プログラミングおよび実習 I・第 12 回

2003 年 7 月 2 日

1 コマンドライン引数と関数 main の引数

Unix (Linux) や Windows 等の OS 上の C プログラムの実行環境では、プログラムを起動した際にコマンドラインで与えた引数 (文字列) が何であったかを、関数 main の引数として知ることができるようになっています。このとき、main の第 1 引数には「コマンドライン引数の個数 + 1」が int 型の値として渡されます。また、main の第 2 引数には、文字列 (char * 型の値) を要素とした配列の先頭要素のアドレス (char **型) が渡されます。この配列は「コマンドライン引数の個数+2」個の要素からなっており、その先頭要素 (添字 0) はプログラムが起動されたときの「プログラム (コマンド) 名」、添字 1 の要素はコマンドラインでの第 1 引数、添字 2 の要素はコマンドラインでの第 2 引数、 ... となります。配列の最後の要素には (アドレスの) 0 が格納されています。

この様子は、次のようなプログラム args.c を実行してみるとよく分かります。

args.c

```
 1 #include <stdio.h>
 2
 3 int main(int argc, char *argv[])
 4 {
 5     int i;
 6
 7     printf("argc = %d\n", argc);
 8     for (i = 0; i < argc; i++)
 9         printf("argv[%d] = %s\n", i, argv[i]);
10     return 0;
11 }
```

args.c のコンパイルと実行例

```
s1609h017% cc -o args args.c
s1609h017% ./args
argc = 1
argv[0] = ./args
s1609h017% ./args -hello
argc = 2
argv[0] = ./args
argv[1] = -hello
s1609h017% ./args This is a test.
argc = 5
argv[0] = ./args
argv[1] = This
argv[2] = is
argv[3] = a
argv[0] = -hello
argv[4] = test.
s1609h017%
```

もう 1 つ、main の引数を利用したプログラムをみてみましょう。このプログラムは、コマンドライン引数として与えられた (日本語風の) 四則演算を計算して、その結果を表示するものです。

calc.c

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

```
 2 #include <string.h>
 3
 4 int main(int argc, char *argv[])
 5 {
 6     int a, b;
 7
 8     if (argc < 4) {
 9         printf("引数の数が足りません\n");
10         return 1;
11     }
12     if (sscanf(argv[1], "%d", &a) != 1) {
13         printf("第1引数が整数ではありません\n");
14         return 1;
15     }
16     if (sscanf(argv[3], "%d", &b) != 1) {
17         printf("第3引数が整数ではありません\n");
18         return 1;
19     }
20
21     if (strcmp(argv[2], "足す") == 0)
22         printf("%d + %d = %d\n", a, b, a+b);
23     else if (strcmp(argv[2], "引く") == 0)
24         printf("%d - %d = %d\n", a, b, a-b);
25     else if (strcmp(argv[2], "掛ける") == 0)
26         printf("%d * %d = %d\n", a, b, a*b);
27     else if (strcmp(argv[2], "割る") == 0)
28         printf("%d / %d = %d ... %d\n", a, b, a/b, a%b);
29     else {
30         printf("演算の種類が理解できません\n");
31         return 1;
32     }
33
34     return 0;
35 }
```

calc.c の実行例

```
s1609h017% ./calc 23 足す 5
23 + 5 = 28
s1609h017% ./calc 23 掛ける 5
23 * 5 = 115
s1609h017% ./calc 23 割る 5
23 / 5 = 4 ... 3
s1609h017% ./calc 23 引く 5
23 - 5 = 18
s1609h017% ./calc 23引く5
引数が足りません
s1609h017%
```

コマンドライン引数はスペースやタブ文字で区切られますから、「足す」や「割る」の両側にはこれらの区切り記号が必要となります。

このプログラム中で使われている strcmp は、文字列処理のための標準ライブラリ関数の 1 つで、引数 (char 型へのポインタ型) として与えられた 2 つの文字列が等しいときには 0 を、等しくないときには 0 以外の整数を返すものです[1]。この関数を使用する際には string.h というヘッダファイルを

[1] 与えられた 2 つの文字列が異なる場合は、文字列の先頭から順に対応する文字を調べていき最初に異なった文字の文字コードを比較し、第 1 引数の方が第 2 引数より小さい場合は負の整数を、大きい場合は正の整数を返り値として返します

include する必要があります (calc.c の 2 行目)。

2 スタックを使った「待った」の実現

本当のオセロゲームではもちろんルール違反ですが、プレーヤーが「待った」を選択できるようにしてみましょう。最も単純と思われる方法は、ゲームの進行に伴う局面の変化をスタックに保存 (push) しておき、「待った」のときにスタックから過去の局面を取り出 (pop) して復元することです。

