第3章　C言語基礎編

3.1　Cの演算子

「+」や「-」などの演算子は通常の数学の演算子と同じだが、「*」（＿＿＿＿）や「/」（＿＿＿＿）、「%」（＿＿＿＿＿＿＿＿＿＿）は、C言語などのプログラミング言語に特有の書き方なので注意する。また、「/」は演算数が両方とも整数の場合は＿＿＿＿＿＿＿＿になる。

またCの演算子にも優先順位がある。算術演算子については、通常の数学で用いられる規則と同じで、「*」や「/」が、「+」や「-」より優先順位が高い。その他の演算子の優先順位については、参考書を参照すること。

Cでは変数の値を変更するという操作が多用されるので、そのための演算子が多数用意されている。

演算子	意味	例
++	変数の値を1増やす。（＿＿＿＿＿＿＿＿する。）	x++, ++x
--	変数の値を1減らす。（＿＿＿＿＿＿＿＿する。）	x--, --x
+=	x += y は、x = x + y と同じ。	x += 10

他に -=, *=, /=, %=なども、+=と同じように考えることができる。

（参考）　一般に、C言語では、「=」のような代入演算子も含めて、演算子を適用した結果は構文上は「式」であり、値を持っている。代入演算子の場合、その値は、代入後の値である。例えば、

```
x = (y = 2)
```

という式では、y = 2という式の値は2であり、そのため、yだけでなく、xにも2が代入される。さらに=演算子は右結合な（右にあるものが優先される）ので、単に

```
x = y = 2
```

と書くこともできる。

「式」の後にセミコロン（;）をつけると構文上「文」になる。「式」にセミコロンをつけて「文」にした時は、式の値は使わずに捨てることになる。代入演算子の結果やprintfのような入出力関連の関数の呼出しは、その値を捨てることが多い。

問 3.1.1　++xとx++の違いを調べよ。

……
……
……

3.2　データ型

C 言語のデータ型としては、これまでに int 型（＿＿＿＿）、double 型（＿＿＿＿＿＿＿＿＿＿）などを紹介してきた。この他に float 型（double より精度が低い浮動小数点数型）、char 型（「ちゃー」型と読む。＿＿＿＿）などがある。

問 3.2.1 浮動小数点数について調べよ。

.....
.....
.....
.....

char 型は文字型といっても、実際には各文字に対応する整数として表現されている。この対応のことを**文字コード**という。文字コードには、パソコンで利用される ASCII コード、大型コンピュータで利用される EBCDIC コードなど、いくつかの種類がある。例えば'a' という文字は ASCII コードでは 97 という数に対応している。

問 3.2.2 *ASCII* コードについて調べよ。

.....
.....
.....
.....

また、構造体型、配列型などの複合データ型がある。配列型については後ほど紹介する。

データ型のサイズは C 言語の仕様では定められておらず、処理系依存である。bcc32 の場合、char 型は＿＿＿＿＿、int 型と float 型は＿＿＿＿＿、double 型は＿＿＿＿＿である。

3.3　C の制御構造

本格的なプログラムを作成するためには、条件が成り立つかどうかによって処理を分けたり、ある条件が成り立つ間、処理を繰り返したりすることが必要になる。ここでは、そのために必要な＿＿＿＿＿を学ぶ。

3.3.1　条件分岐

C++では、基本的には文は上から順に実行されるが、ある条件が成り立つかどうかによって、処理を分けたいことがよくある。そのような場合、＿＿＿＿あるいは、＿＿＿＿＿＿＿＿を用いる。

例題 3.3.1 （時間の計算） 時刻を入力して、その *2* 時間 *46* 分後の時刻を出力するプログラムを書け。
ファイル addTime.c

```
#include <stdio.h>

main()
{
    int hour1, minute1;
    int hour2, minute2;

    printf("何時ですか? ");
    scanf("%d", &hour1);
    printf("何分ですか? ");
    scanf("%d", &minute1);

    hour2   = hour1 + 2;
    minute2 = minute1 + 46;

    if (minute2 >= 60) {
        hour2 = hour2 + 1;
        minute2 = minute2 - 60;
    }

    printf("その 2 時間 46 分後は、%d 時 %d 分です。¥n", hour2, minute2);

    return 0;
}
```

if 文の一般的な形は

```
if (条件式) 文
```

である。条件式が成り立てば _____。成り立たなければ _____。

上の例の場合、if の後の `minute2 >= 60` が _____と呼ばれる部分である。この条件が成り立つ場合繰り上がりの処理をおこなう。繰り上がりの処理として、

```
hour2 = hour2 + 1;
minute2 = minute2 - 60;
```

