C言語の基礎8

文字列とポインタ・配列

初期化の扱い

- 以下のように文字配列とポインタは似ている（同じように扱える）が…
  - コアダンプを吐く場合とうまくいく場合がある
  - なぜだろうか？

【array_pointer1.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    // 異なる方法での初期化
    char  astr[] = "Hashimoto";
    char  bstr[] = "Yoshimura";
    char *pstr   = bstr;

    // 出力する
    printf("%s\n",astr);
    printf("%s\n",pstr);

    // 変更と出力(OK)
    astr[0] = 'K';
    printf("%s\n",astr);

    // 変更と出力(OK)
    pstr[0] = 'K';
    printf("%s\n",pstr);

    return 0;
}
```

【array_pointer2.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    // 異なる方法での初期化
    char  astr[] = "Hashimoto";
```

```
    char *pstr   = "Yoshimura";

    // 出力する
    printf("%s\n",astr);
    printf("%s\n",pstr);

    // 変更と出力(OK)
    astr[0] = 'K';
    printf("%s\n",astr);

    // 変更と出力(NG)
    pstr[0] = 'K';
    printf("%s\n",pstr);

    return 0;
}
```

ポインタへの再代入

【string_pointer1.c】

```
#include <stdio.h>

int main(int argc, char const *argv[])
{
    // ポインタを初期化
    char *p1 = "Yoshimura";
    printf("%s\n", p1);

    // ポインタに再設定
    p1 = "Hashimoto";
    printf("%s\n", p1);

    return 0;
}
```

【string_pointer2.c】

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    // ポインタを初期化
    char *p1 = "Yoshimura";
    printf("%s\n", p1);
    printf("%p\n", p1);

    // ポインタに再設定
```

```
    p1 = "Hashimoto";
    printf("%s\n", p1);
    printf("%p\n", p1);

    return 0;
}
```

- これらのことから、何が分かるだろうか？
- 文字列を扱う場合、配列・ポインタのどちらが適切なのだろうか？

> https://qiita.com/Reed_X1319RAY/items/3b64d971f5c5cdc4d2fd
>
> https://qiita.com/jiku_jiku/items/72d0a7517bfe7fd9c62a

メモリ領域の違い

- 文字列リテラルは読み取り専用メモリ領域に配置される
- この領域への書き込みは未定義動作を引き起こす
- 文字配列は書き込み可能なメモリ領域に配置される

ポインタの再代入

- ポインタ変数は別の文字列リテラルのアドレスを指すように変更可能
- これは新しい文字列リテラルのアドレスを代入するだけ
- 元の文字列リテラルの内容は変更されない

文字列操作の安全性

- バッファオーバーフローを防ぐため、適切なサイズの配列を確保する
- 文字列操作時は終端文字（`\0`, `NULL`）の存在を確認する
- 標準ライブラリ関数を使用する場合は、バッファサイズを考慮する

文字列操作の推奨事項

- 文字列を変更する必要がある場合は文字配列を使用
- 文字列を参照するだけの場合はポインタを使用
- 可能な限り標準ライブラリ関数を使用する
- バッファサイズのチェックを忘れない

メモリ配置の一般的な例

```
アドレスの大きい方
+--------------------------+
|          スタック          | ← 自動変数、関数呼び出し時の情報など
|   (Stack - 自動的に増減)   |
+--------------------------+
|         未使用領域          |
|         (ガード領域)        |
+--------------------------+
|          ヒープ            | ← malloc/new による動的確保領域
|   (Heap - 手動で管理)      |
+--------------------------+
|       BSSセクション         | ← 初期化されていない静的変数など
|  (.bss - ゼロ初期化)        |
+--------------------------+
|      データセクション       | ← 初期化された静的/グローバル変数
|    (.data - 初期値あり)     |
+--------------------------+
|      テキストセクション     | ← プログラムの実行命令（コード）
|     (.text - 実行可能)     |
+--------------------------+
アドレスの小さい方
```

構造体

構造体の宣言と初期化

- ひとつの名前でまとめられた、異なる型の変数の集まり
- 一緒に格納された複数の値のグループで、各値は構造体のフィールド（またはメンバー）と呼ばれる
- 構造体を使用して、ユーザー定義のデータ型を作成できる
- C言語はオブジェクト指向言語ではないが、構造体を利用することで、オブジェクト指向的な設計も可能

構造体の例

