

プログラミング 参考資料

1. コンピュータとプログラム

1.1. コンピュータとは

{ハードウェア → 本体など
{ソフトウェア → プログラム、ドキュメント (仕様書、操作法などの文書)

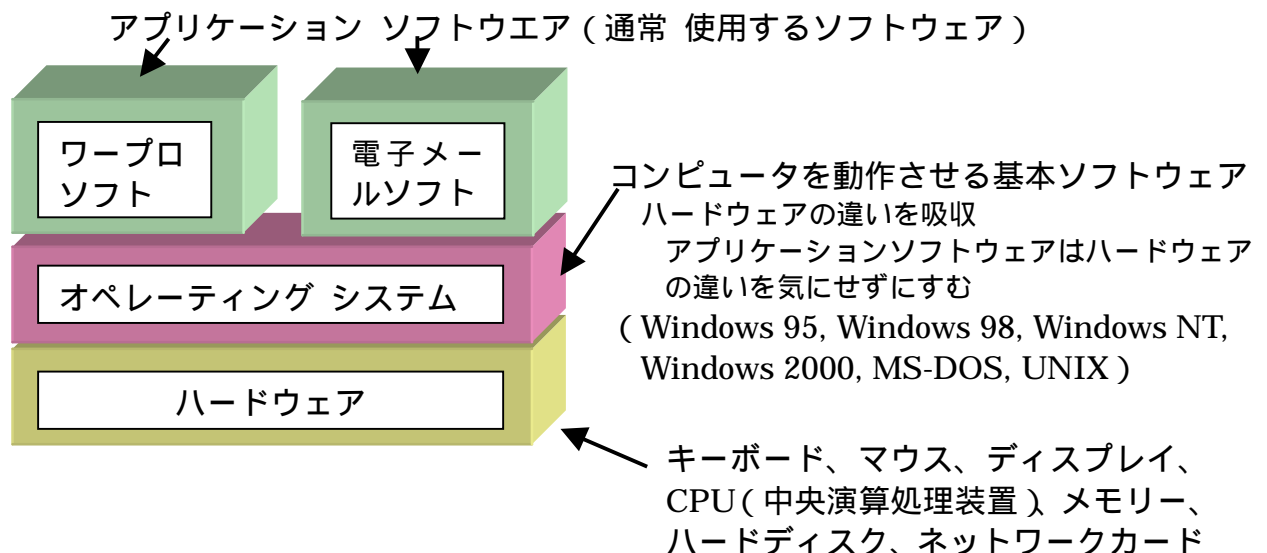
1945年 世界で最初のコンピュータ

1970年末 ~ 1980年頃 パソコン登場

【コンピュータの種類】

- ・ 大型コンピュータ (1960年代 ~) } ⇒ スーパーコンピュータ (超高速計算ができる)
- ・ ミニコン
- ・ ワークステーション (~ 90年代) } ⇒ パソコン (PCワークステーション)
- ・ パソコン (PC) } ⇒ {サーバ...サービスを提供するコンピュータ
{クライアント...サービスを受けるコンピュータ
- ・ PDA(Personal Digital Assistance)...携帯情報端末 (片手で扱えるサイズのデジタル機器の総称)
- ・ 組み込みコンピュータ(携帯電話の中、エアコンの中、洗濯機の中)

【コンピュータの内部構成】



1.2. プログラミング言語

アプリケーション ソフトウェアやオペレーティングシステムなどのソフトウェアを作成するための言語

- ・ アセンブリ言語 (コンピュータが理解できる機械語と直接対応)
- ・ C言語 (幅広く利用されている、教科書 p.12 ~ 13)
 - ・ K & R...古い規格
 - ・ ANSI (米国の標準化団体)、ISO (万国標準化団体)...1985年以來、現在の標準
- ・ B A S I C
- ・ C O B O L (事務処理用)
- ・ F O R T R A N (科学技術計算用だが、あまり用いられなくなった)
- ・ P a s c a l (C言語に似ており、教育用に用いられたが、最近では使われなくなった)
- ・ P r o l o g、L i s p (推論機能を持っており、人工知能研究などに利用されている)
- ・ J a v a (インターネットサービスによく用いられている)

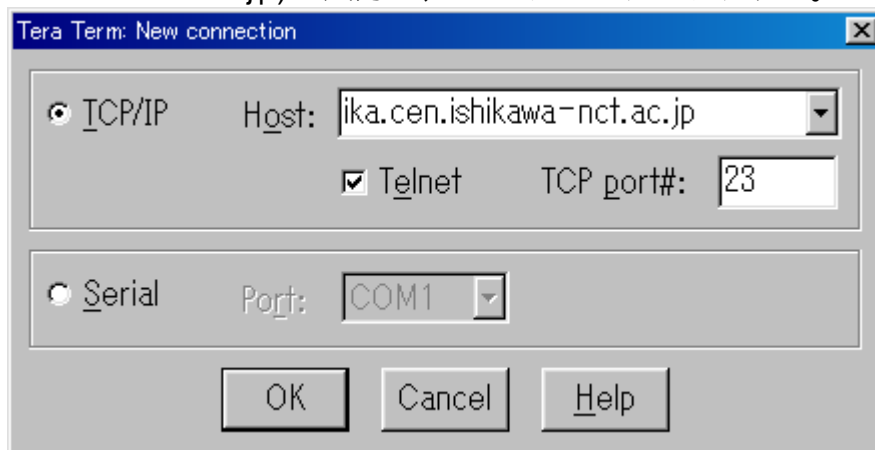
2. C言語の基本的なきまり

2.1. まず使ってみよう

(1) パスワードの変更

パソコンのデスクトップにある Tera Term アイコン  をダブルクリックする。

すると下図のような host 名選択画面が現れるので、Host: 欄に接続したいホスト名 (ika.cen.ishikawa-nct.ac.jp) を入力し、OK ボタンをクリックする。



ログインを促すプロンプトが表示されますから、次のようにユーザー名とパスワードを入力します。

```
login : i02100      --- ユーザー名の入力(学籍番号の前に i をつける)
password :         --- パスワードの入力

%                 --- 命令の入力を促すプロンプトが表示
```

まだパスワードを登録していない場合や変更したい場合は以下の手順でパスワードを設定します。

```
% yppasswd
Changing NIS password for kousen on ika.
Old password:      --- 今までのパスワードを入力します。
                   (初回は、必要ありません)
New password:     --- 新しいパスワードを入力します。
Retype new password: --- もう一度、新しいパスワードを入力します。
NIS entry changed on uni.

%
```

これでパスワードが登録 / 変更できました。

```
% exit
```


これでログアウトしますから、もう一度ログインしてみてください。

(2) Cプログラムの作成、実行 (Visual C を使用した場合)

1. 起動 (デスクトップの以下のアイコンをダブルクリック)

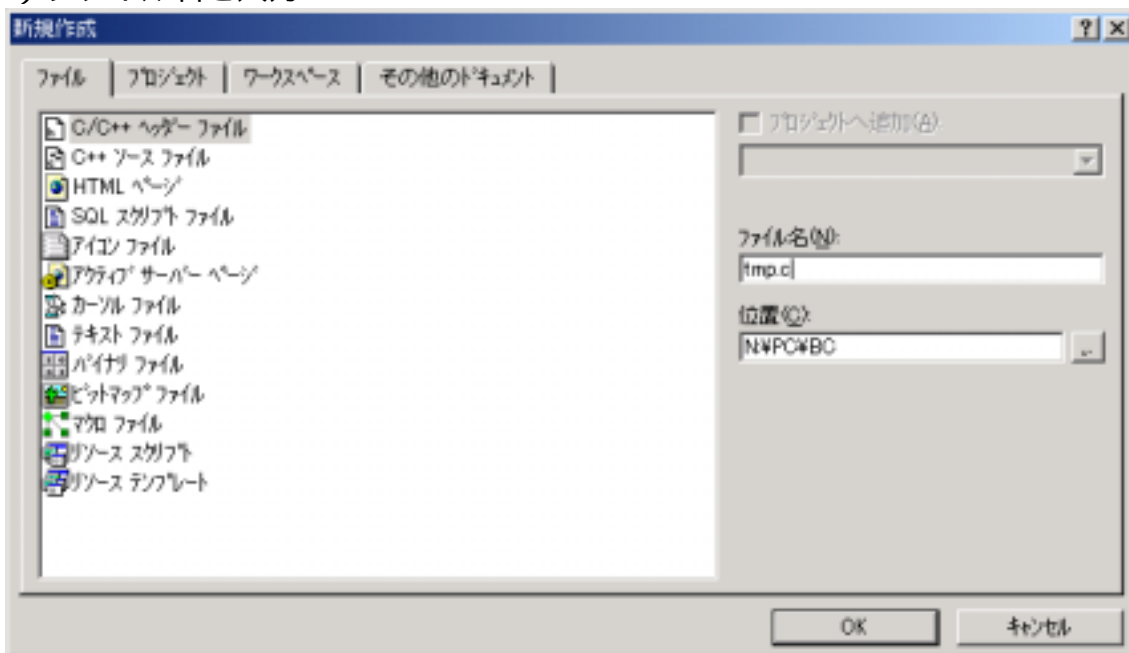


2. 新規 Cプログラムの作成

(a) ファイルメニューの新規作成を選択 (ファイル - 新規作成) 