オセロゲームの一局は、ゲーム盤の $8 \times 8 - 4$ 個の空きマスを埋めていくことですから、たとえ石を置く差し手すべての間に 1 回ずつパスがあったとしても、 119 手目までには終局するはずです。そこで、この数の局面を保存するのに十分な大きさのスタック (OState 型の要素からなる配列) を用意しておくことにします。この局面のスタック (配列とスタックポインタ) と現在の局面の組がオセロゲームのプログラムの状態を表現していると考えることができますから、これらをひとまとめにして、次のような構造体型のデータとして扱うことにします。

```
#define MAX_UNDO        (2*BOARD_SIZE*BOARD_SIZE)

typedef struct {
    OState state;               /* 現在局面 */
    OState undoStack[MAX_UNDO]; /* 局面のスタック */
    int sp;                     /* スタックポインタ */
} GameState;
```

この GameState 型のデータの内、局面のスタックに対する push と pop の動作は、次の 2 つの関数で実現できます。それぞれ、現在局面をスタックへ push したり、スタックから pop したものを現在局面としたりしていることに注意してください。これら 2 つの関数は、 push や pop が成功した場合は TRUE を、失敗した場合は FALSE を返り値として返しています。

```
/* 現在局面をスタックへプッシュする */
Boolean push(GameState *game)
{
    if (game->sp == MAX_UNDO)
        return FALSE;
    game->undoStack[game->sp++] = game->state;
    return TRUE;
}

/* スタックから局面をポップして、それを現在局面とする */
Boolean pop(GameState *game)
{
    if (game->sp == 0)
        return FALSE;
    game->state = game->undoStack[--game->sp];
    return TRUE;
}
```

いつ局面を push すればよいか 「待った」による局面の巻き戻しを実現するためには、局面が変化する (手番が変る) ときに、必ず、その前の局面をスタックに push しておくようにしなければなりません。ところが、関数 inputMove がユーザーから与えられた指示を実行する際に、それで局面が変るかどうかを事前に判定するのは困難です。例えば、ユーザーがある位置に石を置こうとした場合、それ (関数

putDisk の呼び出し) が成功したときには、もう局面は変ってしまっています。そこで、次のいずれかの方針でプログラムを作成しましょう。

方針 (a) 関数 inputMove が呼ばれた時点で、とりあえず現在局面をスタックに push しておく。この局面に対して石が置かれ、関数 inputMove から関数 main に戻るのなら何も問題はないが、パスが起こった場合は、main に戻ることなくユーザーからの入力の処理を続行するため、パスが起こった後の局面もタックへ push する必要がある。次の 2 つの場合にパスが起こるので、これらの場合にその時の局面をスタックへ push する。

(1) 関数 autoMove の呼び出しでパスが選択されたとき

(2) ユーザーがパスを選択して、それが成功したとき

方針 (b) 関数 inputMove がプロンプトを出す前に必ず現在局面をスタックに push しておく。この局面に対して石が置かれたりパスが起こったりして局面が変化するのなら何も問題はないが、ユーザーからの入力ミスなどで局面が変化しなかった場合は、再入力を促すプロンプトを出す際に、もう一度、同じ局面が push されています。そこで、ユーザーの指示を実行しても局面が変化しなかったときには、スタックから局面を pop して元の状態に戻す。局面が変化せずにユーザーに再入力を促す場合としては次の 3 つが考えられるので、これらの場合に pop を呼び出す。

(1) ユーザーからの入力が空だった (関数 getLine が 0 以下の値を返した) り、認識出来なかった場合

(2) ユーザーがパスしようとしたが、それが失敗したとき

(3) ユーザーが石を置こうとしたが、それが失敗したとき

方針 (c) 現在局面をスタックへ保存する必要があるかどうかを示すフラグ (真偽値を記憶する変数) を用意しておく。関数 inputMove がプロンプトを表示する際に、このフラグを調べ、フラグが立っているときのみ現在局面を push して、フラグを下ろす。関数 inputMove が呼ばれた時点でまずフラグを立てておき、inputMove の中でパスが起こって手番が変ったときに、再びフラグを立てるようにする。

一方、「待った」を実行する関数は、次のように定義することができますので、ユーザーが「待った」を選択した場合には、この関数 undo を呼び出した後 main へ戻るようにします。

```
void undo(GameState *game)
{
    int t = game->state.turn;

    pop(game); /* まず、push されている現在局面を pop する */
    do {       /* 自分の手番になるまで過去の局面を pop する */
        if (!pop(game))
            return;
    } while(game->state.turn != t);
}
```

