C 言語入門講義

(文責) 大坂　博幸
立命館大学理工学部助教授

Contents

1 第1回 はじめての C **1**

1.1 この講義では 1

1.2 C 言語の欠点 1

1.3 参考文献について 3

1.4 Hello, world 6

1.4.1 ソースファイル hello.c の作成 6

1.4.2 実行ファイルを作成 8

1.4.3 Hello, world プログラムの説明 9

1.5 cc の動作と用語説明等 9

1.6 練習問題 11

1.7 おまけ 12

1.7.1 コンピュータはどう計算しているのか 12

1.7.2 プログラミング言語は, なぜ必要なのか 12

1.7.3 プログラミング言語の種類 13

2 第2回 C で計算をしよう！ **17**

2.1 基本データ型 17

2.2 インタラクティブなプログラム 18

2.3 整数の演算 19

2.4 繰り返し 21

2.5 計算機イプシロン 23

2.6 近似値と誤差 23

2.6.1 誤差の限界 24

2.6.2 いろいろな誤差 24

2.7 練習問題 26

2.8 付録 27

3 第 3 回 制御文（if else, for) 29
3.1 if else による条件分岐 . . . 29
3.1.1 数の大小比較 . . . 29
3.1.2 数学ライブラリの使い方 . . . 31
3.2 関係演算子と論理演算子 . . . 33
3.3 付録: 二次方程式の解法 . . . 34

4 第 4 回 制御文（while) 37
4.1 基本データ型についての復習 . . . 37
4.2 取り込んだ文字列を変換する . . . 39
4.3 while 文 (フィボナッチ数列) . . . 41
4.4 練習問題 . . . 42

5 第 5 回 関数 43
5.1 目標 . . . 43
5.2 御託宣 . . . 43
5.3 前回までの復習と補足 . . . 44
5.4 関数の宣言と記述 . . . 44
5.4.1 補足: main が返す値は ? . . . 45
5.5 実習 . . . 46
5.5.1 例 1. ユークリッド の互除法 . . . 46
5.5.2 例 2: ヘロンの公式を使った 3 角形の面積 . . . 50
5.5.3 例 3: 再帰 (recursion) . . . 55
5.6 練習問題 . . . 56

6 第 6 回 関数 II 57
6.1 目標 . . . 57
6.2 sin 関数の作成 . . . 57
6.3 台形公式による定積分の計算 (変数のキャスト) . . . 60
6.3.1 台形公式と誤差 . . . 63
6.4 練習問題 . . . 65

7 第 7 回 関数とポインタ（Call by Value, Call by Reference） 67
7.1 今までの講議に関する確認と補足 . . . 67
7.1.1 小数型の入出力 (printf, sscanf) . . . 67
7.1.2 式の値 . . . 68
7.1.3 実行ファイルの名前を変えるには . . . 68
7.2 関数とポインタ . . . 68

7.2.1 引数はコピーされる 69
7.2.2 変数のアドレス 71
7.3 練習問題 77

8 第 8 回 配列 I 79
8.1 目標 79
8.2 配列とは 79
8.2.1 1 次元配列の宣言 79
8.2.2 2 次元以上の配列の宣言 80
8.3 素数表の作成 (エラトステネスの篩 (sieve)) 80
8.4 演習 82
8.4.1 例 1:1000 までの素数表を出力するプログラム 82
8.5 配列と関数および前回までの復習 83
8.5.1 例 2: 内積を用いた角度の計算 83
8.6 さらにポインタ 85
8.7 例 3: 配列とポインタ変数の違い 85
8.8 文字列に関する注意 87
8.9 練習問題 88

9 第 9 回 配列 II: 並び替え（バブルソート） 89
9.1 今までの講義に関する確認と補足 89
9.1.1 式の値 89
9.1.2 配列とポインタ 90
9.2 最大値捜し 93
9.3 並べ替え (バブルソート) 93
9.4 演習 95
9.5 練習問題 96

10 第 10 回
方程式を解く（二分法，ニュートン法） 97
10.1 二分法 97
10.2 ニュートン法 99
10.3 演習 100
10.4 練習問題 101

Chapter 1

第 1 回 はじめての C

1.1 この講義では

C 言語を用いたプログラミングの入門を, 講義します. 計算機を実際に動かしてプログラムを作成してもらいます. 内容は, 主に C 言語の文法の話です. 外国語と同じで, 文法だけではプログラムは書けないのですが, 他のことに触れる時間的余裕は, 残念ながらあまりありません. 各自補うようにしてください.

何故 C 言語をするかは次の理由からです.

1. 最近の流行である.
   書店のコンピュータプログラミングのコーナーにいきますと, C 言語の本が多数並んでいます. また, Netnews でも C 言語に関する newsgroup がありいろいろな議論がかわされています. この様に, 勉強するにあたって色々な情報が身近にあるというのは都合がいいことです.

2. UNIX との親和性が高い
   この講義では, UNIX を用いてプログラミングの実習をします. C 言語はその UNIX の開発言語です. 従って, UNIX には C 言語でのプログラミングを補助する色々なツールがそろっています. (補助する道具については, どの程度授業で述べられるかは, わかりませんが...)

1.2 C 言語の欠点

C 言語は, UNIX の開発に際して assembly 言語（意味は後記のおまけ参照。以下同様）の代わりとなるべく作られました. 従って, 比較的低級な言語で

す. と言うことは, computer の Hardware の簡単な仕組を理解していることが暗黙裏に要求されたりすることがあります. また, 初期の UNIX 開発のターゲットマシンだった Dec PDP-11 の機械語命令の影響を受けていまして, 必ずしも汎用性のある言語仕様だとは言えない部分があります.

さらに, 1980 年代には C 言語に様々な方言が存在しました. これは, 初期の C 言語の言語仕様がきっちりしていなかったことが原因なのですが, これを何とかしようということで, ANSI で標準化作業がなされました. (これに対して例えば Pascal は, 開発者が言語仕様をきちんと決めています.) そして出来たのが ANSI-C です. しかし, この標準化作業がいい格好で終わったとは言えない部分が多くあります. ということで, C 言語は初心者がいきなりはじめるには不都合なところが残念ながら多くあります.

また, 初心者にとっては C 言語の言語仕様の小ささ (機能が少ない) も欠点となります. プログラミング言語としては, 言語仕様の小ささというのは大した問題ではないのですが, 初心者には取っ付きにくいものになります. 例えば, コンピュータのプログラムを書く上で, 入出力というのは基本的に必要なものなのですが, C 言語そのものには, 入出力の仕様がありません. C 言語では, 多くの処理をあらかじめ作られたライブラリというプログラムに任せます. ということは, このライブラリ[1]の使い方をある程度勉強する必要があります.

[1]文字, 文字列, 入出力, 数値関数, 日付の変更, 動的な記憶領域確保など

1.3 参考文献について

はじめのほうは, C 言語に対する参考文献で後半は, それ以外のものを挙げました.

[1] は, C 言語のバイブルです. C でのプログラミングを真面目にしたいのでしたら, 必ず買うようにしてください. なお, 最近日本語訳を修正した版が出版されています. この事実からわかるように, この本の訳は必ずしも良いとは言えず, 明らかな誤訳が多数含まれています. そういう意味では, 原書を手に入れたほうがよいかもしれません.

[9],[10] は一人でC言語勉強するのに最適な本であると思います. これらを読んだあと, [1] を読むことを奨めます.

[2], [3] は, プログラムを書く上での心構えを主に書いた本です. プロフェッショナルを目指すなら, 一読すべきでしょう.

[5], [4] は, C 言語を用いた数値計算法の本です. [5] には, 英語版ですが Pascal で同じ内容の本があります. [4] には, プログラムのソースファイルが入ったフロッピーディスクがついてきます.

[6] は数学者が書いた本です. そういう意味では, 取っ付きやすいかもしれません. ただそこに書いてあるプログラムのプログラミングスタイルに少々問題があります. [7] は, [6] の結び目理論にかかわる部分を, 追加して際構成した本です。

C 言語を自分の PC に移植して学びたい方は, [20] の本がよろしいでしょう. 付録の CD-ROM からコムパイラを簡単にインストールできます.

UNIX のシステムコールに関連したプログラミングの本として [8], [12] を挙げておきます. この二つは共に古いので, 取り敢えず挙げただけです.

[13] は UNIX のネットワークプログラミングの本です.

[14], [15], [16] はプログラミング全般にわたる古典的名著です.

[14] は, わかりやすいプログラミングの心得を書いたものです. 処理系には ratfor と呼ばれる独自の言語を用いて書いてあります. ratfor は皆さんが今使っている機械にもあったと思います. なお, 日本語訳はありませんが, これとまったく同じ内容を Pascal で書いた本が, 同じ著者で出版されています.

[15] は, データ構造とプログラムの関係を書いた本です. 処理系は Pascal を使っています. プログラムというのは, しょせんデータの処理方法の記述ですが, そのデータ構造をうまく解析して与えないと, よいプログラムが書けないと言う視点でプログラミングを解説しています.

[16] は, プログラムに現われる数学的構造を面白く書いた本です. 第 2 巻までは, 日本語訳があります. 著者は, この本の出来上がりが気に入らないといって TEX を開発したのは有名です.

皆さんの使っている機械では, C 言語以外でもコンパイラ系で, Fortran, Pascal, インタプリタ系で, Awk, Perlが動きます. Fortran 以外について [17], [18], [19] に各々の処理系の原典を挙げておきます.

世の中には, プログラミングの本も C 言語の本も山のように出版されています. この中で, 良い本を捜すのは大変です. 伝え聞く限りでは, 多くの本が出版されている割には良書と呼ばれるものは少ないようです. また, 翻訳本にはひどい誤訳もあるそうです. この様な情報は, たまに Net News などでも議論になっているようですから, Net News の C 関係の news group を定期的に読むことをお勧めします.

なおこのノートを書くにあたって久保智史氏に大変お世話になりました. この紙面をもちまして感謝いたします.

Bibliography

[1] カーニハン・リッチー著, The C Programminng Language,
日本語訳 石田晴久訳, プログラミング言語 C 第 2 版, 共立出版.

[2] S. Oualline 著 岩谷宏訳, Practical C Programming 現実的な C プログラミング, ソフトバンク.

[3] 藤原博文著, C プログラミング診断室,　技術評論社.

[4] L. Baker 著 吉田弘一郎訳, C 言語数学関数ハンドブック, 技術評論社.

[5] W. H. Press 他著 丹慶勝市他訳,　Numerical recipies in C, 技術評論社.

[6] 落合豊行著, C 言語による数学解析, 近代科学社 1988 年.

[7] 落合豊行, 山田修司, 豊田英美子著, コンピュータによる結び目理論入門, 牧野書店 1996 年.

[8] Curry 著 アスキー編集部訳, UNIX C プログラミング, アスキー 1991 年.

[9] 内田智史著,　C 言語によるプログラミング [基礎編], [応用編], オーム社　1991 年.

[10] 　結城　浩著,　C 言語プログラミングレッスン [入門編], [文法編], 1994/1995 年, [スーパーレファレンス編] 2000 年, オーム社.

[11] エイチアイ著, 実用 UNIX C/C++言語ハンドブック, ナツメ社 1999 年.

[12] Rochkind 著　福崎訳,　UNIX システムコールプログラミング, アスキー 1987 年.

[13] W. R. Stevens 著 篠田訳, Unix ネットワークプログラミング, トッパン 1992 年.

[14] カーニハン･プローガー著 木村 泉訳, ソフトウェア作法, 共立出版 1981 年.

[15] N. Wirth 著 片山卓也訳, アルゴリズム + データ構造 = プログラム日本コンピュータ協会 1979 年.

[16] Donald E. Knuth, The art of computer programing, vol 1 - vol 3 Addison-Wesley (1 巻と 2 巻は日本語訳有).

[17] K. イェンゼン・N. ヴィルト著 原田賢一訳, Pascal 第 4 版 培風館.

[18] A. W. エイホ, B. W. カーニハン, P. J. ワインバーガー著足立高徳訳, プログラミング言語, AWK アジソン ウェスレイ, トッパン 1988 年.

[19] L. Wall, R. L. Schwartz 著 近藤嘉雪訳, Perl プログラミング, ソフトバンク 1993 年.

[20] 高田美樹, C 言語スタートブック〜基礎 C のエッセンス, 技術評論社, 2001 年.

1.4 Hello, world

1.4.1 ソースファイル hello.c の作成

まずは C 言語を使った最も簡単なプログラミングを作ってみましょう．次のプログラムは，C 言語の入門では必ず一番最初に書くものです．

画面に Hello, world と表示するプログラム．ファイル名 hello.c

Mule を用いて，hello.c という名前のファイルに次の内容を入力し，セーブしてください．`/*` と`*/` で囲まれた部分はタイプしなくて良いです．

```
-----この下から-----
/* Hello, World を出力するプログラム */

#include <stdio.h>

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

-----この上まで-----

/* と*/ で囲まれた部分はコメント文と呼び，プログラムの説明などを書きます．この部分はなくてもプログラムの実行には関係しません．

注意 コメント文が必要な理由の一つは, 基本的にプログラムは他人も見ることを前提に書かれるべきだからです. それ故, 誰が見てもわかるようにコメントを入れておくとそのプログラムの改良などの作業がしやすいわけで, 例の 2000 年問題はその点修正が苦労したそうです. 構造化プログラミングの思想にのっとって, 誰にもわかりやすい, 「数学屋のような癖」のないプログラムを書くようにしましょう.

1.4.2 実行ファイルを作成

ソースファイル hello.c から実行ファイル a.out を作成します.

(1) `cc hello.c` とタイプします.
3 - 5 秒後に再びプロンプトが現われます. a.out という新しいファイルが出来ていることを ls コマンドで確認しましょう.

(2) このファイルは, 実行可能ファイルです. `ls -l a.out` で, ファイルモードが実行可能 (x) になっていることを確認して下さい. (普通はしなくて良い)

(3) `./a.out` と入力します.

上の操作をソースファイルをコンパイルして実行するといいます.

> コマンド cc がうまく実行できなかった人は, 次をチェックしてください.
>
> 1. ファイル名は, hello.c となっているか.
> 2. コマンド 又はファイル名を正しくタイプしたか ?
> 3. hello.c の中味の括弧や quotation は, 正しく入力されているか ?
> 4. #include の # は行の先頭にあるか ?
> 5. printf 文の最後にセミコロン ; がついているか ?

メモ

UNIX のコマンドを忘れた人がいるかも知れませんが, UNIX 内で作成したファイルにはいくつか情報がついてまわります. 例えば, 今作成した実行ファイルを `ls - l a.out` でみると,

-rwxr-x— 1 osaka teacher 5844 Mar 31 11:52 a.out

ここで重要なのは最初の１０項目です. 一番目はファイルの種類を表しています. `"-"`は通常ファイル、`"d"`はディレクトリを意味します. 残り, ９項目は, ３項目づつ所有者, グループ, その他のユーザに対する許可情報を与えています. 今の場合, 所有者に, r(読みとり), w(書き込み), x(実行) 許可が与えられていることがわかります. 先に述べたことは, ここでもし, 所有者情

報で x が- であったら, chmod u+x a.out として許可を与えなければなりません. ここの, u+x は、u(所有者) に+x(実行許可を与える) ということです. これらのことは UNIX の基本でありますので, 時間があるときにでも本などに目を通して下さい. 例えば,「たのしい　UNIX」(坂本文著、アスキー出版社).

1.4.3 Hello, world プログラムの説明

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

#で始まるこの部分は，C プリプロセッサ (preprocessor) cpp への命令です. (とりあえずは，おまじないと思っていて下さい．)

```
main()
```

main と言う関数の始まりを示します．C 言語では，必ず main と言う名前の関数が必要で，ここからプログラムの処理がはじまるようになっています. () の中には，この関数のパラメータ (通常プログラミングとか計算機の分野では，引数という．) が入ります.

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

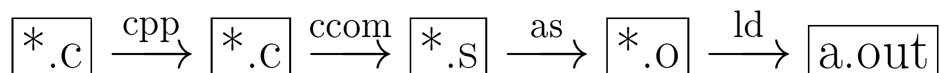
ここが，このプログラムの本体です．C 言語では，一連の処理は中括弧 { と } で挟まれた部分に書きます．中央の