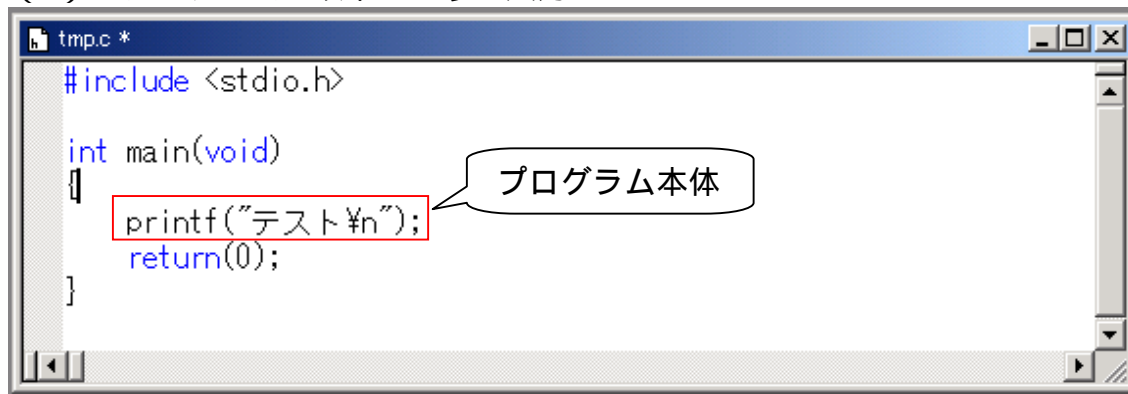
(b) 以下のようにファイルタグの位置に N:\¥PC¥BC を入力


(c) ファイル名を入力




3. Cプログラムの編集

(a) Cプログラムを以下のように入力

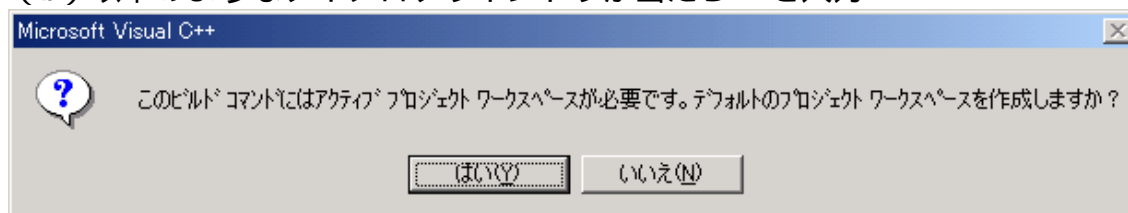


(b) ファイル - 上書き保存 

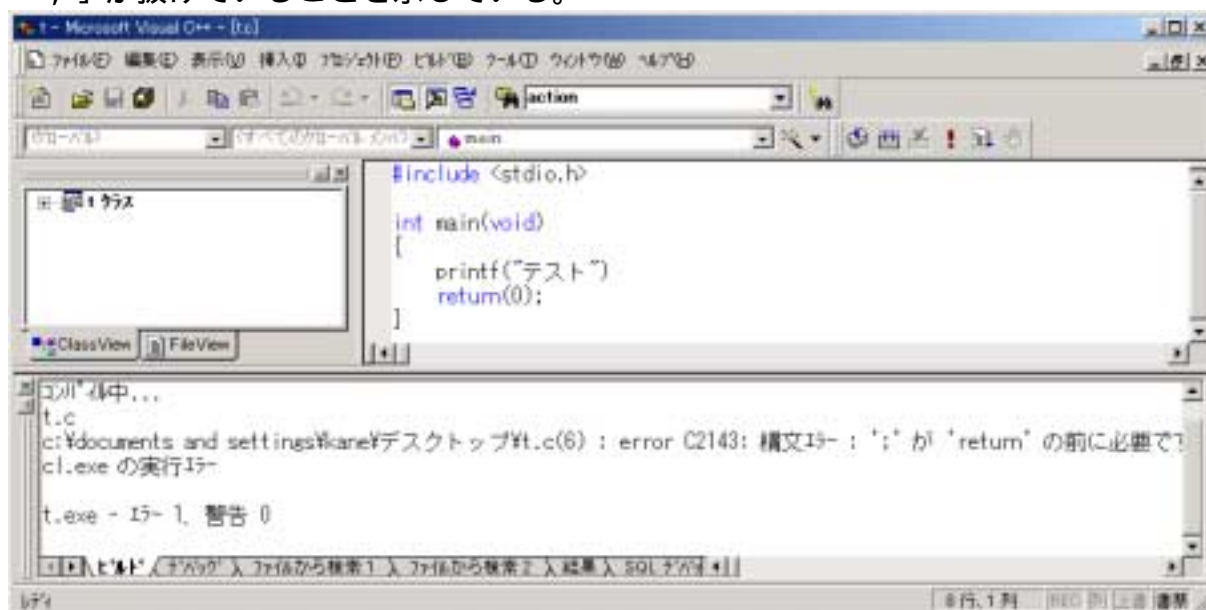
4. コンパイル&リンク

(a) ビルド - ビルド (コンパイル&リンク)  (F7)

(b) 以下のようなダイアログウィンドウが出たら Y を入力

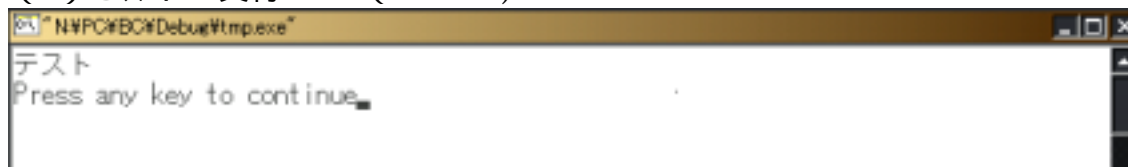


(c) もし、以下のようにエラーが表示されたときには、メッセージに従って誤りを修正後、(1)のコンパイル&リンクをやり直す。以下の例では 6 行目の前の行末に「;」が抜けていることを示している。



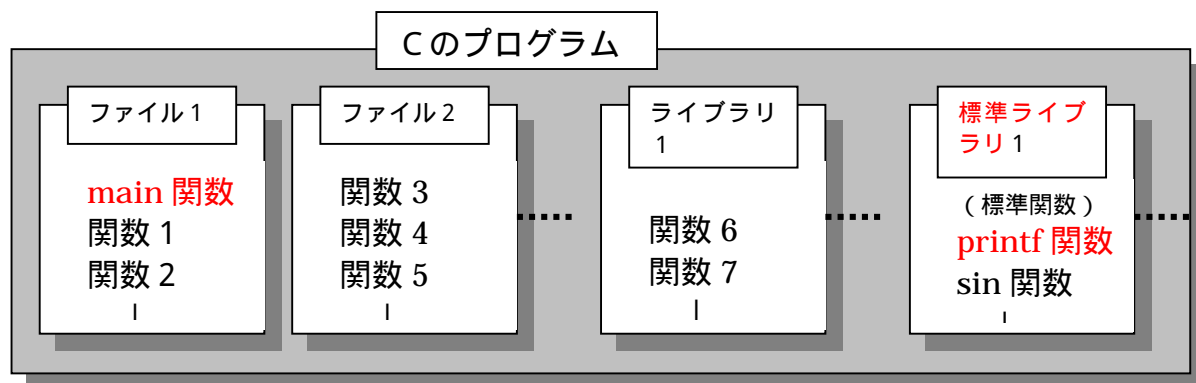
5. 実行

(a) ビルド - 実行  (ctrl+F5)



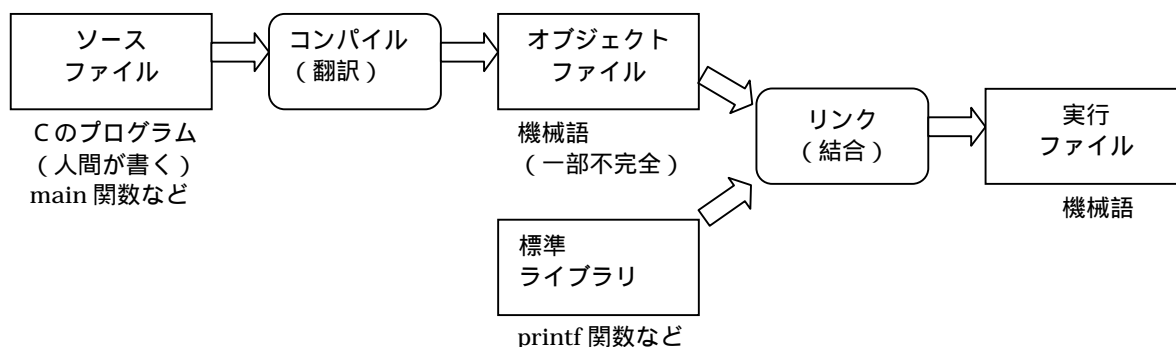
2.2. Cプログラムの構成

以下の図に示すようにCプログラムは、関数の集まりからなっています。関数には main 関数のように自分で作成する関数と予め用意されている関数(標準関数)があります。これらの関数はファイル(ひとかたまりのデータの集まりをファイルという。)またはライブラリの中に記述されています。簡単のため、当面は main 関数だけのファイルと標準関数で構成されるプログラムを扱います。



2.3. 実行プログラムの作成

Visual C などを利用すると自動的に以下のような手順で実行プログラムが作成されます。まず人間が書いたC言語のプログラムを機械語(機械が理解できる言葉)に翻訳(コンパイル)し、オブジェクトファイルを作成します。このオブジェクトファイルは 再利用しやすいように わざと一部不完全な機械語になっています。次に必要なオブジェクトファイルやライブラリ(オブジェクトプログラムをまとめたもの)を接続(リンク)すると、実行可能な機械語プログラム(実行プログラム)ができます。



2.4. 簡単なCプログラム例

以下は、標準出力(通常はディスプレイ)に「テスト」と表示するプログラムです。プログラムは、コメントから書くようにしましょう。コメントの始まりは/*、終わりは*/で指定します。コメントには、何を書いてもかまいませんが、何をやるプログラムか?、使用法、入力と出力、作成日時や更新日時、作者などを必ず書きましょう。

printf()は標準出力(通常はディスプレイ)に文字列を表示する標準関数です。ここでは、「テスト」と表示します。「テスト」に続く¥n は改行を表しています。

pro2_3.c

コメントの始まり

```
/* pro2_3.c  
   テストプログラム  
   標準出力（通常はディスプレイ）に「テスト」と表示する。  
   作成日時：2000年4月16日  
   作成者： 111番 石川太郎
```

コメントの終わり

```
*/
```

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

stdio.h という名前のファイルの内容をインクルードします(含めます)。この場所に stdio.h ファイルの内容を直接書いた場合と同じこととなります。stdio.h にはC言語の基本的な標準関数を使うために必要な情報が書かれています。

```
int main(void)
```

main 関数の始まりであることを示しています。括弧の中の void は、この関数への入力がないことを表しています。最初の int はこの関数の戻り値が整数であることを表しています。

```
{  
    printf("テスト\n");  
    return(0);  
}
```

```
    printf("テスト\n");  
    return(0);
```

関数の戻り値として整数 0 を返して、この関数を終了します。


C言語のプログラムは、関数の集まりでできています。上のプログラムは、自分で書く main 関数と文字列を表示する標準関数 (printf 関数) の集まりでできています。また、関数は文の集まりでできています。上のプログラムにもあるように文の終わりにはセミコロン「;」をつけます。

2.5. 課題

(1) 以下のプログラムを入力、実行せよ。それぞれのプログラムの動作を考えながら行うこと。ALT キーを押しながら左上の「~」キーを押すことにより日本語入力できる状態 (日本語入力モード) になる。再び ALT キーを押しながら左上の「~」キーを押せば、元の英数字入力モードに戻る。

(2) pro2_4_2.c において、一番最初の printf 文の右側の「;」を削除してプログラムをコンパイルせよ。どんなエラーメッセージがでるか記録せよ。

(3) (2)で変更したソースファイルをもとにもどし、pro2_4_2.c の最初の printf 文の右側の「"」を削除して、プログラムをコンパイルせよ。どんなエラーメッセージができるか記録せよ。

(4) (3)で変更したソースファイルをもとにもどし、今度は pro2_4_2.c の最初の printf 文の前に全角の空白を入れよ。コンパイルすると、どんなエラーメッセージができるか記録せよ。全角の空白を入力するには、まず画面右下のペンのアイコンを右クリック(右ボタンを押す)し、日本語入力システムの設定を選択する。つぎに設定のウィンドウが現れるので、基本設定 - スペースの項目を全角スペースに変更する。その後、日本語入力モードにして、空白を一つ入れると、全角スペースが入力できる。全角スペース入力後は、基本設定 - スペースの項目を半角スペースに必ず戻しておくこと。

```
/*-----  
pro2_4_1.c  
俳句を画面に表示する。  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    printf("松島や  ああ松島や      松島や¥n");  
    return(0);  
}  
  
/*-----  
pro2_4_2.c  
俳句を画面に表示する。  
( printf を複数にわけて出力する例 )  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    printf("松島や");  
    printf("  ああ松島や");  
    printf("      松島や¥n");  
    return(0);  
}
```

```

/*-----
pro2_4_3.c
3行に分けて画面上に俳句を出力する。
-----*/
#include <stdio.h>
int main(void)
{
    printf("松島や      ￥n");
    printf(" ああ松島や￥n");
    printf("      松島や￥n");
    return(0);
}

```

```

/*-----
pro2_4_4.c
3行に分けて画面上に俳句を出力する。
(一つのprintf文を用いた場合)
-----*/
#include <stdio.h>
int main(void)
{
    printf("松島や￥n ああ松島や￥n      松島や￥n");
    return(0);
}

```

課題（２）の説明

```
  |
9行目 printf("松島や")
10行目 printf("  ああ松島や");
  |
```

[エラーメッセージ]

```
pro2_4_2.c: In function `main':
```

```
pro2_4_2.c:10: parse error before `printf'
```

C言語はフリーフォーマットなので、10行目で初めて9行目のprintfに対応する文末の;がないことがわかる。

C言語はフリーフォーマット

(1) 1行に2つ以上の文を書いてもO.K.

```
printf("松島や"); printf("  ああ松島や");
```

(2) 2行以上に1つの文を書いてもO.K.

```
printf("松島や")
```

```
;
```

```
printf(
```

```
"  ああ松島や"
```

```
);
```

(3) 1行に全部書いてもO.K.

```
int main(void){printf("松島や");printf("  ああ松島や");return(0);}
```

しかし、見にくく、誤りの原因となるので以上のような書き方をしないこと。

課題（３）の説明

[エラーメッセージ]

```
pro2_4_2.c:11: unterminated string or character constant
```

```
pro2_4_2.c:9: possible real start of unterminated constant
```

