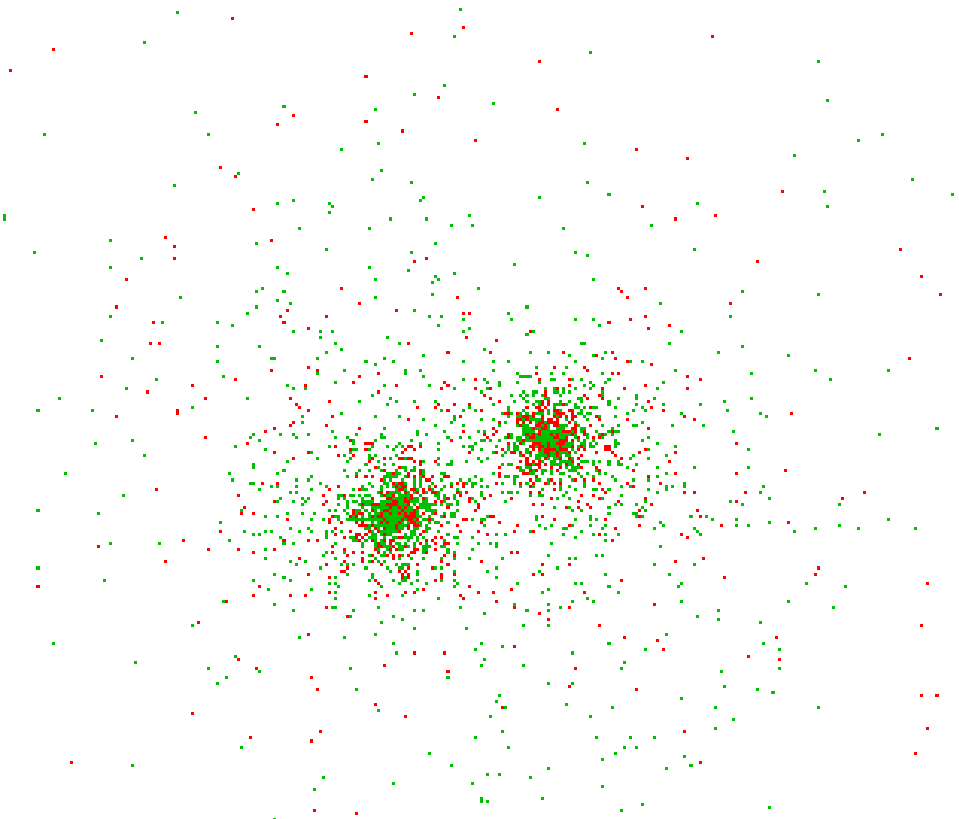


プログラミング言語基礎

吉田　宏
E-mail:yoshidah@fmu.ac.jp
福島県立医科大学　物理学講座

目次

第 0 章

プログラミングを始める前に

0.1　Cygwin の起動

Cygwin とは，Windows の上で UNIX と同じような環境でコマンド (コンピュータに対する命令を「コマンド」という) 等を使うことのできるアプリケーションである。

Windows のデスクトップ上に 

のようなアイコンがあるので，これをダブルクリックする。すると図 0.1 のような window が開き Cygwin の shell が起動する。以下では，図 0.1 の window のことを簡単に “shell” と呼ぶことにする。

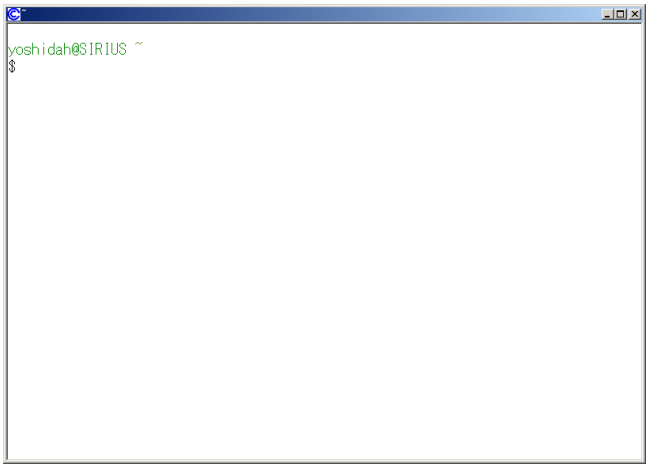


図 0.1: Cygwin の shell

ここで，“$” とあるのが，Cygwin でのプロンプトで，この表示のあとにさまざまなコマンドを入力することができる。また，プロンプトの上の “yoshidah@SIRIUS ~” は “yoshidah” というログイン名の人が “SIRIUS” というパソコンの “~/” というディレクトリで現在作業しようとしている，ことを意味している。実際には “XXXXX@IPCYYYYY ~” のように表示されていることだろう (XXXXX はあなたのログイン名，IPCYYYYY はあなたが現在使っているパソコンの名前)。

0.2　Cygwin/Windows の基礎知識

0.2.1　ディレクトリ（フォルダ）

ファイルをしまっておいたり作業をする場所を，Cygwin では「ディレクトリ」，Windows では「フォルダ」などという。図 0.2 は，Windows の「Explorer」で見たときのフォルダである。

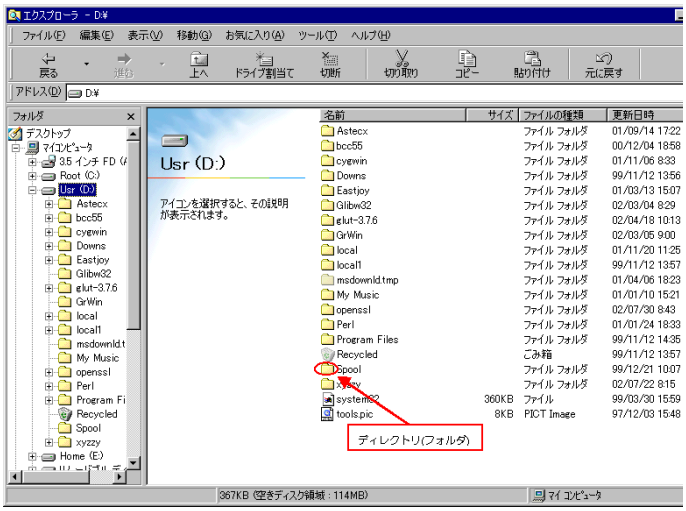


図 0.2: Explorer でみたディレクトリ(フォルダ)

0.2.2　カレントディレクトリ

shell 上で次のように入力してみよう。

```
pwd↵
```

「/home/edu/03/pc/XXXXX」と表示されたろうか？ (XXXXX はあなたのユーザ名（ログイン名）である)。これが，現時点でのカレントディレクトリである。カレントディレクトリとは，「現在作業しているディレクトリ」のことである。「今自分はどこのディレクトリで作業しているのだろう」とふと思ったときに shell 上で「pwd」とすると，カレントディレクトリが表示される。

0.2.3 ホームディレクトリ

自分で作成したファイル (プログラムや文書など) やフォルダ等を保存することのできるディレクトリを「ホームディレクトリ」という。福大の情報センターのコンピュータの設定は，shell 上では”~/”(UNIX 用)，Windows 上では”J:¥”(Windows 用) が各自のホームディレクトリになっている (実際には同じディレクトリである)。

0.2.4 パス（PATH）

shell からある命令を実行するとき，shell はその命令がどこに保存されている命令なのかを探しにいく。命令を探しにいく道筋を「パス (PATH)」と呼び，shell の中で「PATH」という環境変数に登録されている。Cygwin では PATH は各実行形式の命令の保存されているディレクトリを「:」で 区切った形で登録されている。shell 上で

```
set | grep PATH↵
```

と入力してみると, 現在の PATH が表示される。Cygwin がどのディレクトリをたどって命令を探しているかがわかるはずである。

0.3 「メモ帳」を使ってファイルを作成する

プログラムを作成するために，ファイルを編集するアプリケーションを起動する必要がある。このアプリケーションを「エディタ」といい，Windows では「メモ帳」がもっとも簡単に利用できるエディタである。

「メモ帳」は Cygwin では shell 上で “notepad” と入力すると起動できる。新たに “ex0001.c” というファイルを作るときは shell 上で

```
notepad ex0001.c &↵
```

とするだけでよい。一般に shell 上で notepad を使ってファイルを作成するときは「notepad XXXXX &」(XXXXX はファイル名) とする。

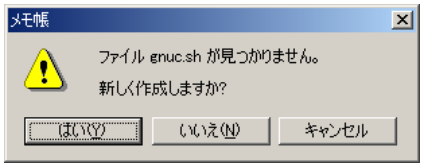


図 0.3 メモ帳からの警告

新たにファイルを作成するときは図 0.3 のような窓が開くが，気にせず「はい (Y)」をクリックすれば先に進める。

「メモ帳」で作成したファイルを保存するときは，「メモ帳」のメニューバーのところの「ファイル (F)」をクリックし，「上書き保存 (S)」をクリックすればよい。また，ファイルを保存して「メモ帳」を終了するときは，メニューバーの「ファイル (F)」をクリックした後，「メモ帳の終了 (X)」をクリックすればよい。

それでは，早速次のようなファイルを作ってみよう。なお，左端の “1.” などの番号およびピリオドは便宜的につけた行番号なのでこれは入力しなくてよい。

List 0.1 ex0001.c

```
1. /* ex0001.c 初めてのプログラム */
2. /* 作成者 XXXXXXX */
3. /* 作成日 2003/04/16 */
4.
5. int main()
6. {
7.   return 0;
8. }
```

日本語以外は全て半角英数字で入力すること。「Alt」キーを押しながら「半角／全角」のキーを押すと日本語の入力ができる。また，「半角英数字」を入力するときは，再び「Alt」キーを押しながら「半角／全角」のキーを押せばよい。

0.4　Cygwin の基本的なコマンド

Cygwin にはファイルをコピーしたり，新たにディレクトリを作成したりするコマンドがある。以下に基本的なコマンドの表を記しておく。

表 0.1 良く使うコマンド

コマンド		意味
shell 内	Windows	
ls	dir /w	現在のディレクトリにあるファイルの名前を表示する
ls -l	dir	現在のディレクトリにあるファイルの名前を表示する
pwd	cd	カレントディレクトリを表示する
cd	cd ¥	ホームディレクトリに移動する
cd sss	cd　sss	現在のディレクトリを別のディレクトリ sss に移動する
cat xxxx	type xxxx	xxxx の内容を表示する
mkdir yyyy	md yyyy	yyyy という名前のディレクトリを作る
rmdir zzzz	rd zzzz	zzzz というディレクトリを削除する
cp aaaa bbbb	copy aaaa bbbb	ファイル aaaa を bbbb という名前でコピーする
mv qqqq pppp	ren qqqq pppp	ファイル qqqq を pppp という名前に変える
rm aaa	del aaa	ファイル aaa を消去する
clear	cls	画面のクリア（消去）

課題 0.1: Cygwin を新たに起動して，その shell の中で次のコマンド (命令) を実行してみよう！何が起こるか，ノートにメモしておくこと。

1. pwd ↵
2. ls -l↵
3. ls↵
4. cat ex0001.c↵
5. mkdir taco↵
6. ls ↵
7. cp ex0001.c aa↵
8. ls↵
9. cat aa↵
10. rm aa↵
11. ls↵
12. rmdir taco↵
13. ls↵
14. mkdir C↵
15. mkdir C/Lec01↵
16. ls↵
17. cd C/Lec01↵
18. ls↵

0.5　今後, プログラミングをする際に

今後プログラミングをする際に，次のことを行うことになる。

0.5.1　「メモ帳」でプログラム (ソースコード) を記述する。

たとえば，ex0101.c というプログラムを新規に作成するときは，上で述べたように，shell 上で

notepad ex0101.c &↵

とする。また，すでにあるプログラムを修正するときも上と同様にすればよい。ここで，作成するソースコードは，必ずプログラム名のあとに「.c」をつける (プログラム名が「ex001」なら，ソースコードの名前は「ex001.c」である)。この「.c」は C のソースコードであることを示す「ファイルの拡張子」と呼ばれるものである。この他にも「.exe」や「.txt」のような拡張子がある。

0.5.2 作成したプログラムを gcc でコンパイルする。

ソースコードのままでは，コンピュータはプログラムを理解することができず実行することができない。コンピュータが理解できるの「0」と「1」からなる「機械語」と呼ばれるもので，これは，私たちには理解できない世界である。したがって，われわれの世界の言葉とコンピュータの世界の言葉との橋渡しをするものが必要になる。これが「コンパイラ」と呼ばれるものである。Cygwin で使える C 言語のコンパイラには「gcc」というものがある。

この授業の中では作成したソースコードを gcc でコンピュータににわかるような形式に変換して（これを「コンパイル」という），実行形式のファイルを作成する。たとえば，ex0001.c というソースコードに対して，shell 上で

```
gcc -o ex0001 ex0001.c↵
```

とする。ここで，gcc のあとの “`-o ex0001`” のところで，プログラム名を ex0001 と指定し、最後の “`ex0001.c`” でコンパイルするソースコードの名前を指定する。もし，プログラム (ex0001.c) の中で，数学関数を使っている時は，

```
gcc -o ex0001 ex0001.c -lm↵
```

のように，最後に”`-lm`” をつける。

このように入力したあと，何もメッセージが現れなかったら，コンパイルに成功したことを示す。何かメッセージが出てきたら，その内容からソースコードのエラーを探し，修正する。

0.6 C 言語

0.6.1 C 言語の特徴

- C 言語の命令はハードウェアに近い
- プログラムが関数を基本とした集合体
- 多様なデータ型を備えている
- 変数にローカル変数・グローバル変数・静的変数などがある
- 構造化プログラミングができる
- 構造体が使える
- 動的に配列がとれる
- ポインターが使える

0.6.2 プログラムの構成

- プリプロセッサ
  - ❋ #include 関数や定数の定義してあるファイルをメモリ上に置く
  - ❁ #define 定数や簡単な関数を定義する
  - ✳ #if #else #endif など コンパイル時の条件判断
- 関数
  - ❋ main 関数 (必ず必要，かつ１つだけ)
  - ❁ その他の関数
- 宣言
  - ❋ 関数内で使用する変数は，予め各データ型に分けて全て宣言しておく

int	float	double	char
整数型	単精度実数型	倍精度実数型	文字型

- 制御文

  繰り返し:　while, for , do ▪ while

  条件分岐:　if ▪ else , switch ▪ case

  ジャンプ:　continue, break, goto

- 代入文
  - ❋ a=210;
  - ❄ b=1e5;
  - ❋ c=func(a,b);
- コメント文　　/* コメント */

0.6.3　C 言語での約束事

- 自由形式

List 0.2 ex0002.c

```
 1. #include <stdio.h>
 2. int main()
 3. {
 4.     int a;
 5.     int b;
 6.
 7.     a=10;
 8.     b=2;
 9.     printf( “%d %d ¥n “,a/b,b/a);
10. }
```

List 0.3 ex0003.c

```
1. #include <stdio.h>
2. int main(){int a;int b;a=10;b=2;printf( “%d %d ¥n “,a/b,b/a);}
```

- コメント（注釈）は /* と */ でくくる

List 0.4 ex0004.c

```
 1. /* ex0003.c ２つの整数の割り算 */
 2. /* プロ基礎　コメントの例 */
 3.
 4. #include <stdio.h>
 5. int main()
 6. {
 7.     int a; /* 整数 a */
 8.     int b; /* 整数 b */
 9.
10.     a=10;b=2;/* 値の代入 */
11.     printf( “%d %d¥n “,a/b,b/a);   /* 結果の出力*/
12.     return 0;
13. }
```

- 変数名・関数名には英文字か _ で始め 31 文字とし，予約語は使わない
  - ☆ 良い例
    - ❃ wa,sa,param,solve,move_particle, gravity, thermal
    - ❄ _body, _sample, mean,__average, sigma8, lambda0
    - ❋ Omega,_OMEGA,_OMEGA_,alpha,beta_, ····
  - ✶ 悪い例
    - ❃ mean-density, 平均値
    - ❄ 0wa,007,%solve,$move_particle, gravity$
  - ✲ 予約語

```
auto,break,case,char,const,continue,default,do,double,else,enum,extern
float,for,goto,ifint,long,register,return,short,signed,sizeof static
struct,switch,typedef,union,unsigned,volatile,void,while
```

- 関数は { で始まり，} で終わる

List 0.5 関数と文

```
 1. /* ex0005.c 関数の例 プロ基礎*/
 2. #include <stdio.h>
 3. int function(int a,int b)
 4. {
 5.     int wa;
 6.     wa=a+b;
 7.     return wa;
 8. }
 9.
10. int main()
11. {
12.     int a=10,b=3;
13.     printf("%d %d¥n",function(a,b));  /* 結果の出力*/
14.     return 0;
15. }
```