```
        printf("Hello, world\n");
```

が，このプログラムがする全てです．**printf** は，標準入出力ライブラリにある関数の名前です．２重引用符 (") で囲まれた文字列を標準出力（ディスプレイ）に表示します．最後の\n は改行を表わします.

1.5 cc の動作と用語説明等

コマンド cc は一つのプログラムではなく，少なくとも次のプログラム (プロセス) を作り出します．下の図の四角く囲まれた部分に書かれているのは，その時に使うファイル名です.



cpp C preprosessor: C compiler (ccom) が処理をする前に，文字列の処理の部分のほとんどを受け持って，事前に処理をするものだと思ってください．

ccom C compiler : C 言語の文法で記述された処理内容を，assembly 言語に変換するプログラム．

as Assembler : assembly 言語で書かれた内容を，機械語 (すなわち 2 進法で記述されもの) に変換するプログラム．

ld Link editor : リンカともいう．上の hello.c ではコンパイルした結果は，`Hello，world\n` という文字の並びを関数 printf に受け渡すという命令だけが，作り出されます． printf の実体は，ライブラリとしてコンピュータのファイル

```
/usr/lib/libc.a
```

に入っています．ld は，hello.c のコンパイルしてできた結果と上のファイルにある printf を結び付けて (リンクするという)，さらにシステムが，実行時に要求する情報をつけて実行可能ファイルを作り上げます．

Library プログラムを組むと分かりますが，画面の入出力などはどのプログラムでも同じ様なことをします．これをいちいちゼロからプログラムしたのでは，無駄が多いのです．そこで，よく使う処理はあらかじめ 2 進法レベルのものを作成，まとめたのが必ずあります．これがライブラリ (library) です．ライブラリは，compiler(実は，リンカ) に指示することによって，自分のプログラムにくっつけることが出来ます．また，C 言語では標準ライブラリというのが ANSI により定義されており，多くの処理はこれを使って実現します．

ファイル名 C 言語のプログラミングにおいてファイル名は，次の規則にしたがって，名前をつけます．

1. *.c :C 言語のプログラムを記述したファイル．
2. *.h :ヘッダ (header) ファイル．定数とか関数の定義を記述する．
3. *.s :assembly 言語のプログラムを記述したファイル．
4. *.o :オブジェクト (object) ファイル．2 進法で記述した機械語が入っている．

5. a.out :特別なオプションを指定せずにコンパイルしたときに出来る実行可能ファイル.

上のファイルのうちいくつかは，hello.c をコンパイルするときには現われませんでしたが，それは単に目に見えないだけで，コンピュータの内部ではちゃんと作られています.

1.6 練習問題

問題 1.1. 次のように表示するプログラムを作成せよ.

```
---------------ここからはじまる--------

プログラミングはおもしろい.

Programing is interesting.

---------------ここでおわり-----------
```

二つの文の間に空行を入れていることに注意.

1.7 おまけ

1.7.1 コンピュータはどう計算しているのか

今のコンピュータは, 電気で仕事をしています. 数字は電気を用いて表されています. 例えば, ある基準電位から見て電圧がある時に 1, ないときに 0 というふうに決めてあります. このようにして, 電気を用いて 0 と 1 が表現できます. 従って, 2 進数を用いれば電気を用いて数が表現できることがわかります.

今のコンピュータの内部では, 全てのものが 2 進数で表現されています. また, 上のような電気の 0 と 1 の世界にうまくスウィッチを組み合わせると 2 進数での加法とか, 乗法をする電気回路が作れます. この回路をうまく利用して, コンピュータは計算をしたり文書処理をしたりしているわけです. また, この電気回路の利用の仕方の指定を変えれば, 一つの計算機が, いろいろな用途に使えます. この利用の仕方を変えることが, プログラミングなのです. そして, 利用の仕方を変えるのを記述する言語がプログラミング言語で, 利用の仕方の記述の事をプログラムといいます.

電子式として最初に活躍したコンピュータ ENIAC では[2], 配線を付け変えるという作業を人間がして, 利用の仕方の変更をしました. しかしこの方法では, ちょっとした計算にも膨大な人手がかかる事は明らかです.

現在のコンピュータのハードウェア (電気回路) には, この利用の仕方を与える幾つかの命令を, 用意するようにしました (von Neumann のアイデア). この命令を使って, コンピュータの利用の仕方を制御しております. そして, この命令も 0 と 1 の並びで記述されます. この 0 と 1 の並びで記述される命令もプログラミング言語で, 機械語といいます.

1.7.2 プログラミング言語は, なぜ必要なのか

さて, コンピュータにとっては便利な 2 進法も人間にとっては不便なことこの上ありません. 0 と 1 の並びでそこに何が記述してあるかを読み取るのは, 簡単な場合以外はまず不可能です.

そこで, 人間が用いる言葉に近い記述でプログラミングができるように工夫がなされました. そうして生まれたのが, プログラミング言語です. プログラミング言語というのは, コンピュータに対する命令を記述するための, ある人工的な文法をもった言語です. より広くはその言語の処理系 (その言語を用いて, コンピュータを働かせる仕組全体) のことも指します. コンピュー

[2]実はイギリスで開発されたコロッサスが一番最初らしい. 一松　信著 「暗号の数理」 (Blue Backs) 参照

タが, 文法に従って書かれた記述を自分の本来の言葉である, 機械語に翻訳し (実行し) ます. つまり, 人間にとって面倒なことをコンピュータにやらせるようにしたのです.

プログラミング言語を用いたプログラミングとは, 処理内容をその言語の文法にしたがって記述することです.

さらに処理内容の記述する時にも, コンピュータを利用するようになりました. プログラミングというのは, 処理内容を記述した文書を作ることですが, この文書を作るという処理にもコンピュータを使うわけです. この事が一般的になったのは, 意外にも比較的最近のことです.

1.7.3 プログラミング言語の種類

von Neumann 型のコンピュータが生みだされて以来約 40 年たちますが, その間に様々なプログラミング言語が開発されました.

まず, プログラミング言語はプログラムを処理する形態で, 次の二つに大きく分類されます.

Interpreter(通訳系と訳す事がある)
: 処理を記述したものから, 一部分のまとまりのある処理内容を機械語に変換しては実行し, それが終わると次の部分に移る, ということを繰り返して, 処理をします. つまり, 同時通訳の人が外国語から日本語に通訳するように, プログラミング言語の内容をその場で機械語に通訳して, 計算機に実行させているのです. awk は interpreter です.

Compiler(翻訳系と訳す事がある)
: 処理の記述を一度に全部機械語に翻訳してしまい, その後にそれを実行します. コンピュータは, 機械語に翻訳された内容を読みこみ, それを実行します. この授業では, compiler を取り上げます.

また, プログラミング言語は, その文法が違えば異なる言語です. 文法のなかで, 良く知られているものを挙げると次のようなものがあります. 注意して欲しいのは, 同じ文法を持つ言語でも compiler だったり interpreter だったりすることがあることです.

assembly 言語 プログラミング言語を作るに当って, 最も単純な発想はコンピュータに対する命令である 2 進法の並びを, 一対一で人間が理解できる言葉に対応させることです. この対応で一つの言語が作れます. それが, assembly 言語です. assembly 言語の処理系のことを assembler といいます. この言語を作るのは簡単ですが, 機械語で出来ることがあ

まりにも単純な事しかできないため, この言語を使うプログラミングは人間にとってはやさしいことではありません. 例えば, 画面に 1 を表示させるのも, assembly 言語を使うと結構大変な仕事になります. また, 機械語と一対一に対応しているということから, コンピュータが違えば, assembly 言語も違うということに注意してください. すなわち, assembly 言語という名前の全てのコンピュータに共通の言語があるわけではありません.

assembly 言語のようにコンピュータに密着した言語を, 低級言語といいます. それに対し, より人間に近い言語を高級言語といいます. 高級言語と呼ばれるものの特徴は, コンピュータが違っても (原理上は) まったく同じ記述でプログラミングが出来るということです.

Fortran Formula transformation の略です. 最も早い時期に作られた, 高級言語です. 主に科学技術用数値計算をするためのプログラミング言語です. 古くからあるために, 数値処理のためのパッケージなどが豊富で, 現在でも用いられています. ただ, プログラミング理論などが無い時代の産物なので, 最近の理論から見ると良いプログラミング言語とはいえないようです. 実際問題として, 特に Fortran 用のパッケージを使う必要性がないのなら, もう勉強する必要もないと思います

COBOL Fortran と同時期に作られた高級言語ですが, こちらはもっぱら事務処理用言語として開発されました. 時々マスコミを賑わす 2000 年問題の多くは, COBOL を用いて作られたプログラムで発生します.

ALGOL Fortran, COBOL と同時期に開発された言語です. こちらの方は, 数学者が中心となって作ったといわれるだけあって理論的にすっきりした言語だそうです. ただ, あまり普及しませんでした. その理由はよくわかりませんが, 想像するに, あまりにも言語仕様が巨大になったことと, 標準での入出力の定義が, 与えられなかった事だろうと思います.

Pascal スイスの N. Wirth という人が, 教育用目的に ALGOL を元に言語仕様を小さくし, さらに標準の入出力を定義して開発しました. 1980 年代には結構流行し, 現在でも比較的使われています. 現時点でも, プログラミング入門用言語としては, 良いものだと思います. ただ, 元の言語仕様では大きなプログラムを書くにはちょっと不便です. なお, パスカルから派生した言語として modula-2 と Ada があります. またその孫ともいえる, Oberon という処理系もあります. Oberon は, IBM-PC 互換機とか Macintosh では無料で処理系が ETH(何の略かは忘れました) から入手できます.

LISP List Processor の略です. プログラム言語としては, かなりの歴史を持っています. やたら括弧をつかってプログラムをするのが, 特徴です. これと親戚の言語として, Scheme というのもあります. 人工知能や Expert System の分野では, 良く用いられます. 例えば数学では, 数式処理系 (数式のまま微積分をしたり, 行列計算をしたりする処理系) のひとつ Reduce は, LISP をつかって開発されています. emacs の拡張機能も, LISP の親戚である Emacs LISP で作られます. 以前は方言の多い言語でしたが, 現在では, Common LISP と言う規格が作られました. ただ, Common LISP の言語仕様もかなり大きなものになっています. Unix 用では, 京都大学で開発された KCL (Kyoto Common LISP) が, 有名です.

Prolog 日本のどこかの省庁の次世代コンピュータの開発言語として採用されました. 三段論法をそのままプログラムしていくような言語です.

C 1970 年代に Unix のシステム記述用言語として開発されました. OS の開発にはそれまでは, assembly 言語を用いていましたから, そこに高級言語を用いたということで, これは画期的なことでした. しかし, そもそもは assembly 言語の代わりとして作られたという経緯があるため, 比較的低級な言語になっています.

SmallTalk 1970 年代に, 新しいプログラミング言語として Xerox の研究所で開発されました. オブジェクト指向という新しいプログラミング規範 (paradigm) を導入した言語です. もともとは, 言語だけでなく OS も一体となった環境でしたが, 残念ながらその環境はあまり世に受け入れられませんでした. (値段の問題だと思います) OS の環境のほうは, 良く似た考え方が Macintosh に採用され最近の主流となっています. オブジェクト指向プログラミングの考え方も, C, Pascal, Oberon LISP などに導入されて, Objective-C, C++, CLOS 等の言語となっています. 当時の Xerox では, 現在の主流となっている技術, 例えばワークステーションの概念とか, EtherNet によるネットワークがすでに考えられており, Xerox なしでは現在のコンピュータはありえないのですが, 当の Xerox がそこから余り利益を得ていないのは, 歴史の皮肉です.

BASIC アメリカで, プログラミング入門用として開発されました. (開発者たちの名前は忘れました) 多くのパソコンに入っていたため, 一躍有名になりました. 本来の BASIC の言語仕様は, 割ときちんとしているのですが, パソコンに移植されたものが言語仕様の一部分だったため,「良い」言語としては, あまり普及していません. (本来の言語仕様を True

BASIC というそうです) 手軽ではありますが, パソコンに付いてくるのを使ってプログラミングの入門をするのは, 余りいいことだとは思いません. なお, 新指導要領では高校の数学で, この BASIC を使用するそうです.

Perl 最近少し流行している処理系です. これは, compiler ではなく interpreter です. だいたい $awk + shell + C$ の感じです. つまり, shell のように command を実行させることが出来, awk のように手軽な記述が可能で, C のように何でもできる言語です. (C の中で, command 実行はしない事はないが, 処理系に依存するので大変. shell は command 実行以外はほとんど出来ない. awk も文書処理以外の処理は, 余り得意でない.) UNIX のシステム管理者を目指すなら, 勉強して損は無い言語です.

Java WWW で閲覧するページにいろんな機能を付け加えるために開発された言語です. C 言語に似た構文とオブジェクト指向な環境が備わっています. 最近の WWW の流行にあわせて, これも流行しているようです.

Chapter 2

第 2 回 Cで計算をしよう！

今回は, C で簡単な数値計算のプログラムを書くことを目標にします。
その前に, C で用いられる変数についてお話しします。

2.1 基本データ型

C を用いたプログラミングでは，変数を使うときには必ずそのデータの型を指定して，その名前を宣言する必要があります.
変数の名前は，アルファベットで始まる 31 文字以下の文字の並びです. (31 文字を超える変数名を使うと，32 文字目以降は無視される．) 実際にプログラムを動かすときデータは，主記憶に格納されます．機械語のレベルでは，この時に必要な記憶容量を正確に知っておく必要があります．つまり Compiler は，変数の宣言によって必要となる記憶容量を計算し，これを元にして，プログラムの内容を機械語に翻訳していくのです．この記憶要領の計算の元になるのが，データの型と言われるものです.
C 言語におけるもっとも基本的なデータの型は，以下のとおりです．これ以外にも，データの型というものがあるのですが，それらについては後の講義でふれます.

char 1 文字を表す．通常は， 1 byte = 8 bit．皆さんの使っている処理系もそうです．ただ，これでは日本語の漢字等は表示できないので，漢字表示が必要な場合には特別なことをします．この講義ではその事には触れません.

int 整数型．これは数学的な無限の整数を扱える訳ではなく，有限の範囲の整数しか扱えません．扱える範囲は処理系によって様々です．RAINBOW

のマシンでは int のサイズは 4 byte = 32 bit です.

float 浮動小数点型．実数を扱うときに使用します．現在ではあまり使いません．RAINBOW のマシンでは float のサイズは 4 byte = 32 bit です.

double 倍精度浮動小数点型．float では計算の誤差が多く，精度が確保できないときに使うために，設けられたのが起源です．しかし現在では，この double を使うのが普通です．RAINBOW のマシンでは double のサイズは 8 byte = 64 bit です.

四則計算にはいる前に, 前回の補足をします.

2.2 インタラクティブなプログラム

インタラクティブ (interactive) とは，会話的という意味です．正確な意味は辞書に載っています.

実行の途中で文字の入力を要求するようなプログラムを作成します．以下の例では，改行までの入力文字列を char 型の配列に変換する **gets()** という関数を利用しています．次のプログラムを 2-1.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい.

