§. 1. ネットワーク・コンピュータの基本用語

この節では，ネットワークを介して計算サーバー (後述) を利用する際に必要となる，計算機およびネットワークの基本的知識を，用語を中心に解説する．ここでネットワークとは，複数の計算機を配線で結合し，それぞれの計算機のデータの交換を可能とした仕組みを言う．

1．1．端末 (コンソール)

端末 (**コンソール**) とは，キーボード，マウス，モニター等の入出力装置を言う．計算機内部およびネットワークを介した他の計算機に対して，これら入出力装置は，ひとつの終端 (図 1. 1 参照) であるのでこの名称を用いる．手元の計算機全体を指さない．

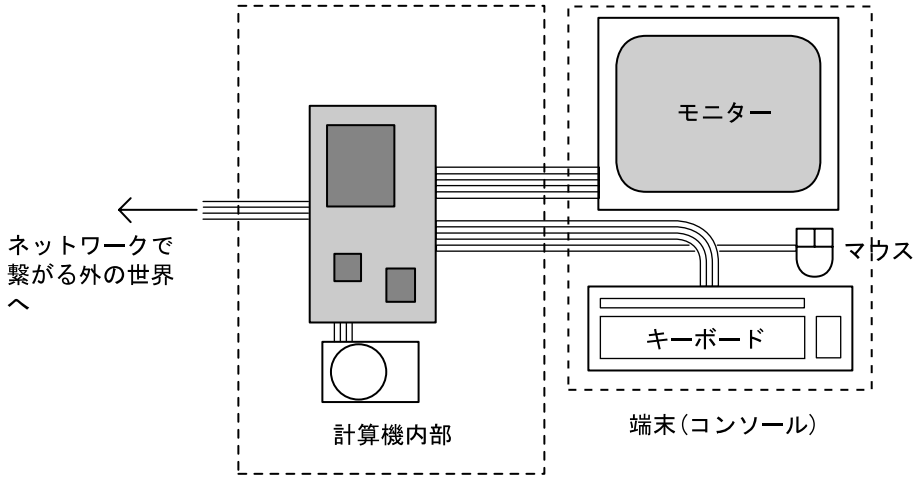

図 1. 1.

1．2．ホスト

すべての計算機一つ一つを，**ホスト**と呼ぶ．計算機利用者にその計算機がサービスを提供するので，計算機はホストと呼ばれる．

ネットワーク上の利用者からの “距離” により，さらに細かい分類がある．

1．2．1．ローカル・ホスト

手元にある，その端末を操作している計算機を**ローカル・ホスト**と呼ぶ．

1．2．2．リモート・ホスト

ネットワークを介してそのサービスを利用している計算機を，**リモート・ホスト**と呼ぶ．例えば，現在操作中の計算機の隣にあっても，その利用がネットワー

ク経由であり，その端末から直接に操作していない計算機は，リモート・ホストである．日本語訳は “遠隔計算機” であるが，物理的な遠近を言うのではない．

1．2．3．サーバー

サービスの供受の側面に注目したホストの分類もある．ネットワークを介してサービスを提供するホストを，**サーバー**と呼ぶ．

1．2．4．クライアント

一方，ネットワークを介してサービスを受けるホストを，**クライアント**と呼ぶ．

1．3．ワークステーション

処理能力の高い計算機は非常に高価であり，多人数で共同利用される．計算機には，部外者に知られては不都合な情報が保存されている場合もあるし，また，部外者が正規の共同利用者の許可なく計算機を使用すると，計算機を故障させる心配もある．そこで，共同利用される計算機には管理者が専任され，計算機の管理を行なうことが多い．

管理者がその計算機を利用できる者を事前に登録し，多人数利用に供される計算機を**ワークステーション**と呼ぶ．ワークステーションは，利用登録のない者は不正な方法を用いない限り，自身を利用できない仕組みを持っている．Microsoft 社の Windows NT の登場以前は，RISC 型 CPU を搭載し UNIX を OS とする計算機を，ワークステーションと呼んだ．現在では，上記の利用形態に注目した概念に変化している．

1．3．1．認証 (**ログオンまたはログイン**)

ワークステーションを利用するには，まず，正規の共同利用者 (登録ユーザーと呼ぶ) である確認を受けなければならない．このワークステーション (管理者ではない) による確認手続きを，**認証**と呼ぶ．認証は，端末の指示に従って，管理者から与えられた**ユーザー名** (**ログイン名**) をキーボードから入力し，続いて**パスワード**を入力することで行う．ワークステーションは，利用者の入力したユーザー名・パスワードが正しいものであることを確認したら，利用者にコマンド入力を許す．パスワードは，初めての利用の際には，管理者から与えられたものを利用し，一度認証を済ませてから，利用者個人で変更する．パスワード入力中は，端末モニターに入力がエコー・バックされない．すなわち，キーボードから入力した文字列が，モニターに表示されない．

1．3．2．リモート・ログイン

ネットワーク上のワークステーションであるリモート・ホストへ，ネットワークを介して，ローカル・ホストからログインすることを**リモート・ログイン**と呼ぶ．この時，ローカル・ホストの端末を (ワークステーションの) **仮想端末**と呼ぶ．

1．4．リモート・ホストのサービスの種類と実現の仕組み

ネットワーク上，サーバーはクライアントに向けて様々なサービスを提供している．ワークステーションの利用の場合のような認証を，必要としないサービスがその大半であり，日々拡大するインターネット上では，不特定多数のクライアントがそのサービスを利用する．サーバーもまた，日々その数を増加させている．

サービスの提供・利用の仕組みには，統一規格が存在し，この規格に準拠したハードウェアー・ソフトウェアーを各ホストが実装することで，インターネット上の各ホストの個別独立な行動にもかかわらず，サービスの供受に不整合をきたしていない．ソフトウェアーに関するこの規格を**プロトコル**という．

1. 4. 1. サービスの種類と準拠プロトコル

規格は，個別の具体的サービスに対応する最上層の規格から，それらに共通の基底的規格へと階層構造をなしている．ホームページの閲覧，E-メールの送受等の具体的サービスに関する規格は，一階層下のトランスミッション・コントロール・プロトコル/インターネット・プロトコル (TCP/IP) と呼ばれる基底的規格の上に位置している．以下，代表的サービスに関して，その準拠プロトコル，サーバー・ソフト，クライアント・ソフトを表にまとめる．ここで，サーバー・ソフトとは，サーバーがそのサービスの提供を実現する為のソフトを意味し，クライアント・ソフトとは，クライアントがそのサービスを受ける為のソフトを意味する．

サービス	準拠プロトコル	サーバー・ソフト (デーモン)	クライアント・ソフト
ホームページの閲覧	Hypertext Transfer Protocol HTTP	Apache(httpd)	Internet Explorer
ファイルの送受	File Transfer Protocol FTP	wu-ftpd (wu-ftpd)	FFFTP 等
E-メールの送受	Simple Mail Transfer Protocol Post Office Protocol 3 SMTP, POP3	qmail, postfix 等 (smtpd, pop3d)	Almail 等
リモートログイン	telnet	telnetd(telnetd)	TeraTerm 等

1. 4. 2. デーモン (daemon)

サーバーにおいて，クライアント (サーバー自身の端末からの操作を含む) のサービス提供要求に対して，自動応答する常駐プログラム (“常駐” とは，ホストの主記憶上に常にロードされて (書き込まれて) いることを言う) を**デーモン**と言う．つまり，クライアントからのサービスの提供要求には，サーバーであるホストの管理者が応答するのではなく，サーバー自身が応答する．上記表において，wu-ftpd, telnetd はそれを実現する固有名詞を持つソフトウェアーは存在せず，デーモン名がプログラム名となっている．

