

計算機通論・情報工学通論：C言語入門

九州工業大学・情報工学部・生命情報工学科
松山明彦

2017年4月

はじめに：

生命情報工学科の編入生向けの情報工学通論と計算機通論のテキストです。内容は、C言語入門です。Java言語も少し紹介します。1回の授業で1章を進みます。講義ではいろいろ図も書いていく予定なので、テキストに無い説明などは各自で書き入れてください。各自でノートを作るのも良いでしょう。テキストには必要最小限のことしか書いてありません。他のC言語の参考書を買うことをお勧めします。

半期で通常の一年分は進みます。しっかり復習をしましょう。

Moodleに、演習問題の回答例などを載せていきます。

では、がんばってください。

2017年4月

生命情報工学科 松山明彦

目次

第 1 章	C 言語の第一歩	9
1.1	C 言語の基本	9
1.2	プログラミングの過程	9
1.3	プログラミングの基本	10
1.3.1	コメント文	10
1.3.2	プログラムの基本構造	10
1.3.3	挨拶プログラム	10
1.3.4	和を求めるプログラム	11
1.3.5	printf() 関数	11
1.3.6	文字列入力をするプログラム	12
1.3.7	たし算のプログラム	12
1.3.8	scanf() 関数	13
1.4	条件による分岐	13
1.4.1	2つの整数の差の絶対値を求めるプログラム	13
1.4.2	閏年の判定プログラム	14
1.4.3	switch-case 文	15
1.4.4	四則演算プログラム (switch-case)	16
1.4.5	プリプロセッサ命令	17
1.5	演習問題	18
第 2 章	繰り返しの処理と配列	21
2.1	while 文	21
2.1.1	1 から 5 までの 2 乗を求める	21
2.2	for 文	22
2.2.1	1 から 5 までの 3 乗を求める	22
2.2.2	for の入れ子構造	22
2.3	1 次元配列	23
2.3.1	入力の逆順に数値を表示する	23
2.3.2	ソート (整列)	24
2.4	2 次元配列	25
2.4.1	2 次元配列を用いた簡単なプログラム	25
2.4.2	素数を求める	27
2.4.3	多次元配列	28

2.5	演習問題	28
第3章	実数型変数と標準関数	29
3.1	実数型データ	29
3.1.1	float 型	29
3.1.2	double 型	29
3.1.3	実数型と整数型の混在	29
3.1.4	台形の面積	30
3.1.5	平方根を求める	30
3.2	標準関数	32
3.2.1	c 言語での関数の呼び出し	33
3.2.2	標準関数による平方根を求める	33
3.2.3	2 次方程式の根	33
3.2.4	sin と cos の表を作る	34
3.3	C 言語のいろいろな演算子	35
3.3.1	インクリメント演算子	35
3.3.2	入力の逆順に数値を表示する	36
3.3.3	複合代入演算子	36
3.3.4	printf() 関数の書式指定	36
3.3.5	エスケープ文字	37
3.4	演習問題	37
第4章	文字データの取り扱い	39
4.1	文字型データ	39
4.1.1	文字型変数の宣言	39
4.2	文字型と整数型	39
4.2.1	文字コード	40
4.2.2	getchar() と putchar() 関数	40
4.2.3	判定プログラム	41
4.2.4	文字型データの算術演算	41
4.3	文字列の扱い	42
4.3.1	文字型配列の宣言	42
4.3.2	文字列定数	42
4.3.3	文字列の入出力	43
4.3.4	文字の長さを答えるプログラム	43
4.3.5	文字列のコピー	44
4.3.6	文字列の接続	45
4.4	2次元の文字型配列	46
4.4.1	2次元文字型配列の宣言	46
4.5	演習問題	47

	5
第 5 章 ポインタの取り扱い	49
5.1 ポインタ (番地)	49
5.1.1 ポインタ変数の宣言	49
5.1.2 ポインタ演算子	49
5.1.3 ポインタを使った四則演算	51
5.1.4 配列とポインタ	51
5.2 ポインタによる文字列の取り扱い	53
5.2.1 文字列のコピー	53
5.2.2 文字列の分割	54
5.2.3 配列とポインタ変数の違い	55
5.2.4 見掛け上のコピー	55
5.2.5 2つの文字配列を接続する	56
5.3 演習問題	57
第 6 章 ファイルの入出力	59
6.1 ファイルの基本操作	59
6.1.1 ファイル・ポインタ	59
6.1.2 ファイルのオープン	59
6.1.3 ファイルのクローズ	59
6.1.4 ファイルの読み出し	60
6.1.5 文字数を数える	60
6.1.6 ファイルの書き込み	61
6.1.7 ファイルをコピー	61
6.2 データの書式付き入出力	63
6.2.1 ファイルに書き込むプログラム	63
6.2.2 ファイルを読み込んで表示する。	64
6.3 演習問題	65
第 7 章 関数の作り方	67
7.1 関数	67
7.1.1 引数を持たない関数	67
7.1.2 引数を持つ関数	68
7.1.3 引数の受け渡しの例	69
7.1.4 引数と戻り値を持つ関数	70
7.1.5 関数のプロトタイプ宣言	70
7.2 変数の特性：スコープ	71
7.2.1 int 型と double 型の関数 (スコープ)	72
7.2.2 変数の記憶クラス	73
7.3 文字列を渡す	74
7.4 演習問題	75

第 8 章	関数とポインタ	77
8.1	関数の引数とポインタ	77
8.1.1	ポインタ渡し	77
8.1.2	複数のデータを返す関数	78
8.2	関数の引数と配列	79
8.2.1	配列の受け渡し	79
8.2.2	配列とポインタ	80
8.3	main() 関数の引数	81
8.3.1	main() 関数の引数渡し	82
8.4	関数を指すポインタ	83
8.5	関数へのポインタの配列	84
8.6	演習問題	85
第 9 章	構造体と列挙型	87
9.1	構造体	87
9.1.1	複素数を扱うプログラム	87
9.1.2	新しいデータ型の定義	88
9.1.3	構造体を使ったプログラム	88
9.1.4	構造体と関数	89
9.1.5	構造体とポインタ	91
9.1.6	アロー演算子: \rightarrow	91
9.2	列挙型とデータの定義	93
9.2.1	データ型の定義	95
9.3	自己参照構造体	95
9.3.1	自己参照構造体の宣言	96
9.4	演習問題	99
第 10 章	Java 言語の基本	101
10.1	Java プログラムの実行	101
10.1.1	Hello プログラム	101
10.2	Java プログラムの形式	102
10.3	変数	103
10.4	クラス・メソッド	104
10.5	インスタンス	105
10.5.1	インスタンスの生成	105
10.6	オブジェクトの引き渡し	106
10.7	継承	107
10.8	演習問題	109
第 11 章	Java の制御文・配列・関数・その他	111

第 12 章 アプレット・グラフィックス	113
12.1 画面の基本	114
12.2 正方形・長方形	115
12.3 円・楕円・直線	116
12.4 多角形と折れ線グラフ	117
12.5 関数を描く	119
12.6 アニメーション	120
12.7 演習問題	122
第 13 章 高分子のランダムウォーク	123
13.1 正方形の壁で囲まれた 2 次元領域をランダムに動く高分子	123
13.2 演習問題	127

第1章 C言語の第一歩

1.1 C言語の基本

現在，広く使われているプログラミング言語として，FORTRAN, COBOL, LISP, BASIC, C, C++, Java などがあります。その中でもC言語はUNIXの基本言語であり，豊富なライブラリーを利用できるなど，いろいろな利点があります。最近ではオブジェクト指向の機能を追加したC++や，JAVAなどが主流になりつつありますが，C言語はそれらの基礎としても重要です。基本的には言語は何でも構わない，一つの言語をしっかりと学べばやりたいことはある程度出来るとおもいます。プログラム言語は英語の学習とよくにているところがあります。実際やってみることでどんどん勉強していくこと。これが上達への近道です。

1.2 プログラミングの過程

プログラムを

- (1) 編集 … vi や emacs などの編集ソフトでプログラムを記述する。
- (2) コンパイル … c 言語などの高級言語で書かれたソース・プログラムはそのままではコンピュータに理解してもらえません。ソース・プログラムはコンパイラと呼ばれる翻訳プログラムによって機械語に翻訳されます。機械語に翻訳されたプログラムを、オブジェクト・プログラムという。
- (3) リンク … c 言語ではよく用いる機能や関数は標準関数として用意されています。標準関数を取り出し、オブジェクト・プログラムに付加する作業をリンクと呼びます。
- (4) 実行 … プログラムの実行

Unix での操作例

```
% cc -o test test.c …… コンパイルとリンク
% test …… プログラムの実行
あるいは、
% cc test.c …… コンパイルとリンク
% a.out または、 ./a.out .. プログラムの実行
```

1.3 プログラミングの基本

1.3.1 コメント文

```
/* コメントの開始      */ コメントの終了
/*この中の文章はコンパイルされないで無視される。メモがわりに使う*/
   コメント文を書くことで、後でプログラムを見たときに理解しやすい。
```

1.3.2 プログラムの基本構造

main() 関数の定義、プログラムは exit(0) , 又は, return (0) で終了する。

```
/******
program.c 2003.12.10 .. プログラムに関する情報などを書く (プログラム・ヘッダー)
*****/
#include <stdio.h>
int main(void)
{

/* ここに実行文などを書く。*/

exit(0);
}
```

1.3.3 挨拶プログラム

```
0: /**101.c***/
1: #include <stdio.h>
2: int main(void)
3: {
4:     printf("Hello\n");
5:
6:     return(0);
7: }
8: /***/
```

- 1: ヘッダーファイルである stdio.h を読み込むことを示す。printf() 関数などを使うために必要。
- 2: main() 関数の定義。ここで、int は main() 関数の返す値が整数値であることを指定している。また void は main() 関数に渡されるデータがないことを示す。
- 3: main() 関数の始まりを示す。
- 4: printf() 関数で文字を出力する。

6: main() 関数にゼロを返して、終了する。正常終了の時はゼロ、異常終了の時は 1 を返す。1 の場合はエラーメッセージが出る。

7: main() 関数の終了。

1.3.4 和を求めるプログラム

```
0: /**** 102.c *****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int answer;
5:     answer=10+20;
6:     printf("ans=%d\n", answer);
7:
8:     return(0);
9: }
10: /*****/
```

4: 整数型変数 answer の宣言。

5: 10+20 を変数 answer に入れる。(数字を answer という箱に入れるイメージを持つとよい。必ず箱は左に、入れる数字は右に書く。)

6: 結果を出力する。

1.3.5 printf() 関数



```
書式指定文字列
printf("format", arg1, arg2, ..., )
```

図 1.1: printf() 関数の書式

printf() 関数は文字を出力する関数である。” ”内の文字はそのまま出力される。
\n (または、¥ n) は改行の意味。

変換方法を指定する例

1. printf("ans= %d \n", nn); 変数 nn の値を%d に入れ、” ”内をそのまま出力。
2. printf("%d \n", nn); 3 0 を出力してから、改行する。
3. printf("%d ", nn); 3 0 を出力するだけ、改行しない。

%d などを変換仕様という。%d は10進数、%x は16進数、%f は浮動小数点、%c は文字、%s は文字列を出力する時に使い分ける。

1.3.6 文字列入力をするプログラム

```
1: /****103.c*****/
2: #include <stdio.h>
3: int main(void)
4: {
5:     char ss[80];
6:
7:     printf("Input ss>>");
8:     scanf("%s", ss);
9:
10:    printf("Moji_Retsu=%s\n",ss);
11:    return(0);
12: }
13: /*****/
```

5: 文字型変数 ss[] の文字宣言。[80] は80文字格納出来る。

7: ” ”内の文字を出力する。

8: scanf(”%s”, ss) 関数はキーボードから文字列を入力する関数。文字列を入力する時は、%s を使う。

10: 文字列を出力

1.3.7 たし算のプログラム

```
0: /****104.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int first_num, second_num;
5:     int answer;
6:
7:     printf("Input Number 1 >>");
8:     scanf("%d", &first_num);
9:     printf("Input Number 2 >>");
10:    scanf("%d", &second_num);
11:
```

```

12:     answer=first_num+second_num;
13:     printf("%d+%d=%d \n",first_num,second_num,answer);
14:
15:     return(0);
16: }
17: /*****
```

8, 10: scanf() 関数はキーボードからデータを読み込む関数。整数を入力する時は、%d を使う。

1.3.8 scanf() 関数

scanf() 関数はキーボードからデータを読み込む関数である。変数名の前に & (アンパサンド) を付けること。ただし、配列を読み込むときは、この&はいらないことに注意 (1.3.6 を参照)。

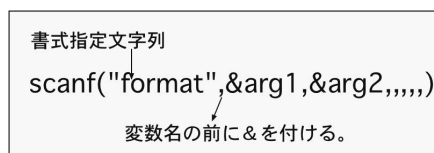


図 1.2: scanf() 関数の書式

1.4 条件による分岐

1.4.1 2つの整数の差の絶対値を求めるプログラム

```

0:/**105.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int first_num, second_num;
5:     int answer;
6:
7:     printf("Input Number 1 >>");
8:     scanf("%d", &first_num);
9:     printf("Input Number 2 >>");
10:    scanf("%d", &second_num);
11:
```

```

12:     if(first_num>second_num){
13:         answer=first_num-second_num;
14:     }
15:     else{
16:         answer=second_num-first_num;
17:     }
19:
20:     printf("|%d-%d|=%d \n",first_num,second_num,answer);
21:
22:     return(0);
23: }
24: /*****/

```

```

if(条件 1) {
    文 1 ;
}
else if (条件 2)
    /* 必要な回数だけ繰り返すことができる。*/
    文 2 ;文 3 ;
}
else (条件 3) {
    文 4 ;
}

```

図 1.3: if()-else 文の構文; 条件が真の時文 1 , 2 を実行、条件が偽の時文 3 を実行

1.4.2 閏年の判定プログラム

閏年は 4 で割り切れるが、100 で割り切れない。ただし 400 で割り切れる場合も閏年である。「4 で割り切れる」「100 で割り切れない」「400 で割り切れる」という 3 つの部分にわけると簡単な論理演算子で書けます。

```

0: /*****106.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int year;
5:
6:     printf("Input Year 1>>");
8:     scanf("%d", &year);
9:

```

```

10:     if( (year % 4 ==0 && year % 100 !=0) || year % 400 ==0){
11:         printf("%d : Uruu-Doshi ! \n",year);
12:     }
13:     else{
14:         printf("%d : Futuu-no toshi ! \n",year);
15:     }
16:
17:     return(0);
18: }
19: /*****/

```

10: 西暦を4で割った余りがゼロは「year % 4 ==0」と書けます。「year % 100 !=0」は100で割って余りがゼロでないを示します。

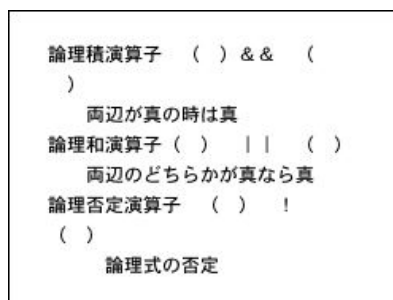


図 1.4: 論理演算子

1.4.3 switch-case 文

1 から 3 の整数を入力すると、それに対応する英語を出力するプログラムを以下に示す。

```

0: /****107.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int select; /* 入力される整数 */
5:
6:     printf("Input Number 1 or 2 or 3 >>");
8:     scanf("%d", &select);
9:
10:    switch(select){
11:        case 1: printf(" One \n ");

```

```

12;                break;
13:                case 2: printf(" Two \n ");
14;                break;
15:                case 3: printf(" Three \n ");
16;                break;
17:                default:
17:                }
18:                return(0);
19:                }
20:                /*****/

```

12: break 文がないと 1 3 行目に進む。

14: break 文がないと 1 5 行目に進む。

switch の後ろのカッコ内に書かれた変数や整数の値が case に続く定数と一致すると、その case 以降に書かれた実行文が全て実行される。どの case に続く定数とも一致しない場合は case で書かれた文を全部飛ばして、下 (default) に行く。break 文は if 文や switch-case 文などの分岐や繰り返し制御から強制的に抜け出す働きをする。

```

switch (整数) {
    case 整数定数 1 : 実行文 1 ;

    break;
    case 整数定数 2 : 実行文 2 ;
                    break;
    case 整数定数 3 : 実行文 3 ;

    break;
    default: 実行文 z ;
} /* 強制脱出breakで
switch{}を抜ける */

```

図 1.5: switch-case 文と break

1.4.4 四則演算プログラム (switch-case)

1 から 4 の整数番号を入力して、2 つの整数の四則演算を行うプログラムです。ただし、1 は和、2 は差、3 は積、4 は商に対応しているとする。

```

0: /****108.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int first_num, second_num; /* 入力される 2 整数 */

```



```

5:      int operation;                /* 演算の種類 */
6:      int answer;                  /* 答え */
7:
8:      printf("Input Integer Number 1 >");
9:      scanf("%d", &first_num);
10:     printf("Input Integer Number 2 >");
11:     scanf("%d", &second_num);
12:     printf("operation (1) +, (2) -, (3) *, (4) / >>");
13:     scanf("%d", &operation);
14:
15:     switch(operation){
16:         case 1: answer=first_num+second_num;
17:             printf(" %d+%d=%d \n ", first_num, second_num, answer);
18:             break;
19:         case 2: answer=first_num-second_num;
20:             printf(" %d-%d=%d \n " , first_num, second_num, answer);
21:             break;
22:         case 3: answer=first_num*second_num;
23:             printf(" %d * %d=%d \n " , first_num, second_num, answer);
24:             break;
25:         case 4: answer=first_num/second_num;
26:             printf(" %d / %d=%d \n " , first_num, second_num, answer);
27:             break;
28:         default: printf("Error: Operation (1-4) \n");
29:     }
30:     return(0);
31: }
32: /*****/

```

28: 1 - 4 以外の数字を選ぶと 28 行目に飛ぶ。

1.4.5 プリプロセッサ命令

前のプログラムの定数のようにプログラム中に現れる見ただけでは意味のわからない定数を、マジック・ナンバーと呼びます。case 1 case2 などの数字です。マジックナンバーはプログラムを難解にします。1 が和、2 が積に対応しているのは、プログラムの中身を読まないといけない。そこで、プリプロセッサ命令による定数定義を用いるとプログラムが理解しやすくなります。

```

/*****/
#include <stdio.h>

```

```
#include <stdio.h>
#define 定数名 定数
/*
  定数 1 を定数名 ADD に付ける場合、
  #define ADD 1
  と書く。
  */
```

図 1.6: #define 命令。セミコロン ; はいらぬ。定数名は大文字にするのが習慣

```
#define ADD 1 /* 足し算 */
#define SUB 2 /* 引き算 */
#define MULT 3 /* かけ算 */
#define DIV 4 /* 割り算 */

int main(void)
{
    略

    switch(operation){
        略

    }
    return(0);
}
/*****/
```

1.5 演習問題

- 1.3.7 を参考にして、printf() と scanf() 関数を用いて 2 つの整数の四則演算を行うプログラムを作りなさい。
- 1.4.1 を参考にして、if()— else 文を使って、2 つの整数を入力し、その大きい方を小さい方で割った商と余りを求めるプログラムを作りなさい。
- 試験の成績が 90 以上ならば A, 90 点未満 70 点以上ならば B, 70 点未満 60 点以上ならば C, 60 点以下は不可 (Out) として、試験の点数を入力すると、評価を出力するプログラムを作りなさい。if()— else if — else を用いて考えてください。
- 1.4.4 のプログラムでゼロで割るとどうなるか？またこれを避けるようなプログラムを書いてください。

5. 1.4.5 を参考にして#define 命令を用いて 1.4.4 の四則演算プログラムを書き換えて下さい。

6. #include <stdio.h>

などの、.h が付くファイルをヘッダーファイルという。C 言語で使う、main() や printf() などを使うためにプログラム中へリンクしている。では、この

<stdio.h>

は Unix のツリー構造のなかのどこにあるでしょうか？絶対パス名で教えてください。

第2章 繰り返しの処理と配列

2.1 while文

2.1.1 1から5までの2乗を求める

```
While(条件) {  
    文1 ;  
    文2 ;  
  
}
```

図 2.1: while() 文の構文

```
0:/**201.c*****/  
1: #include <stdio.h>  
2: int main(void)  
3: {  
4:     int number; /* 2乗する数 */  
5:     int answer; /* 答え */  
6:  
7:     number=1;  
8:     while(number<=5){ /* while文の始まり */  
9:         answer=number*number;  
10:        printf(" %d^2=%d \n", number, answer);  
11:        number=number+1; /* numberに1加える */  
12:    } /* while文の終わり */  
13:    exit(0);  
14: }  
15: /*****/
```

7: number に初期値 1 を与える。

8: while(条件) 文: 条件が真なら while 文に書かれた文 が繰り返し実行される。つまり、number

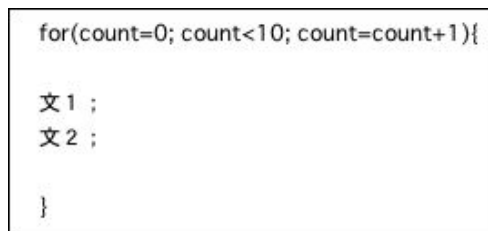
が5以下であるとき、while 文を繰り返す。

11: number に1を加える。

2.2 for 文

2.2.1 1から5までの3乗を求める

```
0: /**202.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int number; /* 3乗する数 */
5:     int answer; /* 答え */
6:
7:
8:     for(number=1; number<=5; number=number+1){ /* for 文の始まり */
9:         answer=number*number*number;
10:        printf(" %d^3=%d \n", number, answer);
11:
12:    } /* for 文の終わり */
13:    exit(0);
14: }
15: /***/
```



```
for(count=0; count<10; count=count+1){
文1;
文2;
}
```