- 文はセミコロン ; で区切る
- 1 つの関数は概ね 50〜60 行までとする

第 1 章

簡単なプログラムを作ろう！

とにかく何か出力するプログラムを作ってみる。ここでは、ほとんどの C 言語の解説書の冒頭にあるプログラムを紹介する。

「Hello, World!」と出力するプログラム ex0101.c を作る。

List 1.1 「Hello，World!」と出力するプログラム ex0101.c

```
 1. /* ex0101.c "Hello, World"を出力するプログラム */
 2. /* 作成日   2002/4/23 */
 3. /* 作成者   xxxxx */
 4. #include <stdio.h>
 5.
 6. /* 開始  */
 7. int main()
 8. {
 9.   printf("Hello, World!¥n");
10.   return 0;
11. }
12. /* 終了  */
```

≪プログラムの解説≫

1〜3 行: いつ誰がどのようなプログラムを作ったのかをコメント行としてメモしておく。必要ではないが，このようなメモをプログラムの中に記述しておく習慣をつけておくと良い。

4 行: ヘッダーファイル stdio.h を挿入する。 stdio.h には標準的な入出力関係の関数が定義されている。最初のうちは，「おまじない」だと思っていても差し支えない。必ずプログラムの始めの方に記述する。

6 行: プログラムの開始をコメントとして記述する。いつも必要というわけでもない。

7 行: プログラムは必ず main 関数で始まる。

8 行: main 関数を { で始める。

9 行: printf 関数を使って，"Hello，World!"という文字列を出力する。

10 行: return 文で，処理をユーザに返す。

11 行: main 関数を { で括る。

12 行: プログラムの終了をコメントとして記述する。いつも必要というわけでもない。

課題 1.1: カレントディレクトリを C/Lec01 とせよ。

課題 1.2: このプログラムを ex0101.c という名前で C/Lec01 のディレクトリに保存すること。

課題 1.3: ex0101.c が C/Lec01 のディレクトリ内にあるか確認すること。確認するには，shell の中で，「ls」としたとき，ex0101.c が表示されているかを確認すればよい。

課題 1.4: shell の中で，

gcc -o ex0101 ex0101.c↵

と入力する。エラーがなければ，次に shell の中で

ls ↵
と入力する。ex0101.exe が表示されれば，とりあえずはコンパイル終了である。

課題 1.5: shell の中で，
./ex0101↵
と入力して，出力を確認せよ。(‘‘./’’ を忘れずに)

課題 1.6: ex0101.c の 9 行目の「printf("Hello, World!¥n");」を次の２行に書き換えたとき，出力はどうなるか？プログラムを ex0102.c として保存すること。

```
printf("Hello, ");
printf("World!¥n");
```

課題 1.7: ex0101.c の 9 行目の「printf("Hello, World!¥n")；」を次のように書き換えたとき，出力はどうなるか？プログラムを ex0103.c としてプログラムを保存すること。

```
printf("Hello,¥nWorld¥n");
```

課題 1.8: ex0101.c の 9 行目の「printf("Hello, World!¥n")；」を次のように書き換えたとき，出力はどうなるか？プログラムを ex0104.c として保存すること。

```
printf("Hello,¥n¥n¥nWorld¥n");
```

参考 1　書式付出力関数　printf

printf 関数は「書式付出力関数」と呼ばれるもので，" "で囲まれた書式に基づいて標準出力（多くの場合ディスプレイ）に出力される。ex0101.c のプログラムでは「"Hello, World¥n"を標準出力に出力しなさい」 という意味である。では，最後 ¥n は何だろうか？。ex0103.c と ex0104.c のプログラムでおおよそ察しがつくと思うが，敢えていえば，「改行」という意味の特殊な記号（エスケープ文字列）である。

また，数値を出力するときは，

```
printf("%d¥n",30000);
printf("%6d¥n",30000);
printf("%f¥n",3.141592564);
printf("%7.3f¥n",3.141592654);
printf("Pi=%10.6f e=%10.6f¥n",3.141592654,2.718281828);
printf("A=%10.6e B=%10.6e¥n",82918.2992,0.0045681828);
```

などのようにする。ここで，" "で囲まれたところが出力するときの書式を指定する部分で，実際に出力される値は ，で区切った後の部分である。また，%d は整数を，%f は浮動小数 (いわゆる実数) である。また，%7.3f は小数点以下 3 桁，全体で 7 桁で表示されることを意味している。

printf 関数の詳細は回を追うに従って少しずつ説明していく。

課題 1.9: ex0101.c の 9 行目の「printf("Hello, World!¥n")；」を次のように書き換えたとき，出力はどうなるか？プログラムを ex0105.c として保存すること。

```
printf("%d¥n",30000);
printf("%6d¥n",30000);
printf("%f¥n",3.141592564);
printf("%7.3f¥n",3.141592654);
printf("Pi=%10.6f e=%10.6f¥n",3.141592654,2.718281828);
printf("A=%10.6e B=%10.6e¥n",82918.2992,0.0045681828);
```

第 2 章

変数

C 言語では，プログラム中に出てくる全ての変数は，その変数が使われる関数内 (main 関数も含む) の冒頭で宣言しなくてはならない仕様になっている。これは，BASIC 等の他の言語での型の違いから生じる致命的なエラーを回避し，使う変数は作成者が全て責任を持って使うことができる，というありがたい仕様でもある。

2.1 基本的な変数型

変数にはいくつかの基本的な型がある。整数を扱うときは int ， 文字を扱うときは char，実数を扱うときは float, double などといったものであり，それぞれの用途にしたがって使用する。

表 2.1 主な変数型

型	データ	バイト幅	範囲
char	文字	1byte(8bit)	文字または −128 ~ 127 の整数
int	(符号付) 整数	4byte(32bit)	$-2^{31} \sim 2^{31}-1$ の範囲の整数
float	単精度実数	4byte(32bit)	約 $10^{-38} \sim 10^{38}$(7 桁有効)
double	倍精度実数	8byte(64bit)	約 $10^{-308} \sim 10^{308}$(15 桁有効)

char 型には 1byte のメモリが割り当てられる。1byte = 8bit = 256(2 の 8 乗) の区別ができる。この型を使って半角英数文字が表現できる。また，−128 ~ 127 の整数として使っても良い。C 言語では char 型を使って１つ文字を表現する。では，”Hello”などのような「文字列」はどのように表現するかというと，後で学習する char 型の配列というものを用いる。

int 型には 4byte のメモリが割り当てられる。これによって，4byte = 32bit = 2 の 32 乗の区別が可能となる。具体的には，符号付で $-2^{31} \sim 2^{31}-1$ の整数が表現できる。

float 型には 4byte のメモリが割り当てられる。4byte = 32bit をそれぞれ，符号に 1bit,「指数部」と呼ばれるものに 8bit = −128 ~ 127,「仮数部」と呼ばれるところに残りの 23bit(最小単位 2^{-23}) が割り当てられる。float 型で宣言された数値は単精度実数（単精度浮動小数）と呼ばれる。f を float 型で宣言したとき，f =「符号」×「仮数部」×$10^{\text{「指数部」}}$ で表現される。 例として 3982.2313454 は「符号」が + で,「仮数部」が 0.39822313454 で「指数部」が 4 となる。

double 型には 8byte のメモリが割り当てられる。8byte = 64bit を符号に 1bit,「指数部」と呼ばれるものに 11bit= −1028 ~ 1027,「仮数部」と呼ばれるところに残りの 52bit(最小単位 2^{-52}) が割り当てられる。double 型で宣言された数値は倍精度実数（倍精度浮動小数）と呼ばれる。double 型で宣言された数値も float 型で宣言された数値と同様に「符号」×「仮数部」×$10^{\text{「指数部」}}$ と表現される。

2.2 変数型を変えてみよう

次に，いくつかの変数を使用したプログラムを作成してみよう。 変数 x にある値を入力して，$y=x^2+2x+1$ の値を出力するプログラム ex0201.c を作る。

List 2.1 $y = x^2 + 2x + 1$ を計算するプログラム ex0201.c

```
 1. /* ex0201.c y=x*x+2*x+1 を計算するプログラム */
 2. /* 作成日   2002/4/23 */
 3. /* 作成者   xxxxx */
 4. #include <stdio.h>
 5.
 6. /* 開始  */
 7. int main()
 8. {
 9.     int x,y; /*  x,y を整数型変数として宣言。 （宣言文） */
10.
11.     printf("x を入力してください!!");
12.     scanf("%d",&x);/* x に値を整数型のデータとして入力 */
13.
14.     y=x*x+2*x+1; /* y に値を代入 （代入文）*/
15.
16.     printf("x が%d のとき，y=x*x+2*x+1 は %d です ¥n",x,y);
17.     return 0;
18. }
19. /* 終了  */
```

≪プログラムの解説≫

1〜3 行: 略

4 行: ヘッダーファイル stdio.h を挿入する。

6 行: 略

7 行: プログラムは必ず main 関数で始まる。

8 行: main 関数を"{"で始める。

9 行: x,y を整数型変数として宣言する。

10 行: 切れの良いところに空行を入れておくと，プログラムを見やすくなる。13,15 行も同様。

11 行: printf 関数を使って，"x を入力してください!!" という文字列を出力する。

12 行: scanf 関数を使って，標準入力 (キーボード) から x に整数値を入力する。

13 行: 略

14 行: y に x*x+2*x+1 を代入する。

15 行: 略

16 行: printf 関数を使って，x,y の値を標準出力に出力する。

17 行: return 文で，処理をユーザに返す。

18 行: main 関数を"}"で括る。

19 行: 略

課題 2.1: ~/C/Lec02 のディレクトリを作成し，カレントディレクトリを~/C/Lec02 とせよ。

課題 2.2: このプログラムを ex0201.c という名前で保存し，shell の中で，コンパイルし，実行せよ。また，x にいくつかの数を代入し，計算が正しいかチェックせよ。

課題 2.3: ex0201.c の 9 行目の int を float と置き換えたプログラムを ex0202.c として保存し，コンパイル・実行し，正しく計算されているかを確認せよ。

課題 2.4: ex0202.c の 12 行目の %d を %f と置き換えたプログラムを ex0203.c として保存し，コンパイル・実行し，正しく計算されているかを確認せよ。

課題 2.5: ex0203.c の 16 行目の %d を %f と置き換えたプログラムを ex0204.c として保存し，コンパイル・実行し，正しく計算されているかを確認せよ。

参考 2　書式付入力関数　scanf

scanf 関数は「書式付入力関数」と呼ばれるもので，" "で囲まれた書式に基づいて標準入力（多くの場合キーボード）で入力されたデータをプログラム内に取り込む関数である。ex0201.c のプログラムでは「変数 x に 整数データとして 標準入力から値を入力しなさい」という意味である。ここで，「整数データとして」というのは，scanf 内の “%d” で指定されている。また，“&x” の &は，今のところ，データを入力する際の，「おまじない」だと思って差し支えない（後で，「アドレスとポインタ」(§§6.4.1) について学習した際に，理解できる筈である）。各変数型のデータを入力する際は次のようにする (printf 関数の書式も参照)。

表 2.2 各種データ型に対する printf,scanf 関数

変数型	宣言文	printf の中身	scanf の中身
整数	int i;	printf("%d¥n",i);	scanf("%d",&i);
文字	char c;	printf("%c¥n",c);	scanf("%c",&c);
文字列	char a[10];	printf("%s¥n",a);	scanf("%s",a);
単精度実数	float x;	printf("%f¥n",x);	scanf("%f",&x);
倍精度実数	double y;	printf("%f¥n",y);	scanf("%lf",&y);

などのようにする。ここで，""で囲まれたところが入力するときの書式を指定する部分で，実際に入力される変数は「,」で区切った後の部分である。また，複数のデータを１度に入力する際は

```
scanf("%d %d",&i,&j);
```

とする。書式で指定したデータの数 (%の数と思ってよい) と入力する変数の数が一致していなければならない。さらに，printf では小数点以下 6 桁まで出力するときに，%10.6f と書式を指定したが，入力 (scanf) のときはこのようには入力しない。

scanf 関数のについても，回を追うに従って少しずつ説明していく。

課題 2.6: ex0204.c の 9 行目の float を double と置き換えたときに，正しく動くプログラムを作り ex0205.c として保存せよ。もちろん，コンパイル・実行し，正しく計算されているかを確認すること。

2.3　変数型と演算

角度を「度」から「ラジアン」に変換するプログラムを考えてみよう。まず，次のプログラムを ex0206.c としてディレクトリ　”~/C/Lec02” に保存・コンパイル・実行してみよう。

List 2.2 角度を「度」から「ラジアン」に変換するプログラム

```
 1. /* ex0206.c 角度を「度」から「ラジアン」に変換するプログラム */
 2. /* 作成者 xxxx */
 3. /* 作成日 2002/4/30 */
 4.
 5. #include <stdio.h> /* stdio.h をインクリュードする */
 6. int main() /* main 関数 */
 7. {
 8.     int radian,degree=0; /* 整数型 radian, degree の宣言 */
 9.     float PI=3.141592654; /* PI  円周率 */
10.
11.     while(degree <= 180){
12.         radian = degree*PI/180; /* ラジアンの換算 */
13.         printf("%d %d¥n",degree,radian); /* 結果の出力 */
14.         degree+=10; /* degree を 10 増やす */
15.     }
16.     return 0;  /* 終了 */
17. }
```

≪プログラムの解説≫

5 行: ヘッダーファイル stdio.h を挿入する。

6 行: プログラムは必ず main 関数で始まる。

7 行: main 関数を { で始める。

8 行: radian,degree を整数型変数として宣言し，degree に 0 を初期値として代入する。

9 行: PI を単精度実数型変数として宣言し，これに 3.141592654 を代入する。

11 行: while 文: degree が 180 以下だったら，{ と } で囲まれた部分 (12 行目から 14 行目まで) を処理する。

12 行: radian に，degree に PI をかけ更に 180 で割った値を代入する。

13 行: printf 関数を使って，"%d %d" の書式で，degree と radian を出力する。

14 行: degree の値を 10 増やす (degree の値に 10 加えたものを新たに degree の値とする) 。

15 行: 11 行目の while 文の { に対応させる } を必ずつける。{ } で囲まれたところが while 文の対象であることを明確にする。

16 行: return 文で，処理をユーザに返す。

17 行: main 関数を } で括る。

課題 2.7: このプログラムの出力で各行の桁が揃うようにプログラムを書き換えて，ex0207.c として保存せよ。

課題 2.8: ex0206.c の 12 行目では，degree*PI/180 であったが，これを degree/180*PI とするとどうなるか，書き換えたプログラムを ex0208.c として保存・コンパイル・実行して，ex0206.c のときの結果と比較せよ。

参考 3. while 文

while 文はある条件下での繰り返し（ループ）を実行するための制御文である。条件を自由に設定できる点が，あとで説明する for 文に比べて良い点である。while 文の書式は次の通り。

```
while(条件 1){ ... ①
    処理 1;
    処理２;
    ....
}.................②
```

() の中には，ある条件（ここでは「条件１」）を示す式が入っている。while 文はこの条件が真であるうちは {} で囲まれた部分の処理（ここでは「処理１」，「処理２」，‥）を何度も繰り返し実行する。ここで，① の { と② の } で挟まれている複数の文（ここでは，「処理１;」,「処理２;」など）を複文という。

複文は，後で説明する制御文でよく使われる。

2.3.1 ≪ ex0206.c と ex0208.c の違い≫

さて，課題 2.8 で作った ex0208.c の出力結果と ex0206.c の出力結果はどのように違うだろうか？

ex0206.c の 12 行目が「radian=degree*PI/180;」だったのが, ex0208.c では「radian=degree/180*PI;」に変わっただけである。数学の世界では，a*b/c=a/c*b はいつも成立する。コンピュータでは degree*PI/180 は degree/180*PI と異なるのだろうか？これは，① 演算の順序，② 計算する変数の型，に原因がある。

まず ① について: コンピュータ内の計算は基本的にプログラムの各行の左から右に計算される。例えば，3*4+9 は「3 に 4 をかけてから，これに 9 を加える」という意味になる。したがって，radian に代入されるのは ex0206.c では「degree に PI をかけて，更にその値を 180 で割った値」であり，ex0208.c では「degree を 180 で割った値に，PI をかけた値」となる。

ここまでだったら，ex0206.c も ex0208.c も radian に代入される値自体何も変わらないような気がするが，① の演算の順序に加えて ② の「計算する変数の型」が大きく関わっている。

