入門講習会 第五回

1. 変数のサイズと long long 型

一つの変数に入れられる値には限界がある。

例えば、int 型は(一般的には)$-2^{31}-1 \sim 2^{31}$ の値が入る。

これより大きかったり小さかったりする値を入れると、表現できる限界を超えてしまうため、桁があふれておかしな数値になってしまう。この現象を「算術オーバーフロー(桁あふれ)」という。

もう少し大きな値を扱いたい場合は、`long long` 型を用いる。これは long long int 型の略で、一般的には$-2^{63}-1 \sim 2^{63}$ の値を入れられる。

これより大きな値を使うことはあまりない。組み合わせの問題は答えが非常に大きくなることが多いため、「○○で割った余りを出力してください」と問題文に指定されていることが多い。

余りを求める問題では、何か計算をするたびに余りを求めるようにしなければならない、「やるべき計算をすべて終えてから余りを求める」という方法をとってしまうと、その計算途中に算術オーバーフローしてしまう危険性がある。

[補足]

long long 型は純粋な C 言語(C90)では未実装。しかし gcc では実装されている。long long を知らないと解けない問題もあるので知っておこう。

2. 多次元配列

配列の配列を宣言することができる。厳密には「配列を要素とする配列」というのが正しいのだが、その使われ方から「多次元配列」と呼ばれている。

多次元配列を宣言するときは、配列の宣言における[要素数]の部分を複数書く。

例えば int 型の 3×4 の二次元配列 a を宣言する場合は次のように書く。

```
int a[3][4];
```

要素数 4 の配列を要素する、要素数 3 の配列を宣言することを意味する。

二次元配列の初期化はたとえば以下のように行える。

```
int a[3][4] = {{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 6, 9, 12}};
```

配列の初期化には{}を用いていた。だから、配列の中の要素が配列なら、各要素を書く際に{}を使うのは当然と言えるだろう。

二次元的なものは二次元配列で管理することがある。例えば、平面や行列などである。いくつか例を見てみよう。

[二次元配列の表示]

以下は、5×5 の整数型の配列を宣言し、入力を行った後、それぞれの要素に 2 を加え

て出力するだけのプログラムである。

```c
#include <stdio.h>

int main(void) {
    int a[5][5];
    int i, j;

    for(i = 0; i < 5; i++) {
        for(j = 0; j < 5; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for(i = 0; i < 5; i++) {
        for(j = 0; j < 5; j++) {
            printf("%d ", a[i][j] + 2);
        }
        printf("¥n");
    }

    return 0;
}
```

実行結果は次のようになる。

```
4 3 5 6 8（入力）
2 10 8 9 2（入力）
0 0 1 2 3（入力）
4 8 9 12 3（入力）
0 0 0 0 0（入力）
6 5 7 8 10
4 12 10 11 4
2 2 3 4 5
6 10 11 14 5
2 2 2 2 2
```

初めの二重ループで、scanf を 5×5=25 回呼び出して、二次元配列に要素を入力している。次の二重ループでは、各々の要素に 2 を加えたものを printf で出力している。内側のループを抜けた直後に改行を行うことで、このように 5×5 の行列のような出力をす

ることができる。

[行列の積]
行列の積を計算するプログラムを作ってみよう。今回 A を 4×3 行列、B を 3×5 行列として、AB を計算する。

```c
#include <stdio.h>

int main() {
    int A[4][3], B[3][5], C[4][5] = { 0 };
    int i, j, k;

    /*入力部分*/
    printf("A:¥n");
    for(i = 0; i < 4; i++) {
        for(j = 0; j < 3; j++) {
            scanf("%d", &A[i][j]);
        }
    }
    printf("B:¥n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 5; j++) {
            scanf("%d", &B[i][j]);
        }
    }

    /*積を計算する部分*/
    for(i = 0; i < 4; i++) {
        for(j = 0; j < 5; j++) {
            for(k = 0; k < 3; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    /*出力部分*/
    for(i = 0; i < 4; i++) {
```

```
        for(j = 0; j < 5; j++) {
            printf("%d ", C[i][j]);
        }
        printf("¥n");
    }

    return 0;
}
```

積の部分は三重ループになっている。分かりにくければ紙などに書いてループの動きを追ってみよう。行列の積の計算になっていることが分かるだろう。

[文字型の二次元配列を入力]

文字型の二次元配列を scanf で入力する際は注意が必要である。

例えば文字型の二次元配列 char c[10][10]について、次のように入力したとしよう。

```
int i, j;
char c[10][10];

/*5×5 個の文字を入力(配列の要素は多めにとっている)*/
for(i = 0; i < 5; i++) {
    for(int j = 0; j < 5; j++) {
        scanf("%c", &c[i][j]);
    }
}
```