```
#include <stdio.h>

main()
{
    char        nyuuryoku[80];   /* 半角 80 文字まで入力できます */

    printf("文字列を入力して下さい >> ");

    /* 改行までの入力文字を char 型の配列に変換する */
    gets(nyuuryoku);

    printf("あなたの入力した文字は\n\t%s\n です\n", nyuuryoku);
}
```

printf の新しい使い方が出てきています.

関数 printf の最初の引き数の中の`%s` は引き数が char 型の配列名のとき，”文字列に直して出力せよ.” という意味です．`\t` はタブ（右にずらし特定の列に出力する機命令）です.

これは UNIX の echo コマンドの簡単な version です。(実際の echo コマンドは main 関数部分を,
`main(int argc, char *argv[])` 書くことにより, コマンドラインからの情報を文字列して取得できますが, ポインタの概念が必要なので後で詳しく触れます.)
では、整数の四則計算をしましょう.

2.3 整数の演算

次のプログラムを 2-2.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    int     a, b;       /* 変数 a,b が整数型であることを宣言 */
    int     sum, diff, prod, quot;  /* 同じく型宣言 */

    a = 5;  /* a に 5 を代入しています */
    b = 3;  /* 同じく代入しています */

    sum = a + b;        /* sum に a+b を代入します */
    diff = a - b;       /* diff に a-b を代入します */
    prod = a * b;       /* 掛け算は * という記号を使います */
    quot = a / b;       /* 割算は / という記号を使います */

    /* printf は"で囲まれた書式に従って出力します */
    printf("summation = %d, difference = %d\n", sum, diff);
    printf("product = %d, quotient = %d\n", prod, quot);
}
```

関数 `printf` に渡されるパラメータは，２重引用符 (”) で囲まれた文字列, 変数 sum，変数 diff の３つです.（プログラミングでは関数のパラメータを引き数といったりします.)
最初の引き数（”で囲まれた文字列）の中の最初の%という文字とそれに続く文字は**２番目の引き数の値**が代入されます．同様に，２つ目の%という文字とそれに続く文字は，**３番目の引き数の値**に取って代わられます．

１行目の最初の%d **は ”整数 sum の値を出力せよ.”** という意味です． ２番目の%d は整数 diff の値を出力せよ．という意味です． ２行目の printf 文についても同様です．実数型 (double) の変数の出力には%f を使います.

このように，関数 printf の最初の引き数は，文字列を出力するだけでなく，２番目以降の引き数の値の表示方法を指定します.

問題 上の printf の中の書式指定において%d を%f に変えたらどうなるか調べよ.

今度は, キーボードから数を入力して計算させるプログラムを作ります.（つまり、標準入力からのデータに対してのプログラム）

先に, 文字列を入力する場合, 関数 gets を利用しました. 今度は, 文字列以外にも適用できる関数 scanf をもちいます.

次のプログラムを 2-3.c という名前で作成して下さい．(プログラム 2-2.c を利用して作成せよ.)

```
#include <stdio.h>

main()
{
    int     a, b;        /* 変数 a,b が整数型であることを宣言 */
    int     sum, diff, prod, quot;  /* 同じく型宣言 */

    printf("二つの整数 a,b の四則計算をします.>>");
    printf("a, b に代入する数を空欄をいれて連続して入力して下さい");
    scanf("%d %d", &a, &b);          /* a,b にデータ入力 */


    sum = a + b;         /* sum に a+b を代入します */
    diff = a - b;        /* diff に a-b を代入します */
    prod = a * b;        /* 掛け算は * という記号を使います */
    quot = a / b;        /* ひき算は / という記号を使います */

    /* printf は"で囲まれた書式に従って出力します */
    printf("summation = %d, difference = %d\n", sum, diff);
    printf("product = %d, quotient = %d\n", prod, quot);
}
```

関数 **scanf** の使い方は,

```
scanf("%d",&a)
```

で, int 型の変数 a にデータを入れます. ただし、double 型の変数 x の場合は,

```
scanf("%lf",&x)
```

となります. "l"(エル) **を忘れずに!!**.

変数 a の前の記号&は、**アンパサンド**といい, 変数 a のアドレスを与えています. 今は, おまじないと思って下さい.

メモ 役に立つ変換文字の表示記号を表にまとめてみました.

%d	整数を表示する
%c	1 文字を表示する
%s	文字列を表示する
%f	double 型実数を表示する
%p	ポインタとしての表示

ポインタについては第 7 回を参照.

2.4 繰り返し

次に繰り返しを使ったプログラムを作成してみます.

次のプログラムを 2-4.c という名前で作成して下さい. 出来たら, コンパイルして実行してみて下さい.

```
#include <stdio.h>

main()
{
    int     i;          /* i = 0, 1, ..., 5 */
    int     sum;        /* 0から4 までの和 */

    sum = 0;    /* 変数を初期化します */

    /* for 文は繰り返しの計算に使われます */
    for (i = 0; i < 5; i++) {
```

```
        sum = sum + i;
    }

    printf("sum = %d\n", sum);

}
```

まずは，**for (...)** の後ろの {} で括られた部分を見てみましょう．

```
sum = sum + i;
```

は sum の値に i の値を加えたものを新しい sum の値とせよ．という意味です．この部分は

```
sum += i;
```

と書くことも出来ます．

for 文は次のような形で使用します．

```
for（式１；式２；式３）{
    …;
}
```

式１は，for 文の中の処理を行う前に１度だけ行います．次に式２を判定し，その結果が真ならば，{} で囲まれた部分を実行します．その後，式３を実行し，再び式２を判定します．あとはこの繰り返しです．

上の例の式３には

```
i++
```

が書かれています．これは i を１増やせという意味で**インクリメント**と呼ばれます．この部分は，

```
i = i + 1;
```

あるいは，

```
i += 1;
```

と書いても同じことです．

問題 1 から 100 までの奇数の和を求めるプログラムを作成せよ．

2.5 計算機イプシロン

計算機イプシロンとは

$$1.0 + \epsilon > 1.0$$

となる最小の正の数 ϵ のことです．もちろん，実数の場合は,（有理数でも）上の式をみたす ϵ はいくらでも 0 に近くとれるので意味がありません.（数学では最小値は存在しないというのでした.）

計算機の場合，実数型はトビトビの値を取りますから，上記の ϵ が定まります．次のプログラムを 2-5.c という名前で作成して下さい．出来たら，コンパイルして実行してみて下さい．

```
#include <stdio.h>

main()
{
    int         i;      /* 繰り返しの数を数える */
    double      e;      /* 計算機イプシロン */

    e = 1.0;          /* 実数型では，代入する数も少数にします */

    /* 2で割っていき，1.0 + e = 1.0 となったらループを抜けます */
    for (i = 0; 1.0 + e > 1.0; i++) {
        e /= 2.0;
    }

    /* ループを抜ける一つ手前の値が計算機イプシロンです */
    printf("machine epsilon = 2^(%d)\n", -(i-1));
}
```

プログラム 2-5.c の中で, `e /= 2.0` の意味は、`e = e/2.0` です.
for 文の中に入る命令が一文だけなら { } は省略できます.

2.6 近似値と誤差

実際上の問題で，コンピュータなどを用いて数値計算を行う場合には，近似値を扱うのが普通です．そのため，必然的に誤差の問題が生じてきます．

2.6.1 誤差の限界

真の値 A の近似値 a に対して，近似値と真の値の差

$$a - A$$

を近似値 a の**誤差**といいます．その絶対値 $\mid a - A \mid$ を誤差ということもあります．また，ある小さい正の数 ϵ に対して

$$\mid a - A \mid \leq \epsilon$$

が成り立つとき，ϵ を近似値 a の誤差の限界といいます．このとき，真の値の存在する範囲は，次のように定まります:

$$a - \epsilon \leq A \leq a + \epsilon$$

例えば，$\sqrt{2}$ =1.41421356$\cdots\cdots$ の小数第 6 位を四捨五入した値 $a = 1.41421$ を $\sqrt{2}$ の近似値にとると

$$a - 0.000005 \leq \sqrt{2} < a + 0.000005$$

よって，a は不等式 $\mid a - \sqrt{2} \mid \leq 0.000005$ を満たし，この近似値の誤差の限界は 0.000005，すなわち 5×10^{-6} です．

2 つの地点 P, Q 間の距離の測定値は $100m$，別の 2 地点 R, S 間の距離の測定値は $42195m$ で，これらの誤差の限界はともに $10cm$ であったとします．このとき，誤差の限界は同じでも，P, Q 間の距離と，R, S 間の距離の大きさが違いすぎるため，測定値の近似の程度はかなり異なると考えられます．

このようなとき，真の値 A の近似値 a に対して，近似の程度をみるには，次の**相対誤差**を考えるといいです:

$$\frac{a - A}{A} \quad \text{または} \quad |\frac{a - A}{A}|$$

2.6.2 いろいろな誤差

コンピュータや電卓では，入力された 10 進数は内部で 2 進数に変換されて，計算などの処理が行われています．2 進数でも，10 進数の場合と同様に小数が考えられます．例えば，2 進数の 101.011 とは，

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

と表される数のことであり，10 進数で表すと 5.375 になります．10 進数の有限小数は，2 進数に変換すると無限小数になる場合があります．例えば，10 進数の小数 0.1 を 2 進数に変換すると

$$0.0001100110011001100\cdots\cdots$$

と，無限小数となります．コンピュータや電卓には，0 と 1 を記憶する場所が非常にたくさんありますが，無限ではなく限りがあり, そのため，上のような無限小数は，何桁目かで切り捨てや切り上げなどの，いわゆる「丸める」操作が施された有限小数として記憶されています．また，計算の途中に現れる数値にも,「丸める」操作がしばしば施されます．このような操作のために生じる誤差を，**丸め誤差**といいます．例えば，電卓で $(1\div3)\times3$ の計算を行うとき，丸め誤差のために，電卓によっては正しい答 1 ではなく，0.9999999 と表示されることがあります.

コンピュータは，非常に多い回数の計算を高速でおこないますが，無限回の計算を行うことはできません．弧度法で表された角 x の三角関数 $\sin x$ は

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\cdots$$

と，項が無限に続く式で表されます．例えば，この式を用いてコンピュータで $\sin x$ の値を求める場合，無限回の計算は行えないから，右辺を有限個の項で打ち切って，その近似値を求めることになります．このような事情のために生じる誤差を，**打ち切り誤差**という.

コンピュータを利用するときに生じる上のような誤差のほかに，有効数字の極端な桁落ちによる誤差もあります．例えば，有効数字が 5 桁の非常に近い 2 つの数

$$a = 2.1438, \qquad b = 2.1425$$

の差は $a-b = 1.3\times10^{-3}$ となり，有効数字が 2 桁に落ちます．そのため，この差に b を掛けた

$$(a-b)\times b = 1.3\times10^{-3}\times2.1425 = 2.78525\times10^{-3}$$

では，意味のある有効数字は，最初の 2.7 の 2 つだけになります.

以上のような誤差は，1 回の計算では微小であっても，例えば，コンピュータが得意とする繰り返し計算などでは，計算回数が多くなるにしたがって誤差が累積し，無視できなくなる可能性があることを認識しておくことは大切です．例えば、2 次方程式の解を求めるプログラムを組むとき, 解の公式では誤差が影響するため, 解と係数の関係を用いて作成するのが最良です.

2.7 練習問題

問題 2.1. (1) *for* 文による繰り返しを使って，2^{10} を求めるプログラムを作成せよ.

(2) 付録のプログラムを参照して 2^n を求めるプログラムを作成せよ.

問題 2.2. 計算機イプシロンの定義を

$$\epsilon > 0.0 \text{ となる最小の正の数 } \epsilon$$

と変えたとき、ϵ を求めるプログラムを作成し，*2-5.c* の結果と異なることを確認せよ.

2.8 付録

多項式の計算

繰り返しを使って, n 次の多項式

$$a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$$

の値を出力するプログラムを作りましょう. 但し、$x, a_n, a_{n-1}, \cdots, a_0$ は実数です.

プログラムを作成する前に, その**手順 (アルゴリズム)** について考える必要があります. 紙の上では、単純に直接元を多項式に放り込む方法が浮かびますが, コンピュータがする計算量を意識するとあまりよくないことがわかります. 実際, 乗法・加法を合わせて, $(2n-1)+n$ 回計算が必要になります (何故か).

計算量を減らして、なおかつ有効なアルゴリズムとして, 次の**ホーナー法**があります.

いま, $b_n = a_n$, $b_{n-1} = b_n * x + a_{n-1}$, $\cdots$, $b_0 = b_1 * x + a_0$ と繰り返していくと, b_0 がもとめる多項式の値です. この場合, 乗法・加法に必要な計算量は $2n$ と前のものより少なくすみます. このように, プログラムを組む場合において, 問題を明確化し適切なアルゴリズムをまず作成しなければなりません. また, そのときの計算量を意識しなければならなりません. あまりに大きいと（例えば行列式の計算とか）時間が途方にかかり, コンピュータがダウンしてしまいます. もう一つ大事なこととして計算誤差があり, それについては後で触れます.

次のプログラムを読んで下さい. 時間がありましたら, 作成し実行してみましょう.

```
#include <stdio.h>

main()
{
     double a, x, r;                          /* 変数a,x,rが実数型
であることを宣言 */
     int n, j;                                /* 同じく型宣言 */

     printf("次数nを入力して下さい>>");
     scanf("%d", &n);                         /* nにデータを入力 */
     printf("求める値xを入力して下さい>>");
```

```
    scanf("%lf",&x);                          /* xにデータを入力 */

    r= 0.0;                                   /*rの初期化*/
    for(j = n; j > = 0;j--){
     printf("係数a[%d]を入力して下さい>>",j);
     scanf("%lf",&a);                          /* ホーナー法のアル
ゴリズム */
     r = x*r + a;
   }
  printf("求める値は、%f.\n", r);
}
```

上の例の式3には

```
    j--
```

が書かれています．これは j を1減らせという意味です(デクリメントといいます)．この部分は，

```
    j = j - 1;
```

あるいは，

```
    j -= 1;
```

と書いても同じことです．

また，> =は，≧の意味です．その他，=(等しい)，\not=(等しくない)，\leq(以下)は，それぞれ、==，!=，<=と書きます．これらを総称して関係演算子と呼びます．プログラムの中で演算子の記述間違いがあっても，コンパイルがとおることがありますので注意しましょう．

Chapter 3

第3回 制御文（if else, for)

今回は, if else, for 文を使った制御文について学びます．内容として高校で取り上げるであろうものです.

次回は優しい文字列操作と while 文について触れるつもりです.

3.1 if else による条件分岐

3.1.1 数の大小比較

前回使用した for 文は，プログラムを繰り返し実行させる制御という機能を持った文です．繰り返しと並んで重要な制御のための文は **if** 文です．if 文は条件により，処理の分岐を行います．条件分岐のコマンドにはこの他に **switch** 文があります.

if 文と for 文があれば，殆どのプログラムを作ることが出来ます．次回はさらに **while** 文も扱います．while 以外の繰り返しを行うための命令は，**do while** があります.

do while と switch の機能については自習して下さい.

if 文を使ったプログラムを作成しましょう（ファイル名 3-１.c）.

```
/* ２つの整数を比較する */
#include <stdio.h>

main()
{
  int      m, n;
```

```
  printf("２つの整数を空白で区切って入力して下さい >> ");

  scanf("%d %d", &m, &n);

  if (m > n) {
    printf("%d は %d より大きい\n", m, n);
  } else if (m == n) { /* 等しいかどうかの比較は == を使用 */
    printf("%d は %d と等しい\n", m, n);
  } else {
    printf("%d は %d より小さい\n", m, n);
  }
}
```

前回使用した関数 scanf は書式付き入力関数と呼ばれるもので, 標準入力 (今の場合キーボードからの入力) から入力されたデータを指定した書式に変換して変数に格納します. プログラム 3-1.c にある例

```
scanf("%d %d", &m, &n)
```

は, m に最初の整数 (d) の値が代入され、n に次の整数 (d) の値が代入されるということです. (三つ以上も同じ) 注意として, 変数の前に&(アンパサンド) を忘れずに.

if 文の使い方は次の通りです.
条件が真のときのみ，ある処理を行いたいならば，

```
if (条件式) {
  条件式が真ならここを実行
}
```

条件式が偽のとき別の処理をしたいならば,

```
if (条件式) {
  条件式が真ならここを実行
} else {
  条件式が偽ならここを実行
}
```

条件式が偽のとき，別の条件を調べたいなら

```
if (条件式１) {
   条件式１が真ならここを実行
} else if (条件式２){
   条件式１が偽で，かつ，条件式２が真ならここを実行
}
```

3.1.2 数学ライブラリの使い方

数学ライブラリを使うソースファイルを if 文を用いて書きます．前回の問題 1 を関数 `pow(double x, double y)` を用いて書きましょう (ファイル名 3-2.c)．　但し, 求める指数の値は任意の実数とします.

