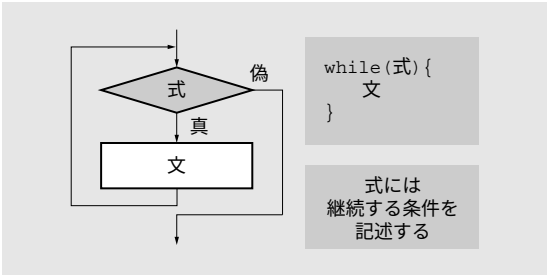

図3.2.4　前判定反復の構造

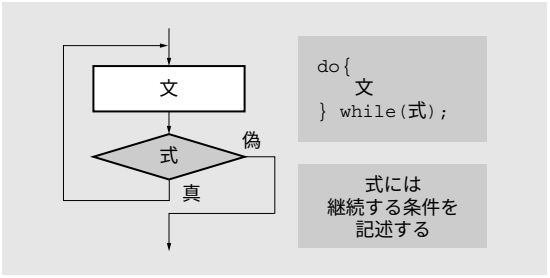

図3.2.5　後判定反復の構造

し，～だったら･･･」と解釈できます．

図にも付記しているように，`if( 式 )`の小括弧(parenthesis)の中に条件を記述します．基本的には関係演算子や論理演算子を組み合わせた論理式です．つまり指定の条件で`( )`の中が真か偽を判定して分岐します．C言語の場合，条件の式は，もし変数`a`と`b`が等しかったらというときには，

```
if ( a == b )
```

などと，論理演算子を使います．この変形として，else時の処理がない，つまりある条件が成立したときのみ処理があるという形もあります．**図3.2.3**に示すように，真の場合の処理が終わったところで合流します．

3.2.3　反復

繰り返しのことです．所定回数あるいは条件が整っている間，繰り返し同様の処理を反復するというものです．この反復(iteration)の構造ができることから，小さいプログラムでも大量のデータ処理が可能になります．C言語では，3種類の反復の表現ができます．前判定反復，後判定反復，そして所定回数反復です．

■ while文

前判定反復には**図3.2.4**に示すように，`while( )`というキーワードを利用します．カッコのなかの条件が真であるかぎり，以下の文あるいは複文を繰り返し処理します．記述形式は，

```
while ( 条件 ){
  文;
}
```

の形です．`while`の`( )`内の条件が成立しているかぎり，文で記述された処理を続けるという意味になります．条件が成立していなければ，1回も繰り返しのループに入らないこともあります．

■ do文

後判定反復では，**図3.2.5**に示すように，

```
do{
  文;
} while( 式 );
```