PCA ゼミ No.2

★★

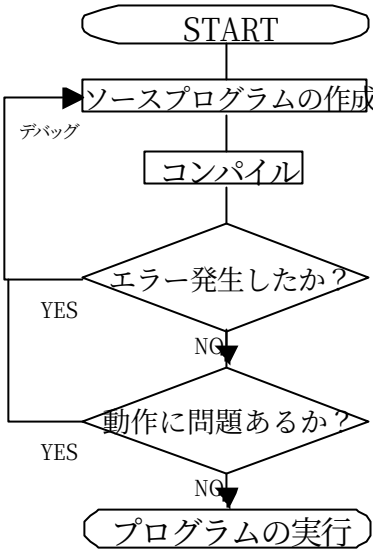
どうやってプログラムを作るか

★★

■ソースプログラムの作成

まず、エディタでソースプログラムを作成する。このとき、Cソースプログラムをファイルに保存するときは .c というタイプ名を付ける。

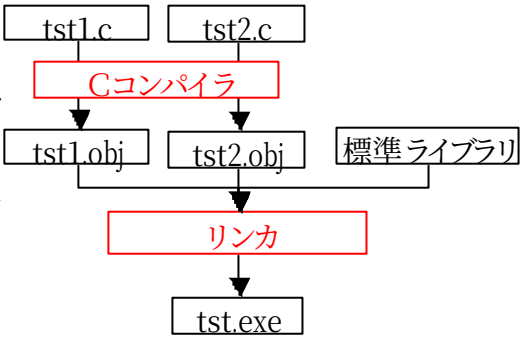
例） tstprg.c



■ソースプログラムをコンパイルする。

ソースプログラムを作成しただけでは、コンピュータは何もしません。コンピュータが理解できる唯一の言語”機械語”に変換する必要がある。その仕事をするのがCコンパイラと呼ばれるものである。

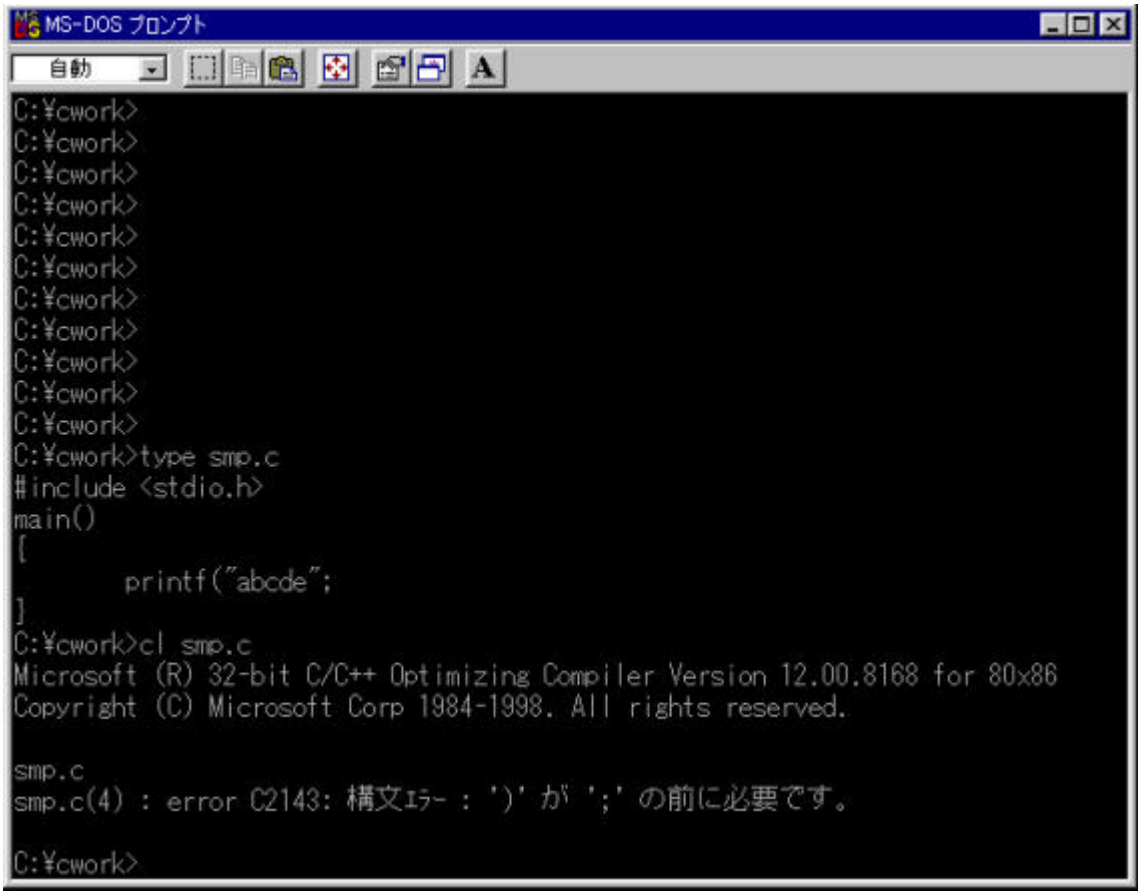
これでソースプログラムが機械語プログラムに変換される。この機械語プログラムを”オブジェクトモジュール”または”オブジェクトファイル”という。このオブジェクトモジュールは完全なプログラムにはなっていない。たとえば基本的な動作を受け持つ「標準ライブラリルーチン」というものが組み込まれていない。またプログラムコードをメモリ上のどの番地におくかなども未決定のままである。こういったものは別のモジュールとして存在しているので、それらをまとめて互いにつながりのある状態にする必要がある。これを”リンク”といい、この仕事をするのがリンカと呼ばれるものである。



うまくリンクできると .exe というプログラム実行ファイルが新たに作成される。

通常のコンパイラ作業では、コンパイルとリンクは連続して実行される。コンパイル処理は、開発環境によって異なるものであり、以下にいくつか例を示す。

開発環境	コンパイル指示
Turbo C	C>tc tstprg.c
Quick C	C>qc tstprg.c
Microsoft C	C>cl tstprg.c
UNIX	% cc tstprg.c -o tstprg.exe



■エラーがでたら

プログラムをコンパイルすると、正しくない部分を報告してくる。これが「エラー(Error)メッセージ」である。エラーメッセージは、”正しくない部分の報告”であるが、実際にはもう一つ「警告(Warning)メッセージ」というものもある。これは”謝りではないが望ましくない部分の報告」ということになる。

コンパイルが正しく行われても、それでプログラムが修了したわけではない。コンパイルしたプログラムを実行してみて、意図したとおりの動作をするかどうかのチェックをする必要がありる。この作業を”デバッグ”という。

■完成したプログラムの実行

無事完成して実行ファイルができると、そのプログラムを実行させることができます。実行するには、MS-DOS 上で tstprg という実行ファイルができているときは、

