データ構造とアルゴリズム

第1回　構造体と共用体

メディア情報コース
平居　悠（ひらい　ゆたか）

データサイエンス・AI教育プログラム

データサイエンスやAIに関する基礎的な知識と技術、及びその知識や技術を他の科目や学修で応用する能力

MDASH Literacy 数理・データサイエンス・AI 教育プログラム認定制度 リテラシーレベル

MDASH Advanced Literacy 数理・データサイエンス・AI 教育プログラム認定制度 応用基礎レベル

「データサイエンス・AI教育プログラム」の流れ （※8単位以上修得）

1年次秋学期	2年次春学期	2年次秋学期
データリテラシー	基礎プログラミングⅠ	基礎プログラミングⅡ

選択必修科目：「AIと社会」「セキュリティ論」「日経講座：デジタル社会論」
選択科目：「数学a」「数学b」「統計学a」「統計学b」「データサイエンス入門b」「数値情報処理b」「画像情報処理」「データ分析手法b」「経営工学a」「経営工学b」「データベース論」「データベース演習」「データ構造とアルゴリズム」「応用プログラミング」「機械学習入門a」「機械学習入門b」

授業概要

C言語は、現在主流な多くのプログラミング言語の基礎となっており、現在でも広く使われている。C言語を学ぶことで、プログラミング言語やコンピュータの動作を理解することができる。本講義では、「応用プログラミング」で学んだことをベースにC言語のより高度な文法を学ぶ。これにより**目的に応じて適切で効率的なプログラムを自由に設計**できるようになることを目指す。

到達目標

- C言語を通じてデータ構造とアルゴリズムを理解する。
- 目的に応じて効率的なプログラムを自由に設計できる。

授業計画

第１回　構造体と共用体
第２回　探索とソート
第３回　メモリの動的確保と線形リスト
第４回　文字列の類似度計算、再帰、動的計画法
第５回　自由課題作成
第６回　自由課題作成
第７回　自由課題発表

成績評価

平常課題（６割）

自由課題（４割）

授業ページ

学生専用サーバ→平居→データ構造とアルゴリズム

学内：http://roy.e.koeki-u.ac.jp/
学外：https://www.koeki-prj.org/roy/

質問等

- s4グループ
- メール (yutaka.hirai@koeki-u.ac.jp)
- オフィスアワー（H2研究室木曜2限）

参考書


NEKO series
猫でもわかる
C言語プログラミング
第3版
粂井康孝 著
Yasutaka Kumei
SoftBank Creative


新・明解
柴田望洋
BohYoh Shibata
C言語
実践編
第2版
Textbook MEIKAI
by Dr.BohYoh Shibata
SB Creative


新・標準
プログラマーズ
ライブラリ
C言語
ポインタ
完全制覇
前橋和弥
Kazuya Maebashi
Cはなんでこんな言語に「なっちゃった」のか。そもそもこの悪名高いポインタとは何か。
初心者が必ずひっかかる、配列とポインタのまぎらわしい文法とは。
Cはメモリを実際にどんなふうに使うのか。Cの宣言は英語で読め。
ポインタの真の使い方は。Cの文法を深く知ることで見えてくること納得できること。
技術評論社

今回

第１回　構造体と共用体
第２回　探索とソート
第３回　メモリの動的確保と線形リスト
第４回　文字列の類似度計算、再帰、動的計画法
第５回　自由課題作成
第６回　自由課題作成
第７回　自由課題発表

今回の目標

構造体を定義してプログラムで使えるようになる。

今回学ぶこと

1.C言語基本文法の復習
2.構造体の宣言
3.構造体へのポインタ
4.共用体

今回の問い

- 構造体を宣言すると何ができるのか？
- 構造体ポインタが指す構造体のメンバにアクセスするには、どのようにすれば良いか？
- 共用体とは何か？

今回学ぶこと

1.C言語基本文法の復習

2.構造体の宣言

3.構造体へのポインタ

4.共用体

なぜC言語を学ぶのか？

- 多くのソフトウェアやOSはC言語で書かれている
- 処理が速く、メモリ消費量が少ない
- C言語を知っていると他のプログラミング言語の習得が容易になる

インタプリタ言語とコンパイラ言語

インタプリタ言語

Ruby
A PROGRAMMER'S BEST FRIEND

```
print “Hello, world!”
```

コードを1行ずつ翻訳

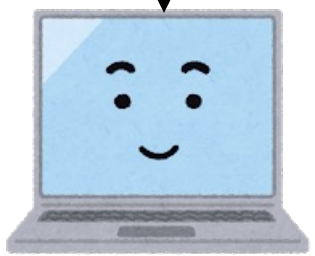
コンパイラ言語

C

```
#include <stdio.h>

int main()
{
   printf(“Hello, world!\n”);
   return 0;
}
```

