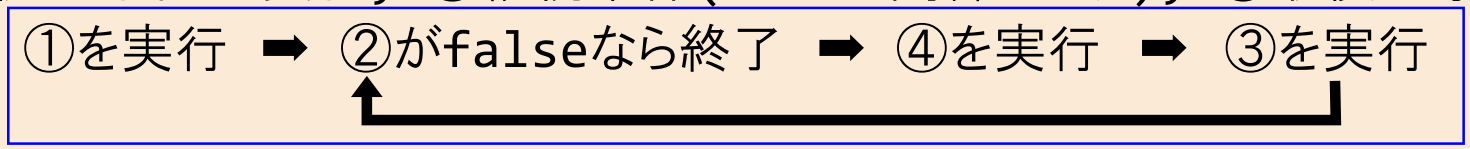
画像処理
(4J)

第02回
～C言語の復習（2）～

前回のまとめ

- 標準ライブラリヘッダの取り込みと、main()関数の書き方
  - #include <stdio.h>
  - int main(void){ }
- 変数宣言など、基本的なCプログラムの書き方
  - int a=10;　　"文字列リテラル"　　基本的な演算子　　大文字と小文字の区別等
- printf()関数の基本的な使い方
  - 書式指定文字 %d, %s
- for文の使い方
  - for (①最初に1回だけ実行; ②継続条件(trueの間繰り返す); ③最後に毎回実行) { ④ }



- 変数のスコープ
  - ローカル変数(局所変数) ⇔ グローバル変数(大域変数)

今回の内容

- 変数のスコープの話(続)
- 変数の宣言
  - 型、定数
  - 初期化
  - 暗黙的な型変換、明示的な型変換(キャスト)
- 条件式
  - 真(true)と偽(false)
  - if文
  - switch文
  - 三項演算子
- sizeof演算子

変数のスコープの話(続)

変数の有効範囲 ”スコープ”の話

- スコープが違うと、変数にアクセスできない
- スコープが違う場所なら、同名の"別の変数"も定義可能

```c
// Example: A-1
#include <stdio.h>

int main(void){
    int a = 999;
    printf("main: a = %d¥n", a);

    for (a = 0; a < 5; a++){
        printf("for内: a = %d¥n", a);
    }

    printf("main: a = %d¥n", a);
    return 0;
}
```

```c
// Example: A-2
#include <stdio.h>

int main(void){
    int a = 999;
    printf("main: a = %d¥n", a);

    for (int a = 0; a < 5; a++){
        printf("for内: a = %d¥n", a);
    }

    printf("main: a = %d¥n", a);
    return 0;
}
```

変数の有効範囲 ”スコープ”の話

- スコープが違うと、変数にアクセスできない
- スコープが違う場所なら、同名の"別の変数"も定義可能

```c
// Example: B-1
#include <stdio.h>

void sub(void) {
    int a = 10;  // ローカル変数
    printf("sub: a=%d¥n", a);
    a = 999;
}

int main(void){
    int a = 10;  // ローカル変数
    sub();
    printf("main: a=%d¥n", a);
    return 0;
}
```

```c
// Example: B-2
#include <stdio.h>

int a = 10;  // グローバル変数

void sub(void) {
    printf("sub: a=%d¥n", a);
    a = 999;
}

int main(void){
    sub();
    printf("main: a=%d¥n", a);
    return 0;
}
```

変数の有効範囲 ”スコープ”の話

●{}で囲うことで、スコープを限定することもできる
(読みにくくなるのでおすすめしないが、前述のようにfor文などではよく使われている)

```
// Case: C-1
#include <stdio.h>

int main(void){
    int a = 999;
    printf("1: %d¥n", a);
    {
        int a = 333;
        printf("2: %d¥n", a);
        {
            int a = 111;
            printf("3: %d¥n", a);
            {
                int a = 0;
                printf("4: %d¥n", a);
            }
            printf("5: %d¥n", a);
        }
        printf("6: %d¥n", a);
    }
    printf("7: %d¥n", a);

    return 0;
}
```

```
// Case: C-2 (※コンパイルエラーになる)
#include <stdio.h>

int main(void){
    int a = 999;
    printf("1: %d¥n", a);

    int a = 333;
    printf("2: %d¥n", a);

    return 0;
}
```

```
Main.c:8:6: error: redefinition of 'a'
        int a = 333;
            ^
Main.c:5:6: note: previous definition is here
        int a = 999;
            ^
1 error generated.
```