では ② について考えてみよ。 radian，degree はどちらのプログラムでも整数型の変数として定義されている。ex0206.c では，まず degree に実数の PI がかけられている。この演算によって，(degree*PI) は実数になる。次に (degree*PI) の実数を「180」という整数で割る。実数の整数型への代入では，小数点以下が切り捨てられてしまうので，この場合 (degree*PI) が 180 より小さいときは 0 が，180 以上 360 未満のときは 1 が，360 以上 540 未満のときは 2 が，540 以上 720 未満のときは 3 が radian に代入される。これに対して ex0208.c では，まず degree を 180 で割っている。したがって，(degree/180) は degree が 180 より小さいときは 0，degree=180 のときは 1 となる。この値に，PI をかけると，0.0000 か 3.1415... が radian に代入される。ところで，radian は整数型の変数として宣言されていたので，実際に radian に代入される値は，小数点以下は切り捨てられた値になる。

ex0206.c と ex0208.c の出力の違いが理解できた (?) ところで，プログラムの改良に取りかかろう。ex0206.c や ex0208.c の出力は，小学生の円周率を使った計算のようで，数値的に余りにもお粗末である。ex0206.c のプログラムをもう少しいじってみよう。

課題 2.9: radian を小数点以下３桁まで表示し，各行の桁が揃うように出力するプログラムを作成し，ex0209.c という名前で保存し，コンパイル・実行せよ。

参考 4. 演算と順序 (優先順位)

まずは四則演算: 「たす」,「ひく」,「かける」,「割る」はそれぞれ「+」,「-」,「*」,「/」で表現する。また，a,b が共に整数のときは「a%b」で「a を b で割ったときの余り」を表現することができる。また，四則演算の演算の優先順位は数学での優先順位と同じである。a+b*c は b*c が先に計算され，その後で，a に先の値が加えられる。

すでに，ex0206.c の中に現れているが，C 言語では代入文に 「=」,「+=」,「-=」,「*=」,「/=」の表現がある。これらは代入演算子と呼ばれている。

表 2.3 代入演算子

a=b	a に b を代入	a+=b	a に a+b を代入
a-=b	a に a-b を代入	a*=b	a に b かける
a/=b	a を b で割る		

また，代入演算子と通常の演算の優先順位は次のとおりである。

表 2.4 代入演算子と優先順位

a*=b+c;	a に a*(b+c) を代入
a+=b=c;	b に c を代入し，a に a+b を代入
a=b+=c;	b に b+c を代入し，a に b を代入
a+=b-=c;	b に b-c を代入し，a に a+b を代入

2.4 配列

40 人のクラスの英語のテストの成績を出席番号順に処理したいときに，

```
int a01,a02,a03,a04,a05,a06,a07,...,a40;
```

のように各生徒の成績ごとに変数名を変えて宣言していたのでは，変数の宣言だけでも長く読みづらいプログラムとなってしまうし，後でプログラムを手直ししたりするときも大変な作業となってしまう。

このようなときは，「配列」と呼ばれる同じ変数型で同種のデータをひとまとめにしたものを使うと便利である。ここでは，簡単に配列について説明する。

2.4.1 配列の宣言の仕方

配列の各要素は，通常の変数 (整数型，実数型，倍精度実数型，文字型) と同じなので，配列自身は通常の変数のように宣言する。

- `int n[10];` ..... 10 個の整数型の変数 n[0],n[1],···,n[9] を宣言
- `float f[10];` ..... 10 個の実数型の変数 f[0],f[1],···,f[9] を宣言
- `double a[10];` ..... 10 個の倍精度実数型の変数 a[0],a[1],···,a[9] を宣言
- `char s[10];` ..... 10 個の文字型の変数 s[0],s[1],···,s[9] を宣言

10 個の要素の配列を宣言する場合 (例えば，`int n[10];` としたとき)，実際にプログラムの中で使える 10 個の要素（変数）は `n[0],n[1],n[2],n[3],n[4],n[5],n[6],n[7],n[8],n[9]` になることに注意。この場合，配列の要素としての `n[10]` はなく，`n[10]` は「配列全体の名前」とでも考えていただければよい。プログラムの中で `n[10]` や `n[11]` などを変数として使おうとすると，エラーもしくはプログラムの暴走を招くことがある。**宣言文で `n[10]` ときは，変数としては `n[0]` から `n[9]` までしか使えない** ということを肝に銘じていただきたい。

また，あらかじめ配列の要素のがわかっているときは，次のように宣言することができる。

数の配列

❶ `int a[3]={2,34,23};`
❷ `int a[2][3]={{1,34,23},{1,33,67}};`

❶，❷の例は整数型の１次元 (ベクトル)，２次元 (行列) の例である。整数型に限らず，`int` を `float` や `double` に変えれば，実数型や倍精度実数型に対しても使うことができる。

❷の例は行列の配列であることは述べたが，このままだと，この行列は 2×3 の行列なのか 3×2 の行列なのか，また，各行列要素が何なのかよくわからない。そこで，次のようなプログラム (ex0210.c) で確認してみよう。

List 2.3 ex0210.c

```
 1. /* ex0210.c 行列と配列の対応 */
 2. /* 作成者 XXXXXX */
 3. /* 作成日 2003/5/21 */
 4. #include <stdio.h>
 5.
 6. int main()
 7. {
 8.   int a[2][3]={{11,12,13},{21,22,23}};
 9.   int i,j;
10.
11.   i=j=0;
12.   while(i<2){
13.     while(j<3){
14.       printf("a の%1d 行%1d 列の要素 a_{%1d,%1d}=%2d¥n",i+1,j+1,i+1,j+1,a[i][j]);
15.       j++;
16.     }
17.     j=0;i++;
18.   }
19.   return 0;
20. }
```

課題 2.10:　上のプログラムを `ex0210.c` 作成し，コンパイル・実行してみよ。また，このプログラムの中で宣言された `a` は何行何列の行列か？。さらに，各行列要素を調べよ。

課題 2.11:　上のプログラム `ex0210.c` の１４行目の `printf` 文の中で `i+1,j+1` のように`+1` となっているのはなぜか？答えよ。

文字の配列 (文字列)

❸ `char s[10]="abcdefg";`
❹ `char s[2][10]={"abcdefg","hijklmn"};`

また❸，❹の例は文字変数に対する１次元，２次元の配列の例で，これらは文字列と呼ばれる。❸の例では，`s[0]='a', s[1]='b',s[2]='c',s[3]='d',s[4]='e',s[5]='f',s[6]='g'` としたことになっている。このとき，`s[7],s[8],s[9]` には，特に文字が代入されていないように見えるが，実は `s[7]` には文字列が終わりである記号 "`¥0`" が代入されている (これは，C 言語での約束事の１つである)。

また，文字列として日本語を使うこともできる。この場合は，日本語の文字は「２バイト文字」と呼ばれていて，半角英数字の２倍のメモリスペースが必要になる。したがって，

`char s[11]="文字列の例";`

と宣言すると s[0] と s[1] に「文」, s[2] と s[3] に「字」，s[4] と s[5] に「列」，s[6] と s[7] に「の」，s[8] と s[9] に「例」が代入され，最後に s[10] に ¥0 が入ることになる。

2.4.2 ベクトルの計算

それでは，実際に配列を使ったプログラムを作成してみよう。ただし，次のプログラムには数学関数 (acos，sqrt) が使われているので，コンパイルするときは，最後に “-lm” をつけること (§0.5.2 を参照)。

課題 2.12: 次のプログラムを ex0211.c という名前で作成し，コンパイル・実行せよ。また，下のプログラムを５次元のベクトルに適応できるように修正したものを ex0212.c という名前で作成せよ。

List 2.4 ex0211.c

```
 1. /* ex0211.c ベクトルの内積 */
 2. /* 作成者 XXXXXX */
 3. /* 作成日 2003/05/21 */
 4. #include <stdio.h>
 5. #include <math.h>
 6. #define NDIM 2
 7.
 8. int main()
 9. {
10.   double alpha,la,lb,ab,a[NDIM],b[NDIM];
11.   int i=0;
12.
13.   while(i<NDIM){
14.     printf("ベクトル a の第%d 成分を入力してください-->",i+1);
15.     scanf("%lf",&a[i]);
16.     printf("ベクトル b の第%d 成分を入力してください-->",i+1);
17.     scanf("%lf",&b[i]);
18.     i++;
19.   }
20.
21.   la=lb=ab=0.0;
22.   i=0;
23.   while(i<NDIM){
24.     la+=a[i]*a[i];
25.     lb+=b[i]*b[i];
26.     ab+=a[i]*b[i];
27.     i++;
28.   }
29.   alpha=acos(ab/sqrt(la*lb));
30.   printf("a と b の成す角は%10.3f 度です ¥n",alpha/M_PI*180);
31.   return 0;
32. }
```

第 3 章

フローチャート（流れ図）を作る

フローチャートはプログラミングする際の設計図に当たるものである。複雑なプログラムになればなるほど，あらかじめフローチャートを作っておくと，プログラムの見通しが良くなる。込み入ったプログラを作成する前に、フローチャートをマスターしよう。

3.1　フローチャートの部品

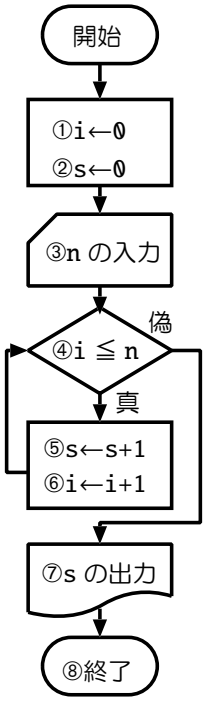
フローチャートの部品には次のようなものがある*(注 1)。

表 3.1 フローチャートの部品

開始/終了	入力	処理 (代入)	条件分岐	定義済み処理	出力
					

3.2　フローチャートの例

ある整数 *n* を入力して，1 から n までの和をとるプログラムのフローチャートを考えよう。 フローチャートの各記号とその意味，更にそれをもとに実際にプログラムを組んだときの例を対応させてみよう。このプログラムで変数 i,s を整数として使用することを宣言する。



フローチャートの意味

① i の初期化 (i を 0 に設定する)
② s の初期化 (s を 0 に設定する)
③ n の入力
④ i が n 以下か？
　(ア) “真”なら
　　⑤ s に i を加える
　　⑥ i を 1 つ増分させる。
　(イ) ”偽”なら
　　⑦ に続く
⑦ s の出力
⑧ 終了

*(注 1) 「定義済み処理」とは，プログラム中で定義された関数である。

課題 3.1: ~/C/Lec03 のディレクトリを作成し，カレントディレクトリを~/C/Lec03 とせよ。

課題 3.2: 上のフローチャートをもとに，ある整数 n を入力して，1 から n までの和をとるプログラムを作成し，ex0301.c という名前で~/C/Lec03 内に保存し，コンパイル・実行せよ。

課題 3.3: 同じようにして，ある整数 n を入力して，1 から $1/n^2$ までの和をとるプログラムを作成し，ex0302.c という名前で保存し，コンパイル・実行せよ。

List 3.1 1 から n までの和を求めるプログラム

```
 1. /* ex0301.c 1 から n までの和を求めるプログラム*/
 2. /* 作成者 xxxxxxxx */
 3. /* 作成日 2002/6/4 */
 4. #include <stdio.h>
 5.
 6. int main()
 7. {
 8.     int s,n,i; /* i,s,n を整数型として宣言 */
 9.
10.     i=0; /* i の初期化 */
11.     s=0; /* s の初期化 */
12.
13.     printf("Input n -->");
14.     scanf("%d",&n); /* n の入力 */
15.
16.     while(i<=n){ /* i が n より小さかったら */
17.         s+=i; /* s に i を加える */
18.         i++; /* i を 1 増やす */
19.     }
20.     printf("%5d %5d¥n",n,s); /* 結果の出力 */
21.     return 0;
22. }
```

第 4 章

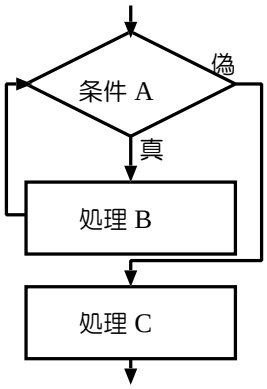
制御文

前回 while 文について学習した。while 文は制御文と呼ばれるものの１つで，ある条件を満たしている間は，“{” と “}” の間にある処理を繰り返すという構文である。ここでは，while 文以外の制御文について学習しよう。

制御文には while 文のほか，for 文，do while 文の繰り返しを制御する文と，if 文，switch 文などの条件によって処理を分ける文，更に，continue 文，break 文，goto 文などがある。ここでは，これらについて順に学習して行く。

4.1　while 文

まず while 文について復習する。while 文は §2 でも学習したように，「ある条件下で繰り返しを実行するための制御文」である。これをフローチャートで描くと次のようになる。



処理が while 文に差し掛かると，まず「条件 A」を評価する。「条件 A」を満たしていると，「処理 B」を実行し再び「条件 A」を評価する，「条件 A」を満たしていないと，そこでは何の処理もされず次の処理「処理 C」へ移る。これをプログラムでは次のように書く。

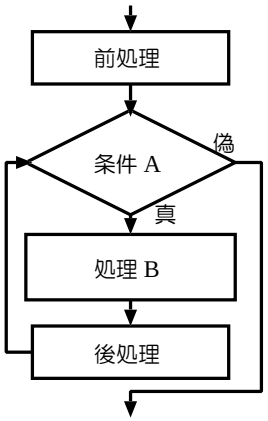
while 文

```
while(条件 A){
    処理 B；
}
処理 C;
```

ここで，“処理 B” は 1 つ (1 行) の処理である必要はない。また，“処理 B” が “条件 A” に関係のない処理であるとき (“処理 B” が何度実行されても “条件 A” の評価が変わらない場合)，この while 文は「無限ループ」と呼ばれるものになり，このプログラムを実行すると，終了しないプログラムになるので，注意が必要である。

4.2 for 文

for 文も while 文と同じように繰り返しを実現する為の制御文である。予め繰り返す回数がわかっている場合に for 文を使うと，プログラムの見通しがよくなる。for 文の使い方の１つをフローチャートで描くと次のようになる。



「条件 A」が真のとき，「処理 B」が実行され，指定した処理が終了したら「後処理」を行い再び「条件 A」を評価する。処理が for 文にさしかかると，最初の１回だけ「前処理」が実行され，次に「条件 A」が評価される。「条件 A」が真の間は，「処理 B」と「後処理」が順次実行される。また，「条件 A」が偽となると，処理が for 文の外に出る。 これをプログラムでは次のように書く。

for 文

```
for(前処理; 条件 A;後処理){
    処理 B;
}
```

課題 4.1: ~/C/Lec04 のディレクトリーを作成し，カレントディレクトリを~/C/Lec04 とせよ。

課題 4.2: 次のプログラムを ex0400.c としてこのディレクトリに保存し，コンパイル・実行してみよう。

List 4.1 ex0400.c

```
 1. #include <stdio.h>
 2. int main(){
 3.   int i,j;
 4.
 5.   for(i=0;i<10;i++){
 6.     for(j=0;j<=i;j++) printf("*");
 7.     printf("¥n");
 8.   }
 9.
10.   for(i=9;i>=0;i--){
11.     for(j=0;j<=i;j++) printf("*");
12.     printf("¥n");
13.   }
14.   return 0;
15. }
```

for 文も while 文も基本的には繰り返しの制御文なので，while 文で書かれたプログラムを for 文で書き換えることが可能である。§2 でつくった ex0209.c のプログラムを ex0401.c とコピーして，while 文の箇所を for 文に書き換えてみよう。

List 4.2 角度を「度」から「ラジアン」に変換するプログラム

```
 1. /* ex0401.c 角度を「度」から「ラジアン」に変換するプログラム */
 2. /* 作成者 xxxx */
 3. /* 作成日 2002/5/28 */
 4.
 5. #include <stdio.h> /* stdio.h をインクリュードする */
 6. int main() /* main 関数 */
 7. {
 8.     int degree; /* 整数型 degree の宣言 */
 9.     float radian; /* 単精度実数型 radian の宣言 */
10.     float PI=3.141592654; /* PI  円周率 */
11.
12.     for(degree=0;degree<=180;degree+=10){
13.         radian = degree*PI/180; /* ラジアンの換算 */
14.         printf("%4d %8.3f¥n",degree,radian); /* 結果の出力 */
15.     }
16.     return 0;  /* 終了 */
17. }
```

≪プログラムの解説≫

5 行: ヘッダーファイル stdio.h を挿入する。

6 行: プログラムは必ず main 関数で始まる。

7 行: main 関数を"{"で始める。

8 行: degree を整数型変数として宣言する。

9 行: radian を単精度実数型変数として宣言する。

10 行: PI を単精度実数型変数として宣言し，これに 3.141592654 を代入する。

12 行: for 文: 「前処理」として degree に 0 を代入。「条件」として「degree が 180 以下？」とし，「後処理」を「degree を 10 増やす」としている。