11: 閉じていない文字列または文字定数がある

9: 文字定数の可能性のある実際の始まり

課題（４）の説明

[エラーメッセージ]

```
pro2_4_2.c: In function `main':
```

```
pro2_4_2.c:9: parse error before character 0241
```

9: 文字0241（全角スペースに対応する文字番号）の前で構文エラー

全角スペースは漢字であるため、エラーとなる。全角スペースは見つけにくいので注意すること。

2.6. エスケープシーケンス

```
/*-----
pro2_6_1.c
 エスケープシーケンスを理解する。
 二重引用符「"」を出力するには?
-----*/
#include <stdio.h>
int main(void)
{
    printf("She said, ¥\"I like C.¥\"¥n");
    return(0);
}

/*-----
pro2_6_2.c
 エスケープシーケンスを理解する。
 ビープ音を鳴らす
-----*/
#include <stdio.h>
int main(void)
{
    printf("¥a");
    return(0);
}

/*-----
pro2_6_3.c
 エスケープシーケンスを理解する。
 「タブスペース」を使う。
-----*/

#include <stdio.h>
int main(void)
{
    printf("英語¥t 日本語¥t 色¥n");
    printf("orange¥t みかん¥t 黄色¥n");
    printf("apple¥t りんご¥t 赤色¥n");
    printf("grape¥t ぶどう¥t 紫色¥n");
    return(0);
}
```

エスケープシーケンス一覧

記号	文字	意味
¥a	警告(ベル)文字	ベルを鳴らす。
¥n	改行	改行する。
¥t	タブ	タブ位置にカーソルを移動。
¥¥		¥を表示。
¥?	疑問符	?を表示。
¥'	1重引用符	'を表示。
¥"	2重引用符	"を表示。
¥ooo	3桁の8進数	与えられた任意の8進数を表示。
¥xhh	2桁の16進数	与えられた任意の16進数を表示。

2.7. 変数を使ったプログラム

変数...データを覚えておくための箱

変数の型...箱の種類

文字型 char 1文字の英数字を覚えることができる。(1バイト)

整数型 short -32,768 ~ 32,767($2^{15} - 1$)までの整数を記憶できる。(2バイト)

long -2,147,483,648 ~ 2,147,483,647($2^{31} - 1$)までの整数を記憶できる。
(4バイト)

int 処理系によって記憶できる大きさが異なる。UNIXではlongと同等。
(4バイト)

実数型 float $\pm 3.4 \times 10^{-38} \sim \pm 3.4 \times 10^{38}$ の範囲の実数(小数点を持つ数)を記憶できる形式で、10進数で有効数字約7桁である。(4バイト)

double $\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{308}$ の範囲の実数を記憶できる形式で、10進数で有効数字約15~16桁である。(8バイト)

long double $\pm 3.4 \times 10^{-4932} \sim \pm 1.7 \times 10^{4932}$ の範囲の実数を記憶できる形式である。(10バイト)

Printf()の書式

%d int型を表示
 %ld long型を表示
 %f double型、float型を表示
 %lf long double型を表示
 %c char型(文字型)を表示
 %s 文字列を表示

例

```
/*-----  
pro2_7_1.c  
 2つの整数型の変数の和、差、積、商、剰余を求める。  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    int x, y;          /* x, y という名前の整数型の箱 (変数) を2つ用意する */  
                      /* 宣言の後は1行空けると見やすい */  
    x = 8;             /* x に8を代入 */  
    y = 3;             /* y に3を代入 */  
    printf("x+y=%d\n", x + y);      /* x と y の和を表示 */  
    printf("x-y=%d\n", x - y);      /* x と y の差を表示 */  
    printf("x×y=%d\n", x * y);      /* x と y の積を表示 */  
    printf("x÷y=%d\n", x / y);      /* x を y で割ったときの商を表示 */  
    printf("x%%y=%d\n", x % y);     /* x を y で割ったときの余りを表示 */  
    return(0);  
}
```

<実行結果>

x+y=11

x-y=5

x×y=24

x÷y=2 (整数同士の割り算では小数点以下は切り捨てられる。)

x%y=2

```
/*-----  
pro2_7_2.c  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    int a;             /* a という名前の整数型の箱 (変数) を用意する */  
  
    a = 3 + 4 * 2 + 2 / 3;      /* 3 + 4×2 + 2/3 を計算し a に代入 */  
    printf("a=%d\n", a);       /* a の値を表示 */  
    return(0);  
}
```

<実行結果>

a=11 (a は整数しか記憶できないので、小数点以下は切り捨てられる。)

```

/*-----
pro2_7_3.c
-----*/
#include <stdio.h>
int main(void)
{
    double    a;          /* a という名前の実数型の箱 (変数) を用意する */

    a = 3 + 4 * 2 + 2 / 3;    /* 3 + 4×2 + 2/3 を計算し a に代入 */
    printf("a=%f\n", a);    /* a の値を表示 */
    return(0);
}

```

<実行結果>

a=11.000000

(整数 ÷ 整数は整数になってしまうため、2 / 3 の部分が小数点以下(0.6666...)を切り捨てて0になってしまう。)

```

/*-----
pro2_7_4.c
-----*/
#include <stdio.h>
int main(void)
{
    double    a;          /* a という名前の実数型の箱 (変数) を用意する */

    a = 3 + 4 * 2 + 2.0 / 3.0;    /* 3 + 4×2 + 2/3 を計算し a に代入 */
    printf("a=%f\n", a);    /* a の値を表示 */
    return(0);
}

```

<実行結果>

a=11.666667

(実数 ÷ 実数は実数と考えるため、2 .0/ 3.0 の部分は 0.6666...となる。最後の桁は四捨五入される。)

変数名

- ・変数名として使用できる文字 ... a~z, A~Z, 0~9, _(アンダーバー)
- ・変数名の最初の文字 ... 英字またはアンダーバー
- ・変数名の長さ ... 32文字までが有効
- ・予約語(教科書 P.16~17 に書いてあるもの)は、使用不可
- ・大文字と小文字は区別される

変数の記憶サイズとアドレス

データ型	記憶サイズ	表現範囲
short	2 バイト	-32,768(2^{15}) ~ 32,767($2^{15}-1$) の整数
int	2 または 4 バイト	
long	4 バイト	-2,147,483,648(2^{31}) ~ 2,147,483,647($2^{31}-1$) の整数
float	4 バイト	$\pm 3.4 \times 10^{-38} \sim \pm 3.4 \times 10^{38}$ の範囲の実数
double	8 バイト	$\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{308}$ の範囲の実数
char	1 バイト	1 文字の英数字

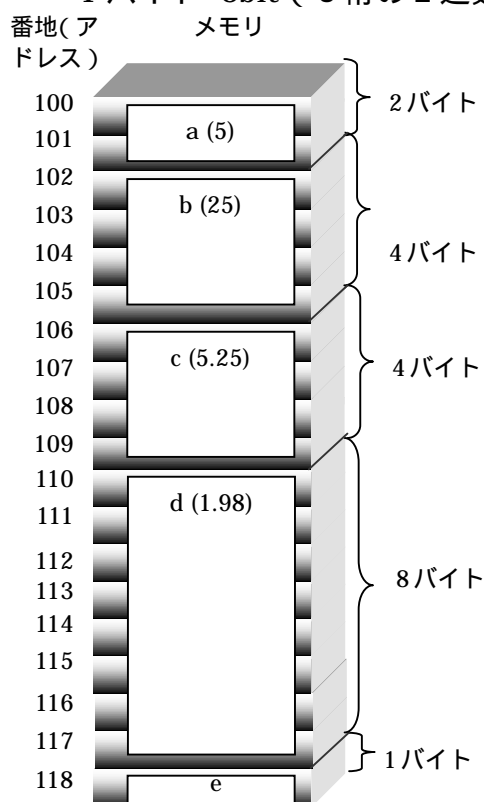
1 バイト=8bit (8 桁の 2 進数)

(1) データ型によって、記憶に必要なメモリ (記憶装置) の占有量が上の表のように異なる。

(2) 各変数は、メモリ上のどこかに記憶されている。その場所のことをアドレス (番地) という。例えば、以下のように宣言すると、各変数は右図のように記憶される。ただし、どの変数をどのアドレス (番地) に割り当てるかは、オペレーティングシステムによって行われる。

```
short    a=5;
long     b=25;
float    c=5.25;
double   d=1.98;
char     e;
```

例えば、変数 b は、102 番地から始まる long 型の箱に割り振られている。



練習

(1) 以下のプログラムの実行結果を予想せよ。

```
/*-----  
pro2_7_5.c  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    int x, y, z;      /* x, y, z という名前の整数型の箱 (変数) を用意する */  
  
    x = 8;           /* x に 8 を代入 */  
    y = 3;           /* y に 3 を代入 */  
    z = x + y;  
  
    printf("(1) %d\n", 11);  
    printf("(2) %d\n", 8 + 3);  
    printf("(3) %d\n", x + y);  
    printf("(4) %d\n", z);  
  
    printf("(5) x+y=%d\n", z);  
    printf("(6) 8+3=%d\n", z);  
    printf("(7) %d+%d=%d\n", x, y, z);  
    printf("(8) %d+%d=%d\n", x, y, x + y);  
    return(0);  
}
```

(2) 以下のプログラムの実行結果を予想せよ。

```
/*-----  
pro2_7_6.c  
-----*/  
#include <stdio.h>  
int main(void)  
{  
    double x, y, z;  /* x, y, z という名前の実数型の箱 (変数) を用意する */  
    int    a;        /* a という名前の整数型の箱 (変数) を用意する */  
  
    x = 1.2;         /* x に 1.2 を代入 */  
    y = 1.7;         /* y に 1.7 を代入 */  
    z = x + y;  
  
    printf("(1) %f\n", 1.2 + 1.7);  
    printf("(2) %f\n", x + y);  
    printf("(3) %f\n", z);  
    printf("(4) %f+%f=%f\n", x, y, z);  
    printf("(5) %d\n", (int)z); /* (int)は整数型への型変換を行う(切り捨て) */  
    printf("(6) %d\n", (int)(z + 0.5)); /* (四捨五入) */  
    return(0);  
}
```

(3) 円の半径 1.5[cm]を double 型の変数 r に代入し、円周[cm]、円の面積[cm²]を求めよ。ただし円周率は 3.1415 とする。

(4) 直方体の 3 辺の長さが 1.2[cm] , 2.3[cm] , 3.4[cm] であるとき、各辺の長さを double 型の変数 x , y , z に代入し、体積を求めるプログラムを作成せよ。

(5) 42.195[km]の距離を時速 4[km/h]で歩いたとする。何時間要するかを求めるプログラムを作成せよ。

[ヒント] 距離を x[km]、速さを v[km/h]、要する時間を t[h]としたとき、

$$t = x / v$$

である。

【研究課題】(5) の答えが 時間 分と表示されるように改良せよ。

[ヒント] 要する時間を t[h]としたとき、以下の関係がある。

60 * t 要する時間[分]

(int)(60 * t) 要する時間[分]を切り捨て

(int)(60 * t + 0.5) 要する時間[分]を四捨五入

また、要する時間[分]を四捨五入したものが整数型の変数 m に代入されているとき、

m = (int)(60 * x / v + 0.5) ;

m / 60 要する時間[時間]

m % 60 残りの分 (m を 60 で割った余り)

で 時間 分が求まる。

2.8. キーボードからのデータ入力： scanf

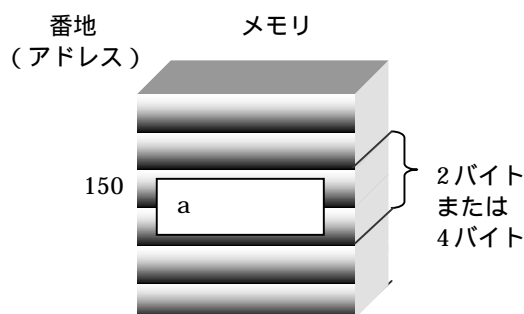
例 1

キーボードから入力した値（整数）を int 型の変数 a に格納する（記憶させる）。

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

まず int a; を宣言することによって、右図のように整数型 (int 型) の箱が用意されます。この箱のアドレス (番地) が例えば 150 番地であるとしてみます。

