# C から入る C++

担当: 金丸隆志

第 4 回

## 1 ソートプログラムの作成

### 1.1 ソートとは

今回は前回に作成した swap 関数 (ポインタ引数または参照引数) を用いたソートプログラムを作成してもらう.

ソートとは, 集合の要素を何らかの基準に従って並べ変えることを言う.

例えば, Windows には「アイコンの整列」という機能があり, アイコンを「名前順」,「種類順」,「サイズ順」などに並べ変えることができるが, これもソートの一例である.

今回扱うのは, 正の整数の列 (例えば「2, 3, 4, 1, 3」) を小さい順に (「1, 2, 3, 3, 4」のように) 並べ変える, という例である.

なお, 本章で扱うソートは「選択ソート」と呼ばれるアルゴリズムに近いが, この選択ソートのアルゴリズムを学ぶことを重視しているわけではない. むしろ,「考えたアルゴリズムをプログラムの言葉に翻訳する」技術を身につけることが目的である.

### 1.2 基本方針

以下の方針でプログラムを作成することにしよう.

(1) 固定長 (SIZE とする) の int 型配列 x[SIZE] を確保する.

(2) 0 ∼ SIZE−1 の範囲の整数の乱数を生成し, 配列 x[] に順に格納する.

(3) 配列 x[] とその長さ SIZE を関数 sort_func に渡す.

(4) sort_func 関数の処理が終了したとき, 配列 x[] にはソートされた整数が格納されているとする.

今回は sort_func 関数を作ることに集中してもらうため, 乱数を生成する方法などはあらかじめこちらで提示する.

### 1.3 マクロ定義

今回, 配列の大きさ (つまり, ソートする数列の長さ) をいろいろ変えてプログラムを実行してもらうことになる.

そのような時, あらかじめ配列の大きさをマクロ定義しておくと便利である (前回,「マクロ定義は扱わない」と書いたのですが, 結局使うことになりました). 図 1 のリスト 1 行目のようにマクロ SIZE

```
#define SIZE 5      /* セミコロンはつけない */

int main(void){
    int x[SIZE];
     for(int i=0 ; i<SIZE ; i++){
        x[i] = /* 乱数 のセット */
    }
     ...
```

図 1: マクロ定義による, 配列 x[5] の定義

を 5 と定義すると, コンパイルの時にプログラム中の "SIZE" が全て 5 に置き換えられる. あくまで置き換えるだけなので,「SIZE = 10」などのように変数として扱うことはできない.

配列の大きさを変えたい時は 1 行目の「5」を別の数字に書換えて再コンパイルすれば良い. (マクロの利用により, 書き換えるべき箇所が減るのがわかるだろうか？)

なお, for 文の中で int 型の整数 i が定義されているが, これは C++ で採り入れられた機能である. 便利な機能なので, 以後も利用することにする.

### 1.4 乱数生成

乱数生成は C の標準ライブラリの rand 関数を利用することにする. rand 関数は <stdlib.h> で定義されているので, これをインクルードする必要がある. 以下のリストで配列 x[] に 0 ～ SIZE−1 の正の整数乱数をセットすることができる.「srand(1)」

```
#include <stdlib.h>
#define SIZE 5

int main(void){
   int x[SIZE];

    srand(1);
   for(int i=0 ; i<SIZE ; i++){
       x[i] = rand()%SIZE
   }
```

図 2: rand() を用いた乱数のセット

は, 乱数の系列をセットしている. srand に渡す値を 1 から変更すれば, 異なる系列の乱数が得られる (逆に言うと, 1 から変更しなければ, 毎回同じ乱数系列が得られるということ [3]).

rand()%SIZE は, rand() が生成した乱数 (十分大きい数と思って良い [4]) を SIZE で割った余りを求めている. これにより, 0 ～ SIZE−1 の乱数が得られる (例えば, ある整数を 5 で割ればその余りは 0 ～ 4 になることに注意しよう). なお, 乱数であるから, 同じ数が複数回出ることや, ある数が一度も出ないこともあることに注意 (図 3 はその様な例になっている).