C>tstprg

と入力する。また、Windows 上でエクスプローラなどを利用して実行することもできる。

★★

８つの基本ステップ

★★

■変数宣言

　取り込んだデータを使っていろんな処理をするときに、一時的にデータを保管する必要がある。その保管場所を確保する。

■入力

　データの保管場所を決めたら、そこにデータを持ってくることができるようになる。用意した所定の場所に、数値や文字といったデータを入れる作業。

■演算

　入力されたデータを使って、計算する作業。

■条件判断

　-データをすべて加算する。ただし 1000 未満のものについては加算しない。
といった処理をするときに、条件を設定する作業。

■繰り返し

　同じ計算を何回もするときに行う作業。

■関数

　同じような処理が連続して、繰り返しでてくるような場合、それを「よく使われる処理」として登録し、名前を付けて保管しておくと便利である。この名前を付けられた「よく使われる処理」を関数という。この関数を設定・利用する作業。

■出力

　データを使って計算、処理した結果をディスプレイに表示したりプリントアウトしたりする作業。

■ファイル処理

　データの入力数が多いときなどキーボードから一つずつ打ち込むのは大変だし、入力ミスを起こしやすい。そこで、あらかじめデータファイルを作っておき、一気に入力させたる作業。また逆に、結果をデータとして保存したい場合、ファイルに出力すると良い。このファイルに出力するための作業。

★★

プログラミング

★★

■文字を出力する

次のプログラムを作ってみよう。smp1.c

```
#include <stdio.h>

main()
{
        printf("Howare you?¥n");
}
```

● main は関数

すべてのプログラムは関数で構成される。

プログラムには main 関数は 1 つだけ。

●関数の有効範囲 ｛ から ｝まで

●空行は無視される

●大文字と小文字は区別される

● printf で出力する

●¥n は復改する

```
printf("How");
printf("are");
printf("you?¥n");
```

```
printf("How¥n");
printf("are ¥n");
printf("you?¥n");
```

●#include で読み込む

最初のうちはとにかく #include <stdio.h>

●文にはセミコロン ’; ’ が必要

Ｃは行の概念を持たない

●字下げをすると読みやすい

●空白を取ってもかまわない

画面に「今日はよい天気です。」と表示するプログラム　smp2.c

```
/* sample program */
#include <stdio.h>

main()
{
        printf("今日はよい天気です。¥n");

        printf("今日は"); printf("よい天気"); printf("です。¥n");

}
```

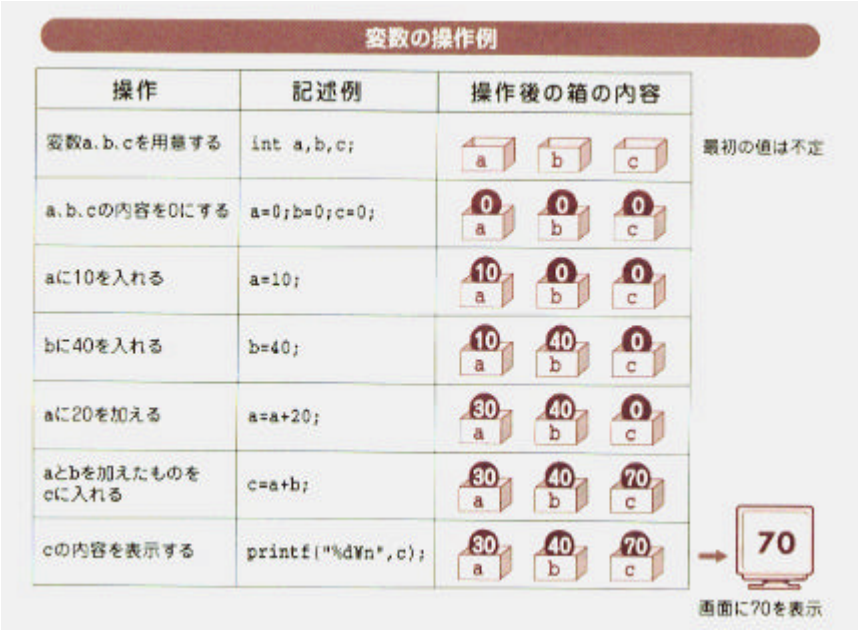
●漢字の出力
●１行に複数の文を書ける
●コメント /* ではじまり */ でおわる。

例題１　画面に次のように表示するプログラムを作る

```
a
ab
abc
abcd
```

■変数宣言

Ｃ言語では、すべての変数はそれを使用する前にあらかじめ名前と型名が宣言されていなければならない。



「１０＋２０」の答えを計算して表示するプログラム　smp3.c

```
#include <stdio.h>

main()
{
        int a,b,c;

        a=10;
        b=20;
        c=a+b;
        printf("%d¥n",c);
}
```

●整数型変数の宣言

int （integer:整数） １　２　１０　といった小数点以下の値のない数字

●値の代入

a=10; は a ← 10; といった雰囲気

a=a+100; は a ← a+100; といった雰囲気

●整数型変数の出力

%d は、対応する整数の値を表示しなさい　という指示。

たとえば、printf("%d¥n",a+b+100); でもよい。

●変換文字列

%d を変換文字列という。その他の変換文字列を以下に示す。

変換文字列	意味
%o	８進数で出力する
%d	１０進数で出力する
%x	１６進数で出力する
%f	浮動小数点数として出力する
%c	文字として出力する
%s	文字列として出力する

●メッセージを含む出力

例１： printf("a の値＝%d¥n",a);

例２： printf("a の値は %d です¥n",a);

例３： printf("a の値は %d　b の値は %d です。¥n",a,b);

半径から円の面積を計算するプログラム smp4.c

```
#include <stdio.h>

main()
{
        float pi,r,s;

        pi=3.14159;
        r=2.0;
        s=pi*r*r;
        printf("面積は %f です。¥n",s);
}
```

●実数型変数の宣言　float

●実数型変数の出力　%f

●変数の型

種類	型指定	宣言例	範囲
整数	int	int a;	-128 〜 127
倍長整数	long	longb;	-32768 〜 32767
単精度実数	float	float c;	-2147483648 〜 2147483647
倍精度実数	double	doubled;	有効桁 15 桁
文字	char	chare;	
文字列		char f[80];	

●配列を作る

int dt[10]; とすると、int 型の変数を 10 個用意してくれる。その変数には 0 からの通し番号が与えられる。すなわち、

dt[0],dt[1],dt[2],dt[3],dt[4],dt[5],dt[6],dt[7],dt[8],dt[9]