次に scanf 関数はキーボードなどから入力した値を変数に記憶させる関数です。scanf 関数の中の %d は int 型の値を入力することを表しています。また &a は、変数 a のアドレスを表しています。すなわち & 記号はアドレスという意味で、この場合、&a (変数 a のアドレス) は 150 番地です。よって、scanf("%d", &a) は、キーボードから入力される整数 (%d) を、150 番地 (&a) から始まる整数型の箱 (変数 a) に代入するという意味になります。



scanf() の書式

- %hd** short 型を入力
- %d** int 型を入力
- %ld** long 型を入力
- %f** float 型を入力
- %lf** double 型を入力 (printf のときと違うので注意)
- %Lf** long double 型を入力
- %c** char 型 (文字型) を入力
- %s** 文字列を入力

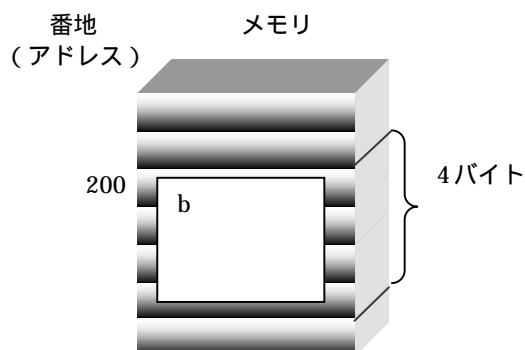
例 2

キーボードから入力した値（実数）を float 型の変数 b に代入する。

```
float  b;  
scanf("%f", &b);
```

float b; を宣言することによって、右図のように float 型の箱が用意され、この箱のアドレス (番地) が例えば 200 番地であるとしてみます。

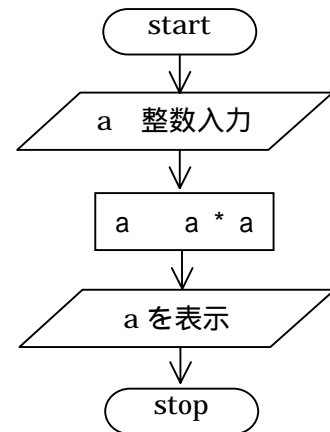
次に scanf("%f", &b) で、キーボードから入力される実数 (%f) を、200 番地 (&b) から始まる float 型の箱 (変数 b) に代入します。



例 3

```
/*-----  
pro2_8_3.c  
キーボードから1つの整数を入力し、その2乗を表示する。  
-----*/  
  
#include <stdio.h>  
int main(void)  
{  
    int a;  
  
    /* 入力(キーボードから1つの整数を入力) */  
    printf("整数を1つ入力して下さい: ");  
    /* 何を入力すればよいか必ず指定すること */  
    scanf("%d", &a);  
  
    /* 処理(2乗を求める) */  
    a = a * a;  
  
    /* 出力 */  
    printf("入力された整数の2乗は%dです。¥n", a);  
    return(0);  
}
```

フローチャート(流れ図)



<実行結果例>

```
整数を1つ入力して下さい: 3  
入力された整数の2乗は9です。
```

2.9. 課題

- (1) キーボードから円の半径[cm](実数)を入力し、円周[cm]、円の面積[cm²]を求めよ。
- (2) 2つの整数を入力し、その和、差、積、商、余りを求めよ。
- (3) 2つの実数 x, y を入力し、x+y, x-y, x×y, x÷y を求めよ。
- (4) 直方体の3辺の長さを入力し、体積を求めよ。
- (5) 距離 x, 時速 v を入力し、x[km]の距離を v[km/h]で移動したときに、何時間何分要するかを求めよ。

2.10. 出力桁数の指定 (printf)

(1) 整数型

```
int a = 19;
printf("%d\n", a);      /* 左端に表示 */ 1 9
printf("%4d\n", a);    /* 4桁で表示 */  1 9
printf("%7d\n", a);    /* 7桁で表示 */   1 9
```

```
int a = 56, b=619, c=1234;
printf("%d\n", a);     /* %d で表示 */  5 6
printf("%d\n", b);     6 1 9
printf("%d\n", c);     1 2 3 4
```

```
printf("%4d\n", a);    /* %4d で表示 */   5 6
printf("%4d\n", b);    6 1 9
printf("%4d\n", c);    1 2 3 4
```

(2) 実数型

```
double a = 1.5;
printf("%f\n", a);     /* 左端に表示 */ 1 . 5 0 0 0 0 0
/* 全部で6桁、小数点以下2桁で表示 */
printf("%6.2f\n", a);  1 . 5 0
```

【指数表現】

200	2×10^2	2.00 E 02
20000	2×10^4	2.00 E 04
2000000000	2×10^9	2.00 E 09
0.02	2×10^{-2}	2.00 E-02
0.000000002	2×10^{-9}	2.00 E-09

```
double a = 0.0000012345;
printf("%e\n", a);     1 . 2 3 4 5 0 0 e - 0 6
printf("%E\n", a);     1 . 2 3 4 5 0 0 E - 0 6
```

2.11. 数学関数の利用

$\sqrt{3}$... `sqrt(3)`
 x^y ... `pow(x, y)`
 e^x ... `exp(x)`
 $\sin x$... `sin(x)`
 $\cos x$... `cos(x)`
 $\tan x$... `tan(x)`

例 1

```
/*-----  
pro2_11.c  
平方根を表示する例。  
-----*/  
#include <stdio.h>  
#include <math.h>          /* 数学関数を使用する際に必要 */  
int main(void)  
{  
    double x, y;  
  
    printf("平方根を計算します。¥n 実数を 1 つ入力して下さい: ");  
    scanf("%lf", &x);  
  
    y = sqrt(x);  
    printf("%f の平方根は%f です。¥n", x, y);  
  
    y = 3.0 * sqrt(x) + 2.0;  
    printf("3  %f + 2 = %f¥n", x, y);  
  
    printf("%f の平方根は%f です。¥n", x, sqrt(x));  
    printf("3  %f + 2 = %f¥n", x, 3.0 * sqrt(x) + 2.0);  
  
    return(0);  
}
```

練習

実数 x, y を入力し、 x^y を求めるプログラムを作成せよ。

2.12. 文字と文字コード

(1) ASCII コード (American Standard Code for Information Interchange) P.38

文字 (英数字、記号)	数字 (7bit, 0 ~ 127)
1	49
2	50
A	65
B	66
a	97

31 以下 特殊機能

32 ~ 127 英数字、記号

(2) 日本語コード

現在使用されている主要な文字コードは、J I Sコード、S J I Sコード、E U Cコード、U n i c o d eの4つです。

J I Sコード 電子メール等、インターネットで良く用いられているコード
エスケープシーケンスにより、文字の種類を決めます。

エスケープシーケ ンス	1 バイト 目	2 バイト 目	文字の種類
1B 28 42 ESC (B	20-7E		A S C I I
1B 28 4A ESC (J	20-7E		J I S ローマ字 (ASCII のバックスラ ッシュが¥になった文字集合)
1B 24 40 ESC \$ @	21-7E	21-7E	旧 J I S 漢字 (1978 年制定)
1B 24 42 ESC \$ B	21-7E	21-7E	新 J I S 漢字 (1983 年制定)
1B 28 49 ESC (I	21-5F		J I S カナ

シフトJ I Sコード (S J I S) Microsoft が決めたコードで Windows や
MS-DOS や Mac などで使用されています。

エスケープシーケンスを使用しない。

半角カナを1バイトで扱えるため、表示桁数と内部バイト数が一致する。

第1バイトは必ず最上位ビットが1ですが、第2バイトは1でないこともあるので、
逆方向文字列検索プログラムをする場合は、注意が必要。

E U C (Extended Unix Code) 日本語U N I Xで使われているコード

エスケープシーケンスを使用せず、A S C I I以外の文字は、J I Sコードの上位
ビットを1にすることにより識別。

U n i c o d e Windows NT や Java などで使用されているコード

例 1 文字型(char 型)

```
char    m;  
m = 'A';          /* 1バイトの文字型変数 m に 'A' (65) を代入 */  
printf("%c\n", m); /* A が表示される */  
m = m + 1;  
printf("%c\n", m); /* B(66) が表示される */
```

(注 1) `m = 'A';` の代わりに

```
m = 65;          /* 10進数 */  
または  
m = 0x41;       /* 16進数 */  
と書いてもよい。
```

(注 2) 'A' A という文字を表す
"A" A という文字列を表す
"ABC" ABC という文字列を表す

例 2 文字の入力 (入力した大文字を小文字に変換)

```
char    moji;  
  
printf("大文字を 1 文字入力して下さい(A ~ Z)\n");  
scanf("%c", &moji);  
  
printf("%c の小文字は%c です。 \n", moji, moji - 'A' + 'a');
```

(参考) 小文字 大文字 - 'A' + 'a'
 'b' 'B' - 'A' + 'a'

大文字 小文字 - 'a' + 'A'
 'B' 'b' - 'a' + 'A'

小文字	大文字	小文字 - 大文字 (差)
a(97)	A(65)	32 ('a' - 'A')
b(98)	B(66)	32
z(122)	Z(90)	32

例 3

1文字単位の入出力

aを表示するには

```
printf("a");
```

または、

```
putchar('a');
```

同様に文字変数 moji を表示するには

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

または、

```
putchar(moji);
```

1文字入力するには

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

または、

```
moji = getchar();
```

(注)

```
printf("a¥n");
```

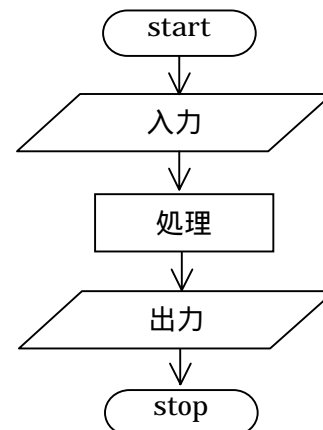
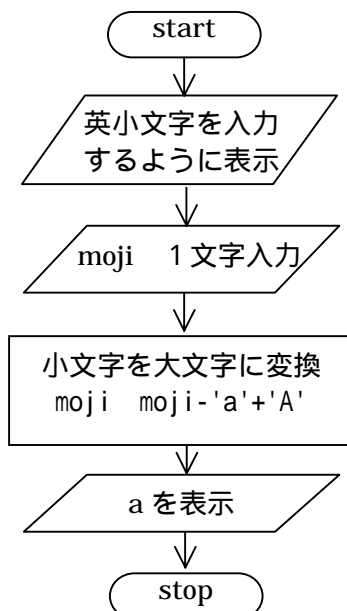
||

```
putchar('a');
```

```
putchar("¥n");
```

練習

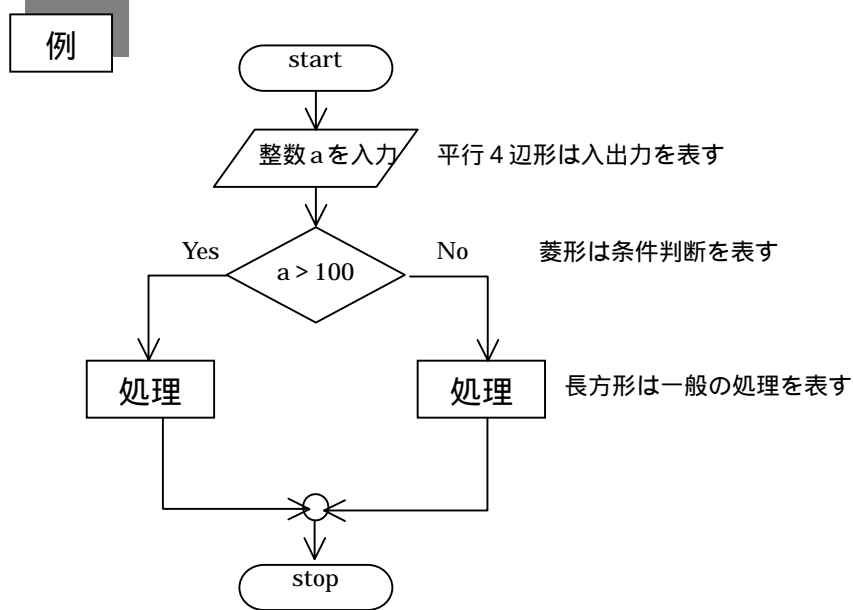
putchar(), getchar()を用いて、キーボードから入力した小文字を大文字に変換するプログラムを作成せよ。