1. 4. 3. プログラム演習の実際

大学のプログラム演習などで，実際に行なっているホストの利用法を，今まで

に説明してきた用語を用いてまとめておこう．数値解析を実行する為のホストである，計算サーバーがネットワーク上に準備されている．この計算サーバーは，リモートログインによって，ローカル・ホストの端末を仮想端末として利用する．当然ながら，このサーバーは多人数利用のワークステーションである．学生は管理者である教師によって，このワークステーションに登録されている (**アカウントを持つ**，とも言う)．このリモートログインは，telnet と言うプロトコルにしたがった，サーバー・ソフト，クライアント・ソフトにより実現されており，学生はローカル・ホストに設定されている TeraTerm 等のソフトを利用することになる．

リモート・ログインによりリモート・ホストを利用する場合，２節以降で説明するソース・ファイルの編集，コンパイルといった作業を行なっているのは，今現に操作しているローカル・ホストではなく，リモート・ホストである．ローカル・ホストはその端末にリモート・ホストの送信するデータを表示する働きをしているだけである．

§．2．実行型ファイルの構築

2．1．ファイルの種類

この節では，実行型ファイルと呼ばれる，計算機を動作させる＋・－の列からなる情報のひとまとまりを作成する過程の概略を述べる．計算機で処理される情報 (信号) のひとまとまりを，**ファイル**と呼ぶ．ファイルは，電圧のプラス，マイナスの２値の列で記述され，磁気ディスク等に保存される．ファイルの磁気ディスクへの保存は，プラス，マイナスの情報を，磁気の N 極，S 極を用いて記述することで行なわれる．ファイルにおいて，通常の文章の単語に当たる，＋・－２値列の単位を**コード**と呼ぶ．市販されている，いわゆるアプリケーション・ソフトウェアーは，関連しあう複数のファイルから構成されており，前述の磁気媒体または光学記憶媒体 (CD-ROM) に保存して (書き込んで) 販売されている．

計算機が扱うファイルには，ソフトウェアーのように計算機を制御する (動作させる) コードからなるものと，単なる文字情報のみからなるものの２種類がある．前者を**実行型ファイル**，後者を**テキスト・ファイル**と呼ぶ．実行型ファイルは，OS[1]によってそれが主記憶上に読み込まれる (ロードされる) と，その情報 (コード) が順次，中央演算処理装置 (CPU) で処理され，計算機を “動かす”．一方，テキスト・ファイルは，文字情報のみからなるファイルで，主記憶上に読み込まれても，それ自身で計算機を制御しない．計算機内では，アスキー・コードと呼ばれる文字と数値 (１６進数) との対応付けの規則にしたがって，文字は，数値に置き換えら

[1]オペレーション・システム．一般のソフトウェアーの動作を管理・制御する，いわば一階層基底的なソフトウェアー．一般ソフトウェアーへ，計算機利用者による入力装置を介した命令を伝達したり，一般ソフトウェアーの入出力装置の利用要求を処理する．

れるが，テキスト・ファイルは，アスキー・コードにしたがった文字情報のみからなるファイルである．

テキスト・ファイルの対義語は，正確には**バイナリー・ファイル**で，実行型ファイルは，バイナリー・ファイルの一つの種類である．実行型ファイルではないバイナリー・ファイルも存在する．例えば，近年，ネットワークで頒布される (図版を含む) 文書の形式として主流を占めつつある PDF ファイルなどがその代表である．PDF ファイルは，図版を含まない場合，文字情報のみから成るが，アスキー・コードからなるファイルではなく，テキスト・ファイルとは呼ばない．

2．2．コンパイラー

実行型ファイルは，マシン語と呼ばれる制御コードをじかに編集することで，直接に作成可能であるが，作業効率が極度に悪い．そのため，自然言語に近い人間の理解しよい文法にしたがって記述されたテキスト・ファイルを，実行型ファイルに変換するソフトウェアーが開発され，広く利用されている．これらソフトウェアーを**コンパイラー**と呼ぶ．コンパイラーは，実行型ファイルを生成する元となるテキスト・ファイルを記述する言語 (または，文法．**プログラム言語**と呼ばれる) とともに開発され，C 言語，FORTRAN，PASCAL 等の言語に対応して，それぞれ，C コンパイラー，FORTRAN コンパイラー，PASCAL コンパイラー等がある．

CPU の命令と１対１に対応付けられた，自然言語の単語からなるテキスト・ファイルを，実行型ファイルに変換するソフトウェアーは**アッセンブラー**と呼ばれ，コンパイラーとは区別される．コンパイラーでは，ひとつの命令 (単語) から，組織的に一連の制御コードを生成し，アッセンブラーに比べ，実行型ファイル作成の効率がよい．

2．3．実行型ファイル構築の手順

まずは，コンパイラーの文法にしたがって，実行型ファイルに変換するためのテキスト・ファイルを作成する．このテキスト・ファイルを**ソース・ファイル**と呼ぶ．次に，作成したソース・ファイルをコンパイラーを用いて，実行型ファイルに変換する．つまり，実行型ファイルの構築には，ソース・ファイルを作成する**エディター**と呼ばれるソフトと，コンパイラーの２つが必要である．

2．3．1．エディター

テキスト・ファイルを作成するソフトウェアーをエディターと呼ぶ．エディターには多くの種類があるが，出来上がるテキスト・ファイルは，すべて全く同じである．利用者の使用感が違うだけで，各人の好みに応じて多くのエディターの中からひとつを選べばよい．エディターを用いて，テキスト・ファイルを作成する，また，すでに作成されたファイルを修正することを，**テキスト・ファイルを編集する**，と呼ぶ．雑誌等の編集作業は，“多くの” 原稿を一冊の本にまとめることを言うが，コンピューター用語では，“ひとつ” の原稿 (ファイル) の作成・修正作業

を指す.

UNIX では，**Emacs**(または，多種の言語，つまり，日本語，韓国語等に適応した変種である **Mule**) が，一般的である．また，UNIX 上必ず存在するエディターに **vi** がある．Emacs に比べ，機能面・操作性ではるかに劣るが，OS の障害時等 Emacs が使用不能の状況でも利用可能であることが多く，必ず使用法をマスターしておくべきエディターである．

2. 3. 2. コンパイル＋リンク＝メイク (構築)

アッセンブラーによって，ソース・ファイルから作られる実行型ファイルを，“目的のファイル” という意味で**オブジェクト・ファイル**と呼ぶ．コンパイラーにとっても，実行型ファイルは目的のファイルであるはずであるが，コンパイラー使用時のオブジェクト・ファイルは実行型ファイルとは異なる．

“ソース・ファイルをコンパイルする” という場合，通常，狭義のコンパイルとリンクの２つの操作を続けて行うことを指す．このより細分化された操作に対応して，通常広義にコンパイラーと呼んでいるソフトウェアーは，狭義にはコンパイラーとリンカーの２つからなる．例えば，C コンパイラーの代表的頒布ソフトである gcc 等は，この２つの操作を分けて実行することも可能であるが，ソフトウェアーとしては単一のソフトウェアーに統合されたもので，通常，単に C コンパイラーと呼ばれる．

