# プログラミング2　第1回

# 本講義の目的

- 目的
  - C言語の修得を通じて、**計算機システムに関する理解を深める**
    - 特にメモリ操作
  - C言語の修得を通じて、**データ構造やアルゴリズムに関する理解を深める**
    - リスト、ソートなど

# 担当教員・TA

- 教員
  - 星野 哲也（情報基盤センター 准教授、情報システム学専攻 片桐・星野研）
    - 高性能計算に関する研究
    - 名大スパコン「不老」の調達や運用が主な業務
  - HUANG Wen-Chin ホワン ウェンチン（情報学研究科 助教、知能システム学専攻 戸田研）
    - 音声信号処理
    - 深層学習

- TA（Teaching Assistant）
  - 棗　恒輝（情報学研究科 修士1年）
  - 森田　光貴（〃）

# 講義計画（暫定）

- 第1回(11/28(金))　ガイダンス、Cプログラミングの基礎（データ型、変数、四則演算、制御構文）
- 第2回(12/05(金))　ビット演算、配列、関数
- 第3回(12/12(金))　配列と関数、文字列、ソート
- 第4回(12/19(金))　再帰関数、ポインタ(1)
- 第5回(12/26(金))　ポインタ(2)
- 第6回(01/ 9(金))　構造体・共用体、リスト
- 第7回(01/23(金))　Cプログラミング実用上必須のもの（defineマクロ、ファイル操作、複数ソースコンパイルなど）
- 第8回(01/30(金))　期末試験（**出席必須！**）

# 講義資料・教科書・参考書










- 講義資料
  - TACTにアップロード
- 教科書（一応）
  - ハーバード・シルト 著、トップスタジオ 訳、柏原正三 監修、独習C 第4版、翔泳社、2007年
  - 本講義資料は当初は※本書籍に沿って作られた
- 参考書（一応）
  - B.W. カーニハン & D.M. リッチー 著、石田晴久 訳、プログラミング言語C 第2版、共立出版、1989年
  - 片桐孝洋, 大島聡史 著、C&Fortran 演習で学ぶ数値計算、共立出版、2022年
    - 前任者の本
- **書籍の購入は必要ない**
  - Web上の資料で十分だと思う
  - 体系的に学びたい場合は本を買ってください

※ 前々任者が作成した資料を加筆・修正してきている

# 成績評価方法・基準

- 評価方法
  - 演習課題（ほぼ毎週、全６回）
  - 期末試験
- 評価基準
  - 演習課題：50点
    - 60点分課題を出し、50点を超えた場合は50点とする
  - 期末試験：50点
    - 60点分問題を出し、50点を超えた場合は50点とする
  - 合計100点満点で60点以上が合格
    - 演習60点でも期末5点なら合計55点で不合格なので注意
  - 成績(A+, A, B, C, C-）への変換は、標準的な換算方法**ではなく**、相対評価で行う
    - 一昨年度実績： A+: 100, A: 85-99, B: 70-84, C: 65-69, C-: 60-64
    - 昨年度実績：標準と同等

# 授業の進め方

- オンラインなし現地のみ
  - 出席点はつけないので、出席の義務はなし
  - **ただし期末試験は必ず出席してください**
- 3限座学、4限演習
  - 3限早めに終わらせて、演習長めにとります
- 演習課題
  - 毎回出します
  - 締め切りは基本2週間後

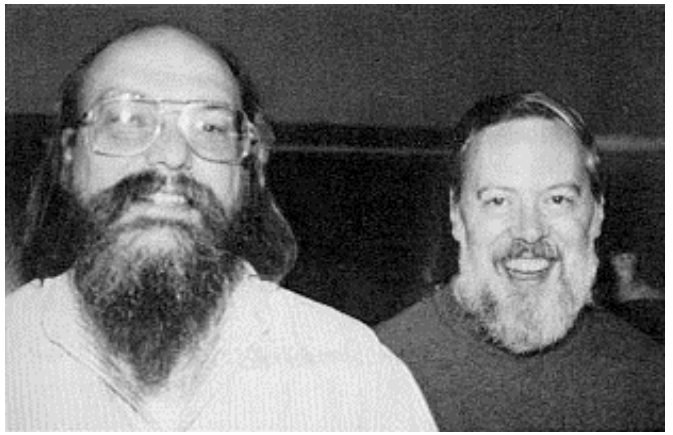
# C言語の歴史

- 1966年　Martin Richards（ケンブリッジ大学）がBasic Combined Programming Language(BCPL)を設計
  - このときmrは26歳くらい
- 1969年　Ken Thompson（AT&Tベル研究所）がBCPLを元にB言語を設計
  - このときkenは26歳くらい
- 1972年　Dennis M. Ritchie(AT&Tベル研究所)がB言語を発展させたC言語を設計
  - このときdmrは31歳くらい



https://www.cst.cam.ac.uk/people/mr10

**Martin Richards**



**Ken Thompson (Left)**

**Dennis M. Ritchie (Right)**

# 1970年代のコンピュータ

- IBM System/370[*1] (1972~)
  - メモリ240 KB
    - ↑ **今時の普通のPCの1万分の１以下！**
- Cray-1[*2] (1976-1982年まで世界最速のスーパーコンピュータ)
  - メモリ8.3 MB
    - ↑ **今時の普通のPCの1000分の1程度！**
  - 演算性能：160 MFlops (一秒間に1億6000万回の計算)
    - ↑ **今時のスマホの1000分の1以下！**