3. 処理の流れの制御

3.1. 処理の流れの表現

(1) フローチャート (流れ図) 教科書 P.46



(2) SPD (NEC などで使用されている)

(3) PAD (日立)

(4) HCP チャート (NTT)

3.2. 構造化プログラム

基本的な制御構造だけを用いて、理解しやすいプログラムを作成する方法

信頼性が高くなる

わかりやすくなる

保守・拡張が容易

基本的な制御構造

接続 (これまで使用)

判断 (条件判定) if else

多方向分岐 switch case

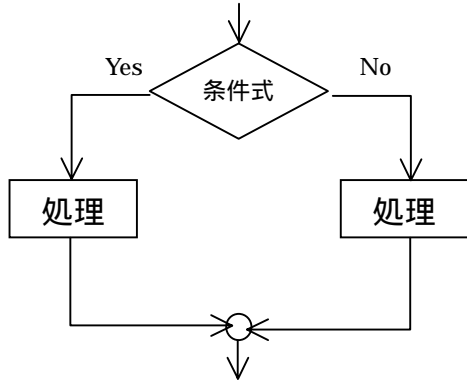
前判定反復 while

後判定反復 do while

所定回反復 for

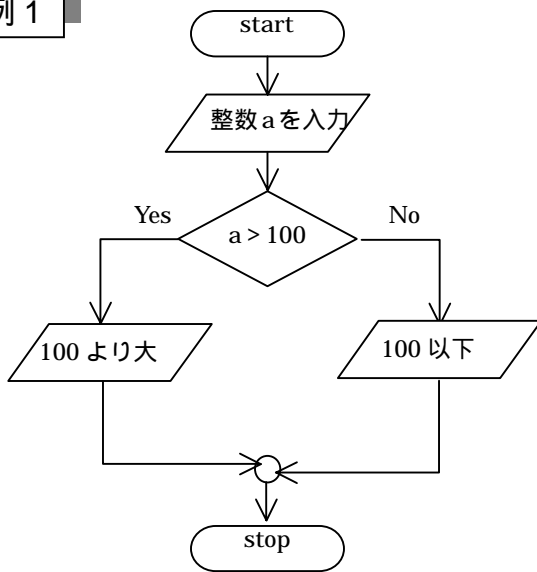
3.3. if else 文 (条件判断) P.48

(1) 書式



```
if (条件式) {
    処理
}
else {
    処理
}
```

例 1



```
/*-----
pro3_3_1.c   if else 文
整数を入力し 100 より大きいかどうかを判定
-----*/
#include <stdio.h>
int main(void)
{
    int a;

    printf("整数を1つ入力して下さい: ");
    scanf("%d", &a);

    if(a > 100){
        printf("100 より大\n");
    }
    else {
        printf("100 以下\n");
    }

    return(0);
}
```

(注) 処理 1、処理 2 が 1 文しかないときは、以下のように { } を省略できるが、間違えやすいので当面使用しない。

```
if(a > 100)
    printf("100 より大\n");
else
    printf("100 以下\n");
```

(2) 条件式

$a > 100$ a が 100 より大きい

$a < 100$ a が 100 より小さい

$a == 100$ a が 100 に等しい

$a \geq 100$ a が 100 以上

$a \leq 100$ a が 100 以下

$a != 100$ a が 100 に等しくない

$0 \leq a \ \&\& \ a \leq 100$ a が 0 以上で、かつ(AND) 100 以下

【注意】 $0 \leq a \leq 100$ という表現はだめ。

$a < 0 \ \|\| \ a > 100$ a が 0 より小さいかまたは(OR) a が 100 より大きい

$!(0 \leq a \ \&\& \ a \leq 100)$ a が 0 以上かつ 100 以下でない

$'A' \leq c \ \&\& \ c \leq 'Z'$ 文字変数 c が 'A'(65)以上でかつ'Z'(90)以下
(アルファベットの大文字)

【練習】

(a) 教科書 P.50 練習問題 6

(b) a が 0 以上 100 以下のとき「範囲内」と表示し、それ以外のとき「範囲外」と表示

(c) P.52 問 4

(d) P.52 例題 7

(e) P.52 練習問題 7

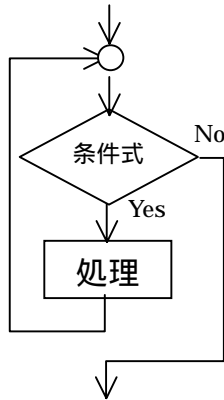
(f) キーボードから西暦を入力し、うるう年かどうか判定する。

うるう年: 4 で割り切れる かつ (100 で割り切れない または 400 で割り切れる)

(g) 2 次方程式 $ax^2 + bx + c = 0$ の係数 a, b, c を入力すると解を出力する。

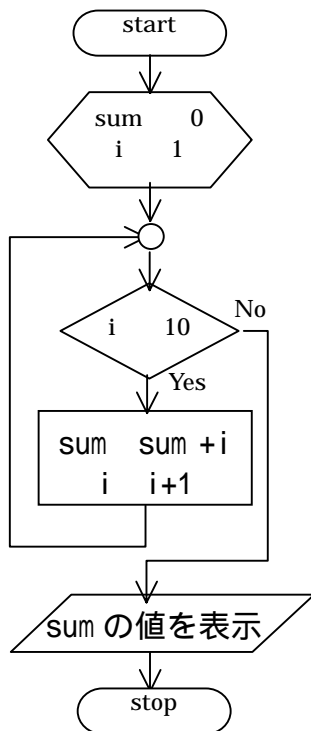
3.4. while 文 (繰り返し処理 1) P.56

(1) 書式



```
while (条件式) {
    処理
}
```

例 1



```
/*-----
pro3_4_1.c   while 文
1+2+3+...+10 を求める
-----*/
#include <stdio.h>
int main(void)
{
    int i, sum;

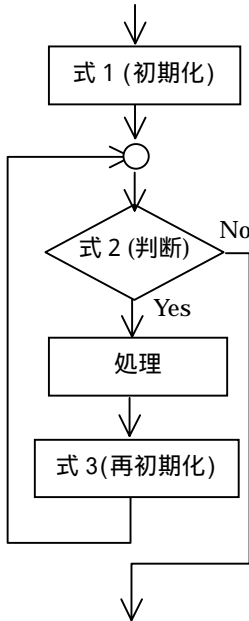
    sum = 0;
    i = 1;
    while( i <= 10 ) {
        sum = sum + i;
        i = i + 1;
    }
    printf("1+2+3+...+10 = %d\n", sum);
    return(0);
}
```

(注) 点線の部分は以下のように省略して記述されることが多い。

```
sum += i; /* sum の値を i だけ増やす */
i ++;    /* i の値を 1 つ増やす (インクリメント) */
```

3.5. for 文 (繰り返し処理 2) P.53

(1) 書式

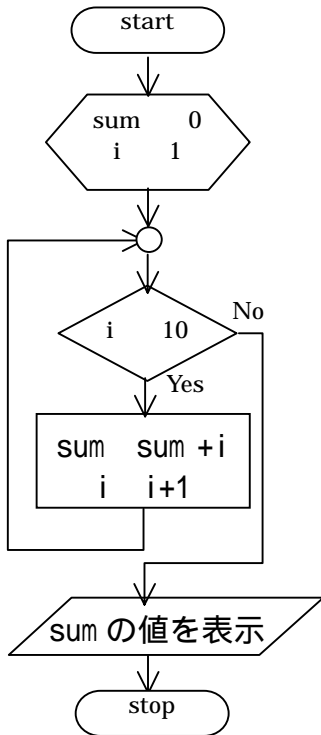


```

for (式 1; 式 2; 式 3) {
    処理
}
  
```

指定回数の繰り返しのとき、特に便利

例 1



```

/*-----
pro3_5_1.c  for 文
1+2+3+...+10 を求める
-----*/
#include <stdio.h>
int main(void)
{
    int i, sum;

    sum = 0;
    for(i = 1; i <= 10; i ++ ) {
        sum += i;
    }
    printf("1+2+3+...+10 = %d\n", sum);
    return(0);
}
  
```

(2) 練習

- (a) 3+4+...+20 を求める。
- (b) 1+3+5+...+19 を求める。
- (c) 1×2×3×...×10 を求める。
- (d) 5つの整数を入力し、その合計と平均を求める。
- (e) 整数nの値を入力後、さらにn個の実数を入力し、合計と平均を求める。
- (f) 整数nの値を入力後、さらにn個の実数を入力し、最大値を求める。
- (g) 整数nの値を入力後、さらにn個の実数を入力し、最小値を求める。
- (h) $= 0^\circ, 5^\circ, 10^\circ, \dots, 90^\circ$ について \sin の値を表示する。
- (i) for 文を用いて、5 4 3 2 と表示する。
- (j) 教科書 P.55 練習問題 8

3.6. データの終わりと EOF

データの数わからないとき、データの終了まで処理を続けたいことがある。データの終了は、ファイルまたはデータの終了を示すコード (MS-DOS などは「^Z」、CTRL+Z、UNIX では^D) が入力されるかどうかで判定すればよい。ファイルまたはデータの終了を示すコードが入力されると、scanf()関数などは EOF(End Of File、通常は-1)を返す。

例

```
-----
pro3_6_1.c      EOF
データが終了するまでデータ入力し、合計と平均を求める。
-----*/
#include <stdio.h>
int main(void)
{
    int      i;
    double   dat, sum;

    printf("合計と平均を求めます。必要なだけデータを入力して下さい。(終了:^Z\n");
    for(sum = 0, i = 0; scanf("%lf", &dat) != EOF; i ++ ) {
        sum += dat;
    }
    if(i != 0)
        printf("合計 = %f¥t 平均 = %f¥n", sum, sum / (double)i);
    else
        printf("データがありません。¥n");
    return(0);
}
```

練習

データが終了するまでデータ入力し、合計、平均、最小値、最大値を求める。

3.7. 多重ループ

(1) 九九の表を表示したい。for 文を入れ子にして、作成せよ。(P.62 参照)

練習

(a) 以下のような形の長方形を表示せよ。

```
*****
*****
*****
```

(b) 以下のような形の三角形を表示せよ。

```
*
**
***
****
*****
```

(c) 以下のような形の三角形を表示せよ。

```
*****
****
***
**
*
```

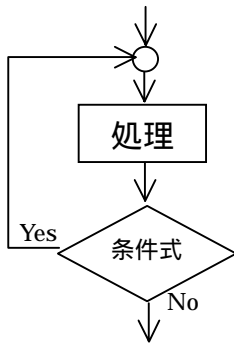
(2) break 文

```
/*-----
   pro3_7_2.c    break
   1,000 以下の素数を求める。
   -----*/
#include <stdio.h>
int main(void)
{
    int    i, x;

    for ( x = 2; x <= 1000; x ++ ) {
        for ( i = 2; i < x; i++ ) {
            if ( (x % i) == 0 ){ /* x が i で割り切れたら、*/
                break;          /* 内側のループから脱出 */
            }
        }
        if ( x == i ) {          /* x が素数なら ( 2 ~ x-1 の全ての数で
                                   割り切れなかったら ) */
            printf("%4d", x );
        }
    }
    return(0);
}
```

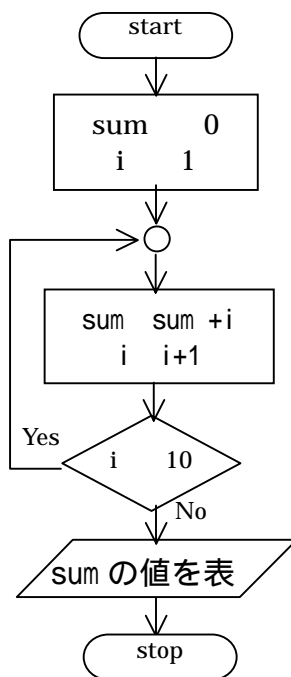
3.8. do while 文 (繰り返し処理 3)