狭義のコンパイルによって生成されるファイルを，オブジェクト・ファイルと呼ぶ．確かに，狭義のコンパイラーにとっては，これが目的のファイルとなっている．オブジェクト・ファイルは，実行型ファイルの元となる，いわば，“中間生成物” で，実行型ファイルの骨格または設計図に相当する．この骨格・設計図に実行型ファイルの既製の部品 (これを**ライブラリー**と呼ぶ) を組み込む，または，組み込むための組み込み口を作る操作を**リンク**と呼ぶ．ライブラリーの組み込みは，実行型ファイルのファイル・サイズを小さくする目的で，その実行型ファイルの実行時に主記憶上で行うことがほとんどである．つまり，実行型ファイルそのものに，一体のファイルとしてライブラリーは組み込まれない．完全に独立な実行型ファイルを生成する必要があるときは，ファイルとして組み込みことも可能で，この操作を**静的にリンクする**と呼ぶ．PC/DOS 上の C コンパイラーの場合，OS の性格上，リンクは必ず静的に行われ，UNIX では，通常，実行時組み込み型となっている．Windows も UNIX に習い，ソフトウェアーの実行時に，Windows 自身が持っている，さまざまなソフトウェアーに共通の実行時部品 (ダイナミック・リンク・ライブラリーと呼ばれ，ファイル名は *.dll) を組み込んでいる．コンパイルとリンクの操作をあわせて，**メイク** (**構築**) と呼ぶ.

§. 3. C 言語の基礎

この節では，C 言語を，関数，ポインターに焦点を当てて解説する．文法を順を追ってすべて解説することは止めて，C 言語の長所であり，最初 (最大？) の難関であるポインターの理解に力点をおく．一般的な文法の基礎は [1] などを参照されたい．[2] や，開発者自身による解説書 [3] 等も適宜参照されたい．

プログラム言語の文法にしたがって，ソース・ファイルを編集することを，**プログラムを書く**，と言い，ソース・ファイル自体をプログラムと言う．C 言語のプログラムは，関数と関数を組み合わせた制御構文からなる．

3．1．関数 (１)

まず，C 言語における関数について説明する．C 言語の関数は，数学の関数と類似点も持つが，異なる点も多い．

3．1．1．類似点

数学の関数 $y = f(x)$ の独立変数 x，従属変数 y に対応して，C 言語の関数は，それぞれ，**引数**，**戻り値**を持つ (正確には，引数は，独立変数に代入される ‘値’，戻り値は，従属変数に与えられる ‘値’ である．独立変数に対応する概念は，正確には，仮引数である．仮引数の詳細は 3.4.3 節で述べる)．

3．1．2．相違点

C 言語の関数は，引数または戻り値のどちらか一方，または両方を持たないことを許される．数学の関数は，出力である従属変数が重要であるが，C 言語の関数においては戻り値を返すまでの関数内部の処理そのものが重要な場合が多く，戻り値には２次的な重要性しかないこともある．

関数の名前は，数学の場合，概ね一文字であるが，C 言語では関数名からある程度実態が推測しやすいように，単語，またはその省略を用いることが多い．変数についても同様である (この点は，後でもう一度触れる)．

3．1．3．プログラムの骨格

プログラムの骨格は，**main** と名づけられた関数で，main 関数はソース・ファイルに

```
main()
{

                    **********
                      プログラムの本体
                    **********



     return 0;
}
```

と言う形式で記述する．

実行型ファイルの実行は，OS が main 関数を呼び出すことで行なわれる．main 関数は，引数を取っても，取らなくともよく，戻り値は通常，整数 0 である．この 0 は，実行型ファイルの実行が終了したことを示す為に，このソフトウェアを呼び出したソフトウェア，つまり OS，に戻される．`return 0;` が，この 0 を OS に戻すという命令を記述している．当然，OS に戻される値 0 には，その実行型ファイル自身の動作にとって，あまり重要性はない．

3．2．簡単なプログラム

プログラム例を一つ示し，プログラムの実際を解説する．

3．2．1．プログラム例 1．

```
#include <stdio.h>

main()
{
     printf("Hello, world\n");

     return 0;
}
```

このプログラムは，端末モニター上に `Hello, world` と表示し改行する，という動作を計算機に行わせる．関数 `printf` は `"  "` の中に書かれた文字列を，書かれたままに，標準出力 (端末モニター) に出力する．ただし，`\n` は改行を指示す命令であり，書かれたままには表示されない．`main()` は，() の中が空白であることで，この main 関数が引数を取らないことをあらわしている．`printf` に続き `return 0;` で OS に 0 を戻り値として返し，このプログラムは終了する．各行の末尾にはセミコロン ; をおく．

`printf(` ···，の前の字下げは，文法により要求されたものでなく，字下げは行なっても行なわなくともよい．また，次に続く空行の挿入も，行なっても行わなくともよい．極端な例として，`#include <stdio.h>` 以外の全てを

```
main(){ printf("Hello, world\n"); return 0;}
```

と 1 行に記述しても，文法上は問題ない．しかし，プログラムが複雑になればなるほど，この字下げや次の行の空行等が，プログラムの “見易さ” に威力を発揮する．通常，字下げは 5 文字前後であるが，特に決まりはなく，好みで調整してよい．

3．2．2．ヘッダ・ファイル

第 1 行の `#include <stdio.h>` について，簡単に解説する．かぎ括弧 < > の中の stdio.h はヘッダ・ファイルと呼ばれるファイル群の中のひとつのファイル名である．

ヘッダ・ファイルを説明する為に，まず，関数のプロトタイプを解説しよう．C 言語の関数は，3. 1 で述べたように，数学の関数の独立変数に相当する引数 (複数であってもよい．つまり，数学の関数の多変数関数に相当する関数を許す) と従属変数に相当する戻り値を持つ．C 言語は他のプログラム言語と同様に，後に解説するように，いくつかの変数の “型” を持つ．ここで変数は，一先ず，数学同様に数値の代入先と考えてほしい．数値が実数であるか整数であるかに応じて，その数値の代入先である変数は別の種類のものが用意されており，この変数の種類を変数 (の) 型 (後に詳述) と呼ぶ．関数の**プロトタイプ** (**型式**) とは，その関数が引数として，どのような型の変数たちを持ち，戻り値としてどのような型の変数を持つかを示したものを言う．

C 言語では，main 関数以外の関数を使用するには，main 関数の外で，その関数の**プロトタイプ宣言**を行わなければならない．プロトタイプ宣言とは，その関数の型式を一定の書式にしたがって，プログラムに書き込むことを言う．例えば，引数として，実数型，整数型，文字型 (文字も数値同様に，変数へ入力される “数値” として扱われる) の変数をそれぞれひとつずつ取り，戻り値として，実数型の変数を取る func と名づけた関数をつくり，それを利用する場合 (関数の構成の詳細は後述),

```
float func(float x, int n, char a);
```

と記述する．

このプロトタイプ宣言は，C 言語に元々標準で準備されている関数についても，平等に行わなければならない．ところが，毎回，標準に装備された関数にまで，プロトタイプ宣言を行うのは不経済なので，その代用として，ヘッダ・ファイルを**インクルード**する．つまり，プログラムの最初に

```
#include <stdio.h>
```

と記述する．ヘッダ・ファイルは標準装備関数をいくつかのグループに分けて，それらのプロトタイプ宣言を集めたもので，いわば，“プロトタイプ宣言集” である．

関数 printf は，文字型変数 (の配列) を引数 (独立変数) とし，main 関数に整数を返す．つまり，戻り値 (従属変数) の変数型は整数型である．この戻り値を戻す過程で，端末モニターに，引数として与えられた文字配列を描画する．