**コンピュータの性能が乏しい時代に**
**C言語は作られた**



**Cray 1：世界一高価な椅子(当時$500万)**

*1: https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3135.html
*2: https://www.hpe.com/jp/ja/compute/hpc/cray.html

# C言語の特徴

- 速い
  - C言語の唯一最大のメリット
  - メモリを直接的に効率良く操作できる
  - 速さや省リソースが求められる分野では未だ一線級
    - 高性能計算、OS、組み込みシステム、省エネ〇〇
- 使いづらい
  - 直感的じゃない
    - コンピュータ側の都合が言語に反映されているため
  - Pythonで提供されている便利機能はほとんどない
- コンパイラ形式
- 静的型付け
- 手続き型（関数型）

# C言語の規格

- K&R (B.W. Kernighan & D.M. Ritchie)
  - ANSI標準化以前のC言語
  - 現在ではK&R表記によるプログラミングは推奨されない
- C89/90 (ISO/IEC 9899: 1990) (ANSI C)
  - ANSI及びISOによる最初のC言語標準規格
- C99 (ISO/IEC 9899: 1999)
  - ブロック先頭以外でも変数宣言が可能
  - ブール代数_Bool型や複素数_Complex型を追加
  - “//”によるコメントアウト

最近のCコンパイラはC99の機能くらいまではオプション不要で利用可能

- C11 (ISO/IEC 9899: 2011)
  - アラインメント指定
  - 無名構造体と共用体
  - マルチスレッド機能
  - Unicodeサポート改善

C11以降の機能を使う場合はコンパイル時に追加オプションが必要なことが多い

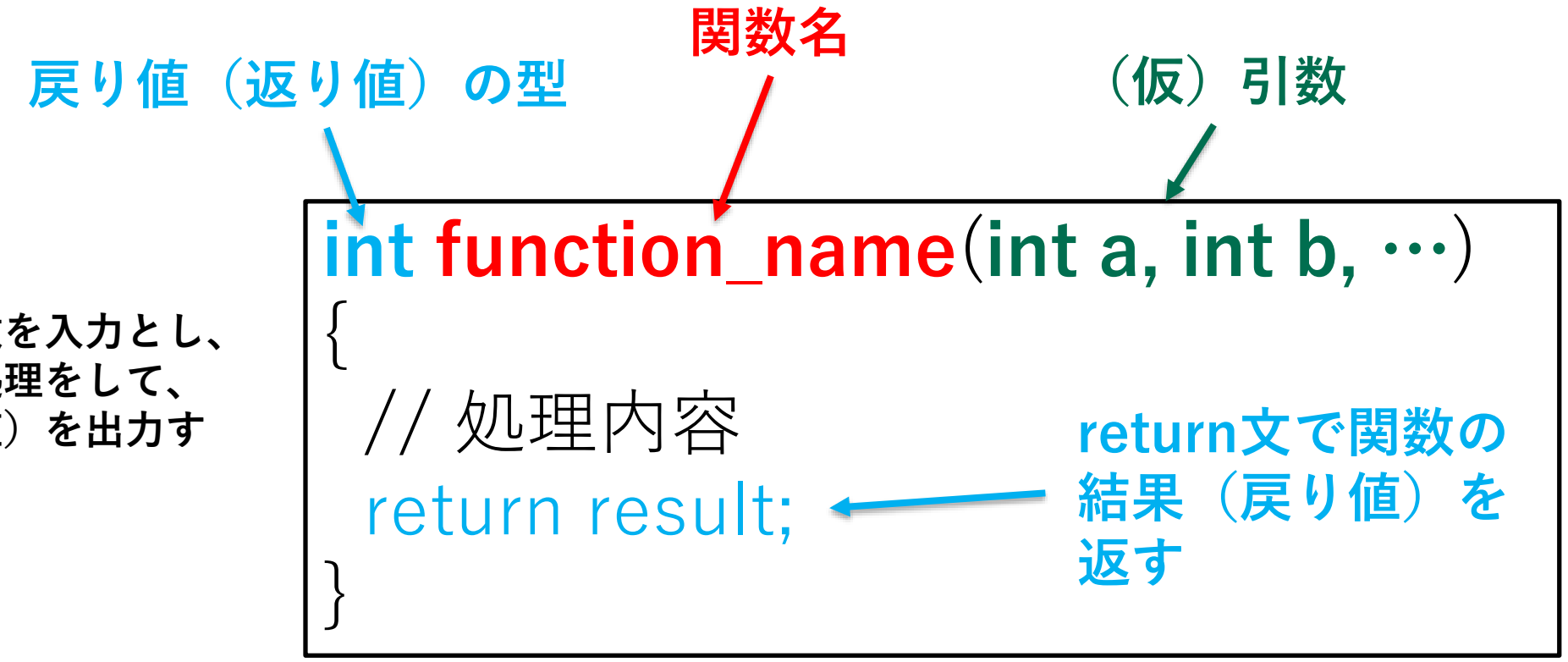
- C17 (ISO/IEC 9899:2018)
  - 通称C17またはC18
  - 欠陥修正がメインのマイナーアップデート