```c
struct Person {
  char *name;
  int age;
  int height;
};
```

- 構造体 `Person` を定義する。
    - メンバー変数は　名前、年齢、身長を格納できる
- 構造体 `Person` を格納する変数 `p1` を宣言する場合

```
struct Person p1;
```

- 構造体のメンバーに値を格納する場合

```
    p1.age = 20;
    p1.height = 170;
```

- 宣言と初期化を行う場合

```
    struct Person p1 = {"Whoami", 30, 175};
```

構造体とポインタ

- 構造体メンバーを利用する場合は `.` （ピリオド）を使用する
- ポインタを介して構造体メンバーを利用する場合は、 `->` （アロー演算子）を使用する

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

// 構造体の定義
struct Person {
    char name[50];
    int age;
};

int main() {
    // 構造体のインスタンス化と初期化
    struct Person person1;
    strcpy(person1.name, "John");
    person1.age = 25;

    // ポインタを使用して構造体にアクセス
    struct Person *personPtr = &person1;

    // ポインタを介して構造体メンバにアクセス
    printf("Name: %s\n", personPtr->name);
    printf("Age: %d\n", personPtr->age);
```

```
    return 0;
}
```

- リスト構造

  > https://daeudaeu.com/list-structure/

- 木構造

  > https://daeudaeu.com/bintree/

様々なアルゴリズムを実装する場合に、構造体（struct）は、よく使用される。

練習

以下のプログラムを作成してください。サンプルを動作させるmain関数も作成してください。

- 問題1: 以下の構造体を定義し、ポインタを使用してメンバにアクセスしてください（prog1.c）
  - 構造体Pointのx,yに値を代入し、出力する

```
struct Point {
    int x;
    int y;
};
```

- 問題2: 以下の構造体を定義し、ポインタを使用して関数内で構造体のメンバを変更してください（prog2.c）
  - 関数 `doubleSize` を実装し、与えられた `Rectangle` 構造体の幅と高さを2倍に更新する
  - 関数 `diagonal` を実装し、与えられた `Rectangle` 構造体の幅と高さから対角線の長さ計算する

```
struct Rectangle {
    int width;
    int height;
};
```

```
void  doubleSize(struct Rectangle *rect);
float diagonal(struct Rectangle rect);
```

【prog2.c】

```
int main(int argc, char const *argv[]) {
    struct Rectangle obj1;
    obj1.width  = 50;
    obj1.height = 100;

    doubleSize(&obj1);

    printf("width : %d\n", obj1.width);
    printf("height : %d\n", obj1.height);

    printf("diagonal : %f\n", diagonal(obj1));

    return 0;
}
```

補足

- ライブラリを使用することが今後増えると思いますが、以下のような点に注意してください
  - 数学関数を `#include <math.h>` で利用する場合、コンパイル時に `-lm` をオプションに付ける

    ```
    $ gcc prog1.c -lm
    ```

  - 数学関数ライブラリ `libm.so` は、標準ライブラリ `libc` に含まれていないため、リンカに対して明示する必要がある

    ```
    $ gcc hoge.c -lxxxx
    ```

    - hoge.cをコンパイル→リンクするときに、 `libxxxx.so` を使用するという意味
    - 接頭辞 `lib` と、接尾辞 `.so` は、省略する

- math.hを利用したプログラムのコンパイル
  【math_err.c】

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

int main(int argc, char const *argv[]) {
    float a = 2.0;
    printf("sqrt(%f)=>%f", a, sqrt(a));
    return 0;
}
```

```
$ gcc math_err.c
/usr/bin/ld: /tmp/ccBVRyVx.o: in function `main':
math_err.c:(.text+0x34): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

    - `ld`とは、リンカプログラムのこと。

- -lxxxxを使用しなくても、コンパイル可能なもの(抜粋)

