# 第7章 配列

## 対応C規格

* **主要対象:** C90
* **学習内容:** 次元配列、多次元配列、文字列配列、配列の初期化、配列とポインタの関係

## 学習目標

この章を完了すると、以下のことができるようになります：

* 次元配列の宣言・初期化・使用ができる
* 多次元配列を理解して活用できる
* 文字列配列の扱い方を完全に理解する
* char配列とchar\*配列の違いを把握する
* 配列を使った実践的なプログラムを作成できる

## 概要と詳細

### 配列とは？

配列（はいれつ）は、プログラミングで最も重要なデータ構造の一つです。

#### 日常生活での配列

配列を理解するために、身近な例を考えてみましょう：

1. **教室の座席**
   * 5行6列の座席 = 2次元配列
   * 各座席には生徒が座る = 配列の要素
   * 「3行目の4番目の席」= 配列の[3][4]
2. **アパートの部屋番号**
   * 101号室、102号室… = 1次元配列
   * 各部屋に住人がいる = 配列の要素
3. **カレンダー**
   * 曜日×週 = 2次元配列
   * 各マスに日付 = 配列の要素

#### なぜ配列が必要なの？

配列がないと、たくさんのデータを扱うのが大変になります：

/\* 配列を使わない場合（5人の点数） \*/  
int score1 = 85;  
int score2 = 92;  
int score3 = 78;  
int score4 = 91;  
int score5 = 88;  
  
/\* 配列を使う場合 \*/  
int scores[5] = {85, 92, 78, 91, 88};

配列を使えば、100人でも1000人でも簡単に管理できます！

### 配列の基本概念

配列は同じデータ型の要素を連続したメモリ領域に格納するデータ構造です。

#### 次元配列の宣言

データ型 配列名[要素数];

#### 基本的な配列の使用例

#include <stdio.h>  
  
int main(void)  
{  
  
 int numbers[5]; /\* 5個のint型要素を持つ配列 \*/  
 int i;  
   
 /\* 配列への値の代入 \*/  
 numbers[0] = 10;  
 numbers[1] = 20;  
 numbers[2] = 30;  
 numbers[3] = 40;  
 numbers[4] = 50;  
   
 /\* 配列の値を出力 \*/  
 for (i = 0; i < 5; i++) {  
 printf("numbers[%d] = %d\n", i, numbers[i]);  
 }  
   
   
 return 0;

### 配列の初期化

配列を使うときは、最初に値を設定（初期化）することが大切です。

#### 初心者が陥りやすいミス

int numbers[5]; /\* 初期化なし \*/  
printf("%d", numbers[0]); /\* 何が表示される？ \*/

初期化していない配列には「ゴミ値」が入っています。必ず初期化しましょう！

#### 宣言時の初期化

/\* 方法1: 全要素を明示的に初期化 \*/  
int scores[5] = {85, 92, 78, 91, 88};  
  
/\* 方法2: 部分的な初期化（残りは0で初期化） \*/  
int values[10] = {1, 2, 3}; /\* values[0]=1, values[1]=2, values[2]=3, 残りは0 \*/  
  
/\* 方法3: サイズを省略（要素数から自動決定） \*/  
int grades[] = {95, 87, 92, 76, 89}; /\* サイズは5になる \*/  
  
/\* 方法4: 全要素を0で初期化 \*/  
int zeros[10] = {0}; /\* 全要素が0 \*/

#### C90での初期化の制限

int main(void)  
{  
  
 int size = 5;  
 /\* int arr[size]; <- C90では不可（Variable Length Array） \*/  
   
 /\* C90では定数でサイズを指定 \*/  
 #define ARRAY\_SIZE 5  
 int arr[ARRAY\_SIZE] = {1, 2, 3, 4, 5};  
   
 return 0;

### 配列の操作

#### 配列要素へのアクセス

#include <stdio.h>  
  
int main(void)  
{  
  
 int data[5] = {10, 20, 30, 40, 50};  
 int i;  
   
 /\* 読み取り \*/  
 printf("3番目の要素: %d\n", data[2]); /\* 30が出力 \*/  
   
 /\* 書き込み \*/  
 data[2] = 35;  
   
 /\* ループでのアクセス \*/  
 for (i = 0; i < 5; i++) {  
 data[i] = data[i] \* 2; /\* 各要素を2倍 \*/  
 }  
   
   
 return 0;

#### 配列のサイズ計算

#include <stdio.h>  
  
int main(void)  
{  
  
 int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 int size;  
 int i;  
   
 /\* 配列のサイズを計算 \*/  
 size = sizeof(numbers) / sizeof(numbers[0]);  
   
 printf("配列のサイズ: %d\n", size); /\* 10が出力 \*/  
   
 for (i = 0; i < size; i++) {  
 printf("%d ", numbers[i]);  
 }  
   
 printf("\n");  
   
 return 0;

### 文字配列（文字列）

C言語では、文字列は文字（char）の配列として扱います。これは初心者にとって重要な概念です。

#### 文字列の仕組み

文字列は「文字の並び + 終端文字（\0）」です：

H e l l o \0  
↑ ↑ ↑ ↑ ↑ ↑  
0 1 2 3 4 5番目

終端文字（\0）は「ここで文字列が終わり」という目印です。

#### char配列による文字列

#include <stdio.h>  
  
int main(void)  
{  
  
 char str1[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; /\* 明示的初期化 \*/  
 char str2[6] = "Hello"; /\* 文字列リテラル \*/  
 char str3[] = "Hello"; /\* サイズ自動決定 \*/  
   
 printf("str1: %s\n", str1);  
 printf("str2: %s\n", str2);  
 printf("str3: %s\n", str3);  
   
 return 0;