- 「教科書」はANSI Cに基づいている
- 本講義はANSI Cベースだが一部C99の機能を使用する

# C言語の構造

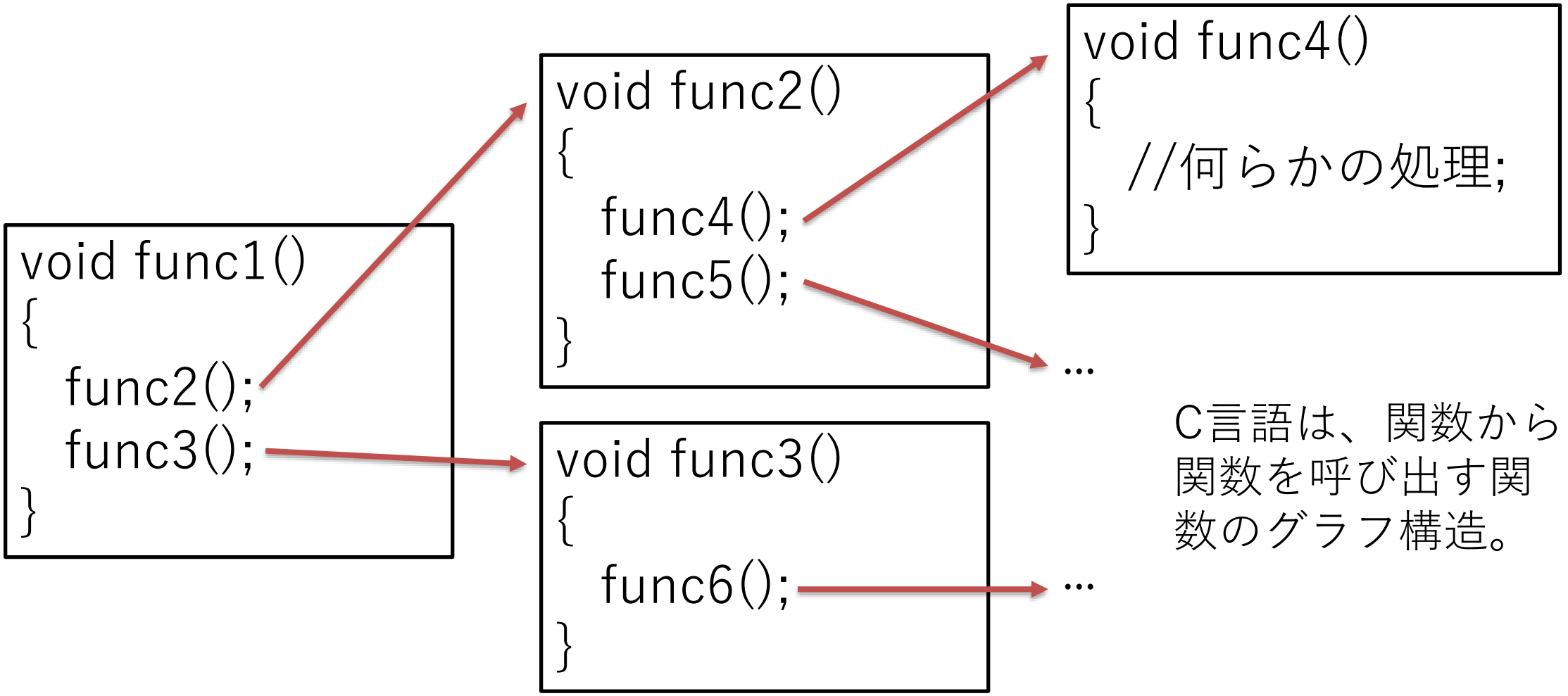
- C言語は手続き型（関数型）言語と呼ばれ、関数の集合でプログラムが構成される
- 関数とは

