# 第7章 配列 - 演習問題

# 演習問題

## 演習の目的

* 1次元配列と多次元配列の理解を深める
* 文字列配列の扱い方を習得する
* 配列を使った実践的な問題解決能力を養う

## 演習問題

### 演習6-1: 配列の基本操作

整数配列に対する各種操作を実行するプログラムを作成してください。

**要件:** - 5個の整数を入力して配列に格納 - 以下の操作を実装: - 配列の全要素を表示 - 最大値と最小値を見つける - 平均値を計算 - 配列を逆順に並べ替える - 指定した値が配列内に存在するか検索

**実装のヒント:** - 配列のサイズは#defineで定義 - 逆順は別の配列にコピーまたはその場で交換

### 演習6-2: 成績管理システム

複数の学生の複数科目の成績を管理するプログラムを作成してください。

**要件:** - 5人の学生、4科目（国語、数学、英語、理科）の成績を2次元配列で管理 - 各学生の合計点と平均点を計算 - 各科目の平均点を計算 - 最高得点の学生と科目を表示 - 成績表を見やすく表示

**実装のヒント:** - 2次元配列 grades[][] を使用 - 行が学生、列が科目

### 演習6-3: 文字列配列の操作

都道府県名を管理するプログラムを作成してください。

**要件:** - 10個の都道府県名を文字列配列に格納 - 以下の機能を実装: - 全都道府県名をアルファベット順に表示 - 文字数が最も長い都道府県名を見つける - 指定した文字で始まる都道府県名を検索 - 都道府県名に「県」が含まれるものをカウント

**実装のヒント:** - char prefecture[][] またはchar \*prefecture[] を使用 - strcmp()関数で文字列比較

### 演習6-4: 行列演算

2次元配列を使って行列演算を実行するプログラムを作成してください。

**要件:** - 3x3の行列を2つ入力 - 以下の演算を実装: - 行列の加算 - 行列の減算 - 行列の乗算 - 転置行列の計算

**実装のヒント:** - 行列の乗算: result[i][j] = Σ(a[i][k] \* b[k][j]) - 各演算結果を見やすく表示

### 演習6-5: 簡易辞書プログラム

英単語と日本語訳のペアを管理する辞書プログラムを作成してください。

**要件:** - 英単語と日本語訳のペアを10個程度登録 - 以下の機能を実装: - 英単語を入力すると日本語訳を表示 - 日本語を入力すると英単語を表示 - 登録されている全単語を一覧表示 - 単語の追加機能（配列に空きがある場合）

**実装のヒント:** - 2つの文字列配列（英語用と日本語用）を使用 - または構造体の配列を使用（先取り）

## チャレンジ問題

### チャレンジ6-1: ソートアルゴリズムの実装

配列のソートアルゴリズムを複数実装して比較してください。

**要件:** - 以下のソートアルゴリズムを実装: - バブルソート - 選択ソート - 挿入ソート - 各アルゴリズムの実行時間を測定 - ランダムな配列、ソート済み配列、逆順配列で性能比較

**追加要件:** - 比較回数と交換回数をカウント - 結果をグラフィカルに表示（\*を使った簡易グラフ）

### チャレンジ6-2: ライフゲーム（Conway’s Game of Life）

セル・オートマトンの一種であるライフゲームを実装してください。

**要件:** - 20x20のグリッドで実装 - 初期パターンを設定可能 - 以下のルールを実装: - 生きているセルの周囲に2-3個の生きたセルがあれば生存 - 死んでいるセルの周囲にちょうど3個の生きたセルがあれば誕生 - それ以外は死亡 - 世代を進めるごとに画面をクリアして表示

**実装のヒント:** - 2つの2次元配列を使用（現在と次世代） - 境界条件の処理に注意

## 提出方法

1. 各演習問題ごとに独立したCファイルを作成
2. ファイル名は ex6\_1.c, ex6\_2.c のような形式で
3. 配列のサイズや境界チェックに特に注意
4. メモリアクセスエラーが起きないよう十分にテスト

## 期限

配列は多くのプログラムの基礎となるデータ構造です。 しっかりと理解してから次の章へ進みましょう。

# 解答例

## 解答例一覧

### 演習6-1: 数値の統計処理

* **ファイル**: ex6\_1.c, ex6\_1\_c99.c
* **学習内容**: 1次元配列の操作、最大値・最小値・平均値の計算
* **ポイント**: 配列の初期化、ループによる要素の処理

### 演習6-2: 配列内の検索

* **ファイル**: ex6\_2.c, ex6\_2\_c99.c
* **学習内容**: 線形検索アルゴリズム、見つかった要素のインデックス取得
* **ポイント**: 配列の走査、条件判定による検索処理

### 演習6-3: 配列のソート

* **ファイル**: ex6\_3.c, ex6\_3\_c99.c
* **学習内容**: バブルソートアルゴリズム、配列要素の交換
* **ポイント**: ネストしたループ、要素の入れ替え処理

### 演習6-4: 2次元配列による行列演算

* **ファイル**: ex6\_4.c, ex6\_4\_c99.c
* **学習内容**: 2次元配列の操作、行列の加算・乗算
* **ポイント**: 多次元配列のインデックス操作、行列計算アルゴリズム

### 演習6-5: 文字列の操作

* **ファイル**: ex6\_5.c, ex6\_5\_c99.c
* **学習内容**: 文字配列による文字列処理、文字列関数の実装
* **ポイント**: 文字配列と文字列の関係、null終端文字の扱い

## チャレンジ問題

### チャレンジ6-1: ソートアルゴリズムの実装と比較

* **ファイル**: ex6\_challenge1.c, ex6\_challenge1\_c99.c
* **学習内容**: バブルソート、選択ソート、挿入ソートの実装と性能比較
* **ポイント**: アルゴリズムの性能測定、実行時間と比較・交換回数の計測

### チャレンジ6-2: ライフゲーム（Conway’s Game of Life）

* **ファイル**: ex6\_challenge2.c, ex6\_challenge2\_c99.c
* **学習内容**: 2次元配列を使ったセルオートマトンの実装
* **ポイント**: 複雑なルールの実装、境界条件の処理、世代管理

