# C 言語によるプログラミング

１．C プログラミングのための環境整備
２．C プログラムの開発手順
３．C プログラムの書き方
４．変数の使い方
５．標準入出力
６．関係演算子（比較）
７．条件判断構造（if 文）
８．繰り返し構造（for 文）
実力テスト
９．もうひとつの条件判断構造（switch 文）と繰り返し構造（while 文）
実力テスト
１０．関数によるプログラムの分割
１１．構造体

佐井・李研究室

## １．Ｃプログラミングのための環境整備

◇必要な道具　－－－　エディタ，コンパイラ，**CUI** ベースの端末

● エディタ

プログラムを作成・編集するときに使うツール．テキスト形式のファイルを作成・編集できれば何でも可．

Windows 系の場合：　「メモ帳」，「ワードパッド」，「Microsoft Word」など．

Linux 系の場合：　「Emacs」，「Mule」，「Vi」など．

● コンパイラ

人間が作成したプログラムを，コンピュータが理解できる言葉に翻訳するためのプログラム．

＜代表的なコンパイラ＞

・GNU　C　コンパイラ

・Visual　C++　コンパイラ

・Borland　C++　コンパイラ

Linux 系の OS では，GNU C ライブラリ（コンパイラを含む）が標準で提供されている．

コンパイラのコマンド名は「gcc」．

● CUI ベースの端末

CUI とは文字を介してコンピュータへの入出力を行うインターフェースである．

Windows 系の場合：　「スタート」→「プログラム」→「アクセサリ」→「MS－DOS プロンプト」

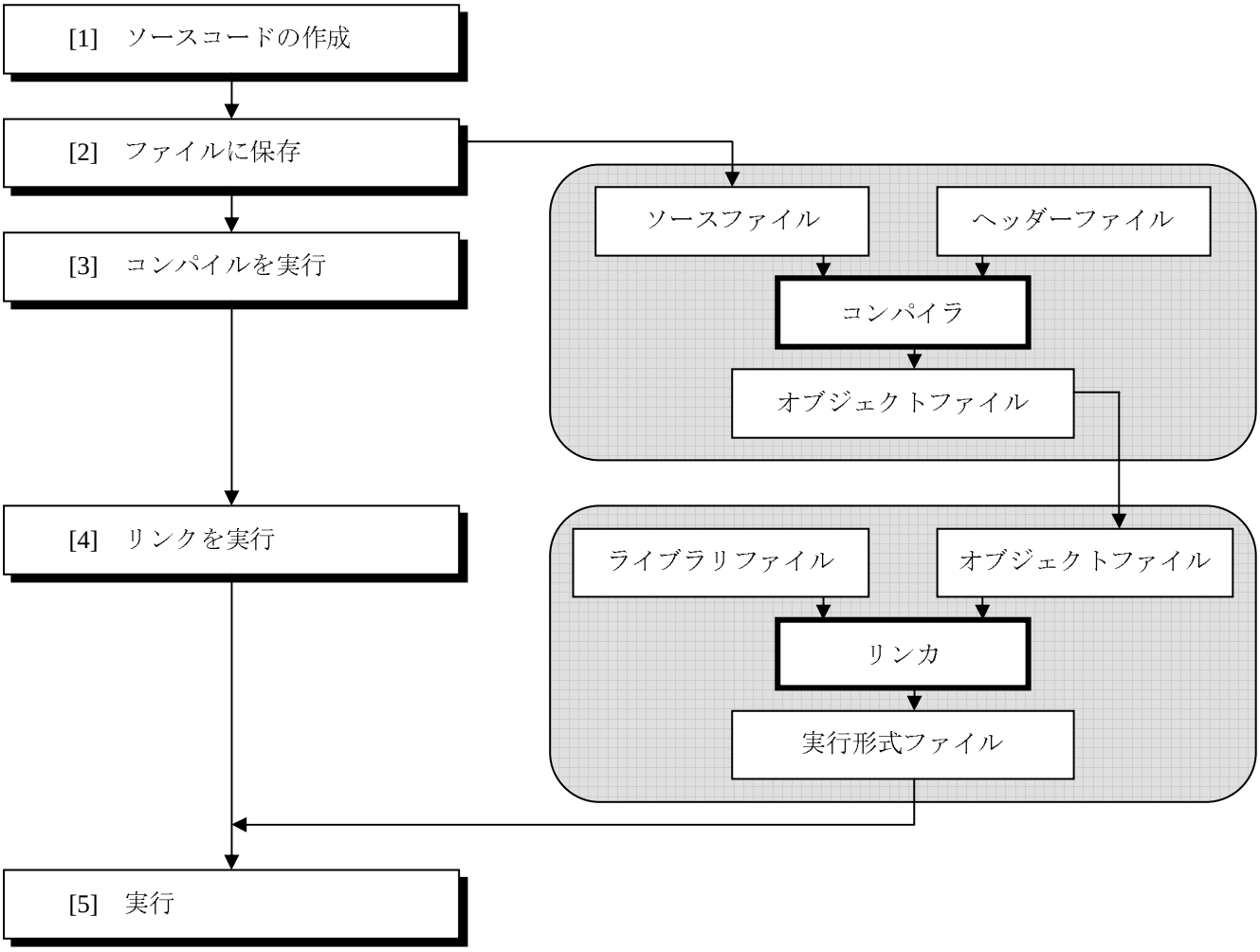
Linux 系の場合：　「Kterm」「Console 端末」など

がそれに相当する．Linux 系の OS で CUI ベースの端末は，プログラミングを行うためには必須である．

なお，Windows での統合開発環境（Visual C++, Borland C++など）では，CUI ベースの端末は必要ない．

本文の中では，以降 kterm を利用することを前提とする．

## ２．Ｃプログラムの開発手順




[1] エディタでソースコードを書く

プログラミング言語で記述するプログラムのことを**ソースコード**という呼ぶ．「コード」とは，記述した命令の一つ一つを指す．その命令がたくさん集まったものという意味で「ソースリスト」と呼ぶこともある．通常は単に「ソース」と呼ばれることが多い．

[2] ソースコードをファイルに保存する．

C 言語で作成したプログラムは「****.c」という名前で保存するのが決まりである．つまり，拡張子は「C 言語で作成したプログラム」を表す「.c」でなければならない．プログラムを保存したファイルを「**ソースファイル**」と呼ぶ．

[3] コンパイルを実行する

プログラミング言語からコンピュータ理解できる「０」と「１」の組み合わせに置き換えることを，**コンパイル**と呼ぶ．この作業に使うのが「コンパイラ」と呼ばれるプログラムである．

コンパイルに成功する，つまりはプログラムが文法的に正しければ，**オブジェクトファイル**が自動的に作成される．オブジェクトファイルは「****.o」という形をしている．

コンパイルに失敗した場合は，エラーメッセージが表示される．エラーメッセージは，プログラム中の何行目にどのような間違いがあるのかを表すことが多い．

[4] リンクを実行する

リンクとは，プログラムの実行に必要なファイル（ライブラリファイル）をオブジェクトファイルに付け足す作業を指す．この作業を実行するのがリンカと呼ばれるプログラムである．コンパイラによっては，コンパイル時に自動的にリンカを実行するものもある．

[5] 実行する

コンパイル，リンクを実行すると「０」と「１」に翻訳されたファイルが出来上がる．これを実行形式ファイルと呼ぶ．このファイルを使って，作成したプログラムを実行する．

もしもプログラムが期待したとおりに動かないとしたら，それはプログラムのどこかに間違い（バグ）があるからである．コンピュータは，指示されたとおりに動くことしかできない．この場合，手順[1]に戻ってプログラムを見直す作業をしなければならない．