13 行: 「条件」が真だったら，radian に degree に PI をかけ，更に 180 で割った値を代入する。

14 行: printf 関数を使って，"%4d %8.3f"の書式で，degree と radian を出力する。

15 行: 12 行目の for 文の"{"に対応させる"}"を必ずつける。"{ }"で囲まれたところが for 文の対象であることを明確にする。

16 行: return 文で，処理をユーザに返す。

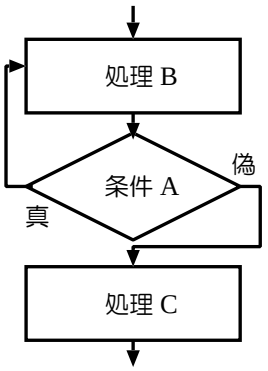
17 行: main 関数を"}"で括る

課題 4.3: 上記のように ex0401.c をつくって，コンパイル・実行し，ex0209.c の結果と比較せよ。

課題 4.4: for 文を使って ex0301.c および ex0302.c を書き換えたものを，ex0402.c および ex0403.c として保存・コンパイル・実行し，実行結果を以前の ex0301 および ex0302 の実行結果と比較せよ。

4.3 do while 文

do while 文は while 文の処理の順番をひっくり返したもので，基本的に while 文と代わりがない。フローチャートは次の通りである。



まず「処理 B」を処理して，「条件 A」を評価する。「条件 A」が真なら再び「処理 B」に戻る。偽なら while 文の外に出る。while 文との大きな違いは，while 文では最初から「条件 A」が真でなければ「処理 B」は全く処理されずに while 文の外に出てしまうのに対して，do while 文では「処理 B」は少なくとも１回は処理される，というところにある。プログラムでは次のようになる。

do while 文

```
do {
処理 B；
} while(条件 A);
```

while 文を for 文で書き換えたのと同じように，ex0209.c のプログラムを ex0404.c とコピーして，while 文の箇所を do while 文を使って書き直してみよう。

List 4.3 角度を「度」から「ラジアン」に変換するプログラム

```
 1. /* ex0404.c 角度を「度」から「ラジアン」に変換するプログラム */
 2. /* 作成者 xxxx */
 3. /* 作成日 2002/5/28 */
 4.
 5. #include <stdio.h> /* stdio.h をインクリュードする */
 6. int main() /* main 関数 */
 7. {
 8.     int degree=0; /* 整数型 degree の宣言 */
 9.     float radian; /* 単精度実数型 radian の宣言 */
10.     float PI=3.141592654; /* PI  円周率 */
11.     do{
12.         radian = degree*PI/180; /* ラジアンの換算 */
13.         printf("%4d %8.3f¥n",degree,radian); /* 結果の出力 */
14.         degree+=10;
15.     }while(degree<=180);
16.     return 0;  /* 終了 */
17. }
```

課題 4.5: 上記のように ex0404.c をつくって，コンパイル･実行し，ex0209.c の結果と比較せよ。

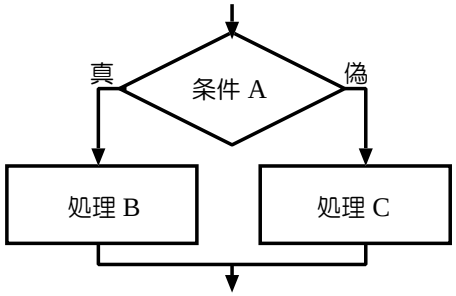
課題 4.6: do while 文を使って ex0301.c および ex0302.c を書き換えたものを，ex0405.c および ex0406.c として保存・コンパイル・実行し，実行結果を以前の ex0301 および ex0302 の実行結果と比較せよ。

注意！！ do while 文の while(条件) の後には必ず ;(セミコロン) を付けること。これを忘れると，コンパイルでエラーがでる。

4.4 if 文

次に，単純な条件判断をする制御文である if 文について学習しよう。if 文を使うことによって，そこまでの処理の結果に応じて２つの異なった処理に分岐させることができる。ここでは，条件の判断のさせ方，処理の分岐のさせ方について学習する。

if 文のフローチャートは左図の通りである。



この意味は，「条件 A」が真であれば「処理 B」を，偽であれば「処理 C」が実行される。プログラムでは次

のようにかく。

if 文

```
if(条件 A){
    処理 B;
} else {
    処理 C;
}
```

「処理 B」や「処理 C」が 1 つの文であるときは，処理を { } で囲む必要が無い。例えば，次のようになる。。

```
if(a>1)
    printf("a(%d) は 1 より大きい ¥n",a);
else
    printf("a(%d) は 1 以下 ¥n",a);
```

条件の表し方は次のとおりである。

表 4.1 条件の表し方

条件	プログラムでの条件の表現	条件	プログラムでの条件の表現
$a > b$	a>b	$a < b$	a<b
$a \geqq b$	a>=b	$a \leqq b$	a<=b
$a = b$	a==b	$a \neq b$	a!=b

また，２つ以上の条件をいっしょに判断する場合は

表 4.2 複数の条件の表し方

条件	表現法
“条件 A” かつ “条件 B”	条件 A&&条件 B
“条件 A” または “条件 B”	条件 A||条件 B

のように&&（「かつ」）または||（「または」）を間に挟む。この条件の表し方は if 文特有の表現ではなく，前述の while 文，for 文, do while 文でも同じように条件を表現する。

課題 4.7:　上記のように ex0407.c をつくって，コンパイル・実行し，正しい結果が得られるか調べよ。

List 4.4 if 文を使ったプログラム

```
 1. /* ex0407.c if 文を使ったプログラム */
 2. /* 作成者 xxxxxx */
 3. /* 作成日 2002/6/04 */
 4. #include <stdio.h>
 5.
 6. int main()
 7. {
 8.     int a,b;
 9.
10.     printf("a と b を入力してください-->");
11.     scanf("%d %d",&a,&b);
12.     if(a>b){
13.         printf("a(%d) は b(%d) より大きい ¥n",a,b);
14.     } else {
15.         printf("a(%d) は b(%d) より大きくない ¥n",a,b);
16.     }
17.     return 0;
18. }
```

課題 4.8: ex0407.c の 12 行目の {，14 行目の } と {，16 行目の } をはずしたものを ex0408.c とし，保存・コンパイル・実行し，ex0407.c と同じ結果になることを確認せよ。

また，本来 if 文は２つの条件分岐で使われるものだが，次のように書くことによって複数の分岐を行うことができる。

```
if(条件 1){
    処理１;
} else if(条件２){
    処理２;
} else if...
    ...
} else if(条件 n){
    処理 n;
} else {
    その他の処理;
}
```

List 4.5 複数の分岐を持つプログラム

```
 1. /* ex0409.c 複数の分岐を持つプログラム */
 2. /* 作成者 xxxxxxx */
 3. /* 作成日 2002/6/04 */
 4. #include <stdio.h>
 5.
 6. int main()
 7. {
 8.     double a;
 9.
10.     printf("実数 a を入力してください-->");
11.     scanf("%lf",&a);
12.     if(a>=1000.0){
13.          printf("a は 1000 以上 (a=%10.3f)¥n",a);
14.     }else if(a>=500.0){
15.          printf("a は 500 以上 1000 未満 (a=%10.3f)¥n",a);
16.     }else if(a>=100.0){
17.          printf("a は 100 以上 500 未満 (a=%10.3f)¥n",a);
18.     }else if(a>=50){
19.          printf("a は 50 以上 100 未満 (a=%10.3f)¥n",a);
20.     }else if(a>=10){
21.          printf("a は 10 以上 50 未満 (a=%10.3f)¥n",a);
22.     }else if(a>=0){
23.          printf("a は 0 以上 10 未満 (a=%10.3f)¥n",a);
24.     }else{
25.          printf("a は 負 (a=%10.3f)¥n",a);
26.     }
27.     return 0;
28. }
```

課題 4.9: 上記のように ex0409.c をつくって，コンパイル・実行し，正しく出力されているかを確認せよ。

4.5 ちょっと Cool な条件文

また，いろいろなプログラムを見ると，「if(a)」とか「if(!a)」等と書かれたものがある。この意味は， if(a) では a の値が 0 以外のとき「真」，（整数で）0 のとき「偽」となる。また，if(!a) では a の値が（整数で）0 のとき「真」，0 以外のとき「偽」となる。

課題 4.10: 次のように ex0410.c をつくって，コンパイル・実行し，どのような出力をするか調べよ。また，39 行目，42 行目の if 文はどのよう

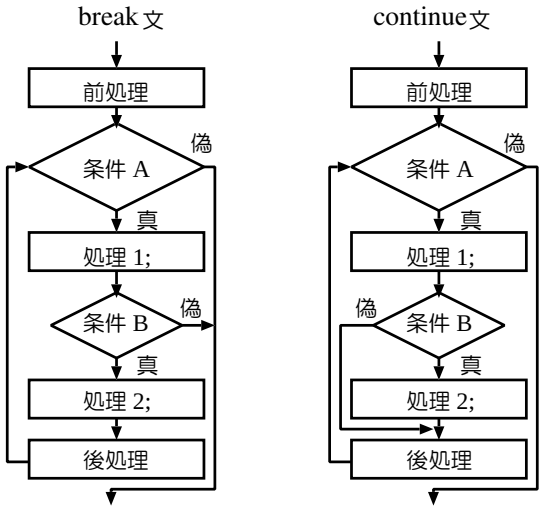
なときに真，偽となるかを調べよ。

List 4.6 ex0410.c

```
 1. /* ex0410.c */
 2. /* 作成者   xxxxx */
 3. /* 作成日 2002/6/4 */
 4. #include <stdio.h>
 5.
 6. int main()
 7. {
 8.     int a,b,n,m;
 9.
10.     n=m=0;
11.     printf("整数 a を 0 から 10 の範囲で入力してください-->");
12.     scanf("%d",&a);
13.     while(a<0||a>10){ /*  「a<0 または a>10」のとき */
14.        ++n;
15.        if(n>3) {
16.            printf("しつこい奴だ！ ¥na は 7 とするぞ！ ¥n");
17.            a=7;
18.        }else{
19.
20.        printf("もう一度，整数 a を 0 から 10 の範囲で入力してください-->");
21.        scanf("%d",&a);
22.        }
23.     }
24.
25.     printf("整数 b を 0 から 10 の範囲で入力してください-->");
26.     scanf("%d",&b);
27.     while(b<0||b>10){ /*  「b<0 または b>10」のとき */
28.        ++m;
29.        if(m>3){
30.            printf("しつこい奴だ！ ¥nb は 6 とするぞ!¥n");
31.            b=6;
32.        }else{
33.            printf("もう一度，整数 b を 0 から 10 の範囲で入力してください-->");
34.            scanf("%d",&b);
35.        }
36.     }
37.     printf("a は%2d,  b は%2d です。¥n",a,b);
38.
39.     if(a-b) printf("38 行目:(真) a と b は等しくない。 a-b=%2d¥n",a-b);
40.     else printf("39 行目:(偽) a と b は等しい。 a-b=%2d¥n",a-b);
41.
42.     if(!(a-b)) printf("41 行目:!(真) a と b は等しい (a-b)=%2d¥n",a-b);
43.     else printf("42 行目!(偽) a と b は等しくない。 a-b=%2d¥n",a-b);
44.
45.     return 0;
46. }
```

4.6 break 文と continue 文

break 文と continue 文は，for 文，while 文，do while 文などの繰り返し処理から脱出したり (break 文)，繰り返しの処理をある条件のときだけスキップしたいとき (continue 文) に使う。for 文を使った繰り返しの処理の中で用いられる例では，次のようなフローチャートが書ける。



左の図が break 文のフローチャートで，これを使うと，繰り返しの中で「条件 B」が真になったとき繰り返しのループから脱出することができる。一方，右の図は continue 文のフローチャートで，これを使うと，繰り返しの中で「条件 B」が真になったとき「処理２」をスキップして次の繰り返しへ移ることができる。上のフローチャートは，プログラムの中ではそれぞれ次のように書くことができる。プログラムに書くと非常に似た形をしているが，機能的には全く異なることに注意しよう。

break 文

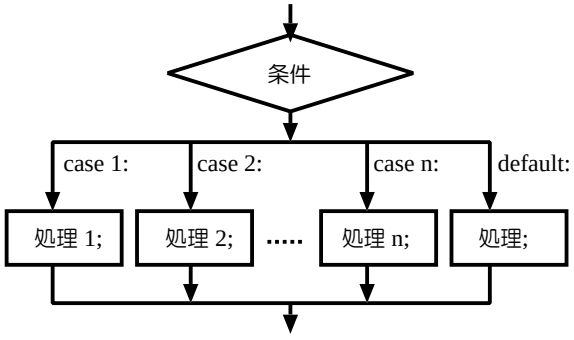
```
for(前処理; 条件 A; 後処理){
    処理 1;
     if(条件 B) break;
    処理 2;
}
```

continue 文

```
for(前処理; 条件 A; 後処理){
    処理 1;
     if(条件 B) continue;
    処理 2;
}
```

4.7 switch 文

いくつかの場合分けをするときに switch 文を使うと便利なことがある。if 文での分岐は「真」か「偽」かで２つの場合にしか分岐できなかったが，switch 文では「いくつかの case の中からどれを実行するか」を処理に応じて分岐させることができる。フローチャートは次のようになる。



これをプログラムで書くと次のようになる。

switch 文

```
int m;
switch(m){
    case 1:
     処理１;
      break;
    case 2:
     処理 2;
      break;
       :
       :
    case n:
     処理 n;
      break;
    default:
     処理;
}
```

ここで，m が 1 のとき case 1 で指定された処理が行われ，$m = 2, 3, \ldots, n,$ でそれぞれの処理が実行される。m が 1 から n までのどれにも該当しなかったときには default で指定された処理が実行される。また，各 case

での処理が終了した時点で break 文を 1 つ入れておく必要がある。break 文を省略したときは次の case で指定された処理も実行される。例えば case 3 と case 4 の間に break 文がないときは，*m* が 3 のとき，case 3 と case 4 で指定された処理が実行されることになる。また，switch 文で評価する式（上の例では m の値）は整数型でなければならない。

4.8　繰り返しの制御文の応用 — 円周率を求めよう —

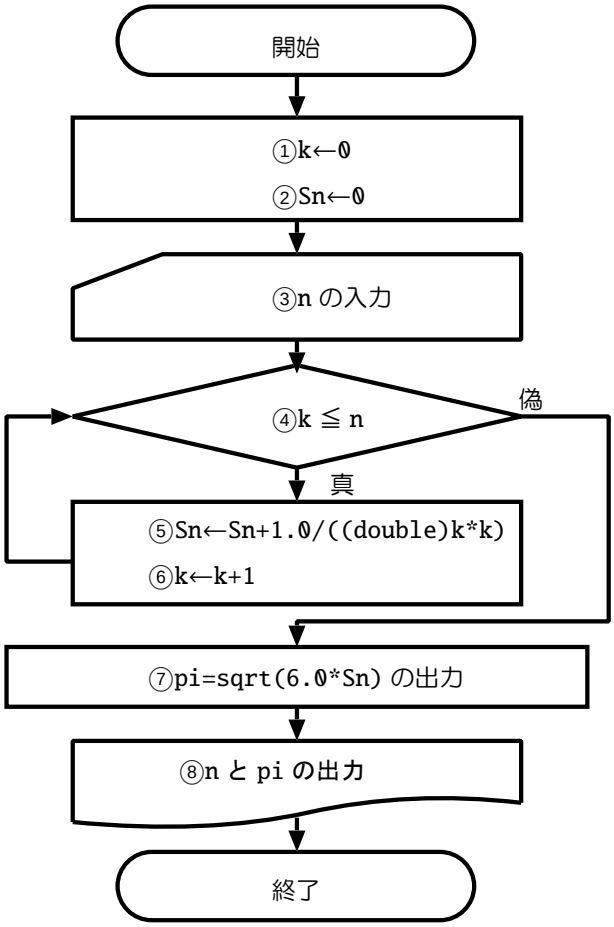
繰り返しの制御文では同じような処理を行うのに適している。例えば数列を求める問題や数列の総和を求める問題には，欠くことのできない処理である。ここでは，簡単な数列の総和を求めるプログラムをつくろう。

次のような級数を考えよう。

$$S_n = \sum_{k=1}^{n} \frac{1}{k^2}$$

この数列の極限 ($n \to \infty$) は $\pi^2/6$ となることが知られている。計算機の上で無限大を表現することはできないので，適当な n までの和を求めて，得られた総和を 6 倍しそれの平方根 (sqrt 関数を使う) をとれば，比較的?π に近い値が得られるはずである。