この例では, 数学関係ライブラリを使うのでそこで扱う関数を定義するために最初に **math.h** というファイルを include しています.

```
/* 2^n を求めるプログラム */
#include <stdio.h>
#include<math.h>

main()
{

  double n;                      /* 指数 */
  double power;                    /* 求める解*/


  printf("求める指数 n を入力して下さい\n");
  scanf(  "%lf", &n);     /* n に値を代入 */

  power = pow(2.0,n);
  printf("求める値は%f です\n", power);
  }
```

数学関数ライブラリの使い方

上のプログラムは, 通常の cc コマンドではコンパイルできません.

```
% cc 3-2.c
```

を実行して, 出て来るメッセージを確認してください．私の環境では, 次のようになりました.

```
%cc 3-2.c
Undefined                         first referenced
 symbol                               in file
pow                                 3-2.o
ld: fatal: Symbol referencing errors. No output written to a.out
```

エラーメッセージは, 上のような表形式ででます. 1-2 行目は, 左側に未定義シンボルと書いてあって, 右側に最初に参照されたファイルと書いてあります. 単に翻訳しただけですが, この程度の翻訳はできるのが, 大学生の常識です[1]. 3 行目は, 上の物の内容で, 未定義シンボルが pow, 参照されたファイルが, 3-2.o という意味です. 4 行目の先頭は, このメッセージを出したコマンドで ld (link editor リンカ) でその後に書いてあるような事を言っています. (さて, なんて書いてあるのでしょう ?)

これは, ld がプログラム a.out を作り上げる際に pow という関数がどこにも無かったので, 最後まで実行できなかったため出て来たエラーメッセージです. UNIX の cc(ld) は, デフォルトでは標準 C ライブラリ /usr/lib/libc.a(/usr/lib/libc.so) にある関数しか, ライブラリをリンクしません. (自分でチェックしよう) pow は数学関数ライブラリにありますから, このような事が起こります. 標準ライブラリ以外のライブラリにある関数をリンクするには, オプション -l を cc に加えてコンパイルします. ちなみに, 数学関数ライブラリは, /usr/lib/libm.a(/usr/lib/libm.so) ですが, これをリンクするには次のようにコマンドを打ちます.

```
% cc 4-1.c -lm
-
```

-l の後の文字 m の意味は上のファイルから類推できるでしょう[2]. 他のライブラリをリンクする時も, 同様の規則で -l の後のアルファベットを指定します.

scanf についての注意

ソースファイル 3-2.c の 11 行目にある `scanf` は, double 型の変数に値を代入しますので`%lf` としなければなりません. ここが, 関数 `printf` と多少違います.

その他の注意

[1] ほとんどの大学生が常識が無いと言う, 悲しい現実がありますが

[2] もちろん m は mathematical function の m です

関数 pow の引数 (変数) は double 型であるので， 代入する整数は 2.0 のように double 型にしたほうがよろしいです．そのようにしない場合は, たとえコンパイルが通っても結果がおかしくなることがあります.

問題 3.1. *0* 以上の数を，いくつか順に入力して，最後に負の数を入力し，入力された数のうち最大のものを出力するプログラムを作成せよ.

3.2 関係演算子と論理演算子

for 文，if 文などに出てきた大小比較の記号 `<`，`>` は正しくは関係演算子と呼ばれます．関係演算子はこの他に次のものがあります．

<= 等しいか，より小さい．≦と同じ意味である．

>= 等しいか，より大きい．≧と同じ意味である．

!= 等しくない．≠と同じ意味である．

同じく制御文によく使われる論理演算子と呼ばれるものがあります．論理演算子は次の 3 つです．

論理積 記号`&&`,（`A && B` は A, B が共に真のとき真となる）.

論理和 記号`||`,（`A || B` は A, B どちらかが真のとき真となる）.

否定 記号`!`,（`!A` は A が真のとき偽，A が偽のとき真となる）.

これらを組合せて条件式を作ることができます．例えば，下の式は，いつ真になるか考えましょう．

```
5<a && a<10
```

上記の演算の結果は整数に値を持ちます．例えば，`a = 5 < 3`, `a = 3 < 5`, `a = 1 || 0`, `a = 1 && 0`, `a = !1`, `a = !3`, `a = !0`, などとすると `a` に整数が代入されます．どんな値が代入されているか調べてみるとよろしいです．論理積，論理和という言葉の意味がわかるでしょう．他の例も調べてみよう.

3.3 付録: 二次方程式の解法

二次方程式の解を求めるプログラムを if 文を用いて書きます. すぐに思いつくのは, 解の公式を用いた次のプログラムです.

この例でも, 数学関係ライブラリを使うのでそこで使う関数を定義するために最初に math.h というファイルを include しています. 使う関数は, `sqrt` で平方根をとる関数です.

```
/* 二次方程式の実数解の解法１ */
#include <stdio.h>
#include<math.h>

main()
{

  double a, b, c;                          /* 二次方程式の係数 */
  double x1, x2, D;                        /* 求める解と判別式  */

  printf("係数をａ ｂ ｃ 入力して下さい\n");
  scanf(  "%lf %lf %lf", &a, &b, &c);      /* a, b, c に値を代入 */
  if(a==0.0)
    {
  if(b==0.0)
    {  printf("係数がおかしい\n");
     }
  else
    { x1 = -c/b;
         printf("解は%f です\n", x1);
    }
    }
else
    {
  D = b*b - 4.0*a*c;                              /* 判別式Ｄの宣言*/
     if(D >=  0.0)
       {
         x1 = (-b + sqrt(D))/(2.0*a);
         x2 = (-b - sqrt(D))/(2.0*a);
         if(D==0.0)
```

```
        {
          printf("解は重解%f です\n",x1);
         }
       else
       {
          printf("解は%f %f です\n", x1,x2);
         }
     }
   else
     {
       printf("解は虚数解です\n");
      }
   }
}
```

実は解法 1 のプログラムは良いプログラムでありません桁落ちという誤差により, 真の解からのずれが大きい場合が起こります. 例えば, 上のプログラム内の $x1, x2$ の型を float 型にして次の方程式の解を求めてみてください. double 型のときと異なった解が帰ってきます.

$$x^2 + 21000x + 0.76543 = 0.$$

そこで解と係数の関係を用いたプログラムを一般に用います.

問題 3.2. *入力された係数 b が 0 以上であるとき, 解の公式から x2 を求め, x1 を解と係数の関係から求める. 逆に, b が 0 より小さいとき, 解の公式から x1 を求め, x2 を解と係数から求めるプログラムを作成せよ.*

Chapter 4

第4回 制御文（while)

4.1　基本データ型についての復習

変数のデータ型について，関数 printf を使って復習します．変数の型の主なものは，次の４つでした．

char 文字型，1 byte = 8 bit

int 整数型，4 byte = 32 bit

float 小数（現在はあまり使わない）4 byte = 32 bit

double 小数 8 byte = 64 bit

char 型は文字を代入することが多いというだけで，1 byte で表わされる範囲の整数を扱うことも出来ます．

1 byte = 8 bit ですので，0 から $2^8 = 256$ までの大きさの整数，または，$-2^7 = -128$ から，$2^7 - 1 = 127$ までの整数が扱えます．どちらの範囲の整数が使用できるかは，機械（コンパイラー）によって異なりますので，0 から 127 までを使うようにするのが無難でしょう．

整数型は $-2^{32} = -2147483648$ から $2^{32-1} = 2147483647$ まで取り扱うことができます．

次のプログラムを見て，変数の型とその表示の方法について復習しましょう．

```
/* char，int，double，の変数と文字型配列を表示 */
```

```
#include <stdio.h>

main()
{
  char    moji;    /* 主に文字を扱う */
  int     suji;    /* 整数 */
  double  shosu;   /* 実数（浮動小数）*/
  char    bun[80]; /* 文字列は配列に代入する*/

  moji = '!';       /* 文字を代入するときは，' で囲む*/
  suji = 8;        /* 数字を代入するときはそのまま */
  shosu = 7.0;     /* double 型への代入は小数にしてから */

  /* 文字列は"で囲んで strcpy で配列名に代入 */
  strcpy(bun, "is my birthday");

  /* %2.0f とは小数を２桁で小数点以下を０桁で出力せよ */
  /* という意味 */
  printf("%d %2.0f %s %c \n", suji, shosu, bun, moji);

}
```

文字列を char 型の配列に代入するには，関数 strcpy を使います．最初の引き数に char 型の配列の配列名，２番目の引き数には，文字列を"で括って書きます．その際，代入する文字列よりも配列の長さを長くとっておきます．%2.0f は実数を二桁で表示し，そのうち小数点以下を０桁表示することを意味します．

> **注意**
> strcpy(bun, "is my birthday");
> **の部分は**
> bun = "is my birthday";
> **とすることは出来ません．**

その他の部分のプログラムの説明はコメントに書いてある通りです．
では上のプログラムをタイプして実行してみて下さい．（ファイル名 4-1.c）

次に文字型変数に整数を代入して出力する例をみてみましょう．次のプログラムを良く読んで出力結果を予想して下さい，それから実際に作成して実行してみて下さい．（ファイル名 4-2.c）

```
#include <stdio.h>

main()
{
  char    a, b, c, one, two, three;

  /* 次の 10 進数は，16 進数では幾つになるでしょう？ */
  a = 65; b = 66; c = 67;

  /* 10 進整数として出力するときは，%d を使用します */
  printf("%d %d %d\n", a, b, c);

  /* 16 進整数として出力するときは，%x を使用します */
  printf("%x %x %x\n", a, b, c);

  /* %c を使用すると ASCII コードに対応する文字を出力します */

  printf("%c %c %c\n", a, b, c);


  /* 16 進数の代入は数字の前に 0x を書きます*/
  one = 0x31;   /* 文字'1' に対応するコードは 0x31 = 49 */
  two = 0x32;   /* 文字'2' に対応するコードは 0x32 = 50 */
  three = 0x33; /* 文字'3' に対応するコードは 0x33 = 51 */

  printf("%d %d %d\n", one, two, three);
  printf("%c %c %c\n", one, two, three);
}
```

実験してみてわかるように，文字型変数に代入した' 文字' は対応する ASCII コード（整数）を値として持っています．整数 d を文字として扱うとき, %c で出力されるのは, d を 16 進法で表した数に対応するコードが表現される.

4.2 取り込んだ文字列を変換する

前回使用した関数 gets は，キーボードから入力した文字列をプログラムに取り込むものでした．基本的にキーボードからの入力は文字列になります.

前の節で見たように，文字にはコードが割り当てられているので，入力文字がすべて数字の場合，コード表の規則に従うと整数に変換することが出来ます．

また，以下のように関数 sscanf を使って変換も出来ます．(ファイル 4-3.c)

```
/* 文字列を２つの整数に変換する */
#include <stdio.h>

main()
{
  char    mojiretsu[80];  /* 入力文字列 */
  int     m, n;           /* 変換された整数を m, n に保存 */

  printf("２つの整数を空白で区切って入力して下さい >> ");

  /* キーボードからの入力を mojiretsu に格納します */
  gets(mojiretsu);

  /* 文字配列 mojiretsu から整数を取り出します */
  /* m, n の前に & （アンパサンド）が必要です */

  sscanf(mojiretsu, "%d %d", &m, &n);

  /* m, n を使った計算結果の表示 */
  printf("m = %d, n = %d m+n = %d\n", m, n, m+n);
}
```

関数 sscanf の最初の引き数は，文字列の入っている配列の名前です． 2 番目の引き数は取り出すデータの型を”で囲んで記述します．printf と大体同じです．double 型のデータを受け取るときは，%f を使用します．**変換された整数を受け取る変数の前には & が必要です．**

確認

scanf と sscanf の違いを確認しましょう．

確認 上のプログラムで 3 個以上の整数を入力しても 最初の 2 個以外は無視されることを確かめよ．

キーボードからの入力が必要なプログラムについては，gets, sscanf を組合せて使うと幅広いプログラムが作成できます．

4.3 while 文 (フィボナッチ数列)

第 3 回で for 文を用いて簡単な数列の和について求めましたが、フィボナッチ数列 $\{f_n\}$ を **while** 文を使用して考えてみましょう．．while 文は次のように使います.

```
while (条件式) {
  条件式が真ならここを実行
}
```

では次のプログラムを入力し，実行してみて下さい.（ファイル名 4-4.c）
フィボナッチ数列の漸化式は,

$$f_1 = 1, f_2 = 1, f_{n+2} = f_{n+1} + f_n \quad (n = 1, 2, \cdots)$$

です.
この数列を与えられた正の数までの表を作りましょう.

```
#include<stdio.h>

main()
{

  int fn, fn1, fn2, m;    /* 変数 fn, fn1, fn2, m の整数型の宣言*/
  fn = 1, fn1 =1, fn2 =2; /* 変数 fn, fn1, fn2 の初期化  */

  printf("正の整数を入力して下さい  >>");
  scanf("%d", &m);

  printf("%d までのフィボナッチ数列は以下のとおり\n",m);
  printf("-----------------------------------------------\n");
  printf("%d, %d,", fn, fn1);
  while(fn2 <=m){
    fn = fn1, fn1 = fn2, fn2 = fn + fn1;  /*フィボナッチ数列*/
    printf("%d,", fn2);
       }
}
```

例えば 100 までのフィボナッチ数列は，1,1,2,3,5,8,13,21,34,55,89 となります.

4.4 練習問題

問題 4.1. 上のプログラムを改良し, 与えられた整数に対して, それより小さい中で一番近いフィボナッチ数列の値を求めるプログラムを作成せよ.

問題 4.2. 第 *3* 回の問題 *3.1* のプログラムを *while* 文を使って作成せよ.

問題 4.3. ユークリッドの互除法を用いて，２つの正の整数の最大公約数を求めるプログラムを作成せよ．(ヒント：`r = x%y` とすると `x` を `y` で割った余りが，`r` に代入される.)

Chapter 5

第5回 関数

5.1 目標

- while 文の復習
- C 言語における関数の宣言の仕方, 書き方を知る.
- 「再帰」と言う言葉を知る.

5.2 御託宣

今回は関数[1]の書き方をやります. 例えば, main() と言うのは関数ですし, これまでに, printf(), gets(), sscanf() も関数です. 関数と言うのは, 通常は変数 (プログラミングでは引数と言う) を受け取ってその値から別の値を計算して出力するものですが, C 言語では printf の様に計算機に何らかの操作を与えるものも関数と言います. printf(), gets(), sscanf() 等を使う時に ( ) の括弧の中にあるものは, 引数と言われます. printf 等は実は値を返しています. これまでのプログラムでは, その値を使っていないだけです. どのような値を返すかは, **man** コマンドで調べる事ができます.

なお, 関数は数学のように何らかの処理をして値を返すという行為を実現するためだけにあるのではありません. 実際のプログラミングでは, この事

[1]本来は関数ではなく, 函数です. function の音読みが中国語になってそのまま日本に輸入された言葉です. function の本来の意味は, 辞書で索いて下さい. その意味を知っておいた方が, より理解がしやすいです. 例えば, キーボードの最上部に F1 から F15 までの刻印があるキーがありますが, これはファンクションキーと呼ばれています. これを「関数鍵」と訳すと変ですよね.

よりもプログラムの処理を適当な大きさの単位に分割するという目的で主に使われます．人間が一度に把握できる内容は, 概ね画面に表示可能な量の情報であると経験上知られており, それ以下の単位に処理を分割するというのは, 大きなプログラムを間違いなく書くための基本的なテクニークです．今日書くプログラムも, 関数を使わずに実現する事が可能ですが, 実際には今日のようにプログラムする方が良いわけです．これは, 構造化プログラミングのための仕組みの一つです.

5.3 前回までの復習と補足

これまでに, 変数を宣言する事を学びました．さて, 変数が宣言された時点で, その値はいくらなのでしょうか ? 実は, C 言語には, それを決めるための仕様はありません．従って, 変数は代入をしない限りどんな値を持っているかはわからないのです．ちなみに次のプログラムを書いてコンパイルして実行して下さい.

```
/* Variable testing program */

#include <stdio.h>

main()
{
     int r;
     printf("The value of r is %d.\n", r);
}
```

私が実験したところ r の値は -2146890064 でした．例えば, その後 r の値が 0 か否かを判定する while ループがあるとその while ループは実行されません.