Linux でのエディタ　→　emacs

コンパイルの仕方　→　gcc

UNIX のコマンド：　pwd, ls, cd, cp, mkdir, rm

詳細は医用画像工学実験のホームページおよび参考書（プログラミング実践の基礎）P38-46 を参照

▼画像表示プログラムを動かしてみよう．　※よくわからない場合は後回しで良い

2byte データの画像を読み込み，ディスプレイに表示する．

| 手　順 | kterm 上での入力（☆は半角スペース） |
|---|---|
| ①ホームディレクトリに，ディレクトリ sliceView を作成する． | mkdir☆sliceView |
| ②sliceView に移動する． | cd☆sliceView |
| ③/home/lee/sliceView から，カレントディレクトリに各ファイルを（maic.c, xdisp.c, rw_data.c, byte_swap.c, trans_wl.c, Makefile, liver_1_se1_145.raw)をコピーする． | cp☆/home/lee/sliceView/main.c☆./ |
| ④コンパイルを実行． | make |
| ⑤プログラムを実行． | ./ViewImage☆liver_1_se1_145.raw☆512☆512 |

「卒研前準備－－課題」のホームページのプログラムも同様に動かしてみよう．

↑

研究室ホームページ（佐井 or 李）からリンク有り

## ３．C プログラムの書き方

◇ プログラムの基本形（何もしないプログラム）

```
#include <stdio.h>
int main(int argc, char **argv)
{
    return 0;
}
```

どんなプログラムもこの基本形から作成することができる．

● #include <stdio.h>

C 言語プログラムの先頭には，必ずこの命令が必要である．

これはプリプロセッサ指令と呼ばれもので，

**「stdio.h」というファイルを読み込め**

という命令である．「読み込め」の定義は，「stdio.h」の中身を#include <stdio.h>の行に置き換えることである．つまり，#include <stdio.h>という１行で，stdio.h というファイルの中身すべてを記述したことになる．

stdio.h はコンパイラにあらかじめ用意されているプログラムで，キーボードから値を入力したり，画面に文字を出力したりするための基本的な命令（scanf 文，printf 文など）が定義されているファイルである．

stdio.h のように拡張子が「.h」のファイルをヘッダーファイルと呼ぶ．ヘッダーファイルには，プログラム本体を処理する前にしなければならない処理が記述されている．そのため，main 関数よりも先に記述する必要がある．stdio.h のほかによく使われるヘッダーファイルとして，stdlib.h，string.h，math.h などがある．

● int main(int argc, char **argv)

プログラムの入り口を表し，**main 関数**と呼ばれる．C 言語のプログラムは，必ずこの main 関数から実行される．そのため，実行する処理は，すべて main 関数の中に記述されていなければならない．

**C 言語のプログラムは，int main(int argc, char **argv)から始まる**

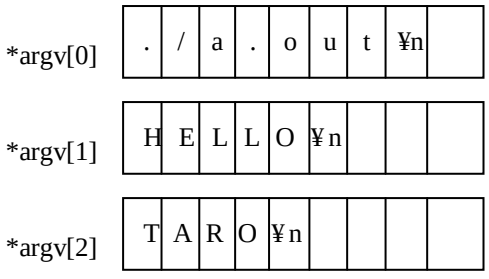
ということを覚えておきましょう．

main の前にある「int」は，「main 関数の最後に整数を返す」ことを意味する．「int argc」は，main 関数が使う引数の個数を表す．実際には，配列 argv の要素数と等しい．「char **argv」は，コマンドラインから入力される文字列を格納する配列へのポインタである．（ポインタに関する詳細は後ほど）

例えば，次のようなプログラムを実行したとき，

./a.out　HELLO　TARO

argc の値は 3 となり，argv には次のような値が代入される．



● {　}

２つのカッコは，main 関数の開始と終わりを表す．プログラムで実行したい処理は，{と}の間（正確には「return 0;」の前まで）に記述する．{　}で囲まれた部分をブロックと呼ぶ．

カッコを置く位置に特に決まりはないが，通常はプログラムの見易さを考えて，ブロックの始まりを表す命令（ここでは main 関数）の先頭位置に合わせる．

● return 0;

main 関数の最後には

**return 0;**

を記述する．これは，「main 関数を実行した結果，０という値を返す」ことを意味する．なお，この記述は省略しても特に問題はないが，return 0;を記述するくせ（？）をつけておくとよい．

■ 命令文の書き方

main 関数の中に書かれている１行１行が命令文である．C 言語のプログラムは，基本的に半角英小文字と半角数字で記述される．原則として，１行には命令文を１つだけ記述し，命令文の終わりには，セミコロン（；）をつける．

【例１】　printf("名前を入力してください.....");

命令文によっては１行に書ききれないほど長くなるものもある．そのときは，命令文中の区切りで複数行に分けることができる．改行した行の末尾にはセミコロンをつけてはいけない．

```
【例１】　printf("名前は　%s　さんですね．間違っていませんか？¥n",
                    name );
```

なお，プログラミング言語によっては，大文字／小文字を区別しないものもあるが，C 言語では大文字の「A」と小文字の「a」は別のものとして扱われる．

■ コメントの書き方

コメントは，プログラムの中に記述するメモのようなものである．処理内容を簡単なメモでプログラム中に残しておくと，後で読み返したときに便利である．C 言語のコメントは，２種類の書き方ある．

・//　記号からその行の終わりまでが，すべてコメントになる．
・/* と */ で囲んだ部分がコメントになる．

２つの記号の使い分けに，特に決まりはない．なお，コメント行はコンパイル時には無視される．このことを利用して，実行したくない命令文をコメントにすることもできる．

■インデント，空行の使い方

プログラムを書くときに，命令文の先頭に入れる空白をインデントと呼ぶ．通常は，ブロックごとにインデントを設定する．プログラムの構造が複雑になれば，たくさんのブロックを使用しなければならない．インデントを利用することによって，ブロックの範囲がわかりやすくなる．なお，大抵はエディタではインデントを入れるときに「タブ」を利用する．

【例】
```
#include <stdio.h>
int main(int argc, char **argv)
{
    int     i, j;
    for ( i = 0; i <= 9 ; i++) {
        for ( j = 0; j <= 9 ; j++) {
            printf ("%d    ", i*j);
        }
        printf ("¥n");
    }
    return 0;
}
```

◆ 以下のサンプルプログラム１（sample1.c）をエディタ（emacs）で作成し，実行してみましょう．

```
#include <stdio.h>
int main(int argc, char **argv)
{
    /* 変数の宣言 */
    char     name[20];     // 名前
    char     food[20]:      // 好きな食べ物

   /* 会話その１ */
   printf("名前を入力してください.....");
   scanf("%s", name);
   printf("%s さん．はじめまして！¥n", name);

   /* 会話その２ */
   printf("好きな食べ物を入力してください.....");
   scanf("%s", food);
   printf("%s は美味しいもんね！¥n", food);

   return 0;
}
```

・実行手順

① kterm（CUI 端末）を開く

② kterm 上で mkdir☆sample と入力し，Enter キーを押す．　*）☆は「スペース」

③ cd☆sample　と入力し，Enter キーを押す．

④ emacs☆sample1.c☆&　と入力し，Enter キーを押す．

⑤ 開いた emacs 上で，上記のプログラムのソースを入力する．

⑥ すべて入力したら，保存する．保存は，「Ctrl」＋「x」，「Ctrl」＋「s」を押せばよい．

「Ctrl」＋「x」は「Ctrl キーを押しながら x キーを押す」ことを意味する．

kterm 上で　ls　と入力し Enter キーを押して，sample1.c というファイルが存在することを確認する．

⑦ プログラム sample1.c をコンパイルする．

コンパイル方法：　gcc☆sample1.c　と入力し，Enter キーを押す．エラーがでなければ，ls　と入力し a.out　というファイルが作成されていることを確認する．もしもエラーメッセージが出たのなら，ソースが間違っていないか確かめる．

⑧ プログラムを実行する．　./a.out　と入力し，Enter キーを押す．

◆ sample1.c の解説

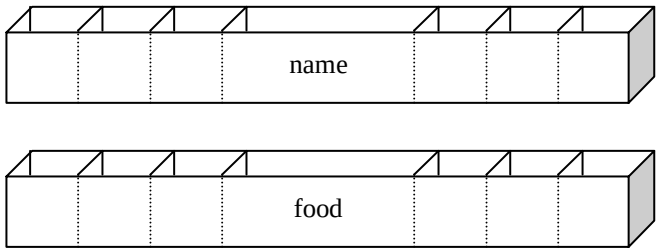
● 入れ物を用意する

入力された値をコンピュータは覚えておかなければならない．そのためには，何か入れ物が必要である．これをいわゆる**変数**と呼ぶ．main 関数の先頭に

```
char    name[20];
char    food[20]:
```