図 2.2: for() 文の構文

2.2.2 for の入れ子構造

以下は、九九の表を出力するプログラムである。

for(初期値設定; 反復条件; 変更処理){ 文1; 文2; } が入れ子構造になっています。

```

0: /**203.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int i,j;
5:
6:     for(i=1; i<=9; i=i+1){
7:         for(j=1; j<=9; j=j+1){
8:             printf(" %d \n", i*j);
9:         }
10:    }
13:    exit(0);
14: }
15: /*****/

```

2.3 1次元配列

変数はデータを入れる箱の様なものであるが、配列はデータを入れる箱がたくさん並んだものと考えられる。配列の宣言は、例えば、

```
int sss[5];    /* 1次元配列 */
```

である。この場合、配列名が `sss` で、配列の要素が5個ある整数型の配列を宣言している。配列の要素は要素番号によって識別される。C言語では要素番号は0から始まる整数である。また、この例のように、配列の要素が1次元に並んだ配列を、1次元配列とよぶ。

1	3	10	2	7	データ
sss[0]	sss[1]	sss[2]	sss[3]	sss[4]	配列要素

図 2.3: 配列要素へのデータの代入: `sss[2]=10;`

2.3.1 入力の逆順に数値を表示する

0から4の5つの整数を入力すると、入力された順番と逆順で表示するプログラム。for文と配列を使ってある。

```

0: /**204.c***/
1: #include <stdio.h>
2: int main(void)
3: {
4:     int array[5];      /* 入力するデータ数は5個 */
5:     int count;
6:
7:     for(count=0; count<5; count=count+1){
8:         printf("Input number %d >", count);
9:         scanf("%d", &array[count]);
10:    }
11:
12:    for(count=4; count>=0; count=count-1){
13:        printf("No.%d : %d ", count, array[count]);
14:    }
15:    exit(0);
16: }
17: /*****/

```

4: 5つの要素を持つ int 型の配列 array[5] を宣言している。

9: 配列要素 array[] にデータを入れる。

配列 array[5] には array[0] から array[4] までの5つの要素がある。配列番号はゼロから始まることに注意。array[5] は存在しないので count=5 はエラーとなる可能性がある。

2.3.2 ソート（整列）

勝手な順序で入力されたデータを大きさの順に並べ替えることをソート（整列）という。隣り合った2つのデータを比較し、正しい順序ならそのままにしておき、逆順であれば入れ替える、という操作を先頭から最後まで繰り返せばよい。これを交換法という。

```

0: /**205.c***/
1: #include <stdio.h>
2: #define N 10          /* データの個数 N を定義している */
3:
4: int main(void) {
5:     int i,j,m,n;
6:     int a[N+1],b;
7:     printf("n=");    /* データの個数 n をキーボードから代入する */
8:     scanf("%d",&n);
9:

```



```

10: if(n>N) exit(1);          /* n>Nの時強制終了 */
11:
12: for(i=1; i<=n; i=i+1){
13:     printf("a[%d]=",i);
14:     scanf("%d",&a[i]);
15: }
16:
17: printf("*****\n");
18:
19: for(m=n-1; m>=1; m=m-1){
20:     for(j=1; j<=m; j=j+1){
21:         if(a[j]>a[j+1]){
22:             b=a[j];
23:             a[j]=a[j+1];
24:             a[j+1]=b;      /* 配列の入れ替えをする */
25:         }
26:     }
27: }
28: for(i=1; i<=n; i=i+1){
29:     printf("a[%d]=%d \n",i,a[i]);
30: }
31: }
32: /*****/

```

2.4 2次元配列

```
int array[3][5];    /* 2次元配列 3行5列*/
```

配列のデータ型 (int) と配列名 (array)、2次元両方向にそれぞれどれだけの要素を持つかを示す2つの要素数を示す。この例の場合、5列3行の配列を宣言する。

2.4.1 2次元配列を用いた簡単なプログラム

3行3列の数表を読み込み、縦の和と横の和を計算するプログラム。

```

0: /**206.c***/
1: #include <stdio.h>
2: int main(void){
3:     int table[3][3];          /* 3行3列の配列の宣言 */
4:     int y_sum[3],x_sum[3];   /* 縦と横の和 */

```

配列名: array[][]

配列要素	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
データ	8	7	6	1	2
	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
	1	3	4	1	100
	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
	10	10	12	2	5

図 2.4: 2次元配列要素へのデータの代入

```

5:     int x;                                /* 行のためのカウンター変数 */
6:     int y;                                /* 列のためのカウンター変数 */
7:
8:     for(x=0; x<3;x=x+1){                  /* 数表の入力 */
9:         for(y=0;y<3;y=y+1){
10:             printf("[%d][%d]=",x,y);
11:             scanf("%d",&table[x][y]);
12:         }
13
14:         y_sum[x]=0;                        /* 初期化：始めにゼロを入れておく */
15:         x_sum[x]=0;
16:     }
17:     /****和の計算*****/
18:     for(x=0;x<3;x=x+1){
19:         for(y=0;y<3;y=y+1){
20:             y_sum[x]=y_sum[x]+table[x][y];
21:             x_sum[x]=x_sum[x]+table[y][x];
22:         }
23:     }
24:     /***** 結果の出力 *****/
25:
26:         for(y=0;y<3;y=y+1){
27:             printf("%d\t %d \t %d \n",table[y][0], table[y][1],table[y][2]);
28:         }
29:         printf("*****\n");
30:         printf("%d \t %d \t %d \n", x_sum[0],x_sum[1],x_sum[2]);
31:         printf("%d \t %d \t %d \n", y_sum[0],y_sum[1],y_sum[2]);
32:
33:     exit(0);

```

```

34:
35:     }
36:     /***** */

```

printf() 文を使って、分かりやすいように表示するようにする。

2.4.2 素数を求める

1000までの素数をすべて求めるプログラムを示します。

考え方(エラトステネスのふるまい):(1)まず2以上の整数を並べておきます。2は素数です。2の倍数はすべて2で割り切れるから、×印をつけて除外します。(2)次に2をのぞく最小の素数は3です。3の倍数に×印をつけて除外します。(3)3より大きい整数で×印の着いていない最小の数は5です。5の倍数に×印をつけて除外します。考慮する最大の数までこれを繰り返せば、×印の付いていないのが素数になります。

```

0:/**207.c*****/
1:   #include <stdio.h>
2:   #define MAX 100           /* 素数チェックの最大数 */
3:   #define PRIME 0          /* 素数 */
4:   #define NOT_PRIME 1     /* 素数でない */
5:
6:   int main(void) {
7:       int number[MAX+1]; /* 素数をチェックする配列 */
8:       int i,j;           /* カウンター変数 */
9:       /***** 素数配列の初期化*****/
10:      for(i=1;i<=MAX; i=i+1){
11:          number[i]=PRIME;
12:      }
13:
14:      /****エラトステネスのふるまい****/
15:      number[1]=NOT_PRIME;
16:      for(i=2; i<=MAX; i=i+1){
17:          if(number[i]==PRIME){
18:              for(j=2*i ; j<=MAX ; j=j+i ){
19:                  number[j]=NOT_PRIME;
20:              }
21:          }
22:      }
23:
24:      /****素数の表示*****/
25:      for(i=1 ; i<=MAX ; i=i+1){

```

```

26:         if(number[i]==PRIME){
27:             printf("%d \n",i);
28:         }
29:     }
30:
31:     exit(0);
32: /*****/

```

3,4: 0 を素数に 1 を否素数に対応させるために PRIME と NOT_PRIME という定数を定義する。

11: 配列の要素番号を調べる数に対応させる。

18: $j=j+i$ であることに注意!

2.4.3 多次元配列

```
int array[5][3][4];    /* 配列宣言例 */
```

2次元以上の配列をあつかうことが出来る。(上の例は3次元の配列)

配列はコンピューター上のメモリに確保されます。たとえば、配列を

```
int array[1000][1000][1000]
```

と定義すると、int 型変数は4バイトであるので、 $4(\text{バイト}) \times 1000(\text{要素}) \times 1000(\text{要素}) \times 1000(\text{要素}) = 4 \times 10^9 \text{ バイト} = 4\text{G バイト}$ のメモリが必要になります。

2.5 演習問題

- 2.2.1 を参考にして、1 から 1 0 0 までの整数の和を計算するプログラムを書いてください。
- フィボナッチ数列は、 $a_0=0, a_1=1$ で始まり、 $n=1,2,3, \dots$ の順に

$$a_{n+1} = a_n + a_{n-1},$$

という規則で生成される数列である。任意の n を与えたときのフィボナッチ数列を計算するプログラムを作りなさい。配列 $a[n]$ を用いること。また、 n の値が最大いくつまで正確な値が出るか調べよ。なぜ正確な値が出なくなるか? その理由を考えよ。(ヒント: ビットとバイト)

- 3×3 の行列を2つ読み込んで、2つの行列の和と積を求めるプログラムを作りなさい。

第3章 実数型変数と標準関数

3.1 実数型データ

1,2などは整数型変数ですが,1.0や2.0とすると実数型変数になります。小数点をつけると実数型の変数になる。

3.1.1 float 型

```
float japan    /*float 型実数 japan の宣言 */
```

最大値 3.4×10^{38} , 最小値 -3.4×10^{38} の実数をあつかえる。7桁の精度を持つ。

3.1.2 double 型

```
double japan   /*double 型実数 japan の宣言 */
```

最大値 1.7×10^{308} , 最小値 -1.7×10^{308} の実数をあつかえる。15桁の精度を持つ。

printf() や scanf() 関数で用いる実数型のための書式文字:

```
%f    float 型  
%lf   double 型
```

3.1.3 実数型と整数型の混在

```
0:/**301.c***/  
1: #include <stdio.h>  
2: int main(void)  
3: {  
4:     int count;  
5:     double a,b;  
6:     count=7;  
7:     a=1.0;  
8:     b=a+count/2;  
9:     printf("%lf",b);  
10:
```

```

11:         exit(0);
12:     }
13: /*****/

```

8: 変数 count や 2 は整数型であるので, count/2 は3となる。従って b=4 となる。

3.1.4 台形の面積

```

0: /***302.c*****/
1: #include <stdio.h>
2: int main(void)
3: {
4:     float a,b,h,s;
5:
6:     printf("a= \n");    /*上辺の長さ*/
7:     scanf("%f",&a);
8:     printf("b= \n");    /* 下辺の長さ */
9:     scanf("%f",&b);
10:    printf("h= \n");    /* 高さ*/
11:    scanf("%f",&h);
12:    s=(a+b)*h/2.0;    /*台形の面積*/
13:    printf("%f",s);
14:
15:    exit(0);
16: }
17: /*****/

```

12: 実数型の計算式に使う定数には小数点を付ける。

3.1.5 平方根を求める

入力した実数の平方根を求めるプログラムを作る。

2分法: たとえば2の平方根を求める場合を考えよう。2の平方根は $f(x) = x^2 - 2$ の2次曲線が x 軸と交わる交点である。 $f(2) = 2$ が正, $f(0) = -2$ が負であることから平方根は0と2の間にあることがわかります。次に $x = 0$ と2の間の中点1を考える。 $f(1) = -1$ は負ですから, x 軸との交点は $x = 1$ と $x = 2$ の間にあります。さらに1と2の中点 $x = 1.5$ を考えて, 2次曲線の x 軸の交点が $x = 1$ と $x = 1.5$ の間にあるか, $x = 2$ と $x = 1.5$ の間にあるかを調べる。この手続きを繰り返していくと, 2次曲線と x 軸の交点を挟む2つの数はどんどん近づき, 平方根の近似値が得られます。

2分法で任意の数 a の平方根を求める手続きは上の方法と同じです。2次曲線 $f(x) = x^2 - a$ を 0 と a の間で探すことになります。

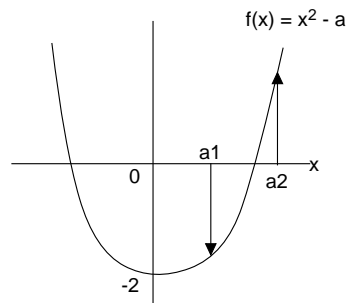


図 3.1: $f(x) = x^2 - 2$ のグラフ。2分法によって解に無限に近づく。

```

0:/**303.c***/
1:  #include <stdio.h>
2:  #define MAX 1000  /* 繰り返しの最大数 */
3:  #define  ACCURACY  1.0e-10  /* 近似の精度 */
4:  int  main(void){
5:
6:      double  a;      /*入力する数: この値の平方根を求める*/
7:      double  a1,a2;      /* 解をはさむ2点 ; a1<a2 とする*/
8:      double  tyu_ten;      /* 中点 */
9:      int     count;      /* カウンタ変数 */
10:
11:  /**** 数値入力と初期化*****/
12:  printf("Input data >>");
13:  scanf("%lf", &a);
14:  a1=0.0;      /* 0.0 と a の間で解を探す。*/
15:  a2=a;
16:  /***** 2分法の繰り返し *****/
17:  for(count=0; count<MAX; count=count+1){
18:      tyu_ten=(a1+a2)/2.0;
19:      if(tyu_ten*tyu_ten==a){
20:          break;      /* 解である場合 条件1 */
21:      }
22:      if(tyu_ten*tyu_ten-a>0.0){
23:          a2=tyu_ten;  }
24:      else{
25:          a1=tyu_ten;  }

```

```

26:
27:     if( ((a2-a1)/a1<ACCURACY) && ((a2-a1)/a1>-ACCURACY) ){
28:         break; }          /* 条件2 */
29:     }
30:
31:     printf("sqrt(%lf)=%lf \n", a, tyu_ten);
32:     exit(0);
33:     }
34: /*****/

```

2: 繰り返しの回数が最大 MAX 回で終了する。

3: 1.0×10^{-10} の精度で解を求めるように変数 ACCURACY を定義している。

23: 中点の値が $f(tyu_ten) > 0$ なら a2 を中点にする。

25: 中点の値が $f(tyu_ten) < 0$ なら a1 を中点にする。

27: a1 と a2 の差 (絶対値) が ACCURACY= 1.0×10^{-10} 以下になった時、解とする。

3.2 標準関数

c 言語では標準関数として多くの数学関数を用意している。標準関数のマニュアルなどを参考にしてください。

代表的な標準関数例 (以下の x や y は double 型の実数)

平方根	sqrt(x)
指数関数	exp(x)
自然対数	log(x)
常用対数	log10(x)
sin 関数	sin(x)
cos 関数	cos(x)
tan 関数	tan(x)
階乗 (x の y 乗)	pow(x, y)

これらの標準関数を使う場合は、プリプロセッサ命令に

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

を取り込まなければいけない。また、プログラム中で標準関数を使うには、コンパイラーで標準関数をまとめたライブラリーにリンクしなければいけない。これは $-lm$ というオプションをつけて cc コマンドを実行しなければいけない。

```
% cc sample.c -lm      または、% cc -o sample -lm sample.c
```


3.2.1 c 言語での関数の呼び出し

- a. 引数を持たない関数

```
function( ); /* 関数名を書きだけ*/
```

- b. 引数がある場合

```
function(x,y); /*カッコ内に引数を書く*/
```

- c. 戻り値を変数 (answer) に代入する場合

```
answer=function(x,y); /*function(x,y)の戻り値はプログラムのなかで利用できる*/
```

3.2.2 標準関数による平方根を求める

```
0:  /**304.c*****/
1:  #include <stdio.h>
2:  #include <math.h> /* math.h を呼び出す */
3:
4:  int main(void){
5:
6:      double a; /*入力する数*/
7:      double answer; /* 平方根*/
8:
9:      /*** 数値入力*****/
10:     printf("Input data >>");
11:     scanf("%lf", &a);
12:
13:     answer=sqrt(a);
14:     printf("sqrt(%lf)=%lf \n", a, answer);
15:     exit(0);
16: }
17:  /***/
```

3.2.3 2次方程式の根

2次方程式, $ax^2 + bx + c = 0$ (ただし $a \neq 0$) の根を求めるプログラムを作りましょう。

```
0:  /**305.c*****/
1:  #include <stdio.h>
2:  #include <math.h> /* math.h を呼び出す */
3:
4:  int main(void){
5:
6:      double a,b,c; /*係数/
```

```

7:     double   d,r,s,bunbo;
8:     double   x,x1,x2;
9:     printf(" a= "); scanf("%lf",&a);
10:    printf(" b= "); scanf("%lf",&b);
11:    printf(" c= "); scanf("%lf",&c);
12:        d=pow(b,2.0)-4.0*a*c;
13:        bunbo=2.0*a;
14:
15:    if(d>0){
16:        x1=(-b+sqrt(d))/bunbo;
17:        x2=(-b-sqrt(d))/bunbo;
18:        printf("x1=%lf \n ", x1);
19:        printf("x2=%lf \n ", x2);
20:    }
21:    else if(d==0){
22:        x=(-b)/bunbo;
23:        printf("x=%lf \n ", x);
24:    }
25:    else{
26:        r=(-b)/bunbo;
27:        s=sqrt(-d)/bunbo;
28:        printf("x1=%lf + %lf i \n ", r,s);
29:        printf("x2=%lf - %lf i \n ", r,s);
30:    }
31:        exit(0);
32:    }
33:    /*****/

```

3.2.4 sin と cos の表を作る

```

0:     /**306.c***/
1:     #include <stdio.h>
2:     #include <math.h>
3:     #define   MAX       20 /*繰り返しの回数*/
4:     #define   PAI       3.1415926535 /*円周率*/
5:
6:     int  main(void){
7:
8:         int  count ;           /*カウンタ変数*/
9:         double  theta;

```

```

10:     double   sin_ans, cos_ans;
11:
12:     /** 0 から 2PAI の間を 20 等分して各点での sin と cos を計算 **/
13:     for(count=0; count<=MAX; count=count+1){
14:         theta=2.0*PAI*(double)count/(double)MAX;
15:         sin_ans=sin(theta);
16:         cos_ans=cos(theta);
17:
18:         printf("%lf (rad): %lf, \t %lf \n", theta, sin_ans, cos_ans);
19:
20:     }
21:     exit(0);
22: }
23:
24: /*****/

```

14: 変数 count と MAX をキャスト演算子によって型変換している。すべてを実数型変数として宣言してもよい。その場合、小数点を忘れないように。

3.3 C 言語のいろいろな演算子

3.3.1 インクリメント演算子

インクリメント演算子は数値などの値を 1 だけ増やしたりする演算子です。

- a. インクリメント演算子: ++
count++; は count=count+1; と等価。
- b. インクリメント演算子: --
count--; は count=count-1; と等価。

前置きインクリメント演算子

```

count=2;
number=++count;
/* count の値を 1 増やして, number に代入する。結果は count=3, number=3 となる。*/

```

後置きインクリメント演算子

```

count=2;
number=count++;
/* count の値を number に代入する。その後, count の値を 1 増やす。結果は count=3, number=2 となる。*/

```

3.3.2 入力の逆順に数値を表示する

```

0:    /**307.c*****/
1:    #include <stdio.h>
2:
3:    int main(void){
4:        int array[5];
5:        int count;           /*カウンタ変数*/
6:
7:        for(count=0; count<5; count++){
8:            printf("Input No.%d >>", count);
9:            scanf("%d",&array[count]); /* 整数型変数 */
10:       }
11:
12:       for(count=4; count>=0; count--){
13:           printf("No. %d : %d \n", count, array[count]);
14:       }
15:       exit(0);
16:   }
17:   /***/

```

3.3.3 複合代入演算子

複合代入演算子	使用方法	等価な文
+=	count+=5;	count=count + 5;
-=	count-=5;	count=count - 5;
=	count=5;	count=count * 5;
/=	count/=5;	count=count / 5;
%=	count%=5;	count=count % 5

3.3.4 printf() 関数の書式指定

```

"%4d"    /* 10進整数4桁で右寄せで表示する。 */
"%-4d"   /* 10進整数4桁で左寄せで表示する。 */

```

実数型的小数表示

```

"%f" /* 実数型的小数表示；小数点以下6桁で表示 */
      (表示例) 1.234560, 12345600.000000
"%e" /* 浮動小数点表示；小数点以下6桁で表示 */
      (表示例) 1.234560e+00, 1.234560e+8

```

```

"%g" /* 有効桁数 6 桁で表示 */
      表示例) 1.23456, 1.23456e+8
"%10.3f" /* 10 は表示桁数で 3 は小数点以下桁数を示す */
          表示例) 1.234, 123456.000
"%10.3e" /* 10 桁で小数点以下 3 桁で表示 */
          表示例) 1.234e+00
"%10.3g" /* 10 桁で有効桁数 3 桁で表示 */
          表示例) 1.23 1.23e+10

```

3.3.5 エスケープ文字

エスケープ文字	名称	機能
\b	バックスペース	カーソルを 1 文字左へ動かす
\f	改頁	次の頁の先頭へ移動
\n	改行	次の行へ移動
\t	タブ	空白を挿入する