## C90版とC99版の違い

### C90版（基本ファイル）

* すべての変数を関数の先頭で宣言
* 配列のサイズは定数で指定
* /\* \*/ 形式のコメントを使用
* 従来のC言語の配列操作に準拠

### C99版（\_c99.cファイル）

* 変数を使用する箇所の近くで宣言可能
* 可変長配列（VLA）に対応（一部の演習で使用）
* // 形式のコメントに対応
* より現代的な配列操作

## コンパイルと実行

### C90版

gcc -Wall -Wextra -pedantic -std=c90 ex6\_1.c -o ex6\_1  
./ex6\_1

### C99版

gcc -Wall -Wextra -pedantic -std=c99 ex6\_1\_c99.c -o ex6\_1\_c99  
./ex6\_1\_c99

### Makefileを使用

make all # 全ての解答例をコンパイル  
make solutions # 解答例のみコンパイル  
make ex6\_1 # 個別の解答例をコンパイル  
make ex6\_1\_c99 # C99版の個別コンパイル  
make run-solutions # 解答例を実行  
make clean # 実行ファイルを削除

## 学習のポイント

1. **配列の基本操作**: 宣言、初期化、要素アクセス
2. **配列の走査**: forループによる全要素の処理
3. **アルゴリズムの実装**: 検索、ソート、統計処理
4. **多次元配列**: 2次元、3次元配列の効果的な使用
5. **文字列処理**: 文字配列による文字列操作
6. **メモリ効率**: 配列サイズとメモリ使用量の関係

## 配列の重要概念

### インデックス

* 配列の要素は0から始まるインデックスでアクセス
* 配列の境界を超えないよう注意（バッファオーバーラン対策）

### メモリレイアウト

* 配列要素は連続したメモリ領域に配置
* 多次元配列は行優先順序で格納

### 効率的な処理

* キャッシュ効率を考慮したメモリアクセスパターン
* アルゴリズムの時間計算量を意識した実装

## 注意事項

* 配列の境界チェックは必須（C言語は自動チェックしない）
* 初期化されていない配列要素は不定値
* 文字列配列では null終端文字 (‘\0’) を忘れずに

## 次のステップ

この章をマスターしたら、次の章に進みましょう： - [第7章: 文字列](../strings/) - [第8章: 関数](../functions/) - [第9章: ポインタ](../pointers/) ## ex6\_1.c

```c / 演習 6-1: 配列の基本操作（C90準拠版） \* \* 整数配列に対する各種操作（表示、最大値・最小値、 \* 平均値、逆順、検索）を行うプログラム \*/ #include <stdio.h>

#define ARRAY\_SIZE 10

/\* 配列を表示する関数 \*/ void display\_array(int arr[], int size) { int i; printf(“配列の内容:”); for (i = 0; i < size; i++) { printf(“%d”, arr[i]); } printf(“”); }

/\* 最大値を見つける関数 \*/ int find\_max(int arr[], int size) { int max = arr[0]; int i;

for (i = 1; i < size; i++) {  
 if (arr[i] > max) {  
 max = arr[i];  
 }  
}  
return max;

}

/\* 最小値を見つける関数 \*/ int find\_min(int arr[], int size) { int min = arr[0]; int i;

for (i = 1; i < size; i++) {  
 if (arr[i] < min) {  
 min = arr[i];  
 }  
}  
return min;

}

/\* 平均値を計算する関数 \*/ double calculate\_average(int arr[], int size) { int sum = 0; int i;

for (i = 0; i < size; i++) {  
 sum += arr[i];  
}  
return (double)sum / size;

}

/\* 配列を逆順にする関数 \*/ void reverse\_array(int arr[], int size) { int temp; int i;

for (i = 0; i < size / 2; i++) {  
 temp = arr[i];  
 arr[i] = arr[size - 1 - i];  
 arr[size - 1 - i] = temp;  
}

}

/\* 値を検索する関数 \*/ int search\_value(int arr[], int size, int target) { int i;

for (i = 0; i < size; i++) {  
 if (arr[i] == target) {  
 return i; /\* 見つかった位置を返す \*/  
 }  
}  
return -1; /\* 見つからなかった \*/

}

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ int numbers[ARRAY\_SIZE]; int i; int search\_target; int position;

/\* 配列に値を入力 \*/  
printf("=== 配列の基本操作プログラム ===\n");  
printf("%d個の整数を入力してください:\n", ARRAY\_SIZE);  
  
for (i = 0; i < ARRAY\_SIZE; i++) {  
 printf("%d番目の数: ", i + 1);  
 scanf("%d", &numbers[i]);  
}  
  
printf("\n");  
  
/\* 1. 配列の表示 \*/  
display\_array(numbers, ARRAY\_SIZE);  
  
/\* 2. 最大値と最小値 \*/  
printf("\n最大値: %d\n", find\_max(numbers, ARRAY\_SIZE));  
printf("最小値: %d\n", find\_min(numbers, ARRAY\_SIZE));  
  
/\* 3. 平均値 \*/  
printf("平均値: %.2f\n", calculate\_average(numbers, ARRAY\_SIZE));  
  
/\* 4. 逆順に並べ替え \*/  
printf("\n配列を逆順に並べ替えます...\n");  
reverse\_array(numbers, ARRAY\_SIZE);  
display\_array(numbers, ARRAY\_SIZE);  
  
/\* 5. 値の検索 \*/  
printf("\n検索したい値を入力してください: ");  
scanf("%d", &search\_target);  
  
position = search\_value(numbers, ARRAY\_SIZE, search\_target);  
  
if (position != -1) {  
 printf("値 %d は配列の %d 番目（インデックス %d）にあります。\n",   
 search\_target, position + 1, position);  
} else {  
 printf("値 %d は配列内に存在しません。\n", search\_target);  
}  
  
return 0;

}```

## ex6\_2.c

```c / 演習 6-2: 成績管理システム（C90準拠版） \* \* 5人の学生、4科目（国語、数学、英語、理科）の成績を管理し、 \* 各種統計を計算するプログラム \*/ #include <stdio.h>

#define NUM\_STUDENTS 5 #define NUM\_SUBJECTS 4

/\* 学生名と科目名 \*/ char students[NUM\_STUDENTS][20] = { “田中太郎”, “佐藤花子”, “鈴木一郎”, “高橋美咲”, “伊藤健太” };

char subjects[NUM\_SUBJECTS][10] = { “国語”, “数学”, “英語”, “理科” };