の 2 つの文を実行しなければいけないので、この 2 つの文をひとまとまりにするために、中かっこでくくっている。このように、いくつかの文を中かっこでくくったものを _____と呼ぶ。ブロックは 1 つの文と同じように扱うことができる。

ブロックの中は見やすいように、2〜4 字分、字下げ（_____）をしておく。空白（タブ文字を含む）がいくつ続いても、C のコンパイラは 1 つの空白と同様に扱う。

問 3.3.2

1. 例題 *3.3.1* のプログラムを *2* 時間 *46* 分前の時刻を求めるように変更せよ。
2. 例題 *3.3.1* のプログラムを *hour2* が *24* を越えた場合にも対応できるようにあらためよ。

if 文の場合は、条件式が成り立たなかった場合は何もせずに、次の文に進むが、成り立たなかった場合も何か別の処理を行ないたい場合がある。このような時は if〜 else 文を用いる。

`if` (条件式) 文$_1$ `else` 文$_2$

条件式が成り立つ時は ＿＿を、そうでない時は ＿＿を実行する。

例題 3.3.3 （時刻の判定） 時刻を入力して午前か午後かを判定せよ。

ファイル ifElse.c

```
#include <stdio.h>

main()
{
    int hour;

    printf("何時ですか? ");
    scanf("%d", &hour);

    if (hour < 12) {
        printf("午前ですね。¥n");
    } else {
        printf("午後ですね。¥n");
    }

    return 0;
}
```

この例のように、分岐処理の中の文が 1 つの場合は、必ずしもその周りを中かっこで囲む必要はないが、いつも囲むように習慣付けておく方が良い。（あとで書き換える時に便利である。）

`if`〜 `else` 文は次のようにいくつも続けることができる。

```
    if (hour < 12) {
        printf("おはようございます。¥n");
    } else if (hour < 18) {
        printf("こんにちは。¥n");
    } else {
        printf("こんばんは。¥n");
    }
```

これは単に `else` の次の文が、また `if` 文になっているととらえることができる。

問 3.3.4 （ぶらさがりの *else*）

```
if (hour < 12) if (hour < 6) { printf("A"); } else { printf("B"); }
```

という文では、*else* はどちらの *if* に対応するか? つまり

```
if (hour < 12) {
    if (hour < 6) {
        printf("A");
    } else {
        printf("B");
    }
}
```

```
if (hour < 12) {
    if (hour < 6) {
        printf("A");
    }
} else {
    printf("B");
}
```

のどちらとして解釈されるか? 実際に実行して確かめよ。

.....

.....

3.3.2 インデンテーション

C 言語は、かなり自由に空白や改行を入れることができる。通常の文脈では、2 つ以上の空白は、1 つの空白と同じ意味であるし、改行も空白と同じ意味しか持たない。気をつけなければいけないところは、

- #から始まる命令、つまり`#include`文や`#define`文（後述）、は 1 行に 1 つ記述する。1 行に 2 つ以上の命令を書いてはいけないし、1 つの命令を 2 行以上にわけてもいけない。
- 二重引用符（"）や一重引用符（'）の間では、改行できない。

などである。

プログラムを読みやすくするために、インデンテーションは必ず行なうようにする。以下に典型的なインデンテーションの方法を紹介する。

- 中括弧の中は、必ず外よりも _____
- 開き括弧は改行して、`if`や`for`などの _____と列を揃える（下図左）か、改行せずに続けて書く（下図右）。

```
if (x == 0)
{
    x = 1;
}
```

または

```
if (x == 0) {
    x = 1;
}
```

閉じ中括弧もキーワードと列を揃える、

改行は空白と同等に扱われる。通常は 1 つの行に 1 つの文を書くことが多いが、複数の文を書いても構わない。また長いプログラムの場合はまとまった処理の間に空行を挿入するなど、プログラムを見易くする工夫をする。