3.4 演習問題

(ちょっと多いですが、出来るところまで頑張りましょう。)

- 3.1.3 のプログラムで、答えが実数型の 4.5 となるようにプログラムを書き換えよ。
- 3.1.5 のプログラムで精度 (ACCURACY の値) をいろいろ変えて平方根を求めてみよう。また、次の節で示す標準関数を用いて求められる平方根の精度はどのくらいか調べよう。
- 3.2.3 のプログラムで、 $a = 0$ の場合も解を与えるようにプログラムを書き換えよ。
- 変数 `number` は、`double` 型の実数で `1.2345e6` が格納されている。次の `printf()` 関数を実行した場合どのような出力が得られるか。

(a) `printf("%10.3lf", number);`

(b) `printf("%10.3le", number);`

- 実数型の 3×3 の行列を 2 つ読み込んで、2 つの行列の和と積を求めるプログラムを作りなさい。9 個の行列要素には 1.1 から 1.9 までの実数を 0.1 ずつ増やして入れることにしましょう。
- (a) a_1 から a_n 個の n 個の実数型データの和

$$sum = \sum_{i=1}^n a_i,$$

と平均値 ($s = \text{sum}/n$)、さらに分散、

$$ss = \frac{1}{n} \sum_{i=1}^n (a_i - s)^2,$$

と、標準偏差 $sss = \sqrt{ss}$ を求めるプログラムをつくりましょう。

n 個のデータとして、 $a_i = i^{-2}, (i = 1, 2, \dots, n)$ を与えてください¹。

(b) n を増やしていくと和 (sum) の値はどうなるか？

(c) 時間のある人は、

$$\text{sum} = \sum_{i=1}^{\infty} \frac{1}{n^k},$$

($k=2, 3, 4, \dots, 10$) の場合の関数の和も求めて見ましょう。

7. 「じゃんけん」するプログラムを作って下さい。(考え方) グー、チョキ、パーを 1, 2, 3 に対応させる (switch() 文などを使う)。見られないように二人が交互にキーボードから 1, 2, 3 の内のどれかの数値を打ち込んで、その値を a, b とします。この数値を調べて、「a の勝ち」「b の勝ち」「引き分け」を出力するプログラムを作る。

¹自然数の逆数の冪の級数 = ツェータ関数と呼ばれている。

第4章 文字データの取り扱い

4.1 文字型データ

C言語では、整数型や実数型のデータとして数値が扱えるように、文字型のデータとして、文字を取り扱うことができる。文字型の宣言は char 型で、char 型で扱えるデータは1文字の英数字といくつかの記号です。

4.1.1 文字型変数の宣言

```
char moji;  
    moji='a';
```

文字変数 moji に a を格納する。シングル・クォート 'a' で文字を囲む。printf() や scanf() 関数で文字型データを入出力する場合は、%c を使う。

```
printf("%c", moji);      scanf("%c", &moji);
```

文字の入出力プログラム

```
0:    /**401.c*****/  
1:    #include <stdio.h>  
2:  
3:    int main(void){  
4:        char moji;  
5:  
6:        printf("Input Character >>");  
7:        scanf("%c", &moji);  
8:  
9:        printf("%c \n ", moji);  
10:   exit(0);  
11:   }  
12:   /*****/
```

4.2 文字型と整数型

コンピュータでは文字を直接扱うかわりに、文字に番号(文字コードという)が付けられていて、その番号によって文字を表します。代表的な文字コードはASCIIコードやEUCコードなどがあ

る。char 型変数は-128から128の整数をつかって文字に対応させています。文字コードには整数を使いますから、文字型のデータは整数です。アスキー・コード表などで確認してください。

4.2.1 文字コード

以下は入力された文字コードを出力するプログラムです。

```

0:      /**402.c*****/
1:      #include <stdio.h>
2:
3:      int  main(void){
4:          char moji;      /* 入力される文字*/
5:          int  c_code;    /* 文字コード */
6:
7:          printf("Input Character >>");
8:          scanf("%c", &moji);
9:
10:         c_code=(int)moji;
11:         printf("%c : %d \n ", moji, c_code);
12:         exit(0);
13:     }
14:     /***/

```

10: 文字型変数 moji の内容を int 型に変換して、int 型変数 c_code に代入する。

4.2.2 getchar() と putchar() 関数

scanf() や printf() 関数でも文字型変数の入出力はできますが、文字型データの入出力には getchar() と putchar() 関数があります。両方とも戻り値は int 型である。

文字データの入力	getchar(); /* 引数はない */
文字データの出力	putchar(char moji) ; /* 引数は char 型 */

文字の入出力プログラム

```

0:      /**403.c*****/
1:      #include <stdio.h>
2:
3:      int  main(void){
4:          char moji;      /* 文字型変数の宣言 */
5:
6:          printf("Input Character >>");

```



```
7:     moji=(char)getchar();
8:
9:     putchar(moji);
10:    putchar('\n');    /* 改行の出力 */
10:    exit(0);
11: }
12:  /*****/
```

7: getchar() 関数の戻り値をキャスト演算子で char 型に変換している。

4.2.3 判定プログラム

入力された英文字が大文字か小文字かを判定するプログラムを作りなさい。英文字以外が入力されたときは、エラーメッセージが表示されるようにしなさい。

(考え方): ASCII コードでは英文字は順番に番号付けされています。大文字'A'が65番で'B'が66番、順番にいて、'z'が90番、小文字は'a'の97番から始まって、'z'は122番で終わります。

```
0:  /****404.c*****/
1:  #include <stdio.h>
2:
3:  int main(void){
4:      char moji;    /* 文字型変数の宣言 */
5:
6:      printf("Input Character >>");
7:      moji=(char)getchar();
8:
9:      if(moji>='A' && moji<='Z'){
10:         printf("%c : Large \n", moji); }
11:     else if(moji>='a' && moji<='z'){
12:         printf("%c : Small \n", moji); }
13:     else{
14:         printf("%c : Not Alphabet \n", moji); }
15:
16:     exit(0);
17: }
18:  /*****/
```

4.2.4 文字型データの算術演算

英文字の大文字と小文字の変換を行うプログラムを作りましょう。

(考え方): ASCII コード表によれば、大文字と小文字のアルファベット順に並んでいて、大文字

と小文字の差は32です。英小文字のASCIIコードから32を引くと大文字になります。つまり英小文字の文字コードから'a-'A'を引くと英大文字のコードになります。

```

0:      /***405.c*****/
1:      #include <stdio.h>
2:
3:      int main(void){
4:          char moji;    /* 文字型変数の宣言 */
5:
6:          printf("Input Character >>");
7:          moji=(char)getchar();
8:
9:          if(moji>='A' && moji<='Z'){
10:             moji += ('a'-'A');    }
11:         else if(moji>='a' && moji<='z'){
12:             moji -= ('a'-'A');    }
13:
14:             putchar(moji);
15:             putchar('\n');    /* 改行*/
16:         exit(0);
17:     }
18:     /***/

```

4.3 文字列の扱い

1文字ではなく、単語のように文字が並んだデータを文字列と呼ぶ。文字列は、文字型の配列によって取り扱います。

4.3.1 文字型配列の宣言

```
char string[5];
```

char: 配列のデータ型、配列名 string, 要素数5の定義。

4.3.2 文字列定数

文字列のデータをプログラム中に定数として書き込むために、文字列定数を用います。文字列定数は文字の並びをダブル・クォート ""で囲んで表します。ただし、文字の最後に自動的に

'\0'

(ヌル記号) が代入されます。これは、文字の終わりを示します。

```
"cat"    =====> 'c' 'a' 't' '\0'
```

ヌル記号も配列に格納されるので、要素数は文字数より 1 以上多い値を定義しなければいけない。長い文字列定数を複数行にわけて書く場合は改行の前にバック・スラッシュ「¥」を置く。

4.3.3 文字列の入出力

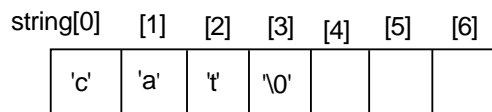


図 4.1: 配列 string[i] への文字列の格納。

printf() や scanf() 関数による文字列の入出力は、書式文字”% s”を使う。

```
char string[100];    /* 100個の配列要素をもつ文字列 string の宣言 */
scanf("%s", string);
printf("%s", string);
```

第二引数に「&」はいらない。

```
0:    /**406.c*****/
1:    #include <stdio.h>
2:
3:    int main(void){
4:        char string[100];    /* 文字列の宣言 */
5:
6:        printf("Input string >>");
7:        scanf("%s", string);
8:
9:        printf("%s \n", string);
10:
11:        exit(0);
12:    }
13:    /***/
```

4.3.4 文字の長さを答えるプログラム

```
0:    /**407.c*****/
```

```

1:     #include <stdio.h>
2:
3:     int  main(void){
4:         char  string[100];    /* 文字列の宣言 */
5:         int  s_length;       /* 文字列の長さ */
6:
7:         printf("Input string >>");
8:         scanf("%s", string);
8:
9:         for(s_length=0; string[s_length] != '\0'; s_length++){
10:            ;        }
11:         printf("%d \n",s_length);
12:         exit(0);
13:     }
14:     /*****/

```

9: s_length の値をゼロから 1 ずつ増やしながら string[s_length] の内容を繰り返し、ヌル記号であるとき for() 文から抜ける。

10: 何もない空文

4.3.5 文字列のコピー

文字配列 a[] に格納されている文字列を、文字配列 b[] にコピーするプログラムを考えよう。

```

0:     /****408.c*****/
1:     #include <stdio.h>
2:
3:     int  main(void){
4:         char  a[100];    /* 入力される文字列の宣言 */
5:         char  b[100];    /* コピー先の文字列の宣言 */
5:         int  count;      /* カウンタ変数 */
6:
7:         printf("Input string >>");
8:         scanf("%s", a);
8:
9:         for(count=0; a[count] != '\0'; count++){
10:            b[count]=a[count] ;
11:        }
12:            b[count]='\0';
13:
14:         printf("input : %s \n", a);

```

```
15:     printf("copy : %s \n", b);
16:     exit(0);
17: }
18:     /*****/
```

12: 文字列の終了を示すヌル記号を入れる。

4.3.6 文字列の接続

入力された2つの文字列をつなぐプログラムを作りましょう。

(考え方): 文字列 `b[]` に格納された文字列に文字列 `a[]` に格納された文字列を接続する場合を考えましょう。`b[]` に格納された文字列の終端を見つけ、その後ろに文字列のコピーと同様な処理を用いて、コピーしていく。

```
0:     /****409.c*****/
1:     #include <stdio.h>
2:
3:     int main(void){
4:         char a[100]; /* 入力される文字列の宣言 */
5:         char b[100]; /* コピー先の文字列の宣言 */
5:         int a_count, b_count; /* カウンタ変数 */
6:
7:         printf("Input string 1 >>");
8:         scanf("%s", a);
9:         printf("Input string 2 >>");
10:        scanf("%s", b);
11:        /*** 文字列の終端を探す***/
12:        for(b_count=0; b[b_count] != '\0'; b_count++){
10:            ;    }
11:
12:        /***文字列のコピー *****/
13:        for(a_count=0; a[a_count] != '\0'; a_count++){
13:            b[b_count+a_count]=a[a_count];
14:        }
15:        b[b_count+a_count]='\0';
16:        printf("string : %s \n", a);
17:        printf("copy : %s \n", b);
18:        exit(0);
19:    }
20:    /*****/
```

4.4 2次元の文字型配列

文字型の配列でも2次元の配列を用いることができる。

4.4.1 2次元文字型配列の宣言

```
char string[3][5];
```

char string[3][5]

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
'a'	'b'	'c'	'\0'	
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
'y'	'a'	'm'	'a'	'\0'
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
'k'	'a'	'w'	'a'	'\0'

図 4.2: 文字型の2次元配列

string[0] は文字列 'abc' を、string[1] は文字列 'yama' を、string[2] は文字列 'kawa' を表す。printf("%s ¥n", string[0]) は第0番目の文字列を表す。

入力された整数に対応する英単語を答えるプログラムを考えましょう。

(考え方): "one", "two", "three" をあらかじめ配列 name[][] に格納しておく。name[0], name[1], name[2] によって文字列を呼び出す。

```
0:      /**410.c*****/
1:      #include <stdio.h>
2:
3:      int main(void){
4:          char name[3][6]={"one", "two", "three"};      /* 英語の格納 */
5:          int number;          /* 入力される数 */
6:
7:          printf("Input number (1 or 2 or 3) >>");
8:          scanf("%d", &number);
9:
10:         printf("%s \n", name[number-1]);
11:         exit(0);
12:     }
13:     /***/
```

4: 英単語をあらかじめ、初期化付き宣言で2次元文字配列 name[][] の中に格納しておく。

10: number-1 は0行目から2行目までの文字列に対応している。

4.5 演習問題

1. 4.2.1 のプログラムを使って、いろいろな文字コードを調べてみよう。
2. 4.2.1 のプログラムをヒントに、a から z までと、A から Z までの文字と文字コードを出力するプログラムを書いてください。文字は文字型変数で扱えるので、`for(c_code='a'; c_code<='z'; c_code++)` を使える。
3. 4.4.1 のプログラムを参考にして、1 から 10 までの数に対応する英語を答えるプログラムを作りなさい。またこの範囲外の入力があった場合に、何らかのエラー処理を行うようにしなさい。
4. 文字コードのプログラムを参考に、0 から 127 までの数字を入力した時に、その数字（文字コード）に対する文字を出力するプログラムを作ってください。
5. 2次元文字型配列のプログラムを参考にして、日曜から土曜までに対応する英語を答えるプログラムを作りなさい。
6. 入力されたでたらめな文字列を、ABCD 順で出力するプログラムを作ってください。
7. 文字列 a と文字列 b を繋いだものを文字列 c に入れるプログラムを作ってください。

第5章 ポインタの取り扱い

5.1 ポインタ(番地)

C言語では、データは変数に格納されます。変数はメモリ上に置かれています。メモリ上で変数が置かれている番地を示しているのがポインタです。ポインタは変数のメモリ上の番地(アドレス)を示します(整数で表された番号)。ポインタを格納する変数をポインタ変数とよぶ。

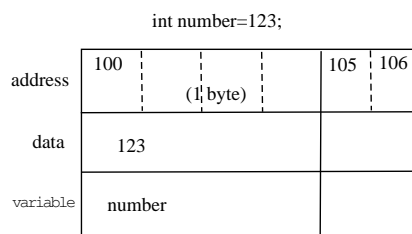


図 5.1: int 型の 4 バイトのデータが 100 番地に格納されている。変数 number の先頭番地が 100 である。番地はコンピューターシステムが自動的に決める。

5.1.1 ポインタ変数の宣言

100 番地(アドレス)という数字のかわりに、番地に自分の好きな名前(たとえば、int_ptr)を付ける事が出来る。

```
int *int_ptr; /* int 型のデータを示すポインタ変数 int_ptr の宣言を示す。*/
```

int *int_ptr; の意味は、アドレス(int_ptr)に入っている数値は int 型の整数であることを示す。

* (アスタリスク) は int_ptr がポインタ変数であることを示す。

5.1.2 ポインタ演算子

演算子名	演算子	使用例
アドレス演算子	& (アンパサンド)	int_ptr=&number;
ポインタ演算子	* (アスタリスク)	number=*int_ptr;

変数 `number` にアドレス演算子 (`&`) が適用されると、変数の置かれたアドレス (番地) が得られる。また、ポインタ変数 `int_ptr` にポインタ演算子 (`*`) が適用されると、ポインタ変数に格納されているデータが得られる。

実際にポインタがどんな値になっているか表示してみましょう。

```
0:      /***501.c*****/
1:      #include <stdio.h>
2:
3:      int  main(void){
4:          int    n=7;
5:          double h=20.0;
6:          double k=1.23;
7:
9:          printf(" &n= %d \n", &n);
10:         printf(" &n= %d \n", &h);
11:         printf(" &n= %d \n", &k);
12:         exit(0);
13:     }
14:     /******/
```

以下は、アドレス演算子とポインタ演算子の使用例を示します。`int_val` は宣言しただけで何も代入していないが、結果は2を表示するのがおもしろい。

```
0:      /***502.c*****/
1:      #include <stdio.h>
2:
3:      int  main(void){
4:          int    int_val;      /* int 型の変数 */
5:          int    *int_ptr;     /* int 型のポインタ変数 */
6:
7:          int_ptr=&int_val;
8:          *int_ptr=2;
9:
10:         printf(" int_val= %d \n", int_val);
11:         printf(" *int_ptr= %d \n", *int_ptr);
18:         exit(0);
19:     }
20:     /******/
```

7: 変数 `int_val` のアドレスをポインタ変数 `int_ptr` に代入している。

8: ポインタ演算を行ったポインタ変数 `int_ptr` に2を代入している。

`int_val` は宣言しただけで何も代入していないが、結果は2を表示する。

5.1.3 ポインタを使った四則演算

ポインタを使って $2 + 3 = 5$ の計算をやってみましょう。

```
0:    /***503.c*****/
1:    #include <stdio.h>
2:
3:    int main(void){
4:        int    *a, *b, *c;        /* int 型のポインタ変数 */
5:        int    ia, ib, ic;       /* int 型の変数 */
6:
7:        a=&ia; b=&ib; c=&ic;     /* 番地の格納 */
8:        *a=2; *b=3;
9:        *c=*a+*b;
10:
11:    printf(" *a=%d, *b=%d, *c=%d \n", *a, *b, *c);
12:    exit(0);
13: }
14:    /***/
```

5.1.4 配列とポインタ

ポインタは、メモリ上の変数を指し示すことが出来るだけでなく、配列の要素を指し示す事も出来る。c 言語では、「配列名は配列の先頭の要素のアドレスを表す」という約束になっています。つまり、

```
int array[5]
```

の array は配列の最初のアドレス (番地) を示しているのです。例えば int 型変数は 4 バイトなので、array が 1 0 0 番地、array[1] は 1 0 4 番地、array[2] は 1 0 8 番地の様になります。以下のプログラムで確かめて見ましょう。

```
0:    /***504.c*****/
1:    #include <stdio.h>
2:
3:    int main(void){
4:        int    a[5]; /* int 型の配列 */
5:
6:        printf(" a= %d \n", a);
7:        printf(" &a[0]= %d \n", &a[0]);
8:        printf(" &a[1]= %d \n", &a[1]);
9:        printf(" &a[2]= %d \n", &a[2]);
```

```

10:     printf(" &a[3]= %d \n", &a[3]);
11:     printf(" &a[4]= %d \n", &a[4]);
12:
13:     exit(0);
14: }
15:  /*****/

```

		int array[3] int *int_ptr					
address		100	104	108	112	116	120
		[0]	[1]	[2]	int_ptr		
data							

図 5.2: 配列とポインタ

ポインタと配列の関係を以下のプログラムで確認してみましょう。図 5.2 を参照。

```

0:     /***505.c*****/
1:     #include <stdio.h>
2:
3:     int main(void){
4:         int array[3]; /* int 型の配列 */
5:         int *int_ptr; /* int 型のポインタ変数 */
6:         int count; /* カウンター変数 */
7:
8:         int_ptr=array; /* ポインタ int_ptr は 1 0 0 番地を示すようになる*/
9:
10:        for(count=0; count<3; count++){
11:            *int_ptr=count; /* 1 0 0 番地から順番にデータ (count) が格納される*/
12:            int_ptr++;
13:        }
14:        for(count=0; count<3; count++){
15:            printf(" array[%d]=%d \n", count, array[count]);
16:        }
17:        exit(0);
18:    }
19:    /*****/

```

8: ポインタ変数 `int_ptr` には配列 `array[]` の先頭要素 `[0]` のアドレスが代入される。

11: ポインタ変数を用いて数値を代入する。`array[0]` に `count` が代入される。

12: ポインタ変数にインクリメント演算子を適用すると、次の配列要素 `[1]` を指し示す。

5.2 ポインタによる文字列の取り扱い

ポインタの得意な技として、文字列の処理がある。

5.2.1 文字列のコピー

4.3.5 節で取り扱った文字列のコピーをポインタを使って作りましょう。(配列の要素を直接操作するかわりに、ポインタ変数に格納されたポインタを用いて、配列の要素を操作する。)

```
0:      /***506.c*****/
1:      #include <stdio.h>
2:
3:      int  main(void){
4:          char  a_str[100];    /* 入力される文字列 */
5:          char  b_str[100];    /* コピー先の文字列*/
6:          char   *a_ptr, *b_ptr;    /* char 型のポインタ変数の定義 */
7:
8:          a_ptr=a_str; /* 文字配列の先頭アドレス */
9:          b_ptr=b_str;
10:         printf("Input String >> ");
11:         scanf("%s", a_ptr);
12:
13:         for(   ; *a_ptr != '\0' ; a_ptr++, b_ptr++){
14:             *b_ptr=*a_ptr; }
15:             *b_ptr='\0';
16:
17:             printf(" Input : %s \n", a_str);
18:             printf(" Copy : %s \n", b_str);
19:
20:             exit(0);
21:         }
22:         /***/
```

8, 9: 2つのポインタに2つの char 型配列の先頭アドレスを代入している。ポインタ変数 a_ptr と b_ptr はそれぞれ、a_str[0] と b_str[0] を示すことになる。