/\* 成績表を表示する関数 \*/ void display\_grade\_table(int grades[][NUM\_SUBJECTS]) { int i, j;

printf("\n=== 成績表 ===\n");  
printf("%-12s", "学生名");  
for (j = 0; j < NUM\_SUBJECTS; j++) {  
 printf("%-8s", subjects[j]);  
}  
printf("%-8s%-8s\n", "合計", "平均");  
  
printf("------------------------------------------------\n");  
  
for (i = 0; i < NUM\_STUDENTS; i++) {  
 int total = 0;  
 printf("%-12s", students[i]);  
   
 for (j = 0; j < NUM\_SUBJECTS; j++) {  
 printf("%-8d", grades[i][j]);  
 total += grades[i][j];  
 }  
   
 printf("%-8d%-8.1f\n", total, (double)total / NUM\_SUBJECTS);  
}

}

/\* 各科目の平均を計算・表示する関数 \*/ void display\_subject\_averages(int grades[][NUM\_SUBJECTS]) { int i, j; int total;

printf("\n=== 科目別平均点 ===\n");  
for (j = 0; j < NUM\_SUBJECTS; j++) {  
 total = 0;  
 for (i = 0; i < NUM\_STUDENTS; i++) {  
 total += grades[i][j];  
 }  
 printf("%s: %.1f点\n", subjects[j], (double)total / NUM\_STUDENTS);  
}

}

/\* 最高得点の学生と科目を見つける関数 \*/ void find\_highest\_score(int grades[][NUM\_SUBJECTS]) { int i, j; int max\_score = grades[0][0]; int max\_student = 0; int max\_subject = 0;

for (i = 0; i < NUM\_STUDENTS; i++) {  
 for (j = 0; j < NUM\_SUBJECTS; j++) {  
 if (grades[i][j] > max\_score) {  
 max\_score = grades[i][j];  
 max\_student = i;  
 max\_subject = j;  
 }  
 }  
}  
  
printf("\n=== 最高得点 ===\n");  
printf("学生: %s\n", students[max\_student]);  
printf("科目: %s\n", subjects[max\_subject]);  
printf("得点: %d点\n", max\_score);

}

/\* 学生ランキングを表示する関数 \*/ void display\_student\_ranking(int grades[][NUM\_SUBJECTS]) { int i, j, k; int totals[NUM\_STUDENTS]; int ranking[NUM\_STUDENTS];

/\* 各学生の合計点を計算 \*/  
for (i = 0; i < NUM\_STUDENTS; i++) {  
 totals[i] = 0;  
 for (j = 0; j < NUM\_SUBJECTS; j++) {  
 totals[i] += grades[i][j];  
 }  
 ranking[i] = i;  
}  
  
/\* 簡易バブルソート（合計点の降順） \*/  
for (i = 0; i < NUM\_STUDENTS - 1; i++) {  
 for (j = 0; j < NUM\_STUDENTS - 1 - i; j++) {  
 if (totals[ranking[j]] < totals[ranking[j + 1]]) {  
 int temp = ranking[j];  
 ranking[j] = ranking[j + 1];  
 ranking[j + 1] = temp;  
 }  
 }  
}  
  
printf("\n=== 学生ランキング（合計点順） ===\n");  
for (i = 0; i < NUM\_STUDENTS; i++) {  
 int student\_idx = ranking[i];  
 printf("%d位: %s (%d点, 平均%.1f点)\n",   
 i + 1,   
 students[student\_idx],   
 totals[student\_idx],  
 (double)totals[student\_idx] / NUM\_SUBJECTS);  
}

}

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ int grades[NUM\_STUDENTS][NUM\_SUBJECTS]; int i, j;

printf("=== 成績管理システム ===\n");  
printf("5人の学生の4科目の成績を入力してください。\n\n");  
  
/\* 成績の入力 \*/  
for (i = 0; i < NUM\_STUDENTS; i++) {  
 printf("--- %s の成績 ---\n", students[i]);  
 for (j = 0; j < NUM\_SUBJECTS; j++) {  
 printf("%s: ", subjects[j]);  
 scanf("%d", &grades[i][j]);  
 }  
 printf("\n");  
}  
  
/\* 各種統計の表示 \*/  
display\_grade\_table(grades);  
display\_subject\_averages(grades);  
find\_highest\_score(grades);  
display\_student\_ranking(grades);  
  
return 0;