#### 文字配列の操作

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 char buffer[50];  
 char name[20];  
 int i;  
   
 /\* 文字列のコピー \*/  
 strcpy(buffer, "Hello, World!");  
 printf("buffer: %s\n", buffer);  
   
 /\* 文字づつのアクセス \*/  
 strcpy(name, "Alice");  
 for (i = 0; name[i] != '\0'; i++) {  
 printf("name[%d] = '%c'\n", i, name[i]);  
 }  
   
   
 return 0;

### 文字列配列の扱い

複数の文字列を扱う方法は、C言語学習者がよく迷うポイントです。ここでは2つの方法を詳しく説明します。

#### 方法1: 2次元char配列

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 /\* 各文字列の最大長を20文字とする5つの文字列 \*/  
 char names[5][20]; /\* 19文字+null終端文字 \*/  
 int i;  
   
 /\* 文字列配列への代入 \*/  
 strcpy(names[0], "Alice");  
 strcpy(names[1], "Bob");  
 strcpy(names[2], "Charlie");  
 strcpy(names[3], "David");  
 strcpy(names[4], "Eve");  
   
 /\* 文字列配列の表示 \*/  
 for (i = 0; i < 5; i++) {  
 printf("names[%d] = %s\n", i, names[i]);  
 }  
   
   
 return 0;

#### 方法2: char\*配列（文字列ポインタ配列）

#include <stdio.h>  
  
int main(void)  
{  
  
 /\* 文字列ポインタの配列 \*/  
 char \*fruits[] = {  
 "Apple",  
 "Banana",   
 "Cherry",  
 "Date"  
 };  
 int i;  
   
 /\* 文字列ポインタ配列の表示 \*/  
 for (i = 0; i < 4; i++) {  
 printf("fruits[%d] = %s\n", i, fruits[i]);  
 }  
   
   
 return 0;

#### 文字列配列の初期化パターン

/\* パターン1: 2次元char配列の初期化 \*/  
char cities[3][10] = {  
 "Tokyo",  
 "Osaka",   
 "Kyoto"  
};  
  
/\* パターン2: char\*配列の初期化 \*/  
char \*colors[] = {  
 "Red",  
 "Green",  
 "Blue",  
 "Yellow"  
};  
  
/\* パターン3: 混合初期化 \*/  
char languages[4][15] = {  
 "C", /\* 文字列リテラル \*/  
 "Python", /\* 文字列リテラル \*/  
 "Java", /\* 文字列リテラル \*/  
 "JavaScript" /\* 文字列リテラル \*/  
};

### char配列 vs char\*配列の違い

これは文字列配列を扱う上で重要な概念です。

#### メモリ配置の違い

#include <stdio.h>  
  
int main(void)  
{  
  
 /\* 2次元char配列 \*/  
 char matrix[3][4] = {"ABC", "DEF", "GHI"};  
   
 /\* char\*配列 \*/  
 char \*pointers[] = {"ABC", "DEF", "GHI"};  
   
 printf("=== 2次元char配列 ===\n");  
 printf("全体サイズ: %lu バイト\n", sizeof(matrix)); /\* 12バイト \*/  
 printf("1行のサイズ: %lu バイト\n", sizeof(matrix[0])); /\* 4バイト \*/  
   
 printf("\n=== char\*配列 ===\n");  
 printf("全体サイズ: %lu バイト\n", sizeof(pointers)); /\* 24バイト（8バイトx3） \*/  
 printf("要素のサイズ: %lu バイト\n", sizeof(pointers[0])); /\* 8バイト（ポインタサイズ） \*/  
   
 return 0;

#### 変更可能性の違い

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
  
 /\* 2次元char配列（変更可能） \*/  
 char mutable\_array[3][10] = {"Hello", "World", "Test"};  
   
 /\* char\*配列（文字列リテラルは変更不可） \*/  
 char \*immutable\_array[] = {"Hello", "World", "Test"};  
   
 /\* 配列の内容変更 \*/  
 strcpy(mutable\_array[0], "Hi"); /\* OK: 内容変更可能 \*/  
 /\* strcpy(immutable\_array[0], "Hi"); <- NG: 文字列リテラルは変更不可 \*/  
   
 /\* ポインタの変更 \*/  
 /\* mutable\_array[0] = "New"; <- NG: 配列名は変更不可 \*/  
 immutable\_array[0] = "New"; /\* OK: ポインタ変更可能 \*/  
   
 printf("mutable\_array[0]: %s\n", mutable\_array[0]); /\* "Hi" \*/  
 printf("immutable\_array[0]: %s\n", immutable\_array[0]); /\* "New" \*/  
   
 return 0;

#### 用途に応じた使い分け

/\* 固定文字列の配列（読み取り専用） - char\*配列が適している \*/  
char \*error\_messages[] = {  
 "Success",  
 "File not found",  
 "Permission denied",  
 "Out of memory"  
};  
  
/\* 動的に変更する文字列配列 - 2次元char配列が適している \*/  
char user\_inputs[10][100]; /\* ユーザー入力を格納 \*/  
  