ヘッダファイル	説明	例
<stdio.h>	標準入出力ライブラリ	`printf`, `scanf`, `fopen`
<stdlib.h>	標準ライブラリユーティリティ	`malloc`, `free`, `atoi`, `rand`
<string.h>	文字列操作	`strcpy`, `strlen`, `strcmp`, `strcat`
<ctype.h>	文字の分類と変換	`isdigit`, `isalpha`, `tolower`, `toupper`
<math.h> ※	数学関数	`sqrt`, `sin`, `cos`, `tan` ※ 環境により-lmが不要
<time.h>	時刻と日付	`time`, `difftime`, `strftime`
<limits.h>	各種定数の制限値	`INT_MAX`, `CHAR_BIT`
<float.h>	浮動小数点の制限値	`FLT_MAX`, `DBL_MIN`
<assert.h>	アサーション（実行時検証）	`assert` マクロ
<errno.h>	エラーナンバー	`errno` 変数, `EDOM`, `ERANGE`
<signal.h>	シグナル処理	`signal`, `raise`, `SIGKILL`
<stddef.h>	標準型定義	`size_t`, `NULL`, `offsetof`
<stdbool.h>	ブール型	`bool`, `true`, `false`
<stdint.h>	固定幅整数型	`int8_t`, `int16_t`, `int32_t`
<iso646.h>	代替演算子	`and`, `or`, `not`

構造体の配列とポインタ

- 構造体の配列をポインタで操作する例を確認してください
    - どのような動作を行ない、メモリ上のどこを指しているのか？　を意識して読んでください

【struct_4.c】

```c
#include <stdio.h>

struct Student {
    int no;
    char name[20]; // 氏名
    float height;  // 身長
    float weight;  // 体重
};
```

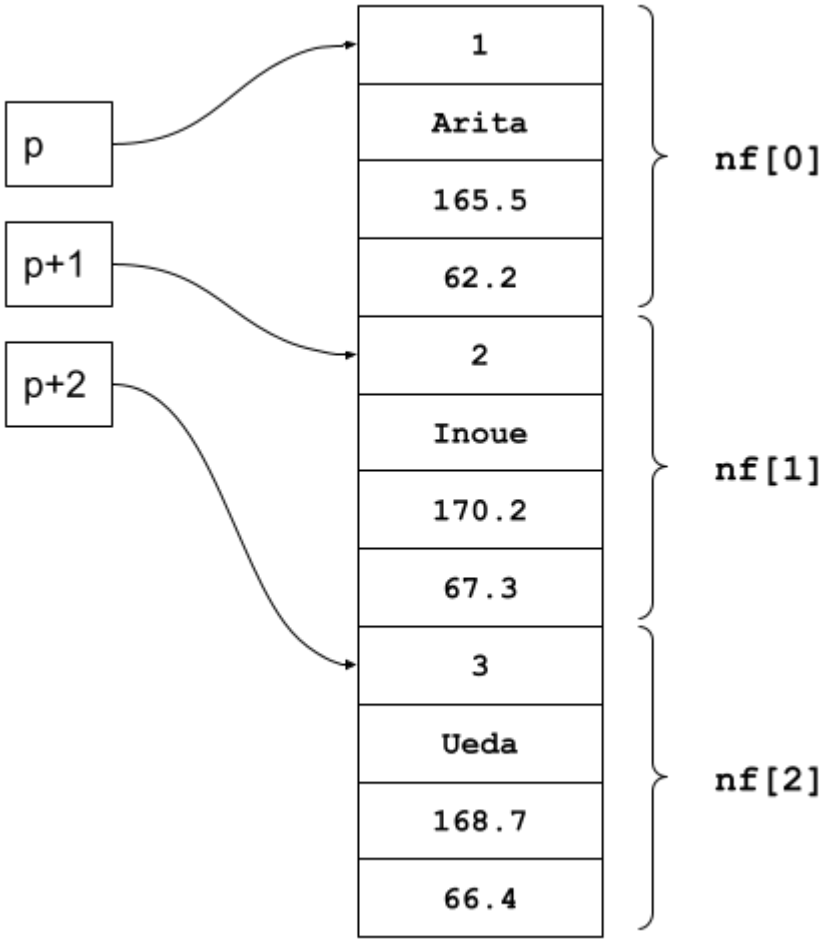
```c
int main(int argc, char const *argv[]) {
    struct Student nf[] = {
        {1, "Arita", 165.5, 62.2},
        {2, "Inoue", 170.2, 67.3},
        {3, "Ueda", 168.7, 66.4},
    };

    struct Student *p = nf;

    for (int i = 0; i < (sizeof(nf) / sizeof(struct Student)); i++) {
        printf("%d : %s\n", (p + i)->no, (p + i)->name);
    }

    return 0;
}
```