11: scanf() 関数の引数として、ポインタ変数が配列名と同じようにつかえる。

13: for 文の第1フィールドには何も書かれていない。第2フィールドの繰り返し条件は a_ptr のポインタが指し示すデータが「ヌル '\0」文字でないことである。

14: ポインタのコピー

5.2.2 文字列の分割

入力された氏名を名前と名字に分割するプログラムを作りましょう。ただし名前を代入するときには「Ichiro/Suzuki」の様に、名前と名字の間にスラッシュ「/」を入れるようにする。

(考え方): char 型配列 name[] と 2 つのポインタ変数 a_ptr, b_ptr を用意する。配列 name[] に名前を格納する。「/」の位置に「ヌル ¥ 0」を書き込めば、文字列は 2 つに分かれます。あとは、文字列の開始をポインタで指定すればよい。

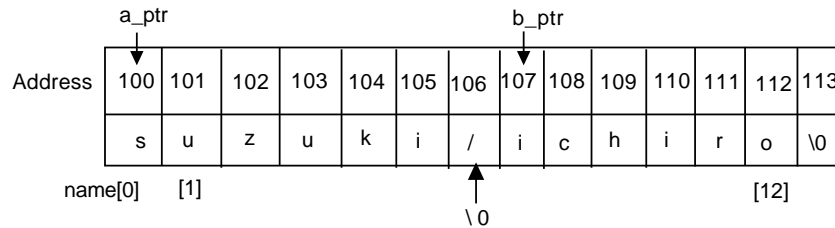


図 5.3: 文字列の分割

```

0:  /**507.c*****/
1:  #include <stdio.h>
2:
3:  int main(void){
4:      char name[100]; /* 氏名の配列 */
5:      char *a_ptr, *b_ptr; /* char 型のポインタ変数の定義 */
6:
7:
8:      printf("Input First name/Family name >> ");
9:      scanf("%s", name);
10:     a_ptr=name;
11:     b_ptr=name;
12:
13:     while( *b_ptr != '/' && *b_ptr != '\0'){
14:         b_ptr++; }
15:     if(*b_ptr == '/'){
15:         *b_ptr= '\0' ;
16:         b_ptr++;
16:     }
17:     printf(" First Name : %s \n", a_ptr);
18:     printf(" Family Name : %s \n", b_ptr);
19:
20:     exit(0);
21: }
22:  /***/

```

15: 氏名を区切る '/' が見つかり、その文字を文字列の終端をしめす、「ヌル」 '¥ 0' に置き換える。この時点で、文字列 `name []` の終端は '/' が格納されていた番地で終り、分割される。

17: `printf()` 関数の引数として文字列の開始アドレスを格納したポインタ変数を与えても、文字列を表示できる。

5.2.3 配列とポインタ変数の違い

配列の宣言、

```
char array[5];
array[0]='a';
```

では、配列のデータ型と要素数に合わせてデータの格納場所が確保されます。しかしポインタ変数の宣言

```
char *point;
*point='a';
```

では、ポインタの値（アドレス）の格納場所が確保されるだけで、データの格納場所は確保されません。

5.2.4 見掛け上のコピー

以下のプログラムは、ポインタ変数 `a_ptr` から `b_ptr` へ、ポインタの値（アドレス）がコピーされ、両者が同じ文字列を示すようにしただけです。文字列データ自体がコピーされたプログラムではない。

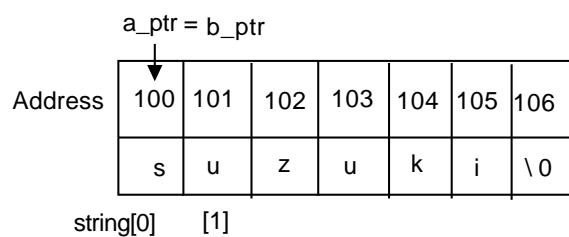


図 5.4: プログラムの動作

```
0:  /**508.c*****/
1:  #include <stdio.h>
2:
3:  int main(void){
4:      char string[100]; /* 入力される配列 */
5:      char *a_ptr, *b_ptr; /* char 型のポインタ変数 */
```

```

7:
8:     a_ptr=string;
9:     printf("Input String >> ");
10:    scanf("%s", a_ptr);
11:
12:    b_ptr=a_ptr;
13:
14:    printf(" Input : %s \n", a_ptr);
15:    printf(" copy : %s \n", b_ptr);
16:
17:    exit(0);
18: }
19:  /*****/

```

8: string の先頭アドレスをポインタ変数 a_ptr に代入する。

10: scanf() 関数で a_ptr に文字列を代入する。

12: b_ptr に a_ptr の内容を代入する。

14: printf() 関数で表示。

5.2.5 2つの文字配列を接続する

ここでは、2つの文字列たとえば、{a,b,c,d} と {a,b,c,d} を接続して、{a,b,c,d,a,b,c,d} と出力するプログラムをポインタを使って作ってみよう。

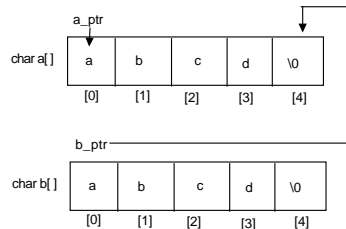


図 5.5: 文字配列の接続とポインタ操作

```

0:  /***509.c*****/
1:  #include <stdio.h>
2:  #define MAX 100
3:
4:  int main(void){
5:      char a[MAX];
6:      char b[MAX];

```



```
7:      char   *a_ptr, *b_ptr;          /* char 型のポインタ変数の定義 */
8:      int    i, a_count;             /* カウンター変数 */
9:
10:     a_ptr=a;                       /* 文字配列の先頭アドレス */
11:     b_ptr=b;                       /* 文字配列の先頭アドレス */
12:     printf("Input String 1>> ");
13:     scanf("%s", a_ptr);
14:     printf("Input String 2>> ");
15:     scanf("%s", b_ptr);
16:
17:     for( ; *a_ptr != '\0'; a_ptr++){
18:         /* 文字配列 a の終端'\0' を探す */
19:     }
20:     *a_ptr=*b_ptr;                 /* a の終端に配列 b の先頭を代入する */
21:     for(i=1;i<MAX; i++){
22:         *(a_ptr+i)=*(b_ptr+i);     /* 配列 b を配列 a に代入する*/
23:         if(*(b_ptr+i)=='\0'){ /* 文字配列 b の終端'\0' で終了する */
24:             break;
25:         }
26:     }
27:     printf(" Connect : %s \n", a);
28:     exit(0);
29: }
30: /*****/
```

5.3 演習問題

1. 5.1.3 のプログラムを参考にして、ポインタを使って四則演算を行うプログラムを作りなさい。
2. 5.1.4 のプログラムを参考にして、int 型、float 型、double 型の配列 a[5] の番地を表示して下さい。
3. int 型の配列 array[5]={1,2,3,4,5} に対して、array[i] と *(array+i) を出力するプログラムを作り両者を比べなさい。何がわかるか？
4. 5.2.5 のプログラムを参考にして、接続後に 2 つの文字列の間に '/' (スラッシュ) を入れるようにプログラムを改良してください。
5. キーボードから文字を 1 文字ずつ入力し出力するプログラムを、配列とポインタを使って作ってください。ただし、実行結果例が以下のようになるようにしてください。(実行結果例の cat などの文字や文字数は何でもよい)

```
%. /a.out
1文字入力してください>>c
a[0]=c
1文字入力してください>>a
a[1]=a
1文字入力してください>>t
a[2]=t
文字列 a=cat
%
```

6. 509.c を参考に、3つの文字配列を接続するプログラムを作ってください。
7. 名字と名前を入力し、名字 名前の順と名前 名字の順で出力するプログラムをポインタを使って作ってください。ただし、名字と名前の間にスペースをアスキー文字コードを使って1個入れるようにしてください。（スペースのアスキー文字コードは32です。）

第6章 ファイルの入出力

これまで用いてきたプログラムは、入力や出力はキーボードやディスプレイを用いたものでした。しかし、コンピューターのハードディスクに直接データを保存したり、プログラム上に読み込んだりする場合もあるでしょう。ここではファイルについて学びましょう。

6.1 ファイルの基本操作

6.1.1 ファイル・ポインタ

様々なデータをテキストファイルとして保存しておく、読んだり、変更したり出来て便利です。ファイルの操作は FILE 型のポインタ変数（ファイル・ポインタ）を用いて行われる。

```
/** ファイル・ポインタの宣言 ****/  
FILE *inp_file;
```

fopen() 関数：ファイルを開く標準関数、fgetc() や fputc() はファイルを読んだり書いたりするための標準関数、fclose() 関数は、ファイルを閉じる。

6.1.2 ファイルのオープン

ファイルを開くには fopen() 関数を使う。使い方は、

```
FILE *fp; /* ファイル・ポインタ */  
fp=fopen(t_file, "r");
```

第一引数”t_file”は開きたいファイルの名前、第2引数”r”はファイルを読むためのもの。書き込む時は”w”を使う。fp という名前の FILE 型ポインタ変数に fopen() 関数の戻り値（ファイル・ポインタ）を代入する。

6.1.3 ファイルのクローズ

ファイルを閉じるには、fclose() 関数を使う。fclose() 関数は引数として閉じたいファイルのファイル・ポインタを使う。

```
FILE *fp; /* ファイル・ポインタ */  
int flag;  
flag=fclose(fp);
```

fclose() 関数は、戻り値として、成功すれば(0)を失敗すれば(1)を返す。ファイルを開いたら必ず fclose() 関数でファイルを閉じるようにする。バッファド I/O 方式¹。

6.1.4 ファイルの読み出し

fgetc() 関数：ファイルから一文字の文字の読み出しを行う標準関数。fgetc() 関数は、呼び出される度に、ファイルの先頭から順に1文字ずつ文字を読み出します。fgetc() 関数の引数は、読みたいファイルのファイル・ポインタを指定する。返り値は読み出された文字か EOF を返す。

```
int moji;
FILE *fp;
moji=fgetc(fp);
```

何らかの理由で文字が読み出せない場合、fgetc() 関数は EOF の値を返します。EOF はファイルの終わり (End of File) を意味する。一般には (-1) という int 型の値を持っています。このような事情で、fgetc() 関数は int 型の変数を返す様に定義されている。

6.1.5 文字数を数える

ファイルに含まれる、文字数を数えるプログラムを作りましょう²。
考え方：fgetc() 関数でファイルを読んでいって、返り値が EOF になるまでの回数を数えればよい。

```
0:    /**601.c*****/
1:    #include <stdio.h>
2:    #define FOREVER 1
3:
4:    int main(void){
5:        char f_name[100]; /* ファイル名 */
6:        FILE *inp_file;   /* ファイル・ポインタ */
7:        int count;       /* 文字数を数える */
8:        int eof_flag;    /*読み込んだ文字を代入するところ */
9:
10:   printf("Input file name:");
11:   scanf("%s", f_name);
12:   inp_file=fopen(f_name, "r");
13:
14:   if(inp_file==NULL){
15:   printf("%s does not exist \n", f_name);
```

¹入出力が行われるたびに実際にファイルへ出力するのではなく、バッファと呼ばれる一時記憶領域にデータを貯めておく

²このプログラムを動かすには、あらかじめ、適当な名前のファイルを作りその中に文字を書き入れておく必要がある。数字を適当に並べたファイルを作りましょう。ファイル名は file1 としておきましょう。

```
16: exit(0);
17: }
18: count=0;
19: while(FOREVER){      /* 文字の終点を探す */
20:   eof_flag=fgetc(inp_file);
21:   if(eof_flag==EOF){
22:     break;
23:   }
24:   count++;          /*文字数を数える*/
25: }
26: fclose(inp_file);
27: printf("%s: %d chars \n", f_name,count);
28: exit(0);
29: }
30: /*****/
```

11: scanf() 関数でファイル名を読み込む

12: 読み出しモード”r”でファイルを開く

14: ファイル・ポインタが NULL (ファイルが存在しないときは NULL を返す) の時、ファイルのオープンに失敗したので exit(0) 関数でプログラムを終了する。

20: fgetc() 関数で文字を読み込んで、その文字を eof_flag に代入する。

21: eof_flag が EOF の時終了する。

6.1.6 ファイルの書き込み

ファイルへ 1 文字の書き込みを行うには、fputc() 関数を使う。fputc() 関数は呼び出される度に、ファイルの末尾に順に 1 文字ずつ書き込んでゆく。

```
int flag;
char moji;
FILE *fp;
flag=fputc(moji, fp);
```

/* moji: 書き込みたい文字、fp: 書き込みたいファイルのファイル・ポインタ*/
fputc() 関数は帰り値として、int 型を返す。

6.1.7 ファイルをコピー

ファイルの内容をコピーするプログラムを作りましょう。

考え方: 6-1.c のプログラムと同様にファイルから文字を読み出して EOF を探す。ただし、繰り返

しの中で、読み出した文字をそのまま別のファイルに書き込めばよい³。

```
0:     /**602.c*****/
1:     #include <stdio.h>
2:     #define  FOREVER  1
3:
4:     int  main(void){
5:         char  in_name[100]; /* 入力ファイル名 */
6:         char  out_name[100]; /* 出力ファイル名 */
7:         FILE  *in_file;     /* ファイル・ポインタ */
8:         FILE  *out_file;    /* ファイル・ポインタ */
9:         int  moji;         /* 読み込んだ文字 */
10:
11:        printf("Input file name:"); /*入力ファイル名 (file1) を入力 */
12:        scanf("%s", in_name);
12:        in_file=fopen(in_name, "r"); /*入力ファイルのオープン */
13:
14:        if(in_file==NULL){
15:            printf("%s does not exist \n", in_name);
16:            exit(0);
17:        }
18:
19:        printf("Output file name:"); /*出力ファイル名 (file2) を入力 */
20:        scanf("%s", out_name);
21:        out_file=fopen(out_name, "w"); /*出力ファイルのオープン, "w"で書き込む */
22:
23:        if(out_file==NULL){
24:            printf("%s does not exist \n", out_name);
25:            exit(0);
26:        }
27:
19:        while(FOREVER){ /* 文字の終点を探す */
20:            moji=fgetc(in_file);
21:            if(moji==EOF){
22:                break;
23:            }
24:            fputc((char)moji, out_file); /*文字 moji を書き込む*/
25:        }
26:        fclose(in_file);
27:        fclose(out_file);
```

³入力ファイル名には 6-1.c で用いたファイル名を使うとよい。出力ファイル名は file2 としておきましょう。

```

28:         exit(0);
29:     }
30:     /*****
```

21: EOF は int 型の値を返すので、moji は int 型変数として定義してある。

24: 文字 moji を out_file へ 1 文字ずつ書き込む。

6.2 データの書式付き入出力

前の章では 1 文字の入出力を行うための関数を扱いました。1 文字だけでなく、様々なデータの入出力を行うために、fscanf() と fprintf() 関数があります。書式は以下ようになります。printf() や scanf() 関数の引数に、ファイル・ポインタ (in_file や out_file) が付け加わっただけです。

```

flag=fscanf(in_file, "%d," &val);
flag=fprintf(out_file, "%d", val);
```

fscanf() 関数の戻り値は、int 型で、ファイルが終了して読み込むデータが無い場合は EOF となる。fprintf() 関数の戻り値は、通常はファイルの書き込まれた文字数です。しかしファイルへの書き込みを失敗すると EOF を返します。

6.2.1 ファイルに書き込むプログラム

sin 関数と cos 関数の 0 から 2π まで $\pi/10$ 刻みで関数表を作り、ファイルに書き込むプログラムを示します。⁴

```

0:  /****603.c*****/
1:      #include <stdio.h>
2:      #include <math.h>
3:      #define    MAX    20
4:      #define    PAI    3.1415926535
5:
6:      int  main(void){
7:
8:          int  count;                /* カウンタ変数 */
9:          double  theta;            /* 角度 */
10:         double  sin_ans, cos_ans; /* sin,cos の値 */
11:         char  out_name[100];      /* 出力ファイル名 */
12:         FILE  *out_file;          /* 出力ファイル・ポインタ */
```

⁴実行した結果をグラフにしてみよう。% gnuplot を使って、
gnuplot > plot "data1" using 1:2 with lines, ¥
> "data1" using 1:3 with lines

```

13:
14:     printf("Output file name: "); /* 出力ファイル名の入力 */
15:     scanf("%s", out_name);
16:     out_file=fopen(out_name, "w"); /*出力ファイルのオープン */
17:
18:     if(out_file ==NULL){
19:         printf("file open error <%s> \n", out_name);
20:         exit(0);
21:     }
22:
23:     for(count=0; count<=MAX; count++){
24:         theta=2.0*PAI*(double)count/(double)MAX;
25:         sin_ans=sin(theta);
26:         cos_ans=cos(theta);
27:
28:         fprintf(out_file, "%lf \t %lf \t %lf \n",theta, sin_ans, cos_ans);
29:     }
30:
31:     fclose(out_file);
32:     exit(0);
33:     }
34:     /*****/

```

14: 出力ファイル名は”data1”としておきましょう。

18: オープン失敗の場合は終了。

28: ”書式に注意”

6.2.2 ファイルを読み込んで表示する。

テキスト・ファイルに書かれた整数のデータを全て加算して、その結果をディスプレイに表示するプログラムを作りましょう。(このプログラムを実行させるあたって、入力ファイルには1行に1文字ずつの整数データがテキストファイルですすでに書かれている必要がある。演習問題2でファイルを作りましょう。)

```

0:  /****604.c*****/
1:  #include <stdio.h>
2:  #define FOREVER 1
3:
4:  int main(void){
5:      char in_name[100]; /* 入力ファイル名 */
6:      FILE *in_file; /*入力ファイル・ポインタ*/

```



```
7:         int number;          /* 読み出した整数 */
8:         int flag;            /* EOF のチェック */
9:         int total;          /* 読んだ整数の総和 */
10:
11:         printf("Input file name:");    /* 入力ファイル名の入力 */
12:         scanf("%s", in_name);
13:         in_file=fopen(in_name,"r");    /* 入力ファイルのオープン */
14:
15:         if(in_file==NULL){          /* オープン失敗の場合 */
16:             printf("%s does not exist. \n", in_name);
17:             exit(0);
18:         }
19:
20:         total=0;                    /* 初期値 */
21:         while(FOREVER){
22:             flag=fscanf(in_file, "%d", &number);
23:             if(flag==EOF){
24:                 break;
25:             }
26:             total += number;        /* 整数の和を求める */
27:         }
28:         fclose(in_file);
29:         printf("total=%d \n", total);
30:         exit(0);
31:     }
32: /*****/
```

6.3 演習問題

1. 6.1.5 のプログラム 601.c を参考にして、テキスト・ファイル中での行数を数えるプログラムを作りなさい。テキスト・ファイル中では、行末には改行文字 (' $\backslash n$ ') があることを利用してください。
2. 1 から 100 までの int 型整数を 1 行に 1 つずつ書き込むプログラムを作ってください。ただしファイルの名前は "data2" としていきましょう。604.c のプログラムの実行の時につかってみよう。
3. $y = (x - 1)^2 - 3$ を x の値を -1 から 3 の間で 0.01 ずつ計算し、 x と y の結果を 2 列でファイル data3 に書き込んで下さい。さらに gnuplot を用いてグラフ表示してください。
4. ファイル data2 を読み込んで、各データに 1 を加えて、その結果を同じファイル data2 に

書き込むプログラムを作りましょう。(実行結果を `cat` コマンドや `less` コマンドで見てください。)

5. ファイル `data2` を読み込んで、各データを2乗して、その結果をファイル `data4` に書き込むプログラムを作りましょう。
6. プログラム `602.c` を参考にして、2つのテキスト・ファイルの内容をつないで、別のファイルに書き込むプログラムを作ってください。

第7章 関数の作り方

c 言語では一連の手続きを関数という形でまとめておきます。関数の形にまとめた手続きは、その関数を呼び出すことで実行されます。すでに、`printf()` 関数や、`sin()` 関数などの標準関数などを利用してきました。`main()` も関数です。`printf()` 関数は出力の手続きをまとめた関数で、これを呼び出すことでデータの出力が行えます。便利な関数を自分で作っていきましょう。

7.1 関数

関数の基本的な構造は以下のようになります。

```

/*****
function() 関数 の定義
*****/
function() {
関数のプログラム
return;
}

```

関数の行う手続きは `return` 文によって終了します。外見上は `main()` 関数を書いたときとまったく同じ文法に従って、プログラムを書けばよい。任意の関数は `main()` 関数から呼び出して使うことができます。

7.1.1 引数を持たない関数

関数呼び出しの様子を理解するために、簡単なプログラムで考えていきましょう。以下のプログラムは `printf()` 関数を用いて、コメントを出力するだけですが、`main()` 関数から `function()` 関数を呼び出して、実行していることがわかります。

```

0:      /***701.c*****/
1:      #include <stdio.h>
2:
3:      int main(void){
4:          printf("Main >>1 \n");
5:          printf("Main >>2 \n");
6:          function();      /* function() 関数の呼び出し */

```

```

7:     printf("Main >>3 \n");
8:     exit(0);
9: }
10:  /*****/
11:     function(){
12:         printf("Function >>1 \n");
13:         printf("Function >>2 \n");
14:         return;
15:     }
16:  /*****/

