C 言語入門（基礎編）

2009-04-28 熊谷知久、泉 聡志

1. 実行するには？

C プログラム hoge.c を書いたら,

>*gcc hoge.c*　　(数値計算の場合は>*gcc –lm hoge.c*)

のようにコンパイルを行う．この場合, a.out という実行ファイルが作られる．

実行ファイルは

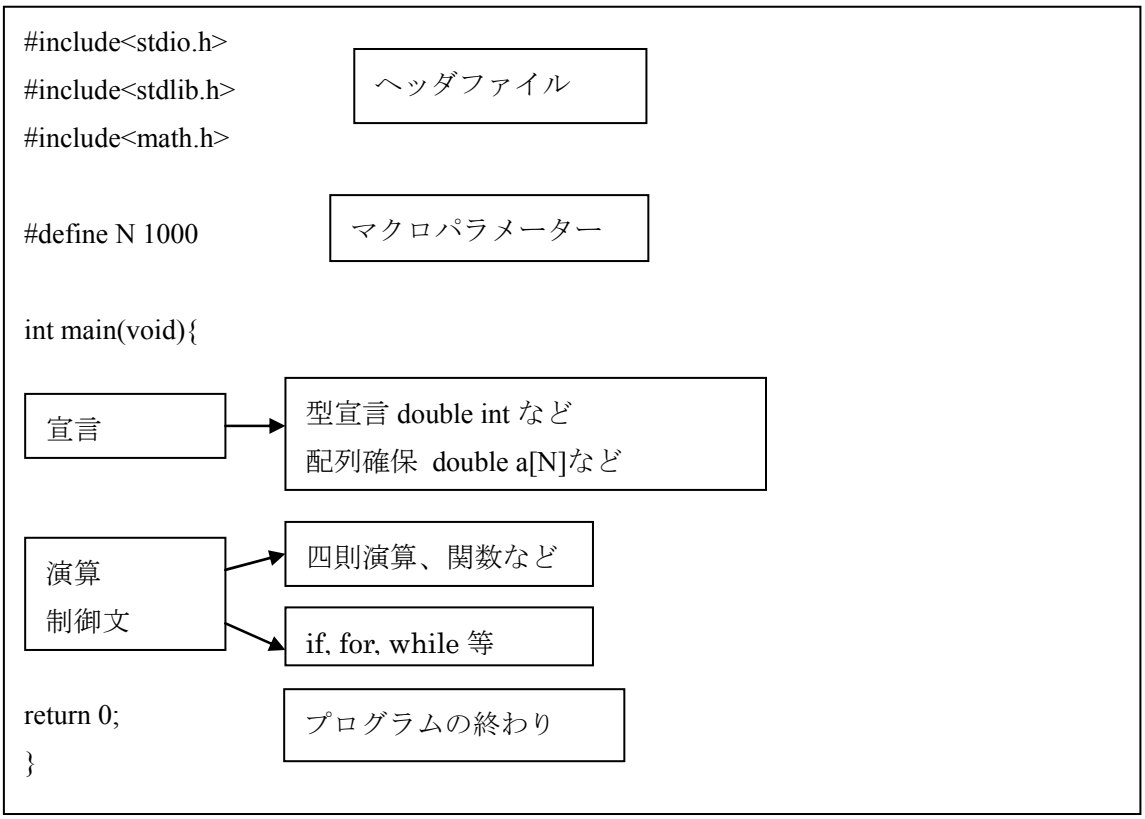
>*./a.out*

とすれば実行できる．

2. 単純なCプログラムの形式

- プログラムの冒頭に，必要なヘッダファイルをインクルードする（書いておく）必要がある．一般的に使われるのは stdio.h, stdlib.h, math.h, string.h 程度であり、面倒なら全て書いておけばよい．math.h をインクルードした場合，コンパイルオプションに-lm が必要な場合があるので注意（常につけていても問題はない）．
- #define から始まるマクロパラメータ（プログラム中で共通で規定の数）も冒頭で宣言するのが一般的である．
- 基本的に main という名前の関数の中身が実行される．
- main 関数の中身は宣言，演算&制御文という順番で書く．
- 制御文は中括弧{}でくくられる．
- 一つの単位を文と呼ぶが，文の最後には;(セミコロン)を書かねばならない（改行しただけでは文とはならない）．
- 小文字と大文字とは別の文字として認識される
- /*から*/までがコメントアウト（プログラムと認識されない）される．行の//以降もコメントアウトされる．
- プログラムを途中で終了したい場合には終了させた居場所に exit(0);と書いておく(要 stdlib.h).
- void は「空」というような意味で返り値をもたない関数や引数がない場合に用いられる．