- 図示すると、以下のようなイメージになる



typedefの利用

- キーワード `typedef` は、既に定義されている型に，別の新しい名前をつけて定義することができる
    - 型(type)定義(definition)の省略形
- 定義の仕方

```
typedef　定義されている型　定義する新しい型名;
```

- 例えば、doubleという型を別の名前で定義することができる

```
typedef double dob;
```

    - 以降 `dob` はdoubleと同じ扱いとなる。

> https://daeudaeu.com/c-typedef/
>
> よく読んで、理解してほしい。

- これを構造体の定義と一緒に使うことで、記述が簡単になる
    - 先の `struct_4.c` をtypedefを使用して書き直してみる

```c
#include <stdio.h>

typedef struct {
    int no;
    char name[20]; // 氏名
    float height;  // 身長
    float weight;  // 体重
} Student;  // Student型を定義

int main(int argc, char const *argv[]) {
    Student nf[] = {
        {1, "Arita", 165.5, 62.2},
        {2, "Inoue", 170.2, 67.3},
        {3, "Ueda", 168.7, 66.4},
    };

    Student *p = nf;

    for (int i = 0; i < (sizeof(nf) / sizeof(Student)); i++) {
        printf("%d : %s\n", (p + i)->no, (p + i)->name);
    }
```

```
    return 0;
}
```

- 構造体を使用する場合、ほとんどの場合 `typedef` を用いて、型定義を一緒に行う
    - 記述が簡単になり、読みやすくなる

練習

- 先の練習で作成した、 `prog1.c` 、 `prog2.c` を `typedef` を用いて書き直してください
    - プログラム名は同一でOK

- 与えられた構造体の配列を受け取り、配列内の要素の中でxの最大値を持つ要素のポインタを返す関数 `findxMax` を実装してください
  関数は次の通りです。(prog3.c)

```
struct Point {
    int x;
    int y;
};
```

```
struct Point *findxMax(struct Point *points, int size);
```

- prog3.cをtypedefを使用して書き換えてください（prog4.c）

課題

以下の構造体を作成してください。

- 以下の情報を表す構造体 Book を定義し、それを使用して本の情報を入力し、表示するプログラムを作成してください（prog5.c）

```
本のタイトル （文字列）
著者の名前 （文字列）
出版年 （整数）
価格 （実数）
```

- 構造体の定義は、`typedef`を使用してください
    - 文字列は100文字まで格納できるようにしてください
- 関数mainだけでOK

【実行例】

```
$ ./a.out
本の情報を入力してください
タイトル : C言語の練習
著者名 : yoshimura
出版年 : 2023
価格 : 1980

== 本の情報 ==
タイトル: C言語の練習
著者名: yoshimura
出版年: 2023
価格  : 1980.00 円
```

- 上記プログラムで、複数の書籍が管理できるよう、構造体の配列を用意し動作するようにしてください
    - あらかじめ、最大20冊管理できるようにする
    - 3冊分入力し、その後、すべてのデータを出力するようにする(prog5_2.c)

【実行例】

```
$ ./a.out
No:1 本の情報を入力してください (@QUITで終了)
タイトル : あいう
著者名 : えお
出版年 : 2001
価格 : 100
No:2 本の情報を入力してください (@QUITで終了)
タイトル : かきく
著者名 : けこ
出版年 : 2002
価格 : 200
No:3 本の情報を入力してください (@QUITで終了)
タイトル : さしす
著者名 : せそ
出版年 : 2003
価格 : 300
No:4 本の情報を入力してください (@QUITで終了)
タイトル : @QUIT
```

```
== 本の情報 ==
タイトル: あいう
著者名: えお
出版年: 2001
価格  : 100.00 円

== 本の情報 ==
タイトル: かきく
著者名: けこ
出版年: 2002
価格  : 200.00 円

== 本の情報 ==
タイトル: さしす
著者名: せそ
出版年: 2003
価格  : 300.00 円
```

以下の構造体を利用したプログラムを作成してください

- 以下の情報を表す構造体 `Rectangle` を定義する
  - 構造体 `Rectangle` を使用して長方形の情報を入力し、面積と周囲の長さを計算して表示するプログラム（prog6.c）
  - 長方形の幅 (実数)
  - 長方形の高さ (実数)
- 関数名は次の通りとする
  - 面積を求める関数：`calcArea()`
  - 周囲の長さを求める関数：`calcPerimeter()`
- 動作を確認するためのmain関数も作成してください