このような変数の集まりを ”配列 ” という。

例題２　int型変数a に5を代入し、aの値、aの２乗値、aの３乗値を表示する。

例題３　aの２乗値を変数aaに、aの３乗値を変数aaaに一度格納してから表示する。

例題４　float型変数bに2.44949を代入し、その２乗値を変数ｃに入れてから出力する。

■数字の計算をする.演算

１００と３の加減乗除の計算をするプログラム　smp5.c

```
#include <stdio.h>

main()
{
        int a,b,wa,sa,seki,syo,amari;

        a=100;
        b=3;
        wa=a+b;
        sa=a-b;
        seki=a*b;
        syo=a/b;
        amari=a%b;

        printf("和=%d¥n",wa);
        printf("差=%d¥n",sa);
        printf("積=%d¥n",seki);
        printf("商=%d¥n",syo);
        printf("余り= %d¥n",amari);
}
```

●演算子　+ - * / %

●インクリメント演算子とデクリメント演算子

プログラムの中で 1 だけ加算する、1 だけ減算するという処理は非常によく用いられるため、特別にコンパクトに記述できるようになっている。

++a; または a++;　→　a = a+1; と同じ

--a; または a--;　→　a = a-1; と同じ

０～９の自乗値を出力するプログラム　smp6.c

```
#include <stdio.h>

main()
{
        int i,a[10];

        for(i=0; i<=9; i++)
                a[i]=i*i;

        for(i=0; i<=9; i++)
                printf("%d¥n",a[i]);
}
```

インクリメント演算子は、他の式と組み合わせて用いられることもある。このときは++が変数より先に置かれるか、後に置かれるかで動作が異なってくる。

a=++b;

と書くと、これは、

b=b+1;　・・・・先に加算してから

a=b;　・・・・本来の処理をする

また、a=b++; とすると

a=b;　・・・・本来の処理をしてから

b=b+1;　・・・・後で加算する

なおこれはデクリメント演算子でも同様である。

例題５　3678秒を，時間，分，秒に変換する．

答え：1時間 1分 18秒

■条件判断と繰り返し

・定価が 1000 円以上なら 2 割引する．
そうでないときは 1 割引する．

・２つの数値を入力し，大きい方を判断する．

この二つのプログラムを作ります．

```
/* smp7.c */
#include<stdio.h>

main()
{
        int kane, a, b;

        printf("定価：");
        scanf("%d", &kane);

        if(kane>=1000){
                kane = kane*0.8;
        }
        else{
                kane = kane*0.9
        }

        printf("価格＝  %d ¥n",kane);

/*  ２つの数値を入力し，大きい方を判断する */

        printf("数値１＝");
        scanf("%d", &a);
        printf("数値２＝");
        scanf("%d", &b);
        if(a>b)
                printf("数値１の方が大きい¥n");
        else if(a<b)
                printf("数値２の方が大きい¥n");
        else
                printf("数値１と数値２は等しい¥n");

}
```

● scanf

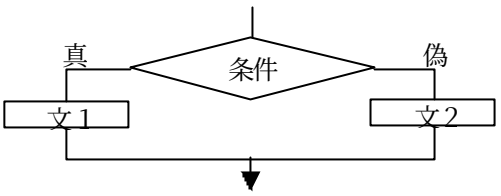
数値の入力には scanf **を使う．** scanf **は変換文字列の指定により，数値だけでなく，文字や文字列の入力もできる．** scanf **は** printf **とちょうど逆の働きをする関数である．**

この関数を使うとき，&を数値変数に付けるのを忘れないように注意する

● if **文で分岐する**

条件によって A **の処理をするか** B **の処理をするかを決める，働きをする．**

if(条件){
　条件がとれたときに実行する文１
}
else{
　条件がとれないとき実行する文２
}



●ブロック文

　　この if 文やあとで出てくる for 文で制御される文はいくつあってもよい．ただし，どこからどこまでが制御される範囲なのかを明確に示すためにブロック文を使う．ブロック文は{}で示す．

　　{　　---ここから
　　}　　---ここまで

●省略　ブロック文，else 文

　実行する文が１つだけの場合，特別にブロック文を省略できる．

例　　if(kane >= 1000)
　　kane=kane*0.8;
　　else
　　kane=kane*0.9;

　if 文の基本形は　**if-else** 型であるが，else が不要な場合はこれを省略することができる．たとえば”定価が 1000 円以上なら 2 割引する”だけならば

例　　if(kane >= 1000){
　　kane=kane*0.8;
　　}

●関係演算子

　if 文の条件は「条件式」で記述する．その条件式は「関係演算子」という演算子を使う．関係演算子は”２つの値の大小関係や等値関係を判定するもの”である．

演算子	説明	記述例	意味
>	大きい	if(a>100)	a が 100 より大なら
>=	大きいか等しい	if(a>=100)	a が 100 以上なら
<	小さい	if(a<100)	a が 100 より小なら
<=	小さいか等しい	if(a<=100)	a が 100 以下なら
= =	等しい	if(a==100)	a が 100 なら
! =	等しくない	if(a! =100)	a が 100 でないなら

●条件式の意味

　　ここで「条件が真」というのは，条件部に書かれた式が，その条件を満たしていることを示す．

例：　　if(a==10)

条件式：　a==10
真：　　a の値が 10 のとき ”真 ” になる
偽：　　a の値が 10 以外のとき ”偽 ” になる

● else　if 文

if 文は本来２方向に分岐するもだが，else　if　を使うことによって多様高にも分岐できる．

・西暦から ”うるう年 ” かどうか判定するプログラム

```
/* smp8.c */
#include<stdio.h>

main()
{
        int n;

        printf("年＝") ;
        scanf("%d", &n);

        if(n%4 = = 0 && n%100!=0){
                printf("うるう年です¥n");
        }
        else if (n%400 = = 0){
                printf("うるう年です¥n");
        }
        else{
                printf("ちがいます¥n");
        }

}
```

●論理演算子

”かつ ”, ” または ” に相当するものを論理演算子という．

うるう年は,

” ４で割り切れ　**かつ**　100 で割り切れない年 ” または ” 400 で割り切れる年 ” に当てはまる年である．この ” かつ ” に相当するのが && である．

また，”a の値が 10 未満または 100 よりだいなら ” という条件は,

if(a<10||a≧100)

と記述する．この || が ” または ” に相当する論理演算子である．

他にも ” ではない ” という否定を表す演算子として ！ がある．

演算子	説明	記述例	意味
&&	論理積（かつ）	if(a==3&&b==4)	a が 3 かつ b が 4 なら
||	論理和（または）	if(a==3||b==4)	a が 3 または b が 4 なら
!	否定（でない）	if(!(a==3&&b==4))	a が 3 かつ b が 4 でないなら

例題６　入力された数字が偶数か奇数かを判別する．

for を使ったプログラム

```
/* smp9.c */
#include<stdio.h>

main()
{
        int i, sum, mul;

        sum=0;
        mul=0;
        for(i=1; i<=5;i++){
                sum=sum+i;
                mul=mul*1;
        }
        printf("和＝%d¥n",sum);
        printf("積＝%d¥n",mul);

        for(i=40; i>=20;i=i-2){
                printf("%d",i);
        }

}
```

● for 文

これが最も基本的な for 文の用例である.

