情報基礎シリーズ4

アルゴリズムとデータ構造

電子開発学園出版局

情報基礎シリーズ４

アルゴリズムとデータ構造

電子開発学園出版局

はじめに

コンピュータ技術の進展により、コンピュータは人類にとって不可欠なものになっています。コンピュータはその規模に応じて、スーパーコンピュータ、汎用コンピュータ、ワークステーション、パーソナルコンピュータなどに分類されます。ただ、どのようなコンピュータも基本構成は同じで、実際に計算をしたり、通信をしたりするハードウェアと、それを制御するソフトウェアから構成されています。ソフトウェアもその使途により、オペレーティングシステム、ワープロソフト、表計算ソフト、銀行の為替管理ソフト、自動車会社の生産管理ソフトなどに分類できますが、どのようなソフトウェアもプログラムによって構成されています。つまり、適切に作成されていないプログラムでは、コンピュータは意味をなさないものとなります。

適切なプログラムを作成するにあたっては、処理すべき機能（問題）の論理を正しく把握し、その論理に合った処理手順（アルゴリズム）を正確に組み合わせることが大切です。このためには、処理すべきデータがどのような構造になっていて、どのようなアルゴリズムが必要になるかを判断できる能力が不可欠となります。

本書は、代表的な「データ構造」および「アルゴリズム」を学習することで、この能力を養成することを目的としています。アルゴリズムの表記には、いくつかの方法がありますが、本書では第２章においてプログラム流れ図（フローチャート）を、第３章以降は擬似言語を用いて、段階的に学習することを可能としています。

第１章：アルゴリズムを学習するにあたって、プログラムの作成の基本的な知識を学習します。
第２章：フローチャートをもとに、アルゴリズムの基本を学習します。
第３章：第２章のアルゴリズムの基本的な処理を擬似言語に置き換えて擬似言語の記述方法を学習します。
第４章：基本的なデータ構造として「配列」を取り上げ、「配列」を利用する各種アルゴリズムを学習します。
第５章：応用的なアルゴリズムとして問題向けデータ構造と再帰のアルゴリズムを学習します。
第６章：事務処理系プログラムを対象としたファイル処理のアルゴリズムを構造化した擬似言語プログラムを用いて学習します。

付録１：アルゴリズムの評価方法を紹介しています。
付録２：構造化チャートによる表記法を紹介しています。

本書では、章ごとに多くの【練習問題】を用意しています（ダウンロード方式)。ぜひ、多くの練習問題にチャレンジして「データ構造」「アルゴリズム」を身につけて下さい。また、プログラム言語を学習されている方は、学習した「データ構造」「アルゴリズム」をそのプログラム言語でプログラミングすると、いっそう理解が深まると思います。

編著者

目　　次

情報基礎シリーズ４

アルゴリズムとデータ構造

第１章　プログラム作成の基礎知識

１．１　プログラム作成とアルゴリズム

プログラム作成の技術を身につけるためには、「アルゴリズムを学び、論理的に手順を組み立てる力を養う」と同時に、プログラムで処理される「データの構造について知識を持つこと」が大切である。

１．１．１　プログラム作成とは

プログラムはコンピュータに対する命令を記述したものである。コンピュータでデータ処理を行うには処理させる手順に従って記述したプログラムを作成しなければならない。プログラムの作成(プログラミング）は以下の 3 段階を経て行われる。

① 問題を分析して、何をするプログラムを作るのかを決める。

↓

プログラム仕様書

↓

② どのような処理手順、データ構造を使用するのかを決める。

↓

プログラム(内部)設計書

↓

③ 処理手順をプログラム言語で記述する（この作業はコーディングという)。

↓

プログラム

本書では、主に「① 何をするプログラムを作るのか」「② どのような処理手順、データ構造を使用するのか」について学習する。

１．１．２　アルゴリズムとは

私たちの日常生活でも、ある作業をするときには、その作業の手順に従って行う必要がある。**「アルゴリズム(algorithm)」**とは、この作業の**「処理手順」**である。良い「処理手順」を用いれば、作業も速くなり、間違いも少なくなる。

コンピュータでデータ処理を行う場合も、「良いアルゴリズム」を用いることが大切である。良いアルゴリズムは次のことが上げられる。

・正しく処理が行われ信頼性が高いこと。
・効率が良いこと。
・わかりやすいこと。

アルゴリズムを表現する方法として、図式を用いる「フローチャート(流れ図)」や、各種プログラミング言語によるプログラムがある(あくまでもプログラムはアルゴリズムを表現したものである。プログラムは著作権法で保護の対象となっているが、アルゴリズムそのものは保護の対象となっていない)。

本書ではアルゴリズムの学習に「フローチャート」と「擬似言語」を用いる。基本情報技術者試験の擬似言語は平成 13 年春の問題から出題に用いられている言語である。試験のために作られた言語でコンピュータによる実行はできない。

１．１．３　データとデータ構造の概要

（1）データと記憶領域

コンピュータのデータ処理は、データを入力して、データを加工し、結果を出力することである。入力したデータや加工したデータ、出力したデータは、主記憶装置の中に記憶していなければならない。

これら種々のデータを記憶する場所を総称して**記憶領域**という。プログラムを作成するときは、記憶領域にどのようなデータを記憶させるかを考え、適切な名前(データ名）をつける。

例　「単価と数量を入力し、金額を求めて金額を出力する。」

この例では、主記憶装置内に、

・入力する単価を記憶する場所
・入力する数量を記憶する場所
・計算によって求められる金額を記憶する場所

を決めて、データを格納するようにする。また、それぞれの場所に名前(データ名)をつけて表記する。

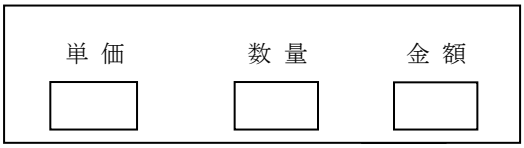


図 1.1　主記憶装置

（２）データ構造

プログラムで扱うデータは同じ性質を持つデータをまとめて**「データ型」**として分類され、同じデータ型の記憶領域に記憶される。使用されるデータの種類によって処理に応じて適切なデータ型を選択する必要がある。また、誤差等の問題が発生する場合もあるので、内部でどのように記録されているか知っておくことも大切である。