コードを全体を翻訳



コンパイル

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

gccコマンド：コンパイルのため、言語処理系GCCを呼び出す

“-o”オプション：実行形式ファイル名を指定。
指定しないと“a.out”という実行形式ファイルが生成される。

式と演算子

演算子：特定の操作機能を有した記号 (+, −, *, /など)

オペランド：演算子の作用対象

配列の基本

配列：変数の集まり

宣言方法：

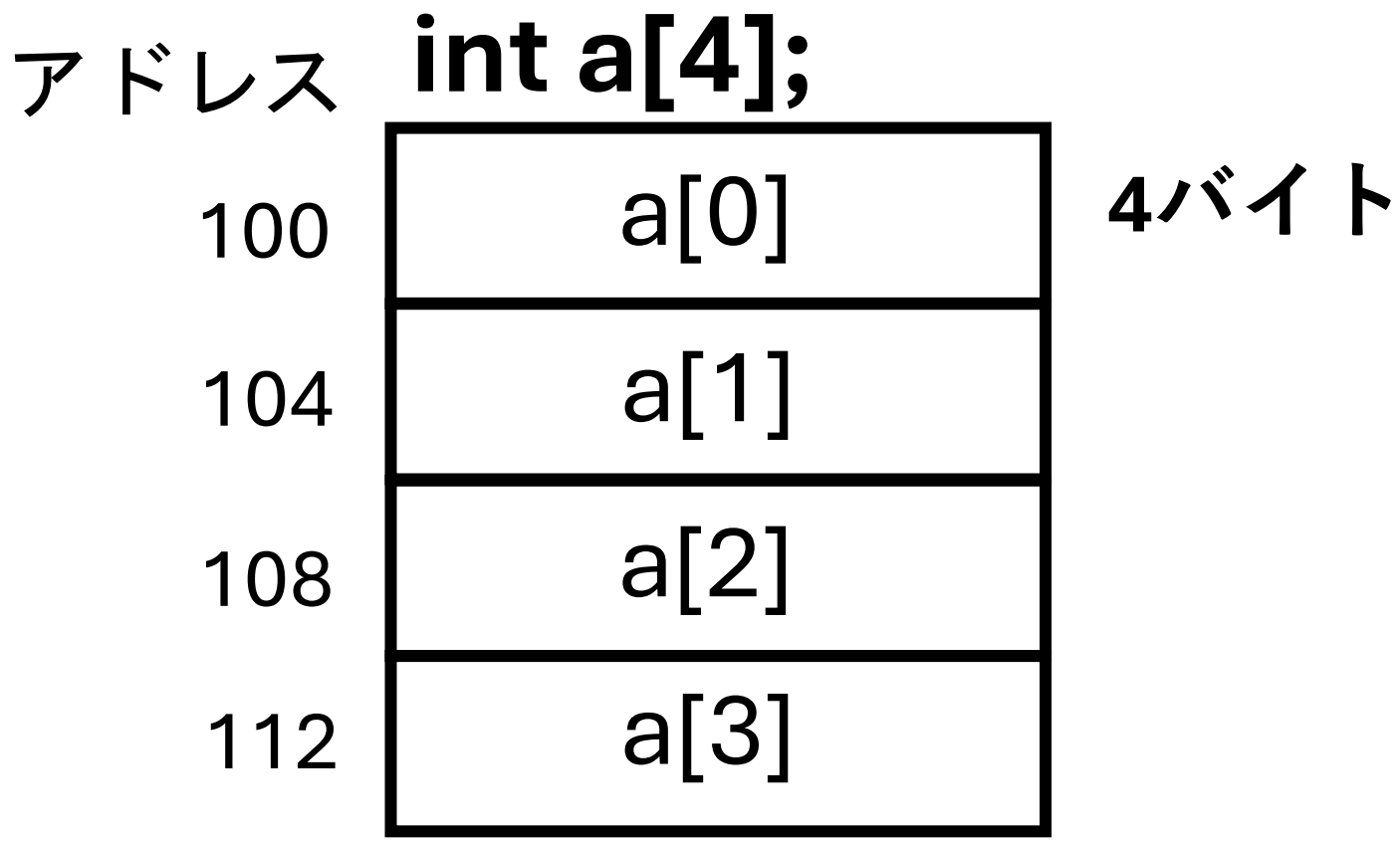
データ型　配列名[配列の大きさ];