"変数 i が 1 から 5 になるまで{}を繰り返せ”という意味で，詳しくは

```
for(i=1; i<=5; i++){
}
```

for(i を 1 にしろ(初期化);i が 5 以下の間ループしろ(継続条件);ループのたびに i に 1 を足せ(最初期化))

ということになる.

for の決まり文句は,

```
for (i=1; i<=回数; i++){
        実行したい文の集まり
}
```

・０～１９までの２乗値と３乗値を計算するプログラム

```
/* smp10.c */
#include<stdio.h>

main()
{
        int i;
        int sqr[20], cube[20];

        for(i=0; i<=19;i++){
                sqr[i]=i*i;
                cube[i]=i*i*i;
        }

        printf("数　２乗　３乗¥n");

        for(i=0; i<=19;i++){
                printf("%2d　%4d　%4d¥n",i,sqr[i],cube[i]);
        }

}
```

●配列の宣言

●配列へのデータ設定

配列にデータを設定するときは添え字をうまく活用する．つまり

sqr[i]　と表現し，変数 i の値を 0 ～ 19 と変更してやれば，sqr[0]～ sqr[19]を指定したことになる．

●数字の表示幅指定

数字を表示させるとき，桁数をそろえたいことがある．たとえば

1		1
23	ではなく	23
337		337
5262		5262

としたいとき，**%4d** ”数字を 4 文字幅で表示しろ”といったふうに記述します．

例題７　九九の表を表示する．

ここでは10から20までのかけ算の結果を表示する．

できれば多重ループを使って欲しい．

出力例：

```
      10  11  12  13  14  15  16  17  18  19  20
10   100 110 120 130 140 150 160 170 180 190 200
11   110 121 132 143 154 165 176 187 198 209 220
12
・                          省略
20   200 220 240 260 280 300 320 340 360 380 400
```

・入力された数字の合計を計算するプログラム

```
/* smp11.c */
#include<stdio.h>

main()
{
        int dt=-1, sum=0;

        while(dt != 0){
                printf("?");
                scanf("%d", &dt);
                sum=sum+dt;
        }

        printf("合計＝%d¥n",sum)；

}
```

● while 文

for 文は”繰り返す回数がはじめからわかっている”ときのみ使える．しかし，”繰り返す回数が最初はわからない”ときは必ずある．そういったときに用いるのが **while** である．while 文は，ループのたびに条件を見て，それが正しければ{}の処理を続行する．条件が正しくなければ処理をうち切る．

ここでは，dt ! = 0 なので”dt が 0 でない間ループしろ”という意味になっている．よってこのループ条件に使われる dt という変数は，毎回の{}内処理ごとに値が変わるように記述しなければならない．さもないと永久にループし続けることになる．

while ループの決まり文句は以下の通りである．

```
while(繰り返し条件){
    実行したい処理の集まり
    繰り返し条件要素の再設定
}
```

● while の前処理

ここの while ループの条件が，”dt の値が 0 でないこと”であるので，一番最初の dt の値を-1 に仮に設定してある．最初に dt を 0 としてしまうと，while ループを一度も処理しないプログラムになってしまうので注意する必要がある．

例題８　10000を越えない最大の倍数を見つける．

数字を入力する．その数字を2倍，3倍...と計算していき，10000を越えたらwhileループを打ち切る．

・switch 文で分岐する

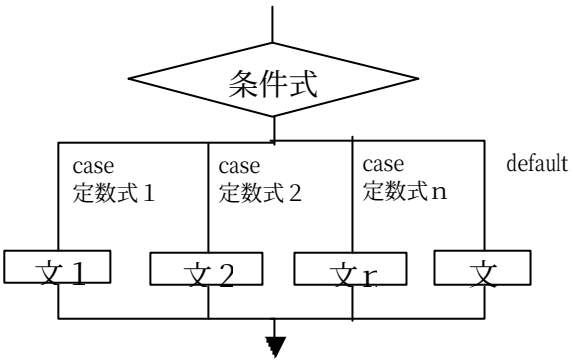
```
/* smp12.c */
#include<stdio.h>

main()
{
      int a;

      for(a=1; a<=5; a++){
            printf("----------%d\n ",a);
            switch(a){
            case 1:
                  printf("a=1\n");
                  break;
            case 3:
                  printf("a=3\n");
                  break;
            case 5:
                  printf("a=5\n");
                  break;
            default:
                  printf("others\n");
            }
      }
}
```

● switch 文

多方向分岐をする.

書式：

```
switch(式){
  case 定数式１:
                文１
                break;
  case 定数式２:
                文２
                break;
  case 定数式ｎ:
                文ｎ
                break;
  default:
                文
  }
```



・式の値を評価し，その値と一致する定数式の case 部分にジャンプし，該当する文を実行する.

・break 文に出会うと処理を終了して switch 文全体を終了する.

・一致する定数式がないときは default 部に記述された文を実行する.

・default 部は不要なら省略できる.

・switch(式)の式は整数を結果とするものでなければならない．また case 定数式部分も整数，文字定数，定数式しか書けない．

正しい例：

case5:　　case 'A':　　case 'B'+20 :　　case MAXVAL:

誤った例：

case5.6:　　case a>10:　　case "ABC":

・case 句の処理文の最後には break 文を入れる必要がある．

例えば，先のプログラムで break 文を削除してみると・・・

● break 文で打ち切る

break 文は switch 文または，for,while などのループから脱出するための文である．

break 文を使うと goto 文なしでループを脱出できる．

ループが多重構造になっているときは，break 文の存在するループを１つだけ抜ける．

・break 文の働きを確認するプログラム

```
/* smp13.c */
#include<stdio.h>

main()
{
      int i, n, sum;

      printf("for loop\n");
      sum=0;
      for(i=1; i<=10; i++){
            scanf("%d",&n);
            if(n = = -1)
                  break;
            sum=sum+n;
            }
      printf("sum=%d\n",sum);

      printf("while loop\n");
      i=10;
      while(1){
            printf("%d\n",i);
            if(i= =13)
                  break;
            ++i;
      }
      printf("end\n");
}
```

10 個の入力データ，または-1 が入力されるまでのデータの合計をとる．

i の値が 13 になったらループを終了する．

■入力と出力

●数値の入出力(scanf,printf)

書き方：

int n;	変数を宣言する
printf("%d¥n",n);	printf のときは n を使う
scanf("%d",&n);	scanf のときは&n を使う

printf のオプション指定

・フィールド幅指定子

数値の出力幅を指定する．(p13 参照)

例：変数の値が 123 のとき

%d	123	必要な幅で出力される
%5d	－－123	先行スペースを入れて５文字で出力
%2d	123	指定の方が小さくても必要幅を確保

・精度指定子

実数の小数点以下の桁数を指定する．

例：変数の値が 654.321 のとき

%f	654.321000	標準の幅で出力
%12f	－－654.321000	小数点を入れて 12 桁で出力 小数点以下は標準の桁数となる
%9.2f	－－－654.32	小数点を入れて 9 桁で出力 小数点以下 2 桁で出力