### 1.5 ソートアルゴリズム

以上で, 配列 x[SIZE] に 0 ～ SIZE−1 の乱数をセットすることができた. では, 本節では配列 x[SIZE] を小さい順にソートすることを考えよう.

本章で取り扱うソートアルゴリズムの具体的なアイディアを図 3 に示す. 図の流れの解説は以下の通り.

(1) 基準点を i=0 に, 比較対象点を j=1 にセットし, j を順に動かしながら, 2 つの数 (x[i] と x[j]) を比較していく

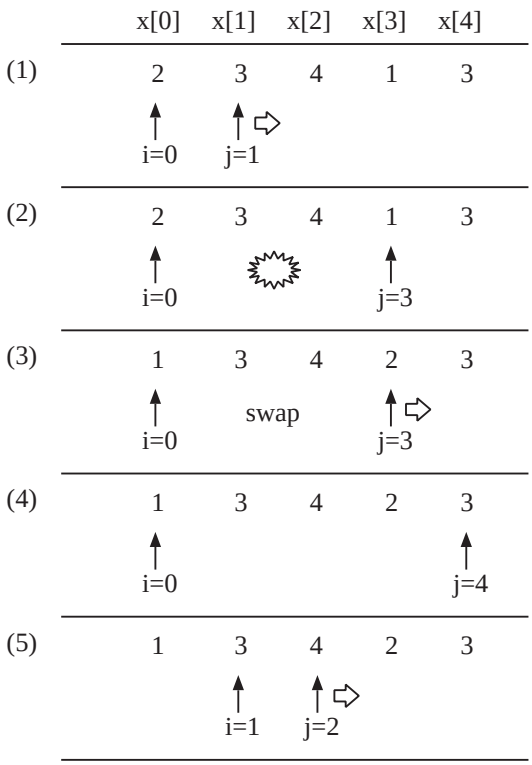


図 3: 本章で取り扱うソートの具体例 (SIZE が 5 の場合)

(2) i の位置の数よりも j の位置の数が小さい場合が見つかる

(3) swap を実行し, 値を交換する.

(4) j が配列の最後の数 (ここでは j=4) に達するまで比較を続ける. この時点で, i=0 には数列の最小値が来ているはずである.

(5) j が j=4 に達したら, i=1, j=2 にセットしなおし, 再び j をずらしながら比較してゆく

(6) i=3, j=4 まで比較が終了したらソート終了

以上の流れをプログラム化することが目標である.

### 1.6 プログラムの骨組み

以上をまとめると, プログラムの骨組みは以下のリストのようになるだろう.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

void swap_r(int &x, int &y);
void sort_func(int *x, int N);

int main(void){
    int x[SIZE];

    srand(1);
    for(int i=0 ; i<SIZE ; i++){
        x[i] = rand()%SIZE;
    }

    printf("------before------\n");
    for(int i=0 ; i<SIZE ; i++){
        printf("%d\n",x[i]);
    }

    sort_func(x,SIZE);

    printf("------after------\n");
    for(int i=0 ; i<SIZE ; i++){
        printf("%d\n",x[i]);
    }
    return 0;
}