3．3．変数

前節まであいまいな定義のまま使用してきた，変数の概念を明らかにしよう．数値などのデータは，計算機の主記憶に書き込まれるが，そのためにはまず，主記憶上にデータを書き込む (格納する) 区画が，確保されなければならない．また，確保されたそれぞれの区画を区別する為に，何らかの識別子，つまりは名前を付け

る必要がある．さらには，データの格納・呼び出しの為の手段も必要である．プログラム言語において**変数**が，この場所の識別子，格納・呼び出しの手段である．主記憶上の格納区画そのものも，通常，変数と呼ばれる．つまり，変数 x に値 0.034 を格納する，などと言う言葉の使い方は，一般的である．格納区画の確保は，変数を**宣言**することで行う．変数宣言は以下に述べるように，変数型と変数につけた名前を，プログラムに書き込むことで行なう．

3. 3. 1. 変数型

計算機で扱うデータには，実数，整数，文字 (列) がある．実数は，計算機内での処理方法に由来する名称として，**浮動小数点数**と言う呼び名が与えられている．実数は，例えば，351.49837 の場合，

$$351.49837 = 0.35149837 \times 10^3$$

として，符号 + ，35149837，3(10 の肩の指数) に分解して記憶される．35149837 を**仮数**，3 を**指数**と呼ぶ．通常の浮動小数点数の場合，仮数部の桁数は 6 桁であり，

$$\{+,\ 351498,\ 3\}$$

の３つのデータの組として記憶される．つまり，仮数部は 6 桁に丸めて記憶される．指数部に記憶可能な数は，-37 から 38 の間の整数である．

データの種類が，浮動小数点数，整数，文字のいずれであるかに応じて，主記憶上の格納区画の大きさは異なる．例えば，浮動小数点数一つの場合，4 バイト (32 ビット) の記憶容量が必要であり，文字 (半角英数) 一つは，1 バイト (8 ビット) である．つまり変数は，格納されるデータの種類に対応して，それぞれ別の型が用意されており，**浮動小数点型**，**整数型**，**文字型**の３つに大別される[2]．そこで，扱うデータに応じて，異なる大きさの格納区画を確保する必要がある．この区画の確保，及び区画の識別子 (変数名) の登録のために，変数は，実際にデータの格納・呼び出し (**参照**と言う) に使用される前に，**変数型**を明示して宣言される必要がある．例えば，浮動小数点数を格納する区画を，変数名 x で確保する為には，

```
float x;
```

とする．変数名 x の前に float と記し，x が浮動小数点数型の変数であることを示す (他の型の変数の宣言は，順次，プログラム例の中で示す．一度に記憶しようとせず，解説の進行とともに徐々に学習する方法を採る)．

3. 3. 2. プログラム例 2.

変数宣言の実際と変数による数値の操作を，次の例で見てみよう．

[2]さらに詳細に区別された変数型が用意されている．参考文献 [1]，[2] を参照されたい．各型で扱える数値は，当然，無限ではなく決まった範囲の数値のみである．整数型の場合この範囲が意外に狭いので，一度確認しておくことを勧める．

```
[1] #include <stdio.h>
[2]
[3] main()
[4] {
[5]       int a, b, c; //変数宣言
[6]
[7]       a = 2; //変数 a に値 2 を格納
[8]       b = 4; //変数 b に値 4 を格納
[9]
[10]      c = a + b; //変数 a, b の値を参照し (呼び出し), そ
[11]                 //の値を足し合わせた結果を変数 c に格納
[12]
[13]      printf("c = %d\n", c);
[14]
[15]      return 0;
[16]}
```

[5] で整数型変数 a, b, c を宣言している．int と記すことで，これらが整数型の変数であることを示している．この記述で，主記憶上に a, b, c の名前で参照される整数を格納する区画が確保される．この例のように同じ型の変数を複数，宣言する場合，型名 (int) の後に変数名をコンマで区切って列挙する．

```
int a;
int b;
int c;
```

としてもよいが，かえって読みづらい．行末には，printf などを使用した場合と同じようにセミコロン ; をおく．

[7]，[8] は，変数 a, b (と名づけられた格納区画) へ，それぞれ，値 2, 4 を格納する操作を示す．C 言語において = は数学の等号ではなく，**代入命令**と呼ばれる命令である．また，[7]，[8] を**代入構文**と呼ぶ．= の左辺の変数に，右辺の値を格納する．見方を変えると，a, b は，主記憶内の a, b と名付けられた格納区画に対する操作のリモコンにもなっている．つまり，a, b と名付けた格納区画への数値の格納は，リモコン a, b に代入命令を施すことで実現される (図 3. 1 参照)．

次に [10] を見る．= の右辺にも変数がある．右辺の a, b は既に格納されている値を参照するリモコンである．右辺 a + b は，a, b と名付けられた格納区画へ格納されている値 2, 4 をレジスター (CPU 内の数値の一時格納場所．あらゆる操作は，いったんレジスターへ数値を呼び出し処理される) へ呼び出し，その値の和を計算し，結果をレジスターへの書き込む操作を示す (図 3. 2 参照)．= の右辺と左辺は非対称で，そこにおかれる変数の役割は異なる．= の右辺の数値または右

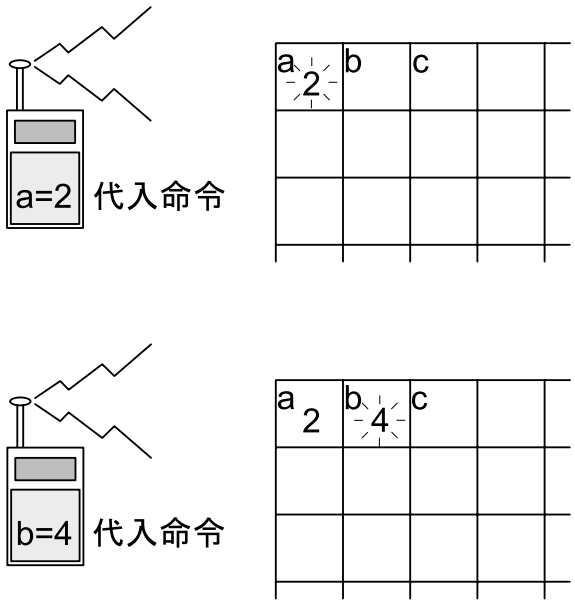

図 3. 1.

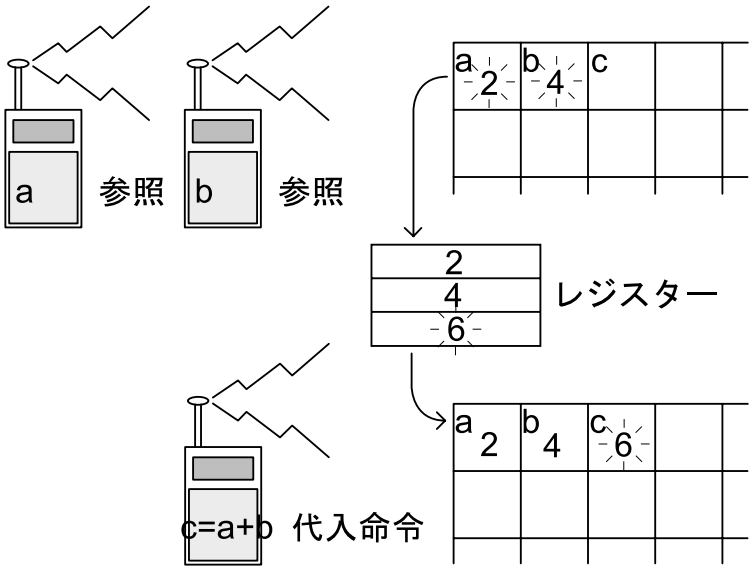

図 3. 2.

