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 (2)

構造体を返す関数 （例）

- 点 **p1** を (**dx, dy**) だけ移動した点を返す **movePoint** 関数の宣言
  - 返却値の型に **POINT** 型を指定

```
POINT movepoint(POINT p1, int dx, int dy);
```

- **movePoint** 関数の定義

```
POINT movePoint(POINT p1, int dx, int dy){
        POINT p2;

        p2.x = p1.x + dx;
        p2.y = p2.y + dy;
        return p2;
}
```

- 引数と同じように，関数の返却値にも構造体が使える
- 点 **p1** を **(dx, dy)** 移動した点を返す関数
- **POINT** 型の局所変数 **p2** を定義
- 移動後の点を **p2** に格納

- 関数の呼び出し

```
POINT pt1 = {3, 7};
POINT pt2;
pt2 = movePoint(pt1, 2, 5);
```

- **POINT** 型の変数 **pt2** をさらに定義
- **movePoint** 関数の返却値を変数 **pt2** にコピー

【例題 6-1】 構造体を返す関数のプログラム例

```
#include <stdio.h>
/* 入力された点の座標を移動するプログラム */
typedef struct point{
     int x;      /* x 座標 */
     int y;      /* y 座標 */
} POINT;

POINT movePoint(POINT p1, int dx, int dy){
     POINT p2;

     p2.x = p1.x + dx;
     p2.y = p1.y + dy;
     return p2;
}

int main(void){
     POINT pt1;                         /* 移動前の座標 */
     POINT pt2;                         /* 移動後の座標 */
     int dx, dy;
     printf("> (x1, y1) ");
     scanf("%d %d", &pt1.x, &pt1.y);       /* pt1 を入力 */
     printf("> (dx, dy) ");
     while(scanf("%d  %d", &dx, &dy) != EOF){
          pt2 = movePoint(pt1, dx, dy);
          printf("Original: (%d, %d), Shift: (%d, %d), Result: (%d, %d).¥n",
               pt1.x, pt1.y, dx, dy, pt2.x, pt2.y);
          pt1 = pt2;          /* 構造体 pt2 を pt1 にコピー */
          printf("> (dx, dy) ");
     }
     return 0;
}
```

【課題 6-1】

POINT 型（例題 6-1 参照）の 2 つの引数の中点を返す **getMiddlePoint** 関数を次のように定義し，入力した 2 点の中点を出力するプログラムを作成せよ．ただし，下のような実行例を想定している．

```
POINT getMiddlePoint(POINT p1, POINT p2);
```

実行例

```
% ./a.out
> (x1, y1) 1 2
> (x2, y2) 5 8
Midpoint between (1, 2) and (5, 8) is (3, 5).
```

構造体へのポインタ引数

- 「ポインタ」で学んだように，関数が呼び出された時，実引数の値が仮引数にコピーされてから関数が実行される．引数が構造体の場合も同様.
- 関数内で仮引数の構造体のメンバの値を変更しても，実引数のメンバの値は変化しない.

（例）

```
void moveVpoint(POINT p1, int dx, int dy){
        p1.x = p1.x + dx;
        p1.y = p1.y + dy;
}
```

関数内で仮引数 **p1** のメンバの値を変更しているが，実引数のメンバの値は変化しない.

- 実引数の構造体のメンバの値を関数で変更したい時は，構造体へのポインタを引数に指定

```
void movePpoint(POINT *p1, int dx, int dy);
```

- 構造体へのポインタを使用した時，そのメンバを参照するには，メンバ参照演算子「**.**」ではなく，間接メンバ演算子「**->**」を用いる.

```
構造体へのポインタ->メンバ名
```

（例）構造体へのポインタを使って **moveVpoint** を書き換えると

```
void movePpoint(POINT *p1, int dx, int dy){
        p1->x = p1->x + dx;
        p1->y = p1->y + dy;
}
```

- 構造体のアドレスを実引数に指定して，次のように **movePpoint** 関数を呼ぶと，構造体 **pt1** の値は **(5, 12)** に変更される.

```
POINT pt1 = {3, 7};
movePPoint(&pt1, 2, 5);
```

（参考）

- 変数 **p1** が構造体へのポインタの時，間接参照演算子「*****」を使うと，***p1** で構造体自身を参照できるので，メンバ **x** は，**(*p1).x** でも参照できる.
- しかし，括弧を使ったりして複雑なので，一般には，**p1->x** と書く.

> メンバ参照演算子「**.**」の優先順位は，間接参照演算子「*****」より高いため，***p1.x** は ***(p1.x)** の意味になる．そのため，**(*p1).x** と書かなければならない.

