C言語の基礎2

制御構造

if / if-else

```
if ( expression ) statement_true
```

```
if ( expression ) statement_true else statement_false
```

- `statement` が、1文以上の場合、ブロック `{}` で記述することができる。

【if_1.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int x = 10;

    if (x > 5)
    printf("xは5より大きいです\n");
    printf("if文の後に実行されます\n");

    return 0;
}
```

- この書き方はお勧めしない

【if_2.c】

- 同じ動作をするコード

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int x = 10;

    if (x > 5) {
        printf("xは5より大きいです\n");
    }
    printf("if文の後に実行されます\n");

    return 0;
}
```

【if_else_1.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
   int x = 3;

   if(x > 5) {
      printf("xは5より大きいです\n");
   } else {
      printf("xは5以下です\n");
   }

   printf("if-else文の後に実行されます\n");

   return 0;
}
```

比較演算子

比較演算子は、条件を判定し、その条件が論理的に**真**の場合は **1** を、**偽**の場合は **0** を返す、二項演算子です。

演算子	演算子の名前	例	説明
==	等しい	a == b	**a** が **b** と等しい
!=	等しくない	a != b	**a** が **b** と等しくない
<	より小さい	a < b	**a** が **b** より小さい
>	より大きい	a > b	**a** が **b** より大きい
<=	より小さいまたは等しい	a <= b	**a** が **b** より小さいまたは等しい
>=	より大きいまたは等しい	a >= b	**a** が **b** より大きいまたは等しい

- C言語では、Boolean型がないため、0を `False` 、0以外を `True` として扱う。
    - for / while もBooleanは同じ扱い

【if_else_2.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int x = 3;

    if (x) {  // ← 正しい条件式 省略しないで書くと x != 0
        printf("xは0以外です\n");
    } else {
        printf("xは0です\n");
    }

    printf("if-else文の後に実行されます\n");

    return 0;
}
```

for

【for_1.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int i;

    for (i = 1; i <= 5; i++) {
        printf("%d回目のループです\n", i);
    }

    printf("for文の後に実行されます\n");

    return 0;
}
```

- 現在は、以下のように書いても良い
  【for_1_c99.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {

    for (int i = 1; i <= 5; i++) {
        printf("%d回目のループです\n", i);
    }

    printf("for文の後に実行されます\n");

    return 0;
}
```

> C言語でfor文で変数の宣言と初期化を同時に行う機能は、C99規格から導入されました。
>
> C99で導入された「ブロックを中括弧で囲まなくても変数の宣言が可能」という機能とともに導入されました。この機能を「ブロックスコープ宣言」と呼びます。
>
> このため、C99以降のコンパイラであれば、for文で変数宣言を行うことができます。ただし、C89規格以前のコンパイラでは、この機能はサポートされていない場合があります。したがって、古いコンパイラを使用している場合は、for文の外で変数を宣言してからfor文で初期化する必要があります。

- スコープに注意
  - `for (int i = 1; i <= 5; i++)` の `i` は、for内でしか使用できない。
    【for_1_c99_outscope.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {

    for (int i = 1; i <= 5; i++) {
        printf("%d回目のループです\n", i);
    }

    printf("i=%d\n", i); // ここでエラーが発生します
    printf("for文の後に実行されます\n");

    return 0;
}
```

無限ループ

- forで永久にループする場合を記述する場合は以下のようにする。
    - ループを抜ける経路を必ず用意すること。
    - 終わらない場合は、`ctrl` + `c` で中断する。

【for_2.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int i = 0;

    for (;;) {
        if (i++ >= 5) {
            break;
        }
        printf("%d回目のループです\n", i);
    }

    printf("for文の後に実行されます\n");

    return 0;
}
```

【for_2.c】終わらない例

- コメント/アンコメントは、範囲選択後、`ctrl` + `/`
    - 範囲選択していない場合は、カーソルの存在する行に適用

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

```c
int main(int argc, char const *argv[]) {
    int i = 0;

    for (;;) {
        // if (i++ >= 5) {
        //     break;
        // }
        printf("%d回目のループです\n", ++i);  // ← 変更
    }

    printf("for文の後に実行されます\n");

    return 0;
}
```

while

- 条件が真の間、ループを繰り返す。

【while_1.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int i = 1;

    while (i <= 5) {
        printf("%d回目のループです\n", i);
        i++;
    }

    printf("while文の後に実行されます\n");

    return 0;
}
```

無限ループ

- forと同じく、whileでも無限ループを実行できる
    - 以下の例は終了条件が無いので、`ctrl`+`c`で中断すること

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int i = 1;

    while (1) {
        printf("%d回目のループです\n", i);
        i++;
    }

    printf("while文の後に実行されます\n");

    return 0;
}
```

- C言語では、0以外が真（true）になることを利用する

```
while(1) {
```

- 一般的には、上記のような書き方が多い。

- bool型はC言語には存在しないので、true/falseは標準では使用できない。
    - C99以降は、`stdbool.h`を利用することで、bool型のような処理を書くことが可能