課題 4.11:　右のフローチャートをもとに，1 から入力した n までの逆数の 2 乗の和 を求め，得られた結果を 6.0 倍して平方根をとった値 (pi とする) を，n と共に表示するプログラム ex0411.c を作成せよ。



《ヒント》

- このプログラムで使用する変数は k,n(整数型)，Sn,pi(倍精度実数型) である。
- 平方根をとる関数は sqrt 関数で math.h で定義されている。プログラムの始めのところで，

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

とする。（sqrt の使い方は⑦を参照。）
- 1.0/((double)k*k) を Sn に加えていく。

第 5 章

ファイル操作

ここではファイルに格納されているデータを読み込む方法およびデータをファイルに書き込む方法について学習する。

データの数が多くなるとデータを１つ１つ入力したり，１つ１つ計算結果をノートに記録したりするのが大変になる。また，プログラム上では正しく計算しているのに，入力ミスや記録ミスなどがあったのではデータとして使うことができない。そこで，この章では計算させるデータをキーボードから入力するのではなく，予め作成したファイルにあるデータをプログラムの中から読み込ませ，そのデータを使ってデータ処理し，最終的な結果も別のファイルに格納する方法を学習する。

5.1　ファイルの取り扱い

プログラムの中で，ファイルに保存されているデータを読み込んだり，計算結果をあるファイルに書き込ませたりする為には，

1. 目的のファイルを開いて（プログラムの中で認識させる）
2. データを読み込ませる/書き込ませる
3. ファイルを閉じる（メモリー上から解放する）

という作業をする必要がある。

5.2　ファイルの読み込み

まず① を行うには fopen 関数を使う。この関数を使って，「なんと云う名前」のファイル（ファイル名指定）を「どういうモード」（モード指定）で「どこ」（メモリー上のどこ）に開くのかを指定する。最初のファイル名指定とモード指定は fopen 関数の２つの引数で，また，fopen 関数の戻り値でメモリー上のどこにそのファイルを開くのかを指定する。例えば次のように:

```
FILE *fin;  /* ファイルポインタ fin を宣言する。 宣言文の１つ */
fin=fopen("file.dat","r");
```

この例では「file.dat”というファイルを r モード（読み込みモード）でファイルポインタ fin に開く」ということを示している。この関数を実行することによって，メモリー上の fin で指定されたところに，tmp.dat の内容が確保される。もし，tmp.dat がないときには NULL という値が fin に代入される。fin を宣言するときは，必ず fin の前に “*” がなければならない。このようにすれば，fin がファイルポインタとして宣言されていると認識される。また，ファイル名を指定するときは文字列として指定しなければならないので，ファイル名を”で囲む必要がある。指定したファイルからデータを読んだりデータを書き込ませたりする作

業② についての概要を説明する。この作業を行うためにいくつかの関数が定義されているが，ここでは最も簡単な関数について説明をしよう。キーボードからデータを入力するときは scanf 関数を使った。これと殆ど同じ関数がファイルからデータを入力する（読み込む）ために fscanf 関数が用意されている。fscanf 関数は fscanf(ファイルポインタ，入力データの書式，値を代入する変数) のように使う。これまでの scanf 関数と異なるのは，この関数の最初の引数がファイルポインタになっている点だけである。第２・第３引数は scanf 関数の第１．第２引数と全く同じである。ここまでが② の作業である。

最後に fclose 関数で fin を通して開いていたファイルを閉じ，fin に確保されていたメモリーを解放する必要がある。これが③ の作業である。

ファイル”tmp.dat”に以下のようなデータがあるとき，プログラムの中でデータの読み込む例を次に示す。

List 5.1 ex0501.c のプログラム

```
 1. /* ex0501.c データを読み込ませるプログラム */
 2. /* 作成者   xxxxxx */
 3. /* 作成日   2002/07/xx */
 4. #include <stdio.h>
 5. #include <stdlib.h>
 6. #define N 5
 7.
 8. int main()
 9. {
10.     FILE *fp;
11.     char name[N][20];
12.     int i ;
13.     int num[N],kokugo[N],sansuu[N];
14.     double hyouka[N];
15.
16.     fp=fopen("tmp.dat","r");
17.     if(fp==NULL){
18.         printf("ファイルがありませんでした");
19.         exit(1);
20.     }
21.     for(i=0;i<N;i++)
22.       fscanf(fp,"%d %s %d %d %lf",
23.         &num[i],name[i],&kokugo[i],&sansuu[i],&hyouka[i]);
24.     fclose(fp);
25.
26.     for(i=0;i<N;i++)
27.       printf("%2d %s %4d %4d %5.2f¥n",
28.         num[i],name[i],kokugo[i],sansuu[i],hyouka[i]);
29.     return 0;
30. }
```

tmp.dat の内容（スペースは全て半角）

1	浅井清和	56	45	2.3
2	井上ひかる	45	66	2.5
3	宇野泰三	91	21	4.1
4	遠藤芳美	54	25	5.4
5	岡村文雄	87	41	7.3

《プログラムの解説》

５行:　stdlib.h には標準的な関数がされている。ここに定義されている関数の１つである exit 関数（１９行目）をプログラムの中で使う為，ここで ex0501.c の中で include する。

６行:　N を 5 と定義して，プログラムの中で定数として使う。

１０行:　fp をファイルポインタとして宣言。

１１行:　２０個の文字の並びを確保する文字列 name を N 個確保する。

１３行:　num,kokugo,sansuu を整数型の変数として N 個ずつ確保する。ここで，

int num[10];

のように宣言すると，num は num[0] から num[9] までの 10 個の変数が定義されることになる。このような変数を配列という。int num[10]; の例では 10 個の成分をもった整数のベクトルと思えば

よい。更に，

int A[10][3];

と宣言される場合は各成分 A[i][j] が整数 (int) の 10×3 の行列の成分 A_{ij} ようなものと考えればよい。9 行目の

char name[N][20];

は文字が 20 個入るベクトルを N 個用意するという意味である。

１４行: hyouka を倍精度実数型の変数として N 個確保する。

１６行: tmp.dat の内容を「読み込みモード (r モード)」でファイルポインタ fp に記憶させる（ファイルを開く。① の作業）。fp で取り込むはずの tmp.dat がないときは，fp に NULL が代入される。

１７行～２０行: fp で取り込むはずだった tmp.dat がないとき，「ファイルがありません」と出力させて，プログラムを exit 関数で終了させる。

２１行～２３行: for 文を使って，ファイルポインタ fp に記憶されているファイル（今の場合は”tmp.dat”）のデータを，整数 (%d)，文字列 (%s)，整数 (%d)，整数 (%d)，倍精度 1 実数 (%lf) の要領で，num[i],name[i],kokugo[i],sansuu[i],hyouka[i] の各変数に取り込む（② の作業）。ここで，scanf 関数・fscanf 関数共に注意を要するのは，1 つの変数を読み込むときには必ず変数の前に&をつけ，文字列等の複数の データを読み込むときは& はつけない点である。

２４行: ファイルポインタ fp で取り込んだファイルをメモリー上から解放する（ファイルを閉じる。③ の作 業）。

２６行～２８行: for 文を使って，各変数に取り込んだデータを画面上に出力する。

課題 5.1: ~/C/Lec05 のディレクトリを作成し，カレントディレクトリを~/C/Lec05 とせよ。

課題 5.2: tmp.dat および ex0501.c を ~/C/Lec05 に作成し，更に，ex0501.c をコンパイル・実行して，出力結果が tmp.dat と同じようなものになっていることを確認せよ。

5.3 ファイルへの書き込み

ファイルに関する手順は基本的に「2. ファイルの読み込み」で行ったのと同じである。まず① で fopen 関数を使ってファイルポインタ fout に書き込みモードで書き込ませるファイルの内容をメモリー上に確保する。例えば

```
FILE *fout;
fout=fopen("kekka.dat","w");
```

のような具合である。この例では、ファイル名を kekka.dat とした。また，fopen 関数の 2 つ目の引数では，”w”としてあるが，これはファイル kekka.dat を「書き込みモード」でオープンしなさい，という命令である。

更に「書き込みの場合注意しなければならない」のは，”w”を指定すると，それ以前にあった kekka.dat へ，新たに上書き保存してしまう，という点である。せっかく苦労して作ったファイルも，プログラムで書き込みモードでファイルがオープンされると一瞬のうちにその内容が消されてしまう。これを避ける為に，「書き込みモード」の変わりに，「アペンドモード」が用意されている。これは，「新たに書き込むデータはそれ以前に作成したデータの一番下から書き加える」というモードである。これは fopen 関数の第 2 の引数で”w”の変わりに”a”を指定すればよい。

以上でファイルへ書き込むときの手順① は終了。次に，実際にデータをファイルへ出力する方法② を説明する。「2. ファイルの読み込み」の② のところの fscanf 関数の変わりに fprintf 関数を使うことで，簡単にファイルにデータを出力することができる。fprintf 関数の使い方は基本的に printf 関数と同じである。キーボードからの入力が scanf 関数であったのに対してファイルから入力する命令が fscanf 関数であったのと同様に，画面への出力が printf 関数であるのに対して，ファイルに出力するのが fprintf 関数である。したがって，fprintf 関数は fprintf(ファイルポインタ，出力データの書式，出力する変数) のように使う。ファイルを閉じるのは「2. ファイルの読み込み」の③ と全く同じである。 さて，下の例では ex0501.c のプログラムを少しいじって，tmp.dat のデータを使ってある計算をしてその結果を kekka.dat に出力させるプログラムを考えることにする。

List 5.2 ex0502.c のプログラム

```
 1. /* ex0502.c データを読み込ませ,
 2.       計算結果をファイルに書き込ませるプログラム */
 3. /* 作成者   xxxxxx */
 4. /* 作成日   2002/07/xx */
 5. #include <stdio.h>
 6. #include <stdlib.h>
 7. #define N 5
 8.
 9. int main()
10. {
11.      FILE *fin,*fout;
12.      char name[N][20];
13.      int i ;
14.      int num[N],kokugo[N],sansuu[N],total;
15.     double hyouka[N];
16.
17.      fin=fopen("tmp.dat","r");
18.      if(fin==NULL){
19.          printf("ファイルがありませんでした");
20.           exit(1);
21.      }
22.     for(i=0;i<N;i++)
23.       fscanf(fin,"%d %s %d %d %lf",
24.         &num[i],name[i],&kokugo[i],&sansuu[i],&hyouka[i]);
25.     fclose(fin);
26.     fout=fopen("kekka.dat","a");
27.     for(i=0;i<N;i++) {
28.         total=kokugo[i]+sansuu[i];
29.         printf("%2d %s %4d¥n",num[i],name[i],total);
30.         fprintf(fout,"%2d %s %4d¥n",num[i],name[i],total);
31.     }
32.      fclose(fout);
33.      return 0;
34. }
```

《プログラムの解説》

１１行:　fin,fout をファイルポインタとして宣言する。fin は既存のファイル用に，fout は計算結果をしまうファイル用に宣言する。

２６行:　ファイル kekka.dat をファイルポインタ fout に「アペンドモード」でオープンする（① の作業）。

２７行～３１行:　for 文を使って，各生徒の kokugo[i] と sansuu[i] の和を total に代入し，その結果を画面および fout に出力する（② の作業）。

３２行:　ファイルポインタ fout を使って書き込んだファイルをメモリー上から解放する（③ の作業）。

課題 5.3:　ex0502.c のプログラムを作成，コンパイル，実行し出力結果および kekka.dat の内容が正しいか確認せよ。

課題 5.4:　次のデータを planet.dat として作成し，そのデータから惑星の公転周期の２乗を軌道半径の３乗で割ったものが惑星に依らず殆ど一定であることを示すプログラム planet.c を作成せよ。planet.dat の内容（１列目は惑星名，２列目が公転周期，３列目は軌道半径である。また全てのスペースは半角スペースであることに注意せよ）

planet.dat の内容

水星	0.24085	0.371
金星	0.61521	0.724
地球	1.00004	1.000
火星	1.88089	1.508
木星	11.86223	5.193
土星	29.45772	9.516
天王星	84.013	19.14
海王星	164.79	30.08
冥王星	248.4	36.78

参考 fopen 関数

使い方：

```
FILE *fp;/* ファイルポインタ */
        fp=fopen(ファイル名, オープンモード);
```

意味：ファイル名（文字列で指定。文字列は必ず”で囲む）で指定したファイルをオープンモードにしたがってオープンする。ファイルが正しくオープンされると，指定したファイルポインタにファイルの内容が確保される。ファイルがないときなどエラーの場合は NULL がファイルポインタの値として代入される。

オープンモード；

- ”r”は「読み込みモード」
- ”w”は「上書きモード」
- ”a”は「アペンド（追加）モード」

”w”,”a”共に書き込みモードで使用する。”w”,”a”では指定したファイルがなければ，新たにそのファイルを作成する (このときエラーはでない。したがってファイルポインタに NULL は代入されない)。

第 6 章

関数

C 言語のプログラムは関数の寄せ集めで構成されている。これまでにも良く出てきた main 関数もその１つである。また，同じ操作をパラメータを変えて何度も行うときに，同じ操作を示すプログラムを羅列するより，予めその操作を関数として定義しておき，関数の引数だけ変えて値を求めたほうが，プログラムを短くかつ明瞭に記述することができる。また，１度しか実行されない処理も，関数として定義してやることによって，明瞭な流れのプログラムを記述することができる。兎にも角にも，C 言語をマスターするには関数に対する理解は不可欠である。ここでは，関数の作り方，利用の仕方等について説明する。

6.1　関数の基本形

ここでは，関数の基本な形について紹介する。関数は基本的に次のよう構成になっている。

関数の基本形

```
戻り値のデータ型 関数名(引数のデータ型 引数, ...){
    関数内で使う変数の宣言;
        処理;
    return 戻り値;
}
```

① **戻り値のデータ型**には，int,char,float,double,void などを書き，その関数を実行して得られる結果が，関数の呼び出し側でどのようなデータ型 (char,int,float,double) で扱わなければならないかを指定する。また，関数の呼び出し側に値を戻さないときは，void 型で関数を宣言する。

② **関数名**は半角英数文字で始まる任意の名前を付けることができる（ただし，予約語は使うことはできない）。しかし，同じプログラム内で異なった内容の関数を同じ名前にすることはできない。

③ 関数名の後に**引数**と呼ばれる並びを”(”と”)”で囲む。関数を定義するときは，**引数のデータ型**も宣言しておかなければならない。
例えば三角関数の sin(x) は sin が関数名で，x が引数になっている。この例で sin(x) の引数 x は実数であることは暗黙の了解とされている（実は x が複素数でも sin 関数は定義されている）。
しかし，「計算しろ」と命令するほうには暗黙の了解があっても，命じられた側ではそのような了解は通用しない（ましてや相手は機械—“コンピュータ”—なのだから人の常識など全く通用しない）。また，$S(x,n) = \sin(n\pi x)$ と定義したとき，n が実数なのか整数なのか不明である。機械が計算するときは，実数か整数かでメモリーの確保の仕方が異なるために，n のデータ型が不明のままでは計算できない。したがって関数を定義するときは引数も必ずデータ型を指定しなければならない。

④ **関数の内容**は次の行の “{” から一番下の “}” の間で指定する。関数の中 (“{” から “}” まで) の書き方は main 関数を書くときと全く同じである。ただ，この関数で使われる引数（関数名の後の “（” と “）” で囲まれている部分）に関しては関数内で宣言しない。関数内で宣言でしなければならないのは，その関数の中で初めて使われる変数に対してのみである。また，この中で使われる変数はこの中でしかその意味を持たない。ある関数 funcA と別の関数 funcB 内で同じ変数 a を宣言し使っても，それぞれの a の間には何ら関係がない。このような変数を「**ローカル変数**」という。これに対して，プログラム全体で共通の値を使う場合は全ての関数の外でその変数を宣言する必要がある。

このような変数を「**グローバル変数**」という。このグローバル変数を多用すると，予期できない計算ミスを招くこともあるので，これを使用するときは注意なければならない。

⑤ 関数の中の一番後に，関数内で得られた結果を return 文で呼び出し側に戻す。この戻す値のデータ型が「戻り値のデータ型」である。実は，戻り値のない関数も定義できる。このときは「戻り値のデータ型」は “void” で指定し，最後の return 文は省略できる。

以上の観点からこれまで作成してきたプログラムの main 関数を見てみよう。main 関数はいつも “int main()” という形で始まっている。int は機械に返す戻り値である。この戻り値は main 関数の一番後 の “return 0” で指定している（機械は戻り値が 0 のとき正常終了とみなすように設計されている）。関数名は main，引数がないので () の中は空欄になっている。“int main()” の代わりに，“int main(void)” として，この関数には引数がないということを明示しても良い。また，main 関数も “{” と “}” で囲まれている。したがって，main は関数としての体裁を備えていることがわかる。次の関数の例を見てみよう。