例：

int a[4];

配列とアドレス

a[0], a[1], a[2], a[3]の順でメモリ上に隙間なく並んでいる



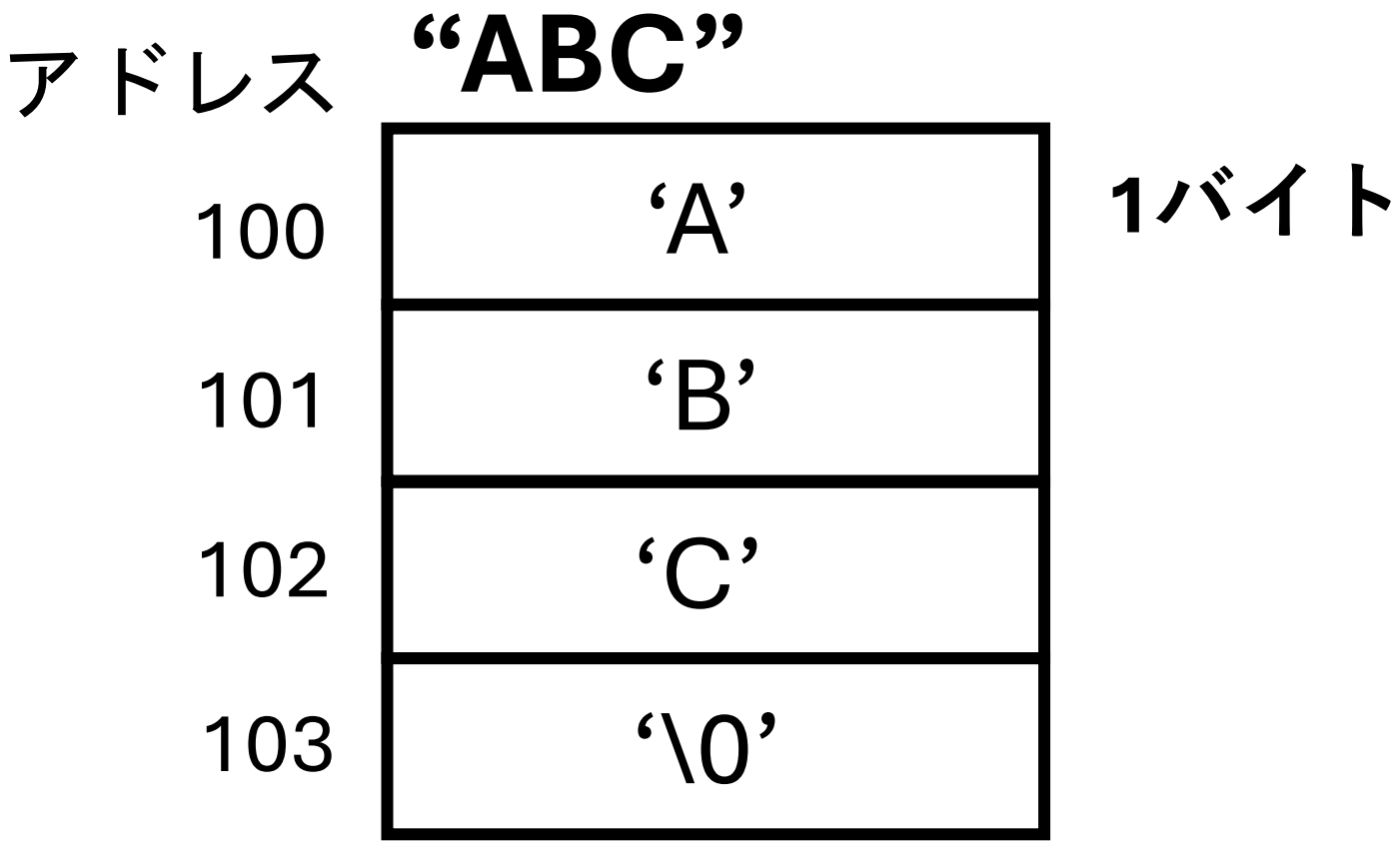
配列のメモリ内の状態

文字列

- ABCのような**文字列**という。
- “ABC”とダブルクォーテーションで囲ったものを**文字列リテラル**という。
- printf関数で文字列出力するときの変換指定子は**%s**

