

プログラミング演習 第 4 回

問 A [04A]: 関数の実装

図 1 のプログラムにおいて、スタックに要素を追加する関数 `push` とスタックから要素を取り出す関数 `pop` をプログラミング通論の講義内容に沿って実装し、以下の実行例のように出力されるよう、プログラムを完成させよ。`main` 関数の内容を変更してはいけない。スタックのオーバーフローやアンダーフローに対するエラー処理を含めること。

また、`push`、`pop` の引数において、`stack` 構造体はポインタとして受け取っている。一方、構造体を表示する関数 `print_stack` ではポインタではない。`push`、`pop` の引数を「`stack` 構造体」ではなく、「`stack` 構造体のポインタ」とする理由を考察し、レポートに含めよ。

実行例 04A

```
[1] # push 1
[1, 2] # push 1
[1, 2, 3] # push 1
[1, 2, 3, 4] # push 1
[1, 2, 3, 4, 5] # push 1
[1, 2, 3, 4] # pop 1
[1, 2, 3] # pop 2
[1, 2, 3, 9] # push 2
[1, 2, 3] # pop 1
[1, 2] # pop 2
[1, 2, 12] # push 2
[1, 2] # pop 1
[1] # pop 2
[1, 14] # push 2
[1] # pop 1
[] # pop 2
[15] # push 2
```

```

#include <stdio.h>
#include <stdlib.h>
#define STACKSIZE 10 // stack の大きさ
typedef int elemtype; // stack 要素の型
struct stack{
    int top;
    elemtype elem[STACKSIZE];
};
void initstack( struct stack *s){
    s->top = -1;
}

void push( struct stack *s, elemtype val ){ /* この関数を実装する */}
elemtype pop(struct stack *s){/* この関数を実装する */}

/* stack の内容を表示する関数 */
void print_stack(struct stack s,char *hint) {
    int p;
    printf("[");
    for (p = 0; p <= s.top; p++) {
        printf("%d", s.elem[p] );
        if (p <= s.top - 1) printf(", ");
    }
    printf "]" # "%s\n", hint);
}

int main(void) {
    struct stack st;
    int i;
    initstack( &st );
    for (i = 1; i <= 5; i++) {
        push(&st, i);
        print_stack(st, "push 1");
    }
    for (i = 0; i < 4; i++) {
        int x, y;
        x = pop(&st);
        print_stack(st, "pop 1");
        y = pop(&st);
        print_stack(st, "pop 2");
        push(&st, x + y);
        print_stack(st, "push 2");
    }
    return 0;
}

```

図 1: 04A のプログラム

問 B [04B]: 整列

図 2 のプログラム片は $N(N \geq 1)$ 個の与えられた整数の並びを昇順に並び替えるものである。(ア) から (オ) に適当な C 言語の文を埋めてプログラムを完成させよ。 N はマクロで 5 と定義せよ。また、入力される整数は -100 から 100 の範囲と仮定してよい。

ただしキューが保持する要素 (element) は int 型とし、initqueue、queueempty、enqueue(または putq)、dequeue(または getq) は講義で示されたものとする (名前は違うものもある)。キューのサイズ QUEUESIZE は N より大きな正整数とする (たとえば 100)。

入出力例 04B

入力

2011

10

31

9

0

出力

(0 9 10 31 2011)

ヒント

- キューの実装はプログラミング通論の講義の通りとする。
- プログラムを作成する前にアルゴリズムを考えよ。□の中に適当な数を入れ、整列するアルゴリズムを作成せよ。
 1. N 個の整数を与えられた順にキュー qin に格納する。
(このとき qin の内容は、□□□□□である。)
 2. $i=0, \dots, (\text{ア})-1$ について以下の操作を繰り返す。
 - (a) キュー qin から値を 1 つ取り、 x に代入する。
(このとき qin の内容は、□□□□□である。)
 - (b) $j=i, \dots, (\text{イ})-1$ について以下の操作を繰り返す。
 - i. キュー qin から値を 1 つ取り、 y に代入する。
(このとき qin の内容は、□□□□□である。)
 - ii. もし、 $x < y$ なら (ウ) の処理を行う。
 - iii. そうでないなら (エ) の処理を行う。
(このとき qin の内容は、□□□□□である。)
 - (c) qout に (オ) を追加する。
(このとき qout の内容は、□□□□□である。)
 3. qout の内容を出力しプログラムを終了する。- 意図する通りにプログラムが動作しないときは、逐一、関数 printq や printf を使って、キューや変数の値をチェックせよ。

```

#include <stdio.h>
#define QUEUESIZE 10
#define N 5
typedef int elemtype;
struct queue{
    int head, tail;
    elemtype elem[QUEUESIZE];
};
void enqueue(struct queue *q, elemtype val); /* この関数を実装する */
elemtype dequeue(struct queue *q); /* この関数を実装する */
void initqueue(struct queue *q); /* この関数を実装する */
int queueempty(struct queue *q); /* この関数を実装する */

/* 与えられたキューの内容を表示する。*/
void printq(struct queue q) {
    printf("(");
    if (!queueempty(&q)) {
        printf("%d", dequeue(&q));
        while (!queueempty(&q)) printf(" %d", dequeue(&q));
    }
    printf(")\n");
}

int main(void) {
    struct queue qin, qout;
    char buf[100];
    int i, j, x, y;

    initqueue(&qin);
    initqueue(&qout);
    /* N 個の整数を与えられた順にキュー qin に格納する。*/
    for (i = 0; i < N; i++) {
        enqueue(&qin, atoi(fgets(buf, BUFSIZE, stdin)));
    }
    /* キュー qin に格納された整数を昇順に並び替えて キュー qout に格納する。*/
    for (i = 0; i < /*(ア)*/ ; i++) {
        x = dequeue(&qin);
        for (j = i; j < /*(イ)*/ ; j++) {
            y = dequeue(&qin);
            if (x < y) { /*(ウ)*/ } else { /*(エ)*/ }
        }
        enqueue(&qout, /*(オ)*/ );
    }
    printq(qout);
    return 0;
}