関数の例

```
1. double MaX(double a,double b)
2. {
3.    double c;
4.
5.    if(a>=b) c=a;
6.    else c=a;
7.    return c;
8. }
```

この関数は，関数を呼び出し側で a,b の値を関数 MaX に渡し，関数 MaX の中で，その値を使って 5 行目から 6 行目までの処理を実行し，7 行目で return 文を使って最終的な値を関数の呼び出し側へ返す，という一連の処理を行っている。1 行目の冒頭に double とあるのは，呼び出し側に double の変数型でデータを返すことを意味している。() の中は 2 つの倍精度実数型の変数を使ってこの関数が処理されることを意味している。

この関数は，例えば次のようにプログラムの中で定義され，使われる。

List 6.1 ex0601.c

```
 1. /* ex0601.c 関数を使ったプログラム */
 2. /* 作成者 xxxxx */
 3. /* 作成日 2002/07/09 */
 4. #include <stdio.h>
 5. double MaX(double a,double b);    /* 関数のプロトタイプの宣言を定義する */
 6. int main() /* main 関数 */
 7. {
 8.    double aa,bb,cc;
 9.
10.    printf("数値を入力してください a-->");
11.    scanf("%lf",&aa);
12.    printf("もう１つ数値を代入してください b-->");
13.    scanf("%lf",&bb);
14.
15.    cc=MaX(aa,bb); /* 関数 MaX を aa,bb を使って呼び出している。cc に結果が戻される */
16.    printf("%f と%f では%f のほうが大きい ¥n",aa,bb,cc);
17.    return 0;
18. }
19. double MaX(double a,double b)
20. {
21.    double c;
22.
23.    if(a>=b) c=a;
24.    else c=b;
25.    return c;
26. }
```

《プログラムの解説》

5 行目: プログラム中で使われる関数の仕様を予め宣言している。これをプロトタイプ宣言という。この宣言によって，「２つの double 型の値を引数とし，実行すると double 型のデータを呼び出し側に返す “MaX” と名付けられた関数がプログラム内で使用されるぞ！」と機械に心の準備 (?) をさせる。また，この行末には必ず “;” をつける必要がある。簡単に自作関数のプロトタイプ宣言をするには，まず作った関数の１行目をコピー (Ctrl+c) し，次に main 関数の前あたりにそれを貼り付け (Ctrl+v) て，最後にその行末にセミコロン（;）を付け加えれば良い。

15 行目: 関数 MaX が aa と bb の値を使って呼び出され，得られた結果を cc（倍精度実数型で宣言）に代入する。

19 行目～26 行目: 関数 MaX の定義をしている。

19 行目: MaX は① double 型の戻り値を呼び出し側に返す関数であることを “ double MaX(...” で明示している。② 関数 MaX は double 型の変数 a と b の引数を使用することを，（）の中で宣言している。

20 行目: 関数 MaX は { からその内容が記述される。

21 行目: 関数内で使用される変数を宣言する。ここでは double 型で c が宣言されている。c に代入される値はこの関数内でのみ有効である (ローカル変数)。

25 行目: 戻り値として，c の値を呼び出し側に返す。

26 行目: } で関数 MaX の定義が終了したことを示す。

課題 6.1: ~/C/Lec06 のディレクトリを作成し，カレントディレクトリを~/C/Lec06 とせよ。

課題 6.2: 上記のプログラムを ex0601.c として作成し，コンパイル・実行せよ。また，実行したときの出力結果が正しいか検討してみよ。

課題 6.3: 上記のプログラムを参考に，小さいほう

の値を求める関数を MiN を定義し，main で「小さいほうの値」(MiN による結果) と「大きいほうの値」(MaX による結果) をそれぞれ出力させるプログラム ex0602.c を作成・コンパイル・実行し，出力結果が正しいことを確認せよ。

6.2 値を返さない関数・引数のない関数

6.2.1 戻り値のない関数

戻り値のない関数は void 型と呼ばれる型で宣言する。この関数を用いた例を簡単に紹介しよう。以前 ex0400.c のプログラムで＊印の三角形を作った。このプログラムを，関数を使って書き直してみよう。

List 6.2 void 関数の例 ex0603.c

```
 1. /* ex0603.c void 関数の例 */
 2. /* 作成者   xxxx */
 3. /* 作成日   2002/07/09 */
 4. #include <stdio.h>
 5. void star(int n);
 6.
 7. int main()
 8. {
 9.    int i,j;
10.    for(i=0;i<10;i++) star(i);
11.
12.   for(i=9;i>=0;i--) star(i);
13.   return 0;
14. }
15.
16. void star(int n)
17. {
18.   int i;
19.   for(i=0;i<n;i++)printf("*");
20.   printf("¥n");
21. }
```

課題 6.4: ex0603.c を作成・コンパイル・実行し，ex0400.c と同じ結果になるかを調べよ。

6.2.2 引数のない関数

戻り値もなく，引数もないような関数も実は存在する。例えば，決まりきった出力を何度もするときは，このような関数が使われるし，１度しか登場しない処理もプログラムの流れを明確にするために関数として使用したりする。例えば次のようなプログラムを書くと，main 関数を見ただけでそのプログラムが何をするプログラムなのか把握できる。

```
#include <stdio.h>
.................
void Start(void);
double Sekibun(double a,double b);
double func(double x);
void End(double x);

int main()
{
  int .................;
  double s,a,b,........;
  ......................;

  Start();
  s=Sekibun(a,b);
  End(s);
  return 0;
}

void Start(void)
{
    printf("プログラムの始まり始まり ¥n");
}
void End(double x)
{
  printf("積分結果は%15.6f です ¥n",x);
}
double Sekibun(double a,double b)
{
    ..............;
}
double func(double x)
{
    ..............;
}
```

6.3　データの渡し方

6.3.1　数値を渡す (Call by value)

一般に関数を呼び出す (call する) ときには，呼び出し側から関数に渡すパラメータ（引数）がある。先の関数の例 MaX では，関数の呼び出し側 (ex0601.c の例では main 関数) で変数 aa と bb に代入された値が関数 MaX にデータとして渡されている。また，関数 MaX では main 関数で指定されたデータを変数 a,b として受け取り所定の処理を行って，結果を変数 c に代入し main 関数に値を渡している。main 側では，MaX から戻ってきた値を変数 cc として受け取る。このように関数の間でデータのやり取りをすることを「データ渡し」特に，データが数値の場合を「数値渡し (Call by value)」という。

しかし，「数値渡し」の方法では関数の呼び出し側で，いくつかのデータを関数へ渡しても，関数から返ってくるデータは１つしか返って来ない，という限界がある。例えば，上記の MaX は「２つのデータ a,b を関数に渡して，大きいほうの値 c が戻ってくる」という関数であったが，「２つのデータ a,b を関数に渡して，大きいほうのデータを c, 小さいほうのデータを d として呼び出し側に戻す」ような関数は「数値渡し」の方法ではできない。このような関数は，下記 (§§ 6.4) の「アドレス渡し (Call by address)」の方法で関数間のデータの受け渡しを行う。

6.3.2　アドレスを渡す (Call by address)

関数間のデータのやり取りは数値だけでなく，複数の数値や文字列といった「配列」である場合もある。関数間で配列に入っているデータを渡すのに，配列の各要素を順々に渡していたのでは時間がかかってしまう。そこで，この場合には配列の記憶されている場所（アドレス）だけを関数に渡して，あとは関数に下駄を預けてしまう，というちょっとズルイ技がある。これを「アドレス渡し (Call by address)」という。

「アドレス」とはその変数が記憶されているメモリー上の番地である。プログラム内で変数を宣言すると，宣言された変数の型に応じてメモリー上で使われていない場所（その場所は固有の番地–address–を持つ）に必要バイト数の領域が確保される。通常の変数については，アドレスについて特に意識する必要はないが，配列などを使う場合には「アドレス」という概念について知っておいたほうが良いこともある。その１例が，関数での「アドレス渡し (Call by address)」である。

宣言された変数が N 個の要素を持つ配列の場合では，その変数型に応じたバイト数分の N 個の連続した領域がメモリー上に確保される。したがって配列の場合には，関数を呼び出す側としては，その配列の先頭のデータがメモリー上のどの場所（アドレス）にあって，その類のデータがいくつ連続しているかを教えてやれば，後は関数にすべて任せてしまえる。一方関数は，そのアドレスに記憶されているデータを順々に取り出して，自分のこなさなければならない処理を勝手にすることができる。

これは，新聞配達のバイト学生？（関数）と雇い主（関数の呼び出し側）の関係に似ている。雇い主はバイトの学生に最初に配達する家の住所（アドレス）と，そこから連続して 10 軒（配列の大きさ）の家に新聞を配達するように依頼する（一軒一軒の住所と配達先の名前をバイト学生に知らせる必要はない。ホント？）。バイト学生は地図を頼りに，最初の配るべき家を見つけそこから連続 10 軒の家に新聞を配り，月末になったら各家に言って集金して，これを雇い主に渡せば彼の仕事は終わりである。

数値入っている配列を関数に渡す場合は，実際に配列に入っているデータがいくつなのかを呼び出し側で責任もって教えなければならない。上の例えで言えば，「バイト学生に何軒新聞を配るのかを教えない雇い主などいない」のと同じことである。

文字列を関数に渡す場合は更に簡単である。というのは，「文字列の終わりには必ず’¥0’ をつける」というのが C 言語の仕様になっている。したがって，関数のほうで’¥0’ を見つけた時点でアドレスをたどる作業を止めればよいわけである。最後に配った家の隣の家に「うちは XX 新聞は取っていません」と新聞受けに張り紙があるようなものである。

さて，配列を関数に渡すプログラムを実際に組んでみよう。次の例は 2 つの実数型の配列 a,b を関数に渡して、２つの内積を計算するプログラムである。

List 6.3 ex0604.c

```
 1. /* ex0604.c 数値の配列を関数に渡すプログラム　N 次元の内積 */
 2. /* 作成者　xxxxxx */
 3. /* 作成日 2002/07/16 */
 4. #include <stdio.h>
 5. double ScalarProduct(double a[],double b[], int n);
 6.
 7. int main()
 8. {
 9.   double a[10],b[10],sprd;
10.   int N,i;
11.
12.   printf("ベクトルの次元はいくつ?-->");
13.   scanf("%d",&N);
14.
15.   for(i=0;i<N;i++){
16.     printf("ベクトル a の%2d 番目の成分の値を入力してください。-->",i+1);
17.     scanf("%lf",&a[i]);
18.   }
19.
20.   for(i=0;i<N;i++){
21.     printf("ベクトル b の%2d 番目の成分の値を入力してください。-->",i+1);
22.     scanf("%lf",&b[i]);
23.   }
24.
25.   sprd=ScalarProduct(a,b,N); /* a と b の内積 */
26.   printf("a と b の内積は%5.2f です。¥n",sprd);
27.
28.   return 0;
29. }
30.
31. double ScalarProduct(double x[],double y[], int n) /* 内積の定義 */
32. {                        /* n でデータが入っている配列の数を指定する。*/
33.   int i;
34.   double sp=0.0;
35.
36.   for(i=0;i<n;i++) sp+=x[i]*y[i];
37.   return sp;
38. }
```

《プログラムの解説》

5 行目: 関数 ScalarProduct のプロトタイプ宣言

9 行目: a,b を倍精度実数型変数としてメモリー上に 10 個ずつ確保する。

12,13 行目: ベクトルの次元を標準入力から入力させる。

15〜23 行目: ベクトル a,b の各成分を入力する。

25 行目: 関数 ScalarProduct で内積の計算をさせ，得られた結果を sprd に代入させる。

26 行目: 結果の出力。

31〜37 行目: 関数 ScalarProduct の定義。

ここで，31 行目:ScalarProduct の引数が ”double x[], double y[], int n”となっている。これが，配列を関数に渡す方法である。このようにすると，関数呼び出し側 (今の場合 main 関数) から渡された配列の先頭のアドレスが x,y に入ることになる。

課題 6.5: ex0604.c を作成・コンパイル・実行し，期待通りの動作をするか確認せよ。

さて，関数の呼び出し側では，25 行目で”sprd=ScalarProduct(a,b,n)”としている。これをみると，関数に渡っているのは，アドレスではなく，配列に入っている数値そのもののように見える（確かに a,b は配列の名前なので，通常の変数の場合と全く同じである)。しかし，配列の場合ちょっと事情が異なっている。というのは，実は 25 行目の a,b は配列の名前であると同時にアドレスでもある。

```
double a[10];
```

と宣言したとき，a[0],...,a[9] は確かに変数なので，プログラム中に

```
a[2]=3.141592654;
```
と数値を代入することができるのに対して，
```
a=2.7182818;
```
とすることはできない。宣言文以外の場所に，配列として宣言した配列名（今の場合だと「a」）そのものがあるとき，それは a[0]〜a[9] までの配列の先頭のアドレスを意味する。したがって，ScalarProduct 関数の引数で，呼び出し側が関数に渡しているのは配列 a,b のアドレスなのである。関数側では，呼び出し側から渡されたものがアドレスであるように 31 行目の関数の引数の中で「double x[],double y[],...」として，待ち受けているのである。これは，文字列に関しても事情は全く同じである。

とにかく，配列を関数で渡すとき，関数の引数で通常の変数と同じように配列名をそのまま指定しているように見えるが，実はその配列の先頭のアドレスを指定している，という事がわかってもらえれば良い。

6.4　複数の戻り値を持つ関数

Call by value の方法では，関数の呼び出し側は関数から１つのデータしか受け取ることはできなかった。実際には関数にデータを渡して，複数の結果としてデータを受け取りたいことがある。§§ 6.3.1 の終わりで述べた，「２つのデータ a,b を関数に渡して，大きいほうのデータを c, 小さいほうのデータを d として呼び出し側に戻す」がその１つである。この場合も上のアドレス渡しの方法を利用する。関数の呼び出し側で，関数から渡される複数のデータの受け皿をあらかじめ用意しておいて，その受け皿のアドレスを関数側に渡すのである。

6.4.1　アドレスとポインタ

上で述べたように，配列だけでなく全ての変数はメモリー上にその値を記憶させるための場所が確保される。その場所には固有のアドレスが対応している。このアドレスを値とする変数をポインタという。通常の変数 a のアドレスを知るには「&a」のように変数の前に「&」をつける。これに対して，ポインタと呼ばれる変数 p は p そのものがアドレスの値をとる。また，ポインタ p はメモリー上のアドレスなので，そのアドレスにあるデータが記憶されている。このデータを知るには，ポインタ変数 p の前に「*」をつけて，*p として，p のアドレスに記憶されているデータを参照することができる。

先に変数のところで述べたように，各変数型に対応して，各変数のメモリー上に確保される記憶容量が異なる。したがって，ポインタに対しても各変数型に対応したポインタ変数の宣言をする必要がある。整数型のポインタ変数なら「int *p;」 倍精度実数型のポインタ変数なら「double *p;」，文字型のポインタ変数なら「char *p」のようにポインタ変数を宣言する。次の例を考えてみよう。整数型の変数 a,b はそれぞれ，アドレス「9801」番地と「9805」番地にデータ 100(a),200(b) が記憶されているとする。

```
1.  int a=100,b=200;
2.  int *p;
3.  p=&a;
4.  *p=20;
5.  p=&b;
6.  *p=30;
```

1 行目:　a,b を整数型の変数として宣言。初期設定として，a は 100,b は 200 としている。

2 行目:　整数型のポインタ変数 p を宣言。

3 行目:　p に a のアドレスを代入（p の矛先を a に向ける）。このとき p の値は a のアドレス「9801」番地をさしている。

4 行目:　p のアドレスで指定される場所にデータ 20 を記憶させる (3 行目で p は a のアドレスとなっていたので，4 行目の操作で，a に 20 が代入されることになる) 。

5 行目:　p に b のアドレスを代入（p の矛先を b に向ける）。このとき p の値は b のアドレス「9805」番地をさしている。

6 行目:　p のアドレスで指定される場所にデータ 30 を記憶させる (5 行目で p は b のアドレスとなっていたので，6 行目の操作で，b に 30 が代入されることになる) 。

上の例では，単に変数とアドレスの対応をポインタを使って見るだけの例で，通常のプログラミングではポ

インタをこのように使うことは殆どないと云って良い。いたずらにポインタを使うと，プログラムが複雑になり，何をしているのかわからなくなってしまうこともある。では，どのようなときにポインタを使うと良いのだろうか？次の例はこのセクションの本題である「複数の戻り値を持つ関数」について説明する。