また、これまでのオセロプログラムでは、関数 main で用意した OState 型の変数で現在の局面の状態を記憶していましたが、この変数の代わりに GameState 型の変数 (例えば game) を用意しておいて、関数 reset や showBoard には &game.state を、関数 inputMove には &game を渡すようにします。

```
int main()
{
    GameState game;

    game.sp = 0;
    reset(&game.state);
    do {
        showBoard(&game.state);
    } while (!inputMove(&game));
    return 0;
}
```

3 演習問題

3.1 コマンドライン引数で自動対局が指定できるようにする

まず、前回の演習問題で作成したプログラム othello25.c を othello26.c にコピーしなさい。次に、この othello26.c を変更して、次の実行例のように「-b」や「-w」をコマンドライン引数としてこのプログラムを起動すると、それぞれ、黒や白の手番にプログラムが自動的に差し手を選ぶようにしなさい。

othello26.c の実行例

```
s1609h017% ./othello26 -b
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番（黒2/白2）= a2
 a b c d
1□□□□
2★●●□
3□●○□
4□□□□
白番（黒4/白1）= c1
 a b c d
1□□☆□
2●●○□
3□●○□
4□□□□
黒番（黒3/白3）= d1
 a b c d
1□□○★
2●●●□
3□●○□
4□□□□
白番（黒5/白2）= a3
 a b c d
1□□○●
2●○●□
3☆○○□
4□□□□
黒番（黒3/白5）= a4
 a b c d
1□□○●
2●○●□
3●●○□
4★□□□
白番（黒6/白3）= d2
 a b c d
```

実行例の続き

```
1□□○●
2●○○☆
3●●○□
4●□□□
黒番（黒5/白5）= b1
 a b c d
1□★●●
2●●○○
3●●○□
4●□□□
白番（黒8/白3）= a1
 a b c d
1☆●●●
2●○○○
3●●○□
4●□□□
黒番（黒7/白5）= d3
 a b c d
1○●●●
2●○●●
3●●●★
4●□□□
白番（黒11/白2）= d4
 a b c d
1○●●●
2●○●●
3●●○●
4●□□☆
黒番（黒10/白4）= b4
 a b c d
1○●●●
2●○●●
3●●●●
4●★□○
白番（黒12/白3）= p
黒番（黒12/白3）= p
黒の勝ちです（黒12/白3）
s1609h017%
```

この実行例での黒番の差し手 (例えば a2 や p) は、ユーザーがキーボードから入力したのではなく、このプログラム othello26.c が自分で表示したものです。

まず、関数 inputMove に int 型の引数 (たとえば mode) を追加し、プログラムが自動的に差し手を決める手番をこの引数で受け取るようにします。この引数 (mode) が NONE であれば、inputMove にはこれまでと同じ動作をさせます。関数 inputMove は、プロンプトを出力した後、この引数 mode の値を調べて、それが現在の手番 (state->turn) と等しければ、関数 autoMove を呼び出して自動的に差し手を決めます。このとき、autoMove が選んだ手を、ユーザーが差し手を入力したときと同じように表示しましょう。autoMove が石を置いたのか、それともパスしたのかは、autoMove の返り値で判定できます。石を置いた場合、その位置は state->lastX と state->lastY が保持しているはずです。これらの情報に基づいて autoMove が選んだ差し手を表示するようにしましょう。

この othello26.c の関数 main では、次の例のように、コマンドライン引数 (第 1 引数) の内容を調べて、「-b」や「-w」の場合に、inputMove に追加した引数を、それぞれ BLACK や WHITE として呼び出すようにすれば ok です。

othello26.c の main の定義例

```
int main(int argc, char *argv[])
{
    OState state;
    int mode = NONE;

    if (argc > 1) {
        if (argc != 2) {
            printf("引数の数が多すぎます\n");
            return 1;
        }
        if (strcmp(argv[1], "-b") == 0)
            mode = BLACK;
        else if (strcmp(argv[1], "-w") == 0)
            mode = WHITE;
        else {
            printf("「%s」は認識できないオプションです\n", argv[1]);
            return 1;
        }
    }
    reset(&state);
    do {
        showBoard(&state);
    } while (!inputMove(&state, mode));
    return 0;
}
```

3.2 「待った」ができるようにする

まず、プログラム othello26.c を othello27.c にコピーしなさい。次に、この othello27.c を変更して、第 2節で解説した方法を使って、キーボードから u (あるいは U) の 1 文字を入力すると、1 つ前の自分の手番まで局面を戻すようにしなさい。

othello26.c では、関数 main が関数 inputMove を呼び出す際に、第 2 引数として自動対局の状態 (mode) を渡していましたが、これもオセロゲームのプログラムの状態の一部と考えられますから、次のように GameState 型の一部に取り込んでしまうのが自然でしょう。

```
typedef struct {
    int mode;                   /* 自動対局の状態 (BLACK/NONE/WHITE) */
    OState state;               /* 現在局面 */
    OState undoStack[MAX_UNDO]; /* 局面のスタック */
    int sp;                     /* スタックポインタ */
} GameState;
```

