コンピュータサイエンスとプログラミング 演習課題

構造体（1）

構造体

- 複数の異なる種類のデータを 1 つにまとめて扱えるようにする

構造体型の宣言

- 構造体がどのような型のデータから構成されるかを宣言し，構造体タグの名前を持つ構造体型が利用できるようにする
  - 構造体タグ： この宣言を参照するための名前
  - メンバ： 構造体を構成する要素

```
struct 構造体タグ{
   型名  メンバ名 1;
   型名  メンバ名 2;
   ……
};
```

```
struct point{
   int x; /* x 座標 */
   int y; /* y 座標 */
};
```

```
struct temperature{
   int date;           /* 日 */
   double temerature; /*温度*/
};
```

構造体の定義

```
struct 構造体タグ  構造体変数名;
```

```
struct point p1, p2;
```

構造体の初期化

```
struct 構造体タグ  構造体変数名 = {メンバ 1 の初期値, メンバ 2 の初期値, …};
```

```
struct point p1 = {1, -1};
```

※初期値の個数がメンバの個数よりも少なかった場合は，残りのメンバは整数型では 0，浮動小数点型では 0.0 で初期化される．

メンバの参照

```
構造体変数名. メンバ名
```

「.」はメンバ参照演算子

（例）構造体型 **point** の変数 **p1** から原点までの距離

```
double d = sqrt(p1.x * p1.x + p1.y * p1.y);
```

メンバに値を代入

```
構造体変数名. メンバ名 = 式;
```

（例）

```
p1.x = p1.x + 10;
p1.y = p1.y + 10;
```

メンバのアドレスの取得

```
&構造体変数名. メンバ名
```

（例）

```
scanf("%d %d", &p1.x, &p1.y);
```

【例題 5-1】構造体のメンバを参照するプログラム

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

struct point{
    int x;      /* x 座標 */
    int y;      /* y 座標 */
};

int main(void){
    struct point p1;
    double d;
    printf("> (x, y) ");
    scanf("%d %d", &p1.x, &p1.y);
    d = sqrt(p1.x * p1.x + p1.y * p1.y);
    printf("Distance between (%d, %d) and (0, 0) is %.2f.¥n", p1.x, p1.y, d);
    return 0;
}
```

- 点の座標を入力し，原点からの距離を計算するプログラム
- **sqrt** 関数を用いるために **math.h** をインクルード
- コンパイル時に-lm のオプション ※注

構造体の比較

構造体を比較するためには，それぞれのメンバ同士を比較する必要がある．

（例） 2 点が同一の点かどうかを調べる

```
if (p1.x == p2.x && p1.y == p2.y){
        2 点が同一の点である時の処理
}
```

【例題 5-2】 構造体のメンバを比較するプログラム

入力された点の座標が予め決められた点と同一かどうかを判定する

```
#include <stdio.h>

struct point{
    int x;      /* x 座標 */
    int y;      /* y 座標 */
};

int main(void){
     struct point p1 = {3, 2};
     struct point p2;
     while(scanf("%d  %d", &p2.x, &p2.y) != EOF){
       if (p1.x == p2.x && p1.y == p2.y){
              printf("(%d, %d) is correct.¥n", p2.x, p2.y);
              return 0;
       } else{
              printf("(%d, %d) is wrong.¥n", p2.x, p2.y);
              }
       }
       return 0;
}
```

ファイルやキーボードから複数のデータを入力する方法

```
struct point p1;

while(scanf("%d  %d", &p1.x, &p1.y) != EOF){
                構造体 p1 の処理
}
```

data.txt の内容の例

```
1 1
2 1
2 2
3 1
3 2
```

- 入力リダイレクト機能

```
% ./a.exe < data.txt
```

  - ファイル **data.txt** の先頭から 1 行ずつデータを入力し，**scanf** 関数の実引数に格納される．
  - ファイルの最後に到達すると，**scanf** 関数は **EOF (End of File)** を返すので，**while** 文の繰り返しが終了する．

- キーボードからのデータ入力
  - データの入力を終了する時に **Ctrl-D** を入力すると， **scanf** 関数は EOF を返すので，**while** 文の繰り返しが終了する．

構造体のコピー

```
struct point p1, p2={3, 2};

p1 = p2;
```

それぞれのメンバの値を代入するのと同じ

```
p1.x = p2.x;
p1.y = p2.y;
```

配列型のメンバを持つ構造体（例）

- 宣言

```
struct callinfo{
    int history[31];  /* 日ごとの通話時間 */
    int total;        /* 通話時間の合計 */
};
```

- 初期化