5.4 関数の宣言と記述

関数は, 使う前に宣言しなければなりません．その意味で変数と同じ扱いを受けます．コンパイラは, その宣言を見て機械語を作り出すのです.

printf 等の関数をこれまで使って来ましたが, これらも実は宣言してから使っています．これらの宣言は今まで記述して来た,

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

で自動的になされた事になっているのです. この文は, **stdio.h** というファイルを取り込むと言う意味で, stdio.h には, printf が宣言されています. 不等号 <, > で囲まれている場合/usr/include ディレクトリにあるファイルを読み込みます.

```
% less /usr/include/stdio.h
```

を実行して, printf が宣言されている事を確かめてください.

関数の宣言は, 次の形になります.

関数の返り値の型　関数の名前 (引数の型 引数名, ･･･)

関数の宣言では, 次の決まり (言語としての仕様) があります.

- 関数の返り値の型および引数の型として許されるのは, 基本データ型, ポインタ, 構造体, 共有体で, 配列型を使う事はできない.
- 関数の返り値の型が指定されていない時には, 自動的に整数型を返す関数となる.

従って, 今まで何も気にしていませんでしたが, main() は整数型の値を返す関数です.

また, 関数そのもの記述は次のようになります.

```
関数の返り値の型　関数の名前 (引数の型 引数名, ･･･ )
{
        関数内で用いる変数 (局所変数) の宣言 ;

        関数の処理 ;

        return 式 ;
}
```

最後の return の部分で, この名前の関数は, そこの式の値を返します.

5.4.1 補足: main が返す値は ?

上でも述べた通り, main() は整数値を返す関数です. もし, main() が値を返すようにプログラムを書いたら, その値はどうなるのでしょうか ? 実は, こ

れは OS によって扱いが異なります. Macintosh OS では, この値は何も意味を持ちません. しかし, 皆さんがプログラムを作っている UNIX では, その値は Shell 変数 status に格納されます. この値によってコマンドの実行が成功したか失敗したかを知る事ができます. したがって UNIX では, コマンド実行が成功すれば 0, 失敗すればそれ以外の値を返すようにするのが, 望ましいプログラミングです. また, より実践的には, その値によってプログラムがどの段階で失敗しているかを知る事ができるように, プログラムを組みます.

5.5 実習

5.5.1 例 1. ユークリッドの互除法

前回の問題 4.3 にあげていた問題でした. プログラムは以下のようになります.

```
#include <stdio.h>

main()
{
  int a, b, r;       /* 変数a,b,rが整数型であることを宣言*/
  int m;

    r = 1 ;

  printf("二つの整数を入力して下さい >>");
  scanf("%d %d", &a, &b);  /* & を忘れずに */

         while(r > 0)
     {       /* 余りrが0でない限りwhile文の中身を繰り返す*/
        r = a%b;
        a =b;
        b = r;
      }
     printf("最大公約数は%dです\n", a);

}
```

While 文の中味が最大公約数 (GCD) を求めるアルゴリズムです.

この部分を GCD(a,b) 関数として定義し, プログラムを書き直すと以下のように, main 内部がすっきりし処理の内容が理解できやすくなります.

ファイル名を 5-1.c とし書いてみましょう.

```
#include<stdio.h>

int GCD(int, int);

main()
{

  int a, b;

  printf("二つの整数を入力して下さい >>");
  scanf("%d %d", &a, &b);
  printf("二つの整数%d %d の最大公約数は%dです\n", a, b, GCD(a,b));
}


int GCD(int a, int b)
{
     int m, r;
     r = 1 ;

         while(r > 0)
     {        /* 余り r が 0 でない限り while 文の中身を繰り返す*/
        r = a%b;
        a =b;
        b = r;
      }
      return a;
}
```

負の整数も意識したプログラム

```
#include<stdio.h>

int GCD(int, int);
int abs(int);

main()
{
  int a, b;

  printf("二つの0でない整数を入力して下さい >>");
  scanf("%d %d", &a, &b);
  printf("二つの整数%d %d の最大公約数は%dです\n", a, b, GCD(a,b));
}


int GCD(int a, int b)
{
     int m, r;
     r = 1;

     if(abs(b) > abs(a))
    {
      m = a;
      a = b;
      b = m;
     }

         while(r > 0)
     {       /* 余りrが0でない限りwhile文の中身を繰り返す*/
        r = a%b;
        a =b;
        b = r;
      }
      return abs(a);
}
```

```
int abs(int x)
 {
    if(x < 0) x = - x;
    return x;
}
```

5.5.2 例 2: ヘロンの公式を使った 3 角形の面積

次は, 3 角形の 3 辺からその面積を求めるプログラムをヘロンの公式を使って書いて見ます.

この例では, 以前一度使いました数学関数ライブラリを使うのでそこで使う関数を宣言するために, 最初に math.h というファイルを include しています. 使う関数は, sqrt で 平方根をとる関数です. 数学関数ライブラリでどのような関数が利用できるかは, ファイル /usr/include/math.h を読めばわかります[2].

```
/* File name 5-2.c */
/* Computes the area of triangle */

#include <stdio.h>
#include <math.h>

double heron(double a, double b, double c);

main()
{
     double a, b, c, area;  /* a, b, c are edges.  */
     char nyuryoku[80];

     printf("3 辺の長さを空白で区切って入力してください>>");
     gets(nyuryoku);
     sscanf(nyuryoku, "%lf %lf %lf", &a, &b, &c);

     area = heron(a, b, c);
     printf("与えられた 3 角形の面積は %f\n", area);
```

[2] これらの include ファイルを一度は読む事

```
}

double heron(double a, double b, double c)
{
     double s;

     s = (a + b + c)/2.0 ;

     return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

再び数学関数ライブラリの使い方

上のプログラムは, 通常の cc コマンドでは, コンパイルできません.

```
% cc 5-2.c
```

を実行して, 出て来るメッセージを確認してください. 私の環境では, 次のようになりました.

```
Undefined                       first referenced
 symbol                             in file
sqrt                                5-2.o
ld: fatal: Symbol referencing errors. No output written to a.out
```

(意味は第2回を参照)

```
% cc 5-2.c -lm
```

とコンパイルをすればいいですね.

補足 1: 変数のスコープ

上のプログラムでは, heron という関数の中で s という double 型の変数を宣言しています. しかし, この形の宣言では, この変数を参照できるのはこの関数だけで, この値を他で利用する事はできません. 同様に, main で宣言されている nyuryoku も, heron に何らかの形で引数として渡さない限り, 参照する事はできません.

すなわち, 関数内で宣言された変数は, 通常はその関数の中だけで有効なのです. これは, プログラミングにとってとても重要な事です. プログラミングは, つまるところデータの変形方法の指定です. 一つの変数の値が, 至る所で変化するとプログラムの流れを追うのがとても大変になります. だから, そのような事にならないように, 言語の仕様の方で工夫してあると考えてください.

補足 2: エラー処理

上のプログラムで, 例えば 3 辺の値として 1, 2, 5 を代入するとエラーが発生します (実験せよ). これは, 1, 2, 5 を 3 辺とする 3 角形が存在しないからです. ということで, このエラー処理まで考慮に入れたプログラムの例を以下に書いておきます. [3]

三角形ができない場合は, 「三角形になりません」と出力し, プログラムを終了させます.

ここでは, main() は正常終了したときに, shell に 0 を返すようにしました. また, **fprintf** は標準エラー出力にメッセージを出すために使われており, **exit()** は shell に括弧内の値を返した後プログラムを終了する関数です.

```
/* File name 5-2-1.c */
/* Computes the area of triangle */

#include <stdio.h>
#include <math.h>

double heron(double a, double b, double c);

int main()
{
     double a, b, c, area;
     char nyuryoku[80];

     printf("3 辺の長さを空白で区切って入力してください>>");
     gets(nyuryoku);
     sscanf(nyuryoku, "%lf %lf %lf", &a, &b, &c);

     area = heron(a, b, c);
     printf("与えられた 3 角形の面積は %f\n", area);
     return 0;
}

double heron(double a, double b, double c)
{
     double s, s2;
```

[3] 実は, センター試験のプログラムも数字以外の文字を入れるとエラーを発生させます.

```
    s = (a + b + c)/2.0 ;
    s2= s*(s-a)*(s-b)*(s-c);
    if(s2 <= 0){
    fprintf(stderr,"%lf, %lf, %lf は三角形になりません\n",a,b,c);
     exit(1);
    }
    return sqrt(s2);
}
```

5.5.3 例 3: 再帰 (recursion)

再帰と言うのは, 感覚的に漸化式をそのままプログラムする様なものです. 例えば, $S(n) = \sum_{k=1}^{n} k$ は漸化式 $S(n) = S(n-1) + n,\ S(1) = 1$ を満たします. ほとんどの高級言語では, これをそのままプログラムで表現する事ができます. C 言語では, 次のようになります. 次の例の sum がまさに上の漸化式の定義通りになっています.

```
/* filename 5-3.c */

#include <stdio.h>

int sum(long n);

main()
{
       int n;

       printf("正の整数を入力してください >> ");
       scanf("%d", &n);
       printf("1 から %d までの和は, %d です. \n", n, sum(n));
}

int sum(long n)
{
      if ( n==1 )
           return 1;
      else
           return sum(n-1)+n;
}
```

5.6 練習問題

問題 5.1. 再帰を用いて *Fibonacci* 数列 $f_n = f_{n-1} + f_{n-2}$, $f_0 = f_1 = 1$ の第 n 項を出力するプログラムを書け.

問題 5.2. 入力された整数 x の n 乗を求めるプログラムを関数 `power(x,n)` を定義して書け.

問題 5.3. 階乗 $n!$ を求めるプログラムを, 再帰を用いた関数 `kaijo(n)` を定義して書け.

問題 5.4. さらに拡張して *2* 項係数 $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$ を求めるプログラムを関数 `comb(n,k)` を定義して作れ.

Chapter 6

第6回 関数II

6.1 目標

- 関数 sin をつくる.
- 台形公式による定積分の計算

6.2 sin 関数の作成

sin 関数は C 言語の数学ライブラリーに定義されておりますが, 勉強のために自分の sin 関数 (= mysin) を作ってみましょう.

$x = 0$ におけるマクローリン展開は,

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

です. よって, mysin(double) のアルゴリズムは, 再帰性を使って書くと,

```
double mysin(double x)
{
    double xx, kai, diff, sum;
    int p;

    p = 3;
    sum = x;                             /* 第一項までの和 */
    xx = x * x;
    kai = 1;                             /* 分母の初期化 */
```

```
    while(1)                          /* 無限ループ  */
    {
      x = -x*xx;                      /* 分子の計算 */
      kai *= (p - 1)*p;
      p += p + 2;
      diff = xx/kai                  /* sin 関数の p 項 */
      sum += diff;                   /* 第 p 項までの和 */
      if(fabs(diff) < eps)           /* 求めた第 p 項が  */
       {                             /* 許容誤差 eps 以下で */
         break;                      /* ループを抜け出る */
         }
      }
        return sum;
}
```

作った mysin() 関数とライブラリーに登録されている sin() 関数を比較するプログラムを書きましょう. 但し, 範囲は $0 \leq x \leq 3.0$ です.

ファイル名は, 6-1.c です.

```
#include<stdio.h>
#include <math.h>
#define eps 10e-11

double mysin(double);

main()
{
   double x, y;

   printf("\tx\t\tsin x\t\tmysin x\n");
   for(x=0.0; x < 3.1; x += 0.1)
   {
     printf("\t%3.1f\t%12.10f\t%12.10f\n", x, sin(x), mysin(x));
     }

}
```

```
double mysin(double x)
{
    double xx, kai, diff, sum;
    int p;

    p = 3;
    sum = x;                          /* 第一項までの和 */
    xx = x * x;
    kai = 1;                          /* 分母の初期化 */

    while(1)                          /* 無限ループ   */
    {
      x = -x*xx;                      /* 分子の計算 */
      kai *= (p - 1)*p;
      p = p + 2;
      diff = x/kai;                  /* sin 関数の p 項 */
      sum += diff;                   /* 第 p 項までの和 */
      if(fabs(diff) < eps)           /* 求めた第 p 項が   */
       {                             /* 許容誤差 eps 以下で */
         break;                      /* ループを抜け出る */
         }
      }
        return sum;
}
```

main 内の printf 関数で使われている`%12.10f` の意味は、実数を 12 桁で表示し, そのうち小数点以下 10 桁とする, ということです.

while 文内の if 文にある許容誤差 eps は main() 関数の前で定義されています. 特定の数や言葉などを使う場合は

`define（定義名）（定義したいもの）`

であらかじめ定義しておくと, エラーを防げると同時に, 間違いを訂正しやすいです.[1]

上の場合, 小数点以下 10 桁を表示しますから, 許容誤差として 10^{-11} でよろしいわけです.

[1]間違いを訂正することを「デバックをする」といいます

`cc 6-1.c -lm` でコンパイルした後 `./a.out` で実行さすると, データが多いためウインド ウないで全て見れないかも知れません. そのような場合, リダイレクションをつかってデータをファイルに保存すればよろしいです. 例えば,

```
%./a.out > sindata
```

とすれば, データは, `sindata` に保存されていますので, less コマンドでファイルの中身を見ればよろしいです.

6.3 台形公式による定積分の計算 (変数のキャスト)

計算機による数値計算の 1 例として, 台形公式を用いた定積分の計算を取り上げます. ここでは, $\int_0^1 \frac{4\,dx}{1+x^2} = \pi$ を取り上げましょう.

以下は「高等学校 数学C （数研出版)」から抜粋です.

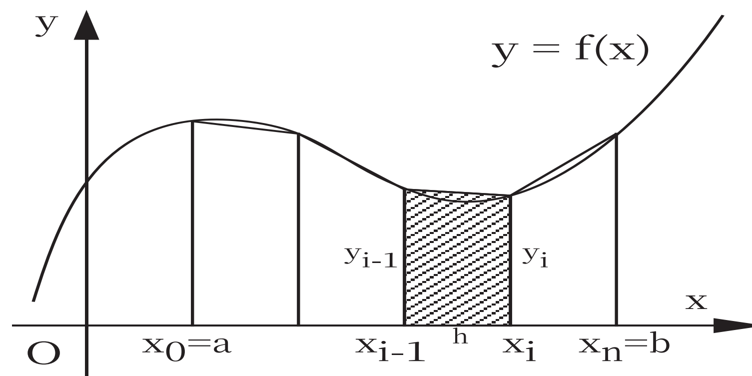
連続な関数 $y = f(x)$ は，区間 $[a, b]$ で常に $f(x) \geq 0$ であるとする．図 ?? のように，この区間を n 等分して，分点を小さい方から順に

$$a = x_0 < x_1 < \cdots < x_n = b$$

とし，次のようにおく．

$$h = \frac{b-a}{n}, \quad y_i = f(x_i)$$

小区間 $[x_{i-1}, x_i]$ において，区間の幅 h を高さとし，y_{i-1} と y_i をそれぞれ，



上低，下低とする台形の面積を S_i とすると

$$S_i = \frac{h}{2}(y_0 + y_1), \quad S_2 = \frac{h}{2}(y_1 + y_2), \quad \cdots, S_n = \frac{h}{2}(y_{n-1} + y_n)$$

これらの台形の面積の総和を T_n とすると

$$T_n = S_1 + S_2 + \cdots + S_n$$

は，n が十分に大きいとき，定積分 $\displaystyle\int_a^b f(x)\,dx$ の近似値を与える．

したがって，定積分の近似式を与える次の **台形公式** が得られる．

$$\int_a^b f(x)\ dx \ \approx\ \frac{b-a}{2n}\{y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n\}$$

この台形公式は，常に $f(x) \geq 0$ でない場合でも成り立つ．

それでは例を考えましょう.
プログラムの中で, daikei と言う関数が台形公式を与え, それの引数は順に, 積分区間の右端, 左端, 分割の個数, と関数です.

math.h を include して, main() の中の daikei という関数の引数の fun の部分を例えば sin に変えると, このままで $\int_0^1 \sin x dx$ の近似値計算ができます (この場合当然数学関数もリンクしなければなりません). 他の引数に付いても同様ですし, 関数 fun の記述を変える事により, いろいろな定積分の近似計算ができます.

プログラムで注意して欲しいのは, 下から 9 行目の fun の引数の計算 `a + interval*i` の部分です. ここでは a, interval は double 型, i は int 型で宣言されています. このような異なる型の変数の間の計算は, 変数の型変換が自動的に行われます. 今の場合 i を double 型に変換して計算が実行されます.

型変換の規則は, **char** $<$ **int** $<$ **double** で与えられます. すなわち, 計算時にはこれらの内, より右側の型に値の型を合わせて計算されます.