・printf のオプション指定子の機能を確認するプログラム

```
/* smp14.c */
#include<stdio.h>

main()
{
      int a;
      double b;

      printf("フィールド指定子の確認．4桁の整数を入力して下さい：");
      scanf("%d",&a);

      printf("d    :%d:\n",a);
      printf("2d   :%2d:\n",a);
      printf("10d  :%10d:\n",a);

      printf("精度指定子の確認．小数点以下3桁の実数を入力して下さい：");
      scanf("%lf",&b);

      printf("f    :%f:\n",b);
      printf("12f  :%12f:\n",b);
      printf("9.2f  :%9.2f:\n",b);
      printf("9.f   :%9.f:\n",b);
      printf(".2f   :%.2f:\n",b);

}
```

・1 文字入出力プログラム

```
/* smp15.c */
#include<stdio.h>

main()
{
      int c;

      c=getchar();
      putchar(c);


}
```

● getchar

1 文字入力を行うための関数．その文字を入れる変数に char 型を使うとうまく動作しないことがあるので，原則として

getchar 入力は必ず int 型変数を使う．

● putchar

1 文字を画面に出力する関数．変数の代わりに直接文字を指定することもできる．

例： putchar('c'); putchar('3'); putchar(65);・・・A と表示

●文字コード

コンピュータは文字を，文字コード（キャラクタコード）と呼ぶ連続番号で管理している．基本となる ASCII（アスキー）文字コード表は 0 から 127 までの番号からなる．例えば，

ch='A'; とすると変数 ch には英文字'A'に相当する文字コード 65 が代入される．

日本においては，このコード表を拡張した JIS 文字コード表が使われている．これは ASCII 文字コード表の一部を手直しし，カタカナコードなどを追加したものである．ASCII 文字コードのうち，0 から 31 までの特殊コードを除外した英数字・記号は次のコードになっている．

32	空白	53	5	74	J	95	_	116	t
33	!	54	6	75	K	96	`	117	u
34	"	55	7	76	L	97	a	118	v
35	#	56	8	77	M	98	b	119	w
36	$	57	9	78	N	99	c	120	x
37	%	58	:	79	O	100	d	121	y
38	&	59	;	80	P	101	e	122	z
39	'	60	<	81	Q	102	f	123	{
40	(	61	=	82	R	103	g	124	|
41	)	62	>	83	S	104	h	125	}
42	*	63	?	84	T	105	i	126	~
43	+	64	@	85	U	106	j	127	DEL
44	,	65	A	86	V	107	k		
45	-	66	B	87	W	108	l		
46	.	67	C	88	X	109	m		
47	/	68	D	89	Y	110	n		
48	0	69	E	90	Z	111	o		
49	1	70	F	91	[	112	p		
50	2	71	G	92	\	113	q		
51	3	72	H	93	]	114	r		
52	4	73	I	94	^	115	s		

・１行入出力プログラム

```
/* smp16.c */
#include<stdio.h>

main()
{
        char ss[80];

        gets(ss);
        puts(ss);


}
```

●文字列格納用変数

char s s[80]; は文字列を格納するための変数である．この文字列変数には最大 80 文字まで格納できる．それより長い文字列が入力されると，オーバーした文がメモリ内部に無理矢理格納され，メモリの一部を破壊してしまう．したがって最大何文字の文字入力があり得るかを考慮して，十分なサイズの文字列変数を用意する必要がある．

●文字列終端マーク

文字列を変数に格納する場合，Cではどこまでが文字列かを判断するために，文字列終端マーク **0** を用意している．例えば，"ABC"という３文字からなる文字列を入力すると，文字列変数 s に格納され，

s[0]=65 s[1]=66 s[2]=67 s[3]=0 s[4]...

となり，最後に必ず **0** が格納される．だから char ss[80]; と宣言したとき，実際に入力できる文字列の長さは 79 文字となる．

● gets

キーボードから文字列を入力するための関数．

● puts

文字列変数に格納されている変数を出力する関数．

puts は printf とは違って，出力した後，自動的に復改する．

● EOF をチェックする．

EOF とは[End Of File]のことで，ファイルが終わりになったときに返される値を意味する記号定数である．getchar 関数，scanf 関数で入力の終了を知らせるときにはこの EOF を用いる．この EOF をキーボードから入力するときには，特別に **CTRL + Z** と入力する．

・キーボードから入力された文字が小文字だったら大文字に変換するプログラム

```
/* smp17.c */
#include<stdio.h>
#include<ctype.h>     /* toupper */

main()
{
      int c;

      while((c=getchar()) != EOF){
            c = toupper(c);
            putchar(c);
      }
}
```

入力例　FORTRAN andPascal　　Prolog or LISP　CTRL+Z

● toupper 関数と ctype.h ファイルの include

●while((c=getchar()) != EOF) 定石記法

・まず while 文の実行に先立ちc=getchar()を実行する

・次に while を実行する．すなわち今入力した c の値が EOF でなければループする．

この記述は，連続１文字入力の基本パターンである．

例題９　EOFが入力されるまで，入力された数字を加算，または減算するプログラム．

ヒント：実数型変数，while定石記法の応用，scanf

■関数

Ｃプログラムは関数の寄せ集めで構成される．main 関数ですら構成上は単なる関数でしかない．ここでは関数について学ぶ．

一般にプログラミング言語では，メインプログラムの他に

１．サブルーチン

２．ファンクション（関数）

といったプログラム単位を用いる．ここでサブルーチンとは”呼び出すことで何か仕事をしてくれるプログラム”である．一方ファンクションは”それを呼び出すと何かの計算をしてくれて，その値を返してくれるプログラム”である．これらの違いは値を返してくれるかどうかでだけである．そこでＣでは，両者をまとめて単に関数(function)とした．だからＣでは値を返す関数と，値を返さない関数がある．またＣではメインルーチンに相当するものを，main という名前の関数としている．

・ベキ（累乗）計算を行う関数 beki をりようしてベキ計算をするプログラム

```
/* smp18.c */
#include<stdio.h>
int beki(int a, int b);

main()
{
      int n;

      n=beki(2, 3);
      printf("%d\n",n);

      printf("%d\n",beki(3, 4));
}

int beki(int a, int b)
{
      int i, ans;

      ans=1;
      for(i=1; i<=b; i++){
            ans=ans*a;
      }
      return ans;
}
```