文字列とアドレス

- 文字列の式の値は先頭文字の置かれたアドレス
- 文字配列最後の要素は必ず **‘\0’ (ヌル文字**)



配列のメモリ内の状態

３つのデータ型

- 整数型
- 浮動小数点数型
- 文字型

キャスト演算子

式の型を変更する。

(型名X)式

ファイル入出力

- ファイルは**fopen**関数で開き、**fclose**関数で閉じる。
- ファイルへの出力は**fprintf**関数を用いる。

デバッグ

プログラムのバグを取り除く作業

GDB (GNU Debugger)

GNUプロジェクトで開発されたデバッガ。多くのUnix系システムで利用できる。

関数定義

```
戻り値の型　関数名(引数)
{
　やりたい処理;

　return 戻り値;
}
```

関数の再帰呼び出し

関数内で自分自身を呼び出すこと。

寿命とスコープ

```
int func(int x, int y)
{
      int kotae;
      kotae = x * y;
      return kotae;
}

int main()
{
   int c;
   c = func(10, 20);
   printf("10 x 20 = %d\n", c);

   return 0;
}
```

kotaeの寿命

変数kotaeはこの範囲（スコープ）内でのみ可視。main関数からは見えない。

cの寿命

変数cはこの範囲（スコープ）内でのみ可視。func関数からは見えない。

プロトタイプ宣言

後で利用する関数について、その引数の個数と型、返す結果の型を先に宣言すること。

Makefile

makeにファイル依存関係を教えておくと、makeを起動したときに適切な処理を行ってくれる。makeに依存関係を教えるには、作業ディレクトリに**Makefile**というファイルを置き、そこに依存関係を記述する。