以下は othello27.c の実行例です。

実行例

```
s1609h017% ./othello27
 a b c d
1□□□□
2□○●□
3□●○□
4□□□□
黒番（黒2/白2）= a2
 a b c d
1□□□□
2★●●□
3□●○□
4□□□□
白番（黒4/白1）= a3
 a b c d
1□□□□
2●●●□
3☆○○□
4□□□□
黒番（黒3/白3）= d4
 a b c d
1□□□□
2●●●□
3○○●□
```

続き

```
4□□□★
白番（黒5/白2）= u
 a b c d
1□□□□
2★●●□
3□●○□
4□□□□
白番（黒4/白1）= a1
 a b c d
1☆□□□
2●○●□
3□●○□
4□□□□
黒番（黒3/白3）= d3
 a b c d
1○□□□
2●○●□
3□●●★
4□□□□
白番（黒5/白2）= a3
 a b c d
1○□□□
2○○●□
```

続き

```
3☆●●●
4□□□□
黒番（黒4/白4）= b1
 a b c d
1○★□□
2○●●□
3○●●●
4□□□□
白番（黒6/白3）= c1
 a b c d
1○○☆□
2○○●□
3○●●●
4□□□□
黒番（黒4/白6）= p
白番（黒4/白6）= U
 a b c d
1○★□□
2○●●□
3○●●●
4□□□□
白番（黒6/白3）= q
s1609h017%
```

プログラミングおよび実習 I・第 12 回・終り

付録: オセロプログラムの強さを試す

みなさんが作成した自動対局プログラムの相手をしてくれるプログラムを用意しました。次のように使ってください。

(1) 自分が作ったプログラムの名前は `othello.c` とする。

(2) ゲーム盤の大きさは 8×8 にしておく。

(3) キーボードから `t` が入力されたときには、3 秒以内に手を選んで差すようにしておく。

(4) 次の実行例のように端末ウィンドウから「`mprog1 othello.c`」を実行して対戦する。

対戦の例

```
s1609h017% cc -o othello othello.c
s1609h017% mprog1 othello.c

あなたのプログラムの相手を選んでください

    1. ひろし
    2. あきら
    3. たかし
    4. たけし
    5. けんじ
    6. 全員と6番勝負

あなたの相手（1〜6）？ 1

あなたのプログラムの手番を入力してください
あなたの手番（1=黒/0=白）？ 1
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
黒番（黒2/白2）= t
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□★●●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
白番（黒4/白1）= e3
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□☆□□□
4□□●●○□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
```

```
黒番（黒3/白3）= t

       ⋮

白番（黒37/白26）= p
黒番（黒37/白26）= t
 a b c d e f g h
1○○○○○○○○
2○●●●●●●★
3○○●●●●●○
4○●●●○●●○
5○●●○●●●○
6●●○●●●●○
7●○●●●●●○
8○●●●●●○○
黒の勝ちです（黒39/白25）

s1609h017%
```

あなたのプログラムの対戦相手を 5 種類 (5 人) の差し手から選ぶことができます。それぞれが選ぶ差し手は、対局する度にある程度違ってくるはずです。あなたが Enter キーを押すごとに 1 手ずつ局面が進んでいきます。

次の実行例のように 5 人全員と 6 番勝負をすることもできます。この場合は、1 手ごとに Enter キーを押す必要はありません。1 局ごとに先手後手が入れ替わるようになっています。最初はあなたのプログラムが先手 (黒番) です。

5 人全員との 6 番勝負

```
s1609h017% mprog1 othello.c

あなたのプログラムの相手を選んでください

    1. ひろし
    2. あきら
    3. たかし
    4. たけし
    5. けんじ
    6. 全員と6番勝負

あなたの相手（1〜6）？ 6

1. ひろしとの対戦: 勝 勝 負 勝 勝 勝: 5勝1敗0分け
2. あきらとの対戦: 負 勝 分 負 負 負: 1勝4敗1分け
3. たかしとの対戦: 負 負 負 負 負 勝: 1勝5敗0分け
4. たけしとの対戦: 負 負 負 負 負 負: 0勝6敗0分け
5. けんじとの対戦: 負 負 負 負 負 負: 0勝6敗0分け

s1609h017%
```