C言語プログラミング教材

演習問題集

C言語学習教材プロジェクト

2025年07月01日

Table of Contents

# 目次

* 第1章 導入・環境構築
* 第2章 基本文法・Hello World
* 第3章 データ型と変数
* 第4章 演算子
* 第5章 制御構造（条件分岐）
* 第6章 制御構造（ループ）
* 第7章 配列
* 第8章 文字列処理
* 第9章 関数
* 第10章 ポインタ基礎
* 第11章 構造体とポインタ
* 第12章 関数ポインタ
* 第13章 複数ファイル・発展技術
* 第14章 C23の新機能（オプション）

# 第1章 導入・環境構築

## 演習問題

## 演習の目的

この演習を通して、C言語の開発環境が正しく構築されているかを確認し、基本的なコンパイル手順を習得します。

## 演習課題

### 基礎問題

#### 問題1: 環境構築確認

サンプルプログラム examples/environment\_check.c を使用して、以下の作業を実行してください。

1. **C90準拠でコンパイル・実行**

* make c90

1. **C99準拠でコンパイル・実行**

* make c99

1. **C11準拠でコンパイル・実行**

* make c11

1. **C17準拠でコンパイル・実行**

* make c17

1. **実行結果の確認**
   * 各規格でコンパイルしたプログラムの出力結果を比較してください
   * 規格による違いがあるかを確認してください

#### 問題2: コンパイラ情報の確認

以下のコマンドを実行して、開発環境の情報を調べてください。

1. **コンパイラバージョンの確認**

* gcc --version

1. **対応規格の確認**

* gcc -std=c90 --help  
  gcc -std=c99 --help  
  gcc -std=c11 --help  
  gcc -std=c17 --help

1. **プリプロセッサマクロの確認**

* echo | gcc -dM -E -

#### 問題3: 基本的なプログラム作成

solutions/ex1\_3\_hello\_name.c というファイルを作成し、以下の機能を実装してください。

**要求仕様:** - ユーザーに名前の入力を求める - 入力された名前を使って挨拶メッセージを表示する - C90準拠で記述する

**実行例:**

あなたの名前を入力してください: 田中  
こんにちは、田中さん！

### 応用問題

#### 問題4: 規格比較レポート

C90とC99の主な違いについて調べ、以下の点についてまとめてください。

1. **新しく追加されたデータ型**
2. **新しく追加された機能**
3. **プログラムの書き方で変わった点**
4. **組込み開発でC90が使われ続ける理由**

回答は exercises/standards\_comparison.md に記述してください。

#### 問題5: コンパイル手順の詳細調査

以下のコンパイラオプションを使用して、コンパイル過程を段階的に確認してください。

1. **プリプロセッサ出力の確認**

* gcc -E examples/environment\_check.c > preprocessed.i

1. **アセンブリコード生成**

* gcc -S examples/environment\_check.c

1. **オブジェクトファイル生成**

* gcc -c examples/environment\_check.c

1. **実行ファイル生成**

* gcc environment\_check.o -o environment\_check

各段階で生成されるファイルの内容を確認し、exercises/compilation\_report.md にまとめてください。

#### 問題6: 環境固有情報の調査

以下の情報を調べて、solutions/ex1\_6\_system\_info.c プログラムを作成してください。

**表示する情報:** - 使用しているOS - コンパイラの種類とバージョン - CPUアーキテクチャ（32bit/64bit） - エンディアン（ビッグエンディアン/リトルエンディアン） - 各データ型のサイズ

## 提出形式

各問題の回答は以下のファイルに記述してください：

* **問題3**: solutions/ex1\_3\_hello\_name.c
* **問題4**: exercises/standards\_comparison.md
* **問題5**: exercises/compilation\_report.md
* **問題6**: solutions/ex1\_6\_system\_info.c

## 評価基準

### 基礎問題

* 環境確認プログラムが正常にコンパイル・実行できる
* 各規格でのコンパイルができる
* ex1\_3\_hello\_name.cが要求仕様を満たしている
* C90準拠で記述されている

### 応用問題

* 規格比較が正確に記述されている
* コンパイル手順が理解できている
* 環境固有情報の取得ができている
* レポートが詳細で分かりやすい

## 参考資料

* [GCC公式ドキュメント](https://gcc.gnu.org/documentation.html)
* [C言語規格書](https://www.iso.org/standard/74528.html)
* [GNU Make マニュアル](https://www.gnu.org/software/make/manual/)

## 次の段階

演習課題が完了したら、[基本文法・Hello World](../../basics-syntax/README.md) に進んでください。

## 解答例

この章では、C言語の開発環境構築の確認と基本的なプログラムの作成練習を行いました。

## 解答ファイル一覧

### 問題3: 基本的なプログラム作成

ユーザーに名前の入力を求め、挨拶メッセージを表示するプログラムです。

* <ex1_3_hello_name.c> - C90準拠版
* <ex1_3_hello_name_c99.c> - C99準拠版（//コメント、変数宣言位置の自由化）

**学習ポイント:** - 標準入出力（printf、scanf）の基本的な使い方 - 文字配列（文字列）の扱い方 - 入力エラーの処理 - C90とC99のコーディングスタイルの違い

### 問題6: システム情報調査プログラム

実行環境の詳細情報を調査・表示するプログラムです。

* <ex1_6_system_info.c> - C90準拠版
* <ex1_6_system_info_c99.c> - C99準拠版（bool型、stdint.h、複合リテラル使用）

**学習ポイント:** - プリプロセッサディレクティブによる条件コンパイル - エンディアン判定の実装方法 - データ型のサイズとlimits.hの使用 - C99で追加された機能の確認方法

## コンパイルと実行

各解答例は以下のようにコンパイルできます：

# C90版のコンパイル例  
gcc -Wall -Wextra -pedantic -std=c90 ex1\_3\_hello\_name.c -o ex1\_3\_hello\_name  
  
# C99版のコンパイル例  
gcc -Wall -Wextra -pedantic -std=c99 ex1\_3\_hello\_name\_c99.c -o ex1\_3\_hello\_name\_c99  
  
# 実行  
./ex1\_3\_hello\_name

親ディレクトリのMakefileを使用する場合：

# すべての解答例をコンパイル  
make solutions  
  
# 個別にコンパイル  
make ex1\_3\_hello\_name  
make ex1\_6\_system\_info\_c99  
  
# 実行  
make run-ex1\_3\_hello\_name

## 学習のポイント

### 1. C90とC99の主な違い

各解答例にはC90版とC99版の両方を用意しています。主な違いは：

* **コメント**: C90は/\* \*/のみ、C99は//も使用可能
* **変数宣言**: C90は関数の先頭、C99は使用箇所で宣言可能
* **新しい型**: C99ではbool、long long、stdint.hの固定幅整数型が追加
* **その他**: 可変長配列、複合リテラル、designated initializerなど

### 2. 入力処理の安全性

ex1\_3\_hello\_name.cでは、バッファオーバーフローを防ぐため：

scanf("%99s", name); /\* 最大99文字に制限 \*/

### 3. システム情報の取得方法

ex1\_6\_system\_info.cでは、以下の技術を使用： - **エンディアン判定**: unionを使用したバイト順の確認 - **アーキテクチャ判定**: ポインタサイズによる判定 - **OS判定**: プリプロセッサマクロの利用 - **コンパイラ判定**: コンパイラ固有のマクロ

### 4. エラー処理の重要性

すべてのプログラムで適切なエラー処理を実装： - 入力エラーのチェック - 戻り値による状態の通知 - エラーメッセージの表示

## 発展的な学習

これらの基本的な実装を理解したら、以下に挑戦してみましょう：

1. **入力の拡張**: 複数の単語を含む名前の入力（fgetsの使用）
2. **国際化**: 日本語以外の言語への対応
3. **クロスプラットフォーム**: より多くのOSやコンパイラへの対応
4. **ビルドシステム**: CMakeなどの高度なビルドツールの使用

## 注意事項

* 実装例はあくまで一例です。他の実装方法も検討してみましょう
* 実際のプロジェクトでは、より堅牢なエラー処理が必要です
* システム固有の情報は環境によって異なる結果となります ### ex1\_3\_hello\_name.c

/\*  
 \* ファイル名: ex1\_3\_hello\_name.c  
 \* 演習1-3: 基本的なプログラム作成  
 \* 説明: ユーザーに名前の入力を求め、挨拶メッセージを表示する  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\*  
 \* メイン関数  
 \*/  
int main(void)  
{  
 char name[100]; /\* 名前を格納する配列 \*/  
  
 /\* 名前の入力を求める \*/  
 printf("あなたの名前を入力してください: ");  
  
 /\* 標準入力から名前を読み込み \*/  
 if (scanf("%99s", name) == 1)  
 {  
 /\* 挨拶メッセージを表示 \*/  
 printf("こんにちは、%sさん!\n", name);  
 }  
 else  
 {  
 /\* 入力エラーの場合 \*/  
 printf("入力エラーが発生しました。\n");  
 return 1;  
 }  
  
 return 0;  
}

### ex1\_6\_system\_info.c

/\*  
 \* ファイル名: ex1\_6\_system\_info.c  
 \* 演習1-6: 環境固有情報の調査  
 \* 説明: 環境固有の情報を調査・表示する  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <limits.h>  
  
/\*  
 \* エンディアンを判定する関数  
 \*/  
void check\_endianness(void)  
{  
 union  
 {  
 unsigned int i;  
 unsigned char c[sizeof(unsigned int)];  
 } test;  
  
 test.i = 0x12345678;  
  
 printf("エンディアン判定:\n");  
 if (test.c[0] == 0x78)  
 {  
 printf("- リトルエンディアン\n");  
 }  
 else if (test.c[0] == 0x12)  
 {  
 printf("- ビッグエンディアン\n");  
 }  
 else  
 {  
 printf("- 不明なエンディアン\n");  
 }  
}  
  
/\*  
 \* アーキテクチャ情報を表示する関数  
 \*/  
void show\_architecture\_info(void)  
{  
 printf("アーキテクチャ情報:\n");  
  
 if (sizeof(void \*) == 4)  
 {  
 printf("- 32ビットアーキテクチャ\n");  
 }  
 else if (sizeof(void \*) == 8)  
 {  
 printf("- 64ビットアーキテクチャ\n");  
 }  
 else  
 {  
 printf("- 不明なアーキテクチャ（ポインタサイズ: %lu bytes）\n",  
 (unsigned long)sizeof(void \*));  
 }  
}  
  
/\*  
 \* OS情報を表示する関数  
 \*/  
void show\_os\_info(void)  
{  
 printf("オペレーティングシステム:\n");  
  
#ifdef \_WIN32  
 printf("- Windows\n");  
#elif defined(\_\_APPLE\_\_) && defined(\_\_MACH\_\_)  
 printf("- macOS\n");  
#elif defined(\_\_linux\_\_)  
 printf("- Linux\n");  
#elif defined(\_\_unix\_\_)  
 printf("- Unix系OS\n");  
#else  
 printf("- 不明なOS\n");  
#endif  
}  
  
/\*  
 \* コンパイラ情報を表示する関数  
 \*/  
void show\_compiler\_info(void)  
{  
 printf("コンパイラ情報:\n");  
  
#ifdef \_\_GNUC\_\_  
 printf("- GCC %d.%d.%d\n", \_\_GNUC\_\_, \_\_GNUC\_MINOR\_\_, \_\_GNUC\_PATCHLEVEL\_\_);  
#endif  
  
#ifdef \_MSC\_VER  
 printf("- Microsoft Visual C++ %d\n", \_MSC\_VER);  
#endif  
  
#ifdef \_\_clang\_\_  
 printf("- Clang %d.%d.%d\n", \_\_clang\_major\_\_, \_\_clang\_minor\_\_, \_\_clang\_patchlevel\_\_);  
#endif  
  
#ifdef \_\_STDC\_VERSION\_\_  
 printf("- C規格: ");  
 if (\_\_STDC\_VERSION\_\_ >= 201710L)  
 {  
 printf("C17\n");  
 }  
 else if (\_\_STDC\_VERSION\_\_ >= 201112L)  
 {  
 printf("C11\n");  
 }  
 else if (\_\_STDC\_VERSION\_\_ >= 199901L)  
 {  
 printf("C99\n");  
 }  
 else  
 {  
 printf("C90\n");  
 }  
#else  
 printf("- C規格: C90またはコンパイラ固有\n");  
#endif  
}  
  
/\*  
 \* データ型のサイズを表示する関数  
 \*/  
void show\_type\_sizes(void)  
{  
 printf("データ型サイズ:\n");  
  
 printf("- char: %lu bytes\n", (unsigned long)sizeof(char));  
 printf("- short: %lu bytes\n", (unsigned long)sizeof(short));  
 printf("- int: %lu bytes\n", (unsigned long)sizeof(int));  
 printf("- long: %lu bytes\n", (unsigned long)sizeof(long));  
 printf("- float: %lu bytes\n", (unsigned long)sizeof(float));  
 printf("- double: %lu bytes\n", (unsigned long)sizeof(double));  
 printf("- void\*: %lu bytes\n", (unsigned long)sizeof(void \*));  
  
 printf("\nデータ型の範囲:\n");  
 printf("- char: %d 〜 %d\n", CHAR\_MIN, CHAR\_MAX);  
 printf("- short: %d 〜 %d\n", SHRT\_MIN, SHRT\_MAX);  
 printf("- int: %d 〜 %d\n", INT\_MIN, INT\_MAX);  
 printf("- long: %ld 〜 %ld\n", LONG\_MIN, LONG\_MAX);  
}  
  
/\*  
 \* メイン関数  
 \*/  
int main(void)  
{  
 printf("=== システム情報調査プログラム ===\n\n");  
  
 /\* OS情報の表示 \*/  
 show\_os\_info();  
 printf("\n");  
  
 /\* コンパイラ情報の表示 \*/  
 show\_compiler\_info();  
 printf("\n");  
  
 /\* アーキテクチャ情報の表示 \*/  
 show\_architecture\_info();  
 printf("\n");  
  
 /\* エンディアン判定 \*/  
 check\_endianness();  
 printf("\n");  
  
 /\* データ型サイズの表示 \*/  
 show\_type\_sizes();  
  
 printf("\n=== 調査完了 ===\n");  
  
 return 0;  
}

# 第2章 基本文法・Hello World

## 演習問題

## 基礎課題

### 課題2-1: 基本的な出力

プログラムを作成して以下を出力してください：

こんにちは、C言語の世界へ！  
私の名前は [あなたの名前] です。  
今日からプログラミングを始めます。

**要求事項:** - printf関数を使用すること - 適切なエスケープシーケンスを使用すること - [あなたの名前]の部分は実際の名前に置き換えること

**ファイル名:** ex2\_1\_hello\_intro.c

### 課題2-2: 変数と基本データ型

以下の変数を定義し、値を代入して出力するプログラムを作成してください：

**変数一覧:** - 整数型: 年齢（例：25） - 浮動小数点型: 身長（例：170.5） - 文字型: 血液型（例：‘A’） - 整数型: 好きな数字（例：7）

**出力例:**

=== 自己紹介データ ===  
年齢: 25歳  
身長: 170.5cm  
血液型: A型  
好きな数字: 7  
================

**要求事項:** - 適切なデータ型を選択すること - printfの書式指定子を正しく使用すること - コメントで各変数の説明を記述すること

**ファイル名:** ex2\_2\_personal\_data.c

### 課題2-3: 入力と出力

ユーザーから名前と年齢を入力してもらい、それを使って計算結果を表示するプログラムを作成してください。

**機能:** 1. 名前を入力してもらう 2. 年齢を入力してもらう 3. 10年後の年齢を計算する 4. 結果を表示する

**出力例:**

お名前を入力してください: 田中太郎  
年齢を入力してください: 25  
  
こんにちは、田中太郎さん！  
現在の年齢: 25歳  
10年後の年齢: 35歳  
プログラミングの学習、頑張ってください！

**要求事項:** - scanf関数を使用すること - 適切なバッファサイズを設定すること - 計算結果を変数に保存すること

**ファイル名:** ex2\_3\_age\_calculator.c

## 応用課題

### 課題2-4: 書式指定子の練習

様々な書式指定子を使って、数値を異なる形式で表示するプログラムを作成してください。

**表示する数値:** 123, 3.14159, 255

**出力例:**

=== 書式指定子のデモ ===  
整数 123 の表示:  
 10進数: 123  
 16進数: 7b  
 8進数: 173  
 フィールド幅5: | 123|  
 ゼロ埋め: |00123|  
  
実数 3.14159 の表示:  
 デフォルト: 3.14159  
 小数点以下2桁: 3.14  
 指数表記: 3.14159e+00  
 フィールド幅10.2: | 3.14|  
  
文字コード 255:  
 文字として: ÿ  
 16進数: ff  
 10進数: 255  
====================

**要求事項:** - 各種書式指定子を使用すること（%d, %x, %o, %f, %e, %c など） - フィールド幅とゼロ埋めを実演すること - 適切なコメントを記述すること

**ファイル名:** ex2\_4\_format\_demo.c

### 課題2-5: 簡単な計算機

四則演算を実行する簡単な計算機プログラムを作成してください。

**機能:** 1. 2つの数値を入力してもらう 2. 四則演算（+, -, \*, /）の結果を表示する 3. 割り算では整数除算と実数除算の両方を表示する

**出力例:**

簡単な計算機プログラム  
===================  
第1の数値を入力してください: 7  
第2の数値を入力してください: 2  
  
計算結果:  
 7 + 2 = 9  
 7 - 2 = 5  
 7 \* 2 = 14  
 7 / 2 = 3 (整数除算)  
 7 / 2 = 3.50 (実数除算)

**要求事項:** - 整数除算と実数除算を区別すること - ゼロ除算のチェックは不要（基礎課題のため） - 適切な変数名を使用すること

**ファイル名:** ex2\_5\_simple\_calculator.c

## 挑戦課題

### 課題2-6: 文字とASCIIコード

文字とASCIIコードの関係を学ぶプログラムを作成してください。

**機能:** 1. ユーザーから文字を入力してもらう 2. その文字のASCIIコードを表示する 3. ASCIIコード表の一部を表示する

**出力例:**

文字を1つ入力してください: A  
入力された文字: A  
ASCIIコード: 65  
  
ASCII表（32-126）の一部:  
 32: 33: ! 34: " 35: # 36: $ 37: % 38: & 39: '  
 40: ( 41: ) 42: \* 43: + 44: , 45: - 46: . 47: /  
 48: 0 49: 1 50: 2 51: 3 52: 4 53: 5 54: 6 55: 7  
 56: 8 57: 9 58: : 59: ; 60: < 61: = 62: > 63: ?  
 64: @ 65: A 66: B 67: C 68: D 69: E 70: F 71: G

**要求事項:** - getchar()またはscanf(" %c", &ch)を使用すること - ループを使ってASCII表を表示すること - 適切な書式で表を整列させること

**ファイル名:** ex2\_6\_ascii\_explorer.c

## 提出について

### ファイル構成

solutions/  
├── ex2\_1\_hello\_intro.c  
├── ex2\_2\_personal\_data.c  
├── ex2\_3\_age\_calculator.c  
├── ex2\_4\_format\_demo.c  
├── ex2\_5\_simple\_calculator.c  
└── ex2\_6\_ascii\_explorer.c

### コンパイルと実行

# コンパイル例  
gcc -o hello\_intro ex2\_1\_hello\_intro.c  
gcc -o personal\_data ex2\_2\_personal\_data.c  
gcc -o age\_calculator ex2\_3\_age\_calculator.c  
  
# 実行例  
./hello\_intro  
./personal\_data  
./age\_calculator

### 評価ポイント

1. **基本文法**: 正しいC言語の文法で記述されているか
2. **コーディング規約**: 適切なインデント、変数名、コメントが使用されているか
3. **機能性**: 要求された機能が正しく実装されているか
4. **入出力**: printf/scanfが適切に使用されているか
5. **書式指定**: 適切な書式指定子が使用されているか

### 学習のポイント

* C言語の基本的な入出力方法
* 変数の宣言と初期化
* 基本データ型の理解
* 書式指定子の使い方
* コメントの書き方
* プログラムの基本構造

頑張って取り組んでください！

## 解答例

## 解答例一覧

### 演習2-1: 自己紹介プログラム

* **ファイル**: <ex2_1_hello_intro.c>, <ex2_1_hello_intro_c99.c>
* **学習内容**: printf文の基本的な使い方、改行文字の使用
* **ポイント**: 複数行の出力とフォーマットの基本

### 演習2-2: 個人データ表示

* **ファイル**: <ex2_2_personal_data.c>, <ex2_2_personal_data_c99.c>
* **学習内容**: 変数の宣言と初期化、各種データ型の使用
* **ポイント**: int, float, char型の実用的な使い方

### 演習2-3: 年齢計算プログラム

* **ファイル**: <ex2_3_age_calculator.c>, <ex2_3_age_calculator_c99.c>
* **学習内容**: scanf関数による入力、計算処理
* **ポイント**: ユーザー入力を受け取って計算結果を表示

### 演習2-4: フォーマット指定子の実践

* **ファイル**: <ex2_4_format_demo.c>, <ex2_4_format_demo_c99.c>
* **学習内容**: printf文の書式指定子、桁数指定
* **ポイント**: %d, %f, %c, %sの実用的な使い方

### 演習2-5: 簡単な計算機

* **ファイル**: <ex2_5_simple_calculator.c>, <ex2_5_simple_calculator_c99.c>
* **学習内容**: 四則演算、浮動小数点数の処理
* **ポイント**: 複数の入力と計算結果の表示

### 演習2-6: ASCII文字探索

* **ファイル**: <ex2_6_ascii_explorer.c>, <ex2_6_ascii_explorer_c99.c>
* **学習内容**: 文字と数値の変換、ASCIIコードの理解
* **ポイント**: char型と整数型の関係性

## 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
* /\* \*/ 形式のコメントを使用
* 従来のC言語の書き方に準拠

### 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
* // 形式のコメントに対応
* より現代的なC言語の書き方

## コンパイルと実行

### C90版

gcc -Wall -Wextra -pedantic -std=c90 hello\_intro.c -o hello\_intro  
./hello\_intro

### C99版

gcc -Wall -Wextra -pedantic -std=c99 hello\_intro\_c99.c -o hello\_intro\_c99  
./hello\_intro\_c99

### Makefileを使用

make all # 全ての解答例をコンパイル  
make clean # 実行ファイルを削除  
make run-hello # hello\_introを実行

## 学習のポイント

1. **基本的な出力**: printf文の正しい使い方を習得
2. **変数の扱い**: 各種データ型の特徴を理解
3. **入力処理**: scanf関数の安全な使用方法
4. **書式指定**: 適切なフォーマット指定子の選択
5. **エラー処理**: 基本的な入力検証の考え方

## 注意事項

* すべての解答例は教育目的で作成されています
* 実際の開発では、より厳密なエラーハンドリングが必要です
* C90版を基本として学習し、C99版で現代的な書き方を確認してください

## 次のステップ

この章をマスターしたら、次は以下の章に進みましょう： - [第3章: データ型と演算子](../data-types/) - [第4章: 制御構造（条件分岐）](../control-if/) ### ex2\_1\_hello\_intro.c

/\*  
 \* 課題2-1の解答例: 基本的な出力  
 \* ファイル名: ex2\_1\_hello\_intro.c  
 \* 説明: printf関数とエスケープシーケンスの基本的な使用  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 基本的な文字列出力 \*/  
 printf("こんにちは、C言語の世界へ！\n");  
 printf("私の名前は 田中太郎 です。\n");  
 printf("今日からプログラミングを始めます。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. #include <stdio.h> - 標準入出力ライブラリの取り込み  
2. int main(void) - プログラムのエントリーポイント  
3. printf関数 - 文字列の出力  
4. \n - 改行を表すエスケープシーケンス  
5. return 0 - プログラムの正常終了を示す  
\*/

### ex2\_2\_personal\_data.c

/\*  
 \* 課題2-2の解答例: 変数と基本データ型  
 \* ファイル名: ex2\_2\_personal\_data.c  
 \* 説明: 基本データ型の変数宣言、初期化、書式指定子の使用  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言と初期化 \*/  
 int age = 25; /\* 年齢（整数） \*/  
 float height = 170.5f; /\* 身長（浮動小数点数） \*/  
 char blood\_type = 'A'; /\* 血液型（文字） \*/  
 int favorite\_number = 7; /\* 好きな数字（整数） \*/  
   
 /\* 自己紹介データの表示 \*/  
 printf("=== 自己紹介データ ===\n");  
 printf("年齢: %d歳\n", age);  
 printf("身長: %.1fcm\n", height);  
 printf("血液型: %c型\n", blood\_type);  
 printf("好きな数字: %d\n", favorite\_number);  
 printf("================\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 基本データ型の使い分け:  
 - int: 整数値  
 - float: 浮動小数点数（単精度）  
 - char: 1文字  
   
2. 変数の初期化:  
 - 宣言と同時に値を代入  
 - float型には f を付ける（170.5f）  
 - char型には シングルクォート（'A'）  
   
3. printf関数の書式指定子:  
 - %d: 整数（decimal）  
 - %f: 浮動小数点数（float）  
 - %c: 文字（character）  
 - %.1f: 小数点以下1桁表示  
   
4. コメントの書き方:  
 - スラッシュとアスタリスクで囲む（C90準拠）  
 - 変数の説明を記述  
\*/

### ex2\_3\_age\_calculator.c

/\*  
 \* 課題2-3の解答例: 入力と出力  
 \* ファイル名: ex2\_3\_age\_calculator.c  
 \* 説明: scanf関数を使った入力処理と計算  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 char name[50]; /\* 名前を格納する文字配列（50文字まで） \*/  
 int current\_age; /\* 現在の年齢 \*/  
 int future\_age; /\* 10年後の年齢 \*/  
   
 /\* ユーザーからの入力 \*/  
 printf("お名前を入力してください: ");  
 scanf("%49s", name); /\* 49文字まで読み込み（バッファオーバーフロー対策） \*/  
   
 printf("年齢を入力してください: ");  
 scanf("%d", &current\_age);  
   
 /\* 10年後の年齢を計算 \*/  
 future\_age = current\_age + 10;  
   
 /\* 結果の表示 \*/  
 printf("\n");  
 printf("こんにちは、%sさん！\n", name);  
 printf("現在の年齢: %d歳\n", current\_age);  
 printf("10年後の年齢: %d歳\n", future\_age);  
 printf("プログラミングの学習、頑張ってください！\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. scanf関数の使用:  
 - %s: 文字列の入力  
 - %d: 整数の入力  
 - & (アドレス演算子): 変数のアドレスを指定  
   
2. 文字配列の宣言:  
 - char name[50]: 50文字まで格納可能  
 - scanf("%49s", name): バッファオーバーフロー対策  
   
3. 計算処理:  
 - future\_age = current\_age + 10  
 - 計算結果を変数に保存  
   
4. 改行文字の使用:  
 - \n で改行  
 - printf("\n") で空行挿入  
   
注意点:  
- scanf使用時はバッファサイズに注意  
- 実際のプログラムでは入力エラーのチェックも必要  
- 配列のサイズは格納する文字数+1（終端文字\0のため）  
\*/

### ex2\_4\_format\_demo.c

/\*  
 \* 課題2-4の解答例: 書式指定子の練習  
 \* ファイル名: ex2\_4\_format\_demo.c  
 \* 説明: printf関数の様々な書式指定子の使用方法  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int integer\_num = 123;  
 double real\_num = 3.14159;  
 unsigned char char\_code = 255;  
   
 printf("=== 書式指定子のデモ ===\n");  
   
 /\* 整数の様々な表示形式 \*/  
 printf("整数 %d の表示:\n", integer\_num);  
 printf(" 10進数: %d\n", integer\_num);  
 printf(" 16進数: %x\n", integer\_num);  
 printf(" 8進数: %o\n", integer\_num);  
 printf(" フィールド幅5: |%5d|\n", integer\_num);  
 printf(" ゼロ埋め: |%05d|\n", integer\_num);  
 printf("\n");  
   
 /\* 実数の様々な表示形式 \*/  
 printf("実数 %.5f の表示:\n", real\_num);  
 printf(" デフォルト: %f\n", real\_num);  
 printf(" 小数点以下2桁: %.2f\n", real\_num);  
 printf(" 指数表記: %e\n", real\_num);  
 printf(" フィールド幅10.2: |%10.2f|\n", real\_num);  
 printf("\n");  
   
 /\* 文字コードの表示 \*/  
 printf("文字コード %d:\n", char\_code);  
 printf(" 文字として: %c\n", char\_code);  
 printf(" 16進数: %x\n", char\_code);  
 printf(" 10進数: %d\n", char\_code);  
   
 printf("====================\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 整数の書式指定子:  
 - %d: 10進数表示  
 - %x: 16進数表示（小文字）  
 - %X: 16進数表示（大文字）  
 - %o: 8進数表示  
   
2. フィールド幅指定:  
 - %5d: 5桁の幅で右寄せ  
 - %05d: 5桁の幅でゼロ埋め  
 - |%5d|: 表示範囲を見やすくするための記号  
   
3. 浮動小数点数の書式指定子:  
 - %f: 通常の小数表示  
 - %.2f: 小数点以下2桁まで表示  
 - %e: 指数表記（科学的記数法）  
 - %10.2f: 幅10、小数点以下2桁  
   
4. 文字の書式指定子:  
 - %c: 文字として表示  
 - %d: 文字コード（整数）として表示  
   
応用ポイント:  
- 複数の書式を組み合わせて使用可能  
- 表の整列や見やすい出力に活用  
- 数値の表示形式を用途に応じて選択  
\*/

### ex2\_5\_simple\_calculator.c

/\*  
 \* 課題2-5の解答例: 簡単な計算機  
 \* ファイル名: ex2\_5\_simple\_calculator.c  
 \* 説明: 四則演算の実装と整数除算・実数除算の違い  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int num1, num2; /\* 計算に使用する2つの整数 \*/  
 int int\_result; /\* 整数演算の結果 \*/  
 double double\_result; /\* 実数演算の結果 \*/  
   
 /\* プログラムのタイトル表示 \*/  
 printf("簡単な計算機プログラム\n");  
 printf("===================\n");  
   
 /\* 数値の入力 \*/  
 printf("第1の数値を入力してください: ");  
 scanf("%d", &num1);  
   
 printf("第2の数値を入力してください: ");  
 scanf("%d", &num2);  
   
 printf("\n");  
   
