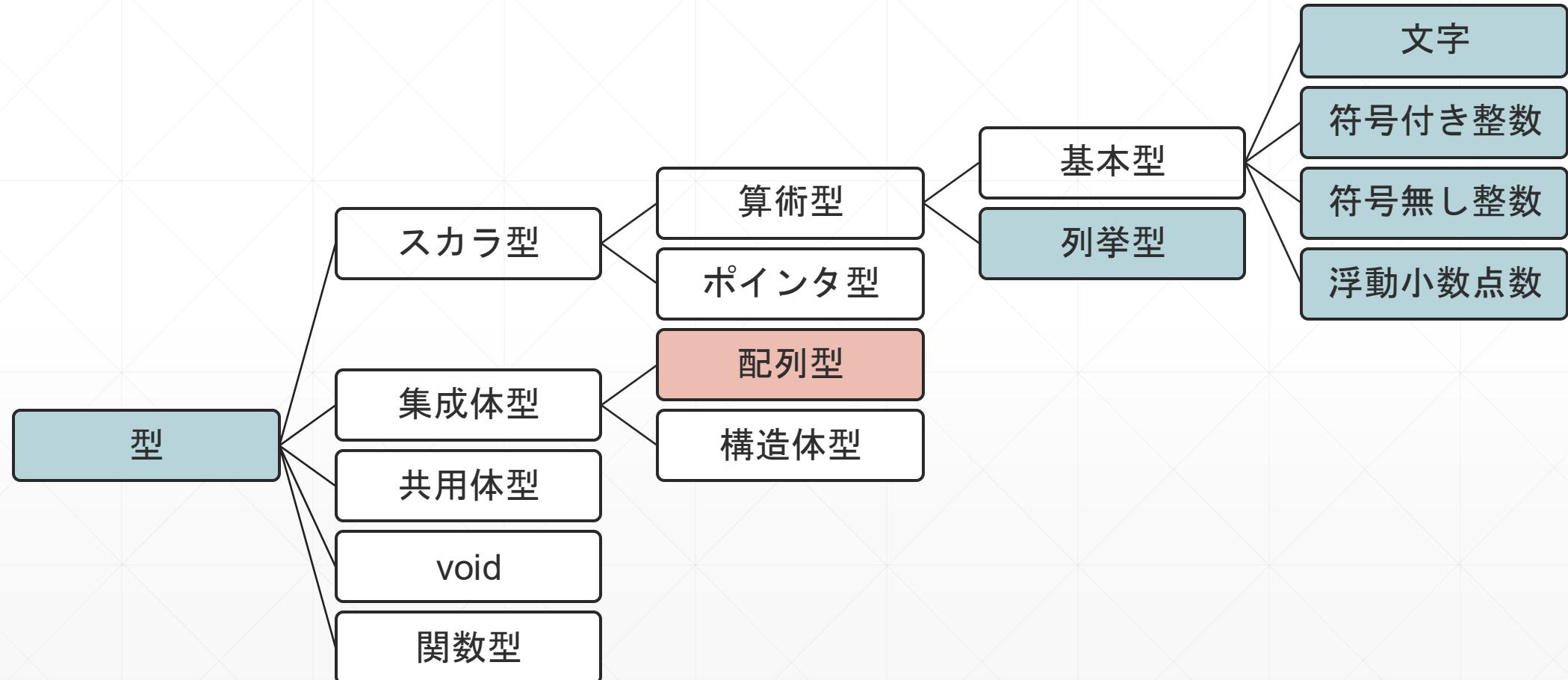


コンピュータとプログラミング C言語編

3. 配列、関数

阿部洋丈 / ABE Hirotake
habe@cs.tsukuba.ac.jp

C言語における型の一覧（再掲）



配列

- 同じ型の変数を複数個並べたもの。添字(index)を使って個々の変数にアクセスできる（前半で説明済）
- 境界検査が無いので細心の注意が要
- Python におけるリストとタプルの中間的な存在
 - 長さは固定。なので途中挿入や末尾追加もできない
 - 値を書き換えることが可能
- リストやタプルのように多次元配列も可能
- 文字列は、char 型の配列として実現されている