単純な C プログラムの典型例



・

3. 宣言・定数

3.1. マクロ

プログラムの冒頭で定義することの多いマクロは以下のように定義する

```
#define マクロの名前 値
```

マクロは実数でも整数でもよいが, 演算中に変更することはできない. 例えば

```
#define N 1000
#define PI 3.1415926535
```

とすればプログラム中の N はすべて 1000 として, PI はすべて 3.14…としてあらかじめ変換される.

3.2. 変数の型

整数型は int, 実数型は double を使うと良い. 宣言は以下のように行う.

```
int i,j,k;
double a,b,c;
```

3.3. 配列

後の 4.1.のように, a,b,c…と多くの変数を定義するのは面倒なので, それらを a[0], a[1], a[2]という 1 次元配列で定義する. 例えば, 配列のサイズが 3 の実数型の場合 double a[3]と宣言する. 整数型で定義したい場合は int a[3]のように宣言する. ここで配列のサイズより 1 だけ小さい配列要素までしか使えないことに注意が必要である. 即ち配列サイズが 3 の場合, a[0], a[1],

a[2]を利用することができる. 配列のサイズは#define で定義されるマクロ変数を使う場合が一般的である. また, 多次元の配列を利用することもできる. 例えば 2 次元配列の場合, 宣言は a[2][3]のように行い, 変数 a[0][0], a[0][1], a[0][2], a[1][0], a[1][1], a[1][2]を使うことができる. 3 次元以上の多次元配列も可能である.

```
#define M 100
#define N 1000
…..
double a[3],b[M],c[M][N];
int i[3],j[M],k[M][N];
```

3.4. 定数

定数の表現を以下の表にまとめる. 指数表記も可能である.

定数	C における表現
整数　5	*5*
実数　5.0	*5.0*
実数　5.0×10^{6}	*5.0e6,5.0e+6*
実数　5.0×10^{-6}	*5.0e-6*
実数　0.001	*1.0e-3*
実数　1000.0	*1.0e+3*
実数　0.001	*1.0e-3*
実数　1000.0	*1.0e+3*

4. 演算子

色々あるが, よく使うものを紹介する.

4.1. 代入演算子

=(イコール)のことである. =の左側の変数に=の右側の値を代入する.

以下にサンプルを示す.

```
int i,j;
double a,b,c;
i=10;       //i に 10 を代入
b=3.0;     //b に 3.0 を代入
c=7.0;     //c に 7.0 を代入
j=i:         //j に i の値を代入
a=b+c;   //a に b+c の値を代入
i=i+5;     //i の値を 5 増加
```

4.2. 算術演算子

四則演算と余りを計算するための演算子.

算術演算	C
加算	＋
減算	－
乗算	*
除算	/
余り	%

4.3. インクリメント, デクリメント演算子

インクリメント演算子(++)は整数型変数を 1 増加させる演算子,デクリメント演算子(--)は 1 減少させる演算子. 以下に使用例を示す.

```
int i;
i=0;
i++;
i--;
```

変数の前に付けるか, 後につけるかで演算されるときが異なるが, 最初は取りあえず後ろにつけておけば問題ない. インクリメント演算 i++は i=i+1 と同じで, デクリメント演算 i--は i=i-1 と同じである. 後述する for 文で頻繁に用いられる.

4.4. 関係演算子

論理式は真(true)か偽(false)の 2 値を持つ. もっとも多い使われ方は数の関係演算子による大小比較である. 以下に代表的な使い方を示す. if 文, for 文に用いられる.