}```

## ex6\_3.c

```c / 演習 6-3: 文字列配列の操作（C90準拠版） \* \* 都道府県名を管理し、各種操作（ソート、検索、 \* 文字数カウント等）を行うプログラム \*/ #include <stdio.h> #include <string.h>

#define NUM\_PREFECTURES 10 #define MAX\_NAME\_LENGTH 20

/\* 都道府県名のサンプルデータ \*/ char prefectures[NUM\_PREFECTURES][MAX\_NAME\_LENGTH] = { “東京都”, “神奈川県”, “大阪府”, “愛知県”, “埼玉県”, “千葉県”, “兵庫県”, “北海道”, “福岡県”, “静岡県” };

/\* 文字列配列をアルファベット順（辞書順）にソートする関数 \*/ void sort\_prefectures(char arr[][MAX\_NAME\_LENGTH], int size) { char temp[MAX\_NAME\_LENGTH]; int i, j;

/\* バブルソート \*/  
for (i = 0; i < size - 1; i++) {  
 for (j = 0; j < size - 1 - i; j++) {  
 if (strcmp(arr[j], arr[j + 1]) > 0) {  
 strcpy(temp, arr[j]);  
 strcpy(arr[j], arr[j + 1]);  
 strcpy(arr[j + 1], temp);  
 }  
 }  
}

}

/\* 全都道府県名を表示する関数 \*/ void display\_all\_prefectures(char arr[][MAX\_NAME\_LENGTH], int size) { int i;

printf("=== 都道府県一覧 ===\n");  
for (i = 0; i < size; i++) {  
 printf("%2d. %s\n", i + 1, arr[i]);  
}  
printf("\n");

}

/\* 最も文字数が長い都道府県名を見つける関数 \*/ void find\_longest\_name(char arr[][MAX\_NAME\_LENGTH], int size) { int i; int max\_length = 0; int max\_index = 0;

for (i = 0; i < size; i++) {  
 int length = strlen(arr[i]);  
 if (length > max\_length) {  
 max\_length = length;  
 max\_index = i;  
 }  
}  
  
printf("=== 最も文字数が長い都道府県名 ===\n");  
printf("都道府県名: %s\n", arr[max\_index]);  
printf("文字数: %d文字\n\n", max\_length);

}

/\* 指定した文字で始まる都道府県名を検索する関数 \*/ void search\_by\_first\_char(char arr[][MAX\_NAME\_LENGTH], int size, char first\_char) { int i; int found = 0;

printf("=== '%c'で始まる都道府県名 ===\n", first\_char);  
  
for (i = 0; i < size; i++) {  
 if (arr[i][0] == first\_char) {  
 printf("- %s\n", arr[i]);  
 found = 1;  
 }  
}  
  
if (!found) {  
 printf("見つかりませんでした。\n");  
}  
printf("\n");

}

/\* 「県」が含まれる都道府県をカウントする関数 \*/ int count\_ken\_prefectures(char arr[][MAX\_NAME\_LENGTH], int size) { int i; int count = 0;

for (i = 0; i < size; i++) {  
 if (strstr(arr[i], "県") != NULL) {  
 count++;  
 }  
}  
  
return count;

}

/\* 都道府県の種類別カウントを表示する関数 \*/ void display\_type\_count(char arr[][MAX\_NAME\_LENGTH], int size) { int i; int ken\_count = 0; int fu\_count = 0; int to\_count = 0; int do\_count = 0;

for (i = 0; i < size; i++) {  
 if (strstr(arr[i], "県") != NULL) {  
 ken\_count++;  
 } else if (strstr(arr[i], "府") != NULL) {  
 fu\_count++;  
 } else if (strstr(arr[i], "都") != NULL) {  
 to\_count++;  
 } else if (strstr(arr[i], "道") != NULL) {  
 do\_count++;  
 }  
}  
  
printf("=== 都道府県の種類別カウント ===\n");  
printf("県: %d個\n", ken\_count);  
printf("府: %d個\n", fu\_count);  
printf("都: %d個\n", to\_count);  
printf("道: %d個\n", do\_count);  
printf("\n");

}

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ char working\_array[NUM\_PREFECTURES][MAX\_NAME\_LENGTH]; char search\_char; int i; int ken\_count;

printf("=== 都道府県管理システム ===\n\n");  
  
/\* 作業用配列にコピー（元データを保持するため） \*/  
for (i = 0; i < NUM\_PREFECTURES; i++) {  
 strcpy(working\_array[i], prefectures[i]);  
}  
  
/\* 元の順序で表示 \*/  
printf("--- 元の順序 ---\n");  
display\_all\_prefectures(prefectures, NUM\_PREFECTURES);  
  
/\* ソートして表示 \*/  
sort\_prefectures(working\_array, NUM\_PREFECTURES);  
printf("--- 辞書順ソート後 ---\n");  
display\_all\_prefectures(working\_array, NUM\_PREFECTURES);  
  
/\* 最も長い都道府県名を表示 \*/  
find\_longest\_name(prefectures, NUM\_PREFECTURES);  
  
/\* 指定文字で始まる都道府県を検索 \*/  
printf("検索したい最初の文字を入力してください: ");  
scanf(" %c", &search\_char); /\* 先頭の空白でバッファをクリア \*/  
search\_by\_first\_char(prefectures, NUM\_PREFECTURES, search\_char);  
  
/\* 「県」を含む都道府県をカウント \*/  
ken\_count = count\_ken\_prefectures(prefectures, NUM\_PREFECTURES);  
printf("「県」が含まれる都道府県: %d個\n\n", ken\_count);  
  
/\* 種類別カウントを表示 \*/  
display\_type\_count(prefectures, NUM\_PREFECTURES);  
  
return 0;

}```

## ex6\_4.c

```c / 演習 6-4: 行列演算（C90準拠版） \* \* 2次元配列を使って行列演算（加算、減算、乗算、転置）を \* 実行するプログラム \*/ #include <stdio.h>

#define ROWS 3 #define COLS 3

/\* 行列を表示する関数 */ void display\_matrix(int matrix[][COLS], int rows, int cols, const char* title) { int i, j;

printf("=== %s ===\n", title);  
for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 printf("%4d ", matrix[i][j]);  
 }  
 printf("\n");  
}  
printf("\n");

}

/\* 行列の加算を行う関数 \*/ void add\_matrices(int a[][COLS], int b[][COLS], int result[][COLS], int rows, int cols) { int i, j;

for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 result[i][j] = a[i][j] + b[i][j];  
 }  
}

}

/\* 行列の減算を行う関数 \*/ void subtract\_matrices(int a[][COLS], int b[][COLS], int result[][COLS], int rows, int cols) { int i, j;

for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 result[i][j] = a[i][j] - b[i][j];  
 }  
}

}

/\* 行列の乗算を行う関数 \*/ void multiply\_matrices(int a[][COLS], int b[][COLS], int result[][COLS], int rows, int cols) { int i, j, k;

/\* 結果行列を初期化 \*/  
for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 result[i][j] = 0;  
 }  
}  
  
/\* 行列の乗算: result[i][j] = Σ(a[i][k] \* b[k][j]) \*/  
for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 for (k = 0; k < cols; k++) {  
 result[i][j] += a[i][k] \* b[k][j];  
 }  
 }  
}

}

/\* 転置行列を計算する関数 \*/ void transpose\_matrix(int matrix[][COLS], int result[][ROWS], int rows, int cols) { int i, j;

for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 result[j][i] = matrix[i][j];  
 }  
}

}

/\* 転置行列専用の表示関数（行と列が入れ替わる） */ void display\_transposed\_matrix(int matrix[][ROWS], int rows, int cols, const char* title) { int i, j;

printf("=== %s ===\n", title);  
for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 printf("%4d ", matrix[i][j]);  
 }  
 printf("\n");  
}  
printf("\n");

}

/\* 行列に値を入力する関数 */ void input\_matrix(int matrix[][COLS], int rows, int cols, const char* name) { int i, j;

printf("=== %s の入力 ===\n", name);  
for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 printf("%s[%d][%d]: ", name, i, j);  
 scanf("%d", &matrix[i][j]);  
 }  
}  
printf("\n");