実は、このプログラムは正しい入力が行われない。

かなり雑にに理由を述べるなら、scanf における%c での入力は、改行文字まで入力に入れてしまうからである。

この問題を回避するために、文字型の配列は文字列とみなせることを思い出そう。すると、c[i]は c[i][1], c[i][2], c[i][3], … の文字が並んだ文字列とみなせる。つまり、次のように書ける。

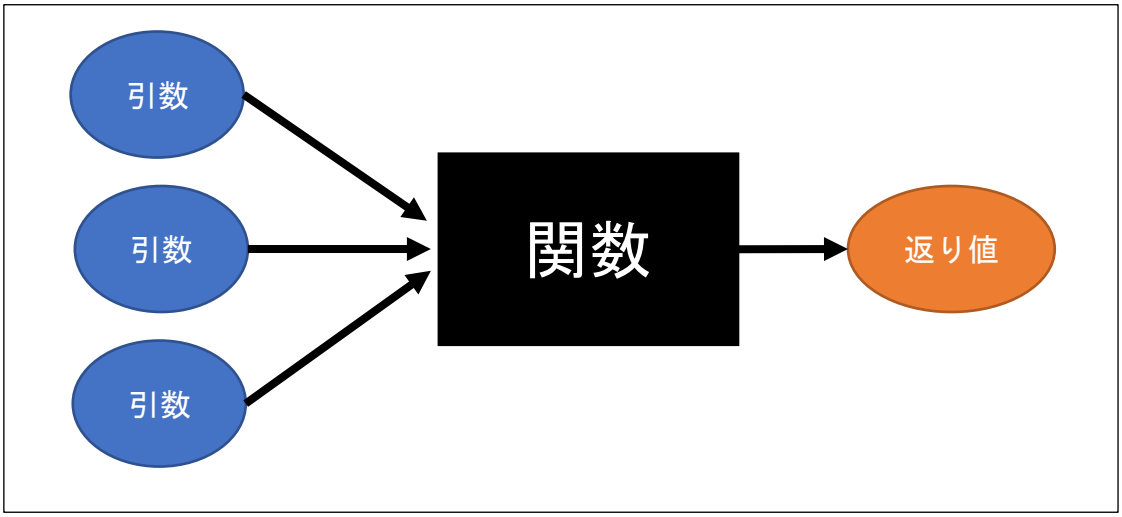
```
int i, j;
char c[10][10];

/*5 個の文字列を入力(配列の要素は多めにとっている)*/
for(i = 0; i < 5; i++) {
    scanf("%s", c[i]);
}
```

[補足]
こうしなければならないのは、scanf の性質によるものである。C++では基本的に入力には cin を利用するので、このような記法で書く必要はない。

3. 関数

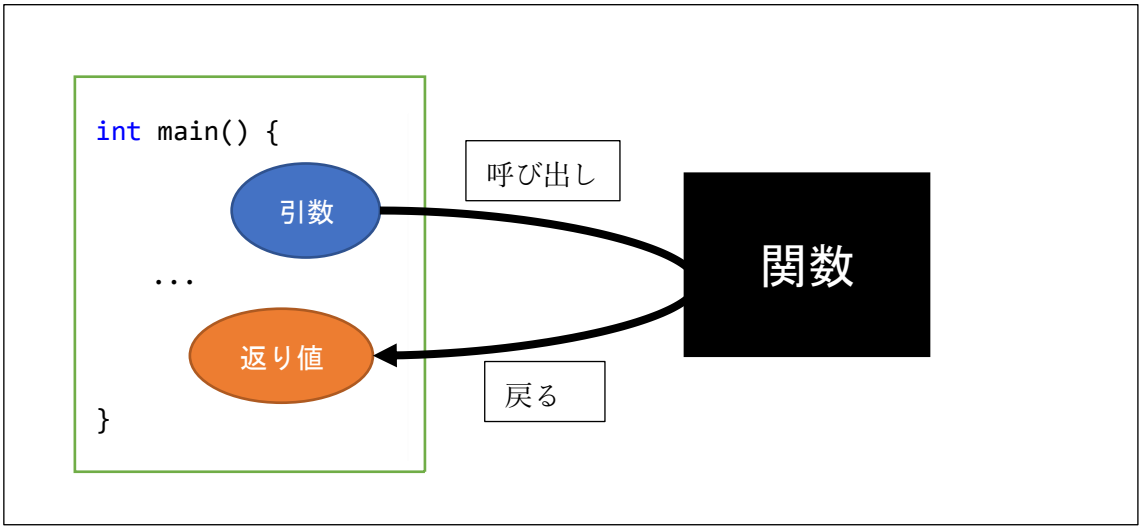
関数とは、端的に言えば、
　入力に対してなんか色々処理して、何かの値を出力する機械のようなもの
である。入力値のことを引数、出力値のことを返り値(戻り値)という。