**関数は、引数を入力とし、なんらかの処理をして、結果（戻り値）を出力するもの。**



# C言語の構造

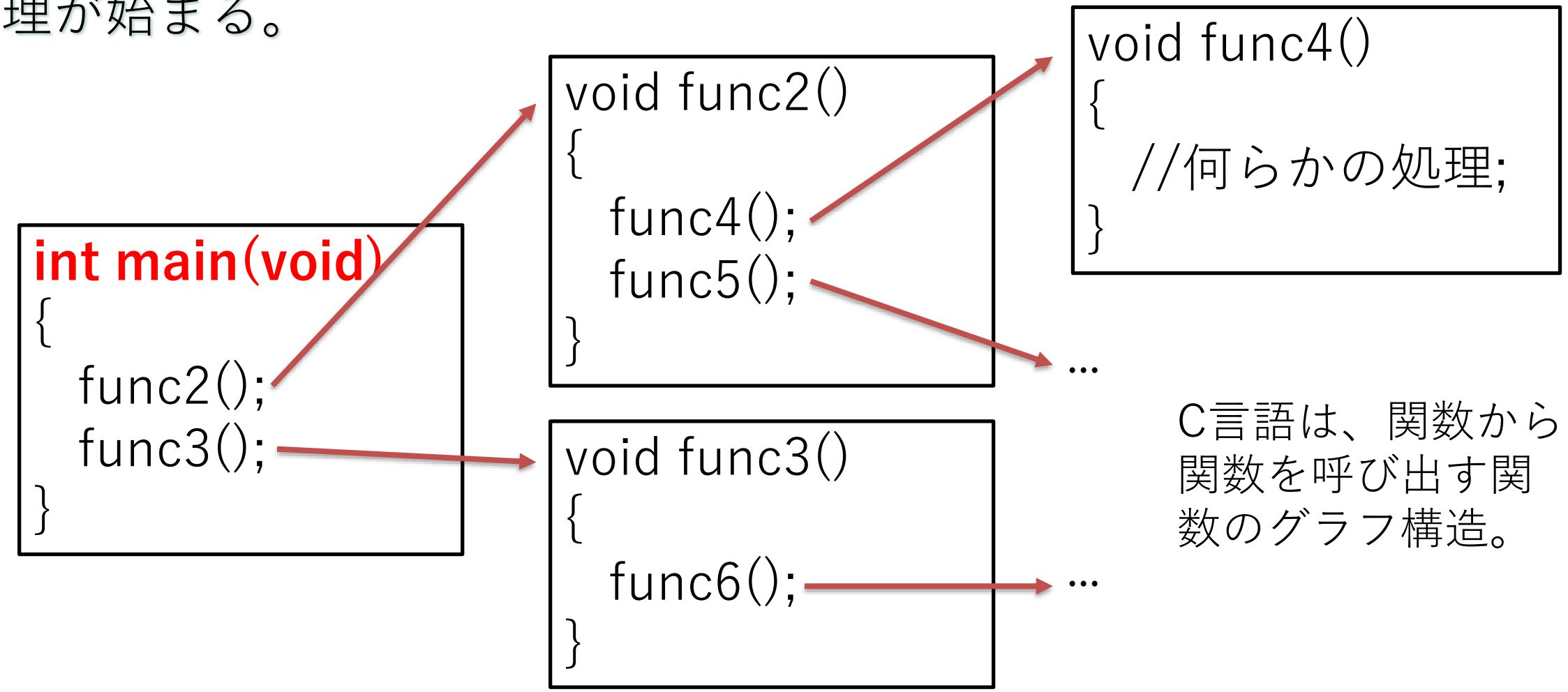
- C言語は手続き型言語と呼ばれ、関数の集合でプログラムが構成される




C言語は、関数から関数を呼び出す関数のグラフ構造。

# main関数

- 一番最初に呼ばれる関数をmain関数と呼ぶ。C言語はmain関数から処理が始まる。




C言語は、関数から関数を呼び出す関数のグラフ構造。

# main関数

- C言語ではmain関数が必須
  - ひとまずmain関数さえあれば、それ以外の関数呼び出しがなくともC言語として成立する

```
#include <stdio.h>

int main(void)
{
  printf("Hello, World!¥n");
  return 0;
}
```

**最も有名なC言語プログラム Hello, World!**

# C言語の書き方

main関数は必須

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!¥n");
    return 0;
}
```

セミコロンまでが
一つの文。

C言語ではスペース・タブ・改行はただの区切り文字。見やすいように自由に入れていい。

main関数の戻り値はプログラム自体の実行結果として使うことができる。

# Hello World！

ヘッダーファイル。stdio.hには、キーボードからの入力や画面出力など、標準入出力関係の関数の宣言などが書かれている。printf関数を使うのに必要。

```
#include <stdio.h>

int main(void)
{
  printf("Hello, World!¥n");
  return 0;
}
```

printf関数を使って、文字列を画面に出力する。

PowerPointの都合で円記号が表示されているが、実際には半角のバックスラッシュ（＼）

# Cプログラムのコンパイル

- テキストエディタでソースファイル(source file)を作成
  - ファイル名の拡張子は ".c"
- Cコンパイラでソースファイルを実行可能ファイル(executable file)に変換
  - Pythonなどインタプリタ形式言語のように逐次解釈しながらの実行はできない
  - ソースファイルを機械（CPU）が読める機械語（machine language）に翻訳（コンパイル）する必要がある
    - よって、ソースファイルを書き換えたらコンパイルし直す必要がある
  - 本講義資料ではCコンパイラとしてgccを用いる
    - 環境構築する際はgcc推奨だが、gcc以外（clangなど）でも可
- 実行ファイルを実行
  - 実行ファイル名がそのまま実行コマンドになる

# Cプログラムのコンパイルと実行

- コンパイル

コンパイルしたいソースファイル名