}

int main(void) { /\* C90では先頭で全ての変数を宣言 */ int matrix\_a[ROWS][COLS]; int matrix\_b[ROWS][COLS]; int result[ROWS][COLS]; int transposed[COLS][ROWS]; /* 転置行列は行と列が入れ替わる \*/

printf("=== 行列演算プログラム ===\n");  
printf("3x3の行列2つで各種演算を行います。\n\n");  
  
/\* 行列の入力 \*/  
input\_matrix(matrix\_a, ROWS, COLS, "行列A");  
input\_matrix(matrix\_b, ROWS, COLS, "行列B");  
  
/\* 入力された行列を表示 \*/  
display\_matrix(matrix\_a, ROWS, COLS, "行列A");  
display\_matrix(matrix\_b, ROWS, COLS, "行列B");  
  
/\* 行列の加算 \*/  
add\_matrices(matrix\_a, matrix\_b, result, ROWS, COLS);  
display\_matrix(result, ROWS, COLS, "行列A + 行列B");  
  
/\* 行列の減算 \*/  
subtract\_matrices(matrix\_a, matrix\_b, result, ROWS, COLS);  
display\_matrix(result, ROWS, COLS, "行列A - 行列B");  
  
/\* 行列の乗算 \*/  
multiply\_matrices(matrix\_a, matrix\_b, result, ROWS, COLS);  
display\_matrix(result, ROWS, COLS, "行列A x 行列B");  
  
/\* 行列Aの転置 \*/  
transpose\_matrix(matrix\_a, transposed, ROWS, COLS);  
display\_transposed\_matrix(transposed, COLS, ROWS, "行列Aの転置");  
  
/\* 行列Bの転置 \*/  
transpose\_matrix(matrix\_b, transposed, ROWS, COLS);  
display\_transposed\_matrix(transposed, COLS, ROWS, "行列Bの転置");  
  
return 0;

}```

## ex6\_5.c

```c / 演習 6-5: 簡易辞書プログラム（C90準拠版） \* \* 英単語と日本語訳のペアを管理する辞書プログラム \* 英和・和英検索、一覧表示、単語追加機能を実装 \*/ #include <stdio.h> #include <string.h>

#define MAX\_WORDS 50 #define MAX\_WORD\_LENGTH 50

/\* 辞書データ構造 \*/ char english\_words[MAX\_WORDS][MAX\_WORD\_LENGTH]; char japanese\_words[MAX\_WORDS][MAX\_WORD\_LENGTH]; int word\_count = 0;

/\* 初期データを設定する関数 \*/ void initialize\_dictionary(void) { strcpy(english\_words[0], “apple”); strcpy(japanese\_words[0], “りんご”);

strcpy(english\_words[1], "book");  
strcpy(japanese\_words[1], "本");  
  
strcpy(english\_words[2], "cat");  
strcpy(japanese\_words[2], "猫");  
  
strcpy(english\_words[3], "dog");  
strcpy(japanese\_words[3], "犬");  
  
strcpy(english\_words[4], "elephant");  
strcpy(japanese\_words[4], "象");  
  
strcpy(english\_words[5], "flower");  
strcpy(japanese\_words[5], "花");  
  
strcpy(english\_words[6], "guitar");  
strcpy(japanese\_words[6], "ギター");  
  
strcpy(english\_words[7], "house");  
strcpy(japanese\_words[7], "家");  
  
strcpy(english\_words[8], "internet");  
strcpy(japanese\_words[8], "インターネット");  
  
strcpy(english\_words[9], "japan");  
strcpy(japanese\_words[9], "日本");  
  
word\_count = 10;

}

/\* 全単語を一覧表示する関数 \*/ void display\_all\_words(void) { int i;

printf("\n=== 辞書一覧 ===\n");  
printf("%-3s %-20s %-20s\n", "No.", "English", "Japanese");  
printf("--------------------------------------------\n");  
  
for (i = 0; i < word\_count; i++) {  
 printf("%-3d %-20s %-20s\n", i + 1, english\_words[i], japanese\_words[i]);  
}  
printf("\n");

}

/\* 英単語で検索する関数（英和） \*/ void search\_english\_to\_japanese(void) { char search\_word[MAX\_WORD\_LENGTH]; int i; int found = 0;

printf("英単語を入力してください: ");  
scanf("%s", search\_word);  
  
for (i = 0; i < word\_count; i++) {  
 if (strcmp(english\_words[i], search\_word) == 0) {  
 printf("結果: %s = %s\n\n", english\_words[i], japanese\_words[i]);  
 found = 1;  
 break;  
 }  
}  
  
if (!found) {  
 printf("「%s」は見つかりませんでした。\n\n", search\_word);  
}

}

/\* 日本語で検索する関数（和英） \*/ void search\_japanese\_to\_english(void) { char search\_word[MAX\_WORD\_LENGTH]; int i; int found = 0;

printf("日本語を入力してください: ");  
scanf("%s", search\_word);  
  
for (i = 0; i < word\_count; i++) {  
 if (strcmp(japanese\_words[i], search\_word) == 0) {  
 printf("結果: %s = %s\n\n", japanese\_words[i], english\_words[i]);  
 found = 1;  
 break;  
 }  
}  
  
if (!found) {  
 printf("「%s」は見つかりませんでした。\n\n", search\_word);  
}

}

/\* 新しい単語を追加する関数 \*/ void add\_new\_word(void) { char new\_english[MAX\_WORD\_LENGTH]; char new\_japanese[MAX\_WORD\_LENGTH]; int i;

if (word\_count >= MAX\_WORDS) {  
 printf("辞書が満杯です。これ以上追加できません。\n\n");  
 return;  
}  
  
printf("新しい英単語を入力してください: ");  
scanf("%s", new\_english);  
  
/\* 重複チェック \*/  
for (i = 0; i < word\_count; i++) {  
 if (strcmp(english\_words[i], new\_english) == 0) {  
 printf("その単語は既に登録されています。\n\n");  
 return;  
 }  
}  
  
printf("日本語訳を入力してください: ");  
scanf("%s", new\_japanese);  
  
/\* 新しい単語を追加 \*/  
strcpy(english\_words[word\_count], new\_english);  
strcpy(japanese\_words[word\_count], new\_japanese);  
word\_count++;  
  
printf("「%s = %s」を追加しました。\n\n", new\_english, new\_japanese);

}

/\* メニューを表示する関数 \*/ void display\_menu(void) { printf(“=== 簡易辞書プログラム ===”); printf(“1. 英和検索（英語→日本語）”); printf(“2. 和英検索（日本語→英語）”); printf(“3. 辞書一覧表示”); printf(“4. 新しい単語を追加”); printf(“5. 終了”); printf(“選択してください (1-5):”); }

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ int choice;

/\* 初期データを設定 \*/  
initialize\_dictionary();  
  
while (1) {  
 display\_menu();  
 scanf("%d", &choice);  
 printf("\n");  
   
 switch (choice) {  
 case 1:  
 search\_english\_to\_japanese();  
 break;  
 case 2:  
 search\_japanese\_to\_english();  
 break;  
 case 3:  
 display\_all\_words();  
 break;  
 case 4:  
 add\_new\_word();  
 break;  
 case 5:  
 printf("辞書プログラムを終了します。\n");  
 return 0;  
 default:  
 printf("無効な選択です。1-5の数字を入力してください。\n\n");  
 break;  
 }  
}  
  
return 0;

}```

## ex6\_challenge1.c