```

7.1.2 引数を持つ関数

function() 関数に2つの整数型引数 a, b を与える例。引数は何個でも定義できる。

```

function(int a, int b) {
    関数のプログラム
    return;
}

```

main() 関数から function() 関数を呼び出す時、引数のデータ型が一致していなくてはいけない。以下の例で考えてみよう。main() 関数から test_fn() 関数を呼び出すプログラムです。

```

0:     /***702.c*****/
1:     #include <stdio.h>
2:
3:     int main(void){
4:         int main_val;
5:         main_val=2;
6:         printf("p1: main_val=%d \n", main_val);
7:         test_fn(main_val);
8:         printf("p4: main_val=%d \n", main_val);
9:         exit(0);
10:    }
11:    /*****
12:        関数名: test_fn
13:        引数: int test_va;
14:    *****/
15:    test_fn(int test_val){
16:        printf("p2: test_val=%d \n", test_val);
17:        test_val++;

```

```

18:     printf("p3: test_val=%d \n", test_val);
19:     return;
20: }
21:  /*****/

```

4: int 型変数 main_val を定義。

7: 変数 main_val を実引数として test_fn() 関数が呼び出される。

14: 呼び出された test_fn() 関数では、仮引数 test_val に値を引き取る。

15: 引数の受け渡しが成功しているかを確かめるために、printf() 関数で仮引数 test_val の値を表示します。

18: test_fn() 関数が終了して main() 関数の 8 行目に戻ります。

7.1.3 引数の受け渡しの例

2つの整数を入力し、大きい方を表示するプログラムを作りましょう。2つの整数を引数として受け取り、大きい方を表示する max_out() 関数を作り、main() 関数から呼び出すプログラムにします。

```

0:  /***703.c*****/
1:  #include <stdio.h>
2:
3:  int main(void){
4:      int a, b; /* 入力する2整数 */
5:      printf("Input Number 1 >> ");
6:      scanf("%d", &a);
7:      printf("Input Number 2 >> ");
8:      scanf("%d", &b);
9:
10:     max_out(a, b);
11:     exit(0);
12: }
13:  /*****
14:     関数名: max_out
15:     引数:   int a, int b
16:  *****/
17:  void max_out(int a, int b){
18:      if(a>b){
19:          printf("%d \n", a);
20:      }
21:      else{

```

```

22:         printf("%d \n", b);
23:     }
24:     return;
25: }
26:  /***** */

```

17: 戻り値を持たない void 型 max_out() 関数の宣言

7.1.4 引数と戻り値を持つ関数

関数から計算結果などのデータを返す方法として、戻り値を用いることが出来る。例えば、標準関数 $\sin(x)$ 関数は引数が double 型変数 x で、戻り値として $\sin(x)$ の値を返す関数です。戻り値を返すためには、関数定義で関数のデータ型を宣言するとともに、return 命令で戻り値として返すデータを指定する。もし、引数や、戻り値が無い場合は、void 型を用いる。

```

int function(int a, int b) { /*int 型の変数を戻り値として持つ function() 関数 */
    int count;
    count=0;

    return(count); /* 戻り値 count */
}

```

7.1.5 関数のプロトタイプ宣言

関数はプロトタイプ宣言を行うこと。関数を呼び出す前にプロトタイプ宣言を行うと、コンパイラはあらかじめ関数の型や引数について知ることが出来、データの引き渡しの間違いを避けます。#include と main() 関数の間に以下の書式で書く。

```

int function(int, int ); /**関数のプロトタイプ宣言 ***/

```

2 整数の差を求めるプログラムと作りましょう。ただし、2 整数の差を求める関数 num_diff() を作り、main() 関数から呼び出すようにしましょう。

```

0:  /***704.c**** */
1:  #include <stdio.h>
2:  int  num_diff(int, int);
3:
4:  int  main(void){

```

```
5:         int  a, b; /* 入力する2整数 */
6:         int  answer;
7:         printf("Input Number 1 >> ");
8:         scanf("%d", &a);
9:         printf("Input Number 2 >> ");
10:        scanf("%d", &b);
11:
12:        answer=num_diff(a, b);
13:        printf(" Difference=%d \n ", answer);
14:        exit(0);
15:    }
16:    /*****
17:    関数名：  num_diff
18:    引数：    int  a, int b
19:    戻り値    diff
19:    *****/
20:    int num_diff(int  a, int b){
21:        int diff;
22:        if(a>b){
23:            diff=a-b;
24:        }
25:        else{
22:            diff=b-a
26:        }
27:        return(diff);
28:    }
29:    /*****/
```

2: int 型の戻り値をもつ num_diff(int, int) 関数のプロトタイプ宣言。引数は int 型の変数 2 個。

12: num_diff(int, int) 関数を呼び出す。20 行目に行く。

27: int 型の戻り値 (diff) を返し、その値が 12 行目の answer に代入される。

7.2 変数の特性：スコープ

宣言された変数があどの範囲で使用できるかを示すのがスコープといいます。変数名は重複しないように定義しなくてはいいけない。関数 { } 内で定義された変数はローカル変数とよばれ、その関数内だけで使える。これに対して、main() 関数の上で定義された変数はグローバル変数とよばれ、プログラムの全範囲で有効である。

7.2.1 int型とdouble型の関数(スコープ)

以下のプログラムはint型とdouble型の関数を呼び出す例です。aとbの値を関数で計算してmain()関数内でaとbの和を計算しています。

```

0:      /**705.c*****/
1:      #include <stdio.h>
2:      int  baiint(int);
3:      double baidbl(double);
3:      double s;      //グローバル変数
4:      int  main(void){
5:          int  a;      // ローカル変数
6:          double  b,c;      // ローカル変数
7:              s=1.0;
8:          a=baiint(123);
9:          printf("%d \n",a);
10:
11:          b=baidbl(1.2);
12:          printf("%lf \n",d);
13:          c=(double)a+b;
14:          printf("%lf \n",c);
15:          exit(0);
16:      }
17:      /*****
18:      関数名:  baiint()
19:      引数 :      int  a_val      // ローカル変数
20:      戻り値      ib      // ローカル変数
21:      *****/
22:      int  baiint(int  a_val){
23:          int  ib;
24:          ib=2*a_val*s;
25:          return(ib);
26:      }
27:      /*****
28:      関数名:  baidbl()
29:      引数 :      double  b_val      // ローカル変数
30:      戻り値      db      // ローカル変数
31:      *****/
32:      double  baidbl(double  b_val){
33:          double  db;
34:          db=2.0*b_val*s;

```



```
35:         return(db);
36:     }
37:     /*****
```

7.2.2 変数の記憶クラス

変数のスコープと並んで、変数の重要な性質として記憶クラスがあります。例えば、グローバル変数は全ての関数から呼び出される可能性があるため、プログラムの実行から終了までずっと有効です。このように実行中ずっと有効であるような変数は記憶クラスが恒久的 (static 変数) であると呼びます。

一方、ローカル変数は、ローカル変数を定義したブロック内 { } だけで有効であり、ブロック内のプログラムが終了すると破棄されます。このように変数の寿命が限られている変数は、記憶クラスが一時的 (auto 変数) であると呼びます。ローカル変数の記憶クラスを auto にすることは、メモリの有効利用に効果的です。

```
static int a; /* static 変数 a の宣言 */
auto int b; /* auto 変数 b の宣言 */
```

自分が呼び出された回数を返り値として返す関数を作り、その関数を 10 回呼び出すプログラムを作りましょう。

```
0:     /***706.c*****/
1:     #include <stdio.h>
2:     int call_num(void);
3:
4:     int main(void){
5:         int num; /* 関数の呼び出し回数 */
6:
7:         do{
8:             num=call_num();
9:             printf("call_num >> %d \n", num);
10:        }while(num<10);
11:
12:        exit(0);
13:    }
14:    /*****
15:        関数名: call_num
16:        引数:   なし
17:        返り値 c_num
18:        *****/
19:    int call_num(void){
```

```

20:         static int c_num=0;
21:
22:         c_num++;
23:         return(c_num);
24:     }
25:     /*****/

```

20: static 変数 `c_num` の定義。static 変数は関数が呼び出されていない間もデータを保持し続けます。このため、static 変数の `c_num` は、前回呼び出された時の値を保持しています。また static 変数の初期化はプログラムの実行時に一度だけ行われます。

7.3 文字列を渡す

`char` 型の文字列変数を関数に渡す例を示します。`strout()` 関数を宣言して、引数を文字列型変数 `string` にして文字を渡します。

```

0:     /****707.c*****/
1:     #include <stdio.h>
2:
3:     void strout(char ss[]);
4:
5:     int main(void){
6:         char string[100];
7:
8:         printf(" Input string >>");
9:         scanf("%s",string);
10:
11:         strout(string);
12:         exit(0);
13:     }
14:     /*****/
15:     void strout(char ss[]){
16:         int i;
17:         printf("ss=%s \n",ss);
18:         for(i=0; ss[i] !='\0'; i++){
19:             printf("%x\t",ss[i]);
20:         }
21:         printf("\n");
22:         return;
23:     }

```

24: /*****/

11: 文字配列 string を引数として渡す。文字列 string の先頭アドレスが渡される。

18: 文字列の最後は'¥0' で終わることを利用して, strout() 関数を終了する。

19: 文字コードを 16 進数で表示する (21: 改行をしめす)。

7.4 演習問題

1. 7.1.5 を参考にして, 3 つの double 型変数の和を求めるプログラムをつくりましょう。ただし, 3 つの double 型変数の和を求める関数 num_sum() を作り, main() 関数から呼び出すようにしましょう。
2. 7.2.2 のプログラムで, call_num() 関数中の変数 c_num を, static 変数ではなく, auto 変数として宣言して見てください。プログラムを実行すると何が起こるか?
3. 2 つの整数 m と n をキーボードから入力して, 大きい方の数字を表示するプログラムを関数 imax(m,n) を使って書いてください。
4. 実数型引数 x について, x の範囲を -1 から 2 まで 0.01 ずつ変化させた時, $f(x) = x + \exp(-x^2)$ の値を返すプログラムを関数を用いて書いてください。ただし, x の値と $f(x)$ の値をファイル data4 に保存するようにしてください。得られた結果を gnuplot で表示して下さい。
5. $ax^2 + bx + c = 0$ の解を求めるプログラムを関数を用いて書いてください。ただし, a,b,c はキーボードから入力し, $a = 0$ の場合も解を与えるようにして下さい。また, 解が存在しない時は, なんらかのメッセージを出すようにして下さい。
6. 関数を使ったプログラムを作ってください。テーマはなんでも構いません。

第8章 関数とポインタ

8.1 関数の引数とポインタ

ここでは、関数を呼び出す際の引数として、変数のかわりにアドレス（ポインタ）を引数として渡す場合を考えましょう。

8.1.1 ポインタ渡し

アドレスを引数として渡す例を以下に示します。

```

/*****/
int main(void){
int number;
    プログラム略
function(&number);
exit(0);
}
/*****/
void function(int *arg){
*arg=*arg*2;
return;
}

```

main() 関数で &number を実引数として、function() 関数を呼び出している。実引数 &number は int 型の変数 number のアドレスです。function() 関数は、仮引数として int 型のポインタ引数 arg を用いて、実引数のアドレスを受け取ります。ポインタ引数 arg には main() 関数の変数 number のアドレスが格納されています。したがって、*arg には main() 関数の変数 number の値を示すことになり、*arg への代入は number への代入と同じ効果になります。

以下は入力された数値を2倍にする「ポインタ渡し」によるプログラムです。

```

0:     /***801.c*****/
1:     #include <stdio.h>
2:     void func(int *); /* ポインタを引数とした関数のプロトタイプ宣言 */
3:
4:     int main(void){

```

```

5:         int number; /* 入力される数値 */
6:
7:         printf("Input Number >>");
8:         scanf("%d",&number);
9:
10:        func(&number);
11:        printf("answer=%d \n",number);
12:        exit(0);
13:    }
14:    /*****
15:        関数名: func()
16:        引数:   int *arg      2倍するデータのポインタ
17:        *****/
18:    func(int *arg){
19:        *arg=*arg*2;
20:        return;
21:    }
22:    /*****/

```

8.1.2 複数のデータを返す関数

ポインタ渡しを用いた関数の呼び出しを利用すると、データを1つしか返せない(戻り値が1つ)というC言語の関数宣言を解決することが出来ます。ポインタ渡しの方法で、複数の変数のポインタを渡すことによって、呼び出した側の変数を書き換えることが可能になります。

2つのint型の変数の割り算を行い、商と余りを求める関数を作りましょう。

考え方: 2つのint型の変数を引数にとり、商と余りをポインタ引数で返すようにする。

```

0:    /***802.c*****/
1:    #include <stdio.h>
2:    void warizan(int , int, int *, int *); /* 割り算関数 */
3:
4:    int main(void){
5:        int a, b; /* 入力される2つの数値 */
6:        int div, mod; /* 商と余りの変数宣言 */
7:
8:        printf("Input Number , a >>");
9:        scanf("%d",&a);
10:       printf("Input Number, b >>");
11:       scanf("%d",&b);
12:

```

```

13:     warizan(a,b, &div, &mod);
14:
15:     printf("div=%d \n",div);
16:     printf("mod=%d \n",mod);
17:     exit(0);
18: }
19: /*****
20:     関数名:  warizan()
21:     引数 :   int a, b   割り算するデータ
22:             int *div_ptr, *mod_ptr   商と余りのポインタ
23:     *****/
24: void warizan(int a_val, int b_val, int *div_ptr, int *mod_ptr){
25:     *div_ptr=a_val/b_val;
26:     *mod_ptr=a_val % b_val;
27:     return;
28: }
29: /*****/

```

13: アドレスが関数の引数になっている。

8.2 関数の引数と配列

8.2.1 配列の受け渡し

関数の呼び出しの際に引数として配列を渡すことができます。

文字列をコピーするプログラムを作りましょう。ただし文字列コピーをする関数 `strcpy()` を作り、プログラムを作りましょう。

```

0:     /***803.c*****/
1:     #include <stdio.h>
2:     void  strcpy(char [], char [] );
3:
4:     int  main(void){
5:         char a_str[100];    /* 入力される文字列 */
6:         char b_str[100];   /* コピー先の文字列 */
7:
8:         printf("Input string 1 >>");
9:         scanf("%s",a_str);
10:
11:         strcpy(a_str, b_str);

```

```

12:
13:     printf("Input: %s \n",a_str);
14:     printf("Output: %s \n",b_str);
15:     exit(0);
16: }
19: /*****
20:     関数名: strcpy()
21:     引数:   char   a_arg[], b_arg[]
22:           a_arg[] から b_arg[] にコピーする関数
23:     *****/
24: void strcpy(char a_arg[], char b_arg[]){
25:     int count;
26:
27:     for(count=0; a_arg[count] != '\0' ; count++){
26:         b_arg[count]=a_arg[count];
27:     }
28:     b_arg[count]= '\0' ;
29:     return;
30: }
31: /*****/

```

2: 関数の定義で、仮引数の配列にはカッコ () の中に要素数は書かない。

11: 引数として配列を渡すときは、配列の先頭アドレスを渡している。

24: 配列は値渡しで引数が受け渡されるのではなく、ポインタ渡しと同じ方法で、アドレスが渡される。配列のように大きなデータを渡すときは、全てのデータ変数を渡すより、アドレスを渡すことで、メモリ使用の削減や速度のアップを行っています。

8.2.2 配列とポインタ

プログラム 803.c で使ったように、配列を引数として関数を呼び出す場合は、実際には配列のデータではなく配列の先頭アドレスが呼び出される関数に渡されます。この節では、配列の形式で受け取るかわりに、呼び出される関数の引数をポインタ引数として受け取る方法を考えましょう。

次のプログラムは、ポインタを利用した、文字列をコピーするプログラムです。

```

0:     /***804.c*****/
1:     #include <stdio.h>
2:     void strcpy(char *, char *);
3:
4:     int main(void){
5:         char a_str[100];     /* 入力される文字列 */

```



```

6:      char b_str[100];    /* コピー先の文字列 */
7:
8:      printf("Input string 1 >>");
9:      scanf("%s",a_str);
10:
11:      strcpy(a_str, b_str);
12:
13:      printf("Input: %s \n",a_str);
14:      printf("Output: %s \n",b_str);
15:      exit(0);
16:  }
19:  /*****
20:      関数名:  strcpy()
21:      引数:   char *a_arg, *b_arg
22:             a_arg[] から b_arg[] にコピーする関数
23:  *****/
24:  void strcpy(char *a_arg, char *b_arg){
25:
26:
27:      for( ; *a_arg != '\0' ; a_arg++, b_arg++){
26:          *b_arg=*a_arg;
27:      }
28:      *b_arg= '\0' ;
29:      return;
30:  }
31:  /*****/

```

2: 関数の引数はポインタになっている。

27: ポインタ演算によってコピーを行っている。

8.3 main() 関数の引数

ファイルをコピーするコマンド (cp) は

```
% cp file1 file2
```

で、file1 の内容が file2 へコピーされます。c 言語のプログラムがコマンド・ラインからパラメータを得るためには main() 関数に次のような仮引数を持つことができます。

```
int main( int argc, char *argv[])
```

仮引数には `argc` や `argv` という名前が使われるのは慣例となっている。仮引数 `argc` は、パラメータの個数が格納されます。プログラムの名前自身もパラメータとして扱われます。前の例では `argc` は `cp` と `file1` と `file2` で `argc=3` となります。第 2 引数の `argv` は `char` 型のポインタ引数です。`argv[0]` は最初のパラメータの文字列 "cp" を示しています。同様に `argv[1]` は `file1` を、`argv[2]` は `file2` を指しています。次のプログラムを実行してみましょう。

```

0:      /**805.c*****/
1:      #include <stdio.h>
2:
3:      int main( int  argc, char *argv[]){
4:          int  count; /* カウンタ変数 */
5:
6:          for(count=0; count<argc; count++){
7:              printf("No. %d : %s \n", count, argv[count]);
8:          }
9:          exit(0);
10:     }
11:     /***/

```

% cc 805.c でコンパイルした場合は

% ./a.out file1 file2

で実行してみる。

8.3.1 main() 関数の引数渡し

入力された 2 つの数値を合計するプログラムを作りましょう。ただし、プログラム名は `806.c` とし、入力する変数をコマンドラインで入力するようにしなさい。

```

0:      /**806.c*****/
1:      #include <stdio.h>
2:      #include <stdlib.h> /* atoi() 関数を使うためのヘッダー */
3:
4:      int main( int  argc, char *argv[]){
5:          int  sum;
6:
7:          if(argc !=3){
8:              printf(" Error \n");
9:              exit(1); /* エラーならプログラムはここで終了する */
10:         }
11:         sum=atoi(argv[1])+atoi(argv[2]);
12:         printf("sum=%d \n", sum);

```

```

13:
14:     exit(0);
15: }
16:  /*****/

```

7: 引数の個数をチェックして、argc=3 でない場合は終了する。

11: atoi() 関数は文字列を int 型の数値に変換する関数です。

実行例

```
% cc -o 806 806.c
```

```
% 806 100 200
```

8.4 関数を指すポインタ

関数をポインタとして使うには、次のように関数へのポインタの宣言をしなければいけない。

```
int (*function)(); /* function は int 型を返す関数へのポインタである。*/
```

*function は int 型を返す関数を意味するから「function」はそれへのポインタである。

次のプログラムはポインタ function を使いながら、あるときは sum() 関数を実行し、またあるときは dif() 関数を実行しています。

```

0:  /***807.c*****/
1:  #include <stdio.h>
2:
3:  void (*function)(int a, int b); /* 関数へのポインタ function の宣言 */
4:  void sum(int a_val, int b_val);
5:  void dif(int a_val, int b_val);
6:
7:  int main(void){
8:      int a=100;
9:      int b=100;
10:
11:     function=sum; /* 関数アドレス sum を設定し、 */
12:     function(a,b); /* そのアドレスを実行する。 */
13:     function=dif; /* 関数アドレス dif を設定し、 */
14:     function(a,b); /* そのアドレスを実行する。 */
15:
16:     exit(0);
17: }
18:  /*****/

```

```

19: void sum(int a_val, int b_val){
20:     printf("a+b=%d \n", a_val+b_val);
21:     return;
22: }
23: /*****/
24: void dif(int a_val, int b_val){
25:     printf("a-b=%d \n", a_val-b_val);
26:     return;
27: }
28: /*****/

```

3: int 型の引数を 2 つ使用し、戻り値は void である。

8.5 関数へのポインタの配列

関数へのポインタは配列とすることも出来ます。その配列要素はいくつもの関数の入り口アドレスを持ちます。たとえば、

```
void (*keisan[4])(int a , int b );
```

とすると、keisan は、2 つの int 型の引数を持ち、戻り値が void 型である関数への入り口アドレスを 4 つ持つことが出来る、ポインタ配列となります。

次のプログラムは、ポインタ keisan に、wa, sa, seki, syo という 4 つの関数を割り当てたものです。

```

0:  /**808.c*****/
1:  #include <stdio.h>
2:
3:  void (*keisan[4])(double a, double b);/* 関数へのポインタfunctionの宣言 */
4:  void wa(double a, double b);
5:  void sa(double a, double b);
6:  void seki(double a, double b);
7:  void syo(double a, double b);
8:
9:  int main(void){
10:     int i;
11:     keisan[0]=wa;          /*****/
12:     keisan[1]=sa;          /* 4つの関数のアドレスを設定する */
13:     keisan[2]=seki;        /*****/
14:     keisan[3]=syo;        /*****/
15:
16:     for(i=0; i<=3; i++){