A＜B	A が B より小さければ真, それ以外偽
A<=B	A が B 以下なら真, それ以外偽
A==B	A と B が等しければ真, それ以外偽
A!=B	A と B が異なれば真, それ以外偽
A＞B	A が B より大きければ真, それ以外偽
A>=B	A が B 以上なら真, それ以外偽

4.5. キャスト演算子

一般的に型が違う変数の算術演算は行わないが, 型が違う変数の演算が必要なケースは非常に多い. この場合, キャスト演算子が用いられる. 多くの場合, コンパイラが勝手に変換(キャスト)してくれるため, 気づかない場合も多いが, バグの原因となることも少なくないので明示的に書いたほうが安全である. 書式は以下のようである.

```
(変数型)キャストする値
```

以下に整数型を実数型に変換(キャスト)する使用例を示す.

```
int i;
double a;
i=5;
a=3.0*(double)i;
```

以下に実数型を整数型に変換(キャスト)する使用例を示す.

```
int i;
double a;
a=5.8;
i=(int)a;
```

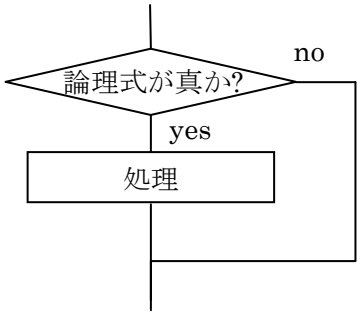
この場合には注意が必要である. a は切捨てされ, i は 5 となる. これは良くバグの原因となる.

5. 制御文

5.1. if文

“もし～が～だったら、～する”というように、論理的判断を行う. さまざまな使い方がある. もっとも最も基本的な書式は

```
if(論理式){
   処理
}
```



である. 論理式が真なら処理が実行され, 偽ならスキップされる. 論理式には例えば前節で述べた関係演算子を用いる. 対応するフローチャートも示す.

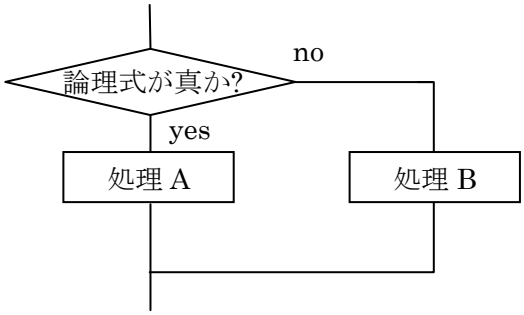
以下に使用例を示す.

```
int i;
double x;
～中略～
if(i==0){
   x=x+3.0;
}
```

これは i がゼロだった場合 x の値を 3 増加させる処理である.

以下の書式の場合,

```
if(論理式){
   処理 A
}else{
   処理 B
}
```



論理式が真なら処理 A が実行され, 偽なら処理 B が実行される.
以下に使用例を示す.

```
double x,y;
～中略～
if(x<0.0){
   y=-x;
}else{
   y=x;
}
```

これは x がゼロ未満だった場合, y に-x を代入し, それ以外の場合(つまり x がゼロ以上), y に x を代入する処理である.

以下の書式の場合

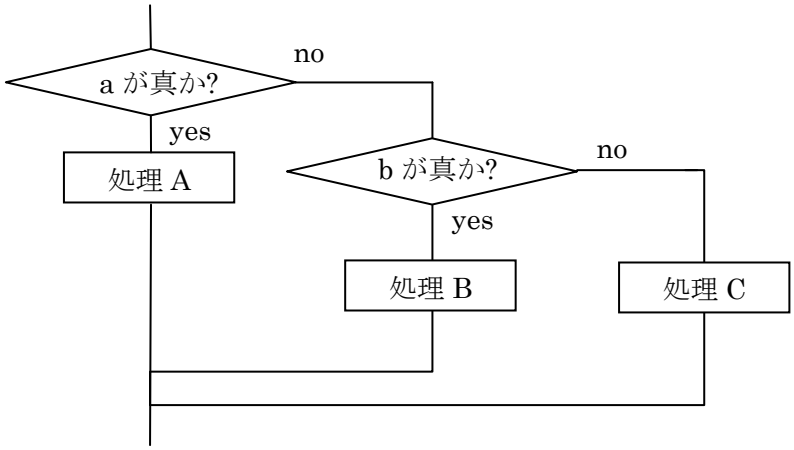
```
if(論理式 a){
   処理 A
}else if(論理式 b){
   処理 B
}
```

論理式 a が真なら処理 A が実行され, 論理式 a が偽で論理式 b が真の場合, 処理 B が実行される. 論理式 a が偽, 論理式 b が偽の場合スキップされる.