3.3.3 関係演算子

関係演算子には、 >, <, >=, <=などの不等号のほか、 ==（等しい）, !=（等しくない）などがある。特に==は、 =と間違えないように注意する。 C では _____演算子である=と、 _____演算子である==とは全く異なる。

問 3.3.5

1. *2* 次方程式の解の係数を入力して、解を判別するプログラムを書け。
2. 三角形の三辺の長さを入力して、三角形の種類（鋭角、直角、鈍角、三角形はできない）を判定するプログラムを書け。（ヒント: 面倒を避けるために、短い辺から順に入力するようにしても良い。）

3.3.4　繰り返し（ループ）

コンピュータは何万回、何億回の繰り返しであろうと文句を言わずにしかも高速に実行してくれる。これこそがコンピュータの最大の存在理由であるといえる。

C で同じ処理を何度も繰り返すためには while 文、for 文、do〜while 文のいずれかを用いる。ここでは、while 文、for 文を紹介する。

while 文

while（条件式 $_1$）文 $_1$

while 文は ________ が成り立つ間、____ の実行を繰り返す。 条件式 $_1$ が最初から成り立たなければ、文 $_1$ は ____________________ ことに注意する。

例題 3.3.6 正の数を入力して、その数だけ '*' を出力する。

ファイル *while1.c*

```
#include <stdio.h>

main()
{
    int num;

    printf("正の数を入力してください。 ");
    scanf("%d", &num);

    while (num > 0) {
        printf("*");
        num--;
    }
    printf("¥n");

    return 0;
}
```

例えば、入力された数値が 10 ならば num は 10, 9, 8, . . . , 1, 0 と減っていく。num が 0 になったときはもう '*' は出力されないので、結局 '*' は 10 回出力される。

for 文

for（式 $_1$; 式 $_2$; 式 $_3$）文 $_1$

for 文は ________________ に、まず ____ を評価する。

____ が成り立つ間、

____ と ____ の評価を繰り返す。

やはり 式 $_2$ が最初から成り立たなければ、文 $_1$ は ____________________。

例題 3.3.7 正の数を入力して、その数だけ '*' を出力する （*for* バージョン)。

ファイル for1.c

```
#include <stdio.h>

main()
{
    int i, num;
    printf("正の数を入力してください。 ");
    scanf("%d", &num);

    for (i = 0; i < num; i++) {
        printf("*");
    }
    printf("¥n");

    return 0;
}
```

先ほどの while 文を使った例題を for 文を使って書き直したものである。今度は num は変化せずに、i が、0, 1, . . . と num まで変化する。i が num に等しくなったときは、'*' は出力されないので、やはり '*' はnum（入力した数）個だけ出力される。num が 0 以下の時はfor の中は一度も実行されないことに注意する。

例題 3.3.8 正の整数 *n* を受け取って *n*!（*n* の階乗）を計算する。

ファイル factorial.c

```
#include <stdio.h>

main()
{
    int i, n, fact;
    printf("正の数を入力してください。 ");
    scanf("%d", &n);

    fact = 1;
    for (i = 1; i <= n; i++) {
        fact = fact * i; /* fact *= i; と書いてもよい */
    }
    printf("あなたの入力した数の階乗は %d です。¥n", fact);

    return 0;
}
```

実行結果:

```
C:¥...>factorial
正の数を入力してください。 10
あなたの入力した数の階乗は 3628800 です。
```

変数の変化の様子:

i	1	2	3	4	5	. . .	10
fact	1	2	6	24	120	. . .	3628800

問 3.3.9 このプログラムにあまり大きな入力を与えると答が変になる。 何故か？

これらの例題からわかるように、for 文には良く使われるパターンがいくつかある。

`for (i = 0; i < n; i++) ...`	i が 0 から n-1 まで、計 _回 ... を繰り返す。
`for (i = 1; i <= n; i++) ...`	i が 1 から n まで、計 _回 ... を繰り返す。
`for (i = n; i > 0; i--) ...`	i が n から 1 まで、計 _回 ... を繰り返す。
`for (i = n - 1; i >= 0; i--) ...`	i が n-1 から 0 まで、計 _回 ... を繰り返す。