広い意味でデータ型を表現・構成したものを**「データ構造」**という。データ構造は、次の図の通り、**「基本データ構造」**、**「問題向きデータ構造」**に分けられる。

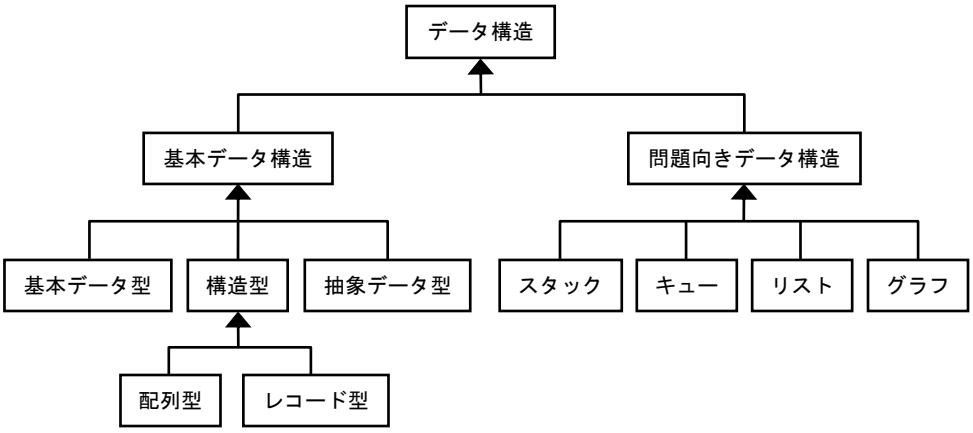


図 1.2　データ構造

（３）基本データ構造

基本データ構造は**「基本データ型」**、**「構造型」**、**「抽象データ型」**がある。ほとんどのプログラム言語で直接表現することができる。

・基本データ型

整数型　整数(小数点以下の数字は表現できない)を扱うデータ型である。

例　1　2　100　256　1024

実数型　実数(小数点以下の数字も表現することができる)を扱うデータ型である。

例　0.1　3.1415　456.7

論理型　論理値(2 通りの状態)を扱うデータ型で、論理演算に使われる。

例　0 または 1　　true または false

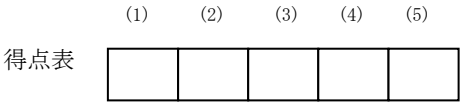
文字型　英字、数字、記号など文字として扱うデータ型である。

例　“A”　“B”　“c”　“1”　“2”　“#”

・構造型

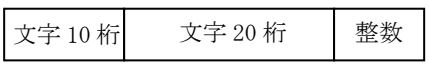
配列型　同一のデータ型・大きさの複数の要素から構成されるデータ型である。

例　整数の記憶領域が 5 回繰り返す。



レコード型　必ずしも同一ではないデータ型、同一ではない大きさの複数のデータから構成されるデータ型である。

例　氏名(文字 10 桁)、住所(文字 20 桁)、電話番号(整数) で構成する。



・抽象データ型

データと操作をひとまとめにしたデータ型。データに対するアクセスは公開されている操作を用いる。その内部処理は隠されている。オブジェクト指向言語で用いられる。

（４）問題向きデータ構造

データをコンピュータで効率よく操作するためには、個々の問題に合わせて、データ間の関連性にもとづき、操作に都合のよいデータ構造を選択する必要がある。この個々の問題に適合したデータ構造を「問題向きデータ構造」という。問題向きデータ構造はプログラム言語によっては表現できないものもあり、前項の基本データ構造を用いて作成しなければならない。

問題向きデータ構造として代表的なものを次に示す。

・スタック：　配列を用いたデータ構造で、配列の一端からのみデータの格納・取出しを行う。最後に格納したデータが最初に取り出される。

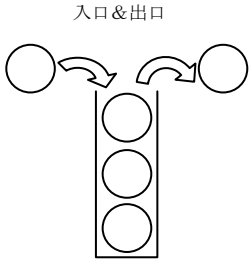


図 1.3　スタック

・待ち行列(キュー)：　配列を用いたデータ構造で、配列の一方の端からデータの格納が行われ、もう一方の端から取り出される。最初に格納したデータは最初に取り出される。

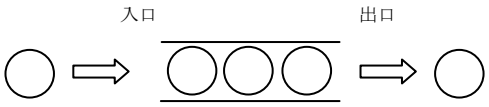


図 1.4　待ち行列

・リスト　：データを論理的な並びとして表す構造である。この構造では、データの物理的な位置には関係なく、データを参照することができる。

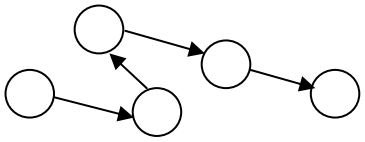


図 1.5　リスト

・木構造　：データ間に階層を持つ構造である。この構造では上位の階層から下位の階層

にデータを順にたどっていくことができる。

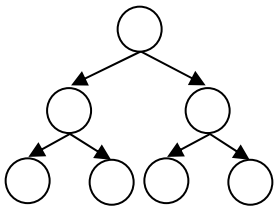


図 1.6　木構造

・グラフ　：道路網のように、データ間の関係(都市などの関係）を表す構造である。
この構造では、木構造と違い、要素間の関係を自由に設定できる。

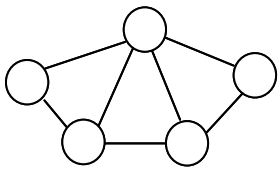


図 1.7　グラフ

（５）アルゴリズムとデータ構造

アルゴリズムはプログラムの処理手順であり、プログラムで処理されるのはデータである。限られたコンピュータ資源を有効に活用するためには、適切なデータ構造を選択し、それにあったアルゴリズムを用いる必要がある。データ構造が変更されると、それに伴ってアルゴリズムの変更も余儀なくされる。つまり、アルゴリズムは、データ構造に左右される。