同一の
スコープ内で、
同名の変数を
宣言することは
できない。
<コンパイル
エラーになる>

(区別がつかない
ので当然)

演習A

① Example: A-1, A-2 を実際に実行して、動作を確認しましょう

② Example: B-1, B-2 を実際に実行して、動作を確認しましょう

③ Example: C-1, C-2 を実際に実行して、動作を確認しましょう

変数の宣言

変数の宣言

●変数宣言：　[記憶クラス] [型修飾子] データ型　変数並び;

- [記憶クラス]　{省略時 = auto, static, extern, register}
- [型修飾子]　{const, volatile}　(※必要なときのみ)
- データ型　{省略時 = signed, unsigned} {char, int, float, double}
  さらに int と double のみ、{short, long} を指定可能
- 例:

```
int a;
unsigned int b;
unsigned long int c;
```

```
const int x;
static int y;
extern int z;
```

●配列

- 例: int a[10];　　･･･ ※「10個分」の宣言なので、添字は0～9。「10」は使えない。
- C言語では、宣言した範囲外へのアクセスもできてしまうので、要注意!!
  int a[10]; と宣言して、 a[15] = 100; などが実行できてしまい、これは即、
  メモリ内容の破壊(例えば他の変数内容が書き換わる)に直結する。

データ型

- 整数型　　char, int
- 浮動小数点型　float, double
- 型なし　　void

基本型はこれしか無い！（文字も数値！）

※struct, union, enum（構造体･共用体･列挙体）および 配列、ポインタ型、などもデータ型（派生型）だが、ここに示した基本型と性格が異なる。

- 定数
  - 10進数　･･･ 0以外 で始まる必要あり。{0,1,2,3,4,5,6,7,8,9}で構成
  - 8進数　･･･ 0 で始まる。{0,1,2,3,4,5,6,7}で構成。
  - 16進数　･･･ 0x で始まる。{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}で構成。
  - 文字定数　･･･ 'A' のように「'」で囲む。 ➡ 文字コード（数値） を意味することになる。
  - 小数　･･･ float型を明示する場合、「F」で終わる。（何も付けなければ double型）
    例: 12.34567　　12.3456F

※同様に、整数の最後に「U」を付けると、unsigned型として扱われる。

数値の扱い


```c
// Example: D-1
#include <stdio.h>

int main(void){
    int a = 43;       // 10進数で 43
    int b = 043;      // 10進数で 35
    int c = 0x43;     // 10進数で 67
    int d = 'T';      // 10進数で 84 ('T'の文字コード)

    printf("%%d: a = %d, b = %d, c = %d, d = %d¥n", a, b, c, d);
    printf("%%o: a = %o, b = %o, c = %o, d = %o¥n", a, b, c, d);
    printf("%%X: a = %X, b = %X, c = %X, d = %X¥n", a, b, c, d);
    printf("%%c: a = %c, b = %c, c = %c, d = %c¥n", a, b, c, d);

    return 0;
}
```