 /\* 四則演算の実行と結果表示 \*/  
 printf("計算結果:\n");  
   
 /\* 加算 \*/  
 int\_result = num1 + num2;  
 printf(" %d + %d = %d\n", num1, num2, int\_result);  
   
 /\* 減算 \*/  
 int\_result = num1 - num2;  
 printf(" %d - %d = %d\n", num1, num2, int\_result);  
   
 /\* 乗算 \*/  
 int\_result = num1 \* num2;  
 printf(" %d \* %d = %d\n", num1, num2, int\_result);  
   
 /\* 除算（整数除算） \*/  
 int\_result = num1 / num2;  
 printf(" %d / %d = %d (整数除算)\n", num1, num2, int\_result);  
   
 /\* 除算（実数除算） \*/  
 double\_result = (double)num1 / num2; /\* キャストを使用 \*/  
 printf(" %d / %d = %.2f (実数除算)\n", num1, num2, double\_result);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 四則演算子:  
 - + : 加算  
 - - : 減算  
 - \* : 乗算（アスタリスク）  
 - / : 除算  
   
2. 整数除算と実数除算の違い:  
 - int / int → 整数除算（小数点以下切り捨て）  
 - (double)int / int → 実数除算（小数点以下あり）  
   
3. 型変換（キャスト）:  
 - (double)num1 → num1を一時的にdouble型に変換  
 - これにより実数除算が行われる  
   
4. 書式指定子:  
 - %d: 整数  
 - %.2f: 小数点以下2桁の実数  
   
5. プログラムの構成:  
 - 入力 → 処理 → 出力の基本的な流れ  
 - 見やすい出力のためのフォーマット  
   
注意点:  
- ゼロ除算のチェックは省略（基礎課題のため）  
- 実際のプログラムでは num2 が 0 でないかチェックが必要  
- キャストは演算の前に行う必要がある  
\*/

### ex2\_6\_ascii\_explorer.c

/\*  
 \* 課題2-6の解答例: 文字とASCIIコード  
 \* ファイル名: ex2\_6\_ascii\_explorer.c  
 \* 説明: 文字とASCIIコードの関係、ループによるASCII表の表示  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 char input\_char; /\* ユーザーが入力した文字 \*/  
 int ascii\_code; /\* ASCIIコード値 \*/  
 int i; /\* ループ用カウンタ \*/  
   
 /\* ユーザーからの文字入力 \*/  
 printf("文字を1つ入力してください: ");  
 scanf(" %c", &input\_char); /\* 空白文字をスキップするため先頭に空白 \*/  
   
 /\* 入力された文字のASCIIコードを表示 \*/  
 ascii\_code = (int)input\_char; /\* 文字を整数に変換 \*/  
 printf("入力された文字: %c\n", input\_char);  
 printf("ASCIIコード: %d\n", ascii\_code);  
 printf("\n");  
   
 /\* ASCII表の一部を表示 \*/  
 printf("ASCII表（32-126）の一部:\n");  
   
 /\* 32から126までのASCII文字を8文字ずつ表示 \*/  
 for (i = 32; i <= 126; i++) {  
 /\* 表示可能文字かチェック \*/  
 if (i == 32) {  
 printf("%3d: ", i); /\* スペース文字は特別に表示 \*/  
 } else {  
 printf("%3d: %c ", i, (char)i);  
 }  
   
 /\* 8文字ごとに改行 \*/  
 if ((i - 31) % 8 == 0) {  
 printf("\n");  
 }  
 }  
   
 /\* 追加情報の表示 \*/  
 printf("\n");  
 printf("ASCII表の説明:\n");  
 printf(" 0-31 : 制御文字（表示されない）\n");  
 printf(" 32 : スペース文字\n");  
 printf(" 33-126: 表示可能文字（英数字、記号など）\n");  
 printf(" 127 : DEL文字（制御文字）\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 文字とASCIIコードの関係:  
 - 文字は内部的には数値（ASCIIコード）として扱われる  
 - (int)char\_var → 文字を整数に変換  
 - (char)int\_var → 整数を文字に変換  
   
2. scanf関数での文字入力:  
 - scanf(" %c", &var) → 先頭の空白で前の入力の改行文字をスキップ  
 - %c 指定子で1文字を読み込み  
   
3. forループの使用:  
 - for (i = 開始値; 条件; 増分) { 処理 }  
 - ASCII表の連続表示に活用  
   
4. 条件分岐:  
 - if文による特別な処理（スペース文字の場合）  
 - 改行タイミングの制御  
   
5. 書式指定子の応用:  
 - %3d: 3桁幅での整数表示  
 - 表を整列させるためのフォーマット  
   
応用ポイント:  
- ASCII表は文字コードの基礎  
- 文字と数値の相互変換は文字列処理の基本  
- ループを使った規則的な処理の例  
- 表形式での見やすい出力方法  
  
注意点:  
- C90では forループ内での変数宣言は不可  
- 変数 i は関数の先頭で宣言する必要がある  
\*/

# 第3章 データ型と変数

## 演習問題

## 基礎問題

### 演習3-1. 変数宣言と初期化

さまざまなデータ型の変数を宣言し、初期化して値を表示するプログラムを作成してください。

**要件:** - 各基本データ型（char, short, int, long, float, double）の変数を宣言 - 適切な値で初期化 - printf関数で型に応じた書式指定子を使用して出力 - 符号付き・符号なしの両方を含める

**期待される出力例:**

char型: 文字 = 'A', 値 = 65  
unsigned char型: 値 = 255  
short型: 値 = -1000  
unsigned short型: 値 = 65535  
int型: 値 = -123456  
unsigned int型: 値 = 4294967295  
float型: 値 = 3.14159  
double型: 値 = 3.141592653589793

### 演習3-2. 四則演算計算機

2つの数値を変数に格納し、四則演算（加算、減算、乗算、除算）の結果を表示するプログラムを作成してください。

**要件:** - 整数型と浮動小数点型の両方で実装 - 除算では整数除算と実数除算の違いを表示 - 各演算結果を見やすく表示

### 演習3-3. データ型サイズの確認

sizeof演算子を使って、各データ型のサイズを表示するプログラムを作成してください。

**要件:** - 基本データ型すべてのサイズを表示 - 配列のサイズも確認 - ポインタのサイズも確認 - サイズをバイト単位で表示

## 応用問題

### 演習3-4. 型変換の理解

整数除算と実数除算の違い、および暗黙的・明示的型変換を確認するプログラムを作成してください。

**要件:** - 同じ数値で整数除算と実数除算を実行し、結果を比較 - 暗黙的型変換が発生する例を実装 - 明示的型変換（キャスト）を使用した例を実装 - 精度の損失が発生する例を示す

### 演習3-5. スコープの実験

グローバル変数、ローカル変数、静的変数を使い分けるプログラムを作成してください。

**要件:** - 同じ名前の変数を異なるスコープで宣言 - スコープの隠蔽（シャドウイング）を実演 - 静的変数を使ったカウンター関数を実装 - 各変数の値の変化を追跡して表示

### 演習3-6. 定数の活用

constと#defineを使って定数を定義し、円の面積と円周を計算するプログラムを作成してください。

**要件:** - 円周率をconstと#defineの両方で定義 - 半径を入力として受け取る - 面積と円周を計算して表示 - 定数を変更しようとした場合のエラーを確認（コメントで説明）

## チャレンジ問題

### 温度変換プログラム

摂氏・華氏・ケルビンの温度単位を相互変換するプログラムを作成してください。

**要件:** - 変換式で使用する定数を適切に定義 - 浮動小数点演算の精度に注意 - ユーザーフレンドリーな入出力 - 変換公式： - 華氏 = 摂氏 × 9/5 + 32 - ケルビン = 摂氏 + 273.15

## 提出方法

1. 各問題に対して別々のCファイルを作成
2. ファイル名は ex3\_1\_variables.c, ex3\_2\_calculator.c のように命名
3. 各プログラムの冒頭にコメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認

## コンパイル例

gcc -std=c90 -Wall -Wextra -pedantic ex\_variables.c -o ex\_variables  
./ex\_variables

## ヒント

* printf関数の書式指定子を正しく使用する
  + %d: int
  + %u: unsigned int
  + %ld: long
  + %f: float/double
  + %c: char
* 初期化されていない変数の使用に注意
* 型の範囲を超えないよう注意
* sizeof演算子の戻り値はsize\_t型（%luで出力）

## 解答例

この章の演習問題の解答例を示します。各解答にはC90準拠版とC99準拠版を用意し、両者の違いを学習できるようにしています。

## 演習解答一覧

### 演習3-1: 変数宣言と初期化

**ファイル:** <ex3_1_variable_declaration.c> | [C99版](ex3_1_variable_declaration_c99.c)

さまざまなデータ型の変数を宣言し、初期化して出力するプログラムです。

**学習内容:** - 各データ型の適切な宣言方法 - 初期化の重要性 - printf関数での型に応じた書式指定子の使用 - sizeof演算子によるサイズ確認 - 定数の定義と使用

### 演習3-2: 四則演算計算機

**ファイル:** <ex3_2_calculator.c> | [C99版](ex3_2_calculator_c99.c)

2つの数値を変数に格納し、四則演算の結果を表示するプログラムです。

**学習内容:** - 整数演算と実数演算の違い - 四則演算子 (+, -, \*, /, %) の使い方 - 整数除算では小数点以下が切り捨てられる - 剰余演算子(%)は整数のみで使用可能 - 混合演算での自動型変換 - 明示的型変換 (キャスト) の使用

### 演習3-3: データ型サイズの確認

**ファイル:** <ex3_3_sizeof_demo.c> | [C99版](ex3_3_sizeof_demo_c99.c)

sizeof演算子を使って、各データ型のサイズを表示するプログラムです。

**学習内容:** - sizeof演算子でデータ型や変数のサイズを確認 - limits.hヘッダで整数型の範囲定数を取得 - float.hヘッダで浮動小数点型の特性を取得 - 配列のサイズと要素数の計算方法 - ポインタのサイズはシステム依存 - C99の固定幅整数型（stdint.h）

### 演習3-4: 型変換の理解

**ファイル:** <ex3_4_type_conversion.c> | [C99版](ex3_4_type_conversion_c99.c)

整数除算と実数除算の違いを確認し、型変換について学習するプログラムです。

**学習内容:** - 整数除算は小数点以下が切り捨てられる - 暗黙の型変換は自動的に行われる - 明示的な型変換（キャスト）で強制的に変換できる - 型変換には優先順位がある - charとintの間でASCII値による変換が可能 - オーバーフローは上位ビットの切り捨てで発生

### 演習3-5: スコープの実験

**ファイル:** <ex3_5_scope_experiment.c> | [C99版](ex3_5_scope_experiment_c99.c)

グローバル変数とローカル変数を使い分け、スコープの概念を学習するプログラムです。

**学習内容:** - グローバル変数はファイル全体でアクセス可能 - ローカル変数は宣言されたブロック内でのみ有効 - 静的ローカル変数は関数呼出間で値を保持 - static指定子でファイル内限定のグローバル変数を作成可能 - 内側のスコープで同名変数を宣言すると外側を隠蔽 - ブロックスコープは {} で区切られた範囲

### 演習3-6: 定数の活用

**ファイル:** <ex3_6_constants_demo.c> | [C99版](ex3_6_constants_demo_c99.c)

constと#defineを使って定数を定義し、物理計算に使用するプログラムです。

**学習内容:** - #defineによるマクロ定数の定義 - const変数による定数の定義 - マクロ関数の作成と使用 - 物理計算での定数の活用 - 定数の命名規則（大文字、アンダースコア） - #define vs const の違い

## C90とC99の主な違い

### C90の特徴

* 変数宣言は関数やブロックの先頭で行う必要がある
* /\* \*/ 形式のコメントのみ使用可能
* 配列サイズは定数式である必要がある
* long long型は使用できない

### C99の新機能

* 変数を使用箇所で宣言可能
* // 形式のコメントが使用可能
* for文内で変数宣言可能 (for (int i = 0; …))
* 可変長配列（VLA）をサポート
* long long型の追加（64ビット整数）
* 固定幅整数型（int8\_t, int16\_t, int32\_t, int64\_t等）

## コンパイル方法

### C90準拠でコンパイル

gcc -std=c90 -Wall -Wextra -pedantic filename.c -o output

### C99準拠でコンパイル

gcc -std=c99 -Wall -Wextra -pedantic filename.c -o output

### 数学ライブラリが必要な場合

gcc -std=c90 -Wall -Wextra -pedantic filename.c -o output -lm

## 重要な学習ポイント

### データ型の選択基準

* **char**: 文字データ、小さな整数
* **short**: 小さな整数（メモリ節約）
* **int**: 一般的な整数計算
* **long**: 大きな整数
* **unsigned**: 負の値を扱わない場合
* **float**: 精度をそれほど要求しない実数
* **double**: 高精度な実数計算

### 変数のスコープと生存期間

* **自動変数**: ブロック開始時に作成、終了時に破棄
* **静的変数**: プログラム開始時に作成、終了まで存続
* **グローバル変数**: プログラム全体で存続

### 定数の使い分け

* **#define**: プリプロセッサによる文字列置換、型情報なし
* **const**: 型付き定数、デバッガで値確認可能、スコープあり

## チャレンジ問題

### 演習3-7: 温度変換プログラム

**ファイル:** <ex3_7_temperature_converter.c> | [C99版](ex3_7_temperature_converter_c99.c)

摂氏・華氏・ケルビンの温度単位を相互変換するプログラムです。

**学習内容:** - #defineマクロによる定数の定義 - 浮動小数点演算の精度管理 - 物理的な意味を持つ計算の実装 - 複数の単位系の相互変換 - 科学技術計算での定数活用

## よくある間違いと対策

### 1. 初期化忘れ

/\* NG: 初期化せずに使用 \*/  
int count;  
printf("%d\n", count); /\* 未定義値が出力される \*/  
  
/\* OK: 初期化してから使用 \*/  
int count = 0;  
printf("%d\n", count);

### 2. 整数除算の結果

/\* NG: 期待した結果にならない \*/  
int result = 5 / 2; /\* 結果: 2 \*/  
  
/\* OK: 実数除算を使用 \*/  
double result = 5.0 / 2.0; /\* 結果: 2.5 \*/

### 3. 型の範囲超過

/\* NG: charの範囲を超える \*/  
char big\_num = 300; /\* オーバーフロー \*/  
  
/\* OK: 適切な型を使用 \*/  
int big\_num = 300;

これらの演習を通じて、C言語の基本データ型、変数のスコープ、定数の活用方法を習得してください。 ### ex3\_1\_variable\_declaration.c

/\*  
 \* ファイル名: ex3\_1\_variable\_declaration.c  
 \* 演習3-1: 変数宣言と初期化  
 \* 説明: さまざまなデータ型の変数を宣言し、初期化して出力する  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 整数型の変数宣言と初期化 \*/  
 char initial = 'C';  
 short year = 2024;  
 int score = 95;  
 long population = 125000000L;  
 unsigned int count = 42U;  
   
 /\* 浮動小数点型の変数宣言と初期化 \*/  
 float temperature = 23.5f;  
 double precision\_value = 3.141592653589793;  
   
 /\* 各種定数の定義 \*/  
 const int MAX\_STUDENTS = 30;  
 const double GRAVITY = 9.8;  
   
 printf("=== 変数宣言と初期化の例 ===\n");  
   
 /\* 整数型変数の出力 \*/  
 printf("文字型 (char) : %c\n", initial);  
 printf("短整数型 (short) : %d\n", year);  
 printf("整数型 (int) : %d\n", score);  
 printf("長整数型 (long) : %ld\n", population);  
 printf("符号なし整数 (unsigned): %u\n", count);  
   
 /\* 浮動小数点型変数の出力 \*/  
 printf("単精度浮動小数点 (float) : %.2f\n", temperature);  
 printf("倍精度浮動小数点 (double): %.15f\n", precision\_value);  
   
 /\* 定数の出力 \*/  
 printf("定数 MAX\_STUDENTS : %d\n", MAX\_STUDENTS);  
 printf("定数 GRAVITY : %.1f\n", GRAVITY);  
   
 /\* データ型のサイズ情報 \*/  
 printf("\n=== データ型のサイズ ===\n");  
 printf("char : %lu バイト\n", sizeof(char));  
 printf("short : %lu バイト\n", sizeof(short));  
 printf("int : %lu バイト\n", sizeof(int));  
 printf("long : %lu バイト\n", sizeof(long));  
 printf("unsigned : %lu バイト\n", sizeof(unsigned int));  
 printf("float : %lu バイト\n", sizeof(float));  
 printf("double : %lu バイト\n", sizeof(double));  
   
 /\* 実際の値の範囲例 \*/  
 printf("\n=== 値の範囲例 ===\n");  
 printf("char 最大値の例: %d\n", 127);  
 printf("char 最小値の例: %d\n", -128);  
 printf("unsigned int 最大値の例: %u\n", 4294967295U);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 各データ型の適切な宣言方法  
2. 初期化の重要性  
3. printf関数での型に応じた書式指定子の使用  
4. sizeof演算子によるサイズ確認  
5. 定数の定義と使用  
6. 各型の値の範囲の理解  
  
データ型の使い分け:  
- char: 文字データ、小さな整数  
- short: 小さな整数（メモリ節約）  
- int: 一般的な整数計算  
- long: 大きな整数  
- unsigned: 負の値を扱わない場合  
- float: 精度をそれほど要求しない実数  
- double: 高精度な実数計算  
\*/

### ex3\_2\_calculator.c

/\*  
 \* ファイル名: ex3\_2\_calculator.c  
 \* 演習3-2: 四則演算計算機  
 \* 説明: 2つの数値を変数に格納し、四則演算の結果を表示する  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 計算に使用する変数の宣言と初期化 \*/  
 int num1 = 15;  
 int num2 = 4;  
 double real1 = 15.5;  
 double real2 = 4.2;  
   
 printf("=== 四則演算計算機 ===\n");  
   
 /\* 整数による四則演算 \*/  
 printf("整数演算 (%d と %d):\n", num1, num2);  
 printf(" %d + %d = %d\n", num1, num2, num1 + num2);  
 printf(" %d - %d = %d\n", num1, num2, num1 - num2);  
 printf(" %d \* %d = %d\n", num1, num2, num1 \* num2);  
 printf(" %d / %d = %d (余り: %d)\n", num1, num2, num1 / num2, num1 % num2);  
   
 /\* 実数による四則演算 \*/  
 printf("\n実数演算 (%.1f と %.1f):\n", real1, real2);  
 printf(" %.1f + %.1f = %.2f\n", real1, real2, real1 + real2);  
 printf(" %.1f - %.1f = %.2f\n", real1, real2, real1 - real2);  
 printf(" %.1f \* %.1f = %.2f\n", real1, real2, real1 \* real2);  
 printf(" %.1f / %.1f = %.2f\n", real1, real2, real1 / real2);  
   
 /\* 混合演算（型変換の例） \*/  
 printf("\n混合演算（整数と実数）:\n");  
 printf(" %d + %.1f = %.2f\n", num1, real2, num1 + real2);  
 printf(" %.1f - %d = %.2f\n", real1, num2, real1 - num2);  
 printf(" %d \* %.1f = %.2f\n", num1, real2, num1 \* real2);  
 printf(" %.1f / %d = %.2f\n", real1, num2, real1 / num2);  
   
 /\* 整数除算と実数除算の比較 \*/  
 printf("\n整数除算 vs 実数除算の比較:\n");  
 printf(" %d / %d = %d (整数除算)\n", num1, num2, num1 / num2);  
 printf(" %d / %d = %.2f (実数除算)\n", num1, num2, (double)num1 / num2);  
 printf(" %.1f / %.1f = %.2f (実数除算)\n", real1, real2, real1 / real2);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 整数演算と実数演算の違い  
2. 四則演算子 (+, -, \*, /, %) の使い方  
3. 整数除算では小数点以下が切り捨てられる  
4. 剰余演算子(%)は整数のみで使用可能  
5. 混合演算での自動型変換  
6. 明示的型変換 (キャスト) の使用  
  
データ型の使い分け:  
- int: 整数計算、カウンタ、インデックス  
- double: 精密な実数計算、科学計算  
- float: メモリ制約がある場合の実数計算  
\*/

### ex3\_3\_sizeof\_demo.c

/\*  
 \* ファイル名: ex3\_3\_sizeof\_demo.c  
 \* 演習3-3: データ型サイズの確認  
 \* 説明: sizeof演算子を使って、各データ型のサイズを表示する  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <limits.h>  
#include <float.h>  
  
int main(void)  
{  
 printf("=== データ型サイズの確認 ===\n");  
   
 /\* 基本データ型のサイズ \*/  
 printf("基本データ型のサイズ:\n");  
 printf(" char : %lu バイト\n", sizeof(char));  
 printf(" signed char : %lu バイト\n", sizeof(signed char));  
 printf(" unsigned char : %lu バイト\n", sizeof(unsigned char));  
 printf(" short : %lu バイト\n", sizeof(short));  
 printf(" unsigned short: %lu バイト\n", sizeof(unsigned short));  
 printf(" int : %lu バイト\n", sizeof(int));  
 printf(" unsigned int : %lu バイト\n", sizeof(unsigned int));  
 printf(" long : %lu バイト\n", sizeof(long));  
 printf(" unsigned long : %lu バイト\n", sizeof(unsigned long));  
 printf(" float : %lu バイト\n", sizeof(float));  
 printf(" double : %lu バイト\n", sizeof(double));  
 printf(" long double : %lu バイト\n", sizeof(long double));  
   
 /\* 変数のサイズ \*/  
 {  
 char c = 'A';  
 int num = 42;  
 double value = 3.14;  
 int array[10];  
   
 printf("\n変数のサイズ:\n");  
 printf(" char変数 c : %lu バイト\n", sizeof(c));  
 printf(" int変数 num : %lu バイト\n", sizeof(num));  
 printf(" double変数 value: %lu バイト\n", sizeof(value));  
 printf(" int配列[10] : %lu バイト\n", sizeof(array));  
 printf(" 配列要素数 : %lu 個\n", sizeof(array) / sizeof(array[0]));  
 }  
   
 /\* ポインタのサイズ \*/  
 printf("\nポインタのサイズ:\n");  
 printf(" char\* : %lu バイト\n", sizeof(char\*));  
 printf(" int\* : %lu バイト\n", sizeof(int\*));  
 printf(" double\* : %lu バイト\n", sizeof(double\*));  
 printf(" void\* : %lu バイト\n", sizeof(void\*));  
   
 /\* 整数型の値の範囲 \*/  
 printf("\n=== 整数型の値の範囲 ===\n");  
 printf("char:\n");  
 printf(" 最小値: %d\n", CHAR\_MIN);  
 printf(" 最大値: %d\n", CHAR\_MAX);  
 printf("signed char:\n");  
 printf(" 最小値: %d\n", SCHAR\_MIN);  
 printf(" 最大値: %d\n", SCHAR\_MAX);  
 printf("unsigned char:\n");  
 printf(" 最大値: %u\n", UCHAR\_MAX);  
   
 printf("short:\n");  
 printf(" 最小値: %d\n", SHRT\_MIN);  
 printf(" 最大値: %d\n", SHRT\_MAX);  
 printf("unsigned short:\n");  
 printf(" 最大値: %u\n", USHRT\_MAX);  
   
 printf("int:\n");  
 printf(" 最小値: %d\n", INT\_MIN);  
 printf(" 最大値: %d\n", INT\_MAX);  
 printf("unsigned int:\n");  
 printf(" 最大値: %u\n", UINT\_MAX);  
   
 printf("long:\n");  
 printf(" 最小値: %ld\n", LONG\_MIN);  
 printf(" 最大値: %ld\n", LONG\_MAX);  
 printf("unsigned long:\n");  
 printf(" 最大値: %lu\n", ULONG\_MAX);  
   
 /\* 浮動小数点型の特性 \*/  
 printf("\n=== 浮動小数点型の特性 ===\n");  
 printf("float:\n");  
 printf(" 精度: %d 桁\n", FLT\_DIG);  
 printf(" 最小値: %e\n", FLT\_MIN);  
 printf(" 最大値: %e\n", FLT\_MAX);  
 printf(" イプシロン: %e\n", FLT\_EPSILON);  
   
 printf("double:\n");  
 printf(" 精度: %d 桁\n", DBL\_DIG);  
 printf(" 最小値: %e\n", DBL\_MIN);  
 printf(" 最大値: %e\n", DBL\_MAX);  
 printf(" イプシロン: %e\n", DBL\_EPSILON);  
   
 printf("long double:\n");  
 printf(" 精度: %d 桁\n", LDBL\_DIG);  
 printf(" 最小値: %Le\n", LDBL\_MIN);  
 printf(" 最大値: %Le\n", LDBL\_MAX);  
 printf(" イプシロン: %Le\n", LDBL\_EPSILON);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. sizeof演算子でデータ型や変数のサイズを確認  
2. limits.hヘッダで整数型の範囲定数を取得  
3. float.hヘッダで浮動小数点型の特性を取得  
4. 配列のサイズと要素数の計算方法  
5. ポインタのサイズはシステム依存（32bit: 4バイト、64bit: 8バイト）  
  
重要な定数:  
- CHAR\_MIN/MAX: char型の範囲  
- INT\_MIN/MAX: int型の範囲   
- FLT\_DIG: float型の有効桁数  
- DBL\_DIG: double型の有効桁数  
- FLT\_EPSILON: float型の最小誤差  
\*/

### ex3\_4\_type\_conversion.c

/\*  
 \* ファイル名: ex3\_4\_type\_conversion.c  
 \* 演習3-4: 型変換の理解  
 \* 説明: 整数除算と実数除算の違いを確認する  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言と初期化 \*/  
 int a = 7, b = 3;  
 float f1 = 7.0f, f2 = 3.0f;  
 double d1 = 7.0, d2 = 3.0;  
   
 printf("=== 型変換の理解 ===\n");  
   
 /\* 整数除算 vs 実数除算 \*/  
 printf("整数除算 vs 実数除算:\n");  
 printf(" %d / %d = %d (整数除算)\n", a, b, a / b);  
 printf(" %.1f / %.1f = %.6f (float除算)\n", f1, f2, f1 / f2);  
 printf(" %.1f / %.1f = %.15f (double除算)\n", d1, d2, d1 / d2);  
   
 /\* 暗黙の型変換 \*/  
 printf("\n暗黙の型変換:\n");  
 printf(" %d + %.1f = %.6f (intからfloatへ自動変換)\n", a, f2, a + f2);  
 printf(" %.1f + %d = %.6f (intからfloatへ自動変換)\n", f1, b, f1 + b);  
 printf(" %d + %.1f = %.15f (intからdoubleへ自動変換)\n", a, d2, a + d2);  
 printf(" %.6f + %.1f = %.15f (floatからdoubleへ自動変換)\n", f1, d2, f1 + d2);  
   
 /\* 明示的な型変換（キャスト） \*/  
 printf("\n明示的な型変換（キャスト）:\n");  
 printf(" %d / %d = %d (整数除算)\n", a, b, a / b);  
 printf(" (double)%d / %d = %.6f (明示的にdoubleに変換)\n", a, b, (double)a / b);  
 printf(" %d / (double)%d = %.6f (明示的にdoubleに変換)\n", a, b, a / (double)b);  
 printf(" (double)%d / (double)%d = %.6f (両方をdoubleに変換)\n", a, b, (double)a / (double)b);  
   
 /\* 小数の整数への変換 \*/  
 printf("\n小数の整数への変換:\n");  
 printf(" %.6f を int に変換: %d (小数点以下切り捨て)\n", f1, (int)f1);  
 printf(" %.6f を int に変換: %d (小数点以下切り捨て)\n", f1/f2, (int)(f1/f2));  
 printf(" %.15f を int に変換: %d (小数点以下切り捨て)\n", d1/d2, (int)(d1/d2));  
   
 /\* 負数の型変換 \*/  
 {  
 int neg\_int = -5;  
 float neg\_float = -2.7f;  
 double neg\_double = -3.14159;  
   
 printf("\n負数の型変換:\n");  
 printf(" %d を float に変換: %.1f\n", neg\_int, (float)neg\_int);  
 printf(" %.1f を int に変換: %d (小数点以下切り捨て)\n", neg\_float, (int)neg\_float);  
 printf(" %.5f を int に変換: %d (小数点以下切り捨て)\n", neg\_double, (int)neg\_double);  
 }  
   
 /\* 文字と数値の変換 \*/  
 {  
 char digit\_char = '5';  
 char letter\_char = 'A';  
 int ascii\_value;  
   
 printf("\n文字と数値の変換:\n");  
 ascii\_value = digit\_char;  
 printf(" 文字 '%c' のASCII値: %d\n", digit\_char, ascii\_value);  
 printf(" 数字文字 '%c' を数値に: %d (ASCII値 - '0')\n", digit\_char, digit\_char - '0');  
   
 ascii\_value = letter\_char;  
 printf(" 文字 '%c' のASCII値: %d\n", letter\_char, ascii\_value);  
 printf(" ASCII値 %d を文字に: '%c'\n", 66, (char)66);  
 }  
   
 /\* 型変換の優先順位 \*/  
 printf("\n型変換の優先順位（算術変換）:\n");  
 printf(" char + int = int型\n");  
 printf(" int + float = float型\n");  
 printf(" float + double = double型\n");  
 printf(" unsigned + signed = unsigned型（同サイズの場合）\n");  
   
 /\* オーバーフローの例 \*/  
 {  
 char small\_char = 100;  
 int big\_int = 300;  
   
 printf("\nオーバーフローの例:\n");  
 printf(" char変数(100) に 200 を加算: %d\n", small\_char + 200);  
 printf(" int値 %d を char に変換: %d (上位ビット切り捨て)\n", big\_int, (char)big\_int);  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 整数除算は小数点以下が切り捨てられる  
2. 暗黙の型変換は自動的に行われる  
3. 明示的な型変換（キャスト）で強制的に変換できる  
4. 型変換には優先順位がある  
5. charとintの間でASCII値による変換が可能  
6. オーバーフローは上位ビットの切り捨てで発生  
  
型変換の優先順位（低→高）:  
char → short → int → long → float → double → long double  
  
注意点:  
- 精度の低い型への変換では情報が失われる可能性  
- 符号付きから符号なしへの変換は予期しない結果を生む場合  
- オーバーフローは未定義動作の原因となる場合がある  
\*/