Pascal（パスカル）というプログラミング言語を設計した Wirth(ビルト)は、

アルゴリズム　＋　データ構造　＝　プログラム

という名言を残している。プログラム作成にあたっては「データ構造にあったアルゴリズムを考えること」、あるいは「アルゴリズムにあったデータ構造を考えること」が大切である。

１．２　プログラム作成とプログラム言語

１．２．１　プログラム言語

（1）プログラム言語の変遷と分類

プログラム言語はハードウェアに依存する**低水準言語**から、ハードウェアに依存しない問題向き言語ともいわれる**高水準言語**へ変遷してきた。プログラミングが容易になるように、また用途・目的に応じて新しい言語が開発され、改良された。それぞれ特徴があり現在もなお使用されている言語もある。

次に、古いプログラミング言語の種類から順にその特徴を示す。

・低水準言語　ハードウェアに依存した言語で機械向き言語とも呼ばれる。コンピュータが直接理解できる**機械語**と、プログラミングが容易にできるように機械語に短い記号や単語を割り当てした**アセンブラ言語**がある。

・高水準言語　人間が日常使用する言語に近い形で作られており、問題向き言語ともいわれている。低水準言語より容易にプログラミングできる。ハードウェアに依存しないため移植性が高くなっている。

（次に分類される言語は広い意味での高水準言語である）

・手続き型言語　代表的な高水準言語。処理手続きを記述することによってアルゴリズムを表現するプログラム言語。
1954 年 FORTRAN、1958 年 ALGOL、1959 年 COBOL、 1964 年 BASIC、
1968 年 Pascal、1972 年 C 言語　など

・非手続き型言語　関数の組合わせ(関数型)や論理式(論理型)によりプログラムを作成する。
1960 年 LISP(関数型)、1972 年 Prolog(論理型)　など

・オブジェクト指向言語　対象をオブジェクトとしてとらえプログラムを作成する。
1983 年 C++、 1995 年 Java、 2000 年 C# など

・スクリプト言語　エンドユーザが容易にプログラミングできる簡易言語であるが、Web アプリケーションシステムの構築ができるものもある。
1987 年 Perl、1991 年 Python、1995 年 JavaScript、1995 年 Ruby

（２）手続き型言語

問題を解く手順と同じように処理手続きを記述することによってアルゴリズムを表現する。プログラムの命令は処理手続きに従って順次実行される。1957 年 FORTRAN が公開されて以来、多くの言語が作られた。

・FORTRAN(FORmula TRANslator)

1957 年に公開された、世界初の高水準言語である。当初は IBM 社が中心となって開発した、数式を中心とした科学技術向きコンパイラ言語である。

・COBOL(COmmon Business Oriented Language)

1959 年に初公開された事務処理向きのコンパイラ言語である。当初は、アメリカ国防総省が中心となり開発し、その後、業界団体 CODASYL(Conference on Data Systems Languages)から ANSI へと標準化の中心が移行し、改良がなされている。

・ALGOL(ALGOrithmic Language)

1950 年代後半、ヨーロッパの研究者が共同開発したコンパイラ言語である。構造化プログラミングを本格的に採用した。その後開発された多くの言語に影響を与えた。

・PL/ I

1960 年代に IBM 社によって、ALGOL を基盤にして開発されたコンパイラ言語である。事務向けと科学技術向けの両方をあつかう言語として注目された。

・Pascal(Philips’ Automatic Sequence Calculator)

1968 年に公開された ALGOL 系の教育用コンパイラ言語である。スイス連邦工科大学の Niklaus Wirth 教授らが開発したもので、パソコンなどで利用された。

・BASIC(Beginners’ All-purpose Symbolic Instruction Code)、Visual BASIC

1964 年に公開されたインタプリタ言語である。FORTRAN をベースに、米国ダートマス大学で開発されたもので、その名の通り初心者向けで、パソコンユーザ間や教育目的で使用された。

・C

1972 年に公開されたコンパイラ言語であり、AT&T 社のベル研究所で、デニス・リッチー(Dennis.Ritchie)を中心に ALGOL を参考に開発された。最初は UNIX 開発向けだったが、その後広く普及している。「ビット操作、ポインタ型のアドレス指定といった機械に近い(低水準：アセンブラに近い) レベルの処理にも適している。

（３）オブジェクト指向言語

オブジェクトは処理とデータを一体として定義したものである。オブジェクトはそれぞれ単独で機能するがオブジェクトを２つ以上あわせてお互いに連携させ別の機能を生み出すことができる。このオブジェクトの集まりとしてプログラムを作成する技法をサポートするプログラミング言語である。

・Java

1995 年に初公開されたオブジェクト指向言語である。旧 Sun Microsystems 社が開発し、それを買収した Oracle 社により継続されている。作成されたプログラムは各 OS の Java 仮想マシン(実行環境)のもとで動作する。そのため OS に依存しないプログラムを作成できる。

大企業などの基幹システムから、携帯電話などで使用される小型のアプリケーションまで、幅広く利用されている。

・C++、C#

C++は、1983 年に AT&T 社によって C 言語の拡張として開発されたオブジェクト指向言語である。「C からの上位互換性」、「一つのクラスが複数のクラスを継承する多重継承が可能」といった特徴がある。

C#は、Microsoft 社が C や C++をベースに Java の一部機能などを取り込んで拡張し、2000 年に公開した言語である。 Java 同様の中間言語へコンパイルし、 .NET Framework が解釈して実行する。