以下の書式の場合

]

```
if(論理式 a){
処理 A
}else if(論理式 b){
処理 B
}else{
処理 C
}
```



論理式 a が真なら処理 A が実行され, 論理式 a が偽で論理式 b が真の場合, 処理 B が実行され, 論理式 a が偽, 論理式 b が偽の場合, 処理 C が実行される. else if(…)はいくらでも挿入することができる. 対応するフローチャートも示す。

以下に使用例を示す.

```
int i;
double x;
～中略～
if(x<0.0){
   i=0;
}else if(x<1.0){
   i=1;
}else if(f<2.0){
   i=2:
}else{
   i=3;
}
```

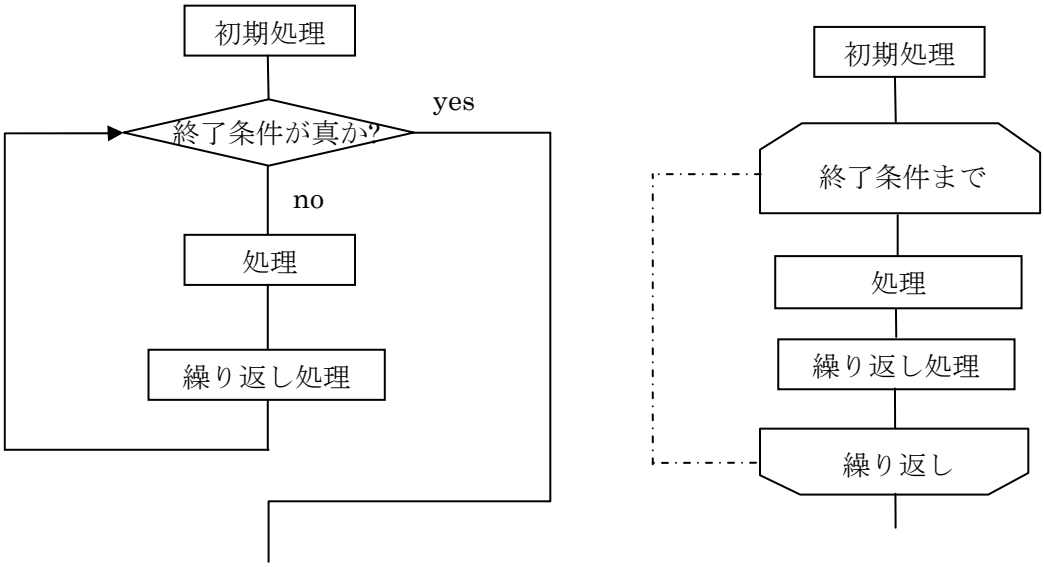
これは x が 0 未満のとき, i が 0, x が 0 以上 1 未満のとき, i が 1, x が 1 以上 2 未満のとき, i が 2, それ以外(x が 2 以上)のとき i=3 となる処理である.

5.2. for文

処理を繰り返し行うために使用する. まず, 初期処理を行い, 論理式で表現される終了条件を満たさないか確認した後, for 文の中身を実行する. for 文の中身を実行した後, 繰り返し処理を行い, 終了条件になっていないか確認し, 終了条件を満たさない場合, for 文の中身を再度実行する. 終了条件を満たす場合は for 文の次の処理を行う. for 文は終了条件を満たすまで実行される. for 文の書式は以下のようである.

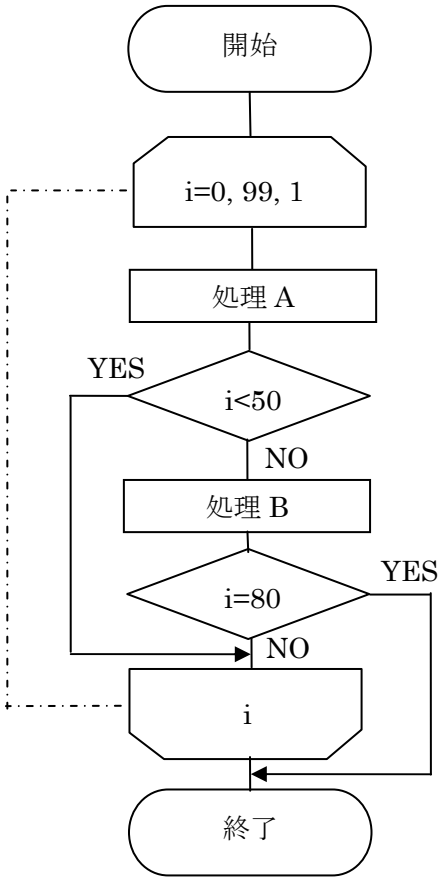
以下に対応するフローチャートを二通り示す.

```
for(初期処理; 終了条件; 繰り返し処理){
   処理
}
```



for 文の中で break が実行されると，強制的に for 文から脱出する．continue が実行されると，for 文の冒頭に戻り，繰り返し処理，終了条件判定が行われる．以下に使用例を示す．

```
for(i=0;i<100;i++){
   処理 A
   if(i<50){
   continue;//処理 B 以下はスキップ
   }
   処理 B
   if(i==80){
   break;//loop 終了
   }
}
```



