第7章
データ構造

データ構造の分類／構造体／共用体

C言語ではさまざまなデータ構造を扱うことができます．またデータの構造の違いにより，その取り扱い手法も変わります．目的のプロジェクト，あるいは情報の集合に対して，どの取り扱いが適しているのか，それにはどの構造がよいのかを充分検討したうえで，決定しなければなりません．そのためには，C言語が取り扱いできる構造を理解しておく，取り扱いの手法を理解しておく，そしてその特質を理解しておくことが必要になります．この章ではデータ構造の一般事項を整理したうえで，構造型，共用型について解説します．

7.1 データ構造の分類

図7.1.1にデータ構造を分類した体系図を示します．基本データ構造と問題向きデータ構造に大別できます．

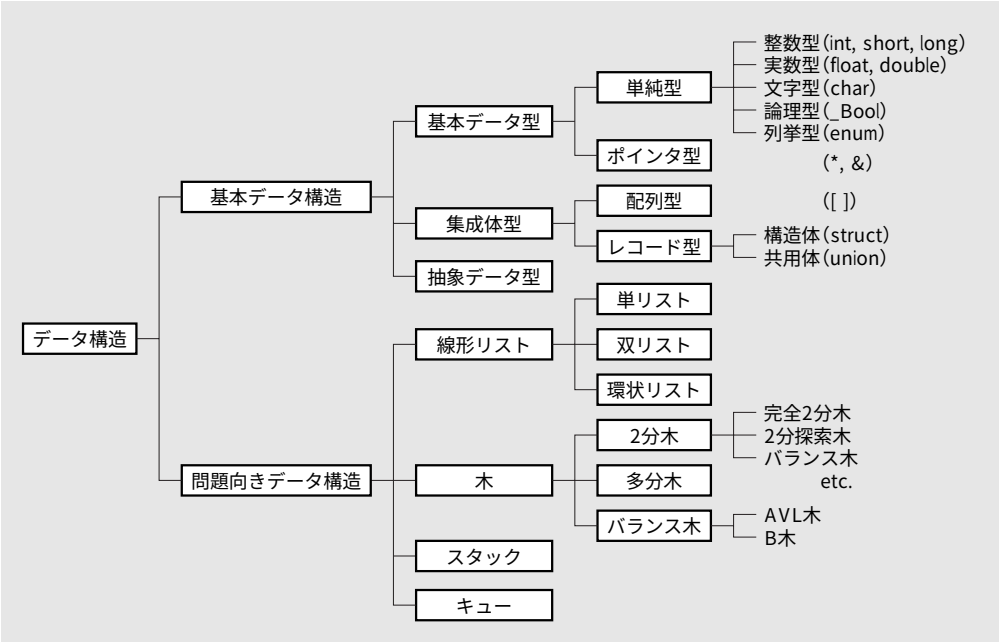

図7.1.1　データ構造の体系

7.1.1　基本データ構造

データの基本単位となっているもので，一部を除いてほとんどの言語であらかじめ準備されている基本の構造です．基本データ構造は，さらに基本データ型，構造型，抽象型に分類できます．

7.1.2　基本データ型

基本データ型(basic data type)は構造をもっていないデータ型です．つまりデータが単体として独立している形式です．基本データ型は単純型とポインタ型に分類できます．

■ 単純型

データそのものを表現する，表現できる型を単純型(simple type)といいます．データがどのような形式で，どの大きさまで収容できるかによって整数型，実数型，文字型，論理型，列挙型に分類されます．

■ ポインタ型

ポインタ(pointer)とは指標のことであり，ポインタ変数は目的のデータのあり場所を指している変数です．目的のデータというのは，通常メモリの内部にありますので，それを指し示すということは，メモリのアドレスを保持している変数ということになります．このようにデータのあり場所を格納する型をポインタ型(pointer type)と呼びます．詳細は第4章を参照してください．

7.1.3　構造型

データが単一のもので構成されているのではなく，複数のデータで構成されているものを構造型(structured type)と呼びます．構成要素が同一の性格をもつものであるか，複数の異なる性格のものを集めたものかによって，配列型とレコード型に分類されます．

■ 配列型

同じ型の複数の要素から構成されている1次元あるいは多次元の並びを配列(array)と呼び，この型を配列型(array type)と呼びます．概要は2.4節で解説しました．多次元の配列構成も可能です．

■ 構造体(レコード型)

前述の配列という構造は，構成する要素は全て同じ型のデータでなければなりません．ここで紹介する構造体(structure type)は，型の違うデータや大きさの違うデータも一緒に塊にできるというものです．配列型を含めて集成体型(aggregate type)と呼びます．

たとえば，会員の名簿を作るという場合にも，名簿のデータとして，

①会員番号
②会員氏名
③郵便番号，住所
④etc.