(1) 書式



```
do {
    処理
} while ( 条件式 );
```

例 1



```
/*-----
   pro3_8_1.c   do while 文
   1+2+3+...+10 を求める
   -----*/
#include <stdio.h>
int main(void)
{
    int i, sum;

    sum = 0;
    i = 1;
    do {
        sum = sum + i;
        i = i + 1;
    } while( i <= 10 );
    printf("1+2+3+...+10 = %d\n", sum);
    return(0);
}
```

(2) 練習

キーボードから 0 以外の実数が入力されるまで、入力を繰り返すプログラムを作成せよ。0 以外の実数が入力されたら、その逆数を表示せよ。

【実行例】

```
0 以外の実数を入力して下さい。: 0.0
0 以外の実数を入力して下さい。: 0
0 以外の実数を入力して下さい。: 2.0
2.0 の逆数は、0.5 です。
```

3.9. switch 文

(1) プリント参照

(2) 練習

(a) 教科書 P.128 例題 30

(b) 配布資料のリスト 3.6 のプログラムを else if 文を用いて書き換えよ。

4. 関数

4.1. 関数とは

長いプログラム ... いくつかの機能単位（個々の仕事を分割した単位）の集まりで構成できる。

機能単位の例：入力、処理 1、処理 2、出力

サブルーチン ... 機能単位をメインルーチンとは別の部分に記述し、必要に応じて呼び出すようにしたもの。

C 言語の場合（2.2 節を参照）

メインルーチン ... main 関数

サブルーチン ... 関数（標準関数、ユーザー関数）

関数 ... サブルーチン + ある値を返す

例： $y = f(x) = ax^2 + bx + c$

入力： x

出力（戻り値）： y

関数を使うメリット

- 同一処理の重複を避けられる。
- 見やすくなる。（理解しやすくなる。）

4.2. プログラムの一般形式

#include <ヘッダーファイル名>

プロトタイプ宣言

```
int main(void)
```

```
{
```

宣言、定義

メインプログラム

```
return(0);
```

```
}
```

関数の型 関数名（仮引数）

```
{
```

宣言、定義

関数のプログラム

```
return 文
```

```
}
```

4.3. ユーザー関数例

(1) 標準的なスタイル

```
/*-----  
   ファイル名: pro4_3_1.c  
   機能: 関数を使用して足し算を行う  
-----*/  
#include <stdio.h>  
  
/* 関数プロトタイプ宣言 */  
/* 引数(入力)は2つで int 型、戻り値は int 型 */  
int add(int, int);  
  
/* main 関数 */  
int main(void)  
{  
    int out;  
    int a=10;  
    int b=20;  
  
    out=add(a,b);    /* a, b の値を関数 add に渡し、戻り値を out に代入する */  
    printf("加算結果 %d\n",out);  
  
    return 0;  
}  
  
/*-----  
   足し算をする関数(何をする関数か?)  
   入力(引数): 2つの整数 x, y          (入力は何か?)  
   出力(戻り値): x + y                (戻り値は何か?)  
   MS-DOS パソコン上では int 型で扱える  
   数の範囲は-32768 ~ 32767 まで      (その他注意事項)  
-----*/  
int add(int x, int y)    /* 呼び出し側の引数(a, b)が仮引数(x, y)に  
                           代入される(x = a; y = b;) */  
{                          /* a, b と x, y は別の変数なので注意すること。 */  
    int out;  
  
    out = x+y;  
    return out;  
}
```

(2) 簡略スタイル

関数定義が呼び出す関数 (main 関数) より前に記述されている場合、以下のように関数プロトタイプ宣言を省略できる。以降は、(1) の標準スタイルを使用する。

```
/*-----  
   ファイル名: pro4_3_2.c  
   機能: 関数を使用して足し算を行う  
-----*/  
#include <stdio.h>  
  
/*-----  
   足し算をする関数 (何をやる関数か?)  
   入力 (引数): 2つの整数 x, y          (入力は何か?)  
   出力 (戻り値): x + y                (戻り値は何か?)  
   MS-DOS パソコン上では int 型で扱える  
   数の範囲は -32768 ~ 32767 まで      (その他注意事項)  
-----*/  
int add(int x, int y) /* 呼び出し側の引数(a, b)が仮引数(x, y)に代入される (x = a; y = b;) */  
{ /* a, b と x, y は別の変数なので注意すること。 */  
    int out;  
  
    out = x+y;  
    return out;  
}  
  
/*----- main 関数 -----*/  
int main(void)  
{  
    int out;  
    int a=10;  
    int b=20;  
  
    out=add(a,b); /* a, b の値を関数 add に渡し、戻り値を out に代入する */  
    printf("加算結果 %d\n",out);  
  
    return 0;  
}
```

(3) 練習

- 実数を与えるとその絶対値を返す関数を作成し、その関数を用いてキーボードから入力した数の絶対値を表示するプログラムを作成せよ。
- キーボードから1つの整数 n を入力し、 $1+2+\dots+n$ を計算し、その結果を表示するプログラムを作成せよ。ただし、整数 n を与えると、 $1+2+\dots+n$ の結果を返す関数を作成し、使用すること。

4.4. 戻り値と引数による関数の分類

{ 値を返す
{ 値を返さない

{ 引数がある
{ 引数がない

(1) 値を返して引数がある場合

```
/* 関数プロトタイプ宣言 */
```

```
int add(int, int); /* 戻り値は int 型、引数 (入力) は 2 つで int 型 */
```

```
/* main 関数 */
```

```
    |  
    out = add(10, 20); /* 10, 20 を関数 add に渡し、戻り値を変数 out に代  
入する */  
    |
```

```
/* add 関数 */
```

```
int add(int x, int y) /* 呼び出し側の引数(10, 20)が仮引数(x, y)に代入される (x  
= 10; y = 20;) */
```

```
{  
    int z;  
    z = x+y;  
    return(z); /* z の値を関数の戻り値として返す */  
}
```

(2) 値を返さず、引数がある場合

```
/* 関数プロトタイプ宣言 */
```

```
void print_msg(int); /* 戻り値はなし、引数 (入力) は 1 つで int 型 */
```

```
/* main 関数 */
```

```
    |  
    int k = 5;  
    print_msg(k); /* k の値(5)を関数 print_msg に渡す */  
    |
```

```
/* print_msg 関数 */
```

```
void print_msg (int m) /* 呼び出し側の引数(k)の値(5)が仮引数(m)  
に代入される (m = k;) */
```

```
{  
    printf("順位 = %d¥n", m);  
}
```

(3) 値を返さず、引数もない場合

```
/* 関数プロトタイプ宣言 */
```

```
void prn_err(void); /* 戻り値なし、引数なし */
```

```
/* main 関数 */
```

```
    |
    prn_err ( );          /* 関数 prn_err を呼び出す */
    |
```

```
/* prn_err 関数 */
```

```
void prn_err (void)
```

```
{
    printf("エラーです\n");
}
```

(4) main 関数の書き方(1)

```
int main(void) /* 整数を返す、引数なし */
```

```
{
    |
    return(0); /* OS に 0 を返す */
}
```

(5) main 関数の書き方(2)

```
void main(void) /* 戻り値なし、引数なし */
```

```
{
    |
}
```

(6) 練習

次の仕様を満たす関数を作成せよ。また作成した関数を呼び出すテストプログラム (main 関数) も作成せよ。

関数名: printj

引数: 整数型で1つ

戻り値: なし

処理内容: 引数の値が1なら「優勝です。」と表示し、それ以外なら「位です。」と表示する。

4.5. 変数の分類

(1) 変数のスコープ (有効範囲)

- グローバル変数 (外部変数) ... プログラム全域で有効 (関数の外で宣言)
- ローカル変数 (局所変数) ... 関数内部 (正確には宣言したブロック{}で囲まれた部分) でのみ有効 (関数の内で宣言)

長いプログラムにおいて、ある変数の値を意図せずに変えてしまう危険性を避けるために、なるべくローカル変数を使用した方が良い。

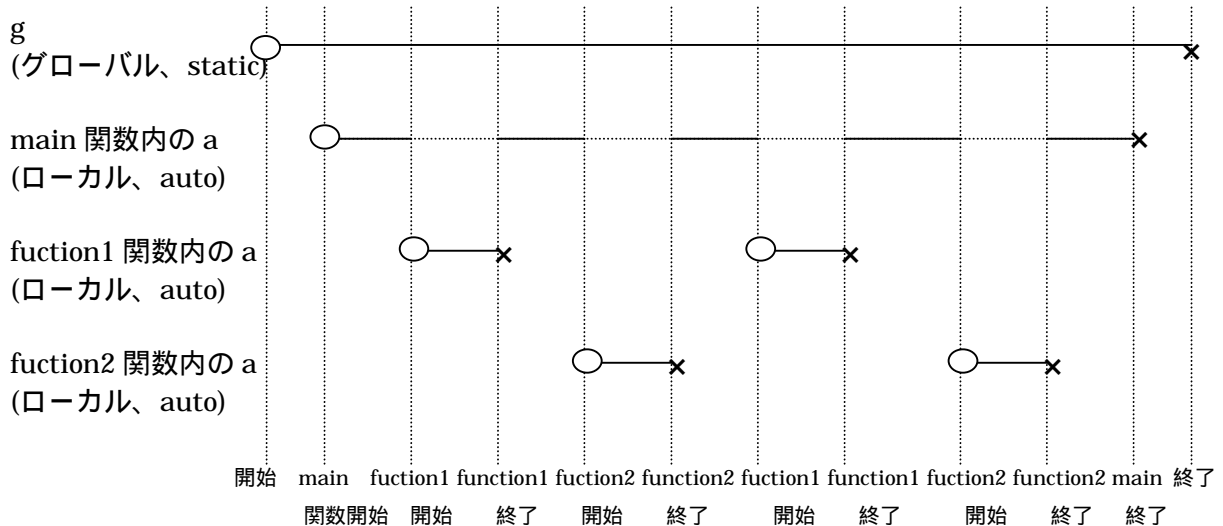
(2) 記憶クラス

- auto 変数 ... 変数が宣言された関数が呼ばれると生成され、関数が終了すると消滅する。(メモリのスタック領域に記憶)
- static 変数 ... プログラム開始時に生成され、プログラム終了時に消滅する。(静的なメモリ領域に記憶)

宣言場所	記憶クラス指定子	スコープ	記憶クラス
関数の外部	なし	プログラム全域 (グローバル)	static
	static		
関数の内部	なし	関数内 (ローカル)	auto
	static		static

[List5-15]の変数の持続時間 (寿命)

○...変数の生成 ×...変数の消滅



5. 配列

5.1. 1次元配列

(1) 配列とは? (教科書 P.68)

40個のデータを記憶する必要があるとき、

【今までの方法】

```
int a0, a1, a2, ..., a39;
a0 = 0;
a1 = 1;
a2 = 2;
  ⋮
a39 = 39;
  ⋮
printf("a0 = %d\n", a0);
printf("a1 = %d\n", a1);
  ⋮
printf("a39 = %d\n", a39);
```

【配列を使う方法】

```
int a[40];
int i;

for(i = 0; i < 40; i++)
    a[i] = i;

for(i = 0; i < 40; i++)
    printf("a[%d] = %d\n", i, a[i]);
```

【練習】

教科書 P.69 問5

(2) 配列データの初期化

【今までの方法】

```
int a0 = 56, a1=67, a2=63, ..., a39=55;
```

【配列を使う方法】

```
int a[40] = {56, 67, 63, ..., 55};
```

【例】

```
int a[40];
int a[40] = {56, 67, 63, ..., 55};
int a[] = {56, 67, 63, ..., 55};
int a[];   これはだめ
char b[3] = {'a', 'b', 'c'};
char b[] = {'a', 'b', 'c'};
double x[] = {1.0, 2.0, 3.0};
```