辺の指示に従って処理された数値が，左辺の変数に格納 (代入) される．したがって，例えば，`2 = a;` と言う記述は誤りである．[10] では，右辺の指示する操作によりレジスターに格納されている数値 6 が，左辺の変数 `c` に代入される．言い換えると，変数 `c` に代入命令を実行することで，`c` と名付けられた格納区画へ 6 が格納される (図 3. 2 参照)．まとめると，右辺の変数は，格納されている値を参

照し，左辺の変数は，値を新たに格納区画へ格納するか，既に格納されている値を更新する．したがって，= の右辺の変数には，必ず，既に値が格納されていなければならない．既に明らかであろうが，+ はその前後に示された２数または２つの変数に格納された数値の和を計算させる命令である．**二項演算子** と呼ばれる命令の一つである．引き算も通常の引き算と同じ記号 - を用いる．掛け算，割り算は *，/ を用いる．

[13] の printf は，プログラム例１と異なり " " に記述された文字をそのまま標準出力に出力しない．" " 内の %d はコンマの後に記述された変数 c に格納された値を受けて，その値を出力する．したがって，このプログラムの実行結果は

```
tateyama@toyama:~$ ./add
c = 6
tateyama@toyama:~$
```

となる．ただし，ホスト toyama のログイン名 tateyama のユーザーにより，このソース・ファイルは，実行型ファイル名を add としてコンパイルされている．add の実行は，安全上の理由から実行型ファイル名の前に ./ をつけて行なう．

関数 printf により変数に格納された値を出力する場合，その変数の変数型が，浮動小数点数型，整数型，文字型のいずれであるかに応じて，%f，%d[3]，%c を用いる．複数の変数の値を受けるには，" " の中に %f，%d，%c を必要な順に書き並べ，コンマの後に，対応する変数を同じ順にコンマで区切って列挙する．例えば，

```
printf("変数%c の値は%f であり，その整数部分は%d である\n", ch, x, xd)
```

において，ch，x，xd はそれぞれ文字型，浮動少数点数型，整数型の変数であり，" " に現れる %c，%f，%d の順に応じて，コンマの後に ch，x，xd の順に列挙してある．

3. 3. 3. 最も重要な例

次の例は，変数，代入構文の，数学との違いを端的に表した，最も重要な例である．

```
n = n + 1; または n = n + a; など
```

数学の等式の場合，上の第１式は明らかに $0 = 1$ に帰着し，誤った等式である．しかし，C 言語の代入構文の場合，正しい処理となっている．なぜなら，この構文は，この行以前の処理で，変数 n に格納されている値 (例えば，3) に 1 を加え，結果 (例の場合，4) を，再び変数 n に格納しなおす処理を意味しているからである (図 3. 3 参照)．n = n + a についても同様である．

[3]整数型の変数はその数値を，１０進数以外の進法で入出力できる．１６進法で出力したい場合，%d の替わりに %x を用いる．数値計算を行なう上で，１０進法以外の表記はあまり用いられないので，これ以上の解説は行なわない．[1]，[2] を参照．

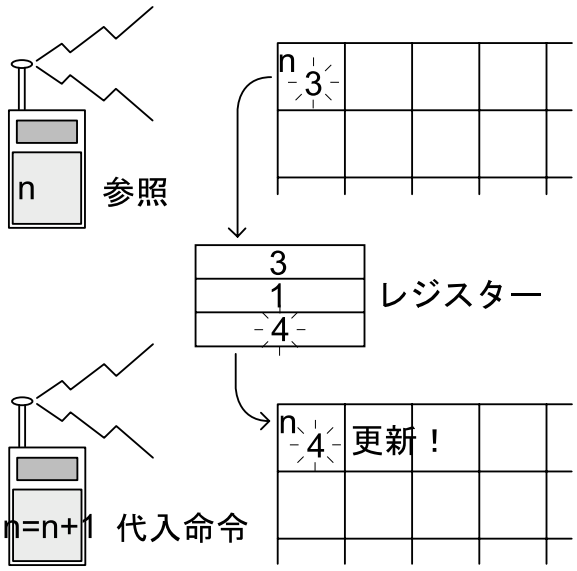

図 3. 3.

3. 3. 4. 変数名

数学の変数の場合は，その名前として，アルファベット１文字を当てることがほとんどであるが，C 言語の場合，なるべく，変数に格納される数値の内容が理解できるように，単語またはその省略が用いられる．変数名に使用可能な文字，文字数には一定の制限があるので，C 言語の文法書 [1]，[2] を参照のこと．

3. 4. 関数 (２)

ここでは，関数を自分で構成し使用するプログラムを書いてみる．C 言語のプログラミングにおいては，いわゆるトップ・ダウン式のプログラミングを実行するが，関数の構成に併せてこのトップ・ダウン式プログラミングも解説する．

3. 4. 1. プログラム例 3.

```
[1]  //２つの数の四則演算を，メニューで選択して実行するプログラム
[2]  #include <stdio.h>
[3]
[4]  //プロトタイプ宣言
[5]  int   choice(void);
[6]  float add(float x, float y);
[7]  float minus(float x, float y);
[8]  float multiply(float x, float y);
[9]  float divide(float x, float y);
[10]
```

```
[11] main()
[12] {
[13]      //変数宣言
[14]      float a, b, c; //四則演算を行なう数値及びその結果を受ける
[15]      int   ch;      //四則演算の選択結果を受ける
[16]      char  cal;     //選択された演算の種類を記憶
[17]
[18]      //２つの数値の入力要求
[19]      printf("----- ２つの数の四則演算を行います．-----\n");
[20]      printf(" ２つの数 a, b を入力してください．\n");
[21]      printf("         a =  ");
[22]      scanf("%f", &a);
[23]      printf("         b =  ");
[24]      scanf("%f", &b);
[25]
[26]      //演算の種類の選択要求
[27]      ch = choice();
[28]
[29]      //ch に応じて，演算処理を選択する
[30]      switch(ch){
[31]      case 1: //足し算
[32]           c = add(a, b);
[33]           cal = '+';
[34]           break;
[35]      case 2: //引き算
[36]           c = minus(a, b);
[37]           cal = '-';
[38]           break;
[39]      case 3: //掛け算
[40]           c = multiply(a, b);
[41]           cal = '*';
[42]           break;
[43]      case 4: //割り算
[44]           c = divide(a, b);
[45]           cal = '/';
[46]           break;
[47]      default: //choice での入力ミスに対応
[48]           printf("選択が誤っています．プログラムを終了します．\n");
[49]           return 0;
```

```
[50]      }
[51]
[52]      //結果の表示
[53]      printf("計算結果は：");
[54]      printf("%f %c %f = %f\n", a, cal, b, c);
[55]
[56]      return 0;
[57] }
[58]
[59] int choice(void){
[60]      int c;
[61]
[62]      printf("演算の種類を選んでください. \n");
[63]      printf("対応する数字を入力してください. \n");
[64]      printf("     1 : 足し算\n");
[65]      printf("     2 : 引き算\n");
[66]      printf("     3 : 掛け算\n");
[67]      printf("     4 : 割り算\n");
[68]
[69]      scanf("%d", &c);
[70]
[71]      return c;
[72] }
[73]
[74] float add(float x, float y){
[75]      float z;
[76]
[77]      z = x + y;
[78]
[79]      return z;
[80] }
[81]
[82] float minus(float x, float y){
[83]      float z;
[84]
[85]      z = x - y;
[86]
[87]      return z;
[88] }
```

```
[89]
[90] float multiply(float x, float y){
[91]      float z;
[92]
[93]      z = x * y;
[94]
[95]      return z;
[96] }
[97]
[98] float divide(float x, float y){
[99]      float z;
[100]
[101]     z = x / y;
[102]
[103]     return z;
[104]}
```