void swap_r(int &x, int &y){
    /* ここは前回の swap 関数 を書く */
}

void sort_func(int *x, int N){
    /* ここにソートアルゴリズムを書く */
}
```

図 4: ソートプログラムの骨組み

[**レポート 課題 1**]

(1) 図 4 を書き写し, 実行してみる. 写し間違いがなければ, 配列に確保された乱数が 2 回表示されるはずである (sort_func 関数が何もしない関数になっているので, 同じ内容が 2 回表示されるだけである). また, srand() の中身の値を変えて再構築→実行すれば, 別の乱数系列が得られることに注意.

(2) swap_r に前回に作成した関数の中身を記述し, さらに, sort_func 関数の中身を記述せよ. sort_func 関数は, 大きさ N の配列 x[N] (つまり, x[0]〜x[N−1] まで使える配列) を小さい順に並び変える関数であり, 内部で swap_r 関数を呼び出すようにすること. また, sort_func 関数内部ではマクロ SIZE を用いず, 変数 N を使うようにすること. なお, sort_func 関数内部で配列 x[N] が使える理由は, 前回資料の「付録 A」を参照.

[**注意**] 参照引数版の swap_r を用いるように書いたが, これはこちらの方が簡単と思われるためである. ポインタ引数版の swap_p を使いたい人はそちらを用いて構わない.

(3) sort_func 関数が完成し, うまく動作することが確認できたら, 3 行目のマクロ SIZE の値を 100, 1000, 10000 などにして再構築→実行してみよ. 大きな配列のソートが行われることになる.

[**注意**] なお, SIZE を大きくする場合, 配列の表示 (printf の部分) はコメントアウト (/* ... */ でくくる ) して無効化した方が良い. そうしないと, コンソールに大量に数字が表示されることになってしまう. 代わりに, sort_func 終了後に「done.」などとメッセージが表示されるようにしておくと分かりやすいだろう.

## 2 標準ライブラリを用いたクイックソートアルゴリズムの利用

前章で記述してもらったソートは,「選択ソート」と呼ばれるアルゴリズムに近いものだが, 実はソートのアルゴリズムとしては最も性能が悪く, 配列サイズが大きいときに用いられることはあまりない (ただし, プログラミングの学習としては適しているので記述してもらったわけである) .

効率の良いソートアルゴリズムの一つとしてクイックソートアルゴリズムが知られており,「C 標準ライブラリ」および「C++ 標準ライブラリ」どちらにも含まれている (C++ は C を拡張した言語であるから, どちらのライブラリも使用できる).

ここではそのどちらかを実行してもらう．まず，レポート課題 を先に提示しておこう．

[**レポート 課題 2**]

(1) 「C 標準ライブラリ版」,「C++ 標準ライブラリ版」のどちらか好きな方を選択し, リストを書き写して実行してみよ．また, SIZE を 100, 1000, 10000 などに変更して, 計算にかかる時間を体感してみよ．課題 1 のソートにかかった計算時間と比較してどうか．

[**注意**] ソートがうまくいっているか確認したい場合は, 配列の最初のいくつかの要素を printf して確認すると良い．(全て printf すると大変なことになるので注意)

(2) クイックソートアルゴリズムは配列サイズ $N$ に対し, 計算量が平均的に $N \log N$ 程度のアルゴリズムであると言われる．それに対し, 課題 1 のソートはどれくらいの計算量か考えたい．図 3 にあるように, 課題 1 のソートは 「2 つの数の比較」を繰り返し行うが, 配列サイズが $N$ のとき, この比較は何回になるだろうか．それは $N \log N$ と比べてどうだろうか．

以下, クイックソートを用いたソートプログラムを提示する．

なお, あらかじめ注意しておくが, クイックソートライブラリを使えるようになることが本章の目的ではない．ライブラリの使い方は, 使うたびにライブラリ辞典やヘルプなどを調べれば良い, というのが我々の立場である．

重視したいのは

- 動くプログラムを書けるようになることは重要であるが,「使える」プログラムを書くにはプログラムの効率を意識しなければならないこともある
- 便利な, 効率のよいアルゴリズムや機能はライブラリで提供されることが多いことを知る．

ということである．

## 2.1 C 標準ライブラリ編

C 標準ライブラリの qsort 関数を用いたクイックソートプログラムを図 5 のリストに示す．