という命令文がある．これは箱を２つ用意して，それぞの箱に name と food という名前をつける命令文である．先頭の char は，「この箱には文字を入れる」ことを意味し，後の[20]は，箱に入れられる最大文字数を表す．この変数については次節「4.変数の使い方」で詳しく述べる．

【変数のイメージ】




● 画面（kterm 上）に文字を出力する

printf という命令文を使うと，画面に数値や文字列を出力することができる．sample1.c では，

```
printf("名前を入力してください.....");
```

のような命令を使っている．この命令を実行すると，画面には，

名前を入力してください

という文字列が出力される．

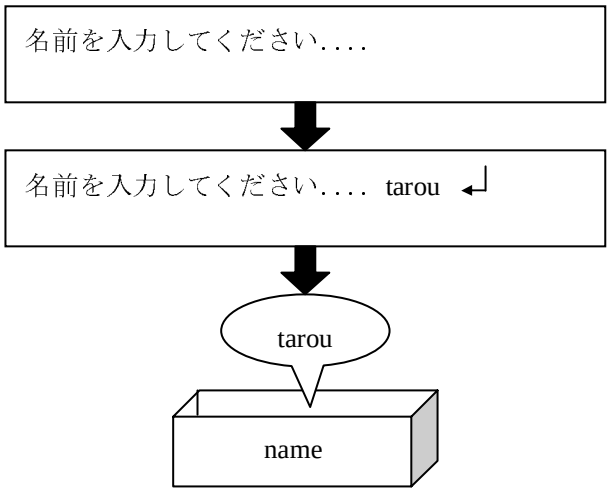
● キーボードからの入力

scanf という命令を使うと，キーボードから入力した値をコンピュータに記憶させることができる．sample1.c では，

scanf("%s", name);

のような命令を使っている．この命令を実行すると，プログラムの実行が一時中断しキーボードからの入力待ち状態になる（kterm 上）．このときにキー入力すると，Enter キーを押すまでに入力した値が，先に用意しておいた「name」という箱に入れられる．

【イメージ図】kterm 上




● 入力された文字を画面に出力する

sample1.c では

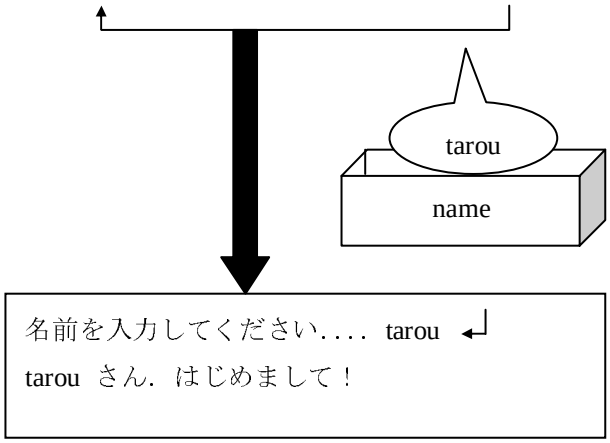
printf("%s さん．はじめまして！¥n", name);

のような命令を使っている．一番最後の「name」は，キーボードから入力した値が入っている箱の名前である．この命令を実行すると，「%s」の部分を箱の中身に置き換えて，

○○○さん．はじめまして！

という文字列が画面（kterm 上）に出力される．

【イメージ図】 printf( " %s さん．はじめまして！¥n ", name );



## ４．変数の使い方

＜変数の概念＞

変数は，プログラムの実行時に使う値を一時的に記憶しておく入れ物である．変数を正確に定義するならば，

メモリ上で値を記憶した領域に付けた名前

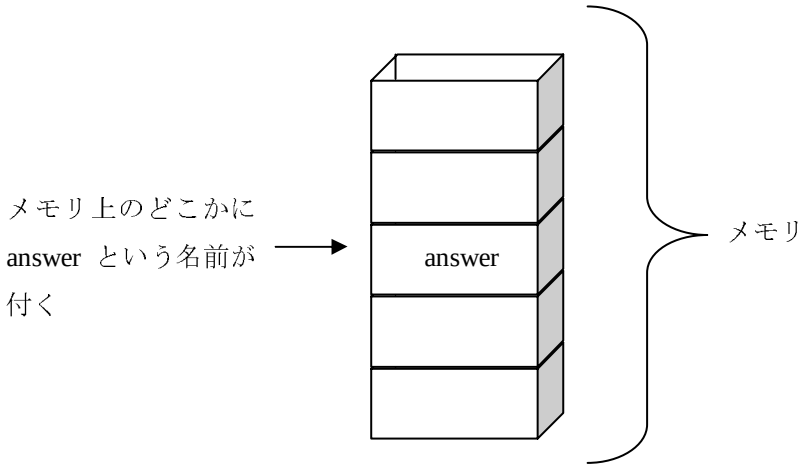
のようになる．

コンピュータは値を記憶するためにメモリを使用する．メモリは，たくさんの入れ物（箱）の集まりだと間得るとよい．それぞれの入れ物には番地（アドレス）が付いており，値を入れた番地を人間が覚えるのは大変である．そこで人間が覚えやすいように使われるのが変数なのである．例えば，

```
int     answer;
```

という命令を実行すると，メモリ上のどこかに「answer」という名前の入れ物が用意される．この入れ物が変数で，answer は変数に付けた名前（変数名）である．先頭の「int」は変数に入れるデータが整数であることを意味する．

【イメージ図】




＜変数の使い方＞

変数を使うには，

① 変数を宣言する

② 変数を初期化する

という作業が必要となる．宣言は入れ物を準備する作業，初期化は入れ物をきれいにする作業である．

● 変数を宣言する

入れ物を用意することを変数の宣言と呼ぶ．変数の宣言には，

・入れ物の大きさを明確にすること

・入れ物に名前を付けること

という２つの仕事がある．書式は次の通りである

---

【書式】　データ型　　変数名；

---

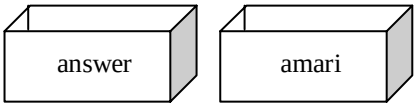
データ型は，入れ物に入れるデータの種類を表す．文字や，整数，実数など入れるデータによって異なる．

さきの例の　int　answer;　のように宣言すると，整数を入れるための箱が用意されることになる．なお，変数の名前は，自由につけることができる．

▼ 複数の変数を宣言する方法

同じ種類のデータ型の変数を宣言する場合は，

```
int　answer, amari ;
```




のように，変数名をカンマ（,）で区切って宣言することができる．上の命令を実行すると，整数を入れられる answer と amari という２つの箱が用意される．

変数には代入する値がわかるように意味のある名前を付けるのが一般的である．しかし，プログラムを作成する人によってバラバラになるのは当然である．そこで，変数の意味をコメントとしておくとわかりやすくなる．コメントを記述するためには，１行に１つの変数を宣言しなければならない．上の命令も，２行に分けて記述すると，わかりやすいプログラムになる．

```
【例】　　int　　answer ;　　// 一人あたりの個数
　　int　　amari ;　　// 余った個数
```

● データの種類

下記表は C 言語のプログラムで扱うことのできる主なデータ型の一覧である．データ型は，データの種類を区別するためのものである．表中の「大きさ」は値を入れるために必要なバイト数で表わされている．

| データ型 | 意味 | 大きさ（バイト数） |
|---|---|---|
| char | 文字（整数） | 1 |
| short | 整数 | 2 |
| int | 整数 | 4 |
| long | 整数 | 4 または 8 |
| float | 浮動小数点数 | 4 |
| double | 浮動小数点数 | 8 |
| long double | 浮動小数点数 | 8 または 12 |

char 型は文字列を扱うデータ型である（整数を扱うこともできる）．通常，整数型を扱うときは int 型，浮動小数点数を扱うときは float 型もしくは double 型を使うとよい．