### ex3\_5\_scope\_experiment.c

/\*  
 \* ファイル名: ex3\_5\_scope\_experiment.c  
 \* 演習3-5: スコープの実験  
 \* 説明: グローバル変数とローカル変数を使い分ける  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* グローバル変数 \*/  
int global\_counter = 0;  
double global\_total = 0.0;  
static int file\_private = 100; /\* ファイル内でのみアクセス可能 \*/  
  
/\* 関数プロトタイプ宣言 \*/  
void increment\_counter(void);  
void add\_to\_total(double value);  
void static\_demo(void);  
void scope\_hiding\_demo(void);  
int get\_next\_id(void);  
  
int main(void)  
{  
 /\* ローカル変数 \*/  
 int local\_var = 10;  
   
 printf("=== スコープの実験 ===\n");  
   
 /\* グローバル変数の初期値表示 \*/  
 printf("初期状態:\n");  
 printf(" global\_counter = %d\n", global\_counter);  
 printf(" global\_total = %.1f\n", global\_total);  
 printf(" local\_var = %d\n", local\_var);  
 printf(" file\_private = %d\n", file\_private);  
   
 /\* グローバル変数の操作 \*/  
 printf("\nグローバル変数の操作:\n");  
 increment\_counter();  
 increment\_counter();  
 add\_to\_total(15.5);  
 add\_to\_total(24.3);  
   
 printf(" 操作後 global\_counter = %d\n", global\_counter);  
 printf(" 操作後 global\_total = %.1f\n", global\_total);  
   
 /\* ローカル変数の操作 \*/  
 printf("\nローカル変数の操作:\n");  
 local\_var += 5;  
 printf(" 操作後 local\_var = %d\n", local\_var);  
   
 /\* 静的変数のデモ \*/  
 printf("\n静的変数のデモ:\n");  
 static\_demo();  
 static\_demo();  
 static\_demo();  
   
 /\* IDジェネレータのデモ \*/  
 printf("\nIDジェネレータのデモ:\n");  
 printf(" ID: %d\n", get\_next\_id());  
 printf(" ID: %d\n", get\_next\_id());  
 printf(" ID: %d\n", get\_next\_id());  
   
 /\* スコープの隠蔽デモ \*/  
 printf("\nスコープの隠蔽デモ:\n");  
 scope\_hiding\_demo();  
   
 /\* ブロックスコープのデモ \*/  
 printf("\nブロックスコープのデモ:\n");  
 {  
 int block\_var = 99;  
 printf(" ブロック内 block\_var = %d\n", block\_var);  
 printf(" ブロック内 local\_var = %d\n", local\_var); /\* main()のローカル変数にアクセス \*/  
   
 {  
 int nested\_var = 88;  
 printf(" さらに内側 nested\_var = %d\n", nested\_var);  
 printf(" さらに内側 block\_var = %d\n", block\_var); /\* 外側のblock\_varにアクセス \*/  
 }  
 /\* nested\_varはここではアクセス不可 \*/  
 }  
 /\* block\_varはここではアクセス不可 \*/  
   
 return 0;  
}  
  
/\* グローバルカウンタをインクリメントする関数 \*/  
void increment\_counter(void)  
{  
 global\_counter++;  
 printf(" increment\_counter(): global\_counter = %d\n", global\_counter);  
}  
  
/\* グローバル合計に値を加算する関数 \*/  
void add\_to\_total(double value)  
{  
 global\_total += value;  
 printf(" add\_to\_total(%.1f): global\_total = %.1f\n", value, global\_total);  
}  
  
/\* 静的ローカル変数のデモ \*/  
void static\_demo(void)  
{  
 static int static\_count = 0; /\* 静的ローカル変数、初期化は1回のみ \*/  
 int auto\_count = 0; /\* 自動変数、関数呼出ごとに初期化 \*/  
   
 static\_count++;  
 auto\_count++;  
   
 printf(" static\_demo() 呼出: static\_count = %d, auto\_count = %d\n",   
 static\_count, auto\_count);  
}  
  
/\* ユニークID生成器 \*/  
int get\_next\_id(void)  
{  
 static int id\_counter = 1000; /\* 初期値1000から開始 \*/  
   
 return id\_counter++; /\* 現在値を返してからインクリメント \*/  
}  
  
/\* スコープの隠蔽（シャドウイング）のデモ \*/  
void scope\_hiding\_demo(void)  
{  
 int global\_counter = 999; /\* グローバル変数を隠蔽するローカル変数 \*/  
   
 printf(" 関数内 global\_counter = %d (ローカル変数)\n", global\_counter);  
 printf(" 実際のグローバル変数は隠蔽されている\n");  
   
 {  
 int global\_counter = 777; /\* さらに内側で隠蔽 \*/  
 printf(" ブロック内 global\_counter = %d (より内側のローカル変数)\n", global\_counter);  
 }  
   
 printf(" ブロック外 global\_counter = %d (関数のローカル変数に戻る)\n", global\_counter);  
}  
  
/\*  
学習ポイント:  
1. グローバル変数はファイル全体でアクセス可能  
2. ローカル変数は宣言されたブロック内でのみ有効  
3. 静的ローカル変数は関数呼出間で値を保持  
4. static指定子でファイル内限定のグローバル変数を作成可能  
5. 内側のスコープで同名変数を宣言すると外側を隠蔽  
6. ブロックスコープは {} で区切られた範囲  
  
スコープの種類:  
- ファイルスコープ: グローバル変数、関数  
- 関数スコープ: 関数内のローカル変数  
- ブロックスコープ: {} 内の変数  
  
変数の生存期間:  
- 自動変数: ブロック開始時に作成、終了時に破棄  
- 静的変数: プログラム開始時に作成、終了まで存続  
- グローバル変数: プログラム全体で存続  
  
実践的な使用例:  
- グローバル変数: 設定値、状態管理  
- 静的ローカル変数: カウンタ、キャッシュ  
- ローカル変数: 一時的な計算、処理用データ  
\*/

### ex3\_6\_constants\_demo.c

/\*  
 \* ファイル名: ex3\_6\_constants\_demo.c  
 \* 演習3-6: 定数の活用  
 \* 説明: constと#defineを使って定数を定義し、計算に使用する  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <math.h>  
  
/\* プリプロセッサ定数（マクロ定数） \*/  
#define PI 3.14159265358979323846  
#define MAX\_STUDENTS 30  
#define PROGRAM\_NAME "定数活用デモ"  
#define VERSION "1.0"  
#define GRAVITY 9.8  
#define LIGHT\_SPEED 299792458L /\* 光速 (m/s) \*/  
  
/\* 計算マクロ \*/  
#define SQUARE(x) ((x) \* (x))  
#define CIRCLE\_AREA(r) (PI \* SQUARE(r))  
#define FAHRENHEIT\_TO\_CELSIUS(f) (((f) - 32.0) \* 5.0 / 9.0)  
  
/\* 関数プロトタイプ \*/  
void const\_variables\_demo(void);  
void macro\_constants\_demo(void);  
void calculation\_demo(void);  
void physics\_calculations(void);  
  
int main(void)  
{  
 printf("=== %s v%s ===\n", PROGRAM\_NAME, VERSION);  
   
 /\* const変数のデモ \*/  
 const\_variables\_demo();  
   
 /\* マクロ定数のデモ \*/  
 macro\_constants\_demo();  
   
 /\* 定数を使った計算のデモ \*/  
 calculation\_demo();  
   
 /\* 物理計算のデモ \*/  
 physics\_calculations();  
   
 return 0;  
}  
  
/\* const変数のデモ \*/  
void const\_variables\_demo(void)  
{  
 /\* const変数の宣言と初期化 \*/  
 const int MAX\_RETRY = 3;  
 const double TAX\_RATE = 0.08;  
 const char GRADE\_A = 'A';  
 const float DISCOUNT = 0.15f;  
   
 printf("\n=== const変数のデモ ===\n");  
 printf("最大リトライ回数: %d\n", MAX\_RETRY);  
 printf("税率: %.1f%%\n", TAX\_RATE \* 100);  
 printf("最高評価: %c\n", GRADE\_A);  
 printf("割引率: %.1f%%\n", DISCOUNT \* 100);  
   
 /\* const変数を使った計算 \*/  
 {  
 int price = 1000;  
 double tax = price \* TAX\_RATE;  
 double discount\_amount = price \* DISCOUNT;  
 double final\_price = price + tax - discount\_amount;  
   
 printf("\n商品価格計算:\n");  
 printf(" 基本価格: %d円\n", price);  
 printf(" 税額: %.0f円\n", tax);  
 printf(" 割引額: %.0f円\n", discount\_amount);  
 printf(" 最終価格: %.0f円\n", final\_price);  
 }  
   
 /\* const変数は変更不可（コンパイルエラーになる） \*/  
 /\* MAX\_RETRY = 5; <- これはコンパイルエラー \*/  
}  
  
/\* マクロ定数のデモ \*/  
void macro\_constants\_demo(void)  
{  
 printf("\n=== マクロ定数のデモ ===\n");  
 printf("円周率: %.15f\n", PI);  
 printf("最大学生数: %d人\n", MAX\_STUDENTS);  
 printf("重力加速度: %.1f m/s²\n", GRAVITY);  
 printf("光速: %ld m/s\n", LIGHT\_SPEED);  
   
 /\* マクロ関数の使用 \*/  
 {  
 double radius = 5.0;  
 double area = CIRCLE\_AREA(radius);  
   
 printf("\n円の面積計算:\n");  
 printf(" 半径: %.1f\n", radius);  
 printf(" 面積: %.2f\n", area);  
 printf(" 計算式: π × %.1f² = %.15f × %.2f = %.2f\n",   
 radius, PI, SQUARE(radius), area);  
 }  
   
 /\* 温度変換マクロ \*/  
 {  
 double fahrenheit = 100.0;  
 double celsius = FAHRENHEIT\_TO\_CELSIUS(fahrenheit);  
   
 printf("\n温度変換:\n");  
 printf(" %.1f°F = %.1f°C\n", fahrenheit, celsius);  
 }  
}  
  
/\* 定数を使った計算のデモ \*/  
void calculation\_demo(void)  
{  
 /\* ローカルconst変数 \*/  
 const double SPHERE\_VOLUME\_COEFF = 4.0 / 3.0;  
 const int ARRAY\_SIZE = 5;  
   
 printf("\n=== 定数を使った計算デモ ===\n");  
   
 /\* 球の体積計算 \*/  
 {  
 double radius = 3.0;  
 double volume = SPHERE\_VOLUME\_COEFF \* PI \* SQUARE(radius) \* radius;  
   
 printf("球の体積計算:\n");  
 printf(" 半径: %.1f\n", radius);  
 printf(" 体積: %.2f\n", volume);  
 printf(" 計算式: (4/3) × π × %.1f³ = %.2f\n", radius, volume);  
 }  
   
 /\* 配列サイズ定数の使用 \*/  
 {  
 const int numbers[5] = {10, 20, 30, 40, 50}; /\* C90では配列サイズは定数が必要 \*/  
 int sum = 0;  
 int i;  
   
 printf("\n配列計算（サイズ: %d）:\n", ARRAY\_SIZE);  
   
 for (i = 0; i < ARRAY\_SIZE; i++) {  
 sum += numbers[i];  
 printf(" numbers[%d] = %d\n", i, numbers[i]);  
 }  
   
 printf(" 合計: %d\n", sum);  
 printf(" 平均: %.1f\n", (double)sum / ARRAY\_SIZE);  
 }  
}  
  
/\* 物理計算のデモ \*/  
void physics\_calculations(void)  
{  
 /\* 物理定数（const変数として定義） \*/  
 const double ELECTRON\_CHARGE = 1.602176634e-19; /\* クーロン \*/  
 const double PLANCK\_CONSTANT = 6.62607015e-34; /\* ジュール秒 \*/  
 const double AVOGADRO\_NUMBER = 6.02214076e23; /\* 個/mol \*/  
   
 printf("\n=== 物理計算デモ ===\n");  
   
 /\* 自由落下計算 \*/  
 {  
 double height = 100.0; /\* 高さ（メートル） \*/  
 double time = sqrt(2.0 \* height / GRAVITY);  
 double velocity = GRAVITY \* time;  
   
 printf("自由落下計算:\n");  
 printf(" 高さ: %.1f m\n", height);  
 printf(" 落下時間: %.2f 秒\n", time);  
 printf(" 着地速度: %.2f m/s\n", velocity);  
 printf(" 重力加速度: %.1f m/s²\n", GRAVITY);  
 }  
   
 /\* 円運動計算 \*/  
 {  
 double radius = 10.0; /\* 半径（メートル） \*/  
 double period = 5.0; /\* 周期（秒） \*/  
 double circumference = 2.0 \* PI \* radius;  
 double velocity = circumference / period;  
 double acceleration = SQUARE(velocity) / radius;  
   
 printf("\n円運動計算:\n");  
 printf(" 半径: %.1f m\n", radius);  
 printf(" 周期: %.1f 秒\n", period);  
 printf(" 円周: %.2f m\n", circumference);  
 printf(" 速度: %.2f m/s\n", velocity);  
 printf(" 向心加速度: %.2f m/s²\n", acceleration);  
 }  
   
 /\* 物理定数の表示 \*/  
 printf("\n物理定数:\n");  
 printf(" 電子電荷: %.3e C\n", ELECTRON\_CHARGE);  
 printf(" プランク定数: %.3e J·s\n", PLANCK\_CONSTANT);  
 printf(" アボガドロ数: %.3e 個/mol\n", AVOGADRO\_NUMBER);  
 printf(" 光速: %ld m/s\n", LIGHT\_SPEED);  
}  
  
/\*  
学習ポイント:  
1. #defineによるマクロ定数の定義  
2. const変数による定数の定義  
3. マクロ関数の作成と使用  
4. 物理計算での定数の活用  
5. 定数の命名規則（大文字、アンダースコア）  
  
#define vs const の違い:  
- #define: プリプロセッサによる文字列置換、型情報なし  
- const: 型付き定数、デバッガで値確認可能、スコープあり  
  
マクロ定数の利点:  
- コンパイル時に値が確定  
- 配列サイズとして使用可能（C90）  
- 条件コンパイルで使用可能  
  
const変数の利点:  
- 型安全性  
- デバッガでの確認が容易  
- スコープによる制御が可能  
  
注意点:  
- マクロ関数は引数を括弧で囲む  
- マクロは副作用に注意（SQUARE(++x)など）  
- 物理定数は国際単位系の値を使用  
\*/

### ex3\_7\_temperature\_converter.c

/\*  
 \* ファイル名: ex3\_7\_temperature\_converter.c  
 \* チャレンジ問題: 温度変換プログラム  
 \* 説明: 摂氏・華氏・ケルビンの温度単位を相互変換するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 温度変換に使用する定数を定義 \*/  
#define FAHRENHEIT\_FACTOR 9.0/5.0  
#define FAHRENHEIT\_OFFSET 32.0  
#define KELVIN\_OFFSET 273.15  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 double celsius;  
 double fahrenheit;  
 double kelvin;  
   
 printf("=== 温度変換プログラム ===\n");  
   
 /\* デモ用の温度値（実際のプログラムでは入力を受け取る） \*/  
 celsius = 25.0;  
   
 printf("入力温度: %.2f℃\n\n", celsius);  
   
 /\* 摂氏から華氏への変換 \*/  
 fahrenheit = celsius \* FAHRENHEIT\_FACTOR + FAHRENHEIT\_OFFSET;  
   
 /\* 摂氏からケルビンへの変換 \*/  
 kelvin = celsius + KELVIN\_OFFSET;  
   
 /\* 結果の表示 \*/  
 printf("=== 温度変換結果 ===\n");  
 printf("摂氏 : %.2f℃\n", celsius);  
 printf("華氏 : %.2f°F\n", fahrenheit);  
 printf("ケルビン: %.2fK\n", kelvin);  
   
 /\* 変換公式の説明 \*/  
 printf("\n=== 変換公式 ===\n");  
 printf("華氏 = 摂氏 × 9/5 + 32\n");  
 printf("ケルビン = 摂氏 + 273.15\n");  
   
 /\* 他の温度での変換例 \*/  
 printf("\n=== その他の変換例 ===\n");  
   
 /\* 水の凝固点 \*/  
 celsius = 0.0;  
 fahrenheit = celsius \* FAHRENHEIT\_FACTOR + FAHRENHEIT\_OFFSET;  
 kelvin = celsius + KELVIN\_OFFSET;  
 printf("水の凝固点: %.1f℃ = %.1f°F = %.1fK\n", celsius, fahrenheit, kelvin);  
   
 /\* 水の沸点 \*/  
 celsius = 100.0;  
 fahrenheit = celsius \* FAHRENHEIT\_FACTOR + FAHRENHEIT\_OFFSET;  
 kelvin = celsius + KELVIN\_OFFSET;  
 printf("水の沸点 : %.1f℃ = %.1f°F = %.1fK\n", celsius, fahrenheit, kelvin);  
   
 /\* 絶対零度 \*/  
 celsius = -273.15;  
 fahrenheit = celsius \* FAHRENHEIT\_FACTOR + FAHRENHEIT\_OFFSET;  
 kelvin = celsius + KELVIN\_OFFSET;  
 printf("絶対零度 : %.2f℃ = %.2f°F = %.2fK\n", celsius, fahrenheit, kelvin);  
   
 /\* 定数の使用例の説明 \*/  
 printf("\n=== 実装のポイント ===\n");  
 printf("- #defineマクロで変換係数を定義\n");  
 printf("- 浮動小数点演算の精度に注意\n");  
 printf("- 物理的に意味のある温度範囲を考慮\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. #defineマクロによる定数の定義  
2. 浮動小数点演算の精度管理  
3. 物理的な意味を持つ計算の実装  
4. 複数の単位系の相互変換  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- #defineで変換係数を定数として定義  
- 浮動小数点演算の精度を考慮  
- 物理的に重要な温度点での変換例を提示  
- デモ版として固定値を使用（実際は入力を受け取る）  
\*/

# 第4章 演算子

## 演習問題

## 基礎問題

### 演習4-1: 四則演算計算機

2つの整数を入力として受け取り、すべての算術演算（加算、減算、乗算、除算、剰余）の結果を表示するプログラムを作成してください。

**要件:** - ユーザーから2つの整数を入力 - 各演算の結果を見やすく表示 - ゼロ除算のチェックを含める

**期待される出力例:**

2つの整数を入力してください: 10 3  
10 + 3 = 13  
10 - 3 = 7  
10 \* 3 = 30  
10 / 3 = 3  
10 % 3 = 1

### 演習4-2: 比較と論理演算

2つの整数を入力として受け取り、関係演算子と論理演算子を使って最大値と最小値を求めるプログラムを作成してください。

**要件:** - 2つの整数を入力 - if文を使わず、条件演算子（三項演算子）のみで実装 - 最大値と最小値を表示

### 演習4-3: インクリメント・デクリメント

前置と後置のインクリメント・デクリメント演算子の違いを確認するプログラムを作成してください。

**要件:** - 変数に対して前置・後置の両方を使用 - 各操作後の値を表示 - 式の中での動作も確認

## 応用問題

### 演習4-4: ビット操作

整数を入力として受け取り、その数値の各ビットを表示し、特定のビット操作を実行するプログラムを作成してください。

**要件:** - 2進数表示 - 指定したビット位置のON/OFF確認 - 特定ビットの設定・クリア・トグル機能

### 演習4-5: 条件演算子の活用

3つの数値を入力として受け取り、条件演算子（三項演算子）のみを使って昇順に並び替えるプログラムを作成してください。

**要件:** - if文、switch文を使用しない - 条件演算子のネストを活用 - 並び替えた結果を表示

### 演習4-6: 演算子優先順位

複雑な式を含むプログラムを作成し、演算子の優先順位による計算結果の違いを確認してください。

**要件:** - 同じ数値で異なる式を評価 - 括弧の有無による結果の違いを表示 - 少なくとも3種類以上の演算子を使用

## 提出方法

1. 各問題に対して別々のCファイルを作成
2. ファイル名は ex4\_1\_calculator.c, ex4\_2\_comparison.c のように命名
3. 各プログラムの冒頭にコメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認

## コンパイル例

gcc -std=c90 -Wall -Wextra -pedantic ex4\_1\_calculator.c -o ex4\_1\_calculator  
./ex4\_1\_calculator

## ヒント

* 演算子の優先順位に注意
* ゼロ除算やオーバーフローに対する考慮
* 可読性を意識したコードを心がける
* 適切なコメントを追加する

## 解答例

このディレクトリには演習問題の解答例が含まれています。

## 解答ファイル一覧

### 基礎問題

1. **演習4-1: 四則演算計算機**
   * <ex4_1_calculator.c> - C90準拠版
   * <ex4_1_calculator_c99.c> - C99準拠版
   * 2つの整数に対する基本的な算術演算を実行
2. **演習4-2: 比較と論理演算**
   * <ex4_2_comparison.c> - C90準拠版
   * <ex4_2_comparison_c99.c> - C99準拠版
   * 条件演算子のみで最大値・最小値を求める
3. **演習4-3: インクリメント・デクリメント**
   * <ex4_3_increment.c> - C90準拠版
   * <ex4_3_increment_c99.c> - C99準拠版
   * 前置と後置の違いを詳細に確認

### 応用問題

1. **演習4-4: ビット操作**
   * <ex4_4_bitwise.c> - C90準拠版
   * <ex4_4_bitwise_c99.c> - C99準拠版
   * ビット演算子を使った各種操作の実装
2. **演習4-5: 条件演算子の活用**
   * <ex4_5_conditional.c> - C90準拠版
   * <ex4_5_conditional_c99.c> - C99準拠版
   * 3つの数値を条件演算子のみで昇順に並び替え
3. **演習4-6: 演算子優先順位**
   * <ex4_6_precedence.c> - C90準拠版
   * <ex4_6_precedence_c99.c> - C99準拠版
   * 複雑な式での演算子優先順位の確認

## 学習のポイント

### C90版とC99版の主な違い

1. **変数宣言位置**
   * C90: ブロックの先頭でのみ宣言可能
   * C99: 使用箇所で宣言可能
2. **bool型の使用**
   * C90: int型で真偽値を表現
   * C99: stdbool.hによるbool型の使用
3. **inline関数**
   * C90: 未対応
   * C99: inline関数による効率化
4. **コメントスタイル**
   * C90: /\* \*/ のみ
   * C99: // も使用可能
5. **固定幅整数型**
   * C90: 標準整数型のみ
   * C99: stdint.hによるuint8\_t等の使用

### 各演習の重要概念

#### 演習4-1: 四則演算計算機

* ゼロ除算のエラー処理
* 整数除算と実数除算の違い
* 代入演算子の使い方

#### 演習4-2: 比較と論理演算

* 関係演算子の評価結果（0または1）
* 論理演算子による条件の組み合わせ
* 条件演算子によるif文なしの条件分岐

#### 演習4-3: インクリメント・デクリメント

* 前置と後置の評価タイミングの違い
* 式の中での副作用
* 未定義動作の回避

#### 演習4-4: ビット操作

* ビットマスクの作成と使用
* 特定ビットの設定・クリア・反転
* ビットシフトによる乗除算

#### 演習4-5: 条件演算子の活用

* ネストした条件演算子の使い方
* 複数条件の組み合わせ
* 可読性とのバランス

#### 演習4-6: 演算子優先順位

* 演算子の優先順位と結合規則
* 括弧による評価順序の制御
* よくある間違いとその回避方法

## コンパイルと実行

# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic ex4\_1\_calculator.c -o ex4\_1\_calculator  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic ex4\_1\_calculator\_c99.c -o ex4\_1\_calculator\_c99  
  
# 実行  
./ex4\_1\_calculator

## 注意事項

* すべての解答例は教育目的で書かれており、実用的なコードよりも理解しやすさを重視しています
* エラー処理は基本的なものに留めており、実際のプロダクションコードではより堅牢な処理が必要です
* 各解答ファイルには詳細なコメントが含まれているので、コードと合わせて読むことを推奨します ### ex4\_1\_calculator.c

/\*  
 \* ファイル名: ex4\_1\_calculator.c  
 \* 演習4-1: 四則演算計算機  
 \* 説明: 2つの整数を入力として受け取り、基本的な算術演算の結果を表示  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int num1, num2;  
 int add\_result, sub\_result, mul\_result;  
   
 printf("===== 四則演算計算機 =====\n");  
 printf("2つの整数を入力してください: ");  
   
 /\* 入力の取得 \*/  
 if (scanf("%d %d", &num1, &num2) != 2) {  
 printf("エラー: 正しい整数を入力してください\n");  
 return 1;  
 }  
   
 /\* 基本的な算術演算 \*/  
 add\_result = num1 + num2;  
 sub\_result = num1 - num2;  
 mul\_result = num1 \* num2;  
   
 /\* 結果の表示 \*/  
 printf("\n=== 計算結果 ===\n");  
 printf("%d + %d = %d\n", num1, num2, add\_result);  
 printf("%d - %d = %d\n", num1, num2, sub\_result);  
 printf("%d \* %d = %d\n", num1, num2, mul\_result);  
   
 /\* 除算と剰余（ゼロ除算チェック付き） \*/  
 if (num2 != 0) {  
 printf("%d / %d = %d (整数除算)\n", num1, num2, num1 / num2);  
 printf("%d %% %d = %d (剰余)\n", num1, num2, num1 % num2);  
   
 /\* 実数除算も表示 \*/  
 printf("%d / %d = %.2f (実数除算)\n",   
 num1, num2, (double)num1 / num2);  
 } else {  
 printf("%d / %d = エラー (ゼロ除算)\n", num1, num2);  
 printf("%d %% %d = エラー (ゼロ除算)\n", num1, num2);  
 }  
   
 /\* 追加情報の表示 \*/  
 printf("\n=== 追加情報 ===\n");  
 printf("入力値の絶対値: |%d| = %d, |%d| = %d\n",   
 num1, (num1 < 0) ? -num1 : num1,  
 num2, (num2 < 0) ? -num2 : num2);  
   
 /\* 代入演算子のデモンストレーション \*/  
 printf("\n=== 代入演算子のデモ ===\n");  
 {  
 int temp = num1;  
 printf("初期値: temp = %d\n", temp);  
   
 temp += num2;  
 printf("temp += %d -> temp = %d\n", num2, temp);  
   
 temp -= num2;  
 printf("temp -= %d -> temp = %d\n", num2, temp);  
   
 if (num2 != 0) {  
 temp \*= num2;  
 printf("temp \*= %d -> temp = %d\n", num2, temp);  
   
 temp /= num2;  
 printf("temp /= %d -> temp = %d\n", num2, temp);  
   
 temp %= num2;  
 printf("temp %%= %d -> temp = %d\n", num2, temp);  
 }  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_1\_calculator  
===== 四則演算計算機 =====  
2つの整数を入力してください: 10 3  
  
=== 計算結果 ===  
10 + 3 = 13  
10 - 3 = 7  
10 \* 3 = 30  
10 / 3 = 3 (整数除算)  
10 % 3 = 1 (剰余)  
10 / 3 = 3.33 (実数除算)  
  
=== 追加情報 ===  
入力値の絶対値: |10| = 10, |3| = 3  
  
=== 代入演算子のデモ ===  
初期値: temp = 10  
temp += 3 -> temp = 13  
temp -= 3 -> temp = 10  
temp \*= 3 -> temp = 30  
temp /= 3 -> temp = 10  
temp %= 3 -> temp = 1  
  
学習ポイント:  
1. 基本的な算術演算子（+, -, \*, /, %）の使用  
2. ゼロ除算のエラーチェック  
3. 整数除算と実数除算の違い  
4. 代入演算子（+=, -=, \*=, /=, %=）の動作  
5. 条件演算子による絶対値の計算  
\*/```  
  
### ex4\_2\_comparison.c  
  
```c  
/\*  
 \* ファイル名: ex4\_2\_comparison.c  
 \* 演習4-2: 比較と論理演算  
 \* 説明: 関係演算子と論理演算子を使って最大値と最小値を求める  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int num1, num2;  
 int max, min;  
 int comparison\_result;  
   
 printf("===== 比較と論理演算 =====\n");  
 printf("2つの整数を入力してください: ");  
   
 /\* 入力の取得 \*/  
 if (scanf("%d %d", &num1, &num2) != 2) {  
 printf("エラー: 正しい整数を入力してください\n");  
 return 1;  
 }  
   
 /\* 条件演算子（三項演算子）のみで最大値と最小値を求める \*/  
 max = (num1 > num2) ? num1 : num2;  
 min = (num1 < num2) ? num1 : num2;  
   
 printf("\n=== 結果 ===\n");  
 printf("入力値: %d, %d\n", num1, num2);  
 printf("最大値: %d\n", max);  
 printf("最小値: %d\n", min);  
   
 /\* 関係演算子の詳細な比較 \*/  
 printf("\n=== 関係演算子による比較 ===\n");  
 printf("%d == %d : %d\n", num1, num2, num1 == num2);  
 printf("%d != %d : %d\n", num1, num2, num1 != num2);  
 printf("%d < %d : %d\n", num1, num2, num1 < num2);  
 printf("%d <= %d : %d\n", num1, num2, num1 <= num2);  
 printf("%d > %d : %d\n", num1, num2, num1 > num2);  
 printf("%d >= %d : %d\n", num1, num2, num1 >= num2);  
   
 /\* 論理演算子の使用例 \*/  
 printf("\n=== 論理演算子の応用 ===\n");  
   
 /\* 両方の数が正の数かチェック \*/  
 comparison\_result = (num1 > 0) && (num2 > 0);  
 printf("両方とも正の数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 少なくとも一方が正の数かチェック \*/  
 comparison\_result = (num1 > 0) || (num2 > 0);  
 printf("少なくとも一方が正の数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 両方の数が偶数かチェック \*/  
 comparison\_result = !(num1 % 2) && !(num2 % 2);  
 printf("両方とも偶数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 条件演算子のネストによる3つの値の比較（拡張例） \*/  
 printf("\n=== 条件演算子の応用 ===\n");  
 {  
 int third\_value = 0;  
 int max\_of\_three;  
 int min\_of\_three;  
   
 printf("3つ目の値として0を使用します\n");  
   
 /\* 3つの値の最大値（ネストした条件演算子） \*/  
 max\_of\_three = (num1 > num2) ?   
 ((num1 > third\_value) ? num1 : third\_value) :  
 ((num2 > third\_value) ? num2 : third\_value);  
   
 /\* 3つの値の最小値（ネストした条件演算子） \*/  
 min\_of\_three = (num1 < num2) ?   
 ((num1 < third\_value) ? num1 : third\_value) :  
 ((num2 < third\_value) ? num2 : third\_value);  
   
 printf("3つの値 (%d, %d, %d) の最大値: %d\n",   
 num1, num2, third\_value, max\_of\_three);  
 printf("3つの値 (%d, %d, %d) の最小値: %d\n",   
 num1, num2, third\_value, min\_of\_three);  
 }  
   
 /\* 値の範囲判定（条件演算子のみ使用） \*/  
 printf("\n=== 範囲判定 ===\n");  
 {  
 const char \*range1 = (num1 >= -10 && num1 <= 10) ?   
 "範囲内（-10〜10）" : "範囲外";  
 const char \*range2 = (num2 >= -10 && num2 <= 10) ?   
 "範囲内（-10〜10）" : "範囲外";  
   
 printf("第1の値（%d）: %s\n", num1, range1);  
 printf("第2の値（%d）: %s\n", num2, range2);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_2\_comparison  
===== 比較と論理演算 =====  
2つの整数を入力してください: 7 12  
  
=== 結果 ===  
入力値: 7, 12  
最大値: 12  
最小値: 7  
  
=== 関係演算子による比較 ===  
7 == 12 : 0  
7 != 12 : 1  
7 < 12 : 1  
7 <= 12 : 1  
7 > 12 : 0  
7 >= 12 : 0  
  
=== 論理演算子の応用 ===  
両方とも正の数: はい  
少なくとも一方が正の数: はい  
両方とも偶数: いいえ  
  
=== 条件演算子の応用 ===  
3つ目の値として0を使用します  
3つの値 (7, 12, 0) の最大値: 12  
3つの値 (7, 12, 0) の最小値: 0  
  
=== 範囲判定 ===  
第1の値（7）: 範囲内（-10〜10）  
第2の値（12）: 範囲外  
  
学習ポイント:  
1. 条件演算子（三項演算子）による最大値・最小値の判定  
2. 関係演算子（==, !=, <, <=, >, >=）の使用  
3. 論理演算子（&&, ||, !）の使用  
4. 条件演算子のネストによる複雑な条件判定  
5. if文を使わない条件分岐の実装  
\*/```  
  
### ex4\_3\_increment.c  
  