```

```
17:         keisan[i]( 100.0, 200.0);
18:     }
19:     exit(0);
20: }
21: /*****/
22: void wa(double a, double b){
23:     printf("a+b=%lf \n", a+b);
24:     return;
25: }
26: /*****/
27: void sa(double a, double b){
28:     printf("a-b=%lf \n", a-b);
29:     return;
30: }
31: /*****/
32: void seki(double a, double b){
33:     printf("a*b=%lf \n", a*b);
34:     return;
35: }
36: /*****/
37: void syo(double a, double b){
38:     printf("a/b=%lf \n", a/b);
39:     return;
40: }
41: /*****/
```

8.6 演習問題

- 2つの実数型変数の四則演算を行うプログラムをポインタを使って作りなさい。いろいろな方法があります。
- 806.c と 807.c を参考にして、コマンドラインから2つの整数を入力して和と積を求めるプログラムを、関数へのポインタを使って作ってください。
- 2つの文字列を接続するプログラムを作りなさい。ただし、文字列を接続する関数を `strjoin()` を作り、`main()` 関数とその関数を呼び出す形式でプログラミングしてください。
 - 803.c を参考にして配列を用いてプログラムしなさい。
 - 804.c を参考にして配列とポインタを用いてプログラムしなさい。
- 定価（税抜き） x 円の商品を、支払い金額 y 円出して購入しました。税込み価格、支払い金額、おつりを表示するプログラムをつくりましょう。ただし、定価（税抜き）と支払い金額

を2つの引数にとり、税込み価格、支払い金額、お釣りをポインタ引数で返す関数 `buy()` を定義して使って下さい。(802.c を参照)

5. 3次元空間上の点 P の位置 (x, y, z) をコマンドラインから実数型で3つ入力して、原点から点 P までの距離を出力するプログラムを作ってください。文字列を実数型に変換する関数、`atof()` を用いる。
6. 任意の関数 $f(x)$ の $x = a$ での1回微分、2回微分を”数値微分”で計算するプログラムを作りましょう。ただし、関数のポインタ配列 `*bibun[]` に `first()`、`second()` という関数を割り当て、それぞれ1回微分、2回微分を代入して下さい。数値微分は

$$f'(x) = (f(x + \delta x) - f(x)) / \delta x,$$

$$f''(x) = (f(x + \delta x) - 2f(x) + f(x - \delta x)) / (\delta x)^2,$$

を用いてください。例として、関数 $f(x) = \log(1+x)$ の $x = 0$ での1回微分と2回微分の値を表示してください。 δx の値は0.001ぐらいにしておきましょう。

7. 任意の関数 $f(x)$ は $x = a$ の近傍で、テーラー級数

$$f(x) = f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \dots$$

に展開することが出来る。前のプログラムを発展させ、 a の値を入力したときに関数 $f(x)$ のテーラー展開を x の二次まで表示するプログラムを作ってください。1例として、 $f(x) = \log(1+x)$ としましょう。その後、任意の関数でやってみましょう。

第9章 構造体と列挙型

整数を取り扱うには `int` 型や `long` 型、実数は `float` 型や `double` 型、文字を扱うには `char` 型など様々なデータ型を用いてきました。C 言語ではこのようなデータ型を作り出す機能が備わっています。この章では、データ型を新たに作り出し、プログラムをスマートに書く方法について学びましょう。

9.1 構造体

9.1.1 複素数を扱うプログラム

C 言語では、複素数 ($3+2i$ など) を扱う複素数型は存在しません。まずは、2つの複素数の足し算を行うプログラムを作りましょう。この例では、実部と虚部をそれぞれ、変数に入力し、その後、実部どうしを足し、虚部どうしを足すことで、複素数の足し算を行っています。

```
0:      /**901.c*****/
1:      #include <stdio.h>
2:
3:      int main(void){
4:          double a_re, a_im; /*入力される複素数の実部と虚部 */
5:          double b_re, b_im; /*入力される複素数の実部と虚部 */
6:          double ans_re, ans_im; /*答えの複素数の実部と虚部 */
7:
8:          printf("Input complex 1 re >>");
9:          scanf("%lf",&a_re);
10:         printf("Input complex 1 im >>");
11:         scanf("%lf",&a_im);
12:         printf("Input complex 2 re >>");
13:         scanf("%lf",&b_re);
14:         printf("Input complex 2 im >>");
15:         scanf("%lf",&b_im);
16:
17:         ans_re=a_re+b_re;
18:         ans_im=a_im+b_im;
19:
```

```

20:         printf("Answer: %lf+%lf i = \n", ans_re, ans_im) ;
21:         exit(0);
22:     /*****/

```

9.1.2 新しいデータ型の定義

プログラム 901.c では複素数のデータを2つの実数型のデータに分けて表しました。しかし、複素数は複素数のためのデータ型で扱った方が合理的です。c言語では構造体を用いて、新しいデータ型を定義できます。配列は同じデータ型のデータをまとめたものです。これに比べて、構造体は異なるデータ型をもつデータを自由にまとめておくことが出来ます。構造体にまとめられた個々の成分はメンバと呼ばれます。

構造体の定義は以下のように行います。

```

struct    complex { /*複素数型の宣言 */
                double    re; /* 実部 */
                double    im; /* 虚部 */
    };

```

一行目は構造体名 complex の定義、2行目は構造体のメンバを示します。ここでは、メンバ名を re と im としました。この例では、complex 構造体は、2つの double 型のメンバ re と im を含んでいます。このように定義された complex 構造体は、新しいデータ型として、そのデータ型をもつ変数を宣言できます。プログラム中での構造体の宣言は以下のように行います。

```

struct    complex    cmplx_val;

```

complex 構造体型で、変数 cmplx_val を宣言しています。

構造体を用いる場合、そのメンバを自由に操作出来なくてはなりません。構造体のメンバは、メンバ名によって識別されます。これを操作するには

```

cmplx_val.re=1.0;

```

のように構造体型の変数名の後ろに「.(ピリオド)またはドット演算子」を挟んでメンバ名を書きます。「構造体変数名.メンバ名」の形で普通の変数や配列などと同様に扱えます。

9.1.3 構造体を使ったプログラム

プログラム 9-1.c を構造体を使って書き換えましょう。

```

0:     /****902.c*****/
1:     #include <stdio.h>
2:     struct    complex{ /* 複素数型 */
3:         double    re; /* 実部 */

```



```

4:             double im; /* 虚部 */
5:     };
6:
7:     int main(void){
8:         struct complex a_val, b_val; /*入力される複素数 */
9:         struct complex ans; /*答えの複素数 */
10:
11:         printf("Input complex 1 re >>");
12:         scanf("%lf",&a_val.re);
13:         printf("Input complex 1 im >>");
11:         scanf("%lf",&a_val.im);
12:         printf("Input complex 2 re >>");
13:         scanf("%lf",&b_val.re);
14:         printf("Input complex 2 im >>");
15:         scanf("%lf",&b_val.im);
16:
17:         ans.re=a_val.re+b_val.re; /* 実部の足し算 */
18:         ans.im=a_val.im+b_val.im; /* 虚部の足し算 */
19:
20:         printf("Answer: %lf+%lf i \n", ans.re, ans.im) ;
21:         exit(0);
22:     /*******/

```

2: complex 構造体の定義。構造体の定義はプログラムの冒頭部分で行うようにする。

8: complex 型変数 a_val と b_val と ans の宣言。各変数は re と im の2つのメンバを持つ。

20: 実部 (ans.re) と虚部 (ans.im) の変数名を使って答えを書き出す。

9.1.4 構造体と関数

構造体として宣言された変数は、関数への引数として渡したり、関数からの戻り値として返したりできます。プログラム 902.c を、構造体を引数や戻り値とする関数を用いて書き換えた例を、以下のプログラムで示します。

```

0:     /**903.c*****/
1:     #include <stdio.h>
2:     struct complex{ /* 複素数型 */
3:         double re; /* 実部 */
4:         double im; /* 虚部 */
5:     };
6:     struct complex get_cmp(void);

```

```
7:     struct complex add_cmp(struct complex, struct complex);
8:     void
9:         put_cmp(struct complex);
10:
11:     int main(void){
12:         struct complex a_val, b_val; /*入力される複素数 */
13:         struct complex ans; /*答えの複素数 */
14:
15:         a_val=get_cmp();
16:         b_val=get_cmp();
17:
18:         ans=add_cmp(a_val,b_val);
19:
20:         put_cmp(ans);
21:         exit(0);
22:     }
23: /*****
24:     get_cmp() 複素数の入力のための関数
25:     引数: なし
26:     戻り値: struct complex 入力された複素数
27: *****/
28: struct complex get_cmp(void){
29:     struct complex inp_val; /*複素数の入力値*/
30:
31:     printf("Input complex re >>");
32:     scanf("%lf",&inp_val.re);
33:     printf("Input complex im >>");
34:     scanf("%lf",&inp_val.im);
35:     return(inp_val);
36: }
37: /*****
38:     add_cmp(struct complex, struct complex) 足し算をする関数
39:     引数: struct complex a_val と struct complex b_val
40:     戻り値: struct complex 足し算の答え。
41: *****/
42: struct complex add_cmp(struct complex a_arg, struct complex b_arg){
43:     struct complex answer; /*足し算の答え */
44:     answer.re=a_arg.re+b_arg.re; /* 実部の足し算 */
45:     answer.im=a_arg.im+b_arg.im; /* 虚部の足し算 */
46:     return(answer);
47: }
```

```

47:  /*****
48:  put_cmp(struct complex) 複素数の出力のための関数
49:  引数: struct complex out_data
50:  戻り値: なし
51:  *****/
52:  void put_cmp( struct complex out_data ){
53:      printf("Answer: %lf+%lf i \n", out_data.re, out_data.im) ;
54:      return;
55:  }
56:  *****/

```

6-8: complex 型の関数 (3 つ) の宣言。

11-12: complex 型の変数 (3 つ) の宣言。

14-15: データ入力のための get_cmp() 関数の呼び出し。

17: 足し算のための、add_cmp() 関数の呼び出し。

19: 結果の出力のための、put_cmp() 関数の呼び出し。

9.1.5 構造体とポインタ

int 型や char 型のポインタと同様に、構造体に対するポインタを考えることが出来る。構造体のポインタ変数の宣言は、以下のように行う。

a. 構造体のポインタ変数の宣言

```

struct complex *cmp_ptr;
    構造体名 ポインタ変数名

```

b. 構造体のポインタ変数によるメンバの操作

```

(*cmp_ptr).re=10.0;
    (*ポインタ変数名).メンバ名

```

c. 構造体のアロー演算子によるメンバの操作

```

cmp_ptr ->re =10.0;
    ポインタ変数名 -> メンバ名 (->はアロー演算子と呼ぶ)

```

9.1.6 アロー演算子: ->

アロー演算子は構造体変数がポインタによって示される時に使います。例として次のような構造体名 seiseki を考えましょう。

```

struct seiseki{
    char name[100];
    int butsuri;
    int program;
}

```

メンバーが char 型の name と、int 型の butsuri と program になってます。構造体変数は通常の変数と同じようにポインタで操作することが出来ます。例えば、

```

struct seiseki  gakusei;
struct seiseki *sp=&gakusei;

```

と宣言すると、ポインタ sp は構造体変数 gakusei の先頭アドレスを示します。(*sp) は値を示すので、ドット演算子でメンバを参照できます。例えば、

```
(*sp).butsuri=80;
```

と書くことが出来ます。アロー演算子を使うと上の文は

```
sp->butsuri=80;
```

となります。アロー演算子を使うと面倒な () 記述が不要になります。

次のプログラムはアロー演算子の利用例です。少しくどいプログラムですが、いろいろな使い方を覚えてください。

```

0:  /****904.c*****/
1:  #include <stdio.h>
2:
3:  struct seiseki{
4:      char name[30];
5:      float butsuri;
6:      float program;
7:  };
8:  int main(void){
9:      struct seiseki  gakusei;    /* seiseki 型をもつ変数 gakusei を宣言 */
10:     struct seiseki  *sp=&gakusei;    /*** ポインタ sp に変数 gakusei の ****/
11:     float  total_mem, total_dot, total_arro; /*** 先頭アドレスを与える ***/
12:
13:     printf("Your Name >>");
14:     scanf("%s",sp->name);
15:     printf("Score of Busturi >>");
16:     scanf("%f",&sp->butsuri);    /* アロー演算子を使ったメンバへの入力 */
17:     printf("Score of Program >>");
18:     scanf("%f",&sp->program);    /* アロー演算子を使ったメンバへの入力 */

```

```

19:
20:     printf("Name: %s \n", sp->name);
21:     printf("Butsuri: %f \n", sp->butsuri);
22:     printf("Program: %f \n", sp->program);
23:
24:     total_mem=gakusei.butsuri+gakusei.program; /*ドット演算子を使った和の計算
*/
25:     total_dot=sp->butsuri+sp->program; /*アロー演算子を使った和の計算*/
26:     total_arro>(*sp).butsuri>(*sp).program; /*ポインタ演算子を使った和の計算
*/
27:
28:     printf("Total_mem: %f \n", total_mem);
29:     printf("Total_dot: %f \n", total_dot);
30:     printf("Total_arro: %f \n", total_arro);
31:
32:     exit(0);
33: }
34: /*****/

```

9.2 列挙型とデータの定義

列挙型 (*enum*) は名前付き定数 (列挙定数) を使って、データの取りえる値を全て列挙したものです。意味の解りづらい数値データを、名前付き定数で操作出来るようにします。

テストの点数を引数として受けて、成績 A,B,C,D を返り値 (数字ではない) として返すプログラムを作りましょう。列挙型の定義は

```
enum seiseki{A, B, C, D};
```

で、列挙型名 (*seiseki*) の後ろに列挙型が取りうる値であるタグ名を並べて書きます。列挙型変数の定義は

```
enum seiseki class;
```

です。この例では、*seiseki* 列挙型の変数 *class* の定義を行っています。変数 *class* は、A, B, C, D の4種類の値だけを取ることが出来ます。

```

0: /****905.c*****/
1: #include <stdio.h>
2:
3: enum seiseki{A, B, C, D}; /* 成績のクラスの列挙型*/
4: enum seiseki get_class(double); /*成績を返す関数の宣言*/

```

```
5:
6: int main(void){
7:     double point;                /**成績**/
8:     enum seiseki class;          /**seiseki 列挙型の変数 class の定義**/
9:
10:    printf("Input point >> ");
11:    scanf("%lf", &point);
12:
13:    class=get_class(point);        /**引数 point で成績を返す**/
14:    switch(class){
15:        case A : printf("Class A \n");
16:                break;
17:        case B : printf("Class B \n");
18:                break;
19:        case C : printf("Class C \n");
20:                break;
21:        case D : printf("Class D \n");
22:                break;
23:        default : printf("Error \n");
24:    }
25:
26:    exit(0);
27: }
28: /*****
29:     get_class(double point )      成績を返す関数
30:     引数 : double point
31:     戻り値 : enum class
32: *****/
33: enum seiseki get_class(double point){
34:     if(point<25.0){
35:         return(D); /**戻り値として D を返す**/}
36:     else if(point<50.0){
37:         return(C); }
38:     else if(point<75.0){
39:         return(B); }
40:     else if(point>75.0){
41:         return(A); }
42: }
43: /*****/
```

9.2.1 データ型の定義

C言語ではデータ型の定義で、データ型に別名を付ける事が出来る。typedefによるデータ型の定義を行う。

```
typedef struct complex cmplx;
```

データ型 (struct complex) に新しい名前 (cmplx) を付ける。以降、名前 (cmplx) を用いてデータ型を宣言することが出来る。

プログラム 903.c の main() 関数をデータ型の定義で書き換えた例を示します。

```
0:      /*****/
1:      #include <stdio.h>
2:      typedef struct complex{      /* 複素数型 */
3:          double re;      /* 実部 */
4:          double im;      /* 虚部 */
5:      } cmplx ;      /* ** <===ここで別名を定義できる**** */
6:      cmplx get_cmp(void);
7:      cmplx add_cmp(cmplx, cmplx);
8:      void          put_cmp(cmplx);
9:
10:     int main(void){
11:         cmplx a_val, b_val; /*入力される複素数 */
12:         cmplx ans;      /*答えの複素数 */
13:
14:         a_val=get_cmp();
15:         b_val=get_cmp();
16:
17:         ans=add_cmp(a_val,b_val);
18:
19:         put_cmp(ans);
20:         exit(0);
21:     }
22:     /*****/
```

9.3 自己参照構造体

構造体の中に自分自身を指し示すポインタを含ませることが出来る。これを自己参照構造体と呼びます。自己参照構造体は同じタイプの構造体を次々に繋いでいき、データのチェーンを作るときに使われます。

例えば大量のデータを大きい順に配列

```
int data[1000];
```

に格納していたとします。2番目のデータを削除したい場合、データ自体は残したまま、図 12.1 の (b) の様にデータ自体は残したまま、1番目と3番目を結ぶことで削除したことと同じことになります。また2番目と3番目の間に新しいデータ6を挿入したいときは(c)のように順番を変えてやればよいのです。

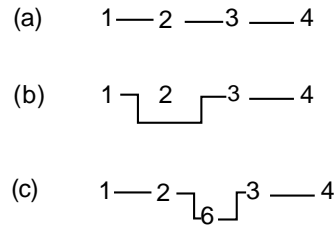


図 9.1: データのチェーン構造

9.3.1 自己参照構造体の宣言

次の例は、名前 (name) と物理の点数 (butsuri) からなるデータをチェーンできる自己参照構造体 (person) を宣言しています。

```
typedef struct person{
    char name[30];
    double butsuri;
    struct person *next;
}Person;
```

struct person *next; は次の person 構造体のアドレスが入る。つまり、ポインタ next が次の構造体のアドレスを示すことが出来ます。チェーンの終了は NULL を返すようにします。図 12.2 の (3) の next に NULL が格納されているときここでチェーンは終了を意味します。

次のプログラムは、キーボードから入力されたデータ (名前と点数) を次々に自己参照構造体に繋いでいき、点数の高い順に出力するようにします。

```
0:  /****906.c*****/
1:  #include <stdio.h>
2:  #include <stdlib.h> /* atof() malloc() 関数を使うためのヘッダー */
3:  #include <string.h> /* strcpy() 関数を使うためのヘッダー */
4:
5:  typedef struct person{ /* 構造体の宣言*/
6:      char name[30];
7:      double butsuri;
```

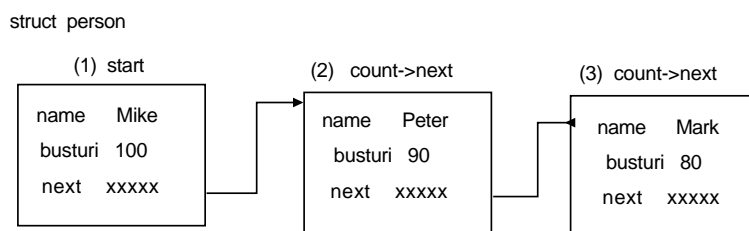



図 9.2: 構造体のモデル

```

8:      struct person *next; /* 自分自身を指し示すポインタ next */
9:  }Person; /* person 構造体の別名 Person */
10:
11:  int main(void){
12:      Person mark={"", 0, NULL}; /*** NULL を終端マークにする***/
13:      Person *start=&mark; /* ポインタ start に先頭アドレスを与える**/
14:      Person *ptr; /* Person 構造体のポインタ宣言 (作業用)**/
15:      Person *count; /* Person 構造体のポインタ宣言 (ループ制御用)**/
16:      char name[30], butsure_data[10]; /***データ入力用**/
17:
18:  while(1){
19:      /***キーボードからデータを入力する***/
20:      printf("Name=");
21:      gets(name);
22:      if(strcmp(name,"")==0){break;} /* 改行が入力されると入力終了する***/
23:
24:      printf("Butsuri=");
25:      gets(butsuri_data);
26:      /***構造体 1 個分のメモリを確保する ***/
27:      ptr=(Person *)malloc(sizeof(Person));
28:      if(ptr==NULL){ /***メモリの確保が失敗したとき Error を出力 **/
29:          printf("Error \n");
30:          exit(1);
31:      }
32:      /***確保した構造体にデータを入れる*/
33:      strcpy(ptr->name, name);
34:      ptr->butsuri=atof(butsuri_data);
35:
36:      /***作成した構造体 ptr を既存のチェーンにはめ込んでいく**/
37:      for(count=start; count->next != NULL; count=count->next){

```

```

38:   if(ptr->butsturi > count->next->butsturi){ /**次のデータより点数が高いとき**/
39:       ptr->next = count->next;                /** チェーンをつなぎ変える.**/
40:       count -> next= ptr;
41:       break;
42:   }
43: }
44: /**最低点の時チェーンの最後につなぐ**/
45:   if(count->next == NULL){ /**チェーンが NULL に行き着いたとき**/
46:       count->next=ptr; /**最後尾に構造体をつける**/
47:       ptr->next=NULL; /** 新たに最後尾マークを付ける */
48:   }
49:
50: } /******* while() 文はここまで繰り返す*****/
51:
52: for(count=start->next; count != NULL; count=count->next){
53:     printf("%s %f \n", count->name, count->butsturi);
54: }
55:
56:     exit(0);
57: }
58: /*******/