問 3.3.10 正の整数 n を受け取って 2^n（2 の n 乗）を計算するプログラムを書け。

問 3.3.11 正の整数 n, r を受け取って ${}_nC_r \left(= \dfrac{n \cdot (n-1) \cdot \cdots \cdot (n-r+1)}{r \cdot (r-1) \cdot \cdots \cdot 1}\right)$ を計算するプログラムを書け。

問 3.3.12 正の整数 n を受け取って n の約数をすべて出力するプログラムを書け。（ヒント: 余りを求める演算子は %演算子をつかう。x を y で割った余りは x % y である。）

問 3.3.13 正の整数 n を受け取って 1 から n までの数の平方根を出力するプログラムを書け。

例題 3.3.14 数学関数のグラフ
SVG 形式を利用すれば、プログラムでグラフィックスを生成することもできる。
ファイル *mathGraph.c*

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

main()
{
    double x,y;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE svg PUBLIC '-//W3C/DTD SVG 1.0//EN'¥n");
    printf("                     'http://www.w3.org/TR/SVG/DTD/svg10.dtd'>¥n");
    printf("<svg width='%d' height='%d'>¥n", 315, 100);

    printf("<polyline fill='none' stroke='blue' points='");
    printf("%d, %d ", 0, 60);   /* グラフの最初の点を出力する。 */
    for (x = -2 * M_PI; x <= 2 * M_PI; x += 0.1 * M_PI) {
        y = sin(x) - sin(2 * x) / 2 + sin(3 * x) / 3 - sin(4 * x)/4;
        printf("%f, %f ", x * 25 + 50 * M_PI, 60 - y * 25);
    }
    printf("' />¥n");

    printf("</svg>¥n");

    return 0;
}
```

注: SVG（一般に XML）の中では、文字列を囲む引用符は二重引用符（"）でも一重引用符（'）でも構わないので、このプログラムでは一重引用符（'）を利用している。C 言語の文字列定数の中に二重引用符（"）を入れる時は、\"と書かなければいけない。

網かけの部分は、SVG を出力するすべてのプログラムに共通である。

このまま実行すると、SVG 形式の出力結果が文字として画面に表示されるだけなので、この出力をファイルに保存する必要がある。

```
matGraph > mathGraph.svg
```

というコマンドを実行すれば、出力が`mathGraph.svg` というファイルに保存される。一般に Windows のコマンドプロンプトで

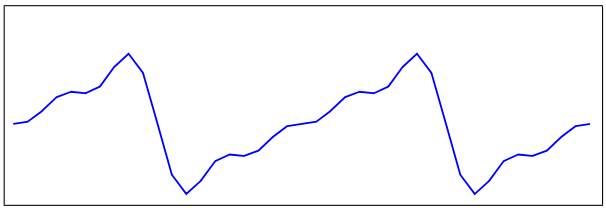
```
コマンド > ファイル
```

という形でコマンドを実行すると、コマンドの出力が画面ではなくファイルに保存される。これを **リダイレクト** という。逆に

```
コマンド < ファイル
```

という形でコマンドを実行すると、キーボードの代わりにファイルから入力が与えられる。

このプログラムは $y = \sum_{k=1}^{4} (-1)^{k-1} \frac{\sin kx}{k}$ のだいたい $[-2\pi, 2\pi]$ の範囲のグラフ（下図）を作成する。



`sin` という数学関数を使っているので、`#include <stdio.h>`の次に

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

という 1 行を入れておく。`M_PI` は定数で π（円周率）のことである。
`y = ...` の行を変えれば、いろいろな関数のグラフに応用できる。

例題 3.3.15 グラデーションを描く

ファイル *gradation1.c*

```
#include <stdio.h>

main()
{
    int i;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE svg PUBLIC '-//W3C/DTD SVG 1.0//EN'¥n");
    printf("                     'http://www.w3.org/TR/SVG/DTD/svg10.dtd'>¥n");
    printf("<svg width='%d' height='%d'>¥n", 256, 64);

    for(i = 0; i < 64; i++) {
        printf("<rect x='%d' y='0' width='4' height='64'¥n", i * 4 );
        printf("      stroke='none' fill='#%02X%02X%02X' />¥n", i * 4, 0, (63 - i) * 4)
    }

    printf("</svg>¥n");