```
#include <stdio.h>
#include <stdlib.h>       /* for qsort */
#define SIZE 10

int compare(const void *x, const void *y);

int main(void){
    int x[SIZE];

    srand(1);
    for(int i=0 ; i<SIZE ; i++){
        x[i] = rand()%SIZE;
    }

    qsort(x, SIZE, sizeof(int), compare);

    printf("done.\n");
    return 0;
}

int compare(const void *x, const void *y){
    return( *((int *)x) - *((int *)y) ) ;
}
```

図 5: C 標準ライブラリ qsort 関数によるクイックソート

このプログラム全てを理解する必要はないが, 興味のある人のために, 理解の助けになるヒントを記しておく．

- qsort 関数には, 配列の先頭へのポインタ (x), 配列サイズ (SIZE), 配列の 1 要素のサイズ (大抵の場合 int 型は 4 バイトだが, ここでは sizeof 演算子を用いて値を渡している), そして比較用関数 compare を渡している．
- 比較用関数 compare は, 並び替えのルールを定めている．return 文の中の x と y を入れ換えれば大きい順にソートされる．なお, return 文の中の「*((int *)x)」は, void 型のポインタ x を int 型のポインタにキャストし, その値を * で求めている．

### 2.2 C++ 標準ライブラリ編

C++ 標準ライブラリ [6] を用いたクイックソートプログラムを図 6 のリストに示す.

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#define SIZE 10
using namespace std;

int main(void){
    vector<int> x;

    srand(1);
    for(int i=0 ; i<SIZE ; i++){
        x.push_back( rand()%SIZE );
    }

    sort(x.begin(), x.end());

    printf("done.\n");
    return 0;
}
```

図 6: C++ 標準ライブラリによるクイックソート

このプログラム全てを理解する必要はないが, 興味のある人のために, 理解の助けになるヒントを記しておく.

- x を配列ではなく, STL (Standard Template Library) [7] の int 型の vector コンテナとして定義している. (これにより, 配列の要素数を意識する必要がなくなる)
- x に値を代入するときは, push_back 関数により, 乱数を x の最後尾にどんどん追加して行く, というイメージである. なお,「x.[関数]()」という形式はクラスのメンバ関数呼び出しとして今後学ぶことになる.
- sort 関数に x の先頭 (x.begin()) および 最後尾 (x.end()) を代入するとソートされる.

なお, x の要素の表示には, 通常通り for 文の中で「printf("%d\n", x[i]);」すれば良い. (C++ 的に #include <iostream> してから「cout << x[i] << endl;」でも構わない)

## 参考文献


[1] Steve Oualline, "C++ 実践プログラミング," オライリー・ジャパン (1996).

[2] B.W. カーニハン/D.M リッチー, "プログラミング言語 C (第 2 版)," 共立出版株式会社 (1989).

[3] 実行ごとに異なる乱数系列にしたい場合は, 実行された時刻を取得して srand に代入するとよい. 興味のある人は調べてみるとよいだろう.

[4] rand 関数は 0 〜 RAND_MAX の整数乱数を返す. RAND_MAX は stdlib.h で定義されており, BCC の場合 0x7fff (16 進数) である. なお, 乱数には「性能」があり, この rand 関数は性能は良くないので研究などには向いていないとされる.

[5] C/C++ 標準ライブラリは, 標準 C/C++ がサポートされている環境で提供されるライブラリ群である. つまり, OS (Windows, linux, Solaris, ···) やコンパイラ (BCC, C++ Builder, Visual C++, gcc, ···) に依らず使えるため, 標準ライブラリをうまく利用すると, 移植性の高いプログラムを書くことができる.

C の標準ライブラリに関しては例えば以下の URL が参考になるだろう.

http://www.bohyoh.com/CandCPP/C/Library/

[6] Nicolai M. Josuttis, "C++ 標準ライブラリ チュートリアル & リファレンス," アスキー (2001).

[7] STL は 1994 年に C++ に加えられ, 1998 年の C++ ANSI 標準化と共に標準化された比較的新しい技術である. そのため, 教科書にはあまり載っていないことが多いが, 今後普及していくことが十分考えられる. そこで, 本演習でも機会をみて少しずつ紹介していくことにしたい.