```
$ gcc -o hello hello.c
```

作成する実行ファイル名を
-o オプションの後に指定
（指定しないとa.outになる）

**この資料の表記方法**
・$の後がコマンド($は実際は入力しない)
・特にない場合が出力

```
$ コマンド
出力
```

- 実行

```
$ ./hello
Hello, World!
$ echo $?
0
```

./（ドットスラッシュ）をつけること
＊Windowsのコマンドプロンプト使用の場合は ./ は不要

- $? は直前に実行したコマンドの終了ステータスを意味する
- プログラム最後のreturnで指定した"0"が表示される

# 変数と型

- C言語で変数を使う際には、型の宣言が必要
  - よく使う型：char（文字）, int（整数）, double（倍精度浮動小数点数）など

```
int n;
double x, y, z;
```

名前がnのint(整数)型変数1個

double型変数3個の一括宣言

- 変数名のルール（関数名も同じルール）
  - 大文字[A-Z], 小文字[a-z], 数字[0-9], アンダースコア"_"を使う
  - 大文字と小文字は区別
  - 最初の1字は数字以外

# なぜ型が必要なのか

- コンピュータは0と1しか扱えない
  - 通電状態を1、絶縁状態を0とする半導体のスイッチの集合

メモリのイメージ

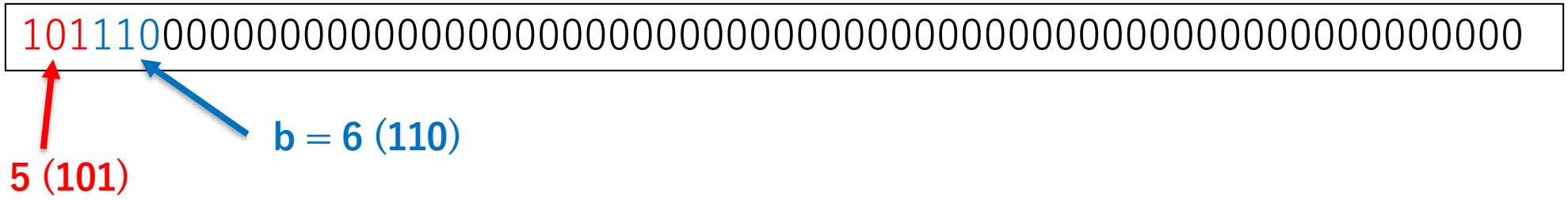
```
01010111000010100100010111010010010101110000101001000101110100 10
```

- コンピュータの中では全てを0と1の2進数で表現
  - 十進数 → 二進数
    - 0 → 0
    - 1 → 1
    - 2 → 10
    - 3 → 11
    - 4 → 100

# なぜ型が必要なのか

問題１. 型が無いと変数が必要とするメモリ領域のサイズがわからない

メモリのイメージ



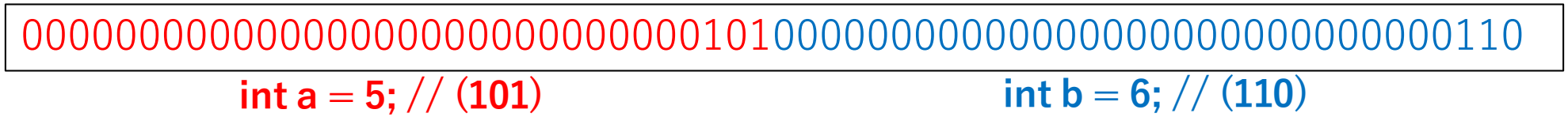

```
a = 5;    // 101
b = 6;    // 110
a = 10;  // 1010
```

a = 10を代入するスペースが無い！

# なぜ型が必要なのか

- よって型のメモリサイズがあらかじめ決まっている
  - ただしサイズはC言語で決まってるわけではなく、処理系に依存
  - int 型なら普通は32 bit

メモリのイメージ