引数と返り値はあってもなくてもよい。ただし指定できる引数は 0 個以上で、返り値は 1 個以下である。
関数は次の目的で使われることが多い。

・処理を意味的に分けたい
・処理を分割して、汎用性を高める
・再帰処理をする

再帰処理については別のときに説明する。
関数には呼び出し元が必要である。例えば int main()中で関数を呼び出せば、呼び出し元は int main()となる。呼び出した関数内で適当な処理を行った後、返り値とともに再び呼び出し元に戻ってくる。



関数は次の書式で定義をする。

```
返り値の型 関数名(引数の宣言部分) {
    なんか色々処理
}
```

例えば、引数として int 型の値 2 つをとり、double 型の値を返す関数 func は次のような書式で書く。

```
double func(int a, int b) {
    なんか色々処理
}
```

先頭の double を int や char に変えれば、返り値の型を int や char にすることができる。また、void と書けば、値を返さない関数を作ることができる。

引数に配列を指定する場合についての話は別の機会に述べる。

引数をなにも書かないときは、()の中には void と書く。

何か値を返して、呼び出し元に戻りたいときは、return 文を用いる。

```
return 値;
```

関数 func を実際に呼び出すときは、呼び出し元で次のように書く。

```
func(値, 値);
```

このように宣言、呼び出し等の文をバラバラに言われてもわかりにくいと思われるので、実際にこれらを組み合わせた使用例を見てみよう。

[等差数列の和を求める関数]

以下は、整数 a, b, c を入力値とし、初項 a、公差 b、項数 c の等差数列の和を求めるプログラムである。

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

```c
int main(void) {
    int a, b, c;

    scanf("%d %d %d", &a, &b, &c);
    printf("%d¥n", c * (2 * a + (c - 1) * b) / 2);

    return 0;
}
```

和を求める部分を関数として分離してみよう。以下のようになる。

```c
#include <stdio.h>

int sumtousa(int a, int d, int n) {
    return n * (2 * a + (n - 1) * d) / 2;
}

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

    scanf("%d %d %d", &a, &b, &c);
    printf("%d¥n", sumtousa(a, b, c));

    return 0;
}
```

後者のソースコードを細かく見てみよう。まず初めの

```c
int sumtousa(int a, int d, int n) {
    return n * (2 * a + (n - 1) * d) / 2;
}
```

において、int 型を返し、int 型の引数 3 つを取る関数 suntoua を定義している。実際に入力される引数の値は呼び出し側の状況によって異なるので、とりあえずその引数を int 型変数 a, d, n としている。関数定義における引数のことを仮引数と呼び、後述する実引数と区別される。