【例題 6-2】 構造体へのポインタ引数をもつ関数の例

```
#include <stdio.h>

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

POINT movePPoint(POINT *p1, int dx, int dy){
        p1->x = p1->x + dx;
        p1->y = p1->y + dy;
}

int main(void){
        POINT pt1;                      /* 移動後の座標 */
        POINT pt2;                      /* 移動前の座標 */
        int dx, dy;
        printf("> (x1, y1) ");
        scanf("%d %d", &pt1.x, &pt1.y);     /* pt1を入力 */
        printf("> (dx, dy) ");
        while(scanf("%d  %d", &dx, &dy) != EOF){
            pt2 = pt1;
            movePPoint(&pt1, dx, dy);
            printf("Original: (%d, %d), Shift: (%d, %d), Result: (%d, %d). ¥n",
               pt2.x, pt2.y, dx, dy, pt1.x, pt1.y);
            printf("> (dx, dy) ");
        }
        return 0;
}
```

pt1 と pt2 の役割が例 6-1 と逆になっていることに注意

【課題 6-2】

原点を中心として，点 **p1** の位置を時計の針と逆方向に **dt** 度回転する **rotatePoint** 関数を下のように定義し，入力した点を順に回転させるプログラムを作成せよ．ただし，回転角度 **dt** はラジアンではなく度とし，右のような実行例を想定するものとする．

```
void rotetePoint(POINT *p1, int dt);
```

```
% ./a.out
> (x1, y1) 10 10
> (dt) 45
(0, 14)
> (dt) 45
(-10, 10)
> (dt) 180
(10, -10)
> (dt) -30
(4, -14)
> (dt) -60
(-10, -10)
```

動的メモリ割り付け (Dynamic memory allocation)

- 任意の大きさのメモリ領域を，OS が管理するメモリ（ヒープと呼ばれる）から確保する機能
- 次の 2 つの関数をペアで用いる （**stdlib.h** をインクルード）

```
void *malloc(size_t size);
```

  - 大きさ **size** バイトのメモリ領域を確保し，その先頭アドレスを返す．
  - メモリ領域を確保できなかった時は **NULL** を返す．

```
void free(void *ptr);
```

  - ポインタ **ptr** が指すメモリ領域を解放し，ヒープへ戻す．

- **malloc** 関数の引数で指定するバイト数は，**sizeof** 演算子を使って計算

```
size_t dsize = sizeof(double);
```

size_t 型は非負の整数を表す型 **(unsigned int)**

- 動的メモリ割り付けの例
  - **int** 型のデータに必要なメモリ領域の確保

```
int *p1;

p1 = malloc(sizeof(int));
if (p1 == NULL){
     fprintf(stderr, "Memory Shortage¥n");
     exit(1);
}
*p1 =0;      /* p1 が指すメモリ領域を初期化 */
```

- **int** 型のデータに必要なメモリ領域を **malloc** 関数を使ってヒープから確保
- その先頭アドレスをポインタ **p1** に代入
- **int** 型以外のメモリ領域を使いたい時は，下線の部分を書き換える
- **NULL** はマクロで，どんな変数へのポインタの値とも等しくならない定数で 0 と定義.

【例題 6-3】動的メモリ割り付けの例 (1)

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

int main(void){
     int x = 19;
     int *p1;
     int *p2;

     p1 = &x;

     p2 = malloc(sizeof(int));
     if (p2 == NULL){
     fprintf(stderr, "Memory Shortage¥n");
     exit(1);
     }

     *p2 = 77;

     printf("HENSU ¥t VALUE ¥t INDIRECT¥n");
     printf("p1 ¥t %10p ¥t %d¥n", p1, *p1);
     printf("p2 ¥t %10p ¥t %d¥n", p2, *p2);
     free(p2);
     return 0;
}
```

- 各値と間接参照される値を出力する
  - 局所変数を指すポインタ （**p1**）
  - **malloc** 関数によって確保したメモリ領域を指すポインタ （**p2**）
- **p1** には局所変数 **x** のアドレスを代入
- **p2** には **malloc** 関数で確保したメモリ領域を初期化
- **p2** が示すメモリ領域は解放しなければならない

【例題 6-4】動的メモリ割り付けの例 (2)

- 任意の個数のコマンドライン引数の平均を計算

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

int main(int argc, char *argv[]){
    int i, num;
    double sum = 0.0;
    double *table;

    if (argc < 2){
        fprintf(stderr, "Arguments are not specified¥n");
        exit(1);
    }

    num = argc -1;
    table = malloc(sizeof(double) * num);
    if (table == NULL){
        fprintf(stderr, "Memory Shortage¥n");
        exit(1);
    }

次のページに続く

    for (i = 0; i < num; i++){
        *(table + i) = atof(argv[i + 1]);
    }
```

- 変数 **table** は配列の先頭要素を指すためのポインタ
- コマンドライン引数で指定した実数の個数を **num** に代入
- **malloc** 関数を使って確保した配列用のメモリ領域の先頭アドレスをポインタ **table** に格納
- ポインタ **table** が指す配列の **i** 番目の要素は，***(table + i)** または

```
    for (i = 0; i < num; i++){
        sum += table[i];
    }

    printf("Average of %d doubles is %.2f. ¥n",
                          num, sum / num);
        free(table);
        return 0;
}
```

table[i]で参照できる.