```c  
/\*  
 \* ファイル名: ex4\_3\_increment.c  
 \* 演習4-3: インクリメントとデクリメント演算  
 \* 説明: 前置と後置のインクリメント・デクリメント演算子の違いを確認  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int a, b, result;  
 int array[5] = {10, 20, 30, 40, 50};  
 int i;  
   
 printf("===== インクリメント・デクリメント演算子 =====\n");  
   
 /\* 基本的な動作確認 \*/  
 printf("\n=== 基本的な動作 ===\n");  
   
 a = 5;  
 printf("初期値: a = %d\n", a);  
   
 result = a++; /\* 後置インクリメント \*/  
 printf("result = a++ → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = ++a; /\* 前置インクリメント \*/  
 printf("result = ++a → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = a--; /\* 後置デクリメント \*/  
 printf("result = a-- → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = --a; /\* 前置デクリメント \*/  
 printf("result = --a → result = %d, a = %d\n", result, a);  
   
 /\* 式の中での動作確認 \*/  
 printf("\n=== 式の中での動作 ===\n");  
   
 a = 5;  
 b = 3;  
 result = a++ \* b;  
 printf("a = 5, b = 3\n");  
 printf("result = a++ \* b → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 b = 3;  
 result = ++a \* b;  
 printf("a = 5, b = 3\n");  
 printf("result = ++a \* b → result = %d, a = %d\n", result, a);  
   
 /\* 複雑な式での使用 \*/  
 printf("\n=== 複雑な式での使用 ===\n");  
   
 a = 10;  
 b = 5;  
 result = a++ + ++b;  
 printf("a = 10, b = 5\n");  
 printf("result = a++ + ++b → result = %d, a = %d, b = %d\n",   
 result, a, b);  
   
 a = 10;  
 b = 5;  
 result = --a - b--;  
 printf("a = 10, b = 5\n");  
 printf("result = --a - b-- → result = %d, a = %d, b = %d\n",   
 result, a, b);  
   
 /\* 配列とポインタでの使用 \*/  
 printf("\n=== 配列での使用 ===\n");  
 printf("配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n");  
   
 i = 0;  
 int temp\_val = array[i++];  
 printf("array[i++] = %d, i = %d\n", temp\_val, i);  
   
 i = 0;  
 temp\_val = array[++i];  
 printf("array[++i] = %d, i = %d\n", temp\_val, i);  
   
 /\* 実用的な例：ループでの使用 \*/  
 printf("\n=== ループでの実用例 ===\n");  
   
 printf("後置インクリメント（i++）:\n");  
 for (i = 0; i < 5; i++) {  
 printf("i = %d, array[%d] = %d\n", i, i, array[i]);  
 }  
   
 printf("\n前置インクリメント（++i）:\n");  
 for (i = 0; i < 5; ++i) {  
 printf("i = %d, array[%d] = %d\n", i, i, array[i]);  
 }  
   
 /\* 注意事項：未定義動作の例（コメントアウト） \*/  
 printf("\n=== 注意事項 ===\n");  
 printf("以下のような式は未定義動作となるので避けるべきです：\n");  
 printf(" i = i++; （同じ変数への複数の副作用）\n");  
 printf(" a[i] = i++; （評価順序が不定）\n");  
 printf(" func(i++, i++); （引数の評価順序が不定）\n");  
   
 /\* ポインタでの使用例 \*/  
 printf("\n=== ポインタでの使用例 ===\n");  
 {  
 int \*ptr = array;  
   
 printf("\*ptr++ = %d （値を取得してからポインタを進める）\n", \*ptr++);  
 printf("現在のptr位置の値: %d\n", \*ptr);  
   
 ptr = array; /\* リセット \*/  
 printf("\*++ptr = %d （ポインタを進めてから値を取得）\n", \*++ptr);  
 printf("現在のptr位置の値: %d\n", \*ptr);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_3\_increment  
===== インクリメント・デクリメント演算子 =====  
  
=== 基本的な動作 ===  
初期値: a = 5  
result = a++ → result = 5, a = 6  
result = ++a → result = 6, a = 6  
result = a-- → result = 5, a = 4  
result = --a → result = 4, a = 4  
  
=== 式の中での動作 ===  
a = 5, b = 3  
result = a++ \* b → result = 15, a = 6  
a = 5, b = 3  
result = ++a \* b → result = 18, a = 6  
  
=== 複雑な式での使用 ===  
a = 10, b = 5  
result = a++ + ++b → result = 16, a = 11, b = 6  
a = 10, b = 5  
result = --a - b-- → result = 4, a = 9, b = 4  
  
=== 配列での使用 ===  
配列: 10 20 30 40 50  
array[i++] = 10, i = 1  
array[++i] = 20, i = 1  
  
=== ループでの実用例 ===  
後置インクリメント（i++）:  
i = 0, array[0] = 10  
i = 1, array[1] = 20  
i = 2, array[2] = 30  
i = 3, array[3] = 40  
i = 4, array[4] = 50  
  
前置インクリメント（++i）:  
i = 0, array[0] = 10  
i = 1, array[1] = 20  
i = 2, array[2] = 30  
i = 3, array[3] = 40  
i = 4, array[4] = 50  
  
=== 注意事項 ===  
以下のような式は未定義動作となるので避けるべきです：  
 i = i++; （同じ変数への複数の副作用）  
 a[i] = i++; （評価順序が不定）  
 func(i++, i++); （引数の評価順序が不定）  
  
=== ポインタでの使用例 ===  
\*ptr++ = 10 （値を取得してからポインタを進める）  
現在のptr位置の値: 20  
\*++ptr = 20 （ポインタを進めてから値を取得）  
現在のptr位置の値: 20  
  
学習ポイント:  
1. 前置と後置の違い（値の返却タイミング）  
2. 式の中での評価順序  
3. 配列やポインタでの実用的な使い方  
4. 未定義動作を避けるための注意点  
5. ループでの効率的な使用方法  
\*/```  
  
### ex4\_4\_bitwise.c  
  
```c  
/\*  
 \* ファイル名: ex4\_4\_bitwise.c  
 \* 演習4-4: ビット演算  
 \* 説明: ビット演算子を使って数値の特定ビットを操作  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\* ビットを表示する関数 \*/  
void print\_bits(unsigned int value, int bits)  
{  
 int i;  
 printf("0b");  
 for (i = bits - 1; i >= 0; i--) {  
 printf("%d", (value >> i) & 1);  
 if (i % 4 == 0 && i > 0) {  
 printf("\_"); /\* 4ビットごとに区切り \*/  
 }  
 }  
}  
  
int main(void)  
{  
 unsigned int num;  
 int bit\_position;  
 unsigned int result;  
 unsigned int mask;  
   
 printf("===== ビット操作 =====\n");  
 printf("整数を入力してください (0-255): ");  
   
 if (scanf("%u", &num) != 1 || num > 255) {  
 printf("エラー: 0から255の範囲で入力してください\n");  
 return 1;  
 }  
   
 printf("\n元の値: %u = ", num);  
 print\_bits(num, 8);  
 printf("\n\n");  
   
 /\* ビット単位の基本演算 \*/  
 printf("=== ビット単位の基本演算 ===\n");  
   
 /\* AND演算 \*/  
 result = num & 0xF0; /\* 上位4ビットを取得 \*/  
 printf("AND演算 (上位4ビット取得):\n");  
 printf("%u & 0xF0 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* OR演算 \*/  
 result = num | 0x0F; /\* 下位4ビットを1にセット \*/  
 printf("OR演算 (下位4ビットセット):\n");  
 printf("%u | 0x0F = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* XOR演算 \*/  
 result = num ^ 0xFF; /\* 全ビット反転 \*/  
 printf("XOR演算 (全ビット反転):\n");  
 printf("%u ^ 0xFF = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* NOT演算 \*/  
 result = ~num & 0xFF; /\* 8ビットのみ表示 \*/  
 printf("NOT演算 (ビット反転):\n");  
 printf("~%u = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* ビットシフト演算 \*/  
 printf("=== ビットシフト演算 ===\n");  
   
 /\* 左シフト \*/  
 result = num << 1;  
 printf("左シフト (1ビット):\n");  
 printf("%u << 1 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf(" (2倍)\n\n");  
   
 /\* 右シフト \*/  
 result = num >> 1;  
 printf("右シフト (1ビット):\n");  
 printf("%u >> 1 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf(" (1/2)\n\n");  
   
 /\* 特定ビットの操作 \*/  
 printf("=== 特定ビットの操作 ===\n");  
 printf("操作するビット位置を入力 (0-7): ");  
   
 if (scanf("%d", &bit\_position) != 1 || bit\_position < 0 || bit\_position > 7) {  
 printf("エラー: 0から7の範囲で入力してください\n");  
 return 1;  
 }  
   
 /\* ビットマスクの作成 \*/  
 mask = 1u << bit\_position;  
   
 /\* ビットのテスト \*/  
 printf("\nビット%dのテスト:\n", bit\_position);  
 if (num & mask) {  
 printf("ビット%dは1です\n", bit\_position);  
 } else {  
 printf("ビット%dは0です\n", bit\_position);  
 }  
   
 /\* ビットのセット \*/  
 result = num | mask;  
 printf("\nビット%dをセット:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* ビットのクリア \*/  
 result = num & ~mask;  
 printf("\nビット%dをクリア:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* ビットの反転 \*/  
 result = num ^ mask;  
 printf("\nビット%dを反転:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* 実用的な例 \*/  
 printf("\n=== 実用的なビット操作例 ===\n");  
   
 /\* 偶数・奇数の判定 \*/  
 printf("偶数・奇数判定: ");  
 if (num & 1) {  
 printf("%uは奇数です (最下位ビットが1)\n", num);  
 } else {  
 printf("%uは偶数です (最下位ビットが0)\n", num);  
 }  
   
 /\* 2のべき乗の判定 \*/  
 if (num > 0 && (num & (num - 1)) == 0) {  
 printf("%uは2のべき乗です\n", num);  
 } else {  
 printf("%uは2のべき乗ではありません\n", num);  
 }  
   
 /\* ビットカウント（立っているビットの数） \*/  
 {  
 unsigned int temp = num;  
 int count = 0;  
 while (temp) {  
 count += temp & 1;  
 temp >>= 1;  
 }  
 printf("立っているビットの数: %d個\n", count);  
 }  
   
 /\* 上位ビットと下位ビットの交換 \*/  
 {  
 unsigned int upper = (num & 0xF0) >> 4;  
 unsigned int lower = (num & 0x0F) << 4;  
 result = upper | lower;  
 printf("\n上位4ビットと下位4ビットの交換:\n");  
 printf("元: ");  
 print\_bits(num, 8);  
 printf("\n結果: ");  
 print\_bits(result, 8);  
 printf(" (%u)\n", result);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_4\_bitwise  
===== ビット操作 =====  
整数を入力してください (0-255): 170  
  
元の値: 170 = 0b1010\_1010  
  
=== ビット単位の基本演算 ===  
AND演算 (上位4ビット取得):  
170 & 0xF0 = 160 = 0b1010\_0000  
  
OR演算 (下位4ビットセット):  
170 | 0x0F = 175 = 0b1010\_1111  
  
XOR演算 (全ビット反転):  
170 ^ 0xFF = 85 = 0b0101\_0101  
  
NOT演算 (ビット反転):  
~170 = 85 = 0b0101\_0101  
  
=== ビットシフト演算 ===  
左シフト (1ビット):  
170 << 1 = 340 = 0b0101\_0100 (2倍)  
  
右シフト (1ビット):  
170 >> 1 = 85 = 0b0101\_0101 (1/2)  
  
=== 特定ビットの操作 ===  
操作するビット位置を入力 (0-7): 3  
  
ビット3のテスト:  
ビット3は1です  
  
ビット3をセット:  
結果: 170 = 0b1010\_1010  
  
ビット3をクリア:  
結果: 162 = 0b1010\_0010  
  
ビット3を反転:  
結果: 162 = 0b1010\_0010  
  
=== 実用的なビット操作例 ===  
偶数・奇数判定: 170は偶数です (最下位ビットが0)  
170は2のべき乗ではありません  
立っているビットの数: 4個  
  
上位4ビットと下位4ビットの交換:  
元: 0b1010\_1010  
結果: 0b1010\_1010 (170)  
  
学習ポイント:  
1. ビット演算子（&, |, ^, ~, <<, >>）の使い方  
2. ビットマスクによる特定ビットの操作  
3. ビット演算を使った効率的な処理  
4. 2進数表示によるビット操作の可視化  
5. 実用的なビット操作テクニック  
\*/```  
  
### ex4\_5\_conditional.c  
  
```c  
/\*  
 \* ファイル名: ex4\_5\_conditional.c  
 \* 演習4-5: 条件演算子  
 \* 説明: 3つの数値を条件演算子のみで昇順に並び替える  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
  
/\* 時間表示用の構造体 \*/  
typedef struct  
{  
 int hour;  
 int minute;  
} Time;  
  
/\* ユーティリティ関数 \*/  
int max(int a, int b);  
int min(int a, int b);  
int sign(int x);  
double absolute(double x);  
char get\_grade(int score);  
const char \*get\_season(int month);  
  
int main(void)  
{  
 /\* C90準拠のために全ての変数を最初に宣言 \*/  
 int number;  
 int score;  
 char grade;  
 int values[] = {23, 89, 45, 12, 67, 91, 34};  
 int size;  
 int array\_max;  
 int array\_min;  
 int range\_min, range\_max;  
 int in\_range\_count;  
 Time current\_time;  
 int hour\_12;  
 const char \*ampm;  
 int age;  
 char gender;  
 int is\_student;  
 int discount\_rate;  
 const char \*discount\_reason;  
 int x, y;  
 char text[] = "Hello, World!";  
 int months[] = {3, 6, 9, 12};  
 int indices[] = {-1, 0, 3, 5, 7, 10};  
 int array\_size;  
 int safe\_array[5] = {10, 20, 30, 40, 50};  
 int i;  
 char ch;  
 const char \*type;  
 int idx;  
 int safe\_idx;  
 int safe\_value;  
  
 printf("=== 条件演算子の応用 ===\n");  
  
 /\* 基本的な使用 \*/  
 printf("基本的な使用:\n");  
 number = 15;  
 printf(" 数値: %d\n", number);  
 printf(" 絶対値: %d (%s)\n",  
 (number >= 0) ? number : -number,  
 (number >= 0) ? "正数" : "負数");  
 printf(" 符号: %c\n", (number > 0) ? '+' : (number < 0) ? '-'  
 : '0');  
 printf(" 偶数/奇数: %s\n", (number % 2 == 0) ? "偶数" : "奇数");  
 printf("\n");  
  
 /\* ネストした条件演算子 \*/  
 printf("ネストした条件演算子:\n");  
 score = 85;  
 grade = (score >= 90) ? 'A' : (score >= 80) ? 'B'  
 : (score >= 70) ? 'C'  
 : (score >= 60) ? 'D'  
 : 'F';  
 printf(" 点数: %d\n", score);  
 printf(" 評価: %c (90以上:A, 80以上:B, 70以上:C, 60以上:D, その他:F)\n", grade);  
 printf("\n");  
  
 /\* 配列操作での使用 \*/  
 printf("配列操作での使用:\n");  
 size = sizeof(values) / sizeof(values[0]);  
  
 /\* 最大値・最小値の検索 \*/  
 array\_max = values[0];  
 array\_min = values[0];  
 for (i = 1; i < size; i++)  
 {  
 array\_max = (values[i] > array\_max) ? values[i] : array\_max;  
 array\_min = (values[i] < array\_min) ? values[i] : array\_min;  
 }  
 printf(" 配列の最大値: %d\n", array\_max);  
 printf(" 配列の最小値: %d\n", array\_min);  
  
 /\* 範囲内の要素数 \*/  
 range\_min = 30;  
 range\_max = 80;  
 in\_range\_count = 0;  
 for (i = 0; i < size; i++)  
 {  
 in\_range\_count += (values[i] >= range\_min && values[i] <= range\_max) ? 1 : 0;  
 }  
 printf(" 範囲[%d-%d]内の要素数: %d個\n", range\_min, range\_max, in\_range\_count);  
 printf("\n");  
  
 /\* 時間表示での使用 \*/  
 printf("時間表示での使用:\n");  
 current\_time.hour = 14;  
 current\_time.minute = 30;  
 printf(" 現在時刻: %d時%d分\n", current\_time.hour, current\_time.minute);  
  
 /\* 12時間表記への変換 \*/  
 hour\_12 = (current\_time.hour == 0) ? 12 : (current\_time.hour > 12) ? current\_time.hour - 12  
 : current\_time.hour;  
 ampm = (current\_time.hour < 12) ? "午前" : "午後";  
 printf(" 12時間表記: %s%d時%d分\n", ampm, hour\_12, current\_time.minute);  
 printf(" AM/PM: %s %d:%02d\n",  
 (current\_time.hour < 12) ? "AM" : "PM",  
 hour\_12, current\_time.minute);  
 printf("\n");  
  
 /\* 複雑な条件 \*/  
 printf("複雑な条件:\n");  
 age = 20;  
 gender = 'M';  
 is\_student = 1;  
  
 printf(" 年齢: %d歳, 性別: %c, 学生: %s\n",  
 age, gender, is\_student ? "はい" : "いいえ");  
  
 /\* 割引率の計算 \*/  
 discount\_rate = (is\_student && age < 25) ? 20 : (age >= 65) ? 15  
 : (age < 18) ? 10  
 : 0;  
 discount\_reason = (is\_student && age < 25) ? "学生割引" : (age >= 65) ? "シニア割引"  
 : (age < 18) ? "子供割引"  
 : "割引なし";  
 printf(" 割引率: %d%% (%s)\n", discount\_rate, discount\_reason);  
 printf("\n");  
  
 /\* 計算での使用 \*/  
 printf("計算での使用:\n");  
 x = 5;  
 y = 3;  
 printf(" x=%d, y=%d\n", x, y);  
 printf(" max(x,y) = %d\n", max(x, y));  
 printf(" min(x,y) = %d\n", min(x, y));  
 printf(" sign(x-y) = %+d\n", sign(x - y));  
 printf("\n");  
  
 /\* 文字列処理での使用 \*/  
 printf("文字列処理での使用:\n");  
 printf(" 文字列: \"%s\"\n", text);  
 printf(" 文字数: %d\n", (int)strlen(text));  
  
 /\* 文字の分類 \*/  
 printf(" 文字分類:\n");  
 for (i = 0; text[i] != '\0'; i++)  
 {  
 ch = text[i];  
 type = (ch >= 'A' && ch <= 'Z') ? "大文字" : (ch >= 'a' && ch <= 'z') ? "小文字"  
 : (ch >= '0' && ch <= '9') ? "数字"  
 : (ch == ' ') ? "空白"  
 : "記号";  
 printf(" '%c': %s\n", ch, type);  
 }  
 printf("\n");  
  
 /\* 季節判定 \*/  
 printf("季節判定:\n");  
 for (i = 0; i < 4; i++)  
 {  
 printf(" %d月: %s\n", months[i], get\_season(months[i]));  
 }  
 printf("\n");  
  
 /\* 配列の範囲制限 \*/  
 printf("配列の範囲制限:\n");  
 array\_size = 5;  
  
 for (i = 0; i < 6; i++)  
 {  
 idx = indices[i];  
 safe\_idx = (idx < 0) ? 0 : (idx >= array\_size) ? array\_size - 1  
 : idx;  
 safe\_value = safe\_array[safe\_idx];  
 printf(" インデックス%d: %s (値:%d)\n",  
 idx,  
 (idx >= 0 && idx < array\_size) ? "範囲内" : "範囲外→制限",  
 safe\_value);  
 }  
  
 printf("==================\n");  
  
 return 0;  
}  
  
/\* ユーティリティ関数の実装 \*/  
int max(int a, int b)  
{  
 return (a > b) ? a : b;  
}  
  
int min(int a, int b)  
{  
 return (a < b) ? a : b;  
}  
  
int sign(int x)  
{  
 return (x > 0) ? 1 : (x < 0) ? -1  
 : 0;  
}  
  
double absolute(double x)  
{  
 return (x >= 0.0) ? x : -x;  
}  
  
char get\_grade(int score)  
{  
 return (score >= 90) ? 'A' : (score >= 80) ? 'B'  
 : (score >= 70) ? 'C'  
 : (score >= 60) ? 'D'  
 : 'F';  
}  
  
const char \*get\_season(int month)  
{  
 return (month >= 3 && month <= 5) ? "春" : (month >= 6 && month <= 8) ? "夏"  
 : (month >= 9 && month <= 11) ? "秋"  
 : "冬";  
}  
  
/\*  
学習ポイント:  
1. 条件演算子の基本構文  
 - condition ? expression1 : expression2  
 - 三項演算子とも呼ばれる  
  
2. ネストした条件演算子  
 - 複数の条件を組み合わせ可能  
 - 可読性のため適度に使用  
 - if-else文との使い分け  
  
3. 実用的な応用  
 - 最大値・最小値の選択  
 - 範囲制限（クランプ）  
 - デフォルト値の設定  
 - 文字列の選択  
  
4. パフォーマンス考慮  
 - 式として評価されるため効率的  
 - 関数呼び出しより高速  
 - コンパイラ最適化が効きやすい  
  
5. 使用上の注意点  
 - 複雑な条件は分割する  
 - 副作用のある式は避ける  
 - 型の一致に注意  
 - 可読性を優先  
  
6. 適用場面  
 - 簡単な値の選択  
 - 初期化での条件分岐  
 - マクロ内での条件処理  
 - 関数の引数での条件  
\*/

### ex4\_6\_precedence.c

/\*  
 \* ファイル名: ex4\_6\_precedence.c  
 \* 演習4-6: 演算子の優先順位  
 \* 説明: 複雑な式を含むプログラムで演算子の優先順位を確認  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 複雑な式の評価を段階的に表示する関数 \*/  
void demonstrate\_evaluation\_order(void);  
void demonstrate\_associativity(void);  
void demonstrate\_common\_mistakes(void);  
  
int main(void)  
{  
 printf("=== 演算子の優先順位 ===\n");  
  
 /\* 基本的な優先順位 \*/  
 printf("基本的な優先順位:\n");  
 int result1 = 2 + 3 \* 4;  
 int result2 = (2 + 3) \* 4;  
 printf(" 2 + 3 \* 4 = %d (3\*4が先に計算される)\n", result1);  
 printf(" (2 + 3) \* 4 = %d (括弧で優先順位変更)\n", result2);  
 printf("\n");  
  
 /\* 除算の優先順位 \*/  
 printf("除算の優先順位:\n");  
 int result3 = 10 / 2 \* 3;  
 int result4 = 10 / (2 \* 3);  
 printf(" 10 / 2 \* 3 = %d (左から右へ: (10/2)\*3)\n", result3);  
 printf(" 10 / (2 \* 3) = %d (括弧で変更)\n", result4);  
 printf("\n");  
  
 /\* 比較と論理 \*/  
 printf("比較と論理:\n");  
 int a = 5, b = 3, c = 4;  
 int result5 = a > b && b < c;  
 int result6 = a > b + 1;  
 printf(" %d > %d && %d < %d = %d (比較が先、論理が後)\n", a, b, b, c, result5);  
 printf(" %d > %d + 1 = %d (加算が先: %d > %d)\n", a, b, result6, a, b + 1);  
 printf("\n");  
  
 /\* 代入の右結合 \*/  
 printf("代入の右結合:\n");  
 int x, y, z;  
 x = y = z = 5;  
 printf(" x = y = z = 5\n");  
 printf(" 結果: x=%d, y=%d, z=%d\n", x, y, z);  
 printf("\n");  
  
 /\* ビット演算の優先順位 \*/  
 printf("ビット演算の優先順位:\n");  
 int result7 = 6 & 3 | 4;  
 int result8 = 6 & (3 | 4);  
 printf(" 6 & 3 | 4 = %d (&が先: (6&3)|4 = %d|4)\n", result7, 6 & 3);  
 printf(" 6 & (3 | 4) = %d (括弧で変更: 6&%d)\n", result8, 3 | 4);  
 printf("\n");  
  
 /\* より詳細なデモ \*/  
 demonstrate\_evaluation\_order();  
 demonstrate\_associativity();  
 demonstrate\_common\_mistakes();  
  
 /\* 実用的な例 \*/  
 printf("実用的な優先順位の例:\n");  
  
 /\* 条件式での優先順位 \*/  
 int score = 85;  
 int is\_passing = score >= 60 && score <= 100;  
 printf(" 点数判定: score=%d\n", score);  
 printf(" score >= 60 && score <= 100 = %d\n", is\_passing);  
  
 /\* 計算式での優先順位 \*/  
 double radius = 5.0;  
 double area = 3.14 \* radius \* radius; /\* 左から右へ評価 \*/  
 double circumference = 2 \* 3.14 \* radius;  
 printf(" 円の面積 (r=%.1f): %.2f\n", radius, area);  
 printf(" 円の周長: %.2f\n", circumference);  
  
 /\* 配列インデックスでの優先順位 \*/  
 int array[] = {10, 20, 30, 40, 50};  
 int index = 2;  
 int value = array[index++]; /\* []が先、then ++ \*/  
 printf(" 配列アクセス: array[%d] = %d, その後index = %d\n", index - 1, value, index);  
  
 printf("=======================\n");  
  
 return 0;  
}  
  
/\* 複雑な式の評価順序デモ \*/  
void demonstrate\_evaluation\_order(void)  
{  
 printf("複雑な式の評価順序:\n");  
  
 /\* 段階的に複雑にしていく \*/  
 int p = 2, q = 3, r = 4, s = 5;  
 printf(" 初期値: p=%d, q=%d, r=%d, s=%d\n", p, q, r, s);  
  
 /\* 式1: 算術演算の組み合わせ \*/  
 int expr1 = p + q \* r - s;  
 printf(" p + q \* r - s:\n");  
 printf(" = %d + %d \* %d - %d\n", p, q, r, s);  
 printf(" = %d + %d - %d (乗算先)\n", p, q \* r, s);  
 printf(" = %d - %d (加算次)\n", p + q \* r, s);  
 printf(" = %d (減算最後)\n", expr1);  
 printf("\n");  
  
 /\* 式2: 比較と論理の組み合わせ \*/  
 int expr2 = p < q && r > s || p == q;  
 printf(" p < q && r > s || p == q:\n");  
 printf(" = %d < %d && %d > %d || %d == %d\n", p, q, r, s, p, q);  
 printf(" = %d && %d || %d (比較先)\n", p<q, r> s, p == q);  
 printf(" = %d || %d (&&次)\n", (p < q) && (r > s), p == q);  
 printf(" = %d (||最後)\n", expr2);  
 printf("\n");  
  
 /\* 式3: 混合演算 \*/  
 int expr3 = p++ + ++q \* r-- > s && q != r;  
 printf(" 複雑な混合演算の結果: %d\n", expr3);  
 printf(" 演算後の値: p=%d, q=%d, r=%d, s=%d\n", p, q, r, s);  
 printf("\n");  
}  
  
/\* 結合性のデモ \*/  
void demonstrate\_associativity(void)  
{  
 printf("結合性のデモ:\n");  
  
 /\* 左結合の例 \*/  
 printf("左結合の例:\n");  
 int result1 = 100 / 5 / 2; /\* (100 / 5) / 2 \*/  
 int result2 = 100 / (5 / 2); /\* 100 / (5 / 2) \*/  
 printf(" 100 / 5 / 2 = %d (左結合: (100/5)/2)\n", result1);  
 printf(" 100 / (5 / 2) = %d (括弧で右結合)\n", result2);  
  
 int result3 = 20 - 10 - 5; /\* (20 - 10) - 5 \*/  
 int result4 = 20 - (10 - 5); /\* 20 - (10 - 5) \*/  
 printf(" 20 - 10 - 5 = %d (左結合: (20-10)-5)\n", result3);  
 printf(" 20 - (10 - 5) = %d (括弧で右結合)\n", result4);  
 printf("\n");  
  