/\* 設定項目など（変更の可能性あり） - char\*配列が適している \*/  
char \*config\_items[] = {  
 "debug=on",  
 "log\_level=info",   
 "max\_connections=100"  
};

### 多次元配列

#### 次元配列の基本

#include <stdio.h>  
  
int main(void)  
{  
  
 int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 };  
 int i, j;  
   
 /\* 2次元配列の表示 \*/  
 for (i = 0; i < 3; i++) {  
 for (j = 0; j < 4; j++) {  
 printf("%d ", matrix[i][j]);  
 }  
 printf("\n");  
 }  
   
 return 0;

#### 次元配列の例

#include <stdio.h>  
  
int main(void)  
{  
  
 /\* 3次元配列: [2][3][4] \*/  
 int cube[2][3][4] = {  
 {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 },  
 {  
 {13, 14, 15, 16},  
 {17, 18, 19, 20},  
 {21, 22, 23, 24}  
 }  
 };  
 int i, j, k;  
   
 for (i = 0; i < 2; i++) {  
 printf("=== Layer %d ===\n", i);  
 for (j = 0; j < 3; j++) {  
 for (k = 0; k < 4; k++) {  
 printf("%d ", cube[i][j][k]);  
 }  
 printf("\n");  
 }  
 printf("\n");  
 }  
   
 return 0;

### 配列の実践的応用

#### 成績管理システム

#include <stdio.h>  
  
#define MAX\_STUDENTS 5  
#define MAX\_SUBJECTS 3  
  
int main(void)  
{  
  
 char students[MAX\_STUDENTS][20] = {  
 "Alice", "Bob", "Charlie", "David", "Eve"  
 };  
 char subjects[MAX\_SUBJECTS][10] = {  
 "Math", "English", "Science"  
 };  
 int scores[MAX\_STUDENTS][MAX\_SUBJECTS] = {  
 {85, 92, 78}, /\* Alice \*/  
 {90, 87, 85}, /\* Bob \*/  
 {95, 91, 89}, /\* Charlie \*/  
 {82, 79, 88}, /\* David \*/  
 {91, 88, 93} /\* Eve \*/  
 };  
 int i, j;  
 int total;  
   
 /\* 成績表の表示 \*/  
 printf("Student ");  
 for (j = 0; j < MAX\_SUBJECTS; j++) {  
 printf("%-10s", subjects[j]);  
 }  
 printf("Average\n");  
 printf("----------------------------------------\n");  
   
 for (i = 0; i < MAX\_STUDENTS; i++) {  
 printf("%-10s ", students[i]);  
 total = 0;  
 for (j = 0; j < MAX\_SUBJECTS; j++) {  
 printf("%-10d", scores[i][j]);  
 total += scores[i][j];  
 }  
 printf("%.1f\n", (double)total / MAX\_SUBJECTS);  
 }  
   
 return 0;

#### 文字列配列のソート

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_NAMES 5  
#define MAX\_LENGTH 20  
  
int main(void)  
{  
  
 char names[MAX\_NAMES][MAX\_LENGTH] = {  
 "Charlie", "Alice", "Eve", "Bob", "David"  
 };  
 char temp[MAX\_LENGTH];  
 int i, j;  
   
 printf("ソート前:\n");  
 for (i = 0; i < MAX\_NAMES; i++) {  
 printf("%s ", names[i]);  
 }  
 printf("\n");  
   
 /\* バブルソート \*/  
 for (i = 0; i < MAX\_NAMES - 1; i++) {  
 for (j = 0; j < MAX\_NAMES - 1 - i; j++) {  
 if (strcmp(names[j], names[j + 1]) > 0) {  
 strcpy(temp, names[j]);  
 strcpy(names[j], names[j + 1]);  
 strcpy(names[j + 1], temp);  
 }  
 }  
 }  
   
 printf("ソート後:\n");  
 for (i = 0; i < MAX\_NAMES; i++) {  
 printf("%s ", names[i]);  
 }  
 printf("\n");  
   
 return 0;

### 配列とポインタの関係

配列名はその先頭要素へのポインタとして動作します。

#include <stdio.h>  
  
int main(void)  
{  
  
 int arr[5] = {10, 20, 30, 40, 50};  
 int \*ptr;  
 int i;  
   
 ptr = arr; /\* arr は &arr[0] と同じ \*/  
   
 printf("=== 配列表記 ===\n");  
 for (i = 0; i < 5; i++) {  
 printf("arr[%d] = %d\n", i, arr[i]);  
 }  
   
 printf("\n=== ポインタ表記 ===\n");  
 for (i = 0; i < 5; i++) {  
 printf("\*(ptr + %d) = %d\n", i, \*(ptr + i));  
 }  
   
 /\* 配列名とポインタの関係 \*/  
 printf("\n=== アドレス比較 ===\n");  
 printf("arr = %p\n", (void\*)arr);  
 printf("&arr[0] = %p\n", (void\*)&arr[0]);  
 printf("ptr = %p\n", (void\*)ptr);  
   
 return 0;

## 実践的なプログラム例

### 基本的な配列操作（C90準拠）

#### 1次元配列の基本

#include <stdio.h>  
  
int main(void)  
{  
 /\* C90では先頭で全ての変数を宣言 \*/  
 int numbers[5] = {10, 20, 30, 40, 50};  
 int total = 0;  
 int i;  
   