```
/* ファイル名 6-2.c */

#include <stdio.h>

double daikei(double a, double b, int n);
double fun(double x);

main()
{
     printf("Pi is about %1.20lf.\n", daikei(0.0, 1.0, 10000));
}

double daikei(double a, double b, int n)
{
     double sum=fun(a);
     double interval=(b-a)/n;   /* The length of the small interval */
     int i;

     for (i=1; i < n ; i++)      /* add to (n-1)-th value */
               sum = sum + 2.0*fun(a+interval*i);
     return interval*(sum + fun(b))/2.0;
```

```
}


double fun(double x)
{
      return 4.0/(1.0 + x*x);
}
```

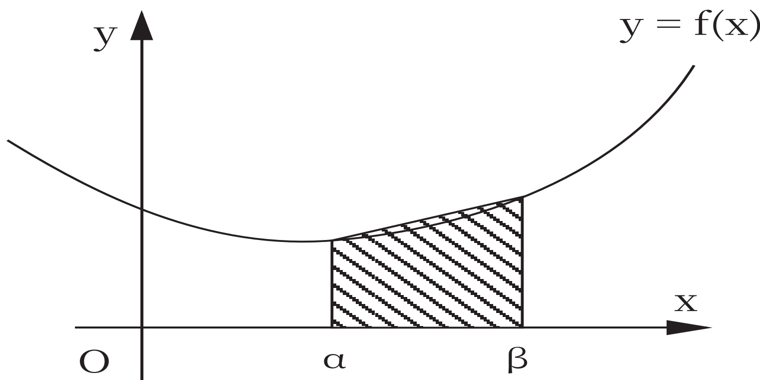
上のプログラムは, π を近似的に求める方法の一つです.

6.3.1 台形公式と誤差

定積分に関する次の部分積分法の公式

$$\int_{\alpha}^{\beta} h(x)g'(x)\,dx = [h(x)g(x)]_{\alpha}^{\beta} - \int_{\alpha}^{\beta} h'(x)g(x)\,dx \tag{6.1}$$

を用いると，台形公式による数値積分の誤差について調べることができます.



定積分 $\int_{\alpha}^{\beta} f(x)\,dx$ は，図 ??の斜線の面積で近似するとその誤差 e は，次のようになります.

$$e = \frac{\beta-\alpha}{2}\{f(\alpha)+f(\beta)\} - \int_{\alpha}^{\beta} f(x)\,dx \tag{6.2}$$

部分積分法の公式を用いて，まず次の等式が成り立つことを示そう.

$$e = \frac{1}{2}\int_\alpha^\beta (x-\alpha)(\beta-x)f''(x)\,dx \tag{6.3}$$

公式 (1) において，$h(x)$ として $(x-\alpha)(\beta-x)$，$g(x)$ として $f'(x)$ をとると，(3) の右辺は次のようになります.

$$\frac{1}{2}[(x-\alpha)(\beta-x)f'(x)]_\alpha^\beta - \int_\alpha^\beta (-2x+\alpha+\beta)f'(x)\,dx \tag{6.4}$$

$$= \frac{1}{2}\int_\alpha^\beta (2x-\alpha-\beta)f'(x)\,dx \tag{6.5}$$

再び公式 (1) において $h(x)$ として $2x-\alpha-\beta$，$g(x)$ として $f(x)$ をとると，(4) の式は次のようになります.

$$\frac{1}{2}\left[(2x-\alpha-\beta)f(x)\right]_\alpha^\beta - \int_\alpha^\beta f(x)\,dx$$

これは (2) の右辺に等しいから，(3) の等式が成り立ちます.

前節において，定積分 $\displaystyle\int_{x_{i-1}}^{x_i} f(x)\,dx$ を S_i で近似したときの誤差は

$$E_i = \frac{x_i - x_{i-1}}{2}f(x_{i-1}) + f(x_i) - \int_{x_{i-1}}^{x_i} f(x)\,dx$$

これは，公式の (2) で $\alpha = x_{i-1}$，　$\beta = x_i$ とおいた式であるから (3) により

$$E_i = \frac{1}{2}\int_{x_{i-1}}^{x_i} (x-x_{i-1})(x_i-x)f''(x)\,dx$$

ここで

$$\frac{1}{2}\int_{x_{i-1}}^{x_i} (x-x_{i-1})(x_i-x)\,dx = \frac{h^3}{12} \quad \text{ただし，} h = x_i - x_{i-1}$$

であるから，小区間 $[x_{i-1}, x_i]$ において，$f''(x)$ の最大値を M_i，最小値を m_i とすると，次の公式が成り立ちます.

$$\frac{h^3}{12}m_i \leq E_i \leq \frac{h^3}{12}M_i$$

関数 $f''(x)$ が連続であるとき，分割の個数 n を大きくとって，小区間の幅 h を小さくすると

$$m_i \ \approx \ f''(x_i) \ \approx \ M_i \quad \text{したがって} \quad E_i \ \approx \ \frac{h^3}{12}f''(x_i)$$

となる．よって，定積分 $\displaystyle\int_a^b f(x)\,dx$ の台形公式による近似値の誤差は

$$E = T_n - \int_a^b f(x)\ dx = \sum_{i=1}^n E_i \ \approx \ \frac{h^2}{12}\sum_{i=1}^n f''(x_i)h$$

ここで，関数 $f''(x)$ の区分求積法による数値積分を考えると

$$\sum_{i=1}^n f''(x_i)h \ \approx \ \int_a^b f''(x)\ dx = [f'(x)]_a^b = f'(b) - f'(a)$$

ゆえに

$$E \ \approx \ \frac{h^2}{12}\{f'(b) - f'(a)\}$$

このことから，この誤差は，小区間の幅の平方 h^2 にほぼ比例し，$|f'(b) - f'(a)|$ が小さければ，誤差も小さいことがわかります.

6.4 練習問題

問題 6.1. 関数 `mycos` を定義し, $0.0 \leq x \leq 3.0$ の範囲の値を求めるプログラムを書き, 出力せよ. 但し, 許容誤差は 10^{-10} の範囲で.

問題 6.2. *Euler* 関数 $\phi(n)$ は, *2* 以上の整数 n に対して *1* から n までの数で, n と互いに素な素数の個数として定義される. 例えば, $\phi(3) = 2$, $\phi(4) = 2$ となる. オイラー関数を C 言語の関数と定義して, n を入力したときその値を出力するプログラムを書け.

(**ヒント** *:* (1) $1 \sim n$ までの $GCR(i, n)$ を調べるアルゴリズムを考える. (2) 次にオイラー関数の特徴をと使う :

$$n = p_1^{r_1} \times p_2^{r_2} \times \cdots \times p_l^{r_l}$$

(p_i は素数, $r_i \geq 1$) のとき

$$\phi(n) = \phi(p_1^{r_1}) \times \cdots \times \phi(p_l^{r_l})$$

)

問題 6.3. 台形公式を用いて $\log 2 = \int_0^1 \frac{1}{1+x} dx$ を計算するプログラムを書き, 区間の分割数を *10, 100, 1000* としたときの値を比較せよ.

Chapter 7

第7回 関数とポインタ（Call by Value, Call by Reference）

7.1 今までの講義に関する確認と補足

7.1.1 小数型の入出力 (printf, sscanf)

この講義では，変数の値の出力は printf，出力には sscanf を使用していますが，小数型 (double) の入出力について確認しておきましょう．小数型 (double) の変数 x にキーボードから入力するには，

```
char mojiretsu[80];
double x;

printf("数字を入力して下さい >>");
gets(mijiretsu);
sscanf(mojiretsu, "%lf" &x); /* & を忘れないこと */.
```

のようにします．

これに対して，double 型の変数 x を出力するときは，

```
printf("%f", x);
```

または，

```
printf("%g", x);
```

とします．入力は，`%lf` 出力は，`%f` となり形式が異なります．

これに対して，整数型 (int) の変数は，入力も出力も `%d` を用います．

7.1.2　式の値

代入文は，**式自体が値を持ちます**．この値は，左辺に代入された値と同じです．下の例で確認してみましょう．

```
a = 1;
while (a) {
    if ( (m % n) == 0 ) {
        break;
    }
    m = n;
    n = r;
}
printf("%d\n", n);
```

２行目の while 文の（）の中には条件式以外も書くことが出来ます．ここが 0 のときは，繰り返しを行わず，0 以外のときは，繰り返しを行います．

３行目の if 文は，r に m を n で割った余りを代入し，その値が 0 であれば，ループを抜けるという意味になります．

7.1.3　実行ファイルの名前を変えるには

file.c という名前のファイルを cc コマンドでコンパイルすると，a.out という実行ファイルが作成されましたが，UNIX のコマンド mv を使って名前をかえることが出来ます．

```
mv  a.out  excute
```

このようにすると，実行ファイルは，excute という名前になりますから，実行させたいときは，excute とタイプします．また，コンパイルの祭に，a.out というファイルを作らず，直接，実行ファイル excute を作るには，次のようにタイプします．

```
cc  file.c  -o  excute
```

7.2　関数とポインタ

前回, 前々回と関数の使い方について学習しました．関数の引数にポインタと呼ばれる変数を渡すことで，本格的なプログラムが組めるようになります．

ポインタは，C 言語の大きな特徴であり，最も大事な部分ですので，参考書などを良く読んでしっかり理解するようにして下さい．カーニハン・リッチーの「プログラミング言語C」では，５章でポインタについて扱っています．

7.2.1 引数はコピーされる

まずは，復習のために "普通の" 変数を引数に持つ関数を考えましょう．
２つの変数の値を入れ替えて出力するプログラムは次のようになります．

```
/* 変数の値を取り替えるプログラム */
#include <stdio.h>

main()
{
  int    x, y, tmp;

  x = 11;  y = 20;          /* 初期値を設定します */

  printf("%d %d\n", x, y);  /* 初期値が出力されます */

  tmp = x;    /* tmp に 11 が代入されます */
  x = y;      /* x に 20 が代入されます */
  y = tmp;    /* y に 11 が代入されます */

  printf("%d %d\n", x, y);  /* 入れ替えた値が出力されます */
}
```

ここで，変数を取り替える処理を行っている部分を swap という関数にしてみたいと思いますが，次のプログラムは，実は期待通りには動きません．

```
/* 変数の入れ替え（関数版，失敗編） */

#include <stdio.h>

void swap(int xx, int yy); /* 使用する関数は，事前に宣言します
                            * 関数 swap には return 文が無いので
                            * その型は void です.
                           */

main()
{
  int    x, y;

  x = 11;  y = 20;          /* 初期値を設定します */

  printf("%d %d\n", x, y);  /* 初期値が出力されます */

  swap(x, y);               /* swap 関数を呼び出します */

  printf("%d %d\n", x, y);  /* 初期値が出力されます */
}

void  swap(int xx, int yy)
{
  int  tmp;

  tmp = xx;
  xx = yy;
  yy = tmp;

}
```

このプログラムが何故，変数の交換を行ってくれないか考えてみます．このプログラムでは，main で宣言した変数 x，y が引数として関数 swap に受け渡されていますが，関数 swap は**変数そのものでなく，値のコピーを受け取ります**．図 7.1 を見て下さい．

上のプログラムでは，x，y に代入された値 11, 20 がそれぞれ，**swap 内部の変数** xx，yy の初期値として代入されますが．swap の最後の 文で main

Figure 7.1: 変数の値渡し

に戻ったとき，main は swap 内部の変数で何がおこったのか知りません．従って，main の変数 `x`，`y` は，swap の後でも同じ値を持ち続けます．

> **注意 1** 上の swap ように関数値を返さない関数を void 型関数と呼びます．main 関数は実は void 型関数です．void 型関数の形式は関数値を返す関数と形式は同じですが, `a = swap(x,y)` のような指定は関数値を返さないので意味がありません．

> **注意 2** swap の変数の宣言で `int xx`，`int yy` の部分をそれぞれ，`int x`，`int y` と変えればうまく動くと思う人がいるかもしれませんが，結果は同じです．関数の内部で宣言された変数は，他の関数の変数と同じ名前であっても，別の変数として扱われます．

7.2.2 変数のアドレス

C 言語では，使用する変数は（関数も）すべて，事前に宣言しなければなりません．宣言された変数は，コンピュータは宣言された変数の値を保存しておくための箱（領域）を確保します．この箱には住所（アドレス，番地）が付けられています．C 言語の特徴の１つとして，アドレスを変数として扱え

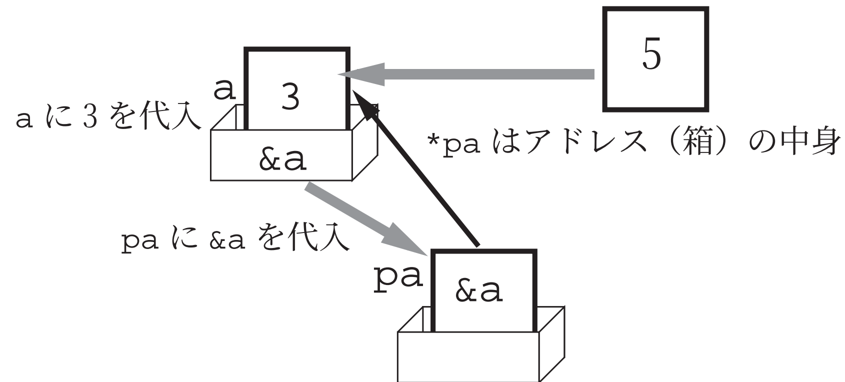


Figure 7.2: ポインタ概念図

るということがあります．アドレスは，変数の箱の位置を指し示すという意味で，ポインタとも呼ばれます．図 7.2 を見て下さい．

ポインタを使用したプログラムの例を見てみましょう．次のプログラムを良く読み，実行結果を予想して下さい．実際にプログラムを組んで結果を確かめてみて下さい.（ファイル名 7-1.c）

```
#include <stdio.h>

main()
{
  int    a;      /* a には整数がはいります */

  /*
   * ポインタを値にもつ変数は次のように * をつけて宣言します
   * 下の例の場合 pa には整数型の変数のアドレスが入ります
   */
  int    *pa;

  a = 3;

  pa = &a;    /* &a とは a のアドレスという意味です */
  *pa = 5;    /* *pa とは pa のアドレスの中身という意味です */

  printf("a = %d\n", a); /* 何が出力されるでしょうか？ */

}
```

このプログラムでは，変数 a は最初に 3 が設定されたあと，何も代入されていません．にもかかわらず，a の値は書き換えられています．

このように，ポインタ変数を利用すると，変数の値を間接的に変更することが出来ます．これを利用すると**関数にアドレスを渡す**ことでもとの変数を書き換えられることがわかります.（図 7.3）

> **注意**　ポインタ型の変数にはアドレスが入りますが，この場合でもそのポインタの行き先の変数の型により，宣言の型も異なります．例えば，double 型の変数のアドレスを格納するポインタ px の型の宣言は，
>
> ```
> double      *px;
> ```
>
> または，
>
> ```
> double *     px;
> ```
>
> とします.

最初のプログラムを正しく書き換えてみましょう．最初のプログラムと図 7.3 を参照しながら，次のプログラムの欠けている部分を補い，完成させて下さい．（ファイル名 7-2.c）「プログラミング言語C」の５章を参照する

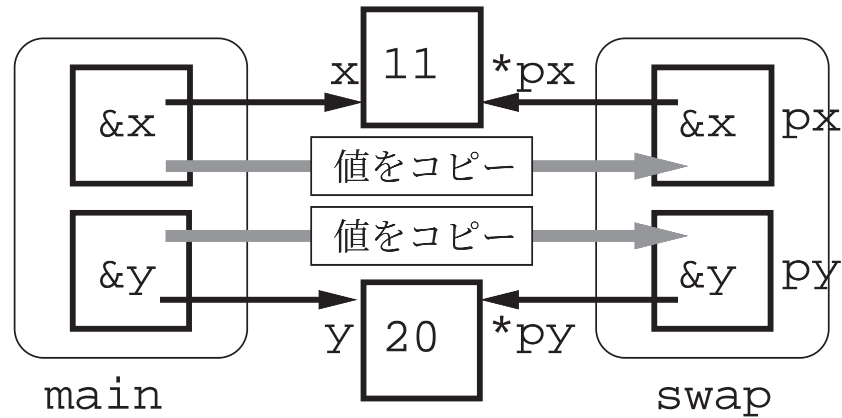


Figure 7.3: 変数のアドレス渡し

のも良いでしょう．