    return 0;
}
```

画面の左端が赤で徐々に青にグラデーションする（右図）。

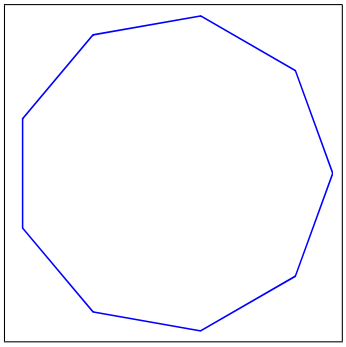
例題 3.3.16 正多角形を描く。

ファイル ngon.c

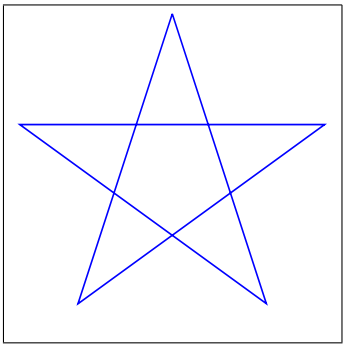
```
#include <stdio.h>
#include <math.h>

main()
{
    int n = 9;
    int i;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE svg PUBLIC '-//W3C/DTD SVG 1.0//EN'¥n");
    printf("                     'http://www.w3.org/TR/SVG/DTD/svg10.dtd'>¥n");
    printf("<svg width='%d' height='%d'>¥n", 200, 200);

    printf("<polyline fill='none' stroke='blue' points='");
    printf("%d, %d ", 200, 100);
    for(i = 0; i < n; i++) {
        double theta = 2 * M_PI * (i + 1) / n;
        printf("%f, %f ", 100 * (1 + cos(theta)), 100 * (1 + sin(theta)));
    }
    printf("' />¥n");

    printf("</svg>¥n");

    return 0;
}
```

やはり #include <math.h>が必要である。これは正 9 角形だが、n を変えると他の正多角形も描くことができる。なお、このプログラムでは、変数theta はfor 文のブロックの中で宣言されている。このようにブロックの中で宣言された変数は、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿有効である。ブロックの外（}より後ろ）ではこのtheta を参照することはできない。

正多角形



星型



問 3.3.17 上図右のような星形を描くプログラムを作成せよ。

問 3.3.18 *mathGraph.c* を参考にして、他の数学関数のグラフを作成せよ。

例題 3.3.19 リサージュ図形を描く

ファイル lissajous.c

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

main()
{
    int i, n = 200, r = 100;
    int a = 3, b = 5;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE svg PUBLIC '-//W3C/DTD SVG 1.0//EN'¥n");
    printf("                     'http://www.w3.org/TR/SVG/DTD/svg10.dtd'>¥n");
    printf("<svg width='%d' height='%d'>¥n", 200, 200);

    printf("<polyline fill='none' stroke='blue' points='");
    printf("%d, %d ", 100, 0);
    for(i = 0; i < n; i++) {
        double theta = 2 * M_PI * (i + 1) / n;
        printf("%f, %f ", r * (1 + sin(a * theta)), r * (1 - cos(b * theta)));
    }
    printf("' />¥n");

    printf("</svg>¥n");

    return 0;
}
```

例題 3.3.20 渦巻き図形を描く

ファイル spiral.c

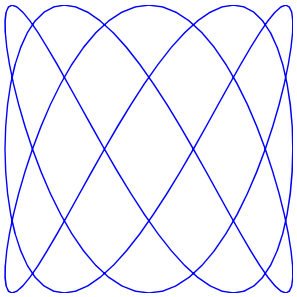
```
#include <stdio.h>
#include <math.h>

main()
{
    int c = 20, r = 100;
    double dt = 3.1, theta, tmp = 1.0 / c / 2 / M_PI;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE svg PUBLIC '-//W3C/DTD SVG 1.0//EN'¥n");
    printf("                     'http://www.w3.org/TR/SVG/DTD/svg10.dtd'>¥n");
    printf("<svg width='%d' height='%d'>¥n", 200, 200);

    printf("<polyline fill='none' stroke='blue' points='");
    printf("%d, %d ", 100, 100);
    for(theta = 0.0; theta < c * 2 * M_PI; theta += M_PI / dt) {
        double x = r * (1 + theta * tmp * cos(theta));
        double y = r * (1 - theta * tmp * sin(theta));
        printf("%f, %f ", x, y);
    }
    printf("' />¥n");

    printf("</svg>¥n");

    return 0;
}
```