配列の定義と使い方

- Python とは違い、事前に型を指定する必要がある

```
int a[10];    // int 型の整数が10個用意される。中身は不定
int b[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};    // 初期化
char c[] = "Hello world!\n";    // 初期化がある場合は長さを省略可
float d[][][3] = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}};
```



```
a[0] = b[1];    // 添字は 0 から始まる (Python と同じ)
b[1] = a[2];    // a は未初期化なので、b[1] に何が入るかわからない (バグのもと)
a[10] = b[3];    // 境界の外へのアクセスのため 非常に危険！ (このあと詳しく)
putchar(c[0]);    // "H" だけが表示される
printf(c);    // "Hello world!\n" 全体が表示される
d[0][0] += d[1][1];    // 二次元配列の使い方
```

配列の境界について

- C言語における配列
 - その格納に必要なメモリ領域を確保するだけ
 - 添字がその領域外になっていても実行できてしまう
- 領域外からの読み込み：
 - 何が返ってくるかわからないので、高確率でバグの原因になる
- 領域外への書き込み：
 - 書き換えていないはずの変数の値が変わってしまうため、発見が非常に難しい厄介なバグが発生する
 - リターンアドレスなど、本来書き換えてはいけない領域を書き換えてしまうとセキュリティホールになり得る
- Python のような負の添字 (e.g. `a[-1]`) は領域外アクセスになるので注意

境界チェックの方法

- 外部から渡された値を添え字として配列にアクセスする前には必ずチェックすることを推奨

```
#define MAX 10

int f(int i) {
    int a[MAX];
    (中略)
    if (0 <= i && i < MAX) {
        a[i] = ...; // 正しい処理
    } else {
        ... // エラー処理
    }
    (以下略)
}
```

```
#include <cassert.h>
#define MAX 10

int f(int i) {
    int a[MAX];
    (中略)
    assert(0 <= i && i < MAX); // false なら停止
    a[i] = ...;
    (以下略)
}
```

もう一つの方法

assert は "#define NODEBUG" を定義すると無効化される

文字列について補足

本来は'＼'(バックスラッシュ)だが、日本語環境では'＼'と表示されることが多い

- C言語には、文字列の終わりを'＼0'(null文字)という特殊文字で表すというルールがある
- char配列を"..."で初期化した場合は自動的に入る
- 初期化されていないchar配列の中身を自分で作る場合は最後に'＼0'を入れる必要がある

```
char s[] = "Hello world"; // 英字10文字 + 空白1文字 + null 1文字 で  
                           // 合計12文字分のcharが確保される  
strlen(s);           // 文字列の長さを数える関数。null文字はカウントされないので  
                     // 結果は11になる。  
s[5] = '\0';         // 6文字目の空白文字の上にnull文字を上書き  
strlen(s);           // 結果は5に変わる。ただし、配列の長さは12のまま  
printf(s);          // Helloと表示される
```

C言語における関数

- Python の def で定義できるものと基本的に同じ
 - 値を受け取って、その値に基づいた計算結果を返す
 - 複数の処理を一つにまとめ、再利用しやすくする
- 使い方や定義の仕方がすこし違う
 - Python は関数なしでもプログラムが書けるが、C言語では関数が必要
 - ソースファイル (.c) の外で定義された関数を使うためにはプロトタイプ宣言が必要

関数の呼び出し方法

- f という関数に a, b, c という3つの引数を渡し、その関数の返戻値を r という変数に保存する場合：

```
r = f(a, b, c);
```

- 他のバリエーション：

```
r1 = f(a, b, c) + g(a, b, c); // 返戻値を使った式  
r2 = f(a, b, g(a, b, c));    // 関数の合成  
f(a, b, c);                  // 返戻値を保存しない
```

Python の関数呼び出しとの違い

- 引数の順番は変えることはできない
 - キーワード引数やデフォルト引数は使えない
- 引数の型や返戻値の型は Python よりも厳密に考える必要がある
 - もし違っていると暗黙の型変換が起きる場合がある
- 返戻値は基本的に1つしか返せない。複数の値を返すためには構造体やポインタを使う必要がある
 - 次回以降詳しく説明

関数の定義

- Python と違い、返戻値や引数の型を明示的に指定

```
int func(int x, int y, int z) {  
    int v = 0;  
    v = x * y + z;  
    return v;  
}
```