（４）スクリプト言語

エンドユーザ向きの言語で簡易プログラミング言語とも呼ばれる。スクリプトと呼ばれる短いプログラムをコンパイルの処理なしにそのままコンピュータで翻訳実行する。データの型の厳密な定義を必要としないものもあり、処理を手軽に記述して実行ができる。

次にあげるスクリプト言語は Web サーバでアプリケーションプログラムを実行する仕組み(CGI:Common Gateway Interface)を記述することができ、Web アプリケーションシステムの構築にも使用されている（Web アプリケーションは「1.2.3（4）」を参照）。

・Perl(Practical Extraction and Report Language)

1987 年ラリー・ウォール(Larry Wall)によって開発されたオープンソースのスクリプト言語である。テキストファイルの読み書きや整形を得意とする。Web アプリケーション、システム管理、テキスト処理などのプログラムを書くのに広く用いられている。

・PHP：Hypertext Preprocessor(Personal HomePage)

1994 年、ラスマス・ラードフ(Rasmus Lerdorf)が作成したオープンソースの埋め込み型スクリプト言語。HTML ファイル内に、処理内容を記述したスクリプトを埋め込み、処理結果に応じて動的に文書を生成し送出する。動的な Web ページを実現できる。

・Python

オランダ人のグイド・ヴァン・ロッサム(Guido van Rossum) が開発したオープンソースのオブジェクト指向型スクリプト言語。強力な動的型付けの機能を持っている。1991 年にバージョン 0.90 が公開されて　Web アプリケーションシステムの開発に用いられている。

・Ruby

1995 年にネットワーク応用通信研究所の松本行弘氏により開発され、発表されたオープンソースのオブジェクト指向型スクリプト言語である。 従来 Perl などのスクリプト言語が用いられてきた領域でのオブジェクト指向プログラミングを実現する。Python の機能とも共通する点が多い。

１．２．２　その他の言語

（１）マークアップ言語

タグと呼ばれる意味をつけた文字列でデータを囲み(マークアップという)、データの修飾を行ったり、データの構成を記述することができる。通常、開始タグと終了タグのペアで用いられる。

・SGML(Standard Generalized Markup Language)

マークアップを、初めて本格的に採用した言語である。マークアップ言語を定義するための言語であり、メタ言語ともいえる。SGML の仕様は大きく複雑だったため、直接の利用はあまり広がっていないが、タグの仕組みや基本的な部分は HTML や XML に受け継がれている。

・HTML(Hyper Text Markup Language)

Web ページの作成に利用されている言語。インターネットを統括する団体：W3C(World Wide Web Consortium)が策定している。SGML がもとになっている。私たちが閲覧している Web ページの表示は、この HTML で書かれたものをブラウザソフトがタグに従って変換したものである。

・XML(eXtensible Markup Language)

1998 年に W3C(World Wide Web Consortium)により、インターネット上などで異なったアプリケーション間でのデータ交換等を容易に行うために策定された言語である。SGML がもととなっている点では HTML と共通である。XML は DTD(Document Type Definition) により作成者が文書構造を定義することができ、文書構造のチェックが可能である。

「XML　Schema」は XML 文書の構造を定義する言語である。DTD の使いづらい面を改善して開発された。データ交換ではお互いに共通の文書構造の定義が必要である。そのため DTD が使われていたが、要素のデータ型が定義できない、文法が XML と異なっている等、XML では使いづらい面があった。

XML は、その活用性・拡張性の高さから、規約の定義など様々な分野でも利用されている。

・DOM (Document Object Model)

W3C により策定された。HTML 文書や XML 文書をアプリケーションから利用するための API である。

・SAX (Simple API for XML)

XML のメーリングリストの有志により策定された。HTML 文書や XML 文書をアプリケーションから利用するための API である。

・SOAP (Simple Object Access Protocol)

主にネットワークを経由したプログラム間でオブジェクトを交換する規約である。

・SVG (Scalable Vector Graphics)

XML をベースとしている。ベクタ形式の画像を表現するための規格である。

・XHTML(eXtensible HyperText Markup Language)

HTML を XML に適合するように定義しなおした。Web ブラウザでも問題なく見られ、かつ XML に準拠した文書を作成できる言語である。

・スタイルシート

Web ページの表示は、近年、よりビジュアル性が求められるようになり、タグが多階層になり分かりにくかったり、タグによる飾り付けでは難しかったりする場合が多い。そこで、マークアップ言語の構造と表示形式を分離するための仕様として考えられたのが スタイルシートである。

HTML に対応するスタイルシートとしては、CSS(Cascading Style Sheets：段階スタイルシート）が使用される。XML では、表示の多様化もあるが、タグで表すデータ特性も同様に多様化している。

XML に対応するスタイルシートとしては、CSS の利用もできるが、表示とデータの双方の多様化に適した XSL(eXtensible Stylesheet Language：**拡張可能なスタイルシート言語**）が使用される。

（２）システム設計に用いる言語

・UML(Unified Modeling Language(**統一モデリング言語**))

オブジェクト指向設計におけるモデリングの表記法を定めたもので、数種類の図を定めた統一的な作成基準である。表記法という意味の言語である。

・ユースケース図

システムにかかわる人や外部システムと、システムで実現する機能を表した図である。

・クラス図

クラス間の関係を定義する。クラス名、ロール名、属性、操作などを定義した図である。

・シーケンス図

オブジェクト(クラスの実体) 間のメッセージのやり取りを時間軸で表した図である。

・オブジェクト図

オブジェクト間の関係を表した図である。

・コラボレーション図

オブジェクト間のメッセージのやり取りを接続関係の視点から表した図である。

・ステートチャート図

オブジェクトの内部状態の変化を表した図である。

１．２．３　プログラミングに関する知識

（１）プログラム作成のためのプログラミング作法

【プログラミング作法】

読みやすく理解しやすいプログラムを作成するには、作成にあたって一定の作法に従って記述する必要がある。これによりバグにつながるプログラミングのミスを発見しやすくなる。会社やプログラム言語によって異なるが、一般的には次のような項目がルール化されている。