 /\* 右結合の例 \*/  
 printf("右結合の例:\n");  
 int a, b, c;  
 a = b = c = 10; /\* a = (b = (c = 10)) \*/  
 printf(" a = b = c = 10 → a=%d, b=%d, c=%d (右結合)\n", a, b, c);  
  
 /\* 条件演算子の右結合 \*/  
 int x = 5;  
 int result5 = x > 0 ? x > 10 ? 10 : x : 0;  
 printf(" 条件演算子: x > 0 ? x > 10 ? 10 : x : 0\n");  
 printf(" x=%d の場合: %d (右結合で評価)\n", x, result5);  
 printf("\n");  
}  
  
/\* よくある間違いの例 \*/  
void demonstrate\_common\_mistakes(void)  
{  
 printf("よくある間違いの例:\n");  
  
 /\* 間違い1: ビット演算と比較の優先順位 \*/  
 printf("間違い1: ビット演算の優先順位\n");  
 int flags = 5; /\* 0101 \*/  
 int mask = 1; /\* 0001 \*/  
  
 /\* 間違った書き方 \*/  
 int wrong = flags & mask == 1; /\* flags & (mask == 1) \*/  
 int correct = (flags & mask) == 1; /\* 正しい \*/  
  
 printf(" flags=%d, mask=%d\n", flags, mask);  
 printf(" flags & mask == 1 = %d (間違い: flags & (mask == 1))\n", wrong);  
 printf(" (flags & mask) == 1 = %d (正しい)\n", correct);  
 printf("\n");  
  
 /\* 間違い2: シフトと算術演算 \*/  
 printf("間違い2: シフトと算術演算\n");  
 int val = 4;  
 int wrong2 = val << 2 + 1; /\* val << (2 + 1) \*/  
 int correct2 = (val << 2) + 1; /\* 正しい \*/  
  
 printf(" val=%d\n", val);  
 printf(" val << 2 + 1 = %d (間違い: val << (2 + 1))\n", wrong2);  
 printf(" (val << 2) + 1 = %d (正しい)\n", correct2);  
 printf("\n");  
  
 /\* 間違い3: ポインタと配列の優先順位 \*/  
 printf("間違い3: ポインタ演算\n");  
 int array[] = {10, 20, 30, 40};  
 int \*ptr = array;  
  
 printf(" \*ptr++ = %d (ポインタを進めてから前の値)\n", \*ptr++);  
 printf(" 現在のptr: %d\n", \*ptr);  
  
 ptr = array; /\* リセット \*/  
 printf(" (\*ptr)++ = %d (値をインクリメント)\n", (\*ptr)++);  
 printf(" array[0] = %d (配列の値が変更された)\n", array[0]);  
  
 array[0] = 10; /\* 復元 \*/  
 printf("\n");  
  
 /\* 間違い4: 条件演算子の優先順位 \*/  
 printf("間違い4: 条件演算子\n");  
 int score = 85;  
 char grade1 = score >= 90 ? 'A' : score >= 80 ? 'B'  
 : 'C';  
 char grade2 = (score >= 90) ? 'A' : ((score >= 80) ? 'B' : 'C');  
  
 printf(" score=%d\n", score);  
 printf(" 評価1: %c (右結合で正しく動作)\n", grade1);  
 printf(" 評価2: %c (括弧で明確化)\n", grade2);  
 printf("\n");  
}  
  
/\*  
学習ポイント:  
1. 演算子の優先順位  
 - 算術 > 関係 > 論理 > 代入  
 - 乗除 > 加減  
 - ビット演算は関係演算より低い  
  
2. 結合性  
 - 左結合: 算術、関係、論理演算  
 - 右結合: 代入、条件演算子  
  
3. よくある間違い  
 - ビット演算と比較の混同  
 - シフトと算術演算の優先順位  
 - ポインタ演算の理解不足  
  
4. 対策  
 - 複雑な式では括弧を使う  
 - 一行に複数の演算子を書かない  
 - 可読性を優先する  
  
5. 実用的なガイドライン  
 - 疑問があれば括弧で明確にする  
 - コンパイラ警告を確認する  
 - 式を分割して可読性を向上させる  
  
重要な優先順位:  
1. 後置演算子（++, --）  
2. 単項演算子（前置++, --, !, ~, \*, &）  
3. 乗除（\*, /, %）  
4. 加減（+, -）  
5. シフト（<<, >>）  
6. 関係（<, <=, >, >=）  
7. 等価（==, !=）  
8. ビット演算（&, ^, |）  
9. 論理演算（&&, ||）  
10. 条件演算子（?:）  
11. 代入（=, +=, -=, ...）  
\*/

# 第5章 制御構造（条件分岐）

## 演習問題

## 演習の目的

* if文、switch文の理解を深める
* 条件演算子の適切な使い方を習得する
* 複雑な条件分岐の設計力を養う

## 演習問題

### 演習4-1: 年齢による料金計算

映画館の料金システムを実装してください。

**要件:** - 3歳未満: 無料 - 3歳以上12歳以下: 子供料金（800円） - 13歳以上18歳以下: 学生料金（1200円） - 19歳以上65歳以下: 大人料金（1800円） - 66歳以上: シニア料金（1200円）

**実装のヒント:** - if-else if文を使用 - 境界値に注意

### 演習4-2: 電卓プログラム

四則演算ができる簡単な電卓を作成してください。

**要件:** - ユーザーから2つの数値と演算子（+, -, \*, /）を入力 - switch文を使って演算を実行 - ゼロ除算のエラー処理を含める

**実装のヒント:** - 演算子は char 型で受け取る - 除算の場合は分母が0でないかチェック

### 演習4-3: BMI判定プログラム

身長と体重からBMIを計算し、判定するプログラムを作成してください。

**要件:** - BMI = 体重(kg) / (身長(m) × 身長(m)) - BMI判定基準: - 18.5未満: 低体重 - 18.5以上25未満: 標準体重 - 25以上30未満: 肥満度1 - 30以上: 肥満度2

**実装のヒント:** - 身長はセンチメートルで入力してメートルに変換 - 浮動小数点数を使用

### 演習4-4: うるう年判定

西暦年を入力してうるう年かどうか判定するプログラムを作成してください。

**うるう年の条件:** 1. 4で割り切れる年はうるう年 2. ただし、100で割り切れる年は平年 3. ただし、400で割り切れる年はうるう年

**実装のヒント:** - 複数の条件を論理演算子で組み合わせる - 条件の順序に注意

### 演習4-5: 成績評価システム

複数の科目の点数から総合評価を出すプログラムを作成してください。

**要件:** - 3科目（国語、数学、英語）の点数を入力 - 平均点を計算 - 平均点による評価（A〜F） - 全科目60点以上の場合のみ「合格」、それ以外は「不合格」 - 条件演算子を使って簡潔に表示

**実装のヒント:** - 平均点の計算は整数演算に注意 - 複数の条件を組み合わせる

## チャレンジ問題

### チャレンジ4-6: じゃんけんゲーム

コンピュータとじゃんけんをするプログラムを作成してください。

**要件:** - ユーザーの手を数値で入力（1:グー、2:チョキ、3:パー） - コンピュータの手はランダムに決定 - 勝敗を判定して表示 - 不正な入力のチェック

**追加要件:** - 3回勝負で最終的な勝者を決定 - 各1回の結果を記録して最後に表示

## 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 ex4\_1.c, ex4\_2.c のような形式で
3. コメントで問題番号と簡単な説明を記載
4. コンパイル・実行確認を必ず実行う

## 期限

演習問題は次の章に進む前に完了させましょう。

## 解答例

## 解答例一覧

### 高度な解答例

#### 成績計算システム

* **ファイル**: grade\_calculator.c, grade\_calculator\_c99.c
* **学習内容**: if-else if連鎖、switch文、複合条件判定
* **ポイント**: 実用的な評価システムの実装
* **機能**: テスト点数、出席率、宿題提出状況による総合評価

## 演習問題の解答

### 演習4-1: 年齢による料金計算

* **ファイル**: ex4\_1\_ticket\_pricing.c, ex4\_1\_ticket\_pricing\_c99.c
* **学習内容**: if-else if文による段階的判定
* **ポイント**: 境界値の適切な処理

### 演習4-2: 電卓プログラム

* **ファイル**: ex4\_2\_calculator.c, ex4\_2\_calculator\_c99.c
* **学習内容**: switch文による多分岐処理
* **ポイント**: ゼロ除算エラーの処理

### 演習4-3: BMI判定プログラム

* **ファイル**: ex4\_3\_bmi\_judge.c, ex4\_3\_bmi\_judge\_c99.c
* **学習内容**: 浮動小数点数の計算と範囲判定
* **ポイント**: 単位変換と健康アドバイス

### 演習4-4: うるう年判定

* **ファイル**: ex4\_4\_leap\_year.c, ex4\_4\_leap\_year\_c99.c
* **学習内容**: 複雑な論理演算子の組み合わせ
* **ポイント**: 複合条件式の正確な実装

### 演習4-5: 成績評価システム

* **ファイル**: ex4\_5\_grade\_system.c, ex4\_5\_grade\_system\_c99.c
* **学習内容**: 条件演算子と複合条件の活用
* **ポイント**: 複数データの統合的な評価

### チャレンジ4-6: じゃんけんゲーム

* **ファイル**: ex4\_6\_janken\_game.c, ex4\_6\_janken\_game\_c99.c
* **学習内容**: ランダム数生成と複雑な勝敗判定
* **ポイント**: ゲームロジックの実装と統計分析

## 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
* /\* \*/ 形式のコメントを使用
* 従来のC言語の制御構造に準拠

### 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
* // 形式のコメントに対応
* より現代的な制御構造の書き方

## コンパイルと実行

### 基本的なコンパイル

gcc -Wall -Wextra -pedantic -std=c90 ex4\_1\_ticket\_pricing.c -o ex4\_1\_ticket\_pricing  
./ex4\_1\_ticket\_pricing

### C99版のコンパイル

gcc -Wall -Wextra -pedantic -std=c99 ex4\_1\_ticket\_pricing\_c99.c -o ex4\_1\_ticket\_pricing\_c99  
./ex4\_1\_ticket\_pricing\_c99

### Makefileを使用

make all # 全てをコンパイル  
make solutions # 解答例のみコンパイル  
make ex4\_1 # 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

### 条件分岐の基本

1. **if文**: 単純な条件判定
2. **if-else文**: 2分岐の処理
3. **if-else if文**: 多段階の条件判定
4. **switch文**: 値による多分岐処理
5. **条件演算子**: 簡潔な条件分岐

### 実用的なテクニック

1. **入力検証**: ユーザー入力の安全性確保
2. **エラーハンドリング**: 適切なエラー処理
3. **境界値処理**: 範囲指定の正確な実装
4. **複合条件**: 論理演算子の効果的な使用
5. **可読性**: 理解しやすいコードの書き方

### 設計パターン

1. **段階的判定**: if-else if連鎖による階層的処理
2. **値による分岐**: switch文による効率的な分岐
3. **状態管理**: フラグ変数による状態の管理
4. **統合的判定**: 複数条件による総合的な評価

## 注意事項

* 条件式では等価演算子（==）と代入演算子（=）を混同しない
* switch文ではbreak文を忘れずに記述
* 浮動小数点数の比較では誤差を考慮
* 論理演算子の優先順位に注意

## 実用的な応用

これらの制御構造は以下のような実用的なプログラムで使用されます：

* **Webアプリケーション**: ユーザー認証、フォーム検証
* **ゲーム**: プレイヤーの行動による分岐処理
* **データ分析**: 条件による分類・集計処理
* **システム管理**: 設定による動作の制御

## 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5章: 制御構造（繰り返し）](../control-loop/) - [第6章: 配列](../arrays/) - [第7章: 文字列](../strings/) - [第8章: 関数](../functions/) ### ex4\_1\_ticket\_pricing.c

/\*  
 \* ファイル名: ex4\_1\_ticket\_pricing.c  
 \* 演習4-1: 年齢による料金計算  
 \* 説明: 映画館の料金システムを実装するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int age;  
 int price;  
 char \*category;  
   
 printf("=== 映画館料金計算システム ===\n");  
 printf("年齢を入力してください: ");  
 scanf("%d", &age);  
   
 /\* 入力値の検証 \*/  
 if (age < 0) {  
 printf("エラー: 年齢は0以上の値を入力してください。\n");  
 return 1;  
 }  
   
 if (age > 120) {  
 printf("エラー: 年齢は120歳以下の値を入力してください。\n");  
 return 1;  
 }  
   
 /\* 年齢による料金分類 \*/  
 if (age < 3) {  
 price = 0;  
 category = "幼児";  
 } else if (age >= 3 && age <= 12) {  
 price = 800;  
 category = "子供";  
 } else if (age >= 13 && age <= 18) {  
 price = 1200;  
 category = "学生";  
 } else if (age >= 19 && age <= 65) {  
 price = 1800;  
 category = "大人";  
 } else {  
 price = 1200;  
 category = "シニア";  
 }  
   
 /\* 結果の表示 \*/  
 printf("\n=== 料金計算結果 ===\n");  
 printf("年齢: %d歳\n", age);  
 printf("区分: %s\n", category);  
   
 if (price == 0) {  
 printf("料金: 無料\n");  
 } else {  
 printf("料金: %d円\n", price);  
 }  
   
 /\* 特別なメッセージ \*/  
 if (age < 3) {  
 printf("お子様は保護者の同伴が必要です。\n");  
 } else if (age >= 66) {  
 printf("シニア割引が適用されました。\n");  
 } else if (age >= 13 && age <= 18) {  
 printf("学生証の提示をお願いします。\n");  
 }  
   
 printf("\nご利用ありがとうございます！\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. if-else if文による段階的な条件判定  
2. 境界値の処理（<=, >=の使い分け）  
3. 入力値の検証とエラーハンドリング  
4. 論理演算子（&&）による範囲指定  
5. 変数の適切な初期化と使用  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 境界値（3歳、12歳、18歳、65歳）を正確に処理  
- ユーザーフレンドリーなエラーメッセージ  
- 明確な結果表示  
\*/```  
  
### ex4\_2\_calculator.c  
  
```c  
/\*  
 \* ファイル名: ex4\_2\_calculator.c  
 \* 演習4-2: 電卓プログラム  
 \* 説明: 四則演算ができる簡単な電卓プログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 double num1, num2, result;  
 char operator;  
 int valid\_operator;  
   
 printf("=== 簡単電卓プログラム ===\n");  
   
 /\* 第一の数値入力 \*/  
 printf("最初の数値を入力してください: ");  
 scanf("%lf", &num1);  
   
 /\* 演算子の入力 \*/  
 printf("演算子を入力してください (+, -, \*, /): ");  
 scanf(" %c", &operator); /\* 空白文字を読み飛ばすため、%cの前に空白 \*/  
   
 /\* 第二の数値入力 \*/  
 printf("二番目の数値を入力してください: ");  
 scanf("%lf", &num2);  
   
 /\* 演算子の妥当性チェック \*/  
 valid\_operator = 1;  
   
 /\* switch文による演算の実行 \*/  
 switch (operator) {  
 case '+':  
 result = num1 + num2;  
 printf("%.2f + %.2f = %.2f\n", num1, num2, result);  
 break;  
   
 case '-':  
 result = num1 - num2;  
 printf("%.2f - %.2f = %.2f\n", num1, num2, result);  
 break;  
   
 case '\*':  
 result = num1 \* num2;  
 printf("%.2f \* %.2f = %.2f\n", num1, num2, result);  
 break;  
   
 case '/':  
 /\* ゼロ除算チェック \*/  
 if (num2 == 0.0) {  
 printf("エラー: ゼロで割ることはできません。\n");  
 return 1;  
 } else {  
 result = num1 / num2;  
 printf("%.2f / %.2f = %.2f\n", num1, num2, result);  
 }  
 break;  
   
 default:  
 printf("エラー: 無効な演算子です。(+, -, \*, /) のいずれかを使用してください。\n");  
 valid\_operator = 0;  
 break;  
 }  
   
 /\* 演算が成功した場合の追加情報 \*/  
 if (valid\_operator && operator != '/') {  
 printf("\n=== 計算完了 ===\n");  
   
 /\* 結果の分析 \*/  
 if (result > 0) {  
 printf("結果は正の数です。\n");  
 } else if (result < 0) {  
 printf("結果は負の数です。\n");  
 } else {  
 printf("結果はゼロです。\n");  
 }  
   
 /\* 整数かどうかのチェック \*/  
 if (result == (int)result) {  
 printf("結果は整数です。\n");  
 } else {  
 printf("結果は小数です。\n");  
 }  
 } else if (valid\_operator && operator == '/') {  
 printf("\n=== 除算完了 ===\n");  
   
 /\* 除算の特別な処理 \*/  
 if (result == 1.0) {  
 printf("割る数と割られる数が同じです。\n");  
 } else if (result > 1.0) {  
 printf("割られる数の方が大きいです。\n");  
 } else {  
 printf("割る数の方が大きいです。\n");  
 }  
 }  
   
 printf("電卓プログラムを終了します。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. switch文による多分岐処理  
2. ゼロ除算のエラーハンドリング  
3. 浮動小数点数の比較  
4. break文の重要性  
5. default句によるエラー処理  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- scanf()での文字入力時の注意点（空白文字の処理）  
- 浮動小数点数の比較（== 0.0の使用）  
- switch文での各caseの適切な処理  
- ユーザーフレンドリーなエラーメッセージ  
\*/```  
  
### ex4\_3\_bmi\_judge.c  
  
```c  
/\*  
 \* ファイル名: ex4\_3\_bmi\_judge.c  
 \* 演習4-3: BMI判定プログラム  
 \* 説明: 身長と体重からBMIを計算し体型判定を行う  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 double height\_cm, height\_m, weight, bmi;  
 char \*category;  
 char \*advice;  
   
 printf("=== BMI判定プログラム ===\n");  
   
 /\* 身長の入力（センチメートル） \*/  
 printf("身長を入力してください（cm）: ");  
 scanf("%lf", &height\_cm);  
   
 /\* 体重の入力 \*/  
 printf("体重を入力してください（kg）: ");  
 scanf("%lf", &weight);  
   
 /\* 入力値の検証 \*/  
 if (height\_cm <= 0 || height\_cm > 300) {  
 printf("エラー: 身長は1cm〜300cmの範囲で入力してください。\n");  
 return 1;  
 }  
   
 if (weight <= 0 || weight > 500) {  
 printf("エラー: 体重は1kg〜500kgの範囲で入力してください。\n");  
 return 1;  
 }  
   
 /\* 身長をメートルに変換 \*/  
 height\_m = height\_cm / 100.0;  
   
 /\* BMI計算 \*/  
 bmi = weight / (height\_m \* height\_m);  
   
 /\* BMI判定 \*/  
 if (bmi < 18.5) {  
 category = "低体重";  
 advice = "栄養バランスの良い食事を心がけ、適度な運動で健康的に体重を増やしましょう。";  
 } else if (bmi >= 18.5 && bmi < 25.0) {  
 category = "標準体重";  
 advice = "理想的な体重です。現在の生活習慣を維持しましょう。";  
 } else if (bmi >= 25.0 && bmi < 30.0) {  
 category = "肥満度1";  
 advice = "軽度の肥満です。食事制限と有酸素運動を始めることをお勧めします。";  
 } else {  
 category = "肥満度2";  
 advice = "高度な肥満です。医師に相談し、本格的な減量プログラムを検討してください。";  
 }  
   
 /\* 結果の表示 \*/  
 printf("\n=== BMI計算結果 ===\n");  
 printf("身長: %.1f cm (%.2f m)\n", height\_cm, height\_m);  
 printf("体重: %.1f kg\n", weight);  
 printf("BMI: %.2f\n", bmi);  
 printf("判定: %s\n", category);  
 printf("\nアドバイス: %s\n", advice);  
   
 /\* 追加の健康情報 \*/  
 printf("\n=== 健康情報 ===\n");  
   
 if (bmi < 18.5) {  
 printf("・BMI 18.5未満は痩せすぎの可能性があります\n");  
 printf("・免疫力低下や骨密度減少のリスクがあります\n");  
 } else if (bmi >= 18.5 && bmi < 25.0) {  
 printf("・BMI 18.5-24.9は標準的な範囲です\n");  
 printf("・生活習慣病のリスクが最も低い範囲です\n");  
 } else if (bmi >= 25.0 && bmi < 30.0) {  
 printf("・BMI 25.0-29.9は軽度肥満の範囲です\n");  
 printf("・糖尿病や高血圧のリスクが上昇します\n");  
 } else {  
 printf("・BMI 30.0以上は高度肥満の範囲です\n");  
 printf("・心血管疾患のリスクが大幅に増加します\n");  
 }  
   
 /\* 理想体重の計算と表示 \*/  
 if (bmi < 18.5 || bmi >= 25.0) {  
 double ideal\_weight\_min = 18.5 \* (height\_m \* height\_m);  
 double ideal\_weight\_max = 24.9 \* (height\_m \* height\_m);  
   
 printf("\n=== 理想体重の目安 ===\n");  
 printf("あなたの理想体重: %.1f kg 〜 %.1f kg\n",   
 ideal\_weight\_min, ideal\_weight\_max);  
   
 if (weight < ideal\_weight\_min) {  
 printf("目標: %.1f kg の増量\n", ideal\_weight\_min - weight);  
 } else if (weight > ideal\_weight\_max) {  
 printf("目標: %.1f kg の減量\n", weight - ideal\_weight\_max);  
 }  
 }  
   
 printf("\n※この判定は一般的な目安です。詳しくは医師にご相談ください。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 浮動小数点数を使った計算  
2. if-else if文による範囲判定  
3. 単位変換（cm→m）  
4. 論理演算子による範囲指定  
5. 計算結果を使った追加処理  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- BMI計算式の正確な実装  
- 境界値（18.5, 25.0, 30.0）の適切な処理  
- ユーザーフレンドリーな結果表示  
- 入力値の妥当性チェック  
\*/```  
  
### ex4\_4\_leap\_year.c  
  
```c  
/\*  
 \* ファイル名: ex4\_4\_leap\_year.c  
 \* 演習4-4: うるう年判定  
 \* 説明: 西暦年を入力してうるう年かどうか判定するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int year;  
 int is\_leap\_year;  
   
 printf("=== うるう年判定プログラム ===\n");  
 printf("西暦年を入力してください: ");  
 scanf("%d", &year);  
   
 /\* 入力値の検証 \*/  
 if (year < 1) {  
 printf("エラー: 西暦1年以降を入力してください。\n");  
 return 1;  
 }  
   
 if (year > 3000) {  
 printf("エラー: 西暦3000年以下を入力してください。\n");  
 return 1;  
 }  
   
 /\* うるう年判定のロジック \*/  
 /\* 1. 4で割り切れる年はうるう年 \*/  
 /\* 2. ただし、100で割り切れる年は平年 \*/  
 /\* 3. ただし、400で割り切れる年はうるう年 \*/  
   
 if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {  
 is\_leap\_year = 1;  
 } else {  
 is\_leap\_year = 0;  
 }  
   
 /\* 結果の表示 \*/  
 printf("\n=== 判定結果 ===\n");  
 printf("入力年: %d年\n", year);  
   
 if (is\_leap\_year) {  
 printf("判定: うるう年です\n");  
 printf("2月は29日まであります。\n");  
 } else {  
 printf("判定: 平年です\n");  
 printf("2月は28日まであります。\n");  
 }  
   
 /\* 詳細な判定理由の表示 \*/  
 printf("\n=== 判定理由 ===\n");  
   
 if (year % 4 != 0) {  
 printf("4で割り切れないため、平年です。\n");  
 } else if (year % 100 != 0) {  
 printf("4で割り切れ、かつ100で割り切れないため、うるう年です。\n");  
 } else if (year % 400 != 0) {  
 printf("100で割り切れ、かつ400で割り切れないため、平年です。\n");  
 } else {  
 printf("400で割り切れるため、うるう年です。\n");  
 }  
   
 /\* 歴史的な情報 \*/  
 printf("\n=== 豆知識 ===\n");  
   
 if (year == 2000) {  
 printf("2000年は特別なうるう年でした（400で割り切れる）。\n");  
 } else if (year == 1900) {  
 printf("1900年は100で割り切れるが400で割り切れないため平年でした。\n");  
 } else if (year == 2100) {  
 printf("2100年も100で割り切れるが400で割り切れないため平年になります。\n");  
 }  
   
 /\* グレゴリオ暦の説明 \*/  
 if (year >= 1582) {  
 printf("グレゴリオ暦（現在の暦）での判定です。\n");  
 } else {  
 printf("注意: 1582年以前はユリウス暦が使われていました。\n");  
 }  
   
 /\* 次のうるう年/平年の計算 \*/  
 printf("\n=== 次のうるう年 ===\n");  
   
 if (is\_leap\_year) {  
 /\* 現在がうるう年の場合、次のうるう年を探す \*/  
 int next\_year;  
 for (next\_year = year + 1; next\_year <= year + 8; next\_year++) {  
 if ((next\_year % 4 == 0 && next\_year % 100 != 0) || (next\_year % 400 == 0)) {  
 printf("次のうるう年: %d年\n", next\_year);  
 break;  
 }  
 }  
 } else {  
 /\* 現在が平年の場合、次のうるう年を探す \*/  
 int next\_year;  
 for (next\_year = year + 1; next\_year <= year + 4; next\_year++) {  
 if ((next\_year % 4 == 0 && next\_year % 100 != 0) || (next\_year % 400 == 0)) {  
 printf("次のうるう年: %d年\n", next\_year);  
 break;  
 }  
 }  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 複雑な論理演算子の組み合わせ  
2. 剰余演算子（%）による割り切れる判定  
3. 条件の優先順位と論理構造  
4. 複合条件式の読みやすい書き方  
5. ループを使った条件検索  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- うるう年判定の複雑なルールを正確に実装  
- 論理演算子（&&, ||）の適切な使用  
- 条件式を段階的に説明  
- 歴史的背景も含めた教育的な内容  
\*/```  
  
### ex4\_5\_grade\_system.c  
  
```c  
/\*  
 \* ファイル名: ex4\_5\_grade\_system.c  
 \* 演習4-5: 成績評価システム  
 \* 説明: 3科目の点数から平均点を計算し総合評価を行う  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int japanese, math, english;  
 int total;  
 double average;  
 char grade;  
 char \*evaluation;  
 int all\_passed;  
 char \*result;  
   
 printf("=== 成績評価システム ===\n");  
   
 /\* 各科目の点数入力 \*/  
 printf("国語の点数を入力してください (0-100): ");  
 scanf("%d", &japanese);  
   
 printf("数学の点数を入力してください (0-100): ");  
 scanf("%d", &math);  
   
 printf("英語の点数を入力してください (0-100): ");  
 scanf("%d", &english);  
   
 /\* 入力値の検証 \*/  
 if (japanese < 0 || japanese > 100) {  
 printf("エラー: 国語の点数は0-100の範囲で入力してください。\n");  
 return 1;  
 }  
   
 if (math < 0 || math > 100) {  
 printf("エラー: 数学の点数は0-100の範囲で入力してください。\n");  
 return 1;  
 }  
   
 if (english < 0 || english > 100) {  
 printf("エラー: 英語の点数は0-100の範囲で入力してください。\n");  
 return 1;  
 }  
   
 /\* 合計点と平均点の計算 \*/  
 total = japanese + math + english;  
 average = (double)total / 3.0; /\* 整数演算を避けるため型キャスト \*/  
   
 /\* 平均点による評価 \*/  
 if (average >= 90) {  
 grade = 'A';  
 evaluation = "優秀";  
 } else if (average >= 80) {  
 grade = 'B';  
 evaluation = "良好";  
 } else if (average >= 70) {  
 grade = 'C';  
 evaluation = "普通";  
 } else if (average >= 60) {  
 grade = 'D';  
 evaluation = "可";  
 } else {  
 grade = 'F';  
 evaluation = "不可";  
 }  
   
 /\* 全科目60点以上の確認 \*/  
 all\_passed = (japanese >= 60 && math >= 60 && english >= 60) ? 1 : 0;  
   
 /\* 条件演算子を使った簡潔な表示 \*/  
 result = all\_passed ? "合格" : "不合格";  
   
 /\* 結果の表示 \*/  
 printf("\n=== 成績評価結果 ===\n");  
 printf("国語: %d点\n", japanese);  
 printf("数学: %d点\n", math);  
 printf("英語: %d点\n", english);  
 printf("--------------------\n");  
 printf("合計点: %d点\n", total);  
 printf("平均点: %.2f点\n", average);  
 printf("評価: %c (%s)\n", grade, evaluation);  
 printf("判定: %s\n", result);  
   
 /\* 詳細な分析 \*/  
 printf("\n=== 詳細分析 ===\n");  
   
 /\* 各科目の個別評価 \*/  
 printf("科目別評価:\n");  
 printf("・国語: %s\n", (japanese >= 60) ? "合格" : "不合格");  
 printf("・数学: %s\n", (math >= 60) ? "合格" : "不合格");  
 printf("・英語: %s\n", (english >= 60) ? "合格" : "不合格");  
   
 /\* 最高点と最低点の科目 \*/  
 if (japanese >= math && japanese >= english) {  
 printf("最高得点科目: 国語 (%d点)\n", japanese);  
 } else if (math >= japanese && math >= english) {  
 printf("最高得点科目: 数学 (%d点)\n", math);  
 } else {  
 printf("最高得点科目: 英語 (%d点)\n", english);  
 }  
   
 if (japanese <= math && japanese <= english) {  
 printf("最低得点科目: 国語 (%d点)\n", japanese);  
 } else if (math <= japanese && math <= english) {  
 printf("最低得点科目: 数学 (%d点)\n", math);  
 } else {  
 printf("最低得点科目: 英語 (%d点)\n", english);  
 }  
   
 /\* 改善アドバイス \*/  
 printf("\n=== 改善アドバイス ===\n");  
   
 if (all\_passed) {  
 printf("全科目合格おめでとうございます！\n");  
 if (average >= 90) {  
 printf("優秀な成績です。この調子で頑張ってください。\n");  
 } else if (average >= 80) {  
 printf("良い成績です。さらなる向上を目指しましょう。\n");  
 } else {  
 printf("全科目クリアしました。より高い点数を目指しましょう。\n");  
 }  
 } else {  
 printf("不合格科目があります。以下の科目の復習をしましょう:\n");  
 if (japanese < 60) {  
 printf("・国語: %d点 (不足 %d点)\n", japanese, 60 - japanese);  
 }  
 if (math < 60) {  
 printf("・数学: %d点 (不足 %d点)\n", math, 60 - math);  
 }  
 if (english < 60) {  
 printf("・英語: %d点 (不足 %d点)\n", english, 60 - english);  
 }  
 }  
   