```c / チャレンジ問題 1: ソートアルゴリズムの実装と比較（C90準拠版） \* \* バブルソート、選択ソート、挿入ソートを実装し、 \* 実行時間と比較・交換回数を測定して性能を比較する \*/ #include <stdio.h> #include <stdlib.h> #include <time.h> #include <string.h>

#define ARRAY\_SIZE 1000 #define MAX\_BAR\_LENGTH 50

/\* ソート統計情報の構造 \*/ typedef struct { long comparisons; long swaps; clock\_t start\_time; clock\_t end\_time; } SortStats;

/\* 配列をコピーする関数 \*/ void copy\_array(int source[], int dest[], int size) { int i; for (i = 0; i < size; i++) { dest[i] = source[i]; } }

/\* 配列をランダムな値で初期化する関数 \*/ void randomize\_array(int arr[], int size) { int i; for (i = 0; i < size; i++) { arr[i] = rand() % 1000; } }

/\* 配列をソート済みの状態にする関数 \*/ void sorted\_array(int arr[], int size) { int i; for (i = 0; i < size; i++) { arr[i] = i; } }

/\* 配列を逆順にする関数 \*/ void reverse\_array(int arr[], int size) { int i; for (i = 0; i < size; i++) { arr[i] = size - i - 1; } }

/\* バブルソート */ void bubble\_sort(int arr[], int size, SortStats* stats) { int i, j; int temp;

stats->comparisons = 0;  
stats->swaps = 0;  
stats->start\_time = clock();  
  
for (i = 0; i < size - 1; i++) {  
 for (j = 0; j < size - 1 - i; j++) {  
 stats->comparisons++;  
 if (arr[j] > arr[j + 1]) {  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 stats->swaps++;  
 }  
 }  
}  
  
stats->end\_time = clock();

}

/\* 選択ソート */ void selection\_sort(int arr[], int size, SortStats* stats) { int i, j, min\_idx; int temp;

stats->comparisons = 0;  
stats->swaps = 0;  
stats->start\_time = clock();  
  
for (i = 0; i < size - 1; i++) {  
 min\_idx = i;  
 for (j = i + 1; j < size; j++) {  
 stats->comparisons++;  
 if (arr[j] < arr[min\_idx]) {  
 min\_idx = j;  
 }  
 }  
 if (min\_idx != i) {  
 temp = arr[i];  
 arr[i] = arr[min\_idx];  
 arr[min\_idx] = temp;  
 stats->swaps++;  
 }  
}  
  
stats->end\_time = clock();

}

/\* 挿入ソート */ void insertion\_sort(int arr[], int size, SortStats* stats) { int i, j; int key;

stats->comparisons = 0;  
stats->swaps = 0;  
stats->start\_time = clock();  
  
for (i = 1; i < size; i++) {  
 key = arr[i];  
 j = i - 1;  
   
 while (j >= 0 && arr[j] > key) {  
 stats->comparisons++;  
 arr[j + 1] = arr[j];  
 stats->swaps++;  
 j--;  
 }  
 if (j >= 0) {  
 stats->comparisons++; /\* 最後の比較もカウント \*/  
 }  
 arr[j + 1] = key;  
}  
  
stats->end\_time = clock();

}

/\* 実行時間を計算する関数（ミリ秒） */ double calculate\_time\_ms(SortStats* stats) { return ((double)(stats->end\_time - stats->start\_time)) / CLOCKS\_PER\_SEC \* 1000.0; }

/\* 簡易グラフを表示する関数 */ void draw\_bar\_graph(const char* label, long value, long max\_value) { int bar\_length; int i;

if (max\_value == 0) {  
 bar\_length = 0;  
} else {  
 bar\_length = (int)((double)value / max\_value \* MAX\_BAR\_LENGTH);  
}  
  
printf("%-15s [", label);  
for (i = 0; i < bar\_length; i++) {  
 printf("\*");  
}  
for (i = bar\_length; i < MAX\_BAR\_LENGTH; i++) {  
 printf(" ");  
}  
printf("] %ld\n", value);

}

/\* ソート結果を表示する関数 */ void display\_results(const char* test\_name, SortStats bubble, SortStats selection, SortStats insertion) { long max\_comparisons, max\_swaps; double max\_time;

printf("\n=== %s ===\n", test\_name);  
  
/\* 最大値を計算（グラフ表示用） \*/  
max\_comparisons = bubble.comparisons;  
if (selection.comparisons > max\_comparisons) max\_comparisons = selection.comparisons;  
if (insertion.comparisons > max\_comparisons) max\_comparisons = insertion.comparisons;  
  
max\_swaps = bubble.swaps;  
if (selection.swaps > max\_swaps) max\_swaps = selection.swaps;  
if (insertion.swaps > max\_swaps) max\_swaps = insertion.swaps;  
  
max\_time = calculate\_time\_ms(&bubble);  
if (calculate\_time\_ms(&selection) > max\_time) max\_time = calculate\_time\_ms(&selection);  
if (calculate\_time\_ms(&insertion) > max\_time) max\_time = calculate\_time\_ms(&insertion);  
  