3. 4. 2. プログラムの概略

プログラム例 3 は，２数の四則演算を行うプログラムで，main 関数は５つの関数とその呼び出し，制御，入力要求また結果の表示からなる．プログラムを作る基本的考え方は，まず，望ましい機能を持つ関数を想定し，その実際の構成は最初には考慮せず全体の構造を作り上げ，その後に各関数の具体的なプログラムを組み上げる，というものである．このように，概略を構成しその後に，各関数を具体的に構築する方法を，**トップ・ダウン式のプログラミング**と呼ぶ．逆に，まず，各関数を設計し，それらを集めて全体を構成する方式を，**ボトム・アップ式のプログラミング**と呼ぶ．C 言語によるプログラミングは，トップ・ダウン式のプログラミングでその本領を発揮し，このことは C 言語が，多人数による分散作業に適していることを意味している．

[1] の // は，その後の次に改行されるまでの入力を，コンパイラーに無視するように指示している．// はプログラムの中身には関係ないが，メモ代わりの覚書をプログラム中に書き込むために用いる．この覚書は，プログラマー本人にとってもプログラムを見直す際に役に立つし，第３者にとっては，そのプログラムを解析するためのヒント代わりにもなる．[1] では，プログラムの概略が手短に説明されている．

[5] から [9] が，関数のプロトタイプ宣言である．これら５つの関数は，それぞれ次のような機能が想定されている．まず，choice は，２つの数に４則演算のどの演算を施すかの選択を，キーボードからの入力で行なうことを要求し，入力された選択を，それが足し算，引き算，掛け算，割り算であることに応じて，整数 1, 2, 3, 4 を戻り値として返す．引数はとらない．したがって，プロトタイプ

は int choice(void) となる．void は引数をとらないことを示す．次に add は，自身が main の中で呼び出される以前に入力済みの，２つの実数を引数に取り (言い換えると，“２つの実数を入力されて”)，その和を返す. 結果である和も当然実数であるので，プロトタイプは float add(fload x, float y) となる．minus，multyply，divide も add と同様に，入力済みの２つの実数の，それぞれ，差，積，商を返す．プロトタイプについても同様である．

[22] の関数 scanf は標準入力 (キーボード) からの入力を，指定された変数に代入する関数である．そのプロトタイプは printf 同様，stdio.h の中に含まれている．[22] では，変数 a への入力が行なわれる．" " の中には，入力先の変数が浮動小数点数，整数，文字に応じて，%f，%d [4]，%c を記述する．入力先の変数は，続いてコンマで区切って& をつけて記述する ([22] では a であるので，&a としてある)．例えば，整数型変数 int に入力する場合は，

```
scanf("%d", &int);
```

とする．

[22] に対応する部分の処理が実行型ファイルによって行なわれる時，キーボードからの入力待ちとなり処理が中断する．そこで，現在，何を入力することを求められているかを，モニターに表示しておくと便利である．[19] から [21] は，この目的の為の画面表示を行なう命令である．[23]，[24] によって，b にもキーボードから数値が代入される．

[27] で関数 choice が呼び出される．引数をとらない関数は，他の関数 (今の場合は，main 関数) の中で使用する場合，この choice のように ( ) の中を空にする．戻り値は整数であるので，choice を呼び出した main 関数の中で宣言された整数型の変数 ch で受ける．戻り値の受け方は，この例のように，関数を右辺とした代入構文とする．既に気付いている読者も多いと思うが，printf，scanf も choice と同様に，戻り値として整数型の変数を持つ．ところが，printf，scanf は，戻り値を main 関数内の変数へ代入していない．実は，n = printf("Hello, world"); としてもプログラムはエラーなくコンパイルされ，実行型ファイルも全く同様に動作する．つまり，printf，scanf は，戻り値を明示的に main 関数に返さなくともよいように設計されている．

[30] 以降は，制御構文のうちの条件分岐，さらにその一種である switch 構文である．swith( ) は ( ) の中の変数に代入されている値を見て，次の処理として，以下に続く case 1，case 2 ･･･ のいずれかを選択的に実行する．一般的は用法は，

```
swith(整数型変数){
     case 1:
```

[4] printf の場合と同じく，整数は，１６進法等でも入力でき，やはり，%x 等を用いる．

```
            整数型変数の値が 1 の場合の処理

            break;
        case 2:

            整数型変数の値が 2 の場合の処理

            break;

                    ⋮


        case n:

            整数型変数の値が n の場合の処理

            break;
        default:

            整数型変数の値が 1 から n 以外の値の場合の処理

            break;
    }
```

今の場合，例えば，ch の値が 2 であれば，case 2([35] から [38]) が実行される．各 case の最後の行の break は，swith 構文を抜け出て，それに続く処理へ進むことを指示している．[38] の break により，その後に続く switch 構文の残りの部分 [39] から [50] が無視されて，[53] へと進む．[47] の default は，ch の値が不適切な場合，今の例では，1 から 4 の整数以外であるときの処理内容である．

最後 [53]，[54] で，結果を表示し，[56] で OS に main 関数の戻り値として 0 を返して，プログラムは終了する．

以上，main 関数を設計するにあたって，main 関数の中で呼び出された各関数の具体的な中身を知る必要はなかった．必要な情報は，それぞれの関数の機能のみ，つまり，引数と戻り値，また，関数が呼び出されている間に実行する処理内容のみである．

3. 4. 3. 関数の設計

まず，choice について考えよう．演算の種類をキーボードからの入力で選択する．引数は持たず (取らず)，戻り値は，選択された演算に応じて，和=1，差=2，

積=3，商=4 を返す．戻り値は，choice 内で宣言された整数型の変数 c に格納された値を [71] で return することで (具体的には，[71] のように，return c; とする) choice を呼び出した関数 main に返される．したがって，[27] のように，文法上は ch = choice(); と記述されるが，実際に代入は，ch = c; で行なわれている。

次に add を見よう．[74] で float add(float x, float y) と記述されている．この x, y を**仮引数**と呼ぶ．仮引数は，引数 (3.3.1 節参照) の代入を受ける変数である．このように記述することで，実は，変数宣言もかねている．つまり，add 内で利用される浮動小数点型変数 x, y が [74] で宣言されたことになる．つまり，仮引数は，その関数内の変数 (**内部変数**と呼ぶ) の一つである．続いて [75] で，浮動小数点型変数 z が宣言され，add は，３つの変数を内部変数として持つことになった．さて，関数の内部変数は，その関数が呼び出されている間のみ機能する．つまり，主記憶上の数値の格納区画も，戻り値を返すと同時に消滅する．さらに重要なことは，(今の場合はそうではないが) main 関数によって呼び出される関数の中で， main 関数で既に使用されている変数名と同じ変数名を持つ変数を，仮引数としてであれ完全にその内部において (つまり，[75] 以降) であれ，宣言しても， main 関数の内部変数である同名の変数とは別の変数として扱われる (図 3. 4 参照)．このことはポインターと関連して C 言語の最も重要な特徴であり，ポインターの解説でもう一度触れる．