```
/* 変数の入れ替え（ポインタ使用） */

#include <stdio.h>

void swap(int *px, int *py); /* ポインタ変数は * をつけて宣言
                               * 返り値が無いので void で宣言しま
す
                               */

main()
{
  int    x, y;

  x = 11;  y = 20;          /* 初期値を設定します */

  printf("%d %d\n", x, y);  /* 初期値が出力されます */

  swap(&x, &y);             /* swap にアドレスを渡します */

  printf("%d %d\n", x, y);  /* 新しい値が出力されます */
}

void swap(int *px, int *py)
{
  int  tmp;

  tmp = *px;
  *px = *****;    /* ここを補って下さい */
  *py = *****;    /* ここを補って下さい */

}
```

もう少し，例を見てみましょう．次のプログラムは，四則演算を行う関数をポインタを利用して書いたものです．先程と同様，欠けている部分を補ってプログラムを完成させて下さい．（ファイル名 7-3.c）

```
#include <stdio.h>

/* 使用する関数は事前に宣言します*/
void wasa(int a, int b, int *pwa, int *psa);
void sekishou(int a, int b, int *pseki, double *pshou);

main()
{
    int       a, b;
    int       wa, sa, seki;
    double    shou;

    a = 2; b = 7;

    wasa(a, b, &wa, &sa);
    sekishou(a, b, &seki, &shou);

    printf("a と b の和は %d 差は %d です\n", wa, sa);
    printf("a と b の積は %d 商は %f です\n", seki, shou);

}

void wasa(int a, int b, int *pwa, int *psa)
{
    *pwa = ****;    /* ここを補って下さい */
    **** = ****;    /* ここを補って下さい */
}

void sekishou(int a, int b, int *pseki, double *pshou)
{
    double    fa, fb;

    *pseki = *********;    /* ここを補って下さい */

    /* 小数型にするために 1.0 を掛けます. */
    fa = 1.0 * a;
    fb = 1.0 * b;
    *pshou =  ********;      /* ここを補って下さい*/
```

```
}
```

7.3 練習問題

問題 7.1. 確立論や統計学で重要な「正規分布」に現れる関数

$$f(x) = \frac{1}{\sqrt{2}} e^{-\frac{x^2}{2}}$$

を *-1.96* から *1.96* まで積分した値の近似値を台形公式，区分求積を用いて求めるプログラムを作成せよ.（ヒント：答えは，*0.950* に近い）分割の個数を *10, 100, 1000, 10000* とし，*0.950* との誤差を比較せよ.

問題 7.2. *Euclid* の互除法のプログラムを関数化して，*main* 中で

```
euclid(m, n, &gcm);
```

とすると *gcm* に *m, n* の最大公約数が代入されるようなプログラムを作成せよ.

問題 7.3. 正の整数の入力に対し，それが素数であるかどうかを判定するプログラムを作成せよ.

Chapter 8

第 8 回 配列 I

8.1 目標

- C 言語における配列型変数の宣言の仕方, 使い方を知る.

8.2 配列とは

今回は変数の配列について詳しくやります.
　今まで書いてきたプログラムでは,

```
char nyuryoku[80];
```

の部分が, char 型の変数の配列の宣言になっています.
　配列は, 同じ型の変数を複数個並べた変数です. 並べ方は, 1 次元的であると限る必要は無く, 2 次元的, 3 次元的であっても構いません. 数学でいうと, ベクトルや行列 (さらにテンソル) だと考えてください.
　このように, 基本データ型からより複雑な別のデータ型を定義する事は, 高級言語では必ずできるようになっています. プログラムに有効なデータ型を定義するのは, 重要な事です.

8.2.1 1 次元配列の宣言

1 次元配列は, 次の形で宣言します.

配列の型　配列変数名 [配列の大きさ];

　例えば上の

```
char nyuryoku[80];
```

は, char 型の変数を 80 個並べた配列の宣言になります. すなわち, コンパイラは 80 文字分の文字を納める記憶領域を確保します. この時, 配列の各成分の値は, nyuryoku[0], nyuryoku[1], ···, nyuryoku[79] で参照します. 数学とは違い添字は, 0 から始まります.

8.2.2 2 次元以上の配列の宣言

2 次元配列は, 次の形で宣言します.

```
配列の型 配列変数名 [配列の第 1 成分の大きさ][配列の第 2 成分の大きさ];
```

例えば, double 型の成分を持つ 4 行 5 列の行列を宣言する場合には,

```
double matrix[4][5];
```

とします. 数学から想像される, double matrix[4, 5] とは異なる事に注意して下さい.

3 次元以上の配列の宣言も同様にできますが, それが必要になる事は少ないので, ここでは省略します.

8.3 素数表の作成 (エラトステネスの篩 (sieve))

配列を用いたプログラムの例として, エラトステネスの篩法で素数表を作成して見ます. エラトステネスの篩は, 小さい素数から順にその数の倍数を消去していくことにより素数を残していく方法で, 素数表の作成には, とても効率が良い事が知られています.

よく知られている事実として, 素数は無限に存在します. 簡単な証明をあげます：

定理 [ユークリッド 原論]
素数は無限に存在する.
(証明)
今仮に素数が有限個存在するとします. 即ち,
$p_1, p_2, \cdots, p_n$.
但し, $p_1 < p_2 < \cdots < p_n$.
ここで, $n = p_1 \cdot p_2 \cdots p_n + 1$ とおくと, これは, 明らかに $p_n < n$ であり, かつ, 新しい素数となります. (何故か?)

故に, 最初の仮定に矛盾するので, 素数は無限個存在する.
(証明終わり)

さて, エラトステネスの篩というのは, 以下のアルゴリズムで素数以外の整数を削除していきます. 簡単な場合として, 38までの整数から素数を拾い上げる方法を考えてみましょう.
配列を利用しますので, 0から数え上げます.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
```

(1) 0と1を素数の候補から外す.

(2) 38以下の2の倍数を素数の候補から外す.

(3) 38以下の3の倍数を素数の候補から外す.

(4) 以下順に, 38以下の $5, 7, 11, \cdots, 38$ の2倍数以上の整数を素数の候補から外す.

(5) 残った整数が素数である.

この場合素数表は,

```
2 3 5 7 11 13 17 19 23 29 31 37
```

となります.

これをプログラムで実行させる場合は, 配列を用いて与えられた整数が素数か否かのシンボルを与えて, 最後に, 素数のシンボルをもつ整数のみを出力すればよろしいです. 即ち, あらかじめ, 全ての整数 i に `isprime[i] = 1` というシンボルを与え, 上の (1), (2), (3) のアルゴリズムを行うに従って, 該当する整数 i のシンボルを `isprime[i] = 0` と変えていき, 最後に, `isprime[i] == 1` の整数 i を出力すればよいことになります.

注意
Cでは, a と b が等しいことは, $a == b$ とかく.

では, 1000までの素数表を出力するプログラムを作成しましょう.

8.4 演習

以下の問題の (1) ～ (6) の空欄を埋めよ. ファイル名は creoportno.8-1 とする.

8.4.1 例 1:1000 までの素数表を出力するプログラム

注意

プログラムの最初の方にある,

```
#define MAX 1000
```

は, マクロ置換というもので, この 1000 の値を変えてコンパイルしなおせば, その値までの素数表が出力されます. (ただし, 配列の添字が取れる範囲の制限が付きます. この値はおそらく処理系依存です.) これを, 1000 とするとプログラムの変更のたびに, 4 箇所の値の変更が必要になり, プログラムのミス (バグ (bug) と言われる) の原因になりますので, このような工夫をします.

また, 配列も他の変数と同様宣言した時の初期値は不定です. これをきちんと初期化しないと, プログラムは動きません.

```
/* Print primes to MAX  */

#include <stdio.h>
#define MAX 1000

main()
{
        char  isprime[MAX];          /* 素数判定の配列  */
        int   i, j;

        for((1))
        {   (2)         }               /* 配列isprime[i]の初期化 */

        /* 篩のアルゴリズム始まり */
        isprime[0] = isprime[1] = (3);      /* 0 と1は素数ではな
い */
        for (i=2; i < MAX; i++){
                if (isprime[i]==1){         /* i が素数のとき */
                        for ((4)){
```

```
                              isprime[i*j] = (5);
                              /* i の 2 倍数以上の倍数を削除 */
                                   }
                                    }
                                 }             /* 篩のアルゴリズムの
終了 */

        for (i=0; i <MAX ; i++){              /* 素数の表示 */
                if ((6))
                        printf("%3d\t", i);  /* 3 桁分の幅で出力
  */
                                }
}
```

8.5 配列と関数および前回までの復習

関数の宣言の時に述べましたように, 配列は関数の返り値や引数にする事はできません. さらに, 配列全体を代入する事もできません. しかし, 関数間での参照ができないと, プログラミングをする上で, とても不便です. 配列は, 前回述べたポインタを通して, 関数間での参照をします[1].

例えば,

```
int a[10];
```

と宣言した時, a は配列の先頭要素へのポインタです. すなわち, a と `&a[0]` はこの意味で, 全く同じ意味を持ちます. このことを利用して, ポインタ経由で配列を関数にわたす事ができます. 次の例は, このことを利用して内積関数を作り, 3 次元空間内の 2 つのベクトルのなす角を弧度法で出力するプログラムです.

8.5.1 例 2: 内積を用いた角度の計算

このプログラムで, 関数 `inn_prod` と言う関数は, double 型のポインタを引数にしていますが, それに double 型の配列変数名を渡しています. もちろん, これは内積を計算する関数です.

[1] この辺の ANSI の規格は不徹底で, 構造体が関数の引数, 返り値にできたり, 代入ができたりしますので, 配列もこれを利用すれば, 引数返り値にもできますし, 代入もできます. 配列だけを特別扱いするのは, 規格上明らかにおかしな事です

また, 最後の角度の計算で **acos** と言う関数が使われていますが, これは cos の逆関数で, 返り値はラジアン単位です. これは, 数学関数ライブラリに入っています. また, 数学関数ライブラリを使いますから, コンパイル時にはオプション -lm を付けて下さい.

```
/* File name 8-2.c                                 */
/* Calculate the angle of two 3-dimensional vectors */

#include <stdio.h>
#include <math.h>

double inn_prod(double *, double *);

main()
{
        char nyuryoku[80];
        double a[3], b[3];
        double angle;

        printf("最初のベクトルの x, y, z 成分を空白で区切って入力し
て下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &a[0], &a[1], &a[2]);

        printf("次のベクトルの x, y, z 成分を空白で区切って入力して
下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &b[0], &b[1], &b[2]);

        angle = acos(inn_prod(a, b)/sqrt(inn_prod(a, a)*inn_prod(b, b)));
        printf("2 つのベクトルのなす角はおよそ %f ラジアンです\n", angle);
}

double inn_prod(double *a, double *b)
{
        return a[0]*b[0]+a[1]*b[1]+a[2]*b[2];
}
```

8.6 さらにポインタ

この講義で幾度とでて来る言葉ですが, 上の配列との関係でもう少し詳しく述べておきます.

そもそもポインタとは,「場所を指し示すための物」という意味で, C 言語でもこの意味で用います.

前回の講義で出ましたが,

```
int *pa;
```

の宣言は, pa をポインタ型の変数として宣言します. すなわち, pa は整数を指し示すための物です.

今回の配列の宣言

```
int a[10];
```

でも a は配列の先頭へのポインタであると言いました.

この 2 つの違いうちの重要な 1 つが,「変数」か否かです[2]. すなわち, 最初の宣言では, pa は int 型へのポインタ変数で, その値をプログラムの中で, 代入やインクリメント (1 増やす), デクリメント (1 減らす) 等の操作で変化させる事ができます. これに反して, 2 番目の宣言では, a は変数ではありません. すなわち, a に値を代入したり, a の値を変化させようとすると, コンパイラエラーになります.

すなわち,「ポインタ変数」と一般的な「ポインタ」と言う言葉は, 集合の包含関係にあることを, 注意してください.

8.7 例 3: 配列とポインタ変数の違い

上に述べた違いを, 次の 3 つで実験してください. 最初は, きちんと動くプログラムです. **putchar** は文字を出力するライブラリ関数です. ポインタや配列は定義と同時に初期値を与えていますが, これらの解説は, 後でします.

プログラムの中で, ポインタ変数の値を変化させている事に注意してください. この変化がどのような影響を及ぼすかは, 後で述べるかも知れませんが, 詳しくは何らかの本を読んでください.

```
/* File name 8-3.c */
#include <stdio.h>
```

[2]他にも違いがありますが, それは今後の講義で触れるかも知れません

```
main()
{
        char *p="This is a test.\n";

        while(*p){
                putchar(*p);
                p++;
                 }
}
```

しかし, p を配列型で宣言した次のプログラムは, コンパイル時にエラーになります. p++ と p の値を変化させようとした事に問題があります.

```
/* File name 8-4.c */

#include <stdio.h>

main()
{
        char p[]="This is a test.\n";

        while(*p){
                putchar(*p);
                p++;
                 }
}
```

しかし, 良く似ていますが次のプログラムはきちんと動きます. ここで *(p+i) という部分では, p+i が指す場所にある内容を取ると言う意味で, p の値を変化させていないからです.

```
#include <stdio.h>

main()
{
        char p[]="This is a test.\n";
        int i=0;
```

```
        while(*(p+i)){
                putchar(*(p+i));
                i++;
                     }
}
```

8.8 文字列に関する注意

C 言語では, 文字列型と言う変数型が存在しません. 文字列は, 全て char 型の配列であり[3], 文字列の最後には必ず数としての 0, C 言語の文字記号で '\0' が付きます (null terminated string と言う). また, 文字列の初期化ではダブルクォーテーション " で囲む事により, 最後の '\0' が自動的付加されます.

上のプログラムで,

```
char *p="This is a test.";
```

となっている所は, 次のように読みます.

- p は, char 型を指すポインタ型変数である.
- p は, 文字列型配列 "This is a test." の最初の文字を指すように, 初期化する.

また,

```
char p[]="This is a test."
```

は, 次のように読みます.

- p[] は, 続く文字列が納めるだけの大きさを持った配列である.
- p[] は, 続く文字列を初期値として持つ.

いずれの場合にも, 文字配列の大きさは, 文字数 + 1 でこの場合だと 16 です. +1 の部分は, 終端文字 '\0' の部分です.

また, **文字列が必ず数の 0 で終る事を利用しる事により, 次の while 文が問題無く実行されているのです**. すなわち, 0 以外はループで回し, 終端の 0 が来た時点でループは終了します.

[3]おそらくこのことが, 関数は配列を引数にできないとか, 関数は配列を返せないとかの規格決定に影響した物だと思いますが, 真実は知りません

8.9 練習問題

問題 8.1. 正の整数の素因数分解を求めるプログラムを作成せよ.
例えば, $6 = 2 * 3$ と表示されるようにする.

問題 8.2. *3* 次元 $\mathbf{R}^3$ の 2 つのベクトルを与えられたとき, それに直交するベクトルを求めるプログラムを作成せよ. なお, 平行なベクトルが与えられた時には, そのことを告げるようにする事.