6. 標準入出力

6.1. scanf文

キーボードからの入力を変数に入れる．書式は以下のようである

整数型の場合,

```
scanf(“%d”,&i);
```

であり，キーボード入力が整数型変数 i に入る．

実数型の場合

```
scanf(“%lf”,&a);
```

であり，キーボード入力が実数型変数 a に入る．

&を忘れないように気をつけてほしい．

6.2. printf文

ディスプレイに出力を行う．書式は以下のようである

整数型の場合,

```
printf(“%d¥n”,i);
```

であり，ディスプレイに整数型変数 i の値が出力される．

実数型の場合

```
printf(“%lf¥n”,a);
```

であり，ディスプレイに実数型変数 a の値が出力される. ¥n は改行記号である.
文字列との組み合わせや複数出力も可能である.

```
printf(“i=%d, a=%lf¥n”,i,a);
```

とすれば，文字列と同時に整数型変数 i,実数型変数 a が出力される.

7. 組み込み数学関数

- 代表的なものを載せる. 後述する通常の関数と同様の使い方をする. 使うためには math.h をインクルードする必要がある. その場合，コンパイルオプションに-lm が必要な場合があるので注意.

数学関数	表現
べき乗 a^3	*pow(a,3)*
sin, cos	*sin( ) , cos( )*
平方根√	*sqrt( )*
絶対値(実数)	*fabs( )*
指数 e^x	*exp( )*

C 言語入門（応用編）

2009-04-28 熊谷知久、泉　聡志

1. 実行するには

>gcc –lm hoge.c –o hoge

実行ファイルを hoge という名前にする.

>./hoge

で実行できる.

2. 文字型，文字列型

2.1. 文字型

文字型は char 型である．文字型は以下の書式で宣言を行う.

```
char c;
```

定数は 1 重括弧で括る.

```
c='L';
```

標準入出力は%c で行う.

```
scanf("%c",&c)
printf("%c¥n",c);
```

2.2. 文字列型

文字列型は char 型の配列で表現する．このため宣言は以下のようになる.

```
char s[100];
```

ただし, 配列の長さより長い文字列は取り扱えない.

文字列型の変数 s として考えても差し支えない使い方ができる.

標準入出力は%s で行う.

```
scanf("%s",s)
printf("%s¥n",s);
```

scanf の変数に&がないことに注意が必要.

操作は string.h に含まれる関数で行うことが多い(必ず string.h をインクルードすること).

例えば, 変数のコピーは strcpy で行う.

```
strcpy(コピー先変数,コピー元変数);
strcpy(コピー先変数,"任意の文字列");
```

任意の文字列は 2 重括弧で括る.