printf()の書式指定

%d	:	10進数として表示
%o	:	8進数として表示
%x, %X	:	16進数として表示
%c	:	文字(1文字)として表示
%s	:	文字列を表示

※「%」を表示したいときは、「%%」と指定

ASCIIコード(文字コード)表

Wikipediaより引用:
https://ja.wikipedia.org/wiki/ASCII

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
2-	SP 0x20 32 040	! 0x21 33 041	" 0x22 34 042	# 0x23 35 043	$ 0x24 36 044	% 0x25 37 045	& 0x26 38 046	' 0x27 39 047	(0x28 40 050	) 0x29 41 051	* 0x2A 42 052	+ 0x2B 43 053	, 0x2C 44 054	- 0x2D 45 055	. 0x2E 46 056	/ 0x2F 47 057
3-	0 0x30 48 060	1 0x31 49 061	2 0x32 50 062	3 0x33 51 063	4 0x34 52 064	5 0x35 53 065	6 0x36 54 066	7 0x37 55 067	8 0x38 56 070	9 0x39 57 071	: 0x3A 58 072	; 0x3B 59 073	< 0x3C 60 074	= 0x3D 61 075	> 0x3E 62 076	? 0x3F 63 077
4-	@ 0x40 64 100	A 0x41 65 101	B 0x42 66 102	C 0x43 67 103	D 0x44 68 104	E 0x45 69 105	F 0x46 70 106	G 0x47 71 107	H 0x48 72 110	I 0x49 73 111	J 0x4A 74 112	K 0x4B 75 113	L 0x4C 76 114	M 0x4D 77 115	N 0x4E 78 116	O 0x4F 79 117
5-	P 0x50 80 120	Q 0x51 81 121	R 0x52 82 122	S 0x53 83 123	T 0x54 84 124	U 0x55 85 125	V 0x56 86 126	W 0x57 87 127	X 0x58 88 130	Y 0x59 89 131	Z 0x5A 90 132	[0x5B 91 133	¥ 0x5C 92 134	] 0x5D 93 135	^ 0x5E 94 136	_ 0x5F 95 137
6-	` 0x60 96 140	a 0x61 97 141	b 0x62 98 142	c 0x63 99 143	d 0x64 100 144	e 0x65 101 145	f 0x66 102 146	g 0x67 103 147	h 0x68 104 150	i 0x69 105 151	j 0x6A 106 152	k 0x6B 107 153	l 0x6C 108 154	m 0x6D 109 155	n 0x6E 110 156	o 0x6F 111 157
7-	p 0x70 112 160	q 0x71 113 161	r 0x72 114 162	s 0x73 115 163	t 0x74 116 164	u 0x75 117 165	v 0x76 118 166	w 0x77 119 167	x 0x78 120 170	y 0x79 121 171	z 0x7A 122 172	{ 0x7B 123 173	| 0x7C 124 174	} 0x7D 125 175	~ 0x7E 126 176	
	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F

文字 = ただの整数(文字コード)

●整数を「文字として表示」することで、文字が表示される。

- 例: 'A' == 65 で 'a' == 97　　※「'A'」と「65」は全く同じに取り扱われる
  ➡ 97-65 = 32　➡ 大文字に +32 すれば小文字に変換できる
- 以下のような計算で文字の表示もできる

```
例1:  printf("%c", 'X' + 32);
例2:  printf("%c", 'Y' + 'a' - 'A');
例3:  for (char i='A'; i <= 'Z'; i++) printf("%c", i);
```

●文字列 = 整数の配列

・・・ヌル文字'¥0'(=整数値「0」)までを一連の文字列として扱う。

- '¥0' == 0 だが、'¥0'とした方が、文字列を扱っていることを明確にできる。
- (配列については、ポインタと絡めて、改めて詳しく取り扱う予定です。)

配列の初期化

●宣言時のみ、{}で列挙して初期化できる。

- `int a[5] = {1,2,3,4,5};`
  - ➡ a[0]==1, a[1]==2, … , a[4]==5 となる。
- `int a[5] = {1,2}; //` 宣言した配列の要素数より、右辺の値の数が少ない場合
  - ➡ a[0]==1, a[1]==2, a[2]==0, a[3]==0, a[4]==0 となる。
    (残りの足りない分は 0 で初期化される)
- `int a[5];`
  - ➡ 宣言のみの場合、値は不定となる。(ただし static の場合 0 に初期化される)
    裏技的?だが、0で初期化したい場合は、 `int a[5] = {0};` などとする。

※次ページで説明

- `int a[] = {1,2,3,4,5};` (この例だと `int a[5]={1,2,3,4,5};` と全く同じ)
  - ➡ 右辺の要素数に合わせて宣言される。

●文字列リテラルでの初期化

- `char str[] = "ABC";` ➡ `char str[4] = {'A','B','C','¥0'};` と同じ
  ※文字列の終わりを示す、ヌル文字('¥0')を含めた 4 要素分を宣言したことになる。