```

21: gets(ss) 関数: 1 行入力して ss に格納する。

27: malloc(size) 関数: size バイトのメモリを確保する。malloc() 関数の戻り値は void 型のポインタであるので、使いたいデータ型にキャストする必要がある。ここでは Person 型にキャストしている。

33: strcpy(ss, "test"): 文字列 ss に"test"をコピーする。

34: atof(ss) 関数: 文字列 ss を double 型に変換して ptr->butsturi に代入する。(4 0 行目)

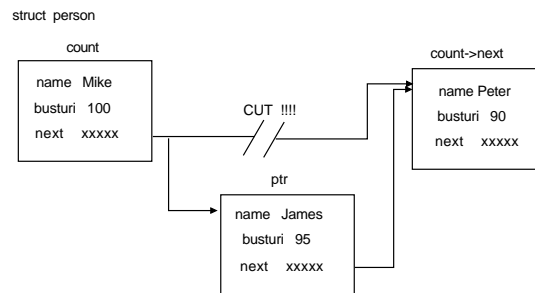


図 9.3: チェーンをつなぎ替え

36: チェーンの繋ぎ替えは

(1)ptr->butsturi=95 が count->next->butsturi(=90) より大きいとき、(38 行目)

- (2) Mike と Peter のラインを切り, Peter のアドレスを ptr->next に代入する。(3 9 行目)
 (ptr->next = count->next;)
 (3) ptr のアドレスを count->next に代入し、Mike→James へのラインを作る。

9.4 演習問題

1. プログラム 903.c を参考にして、複素数の引き算と掛け算を行う関数と加えて、足し算、引き算、掛け算を出力するプログラムを作りなさい。
2. 9.2.1 を参考にして、問 1 をデータ型の定義を用いて書き換えよ。
3. 試験の成績のデータ処理をしたい。氏名と、物理、化学、生物の点数を次の構造体の配列で扱い、各教科の平均値と最大値、最小値を求めるプログラムを作ってください。

```
struct seiseki_data{
    char   name[20];
    double  butsuri;
    double  kagaku;
    double  seibutsu;
};
```

構造体の初期化 (データ入力) はたとえば、

```
struct seiseki_data  data1[20]={
                                { "田中", 100, 90,90},
                                { "鈴木", 80, 90,90},
                                { "松本", 100, 80,70},
                                ..... ,
};
```

のようにする。

4. プログラム 906.c を参考にして、物理、生物の点数を入力してその合計点の高い人順に出力するプログラムを作りなさい。
5. 3次元座標上の2点 $A(x_1, y_1, z_1)$ と $B(x_2, y_2, z_2)$ の間の距離を求めるプログラムを作りましょう。ただし、2点の座標はキーボードから入力する。3次元座標上の任意の点の座標を構造体

```
point_3d{ }
```

をつかって扱うように。

第10章 Java 言語の基本

あと3回の授業で、Javaの基本と簡単なアニメーションについて学びましょう。JavaはC言語を発展させた言語なので、C言語と同じ部分が沢山ありますが、Java独自の書き方も覚えましょう。Javaは1995年にSunマイクロシステム社が発表した、OSを選ばないプログラミング言語で、Sun社のホームページからフリーでダウンロード出来るフリーソフトです。自分のパソコンにインストールして使って下さい。

10.1 Javaプログラムの実行

10.1.1 Helloプログラム

次のプログラムを入力して「s1001.java」という名前でファイルを保存してください。

```
0:    /**s1001.java*****//
1:    public class s1001{           /** クラス名は s1001  ***/
2:        public static void main(String args[]){ /**決まり文句
3:            System.out.println("Hello");      /**出力命令 **/
4:        }
5:    }
6:    /*******//
```

Javaのファイルを保存するときはクラス名(プログラム)と同じ名前にしなければいけない。

コンパイルするには、

```
%javac s1001.java
```

とします。コンパイルが成功すれば「s1001.class」というファイルが自分のディレクトリーに出来ているはずですが、lsコマンドで確かめてください。

プログラムの実行命令は

```
%java s1001
```

です。上の例でプログラム名「s1001」のところを、いろいろなプログラム名に置き変わります。

次のプログラムを入力し実行してください。

```
0:    /**s1002.java*****/
1:    import java.awt.*;
2:    public class s1002{
3:        public static void main(String args[]){
4:            Frame t=new Frame("Sample");
5:            t.setSize(300,100);
6:            t.setVisible(true);    /***又は , t.show(); ***/
7:        }
8:    }
9:    /***/
```

1: java.awt というパッケージ内¹にある全てのクラスをインポートする。

4: クラス Frame のインスタンス t を生成。

5: 画面のサイズを指定するメソッド setSize(i, j)。引数は幅 i=300, 縦 j=100 ピクセルを int 型で指定している。

6: 表示の実行。

小さなウィンドが現れましたね。Java はこういったグラフィックスが容易に出来るのも特徴です。

10.2 Java プログラムの形式

Java にはアプリケーションとアプレットという 2 つの実行形式があります。アプリケーションは普通の使い方で自分のコンピューターに結果を表示する使い方です。一方、アプレットはインターネットのホームページ上で実行する形式を持ちます。遠く離れたインターネット上の他人のコンピューター上で実行されます。

Java の基本的な部分は C 言語とほとんど同じなので、C 言語を勉強した皆さんにとっては理解しやすいでしょう。最近 Java に関して多くの参考書がでています。どれを使うか迷ってしまう程です。自分に合った参考書を 1 冊選んで見て下さい。

Java プログラムの形式は、

```
public class クラスの名前{
    public static void main(String args[]){

        実行プログラム
    }
}
```

です。Java ではこの class(クラス) という単位でプログラムを作成していきます。class の次に書かれた s1000 などがこのクラスの名前になります。

¹java.awt には様々なクラスやメソッドが標準で準備されている。

クラスはさらに メソッド という単位にわかれます。前の例では、s1002 というクラスの中には main という名前のメソッドが1つあることとなります。

クラスやメソッドの前に着いている public, static, void などは修飾子² といいます。

10.3 変数

C 言語と同じように Java でも変数をにいろいろな値を代入できます。基本データの型の変数をまとめておきます。

表 10.1: 基本データ型の変数の種類

種類	型	値	代入方法
整数	byte	-128 ~ 128 (8 ビット)	a=3
	short	-32768 ~ 32768 (16 ビット)	a=2
	int	$-2^{31} \sim 2^{31} - 1$ (32 ビット)	a=2
	long	$-2^{63} \sim 2^{63} - 1$ (64 ビット)	a=2
実数	float	約 $-10^{38} \sim 10^{38}$ (32 ビット)	a=3.0f
	double	約 $-10^{308} \sim 10^{308}$ (64 ビット)	a=2.0
論理値	boolean	true, false (1 ビット)	a=true, b=false
文字	char	文字 (16 ビット)	a='A'

文字列を表示するには、

```
System.out.println("表示したい文字列");
```

を使います。System.out.println(" ") の場合は表示後に改行しますが³、System.out.print(" ") の場合は改行しません。

次のプログラムはプリント文による表示の例を示します。

```
0:    /**s1003.java*****//
1:    public class s1003{
2:        public static void main(String args[]){
3:            int a, b, sum;
4:            a=2; b=3;
5:            sum=a+b;
6:            System.out.println("a+b="+sum); // **a+b=5 と表示される****//
7:            System.out.println(sum);          // **5 ****//
```

²public: どのクラスからもこのメソッド main を呼び出せる; static: このメソッド main はクラスの名前と結びついている; void: 返り値を持たない

³C 言語では printf() 文中の %n に対応しています。

```

8:      System.out.println("a"+"b");          // **ab ****//
9:      System.out.println("Sum"+a+b+sum);    // **Sum235 ****//
10:     }
11:     }
12:     //*****//

```

6-9: System.out.println() 関数のいろいろな使い方。

10.4 クラス・メソッド

クラスを理解するために、2つのクラスを持つプログラム例を示します。まずクラス「s1004」のメソッド main から計算を始めます。5行目でクラス「t1004」のメソッド keisan を呼び出している。

```

0:  //***s1004.java*****//
1:  public class s1004{                               /**クラス s1004 **//
2:      public static void main(String args[]){      /** メソッド main **//
3:          double a,b;
4:          a=2.0;
5:          b=t1004.keisan(a);
6:          System.out.println("a^2="+b);
7:      }
8:  }
9:  //*****//

```

5: t1004.keisan(a) は引数「a」をクラス「t1004」のメソッド「keisan」に引き渡している。C言語でたとえると、t1004.keisan(double a) という関数を定義したのがメソッド「t1004.java」である。

```

0:  //***t1004.java*****//
1:  public class t1004{                               /**クラス t1004 の宣言**//
2:      public static double keisan(double x){ /** メソッド keisan の宣言**//
3:          double y;
4:          y=Math.pow(x,2.0);
5:          return y;
6:      }
7:  }
8:  //*****//

```


2: double 型の返り値を返すメソッド keisan() の宣言。引数は double 型。
 4: クラス「Math」を用いた数値計算。始めから Java に入っている標準関数である。Math.pow(x,2.0) は x の 2 乗を計算する。

実行は以下のように s1004.java をコンパイルする。t1004.class は自動的にできる。

```
% javac s1004.java
% java s1004
```

static がついたメソッドを「クラスメソッド」、static が付かないメソッドを「インスタンスメソッド」といいます。

10.5 インスタンス

先の例では、入力した数字を 2 乗して答えを返すクラス「keisan」を作りました。今度は入力した数字の平方根を返す機能もほしくなるとします。先程と同じように新しく平方根を求めるクラスを作るという方法もありますが、クラスの数だけソースファイルが必要になりますから、その手間は大変です。このクラスの変わりに、ここではインスタンスを使ったプログラムを紹介します。

10.5.1 インスタンスの生成

この例では、s1005 と keisan という 2 つのクラスを使います。keisan クラスに沢山のインスタンスを作ることが出来ます。java では任意のクラスの型で変数を宣言できます（以下の 4 行目：オブジェクト型の変数）。

```
0:  /****s1005.java***/
1:  public class s1005{                               /****クラス s1005 の宣言***/
2:  public static void main(String args[]){          /**** メソッド main の宣言***/
3:      double b;
4:      keisan s; //クラス名 (keisan) 型のオブジェクト変数 s を宣言する //
5:
6:      s=new keisan(); //インスタンスはクラスの前に new を作用させると生成される//
7:      b=s.nijou(3.0); //インスタンス s のメソッド nijou を引数 3 をつけて呼び出す//
8:      System.out.println("a^2="+b);
9:
10:     b=s.root(3.0);
11:     System.out.println("a^{0.5}="+b);
12:
13:     }
14:     }
*****
```

4: 任意のクラス名を型とする変数をオブジェクト型の変数と呼び、宣言した型のクラスのインスタンスが代入される。

6: 引数無しでインスタンスに渡す。new により生成したインスタンスを変数 s に代入する。これにより、s を参照することで、生成したインスタンスを呼び出せるようになる。

```
*****
0:      /**keisan.java*****//
1:      public class keisan{      /**クラス keisan の宣言**//
2:          public double nijou(double x){      //インスタンスメソッド nijou の宣言//
3:              double y;
4:              y=Math.pow(x,2.0);
5:              return y;
6:          }
7:          //*****//
8:          public double root(double x){ //インスタンスメソッド root の宣言//
9:              double y;
10:             y=Math.sqrt(x);
11:             return y;
12:          }
13:          //*****//
14:      }
15://*****//
16:
```

10.6 オブジェクトの引き渡し

引数として int 型や double 型などの変数だけでなく、オブジェクト型の変数も引き渡せます。以下の例では、s1006 と plus の2つのクラスを使います。2数の和をオブジェクト変数 pa, pb を用いて渡しています。

```
0:  /**s1006.java*****//
1:      public class s1006{
2:          public static void main(String args[]){
3:              plus pa, pb;      /* plus のインスタンス (オブジェクト変数) pa, pb の宣言 */
4:              pa=new plus(2); /*2 を引数に plus のインスタンス pa を生成する*/
5:              pb=new plus(5); /*5 を引数に plus のインスタンス pb を生成する*/
6:
7:              s1006 a=new s1006(); /* s1006 のインスタンス a を生成する。引数無し*/
8:              a.hyouji(pa);      /*pa を引数にメソッド hyouji を呼び出す */
9:              a.hyouji(pb);      /*pb を引数にメソッド hyouji を呼び出す */
10:         }
```

```

11:     void hyouji(plus p){ /*plus 型の変数 p で引数 pa を受け取る*/
12:     System.out.println(p.pl(3)); //クラス plus のメソッド pl を引数 3 で呼び出す//
13:     }
14:     }
15:     ://*****//

```

8: 11 行目の p にオブジェクト pa を渡す。p=pa

```

0:     /***plus.java*****//
1:     public class plus{
2:         int y;
3:         plus(int x){ /* s1006 クラスの 4 行目 plus(2) の 2 がここに渡される。x=2 */
4:             y=x;           /*y=2 */
5:         }
6:         public int pl(int w){ //int 型の pl( ) メソッドの宣言。引数は整数型 w//
7:             int z;
8:             z=w+y;
9:             return z;
10:        }
11:    }
12:    ://*****//

```

6: s1006 クラスの 12 行目 p.pl(3) の 3 がここに渡される。w=3

10.7 継承

すでにあるクラスの内容を少しだけ変更したり、付け加えたりして、新しいクラスを作りたい場合などがあります。Java には既存のクラスを変更して新しいクラスを作る仕組みがあります。親となる既存のクラスをスーパークラス、新しく作る子供のクラスをサブクラスといいます。スーパークラスで定義した変数やメソッドはサブクラスの中で自由に使えます。

以下では3つのクラス s1007, Wa, Super1 が登場します。2数の和を求めるサブクラス Wa をスーパークラス Super1 から作る例を示します。

クラス s1007 のプログラム

```

0:     /***s1007.java*****//
1:     public class s1007{           //クラス s1007 の宣言 //
2:         public static void main(String args[]){
3:             int a,b;           //整数型変数 a,b の宣言 //
4:             a=10; b=20;
5:             Wa s;           //クラス Wa のインスタンス s を宣言 //
6:             s=new Wa(); //Wa のインスタンスを生成し s に代入 (引数は無し) //

```

```

7:
8:     s.yset(a);    //s のメソッド yset を引数 a で呼び出す。//
9:     System.out.println("a+b="+s.pl(b)); //s のメソッド yset を引数 b で//
10:    }              //呼び出し結果を出力//
11:  }
11:  ://*****//

```

8: s のメソッド yset(a) を呼び出し y=a とする。

スーパークラス Super1 のプログラム

```

0:  /***Super1.java*****//
1:  public abstract class Super1{ //abstract が付いている//
2:  public int y; //public を付けると、どのクラスからでも呼び出しが出来る//
3:  public void yset(int x){ //メソッド yset() の宣言//
4:      y=x;
5:  }
6:  }
7://*****//

```

1: クラス Super1 はサブクラスである Wa に共通機能を付けるために存在します。このように自分自身ではインスタンスを生成しないクラスを抽象クラスといい、abstract をつけて宣言します。スーパークラスは 1 つしか使用できない。

サブクラス Wa のプログラム

Wa extends Super1 の宣言により、Wa は Super1 のサブクラスとして作られたことをしめす。これにより、クラス Wa はメソッド pl() の他に、スーパークラス Super1 で定義されている変数 y と、メソッド yset() を持つようになる。従って s1007.java の 8 行目で s.yset() でクラス Wa の変数 s でクラス Super1 の中のメソッド yset() が呼び出せる。

```

0:  /***Wa.java*****//
1:  public class Wa extends Super1{ //extends が付いている//
2:  public int pl(int w){
3:      int z;
4:      z=w+y;
5:      return z;
6:  }
7:  }
8://*****//

```

1: Wa extends Super1 は「クラス Wa はクラス Super1 のサブクラスとして作られたことをしめす。

実行方法

```
% javac s1007.java (自動的に Wa.class と Super1.class が出来るはず)
% java s1007
a+b=30
```

10.8 演習問題

1. 自分の名前を表示するプログラムを作りましょう。姓名と名前を改行する場合と、改行しない場合などいろいろ作ってみましょう。
2. s1005.java と keisan.java を参考にして、四則演算を行うプログラムを作りましょう。ここでは wa, sa, seki, waru の4つのインスタンス・メソッドを作ってプログラムを書いてください。(あくまでもクラスとインスタンスの練習のためです。実際、4則演算をするプログラムはc言語で関数を使うように書くことができる、あるいは、そうすべきです。)
3. s1006.java を参考にして、以下の seki クラスを用いて、2つの整数のかけ算を行うオブジェクトクラス ensyu1003.java を作ってください。

```
public class seki{
    int y;
    seki(int x){
        y=x;
    }
    public int pl(int w){
        int z;
        z=w*y;
        return z;
    }
}
```

4. s1007.java を参考にして、2つの整数の和、差、積、割り算を計算するサブクラス Wa, Sa, Seki, Waru をスーパークラス Super1 からつくり、表示するプログラムを作りましょう。

第11章 Javaの制御文・配列・関数・その他

制御文 `if()` , `for()` , `while()` などや、配列 `a[100]`、関数 `func(int a, int b)` など、“ほとんど”C言語と同じように使用できます。ただし、宣言のやりかたなどに多少の違いはあります。ここでは、次の章のプログラムを参考に使い方を覚えることにしておきます。詳しいことはこの授業では省略することにします。

余白スペースがあるので、キーボードからデータを入れる方法を書いておきます。

```
//キーボードからデータを入れる
import java.io.*;
public class keybord1{
    public static void main(String[] args) {
        try{
            int a,b,c;
            //InputStreamReader クラスのインスタンスを生成する
            InputStreamReader reader=new InputStreamReader(System.in);
            //BufferedReader クラスのインスタンスを生成する
            BufferedReader bufreader = new BufferedReader(reader);
            //キー入力
            System.out.print("number1 =");
            String s1=bufreader.readLine();
            a=Integer.parseInt(s1);
            System.out.print("number2 =");
            String s2=bufreader.readLine();
            b=Integer.parseInt(s2);

            c=a+b;
            System.out.println("a+b="+c);

        }
        catch(Exception e){
            System.out.println(e+"Error");
        }
    }
}
```

```
おまけ ;

//データをファイル (data1.txt) にセーブする
import java.io.*;
public class copy{
    public static void main(String[] args) {
        try{

            int c;
            double b;
            PrintWriter fw =new PrintWriter( new FileWriter("data1.txt"));

            for(c=1; c<100; c++){
                b=Math.pow(c,0.5);
                fw.println(c+ "\t"+b+"\t"+c+ "\t"+b);
            }

            fw.close();
        }
        catch(Exception e){
            System.out.println(e+"Error");
        }

    }
}
```


第12章 アプレット・グラフィックス

Java の最大の魅力はホームページ上でプログラムを実行できたり、アニメーションやグラフィックをわりと簡単に出来ることです。しかもホームページ上で!! これを実現してくれるのがアプレットという形式です。まず次のプログラムを実行しましょう

```
0:  /**s1101.java*****//
1:  /*<APPLET CODE="s1101.class" WIDTH=300 HEIGHT=300>
2:  </APPLET> */
3:
4:  import java.applet.*; //図形を表示するために 4,5 行目が必要//
5:  import java.awt.Graphics;
6:
7:      public class s1101 extends Applet{//アプレットクラス s1101 の宣言//
8:      public void paint(Graphics g){ //メソッド paint の宣言 //
9:          g.drawString("Hello", 100,100);
10:     }
11:     }
12://*****//
```

1,2: appletviewer コマンドでアプレットを読み込むための宣言

7: クラス s1101 はスーパークラス Applet のサブクラスである。クラス Applet には様々なメソッド¹が java 言語に標準で用意されている。

8: 画面 (Graphics) の名前として g を宣言した。

9: 図を表示するメソッド。drawString は” ”で挟まれた文字を書くメソッド。

実行方法

```
% javac s1101.java
```

```
% appletviewer s1101.java
```

図 12.1 のような画面が現れる。

ホームページ上で図 12.1 を表示するには以下のような HTML 文書を書いて、例えば s1101.html という名前で保存しておく。その後、WWW ブラウザを立ち上げ、s1101.html を読み込む。

```
0:  <!--s1101.html--> HTML のコメント文
1:  <HEAD>
```

¹直線を引く drawLine(), 長方形を描く drawRect() など。

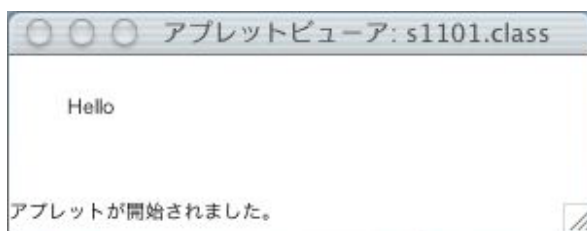


図 12.1: s1101 の実行結果。Hello と書いた画面が現れる。Mac OSX の場合

```

2:     </HEAD>
3:     <BODY>
4:         <APPLET CODE="s1101.class" WIDTH=300 HEIGHT=100>
5:     </APPLET>
6: </BODY>
7: <!------->