▼ 浮動小数点数

小数点を含んだ数値を，プログラムの世界では**浮動小数点数**と呼ぶ．C 言語のプログラムで浮動小数点数を扱うデータ型は float, double, long double の３つがある．float より double のほうが小数点以下の有効桁数が多くなる．そのため，精度が重要な計算には float よりも double を使用するとよい．

下記表は整数を扱えるデータ型の範囲である（long は省略）.

| データ型 | 範囲 |
|---|---|
| char | -128～127 |
| short | -32768～32767 |
| int | -2147483648～2147483647 |

データ型には**符号なしのデータ型**もある．宣言の仕方は通常のデータ型の前に **unsinged** を付ければよい.

| 符号なしのデータ型 | 意味 | 大きさ（バイト数） | 範囲 |
|---|---|---|---|
| unsigned char | 文字（整数） | 1 | 0～255 |
| unsigned short | 整数 | 2 | 0～65535 |
| unsigned int | 整数 | 4 | 0～2147483647×2+1 |

● 変数名の付け方

・名前の先頭は半角英文字

変数名には A～Z，a～z，0～9 および_（アンダーバー）を使うことができる．ただし，先頭は必ず半角英文字でなければならない．数字やアンダーバーを先頭につけると，コンパイル時に大量のエラーが発生する.

【正しい例】　int　answer1 ;
【間違った例】　int　1answer ;

・C 言語の予約語は使えない

予約語とは，C 言語にとって意味のあるキーワードである．変数のデータ型を表わす int や float なども予約語の一部である．これら言葉を変数名に使うことはできない．また，main や printf，scanf など，C 言語で使う標準命令も変数の名前に使うことはできない.

▼変数名の付け方には，上記以外には特に決まりはない.
例えば，最大値を入れておくならば「max_data」のような名前と付ければよい．アンダーバー（_）は，変数名が長くなったときの区切り文字として使うと便利である.

● 宣言をする場所

変数は関数内の先頭で宣言しなければならない．命令文（printf など）の後ろで宣言すると，コンパイル時にエラーが発生する.

● 変数に値を代入する

変数を宣言すると入れ物が容易される．しかし，宣言した直後の入れ物には，何がはいっているかわからない．このままの状態で利用すると，意図しない計算結果になる可能性がある．そのため，通常は実際に変数を使用する前に，**変数の初期化**を行う.

変数に値を代入するには，イコール（＝）を使う．これは**代入演算子**と呼ばれる．プログラムの世界では，

イコールを「代入」の意味で使用するので，算数のイコールと異なる点に注意する．

変数には，宣言したときに指定した種類のデータしか入れることができない．たとえば，

```
int     a ;
```

のように宣言した場合，変数の a には整数しか入れることができない．もしも小数点を含んだデータを入れた場合，小数点以下は切り捨てられる．

【例】　　a　=　10.5　とした場合　a　には 10 が代入される．

★　練習問題

char,　short,　int,　float,　double の各型のバイト数を出力するプログラムを作りなさい．
なお，データ型のバイト数を計算する関数として siseof（型）が C 言語には用意されているのでこれを使いなさい．

## ５．標準入出力

● 標準入出力とは？

標準入力装置からデータを入力する，または標準出力装置にデータを出力する処理

・標準入力装置：　キーボード
・標準出力装置：　画面（モニタ）

標準入出力処理を行う命令を**標準入出力関数**と呼ぶ．C 言語では次のような標準入出力関数がある．

・printf 関数　　：　指定された書式で画面に出力する
・scanf 関数　　：　キーボードから入力された値を受け取る
・getchar 関数　：　キーボードから入力された１文字を受け取る

⋆）これらの標準入出力関数は，stdio.h というヘッダーファイルで定義されている．そのため，プログラムの先頭で，必ずこのファイルをインクルードしなければならない．

プログラムの先頭に

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

を記述する．

● printf 関数と scanf 関数の使い方

---

【書式】　**printf ("表示する文字列および変換文字", 変数名)** ;

---

---

【書式】　**scanf ("変換文字", &変数名)** ;

---

変換文字とは？

画面に出力するデータの種類を表わす文字．通常の文字列同様，ダブルクォーテーション（"）で囲まなければならない．

変換文字一覧

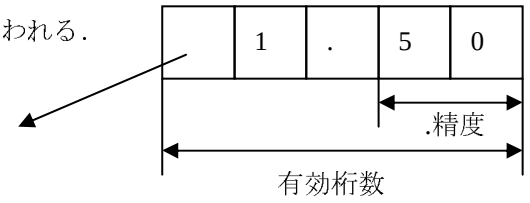
| データの種類 | 変換文字 |
|---|---|
| 整数（10 進数） | %d |
| 整数（8 進数） | %o |
| 整数（16 進数） | %x |
| 浮動小数点数 | %f |
| 文字 | %c |
| 文字列 | %s |

*）printf 関数では，数値の桁数を指定して出力することができる．変換文字を使用するときに次の書式に従って書けばよい．

【変換仕様】　%［有効桁数］[.精度］変換文字

「有効桁数」は，数値を出力するときの最大桁数，「.精度」は，小数点以下の桁数を表わす．

【例】　「1.5」を「%5.2f」という書式で出力すると右のように全体で 5 桁分の領域が確保されそのうち 3 桁が小数点以下の値に使われる．




*）scanf 関数では，値を代入する変数の前に必ずアンパサンド（&）を付けなければならない．ただし，変換文字が%s である場合は，変数名に文字列用の配列変数名を書くだけでよい（&不要，sample1.c 参照）．

「&変数名」の意味

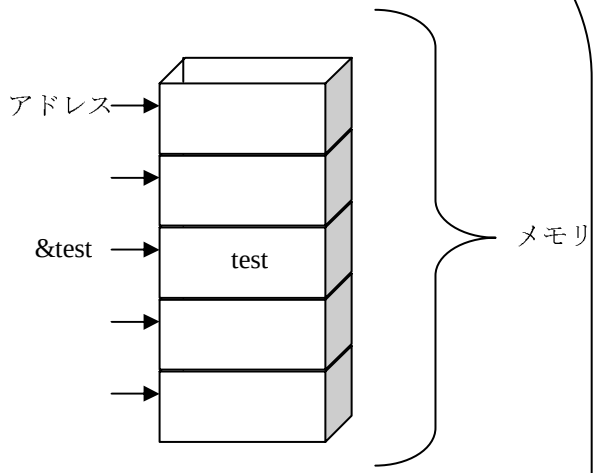
変数名の先頭に「&」をつけて

```
&test
```

と記述すると，「test という名前を付けた場所」＝アドレスを表わすことになる．つまり，

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

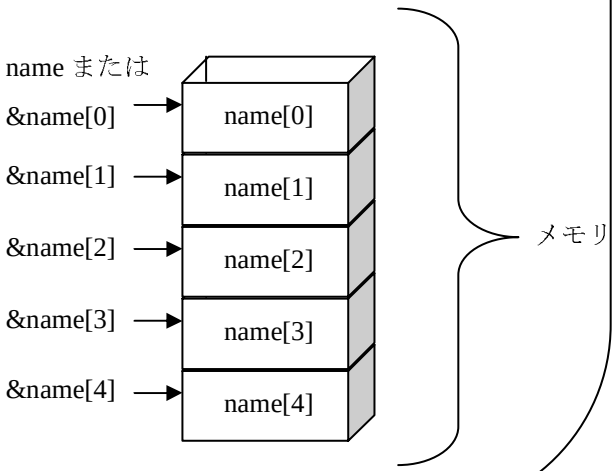
は，「キーボードから入力された値を，メモリ上の test という名前の付いた場所に入れろ」という命令である．




文字列用の配列変数である場合は，「&」は不要である．
例えば，

```
char   name[20];
scanf ("%s", name);
```

のように書けばよい（sample1.c 参照）．これは「キーボードから入力された文字列を，メモリ上の name という名前の付いた場所の先頭から順番に 1 文字ずつ入れろ」という命令である．ここで ha，name という変数名自体が配列 name の最初のアドレスを示す．
つまり，name と &name[0]は同じ意味を持つ．



【標準入出力関数の使用例】　sample2.c

プログラム内容

```
#include <stdio.h>
int main(int argc, char **argv)
{
      int     tosi ;
      printf ("あなたは何歳ですか？　" );
      scanf ("%d", &tosi);
      printf ("あなたは%d 歳ですね¥n", tosi );
      return 0;
}
```

画面上（kterm など）

```
gcc sample2.c        ←コンパイル
./a.out              ←実行
あなたは何歳ですか？　25
あなたは 25 歳ですね
```

★ 練習問題 5-1

sample2.c をもとに，年齢が 10 進数，8 進数，16 進数で出力されるプログラムを作りなさい.