【練習】

- (a) 教科書 P.70 問 6
- (b) 教科書 P.71 例題 12
- (c) 教科書 P.71 練習問題 12
- (d) 教科書 P.72 例題 13
- (e) 教科書 P.73 練習問題 13

```
/*-----  
[ファイル名] array1.c  
[機能] 配列を使って N 人分の点数の度数分布 ( 1 1 段階 ) を表示する  
[方針] 点数は、dat[ ] にいったん格納しておく  
        点数の度数分布値は、hist[11]という配列に格納する  
[手順] 1 . データ ( 点数 ) の入力  
        ・ 人数(N)を入力  
        ・ 人数分のループ i=0 ~ N-1  
          データ入力      dat[i]      キー入力  
  
        2 . 度数 ( 各データが何点か ? ) をカウント  
        ・ hist[0] ~ hist[10]を 0 に初期化  
        ・ 人数分のループ ( i=0 ~ N-1 )  
          rank      dat[i]/10  
          hist[rank]を 1 増やす  
  
        3 . 度数を表示  
        ・ hist[0] ~ hist[10]を表示  
-----*/  
#include <stdio.h>  
  
int main(void)  
{  
    int i;  
    int N;                /* データの数          */  
    int rank;            /* 分類番号          */  
    double dat[100];     /* 点数を格納する配列*/  
    int hist[11];        /* 度数分布          */  
  
    /* 最初にデータの数を入力してもらう */  
    printf("人数は(100 人以内) = ");  
    scanf("%d",&N);  
  
    /* 画面にメッセージを出しつつ配列にデータ格納 */  
    for(i=0;i<N;i++){  
        printf("%3d 番目の点数を入力 = ",i);  
        scanf("%lf",&dat[i]);  
    }  
  
    /* 度数分布値を格納する配列の初期化 */  
    for(i=0;i<11;i++){  
        hist[i]=0;  
    }  
  
    /* dat[ ]の中のすべてのデータを調べ、度数分布値を hist[ ] に格納していく */  
    for(i=0;i<N;i++){  
        rank = (int)(dat[i]/10.0);  
        hist[rank]=hist[rank]+1;  
    }  
  
    /* 度数分布値を表示 */  
    printf("¥n");  
    for(i=0;i<11;i++){  
        printf("%3d:  %d¥n",i*10,hist[i]);  
    }  
  
    return 0;  
}
```

実行結果

```
人数は(100 人以内) = 6  
0 番目の点数を入力 = 95  
1 番目の点数を入力 = 56  
2 番目の点数を入力 = 78  
3 番目の点数を入力 = 85  
4 番目の点数を入力 = 100  
5 番目の点数を入力 = 45  
  
0: 0  
10: 0  
20: 0  
30: 0  
40: 1  
50: 1  
60: 0  
70: 1  
80: 1  
90: 1  
100: 1
```

```

/*-----
--ファイル名-- array1a.c

--機能-- 配列を使って N 人分の点数の度数分布 ( 1 1 段階 ) を表示する
         度数分布は、画面に * を表示することで行なう

--方針-- 点数は、dat[] にいったん格納しておく
         点数の度数分布値は、hist[] という配列に格納する
-----*/

#include <stdio.h>

int main(void)
{
    int i,j;
    int N;          /* データの数 */
    int rank;      /* ランクの数 */
    double dat[100]; /* 点数を格納する配列 */
    int hist[11];  /* 度数分布値を格納 */

    /* 最初にデータの数を入力してもらう */
    printf("人数は(100 人以内) = ");
    scanf("%d",&N);

    /* 画面にメッセージを出しつつ配列にデータ格納 */
    for(i=0;i<N;i++){
        printf("%3d 番目の点数を入力 = ",i);
        scanf("%lf",&dat[i]);
    }

    /* 度数分布値を格納する配列の初期化 */
    for(i=0;i<11;i++){
        hist[i]=0;
    }

    /*
       dat[] 中のすべてのデータを調べ、度数分布値を hist[] に格納
       していく
    */
    for(i=0;i<N;i++){
        rank = (int)(dat[i]/10.0);
        hist[rank]=hist[rank]+1;
    }

    /* 度数分布値を表示 */
    printf("\n");
    for(i=0;i<11;i++){
        printf("%3d:" ,i*10); /* 点数区分を出力 */
        for(j=0;j<hist[i];j++){ /* 度数の分だけ画面に * を出力 */
            printf("*");
        }
        printf("\n");
    }

    return 0;
}

```

実行結果

```

人数は(100 人以内) = 20
0 番目の点数を入力 = 69
1 番目の点数を入力 = 79
2 番目の点数を入力 = 77
3 番目の点数を入力 = 80
4 番目の点数を入力 = 79
5 番目の点数を入力 = 82
6 番目の点数を入力 = 77
7 番目の点数を入力 = 70
8 番目の点数を入力 = 78
9 番目の点数を入力 = 81
10 番目の点数を入力 = 68
11 番目の点数を入力 = 60
12 番目の点数を入力 = 80
13 番目の点数を入力 = 81
14 番目の点数を入力 = 83
15 番目の点数を入力 = 74
16 番目の点数を入力 = 49
17 番目の点数を入力 = 74
18 番目の点数を入力 = 81
19 番目の点数を入力 = 66

0:
10:
20:
30:
40:*
50:
60:****
70:*****
80:*****
90:
100:

```

5.2. 配列と関数を使って平均、最小値、最大値を求める例

```

/* -----
   ファイル名 : mean.c
   機能 : データが終了するまで実数を入力し、合計、平均、
         最小値、最大値を求める。
   方針 : 関数と配列利用の練習
   ----- */
#include <stdio.h>

#define MAX_DAT 100 /* 最大データ数 */

int    dat_no;      /* データ数 */
double dat[MAX_DAT]; /* データ配列 */

int    read_dat(void); /* キーボードからデータが終了する
                       まで実数を入力。戻り値はデータ数 */
double dsum(void);    /* 合計を求める */
double dmean(void);  /* 平均を求める */
double dmin(void);   /* 最小値を求める */
double dmax(void);   /* 最大値を求める */

int main(void)
{
    printf("合計、平均、最小値、最大値を求めます。¥n");
    printf("¥n¥n データ数 = %d¥n", read_dat()); /* データを入力 */
    printf("合計 = %f¥n", dsum());           /* 合計を求める */
    printf("平均 = %f¥n", dmean());          /* 平均を求める */
    printf("最小値 = %f¥n", dmin());         /* 最小値を求める */
    printf("最大値 = %f¥n", dmax());        /* 最大値を求める */
    return(0);
}

/* -----
   キーボードからデータが終了するまで実数をデータ配列に入力。
   戻り値 : データ数
   ----- */
int    read_dat(void){
    dat_no = 0;
    printf("必要なだけデータを入力して下さい。(終了: ^Z)¥n");
    while(scanf("%lf", &dat[dat_no]) != EOF){
        dat_no ++;
        if(dat_no >= MAX_DAT)
            break;
    }

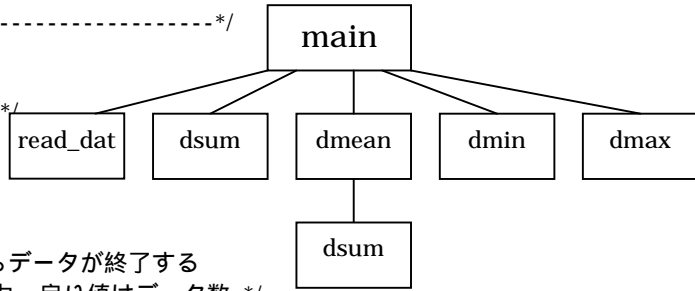
    return(dat_no);
}

/* -----
   データ配列中のデータの合計を求める。
   戻り値 : 合計
   ----- */
double dsum(void){
    int    k;
    double sum = 0.0;

    for(k = 0; k < dat_no; k ++){
        sum += dat[k];
    }

    return(sum);
}

```



```

/* -----
   データ配列中のデータの平均を求める。
   戻り値 : 平均
   ----- */
double dmean(void){
    if(dat_no <= 0){
        printf("データが入力されていません。¥n");
        return(0);
    }
    return(dsum() / (double)dat_no);
}

/* -----
   データ配列中のデータの最小値を求める。
   戻り値 : 最小値
   ----- */
double dmin(void){
    int    k;
    double small;

    if(dat_no <= 0){
        printf("データが入力されていません。¥n");
        return(0);
    }

    small = dat[0];
    for(k = 1; k < dat_no; k ++){
        if(small > dat[k])
            small = dat[k];
    }

    return(small);
}

/* -----
   データ配列中のデータの最大値を求める。
   戻り値 : 最大値
   ----- */
double dmax(void){
    int    k;
    double big;

    if(dat_no <= 0){
        printf("データが入力されていません。¥n");
        return(0);
    }

    big = dat[0];
    for(k = 1; k < dat_no; k ++){
        if(big < dat[k])
            big = dat[k];
    }

    return(big);
}

```

実行結果例

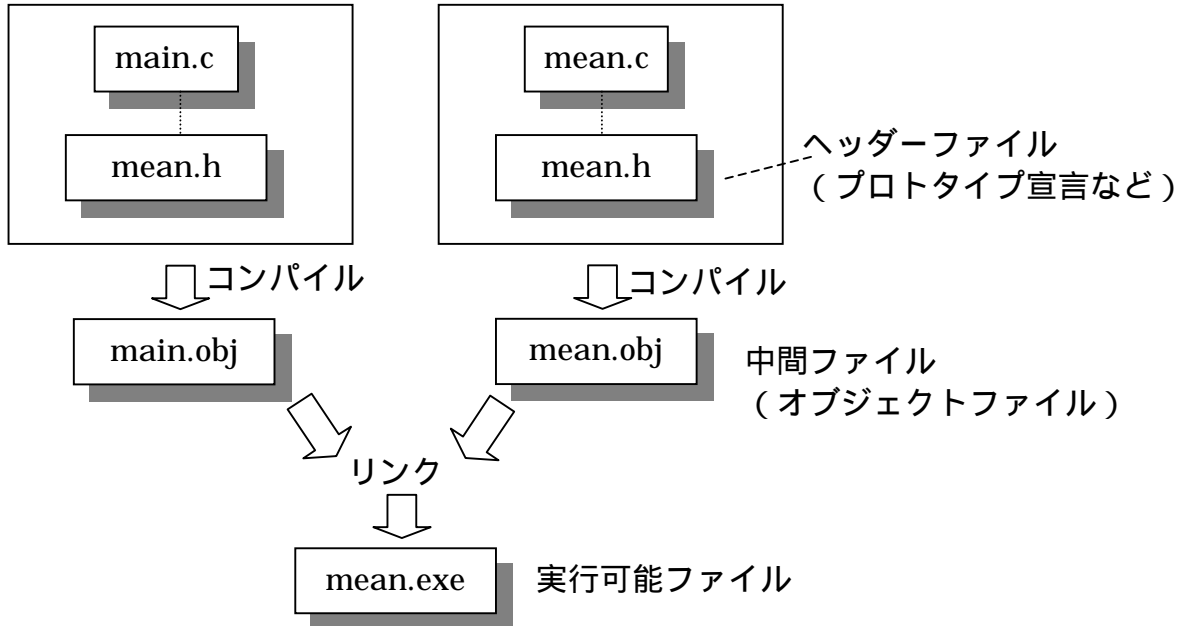
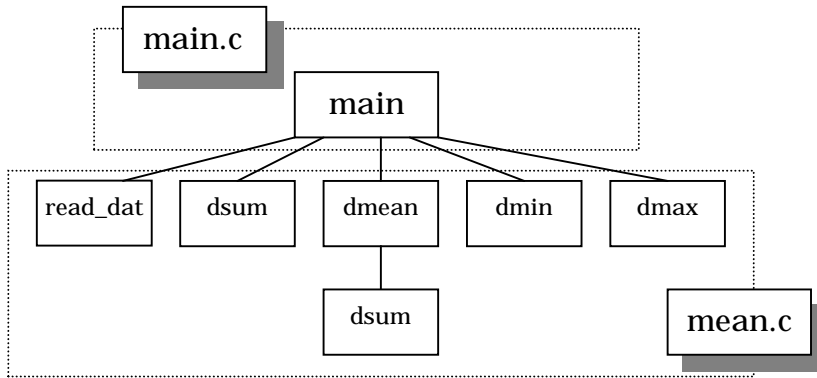
```

合計、平均、最小値、最大値を求めます。
必要なだけデータを入力して下さい。(終了: ^Z)
23.4
12.4
34.5
^Z

データ数 = 3
合計 = 70.300000
平均 = 23.433333
最小値 = 12.400000
最大値 = 34.500000