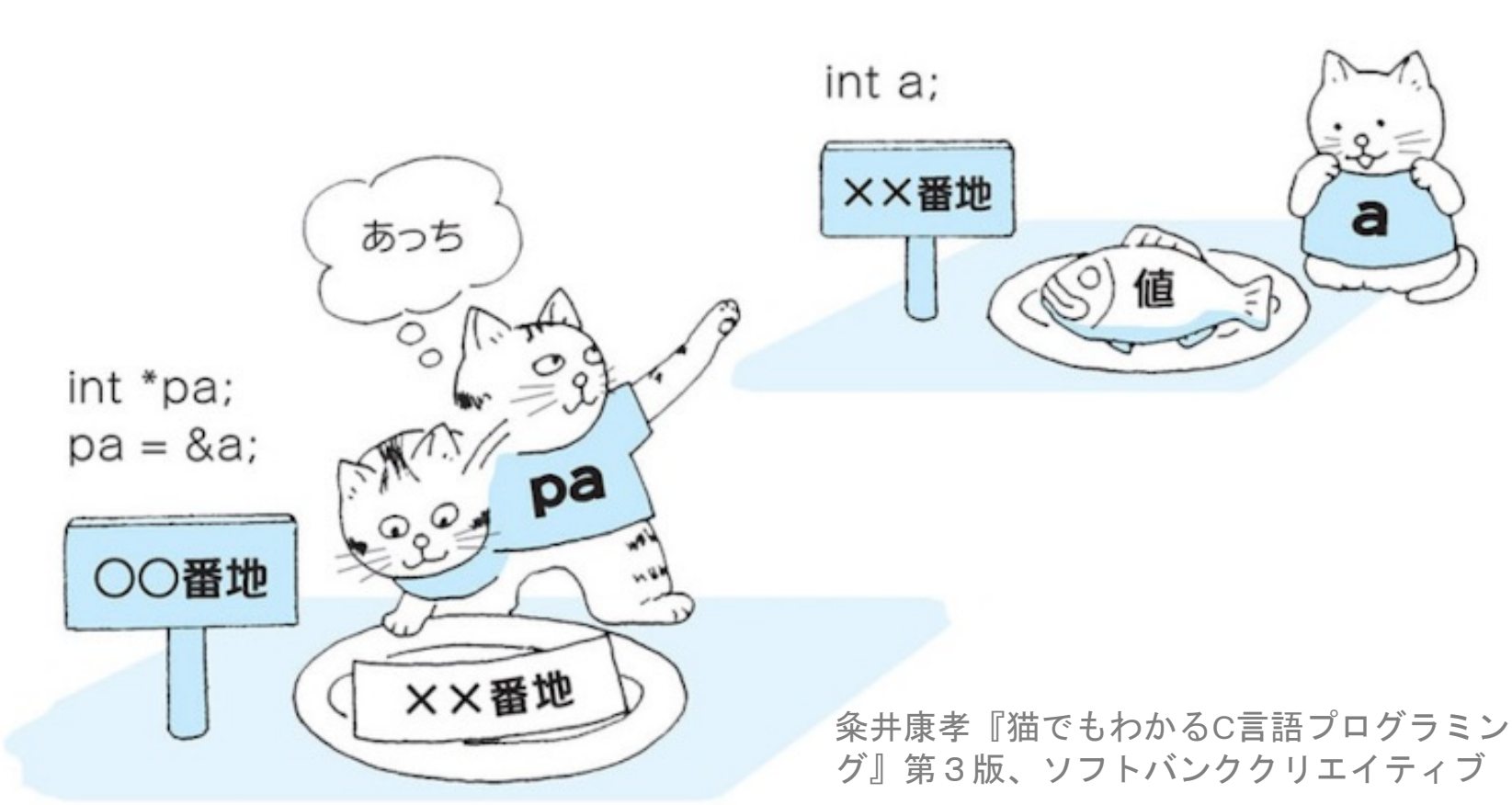
ポインタとは

アドレスを格納するための変数

アドレスの代入

変数aのアドレスをポインタpaに代入する場合：

```
int a;
int *pa;
pa = &a;
```



粂井康孝『猫でもわかるC言語プログラミング』第３版、ソフトバンククリエイティブ

paの値を見れば、aの番地がわかる

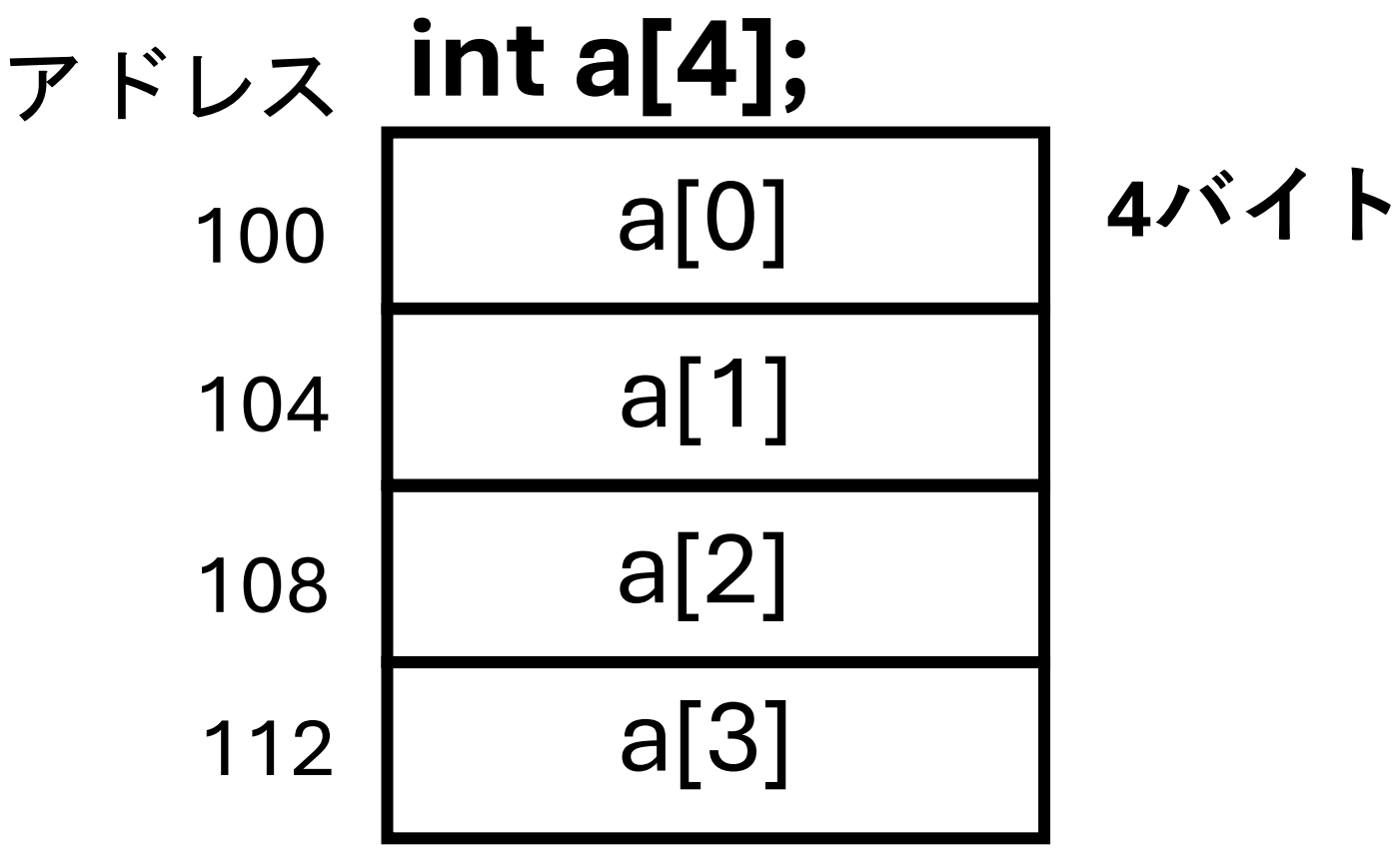
関数の参照呼び出し

変数の値を関数に変更してもらうには、変数のアドレスを渡す。

配列とアドレス

a[0], a[1], a[2], a[3]の順でメモリ上に隙間なく並んでいる。

配列名は、その配列の先頭要素のアドレスに変換される。



配列のメモリ内の状態

文字列とポインタ

“ABC”は先頭文字のアドレスを表しているので、
これをchar型のポインタに代入できる。

```
char *str;
str = “ABC”;
```

または、

```
char *str = “ABC”;
```

今日学ぶこと

1.C言語基本文法の復習

2.構造体の宣言

3.構造体へのポインタ

4.共用体

構造体 (structure)

異なるデータ型をひとまとめにして扱うことができる機能を持つデータ型の一種

構造体の宣言

```
struct 構造体タグ{
    メンバ;
    …;
};
```

メンバ：構造体が有する色々な型のデータ。

構造体のテンプレート

```
struct data{
    char name[32];
    int age;
    double bl;
    double bw;
};
```

テンプレート (templete, 鋳型)を
宣言しただけではまだ使えない。

構造体変数の宣言

struct data man;

構造体変数manはメンバに
name, age, bl, bwを持つ。

構造体テンプレートと構造体変数の宣言

```
struct 構造体タグ{
    メンバ;
    …;
} 構造体変数1, 構造体変数2, …;
```

構造体タグの省略

```
struct {
    メンバ;
    …;
} 構造体変数1, 構造体変数2, …;
```

この場合後から追加で変数を宣言できなくなる。

構造体メンバへのアクセス

構造体のメンバにアクセスするには、**構造体メンバ演算子 (.)**を使う。

構造体メンバへのアクセス

struct01.cというファイルを作成し、下のコードを書いて実行してみよう。

```
#include <stdio.h>

struct Data {