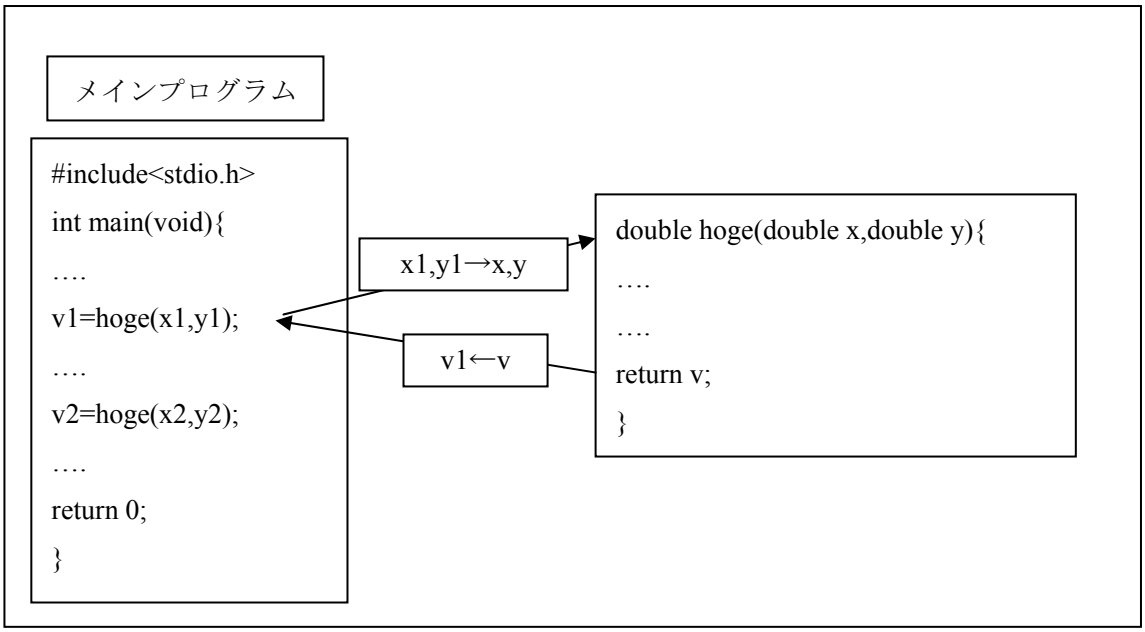
例えば以下のように用いる.

```
char s1[100],s2[100]
strcpy(s1,”abcd”);
strcpy(s2,s1);
```

他にも文字列比較を行う strcmp が良く用いられる.

3. 関数

プログラム中, 似たような処理を繰り返す部分を全て書き下すのは非効率的であるため, 処理のまとまりを関数として定義して呼び出すことが基本的である. また, プログラムをわかりやすく分割するためにも用いられる. main 中で関数を呼び出したり, ある関数中で別の関数を呼び出したりすることができる. 以下にその動作イメージを示す.



関数は引数(ひきすう)を受け取り, 引数の値を使って, 処理を行い, 返り値を返すのが基本であるが, 引数を受け取らない関数や返り値のない関数もある.

3.1. 返り値のある関数

返り値のある関数の場合の関数宣言は以下のようである.

```
返り値の型 関数名(引数 1 の型 引数 1 の名前, 引数 2 の型, 引数 2 の名前,…){
作業関数の宣言
処理
return 返り値
}
```

引数がない場合は( )の中身を void と書いておく.

これを別の箇所から

```
返り値の型の変数=関数名(引数 1, 引数 2,…);
```

のように呼び出す. 引数の数, 型は呼び出す側, 宣言側で必ず一致しなくてはならない.

以下に例を示す.

```
double func(int i,double a,double b){
double v;
if(i==0){
v=a*a;
}else{
v=b*b;
}
return v;
}
```

のように宣言し,

```
int main(void){
int ii;
double v,x,y;
ii=4;
x=3.0;
y=4.0;
v=func(ii,x,y);
return 0;
}
```

のように呼び出す.

3.2. 返り値のない関数(作業関数)

返り値が複数必要な場合などは、3.1 のような使い方は不便である。その場合、返り値のない関数（作業関数）を定義し、引数として与える変数の値を変更して、メインプログラムに戻すことが行われる。

```
void 関数名(引数 1 の型 引数 1 の名前, 引数 2 の型, 引数 2 の名前,…){
作業関数の宣言
処理
return;
}
```

引数がない場合は( )の中身を void と書いておく.

これを別の箇所から

```
関数名(引数 1, 引数 2,…);
```

のように呼び出す. 引数の数, 型は呼び出す側, 宣言側で必ず一致しなくてはならない.

ただし、作業関数の取り扱いには注意が必要である. 引数の整数型, 実数型変数の値を関数中で変更しても, 関数呼び出しから戻った際, 呼び出し元においては, その変更は無効になり, 呼び出し前の値に戻る.

このため, 複数の変数を作業関数において変更したい場合には, 少し工夫を行う必要がある.

例を挙げると
・1. グローバル変数を変更する
・2. ポインタを引数に取る
・3. 配列を引数に取る
・4. 配列形を返り値にする
などである.

ここでは 3.の 1 次元配列を引数にとる方法を説明する. 以下のように定義する.

```
void 関数名(…配列の型 配列名[]…){
作業関数の宣言
処理
return;
}
```

この配列中の値は作業関数による変更が関数呼び出し元でも有効に働く.