- **インデンテーション（字下げ）**　コーディング開始位置を、命令単位や命令のブロック単位などで一定の間隔後ろにずらすこと。開始と終了が明確になり抜けがあると発見しやすい。
- **区切りとしての空白の使用**　演算記号やカッコ等の区切り記号、命令と引数などの前後に空白を入れて見やすくする。
- **プログラム名や変数名等のネーミングルール（命名標準）**　プログラム名や変数名など、その種類や機能、使用目的によって名前の付け方をルール化する。プログラム作成者以外でも名前だけで理解しやすくなる。
- **命令の使用方法や使用禁止命令の設定**　不適切な命令の使用により処理手順が複雑で秩序のない状態のプログラム（スパゲッティプログラムという）にならないよう命令の使用法を制限する。例　飛び越し命令の禁止、繰返しの多重化（ネストの深さ）の制限。
- **コメントの使用法**　定義した変数、命令ブロックの処理内容、プログラムの機能などコメントを付ける。プログラム作成者以外でも理解しやすくなる。

【コーディング基準】

プログラムのソースコードの記述を行なうことを**コーディング**という。コーディングにおいて、その作法を明文化したものが**コーディング基準**である。システム開発を行うプロジェクトでは、この「コーディング基準」をプロジェクトメンバーが共通に使用してプログラム作成を行っている。これによって、作成するプログラムの可読性・保守性の向上が図られる。

（２）プログラムの構造

プログラムの中で同じ処理の記述が繰り返しあった場合、プログラムが長くなりメモリー等の資源を無駄に使うことになる。同じ処理を行う部分は「別のプログラム」にして、必要なときに呼び出すと、「もとのプログラム」もすっきりし無駄もなくなる。この「別のプログラム」を**サブプログラム**または繰り返し使用されるルーチン処理なので**サブルーチン**と呼ぶ。サブル

ーチンに対して「もとのプログラム」を**メインルーチン**または**メインプログラム**と呼ぶ。

大規模なプログラムを作成するとき、機能を持った小さなプログラム(モジュール)に分けて作成し、その機能を持ったプログラムを部品として組み合わせるという方法がとられる。この考え方が**モジュール分割**である。モジュールは同じ機能を持ったモジュールと交換可能である。

モジュール分割はプログラミングにおいて様々な利点がある。

・共通処理をひとつにまとめる。
・思考しやすい大きさにする。
・再利用する。
・機能別に整理する。
・複数の人で分担できる。
・バグの原因を局所化できる。　など

モジュール分割においては、モジュールを変更しても他のモジュールに与える影響が少ないこと、**独立性**が求められる。

独立性を高めるには、他のモジュールとのデータのやり取りは、モジュール内部のデータを直接アクセスするのではなく、引数を用いて単一のデータをやり取りする方法をとるなど、**モジュール結合度**を弱くする。また、単一の機能は１つのモジュールにまとめるなど**モジュール強度(凝集度)**を高める必要がある。

（３）プログラムの品質基準

ISO2001 ではソフトウエア品質特性として次の 6 種類が定められている。システム開発を行う上ではこの項目を目標にするべきである。

・**機能性**　目的を達成するために必要な機能があり、正しく動作し、安全である。
・**信頼性**　機能が正常に動作し続け、障害に強く、復旧が早い。
・**使用性**　操作がわかりやすく、使いやすい。
・**効率性**　速度が速く、使用する資源が少ない。
・**保守性**　プログラムがわかりやすく、変更しやすく、テストしやすい。
・**移植性**　他の環境へ移しやすく、そのまま動作する。他のソフトと共存できる。

（４）Web プログラミング

Web プログラミングは Web アプリケーションのプログラミングをいう。Web アプリケーションはクライアント側のブラウザからの処理要求を入力し Web サーバ内のアプリケーションプログラムが実行され、その結果をブラウザへ返す、という仕組み(CGI:Common Gateway Interface)をもっている。後方にデータベースシステムを連携させ、電子掲示板やブログ、電子商店街、インターネットバンキング、オンライントレードなど様々な Web アプリケーションシステムが構築されている。

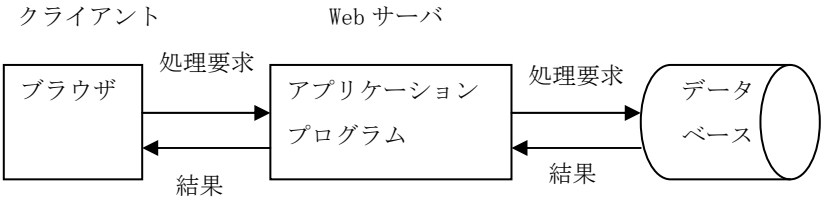


図 1.8　Web アプリケーションシステムの構成

１．３　主な問題向きデータ構造の概要

１．３．１　配列

（１）配列の概要

データ型・大きさが同一のデータが要素となって連続して並んでいるデータ構造。先頭から順番に番号(添字）が付いており各要素の位置はこの番号で指定する。

次に得点表の例を示す。Ａ君の得点、Ｂ君の得点、Ｃ君の得点、Ｄ君の得点、Ｅ君の得点を別々に記憶領域を確保してデータ名を付ける場合、一つ一つデータ名を指定して処理を行う必要がある。配列の場合は、配列の添字を順に変化させることにより順番に連続して処理することができる。