   int data1;
   char data2;
   char data3[32];
};

int main()
{
   struct Data mydata = {10,  ‘A’, “cat”};
   printf(“%d, %c, %s\n”, mydata.data1, mydata.data2, mydata.data3);
   return 0;
}
```

今回の問い

- 構造体を宣言すると何ができるのか？
  - 構造体は異なるデータ型をひとまとめに扱うことができる。
- 構造体ポインタが指す構造体のメンバにアクセスするには、どのようにすれば良いか？
- 共用体とは何か？

今回学ぶこと

1.C言語基本文法の復習

2.構造体の宣言

3.構造体へのポインタ

4.共用体

構造体の配列

構造体変数の配列を作るには、

```
struct Man student;
```

としていたところを、

```
struct Man student[5];
```

などとすればよい。

構造体の配列

student[2]のnameメンバにアクセスするには、

student[2].name

とすればよい。

構造体へのポインタ

構造体変数を

struct Man *student;

のように宣言すると、studentはMan構造体へのポインタとなる。

構造体へのポインタが指す構造体メンバへのアクセス

アロー演算子 **(->)**を用いる

```
student->name;
```

構造体へのポインタ

struct02.cというファイルを作成し、次ページのコードを書いて実行してみよう。

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

struct Data {
   char name[32];
   int age;
   double bl;
   double bw;
   double bmi;
};

int struct_input(struct Data *);

int main()
{
   struct Data mydata;
   char format[] = "%sさん (%d 歳)のプロフィール\n"
      "身長 = %5.1f cm, 体重 = %5.1f kg, BMI = %4.1f\n";
   char buffer[256];

   struct_input(&mydata);
   sprintf(buffer, format, mydata.name, mydata.age, mydata.bl, mydata.bw, mydata.bmi);
   printf(buffer);