【bool_now.c】

```c
#include <stdbool.h>  // C99以降で、bool型を使う場合に必要
#include <stdio.h>

int main(int argc, char const *argv[]) {
    bool a = true;   // bool型を使う場合
    int  b = false;  // int型を使う場合

    if (a) {
        printf("a is true\n");
    }

    if (b) {
        printf("b is true\n");
    }

    return 0;
}
```

【bool_old.c】

- 一般的には以下のような書き方も多い。

```c
#include <stdio.h>

#define FALSE 0
#define TRUE  !0

int main(int argc, char const *argv[]) {
    int a = TRUE;
    int b = FALSE;

    if (a) {
        printf("a is true\n");
    }

    if (b) {
        printf("b is true\n");
    }

    return 0;
```

```
}
```

- `#define` はC言語でのプリプロッセッサへの指示のひとつ。
  プリプロッセッサとは、コンパイルの前に処理を行う（前処理）プログラムのこと。

scanf

- C言語で標準入力から書式に従ってデータを読み込み、指定された変数に格納する関数
- 標準入力は通常キーボードから読み取る

数字（整数）の入力

```
scanf("%d",&num); // intの場合
scanf("%ld",&num);  // longの場合
```

数字（浮動小数点数）の入力

```
scanf("%f",&num); // floatの場合
scanf("%lf",&num);  // doubleの場合
```

1文字の入力

```
scanf("%c",&ch);  // 通常はこれ
scanf(" %c",&ch); // 先に入力がある場合
```

> scanfには、挙動が難しいものがあり要注意。
> 実際には、別の関数を使用するのが一般的。
>
> https://ja.wikipedia.org/wiki/Scanf#scanf.E3.81.AE.E5.95.8F.E9.A1.8C.E7.82.B9.E3.81.A8.E5.9B.9E.E9.81.BF.E6.96.B9.E6.B3.95

バッファオーバーフロー（バッファオーバーラン）

- 文字列の入力では、scanf関数が指定したバッファサイズよりも長い文字列を読み込むと、バッファオーバーフローが発生する可能性がある
- （Windowsにおいては）scanf_sなどの安全なバージョンを使用することを推奨される
- gccには、scanf_sは存在しないので、`fgets()` を使用するのが一般的

【scanf_2.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    char str[10];

    printf("文字列を入力してください：");

    scanf("%s", str);

    printf("入力された文字列は：%s\n", str);
    return 0;
}
```

【scanf_3.c】

```c
// scanfをfgetsに置き換えた例
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[])
{
    char str[10];

    printf("文字列を入力してください：");

    // fgetsで安全に入力を受け取る
    if (fgets(str, sizeof(str), stdin) != NULL) {
        // 入力文字列に改行が含まれていれば削除する
        str[strcspn(str, "\n")] = '\0';
    }

    printf("入力された文字列は：%s\n", str);
    return 0;
}
```

scanfの戻り値

- scanfの戻り値について調べてください。
    - 簡単なサンプルを実行して確認してください。

switch

- if-elseをいくつも使用する場合は、switchで記述する場合がある
    - caseに条件式は記述できない

【switch_1.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int num;

    printf("数字を入力してください：");
    scanf("%d", &num);

    switch (num) {
    case 1:
        printf("1が入力されました。\n");
        break;
    case 2:
        printf("2が入力されました。\n");
        break;
    case 3:
        printf("3が入力されました。\n");
        break;
    default:
        printf("1~3以外の数字が入力されました。\n");
        break;
    }

    return 0;
}
```

- switchを使用する場合、case内では基本的に `break;` が必要

【switch_2.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int num;

    printf("数字を入力してください：");
    scanf("%d", &num);

    switch (num) {
    case 1:
        printf("1が入力されました。\n");
        // break;
    case 2:
        printf("2が入力されました。\n");
        // break;
    case 3:
        printf("3が入力されました。\n");
        // break;
    default:
        printf("1~3以外の数字が入力されました。\n");
        // break;
    }

    return 0;
}
```

- どのような動作になるかを確認し、理解してください。
- あえてbreakを使用しないことで、OR条件にすることができる。

【switch_3.c】

```c
#include <stdio.h>

int main(int argc, char const *argv[]) {
    char ch;

    printf("1文字入力してください：");
    scanf("%c", &ch);

    switch (ch) {
    case 'a':
    case 'A':
        printf("aかAが入力されました。\n");
        break;
    case 'b':
    case 'B':
    case 'c':
    case 'C':
        printf("b,B,c,Cのどれかが入力されました。\n");
        break;
    default:
        printf("a,b以外の文字が入力されました。\n");
        break;
    }

    return 0;
}
```

- 上記のswitch_3.cをif-elseを使用して書き直してください。(switch_3_ifelse.c)

問題

以下のプログラムを作成してください

1. 1から10までの偶数を表示するプログラムを作成してください。（prog1.c）
2. ユーザーが入力した数値が素数かどうかを判定するプログラムを作成してください。(prog2.c)
3. ユーザーが入力した数値の階乗を計算するプログラムを作成してください。(prog3.c)
4. ループにより、ユーザが数字を入力し、0を入力したら入力した数字の合計を出力するプログラムを作成してください。（prog4.c）