●関数の基本構成

・関数の名前

・関数に渡すデータの型と名前

・関数から返されるデータの型

・関数本体の処理

・結果を受け渡すための処理

これらのことを記述して初めて関数となる．その基本形は

return 型　関数名（引数型と仮引数名,・・・）

{

関数内変数の宣言

処理文

}

return 型は，”個の関数は○○という型のデータを返すので適切に処理するように”ということを伝える役割を持つ．

関数名は，他の関数や変数とぶつからないように付ける．

引数とは，呼び出し先から受け渡される数のことで，() の中で型と名前を記述する 必要でなければ()だけ，もしくは void と記述する．

関数内変数は，その関数の中だけで利用する変数である．これをとくにローカル変数という．

● return

処理文で計算・実行された値を，呼び出し先に返す（戻す）ときには return 文を用いる．

呼び出し先に戻す値がない場合には，void 型関数を用いる．

例：

```
voidmyfunc(int dt);
{
        ・・・
        if(dt<=0) return;   /*dt が 0 以下なら処理を打ち切る   */
        ・・・
}
```

また return 文は必要なら，１つの関数の中にいくつ書いてももかまわない

●プロトタイプ宣言

関数を作ったときはその仕様（使い方）を main 関数より前に書いておく必要がある．これをプロトタイプ宣言という．このプロトタイプ処理は，プログラム本体部分の処理に先立って，次のことをコンパイラに伝える役割を持つ．

```
int beki(int a, int b);
```

・このプログラムでは beki という関数を使う

・その関数 beki は，int 型の値を返すので適切に処理して欲しい

・関数 beki の最初の引数は int 型，次の引数は int 型である．これに適合しない引数を使ったら警告して欲しい

● void 型の関数

Ｃの関数には値を返さないものがあるので，”この関数は値を返さない”ということを表明しなければならない．そこで　**void** を用いる．

```
#include<stdio.h>
void putd(int a);

main()
{
      putd(123);

}

void putd(int a)
{
      printf("%d¥n",a);
}
```

この関数 putd は，ただ渡された値を出力するだけなので，return 文で戻すべき値がない．そこで void を用いている．

さらに void は関数定義のときに，”引数がない”という意味でも用いる．

例題10　円の面積を計算する関数を作って，入力された半径から面積を計算しなさい．

●標準ライブラリ関数を使う

C 言語には，入出力や文字列処理を行う機能はなく，これらはすべて関数で実行される．”コンパイラする本体部分はなるべく小さくして，必要なら外部にある機能を取り込む”というのが C 言語の思想になっている．多くの人が必要とする機能は”標準ライブラリ関数”としてあらかじめ用意されているので，必要に応じてこれらを利用する．詳しくは，マニュアル等を見ることにして，ここでは簡単にどのファイルにどういう関数があるのかを紹介しておく．

◇ stdio.h 入出力系

clearerr	指定したファイルのエラーフラグと EOF フラグを 0 にする
fclose	ファイルのクローズ
feof	ファイルエンドの検出
ferror	ファイルエラーの検出
fflush	バッファに入っているすべての文字のファイル書きだし
fgetpos	ファイルポインタの位置情報を読み込む
fgetc	ファイルから１文字入力
fgets	ファイルからの文字列入力
fopen	ファイルのオープン
fprintf	ファイルへの書式付き出力
fputc	ファイルへの１文字出力
fputs	ファイルへの文字列出力
fread	ファイルからのブロックリード
freopen	ファイルの再オープン
fscanf	ファイルからの書式付き入力
fseek	ファイルポインタを移動
fsetpos	fgetpos で読み込んだ位置情報でファイルポインタを設定
ftell	現在のファイルポインタの位置を返す
fwrite	ファイルへのブロックライト
getc	ファイルから１文字入力
getchar	１文字入力
gets	文字列入力
perror	stderr へエラーメッセージを出力
printf	書式付き出力
putc	ファイルへの１文字出力
putchar	１文字出力
puts	文字列出力
remove	ファイルを削除
rename	ファイル名を変更
rewind	ファイルポインタを先頭へ移動
scanf	書式付き入力
setbuf	バッファリングの制御
setvbuf	バッファリングの制御
sprintf	書式付きデータを文字列変数に書き込む
sscanf	書式付きデータを文字列変数から読み込む
tmpfile	テンポラリファイルを作成
tmpnam	衝突しないファイル名を作成
ungetc	ファイルへの１文字戻し
vfprintf	ポインタで引数並びを指示する fprintf
vprintf	ポインタで引数並びを指示する printf
vsprintf	ポインタで引数並びを指示する sprintf

◇ stdlib.h 一般ユーティリティ系

abort	プロセスを強制中断する
abs	int 型データの絶対値
atexit	プログラム終了時点での関数実行
atof	文字列を double 型に変換
atoi	文字列を int 型に変換
atol	文字列を long 型に変換
bsearch	バイナリサーチ
calloc	配列のためのメモリブロック確保
div	商と余りの計算
exit	処理終了
free	メモリブロックの解放
getenv	環境変数の値の取得
labs	long 型データの絶対値
ldiv	long 型データの商と余りを計算
malloc	メモリブロック確保
rand	整数の疑似乱数を返す
realloc	メモリブロック再確保
srand	疑似乱数の発生系列を変更する
strtod	文字列を double 型に変換
strtol	文字列を long 型に変換
strtoul	文字列を unsigned long 型に変換
qsort	クィックソート
system	OS のコマンドを実行

◇ string.h 文字列処理系

memchr	指定した文字列から指定した文字を検索する
memcmp	範囲を指定できる文字列比較
memicmp	大文字と小文字を区別しない memcmp
memmove	コピー領域が重なっていても正しくコピーする memcpy
memcpy	指定した数の文字をあるバッファからあるバッファにコピー
memset	指定した文字列の指定バイト数を指定した文字で初期化
strcat	文字列の連結
strchr	文字列を先頭からサーチし指定文字を探す
strcmp	文字列の比較
strcpy	文字列の複写
strcspn	指定した複数の文字を検索
strerror	エラー番号に対応するエラーメッセージを返す
strlen	文字列の長さを得る
strncat	文字数を指定できる文字列連結
strncmp	文字数を指定できる文字列比較
strncpy	文字数を指定できる文字列複写
strpbrk	指定文字列に含まれるどれかの文字が現れる位置を探す
strrchr	文字列を後ろからサーチし指定文字を探す
strspn	[指定した複数の文字]以外の文字を検索
strstr	文字列の中から指定文字列を検索
strtok	文字列からトークンを切り出す

◇ time.h 時間処理系

asctime	時間情報を文字列に変換
clock	実行開始からの経過秒数を返す
ctime	システムクロック時間を文字列に変換

difftime	time1 と time2 との差を秒単位で計算
gmtime	グリニッジ標準時間
localtime	long 値の秒情報を時刻格納構造体に変換
strftime	日付と時間情報を書式化する
time	現在の時刻を 1970 年 1 月 1 日 00:00:00 からの経過秒で返す