```
a = 5;    // 00000000000000000000000000000101
b = 6;    // 00000000000000000000000000000110
a = 10;   // 00000000000000000000000000001010
```

メモリスペースが同じなので、a = 10を代入できる

# 型の表現範囲

- 各型が何bitかは厳密には決まってない。よって表現範囲も厳密には決まってない。
- C99以降の規格では、固定幅整数型などがある（ここでは扱わない）
  - int32_t, int64_tなど

| 型 | 説明 | メモリサイズ [byte]<br>(1 byte = 8 bit) | 表現できる最小値, 最大値 （メモリサイズが「**普通**」のとき） |
|---|---|---|---|
| char | 文字(character) | 1 | -128, 127 |
| short int | 表現できる範囲がint以下の整数 | int 以下（普通は2） | -32767, 32767 |
| int | 整数(integer) | 2以上（普通は4） | -2147483648, 2147483647 |
| long int※ | 表現できる範囲がint以上の整数 | int 以上（普通は8） | -9223372036854775808,<br>9223372036854775807 |
| long long int | 表現できる範囲がlong int以上の整数 | long int 以上（普通は8） | -9223372036854775808,<br>9223372036854775807 |
| float | 単精度浮動小数点数<br>(floating-point number) | 普通は4 | 1.175494e-38, 3.402823e+38 |
| double | 倍精度浮動小数点数 | 普通は8 | 2.225074e-308, 1.797693e+308 |

※特にWindowsではlong intが4 byteで、intと表現範囲が変わらないことがあるので注意

# なぜ型が必要なのか

問題２．０と１の並びだけでは、何を表したものなのかわからない

01000001010000100100001001000001

– ASCII文字として1 byte ずつ読むと

01000001 | 01000010 | 01000010 | 01000001

A B B A

– 32bit整数として読むと

01000001010000100100001001000001

**1,094,861,377**

– 32bit浮動小数点数として読むと
  - IEEE 754単精度浮動小数点数
  - 浮動小数点数形式そのものは本授業では扱わない

符号 指数部 仮数部

0 | 10000010 | 10000100100001001000001

**約12.1412**

# 変数への代入、初期化

- 代入

```
char c;
int n;
double x;

c = 'A';
n = 10;
x = 123.4;
```

この時点では変数の値は不定。何が入っているかわからない。

char型変数へは引用符(‘　’)を書いて代入

- 初期化
  - 宣言時にそのまま代入することもできる

```
char c = 'A';
int n = 10;
double x = 123.4;
```

# 算術演算子と複合代入演算子

| 演算子 | 構文 | 意味 |
|---|---|---|
| + | a + b | 加算(addition) |
| - | a - b | 減算(subtraction) |
| | - a | 単項マイナス (符号反転) (unary minus) |
| * | a * b | 乗算(multiplication) |
| / | a / b | 除算(division) |
| % | a % b | 剰余 (整数型のみ) (modulo) |
| ++ | ++a | 前置インクリメント、1つ増やす、インクリメント演算子を先に処理 |
| | a++ | 後置インクリメント、1つ増やす、インクリメント演算子以外を先に処理 |
| -- | --a | 前置デクリメント、1つ減らす、インクリメント演算子を先に処理 |
| | a-- | 後置デクリメント、1つ減らす、インクリメント演算子以外を先に処理 |
| += | a += b | 加算代入(addition assignment)、 a = a + b と同じ |
| -= | a -= b | 減算代入(subtraction assignment)、 a = a - b と同じ |
| *= | a *= b | 乗算代入(multiplication assignment)、 a = a * b と同じ |
| /= | a /= b | 除算代入(division assignment)、 a = a / b と同じ |
| %= | a %= b | 剰余代入(modulo assignment)、 a = a % b と同じ |

注意
- べき乗演算子（**）は使えない
- 文字列操作（結合等）には使えない

# 四則演算

```
#include <stdio.h>

int main()
{
  int a,b,c,d;
  a = 5 + 3 * 2;
  b = (5 + 3) * 2;
  c = a / 5;
  d = a % 5;

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

**計算の優先順位に注意**

**フォーマット指定子**

- コンパイルと実行

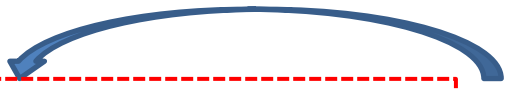
```
$ gcc sansu.c
$ ./a.out
a = 11
b = 16
c = 2, d = 1
```

**小数点以下切り捨て**

# 出力関数printf()

- printf()で変数の値を出力
  - 2重引用符（" "）の後にカンマ区切りで変数（定数も可）を指定
  - 変数の型と出力したい形式に応じて、2重引用符内で値を出力したい位置にフォーマット指定子（format specifier）を記述する
    - %dは10進符号付き整数のフォーマット指定子
    - つまり、**与えられた2進数の型が何だと思って読めばいいか**を指定する。printf()関数は**引数として与えられた変数の型を推論しない**。

```
int h;
h = 3776;
printf("Mt. Fuji is %d meters high¥n", h);
```