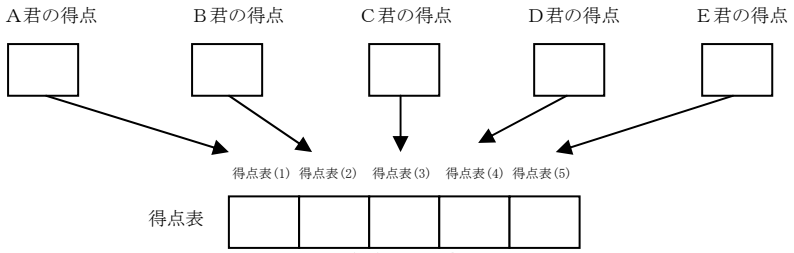


図 1.9　配列

配列要素の指定は、**配列名[添字]** の形式で行う。上記の例で 1 番目の配列要素は、得点表[1]、 2 番目の配列要素は、得点表[2] となる（添字は 0 から始まる場合もある）。

配列の欠点は要素の挿入・削除である。要素を削除する場合、データを詰め直す必要がある。

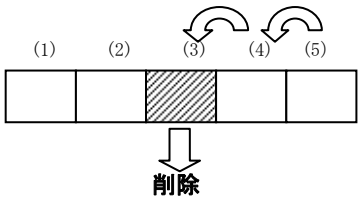


図 1.10　配列の削除

要素を挿入する場合は、後ろにデータをずらして挿入する場所を確保する必要がある。

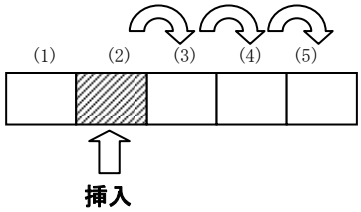


図 1.11　配列挿入

（２）配列に関する知識

添字が１つの配列を１次元配列、添字が２つの配列を２次元配列という。次の例は２次元配列である。要素の指定は、**配列名[行の添字，列の添字]**の形式で行う。

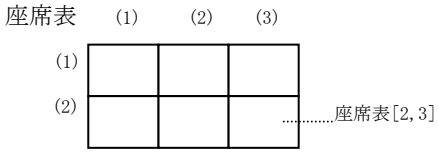


図 1.12　２次元配列(2 行 3 列の例)

上図は２次元配列なので、行添字と列添字の２つの添字を使用している。つまり、添字の個数が次元数である。２次元以上の配列を**「多次元配列」**という。

通常、配列を定義する時点で要素数を指定する。定義した要素数を越えてデータを書き込むことはできない。このような配列を**「静的配列」**という。それに対して、配列を使用する時点で配列の要素数が変わる**「動的配列」**がある。

１．３．２　スタック

配列を用いたデータ構造で、配列の一端からのみデータの格納・取出しを行う。スタックへのデータの格納を**プッシュ**(push)、スタックからのデータの取出しを**ポップ**(pop)と呼ぶ。最後に格納したデータが最初に取り出される。**後入れ先出し**(LIFO: Last In First Out)という。

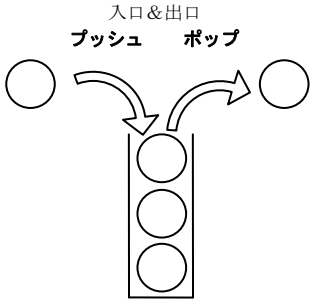


図 1.13　スタック

スタックは後述するクイックソート、サブプログラムの戻り番地の保存、数式処理の逆ポーランド記法や後述する木やグラフの深さ優先探索などに用いられる。

１．３．３　待ち行列(キュー)

配列を用いたデータ構造で、配列の一方の端からデータの格納(エンキュー(enqueue))が行なわれ、もう一方の端から取出し(デキュー(dequeue))がされる。最初に格納したデータは最初に取り出される。**先入れ先出し**(FIFO : First In First Out)という。

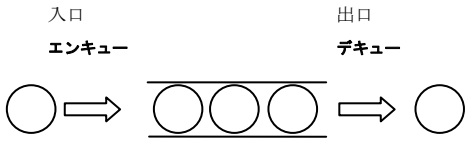


図 1.14　待ち行列

待ち行列は到着した順番に処理されるプリンタの印刷管理、後述する木やグラフの幅優先探索などに用いられる。

１．３．４　リスト

（1）リストの概要

データを論理的な並びとして表現する構造である。各データは自分自身の次のデータへの位置情報(ポインタ)を持っている。このポインタをたどって論理的な順番でデータを参照することができる。そのためデータは物理的な順番に並んでいなくてもよい。