   return 0;
}
```

```c
int struct_input(struct Data *p)
{
   char name[32];

   printf("氏名---");
   fgets(name, sizeof(name), stdin);
   if (strchr(name, '\n') != NULL) {
      name[strlen(name) - 1] = '\0';
   } else {
      while(getchar() != '\n');
   }
   strcpy(p->name, name);
   printf("年齢---");
   scanf("%d", &p->age);
   printf("身長 (cm)---");
   scanf("%lf", &p->bl);
   printf("体重 (kg)---");
   scanf("%lf", &p->bw);
   p->bmi = p->bw * 10000.0 / (p->bl * p->bl);

   return 0;
}
```

sprintf関数

書式にのっとって変換した出力を文字列に格納。

```
int sprintf(
    char *buffer,
    const char *format [, argument]...
);
```

bufferは出力の格納先。formatは書式指定文字列(printf関数と同じ)。

strchr関数

文字列に指定された文字が含まれるかチェックする。

```
char *strchr(
    const char *string,
    int c
);
```

インクルードファイルはstring.h。文字列stringの先頭から文字cを探し、最初に見つかった位置のポインタを返す。見つからなかった場合はNULLを返す。

strlen関数

文字列の長さを返す。

```
size_t strlen(
    const char *string
);
```

getchar関数

標準入力から文字を読み込む。

```
int getchar(void);
```

新しい型の定義

typedefキーワードを使えば、データ型の名前を自由に定義できる。

typedef 既存の型名 新しい型名;

新しい型の定義

```
typedef int Seisu;
```

とすれば、「int」の代わりに「Seisu」が使える。

新しい型の定義

```
typedef struct Data {
    int age;
    double bl;
    double bw;
} MyDataType, *lpMyDataType;
```

とすれば、Data型の構造体変数を宣言するとき、

```
MyDataType data;
```

と書ける。

新しい型の定義

```
typedef struct Data {
    int age;
    double bl;
    double bw;
} MyDataType, *lpMyDataType;
```

この構造体へのポインタを宣言するときは、

```
lpMyDataType lpdata;
```

と書ける。

自己参照的構造体

```
struct data {
    char name[32];
    int age;
    struct data *nextdata;
};
```

のように、構造体メンバには自分自身と同じ型へのポインタを持たせることができる。

自己参照的構造体

struct03.cというファイルを作成し、右のコードを書いて実行してみよう。

```
#include <stdio.h>

typedef struct Data {
   char name[32];
   int age;
   struct Data *nextdata;
} MyDataType;

int main()
{
   MyDataType a = {“佐藤”, 28, },
               b = {“田中”, 35, },
               c = {“鈴木”, 18, };

   MyDataType *lpdata;

   a.nextdata = &b;
   b.nextdata = &c;
   c.nextdata = NULL;

   for(lpdata = &a; lpdata; lpdata = lpdata->nextdata)
      printf(“%s, (%d 歳)\n”, lpdata->name, lpdata->age);

   return 0;
}
```

自己参照的構造体

- aのnextdataメンバにはbのアドレス
- bのnextdataメンバにはcのアドレス
- cのnextdataメンバにはNULL

を代入

```
#include <stdio.h>

typedef struct Data {
   char data3[32];
   int age;
   struct Data *nextdata;
} MyDataType;

int main()
{
   MyDataType a = {“佐藤”, 28, },
                  b = {“田中”, 35, },
                  c = {“鈴木”, 18, };

   MyDataType *lpdata;

   a.nextdata = &b;
   b.nextdata = &c;
   c.nextdata = NULL;

   for(lpdata = &a; lpdata; lpdata = lpdata->nextdata)
      printf(“%s, (%d 歳)\n”, lpdata->name, lpdata->age);

   return 0;
}
```

自己参照的構造体

for文の初期設定式でlpdataに構造体aのアドレスを代入

```
#include <stdio.h>

typedef struct Data {
   char data3[32];
   int age;
   struct Data *nextdata;
} MyDataType;

int main()
{
   MyDataType a = {“佐藤”, 28, },
                  b = {“田中”, 35, },
                  c = {“鈴木”, 18, };

   MyDataType *lpdata;

   a.nextdata = &b;
   b.nextdata = &c;
   c.nextdata = NULL;

   for(lpdata = &a; lpdata; lpdata = lpdata->nextdata)
      printf(“%s, (%d 歳)\n”, lpdata->name, lpdata->age);

   return 0;
}
```

自己参照的構造体

for文の継続条件式はlpdata。これがNULLになるとfor文が終了する。

```
#include <stdio.h>

typedef struct Data {
   char data3[32];
   int age;
   struct Data *nextdata;
} MyDataType;

int main()
{
   MyDataType a = {“佐藤”, 28, },
               b = {“田中”, 35, },
               c = {“鈴木”, 18, };