 /\* 配列の全要素を表示 \*/  
 printf("=== 配列の内容 ===\n");  
 for (i = 0; i < 5; i++) {  
 printf("numbers[%d] = %d\n", i, numbers[i]);  
 total += numbers[i];  
 }  
   
 printf("合計: %d\n", total);  
 printf("平均: %.1f\n", (double)total / 5);  
   
 return 0;  
}

📁 **ファイル**: <examples/array_basics.c>  
📁 **C99版**: <examples/array_basics_c99.c>

### 文字列配列の実例（C90準拠）

#### char配列 vs char\*配列の違い

#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
 /\* C90では先頭で全ての変数を宣言 \*/  
 char cities[3][20] = {"Tokyo", "Osaka", "Kyoto"}; /\* 2次元char配列 \*/  
 char \*fruits[] = {"Apple", "Banana", "Cherry"}; /\* char\*配列 \*/  
 int i;  
   
 printf("=== 2次元char配列 ===\n");  
 for (i = 0; i < 3; i++) {  
 printf("cities[%d] = %s\n", i, cities[i]);  
 }  
   
 printf("\n=== char\*配列 ===\n");  
 for (i = 0; i < 3; i++) {  
 printf("fruits[%d] = %s\n", i, fruits[i]);  
 }  
   
 /\* メモリサイズの違い \*/  
 printf("\n=== メモリサイズ比較 ===\n");  
 printf("cities配列のサイズ: %lu バイト\n", (unsigned long)sizeof(cities));  
 printf("fruits配列のサイズ: %lu バイト\n", (unsigned long)sizeof(fruits));  
   
 return 0;  
}

📁 **ファイル**: <examples/string_arrays.c>  
📁 **C99版**: <examples/string_arrays_c99.c>

### 多次元配列の活用（C90準拠）

#### 3x4行列の操作

#include <stdio.h>  
  
int main(void)  
{  
 /\* C90では先頭で全ての変数を宣言 \*/  
 int matrix[3][4] = {  
 {1, 2, 3, 4},  
 {5, 6, 7, 8},  
 {9, 10, 11, 12}  
 };  
 int i, j;  
 int row\_sum;  
   
 printf("=== 行列の表示 ===\n");  
 for (i = 0; i < 3; i++) {  
 row\_sum = 0;  
 for (j = 0; j < 4; j++) {  
 printf("%3d ", matrix[i][j]);  
 row\_sum += matrix[i][j];  
 }  
 printf("| 行の合計: %d\n", row\_sum);  
 }  
   
 return 0;  
}

📁 **ファイル**: <examples/multidimensional_arrays.c>  
📁 **C99版**: <examples/multidimensional_arrays_c99.c>

### コンパイル方法

#### 基本的なコンパイル（C90準拠）

gcc -std=c90 -Wall -Wextra -pedantic array\_basics.c -o array\_basics

#### Makefileを使用した場合

# 全てのプログラムをコンパイル  
make all  
  
# 特定のプログラムをコンパイル   
make array\_basics  
  
# C99版をコンパイル  
make array\_basics\_c99  
  
# プログラムを実行  
make run-all  
  
# クリーンアップ  
make clean

## 演習課題と解答例

### 利用可能な演習問題

[演習問題詳細](exercises/README.md)では、配列と文字列配列の理解を深める課題が提供されています。

#### 演習6-1: 配列の基本操作（C90準拠）

#include <stdio.h>  
  
int main(void)  
{  
 /\* C90では先頭で全ての変数を宣言 \*/  
 int numbers[10] = {15, 23, 8, 42, 7, 31, 19, 5, 28, 36};  
 int i;  
 int max, min, sum;  
 double average;  
   
 /\* 最大値、最小値、合計の初期化 \*/  
 max = min = sum = numbers[0];  
   
 /\* 配列の全要素を処理 \*/  
 for (i = 1; i < 10; i++) {  
 if (numbers[i] > max) max = numbers[i];  
 if (numbers[i] < min) min = numbers[i];  
 sum += numbers[i];  
 }  
   
 average = (double)sum / 10;  
   
 printf("=== 配列の統計情報 ===\n");  
 printf("最大値: %d\n", max);  
 printf("最小値: %d\n", min);  
 printf("合計: %d\n", sum);  
 printf("平均: %.1f\n", average);  
   
 return 0;  
}

📁 **解答例**: <solutions/ex6_1.c>  
📁 **C99版**: <solutions/ex6_1_c99.c>

#### 演習6-2: 成績管理システム（C90準拠）

#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"  
};  
  
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");  
 }  
   
 /\* 成績表の表示処理などが続く... \*/  
   
 return 0;  
}

📁 **解答例**: <solutions/ex6_2.c>  
📁 **C99版**: <solutions/ex6_2_c99.c>

#### 演習6-3: 文字列配列の操作（C90準拠）

#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"  
};  
  
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];  
 int i;  
   
 printf("=== 都道府県管理システム ===\n\n");  
   
 /\* 作業用配列にコピー \*/  
 for (i = 0; i < NUM\_PREFECTURES; i++) {  
 strcpy(working\_array[i], prefectures[i]);  
 }  
   