変数宣言

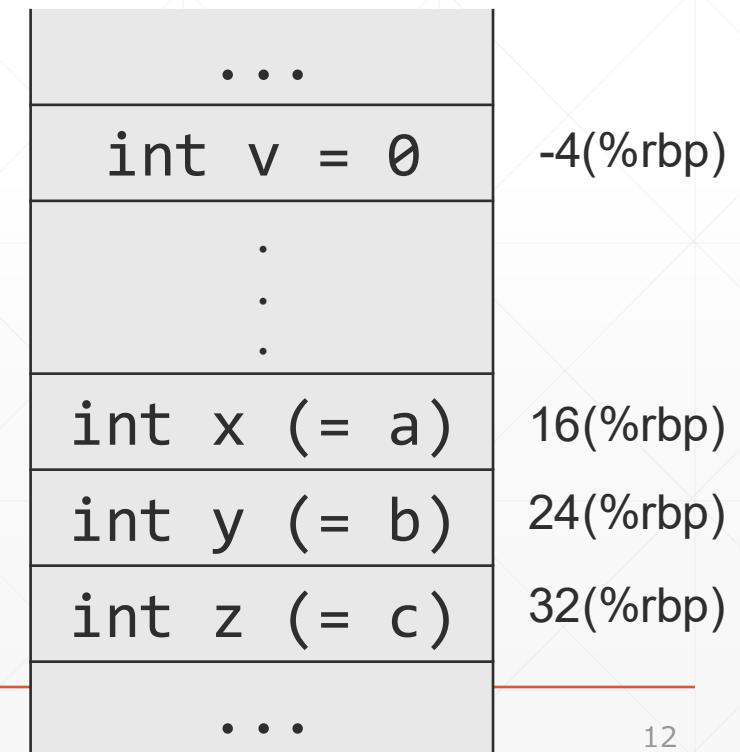
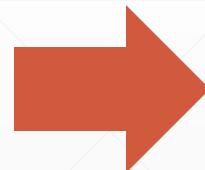
処理の中身

呼び出し元に
返戻値を渡す

関数呼び出しと実行時スタック

- 既に授業の前半で学んでいるように、関数呼び出しは実行時スタックを使って実現されている

```
int func(int x, int y, int z) {  
    int v = 0;  
    v = x * y + z;  
    return v;  
}
```



注1：数が少ない場合はレジスタ経由になる場合もある

注2：実際の相対アドレスは異なる場合がある

変数の種類（寿命の観点から）

- 前ページの例から分かること：
 - 呼び出し側の関数から渡された引数や、呼ばれた側の関数の中で宣言された変数をいくら書き換えても、一旦 return してしまうと別の関数に上書きされてしまう
 - このような変数を自動変数と言う。初期化はされない。
- 関数呼び出しに関係なく存在し続ける変数を作ることもできる。（ただし安易な使用は推奨されない）
 - 外部変数：関数の外側で変数を宣言。複数の関数から利用可
 - 静的変数：関数内で static を付けて宣言。その関数のみ利用可
 - 外部変数と静的変数は0で初期化される。

外部変数、静的変数の例

- 以下の 2 つの関数は、いずれも、自分が何回呼ばれたかを記憶し、これまでに呼ばれた回数を返す

```
int sum_gl = 0;

int increment_global(void) {
    return sum_gl++;
}

int increment_static(void) {
    static int sum_st = 0;
    return sum_st++;
}
```

外部変数

(すべての関数
から利用可能)

静的変数

(定義している関数
のみ利用可能)

プロトタイプ宣言

- ソースファイルの外で宣言された関数を呼ぶ際に、その関数がどんな関数であるか（引数や返戻値の型）をコンパイラに教えるための宣言
- printf を呼ぶために stdio.h を include する理由

```
extern int printf (const char * __restrict __format, ...);
```

この関数の実体は別のソースファイルで定義されているということを表している

変数の数が可変であることを表している
(のちほど詳しく)

main 関数

- C言語で書かれたプログラムは基本的にここから実行が開始される（エントリポイント）
- 返戻値の型は int。処理の結果を表す値を返すようにしておくと便利なことがある（成功したら0、失敗したらそれ以外）
- 引数 : int argc, char **argv
 - プログラム名および起動時の引数が argv に入る。argc はその個数
 - argv は、ひとまずは char の二次元配列のようなものだと理解しておいて良い
(詳しくは次回説明)

```
int main(int argc, char **argv) {  
    int i;  
    for (i = 0; i < argc; i++) {  
        printf("%d: %s\n", i, argv[i]);  
    }  
    return 0;  
}
```