   MyDataType *lpdata;

   a.nextdata = &b;
   b.nextdata = &c;
   c.nextdata = NULL;

   for(lpdata = &a; lpdata; lpdata = lpdata->nextdata)
      printf(“%s, (%d 歳)\n”, lpdata->name, lpdata->age);

   return 0;
}
```

自己参照的構造体

for文の再設定式ではlpdataにnextdataメンバを代入。つまり、lpdataは次の構造体を指すようになる。

```
#include <stdio.h>

typedef struct Data {
   char data3[32];
   int age;
   struct Data *nextdata;
} MyDataType;

int main()
{
   MyDataType a = {“佐藤”, 28, },
                b = {“田中”, 35, },
                c = {“鈴木”, 18, };

   MyDataType *lpdata;

   a.nextdata = &b;
   b.nextdata = &c;
   c.nextdata = NULL;

   for(lpdata = &a; lpdata; lpdata = lpdata->nextdata)
      printf(“%s, (%d 歳)\n”, lpdata->name, lpdata->age);

   return 0;
}
```

今回の問い

- 構造体を宣言すると何ができるのか？
  - 構造体は異なるデータ型をひとまとめに扱うことができる。
- **構造体ポインタが指す構造体のメンバにアクセスするには、どのようにすれば良いか？**
  - **アロー演算子 (->)を用いる。**
- 共用体とは何か？

今回学ぶこと

1.C言語基本文法の復習
2.構造体の宣言
3.構造体へのポインタ
4.共用体

共用体 (union)とは

１つの記憶領域を確保して、そこに色々なデータ型のデータを格納できる機能を持つデータ型

共用体の宣言

```
union 共用体タグ{
    共用体メンバ;
    …;
} 共用体変数;
```

共用体の特徴

- 記憶できるのは１つのメンバだけ。後で別なメンバに値を代入すると、前のメンバの値は無効となる。
- 初期化はできない。

桑井康孝『猫でもわかるC言語プログラミング』
第３版、ソフトバンククリエイティブ

共用体

union.cというファイルを作成し、右のコードを書いて実行してみよう。

```
#include <stdio.h>

union MyUnion {
   int i;
   double d;
   char c;
   char *str;
};

int main()
{
   union MyUnion u;

   u.i = 200;
   printf(“u.i = %d\n”, u.i);

   u.d = 95.25;
   printf(“u.d = %f\n”, u.d);

   u.c = ‘A’;
   printf(“u.c = %c\n”, u.c);

   u.str = “ABCDE”;
   printf(“u.str = %s\n”, u.str);

   return 0;
}
```

今日の問い

- 構造体を宣言すると何ができるのか？
  - 構造体は異なるデータ型をひとまとめに扱うことができる。
- 構造体ポインタが指す構造体のメンバにアクセスするには、どのようにすれば良いか？
  - アロー演算子 (->)を用いる。
- **共用体とは何か？**
  - **１つの記憶領域を確保して、そこに色々なデータを格納できる機能を持つデータ型**

今回学んだこと

1.C言語基本文法の復習
2.構造体の宣言
3.構造体へのポインタ
4.共用体

今回の目標

構造体を定義してプログラムで使えるようになる。

課題

性別、年齢、身長 (cm)、体重 (kg)をメンバに持つ構造体を作成し、これらの値を入力すると基礎代謝量 (kcal/日)を計算して表示するプログラムbee.cを作成せよ。ただし、性別は男性をm、女性をfと入力させることとし、基礎代謝量は以下の式で計算できる。

男性：基礎代謝量=66+(13.7×体重)+(5×身長)−(6.8×年齢)

女性：基礎代謝量=655+(9.6×体重)+(1.7×身長) −(4.7×年齢)

提出先：yutaka.hirai@koeki-u.ac.jp

件名：データ構造とアルゴリズム第1回課題 [学籍番号]

提出物：bee.c

締め切り：12月3日 (木)

次回

第１回　構造体と共用体
第２回　探索とソート
第３回　メモリの動的確保と線形リスト
第４回　文字列の類似度計算、再帰、動的計画法
第５回　自由課題作成
第６回　自由課題作成
第７回　自由課題発表