 /\* 次回の目標設定 \*/  
 printf("\n=== 次回の目標 ===\n");  
 if (grade == 'A') {  
 printf("目標: A評価の維持\n");  
 } else if (grade == 'B') {  
 printf("目標: A評価（平均90点以上）\n");  
 } else if (grade == 'C') {  
 printf("目標: B評価（平均80点以上）\n");  
 } else if (grade == 'D') {  
 printf("目標: C評価（平均70点以上）\n");  
 } else {  
 printf("目標: まずは全科目60点以上\n");  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 複数の変数を使った計算処理  
2. if-else if文による段階的評価  
3. 条件演算子（?:）の実用的な使用  
4. 論理演算子（&&）による複合条件  
5. 型キャスト（int→double）による精密な計算  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 整数演算を避けるための型キャスト  
- 条件演算子による簡潔な条件分岐  
- 複数条件の組み合わせによる判定  
- ユーザーフレンドリーな結果表示  
\*/```  
  
### ex4\_6\_janken\_game.c  
  
```c  
/\*  
 \* ファイル名: ex4\_6\_janken\_game.c  
 \* チャレンジ4-6: じゃんけんゲーム  
 \* 説明: コンピュータとじゃんけんをする3回勝負のプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int user\_choice, computer\_choice;  
 int user\_wins, computer\_wins, draws;  
 int round;  
 char \*hands[3] = {"グー", "チョキ", "パー"};  
 char \*result\_message;  
   
 /\* 勝利数の初期化 \*/  
 user\_wins = 0;  
 computer\_wins = 0;  
 draws = 0;  
   
 /\* 乱数の初期化 \*/  
 srand((unsigned int)time(NULL));  
   
 printf("=== じゃんけんゲーム（3回勝負） ===\n");  
 printf("1: グー, 2: チョキ, 3: パー\n");  
 printf("さあ、始めましょう！\n\n");  
   
 /\* 3回勝負のループ \*/  
 for (round = 1; round <= 3; round++) {  
 printf("--- 第%d回戦 ---\n", round);  
   
 /\* ユーザーの手の入力 \*/  
 do {  
 printf("あなたの手を選んでください (1-3): ");  
 scanf("%d", &user\_choice);  
   
 if (user\_choice < 1 || user\_choice > 3) {  
 printf("エラー: 1〜3の数字を入力してください。\n");  
 }  
 } while (user\_choice < 1 || user\_choice > 3);  
   
 /\* コンピュータの手をランダムに決定 \*/  
 computer\_choice = rand() % 3 + 1;  
   
 /\* 手の表示 \*/  
 printf("あなた: %s\n", hands[user\_choice - 1]);  
 printf("コンピュータ: %s\n", hands[computer\_choice - 1]);  
   
 /\* 勝敗判定 \*/  
 if (user\_choice == computer\_choice) {  
 printf("結果: あいこ\n");  
 result\_message = "あいこ";  
 draws++;  
 } else if ((user\_choice == 1 && computer\_choice == 2) || /\* グー vs チョキ \*/  
 (user\_choice == 2 && computer\_choice == 3) || /\* チョキ vs パー \*/  
 (user\_choice == 3 && computer\_choice == 1)) { /\* パー vs グー \*/  
 printf("結果: あなたの勝ち！\n");  
 result\_message = "勝利";  
 user\_wins++;  
 } else {  
 printf("結果: コンピュータの勝ち！\n");  
 result\_message = "敗北";  
 computer\_wins++;  
 }  
   
 /\* 現在の戦績表示 \*/  
 printf("現在の戦績 - あなた: %d勝, コンピュータ: %d勝, あいこ: %d回\n",   
 user\_wins, computer\_wins, draws);  
 printf("\n");  
 }  
   
 /\* 最終結果の表示 \*/  
 printf("=== 最終結果 ===\n");  
 printf("3回戦の結果:\n");  
 printf("あなた: %d勝\n", user\_wins);  
 printf("コンピュータ: %d勝\n", computer\_wins);  
 printf("あいこ: %d回\n", draws);  
 printf("-----------------\n");  
   
 /\* 総合勝者の判定 \*/  
 if (user\_wins > computer\_wins) {  
 printf("総合結果: あなたの勝利！\n");  
 printf("おめでとうございます！素晴らしい戦いでした。\n");  
   
 if (user\_wins == 3) {  
 printf("完全勝利です！パーフェクトゲーム達成！\n");  
 } else if (user\_wins == 2 && draws == 1) {  
 printf("2勝1分けの安定した勝利です。\n");  
 }  
 } else if (computer\_wins > user\_wins) {  
 printf("総合結果: コンピュータの勝利！\n");  
 printf("残念！次回はきっと勝てますよ。\n");  
   
 if (computer\_wins == 3) {  
 printf("完敗でした。次回に期待しましょう！\n");  
 }  
 } else {  
 printf("総合結果: 引き分け！\n");  
 printf("互角の戦いでした。良いゲームでしたね。\n");  
 }  
   
 /\* 詳細な分析 \*/  
 printf("\n=== ゲーム分析 ===\n");  
   
 /\* 勝率の計算 \*/  
 if (user\_wins + computer\_wins > 0) {  
 double win\_rate = (double)user\_wins / (user\_wins + computer\_wins) \* 100.0;  
 printf("あなたの勝率: %.1f%%\n", win\_rate);  
   
 if (win\_rate >= 75.0) {  
 printf("評価: 優秀な戦績です！\n");  
 } else if (win\_rate >= 50.0) {  
 printf("評価: まずまずの戦績です。\n");  
 } else {  
 printf("評価: 次回頑張りましょう！\n");  
 }  
 }  
   
 /\* あいこの回数による分析 \*/  
 if (draws == 0) {  
 printf("あいこなしの白熱した戦いでした。\n");  
 } else if (draws == 1) {  
 printf("1回のあいこがありました。\n");  
 } else if (draws == 2) {  
 printf("2回のあいこがありました。運命的な戦いでしたね。\n");  
 } else {  
 printf("全てあいこ！これは珍しい結果です。\n");  
 }  
   
 /\* ランダム性の説明 \*/  
 printf("\n=== 豆知識 ===\n");  
 printf("・コンピュータの手は完全にランダムです\n");  
 printf("・じゃんけんの確率論では、各手が出る確率は1/3です\n");  
 printf("・人間は無意識にパターンを作りがちですが、\n");  
 printf(" 本当のランダム性には勝てません\n");  
   
 printf("\nゲーム終了！また遊んでくださいね。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. ランダム数の生成（srand, rand）  
2. 複雑な条件分岐による勝敗判定  
3. ループを使った繰り返し処理  
4. 配列を使った文字列の管理  
5. 統計的な分析とパーセンテージ計算  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- time(NULL)を使った乱数の初期化  
- 複雑な勝敗判定ロジック  
- ユーザー入力の検証  
- 詳細な結果分析とフィードバック  
  
じゃんけんの勝敗ルール:  
- グー(1) > チョキ(2)  
- チョキ(2) > パー(3)   
- パー(3) > グー(1)  
\*/```  
  
  
```{=openxml}  
<w:p><w:r><w:br w:type="page"/></w:r></w:p>

# 第6章 制御構造（ループ）

## 演習問題

## 演習の目的

* for文、while文、do-while文の使い分けを理解する
* break文とcontinue文を適切に使えるようになる
* ネストしたループを活用した問題解決能力を養う

## 演習問題

### 演習5-1: フィボナッチ数列

最初のN個のフィボナッチ数を表示するプログラムを作成してください。

**要件:** - ユーザーから表示する個数Nを入力 - フィボナッチ数列: 0, 1, 1, 2, 3, 5, 8, 13, … - for文を使用して実装

**実装のヒント:** - 最初の2つの数は0と1 - 3番目以降は前の2つの数の和

### 演習5-2: 数当てゲーム（改良版）

1から100までの数当てゲームを作成してください。

**要件:** - プログラムがランダムに1～100の数を決定 - ユーザーが予想を入力 - 「もっと大きい」「もっと小さい」のヒントを出す - 10回以内に当てられなければゲームオーバー - do-while文を使用

**実装のヒント:** - rand() % 100 + 1 でランダムな数を生成 - 試行回数をカウント

### 演習5-3: 素数リスト

指定された範囲内のすべての素数を表示するプログラムを作成してください。

**要件:** - ユーザーから範囲（開始値と終了値）を入力 - その範囲内の素数をすべて表示 - 素数の個数も表示 - ネストしたループとbreak文を活用

**実装のヒント:** - 2からsqrt(n)まで割り切れるか確認 - 効率化のため、偶数は2以外スキップ

### 演習5-4: 図形の描画

アスタリスク(\*)を使って様々な図形を描画するプログラムを作成してください。

**要件:** 以下の4つの図形を描画する機能を実装: 1. 正方形 2. 直角三角形 3. 逆直角三角形 4. ダイヤモンド

**例（サイズ5の場合）:**

正方形:

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

直角三角形:

\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\*

逆直角三角形:

\*\*\*\*\*  
\*\*\*\*  
\*\*\*  
\*\*  
\*

ダイヤモンド:

\*  
 \*\*\*  
\*\*\*\*\*  
 \*\*\*  
 \*

**実装のヒント:** - ネストしたループを使用 - 空白文字とアスタリスクの配置に注意

### 演習5-5: 掛け算表

9×9の掛け算表を表示するプログラムを作成してください。

**要件:** - ネストしたfor文を使用 - 見やすい表形式で出力 - 列と行のヘッダーを含める

**実装のヒント:** - printf()の書式指定子で桁揃えを行う - %3d などを使用して桁数を統一

### 演習5-6: 階乗計算

ユーザーから入力された数の階乗を計算するプログラムを作成してください。

**要件:** - ユーザーから正の整数を入力 - その数の階乗を計算して表示 - while文を使用して実装 - オーバーフローに注意

**実装のヒント:** - n! = n × (n-1) × (n-2) × … × 1 - long long型の使用を検討

## 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 ex5\_1.c, ex5\_2.c のような形式で
3. コメントで問題番号と簡単な説明を記載
4. コンパイル・実行確認を必ず実行

## 期限

演習問題は次の章に進む前に完了させましょう。

## 解答例

## 解答例一覧

### 演習5-1: フィボナッチ数列

* **ファイル**: ex5\_1\_fibonacci.c, ex5\_1\_fibonacci\_c99.c
* **学習内容**: for文による繰り返し処理、フィボナッチ数列のアルゴリズム
* **ポイント**: 変数の値の更新と循環、初期値の特別処理

### 演習5-2: 数当てゲーム（改良版）

* **ファイル**: ex5\_2\_number\_guess.c, ex5\_2\_number\_guess\_c99.c
* **学習内容**: do-while文による繰り返し、ランダム数生成
* **ポイント**: 複数の終了条件、デモ版として自動実行

### 演習5-3: 素数リスト

* **ファイル**: ex5\_3\_prime\_list.c, ex5\_3\_prime\_list\_c99.c
* **学習内容**: ネストしたループ、break文、効率的な素数判定
* **ポイント**: 数学的最適化、統計情報の計算

### 演習5-4: 図形の描画

* **ファイル**: ex5\_4\_draw\_shapes.c, ex5\_4\_draw\_shapes\_c99.c
* **学習内容**: ネストしたループによる2次元パターン生成
* **ポイント**: 空白とアスタリスクの配置制御、複雑な図形の分解

### 演習5-5: 掛け算表

* **ファイル**: ex5\_5\_multiplication\_table.c, ex5\_5\_multiplication\_table\_c99.c
* **学習内容**: ネストしたfor文、表形式データの生成
* **ポイント**: 書式指定子による桁揃え、ヘッダーと区切り線

### 演習5-6: 階乗計算

* **ファイル**: ex5\_6\_factorial.c, ex5\_6\_factorial\_c99.c
* **学習内容**: while文による繰り返し、オーバーフロー対策
* **ポイント**: long long型の使用、計算過程の表示

## 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
* /\* \*/ 形式のコメントを使用
* 従来のCの制御構造に準拠

### 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
* for文内でのループ変数宣言
* // 形式のコメントに対応
* より現代的な制御構造の書き方

## コンパイルと実行

### 基本的なコンパイル

gcc -Wall -Wextra -pedantic -std=c90 ex5\_1\_fibonacci.c -o ex5\_1\_fibonacci  
./ex5\_1\_fibonacci

### C99版のコンパイル

gcc -Wall -Wextra -pedantic -std=c99 ex5\_1\_fibonacci\_c99.c -o ex5\_1\_fibonacci\_c99  
./ex5\_1\_fibonacci\_c99

### 数学関数を使用する場合（素数判定）

gcc -Wall -Wextra -pedantic -std=c90 ex5\_3\_prime\_list.c -o ex5\_3\_prime\_list -lm  
./ex5\_3\_prime\_list

## 学習のポイント

### ループの基本

1. **for文**: 回数が決まった繰り返し処理
2. **while文**: 条件が真の間の繰り返し処理
3. **do-while文**: 最低1回は実行される繰り返し処理
4. **ネストしたループ**: 2次元的な処理パターン

### 制御文の活用

1. **break文**: ループの早期終了
2. **continue文**: 次の繰り返しへのスキップ
3. **条件分岐との組み合わせ**: 複雑な制御フロー

### 実用的なテクニック

1. **計算アルゴリズム**: フィボナッチ、階乗、素数判定
2. **パターン生成**: 図形描画、表形式データ
3. **入力検証**: エラーハンドリングとユーザビリティ
4. **最適化**: 効率的なアルゴリズムの実装

### プログラム設計

1. **段階的な実装**: 簡単な処理から複雑な処理へ
2. **モジュール化**: 機能ごとの分割と整理
3. **デバッグ**: ループの動作確認と修正
4. **可読性**: 理解しやすいコードの書き方

## 注意事項

* ループの無限ループに注意
* 配列の境界を超えないよう注意
* オーバーフローの可能性を考慮
* 効率的なアルゴリズムの選択

## 対話型プログラムについて

このサンプルコードでは、学習効果を高めるため、一部のプログラムは非対話的なデモ版として実装されています。実際の開発では、ユーザー入力を適切に処理する対話型プログラムとして実装してください。

## 実用的な応用

これらの制御構造は以下のような実用的なプログラムで使用されます：

* **データ処理**: 大量データの反復処理
* **ゲーム**: ゲームループと状態管理
* **シミュレーション**: 反復計算による数値解析
* **グラフィックス**: パターン生成と描画処理

## 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6章: 配列](../arrays/) - [第7章: 文字列](../strings/) - [第8章: 関数](../functions/) ### ex5\_1\_fibonacci.c

/\*  
 \* ファイル名: ex5\_1\_fibonacci.c  
 \* 演習5-1: フィボナッチ数列  
 \* 説明: 最初のN個のフィボナッチ数を表示するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int n, i;  
 int first, second, next;  
   
 printf("=== フィボナッチ数列プログラム ===\n");  
 printf("表示する個数を入力してください: ");  
 scanf("%d", &n);  
   
 /\* 入力値の検証 \*/  
 if (n <= 0) {  
 printf("エラー: 正の整数を入力してください。\n");  
 return 1;  
 }  
   
 printf("\n最初の%d個のフィボナッチ数:\n", n);  
   
 /\* 初期値の設定 \*/  
 first = 0;  
 second = 1;  
   
 /\* フィボナッチ数列の表示 \*/  
 if (n >= 1) {  
 printf("%d ", first);  
 }  
 if (n >= 2) {  
 printf("%d ", second);  
 }  
   
 /\* for文を使用してフィボナッチ数列を計算・表示 \*/  
 for (i = 3; i <= n; i++) {  
 next = first + second;  
 printf("%d ", next);  
   
 /\* 次の計算のために値を更新 \*/  
 first = second;  
 second = next;  
 }  
   
 printf("\n\n");  
 printf("フィボナッチ数列の計算が完了しました。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. for文による繰り返し処理  
2. 変数の値の更新と循環  
3. フィボナッチ数列のアルゴリズム  
4. 条件分岐との組み合わせ  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- for文を使用してフィボナッチ数列を計算  
- 初期値（0, 1）の特別な処理  
- 値の更新を正確に行う  
\*/

### ex5\_2\_number\_guess.c

/\*  
 \* ファイル名: ex5\_2\_number\_guess.c  
 \* 演習5-2: 数当てゲーム（改良版）  
 \* 説明: 1から100までの数当てゲーム（デモ版）  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int target\_number;  
 int guess;  
 int attempts;  
 int max\_attempts;  
 int found;  
   
 printf("=== 数当てゲーム ===\n");  
   
 /\* ランダムシードの初期化 \*/  
 srand(time(NULL));  
   
 /\* 1から100までのランダムな数を生成 \*/  
 target\_number = rand() % 100 + 1;  
 max\_attempts = 10;  
 attempts = 0;  
 found = 0;  
   
 printf("1から100までの数を%d回以内で当ててください。\n", max\_attempts);  
 printf("（実際のゲームでは対話的に入力しますが、デモでは自動実行します）\n\n");  
   
 /\* デモ用の予想値配列（実際の実装では scanf で入力） \*/  
 {  
 int demo\_guesses[] = {50, 75, 25, 37, 44, 41, 43, 42};  
 int demo\_count = sizeof(demo\_guesses) / sizeof(demo\_guesses[0]);  
 int i;  
   
 printf("正解は %d です（通常は表示されません）\n\n", target\_number);  
   
 /\* do-while文を使用したゲームループ \*/  
 do {  
 attempts++;  
   
 /\* デモ用の予想値を使用 \*/  
 if (attempts <= demo\_count) {  
 guess = demo\_guesses[attempts - 1];  
 } else {  
 guess = target\_number; /\* 最後は正解にする \*/  
 }  
   
 printf("試行 %d: %d\n", attempts, guess);  
   
 if (guess == target\_number) {  
 printf("おめでとうございます！正解です！\n");  
 printf("%d回で正解しました。\n", attempts);  
 found = 1;  
 } else if (guess < target\_number) {  
 printf("もっと大きい数です。\n");  
 } else {  
 printf("もっと小さい数です。\n");  
 }  
   
 printf("\n");  
   
 } while (!found && attempts < max\_attempts);  
   
 /\* ゲーム終了の処理 \*/  
 if (!found) {  
 printf("残念！%d回の試行で当てられませんでした。\n", max\_attempts);  
 printf("正解は %d でした。\n", target\_number);  
 }  
 }  
   
 printf("ゲーム終了です。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. do-while文による繰り返し処理  
2. ランダム数の生成  
3. 条件による処理の分岐  
4. ループの終了条件の制御  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- srand()とrand()を使用したランダム数生成  
- do-while文でゲームループを実装  
- 複数の終了条件（正解または試行回数上限）  
- デモ版として自動実行する形に変更  
\*/

### ex5\_3\_prime\_list.c

/\*  
 \* ファイル名: ex5\_3\_prime\_list.c  
 \* 演習5-3: 素数リスト  
 \* 説明: 指定された範囲内のすべての素数を表示するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <math.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int start, end;  
 int num, i;  
 int is\_prime;  
 int prime\_count;  
 int sqrt\_num;  
   
 printf("=== 素数リスト プログラム ===\n");  
 printf("開始値を入力してください: ");  
 scanf("%d", &start);  
 printf("終了値を入力してください: ");  
 scanf("%d", &end);  
   
 /\* 入力値の検証 \*/  
 if (start < 2) {  
 start = 2; /\* 最小の素数は2 \*/  
 }  
 if (end < start) {  
 printf("エラー: 終了値は開始値以上である必要があります。\n");  
 return 1;  
 }  
   
 printf("\n%dから%dまでの素数:\n", start, end);  
 prime\_count = 0;  
   
 /\* 各数について素数判定を行う \*/  
 for (num = start; num <= end; num++) {  
 is\_prime = 1; /\* 素数と仮定 \*/  
   
 /\* 2未満は素数ではない \*/  
 if (num < 2) {  
 is\_prime = 0;  
 } else if (num == 2) {  
 /\* 2は素数 \*/  
 is\_prime = 1;  
 } else if (num % 2 == 0) {  
 /\* 2以外の偶数は素数ではない \*/  
 is\_prime = 0;  
 } else {  
 /\* 奇数について3からsqrt(num)まで割り切れるかチェック \*/  
 sqrt\_num = (int)sqrt(num);  
 for (i = 3; i <= sqrt\_num; i += 2) {  
 if (num % i == 0) {  
 is\_prime = 0;  
 break; /\* 割り切れたので素数ではない \*/  
 }  
 }  
 }  
   
 /\* 素数の場合は表示 \*/  
 if (is\_prime) {  
 printf("%d ", num);  
 prime\_count++;  
   
 /\* 10個ごとに改行 \*/  
 if (prime\_count % 10 == 0) {  
 printf("\n");  
 }  
 }  
 }  
   
 printf("\n\n=== 結果 ===\n");  
 printf("範囲内の素数の個数: %d個\n", prime\_count);  
   
 /\* 統計情報の表示 \*/  
 if (prime\_count > 0) {  
 printf("素数の密度: %.2f%%\n",   
 (double)prime\_count / (end - start + 1) \* 100);  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. ネストしたループの使用  
2. break文による早期ループ終了  
3. 効率的な素数判定アルゴリズム  
4. 数学的な最適化（sqrt、偶数のスキップ）  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- sqrt()関数を使用した効率的な素数判定  
- 偶数の特別処理（2以外はスキップ）  
- break文によるループの最適化  
- 統計情報の計算と表示  
\*/

### ex5\_4\_draw\_shapes.c

/\*  
 \* ファイル名: ex5\_4\_draw\_shapes.c  
 \* 演習5-4: 図形の描画  
 \* 説明: アスタリスクを使って様々な図形を描画するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int size;  
 int i, j, spaces;  
   
 printf("=== 図形描画プログラム ===\n");  
 printf("図形のサイズを入力してください: ");  
 scanf("%d", &size);  
   
 /\* 入力値の検証 \*/  
 if (size <= 0) {  
 printf("エラー: 正の整数を入力してください。\n");  
 return 1;  
 }  
   
 /\* 1. 正方形の描画 \*/  
 printf("\n1. 正方形（%d×%d）:\n", size, size);  
 for (i = 0; i < size; i++) {  
 for (j = 0; j < size; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 /\* 2. 直角三角形の描画 \*/  
 printf("\n2. 直角三角形:\n");  
 for (i = 1; i <= size; i++) {  
 for (j = 0; j < i; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 /\* 3. 逆直角三角形の描画 \*/  
 printf("\n3. 逆直角三角形:\n");  
 for (i = size; i >= 1; i--) {  
 for (j = 0; j < i; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 /\* 4. ダイヤモンドの描画 \*/  
 printf("\n4. ダイヤモンド:\n");  
   
 /\* 上半分（中央含む） \*/  
 for (i = 1; i <= size; i += 2) {  
 /\* 先頭のスペース \*/  
 spaces = (size - i) / 2;  
 for (j = 0; j < spaces; j++) {  
 printf(" ");  
 }  
   
 /\* アスタリスクの出力 \*/  
 for (j = 0; j < i; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 /\* 下半分 \*/  
 for (i = size - 2; i >= 1; i -= 2) {  
 /\* 先頭のスペース \*/  
 spaces = (size - i) / 2;  
 for (j = 0; j < spaces; j++) {  
 printf(" ");  
 }  
   
 /\* アスタリスクの出力 \*/  
 for (j = 0; j < i; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 /\* 5. 右直角三角形（空白付き） \*/  
 printf("\n5. 右直角三角形:\n");  
 for (i = 1; i <= size; i++) {  
 /\* 先頭のスペース \*/  
 for (j = 0; j < size - i; j++) {  
 printf(" ");  
 }  
 /\* アスタリスク \*/  
 for (j = 0; j < i; j++) {  
 printf("\*");  
 }  
 printf("\n");  
 }  
   
 printf("\n図形の描画が完了しました。\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. ネストしたループによる2次元パターンの生成  
2. 空白文字とアスタリスクの配置制御  
3. ループ変数の増減による図形の変化  
4. 複雑な図形（ダイヤモンド）の分解と実装  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 外側ループで行、内側ループで列を制御  
- 空白文字の計算と出力による図形の位置調整  
- ダイヤモンドは上半分と下半分に分けて実装  
- 様々なループパターンの組み合わせ  
\*/

### ex5\_5\_multiplication\_table.c

/\*  
 \* ファイル名: ex5\_5\_multiplication\_table.c  
 \* 演習5-5: 掛け算表  
 \* 説明: 9×9の掛け算表を表示するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int i, j;  
 int result;  
   
 printf("=== 9×9の掛け算表 ===\n\n");  
   
 /\* 列ヘッダーの表示 \*/  
 printf(" ");  
 for (j = 1; j <= 9; j++) {  
 printf("%3d", j);  
 }  
 printf("\n");  
   
 /\* 区切り線の表示 \*/  
 printf(" ");  
 for (j = 1; j <= 9; j++) {  
 printf("---");  
 }  
 printf("\n");  
   
 /\* 掛け算表の本体 \*/  
 for (i = 1; i <= 9; i++) {  
 /\* 行ヘッダーの表示 \*/  
 printf("%2d|", i);  
   
 /\* 各列の計算結果を表示 \*/  
 for (j = 1; j <= 9; j++) {  
 result = i \* j;  
 printf("%3d", result);  
 }  
 printf("\n");  
 }  
   
 printf("\n=== 掛け算表の完成 ===\n");  
   
 /\* 簡単な統計情報 \*/  
 printf("\n統計情報:\n");  
 printf("- 最小値: 1 (1×1)\n");  
 printf("- 最大値: 81 (9×9)\n");  
 printf("- 表示された計算結果の総数: 81個\n");  
   
 /\* 特別な値の表示 \*/  
 printf("\n完全平方数:\n");  
 for (i = 1; i <= 9; i++) {  
 result = i \* i;  
 printf("%d×%d=%d ", i, i, result);  
 }  
 printf("\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. ネストしたfor文による表形式データの生成  
2. printf()の書式指定子による桁揃え  
3. 表のヘッダーと区切り線の作成  
4. 二重ループでの計算処理  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- %3d書式指定子で3桁の右詰め表示  
- 行と列のヘッダーで見やすい表を作成  
- ネストしたループでの計算と表示  
- 統計情報と特別な値の追加表示  
\*/

### ex5\_6\_factorial.c

/\*  
 \* ファイル名: ex5\_6\_factorial.c  
 \* 演習5-6: 階乗計算  
 \* 説明: ユーザーから入力された数の階乗を計算するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int n;  
 long long factorial;  
 int i;  
   
 printf("=== 階乗計算プログラム ===\n");  
 printf("階乗を計算する正の整数を入力してください: ");  
 scanf("%d", &n);  
   
 /\* 入力値の検証 \*/  
 if (n < 0) {  
 printf("エラー: 負の数の階乗は定義されていません。\n");  
 return 1;  
 }  
   
 if (n > 20) {  
 printf("警告: 20より大きい数はオーバーフローの可能性があります。\n");  
 printf("計算を続行しますが、結果が正確でない可能性があります。\n");  
 }  
   
 /\* 階乗の計算（while文を使用） \*/  
 factorial = 1;  
 i = 1;  
   
 printf("\n計算過程:\n");  
 printf("%d! = ", n);  
   
 /\* 0の階乗は1 \*/  
 if (n == 0) {  
 printf("1\n");  
 } else {  
 /\* while文による階乗計算 \*/  
 while (i <= n) {  
 factorial \*= i;  
   
 /\* 計算過程の表示 \*/  
 if (i == 1) {  
 printf("%d", i);  
 } else {  
 printf(" × %d", i);  
 }  
   
 i++;  
 }  
 printf(" = %lld\n", factorial);  
 }  
   
 /\* 結果の表示 \*/  
 printf("\n=== 計算結果 ===\n");  
 printf("%d! = %lld\n", n, factorial);  
   
 /\* 追加情報の表示 \*/  
 if (factorial > 1000000) {  
 printf("この値は100万を超えています。\n");  
 }  
   
 /\* 関連する階乗値の表示 \*/  
 if (n <= 10 && n > 0) {  
 printf("\n参考: 1から%dまでの階乗:\n", n);  
 factorial = 1;  
 for (i = 1; i <= n; i++) {  
 factorial \*= i;  
 printf("%d! = %lld\n", i, factorial);  
 }  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. while文による繰り返し処理  
2. 階乗計算のアルゴリズム  
3. オーバーフロー対策（long long型の使用）  
4. 計算過程の表示  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- while文を使用した階乗計算  
- long long型によるオーバーフロー対策  
- 0の階乗（0! = 1）の特別処理  
- 計算過程の詳細な表示  
- 入力値の検証とエラーハンドリング  
\*/

# 第7章 配列

## 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

/\*  
 \* 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("\n");  
}  
  
/\* 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 ===\n");  
 printf("1. 英和検索（英語→日本語）\n");  
 printf("2. 和英検索（日本語→英語）\n");  
 printf("3. 辞書一覧表示\n");  
 printf("4. 新しい単語を追加\n");  
 printf("5. 終了\n");  
 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;  
}```  
  