★ 練習問題 5-2

sample2.c をもとに，月も換算した年齢を入力し，年齢が浮動小数点数で出力されるようにしなさい.
また，出力時に有効桁数を指定しない場合と「有効桁数」5,「.精度」2 に指定した場合を比較しなさい.

月換算年齢の例：　22 歳で 12 月生まれの場合，現在を 6 月と仮定すると，入力を 22.5 とする.

＜**getchar** 関数の働き＞

---

【書式】　変数名 = getchar ( ) ;

---

働き：　キーボードから 1 文字だけを入力する.

注意）C 言語プログラムでは，文字を扱うデータ型は char 型であるが，getchar 関数を使って 1 文字を受け取るときだけは，<u>値を代入する変数を **int** 型に宣言しなければならない</u>．これは，getchar 関数が stdio.h ヘッダーファイルの中で

```
int    getchar (void);
```

と宣言されているからである.

プログラム途中で，適当なキー入力があるまでプログラムの進行を止めたい場合などにもよく使用される.

【プログラム例】

```
#include <stdio.h>
int main(int argc, char **argv)
{
      /*  変数の宣言  */
      int     a;
      /*  メッセージの出力   */
      printf (" 1 文字だけ入力してください" );
      /*  キーボードから 1 文字入力  */
      a = getchar( );
      /*  画面に出力   */
      printf ("%c¥n", a );
      return 0;
}
```

## ６．演算子

C 言語では以下のような代表的な演算子が用意されている．この他にビット操作を行う演算子などもある．

【算術演算子】

| 演算子 | 意味 |
|---|---|
| ＋ | 足し算 |
| － | 引き算 |
| * | 掛け算 |
| / | 割り算の商 |
| % | 割り算の余り |

【代入演算子】

| 演算子 | 例 | 意味 |
|---|---|---|
| = | a = 5; | a に 5 を代入する |
| ＋= | a＋= 5; | a に 5 を足した値で a を更新する |
| －= | a－= 5; | a から 5 引いた値で a を更新する |
| *= | a *= 5; | a に 5 を掛けた値で a を更新する |
| /= | a /= 5; | a を 5 で割った時の商で a を更新する |
| %= | a %= 5; | a を 5 で割った時の余りで a を更新する |

【関係演算子】

| 演算子 | 意味 |
|---|---|
| == | 等しい |
| != | 等しくない |
| > | より大きい |
| < | より小さい |
| >= | 以上 |
| <= | 以下 |

【論理演算子】

| 演算子 | 意味 |
|---|---|
| && | 論理積 |
| \|\| | 論理和 |
| ! | 否定 |

【特殊な演算子】

| 演算子 | 名前 | 例 | 意味 |
|---|---|---|---|
| ＋＋ | インクリメント | a＋＋; | a に 1 を足した値で a を更新する．a = a＋1 と同じ処理． |
| －－ | デクリメント | a－－; | a に 1 を引いた値で a を更新する．a = a－1 と同じ処理． |

● 計算するときの注意

・オーバーフロー　：　扱うことのできる数値の範囲を超えてしまい，データが失われてしまうこと

【例】
```
short  answer;
answer = 1000 * 1000;
```

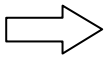
この場合，short 型の変数が扱うことができる範囲は「-32768～32767」である．ところが，「1000×1000」の計算結果は「1000000」となり，short 型の扱える範囲を超えてしまう．そのため，変数 answer に値が代入されるときにオーバーフローが発生してしまう．

・データ型の変換（キャスト）

C 言語プログラムでは変数のデータ型を当初宣言した型以外の型に強制的に変換することができる．そのた

めに用意されているものがキャスト演算子と呼ばれるものである．キャスト演算子は（データ型）で表わされる．

【例】
```
float   answer;
 int    a=7, b=3;
answer = a / b;
printf ("first answer = %f¥n", answer);
answer = (float) a / (float) b;
printf ("second answer = %f¥n", answer);
```



```
first answer = 2.000000
second answer = 2.333333
```

実行結果

この場合，変数 a と b は int 型で宣言してあり，最初の演算 a / b ではそれぞれ int 型で計算される．つまり，7 / 3 の計算が行われその結果は 2 となる．この 2 が float 型で宣言されている answer に代入されるため first answer は 2.000000 となる．これに対し，second answer では int 型で宣言されていた変数 a と b をキャスト演算子を使って強制的に float 型に変換している．(float) a / (float) b は 7.000000 / 3.000000 を意味する．つまり，計算が浮動小数点数で行われることを意味する．このことから計算結果は 2.333333 となり，この値が answer に代入される．

（プログラムを作成するときは常にデータ型を意識するように心掛ける．そのことによってオーバーフローなどが回避でき，正確な計算（自分が意図した計算）が行えるようになる．）

★ 練習問題 6-1

標準入力から２つの整数を入力し，その２つの値の算術演算結果（+，－，*，/，%）を出力するプログラムを作成しなさい．

★ 練習問題 6-2

練習問題 6-1 で作成したプログラムを，算術演算が浮動小数点数で行われ，結果も浮動小数点数で出力されるように変更しなさい．（標準入力は整数型で入力させる）

● 関係演算子の使い方

---

**【書式】　演算結果　＝　（　値 1　関係演算子　値 2　）；**

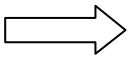
演算結果　　：　右辺の条件式を判断した結果．正しいときは 1，間違っているときは 0

値 1，値 2　：　比較する値

関係演算子　：　比較に使う演算子

---

【例】
```
int   result,   a=7,   b=10;
printf (" a = %d,   b = %d  のとき¥n", a, b );
result = ( a > b )
printf (" a > b : %d¥n", result );
result = ( a < b )
printf (" a > b : %d¥n", result );
```



```
a = 7, b = 10 のとき
a > b : 0
a < b : 1
```

実行結果

★ 練習問題 6-3

キーボードから入力された値が「正の数」かどうかを調べて，
画面に出力するプログラムを作成しなさい．

【出力結果イメージ】

```
./a.out
値      1              ←入力
結果    1              ←正の場合１
./a.out
値      －１           ←入力
結果    0              ←負の場合０
```

● 論理演算子の使い方

---

**【書式】　演算結果　＝　（　条件式 1　）　論理演算子　（　条件式 2　）；**

演算結果　　：　右辺の式を判断した結果．正しいときは１，間違っているときは 0
条件式 1，条件式 2：　比較する条件式
論理演算子　　：　比較に使う演算子

---

【例】

```
int    result,    a=7,    b=10;
printf (" a = %d,    b = %d  のとき¥n", a, b );
result = ( a >= 10 && b >= 10 )
printf ("a >= 10 && b >= 10 : %d¥n", result );
result = ( a <= 10 | | b >=10 )
printf ("a <= 10 | | b >=10 :     %d¥n", result );
```

⇒

```
a = 7, b = 10 のとき
a >= 10 && b >= 10   : 0
a <= 10 | | b >=10       : 1
```

実行結果

★ 練習問題 6-4

キーボードから入力された値が「正の数」かつ
「100 より小さい値」かどうかを調べて，
画面に出力するプログラムを作成しなさい．

【出力結果イメージ】

```
./a.out
値      232            ←入力
結果    1              ←結果
./a.out
値      56             ←入力
結果    0              ←結果
```

## ７．条件判断構造（**if** 文）

---

【構文】 **if** （ （ 条件式 **1-1** ） 論理演算子 （ 条件式 **1-2** ） ・・・・ ） {

実行する処理１；

}

**else if** （ （ 条件式 **2-1** ） 論理演算子 （ 条件式 **2-2** ） ・・・・ ） {

実行する処理２；

}

:

:

**else** {

実行する処理３；

}

条件式 ： 条件式が複数のときは論理演算子を使って書くことができる

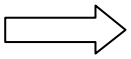
実行する処理 ： 条件式を評価した結果が「真」のときに実行する処理

この処理の中にさらに if 文があってもよい.