/\* 比較回数のグラフ \*/  
printf("\n--- 比較回数 ---\n");  
draw\_bar\_graph("Bubble", bubble.comparisons, max\_comparisons);  
draw\_bar\_graph("Selection", selection.comparisons, max\_comparisons);  
draw\_bar\_graph("Insertion", insertion.comparisons, max\_comparisons);  
  
/\* 交換回数のグラフ \*/  
printf("\n--- 交換回数 ---\n");  
draw\_bar\_graph("Bubble", bubble.swaps, max\_swaps);  
draw\_bar\_graph("Selection", selection.swaps, max\_swaps);  
draw\_bar\_graph("Insertion", insertion.swaps, max\_swaps);  
  
/\* 実行時間の詳細 \*/  
printf("\n--- 実行時間 ---\n");  
printf("Bubble Sort: %.3f ms\n", calculate\_time\_ms(&bubble));  
printf("Selection Sort: %.3f ms\n", calculate\_time\_ms(&selection));  
printf("Insertion Sort: %.3f ms\n", calculate\_time\_ms(&insertion));

}

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ int original\_array[ARRAY\_SIZE]; int test\_array[ARRAY\_SIZE]; SortStats bubble\_stats, selection\_stats, insertion\_stats;

/\* 乱数の種を設定 \*/  
srand((unsigned int)time(NULL));  
  
printf("=== ソートアルゴリズム性能比較プログラム ===\n");  
printf("配列サイズ: %d\n", ARRAY\_SIZE);  
  
/\* テスト1: ランダム配列 \*/  
randomize\_array(original\_array, ARRAY\_SIZE);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
bubble\_sort(test\_array, ARRAY\_SIZE, &bubble\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
selection\_sort(test\_array, ARRAY\_SIZE, &selection\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
insertion\_sort(test\_array, ARRAY\_SIZE, &insertion\_stats);  
  
display\_results("ランダム配列", bubble\_stats, selection\_stats, insertion\_stats);  
  
/\* テスト2: ソート済み配列 \*/  
sorted\_array(original\_array, ARRAY\_SIZE);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
bubble\_sort(test\_array, ARRAY\_SIZE, &bubble\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
selection\_sort(test\_array, ARRAY\_SIZE, &selection\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
insertion\_sort(test\_array, ARRAY\_SIZE, &insertion\_stats);  
  
display\_results("ソート済み配列", bubble\_stats, selection\_stats, insertion\_stats);  
  
/\* テスト3: 逆順配列 \*/  
reverse\_array(original\_array, ARRAY\_SIZE);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
bubble\_sort(test\_array, ARRAY\_SIZE, &bubble\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
selection\_sort(test\_array, ARRAY\_SIZE, &selection\_stats);  
  
copy\_array(original\_array, test\_array, ARRAY\_SIZE);  
insertion\_sort(test\_array, ARRAY\_SIZE, &insertion\_stats);  
  
display\_results("逆順配列", bubble\_stats, selection\_stats, insertion\_stats);  
  
printf("\n=== まとめ ===\n");  
printf("一般的に：\n");  
printf("- 挿入ソート：少ないデータやほぼソート済みデータに効率的\n");  
printf("- 選択ソート：交換回数が少ない\n");  
printf("- バブルソート：理解しやすいが効率は劣る\n");  
  
return 0;