リストは次のデータへの位置情報を持つポインタが 1 つで、1 次元配列と同様に 1 列に並んだデータを扱うため**線形リスト**と呼ぶ。

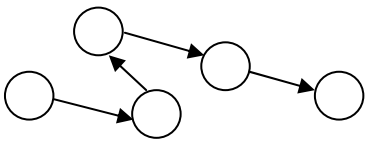


図 1.15　リスト

リストはポインタの付き方で、いくつかの種類がある。

単方向リストは次のデータへのポインタのみで、データを片方向にしか、たどれない。上図は単方向リストである。

双方向リストは次のデータへのポインタと前のデータへのポインタを持つリストである。両方向にデータをたどることができる。

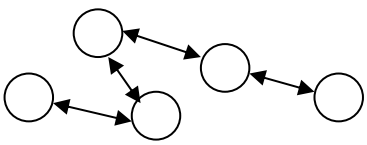


図 1.16　双方向リスト

環状リストは末尾のデータのポインタが先頭のデータの位置を示し、全要素が環状につながっている。

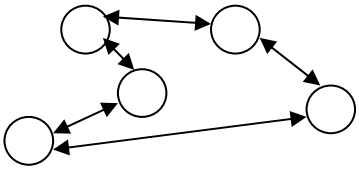


図 1.17　環状リスト

（２）リストの管理

リストは配列の欠点であるデータの挿入・削除を、ポインタを操作することで高速に行うことができる。

挿入する場合、データは空いた場所に入れる。「挿入するデータ」のポインタが「直後のデータ」を指すようにする。

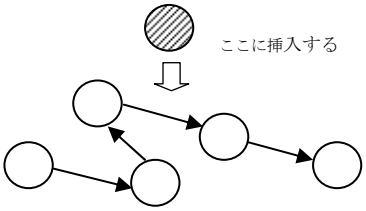


図 1.18　挿入の前

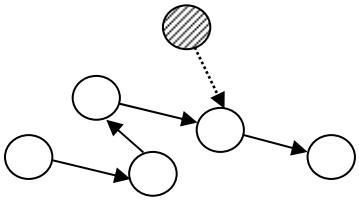


図 1.19　「挿入するデータ」のポインタ

「直前のデータ」のポインタが「挿入するデータ」を指すようにする。

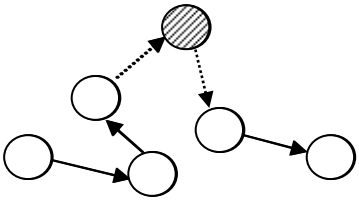


図 1.20　「直前のデータ」のポインタ

削除する場合は、「削除するデータ」の「直前のデータ」のポインタが「直後のデータ」を指すようにする。

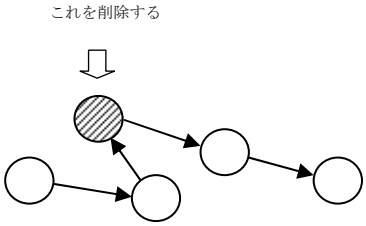


図 1.21　削除の前

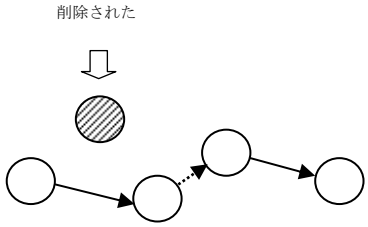


図 1.22　「直前のデータ」のポインタ

１．３．５　木構造

（１）木構造の概要

木構造は次の要素への位置情報(ポインタ)を２つ以上持つデータ構造である。要素間の階層構造を表現でき、最上位の要素を**根**（ルート)、最下位の要素を**葉**（リーフ)、各要素間のつながりを**枝**(ブランチ)という。根から枝を順にたどってデータを参照することができる。上位の階層と下位の階層は親子関係になる。子要素にとって親要素は１つしか存在しない。木構造に含まれる一部分の木構造を**部分木**という。

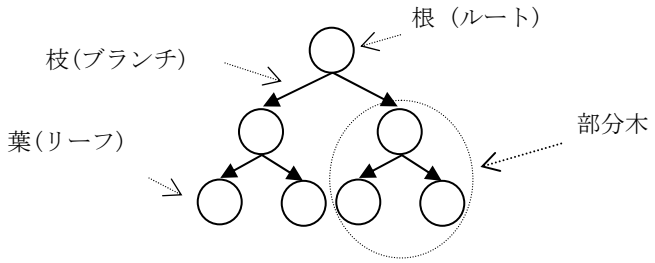


図 1.23　木構造

（２）各種の木構造

要素の順番を保ちながら新たな要素を挿入でき、常に順序(たとえば「左の子 ＜ 親 ＜ 右の子」)が保たれている木を**順序木**という。順序木では根から要素をたどっていくと整列されたデータを取り出すことができる。

データの挿入・削除を行うと特定の葉だけが深くなり、効率が低下する。すべての葉について、深さがほぼ等しくなるように再編成を行う木構造を**バランス木**という。次の図は順序木でもあり、バランス木でもある。

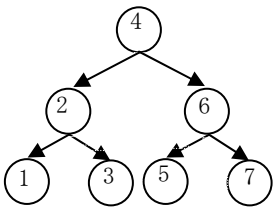


図 1.24　順序木 、バランス木の例

ポインタを 2 つ持つ木構造を **2 分木**、3 つ以上持つ木構造を**多分木**と呼ぶ。2 分木のうち、すべての葉が同じ深さにある場合、**完全 2 分木**という。ただし葉の深さの差が 1 で深い方の葉が木全体の左側に詰めている場合は**完全 2 分木**とみなされる。

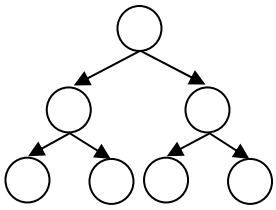


図 1.25　完全 2 分木

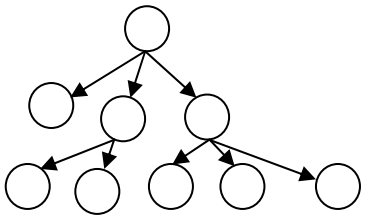


図 1.26　多分木

（３）木構造での探索

探索木はデータが一定の関係を保って格納されており高速で探索が可能である。構造が 2 分木の場合は **2 分探索木**と呼ばれ、データは「左の子 ＜ 親 ＜ 右の子」の関係を保っている。葉の深さの差が少ないバランスのとれた状態であれば、データを高速で探索することができる。

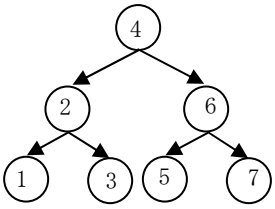


図 1.27　2 分探索木

幅優先探索(横型探索)　最上位階層(根)から始めて、同じ階層のデータを左から右にすべて巡回したら、さらに下の階層へと巡回していく方法。キューが使用され、巡回する順に参照が行われる。

下図の木構造では　4⇒2⇒6⇒1⇒3⇒5⇒7　となる。

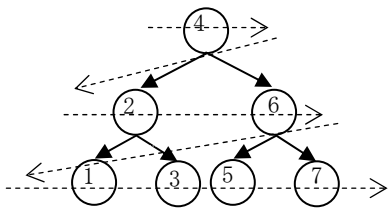


図 1.28　幅優先探索(横型探索)

深さ優先探索(縦型探索)　最上位階層(根)から始めて、左の部分木から順に葉までたどりながら巡回していく方法。スタックが利用され巡回の順番は１通りであるが、データの参照は優先順位のちがいにより次の３通りある。