以下に例を示す.

```
#include<stdio.h>
#define N 3
void work(int n,int a[]){
   int i;
   for(i=0;i<n;i++){
      a[i] = i*i;
  }
  return;
}
int main(void){
  int i;
  int v[N];
  work(N,v);
  for(i=0;i<N;i++){
    printf("v[%d]=%d¥n",i,v[i]);
  }
  return 0;
}
```

作業関数

呼び出し

をコンパイル・実行すると

```
v[0]=0
v[1]=1
v[2]=4
```

のようになる．ここで一回の関数の実行で配列の中の 3 つの変数を変更できたことに注目してほしい．

```
#include<stdio.h>
#define N 3
void work(int i,int a){
a = i*i;
return;
}
int main(void){
   int i,a;
   a=1000;
   for(i=0;i<N;i++){
      work(i,a);
      printf("a_%d=%d¥n",i,a);
   }
   return 0;
}
```

をコンパイル・実行すると、作業関数での計算

```
a_0=0
a_1=1
a_2=4
```

とならず、

```
a_0=1000
a_1=1000
a_2=1000
```

のようになり、a の値はメインプログラム変更されない。

3.3. プロトタイプ宣言

コンパイラによっては上手く行く場合もあるが, 関数はプロトタイプ宣言するほうが無難である. プログラムの冒頭(#define に下にでも)に

```
返り値の型 関数名(引数 1 の型 引数 1 の名前、引数 2 の型、引数 2 の名前,…);
```

を列挙する. 関数宣言と異なり;(セミコロン)が必要なことに注意.

以下にプロトタイプ宣言の例を示す.

```
#include<stdio.h>

double func1(double a,double b);
double func2(double a,double b,double c);

int main(void){
double v1,v2,x,y,z;
x=3.0;
y=4.0;
z=5.0;
v1=func1(x,y);
v2=func2(x,y,z);
return 0;
}

double func1(int i,double a,double b){
return a*a+b;
}

double func2(int i,double a,double b,double c){
return a*a*a+b*b+c;
}
```

4. ファイル入出力

4.1. ファイル入力

違う値を変数に入力するために毎回コンパイルを繰り返すのは非効率的である．このため，入力ファイルを使った変数入力は一般的である．ファイル入力をするためにはまず,ファイルポインタ型変数を宣言する．本テキストの趣旨に反してポインタを使わざるを得ないが，ファイルポインタ型変数は以下に説明するとおりに使えばよいだけである．宣言は以下のようになる．

```
FILE *変数名;
```

この変数名に入力ファイルを結びつけ，ファイルを読み込み(r)モードで開く(fopen).

```
ファイルポインタ型変数 = fopen(“ファイル名”,”r”);
```

次に入力ファイルを読み込む．fscanf を用いれば，第 1 引数にファイルポインタを指定して，scanf と同様の書式でファイルを読み込むことができる．他にも fgetc,fgets などで読み込むことができる.読み終わったら最後にファイルを閉じる(fclose).

```
fclose(ファイルポインタ型変数);
```

以下に input.dat から配列 a に値を読み込む例を示す.

```
double a[N];
FILE *fp;
fp=fopen(“input.dat”,”r”);
for(i=0;i<N;i++){
fscanf(fp,“%lf”,&a[i]);
}
fclose(fp);
```

4.2. ファイル出力

ディスプレイに標準出力すると,後から見るときに不便であるため，ファイルに出力するのが一般的である．ファイル出力するためにはまず fopen によって書き込みモード(w)でファイルを開くか,

```
ファイルポインタ型変数 = fopen(“ファイル名”,”w”);
```

書き込み追加モード(a)でファイルを開くかする．

```
ファイルポインタ型変数 = fopen(“ファイル名”,”a”);
```

書き込みモードで開いた場合はファイルがクリアされてゼロから書き始められるので注意が必要である．

ファイルに書き込みを行うためには fprintf を用いれば，第 1 引数にファイルポインタを指定して, printf と同様の書式でファイルに書きこむことができる．書き終わったら最後にファイルを閉じる(fclose).

以下に output.dat に v の値を書きこむ例を示す.

```
double v;
FILE *fp;
v=0.1;
fp=fopen(“output.dat”,”w”);
fprintf(fp,“v=%lf¥n”,v);
fclose(fp);
```