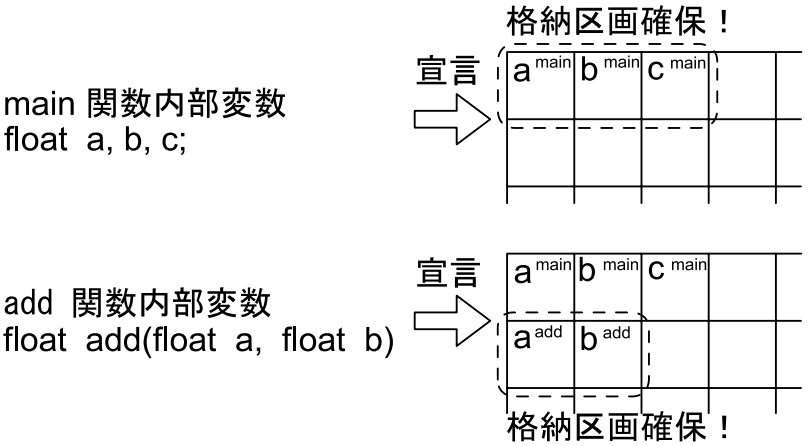

図 3. 4.

[32] をもう一度見よう．add(a, b) は上の説明から，x = a, y = b の二つの代入構文を実行していることとなる．つまり，add の内部変数 x, y へ main 関数の変数 (やはり，main 関数の内部変数と呼ばれる)a, b に格納済みの値が代入されている．したがって，[77] の処理は，この部分だけを見ると，値の格納されていない変数 x, y が右辺に存在し，文法違反のように見えるが，仮引数として宣言された変数は，その関数が呼び出されると同時に値を代入されることが予定され

ているので，正しい処理となることがわかる．なお，関数が引数をとる場合，その仮引数に値を代入するために，代入構文を実行する変数，つまり，c = add(a, b) の a，b 等 (または，a，b に格納されている数値) を，仮引数と明確に区別するために**実引数**とも呼ぶ．

3．5．ポインター

変数は，主記憶上のデータの格納区画の識別子，つまりは名前であり，また，格納・呼び出しの為の手段であった．この格納・呼び出しの為の手段としての役割は，例えて言えば，リモート・コントローラー (リモコン) である．主記憶の内容を，我々は “直に” 覗き見たり，変更することは出来ず，変数というリモコンを介して操作している．ここで注意すべきことは，変数名 a で操作できる主記憶の区画は，変数宣言時に変数名 a で確保された区画のみである．つまり，このリモコンは格納区画 a に対する，専用リモコンであり，他の格納区画に対するリモコンとしては，決して代用できない．

さて，C 言語には，実は，専用リモコン以外に “汎用リモコン” が用意されている．この汎用リモコンが，表題の**ポインター**である．ポインターは，各変数型に対応して，例えば，浮動小数点型変数の汎用リモコンとして使用する場合，浮動小数点型変数用のポインターとして，宣言する必要がある．ポインターは，汎用リモコンであるが，変数型の違いを超えて汎用することは出来ない．ポインターの宣言は次の例のように行う．

```
float sum, *rc;
int   counter, *pc;
```

sum，counter は，それぞれ通常の浮動小数点型，整数型の変数であり，*rc，*pc が，それぞれ浮動小数点型，整数型変数用のポインターである．

ポインターの必要性を説明する為に，次の頻繁に引用されるプログラム例を示す．

3．5．1．ポインターの働きを説明する典型的プログラム (誤)

```
[1] #include <stdio.h>
[2]
[3] void swap(float x, float y);
[4]
[5] main()
[6] {
[7]      float a, b;
[8]
[9]      printf("Input a = ? ");
[10]     scanf("%f", &a);
[11]     printf("Input b = ? ");
```

```
[12]     scanf("%f", &b);
[13]
[14]     swap(a, b);
[15]
[16]     printf("Result: a = %f, b = %f\n", a, b);
[17]
[18]     return 0;
[19]}
[20]
[21]void swap(float x, float y){
[22]     float tmp;
[23]
[24]     tmp = x;
[25]     x = y;
[26]     y = tmp;
[27]}
```

このプログラムは，変数 a と b に格納されている値を，交換することを意図して設計されいる．しかし，実際にはその意図どおりの動作はしない．

プログラムの動作を順に見ていこう．このプログラムは swap と名づけられた関数を用いている．まず，[9] から [12] において，変数 a，b に値の代入を行っている．続いて [14] において，関数 swap を呼び出し，a，b に代入された値の交換を行わせようとしている．関数 swap は，その内部変数として，x，y，tmp の 3 つの変数を持つ．仮引数も内部変数であることを思い出そう．

関数の内部変数は，その関数が別の関数 (このプログラム例では main 関数) から呼び出されている間だけ，主記憶上に確保される．ここでもっとも大切なことは，関数 swap の仮引数 (= 内部変数)x，y を，main 関数で宣言されている変数と同一名で宣言しても，すなわち

```
void swap(float a, float b);
```

と宣言を行っても，やはり，期待通りの入れ替えは実行されない．それは，例え同一名でも，各関数の内部変数は，主記憶のそれぞれ別の格納区画に対応付けられるからであった．つまり，main 関数の a と関数 swap の a は，対応する主記憶上の格納区画が異なり，同じ名前でも異なる変数として処理されるのであった．

一応，上のプログラム例の実行型ファイルの動作を追ってみよう．main 関数の a，b には，[10]，[12] によって，例えば，2, 3 が代入されているとする．[14] により，swap 関数の x，y に a，b に格納されている値 2, 3 が代入される．以後，[21] 以下の処理に移る．[24] で x に格納された値 2 を tmp にも格納する．[25] では，x の値が，y に格納された値 3 に更新される．ここで，x に以前格納されていた値 2 は消滅するが，[24] の操作で，tmp に保持されている．最後に y の値を

tmp に保存されている元々の x の値 2 に更新して，x，y に格納されている値の入れ替えが完了する．注意すべきことは，x，y に格納されていた値の入れ替えは，確かに実行されているが，main 関数の内部変数である a，b の値は，[14] で一度参照される以外，全く何の処理もされることがないことである．これは，void swap(float a，float b) と宣言したとしても，やはり，swap 関数内の a，b の値が入れ替えられるだけで，main 関数の内部変数 a, b は一連の処理に，やはり，無関係である (図 3. 5 参照).