◇ ctype.h 文字処理系

isalnum	英数字なら真
isalpha	英文字なら真
iscntrl	制御文字なら真
isdigit	数字なら真
isgraph	空白を除く印字可能文字なら真
islower	小文字なら真
isprint	印字可能文字なら真
ispunct	区切り文字なら真
isspace	空白，タブ，復帰，改行，垂直タブ，改頁なら真
isupper	大文字なら真
isxdijit	16 進数表示文字なら真
tolower	大文字を小文字に変換する
toupper	小文字を大文字に変換する

◇ setjmp.h ジャンプ処理系

lonjmp	setjump と組み合わせてグローバルジャンプを行う
setjmp	longjump 実行に備えて状態情報を保存する

◇ signal.h シグナル処理系

raise	シグナルを発生させる
signal	OS からの割り込みシグナルに対応する処理を指定する

◇ assert.h 診断処理系

assert	プログラムに診断機能を付加する

◇ math.h 数学処理系

acos	アークコサイン
asin	アークサイン
atan	アークタンジェント
atan2	アークタンジェント（範囲-p ～ p）
ceil	最小整数
cos	コサイン
cosh	ハイパボリックコサイン
exp	指数
fabs	絶対値
floor	最大整数
fmod	浮動小数点の剰余
frexp	浮動小数点の計数部と 2 のベキ乗部に分割
ldexp	x×2^n の値を計算
log	自然対数
log10	常用対数
modf	整数部と小数部の分割
pow	ベキ乗
sin	サイン
sinh	ハイパボリックサイン
sqrt	平方根

tan	タンジェント
tanh	ハイパボリックタンジェント

●数学処理関数 math.h を使ってみる

ここでは，我々がプログラムを組むとき最も使用する割合の高い数学処理系 math.h 関数を利用してみる．

● math.h 数学処理系関数を使うときの注意点

1.**関数の戻り値は** double **型である**

2.**関数の引数は** double **型である**

3.**角度はラジアンで表す．**

サイン 30 **度の値を求めるときは，**

```
d=sin(30.0);
```

ではなく

```
d=sin(30.0*3.14159/180.0);
```

とする．

・sin,cos,tan,**ベキ乗値，常用対数，平方根を求めるプログラム**

```
/* smp19.c */
#include<stdio.h>
#include<math.h>

main()
{
        printf("sin 30 °=%lf",sin(30.0*3.14159/180.0));
        printf("cos 30 °=%lf",cos(30.0*3.14159/180.0));
        printf("tan 45 °= %lf",tan(45.0*3.14159/180.0));
        printf("2の 3乗 =%lf",pow(2.0, 3.0));
        printf("1000 の常用対数= %lf",log10(1000.0));
        printf("10 の平方根= %lf",sqrt(10.0));


}
```

■ファイル処理

プログラムには，データを読み込んだり保存したりする機能が欠かせない．幾つかのデータを読み込むだけなら，手で入力してもいいだろう．しかし，データの数が多くなってくると手でいちいち入力するのは効率が良くないうえに，誤入力の可能性もある．

そこで，プログラムに必要なデータをあらかじめファイル化しておき，それを入力ファイルとして利用するとよい．また，プログラム内で計算した結果をファイルに出力しておくと，結果を考察したり，加工することができる．

ファイル入出力の処理をするとき，いきなりファイルの中の文字を読んだり，書いたりすることは出来ない．次のような手順が必要である．

（1）ファイルを開く

（2）データを読み書きする

（3）ファイルを閉じる

Cでは，ファイルを開くとき fopen 関数を使い，閉じるときには fclose 関数を使う．

```
fp = fopen("sample.txt","r")
```

変数を使ってファイル名を指定するときは，

```
fp = fopen(filename,"r")
```

この意味は，

"sample.txt"というファイルを入力用(r)に
ファイルポインタ fp を使ってオープンしろ　となる．

"r"は，read用すなわち，読み込み用という指示である．その他は以下の通り．

"w"	書き込み
"a"	追加
"r+"	更新（既存のファイルを読み書きできるように開く）
"w+"	更新（既存のファイルの内容は消去される）
"a+"	更新（更新用に開き，その末尾に書く）

ファイルを閉じるときは，下のようになる．

```
fclose(fp);
```

・アルファベットを大文字にするプログラム

```
/* smp20.c */
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define INFILE "data.txt"              /* 入力ファイル名 */
#define OUTFILE "result.txt"           /* 出力ファイル名 */

main()
{
  int c, d;
  FILE *intStream, *outStream;

  if((inStream = fopen(INFILE,"r")) == NULL){
      printf("ファイル%sがオープンできません. \n",INFILE);
      exit(1);
  }

  if((outStream = fopen(OUTFILE,"w")) == NULL){
      printf("ファイル%sがオープンできません. \n",OUTFILE);
      exit(1);
  }

  while((c = fgets(inStream)) !=  EOF){
      d=toupper(c);
      fputc(d, outStream);
  }
  fclose(outStream);
  fclose(inStream);

}
```

● fopen でオープンされたファイルとの基本的な入出力動作は，これまでにやった printf や scanf,putchar,getchar に似ている．

fgetc(ファイルポインタ)　ファイルから 1 文字読み込む

fputc(文字列, ファイルポインタ)　ファイルに 1 文字書き込む

ファイルポインタとは，詳しくは次章で説明するが，ここでは”ファイル名を表している変数”と思ってよい．

●#define

マクロ定義するための処理であるが，ここでは”**以下のプログラム中で INFILE とある場合，data.txt として処理する**”といった定義をしている．

● NULL,EOF

NULL とは，単に 0 であったり，(void　*)0 といった定義がされているが環境によって異なるので，詳しく知る必要はない．

EOF とは，EndofFile でファイルの終端を示す特別な値である．

● toupper 関数

toupper は，アルファベットの小文字を大文字に変換する関数で，ヘッダー・ファイル <ctype.h>の中でプロトタイプ宣言されている．

上のプログラムは，文字を読み込むものであったが，我々が取り扱う数値計算プログラムでは，データファイルから数値を読み込んで，計算をおこない，計算結果を出力ファイルに出力するという作業を行う．そこで，数値データを取り込んで計算を行うプログラムを紹介する．

data.txt ファイルには以下のような数値を入れておく．

```
10
30
100
55
```

プログラムを実行すると，結果が result.txt に記録されているはずである．

● fscanf と fprintf

smp21.c プログラムでは，ファイルから 1 データを読み込むのに fscanf を用い，1 データをファイルに出力するとき fprintf を用いている．

fscanf(ファイルポインタ，フォーマット，引数)　　例:fscanf(fi,"%d",&a);

fprintf(ファイルポインタ，ファーマット，引数)　　例:fprintf(fo,"%d¥n",a);