- 第一引数は2重引用符で括った文字列。変数の値を出力したい部分にはフォーマット指定子を書いておく。
- 第二引数以降に出力したい変数を指定する。

# 主なフォーマット指定子

| フォーマット指定子 | 説明 |
|---|---|
| %c | 文字 |
| %s | 文字列 |
| %hd または %hi | 10進符号付き整数、 short int型 |
| %d または %i | 10進符号付き整数、 int型 |
| %ld または %li | 10進符号付き整数、 long int型 |
| %lld または %lli | 10進符号付き整数、 long long int型 |
| %u | 10進符号なし整数、unsigned int型 |
| %zu | 10進符号なし整数、 size_t型（sizeofの戻り値） |
| %o | 8進数符号なし整数 |
| %x または %X | 16進数符号なし整数 |
| %f | 小数形式浮動小数点数、 float型 |
| %lf | 小数形式浮動小数点数、 double型 |
| %e | 指数形式浮動小数点数 |
| %p | ポインタの値 |

- 表示桁数の指定なども可能：%4d, %08d, %4.2fなど
- 指定子と変数の型が一致しなくても、解釈できる範囲であればそれらしく表示される

# if文とif-else文

- 式の評価結果が真(true)か偽(false)に応じて実行する文を選択

```
if (式) {
  式が真と評価された場合に実行される文;
}
```

```
if (式) {
  式が真と評価された場合に実行される文;
} else {
  式が偽と評価された場合に実行される文;
}
```

```
if (式1) {
  式1が真と評価された場合に実行される文;
} else if (式2) {
  式1が偽かつ式2が真と評価された場合に実行される文;
} else {
  式1も式2も偽と評価された場合に実行される文;
}
```

- 式では関係演算子を使った値の比較を行う
- さらに論理演算子を使って複数の式を使うこともできる

- Cはelse if
- Pythonはelif

# 関係演算子(relation operators)と論理演算子(logical operators)

| | 演算子 | 構文 | 意味 |
|---|---|---|---|
| 関係演算子 | > | a > b | より大きい (greater than) |
| | >= | a >= b | より大きいか等しい (greater than or equal to) |
| | < | a < b | より小さい (less than) |
| | <= | a <= b | より小さいか等しい (less than or equal to) |
| | == | a == b | 等しい (equal to) |
| | != | a != b | 等しくない (not equal to) |
| 論理演算子 | && | p && q | AND (論理積) (logical AND) |
| | \|\| | p \|\| q | OR (論理和) (logical OR) |
| | ! | ! p | NOT (否定) (logical negation) |

＆や｜は一つだけだとビット演算子（整数のビット単位演算）になってしまうため注意
（文法としておかしくなければコンパイラは警告も何も出せない）

# switch文

- 選択肢が多い場合はif文よりswitch文が便利
- switch文の一般的な形式

```
switch (値) {
  case 定数1:
    文;
    break;
  case 定数2:
    文;
    break;

  /* 中略 */

  default:
    文;
    break;
}
```

値と定数（整数）が一致する部分の文が実行される

break文を書かないと次のcaseに進んでしまう
（それを意図的に利用しても良い）

- 条件を書く順番に決まりはない
- いきなりdefault条件を書いても問題ない

どれとも一致しない場合はdefaultの部分が実行される

# forループ

- 繰り返し構造
  – 1つ以上の文で構成されるブロックを所定の回数繰り返し実行
- for文の一般的な形式

```
for (初期値設定部; 条件判定部; インクリメント部) {
  文;
}
```

- 例（1から10までの数を画面に出力）

```
int i;
for (i=0; i<10; i++) {
  printf("%d¥n", i+1);
}
```

# whileループとdo-whileループ

- while文の一般的な形式

```
while (式) {
  文;
}
```

  - 式が真の間だけ文を繰り返し実行する
  - 式の評価は一回目の文の実行前に行うため、「文」が一回も実行されないことがある

- do-while文の一般的な形式

```
do {
  文;
} while (式);
```

  - 式が真の間だけ文を繰り返し実行する
  - 式の評価は文の実行後に行うため、最低1回は「文」が実行される

# C言語学習環境の準備

# C言語開発に必要なソフトウェア環境

- テキストエディタ
  - ソースコードを編集する。テキストが編集できればなんでもいい（メモ帳でも可）
  - VS Code, Emacsなど
- コンパイラ
  - C言語を機械語に翻訳
  - gcc, clang など
- ターミナル
  - コンパイルや実行の命令を出すコマンド入力環境
    - コマンド例

```
gcc main.c -o main
./main
```

  - PowerShell/コマンドプロンプト/ WSL2のUbuntuのターミナル(Windows), ターミナル（mac, Linuxの標準）, VS Codeの内蔵ターミナルなど

**ChatGPT曰く、エディタ＋コンパイラは VS Code＋gcc が推奨。ターミナルはOSによって異なる**

# Windowsの場合

- WSL2（Ubuntu等）＋ VS Code がおすすめ（ChatGPT曰く）
  - 私がChatGPTに聞きながら作った資料があるので、WSL2_VSCode.pdfを参照

- MSYS2（MinGW-w64）＋ VS Code も可
  - 前任者による資料があるので、MinGW_VSCode.pdfを参照

# Macの場合

- Xcodeでclang導入 + VSCodeで開発が一番楽らしい（ChatGPT曰く）
  - ターミナルアプリを開いて以下を入力

```
$ xcode-select --install
```