---

【例】

```
int     a, b;
printf (" a : ¥n );
scanf (" %d", &a );
printf (" b : ¥n );
scanf (" %d", &b );
if ( a == b ) {
      printf ("a と b は等しい¥n" );
 }
else if ( a > b )  {
      printf ("a は b より大きい¥n" );
else  {
      printf ("a は b より小さい¥n" );
 }
```



```
./a.out
a :  7
b :  10
a は b より小さい
./a.out
a :  10
b :  10
a と b は等しい
./a.out
a :  20
b :  10
a は b より大きい
```

実行結果

★ 練習問題 7-1

中間試験と期末試験のそれぞれの平均点が 80 点以上あれば合格となり「補習 0 時間」，どちらかの平均点が 80 点を下回れば「補習 10 時間」，どちらの平均点も 80 点未満であれば「補習 30 時間」が課せられる.

中間試験と期末試験の平均点を入力すれば，補習時間が出力されるプログラムを作成しなさい.

【出力結果イメージ】

```
./a.out
中間試験　85
期末試験　90
補習 0 時間
./a.out
中間試験　75
期末試験　90
補習 10 時間
```

## ８．繰り返し構造（**for** 文）

---

【構文】　**for**　（ カウンターの初期化；　継続条件；　カウンターの更新方法　）　{

実行する処理；

}

カウンタの初期化　：　繰り返す数を数える変数（通常は i をよく使う）の初期化

継続条件　　：　繰り返す回数を決める条件式

カウンタの更新方法　：　繰り返した数（i）を更新する方法

実行する処理　　：　継続条件式を評価した結果が「真」のときに実行する処理

この処理の中でさらに for 文を用いてもよい．

---

【例】

```
int     i;
for ( i = 0; i< 5; i++ ) {
        printf ("  こんにちは%d 回目¥n", i+1 );
}
```

```
./a.out
こんにちは 1 回目
こんにちは 2 回目
こんにちは 3 回目
こんにちは 4 回目
こんにちは 5 回目
```

実行結果

この場合，最初に for 文が実行されるときには i には 0 が代入される．また，継続条件である「i < 5;」は「**もしも i の値が 5 よりも小さければ，繰り返しを継続しなさい**」という意味をもつ．つまり，最初に for 文が実行されるときは i = 0 であるので，継続条件を満たす．継続条件を満たせばブロック{　}内の処理が実行される．このことにより最初の実行時には「こんにちは 1 回目」と表示される．ブロック{　}内の処理が終わると，カウンタが更新され，処理繰り返すかどうか判定される．カウンタの更新にはインクリメントの「i++;」が使われているが，これは i = i + 1;と同じ意味を持つ．つまり，2 度目に for 文が実行されるときには i には 1 が代入されることになる．i = 1 も継続条件「i < 5;」を満たすため，再びブロック{　}内の処理が実行され，「こんにちは 2 回目」と表示される．このような処理を繰り返し i = 5 となった時点でこの for 文による繰り返しは終了する．

注意）無限ループに陥らないように気をつける

【無限ループの例】

```
int    i, a=0;
for ( i = 0; i < 5; a++) {
        printf ("  こんにちは%d 回目¥n", i+1 );
}
```

この場合，継続条件は「i < 5;」であるにも関わらず，カウンタの更新方法では変数 i は使われていない．つまり，いくら繰り返しを行っても継続条件を満たし続けることになり，この for 文から抜け出ることができない．このような状態を**無限ループ**とよぶ．無限ループに陥るとプログラムは終わることができないため，強制終了させなければならない（Unix 上では「Ctrl＋c」で強制終了）．

● 繰り返しを途中でやめる方法

条件判断構造と break という命令を使用する.

---

【構文】 **for** （ カウンターの初期化； 継続条件； カウンターの更新方法 ） {

実行する処理；

**if ( 条件式 ) {**

**break;**

**}**

}

---

【例】

```
int     i;
for ( i = 0; i< 10; i++ ) {
        printf ("  こんにちは%d 回目¥n", i+1 );
        if ( i == 4 ) {
                break;
        }
}
```

```
./a.out
こんにちは 1 回目
こんにちは 2 回目
こんにちは 3 回目
こんにちは 4 回目
こんにちは 5 回目
```

実行結果

この場合，i = 5 になった時点で強制的に for 文から抜けるようになっている.

★ 練習問題 8-1

1 から 100 までの数字を足し算し，その答えを画面に出力しなさい.

★ 練習問題 8-2

掛け算の九九の表を作成し画面に出力しなさい.

[ヒント] 表示する数字の桁数を指定することで
きれいな表が出力できる.

【出力結果イメージ】

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 |

## 実力テスト

**問題１　簡易電卓を作ってみよう．**

キーボードから入力した２つの値を使って，足し算，引き算，掛け算，割り算のいずれからを実行するプログラムをつくりなさい．

【実行例】

```
./a.out
一つ目の値を入力してください　　10
二つ目の値を入力してください　　5
１：足し算　２：引き算　３：掛け算　４：割り算
どの計算を行いますか？　１
計算結果：　15.000000
```

**問題２　最大値を求めよう．**

太郎くん，花子さん，次郎くんが栗拾いにいきました．３人が拾った栗の数を順次入力し，だれが最もたくさん拾ったのかを出力するプログラムを作りなさい．

【実行例】

```
./a.out
太郎くんが拾った栗の数　　30
花子さんが拾った栗の数　　10
次郎くんが拾った栗の数　　20
答え：　太郎くん
```

**問題３　ボーナスを計算しよう．**

会社でボーナス査定システムを作成することになりました．今年のボーナスは，

**ボーナス　＝　（基本給×2.3ヶ月）＋　能力給　＋　ボーナスポイント**

という計算式で求めることになりました．基本給とボーナスポイントは，勤続年数によって決められています．また，能力給はランクによって決まっています．

勤続年数とランクを入力すると，今年のボーナス額を出力するプログラムを作りなさい．

【基本給とボーナスポイント】

| 勤続年数 | 基本給 | ボーナスポイント |
|---|---|---|
| 0～5 年 | 80,000 | 10,000 |
| 6～10 年 | 100,000 | 12,000 |
| 10～15 年 | 120,000 | 15,000 |
| 16 年以上 | 150,000 | 18,000 |

【能力給の計算式】

| ランク | 能力給 |
|---|---|
| A | 基本給×3 |
| B | 基本給×2.5 |
| C | 基本給×2 |
| D | 基本給×1 |

【実行例】

```
./a.out
勤続年数　　8
ランク　　b
ボーナス支給額　　492000 円
```

## ９．もうひとつの条件判断構造（**switch** 文）と繰り返し構造（**while** 文）

● switch 文

switch 文は，変数の値を定数と比較して処理を分岐します．「switch」は電源のスイッチと同じ意味です．変数の値が指定した定数と等しいときに，そのスイッチがオンになり，直後に記述した処理が実行されます．