 /\* ソート、検索処理などが続く... \*/  
   
 return 0;  
}

📁 **解答例**: <solutions/ex6_3.c>  
📁 **C99版**: <solutions/ex6_3_c99.c>

### 演習問題一覧

1. **演習6-1**: 配列の基本操作（最大値・最小値・平均値）
2. **演習6-2**: 成績管理システム（5人×4科目）
3. **演習6-3**: 文字列配列の操作（都道府県管理）
4. **演習6-4**: 行列演算（加算・減算・乗算・転置）
5. **演習6-5**: 簡易辞書プログラム（英和・和英検索）

### チャレンジ問題

* **チャレンジ1**: ソートアルゴリズムの実装と比較
* **チャレンジ2**: ライフゲーム（Conway’s Game of Life）

すべての演習には**C90版**と**C99版**の両方の解答例が提供されています。

## 学習フローとコンパイル方法

### 推奨学習順序

1. **理論学習**: このREADME.mdで配列の概念を理解
2. **サンプルコード**: examples/の基本例を確認
3. **演習課題**: exercises/README.mdで課題を確認
4. **実装練習**: solutions/の解答例を参考に自分で実装

### Makefileを使用したコンパイル

# 全ての例題をコンパイル  
make all  
  
# 特定のプログラムをコンパイル  
make array\_basics # C90版  
make array\_basics\_c99 # C99版  
  
# 全てのプログラムを実行  
make run-all  
  
# C90準拠でのテスト  
make test  
  
# 複数のC標準でのテスト  
make test-standards  
  
# クリーンアップ  
make clean  
  
# ヘルプ表示  
make help

### 手動コンパイル

# C90準拠でコンパイル（推奨）  
gcc -std=c90 -Wall -Wextra -pedantic examples/array\_basics.c -o array\_basics  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic examples/array\_basics\_c99.c -o array\_basics\_c99  
  
# 実行  
./array\_basics

## C90とC99の違い

### C90の特徴（このチュートリアルの基準）

int main(void)  
{  
 /\* すべての変数を関数の先頭で宣言 \*/  
 int numbers[5] = {1, 2, 3, 4, 5};  
 int i;  
 int sum = 0;  
   
 /\* 配列サイズは定数で指定 \*/  
 #define ARRAY\_SIZE 10  
 int arr[ARRAY\_SIZE];  
   
 /\* forループは従来のスタイル \*/  
 for (i = 0; i < 5; i++) {  
 sum += numbers[i];  
 }  
   
 return 0;  
}

### C99の拡張機能

int main(void)  
{  
 int numbers[5] = {1, 2, 3, 4, 5};  
   
 /\* forループ内での変数宣言（C99） \*/  
 for (int i = 0; i < 5; i++) {  
 printf("%d ", numbers[i]);  
 }  
   
 /\* 必要な時点での変数宣言（C99） \*/  
 int sum = 0;  
 for (int i = 0; i < 5; i++) {  
 sum += numbers[i];  
 }  
   
 /\* 可変長配列（VLA）も可能（C99） \*/  
 int size = 10;  
 int vla[size]; /\* C90では不可 \*/  
   
 return 0;  
}

### このチュートリアルでの方針

* **C90版**: 基準となる実装、互換性重視
* **C99版**: 現代的な書き方、読みやすさ重視
* **学習推奨**: まずC90で基礎を固めてからC99を学習

## よくある間違いとデバッグ方法

### 1. 配列の境界外アクセス

#### 間違った例

/\* NG: 配列の範囲を超える \*/  
int arr[5] = {1, 2, 3, 4, 5};  
printf("%d\n", arr[5]); /\* 範囲外アクセス - 未定義動作 \*/

#### 正しい例

/\* OK: 正しい範囲でアクセス \*/  
#include <stdio.h>  
  
int main(void)  
{  
 int arr[5] = {1, 2, 3, 4, 5};  
 int i;  
   
 /\* 安全な配列アクセス \*/  
 for (i = 0; i < 5; i++) { /\* i < 5 であることを確認 \*/  
 printf("arr[%d] = %d\n", i, arr[i]);  
 }  
   
 return 0;  
}

### 2. 文字列配列の初期化忘れ

#### 間違った例

/\* NG: 初期化されていない文字配列 \*/  
char str[20];  
printf("%s\n", str); /\* 未定義動作 \*/

#### 正しい例

/\* OK: 適切な初期化 \*/  
#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
 char str[20] = ""; /\* 空文字列で初期化 \*/  
 strcpy(str, "Hello"); /\* 後から代入 \*/  
 printf("%s\n", str);  
   
 return 0;  
}

### 3. 配列代入の間違い

#### 間違った例

/\* NG: 配列は直接代入できない \*/  
char str1[20], str2[20] = "Hello";  
str1 = str2; /\* コンパイルエラー \*/

#### 正しい例

/\* OK: strcpyを使用 \*/  
#include <string.h>  
  
char str1[20], str2[20] = "Hello";  
strcpy(str1, str2); /\* 正しい文字列コピー \*/

### 4. C90とC99の混在エラー

#### C90でコンパイル時のエラー例

/\* C90ではエラーになる書き方 \*/  
int main(void)  
{  
 int arr[5] = {1, 2, 3, 4, 5};  
 for (int i = 0; i < 5; i++) { /\* C90ではNG \*/  
 printf("%d ", arr[i]);  
 }  
 return 0;  
}

