例題 6　2 次方程式の解の判別

example-6.c

```
#include <stdio.h>

int main(void)
{
        double a, b, c, d;

        printf("a b c を入力してください ¥n");
        scanf("%lf %lf %lf", &a, &b, &c);

        d = b * b - 4 * a * c;
        printf("(%gx^2) + (%gx) + (%g) = 0 は ", a, b, c);

        if (d > 0) {
                printf("異なる 2 つの実数解を持ちます ¥n");
        }
        else if (d == 0) {
                printf("重解を持ちます ¥n");
        }
        else {
                printf("異なる 2 つの虚数解を持ちます ¥n");
        }

        return 0;
}
```

実行結果

```
a b c を入力してください
1 3 2 ←キーボードから入力
(1x^2) + (3x) + (2) = 0 は 異なる 2 つの実数解を持ちます
```

1.4.3 課題

課題 5　$sin(\alpha+\beta)$ をそのまま計算する場合と sin の加法定理を用いて計算する場合の両方のプログラムを作り結果を比較してみよう.

課題 6　数字を入力し，入力された数字が偶数か奇数か判別して画面に表示するプログラムを書いてみよう.

課題 7　例題 6 のプログラムを書き換えて，虚数解も含んだ 2 次方程式の解を表示するプログラムを書いてみよう.

1.5 繰り返し文

今回は**繰り返し文**を学びます．繰り返し文では，条件式が真である限り同じ処理を繰り返し (ループ) ます．if などの条件文と合わせて**制御文**と呼ばれるプログラミングにおいて大変重要な構文です．

1.5.1 インクリメント · デクリメント

繰り返し文の前に**インクリメント**と**デクリメント**について説明します．C 言語ではある変数に 1 を足したり，ある変数から 1 を引いたりすることがよくあります．これはもちろん

```
i = i+1;
i = i-1;
```

のように書くことができますが,

```
i++;
i--;
```

と書くこともできます．++ をインクリメント演算子，--をデクリメント演算子と呼びます．C 言語に慣れてくると後者の書き方の方が読みやすくなります．また，コンピュータ内部の処理の違いにより後者の方が高速で動作します．

例題 7　インクリメント・デクリメント

example-7.c

```
#include <stdio.h>
int main(void)
{
        int i = 10;
        int j = 100;

        i++;    /* インクリメント */
        printf("i = %d¥n", i);

        j--;    /* デクリメント */
        printf("j = %d¥n", j);

        return 0;
}
```

実行結果

```
i = 11
j = 99
```

ほぼ同じ意味で

```
++i;
--i;
```

と書くこともできますが，少し動作が違ってきます．インクリメント · デクリメント演算子を変数の後に置くと (ex i++)，式の中でその変数が使用された後にインクリメント · デクリメントされます．変数の前に置くと (ex ++i) イン

クリメント · デクリメントされてから変数が使用されます．ちょっとややこしいですが，通常は i++ と i--を使用すればいいでしょう．

例題 8　i++ と ++i の違い

example-8.c

```
#include <stdio.h>
int main(void)
{
        int i, j;

        i = 5;
        j = i++;
        printf("i=%d j=%d¥n", i, j);

        i = 5;
        j = ++i;
        printf("i=%d j=%d¥n", i, j);
        return 0;
}
```

実行結果

```
i=6 j=5
i=6 j=6
```

1.5.2　繰り返し for

ある処理を一定の回数繰り返したい場合に使用するのが for 文です．

```
for（初期化式；条件式；増分）{
        条件式が真のときの処理；
}
```

処理が 1 行で済む場合，ブロックを使わずに以下のようにも書けます．

```
for（初期化式；条件式；増分）条件式が真のときの処理；
```

次の例では初期化式で i = 0 とし，条件式で i <= 16 である限り for 以下のブロック内の処理を繰り返します．処理が終わるごとに i が 1 ずつ増えます．簡単に言うと，処理を 16 回繰り返してからループを抜けると言うことです．

例題 9　for 文で 2 のべき乗を求める

example-9.c

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

int main(void)
{
        int n, An, Sn = 0; /* n 項数, An 第 n 項, Sn 第 n 項までの和 */

        printf("nY=t2 の n 乗 Y=t 数列の和 Y=n");
        for (n = 1; n <= 16; n++) {
                An = pow(2 ,n); /* 2 の n 乗を計算 */
                Sn = Sn + An;
                printf("%dY=t%dY=t%dY=n", n, An, Sn);
        }

        return 0;
}
```

実行結果

n	2 の n 乗	数列の和
1	2	2
2	4	6
3	8	14
4	16	30
5	32	62
6	64	126
7	128	254
8	256	510
9	512	1022
10	1024	2046
11	2048	4094
12	4096	8190
13	8192	16382
14	16384	32766
15	32768	65534
16	65536	131070

この例題を繰り返し文を使わずに書くと，かなり大変だということがわかると思います.

1.5.3 代入演算子

例題 9 に

```
Sn = Sn+An;
```

という行がありましたが，次のように書くこともできます.

```
Sn += An;
```