```

5.3. 複数ファイルによるプログラミング例



main.c

```

/* -----
   ファイル名 : main.c
   機能 : データが終了するまで実数を入力し、合計、平均、
         最小値、最大値を求める。
   方針 : 複数ファイルによるプログラミングの練習
   ----- */
#include <stdio.h> /* システムで指定されたディレクトリ(/usr/include など)にファイル(stdio.h)
                  があるとき <> を使用 */

/* 合計、平均、最小値、最大値を求める関数を使用する際に、関数のプロトタイプ宣言などが定義されている
   mean.h をインクルードする */
#include "mean.h" /* カレントディレクトリにファイル(mean.h)があるとき "" を使用 */

int main(void)
{
    printf("合計、平均、最小値、最大値を求めます。¥n");
    printf("¥n¥n データ数 = %d¥n", read_dat()); /* データを入力 */
    printf("合計 = %f¥n", dsum()); /* 合計を求める */
    printf("平均 = %f¥n", dmean()); /* 平均を求める */
    printf("最小値 = %f¥n", dmin()); /* 最小値を求める */
    printf("最大値 = %f¥n", dmax()); /* 最大値を求める */
    return(0);
}

```

mean.h

```
/*-----*/
ファイル名 : mean.h
機能 : データが終了するまで実数を入力し、合計、平均、
      最小値、最大値を求めるプログラム(mean.c)を使用するためのヘッダーファイル
方針 : 複数ファイルによるプログラミングの練習
-----*/
#ifndef MEAN_H /* MEAN_H が定義されていなければ(このファイルが以前に include されていなければ) */
#define MEAN_H 1 /* MEAN_H を定義 */

int read_dat(void); /* キーボードからデータが終了するまで実数を入力。戻り値はデータ数 */
double dsum(void); /* 合計を求める */
double dmean(void); /* 平均を求める */
double dmin(void); /* 最小値を求める */
double dmax(void); /* 最大値を求める */

#endif /* #ifndef MEAN_H に対応 */
```

mean.c

```
/*-----*/
ファイル名 : mean.c
機能 : データが終了するまで実数を入力し、合計、平均、
      最小値、最大値を求める。
方針 : 複数ファイルによるプログラミングの練習
-----*/
#include <stdio.h>
#include "mean.h"
#define MAX_DAT 100 /* 最大データ数 */
int dat_no; /* データ数 (このファイル内で有効) */
double dat[MAX_DAT]; /* データ配列 (このファイル内で有効) */

/*-----*/
/* キーボードからデータが終了するまで実数をデータ配列に入力。
   戻り値 : データ数
   ----- */
int read_dat(void){
    dat_no = 0;
    printf("必要なだけデータを入力して下さい。(終了: ^Z)\n");
    while(scanf("%lf", &dat[dat_no]) != EOF){
        dat_no ++;
        if(dat_no >= MAX_DAT)
            break;
    }
    return(dat_no);
}

/*-----*/
/* データ配列中のデータの合計を求める。
   戻り値 : 合計
   ----- */
double dsum(void){
    int k;
    double sum = 0.0;

    for(k = 0; k < dat_no; k ++){
        sum += dat[k];
    }
    return(sum);
}
```

mean.c (続き)

```
/* -----  
   データ配列中のデータの平均を求める。  
   戻り値 : 平均  
   ----- */  
double dmean(void){  
    return(dsum() / (double)dat_no);  
}  
  
/* -----  
   データ配列中のデータの最小値を求める。  
   戻り値 : 最小値  
   ----- */  
double dmin(void){  
    int    k;  
    double small;  
  
    if(dat_no <= 0){  
        printf("データが入力されていません。¥n");  
        return(0);  
    }  
    small = dat[0];  
    for(k = 1; k < dat_no; k ++){  
        if(small > dat[k])  
            small = dat[k];  
    }  
    return(small);  
}  
  
/* -----  
   データ配列中のデータの最大値を求める。  
   戻り値 : 最大値  
   ----- */  
double dmax(void){  
    int    k;  
    double big;  
  
    if(dat_no <= 0){  
        printf("データが入力されていません。¥n");  
        return(0);  
    }  
    big = dat[0];  
    for(k = 1; k < dat_no; k ++){  
        if(big < dat[k])  
            big = dat[k];  
    }  
    return(big);  
}
```

5.4. 練習問題

(1) 以下のプログラムの空欄を埋め、実行結果例のように動作するプログラムを完成せよ。

```
/*-----  
 配列を使って 点数の度数分布 (11段階) を表示する。  
 点数の入力は、EOF(^Z)が入力されるまで続ける。  
 度数分布は、画面に * を表示することで行なう。  
  
--方針--  
 点数は、dat[ ] といったん格納しておく  
 点数の度数分布値は、hist[ ]という配列に格納する  
-----*/  
#include <stdio.h>  
#define BUFF_MAX 100  
int main(void)  
{  
    int i,j;  
    int N; /* データの数 */  
    int rank; /* 点数のランク */  
    double dat[BUFF_MAX]; /* 点数を格納する配列 */  
    int hist[11]; /* 度数分布値を格納 */  
  
    /* 画面にメッセージを出しつつ配列にデータ格納 */  
    printf("点数を入力して下さい。(100個以内、終了はEOF(^Z)を入力)");  
    for(N = 0; N < BUFF_MAX; N++){  
        if(scanf("%lf",&dat[N]) == EOF)  
            break;  
        if(dat[i] < 0.)  
            dat[i] = 0.;  
        if(dat[i] > 100.)  
            dat[i] = 100.;  
    }  
    if(N >= BUFF_MAX)  
        printf("%d個を越えました。%d個までの度数分布を求めます。", N, N);  
  
    /* 度数分布値を格納する配列の初期化 */  
  
    /* dat[ ]の中のすべてのデータを調べ、度数分布値を hist[ ] に格納していく */  
    for(i = 0; i < N; i++){  
        rank =  
        hist[rank]=hist[rank]+1;  
    }  
  
    /* 度数分布値を表示 */  
    printf("%n");  
    for(i = 0; i < 11; i++){  
        printf("%3d:" ,i*10); /* 点数区分を出力 */  
        /* 度数の分だけ画面に * を出力 */  
  
        printf("%n");  
    }  
    return 0;  
}
```

実行結果例

```
点数を入力して下さい。(100個以内、終了はEOF(^Z)を入力)  
85 75 63 98 78 86 50 79 64 86 ^Z  
0:  
10:  
20:  
30:  
40:  
50:*  
60:**  
70:***  
80:****  
90:*  
100:
```

(2) 入力ファイル「in.dat」には、以下のような点数データが入っている。

85 75 63 98 78 86 50 79 64 86

(a) MS-DOS 上で、(1)のプログラム ex1.exe を使用し、入力ファイル「in.dat」の点数データの度数分布を求め、ディスプレイに表示するには どのようなコマンドを入力すればよいか示せ。

(b) MS-DOS 上で、問題 3 のプログラム ex3.exe を使用し、入力ファイル「in.dat」の点数データの度数分布を求め、出力ファイル「out.dat」に出力するには どのようなコマンドを入力すればよいか示せ。

(3) 以下の表に従って 2 次元配列を初期化し各教科の最高得点を求めるプログラムを作成せよ。

名前	国語	数学	英語
A	80	65	70
B	70	55	75
C	75	80	70
D	80	60	60
E	90	80	75

5.5. 2次元配列

(1) 2次元配列とは

1次元配列

```
int dat[8];
```

dat[0]	dat[1]	dat[2]	dat[3]	dat[4]	dat[5]	dat[6]	dat[7]
--------	--------	--------	--------	--------	--------	--------	--------

2次元配列 (表のようなデータ構造)

```
int ten[5][3];  
5行3列
```

	0列	1列	2列 (column)
0行	ten[0][0]	ten[0][1]	ten[0][2]
1行	ten[1][0]	ten[1][1]	ten[1][2]
2行	ten[2][0]	ten[2][1]	ten[2][2]
3行	ten[3][0]	ten[3][1]	ten[3][2]
4行 (row)	ten[4][0]	ten[4][1]	ten[4][2]

2行1列

(2) 2次元配列の初期化

【方法1】

```
int ten[5][3];  
ten[0][0] = 80;  
ten[0][0] = 80;  
ten[0][1] = 65;  
ten[0][2] = 70;  
ten[1][0] = 70;  
ten[1][1] = 55;  
ten[1][2] = 75;  
ten[2][0] = 75;
```

	0列	1列	2列 (column)
0行	80	65	70
1行	70	55	75
2行	75	80	70
3行	80	60	60
4行 (row)	90	80	75

【方法2】

```
int ten[5][3] = { {80,65,70},  
                  {70,55,75},  
                  {75,80,70},  
                  {80,60,60},  
                  {90,80,75} };
```

【例1】 3行4列の配列を宣言し、1行2列の要素に5を代入

```
int dat[3][4];  
dat[1][2] = 5;
```

【例2】 5行3列の配列を宣言し、1列の要素にすべて0を代入

```
int dat[5][3] = { { ,0, },  
                  { ,0, },  
                  { ,0, },  
                  { ,0, },  
                  { ,0, } };
```

または、

```
int dat[5][3], i;  
for(i = 0; i < 5; i++){  
    dat[i][1] = 0;  
}
```

【例 3】 5 行 3 列の配列を宣言し、すべての要素に 0 を代入

```
int dat[5][3], i, j;
for(i = 0; i < 5; i++){
    for(j = 0; j < 3; j++){
        dat[i][j] = 0;
    }
}
```

(3) 練習問題

(a) 教科書 p.75 例題 14

(b) 教科書 p.75 例題 14 で、各教科の合計を求めるように修正せよ。

(c) 下表を 2 次元配列 a に代入し、縦横の小計及び総合計を表示するプログラムを作成せよ。

```
1  3  2  4
3  4  7  2
5  6  8  9
2  4  7  7
9  8  4  6
```

(d) 以下の表に従って 2 次元配列を初期化し各教科の合計、平均、最低得点、最高得点、各生徒の合計、平均、最低得点、最高得点を求めるプログラムを作成せよ。

名前	国語	数学	英語
A	80	65	70
B	70	55	75
C	75	80	70
D	80	60	60
E	90	80	75

実行結果例：

	国語	数学	英語	合計	平均	最低	最高
A	80	65	70	215	71.7	65	80
B	70	55	75	200	66.7	55	75
C	75	80	70	225	75.0	70	80
D	80	60	60	200	66.7	60	80
E	90	80	75	245	81.7	75	90
合計	395	340	350	1085			
平均	79.0	68.0	70.0		72.3		
最低	70	55	60			55	
最高	90	80	75				90

(e) p.76 練習 14

(f) 以下の表に従って2次元配列 a, b を初期化し、配列の要素が1の部分に*、0の部分にスペースに変換して表示せよ。また、配列 a, b の対応する要素のいずれかが1の部分に*に、その他をスペースに変換して表示せよ。

2次元配列 a
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0

2次元配列 b
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
0 1 1 1 0
1 0 0 1 0
1 0 0 0 1
1 0 0 0 1

実行結果例：

配列 a

```
* * * * *
*       *
*       *
* * * * *
*
*
*
```

配列 b

```
*       *
*       *
* * *
*       *
*       *
*       *
```

配列 a と配列 b の OR

```
* * * * *
*       *
*       *
* * * * *
*       *
*       *
*       *
```

参考文献

- [1] 河西朝雄 「C言語」(ナツメ社)
- [2] 内田智史「C言語によるプログラミング[基礎編]」(オーム社)
- [3] 松林勝志, 小坂敏文, 前田恵三, 舘泉雄治, 柚賀正光, 北村敏也, 東 雄二
「C・C++入門」(森北出版)