#### 正しいC90の書き方

/\* C90準拠の書き方 \*/  
int main(void)  
{  
 int arr[5] = {1, 2, 3, 4, 5};  
 int i; /\* 先頭で宣言 \*/  
   
 for (i = 0; i < 5; i++) {  
 printf("%d ", arr[i]);  
 }  
 return 0;  
}

### デバッグのコツ

* **境界チェック**: 配列アクセス時は必ずインデックスが範囲内かチェック
* **初期化の確認**: 配列や文字列は使用前に必ず初期化
* **コンパイル標準の確認**: -std=c90 または -std=c99 の指定を確認
* **解答例との比較**: solutions/の正解コードと比較して違いを確認

## 次の章へ

配列と文字列配列を理解したら、[文字列処理](../strings/README.md) に進んでください。

## 参考資料

* [C言語配列リファレンス](https://ja.cppreference.com/w/c/language/array)
* [文字列操作関数](https://ja.cppreference.com/w/c/string/byte)
* [多次元配列の詳細](https://ja.cppreference.com/w/c/language/array)

# サンプルコード

## array\_basics.c

/\*  
 \* 配列の基本的な使い方（C90準拠版）  
 \*   
 \* このプログラムは、1次元配列の宣言、初期化、  
 \* 操作の基本的な方法を示します。  
 \*/  
#include <stdio.h>  
  
#define MAX\_SIZE 10  
  
int main(void)  
{  
 /\* C90では先頭で全ての変数を宣言 \*/  
 int numbers[5];  
 int i;  
 int scores[5] = {85, 92, 78, 96, 88};  
 int partial[10] = {1, 2, 3};  
 int auto\_size[] = {5, 10, 15, 20, 25, 30};  
 int size;  
 int test\_scores[] = {75, 88, 92, 67, 85, 91, 78, 84, 90, 86};  
 int n;  
 int sum = 0;  
 int max;  
 int min;  
 int search\_array[] = {15, 23, 8, 42, 16, 4, 11, 29, 36, 19};  
 int search\_value = 42;  
 int found = 0;  
 int position = -1;  
 int original[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 int source[5] = {11, 22, 33, 44, 55};  
 int destination[5];  
 int arr[MAX\_SIZE];  
   
 printf("=== 配列の基本的な使用 ===\n");  
   
 /\* 配列に値を代入 \*/  
 for (i = 0; i < 5; i++) {  
 numbers[i] = (i + 1) \* 10;  
 }  
   
 /\* 配列の内容を表示 \*/  
 printf("配列の内容: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n\n");  
   
 /\* 2. 配列の初期化 \*/  
 printf("=== 配列の初期化 ===\n");  
   
 /\* 完全な初期化 \*/  
 printf("scores配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", scores[i]);  
 }  
 printf("\n");  
   
 /\* 部分的な初期化 \*/  
 printf("partial配列: ");  
 for (i = 0; i < 10; i++) {  
 printf("%d ", partial[i]);  
 }  
 printf("\n");  
   
 /\* サイズを省略した初期化 \*/  
 size = sizeof(auto\_size) / sizeof(auto\_size[0]);  
 printf("auto\_size配列（要素数%d）: ", size);  
 for (i = 0; i < size; i++) {  
 printf("%d ", auto\_size[i]);  
 }  
 printf("\n\n");  
   
 /\* 3. 配列の統計計算 \*/  
 printf("=== 配列の統計計算 ===\n");  
   
 n = sizeof(test\_scores) / sizeof(test\_scores[0]);  
 sum = 0;  
 max = test\_scores[0];  
 min = test\_scores[0];  
   
 /\* 合計、最大値、最小値を計算 \*/  
 for (i = 0; i < n; i++) {  
 sum += test\_scores[i];  
 if (test\_scores[i] > max) {  
 max = test\_scores[i];  
 }  
 if (test\_scores[i] < min) {  
 min = test\_scores[i];  
 }  
 }  
   
 printf("テストの点数: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", test\_scores[i]);  
 }  
 printf("\n");  
 printf("要素数: %d\n", n);  
 printf("合計: %d\n", sum);  
 printf("平均: %.2f\n", (double)sum / n);  
 printf("最高点: %d\n", max);  
 printf("最低点: %d\n\n", min);  
   
 /\* 4. 配列の要素検索 \*/  
 printf("=== 配列の検索 ===\n");  
   
 /\* 線形探索 \*/  
 for (i = 0; i < 10; i++) {  
 if (search\_array[i] == search\_value) {  
 found = 1;  
 position = i;  
 break;  
 }  
 }  
   
 if (found) {  
 printf("値 %d は位置 %d にあります\n", search\_value, position);  
 } else {  
 printf("値 %d は見つかりませんでした\n", search\_value);  
 }  
 printf("\n");  
   
 /\* 5. 配列の逆順表示 \*/  
 printf("=== 配列の逆順表示 ===\n");  
   
 printf("元の配列: ");  
 for (i = 0; i < 10; i++) {  
 printf("%d ", original[i]);  
 }  
 printf("\n");  
   
 printf("逆順: ");  
 for (i = 9; i >= 0; i--) {  
 printf("%d ", original[i]);  
 }  
 printf("\n\n");  
   
 /\* 6. 配列のコピー \*/  
 printf("=== 配列のコピー ===\n");  
   
 /\* 要素を1つずつコピー \*/  
 for (i = 0; i < 5; i++) {  
 destination[i] = source[i];  
 }  
   
 printf("source配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", source[i]);  
 }  
 printf("\n");  
   
 printf("destination配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", destination[i]);  
 }  
 printf("\n\n");  
   