---

【書式】 **switch**（変数）{

**case** 定数 **1**：

実行する処理 **1**；

**break**；

**case** 定数 **2**：

実行する処理 **2**；

**break**；

**:**

**:**

**default**：

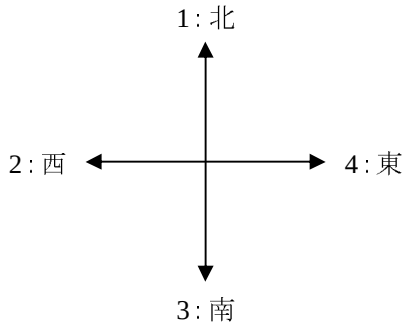
実行する処理 **n**；

}

---

【例】

```
int     direction;
switch (direction) {
    case 1:     /* direction==1 の場合  */
        printf ("北へ進みます．¥n");
        break;
    case 2:     /* direction==2 の場合  */
        printf ("西へ進みます．¥n");
        break;
    case 3:     /* direction==3 の場合  */
        printf ("南へ進みます．¥n");
        break;
    case 3:     /* direction==4 の場合  */
        printf ("東へ進みます．¥n");
        break;
    default:    /* direction が 1～4 以外の場合  */
        printf ("その場で待機します．¥n");
}
```



▼ **if** 文と **switch** 文の違い

・switch 文は，変数の値が定数と等しいかどうかを比較することしかできない．

・switch 文では比較する値の範囲を指定することができない．

⬇

「変数」==「定数」の判定しかできない

switch 文で書くことができない事例：

「もし変数 a の値が 100 より大きければ」

「算数の点数が 80 点以上なら A ランク，60～79 点なら B ランク．．．」．

すべての **switch** 文は **if** 文に置き換えることができるが，その逆はできない．

★ 練習問題 9-1

簡易電卓（実力テスト問題 1 参照）を switch 文を使って作成しなさい．

● while 文

---

【書式】 **while**（継続条件）{

実行する処理；

}

---

「while」は「～の間」という意味である．指定した継続条件を満たしている間だけ，ブロック{ }で囲まれた部分の処理を繰り返し実行する．

【例】

```
int     n = 0;
while ( n < 100 ) {
      n = n + 5;
      printf ("%d¥n",n);
}
printf ("終了¥n");
```

```
./a.out
5
10
:
95
終了
```

実行結果

▼ **for** 文と **while** 文の違い

for 文： 繰り返す回数がわかっているときに使用

while 文： 繰り返す回数がわからないときに使用

★ 練習問題 9-1

キーボードから「q」が入力されるまで，1 文字ずつ入力→表示するプログラムを使って作成しなさい．

「ヒント」 1 文字を直接表わすときにはシングルクォーテーション（'）で囲む．

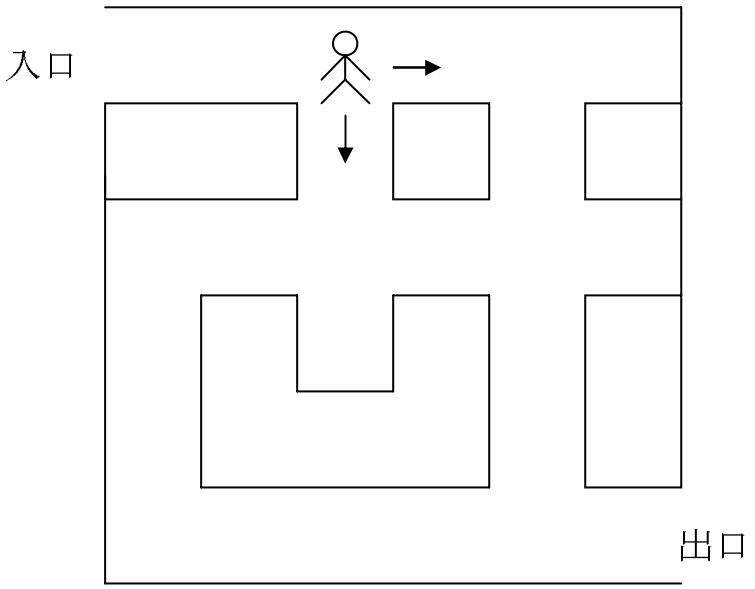
1 文字の比較の例 char moji; と宣言されていると仮定

if ( moji == ' q ' ) /* 変数 moji の中身が q であるならば

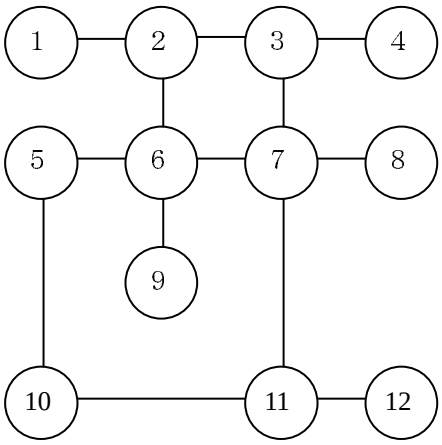
実力テスト

問題４　歩行（?）プログラムを作ってみよう.

次のような迷路を入り口から出発し，上下左右への移動をキーボードから入力させることで（１：上，２：下，３：左，４：右などのように），出口までたどり着くプログラムを作りなさい.




［ヒント］迷路の状態を，「人間が交差点，曲り角あるいは行き止まり地点にいる」ことで表す.



## １０．関数によるプログラムの分割

・C 言語での関数とは　→　　ある目的の動作を行うための独立したプログラム

・今まで出てきた関数　　→　main 関数，標準関数（printf, scanf など）

＜今までのプログラム＞

```
int    main (int argc, char **argv)
{
        処理 A

        処理 B

        処理 C

        return 0;
}
```

＜関数を使ったプログラム＞

```
int    main (int argc, char **argv)
{
    関数 A を実行
    関数 B を実行
    関数 C を実行
        return 0;
}
```

関数 A（処理 A）

関数 B（処理 B）

関数 C（処理 C）

●関数を使うメリット

・処理の流れをつかみやすくなる

まとまった処理を関数で作成し，main 関数ではそれを呼び出すだけにしておくと，処理の流れをつかみやすくなる．呼び出した関数の内容をコメントに残しておくとさらにわかりやすくなる．

・プログラムが扱いやすくなる（＝読みやすく，不具合を発見しやすい）

目的ごとの関数を作成することで，プログラムソースの行数を短くすることができる（見かけ上）．このことはプログラムを読みやすく，不具合を発見しやすくしてくれる．

・同じ処理を行う複数行のコードを何度も書く必要がなくなる

プログラムの中で何度も実行する処理を関数にしておくと，関数を呼び出す１行を記述するだけで，何度も利用することができる．

・ソースコードを修正しやすくなる

ある関数で不具合があった場合，その関数内を修正するだけですむ．もしも，同じコードをプログラムのあちこちに記述していたら，全体を見直して修正しなければならない．

---

★ 練習問題 10- 1

センチをインチに変換する関数を作成し，キーボードからセンチ単位の数値を入力したとき，インチ単位に変換された数値が出力されるプログラムを作りなさい．

★ 練習問題 10-2

キーボードから２つの整数値が入力されたとき，その２つの値の公約数をすべて求めるプログラムを作りなさい．ただし，公約数を求める（標準出力できればよい）部分は関数として作成し，その関数の戻り値は「最大公約数」としなさい．

---

●関数の作り方

---

【書式】　戻り値のデータ型　関数名　（データ型　引数１，　データ型　引数２，...）

```
{
        実行する処理；
        return　戻り値；
}
```

---

【例 1】

```
#include <stdio.h>
void disp_msg (void);   /* プロトタイプ宣言 */
int main ( int argc, char **argv)
{
        printf ("main 関数スタート¥n");
        disp_msg ( );   /* disp_msg ( )関数の呼び出し */
        printf ("main 関数に戻った¥n");
        return 0;
 }
void disp_msg ( void )
{
        printf ("disp_msg 関数実行¥n");
        return;
 }
```

関数を作るときはプロトタイプ宣言を記述するくせをつけておくとよい.

```
./a.out
main 関数スタート
disp_msg 関数実行
main 関数に戻った
```