6.4.2 複数の戻り値を持つ関数

複数の戻り値を持つ場合,「数値渡し (Call by value)」のように return 文で戻り値を呼び出し側に返すことはできない。この場合，関数の引数をポインタにして呼び出し側と関数側で共通のデータを操作することで，見かけ上複数のデータを返すようにする。次の例は「２つのデータ a,b を関数に渡して，大きいほうのデータを c, 小さいほうのデータを d として呼び出し側に戻す」関数 MaXMiN の例である。

List 6.4 ex0605.c

```
 1. /* ex0605.c 複数のデータを返す関数の例 */
 2. /* 作成者 xxxxxx */
 3. /* 作成日 2002/07/16 */
 4. #include <stdio.h>
 5. void MaXMiN(int a,int b,int *m, int *n);
 6. /* void 型関数としてプロトタイプ宣言*/
 7.
 8. int main()
 9. {
10.   int a,b,mx,mn;
11.
12.   printf("2 つ整数を入力してください。");
13.   scanf("%d %d",&a,&b);
14.   MaXMiN(a,b,&mx,&mn);                  /*  関数の呼び出し */
15.   printf("大きいのは %d で，小さいのは %d です。¥n",mx,mn);
16.   return 0;
17. }
18.
19. void MaXMiN(int a,int b,int *m, int *n) /* 関数の定義 */
20. {              /* m,n はポインタと呼ばれるアドレスを値とする変数 */
21.   if(a>=b){
22.     *m=a; /* "*m"は m で指定されたアドレスの値。*/
23.     *n=b;    /* a の値を m で指定したアドレスに記憶させる。*/
24.   }else{
25.     *m=b;
26.     *n=a; /* "*n"は n で指定されたアドレスの値。 */
27.   }          /* b の値を n で指定したアドレスに記憶させる。*/
28. }
```

《プログラムの説明》

5 行目: MaXMin 関数のプロトタイプ宣言。第 3・第 4 の引数がポインタであることを示している。

10 行目: a,b,mx,mn を整数型として宣言する。各変数のアドレスを「9801」,「9805」,「9809」,「9813」とする。

14 行目: MaXMiN 関数を main 関数 (呼び出し側) から実行している。呼び出し側は関数へ a,b の値と mx,mn のアドレス「9809」と「9813」とを渡している。関数が実行されると，mx,mn の示すメモリー上にデータが記録され，結果的に呼び出し側での mx,mn に値が代入される。

19〜28 行目: 関数 MaXMiN の定義。第３・第４の引数はポインタ変数で，関数の呼び出し側から渡されたアドレスが m,n に代入される。今の例では，m に「9809」n に「9813」のアドレスが代入される。ポインタ (アドレスを値とする変数) であることを明示するために，変数の宣言時に「*」がつけられている。

22–26 行目: 変数 a,b の値が m,n の示しているアドレス (「9809」と「9813」) 上に書き込まれる。「*m=a;」は「m で指定しているアドレスに記憶される値に a の値を代入」という意味である。したがって,「*m=a;」で a の値がポインタ m で指定されるアドレスを持つメモリーに記憶される。

課題 6.6: ex0605.c を作成・コンパイル・実行し

期待されるように動作しているか確認せよ。

ここで，13 行目の scanf の 2,3 番目の引数が「&a,&b」となっていることに注目しよう。これは a,b のアドレスが関数側に渡されていることを示している。scanf 関数では標準入力（キーボード）から入力された（複数の）値を関数の呼び出し側の指定したアドレスに値を返す関数である。そういえば,scanf で文字列を入力するときは,「char st[40];」と宣言されている文字列に対して，「scanf(”%s”,st);」とすればよいことになっていた。scanf の２つ目の引数 st は文字型の配列名ではなく,st[0]〜st[39] で確保されるメモリー上の先頭のアドレス（6.3.2 参照）だったのである。

6.5　標準的に定義されている主な関数

6.5.1　入出力関数 <stdio.h>

以下の関数をどれか１つでも利用するときは，stdio.h を必ずインクルードする (プログラムの始めのほうで #include <stdio.h>とする)。

printf 関数

int printf("出力書式", 出力変数);
標準出力（多くの場合画面）に指定した書式で指定した変数の値を出力する。この関数には戻り値があり，正常に処理されたときは出力された文字数が，またエラーが生じたときは”EOF(=-1)”が呼び出し側に返される。
%d,%f,%e,%s では%と d,f,s の間に数値を入れて出力する桁数を指定できる。実数の場合は%12.5f とすると，全体で 12 桁，小数点以下 5 桁の出力をすることができる。また，実数型の出力を%e で指定すると，仮数部と指数部の組み合わせで出力することが可能である。例えば%10.3e という書式で 12345.6789 を出力させると，1.2346e+04 のように出力される。これは，1.2346×10^4 の意味である。

printf 関数

```
1.  /* printf の使用例 */
2.   int a;
3.   float b;
4.   double c;
5.   char r;
6.   char str[100];
7.   ...
8.   printf("%d¥n",a);
9.     /* 整数型の出力 */
10.  printf("%f %f¥n",b,c);
11.    /*単・倍精度実数型の出力 */
12.  printf("%e %e¥n",b,c);
13. /*単・倍精度実数型の出力（指数表示） */
14.  printf("%c¥n",r);
15.    /* １つの文字の出力 */
16.  printf("%s¥n",str);
17.    /* 文字列の出力 */
```

scanf 関数

int scanf("入力書式", 入力変数);
標準入力（多くの場合キーボード）から指定した書式で指定した変数に値を入力する。この関数も整数型で定義された関数で，戻り値は正常に処理されたときは入力されたデータの数が，またエラーが生じたときは”EOF”が呼び出し側に返される。

scanf 関数

```
1.  int a;
2.  float b;
3.  double c;
4.  char str[10];
5.  ...          /*各データ型の入力*/
6.  scanf("%d",&a);/*整数型*/
7.  scanf("%f",&b);/*単精度実数型*/
8.  scanf("%lf",&c);/*倍精度実数型*/
9.  scanf("%s",str);/*文字列*/
```

fopen 関数

FILE *fopen("ファイル名","オープンモード");
FILE *fopen(文字列，"オープンモード");
”ファイル名”や文字列で指定したファイルを指定したモードでオープンし，戻り値としてファイルポインタを呼び出し側に返す。オープンできなければ戻り値として NULL が返される。オープンモードは表 6.1 のとおり。

表 6.1 オープンモード

モード	記号
読み込み	r
書き込み	w
追加書き込み	a

fclose 関数

int fclose(ファイルポインタ);
ファイルポインタで指定されているファイルを閉じる。処理の途中でエラーがでると戻り値に EOF が返される。正常に実行すると 0 が返される。

fopen,fclose 関数

```
1.  /* fopen の使用例 */
2.    FILE *fp; /* ファイルポインタ */
3.    char fname[20]="file.dat";
4.    ...
5.    fp=fopen(fname,"r");
6.  /* fname に記述されている名前の
7.  ファイルを読み込みモードでオープン */
8.    if(fp==NULL){
9.      printf("%s という",fname);
10.     printf("ファイルがない (;_;)¥n")
11.     exit(1);/* ファイルがないときは
12.   メッセージを出して終了 */
13.   }
14.   fclose(fp);
15.  /*ファイルを閉じる*/
```

fprintf 関数

int fprintf(ファイルポインタ,"出力書式", 出力変数);

ファイルポインタへ指定した書式で指定した変数の値を出力する。この関数には戻り値があり，正常に処理されたときは出力された文字数が，またエラーが生じたときは”EOF”が呼び出し側に返される。書式の指定は printf と同じ。

fprintf 関数

```
1.  /* fprintf の使用例 */
2.    FILE *fo; /* ファイルポインタ */
3.    int data;
4.    ...
5.    fo=fopen("result.dat","a");
6.    ...
7.    fprintf(fo,"%10d¥n",data);/*
8.    ...
9.    fclose(fo);
```

fscanf 関数

int fscanf(ファイルポインタ,"入力書式", 入力変数);

ファイルポインタから指定した書式で指定した変数に値を入力する。この関数も整数型で定義された関数で，戻り値は正常に処理されたときは入力されたデータの数が，またエラーが生じたときは”EOF”が呼び出し側に返される。

fscanf 関数

```
1.  /* fscanf, exit の使用例 */
2.    int a[10],i=0;
3.    double c[10];
4.    char fname[20]="file2.dat"
5.    FILE *fin;
6.    ...
7.    fin=fopen(fname,"r");
8.   /* fname に記述されている名前の
9.   ファイルを読み込みモードでオープン */
10.   if(fin==NULL){
11.     printf("%s という",fname);
12.     printf("ファイルはありません ¥n");
13.     exit(1); /* ファイルがないときは
14.              メッセージを出して終了*/
15.   }
16.
17.   while(i<10||fscanf(fin,"%d %lf",
18.           &a[i],&c[i])!=EOF){
19.     i++;
20.   }
```

getchar 関数

int getchar();

標準入力 (通常キーボード) から１文字を入力する。正常終了のときは読み込んだ文字，エラーのときは”EOF”を戻り値として返す。

putchar 関数

int putchar(整数型変数);

標準出力（通常は画面）へ１文字出力する。正常終了のときは読み込んだ文字 (putchar の引数)，エラーのときは”EOF”を戻り値として返す。

getchar,putchar 関数

```
1.  /* getchar,putchar の使用例 */
2.  #include <stdio.h>
3.  int main()
4.  {
5.    int c;
6.
7.    while((c=getchar())!=EOF&&c!='¥n'){
8.   /* ¥n は改行の記号．入力文字に対して
9.    エラーがなく改行記号でもないとき*/
10.     putchar(c);
11.   /* 取り込んだ文字を出力する */
12.   }
13.   return 0;
14. }
```

6.5.2　ユーティリティ関数 <stdlib.h>

以下の関数をどれか１つでも利用するときは，stdlib.h を必ずインクルードする (プログラムの始めのほうで「#include <stdlib.h>」とする)。

exit 関数

void exit(終了コード);

開いている全てのファイルを正常に閉じて，プログラムを終了させる。正常終了時は 0，異常終了時には 1 を引数の終了コードとする。

rand 関数

int rand(void);

0〜RAND_MAX(=32767) までの整数値の擬似乱数を返す。戻り値は擬似乱数。

srand 関数

void srand(負でない整数);

「負でない整数」で擬似乱数の発生系列を変える。

rand 関数

```
1.  /* --rand, srand の使用例 -- */
2.     int i;
3.     double x[100];
4.     srand(time(NULL));
5.
6.     for(i=0;i<100;i++)
7.         x[i]=2.0*rand()/RAND_MAX-1.0;
8.     /* -1 から 1 までの乱数を x[0] から
   x[99] に代入する */
```

6.5.3　文字列関数 <string.h>

以下の関数をどれか１つでも利用するときは，string.h を必ずインクルードする (プログラムの始めのほうで「#include <string.h>」とする)。

strcpy 関数

char *strcpy(文字配列１, 文字列２);

文字配列１に文字列２をコピーする。戻り値は，コピーされた文字配列１。

strcat 関数

char *strcat(文字配列１, 文字列２);

文字配列１の後に文字列２を連結する。戻り値は連結された文字配列１。

strcpy,strcat 関数

```
1.  /* strcpy,srtcat の使用例 */
2.     char st1[100],st2[100];
3.
4.     strcpy(st1,"私は");
5.     /* st1 に"私は"が代入される */
6.     strcpy(st2,"タコです。");
7.     /* st2 に"タコです。"が代入される*/
8.     strcat(st1,st2);
9.     /* st1 は "私はタコです。" となる */
```

6.5.4　数学関数 <math.h>

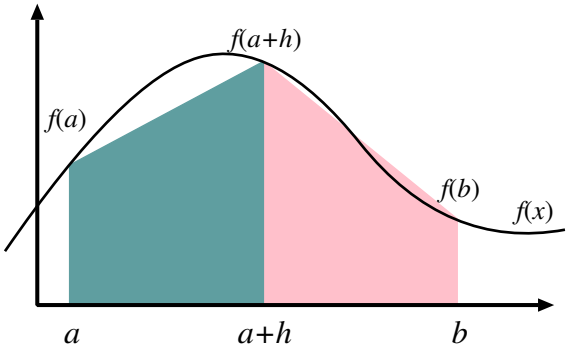
以下の関数をどれか１つでも利用するときは，math.h を必ずインクルードする (プログラムの始めのほうで「#include <math.h>」とする)。

表 6.2 math.h で定義されている主な関数

関数名	意味	引数の型	引数の値	戻り値の型と値
fabs(x)	x の絶対値	double	any	double
sin(x)	$\sin(x)$	double	any(ラジアン)	double
cos(x)	$\sin(x)$	double	any(ラジアン)	double
tan(x)	$\cos(x)$	double	any(ラジアン)	double
asin(x)	$x = \sin(y)$ を満たす y	double	$-1 \sim 1$	double($-\pi/2 \sim \pi/2$)
acos(x)	$x = \cos(y)$ を満たす y	double	$-1 \sim 1$	double($0 \sim \pi$)
atan(x)	$x = \tan(y)$ を満たす y	double	any	double($-\pi/2 \sim \pi/2$)
atan2(x,y)	$\frac{y}{x} = \tan(z)$ を満たす z	double,double	any,any	double($-\pi \sim \pi$)
sinh(x)	$\frac{1}{2}(e^x - e^{-x})$	double	any	double
cosh(x)	$\frac{1}{2}(e^x + e^{-x})$	double	any	double
tanh(x)	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	double	any	double
exp(x)	e^x	double	any	double
log(x)	自然対数	double	$x > 0$	double
log10(x)	常用対数	double	$x > 0$	double
pow(x,y)	x^y	double	$x > 0$,y:any	double
sqrt(x)	$\sqrt{x}$	double	$x > 0$	double
hypot(x)	$\sqrt{x^2 + y^2}$	double,double	any,any	double

6.6　積分しよう!

ここまでのところで，大方のプログラムを書くことができるはずである。そこで，簡単なプログラムの例として，数値積分を取り上げてみよう。



計算機に積分をさせる方法はいくつもあるが，ここでは最も基本的な方法である「台形則」を紹介する。上図のような曲線の $x = a$ から $x = b$ までの積分を考える。$h = (b - a)/2$ とすると， a から $a + h$ までの曲線と x 軸の囲む面積は

$$(a, 0),\ (a, f(a)),\ (a + h, f(a + h)),\ (a + h, 0)$$

で作られる台形の面積で近似できる。したがってこの区間の面積は

$$S_1 = \frac{1}{2}[f(a) + f(a + h)]h$$

と表 せる。同様に，区間 $a + h$ から b までの面積は

$$(a + h, 0),\ (a + h, f(a + h)),\ (b, f(b)),\ (b, 0)$$

で作られる台形の面積

$$S_2 = \frac{1}{2}[f(a + h) + f(b)]h$$

と近似できる。したがって，$x = a$ から $x = b$ までの曲線の積分は

$$S = S_1 + S_2 = \left[\frac{1}{2}f(a) + f(a + h) + \frac{1}{2}f(b)\right]h$$

と近似できる。これが台形則である。ここでは積分区間を２つに分割したときの積分値の近似値を求めたが，この分割が多くなればなるほど近似値は真の値に近くなる。積分区間を 2^N 個に分割した場合の近似値は

$$\begin{aligned} S &= \left[\frac{1}{2}f(a) + f(a + h) + f(a + 2h) + \cdots \right. \\ &\qquad \left. + f(a + (2^N - 1)h) + \frac{1}{2}f(b)\right]h \\ &= \left[\frac{1}{2}f(a) + \sum_{k=1}^{2^N-1} f(a + kh) + \frac{1}{2}f(b)\right] \end{aligned}$$

と表せる。ここで $h = (a - b)/2^N$ である。

以下のプログラムは $\dfrac{4}{1 + x^2}$ を 0 から 1 まで積分するプログラムである。このプログラムでは，main 関数で分割点を 2 倍づつ増やして，その結果を画面と指定したファイルに書き込ませるプログラムである。

課題 6.7:　次の `trape.c` を作成・コンパイル・実行し，得られた結果が円周率に近くなっているかを調べよ。

課題 6.8:　`trape.c` の各行が何を行っているか調べよ。

課題 6.9:　別の関数を積分するときは，`trape.c` のどこを変更すればよいか？別の積分についても正しく計算しているかを調べよ。

List 6.5 trape.c

```
 1. #include <stdio.h>
 2. #include <math.h>
 3.
 4. double func(double x);
 5. double Sekibun(double a,double b,double h);
 6.
 7. int main()
 8. {
 9.   double h,a,b,S;
10.   char fname[40];
11.   FILE *fout;
12.   int i=0,N=1;
13.
14.   printf("ファイル名を入力してください-->");
15.   scanf("%s",fname);
16.   fout=fopen(fname,"w");
17.
18.   a=0.0;
19.   b=1.0;
20.
21.
22.   h=b-a;
23.   while(i<20){
24.     i++;
25.     h*=0.5;
26.     N*=2;
27.     S=Sekibun(a,b,h);
28.     printf("%2d %7d %10.5e %20.16f %15.10e¥n",i,N,h,S,1.-S/M_PI);
29.     fprintf(fout,"%2d %7d %10.5e %20.16f %15.10e¥n",i,N,h,S,1.-S/M_PI);
30.   }
31.   fclose(fout);
32.   return 0;
33. }
34.
35. double Sekibun(double a,double b,double h)
36. {
37.   double x,sum;
38.
39.   x=a;
40.   sum=0.5*(func(a)+func(b));
41.   while((x+=h)<b) sum+=func(x);
42.   sum*=h;
43.   return sum;
44. }
45.
46. double func(double x)
47. {
48.   return 4.0/(1.0+x*x);
49. }
```