先行順は「親」→「左の部分木」→「右の部分木」の順に参照する。

次の例では　4⇒2⇒1⇒3⇒6⇒5⇒7　となる。

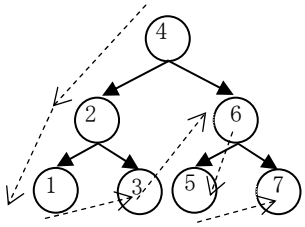


図 1.29　深さ優先探索(縦型探索)

後行順は「左の部分木」→「右の部分木」→「親」の順に参照する。

次の例では　1⇒3⇒2⇒5⇒7⇒6⇒4　となる。

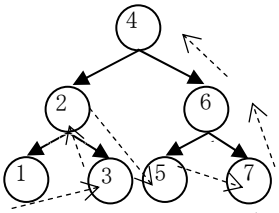


図 1.30　深さ優先探索(縦型探索)

中間順は「左の部分木」→「親」→「右の部分木」の順に参照する。

次の例では　1⇒2⇒3⇒4⇒5⇒6⇒7　となる。2 分探索木で中間順では整列された順に参照される。

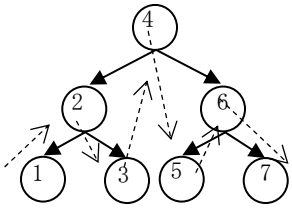


図 1.31　深さ優先探索(縦型探索)

１．３．６　グラフ

（1）グラフの概要

グラフは要素とそのつながりを表現したもので、つながり方に対して特に制約がなく複雑な要素間の関係を表現できる（木構造はグラフの特別な形態と考えることができる）。都市間の道路網交通機関の路線図、電気回路網などを表現できる。

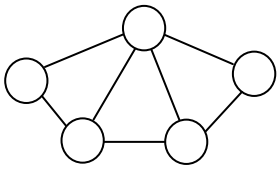


図 1.32　グラフ

上図のとおり、要素と要素間のつながりだけを表現したものを**無向グラフ**という。つながりだけでなく「どの要素からどの要素につながっているか」つながりに方向性を持っているグラフを**有向グラフ**という。

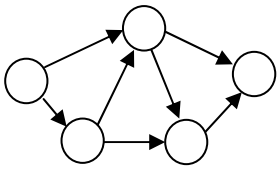


図 1.33　有向グラフ

無向グラフ、有向グラフの、つながりに距離や日数などの重みをつけたグラフを**重みつきグラフ**という。

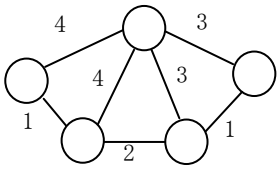


図 1.34　重みつきグラフ

（２）グラフでの探索

グラフの探索ではまず基準となる開始要素を決める必要がある。探索の種類には**幅優先探索**、**深さ優先探索**や、重みつきグラフで要素から要素までの最短経路を探索する**最短経路探索**がある。

幅優先探索は開始要素から始め、隣接するすべての要素を探索する。さらにこれらと新たに隣接するすべての要素に対して同様の処理を繰り返しながら探索を行う。開始要素に近い要素から探索されることになる。処理にはキューが使用される。

下図のグラフでＡから始めてＤを探索すると　A⇒C⇒B⇒E⇒D　の順に探索される。

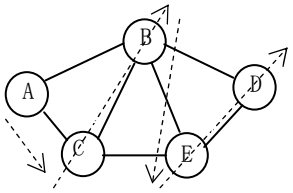


図 1.35　幅優先探索

深さ優先探索は開始要素からひとつの一連のつながりを行けるとこまで探索し、行き詰まったら直近の分岐点まで後戻りして、同様の処理を繰り返す。

下図のグラフでＡから初めてＤを探索すると　A⇒C⇒E⇒D　の順に探索される。

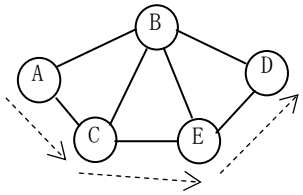


図 1.36　深さ優先探索

重みつきグラフの最短経路探索のひとつに**ダイクストラ法**がある。特定の１つの要素から他の全要素までの最短経路を求める方法である。この方法は開始要素から始め、隣接する要素の距離を順次累積する。複数の経路を経る場合は最短距離の経路を選択し遠回りとなる経路は切り離す。これを繰り返し各要素までの最短経路と最短距離を決定していく。

下図のグラフでＡから他の要素までの最短経路と最短距離を求める。×は遠回りとなる経路である。

Ａの距離を０とし、Ａとつながっている Ｂ と Ｃ までの距離を求める。比較すると A-C の経路が最短である。Ｂ経由でＣに至る経路は遠回りとなるため B-C は切り離す。

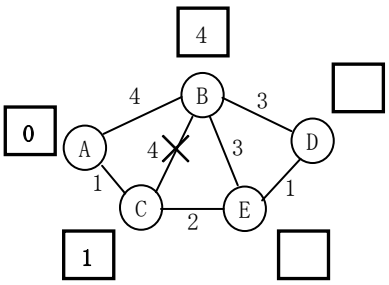


図 1.37　最短経路探索　その１

次にＣとつながっている Ｅ までの距離を求める（Ｂ経由でＥに至る経路は遠回りとなるため B-E は切り離す）。さらにＥとつながっている Ｄ までの距離を求める。Ｂ経由でＤに至る経路は遠回りとなるため B-D は切り離す。これでＡからすべての要素までの最短経路と最短距離が求められた。

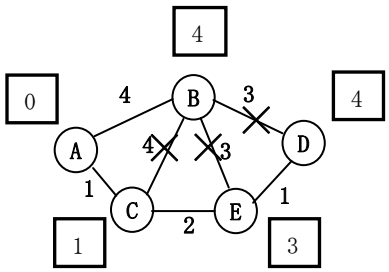


図 1.38　最短経路探索　その２