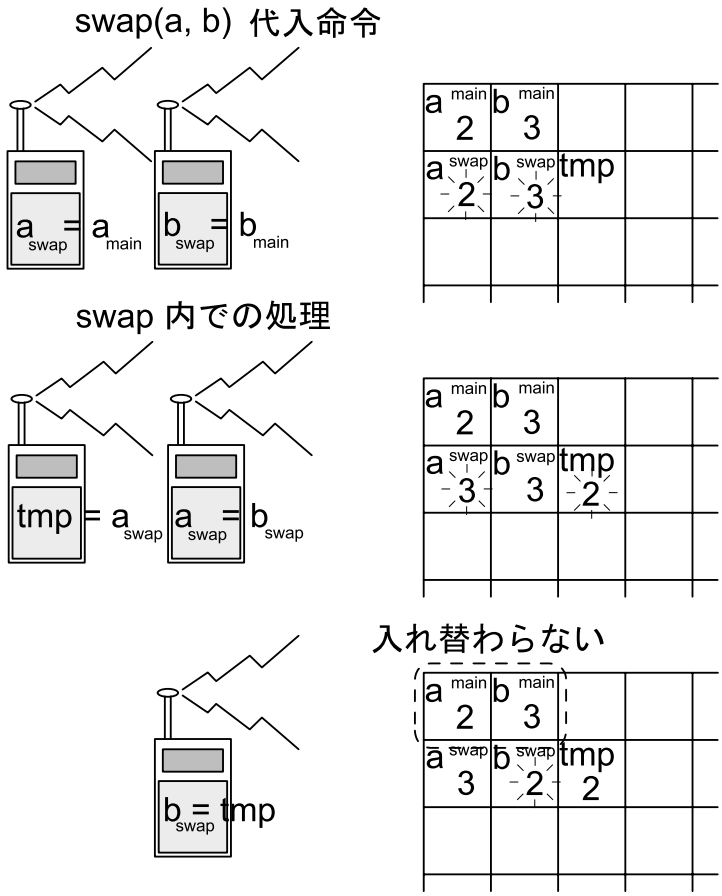

図 3. 5.

3. 5. 2. ポインターの利用法

それでは，意図どおりに main 関数の内部変数 a，b の値の入れ替えを実行できるプログラムは，どのようなプログラムであろうか．このプログラム例を示す前に，ポインターについて，少し説明しておこう．ポインターは汎用リモコンであると説明したが，それでは，実際にこの汎用リモコンを使用するには，どうすればよいのであろうか.

まず，変数は特定のデータ格納区画に対応した，参照・格納の為のリモコンであるが，リモコンとして機能するには，変数はデータ格納場所の “位置” に関する情

報を保持していなければならない．この位置を変数 (の指す主記憶上の区画) の**アドレス**と言う．つまり，各変数は，それぞれ対応するアドレスの情報を保持している．このアドレスは，例えば，変数名 a の変数の場合，&a で参照できる．つまり，

```
printf("変数 a のアドレスは = %d", &a);
```

など参照できる．この使用例からわかるとおり，&a は整数である．

さて，ポインターが

```
float a, b, *t;
```

と宣言されているとする．このとき，*t に対応する主記憶のデータ格納場所は，まだ，決まっていない．汎用リモコンであるので，特定の格納場所の情報を与えない限り，リモコンとして機能しないのである．

```
t = &a;
```

とすると，変数 a のアドレスが *t に与えられ，*t は a に対応した主記憶上のデータ格納区画を操作できるリモコンとなる．*t = &a; ではなく t = &a; であることに注意してほしい．t = &b; とすると，今度は *t は，b に対応した主記憶上のデータ格納場所を操作できるリモコンとなる．

この例は，ポインターの使用法としては無意味なもので，実際のプログラムでは，変数のアドレスを同一関数内のポインターに設定することはまずない．なぜなら，上の例では，そのまま a, b をリモコンとして使用すれば，十分用が足りるからである．ポインターは，ひとつの関数から別の関数を呼び出したとき，呼び出し側の関数の内部変数の代理のリモコンとして，汎用リモコンの機能を利用される．それでは，値入れ替えのプログラムを修正しよう．

3. 5. 3. ポインターの働きを説明する典型的プログラム (正)

```
[1] #include <stdio.h>
[2]
[3] void swap(float *x, float *y);
[4]
[5] main()
[6] {
[7]      float a, b;
[8]
[9]      printf("Input a = ? ");
[10]     scanf("%f", &a);
[11]     printf("Input b = ? ");
[12]     scanf("%f", &b);
[13]
```

```
[14]     swap(&a, &b);
[15]
[16]     printf("Result: a = %f, b = %f\n", a, b);
[17]
[18]     return 0;
[19]}
[20]
[21]void swap(float *x, float *y){
[22]     float tmp;
[23]
[24]     tmp = *x;
[25]     *x = *y;
[26]     *y = tmp;
[27]}
```

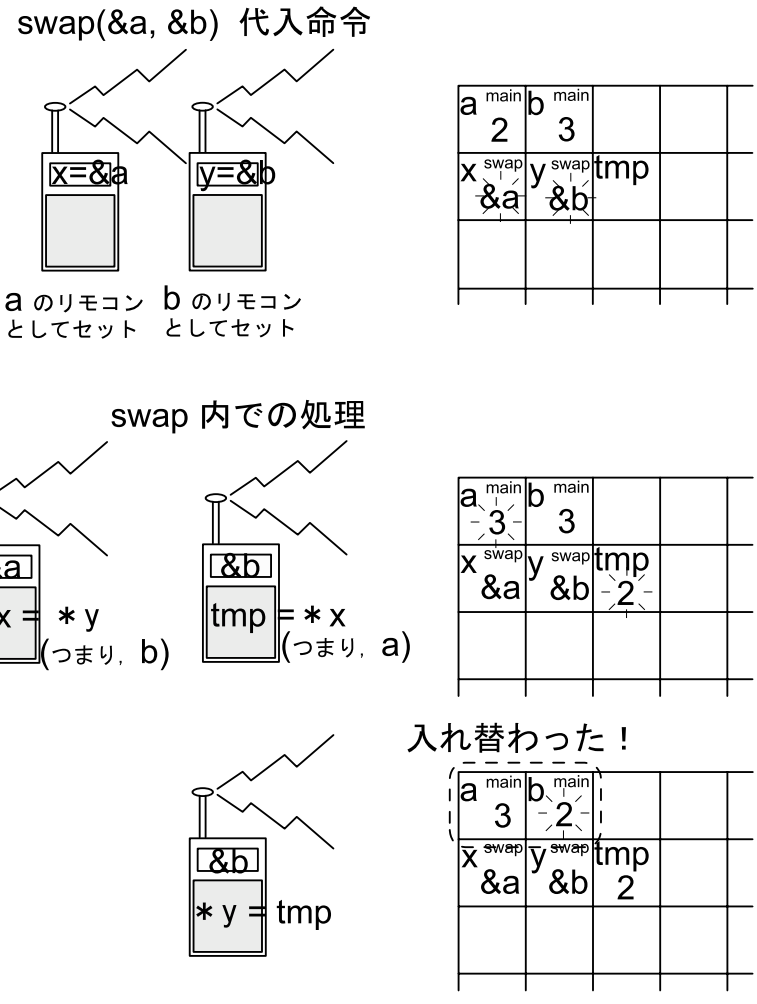

図 3. 6.

修正は [3], [14], [21], [24] から [26] の各行で行っている．まず，[3] では，関数 swap の仮引数を浮動小数点数型のポインターで宣言している．[14] では，ポインターが，特定の主記憶上の区画を指すように，x = &a，y = &b とアドレスの代入を行っている．*x，*y は swap の内部変数ではあるが，swap を呼び出した main 関数の内部変数 a，b に対応する主記憶上の区画に，数値の代入・更新を行える．[25]，[26] で a の値が b に保存された値に更新され，b の値が，tmp に一時記憶された a の元の値に更新される．以上より，今回の修正を施したプログラムは，当初の意図通り，a，b の値を入れ替える (図 3. 6 参照).