  
```{=openxml}  
<w:p><w:r><w:br w:type="page"/></w:r></w:p>

# 第8章 文字列処理

## 演習問題

この章では、C言語における文字列操作の基本から応用まで、段階的に学習できる演習問題を提供しています。

## 演習の概要

文字列処理は、プログラミングにおいて最も基本的かつ重要な技術の一つです。この演習では、以下の3つの段階で文字列操作を習得します。

### 学習目標

1. **基本的な文字列操作の理解**
   * null終端文字列の概念
   * ポインタと配列の関係
   * メモリ安全な文字列操作
2. **文字列配列の管理**
   * 二次元配列の操作
   * 動的な要素の追加・削除
   * ソートと検索アルゴリズム
3. **高度な文字列処理**
   * パターンマッチング
   * アルゴリズムの最適化
   * 実用的な応用例

## 演習問題一覧

### 演習8-1: 基本的な文字列操作

**難易度**: 初級  
**所要時間**: 2-3時間

標準ライブラリの関数を使わずに、基本的な文字列操作関数を実装します。

**実装する関数**: 1. my\_strlen() - 文字列の長さを計算 2. my\_strcpy() - 文字列をコピー 3. my\_strcmp() - 文字列を比較 4. my\_strcat() - 文字列を連結 5. my\_strchr() - 文字を検索 6. my\_strrchr() - 文字を後方検索 7. count\_char() - 特定文字をカウント

**学習ポイント**: - null終端文字列の仕組み - ポインタ演算の基礎 - バッファオーバーフロー対策 - エラーハンドリング

### 演習8-2: 文字列配列の管理

**難易度**: 中級  
**所要時間**: 3-4時間

学生名を格納する文字列配列の管理システムを実装します。

**実装する機能**: 1. **配列管理機能** - 学生名の追加・削除 - 重複チェック - 配列の動的管理

1. **検索とソート機能**
   * 名前による検索
   * アルファベット順ソート
   * 一覧表示機能

**学習ポイント**: - 二次元配列の操作 - 配列要素の挿入・削除 - バブルソートアルゴリズム - 線形検索の実装

### 演習8-3: 高度な文字列検索

**難易度**: 上級  
**所要時間**: 4-6時間

標準ライブラリを使わずに、高度な文字列検索機能を実装します。

**実装する機能**: 1. **検索機能** - 大小文字を区別しない検索 - 単語境界での検索 - 部分文字列検索

1. **パターンマッチング**
   * ワイルドカード検索（\*、?）
   * 複数パターンの同時検索
   * 検索結果の詳細情報

**学習ポイント**: - 高度な文字列マッチングアルゴリズム - メモリ効率的な実装 - 動的プログラミング - パフォーマンス最適化

## 提出要件

各演習について、以下の要件を満たしてください：

### 必須要件

1. **ソースコード**
   * 適切なコメント（日本語）
   * 関数の詳細な説明
   * エラーハンドリング
2. **テストプログラム**
   * 各関数の動作確認
   * 境界値テスト
   * エラーケースのテスト
3. **実行例**
   * 正常ケースの実行結果
   * エラーケースの実行結果

### 推奨要件

1. **コーディングスタイル**
   * 一貫したインデント
   * 意味のある変数名
   * 適切な関数分割
2. **安全性**
   * バッファオーバーフロー対策
   * NULLポインタチェック
   * メモリリークの防止

## 評価基準

### 正確性 (40%)

* 仕様通りに動作するか
* エラーケースが適切に処理されるか
* テストケースが網羅的か

### 安全性 (30%)

* バッファオーバーフローがないか
* メモリ管理が適切か
* 入力検証が十分か

### 効率性 (20%)

* アルゴリズムの計算量が適切か
* 不要な処理がないか
* メモリ使用量が最適か

### 可読性 (10%)

* コードが理解しやすいか
* コメントが適切か
* 命名規則が一貫しているか

## 学習の進め方

### 推奨手順

1. **基礎の理解**
   * 文字列の基本概念を復習
   * ポインタと配列の関係を確認
   * null終端文字列の仕組みを理解
2. **段階的実装**
   * 演習1から順番に取り組む
   * 小さな関数から実装開始
   * テストケースを先に作成
3. **デバッグとテスト**
   * コンパイル時の警告を確認
   * 実行時エラーをチェック
   * メモリ使用状況を監視
4. **最適化と改善**
   * パフォーマンスを測定
   * コードの可読性を向上
   * エラーハンドリングを強化

### 学習のヒント

1. **安全第一**
   * 配列の境界を常に確認
   * NULLポインタをチェック
   * バッファサイズを適切に管理
2. **テスト駆動**
   * 関数作成前にテストケースを考える
   * エッジケースを忘れずに
   * 段階的に機能を拡張
3. **効率性の考慮**
   * 不要な文字列コピーを避ける
   * ポインタ操作を活用
   * メモリ使用量を最適化

## トラブルシューティング

### よくある問題

1. **セグメンテーション違反**
   * 配列の境界越え
   * null終端の忘れ
   * 未初期化ポインタ
2. **予期しない出力**
   * null終端文字の扱い
   * 文字コードの問題
   * バッファの不足
3. **メモリリーク**
   * 動的メモリの解放忘れ
   * 重複した確保
   * 例外時の処理不備

### デバッグ方法

1. **printfデバッグ**
   * 変数の値を確認
   * 関数の呼び出し順序を追跡
   * 配列の内容を表示
2. **gdbの使用**
   * ブレークポイントの設定
   * ステップ実行
   * 変数の監視
3. **静的解析ツール**
   * コンパイラの警告
   * lintツールの使用
   * メモリチェッカー

## 参考資料

### 必読資料

* [文字列操作の基礎](../README.md)
* [実装例とサンプルコード](../examples/)
* [解答例](../solutions/)

### 推奨図書

* 『プログラミング言語C 第2版』（K&R）
* 『Cプログラミング専門課程』
* 『セキュアプログラミング講座』

### オンラインリソース

* C言語標準ライブラリリファレンス
* 文字列処理アルゴリズム
* セキュアコーディング標準

これらの演習を通じて、C言語における文字列操作の基礎から応用まで、体系的に学習することができます。各演習に丁寧に取り組み、安全で効率的なコードの作成技術を身につけてください。

## 解答例

このディレクトリには文字列操作に関する演習問題の解答例が含まれています。

## 解答ファイル一覧

### 演習8-1: 基本的な文字列操作関数の実装

標準ライブラリの関数を使わずに、基本的な文字列操作関数を実装する問題です。

* **C90版**: <ex8_1_string_operations.c>
* **C99版**: <ex8_1_string_operations_c99.c>

**実装する関数**: - my\_strlen() - 文字列の長さを計算 - my\_strcpy() - 文字列をコピー - my\_strcmp() - 文字列を比較 - my\_strcat() - 文字列を連結 - my\_strchr() - 文字を検索

**学習ポイント**: - 文字列がnull終端であることの理解 - ポインタと配列の操作 - 文字列処理の基本アルゴリズム

### 演習8-2: 文字列配列の管理システム

学生名を格納する文字列配列の管理システムを実装する問題です。

* **C90版**: <ex8_2_string_arrays.c>
* **C99版**: <ex8_2_string_arrays_c99.c>

**実装する機能**: - 学生名の追加・削除 - 名前による検索 - アルファベット順ソート - 一覧表示

**学習ポイント**: - 二次元配列の操作 - 文字列の動的管理 - 配列要素の挿入・削除 - 文字列ソートアルゴリズム

### 演習8-3: 高度な文字列検索関数

標準ライブラリを使わずに、高度な文字列検索機能を実装する問題です。

* **C90版**: <ex8_3_string_search.c>
* **C99版**: <ex8_3_string_search_c99.c>

**実装する機能**: - 大小文字を区別しない検索 - 単語境界での検索 - ワイルドカード検索（\*、?） - 複数パターンの同時検索

**学習ポイント**: - 高度な文字列マッチングアルゴリズム - パターンマッチング - 検索結果の詳細情報管理 - メモリ効率的な実装

## C90版とC99版の違い

### C90版の特徴

* ANSI C（C90）に準拠
* 従来の配列とポインタ操作
* 関数の先頭での変数宣言
* int型による真偽値表現

### C99版の特徴

* C99の新機能を活用
* **bool型**: 明確な真偽値表現
* **可変長配列（VLA）**: 実行時サイズ決定
* **複合リテラル**: 一時的なデータ構造
* **指定初期化子**: 特定要素の初期化
* **restrict修飾子**: 最適化ヒント
* **inline関数**: 関数呼び出しコスト削減
* **固定幅整数型**: 明確なデータサイズ

## コンパイルと実行

### 基本的なコンパイル

# C90版  
gcc -Wall -Wextra -pedantic -std=c90 ex8\_1\_string\_operations.c -o ex8\_1  
  
# C99版  
gcc -Wall -Wextra -pedantic -std=c99 ex8\_1\_string\_operations\_c99.c -o ex8\_1\_c99

### Makefileを使用

# 全ての解答をコンパイル  
make solutions  
  
# 特定の解答を実行  
make run-ex8-1  
make run-ex8-1-c99

## 学習の進め方

1. **基礎から応用へ**: 演習8-1から順番に取り組む
2. **C90版を先に学習**: 基本的な実装を理解
3. **C99版で機能拡張**: 新しい言語機能を活用
4. **コードの比較**: 両バージョンの違いを理解
5. **実行とテスト**: 実際に動かして動作を確認

## 各解答の詳細説明

### 演習8-1: 基本的な文字列操作

**C90版の重要ポイント**: - 基本的なポインタ操作 - null終端文字列の理解 - エラーハンドリングの基礎

**C99版の追加機能**: - restrict修飾子による最適化 - bool型による明確な戻り値 - inline関数による性能向上 - 詳細な統計情報と分析機能

### 演習8-2: 文字列配列管理

**C90版の重要ポイント**: - 二次元配列の操作 - 配列要素の動的管理 - 基本的なソートアルゴリズム

**C99版の追加機能**: - 可変長配列による柔軟な処理 - 構造体による詳細情報管理 - 複数の操作モード（検索・削除） - 統計分析機能

### 演習8-3: 高度な文字列検索

**C90版の重要ポイント**: - 基本的なパターンマッチング - 文字列検索アルゴリズム - メモリ効率的な実装

**C99版の追加機能**: - 動的プログラミングによるワイルドカード検索 - 複数パターン同時検索 - 詳細な検索結果情報 - 可変長配列によるメモリ最適化

## 実践的な応用例

これらの解答では、以下のような実践的な応用例も示しています：

1. **ログ解析システム** - 文字列検索を使用
2. **学生管理システム** - 配列操作を使用
3. **文字列統計分析** - 基本操作を組み合わせ
4. **パフォーマンステスト** - 大量データでの動作確認

## トラブルシューティング

### よくある問題

1. **セグメンテーション違反**
   * null終端の確認
   * 配列の境界チェック
   * ポインタの初期化
2. **コンパイルエラー**
   * 標準の指定（-std=c90 または -std=c99）
   * 必要なヘッダファイルのインクルード
3. **予期しない動作**
   * デバッグ情報の有効化（-gフラグ）
   * テストケースでの段階的確認

## 参考資料

* [C言語の文字列操作基礎](../README.md)
* [演習問題の詳細](../exercises/README.md)
* [実行例とサンプルコード](../examples/)

これらの解答例を通じて、C言語における文字列操作の基礎から応用まで、体系的に学習することができます。C90版で基本を固め、C99版で新しい機能と最適化テクニックを身につけてください。 ### ex8\_1\_string\_operations.c

/\*  
 \* ファイル名: ex8\_1\_string\_operations.c  
 \* 演習8-1: 基本的な文字列操作  
 \* 説明: 標準ライブラリを使わずに文字列操作関数を実装  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stddef.h>  
  
/\* 文字列の長さを計算する関数 \*/  
int my\_strlen(const char str[])  
{  
 int length = 0;  
   
 if (str == NULL) {  
 return 0;  
 }  
   
 while (str[length] != '\0') {  
 length++;  
 }  
   
 return length;  
}  
  
/\* 文字列をコピーする関数 \*/  
void my\_strcpy(char dest[], const char src[])  
{  
 int i = 0;  
   
 if (dest == NULL || src == NULL) {  
 return;  
 }  
   
 while (src[i] != '\0') {  
 dest[i] = src[i];  
 i++;  
 }  
 dest[i] = '\0'; /\* 終端文字をコピー \*/  
}  
  
/\* 文字列を比較する関数 \*/  
int my\_strcmp(const char str1[], const char str2[])  
{  
 int i = 0;  
   
 if (str1 == NULL && str2 == NULL) {  
 return 0;  
 }  
 if (str1 == NULL) {  
 return -1;  
 }  
 if (str2 == NULL) {  
 return 1;  
 }  
   
 while (str1[i] != '\0' && str2[i] != '\0') {  
 if (str1[i] < str2[i]) {  
 return -1;  
 } else if (str1[i] > str2[i]) {  
 return 1;  
 }  
 i++;  
 }  
   
 /\* どちらかが終了した場合 \*/  
 if (str1[i] == '\0' && str2[i] == '\0') {  
 return 0; /\* 等しい \*/  
 } else if (str1[i] == '\0') {  
 return -1; /\* str1が短い \*/  
 } else {  
 return 1; /\* str2が短い \*/  
 }  
}  
  
/\* 文字列を連結する関数 \*/  
void my\_strcat(char dest[], const char src[])  
{  
 int dest\_len = 0;  
 int i = 0;  
   
 if (dest == NULL || src == NULL) {  
 return;  
 }  
   
 /\* 連結先の終端文字を見つける \*/  
 while (dest[dest\_len] != '\0') {  
 dest\_len++;  
 }  
   
 /\* 連結元をコピー \*/  
 while (src[i] != '\0') {  
 dest[dest\_len + i] = src[i];  
 i++;  
 }  
   
 /\* 終端文字を追加 \*/  
 dest[dest\_len + i] = '\0';  
}  
  
/\* 文字列内で指定した文字を検索する関数 \*/  
char \*my\_strchr(const char str[], int ch)  
{  
 int i = 0;  
   
 if (str == NULL) {  
 return NULL;  
 }  
   
 while (str[i] != '\0') {  
 if (str[i] == ch) {  
 return (char \*)&str[i]; /\* 見つかった位置を返す \*/  
 }  
 i++;  
 }  
   
 /\* 終端文字を検索する場合 \*/  
 if (ch == '\0') {  
 return (char \*)&str[i];  
 }  
   
 return NULL; /\* 見つからなかった \*/  
}  
  
/\* 文字列内で指定した文字を最後から検索する関数 \*/  
char \*my\_strrchr(const char str[], int ch)  
{  
 int i;  
 int length;  
   
 if (str == NULL) {  
 return NULL;  
 }  
   
 length = my\_strlen(str);  
   
 /\* 後ろから検索 \*/  
 for (i = length; i >= 0; i--) {  
 if (str[i] == ch) {  
 return (char \*)&str[i];  
 }  
 }  
   
 return NULL; /\* 見つからなかった \*/  
}  
  
/\* 文字列内の特定文字をカウントする関数 \*/  
int count\_char(const char str[], char ch)  
{  
 int count = 0;  
 int i = 0;  
   
 if (str == NULL) {  
 return 0;  
 }  
   
 while (str[i] != '\0') {  
 if (str[i] == ch) {  
 count++;  
 }  
 i++;  
 }  
   
 return count;  
}  
  
/\* テスト関数 \*/  
void test\_string\_functions(void)  
{  
 char buffer[100];  
 char \*result;  
   
 printf("=== 自作文字列関数のテスト ===\n");  
   
 /\* my\_strlen のテスト \*/  
 printf("\n1. my\_strlen のテスト:\n");  
 printf("my\_strlen(\"Hello\"): %d\n", my\_strlen("Hello"));  
 printf("my\_strlen(\"\"): %d\n", my\_strlen(""));  
 printf("my\_strlen(\"Hello World\"): %d\n", my\_strlen("Hello World"));  
   
 /\* my\_strcpy のテスト \*/  
 printf("\n2. my\_strcpy のテスト:\n");  
 my\_strcpy(buffer, "Hello");  
 printf("コピー結果: \"%s\"\n", buffer);  
   
 /\* my\_strcmp のテスト \*/  
 printf("\n3. my\_strcmp のテスト:\n");  
 printf("my\_strcmp(\"apple\", \"apple\"): %d\n", my\_strcmp("apple", "apple"));  
 printf("my\_strcmp(\"apple\", \"banana\"): %d\n", my\_strcmp("apple", "banana"));  
 printf("my\_strcmp(\"banana\", \"apple\"): %d\n", my\_strcmp("banana", "apple"));  
   
 /\* my\_strcat のテスト \*/  
 printf("\n4. my\_strcat のテスト:\n");  
 my\_strcpy(buffer, "Hello");  
 my\_strcat(buffer, " World");  
 printf("連結結果: \"%s\"\n", buffer);  
   
 /\* my\_strchr のテスト \*/  
 printf("\n5. my\_strchr のテスト:\n");  
 result = my\_strchr("Hello World", 'W');  
 if (result != NULL) {  
 printf("'W' found at position: %ld\n", result - "Hello World");  
 }  
   
 result = my\_strchr("Hello World", 'x');  
 if (result == NULL) {  
 printf("'x' not found\n");  
 }  
   
 /\* my\_strrchr のテスト \*/  
 printf("\n6. my\_strrchr のテスト:\n");  
 result = my\_strrchr("Hello World", 'l');  
 if (result != NULL) {  
 printf("Last 'l' found at position: %ld\n", result - "Hello World");  
 }  
   
 /\* count\_char のテスト \*/  
 printf("\n7. count\_char のテスト:\n");  
 printf("count\_char(\"Hello World\", 'l'): %d\n", count\_char("Hello World", 'l'));  
 printf("count\_char(\"Hello World\", 'o'): %d\n", count\_char("Hello World", 'o'));  
 printf("count\_char(\"Hello World\", 'x'): %d\n", count\_char("Hello World", 'x'));  
}  
  
int main(void)  
{  
 printf("===== 基本的な文字列操作の実装 =====\n");  
   
 test\_string\_functions();  
   
 printf("\n=== 実用例 ===\n");  
   
 /\* 実用例: 文字列の解析 \*/  
 char text[] = "C programming is fun and challenging!";  
 printf("文字列: \"%s\"\n", text);  
 printf("長さ: %d文字\n", my\_strlen(text));  
 printf("'n'の数: %d個\n", count\_char(text, 'n'));  
 printf("スペースの数: %d個\n", count\_char(text, ' '));  
   
 /\* 'a'の最初と最後の位置 \*/  
 char \*first\_a = my\_strchr(text, 'a');  
 char \*last\_a = my\_strrchr(text, 'a');  
   
 if (first\_a != NULL) {  
 printf("最初の'a'の位置: %ld\n", first\_a - text);  
 }  
 if (last\_a != NULL) {  
 printf("最後の'a'の位置: %ld\n", last\_a - text);  
 }  
   
 /\* 文字列の組み立て \*/  
 char result[200] = "";  
 my\_strcpy(result, "Hello");  
 my\_strcat(result, ", ");  
 my\_strcat(result, "World");  
 my\_strcat(result, "!");  
 printf("\n組み立てた文字列: \"%s\"\n", result);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 標準ライブラリの再実装:  
 - 関数の内部動作の理解  
 - ポインタ操作の練習  
 - エッジケースの処理  
  
2. エラーハンドリング:  
 - NULLポインタのチェック  
 - 境界条件の処理  
 - 適切な戻り値の設定  
  
3. メモリ安全性:  
 - バッファオーバーフローの防止  
 - 終端文字の適切な処理  
 - ポインタの有効性確認  
  
4. 実用的な応用:  
 - 文字列解析  
 - テキスト処理  
 - データ検索  
  
注意点:  
- ポインタ演算時の型安全性  
- 文字列の不変性の維持  
- 効率的なアルゴリズムの選択  
\*/

### ex8\_2\_string\_arrays.c

/\*  
 \* ファイル名: ex8\_2\_string\_arrays.c  
 \* 演習8-2: 文字列配列の管理  
 \* 説明: 学生名を格納する文字列配列の追加・削除・表示機能  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_STUDENTS 10  
#define MAX\_NAME\_LENGTH 50  
  
/\* 学生名配列の表示 \*/  
void display\_students(char students[][MAX\_NAME\_LENGTH], int count)  
{  
 int i;  
   
 if (count == 0) {  
 printf("登録されている学生はいません。\n");  
 return;  
 }  
   
 printf("=== 学生一覧 ===\n");  
 for (i = 0; i < count; i++) {  
 printf("%d. %s\n", i + 1, students[i]);  
 }  
 printf("合計: %d名\n", count);  
}  
  
/\* 学生の追加 \*/  
int add\_student(char students[][MAX\_NAME\_LENGTH], int \*count, const char \*name)  
{  
 if (\*count >= MAX\_STUDENTS) {  
 printf("エラー: これ以上学生を追加できません（最大%d名）\n", MAX\_STUDENTS);  
 return 0; /\* 失敗 \*/  
 }  
   
 if (strlen(name) >= MAX\_NAME\_LENGTH) {  
 printf("エラー: 名前が長すぎます（最大%d文字）\n", MAX\_NAME\_LENGTH - 1);  
 return 0; /\* 失敗 \*/  
 }  
   
 /\* 重複チェック \*/  
 {  
 int i;  
 for (i = 0; i < \*count; i++) {  
 if (strcmp(students[i], name) == 0) {  
 printf("エラー: \"%s\" は既に登録されています\n", name);  
 return 0; /\* 失敗 \*/  
 }  
 }  
 }  
   
 strcpy(students[\*count], name);  
 (\*count)++;  
 printf("\"%s\" を追加しました\n", name);  
 return 1; /\* 成功 \*/  
}  
  
/\* 学生の削除（名前で検索） \*/  
int remove\_student\_by\_name(char students[][MAX\_NAME\_LENGTH], int \*count, const char \*name)  
{  
 int i, found\_index = -1;  
   
 /\* 削除対象を検索 \*/  
 for (i = 0; i < \*count; i++) {  
 if (strcmp(students[i], name) == 0) {  
 found\_index = i;  
 break;  
 }  
 }  
   
 if (found\_index == -1) {  
 printf("エラー: \"%s\" は見つかりませんでした\n", name);  
 return 0; /\* 失敗 \*/  
 }  
   
 /\* 要素を前に詰める \*/  
 for (i = found\_index; i < \*count - 1; i++) {  
 strcpy(students[i], students[i + 1]);  
 }  
   
 (\*count)--;  
 printf("\"%s\" を削除しました\n", name);  
 return 1; /\* 成功 \*/  
}  
  
/\* 学生の削除（インデックス指定） \*/  
int remove\_student\_by\_index(char students[][MAX\_NAME\_LENGTH], int \*count, int index)  
{  
 int i;  
 char name[MAX\_NAME\_LENGTH];  
   
 if (index < 0 || index >= \*count) {  
 printf("エラー: 無効なインデックスです（1-%d の範囲で指定してください）\n", \*count);  
 return 0; /\* 失敗 \*/  
 }  
   
 /\* 削除前に名前を保存 \*/  
 strcpy(name, students[index]);  
   
 /\* 要素を前に詰める \*/  
 for (i = index; i < \*count - 1; i++) {  
 strcpy(students[i], students[i + 1]);  
 }  
   
 (\*count)--;  
 printf("%d番目の学生 \"%s\" を削除しました\n", index + 1, name);  
 return 1; /\* 成功 \*/  
}  
  
/\* 学生の検索 \*/  
int search\_student(char students[][MAX\_NAME\_LENGTH], int count, const char \*name)  
{  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (strcmp(students[i], name) == 0) {  
 return i; /\* 見つかった場合のインデックス \*/  
 }  
 }  
   
 return -1; /\* 見つからない場合 \*/  
}  
  
/\* 学生名のソート（アルファベット順） \*/  
void sort\_students(char students[][MAX\_NAME\_LENGTH], int count)  
{  
 int i, j;  
 char temp[MAX\_NAME\_LENGTH];  
   
 /\* バブルソート \*/  
 for (i = 0; i < count - 1; i++) {  
 for (j = 0; j < count - 1 - i; j++) {  
 if (strcmp(students[j], students[j + 1]) > 0) {  
 strcpy(temp, students[j]);  
 strcpy(students[j], students[j + 1]);  
 strcpy(students[j + 1], temp);  
 }  
 }  
 }  
   
 printf("学生名をアルファベット順にソートしました\n");  
}  
  
/\* 部分一致検索 \*/  
void search\_partial(char students[][MAX\_NAME\_LENGTH], int count, const char \*partial)  
{  
 int i;  
 int found\_count = 0;  
   
 printf("「%s」を含む学生名:\n", partial);  
   
 for (i = 0; i < count; i++) {  
 if (strstr(students[i], partial) != NULL) {  
 printf(" %d. %s\n", i + 1, students[i]);  
 found\_count++;  
 }  
 }  
   
 if (found\_count == 0) {  
 printf(" 該当する学生はいません\n");  
 } else {  
 printf(" %d名が見つかりました\n", found\_count);  
 }  
}  
  
int main(void)  
{  
 char students[MAX\_STUDENTS][MAX\_NAME\_LENGTH];  
 int student\_count = 0;  
 int search\_index;  
   
 printf("===== 学生名管理システム =====\n\n");  
   
 /\* 初期データの追加 \*/  
 printf("1. 学生の追加:\n");  
 add\_student(students, &student\_count, "田中太郎");  
 add\_student(students, &student\_count, "佐藤花子");  
 add\_student(students, &student\_count, "鈴木一郎");  
 add\_student(students, &student\_count, "高橋二郎");  
 add\_student(students, &student\_count, "伊藤三子");  
   
 printf("\n");  
 display\_students(students, student\_count);  
   
 /\* 重複追加のテスト \*/  
 printf("\n2. 重複追加のテスト:\n");  
 add\_student(students, &student\_count, "田中太郎"); /\* 既に存在 \*/  
   
 /\* 検索のテスト \*/  
 printf("\n3. 学生の検索:\n");  
 search\_index = search\_student(students, student\_count, "佐藤花子");  
 if (search\_index != -1) {  
 printf("\"佐藤花子\" は %d番目にいます\n", search\_index + 1);  
 }  
   
 search\_index = search\_student(students, student\_count, "山田太郎");  
 if (search\_index == -1) {  
 printf("\"山田太郎\" は見つかりませんでした\n");  
 }  
   
 /\* 部分一致検索 \*/  
 printf("\n4. 部分一致検索:\n");  
 search\_partial(students, student\_count, "太郎");  
 search\_partial(students, student\_count, "田");  
   
 /\* ソート \*/  
 printf("\n5. ソート:\n");  
 sort\_students(students, student\_count);  
 display\_students(students, student\_count);  
   
 /\* 削除のテスト \*/  
 printf("\n6. 学生の削除:\n");  
 remove\_student\_by\_name(students, &student\_count, "佐藤花子");  
 display\_students(students, student\_count);  
   
 printf("\n");  
 remove\_student\_by\_index(students, &student\_count, 0); /\* 最初の学生を削除 \*/  
 display\_students(students, student\_count);  
   
 /\* エラーケースのテスト \*/  
 printf("\n7. エラーケースのテスト:\n");  
 remove\_student\_by\_name(students, &student\_count, "存在しない学生");  
 remove\_student\_by\_index(students, &student\_count, 100); /\* 無効なインデックス \*/  
   
 /\* 最終状態 \*/  
 printf("\n=== 最終状態 ===\n");  
 display\_students(students, student\_count);  
   
 /\* 統計情報 \*/  
 printf("\n=== 統計情報 ===\n");  
 printf("現在の学生数: %d名\n", student\_count);  
 printf("最大収容数: %d名\n", MAX\_STUDENTS);  
 printf("残り収容可能数: %d名\n", MAX\_STUDENTS - student\_count);  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 配列の動的管理:  
 - 要素の追加・削除  
 - 配列の圧縮（詰める処理）  
 - 容量管理  
  
2. 文字列配列の操作:  
 - strcpy()による安全なコピー  
 - strcmp()による比較  
 - strstr()による部分一致検索  
  
3. エラーハンドリング:  
 - 境界チェック  
 - 重複チェック  
 - 戻り値による成功/失敗の通知  
  
4. 実用的な機能:  
 - 検索機能（完全一致・部分一致）  
 - ソート機能  
 - 統計情報の表示  
  
5. データ構造の設計:  
 - 配列サイズの管理  
 - インデックスの有効性確認  
 - 一貫性のあるAPI設計  
  
注意点:  
- バッファオーバーフローの防止  
- 配列の境界チェック  
- メモリ効率的な削除処理  
- ユーザビリティの向上  
\*/

### ex8\_3\_string\_search.c

/\*  
 \* ファイル名: ex8\_3\_string\_search.c  
 \* 演習8-3: 高度な文字列検索  
 \* 説明: 文字列配列から特定の文字列を検索する高度な機能  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
  
#define MAX\_STRINGS 20  
#define MAX\_LENGTH 100  
  
/\* 完全一致検索 \*/  
int exact\_search(char strings[][MAX\_LENGTH], int count, const char \*target)  
{  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (strcmp(strings[i], target) == 0) {  
 return i; /\* 見つかった位置 \*/  
 }  
 }  
   
 return -1; /\* 見つからない \*/  
}  
  
/\* 大文字小文字を区別しない検索 \*/  
int case\_insensitive\_search(char strings[][MAX\_LENGTH], int count, const char \*target)  
{  
 int i, j;  
 char str\_lower[MAX\_LENGTH];  
 char target\_lower[MAX\_LENGTH];  
   
 /\* ターゲットを小文字に変換 \*/  
 for (j = 0; target[j] != '\0' && j < MAX\_LENGTH - 1; j++) {  
 target\_lower[j] = tolower((unsigned char)target[j]);  
 }  
 target\_lower[j] = '\0';  
   
 for (i = 0; i < count; i++) {  
 /\* 比較対象の文字列を小文字に変換 \*/  
 for (j = 0; strings[i][j] != '\0' && j < MAX\_LENGTH - 1; j++) {  
 str\_lower[j] = tolower((unsigned char)strings[i][j]);  
 }  
 str\_lower[j] = '\0';  
   
 if (strcmp(str\_lower, target\_lower) == 0) {  
 return i;  
 }  
 }  
   
 return -1;  
}  
  
/\* 部分一致検索（すべての該当項目を表示） \*/  
int partial\_search(char strings[][MAX\_LENGTH], int count, const char \*partial,   
 int results[], int max\_results)  
{  
 int i;  
 int found\_count = 0;  
   
 for (i = 0; i < count && found\_count < max\_results; i++) {  
 if (strstr(strings[i], partial) != NULL) {  
 results[found\_count] = i;  
 found\_count++;  
 }  
 }  
   
 return found\_count;  
}  
  
/\* 前方一致検索 \*/  
int prefix\_search(char strings[][MAX\_LENGTH], int count, const char \*prefix,  
 int results[], int max\_results)  
{  
 int i;  
 int found\_count = 0;  
 int prefix\_len = strlen(prefix);  
   
 for (i = 0; i < count && found\_count < max\_results; i++) {  
 if (strncmp(strings[i], prefix, prefix\_len) == 0) {  
 results[found\_count] = i;  
 found\_count++;  
 }  
 }  
   
 return found\_count;  
}  
  
/\* 後方一致検索 \*/  
int suffix\_search(char strings[][MAX\_LENGTH], int count, const char \*suffix,  