- **i+1** 番目のコマンドライン引数を実数に変換し，**table** が指す配列の **i** 番目の要素に格納

構造体へのポインタを返す関数　（例）

- 例題 6-1 の **movePoint** を変更して，構造体へのポインタを返すように **moveMPoint** を定義する
- **malloc** 関数で確保したメモリ領域のアドレスをポインタ **newP** に代入
- 移動した点の座標を **newP** が指す構造体へ格納
- **return** 文で **newP** を返す

【例題 6-5】　例題 6-1 のプログラムを，**moveMPoint** 関数を使うように変更する

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

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

POINT *moveMPoint(POINT p1, int dx, int dy){
     POINT *newP;

     newP = malloc(sizeof(POINT));
     if (newP == NULL){
        fprintf(stderr, "Memory Shortage ¥n");
        exit(1);
     }
     newP->x = p1.x + dx;
     newP->y = p1.y + dy;
     return newP;
}



int main(void){
     POINT pt1;              /* 移動前の座標 */
     POINT *pt2;             /* 移動後の座標 */
     int dx, dy;
     printf("> (x1, y1) ");
     scanf("%d %d", &pt1.x, &pt1.y);       /* pt1 を入力 */
     printf("> (dx, dy) ");
     while(scanf("%d  %d", &dx, &dy) != EOF){
         pt2 = moveMPoint(pt1, dx, dy);
         printf("Original: (%d, %d), Shift: (%d, %d),
               Result: (%d, %d).¥n",
               pt1.x, pt1.y, dx, dy, pt2->x, pt2->y);
         pt1 = *pt2;
```

- **pt2** を構造体へのポインタとして宣言
- 構造体 **pt1** を **(dx, dy)** 移動した点を生成し，その先頭アドレスを **pt2** に代入する

```
        free(pt2);
        printf("> (dx, dy) ");
    }
    return 0;
}
```

【課題 6-3】

例題 6-4 のプログラムで，**malloc** 関数で配列用のメモリ領域を確保し，コマンドライン引数の値を格納する処理を次の関数として定義し，プログラムを作成・実行して動作を確認せよ．ただし，**makeArray** 関数は **int** 型の配列の先頭を返すものとし，コマンドライン引数では整数を指定するように，**int** 型の配列に変更する．

```
int *makeArray(int argc, char *argv[]);
```

なお，次のような実行例を想定する．

```
% ./a.out  13  22  36  41  55
Average of 5 doubles is 33.40.
```

レポートに関して

- ✧ 課題 6-1，6-2，6-3 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 06 月 03 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 6-1 の場合，XXXX-kadai6-1.c
  - 課題 6-2 の場合，XXXX-kadai6-2.c
  - 課題 6-3 の場合，XXXX-kadai6-3.c

Appendix. コマンドライン引数の利用

入力したいデータを実行時に指定できるようにする.

```
% ./a.out a1 22 33.3
```

main 関数を次のように定義する

```
int main(int argc, char *argv[]);
```

argc 個のプログラム引数が **argv** が示す配列に格納されて，**main** 関数が呼ばれる.

【例題 A1】 コマンドライン引数を出力する例

```
#include <stdio.h>
int main(int argc, char *argv[]){
    int i;

    printf("argc = %d¥n", argc);
    for (i=0; i < argc; i++)
        printf("argv[%d] ¥"%s¥"¥n", i, argv[i]);
    return 0;
}
```

printf 関数

- 書式 **%s** は文字列を出力するため
- **¥"** は二重引用符自身を出力するため

実行例

```
% ./a.out a1 22 33.3
argc = 4
argv[0] "./a.exe"
argv[1] "a1"
argv[2] "22"
argv[3] "33.3"
```

コマンドライン引数を数値として利用

```
int atoi(const char *str);
```

- 文字列 **str** を **int** 型の数値に変換した値を返す

```
double atof(const char *str);
```

- 文字列 **str** を **double** 型の数値に変換した値を返す

➢ 注意

- 関数や関数を使う時は，**stdlib.h** をインクルードする
- キーワード **const** は引数の値を関数内で変更しないことを示す

【例題 A2】 コマンドライン引数を数値として利用する例

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
     int x;
     double y;
     x = atoi(argv[2]);      /* int 型に変換*/
     y = atof(argv[3]);      /* double 型に変換*/
     printf("argv[2] %d¥n", x);
     printf("argv[3] %f¥n", y);
     return 0;
}
```

実行例

```
% ./a.out a1 22 33.3
argv[2] 22
argv[3] 33.300000
```