 /\* 7. 配列サイズの注意点 \*/  
 printf("=== 配列サイズの情報 ===\n");  
   
 printf("配列のサイズ: %lu バイト\n", sizeof(arr));  
 printf("要素1つのサイズ: %lu バイト\n", sizeof(arr[0]));  
 printf("要素数: %lu\n", sizeof(arr) / sizeof(arr[0]));  
   
 return 0;  
}

## multidimensional\_arrays.c

```c / 多次元配列の使い方（C90準拠版）  このプログラムは、2次元配列と3次元配列の \* 宣言、初期化、操作方法を示します。 \*/ #include <stdio.h>

int main(void) { /\* C90では先頭で全ての変数を宣言 */ int i, j; int matrix[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}; int grades[5][3] = { {85, 92, 78}, /* 学生1 */ {90, 88, 85}, /* 学生2 */ {78, 85, 92}, /* 学生3 */ {92, 76, 88}, /* 学生4 */ {88, 90, 82} /* 学生5 */ }; char* subjects[] = {“国語”, “数学”, “英語”}; int partial[4][5] = { {1, 2}, /\* 残りの要素は0 */ {3, 4, 5}, /* 残りの要素は0 */ {6} /* 残りの要素は0 */ /* 4行目は全て0 */ }; int identity[5][5]; int image[2][3][3] = { { {255, 0, 0}, /* 赤 */ {0, 255, 0}, /* 緑 */ {0, 0, 255} /* 青 */ }, { {255, 255, 0}, /* 黄 */ {255, 0, 255}, /* マゼンタ */ {0, 255, 255} /* シアン \*/ }}; int original\_matrix[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}; int transposed[4][3]; int sum;

/\* 1. 2次元配列の基本 \*/  
printf("=== 2次元配列の基本 ===\n");  
  
/\* 3x4の2次元配列 \*/  
  
/\* 2次元配列の表示 \*/  
printf("3x4の行列:\n");  
for (i = 0; i < 3; i++)  
{  
 for (j = 0; j < 4; j++)  
 {  
 printf("%3d ", matrix[i][j]);  
 }  
 printf("\n");  
}  
printf("\n");  
  
/\* 2. 成績表の管理 \*/  
printf("=== 成績表（学生x科目）===\n");  
  
/\* 5人の学生、3科目（国語、数学、英語）の成績 \*/  
  
/\* 成績表の表示 \*/  
printf(" ");  
for (j = 0; j < 3; j++)  
{  
 printf("%-6s ", subjects[j]);  
}  
printf("平均\n");  
printf("----------------------------------------\n");  
  
for (i = 0; i < 5; i++)  
{  
 printf("学生%d: ", i + 1);  
 sum = 0;  
 for (j = 0; j < 3; j++)  
 {  
 printf("%-6d ", grades[i][j]);  
 sum += grades[i][j];  
 }  
 printf("%.1f\n", sum / 3.0);  
}  
  
/\* 科目別平均 \*/  
printf("----------------------------------------\n");  
printf("平均: ");  
for (j = 0; j < 3; j++)  
{  
 sum = 0;  
 for (i = 0; i < 5; i++)  
 {  
 sum += grades[i][j];  
 }  
 printf("%-6.1f ", sum / 5.0);  
}  
printf("\n\n");  
  
/\* 3. 部分的な初期化 \*/  
printf("=== 2次元配列の部分初期化 ===\n");  
  
/\* 部分的に初期化（残りは0） \*/  
  
printf("部分的に初期化された配列:\n");  
for (i = 0; i < 4; i++)  
{  
 for (j = 0; j < 5; j++)  
 {  
 printf("%2d ", partial[i][j]);  
 }  
 printf("\n");  
}  
printf("\n");  
  
/\* 4. 単位行列の作成 \*/  
printf("=== 単位行列の作成 ===\n");  
  
/\* 単位行列を作成 \*/  
for (i = 0; i < 5; i++)  
{  
 for (j = 0; j < 5; j++)  
 {  
 if (i == j)  
 {  
 identity[i][j] = 1;  
 }  
 else  
 {  
 identity[i][j] = 0;  
 }  
 }  
}  
  
/\* 単位行列を表示 \*/  
printf("5x5の単位行列:\n");  
for (i = 0; i < 5; i++)  
{  
 for (j = 0; j < 5; j++)  
 {  
 printf("%d ", identity[i][j]);  
 }  
 printf("\n");  
}  
printf("\n");  
  
/\* 5. 3次元配列の例 \*/  
printf("=== 3次元配列（RGB画像データの例）===\n");  
  
/\* 2x3ピクセル、各ピクセルはRGB（3値） \*/  
  
printf("画像データ（2x3ピクセル）:\n");  
for (i = 0; i < 2; i++)  
{  
 printf("行 %d:\n", i);  
 for (j = 0; j < 3; j++)  
 {  
 printf(" ピクセル[%d][%d]: R=%3d, G=%3d, B=%3d\n",  
 i, j, image[i][j][0], image[i][j][1], image[i][j][2]);  
 }  
}  
printf("\n");  
  
/\* 6. 転置行列 \*/  
printf("=== 行列の転置 ===\n");  
  
/\* 転置を計算 \*/  
for (i = 0; i < 3; i++)  
{  
 for (j = 0; j < 4; j++)  
 {  
 transposed[j][i] = original\_matrix[i][j];  
 }  
}  
  
printf("元の行列（3x4）:\n");  
for (i = 0; i < 3; i++)  
{  
 for (j = 0; j < 4; j++)  
 {  
 printf("%3d ", original\_matrix[i][j]);  
 }  
 printf("\n");  
}  
  
printf("\n転置行列（4x3）:\n");  
for (i = 0; i < 4; i++)  
{  
 for (j = 0; j < 3; j++)  
 {  
 printf("%3d ", transposed[i][j]);  
 }  
 printf("\n");  
}  
  
return 0;

}```

## string\_arrays.c