演習B

① Example: D-1 を実際に実行して、動作を確認しましょう。

② for文を用いて、a~zの英文字を出力

```
abcdefghijklmnopqrstuvwxyz
```

③ 文字列をforループ と printf("%c", …) で表示するプログラムを作成👇 ※書式指定子 %c のみ使用

```
#include <stdio.h>

int main(void){
    char str[] = "Ichinoseki KOSEN";   // 例えばの文字列。なんでもいい
    // ここにコードを追加。※文字列の長さは事前には分からず、ヌル文字'¥0'の手前までを出力するものとする

    return 0;
}
```

記憶クラス static (静的変数)


```
// Example: D-2
#include <stdio.h>

void func(void) {        // staticを付けると、関数終了後も保持される
    static int n = 10; // 初期化は1回目の呼び出しのときだけ
    int m = 10;          // auto変数は、関数が終わる度に消える

    printf("n = %2d¥tm = %2d¥n", n, m);
    n = n+1;
    m = m+1;
    return;
}

int main(void){
    func();
    func();
    func();
    func();
    return 0;
}
```

static を付けた場合は宣言だけであっても、暗黙的に「0」に初期化される。

```
// Example: D-3
// 初期化なしの宣言のみに変更すると･･･？
void func(void) {
    static int n;
    int m;
    ... (他は Example: A-2 と同じ)
}

int main(void){
    ...
}
```

記憶クラス extern　…※また改めて説明予定

●"extern" は、「どこか別のところで宣言されてるので、それを使う」というマーク。

➢「どこか」は後で解決される(はず)ので、今は気にせずコンパイルする。

main.c

```
// Example: D-4(1)
// main.c
#include <stdio.h>

int exVal = 100;  // グローバル変数

int main(void){
    func();
    exVal = 200;
    func();
    return 0;
}
```

sub.c

```
// Example: D-4(2)
// sub.c
// 関数のみを定義

extern int exVal;

int func(void){
    printf("exVal = %d¥n", exVal);
    return 0;
}
```

実行結果➡

```
exVal = 100
exVal = 200
```

2つのファイルを
それぞれコンパイル
(この時点では、
sub.c内の
exValは
"どこかにあるはず"
の扱いの状態)
↓
リンクして
実行ファイルを生成
(リンクした際に、
sub.c内の
exValの参照が
解決される)

型修飾子　const

●「値の変更ができない」変数であることを示す、型修飾子

- ➢初期化で値を設定　(コンパイル時に値が決定)
- ➢値の変更をするコードがあるとコンパイルエラーになる

```
// Example: E-1 (※コンパイル不可)
#include <stdio.h>
int main(void){
    const int c = 100;

    c = 99;
    printf("%d¥n", c);
}
```

```
Main.c:6:7: error: cannot assign to variable 'c' with const-qualified type 'const int'
    c = 99;
    ~ ^
Main.c:4:15: note: variable 'c' declared const here
    const int c = 100;
    ~~~~~~~~~~^~~~~~~
1 error generated.
```

```
// Example: E-2
#include <stdio.h>
const int c = 99;

int main(void){
    printf("%d¥n", c);
}
```

```
// Example: E-3
#include <stdio.h>
#define c 99

int main(void){
    printf("%d¥n", c);
}
```

#define を似たような目的(定数)で用いる場合があるが、#defineはプリプロセッサによる単純な文字列置換である。const を用いた場合、組み込み用途などでは、専用の読み出し専用メモリ(ROMやSRAM)に配置される場合があるなど、動作が異なる。