例題 3.3.21 リサージュ図形を描く（*3* 次元版）

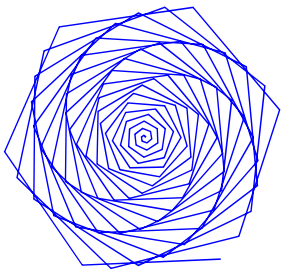
ファイル lissaj3D.c

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

main()
{
    int i, n = 128, r = 5;
    int a = 3, b = 5, c = 2;

    printf("<?xml version='1.0' ?>¥n");
    printf("<!DOCTYPE X3D PUBLIC 'ISO//Web3D//DTD X3D 3.0//EN'¥n");
    printf("          'http://www.web3d.org/specifications/x3d-3.0.dtd'>¥n");
    printf("<X3D profile='Interchange'>¥n");
    printf("<Scene>¥n");
    for(i = 0; i < n; i++) {
        double theta = 2 * M_PI * (i + 1) / n;
        printf("<Transform translation='%f %f %f'>¥n", r * sin(a * theta),
               r * cos(b * theta), r * cos(c * (theta + 0.75 * M_PI )));
        printf("  <Shape>¥n");
        printf("    <Appearance>¥n");
        printf("      <Material emissiveColor='%f %f %f' />¥n", 0.0, 1.0, 0.0);
        printf("    </Appearance>¥n");
        printf("    <Sphere radius='0.1' />¥n");
        printf("  </Shape>¥n");
        printf("</Transform>¥n");
    }
    printf("</Scene>¥n");
    printf("</X3D>¥n");

    return 0;
}
```

リサージュ図形



渦巻き図形



3 次元リサージュ図形（交差法用）

3.4　C のその他の制御構造

引続き、C 言語の制御構造をもう少し詳しく説明する。

3.4.1　繰り返しの構文

ここでは残りの do〜while 文、break 文、continue 文などについて説明する。

do〜while 文

do 文$_1$ while (条件式$_1$);

do〜 while 文は、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿。成り立てば文$_1$を＿＿＿＿＿＿＿＿。条件式$_1$が偽になったときにこのループは終了する。

while 文と for 文は条件が成り立たなければ一度もループが実行されないのに対して、do〜 while 文は少なくとも一度は実行されるという違いがある。

例題 3.4.1 *'?'* を入力したときだけ、*'!'* を出力し、その他の文字はそのまま出力する。¥n で終了。

ファイル *dowhile.c*

```
#include <stdio.h>

main()
{
    char c;
    do {
        c = getchar();
        if (c == '?') {
            putchar('!');
        } else {
            putchar(c);
        }
    } while (c != '¥n');

    return 0;
}
```

例えば、*Hello?*と入力すると *Hello!*と出力する。このプログラムは *C* の入出力関数 *getchar* と *putchar* を用いている。*getchar* は *1* 文字を入力し、*putchar* は *1* 文字を出力する。

break 文と continue 文

break 文は＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿。

例題 3.4.2 入力 n に対して、$(i-1)^2 < n \leq i^2$ となる i（$\sqrt{n}$ を切り上げた数）を出力する。

ファイル *isqrt.c*

```
#include <stdio.h>

main()
{
    int i, n;

    printf("数値を入力してください。");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        if (i * i >= n) break;
    }

    printf("求める整数は %d です。¥n", i);

    return 0;
}
```

問 3.4.3 入力 n に対して、$(i-1)^3 < n \leq i^3$ となる i（$\sqrt[3]{n}$ を切り上げた数）を返す関数を定義せよ。

continue 文は _____。
つまり、continue 文は:
while 文, do〜 while 文の場合　:　条件式の計算に進む。
for 文の場合　:　まず、式 3 を実行し、条件式に進む。
という動作をする。

多重ループ

for や while のループは入れ子（_____とも言う。ループの中にループが入っている状態）にすることができる。

例題 3.4.4 九九の表を表示するプログラムを書け。

ファイル kuku.c