```c / 文字列配列の使い方（C90準拠版） \* \* このプログラムは、文字列配列の宣言、初期化、 \* 操作方法、およびchar配列とchar*配列の違いを示します。* / #include <stdio.h> #include <string.h>

int main(void) { /\* C90では先頭で全ての変数を宣言 */ int i; char str1[20]; char str2[] = “World”; char fruits[5][20] = { “Apple”, “Banana”, “Orange”, “Grape”, “Strawberry” }; char* months[] = { “January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December” }; char cities1[3][20] = {“Tokyo”, “New York”, “London”}; char *cities2[] = {“Tokyo”, “New York”, “London”}; char* programming\_languages[] = { “C”, “C++”, “Java”, “Python”, “JavaScript”, “Ruby”, “Go”, “Rust”, “Swift”, “Kotlin” }; int n\_languages; char search\_lang[20]; int found = 0; char *words[] = {“cat”, “elephant”, “dog”, “hippopotamus”, “ant”, “butterfly”}; int n\_words; int len; char* menu\_items[] = { “1. ファイルを開く”, “2. ファイルを保存”, “3. 設定”, “4. ヘルプ”, “5. 終了” }; int n\_items;

/\* 1. char配列（文字配列）の基本 \*/  
printf("=== char配列（文字配列）===\n");  
  
strcpy(str1, "Hello"); /\* C90では初期化後に代入 \*/  
  
printf("str1: %s（サイズ: %lu）\n", str1, sizeof(str1));  
printf("str2: %s（サイズ: %lu）\n", str2, sizeof(str2));  
  
/\* 文字配列の変更 \*/  
str1[0] = 'h'; /\* 最初の文字を小文字に \*/  
printf("変更後のstr1: %s\n\n", str1);  
  
/\* 2. 文字列配列（2次元char配列）\*/  
printf("=== 2次元char配列による文字列配列 ===\n");  
  
printf("果物リスト:\n");  
for (i = 0; i < 5; i++) {  
 printf("%d: %s\n", i + 1, fruits[i]);  
}  
  
/\* 文字列の変更 \*/  
strcpy(fruits[2], "Mango"); /\* OrangeをMangoに変更 \*/  
printf("\n変更後（3番目をMangoに）:\n");  
for (i = 0; i < 5; i++) {  
 printf("%d: %s\n", i + 1, fruits[i]);  
}  
printf("\n");  
  
/\* 3. char\*配列（ポインタ配列）\*/  
printf("=== char\*配列による文字列配列 ===\n");  
  
printf("月の名前:\n");  
for (i = 0; i < 12; i++) {  
 printf("%2d月: %s\n", i + 1, months[i]);  
}  
printf("\n");  
  
/\* 4. char配列とchar\*配列の違い \*/  
printf("=== char配列とchar\*配列の違い ===\n");  
  
printf("char配列のサイズ: %lu バイト\n", sizeof(cities1));  
printf("char\*配列のサイズ: %lu バイト\n", sizeof(cities2));  
printf("（ポインタのサイズx要素数）\n\n");  
  
/\* 5. 文字列の検索と操作 \*/  
printf("=== 文字列配列の検索 ===\n");  
  
n\_languages = sizeof(programming\_languages) / sizeof(programming\_languages[0]);  
  
printf("検索する言語名を入力: ");  
scanf("%s", search\_lang);  
  
found = 0;  
for (i = 0; i < n\_languages; i++) {  
 if (strcmp(programming\_languages[i], search\_lang) == 0) {  
 printf("%s は %d 番目にあります\n", search\_lang, i + 1);  
 found = 1;  
 break;  
 }  
}  
  
if (!found) {  
 printf("%s は見つかりませんでした\n", search\_lang);  
}  
printf("\n");  
  
/\* 6. 文字列の長さ順にソート（簡易版）\*/  
printf("=== 文字列の長さ順表示 ===\n");  
  
n\_words = sizeof(words) / sizeof(words[0]);  
  
printf("元の順序:\n");  
for (i = 0; i < n\_words; i++) {  
 printf("%s ", words[i]);  
}  
printf("\n");  
  
/\* 長さ順に表示（実際の配列は変更しない） \*/  
printf("\n長さ順:\n");  
for (len = 1; len <= 20; len++) {  
 for (i = 0; i < n\_words; i++) {  
 if ((int)strlen(words[i]) == len) { /\* 型キャストで警告を回避 \*/  
 printf("%s（%d文字） ", words[i], len);  
 }  
 }  
}  
printf("\n\n");  
  
/\* 7. 動的な文字列配列の例 \*/  
printf("=== コマンドメニューシステム ===\n");  
  
n\_items = sizeof(menu\_items) / sizeof(menu\_items[0]);  
  
printf("メニュー:\n");  
for (i = 0; i < n\_items; i++) {  
 printf("%s\n", menu\_items[i]);  
}  
  
return 0;

}```