関数の内容は、単に値

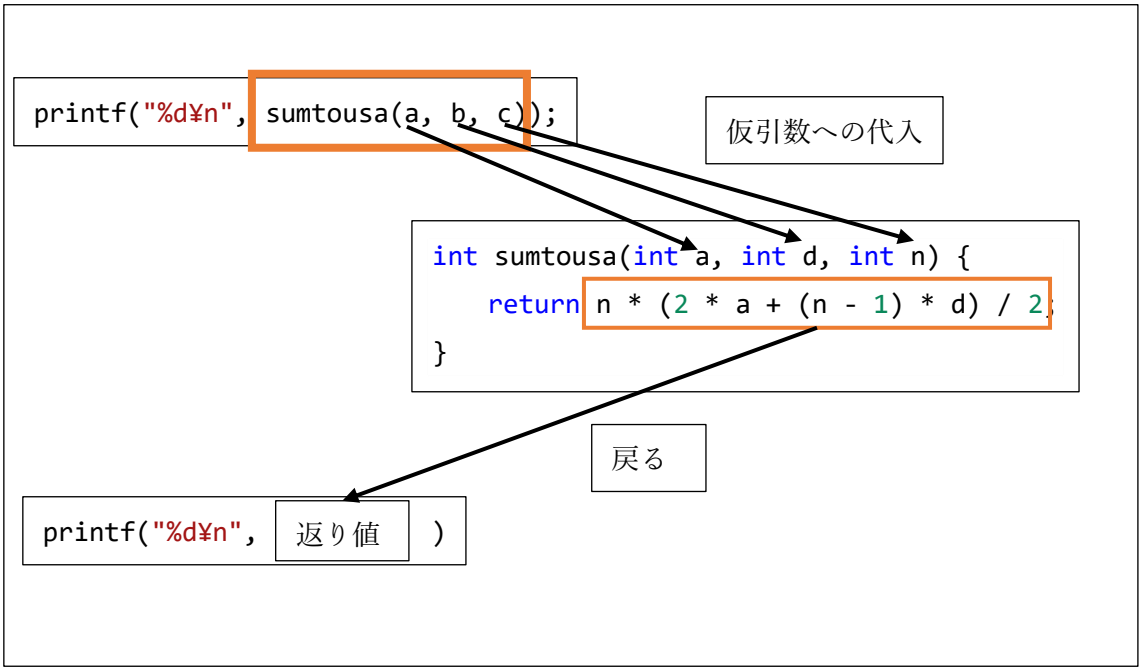
```c
n * (2 * a + (n - 1) * d) / 2
```

を返すだけである。

次に、main 内での関数呼び出し部分を見てみよう。

```c
printf("%d¥n", sumtousa(a, b, c));
```

では、まず sumtousa(a, b, c)を呼び出して、返ってきた値を printf で表示している。ここで、関数呼び出しの時に引数を指定しているが、ここでの引数を実引数と呼ぶ。
関数が呼び出されると、まず実引数の値が仮引数に代入される。つまり今回の例では、実引数 a, b, c の値が、仮引数 a, d, n に代入される。その後関数 sumtousa 内の処理が始まる。処理内容は n * (2 * a + (n – 1) * d) / 2 の計算結果を返すこと。そしてその値が関数呼び出し部分と置き換わる。



[各桁の和を返す関数]
もう少し複雑な関数を作ってみよう、次のプログラムは、ある整数 n が入力されたとき、n の各桁の和を出力するプログラムである。

```c
#include <stdio.h>

int main(void) {
    int n;
    int sum = 0;

    scanf("%d", &n);
    while(n > 0) {
        sum += n % 10;
        n /= 10;
    }
    printf("%d¥n", sum);
```

```
    return 0;
}
```

n の各桁の和を求める部分を関数に分離すると、以下のようなプログラムとなる。

```
#include <stdio.h>

int sumdigit(int num) {
    int sum = 0;
    while(num < 0) {
        sum += num % 10;
        num /= 10;
    }

    return sum;
}

int main(void) {
    int n;

    scanf("%d", &n);
    printf("%d¥n", sumdigit(n));

    return 0;
}
```

処理の内容自体は全く同じである。しかし各桁の和を求める部分を sumdigit という関数にまとめることによって、main が非常にすっきり見える。

[printf で出力するだけの関数]

今度は printf で何か値を出力する関数を作ってみよう。引数として double の値 a, b を取り、a / b の値を printf で出力する関数は以下のように書ける。

```
void showdiv(double a, double b) {
    printf("a / b = %f", a / b);
    return;
}
```

printf で表示をするだけなので、何か値を返すわけではない。よって、返り値はないので関数の型は void となっている。void 型のとき、return 文には値を何も書かない。ま

た return は省略可能なので、次のように書いてもよい。

```
void showdiv(double a, double b) {
    printf("a / b = %f", a / b);
}
```

[関数の引数に関数を指定する例]

ある引数 num に対し、その数に 1 を加えた数を返す関数 suc は次のように書ける。

```
int suc(int num) {
    return num + 1;
}
```

これを使って、次のようなプログラムを作ってみる。

```
#include <stdio.h>

int suc(int num) {
    return num + 1;
}

int main(void) {
    int four;
    four = suc(suc(suc(suc(0))));
    printf("%d¥n", four);
    return 0;
}
```

実行してみると、printf によって 4 が出力される。ここで、

```
four = suc(suc(suc(suc(0))))
```

の部分を見てみよう。

まず右辺の一番内側の suc(0)関数が呼び出され、1 が返ってくる。よって、

```
four = suc(suc(suc(1)))
```

に置き換えられる。次に suc(1)が呼び出され、2 が返ってくる。よって、

```
four = suc(suc(2))
```

に置き換えられる。次に suc(2)が呼び出され、3 が返ってくる。よって、

```
four = suc(3)
```

に置き換えられる。最後に suc(3)が呼び出され、4 が返ってくる。よって、

```
four = 4
```

となり、four には 4 が代入される。これが printf によって出力される。

実は今まで使ってきた printf、scanf や、少しだけ紹介した abs、round、ceil、floor、strlen なども関数である。ではどこでその関数が適宜されているのかというと、stdio.h や math.h や stirng.h の中である。普段書いている#include という命令によってそれらの関数を読み込ませている。

今まで何気なく書いてきた int main(void)と return 0;という文が、関数に似ていることに気づいただろう。実は int main(void)も関数なのである。C 言語では、「プログラムの始まりは必ず main 関数から」という決まりになっている。
main 関数はその定義から、「int 型の値を返し、引数のない関数」だと分かる。実はある特定の引数を指定することも可能なのだが、ここでは割愛する。興味があれば「コマンドライン引数」について調べてほしい。
return 0 によって返された値 0 はどこに返されるのだろうか。それは main 関数の呼び出し元、つまりプログラムを呼び出したところである。0 という値は「終了コード」という形で OS などに返される。