使い方は，難しくないはずである．ファイル入出力のプログラムは，ほとんどこれら 2 つのサンプルと同じであるので，以後参考にすると良い．

・ファイルから整数を読み合計と平均を計算するプログラム

```
/* smp21.c */
#include <stdio.h>
#include <stdlib.h>

#define INFILE "data.txt"          /* 入力ファイル名 */
#define OUTFILE "result.txt"       /* 出力ファイル名 */

main()
{
        int data, sum=0, num=0;
        double avr;
        FILE *inStream, *outStream;

        /* 入力用ファイルのオープン */
        if((inStream = fopen(INFILE, "r")) == NULL){
                printf("ファイル%sがオープンできません. \n",INFILE);
                exit(1);
        }

        /* ファイルから整数を入力して合計する */
        while(fscanf(inStream,"%d",&data) != EOF){
                sum += data;
                num++;
        }

        /* 入力用ファイルのクローズ */
        fclose(inStream);

        /* 出力用ファイルのオープン */
        if((outStream = fopen(OUTFILE, "w")) == NULL){
                printf("ファイル%sがオープンできません. \n",OUTFILE);
                exit(1);
        }

        /* 合計と平均を出力する */
        avr=(double)sum/num;
        fprintf(outStream,"合計：%d,平均：%f\n",sum,avr);

        /* 出力用ファイルのクローズ */
        fclose(outStream);

}
```

もう一つサンプルを紹介する．

下のプログラムは，データファイルから，データを読み込んで配列に値を代入するものである．降雨量から流出量を求めるプログラムなどに利用できるであろうから，参考にするとよい．

ここでは，読み込むときに fread 関数を，用いている．

```
/* smp22.c */
#include <stdio.h>
#include <stdlib.h>

#define MAX 10                          /* データの個数 */
#define INFILE "result2.txt"                    /* 入力ファイル名 */

main()
{
        int i,data[MAX];
        size_t max;
        FILE *inStream;

        /* 入力用ファイルのオープン */
        if((inStream = fopen(INFILE, "r")) == NULL){
                printf("ファイル%sがオープンできません. \n",INFILE);
                exit(1);
        }

        max = fread((void *)data, sizeof(int), MAX, inStream);

        for(i=0; i<(int)max; i++)
                printf("%d\n",data[i]);

        /* 入力用ファイルのクローズ */
        fclose(inStream);
}
```

■ポインタ

C言語の最大の難関がこのポインタである.

ポインタとは，データなどを直接操作するのではなく，”間接的に”操作できるようにする仕組みのことを指す．そのメリットは以下の3点に集約できる.

(1)配列を扱いやすく，コードがすっきりする

(2)関数から呼び出し側の変数の中身を操作できる

(3)動的にメモリーを管理できる.

しかし，通常の変数と配列を使用するだけでも，十分にプログラミングはできるので，無理をして覚えることはない．ただ，ファイル入出力の所でも出てきたように，ちょこちょこと登場するので，まずはさっと流して，のちにじっくりと勉強すると良い.

これ以降，”猫でもわかるプログラム”を参考にしています.

```
int a;
```

とすると、メモリのどこかに変数ａのデータの格納場所が確保されます。その大きさは int 型の大きさです。プログラマは、そのアドレス（住所）を知りません。そのアドレスは、&a で表すことができます。 &a のようなアドレスを値とする変数が考えられます。この変数をポインタと言います。

ptr = &a;

とすれば、ptr がポインタとなります。ptr はどのように宣言するかというと

int *ptr;

のようになります。これをよく見ると *ptr という変数が int 型であると言うようにも見えます。

int (*ptr);

変数に＆をつけると、その変数のアドレスを表す。
ポインタに＊をつけると、そのポインタの示す格納場所の内容を表す。
と覚えてしまいましょう。ちなみに、＆とか＊も演算子の一つです。

```
/* smp23.c */
#include <stdio.h>

main()
{
        int a = 12, b=30;
        int *ptr;

        ptr = &a;                       /* 変数aのアドレスをptrに代入 */
        printf("a=%d\n",*ptr);
        a=*ptr * 3;                     /* ptrの指す内容を3倍してaに代入 */
        printf("a=%d\n",a);
                                /* %xは16進数表示 */
        printf("aのアドレスは%d(%x)\n",&a,&a);
        printf("bのアドレスは%d(%x)\n",&b,&b);
        printf("ptr=%d(%x)\n", ptr, ptr);

}
```

次に配列とポインタの関係.
int a[4];とすると a[0], a[1], a[2], a[3]が
メモリ中のどこかに、**この順**で確保されます。

「この順で」というところ. ここが大事なのです。バラバラの所に確保されるのではなく、「この順に」固まって確保されるのです。仮に、a[0]が１００番地に確保されたとします。int 型が２バイトだとすると a[1]は１０２番地、a[2]は１０４番地に確保されます。そこで、int 型をさすポインタを ptr とすると、

a[0] = 15; a[1] = 20; a[2] = 25; a[3] = 10;と各要素に代入
ptr = &a[0];　　ptr は a[0]のアドレス（１００番地）が代入された・・・当たり前！

*ptr　　a[0]の中身すなわち 15・・・これも当たり前！
&a[0] + 1　　a[1]のアドレスすなわち１０２番地　　・・・エエエエッ？？？
ptr + 1　　上と同じ
&a[0]　　と a は同じ　　・・・・うううううう

どうです？わかりました？多分上の３行目までは理解できると思います。問題は４行目以降です。&a[0]は１００番地を表しているのにどうして &a[0] + 1 が１０２番地を表すの？はい、それはCの約束だから仕方がないのです。つまり、&a[0] + 1 は a[0]が int 型であるため、その次の要素（a[1]）のアドレスを表しているのです。最初に書いた「この順に」という意味がここにあるのです。そしてさらに驚くべきことに 配列名［0］のアドレスは、配列名で表すことができるのです。従って、a[0]のアドレス（&a[0]）はａなのです。 上の例でもう少しやってみましょう。

&a[0] + 2 は　　a[2]のアドレスすなわち　&a[2]と同じ(１０４番地)
*(&a[0] + 2) は a[2]の内容すなわち　２５
ptr + 2　は　　a[2]のアドレスすなわち &a[2]と同じ
*(ptr + 2)は　　a[2]の内容すなわち　２５
&a[0] は　　a[0] のアドレスで　この値は、ptr に代入されている
a は　　a[0]のアドレスを表すので ptr と同じ

どうです。この位例を出せば何となくわかってきたのではないでしょうか。実際に、本当にそうなのかプログラムを作ってみましょう。

```
/* smp24.c */
#include <stdio.h>

main()
{
        int i, a[4], *ptr;

        a[0]=15;
        a[1]=20;
        a[2]=25;
        a[3]=10;

        ptr = &a[0];
        for(i=0; i<=3;i++){
                printf("a[%d]=%d\n",i,a[i]);
                printf("*(&a[%d])=%d\n",i,*(&a[i]));
                printf("*(&a[0]+%d)=%d\n",i,*(&a[0]+i));
                printf("*(ptr+%d]=%d\n",i,*(ptr+i));
                printf("*(a+%d)=%d\n",i,*(a+i));

        }

}
```