Chapter 9

第9回 配列II: 並び替え（バブルソート）

9.1 今までの講義に関する確認と補足

9.1.1 式の値

以前に代入文は，**式自体が値を持つ**ことを学びました．同様に，条件式も値を持ちます．while 文を例に考えてみましょう．

```
while (1) {
    printf("infinite loop!\n");
          }
```

この場合，while の中の中括弧で囲まれたブロックは，永遠に繰り返されます．一般に

```
while (式) {
              ....
           }
```

では，式の値を計算し，それが 0 でない場合，ブロックの中を実行します．条件式はそれが真のとき，0 でない値を取り（1 を取る場合が多い)，偽のときは，0 を取ります．従って，

```
a = 0;
while (a < 3) {
               a = a + 1;
              }
```

というプログラムでは，a が 0, 1, 2 のときの3回繰り返したあと，`a < 3` **の値は，0 になり**，次は，while の中の文は，実行せずに，その下へとプログラムは進みます．これを利用すると，次のようなプログラムを組むことも出来ます．

```
a = -3;
while (a) {
           a = a + 1;
          }
```

文字列の最後には必ず数としての 0, C 言語の文字記号で ’`\0`’ が付きます (null terminated string と言う)．また, 文字列の初期化ではダブルクォーテーション ” で囲む事により, 最後の ’`\0`’ が自動的付加されます．これを利用すると, 次のような場合 while 文が問題無く実行されます．すなわち, `\0`(null) 以外はループで回し, 終端の `\0` が来た時点でループは終了します．

```
#include <stdio.h>

main()
{
        char *p="This is a test.";

        while(*p){
                putchar(*p);
                p++;
                 }
}
```

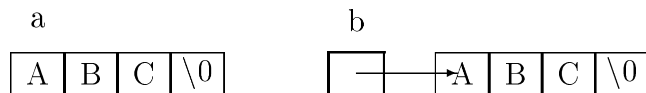
9.1.2 配列とポインタ

配列を宣言すると，配列名はその配列の先頭番地を示すポインタとして扱われました．

例えば,

```
char a[] = "ABC";
char *b = "ABC";
```

の違いは次の図から理解できると思います:



a, b とも配列の先頭を指すポインタでありますが, b はポインタ型変数のため `b +1` とすると意味がありますが (この場合矢印が 1 つ進み B のアドレスを指す), a では `a+1` は意味がありません. 但し, `*(a+1)` は意味があり, B を指します.

関数に配列名を渡すと，その先頭番地がコピーされるので，関数の中で，配列の要素の値を操作することが出来ます.

次のプログラムは，何を行っているか考えてみましょう.

```
void swap_aij(int *pa, int i, int j);
main ()
{
    int  a[10];
    ....
    swap_aij(a, i, j); /* a は配列の先頭番地 */
    ....
}

void swap_aij(int *pa, int ii, int jj)
{
    int    tmp;

    tmp = *(pa + ii);        /* tmp = pa[i] と書いても同じ */
    *(pa + ii) = *(pa + jj);  /* pa[i] = pa[j] と書いても同
じ */
    *(pa + jj) = tmp;        /* pa[j] = tmp と書いても同じ */

    return;
}
```

補足

先週の「内積を用いた角度の計算」のプログラムをポインタで書き直してみますと以下の通りになります.

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

double inn_prod(double *, double *);

main()
{
        char nyuryoku[80];
        double a[3], b[3], *pa, *pb;
        double angle;

        pa = a;        /* ポインタデータの代入*/
        pb = b;       /* ポインタデータの代入*/

        printf("最初のベクトルの x, y, z 成分を空白で区切って
入力して下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &a[0], &a[1], &a[2]);

         printf("次のベクトルの x, y, z 成分を空白で区切って入
力して下さい>>");
        gets(nyuryoku);
        sscanf(nyuryoku, "%lf %lf %lf", &b[0], &b[1], &b[2]);

       angle = acos(inn_prod(pa, pb)/sqrt(inn_prod(pa, pa)
                                *inn_prod(pb, pb)));

        printf("2 つのベクトルのなす角はおよそ %f ラジアンです
\n", angle);
             }

double inn_prod(double *pa, double *pb)
               {
        return (*pa)*(*pb) + (*(pa+1))*(*(pb+1))+
                                   (*(pa+2))*(*(pb+2));
      }
```

注意 上のプログラム内で, 記述が長いため段落を変えたものがあります.

9.2 最大値捜し

与えられた n 個の整数から最大数を求めるプログラムを配列 $d[n]$ をつかって組んでみると, 主要部分は以下の通りになります:

```
for(i=0;i < n; i++)
{  if(d[i] >= d[i+1])
      tmp = d[i];
      d[i]= d[i+1];
      d[i+1] = tmp;
}
```

tmp は, for 文の前で定義されている, 整数型の変数.

この方法を用いると, 最大数は最後の d[n-1] であることがわかります.

もちろん, このほかの方法もありますが, あとで利用するため上の例をあげました.

基本的アイデアは, 2 つの整数の順序を取り替えるアルゴリズム

```
tmp = a;
a = b;
b = tmp;
```

で, 予備の変数 tmp を用いて取り替えをしなければなりません.(何故か)

9.3 並べ替え (バブルソート)

与えられた整数を小さい順に並び替えるバブルソートについて今回学びます.

具体的にどういうものかと言いますと, 今,

```
37 23 11 19 5 7
```

という 6 個の整数が与えられたとき, これを小さい順に並び替えて,

```
5 7 11 19 23 37
```

にするということです.

実際, どのようなアルゴリズムをするでしょうか?

(1) まず一番目の37を2番目の23と比較する.　37 > 23より, これをとりかえる.

(2) 次に2番目の整数 (今の場合は37) と3番目の整数を比較し, 3番目の整数が小さかったら取り替える.

(3) 以下これを繰り返すと,

```
23 11 19 5 7 37
```

となります.

(4) 次に, 上の操作 (1) - (3) を前から5つの整数に行う. そうすると,

```
11 19 5 7 23 37
```

となります.

(5) 以下, 順に比較する整数を減らして, (1) - (3) の操作を行う.

つまり, 比較する整数の集合を配列 $d[n]$ であらわしたとき, 上のアルゴリズムは,

```
for(i = n-1; i>= 0; i--)
{d[0] と d[2], ....,d[i] を比較し, 最大数を d[i] とする}
```

となります.
それでは実際に組んでみましょう.

9.4 演習

以下の問題の空欄 (1) ∼ (6) を穴埋めせよ．但し, プログラム名は creportno.9-1 とする．

与えられた n(500 以下) 個の整数を小さい順に並び替える次のプログラムを完成せよ．

```
#include<stdio.h>
#define MAX 500

main()
{
        int d[MAX];
        int n, tmp, i, j;

        printf("並べ替える整数の個数を入力せよ\n");
        scanf("%d", (1));

        printf("%d 個のデータを入力してください\n",n);

        for((2))                   /* データ入力 */
        { (3)}

        for(i=n-1; i>=0; i--)      /* 並び替え */
         {
            for((4))
            { (5)}
          }

        for(i=0;i < n; i++)       /* データ表示 */
         { (6)}
}
```

9.5 練習問題

次のプログラムを作成せよ.

問題 9.1. 先の *creportno.9-1* を *int d[MAX]* を大きい順に並びかえる *void* 型関数 `void sort_data(void)` を定義して書き換えよ.

問題 9.2. 文字列を入力し, その文字列を逆アルファベット順に並べ替えた文字列を出力するプログラムを作成せよ.　例えば, *chisato* を並べ替えると *tsoihca* となる.

Chapter 10

第10回
方程式を解く（二分法，ニュートン法）

10.1　二分法

方程式 $f(x)=0$ の実数解は，関数 $y=f(x)$ のグラフと，x 軸との共有点の x 座標にほかならない．

関数のグラフを調べて，方程式の実数解の近似値，すなわち近似解を求めることを考えよう．

一般に，関数 $y=f(x)$ のグラフが，ある区間でとぎれることなく，つながっているとき，関数 $f(x)$ は，その区間で連続であるという．区間 $[a,b]$ で連続な関数 $f(x)$ について，次の定理が成り立つ．

> **定理**　$f(a)$ と $f(b)$ の値の符号が異なるとき，すなわち，$f(a)f(b)<0$ ならば，$f(a)=0$ を満たす実数 α が，a と b の間に少なくとも１つ存在する．

この定理を利用して，方程式の近似解を求めることを考えよう．関数 $f(x)$ は，区間 $[a,b]$ において連続で，$f(a)f(b)<0$ を満たすとし，また，方程式 $f(x)=0$ はこの区間でただ１つの解をもつと仮定する．

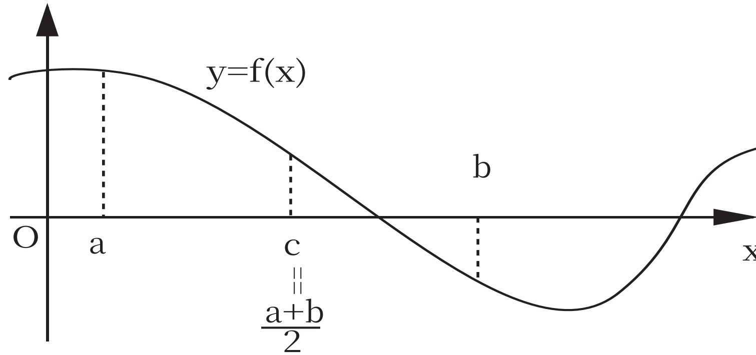
このとき，区間 $[a,b]$ の中点を $c=\frac{a+b}{2}$ とすると $f(c)$ の値は 0 となるか，または $f(a)$,$f(b)$ のいずれかと異符号である．

ここで，$f(c)=0$ なら，c は方程式の解である．

また，$f(a)f(c)<0$ ならば, 方程式は区間 $[a,c]$ に解をもち，$f(b)f(c)<0$ ならば，方程式は区間 $[c,b]$ に解をもつ．

このように，区間の中点を考えることにより，解の存在する範囲を初めの半分に縮小することができる．そして，この操作を繰り返すと，誤差の限界を必要なだけ小さくした近似解が得られる．

このようにして近似解を求める方法を**二分法**という．



一般に，方程式 $f(x)=0$ の近似解を，誤差の限界は ϵ として二分法で求める手順は次のようになる．

1. 変数 a,b に数値を入力して，$f(a)f(b)<0$ を満たすかどうか判断し，正しければ (2) にすすみ，正しくなければ計算不能と表示する．
2. 区間 $[a,b]$ の中点 $c=\frac{a+b}{2}$ を求める．
3. $f(c)=0$ ならば，c を解とする．$f(a)f(c)<0$ ならば，変数 b に c の値を代入し，$f(a)f(c)>0$ ならば，変数 a に c の値を代入する．
4. $|a-b|\leq\varepsilon$ かどうかを判断し，正しければ c の値を近似解とする．正しくなければ (2) に戻り，操作を続ける．

上の (3) において，$f(c)\neq 0$ のとき，変数 a,b の値をこのようにおき換えると，$f(a)f(b)<0$ となり，真の解 α は区間 $[a,b]$ にある．また，このとき，c の値は a か b に等しいから，$|c-\alpha|\leq|a-b|$ となり，c を近似解としたときの誤差の限界は $|a-b|$ である．したがって，(4) で誤差の限界が判定される．

方程式 $f(x)=0$ が区間 $[a,b]$ で 3 つの実数解 l,m,n $\quad(a<l<m<c<n<b)$ をもち, $f(a)>0,f(c)>0,f(b)<0$ とする．
c を a と b の中点として二分法を適用すると，第 2 段階以後区間 $[a,c]$ は無

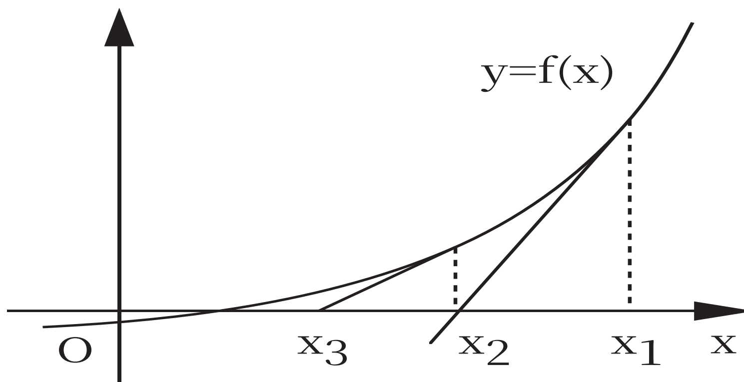
視されて，区間 $[c,b]$ にある解だけを近似することになる．このように，ある区間で方程式が２つ以上の解をもつ場合，二分法で近似解を求めると，１つの解以外は無視される結果になる．また，例えば，方程式 $(x-1)^2=0$ の重解１の近似値は，二分法では求められない．したがって，二分法を適用する場合，あらかじめ関数のグラフの概形を書くなどして，解について調べておくことが必要である．

10.2 ニュートン法

方程式の近似解を求める方法には，前項で学んだ二分法のほかに，曲線の接線を利用する方法がある．

任意の実数ａに対して，関数 $f(x)$ の $x=a$ における微分係数 $f'(a)$ が存在する場合，方程式 $f(x)=0$ の解 α の近似値を求めることを考えよう．

そのため，まず α に近い値 x_1 を１つとり，関数 $y=f(x)$ のグラフ上の点 $(x_1, f(x_1))$ における接線が，x 軸と交わる点の x 座標を x_2 とする．



このとき，接線は，点 $(x_1, f(x_1))$ を通り，傾きが $f'(x_1)$ の直線であるから，その方程式は次のようになる．

$$y = f'(x_1)(x-x_1) + f(x_1)$$

ここで，$y=0$ とおいて，x について解くと x_2 が求まる．

すなわち

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{10.1}$$

次に，点 $(x_2, f(x_2))$ における曲線の接線が，x 軸と交わる点の x 座標を x_3 とし，以下この操作を繰り返すと，数列

$$x_1, x_2, x_3, \cdots, x_n, \cdots \tag{10.2}$$

が得られる．そして，(1) と同様にして， $n = 2, 3, \cdots$ に対しても等式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{10.3}$$

が成り立ち，(2) は漸化式 (3) で定まる．

この数列 (2) は，最初の値 x_1 が α に十分近いとき，多くの場合急速に α に近づき，これを利用して方程式の近似解が効率よく求められる．

このようにして近似解を求める方法を**ニュートン法**といい，最初にとった値 x_1 を**初期値**という．

ニュートン法では，二分法と違って，誤差の限界の正確な判定が難しい．そのため，数列 (2) において，$|x_n - x_{n-1}| < \epsilon$ となったとき，x_n は誤差の限界が ϵ の近似解であると考える．

方程式 $f(x) = 0$ の近似解を，誤差の限界は ϵ として，ニュートン法で求める手順をまとめると，次のようになる．

1. 適当な初期値 a を入力する．

2. $b = a - \frac{f(a)}{f'(a)}$ を求める．

3. $|b - a| < \epsilon$ かどうかを判断し，正しければ，b を近似解とする．正しくなければ，変数 a に b の値を入力して，(2) にもどる．

10.3 演習

以下の問題の空欄 (1) ～ (10) を埋めて二分法のプログラムを完成せよ．但し，プログラム名は, creportno.10-1 とする．

問題 10.1. 初期値を *0, 2(*即ち，区間 *[0,2])*, 誤差を限界 ϵ を $\epsilon = 10^{-6} = 0.000001$ として, 二分法で $x^2 = 2$ の解を出力する下記のプログラムを完成せよ．また, そのとき得られた解を書け．*(Hint:1* ページ目の二分法の手順を参照せよ*)*

```
#include <stdio.h>

double f(double);

main()
{

  double a, b, c, e;
  a = (1);
  b = (2);
  e= 0.000001;                       /* 誤差 */

while((3))
{
  if((4))
     c = (5);
   else
    printf("計算不能");
  if(f(c)== 0)
   printf("求める解は %f です",c);
  if((6))
    b = (7);
      else if((8))
   a = (9);
}
    printf("近似解は %f です", c);
}

double f(double x)
{
   (10);
 }
```

10.4 練習問題

問題 10.2. 初期値 k を入力し，ニュートン法で $x^3 = k$ の解を出力するプログラムを作成せよ．ただし誤差の限界 ϵ は $\epsilon = 10^{-6} = 0.000001$ とする．