暗黙の型変換

- 代入時、右辺と左辺の型が異なっても、暗黙的に型が変換されて代入される
- 計算時には、演算の優先度順に、
  二項演算単位で、表現力の高い型に合わせて暗黙的に型変換される。
  - 同じ型同士での二項演算では、結果も同じ型になる。
    - 例1:「10/3」→ int型同士なので、int型の「3」となる
    - 例2:「10/3.0」→ 表現力の高いdouble型の「3.0」に合わせ、double型の「3.333…」となる
    - 例3:「10/3+1.5」→ まず「10/3」が計算され、「3+1.5」→ double型の「4.500…」となる
    - 例4:「10/3.0+1.5」→「10/3.0」が計算され、「3.333…+1.5」→ double型の「4.833…」となる
  - `int a=10; int b=3;`
    `double c=3.0; double d=1.5;` としたとき、…
    - 例1:「a/b」→ int型同士なので、int型の「3」となる
    - 例2:「a/c」→ 表現力の高いdouble型の c に合わせ、double型の「3.333…」となる
    - 例3:「a/b+d」→ まず「a/b」が計算され、「3+d」→ double型の「4.500…」となる
    - 例4:「a/c+d」→「a/c」が計算され、「3.333…+d」→ double型の「4.833…」となる

明示的な型変換(キャスト)

●変数名や数値、関数名の前に( )で括った型名を書くことで、
明示的に型変換を行うことができる。

➢ int a=10; int b=3;
double c=3.0; double d=1.5; としたとき、

- 例1:「a/(double)b」→ int型とdouble型の演算となり、double型の「3.333…」となる
- 例2:「a/(int)c」→ int型同士の演算となり、int型の「3」となる
- 例3:「a/(double)b+d」→ double型で計算され、「3.333…+d」→ double型の「4.833…」となる
- 例4:「a/(int)c+d」→ int型で計算され、「3+d」→ double型の「4.500…」となる

➢ int a=10; int b=3;
double c=3.0; double d=1.5; としたとき、…

- 例1:「a/b」→ int型同士なので、int型の「3」となる
- 例2:「a/c」→ 表現力の高いdouble型の c に合わせ、double型の「3.333…」となる
- 例3:「a/b+d」→ まず「a/b」が計算され、「3+d」→ double型の「4.500…」となる
- 例4:「a/c+d」→「a/c」が計算され、「3.333…+d」→ double型の「4.833…」となる

←比較用:
全ページの、
キャストなしの
場合と結果

演習C

前ページの「キャストあり」と「無し」の違いを、
実際に確認しなさい

- ヒント:
  printf()で整数型の値を表示する場合、 %d を使う
  printf()で浮動小数点型の値を表示する場合、 %f を使う

条件式、三項演算子

真(true)と偽(false)

●C言語では、真偽値は整数値であり、

- 0 ･･･ 偽(false)
- 1(0以外) ･･･ 真(true)

と扱われる。

●比較演算の結果は、0(false) または 1(true) となる。

- (33 == 33) ･･･ trueなので、「1」　　(33 == 11) ･･･ falseなので、「0」
- (33 != 11) ･･･ trueなので、「1」　　(33 != 33) ･･･ falseなので、「0」
- (11 < 33) ･･･ trueなので、「1」　　(11 > 33) ･･･ falseなので、「0」
- (33 > 11) ･･･ trueなので、「1」　　(33 < 11) ･･･ falseなので、「0」

●論理演算子　　&&, ||, !　　･･･ AND, OR, NOT

- !a ･･･ 真偽値の反転(NOT)　　➡ !0 == 1,　!1 == 0,　!100 == 0
- (a > 100 || b > 0) とか、(a > 100 && b > 0) のように使うことで条件を組み合わせることができる

1文字だけの
& や | は、
ビット演算子という
別の意味になるので
要注意！

if文

基本的な形

```
if (a < b) {
    printf("(a < b)¥n");
    printf("True¥n");
}
else {
    printf("(a < b)¥n");
    printf("False¥n");
}
```

elseは省略可能

```
if (a < b) {
    printf("(a < b)¥n");
    printf("True¥n");
}
```

1行だけなら{}は省略可

```
if (a < b)
    printf("True¥n");
else
    printf("False¥n");
```

```
if (a < b)
    printf("True¥n");
```

elseにif文を続けることも可能

```
if (a == 1) {
    printf("a==1¥n");
}
else if (a == 2){
    printf("a==2¥n");
}
else if (a == 3) {
    printf("a==3¥n");
}
else {
    printf("other¥n");
}
```