% ./a.out abc 123 4.56



argc: 4

| argv: |
|---------|
| ./a.out |
| abc |
| 123 |
| 4.56 |

printf 関数

- さまざまな型の値を使って文字列を生成し、結果を画面に表示する関数
- 第一引数は、文字列のフォーマット。%で始まる部分に値が埋め込まれる
 - %d は整数（10進数）、%f は浮動小数点数、%s は文字列、…
 - 表示桁数やゼロ詰めの有無なども指定することができる

```
printf("%f\n", 123.456);           // 123.456000
printf("%10.2f\n", 123.456);       //      123.46
printf("%010.2f\n", 123.456);      // 0000123.46
```

- それ以降の引数は、フォーマットに埋め込むべき値。フォーマット中に%が複数回出てくる場合は、その個数分だけ並べる（可変長引数）
- 返戻値は、生成した文字列の文字数

scanf 関数

- printf の反対に、指定されたフォーマットに従って標準入力（＝ターミナルのキーボード入力）を読み取る関数
- 読み取った値を格納する先が配列でない場合は変数名の頭に&をつける。配列の場合は&は不要（理由は次回説明）
- 文字列を読み取る場合は配列の境界に注意が必要

```
int i;  
float f;  
char s[10];  
  
scanf("%d", &i);  
scanf("%f", &f);  
scanf("%9s", s); // null 文字の分を考慮
```

scanf の難しいところ

- 期待通りの入力が来ない場合の扱いが厄介
- フォーマット指定にマクロを使うのが面倒
- 読みきれなかった文字は次の scanf に回されてしまう

```
#define MAX 10

scanf("%d", &i);      // 1.2 を与えるとどうなる?
scanf("%f", &f);
scanf("%MAXs", s);   // これは期待通りには動かないで注意!
scanf("%9s%*[^\n]", s); // 10番目より後は読み捨てる
getchar();           // 改行を読み捨てる(次も scanf なら無くても動く)
```

その他の文字入出力関数

- `getchar, putchar`: 文字単位
- `gets, puts`: 文字列単位
 - ただし、`gets` は読み込む文字数を制限できないので危険
`scanf` を適切に使うか、`fgets` という関数を使うことを推奨

```
#define MAX 10

char s[MAX];

gets(s); // 9 文字以上でも読み込んでしまい、結果的にメモリを破壊する
fgets(s, MAX-1, stdin); // 9 文字目で読むのを止める
```

文字列関数の一部 (string.h で定義)

- strlen: 文字列の長さ (\neq 配列の大きさ) を返す
- strcmp: 2つの文字列を辞書式順序で比較する。一致するかの検査の他に、文字列ソートでよく使う。一致する場合は0を返すので要注意
- strstr: 文字列の中から特定の部分文字列の出現位置を見つける
- strncpy: 文字列を指定された文字数分だけコピー
- strncat: 文字列を他の文字列の後ろに連結

数学関数の一部 (math.h)

- fabs, floor, ceil, round, ...
- sqrt, cbrt, pow, exp, log, log10, ...
- sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, ...
- 上記は double 用。 float や long double 用は別の名前になっている
- これらを使う場合はコンパイル時に "-lm" オプションが必要(libm をリンク)

その他、演習等で使いそうな関数 (stdlib.h)

- exit: プログラム実行の終了
 - EXIT_SUCCESS、EXIT_FAILURE 等を引数に与える
- atoi, atol, atof: 文字列を int, long, double に変換
- rand, srand: 亂数の生成とシード設定
- malloc, free, calloc, realloc: メモリ管理用関数（第5週で詳しく）

今日の演習

```
for (i = ...; ...; ...) {  
    for (j = ...; ...; ...) {  
        ...  
    }  
}
```

- 以下の指示に従い二次元配列を扱うプログラムを作成せよ
 - まず、以下の例にならって、 3×3 の行列Aの内容を表示する（二重のfor ループを使う）
 - Aを転置する（同様に二重のfor ループを使う）
 - Aを転置した結果をさらに時計回りに90度回転させる
(転置と同様に二重のfor ループで書けるが、やや複雑)
 - 上の二つの処理を行った結果の行列を再び画面に表示する
 - 4×4 の行列を与えた場合でも正しく動くことを確認せよ

```
#define DIM 3  
double A[DIM][DIM] = {{1.0, 2.0, 3.0},  
                      {4.0, 5.0, 6.0},  
                      {7.0, 8.0, 9.0}};
```



| | | |
|---------|---------|---------|
| 01.0000 | 02.0000 | 03.0000 |
| 04.0000 | 05.0000 | 06.0000 |
| 07.0000 | 08.0000 | 09.0000 |