```

図 2: キューで整列

問 C [04C]: 迷路探索

例に示したような、左上の座標を (0,0) とするマス目でできた迷路があり、'*' は壁で、' ' は空きマスの通路とする。このような迷路において、指定したスタート S からゴール G までの最短距離を求めるプログラムをつくれ。例の迷路ではスタート S の座標は (1,1)、G の座標は (4,3) であり、最短距離は右上から回った 7 である。

手順 1: スタート位置の座標=(1,1) と距離=0 をキューに記録する

手順 2: キューが空になるまで以下を繰り返す。

手順 2-1: キューから位置と距離を取り出し、現在の位置と距離とする。

手順 2-2: もしその位置がゴール G ならば繰返しを抜ける。

手順 2-3: 現在の位置の上下左右を調べ、空きマスならば (現在の距離+1) をそのマスの距離として座標とともにキューに保存する

手順 3: ゴールに到達していれば距離を出力して終了する。

- N は 7 とする。
- ヒント：一度通った道を記録するように工夫することで、効率よく計算することもできる。
- G に到達できないことが分かった時は -1 を出力すること。例えば、迷路が最大でも $N \times N$ マスしかないと利用すると、距離が $N \times N$ を越えても G に辿り着けないと、G に到達できない迷路であることが分かる。「ヒント」にある工夫をしていないプログラムだと、QUEUESIZE を計算機で扱える範囲で大きくしてもオーバーフローしてしまう入力が存在する。工夫をしない場合、QUEUESIZE を (20*1024*1024) にして、それでオーバーフローしない範囲の入力に対して正しい答を出すプログラムを作ること。
- 標準入力を毎回入力することが面倒な場合は、ファイル (仮に maze とする) に内容を書いておき、

```
./a.out < maze
```

のように実行する。

入出力例 04C

入力

```
*****
*S      *
*  **  *
** *G *
*  **  *
**      *
*****
```

出力 (最短距離)

7

```

/* (x,y) 地点とそこまでの距離 c を記録する構造体 */
struct cost {
    int x;
    int y;
    int c;
};

int main(void) {
#define BUFSIZE (N*2)
    char buf[BUFSIZE];
    char maze[N][N];
    int i, j, sx, sy, gx, gy;
    struct cost pos;

    /* 迷路を読み込み、maze にセットする。*/
    for (i = 0; i < N; i++) {
        fgets(buf, BUFSIZE, stdin);
        for (j = 0; j < N && (buf[j] != '\n' && buf[j] != '\0'); j++) {
            if (buf[j] == 'S') {
                sx = j; sy = i; buf[j] = ' ';
            } else if (buf[j] == 'G') {
                gx = j; gy = i; buf[j] = ' ';
            }
            maze[i][j] = buf[j];
        }
        while (j < N) maze[i][j++] = ' ';
    }

    /* ※ */

    /* 最短距離を表示する。*/
    printf("%d\n", pos.c);

    return 0;
}

```

図 3: 最短距離

問 D [04D]: 塗りつぶし

char 型の要素を持つ二次元配列 `canvas[N][N]` があるとする (N は 1 以上の奇数)。`canvas` の中央に位置する要素 (`canvas[N/2][N/2]`) からはじめ、その上下左右にある空白文字を全て文字 `C` に置き換えるプログラムを作成せよ。

また、 N と C はいずれもマクロで、それぞれ 7 および `'.'` とする。

`canvas` にセットする文字列の読み込みと結果の表示する `main` 関数は図 4 のようである。コメント `/* ※ */` の部分に文字の置換えを行うコードを書けばよい。

入出力例 D

入力 1

```
+-----+
|       |
+-----|
|       |
|  -----+
|       |
+-----+
```

入力 1 に対する出力

```
+-----+
|. . . . .|
+-----.|
|. . . . .|
|. -----+
|. . . . .|
+-----+
```

入力 2

```
## ##
#  # #
#    #
#    #
#  #
#  #
#
```

入力 2 に対する出力

```
## ##
#..#..#
#.....#
#.....#
#...#
#.#
#
```

```

#define C '.'
#define N 7

int main(void) {
#define BUFSIZE (N*2)
    char buf[BUFSIZE];
    char canvas[N][N];
    int i, j;

    /* 文字列を読み込み、canvas にセットする。*/
    for (i = 0; i < N; ++i) {
        fgets(buf, BUFSIZE, stdin);
        for (j = 0; j < N && (buf[j] != '\n' && buf[j] != '\0'); ++j) canvas[i][j] = buf[j];
        while (j < N) canvas[i][j++] = ' ';
    }

    /* ※ */

    /* 置き換え後の canvas の内容を表示する。*/
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) printf("%c", canvas[i][j]);
        printf("\n");
    }

    return 0;
}

```

図 4: 塗りつぶし