実行結果

【例 2】

```
#include <stdio.h>
int calc_add (int , int );   /* プロトタイプ宣言 */
int main ( int argc, char **argv)
{
        int   a, b, c;
        printf ("値１    ");
        scanf (" %d", &a );
        printf ("値２    ");
        scanf (" %d", &b );
        c = tasizan ( a, b );   /* calc_add ( )関数の呼び出し*/
        printf ("%d + %d = %d ¥n", a, b, c);
        return 0;
 }
int calc_add ( int value1, int value2 )
 {
       int result;
       result = value1 + value2;
       return result;
 }
```

```
./a.out
値１    10
値２    22
10 + 22 = 32
```

実行結果

# １１．構造体

| 名前 | 性別 | 年 | 月 | 日 | 身長 | 体重 |
|---|---|---|---|---|---|---|
| A | M | 1980 | 12 | 12 | 163.5 | 62.0 |
| B | F | 1978 | 1 | 23 | 156.3 | 50.0 |
| C | M | 1960 | 3 | 10 | 175.5 | 70.3 |

表１

表１のようなデータを管理したいときにはどうすれば良いか？

× すべての項目に変数を用意する （例： char Aname[20]; char Asex; int Ayear; ..... ）

一人につき７つの変数が必要となる．

人数が増えれば増えるほど変数の数が多くなり，プログラムがわかりにくくミスも増える．

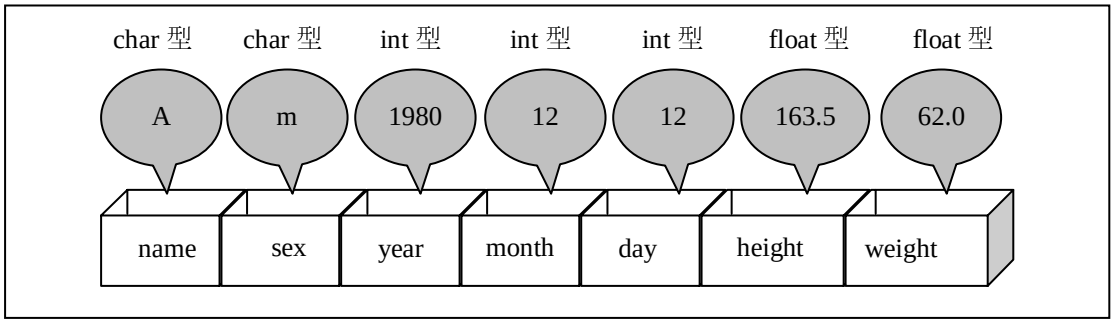
× 配列を利用する （例：char name[3][20]; char sex[3]; int year[3]; ..... ）

項目別に７つの配列を用意する必要がある．

一人のデータを独立した７つの配列で保管するため操作が複雑になる．

○ 構造体を利用する （ 例：struct person hanako; ）

データ型 person として宣言された変数 hanako の例：




構造体とは

・異なるデータ型の変数を一括で管理することができる

・プログラマーが自由に作成することのできるデータ型（構造体の型の宣言）

用語：

構造体タグ　　新に定義したデータ型の名前（例：person）

構造体のメンバー　　構造体に含まれる各変数（例：name, sex, year .....）

●構造体の型の宣言

---

【書式】　**struct**　構造体タグ　{

　　データ型　　メンバー名１；

　　データ型　　メンバー名２；

　　　　:

};

---

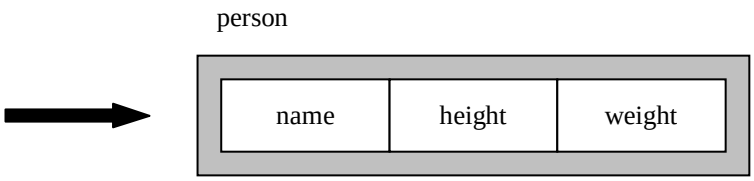
【例】　struct　person　{

　　char　name[20];

　　float　height;

　　float　weight;

};




*) 構造体の型の宣言は，値を入れるための入れ物の設計図を決める作業である．つまり，実際に値を入れるための領域が確保されるわけではない．構造体に値を入れられるようにするには，**構造体型の変数の宣言**を行わなければならない．

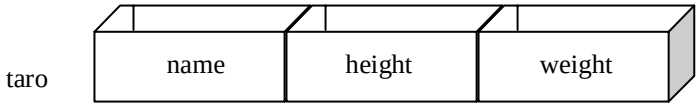
●構造体の宣言

---

【書式】　**struct**　構造体タグ　変数名；

---

【例】　struct　person　taro;




この宣言により，構造体 person 型の入れ物（領域）用意される

●　型の宣言と変数の宣言を同時に行う

構造体の型を宣言すると同時に，その構造体型の変数を宣言することもできる．

次の二つのコードは，どちらも同じ意味を持つ

Ⓐ　型の宣言と変数の宣言を別々に行う

```
struct　person　{
　char　name[20];
　float　height;
　float　weight;
};
struct　person　taro;
```

Ⓑ　型の宣言と変数の宣言を同時に行う

```
struct　person　{
　char　name[20];
　float　height;
　float　weight;
}　taro ;
```

【プログラム例】

```
# include <stdio.h>
int    main (int argc, char **argv)
{
        struct    person    {        /* 構造体の定義 */
                    char    name[20];
                    float    height;
                    float    weight;
        };

        struct    person    taro;     /* 変数の宣言 */

        print ("名前    ");            /* 値の入力 */
        scanf ("%s", taro.name);
        print ("身長    ");
        scanf ("%f", taro.height);
        print ("体重    ");
        scanf ("%f", taro.weight);
        printf("¥n");

        printf ("名前    %s¥n", taro.name);       /* 値の出力 */
        printf ("身長    %f¥n", taro.height);
        printf ("体重    %f¥n", taro.weight);

        return 0;
}
```

構造体のメンバーを参照する方法

【書式】　構造体変数名．メンバー名

左記プログラムの例：

taro.name
taro.height
taro.weight

これらは，通常の変数と同じように扱うことができる．

例：

taro.height = 160.0

また，構造体型の変数は配列を宣言して利用することもできる．

例：

struct　person　human[3];

配列の場合のメンバーの参照法

human[0].name
human[0].height
human[0].weight

---

★ 練習問題 11-1

表１のデータを格納できる構造体配列を用意し，指定した人数分の各データ（名前，性別など）をキーボードから順次入力し，標準出力させるプログラムを作りなさい．

---

▼　構造体を関数に渡す

関数を利用するときの注意点

・main 関数よりも先に，使用する関数を記述する　or 関数のプロトタイプ宣言を記述する．

構造体の型の宣言は関数のプロトタイプ宣言よりも先に行わなければならない．

【プログラム例】

```
# include <stdio.h>

struct    person    {        /*  構造体の定義  */
          char    name[20];
          float    height;
          float    weight;
};

/*  関数のプロトタイプ宣言    */
void    show_struct ( struct    person dat );

int    main (int argc, char **argv)
{

          struct    person    taro;     /*  変数の宣言  */

          print ("名前    ");            /*  値の入力  */
          scanf ("%s", taro.name);
          print ("身長    ");
          scanf ("%f", taro.height);
          print ("体重    ");
          scanf ("%f", taro.weight);
          printf("¥n");

          show_struct (taro);    /*構造体の内容を出力  */
          return 0;
}
/*    show_struct  関数    */
void show_struct (struct person dat)
{
          printf ("名前    %s¥n", dat.name);       /*  値の出力  */
          printf ("身長    %f¥n", dat.height);
          printf ("体重    %f¥n", dat.weight);
          return;
}
```

左のプログラム例では，構造体が保持している値をそのまま関数に渡す方法である．これを<u>値渡し</u>と言うが，この場合，受け取った関数側では，メンバーの値を更新することができない．

受け取った関数側でメンバーの値を更新するためには，<u>参照渡し</u>をしなければならない．

参照渡しの場合

関数の定義

```
void show_struct (struct person *dat)
```

関数内で構造体メンバーを参照する

```
dat->name
dat->height
dat->weight
```

関数を実行するとき

```
show_struct ( &taro);
```

---

★ 練習問題 11-2

上記のプログラムを参照渡しに変更し，show_struct 関数内で構造体メンバーの値が変更できることを確かめてみよう．

---