※「else if」という文があるわけではない

ネストする場合はインデントで分かりやすく

```
if (a == 1) {
    printf("a==1¥n");
    if (b == 1) {
        printf("b==1¥n");
        if (c == 1) {
            printf("c==1¥n");
        }
        else {
            printf("c!=1¥n");
        }
    }
    else {
        printf("b!=1¥n");
    }
}
else {
    printf("a!=1¥n");
}
```

```c
// Example: F

#include <stdio.h>

int main(void){
    printf("True  = %d¥n", (10==10));
    printf("False = %d¥n¥n", (10<3));

    for (int i=-3; i<4; i++) {
        if (i)
            printf("%d: True¥n", i);
        else
            printf("%d: False¥n", i);
    }
    return 0;
}
```

```
True  = 1
False = 0

-3: True
-2: True
-1: True
0: False
1: True
2: True
3: True
```

switch文

基本的な形

```
int i = 1;
// i = 2;
// i = 3;
// i = 10;
printf("i == %d¥n¥n", i);
switch (i) {
     case 1:
          printf("case 1!¥n");
          break;
     case 2:
          printf("case 2!¥n");
          // ココにはbreak無し
     case 3:
          printf("case 3!¥n");
          break;

     default:
          printf("other¥n");
}
printf("fin.¥n")
```

```
switch(整数) {
    case ?:
        ...
        break;
    ...
    default:
        ...
}
```

break; を忘れると、
そのまま次の行を
実行することになるので
注意が必要。

```
i == 1

case 1!
fin.
```

```
i == 2

case 2!
case 3!
fin.
```

```
i == 3

case 3!
fin.
```

```
i == 10

other
fin.
```

逆に言うと、こういう書き方もできる。

```
switch (i) {
  case 1:
  case 2:
  case 3:
    printf("1 or 2 or 3!¥n");
    break;

  default:
    printf("other¥n");
}
```

switch文

文字定数で、こんな書き方もできる。

```
char ch;
...
ch = 'A';
...

switch (ch) {
  case 'A':
    printf("AAA!¥n");
    break;
  case 'B':
    printf("BBBBB!¥n");
    break;
  case 'C':
    printf("CCCCCCCC!¥n");
    break;
}
```

default:
をスペルミスをしても
エラーにはならない
(単なるラベルとして扱われる)ので注意。
※switch文としては
default動作が無い
状態になる。

errorにはならないが、
未使用のラベル「defolt」
があるという、
warningは出してくれる(かも)。

defaultをスペルミスすると…?

```
int i = 10;

switch (i) {
  case 1:
  case 2:
  case 3:
    printf("1 or 2 or 3!¥n");
    break;

  defolt:
    printf("other¥n");
}
printf("fin.¥n");
```

ついでに、C言語におけるラベルについて:
ラベル名+「:」で、ラベルが定義でき、goto文でその場所に飛べます。
でも、可読性が低下するため、通常は使いません。というか、どうかgotoは使わないでください…。

三項演算子

(条件式)? trueの場合 : falseの場合

```
(①)? ② : ③;
```

⇦（ほぼ）同じ⇨

```
if (①)
   ②;
else
   ③;
```

```
int x = -10;
(x>0)? printf("%d", x) : printf("%d", -x);
```

⇦⇨

```
int x = -10;
if (x>0)
   printf("%d", x);
else
   printf("%d", -x);
```

●if文との違いは、値として扱えること･･･くらいかな？

```
int i;
...
printf("%s¥n", (i==10)? "True!" : "False!");
```

```
int i;
int x = -10;
i = (x>0)? x : -x;   // xの絶対値が代入されることになる。
```

sizeof演算子

sizeof 演算子

- sizeof演算子 ･･･ オブジェクトのサイズをバイト単位で返す
  - sizeof(char) ･･･ 型名を引数にすることもできる
  - int a[30];
    sizeof(a); ･･･ 配列全体のバイト数が得られる