```
#include <stdio.h>

main ()
{
    int i, j;
    for (j = 1; j <= 9; j++) {
        for (i = 1; i <= 9; i++) {
            printf("%d * %d = %2d  ", i, j, i * j);
        }
        printf("¥n");
    }

    return 0;
}
```

このプログラムを実行した結果は次のようになる。

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

このプログラムでは、ループが入れ子になっている。（つまり、ループで繰り返す文がまたループになっている。）変数 i に関するループが内側にあるので、i の方が速く変化する。つまり、j が 1 で留まっている間に i は 1 から 9 まで変化する。その次に j が 2 になる。

すなわち、i と j は

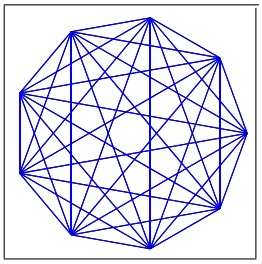
i = _, j = _ ↦ i = _, j = _ ↦ ... ↦ i = _, j = _ ↦
i = _, j = _ ↦ i = _, j = _ ↦ ... ↦ i = _, j = _ ↦
... ↦
i = _, j = _ ↦ ... ↦ i = _, j = _

のような順に変化する。

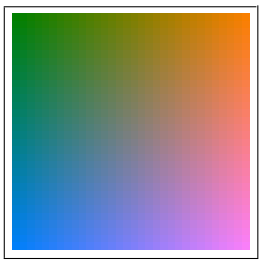
問 3.4.5 正の整数 *n* を受け取って、ディスプレイ上に次のように三角形を表示する関数 *void tri(int n)* を *for* 文を使って書け。
tri(4) の出力

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

問 3.4.6 正 *n* 角形のすべての頂点を結んでできる図形（ダイアモンドパターン）を描画するプログラムを書け。



問 3.4.7 色のグラデーション（2 次元）を作成するプログラムを書け。



問 3.4.8 *mathGraph.c* を参考にして $y = \sum_{k=1}^{n} (-1)^{k-1} \frac{\sin kx}{k}$ （*n* 項までの和）のグラフを描くプログラムを書け。

3.4.2 条件分岐の構文

`if` 文、`if`〜`else` 文についてはすでに説明した。C 言語の条件分岐の構文としてはこの他に `switch`〜`case` 文がある。

```
switch (式) {
    case 定数: 文; ... ; 文; break;
    ...
    case 定数: 文; ... ; 文; break;
    default:   文; ... ; 文; break;
}
```

式を計算して、その結果と一致する `case` を上から順に探し、そのあとの文の並びを実行する。一致する `case` が無い場合は、最後に `default:`があれば、そのあとの文の並びを実行する。`default:`がなければ何もしない。

この場合の `break` 文は _____ という意味になる。

3.4.3　条件式

if 文や for 文、while 文の条件式には、==や>などの関係演算子が使われる。関係演算子と関係の深い演算子として論理演算子（ &&, ||, !）がある。&&は、「〜かつ（AND） 〜」、||は「〜 または（OR） 〜」、!は「 〜でない（NOT 〜）」である。

```
0 <= x && x <= 1   —  0 <= x かつ x <= 1
x < 0 || 1 < x     —  x < 0 または 1 < x
!(x == 0)          —  x == 0 でない（この場合は x != 0 と同じ）
```

不等号などの関係演算子は &&や ||より優先順位が高いので、 0 <= x && x <= 1 で (0 <= x) && (x <= 1) という意味になる。また、 &&は ||より優先順位が高い。つまり a > 0 || b < 0 && c == 0 は (a > 0) || ((b < 0) && (c== 0)) という意味になる。

論理演算子は左から順に計算を行ない、真偽が定まった時点で計算をやめる。例えば x が -1 の時、x < 0 || 1 < x はx < 0 を計算した時点で、真であることがわかるので、1 < x の部分は実行しない。これは時に便利な性質である。

問 3.4.9 うるう年を判定する関数 *int leapyear(int y)* を書け。西暦 *y* 年がうるう年ならば *1* を、うるう年でなければ *0* を返すものとする。（ヒント*:* 原則として西暦で *4* で割れる年がうるう年、ただし *100* で割り切れる年はうるう年ではない。ただしのただし、*400* で割り切れる年はうるう年である。

キーワード*:*

制御構造、if 文、if〜else 文、ブロック、インデンテーション、while 文、for 文、do〜while 文、break 文、continue 文、ネスト、論理演算子、&&、||、!、