}```

## ex6\_challenge2.c

```c / チャレンジ問題 2: ライフゲーム（Conway’s Game of Life）（C90準拠版） \* \* セル・オートマトンの一種であるライフゲームを実装する \* 20x20のグリッドで実装し、世代を進めるごとに画面を更新 \*/ #include <stdio.h> #include <stdlib.h> #include <string.h>

#define GRID\_SIZE 20 #define MAX\_GENERATIONS 100

/\* グリッドの状態を表示する関数 \*/ void display\_grid(int grid[][GRID\_SIZE], int generation) { int i, j;

/\* 画面をクリア（簡易的な方法） \*/  
printf("\n\n=== ライフゲーム - 第%d世代 ===\n", generation);  
printf(" ");  
for (j = 0; j < GRID\_SIZE; j++) {  
 printf("%2d", j % 10);  
}  
printf("\n");  
  
for (i = 0; i < GRID\_SIZE; i++) {  
 printf("%2d", i % 10);  
 for (j = 0; j < GRID\_SIZE; j++) {  
 if (grid[i][j] == 1) {  
 printf(" \*"); /\* 生きているセル \*/  
 } else {  
 printf(" "); /\* 死んでいるセル \*/  
 }  
 }  
 printf("\n");  
}  
printf("\n");

}

/\* 指定位置の周囲の生きているセルの数を数える関数 */ int count\_neighbors(int grid[][GRID\_SIZE], int row, int col) { int i, j; int count = 0; int dr[] = {-1, -1, -1, 0, 0, 1, 1, 1}; /* 行の変位 */ int dc[] = {-1, 0, 1, -1, 1, -1, 0, 1}; /* 列の変位 \*/

for (i = 0; i < 8; i++) {  
 int new\_row = row + dr[i];  
 int new\_col = col + dc[i];  
   
 /\* 境界チェック \*/  
 if (new\_row >= 0 && new\_row < GRID\_SIZE &&   
 new\_col >= 0 && new\_col < GRID\_SIZE) {  
 count += grid[new\_row][new\_col];  
 }  
}  
  
return count;

}

/\* 次世代の状態を計算する関数 \*/ void next\_generation(int current[][GRID\_SIZE], int next[][GRID\_SIZE]) { int i, j; int neighbors;

for (i = 0; i < GRID\_SIZE; i++) {  
 for (j = 0; j < GRID\_SIZE; j++) {  
 neighbors = count\_neighbors(current, i, j);  
   
 /\* ライフゲームのルール \*/  
 if (current[i][j] == 1) { /\* 生きているセル \*/  
 if (neighbors == 2 || neighbors == 3) {  
 next[i][j] = 1; /\* 生存 \*/  
 } else {  
 next[i][j] = 0; /\* 死亡 \*/  
 }  
 } else { /\* 死んでいるセル \*/  
 if (neighbors == 3) {  
 next[i][j] = 1; /\* 誕生 \*/  
 } else {  
 next[i][j] = 0; /\* 死亡のまま \*/  
 }  
 }  
 }  
}

}

/\* グリッドをクリアする関数 \*/ void clear\_grid(int grid[][GRID\_SIZE]) { int i, j; for (i = 0; i < GRID\_SIZE; i++) { for (j = 0; j < GRID\_SIZE; j++) { grid[i][j] = 0; } } }

/\* グリッドをコピーする関数 \*/ void copy\_grid(int source[][GRID\_SIZE], int dest[][GRID\_SIZE]) { int i, j; for (i = 0; i < GRID\_SIZE; i++) { for (j = 0; j < GRID\_SIZE; j++) { dest[i][j] = source[i][j]; } } }

/\* 振動子パターン（ビーコン）を設定する関数 \*/ void set\_beacon\_pattern(int grid[][GRID\_SIZE]) { clear\_grid(grid);

/\* ビーコンパターン \*/  
grid[5][5] = 1; grid[5][6] = 1;  
grid[6][5] = 1; grid[6][6] = 1;  
grid[7][7] = 1; grid[7][8] = 1;  
grid[8][7] = 1; grid[8][8] = 1;

}

/\* グライダーパターンを設定する関数 \*/ void set\_glider\_pattern(int grid[][GRID\_SIZE]) { clear\_grid(grid);

/\* グライダーパターン \*/  
grid[1][2] = 1;  
grid[2][3] = 1;  
grid[3][1] = 1; grid[3][2] = 1; grid[3][3] = 1;

}

/\* ランダムパターンを設定する関数 \*/ void set\_random\_pattern(int grid[][GRID\_SIZE], int density) { int i, j;

clear\_grid(grid);  
  
for (i = 0; i < GRID\_SIZE; i++) {  
 for (j = 0; j < GRID\_SIZE; j++) {  
 if (rand() % 100 < density) {  
 grid[i][j] = 1;  
 }  
 }  
}

}

/\* カスタムパターンを入力する関数 \*/ void set\_custom\_pattern(int grid[][GRID\_SIZE]) { int row, col; char input[10];

clear\_grid(grid);  
  
printf("カスタムパターンを設定します。\n");  
printf("生きているセルの座標を入力してください（0-%d）\n", GRID\_SIZE-1);  
printf("終了するには -1 -1 を入力してください。\n\n");  
  
while (1) {  
 printf("行 列: ");  
 if (scanf("%d %d", &row, &col) != 2) {  
 /\* 入力エラーの場合、バッファをクリア \*/  
 while (getchar() != '\n');  
 continue;  
 }  
   
 if (row == -1 && col == -1) {  
 break;  
 }  
   
 if (row >= 0 && row < GRID\_SIZE && col >= 0 && col < GRID\_SIZE) {  
 grid[row][col] = 1;  
 printf("セル[%d][%d]を生きた状態に設定しました。\n", row, col);  
 } else {  
 printf("無効な座標です。0-%d の範囲で入力してください。\n", GRID\_SIZE-1);  
 }  
}

}

/\* メニューを表示する関数 \*/ int display\_menu(void) { int choice;

printf("=== ライフゲーム ===\n");  
printf("1. グライダー（移動するパターン）\n");  
printf("2. ビーコン（振動するパターン）\n");  
printf("3. ランダムパターン\n");  
printf("4. カスタムパターン\n");  
printf("5. 終了\n");  
printf("選択してください (1-5): ");  
  
scanf("%d", &choice);  
return choice;

}

/\* 2つのグリッドが同じかどうかチェックする関数 \*/ int grids\_equal(int grid1[][GRID\_SIZE], int grid2[][GRID\_SIZE]) { int i, j;

for (i = 0; i < GRID\_SIZE; i++) {  
 for (j = 0; j < GRID\_SIZE; j++) {  
 if (grid1[i][j] != grid2[i][j]) {  
 return 0;  
 }  
 }  
}  
return 1;

}

int main(void) { /\* C90では先頭で全ての変数を宣言 \*/ int current\_grid[GRID\_SIZE][GRID\_SIZE]; int next\_grid[GRID\_SIZE][GRID\_SIZE]; int previous\_grid[GRID\_SIZE][GRID\_SIZE]; int choice; int generation; char input[10];

srand((unsigned int)time(NULL));  
  
while (1) {  
 choice = display\_menu();  
   
 switch (choice) {  
 case 1:  
 set\_glider\_pattern(current\_grid);  
 break;  
 case 2:  
 set\_beacon\_pattern(current\_grid);  
 break;  
 case 3:  
 printf("セルの密度を入力してください (1-50): ");  
 scanf("%d", &generation); /\* 一時的にgeneration変数を使用 \*/  
 if (generation < 1) generation = 1;  
 if (generation > 50) generation = 50;  
 set\_random\_pattern(current\_grid, generation);  
 break;  
 case 4:  
 set\_custom\_pattern(current\_grid);  
 break;  
 case 5:  
 printf("ライフゲームを終了します。\n");  
 return 0;  
 default:  
 printf("無効な選択です。\n");  
 continue;  
 }  
   
 /\* シミュレーション開始 \*/  
 copy\_grid(current\_grid, previous\_grid);  
   
 for (generation = 0; generation < MAX\_GENERATIONS; generation++) {  
 display\_grid(current\_grid, generation);  
   
 printf("Enterキーで次世代へ、'q'で戻る: ");  
 fgets(input, sizeof(input), stdin);  
 if (input[0] == '\n') {  
 fgets(input, sizeof(input), stdin); /\* 実際の入力を読む \*/  
 }  
   
 if (input[0] == 'q' || input[0] == 'Q') {  
 break;  
 }  
   
 /\* 次世代を計算 \*/  
 next\_generation(current\_grid, next\_grid);  
   
 /\* 変化がないか、前の状態に戻った場合は終了 \*/  
 if (grids\_equal(current\_grid, next\_grid)) {  
 printf("安定状態に達しました。\n");  
 break;  
 }  
 if (grids\_equal(next\_grid, previous\_grid)) {  
 printf("振動状態に達しました。\n");  
 break;  
 }  
   
 /\* グリッドを更新 \*/  
 copy\_grid(current\_grid, previous\_grid);  
 copy\_grid(next\_grid, current\_grid);  
 }  
   
 if (generation >= MAX\_GENERATIONS) {  
 printf("最大世代数に達しました。\n");  
 }  
   
 printf("\nメニューに戻ります...\n\n");  
}  
  
return 0;

}```