  - これによってgccコマンドも使えるようになるが、中身はclang（単にシンボリックリンク）
  - VSCodeは別途インストール
  - VSCodeでコードを開発して、ターミナルで実行

- 他と合わせてGNUのgccコンパイラが欲しい場合は、Homebrewあたりで導入

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
$ brew install gcc
```

# 使用するPCについて

- 自分の端末でも、ICEを使っても良い
  - ICEにSSH接続できるようなら、それでOK

# 第1回 演習

# EX-1.1

- 与えられた４桁以下の自然数を２進数表記で表すプログラムを作成せよ
  - 入力の数値が4桁以下の正整数であることを仮定してプログラムを作って良い
  - ２進数はプログラム中で計算すること。あらかじめ計算された２進数を表示するだけは不可
- ただし、出力の２進数は見た目が対応する10進数で代用すること
  - 例えば３を二進数で表現すると11なので、見た目が対応する10進数である11（十一）を出力するプログラムを作れば良い
    - 0 → 0
    - 1 → 1
    - 2 → 10（十）
    - 3 → 11（十一）
    - 4 → 100 （百）

# EX-1.1

ソースコード

```c
#include <stdio.h>

int main()
{
  long long int upTo4Digits = 1023;   // この数字を任意の4桁以下の自然数にする
  long long int binaryNum = 0;


  // ここを実装

  printf("10進数=%lld, 2進数=%lld\n",upTo4Digits,binaryNum);
  return 0;
}
```

※ 敢えての画像です。最初なので、一行目から自分で打ち込みましょう。

実行例

```
$ gcc ex1_1.c -o ex1_1
$ ./ex1_1
10進数=1023, 2進数=1111111111
```

# EX-1.1 注意点

- C言語に冪乗の演算子はありません
  - 標準ライブラリにpow関数があるが、本課題では使わないでください
- 0から始まる数値はC言語では8進数と見做されるので注意
  - 12　　← 10進数で、12
  - 012　← 8進数で、10進数に直すと10
  - 0x12 ← 16進数で、10進数に直すと18
- 整数の表現範囲に注意すること
  - 資料24ページ参照
  - 処理系によって特にlong intのサイズが異なることがあるので注意。以下のプログラムで調べられる。

```
#include <stdio.h>

int main(){
  printf("int:          %d bytes\n",sizeof(int));
  printf("long int:     %d bytes\n",sizeof(long int));
  printf("long long int: %d bytes\n",sizeof(long long int));
  return 0;
}
```

# EX-1.1

- ヒント

13の二進数表記 1101 は、




のように 2 の冪乗*(0 or 1)の足し算の形で表される。

逆に、13を8で割ると商1余5、
余りの5を4で割ると商1余1
余りの1を2で割ると商0余1
余りの1を1で割ると商1余0
となる

# EX-1.2

- 「３の倍数と３の付く数字だけアホになり、かつ８の倍数で犬っぽくなる」プログラムを作成せよ
  - 3の倍数と3の付く数字の時には、「!!」付きで数値を出力
  - 8の倍数の時には、回数分「わん」と鳴く
  - 1から40まで正しく動くことを確認すること

```c
#include <stdio.h>

int main(){

  int length = 40;




  // ここに実装




  return 0;
}
```

# EX-1.2

出力例（24まで）

```
1
2
3!!
4
5
6!!
7
わんわんわんわんわんわんわんわん
9!!
10
11
12!!
13!!
14
15!!
わんわんわんわんわんわんわんわんわんわんわんわんわんわんわんわん
17
18!!
19
20
21!!
22
23!!
わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!わん!!
```

# EX-1.2

- ヒント

if文の複数条件指定には、&&や||を使う

```
(i % 3 == 0) && (i % 8 == 0)
```

← iを3で割った余りが0かつ8で割った余りも0

```
(i % 3 == 0) || (i % 8 == 0)
```

← iを3で割った余りが0または8で割った余りが0

for文を入れ子にした２重ループ

```
int i, j;
for( i = 0; i < 40; i++){
    for( j = 0; j < i; j++){
```

← 内側のループの終了条件などに、外側のループ変数を使うこともできる

# よくある質問

- Q: ソースコードを編集したのに結果が変わりません。
- A: 編集後にコンパイルはしましたか？コンパイルしないと実行ファイルが更新されません。

# 提出

- 演習課題EX-1.1, EX-1.2
- TACTでソースファイルと実行結果（画面に出力されたものをコピー＆ペーストしたテキストファイル）を提出
  - ソースファイルの名前は　学籍番号-1-{課題番号}.c　とすること
    - 例：252301111-1-1.c
  - 結果ファイルの名前は　学籍番号-1-{課題番号}.txt　とすること
    - 例：252301111-1-1.txt
  - 実行可能ファイル（gccにより生成されたファイル）の提出は不要
  - ２課題分計4ファイル提出
- 提出期限
  - **次々回（12/12）講義開始前まで**