```

4: s1101.class を幅 300 ピクセル高さ 100 ピクセルで呼び込む²。

以下のプログラムは、% appletviewer コマンドを用いて実行しましょう。

12.1 画面の基本

図 12.2 の様に、幅 $w=60$ ピクセル、高さ $h=30$ ピクセルの長方形を、左端が $x=20$ ピクセル、上端が $y=30$ ピクセルの位置にくるように表示するプログラムは以下ようになります。

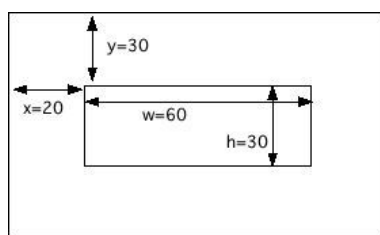


図 12.2: 表示領域の高さは90ピクセル、幅100ピクセル。

```

0:  //***s1102.java***//
1:  /*<APPLET CODE="s1102.class" WIDTH=100 HEIGHT=90>
2:      </APPLET> */
3:      import java.applet.*;
4:      import java.awt.Graphics;

```

²ピクセルはパソコンの画面の単位。現在のパソコンの画面の大きさはおそらく、1024ピクセル(幅)×768ピクセル(高さ)である。アプレットの表示サイズはそれよりも小さくしておくのが安全です。

```
5:
6:     public class s1102 extends Applet{
7:         public void paint (Graphics g){
8:             g.drawRect(20,30,60,30);
9:         }
10:    }
11:    //*****//
```

1: アプレットの表示サイズは大きめに指定するのが安全。

8: `g.drawRect(x,y,w,h)`; 長方形を描くメソッド。引数は `x,y,w,h` の順。図 12.2 を参照。

アプレットを表示するための決まりは

```
/*<APPLET CODE="プログラムの名前.class" WIDTH=300 HEIGHT=300>
</APPLET> */
import java.applet.*;
import java.awt.Graphics;

public class プログラムの名前 extends Applet{
    public void paint (Graphics g){

        ここにプログラムを書く

    }
}
```

12.2 正方形・長方形

正方形や長方形を表示するには

`g.drawRect(左端座標 x, 上端座標 y, 幅 w, 高さ h)`

を使います。1 辺が 1 0 0 ピクセルの正方形と、幅が 1 0 0 ピクセルで高さ 8 0 ピクセルの長方形を描くプログラムを作りましょう。

```
0:    //*****s1103.java***** //
1:    /*<APPLET CODE="s1103.class" WIDTH=300 HEIGHT=300>
2:        </APPLET> */
3:    import java.applet.*;
4:    import java.awt.Graphics;
5:
```

```

6:     public class s1103 extends Applet{
7:         public void paint (Graphics g){
8:             g.drawRect(50,50,100,100);
9:             g.drawRect(200,50,80,100);
10:        }
11:    }
12:    //*****//

```

8: 一辺が 100 ピクセルの正方形は `g.drawRect(左端 x, 上端 y,100,100);` で描けます。

9: 幅が 100 ピクセルで高さ 80 ピクセルの長方形は `g.drawRect(左端 x, 上端 y,80,100);` で描けます。

あとは 2 つの図が重ならないように `x,y` の値を考えればよい。

12.3 円・楕円・直線

円を表示するには

```
g.drawOval(左端座標 x, 上端座標 y, 幅 w, 高さ h)
```

を使います。図 12.3 の様に点 (x_0, y_0) を中心とする半径 r の円を描く場合、左端座標は $x = x_0 - r$ 、上端座標は $y = y_0 - r$ 、幅は $w = 2r$ 、高さ $h = 2r$ となります。この計算式を関数を使って汎用にしておけば、便利です。

次の例は、点 $(100, 100)$ を中心とする半径 70 ピクセルの円を描くプログラムです。`circle1()` 関数を定義して、中心と半径を与えています。

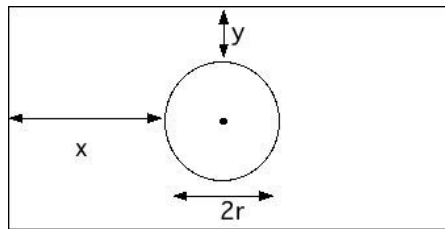


図 12.3: 点 (x_0, y_0) を中心とする半径 r の円を描く

```

0:     //s1104.java //
1:     /*<APPLET CODE="s1104.class" WIDTH=300 HEIGHT=300>
2:         </APPLET> */
3:     import java.applet.*;
4:     import java.awt.Graphics;
5:

```

```

6:     public class s1104 extends Applet{
7:     public void paint (Graphics g){
8:         circle1(g,100,100,70);
9:     }
10:    //中心と半径を与えた時に円を描く関数
11:    static void circle1(Graphics g, int x0, int y0, int r){
12:        int xa, ya, w, h;
13:        xa=x0-r;
14:        ya=y0-r;
15:        w=2*r;
16:        h=2*r;
17:        g.drawOval(xa,ya,w,h);
18:    }
19:
20:    }

```

直線を引く命令は

`g.drawLine(始点の横座標 x0, 始点の縦座標 y0, 終点の横座標 x1, 終点の縦座標 y1)`

です。次の例は 300 × 300 ピクセルの画面上に x 軸と y 軸を引くプログラムを示します。画面の左上の画面座標は (0 , 0) , 右下の画面座標が (3 0 0 , 3 0 0) に対応している。

```

0:    //s1105.java //
1:    /*<APPLET CODE="s1105.class" WIDTH=300 HEIGHT=300>
2:    </APPLET> */
3:    import java.applet.*;
4:    import java.awt.Graphics;
5:
6:    public class s1105 extends Applet{
7:    public void paint (Graphics g){
8:        g.drawLine(0,150,300,150);
9:        g.drawLine(150,0,150,300);
10:    }
11:    }

```

12.4 多角形と折れ線グラフ

多角形の頂点 $P_0, P_1, P_2, \dots, P_{n-1}$ の座標が $(x_0, y_0), (x_1, y_0), (x_2, y_2), \dots, (x_{n-1}, y_{n-1})$ の時, x 軸の座標の配列 $x = (x_0, x_1, x_2, \dots, x_{n-1})$ と y 軸の座標の配列 $y = (y_0, y_1, y_2, \dots, y_{n-1})$ を用いて, 多角形を描くことができます。

`g.drawPolygon(x 軸の座標の配列 x, y 軸の座標の配列 y, 点の数)`

これは、点 $P_0, P_1, P_2, \dots, P_{n-1}$ を順に直線で結んで、最後に点 P_0 と P_{n-1} を結びます。点の数は頂点の数を示します。点の数を多くすれば滑らかな曲線になるはずですが。

`g.drawPolyline(x 軸の座標の配列 x, y 軸の座標の配列 y, 点の数)`

これは、点 $P_0, P_1, P_2, \dots, P_{n-1}$ を順に直線で結びます。

次の例は、正多角形を描くプログラムです。頂点の個数 n , 半径 r , 中心の座標 (x_0, y_0) を与えた時に、正 n 角形を描く関数

`polygon1(Graphics g, int n, int r, int x0, int y0)`

を関数で作ってあります。

```

0: //s1106.java //
1: /*<APPLET CODE="s1106.class" WIDTH=500 HEIGHT=500>
2:    </APPLET> */
3:    import java.applet.*;
4:    import java.awt.Graphics;
5:
6:    public class s1106 extends Applet{
7:    public void paint (Graphics g){
8:        polygon1(g,3,50,100,100);
9:        polygon1(g,4,50,250,100);
10:       polygon1(g,5,50,400,100);
11:       polygon1(g,6,50,100,250);
12:       polygon1(g,7,50,250,250);
13:       polygon1(g,8,50,400,250);
14:    }
15:    //*****
16:    //頂点の個数 n, 半径 r, 中心の座標 (x0,y0) を与えた時に、正 n 角形を描く関数
17:    //*****
18:    static void polygon1(Graphics g, int n, int r, int x0, int y0){
19:        int[] x=new int[121];    //配列 x の定義。要素数を 1 2 1。
20:        int[] y=new int[121];    //配列 y の定義
21:        int i;
22:        double dt=2.0*Math.PI/(double)n;
23:        for(i=0; i<n; i++){
24:            x[i]=x0+(int)(r*Math.sin((double)i*dt)); //頂点の x 座標を計算して配列
25:            y[i]=y0-(int)(r*Math.cos((double)i*dt)); //頂点の y 座標を計算して配列

```

```

26:     }
27:     g.drawPolygon(x,y,n); //正 n 角形を描く関数
28:     }
29: }

```

12.5 関数を描く

関数 $f(x) = x^2$ のグラフを x の値を -1 から 1 までで、描くプログラムを示します。例では 500×500 ピクセルの画面上の中心 $(250, 250)$ を関数座標の原点 $(0,0)$ に、画面上の右下の座標 $(500, 500)$ を関数座標の $(1,-1)$ に対応させています。

```

0: //s1107.java //
1: /*<APPLET CODE="s1107.class" WIDTH=500 HEIGHT=500>
2:     </APPLET> */
3:     import java.applet.*;
4:     import java.awt.Graphics;
5:
6:     public class s1107 extends Applet{
7:     public void paint (Graphics g){
8:         int[] xx=new int[501]; //x 座標の配列の宣言
9:         int[] yy=new int[501]; //y 座標の配列の宣言
10:        double x, y; //関数の変数 x とその計算値 y : 関数座標 (x,y) の宣言
11:        int i, j; //画面上の座標 (i,j) の宣言
12:
13:        for(i=0;i<=500;i++){ //501 点の x 座標点を用意
14:            x=(double)(i)/250.0-1.0; //x の値
15:            y=Math.pow(x,2.0); //x の 2 乗の計算
16:            j=250-(int)(250.0*y); // 画面座標に変換
17:            xx[i]=i; //
18:            yy[i]=j;
19:        }
20:        g.drawPolyline(xx,yy,501); //曲線を描く
21:        g.drawLine(0,250,500,250); // x 軸
22:        g.drawLine(250,0,250,500); //y 軸
23:    }
24: }

```

1: 画面のサイズは 1 行目で行っている。

14: $x = ai + b$ で $i = 250$ の時 $x = 0$, $i = 500$ の時 $x = 1$ を満たすように定数 a, b を決める。

15: java 言語での数学関数は、c 言語の標準数学関数に Math. を付けて呼び出せばよい。

16: $y = cj + d$ で $j = 250$ の時 $y = 0$, $j = 500$ の時 $y = -1$ を満たすように定数 c, d を決める。
描きたい画面の大きさや関数の値によって、定数 a, b, c, d を調節する必要がある。画面座標と関数座標を変換する関数を作っておくとよい。

12.6 アニメーション

Java では画像データを読み込んだり、それを動画（アニメーション）にしたり、ボタンを付けたり、様々な関数（メソッド）が用意されています。

ここでは、その中の一つであるスライドショーについて学びましょう。スライドショーとは、何枚かの画像を一定時間間隔で切り替えて順に見せていく方法です。「一定時間休む」ために Java では `sleep()` という関数があり、スレッド (thread) という方式で処理します。

- スレッドを使う時はアプレットの最初を次のように書きます。

```
public class 名前 extends Applet implements Runnable{

    Thread th;
    public void start(){
        th=new Thread(this);
        th.start();
    }
}
```

- 処理の流れは

```
public void run(){

    }
```

の中に書きます,

- 「一定時間休む」は

```
try{
    Thread.sleep(10);
}
catch(InterruptedException e){
}
```

と書きます。

- 休止時間はミリセカンド (1/1000 秒) 単位の整数値で指定します。

例) 1 秒休む `Thread.sleep(1000);`

5 秒休む `Thread.sleep(5000);`

0.1 秒休む `Thread.sleep(100);`

ここでは、2つの円を左右から動かすプログラムを考えましょう。まず(1)円を描く。(2)それを消して少し右に動かし円を描く(3)それを消して、さらに少し右に動かし円を描く。これを短い時間間隔で繰り返していけば、円が右に動いているように見えるはずです。

```
0: //s1108.java //
1: //2つの円を左右から走らせる
2: /*<APPLET CODE="s1108.class" WIDTH=350 HEIGHT=300>
3: </APPLET> */
4: import java.applet.*;
5: import java.awt.Graphics;
6: import java.awt.Color; //色指定の関数を呼び出す
7:
8: public class s1108 extends Applet implements Runnable{
9:     int x,x1;
10:    Thread th;
11:
12:    public void start(){
13:        th=new Thread(this);
14:        th.start();
15:    }
16:
17:    public void run(){
18:        while(true){
19:            for(x=0; x<280; x++){ //赤玉を右へ動かす
20:                repaint();
21:                try{
22:                    Thread.sleep(10);
23:                }
24:                catch(InterruptedException e){
25:                }
26:            }
27:            for(x=280; x>0; x--){ //赤玉を左へ動かす
28:                repaint();
29:                try{
30:                    Thread.sleep(10);
```

```
31:         }
32:         catch(InterruptedException e){
33:     }
34:     }
35:     } /** while()***/
36: }
37:     public void paint(Graphics g){
38:         x1=280-x;
39:         g.drawLine(0,150,300,150);
40:         g.setColor(Color.red); //赤を指定
41:         g.fillOval(x,130,20,20); //円を上の色で塗る
42:         g.setColor(Color.blue); //青を指定
43:         g.fillOval(x1,130,20,20); //円を上の色で塗る
44:     }
45: }
```

12.7 演習問題

1. 上のプログラムのうちから 2 つ選んで、それぞれの HTML 文書を書いて、WWW ブラウザからプログラムを実行してみましょう。
2. 小さい正方形を 3 行 3 列に規則正しく並べて表示してください。
3. 10 個の円を水平に規則正しく並べて表示してください。
4. s1106.java を参考にして、drawPolyline() を使って多角形を描いてください。
5. s1107.java を参考にして、 $f(x) = x^3$ のグラフを x の範囲を -2 から 2 までで描いてください。
6. s1108.java を参考にして、いろいろな速度で 2 つの円を動かしてみよう。また 2 つの円が衝突したら跳ね返るようにしてみましょう。
7. s1108.java を参考にして、テーマは何でもいいので、アニメーションを作ってください。WWW ブラウザからプログラムを実行してみましょう。

第13章 高分子のランダムウォーク

高分子は n 個のモノマー（セグメントとよぶ）がつながったグニャグニャした分子です。タンパク質やDNAなどの最も簡単なモデルでもあります。熱運動によって各セグメントが、でたらめ（ランダム）に動く様子をアニメーションで実現しましょう。ただし各セグメントはつながっているという制限がついています。

13.1 正方形の壁で囲まれた2次元領域をランダムに動く高分子

次の例は、正方形の壁で囲まれた2次元領域をランダムに動く高分子（ n 個のセグメントがつながった分子）のランダムウォークを示すプログラム例です。ただし、壁を通り抜けてはいけないという条件がついています。

プログラムの構成

- (1) まず高分子を平面上に置きます（初期設定）… `polymer_init()` 関数
- (2) 乱数を引いて n 個のセグメントの内の p 番目のセグメントを選び動かす。動かし方は、ボンド長を一定に保って回転させる。この回転させた座標を p 番目のセグメントの新座標 (x_{new} , y_{new}) とする。… `polymer()` 関数
- (3) 新座標が壁を越えたら新座標を元の座標に戻す。
- (4) (2)(3)の操作を n 回繰り返したものを1ステップとする。
- (5) 新座標を描画する。… `paint()` 関数
- (6) (2)-(5)を `while()` 文で無限に繰り返す。

```
//rwalk2.java //
//反射壁の場合での高分子のランダムウォーク
/*<APPLET CODE="rwalk2.class" WIDTH=550 HEIGHT=550>
</APPLET> */
import java.applet.*;
import java.awt.Graphics;
import java.awt.Color;

//*****
public class rwalk2 extends Applet implements Runnable{
double PI=3.141592;
double b=11.0; //ボンドの長さ
int r=5;          //球の半径
```

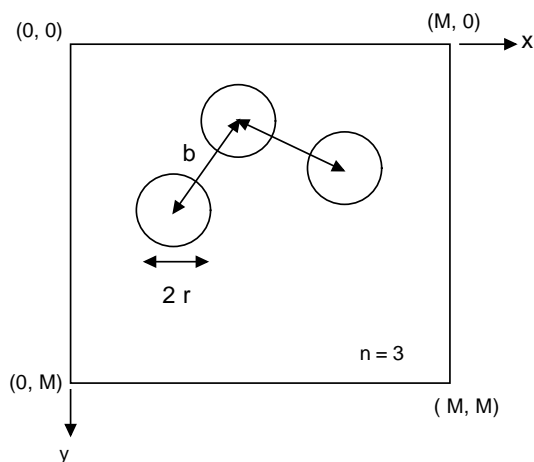


図 13.1: 高分子を配置する

```

int n=3;      //セグメント数
int M =500;   //正方形の一辺(ピクセル)
int[] x= new int[n+3]; //配列 x[0], x[1],,,
int[] y=new int [n+3]; //配列 y[0], y[1],,,
int i,p, t, xp, yp, xp_new, yp_new;
double cos_theta,sin_theta, the;
//*****
    Thread th;
    public void start(){
        th=new Thread(this);
        th.start();
    }
//*****
public void run(){
    x[1]=200; //1 番目のセグメントの初期座標
    y[1]=200;
    polymer_init(); //高分子の初期座標設定
    while(true){ //1 ステップは while の終わりまで。
        polymer(); //各セグメントを動かす
        //*****描画する部分*****
        repaint(); //***paint() 関数を呼ぶ*****
        try{ Thread.sleep(50); }
        catch(InterruptedException e){ }
        //*****
    } //while() の終わり

```

```

} //run() 関数の終わり
//*****
//*****高分子を描く*****
//*****
public void paint(Graphics g){
    g.setColor(Color.blue); //青色の指定
    g.drawRect((int)b,(int)b,M, M);
    g.setColor(Color.red); //赤色の指定
    g.fillOval(x[n]-r, y[n]-r, 2*r, 2*r);
    for(i=1;i<n; i++){
        g.fillOval(x[i]-r, y[i]-r, 2*r, 2*r); //円を塗る
        g.drawLine(x[i], y[i], x[i+1], y[i+1]); //線を引く
    }
}
//*****
//**** public void polymer() ****
//*****
public void polymer(){

    for(i=1; i<=n; i++){
//乱数を引いてp番目のセグメントを動かす
        p=(int)(((double)n+1.0)*Math.random());

        if(p==0){ //ゴミ
            xp_new=0;
            yp_new=0;
        }
        else if(p==1){
            theta=2.0*PI*Math.random();
            xp_new=x[2]+(int)(b*Math.cos(theta));
            yp_new=y[2]+(int)(b*Math.sin(theta));
        }
        else if(p==n){
            theta=2.0*PI*Math.random();
            xp_new=x[n-1]+(int)(b*Math.cos(theta));
            yp_new=y[n-1]+(int)(b*Math.sin(theta));
        }
        else {
            cos_theta=(x[p+1]-x[p])/b;
            sin_theta=(y[p+1]-y[p])/b;

```

```

        xp_new=x[p-1]+(int)(b*cos_theta);
        yp_new=y[p-1]+(int)(b*sin_theta);
    }

    if(xp_new<(int)b || xp_new>M-(int)b ||
       yp_new<(int)b || yp_new>M-(int)b ){
        xp_new=x[p];
        yp_new=y[p];
    }

    x[p]=xp_new;
    y[p]=yp_new;
}
}
//*****
//**** public void polymer_init() **初期座標
//*****
public void polymer_init(){
    for(i=1;i<=n; i++){
        theta=2.0*PI*Math.random();
        x[i+1]=x[i]+(int)(b*Math.cos(theta));
        y[i+1]=y[i]+(int)(b*Math.sin(theta));

        while(true){
            if( x[i+1]<(int)b || x[i+1]>M-(int)b ||
               y[i+1]<(int)b || y[i+1]>M-(int)b ){

                theta=2.0*PI*Math.random();
                x[i+1]=x[i]+(int)(b*Math.cos(theta));
                y[i+1]=y[i]+(int)(b*Math.sin(theta));
            }
            else{break; }
        }

    }
}
//*****
} //end of class

```

```
/** end of program *****/
```

13.2 演習問題

1. 上のプログラムの HTML 文書を書いて, WWW ブラウザからプログラムを実行してみましょう。
2. セグメント数を変えてプログラムを実行してみましょう。たとえば, $n = 200$.
3. 両端を黒丸で表示するようにしてください。
4. 壁を取り外したプログラムを書いて実行してください。
5. 応用問題 (興味があれば, 来年私の研究室でその後をやりませんか?)

- (a) 各ステップ (s) 毎の, 高分子の両端の距離の 2 乗 (R_s^2) を求めて, 1000 ステップ毎に, ステップ数と, R_s^2 の値をファイル (data1.txt) にセーブしてください。 $s = s_{max}$ ステップで実験を終了して平均値

$$\langle R^2 \rangle = \frac{1}{s} \sum_{s=1}^{s_{max}} R_s^2$$

を求めてください。 $s_{max} = 10000$ でまずはやってみましょう。高分子のセグメント数 n を大きくしていくと, $\langle R^2 \rangle = b^2 n$ の関係式が成り立つはずであるが, 確かめてみてください。

- (b) 周期的境界条件を取り入れたプログラムにしてください。
- (c) 各セグメントが重ならないように, プログラムを修正してください。