- sizeofを使って、
  配列の要素数を調べることができる。
  - 配列の要素数を知りたいとき:
    sizeof(a)/sizeof(a[0])
    ･･･ 配列全体のバイト数を、
    要素一つのバイト数で除算する

```
// Example: G
#include <stdio.h>

int main(void){

    int a[100];
    printf("sizeof(a) = %d¥n", sizeof(a));
    printf("sizeof(a[0]) = %d¥n", sizeof(a[0]));
    printf("n = %d", sizeof(a)/sizeof(int));
    return 0;
}
```

まとめ

●変数のスコープの話(続)

●変数の宣言

- 型、定数　`static / const / unsigned / {short, long} / {char, int, float, double}`
  `65 == 0101 == 0x41 == 'A'` (※10進数, 8進数, 16進数, 文字定数)
- 初期化　`int a[5] = {1, 2, 3, 4, 5};` ･･･配列宣言時の初期化
- 暗黙的な型変換、明示的な型変換(キャスト)
  ･･･同じ型同士なら結果も同じ型に、異なる型なら表現力の高い型になる。
  (型名) を頭に付けると、明示的に型変換を指定できる。(=キャスト)

●条件式

- 真(true)と偽(false)　･･･0==偽(False)、1==真(True)　(※判断の際は、0以外は真 と扱われる)
- if文　･･･ `if (条件) {真の場合の処理} else {偽の場合の処理}`
- switch文　･･･ `break;` が無いとそのまま下の行に処理が継続することに注意
- 三項演算子　･･･「`(条件式)? trueの場合 : falseの場合;`」 式全体を値として使える

●sizeof演算子　･･･配列の要素数は `sizeof(a)/sizeof(a[0])` で得られる

課題No.02

提出課題としますので、授業中にできなかった分は宿題として、
（実行結果の例も添えて、）次週までに提出のこと。

課題No.02-A

① for文で整数変数を-10～+10まで1ずつ変化させ、
変数の値とともに　　負の偶数:Negative (Even)
負の奇数:Negative (Odd)
ゼロ:Zero (Even)
正の偶数:Positive (Even)
正の奇数:Positive (Odd)
を出力する。

② 2つの変数a,bに代入した任意の値の範囲
(a～b, a<b)について同様の処理が行えるようにする。

【※提出は②のみ】

```
-10: Negative (Even)
-9: Negative (Odd)
-8: Negative (Even)
-7: Negative (Odd)
-6: Negative (Even)
-5: Negative (Odd)
-4: Negative (Even)
-3: Negative (Odd)
-2: Negative (Even)
-1: Negative (Odd)
0: Zero (Even)
1: Positive (Odd)
2: Positive (Even)
3: Positive (Odd)
4: Positive (Even)
5: Positive (Odd)
6: Positive (Even)
7: Positive (Odd)
8: Positive (Even)
9: Positive (Odd)
10: Positive (Even)
```

課題No.02-B

●`int n[] = {1980, 1990, 2000, 2010, 2015, 2020, 2300};`
のように定義した、任意の西暦年の配列について、
それぞれが“うるう年かどうか”を判定して表示するプログラム

```
1980 : Leap-year
1990 :
2000 : Leap-year
2010 :
2015 :
2020 : Leap-year
2300 :
.
.
.
```

➢うるう年は、以下の条件で判別できる

1．西暦年が4で割り切れる年はうるう年
2．ただし、西暦年が100で割り切れる年は平年（うるう年ではない）
3．ただし、西暦年が400で割り切れる年はうるう年

課題No.02-C


```
char str[] = "abcDEFghi";
```

のように宣言した文字列(A～Z、a～zのみで構成)を、
数値演算のみで、

① すべて大文字に変換して表示

② すべて小文字に変換して表示

③ 大文字と小文字を入れ替えて表示

```
0: abcDEFghi
1: ABCDEFGHI
2: abcdefghi
3: ABCdefGHI
```

●ヒント:

- 文字列は `'¥0'` で終わる

課題No.02-D

- for文で整数値を1～12で変化させ、英語表記での月の短縮名を出力
  - Switch文を使うこと

```
 1: Jan.
 2: Feb.
 3: Mar.
 4: Apr.
 5: May
 6: June
 7: July
 8: Aug.
 9: Sept.
10: Oct.
11: Nov.
12: Dec.
```