付録 A

最後に

これまでで，一通り C 言語によるプログラミングについて説明してきたが，時間不足や言葉足らずで説明不足だったところが多々あった。現在のところ「C 言語の初歩」といった程度のことしか説明ができなかった。残念ながら，この授業の中で「ポインタ」,「構造体」,「プリプロセッサ」などの説明ができなかった。各自自学して頂きたい。

この授業を機に，もっと C 言語を勉強したいと考える人は，各自参考書を購入し，とにかく１冊読破するのが良いかと思われる。そこで，以下に参考となりそうな本を２～３挙げることにする。

①「プログラミング言語 C」第 2 版 ANSI 規格準拠 (共立出版株式会社)
B.W. カーニハン, D.M. リッチー著 石田　晴久訳
この本は C 言語のバイブルとされている本なので，これから C 言語を使っていきたいという人に必須の 1 冊である。

②「改訂　新　C 言語入門　ビギナー編」 (SOFTBANK)
林　晴比古　著
今回の授業を進めるにあたって参考とした本。一通りの C 言語の解説がなされている。

③「改訂　新　C 言語入門　シニア編」 (SOFTBANK)
林　晴比古　著
ビギナー編をマスターしたあとに，C 言語を体系的に理解できるように解説がなされた本。

④「改訂　新　C 言語入門　スーパービギナー編」(SOFTBANK)
林　晴比古　著
「ビギナー編でもちょと」という読者を対象に， C 言語の手短な理解を目指した解説本。

C 言語に関する本は，まだまだたくさんあるので，是非書店に行って自分で手にとって，自分にあった本を探してみのが良い。

また，授業の冒頭で，インターネットで「フリーのコンパイラが手に入る」といったが，これは Borland 社の Web ページ上にある。URL は

http://www.borland.co.jp/cppbuilder/freecompiler

である。ここの指示に従って，名前等を登録すると

freecommandlinetools2.exe

という名前の約 8MB のファイルがダウンロードされる。ファイルが大きいので，ダウンロードするとき注意が必要である。自宅などで電話回線を使っている場合では，ダウンロードするのに時間が非常にかかるし，大学内の LAN からインターネットへつなぐときは，大学内で許されている使用容量の制限（30MB）に引っかかる人もいるかもしれない。注意が必要である。

付録 B

期末試験対策

【1】　次のプログラム tst01.c について設問に答えよ。

List B.1 tst01.c

```
1.  /* tst01.c */
2.  #include <stdio.h>
3.
4.  int main()
5.  {
6.    printf("Hello, World¥n");
7.    return 0;
8.  }
```

問　このプログラムの出力結果を示し，更に 6 行目を

① printf(”Hello,”);printf(” World¥n”);

② printf(”Hello,¥nWorld¥n”);

と書き換えたとき，もとの出力結果とどこが異なるかを①，② についてそれぞれ説明せよ。同じ場合は「同じ」のみで良い。

【2】　次のようなプログラム tst02.c を作成した。以下の設問に答えよ。

List B.2 tst02.c

```
 1.  /* tst02.c */
 2.  #include <stdio.h>
 3.
 4.  int main()
 5.  {
 6.    int x,y;
 7.
 8.    printf("x を入力してください a");
 9.    scanf("%d",&x);
10.
11.    y=x*x+2*x+1;
12.    printf("x が%d のとき，y=x*x+2*x+1 は  %d です ¥n",x,y);
13.    return 0;
14.  }
```

問１　このプログラムは何をするプログラムか説明せよ。

問２　このプログラムの 6 行目の「int」を「double」と変えたとき，プログラムは正常に動作しなかった。その理由を述べ，正しく動作するためにどこをどのように修正したらよいかについても答えよ。

【3】 次のプログラム tst03.c の出力はどのようになるか答え，その理由も述べよ。

List B.3 tst03.c

```
 1.  /* tst03.c */
 2.  #include <stdio.h>
 3.
 4.  int main()
 5.  {
 6.    int a,b,c;
 7.    double x;
 8.
 9.    c=5;
10.    x=2.50;
11.
12.    a=c/2/x;
13.    b=c/x/2;
14.
15.    printf("a=%d b=%d¥n",a,b);
16.    return 0;
17.  }
```

【4】 次のプログラム tst04.c について設問に答えよ。

List B.4 tst04.c

```
 1.  /* tst04.c */
 2.  #include <stdio.h>
 3.
 4.  int main()
 5.  {
 6.    int s,n,i;
 7.
 8.    i=0;
 9.    s=0;
10.
11.    printf("Input n -->");
12.    scanf("%d",&n);
13.
14.    while(i<=n){
15.      s+=i;
16.      i++;
17.    }
18.    printf("%5d %5d¥n",n,s);
19.    return 0;
20.  }
```

問１ このプログラムは何をするプログラムか説明せよ。

問２ このプログラムをコンパイルし実行すると，「Input n -->」と表示される。これに対してキーボードから 10 と入力すると，出力結果はどのようなるか答えよ。

問３ このプログラムを for 文を使って書き換えよ。

【5】　次のプログラム tst05.c の誤りを指摘し，これを正せ。

List B.5 tst05.c

```
 1.  /* tst05.c */
 2.  #include <stdio.h>
 3.  int main()
 4.  {
 5.    int degree, max_degree;
 6.    double radian, PI=3.141592654;
 7.
 8.    printf("何度まで計算しますか？);
 9.    scanf("&d",max_degree);
10.
11.    for(degree=0;degree<=max_degree;degree=+10){
12.      radian = degree*PI/180;
13.      printf(%4d  8.3f¥n,degree,radian);
14.    }
15.    return 0;  /* 終了 */
16.  }
```

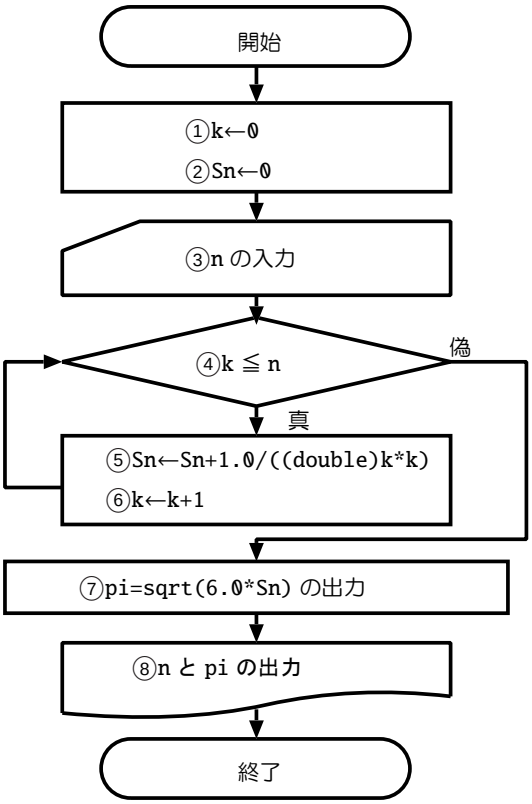
【6】　右のフローチャートをもとに，1 から入力した n までの逆数の 2 乗の和 を求め，得られた結果を 6.0 倍して平方根をとった値 (pi とする) を，n と共に表示するプログラム tst06.c を作成せよ。

《ヒント》

- このプログラムで使用する変数は k,n(整数型)，Sn,pi(倍精度実数型) である。
- 平方根をとる関数は sqrt 関数で math.h で定義されている。プログラムの始めのところで，

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

  とする。（sqrt の使い方は⑦を参照。）
- 1.0/((double)k*k) を Sn に加えていく。



【7】　倍精度実数型の変数 a の値をキーボードから入力して，その絶対値を出力するプログラム tst07.c を if 文を使って作成せよ。

《ヒント》

入力した a が負だったら-a を，正かゼロだったら a を出力するプログラムを作ればよい

【8】 表 1 はある中学生の１，２，３学期の各教科の成績のデータである。このデータが「seiseki.dat」というファイルに保存されているとする。次のプログラム tst08.c はこのファイルを読み込み，読み込んだとおりに画面に出力するプログラムである。空欄① ～⑭ を適切に埋め tst08.c を完成させよ。

List B.6 tst08.c

```
 1.  /* tst08.c */
 2.  #include <stdio.h>
 3.  #include <stdlib.h>
 4.  #define N 5
 5.
 6.  int main()
 7.  {
 8.     FILE *fp;
 9.     char kyouka[N][10];
10.     int i ;
11.     int a[N],b[N],c[N];
12.
13.     fp=fopen( ① , ② );
14.     if( ③ ){
15.       printf("ファイルがありませんでした");
16.       exit(1);
17.     }
18.
19.     for(i=0;i<N;i++)
20.       fscanf( ④ ," ⑤ ", ⑥ , ⑦ , ⑧ , ⑨ );
21.     fclose( ⑩ );
22.
23.     for(i=0;i<N;i++)
24.        printf("%s %2d %2d %2d¥n", ⑪ , ⑫ , ⑬ , ⑭ );
25.     return 0;
26.  }
```

表 1. seiseki.dat

国語	42	45	93
数学	40	60	53
理科	85	37	63
社会	77	95	88
英語	19	68	46

【9】 表 2 はある中学生の１，２，３学期の各教科の成績のデータである。このデータが「seiseki.dat」というファイルに保存されているとする。次のプログラム tst09.c はこのファイルを読み込み，各教科の合計点と平均値を計算し，結果を表 3 のように画面と「kekka.dat」というファイルに出力するプログラムである。空欄 ① ～⑯ を適切に埋め tst09.c を完成させよ。

List B.7 tst09.c

```
 1.  /* tst09.c */
 2.  #include <stdio.h>
 3.  #include <stdlib.h>
 4.  #define N 5
 5.
 6.  int main()
 7.  {
 8.    FILE *fin,*fout;
 9.    char kyouka[N][10];
10.    int i;
11.    int a[N],b[N],c[N],total;
12.    double average;
13.
14.    fin=fopen( ① , ② );
15.    if( ③ ){
16.      printf("ファイルがありませんでした");
17.      exit(1);
18.    }
19.
20.    for(i=0;i<N;i++)
21.      fscanf( ④ ," ⑤ ", ⑥ , ⑦ , ⑧ , ⑨ );
22.    fclose( ⑩ );
23.
24.    fout=fopen("kekka.dat", ⑪ );
25.    for(i=0;i<N;i++) {
26.       total= ⑫ ;
27.       average=total/3.0;
28.       printf("%s %2d %2d %2d %3d %7.2f¥n", ⑬
29.         ,a[i],b[i],c[i],total,average);
30.      fprintf( ⑭ ,"%s %2d %2d %2d %3d %7.2f¥n", ⑮
31.              ,a[i],b[i],c[i],total,average);
32.    }
33.    fclose( ⑯ );
34.
35.     return 0;
36.  }
```

表 2. seiseki.dat

国語	42	45	93
数学	40	60	53
理科	85	37	63
社会	77	95	88
英語	19	68	46

表 3. kekka.dat

国語	42	45	93	180	60.00
数学	40	60	53	153	51.00
理科	85	37	63	185	61.67
社会	77	95	88	260	86.67
英語	19	68	46	133	44.33

【10】 表 4 はある中学生の１，２，３学期の各教科の成績のデータである。このデータが「seiseki.dat」というファイルに保存されているとする。次のプログラム tst10.c はこのファイルを読み込み，各教科の合計点が 240 点以上では'A'，210 以上では'B'，180 以上では'C'，180 未満では'D' と判定した結果を表 5 のように画面と「kekkaABC.dat」というファイルに出力するプログラムである。空欄① ～⑲ を適切に埋め tst10.c を完成させよ。

List B.8 tst10.c

```
 1.  /* tst10.c */
 2.  #include <stdio.h>
 3.  #include <stdlib.h>
 4.  #define N 5
 5.
 6.  char Hantei(int a,int b,int c);
 7.  int main()
 8.  {
 9.    FILE *fin,*fout;
10.    char kyouka[N][10],ABCD;
11.    int a[N],b[N],c[N],i;
12.
13.    fin=fopen( ① , ② );
14.    if( ③ ){
15.      printf("ファイルがありませんでした");
16.      exit(1);
17.    }
18.
19.    for(i=0;i<N;i++) fscanf( ④ ," ⑤ ", ⑥ , ⑦ , ⑧ , ⑨ );
20.    fclose( ⑩ );
21.
22.    fout=fopen("kekkaABC.dat", ⑪ );
23.    for(i=0;i<N;i++) {
24.      ABCD=Hantei(a[i],b[i],c[i]);
25.      printf("%s %2d %2d %2d %c¥n", ⑫ ,a[i],b[i],c[i],ABC);
26.      fprintf( ⑬ ,"%s %2d %2d %2d %c¥n", ⑭ ,a[i],b[i],c[i],ABC);
27.    }
28.    fclose( ⑮ );
29.    return 0;
30.  }
31.
32.  char Hantei( ⑯ )
33.  {
34.    int total;
35.
36.    total=a+b+c;
37.    if(total>=240) return 'A';
38.    else if(total>=210) return ⑰ ;
39.    else if(total>=180) return ⑱ ;
40.    else return ⑲ ;
41.  }
```

表 4. seiseki.dat

国語	42	45	93
数学	40	60	53
理科	85	37	63
社会	77	95	88
英語	19	68	46

表 5. kekkaABC.dat

国語	42	45	93	C
数学	40	60	53	D
理科	85	37	63	C
社会	77	95	88	A
英語	19	68	46	D

【11】 「プログラムが始まりました。」を画面に出力する関数 Start と，「プログラムが終了しました。」を画面に出力する関数 End を作成し，次のプログラム tst11.c を空欄① ～⑩ を適切に埋めて完成させよ。

List B.9 tst11.c

```
 1.  /* tst11.c */
 2.  #include <stdio.h>
 3.
 4.  [①]  Start([②]);
 5.  [③]  End([④]);
 6.  int main()
 7.  {
 8.    Start();
 9.    End();
10.
11.    return 0;
12.  }
13.
14.  [⑤]  Start([⑥])
15.  {
16.    [⑦];
17.  }
18.
19.  [⑧]  End([⑨])
20.  {
21.    [⑩];
22.  }
```

【12】 倍精度実数型の２つの変数 a,b を引数として，a と b の和および差を呼び出し側に返す関数 Wa_Sa を作成せよ。

【13】 0 から 1 までの区間をキーボードから入力した整数 n で等分割し，台形則を使って下の関数を積分するプログラム tst14.c を作成せよ。

①$3x^2$

②$4\sqrt{1-x^2}$

【14】 自分で作成した関数を１つ以上使い，出力結果をファイル (ファイル名は任意) に保存するプログラムを作成せよ

【15】 整数 n を引数として，＊印を横に n 個出力する関数 Star を作成し，表 6 のデータファイル sakana.dat を読み込んで各項目に対して＊印の棒グラフを出力するプログラム tst12.c を空欄① ～⑦ を適切に埋め完成させよ。

List B.10 tst12.c

```
 1.  /* tst12.c */
 2.  #include <stdio.h>
 3.  #include <stdlib.h>
 4.  #define N 5
 5.
 6.  void Star( ① );
 7.  int main()
 8.  {
 9.    FILE *fin;
10.    char fish_name[N][10];
11.    int i,num[N];
12.
13.    fin= ② ;
14.    if( ③ ){
15.      printf("ファイルがありませんでした (;_;)");
16.      exit(1);
17.    }
18.
19.    for(i=0;i<N;i++)
20.      fscanf( ④ );
21.    fclose( ⑤ );
22.
23.    for(i=0;i<N;i++) Star(fish_name[i],num[i]);
24.    return 0;
25.  }
26.
27.  void Star( ⑥ )
28.  {
29.    int i;
30.
31.    printf(%10s ",name);
32.    for(i=0;i<n;i++) printf( ⑦ );
33.    printf("¥n");
34.  }
```

表 5. sakana.dat

アジ	20
サンマ	30
イワシ	37
シャケ	30
サバ	25