```
struct callinfo c1 = {{20, 31, 12}, 0};
```

  配列型のメンバの初期値の個数が配列の大きさよりも少ない時は，配列の残りの要素には 0 が代入される．上の場合は，配列型のメンバ **history** の 3 番目以降の配列要素は全て 0 になる．

```
struct callinfo c1 = {{0}, 0};
```

  メンバ **history** の全ての配列要素とメンバ **total** を 0 で初期化

- 参照

```
x = c1.history[i];
```

- 代入

```
c1.history[i] = 12;
```

- アドレスの取得

```
&c1.history[i]
```

  メンバ参照演算子はアドレス演算子よりも優先順位が高い．

文字列のメンバを持つ構造体（例）

- 宣言

```
struct callinfo2{
    char id[9];       /* 利用者の名前(英数 8 文字) */
    int history[31]; /* 日ごとの通話時間 */
    int total;        /* 通話時間の合計 */
};
```

配列の大きさは，文字列の終端を示す空文字 '¥0' の分を加えて，
格納する文字列の最大長＋1
を指定する

- 初期化

```
struct callinfo2 c1 = {"abcd1234", {0}, 0};
```

- 代入

```
char id[9] = "xyz98765";

strcpy(c1.id, id);   /* 文字列 id を c1.id にコピー*/
```

<string.h>をインクルードする

- 比較

```
if (strcmp(c1.id, id) == 0){
        c1.id と id が等しい時の処理
}
```

strcmp 関数は, 文字列が等しい時に 0 を返す.

構造体の配列

```
struct 構造体タグ　配列名[配列の大きさ];
```

（例）

```
struct point table[100];
```

構造体型 **point** を要素の型とする大きさが 100 の配列 **table** の定義

```
sum += table[i].x;                          参照
table[i].x = (table[i-1].x + table[i+1].x)/2;   代入
```

【課題 5-1】

構造体型 **point** の次の 2 次元配列を使って，入力した点がどの象限に含まれるかを数えるプログラムを作成せよ. ただし, 入力データの最大個数は 100 とする. 座標軸上の点はどの象限にも属さないとして, **table[0]** に格納するものとする．また，下のような実行例を想定する.

```
struct point table[5][100];
```

- 実行例

```
% ./a.exe < point-data.txt
The number of I quadrant (RUQ)    1
( 5,  1)
The number of II quadrant (LUQ)    1
(-6,  2)
The number of III quadrant (LLQ)    2
(-7, -3)
(-8, -3)
The number of IV quadrant (RLQ)    3
( 1, -4)
( 9, -4)
( 8, -4)
```

point-data.txt の内容の例

```
  5   1
  0  91
 -6   2
-92   0
 -7  -3
  0 -93
 94   0
  1  -4
  9  -4
 -8  -3
  8  -4
```

構造体の typedef 宣言

```
typedef struct 構造体タグ 新しい型名;
```

（例）構造体型 **point** に対応する **POINT** という型の宣言

```
struct point{
    int x;  /* x 座標 */
    int y;  /* y 座標 */
};

typedef struct point POINT;
```

→ 構造体型の宣言と **typedef** 宣言をまとめた書き方

```
typedef struct point{
    int x;  /* x 座標 */
    int y;  /* y 座標 */
}POINT;
```

（例）**POINT** 型の変数 **p1** の定義

```
POINT p1;
```

変数 **p1** は **POINT** 型の変数と呼ばれる

構造体の引数

（例） 例題 5-1 の原点から点 **p1** までの距離を計算する処理を **distance** 関数として定義

- 構造体型 **point** を引数に指定して **distance** 関数を宣言

```
double distance(struct point p1);
```

いちいち構造体型 **point** を使うと関数の宣言が長くなるので，**typedef** 名の **POINT** 型を使う

- **typedef 名の POINT** 型を使って **distance** 関数を宣言

```
double distance(POINT p1);
```

- **distance** 関数の定義

```
double distance(POINT p1){
        return sqrt(p1.x * p1.x + p1.y * p1.y);
}
```

- **distance** 関数の呼び出し

```
POINT pt1 = {2, 4};
double d;

d = distance(pt1);
```

実引数に **POINT** 型の構造体を指定する

【課題 5-2】

例題 5-1 のプログラムでは，**main** 関数で処理をしていた．上の **distance** 関数を用いて，例題 5-1 のプログラムを書き換え，動作を確認せよ

レポートに関して

- ✧ 課題 5-1，5-2 を実施し，レポート課題として提出すること（提出期限：2021 年 05 月 27 日 09:00）
- ✧ CLE で提出する際のファイル名（半角英数）は以下の通りとする（XXXX は学籍番号下 4 桁）．
  - 課題 5-1 の場合，XXXX-kadai5-1.c
  - 課題 5-2 の場合，XXXX-kadai5-2.c