などの情報があります．会員番号は整数で取り扱うのが簡便ですが，氏名や住所は文字(あるいは文字列)の形で取り扱わなければならないでしょう．そして氏名と住所では，その文字列の長さも違ってくると思われます．

現実の問題として，データの塊は，このように型も大きさも違うものです．この類のデータを扱いやすくしたのが構造体です．実際のデータ処理においても，構造体の利用は多くあります．そして構造体を構成する各データをC言語ではメンバ(member)と呼んでいます．一般には構造体は一つのレコード(record)の単位であり，メンバはレコード内の各フィールド(field)に対応させることができます．詳細については7.2節で解説します．

□ 構造体の配列

構造体を複数個並べて，一般の変数のように配列にすることもできます．

■ レコード型(共用体)

構造体と似ているのですが，領域を共通に利用する共用体(union)という構造もあります．

7.1.4　抽象データ型

データの構造と，それに対する操作・処理をまとめることを抽象化といいます．抽象データ型（abstract data type；ADT）というのは，抽象化したものをデータとして定義したもののことです．抽象データを使う側は，内部の個々のデータがどういう形式なのか，どういう構造なのかということ，そしてどういう操作がなされているのかなどを意識する必要はありません．関数または手続き（procedure）として，抽象データの型名を指定して呼び出すだけです．

このような抽象データ型は，オブジェクト指向プログラミングの基本となる考え方ですが，C言語そのものでは，組み入れられてはいません．むしろC言語のアプリケーションの範疇で扱われます．C++では定義されています．

■ 情報隠蔽

ソフトウェア設計の考え方に，設計時の決定事項を隠してしまい，最小限の情報だけを外部に示すということがあります．これを情報隠蔽（information hiding）などと呼びます．

一般に情報隠しというのは悪いことと思われがちですが，プログラム・モジュールの独立性を高める，共同開発や保守がしやすい，隠蔽部分を変更しても利用者に影響を与えない，などのメリットもあります．狭い意味では，4.9節で解説した変数のstatic指定なども情報隠蔽の一種です．

■ カプセル化

情報隠蔽をデータ型のレベルで行うことをカプセル化（encapsulation）ということもあります．薬剤を混合して飲みやすくしたカプセル（capsule）と同じです．前述の抽象型データのように，データの内部や操作内容を利用者には見せないで，ユーザにはカプセル化されたモジュールを使うことだけに専念させることができます．データベースや汎用コンピュータの業務用ソフトなどに見られます．これもC言語そのものとしては，アプリケーションの範疇です．

7.2　構造体

実際のデータ処理の際に，データの実体として扱うのは構造体データが多いことは前述のとおりです．この節では構造体の仕組み，実際の利用方法等について解説します．

7.2.1　構造体の中身

構造体は，一般的にいえば型の異なるオブジェクトの集合です．前述のような会員名簿で整数型の会員番号，文字型配列の氏名，そして文字型配列の住所などを一つの塊として定義する例をみてみます．後述の方法により構造体としての定義をしますと，**図7.2.1**に示すように，まとまった領域に，連続に収納スペースが確保されます．処理系によっては奇数バイトの境目には，詰め物が置かれることもありますが，メモリの上にも連続の領域がとられます．このように構造体は，型の異なるデータでも一つのグループとして，まとめた取り扱いができる構造です．

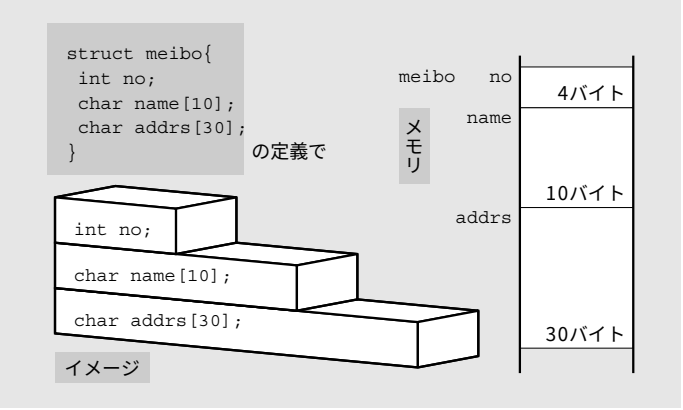

図7.2.1　構造体

7.2.2　構造体の定義

構造体もデータ格納の器であり，代入・参照の対象です．つまりオブジェクトですので，利用前にその型を定義しなければなりません．定義にはキーワードstructを使い，

```
struct  タグ名 {
  型  メンバ名;
  型  メンバ名;
  …
} 変数名;
```

とします．これでタグ(tag)名のオブジェクトが定義できます．

構造体のメンバには，一般の変数を置くことができます．配列を置いてもかまいませんし，ポインタ変数でも，さらに構造体でもかまいません．いわゆるすべてのオブジェクトを置くことができます．それぞれの所要するバイト数が連続に確保されます．具体例は後の章にも出てきます．

また，このオブジェクトを一つのまとまった型として定義することもできます．型として定義し，型に名前を付ければ，以下のプログラムの中で何度も利用することができます．新しい型の定義にはキーワードtypedefを利用します．その場合には，

```
typedef struct {
  型  メンバ名;
  型  メンバ名;
  …
} 構造体型名 ;
```

とすることにより，定義した構造体型名で以下何度でも，構造体型のオブジェクト(変数)の定義に利用することができます．さらにここにタグ名も記述できますので，型宣言と変数実体と同時に定義することもできます．具体的には，

```
typedef struct {
  int    no;
  char   name[20];
  char   addrs[30];
} MEIBO;
```

と定義すると，MEIBOという構造体型(structure type)が定義されたということです．この構造体型について，構造体変数そのものを定義するのには，

```
MEIBO  meibo;
```

とします．これは 型　変数名 と同じ形式ですから，構造体変数の実体が確保されます．詳細や応用については後述しますが，この配置だけは次の例題で確認しておきます．

例題7.2.2　構造体を定義し，各メンバの先頭アドレスを確認する．

上述と同じ構造体で確認します．各メンバの領域の先頭アドレスは，整数変数では変数名に&を付加して，文字配