 int results[], int max\_results)  
{  
 int i;  
 int found\_count = 0;  
 int suffix\_len = strlen(suffix);  
   
 for (i = 0; i < count && found\_count < max\_results; i++) {  
 int str\_len = strlen(strings[i]);  
   
 if (str\_len >= suffix\_len) {  
 if (strcmp(strings[i] + str\_len - suffix\_len, suffix) == 0) {  
 results[found\_count] = i;  
 found\_count++;  
 }  
 }  
 }  
   
 return found\_count;  
}  
  
/\* 正規表現風のワイルドカード検索（簡易版） \*/  
int wildcard\_search(const char \*str, const char \*pattern)  
{  
 const char \*s = str;  
 const char \*p = pattern;  
   
 while (\*p != '\0') {  
 if (\*p == '\*') {  
 /\* ワイルドカード：0文字以上の任意の文字 \*/  
 p++; /\* '\*'をスキップ \*/  
   
 if (\*p == '\0') {  
 return 1; /\* パターンの最後が'\*'なら一致 \*/  
 }  
   
 /\* 次の文字が一致する位置を探す \*/  
 while (\*s != '\0') {  
 if (wildcard\_search(s, p)) {  
 return 1;  
 }  
 s++;  
 }  
 return 0;  
 } else if (\*p == '?') {  
 /\* 任意の1文字 \*/  
 if (\*s == '\0') {  
 return 0; /\* 文字列が足りない \*/  
 }  
 s++;  
 p++;  
 } else {  
 /\* 通常文字 \*/  
 if (\*s != \*p) {  
 return 0;  
 }  
 s++;  
 p++;  
 }  
 }  
   
 return (\*s == '\0'); /\* 両方とも終端に達していれば一致 \*/  
}  
  
/\* ワイルドカード検索（配列用） \*/  
int wildcard\_search\_array(char strings[][MAX\_LENGTH], int count, const char \*pattern,  
 int results[], int max\_results)  
{  
 int i;  
 int found\_count = 0;  
   
 for (i = 0; i < count && found\_count < max\_results; i++) {  
 if (wildcard\_search(strings[i], pattern)) {  
 results[found\_count] = i;  
 found\_count++;  
 }  
 }  
   
 return found\_count;  
}  
  
/\* 検索結果の表示 \*/  
void display\_search\_results(char strings[][MAX\_LENGTH], int results[], int count,  
 const char \*search\_type, const char \*query)  
{  
 int i;  
   
 printf("\n=== %s検索結果: \"%s\" ===\n", search\_type, query);  
   
 if (count == 0) {  
 printf("該当する項目はありません。\n");  
 return;  
 }  
   
 printf("%d件が見つかりました:\n", count);  
 for (i = 0; i < count; i++) {  
 printf(" %d. [%d] %s\n", i + 1, results[i] + 1, strings[results[i]]);  
 }  
}  
  
/\* 文字列の統計情報を表示 \*/  
void display\_statistics(char strings[][MAX\_LENGTH], int count, const char \*query)  
{  
 int i;  
 int total\_length = 0;  
 int min\_length = MAX\_LENGTH;  
 int max\_length = 0;  
 int query\_count = 0;  
   
 printf("\n=== 統計情報 ===\n");  
   
 for (i = 0; i < count; i++) {  
 int len = strlen(strings[i]);  
 total\_length += len;  
   
 if (len < min\_length) min\_length = len;  
 if (len > max\_length) max\_length = len;  
   
 if (strstr(strings[i], query) != NULL) {  
 query\_count++;  
 }  
 }  
   
 printf("総項目数: %d\n", count);  
 printf("平均文字数: %.1f\n", (double)total\_length / count);  
 printf("最短文字数: %d\n", min\_length);  
 printf("最長文字数: %d\n", max\_length);  
 printf("\"%s\"を含む項目数: %d (%.1f%%)\n",   
 query, query\_count, (double)query\_count \* 100 / count);  
}  
  
int main(void)  
{  
 char data[MAX\_STRINGS][MAX\_LENGTH] = {  
 "Apple", "Banana", "Cherry", "Date", "Elderberry",  
 "Fig", "Grape", "apple", "BANANA", "cherry pie",  
 "Apple juice", "grape fruit", "Date palm", "Elder tree",  
 "Fig leaf", "Cherry blossom", "Banana split", "Apple pie",  
 "Grape wine", "Date sugar"  
 };  
 int data\_count = 20;  
 int results[MAX\_STRINGS];  
 int found\_count;  
 int exact\_index;  
   
 printf("===== 高度な文字列検索システム =====\n");  
   
 /\* データ一覧の表示 \*/  
 printf("\n=== データ一覧 ===\n");  
 {  
 int i;  
 for (i = 0; i < data\_count; i++) {  
 printf("%2d. %s\n", i + 1, data[i]);  
 }  
 }  
   
 /\* 1. 完全一致検索 \*/  
 printf("\n1. 完全一致検索:\n");  
 exact\_index = exact\_search(data, data\_count, "Apple");  
 if (exact\_index != -1) {  
 printf("\"Apple\" は %d番目にあります: %s\n",   
 exact\_index + 1, data[exact\_index]);  
 }  
   
 exact\_index = exact\_search(data, data\_count, "apple");  
 if (exact\_index != -1) {  
 printf("\"apple\" は %d番目にあります: %s\n",   
 exact\_index + 1, data[exact\_index]);  
 }  
   
 /\* 2. 大文字小文字を区別しない検索 \*/  
 printf("\n2. 大文字小文字を区別しない検索:\n");  
 exact\_index = case\_insensitive\_search(data, data\_count, "APPLE");  
 if (exact\_index != -1) {  
 printf("\"APPLE\"（大小文字無視）は %d番目にあります: %s\n",   
 exact\_index + 1, data[exact\_index]);  
 }  
   
 /\* 3. 部分一致検索 \*/  
 found\_count = partial\_search(data, data\_count, "Apple", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "部分一致", "Apple");  
   
 /\* 4. 前方一致検索 \*/  
 found\_count = prefix\_search(data, data\_count, "Apple", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "前方一致", "Apple");  
   
 /\* 5. 後方一致検索 \*/  
 found\_count = suffix\_search(data, data\_count, "pie", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "後方一致", "pie");  
   
 /\* 6. ワイルドカード検索 \*/  
 printf("\n=== ワイルドカード検索のテスト ===\n");  
   
 /\* \*ple パターン \*/  
 found\_count = wildcard\_search\_array(data, data\_count, "\*ple", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "ワイルドカード", "\*ple");  
   
 /\* ?ate パターン \*/  
 found\_count = wildcard\_search\_array(data, data\_count, "?ate", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "ワイルドカード", "?ate");  
   
 /\* \*e\* パターン \*/  
 found\_count = wildcard\_search\_array(data, data\_count, "\*e\*", results, MAX\_STRINGS);  
 display\_search\_results(data, results, found\_count, "ワイルドカード", "\*e\*");  
   
 /\* 7. 複合検索の例 \*/  
 printf("\n=== 複合検索の例 ===\n");  
 printf("「Apple」で始まる項目:\n");  
 found\_count = prefix\_search(data, data\_count, "Apple", results, MAX\_STRINGS);  
 {  
 int i;  
 for (i = 0; i < found\_count; i++) {  
 printf(" %s\n", data[results[i]]);  
 }  
 }  
   
 /\* 8. 統計情報 \*/  
 display\_statistics(data, data\_count, "e");  
   
 /\* 9. 対話的検索の例 \*/  
 printf("\n=== 対話的検索のデモ ===\n");  
 {  
 char queries[][20] = {"Banana", "tree", "ju\*", "????"};  
 int i;  
   
 for (i = 0; i < 4; i++) {  
 printf("\nクエリ: \"%s\"\n", queries[i]);  
   
 /\* ワイルドカード文字が含まれているかチェック \*/  
 if (strchr(queries[i], '\*') != NULL || strchr(queries[i], '?') != NULL) {  
 found\_count = wildcard\_search\_array(data, data\_count, queries[i],   
 results, MAX\_STRINGS);  
 printf("ワイルドカード検索: %d件\n", found\_count);  
 } else {  
 found\_count = partial\_search(data, data\_count, queries[i],   
 results, MAX\_STRINGS);  
 printf("部分一致検索: %d件\n", found\_count);  
 }  
   
 if (found\_count > 0 && found\_count <= 3) {  
 int j;  
 for (j = 0; j < found\_count; j++) {  
 printf(" - %s\n", data[results[j]]);  
 }  
 }  
 }  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 多様な検索アルゴリズム:  
 - 完全一致・部分一致・前方一致・後方一致  
 - 大文字小文字を区別しない検索  
 - ワイルドカード検索  
  
2. 効率的な実装:  
 - 結果配列による複数結果の処理  
 - 早期終了による最適化  
 - メモリ効率的な検索  
  
3. 実用的な機能:  
 - 統計情報の表示  
 - 検索結果の整理  
 - エラーハンドリング  
  
4. アルゴリズムの応用:  
 - パターンマッチング  
 - 文字列処理の最適化  
 - 再帰的処理  
  
5. ユーザビリティ:  
 - 分かりやすい結果表示  
 - 複数の検索オプション  
 - 対話的な機能  
  
注意点:  
- バッファオーバーフローの防止  
- パターンマッチングの効率性  
- 文字コードの適切な処理  
- エッジケースの考慮  
\*/

# 第9章 関数

## 演習問題

## 基本課題

### 課題9-1: 基本的な関数作成

以下の関数を作成してください：

1. **数学関数**
   * 2つの整数の最大値を返す関数 int max(int a, int b)
   * 3つの整数の最小値を返す関数 int min(int a, int b, int c)
   * 数値が偶数かどうかを判定する関数 int is\_even(int n)
   * 数値が素数かどうかを判定する関数 int is\_prime(int n)
2. **文字・文字列関数**
   * 文字が英字かどうかを判定する関数 int is\_letter(char c)
   * 文字列の長さを計算する関数 int my\_strlen(const char str[])
   * 文字列内の特定文字を数える関数 int count\_char(const char str[], char ch)
3. **表示関数**
   * 指定した長さの線を描く関数 void draw\_line(int length, char ch)
   * 数値を指定した桁数で表示する関数 void print\_number\_padded(int num, int width)

### 課題9-2: 配列操作関数

以下の配列操作関数を作成してください：

1. **基本操作**
   * 配列の要素の合計を計算する関数 int array\_sum(int arr[], int size)
   * 配列の平均値を計算する関数 double array\_average(int arr[], int size)
   * 配列内の最大値のインデックスを返す関数 int find\_max\_index(int arr[], int size)
2. **検索・ソート**
   * 線形検索関数 int linear\_search(int arr[], int size, int target)
   * 配列を昇順にソートする関数 void sort\_array(int arr[], int size)
   * 配列の要素を逆順にする関数 void reverse\_array(int arr[], int size)
3. **配列統計**
   * 配列の最大値と最小値を同時に求める関数 void find\_min\_max(int arr[], int size, int \*min, int \*max)
   * 配列内の重複する要素の数を数える関数 int count\_duplicates(int arr[], int size)

### 課題9-3: 文字列処理関数

以下の文字列処理関数を作成してください：

1. **基本処理**
   * 文字列をコピーする関数 void my\_strcpy(char dest[], const char src[])
   * 文字列を連結する関数 void my\_strcat(char dest[], const char src[])
   * 文字列を比較する関数 int my\_strcmp(const char str1[], const char str2[])
2. **変換処理**
   * 文字列を大文字に変換する関数 void to\_uppercase(char str[])
   * 文字列を小文字に変換する関数 void to\_lowercase(char str[])
   * 文字列を逆順にする関数 void reverse\_string(char str[])
3. **解析処理**
   * 文字列内の単語数を数える関数 int count\_words(const char str[])
   * 文字列が回文かどうかを判定する関数 int is\_palindrome(const char str[])
   * 文字列内の母音の数を数える関数 int count\_vowels(const char str[])

## 中級課題

### 課題9-4: 複数戻り値を持つ関数

以下の関数を作成してください：

1. **時間計算**
   * 秒数を時分秒に変換する関数 void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds)
   * 日数から年月日に変換する関数（365日/年として） void days\_to\_ymd(int total\_days, int \*years, int \*months, int \*days)
2. **座標計算**
   * 2点間の距離と中点を計算する関数 void calculate\_line\_info(double x1, double y1, double x2, double y2, double \*distance, double \*mid\_x, double \*mid\_y)
   * 円の面積と周囲の長さを計算する関数 void calculate\_circle\_info(double radius, double \*area, double \*circumference)
3. **統計計算**
   * 配列の統計情報を計算する関数 void calculate\_statistics(int arr[], int size, int \*min, int \*max, double \*mean, double \*median)
   * 成績から評価を計算する関数 void calculate\_grade\_info(int scores[], int count, double \*average, char \*grade, int \*pass\_count)

### 課題9-5: エラーハンドリング付き関数

安全性を考慮した以下の関数を作成してください：

1. **安全な基本操作**
   * 安全な除算関数 int safe\_divide(double a, double b, double \*result)
   * 安全な配列アクセス関数 int safe\_array\_get(int arr[], int size, int index, int \*value)
   * 安全な文字列コピー関数 int safe\_strcpy(char dest[], int dest\_size, const char src[])
2. **範囲チェック付き関数**
   * 範囲指定付きランダム数生成関数 int random\_range(int min, int max, int \*result)
   * 配列の範囲チェック付き設定関数 int safe\_array\_set(int arr[], int size, int index, int value)
3. **バリデーション関数**
   * 文字列が数値として有効かチェックする関数 int is\_valid\_number(const char str[])
   * メールアドレスの基本形式をチェックする関数 int is\_valid\_email(const char email[])

### 課題9-6: 構造体を使った関数

以下の構造体を使った関数を作成してください：

typedef struct {  
 double x;  
 double y;  
} Point;  
  
typedef struct {  
 int id;  
 char name[50];  
 int age;  
 double score;  
} Student;  
  
typedef struct {  
 int year;  
 int month;  
 int day;  
} Date;

1. **座標操作**
   * 2点間の距離を計算する関数 double point\_distance(Point p1, Point p2)
   * 点を移動する関数 Point move\_point(Point p, double dx, double dy)
   * 複数の点の重心を計算する関数 Point calculate\_centroid(Point points[], int count)
2. **学生情報処理**
   * 学生情報を表示する関数 void print\_student(Student s)
   * 学生の成績を更新する関数 void update\_score(Student \*s, double new\_score)
   * 学生配列から最高得点者を見つける関数 Student find\_best\_student(Student students[], int count)
3. **日付操作**
   * 日付を表示する関数 void print\_date(Date d)
   * 日付の妥当性をチェックする関数 int is\_valid\_date(Date d)
   * 2つの日付の差を計算する関数 int date\_difference(Date d1, Date d2)

## 上級課題

### 課題9-7: 再帰関数

以下の再帰関数を作成してください：

1. **数学的再帰**
   * 階乗を計算する再帰関数 long factorial\_recursive(int n)
   * フィボナッチ数列を計算する再帰関数 int fibonacci\_recursive(int n)
   * ユークリッドの互除法による最大公約数を求める再帰関数 int gcd\_recursive(int a, int b)
2. **文字列再帰**
   * 文字列が回文かどうか再帰的に判定する関数 int is\_palindrome\_recursive(const char str[], int start, int end)
   * 文字列を再帰的に逆順にする関数 void reverse\_string\_recursive(char str[], int start, int end)
3. **配列再帰**
   * 配列の合計を再帰的に計算する関数 int array\_sum\_recursive(int arr[], int size)
   * 配列の最大値を再帰的に見つける関数 int find\_max\_recursive(int arr[], int size)

### 課題9-8: 高度な文字列処理

以下の高度な文字列処理関数を作成してください：

1. **文字列解析**
   * 文字列をトークンに分割する関数 int tokenize(char str[], char tokens[][50], char delimiter)
   * 文字列内の括弧の対応をチェックする関数 int check\_brackets(const char str[])
   * 文字列から数値を抽出する関数 int extract\_numbers(const char str[], int numbers[], int max\_count)
2. **パターンマッチング**
   * 簡単なワイルドカード（\*、?）パターンマッチング関数 int wildcard\_match(const char str[], const char pattern[])
   * 文字列の置換関数 int string\_replace(char str[], const char old\_substr[], const char new\_substr[])
3. **文字列フォーマット**
   * 文字列を指定幅で中央揃えする関数 void center\_string(char result[], const char str[], int width)
   * CSV形式の文字列を解析する関数 int parse\_csv\_line(const char line[], char fields[][100], int max\_fields)

### 課題9-9: ソートアルゴリズム関数

以下のソートアルゴリズムを関数として実装してください：

1. **基本ソート**
   * バブルソート void bubble\_sort(int arr[], int size)
   * 選択ソート void selection\_sort(int arr[], int size)
   * 挿入ソート void insertion\_sort(int arr[], int size)
2. **高速ソート**
   * マージソート void merge\_sort(int arr[], int left, int right)
   * クイックソート void quick\_sort(int arr[], int left, int right)
3. **特殊ソート**
   * 文字列配列のソート void sort\_strings(char strings[][100], int count)
   * 構造体配列のソート（複数キー対応）void sort\_students(Student students[], int count, int sort\_by)

## 挑戦課題

### 課題9-10: 関数ポインター

以下の関数ポインターを使った課題に取り組んでください：

1. **計算機関数**
   * 四則演算を関数ポインターで切り替える計算機 double calculator(double a, double b, double (\*operation)(double, double))
   * 配列に対する処理を関数ポインターで指定する関数 void process\_array(int arr[], int size, void (\*processor)(int\*))
2. **ソート関数の汎用化**
   * 比較関数を引数に取る汎用ソート関数 void generic\_sort(void \*arr, int size, int elem\_size, int (\*compare)(const void\*, const void\*))
3. **コールバック関数**
   * イベント処理システム void register\_callback(int event\_type, void (\*callback)(int))

### 課題9-11: メモリ効率を考慮した関数

メモリ使用量を最適化した関数を作成してください：

1. **動的メモリ管理**
   * 動的配列を管理する関数群
   * 文字列プールを管理する関数群
   * メモリリークを検出する関数
2. **効率的なアルゴリズム**
   * インプレース（元の配列内で処理）ソート関数
   * メモリ使用量を抑えた文字列処理関数
   * キャッシュ効率を考慮した配列処理関数

### 課題9-12: 総合プロジェクト

複数の関数を組み合わせた総合的なプロジェクトを作成してください：

1. **学生管理システム**
   * 学生情報の登録、検索、更新、削除機能
   * 成績統計の計算機能
   * データのファイル保存・読み込み機能
2. **テキスト解析ツール**
   * ファイルの読み込みと解析
   * 単語頻度の統計
   * 文字列パターンの検索
3. **数値計算ライブラリ**
   * 行列演算機能
   * 統計計算機能
   * 数値積分・微分機能

## 提出要件

各課題について、以下を提出してください：

1. **ソースコード**
   * 関数の実装（.cファイル）
   * ヘッダファイル（.hファイル）
   * テスト用のmain関数
2. **ドキュメント**
   * 各関数の仕様説明
   * 使用例とサンプル出力
   * エラーケースの説明
3. **テストケース**
   * 正常ケースのテスト
   * 境界値のテスト
   * エラーケースのテスト

## 評価ポイント

* **正確性**: 仕様通りに動作するか
* **安全性**: エラーハンドリングが適切か
* **効率性**: アルゴリズムの計算量は適切か
* **可読性**: コードが理解しやすいか
* **再利用性**: 他のプログラムでも使えるか

## 難易度別の推奨学習順序

### 初学者向け（基本課題から開始）

1. 課題9-1: 基本的な関数作成
2. 課題9-2: 配列操作関数
3. 課題9-3: 文字列処理関数

### 中級者向け（中級課題に挑戦）

1. 課題9-4: 複数戻り値を持つ関数
2. 課題9-5: エラーハンドリング付き関数
3. 課題9-6: 構造体を使った関数

### 上級者向け（上級・挑戦課題）

1. 課題9-7: 再帰関数
2. 課題9-8: 高度な文字列処理
3. 課題9-9: ソートアルゴリズム関数
4. 課題9-10: 関数ポインター
5. 課題9-11: メモリ効率を考慮した関数
6. 課題9-12: 総合プロジェクト

## 学習のヒント

1. **段階的実装**
   * 最初は基本機能のみ実装
   * 動作確認後に機能を追加
   * テストを頻繁に実行
2. **エラーハンドリング**
   * NULLポインターのチェック
   * 配列の境界チェック
   * 無効な引数の処理
3. **関数設計の原則**
   * 単一責任の原則
   * 適切な関数名と引数名
   * const修飾子の活用
4. **テスト駆動開発**
   * 関数作成前にテストケースを考える
   * 境界値のテスト
   * エラーケースのテスト
5. **コードレビュー**
   * 他の人に読んでもらう
   * 改善点を見つける
   * ベストプラクティスを学ぶ

## 参考資料

* C言語関数リファレンス
* アルゴリズムとデータ構造の教科書
* セキュアプログラミングガイド
* 関数型プログラミングの考え方

## 実装例の構成

各課題の実装では以下の構成を推奨します：

// ヘッダーファイル（functions.h）  
// - 関数プロトタイプ  
// - 定数定義  
// - 構造体定義  
  
// 実装ファイル（functions.c）  
// - 関数の実装  
// - 静的関数（内部使用）  
// - エラーハンドリング  
  
// テストファイル（test.c）  
// - main関数  
// - 各関数のテスト  
// - 結果の検証

## 発展的な学習内容

1. **可変長引数**
   * va\_list、va\_start、va\_endの使用
   * printfライクな関数の作成
2. **関数ポインター**
   * コールバック関数の実装
   * 関数テーブルの作成
3. **マクロ関数**
   * 関数ライクマクロの作成
   * 条件付きコンパイル
4. **インライン関数**
   * パフォーマンスの最適化
   * 適切な使用場面

このチュートリアルを通じて、C言語の関数について体系的に学習し、実践的なプログラミングスキルを身に付けてください。

## 解答例

この章では、関数の基本から応用まで、様々な演習問題の解答例を提供しています。

## 解答ファイル一覧

### 演習9-1: 基本的な関数作成

素数判定や簡単な計算を行う基本的な関数群の実装例です。

* <ex9_1_prime_functions.c> - C90準拠版
* <ex9_1_prime_functions_c99.c> - C99準拠版（bool型、//コメント使用）

**学習ポイント:** - 関数の基本的な定義と呼び出し - 引数と戻り値の扱い方 - ループと条件分岐の組み合わせ

### 演習9-2: 配列操作関数

配列に対する各種操作（合計、平均、ソート等）を行う関数群の実装例です。

* <ex9_2_array_operations.c> - C90準拠版
* <ex9_2_array_operations_c99.c> - C99準拠版（可変長配列、inline関数使用）

**学習ポイント:** - 配列を引数として受け取る方法 - ポインターを使った配列操作 - 動的メモリ割り当て（calloc/free）

### 演習9-3: 文字列処理関数

文字列の操作（コピー、連結、変換等）を行う関数群の実装例です。

* <ex9_3_string_processing.c> - C90準拠版
* <ex9_3_string_processing_c99.c> - C99準拠版（bool型、inline関数使用）

**学習ポイント:** - 文字配列としての文字列操作 - 標準ライブラリ関数の再実装 - 文字列の解析と変換

### 演習9-4: 複数戻り値を持つ関数

ポインター引数を使って複数の値を返す関数群の実装例です。

* <ex9_4_multiple_returns.c> - C90準拠版
* <ex9_4_multiple_returns_c99.c> - C99準拠版（構造体戻り値、複合リテラル使用）

**学習ポイント:** - ポインター引数による複数値の返却 - 構造体を使った複数値のまとめ方 - 数学的計算と座標変換

### 演習9-5: エラーハンドリング付き関数

エラー処理を含む堅牢な関数群の実装例です。

* <ex9_5_error_handling.c> - C90準拠版
* <ex9_5_error_handling_c99.c> - C99準拠版（enum型、可変長引数マクロ使用）

**学習ポイント:** - エラーコードの設計と管理 - 安全な数値計算（オーバーフロー対策） - 境界チェックとNULLポインターチェック

### 演習9-6: 構造体を使った関数

構造体を引数や戻り値として使用する関数群の実装例です。

* <ex9_6_struct_functions.c> - C90準拠版
* <ex9_6_struct_functions_c99.c> - C99準拠版（designated initializer、複合リテラル使用）

**学習ポイント:** - 構造体の値渡しと参照渡し - 構造体を返す関数の実装 - 複雑なデータ構造の操作

## コンパイルと実行

各解答例は以下のようにコンパイルできます：

# C90版のコンパイル例  
gcc -Wall -Wextra -pedantic -std=c90 ex9\_1\_prime\_functions.c -o ex9\_1\_prime  
  
# C99版のコンパイル例（数学関数を使う場合は-lmを追加）  
gcc -Wall -Wextra -pedantic -std=c99 ex9\_4\_multiple\_returns\_c99.c -lm -o ex9\_4\_c99  
  
# 実行  
./ex9\_1\_prime

## 学習のポイント

### 1. C90とC99の違い

各演習にはC90版とC99版の両方を用意しています。主な違いは：

* **変数宣言**: C90は関数の先頭、C99は使用箇所で宣言可能
* **コメント**: C90は/\* \*/のみ、C99は//も使用可能
* **bool型**: C90は整数で代用、C99はstdbool.hのbool型
* **その他**: C99では可変長配列、複合リテラル、designated initializerなど

### 2. 関数設計の原則

* **単一責任の原則**: 1つの関数は1つの機能に集中
* **適切な名前付け**: 関数名から機能が推測できる
* **エラー処理**: 異常系を考慮した堅牢な実装
* **const修飾子**: 変更しない引数にはconstを付ける

### 3. テストの重要性

各解答例にはテスト用のmain関数が含まれています：

* 正常系のテスト
* 異常系のテスト（エラーケース）
* 境界値のテスト
* 実行結果の表示

### 4. メモリ管理

動的メモリを使用する場合：

* malloc/callocで確保
* 使用後は必ずfreeで解放
* NULLチェックを忘れない
* メモリリークに注意

### 5. 移植性の考慮

* 標準ライブラリのみを使用
* プラットフォーム依存のコードを避ける
* 適切な型の使用（size\_t、int64\_tなど）

## 発展的な学習

これらの基本的な実装を理解したら、以下に挑戦してみましょう：

1. **最適化**: より効率的なアルゴリズムの実装
2. **拡張**: 新しい機能の追加
3. **汎用化**: より汎用的な関数の設計
4. **エラー処理の改善**: より詳細なエラー情報の提供

## 注意事項

* 実装例はあくまで一例です。他の実装方法も検討してみましょう
* 実際のプロジェクトでは、標準ライブラリの使用を推奨します
* セキュリティを考慮した実装を心がけましょう ### ex9\_1\_prime\_functions.c

/\*  
 \* 演習解答例: 素数判定関数とその活用  
 \*  
 \* このプログラムは、素数判定を行う関数を作成し、  
 \* それを使って様々な処理を行います。  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
int is\_prime(int n);  
void print\_primes\_in\_range(int start, int end);  
int count\_primes\_in\_range(int start, int end);  
int nth\_prime(int n);  
  
/\* 素数判定を行う関数 \*/  
int is\_prime(int n)  
{  
 int i;  
  
 /\* 1以下は素数ではない \*/  
 if (n <= 1)  
 {  
 return 0;  
 }  
  
 /\* 2は素数 \*/  
 if (n == 2)  
 {  
 return 1;  
 }  
  
 /\* 偶数は2以外素数ではない \*/  
 if (n % 2 == 0)  
 {  
 return 0;  
 }  
  
 /\* 3からsqrt(n)まで奇数のみをチェック \*/  
 for (i = 3; i \* i <= n; i += 2)  
 {  
 if (n % i == 0)  
 {  
 return 0; /\* 約数が見つかった \*/  
 }  
 }  
  
 return 1; /\* 素数 \*/  
}  
  
/\* 指定した範囲の素数を表示する関数 \*/  
void print\_primes\_in\_range(int start, int end)  
{  
 int i;  
 int count = 0;  
  
 printf("%dから%dまでの素数:\n", start, end);  
  
 for (i = start; i <= end; i++)  
 {  
 if (is\_prime(i))  
 {  
 printf("%d ", i);  
 count++;  
  
 /\* 10個ごとに改行 \*/  
 if (count % 10 == 0)  
 {  
 printf("\n");  
 }  
 }  
 }  
  
 if (count % 10 != 0)  
 {  
 printf("\n");  
 }  
  
 printf("見つかった素数の個数: %d\n", count);  
}  
  
/\* 指定した範囲の素数の個数を数える関数 \*/  
int count\_primes\_in\_range(int start, int end)  
{  
 int i;  
 int count = 0;  
  
 for (i = start; i <= end; i++)  
 {  
 if (is\_prime(i))  
 {  
 count++;  
 }  
 }  
  
 return count;  
}  
  
/\* n番目の素数を求める関数 \*/  
int nth\_prime(int n)  
{  
 int count = 0;  
 int num = 2;  
  
 if (n <= 0)  
 {  
 return -1; /\* 無効な入力 \*/  
 }  
  
 while (count < n)  
 {  
 if (is\_prime(num))  
 {  
 count++;  
 if (count == n)  
 {  
 return num;  
 }  
 }  
 num++;  
 }  
  
 return -1; /\* ここには到達しないはず \*/  
}  
  
int main(void)  
{  
 printf("=== 素数判定プログラム（デモ版）===\n\n");  
  
 /\* デモ1: 単一の数の素数判定 \*/  
 printf("1. 単一の数の素数判定\n");  
 int test\_numbers[] = {2, 3, 4, 17, 18, 97, 100, 101};  
 int num\_tests = sizeof(test\_numbers) / sizeof(test\_numbers[0]);  
 int i;  
   
 for (i = 0; i < num\_tests; i++)  
 {  
 int number = test\_numbers[i];  
 printf(" %d -> ", number);  
 if (is\_prime(number))  
 {  
 printf("素数です\n");  
 }  
 else  
 {  
 printf("素数ではありません\n");  
 }  
 }  
 printf("\n");  
  
 /\* デモ2: 範囲内の素数を表示 \*/  
 printf("2. 範囲内の素数を表示\n");  
 int demo\_ranges[][2] = {{1, 20}, {30, 50}, {90, 110}};  
 int num\_ranges = sizeof(demo\_ranges) / sizeof(demo\_ranges[0]);  
   
 for (i = 0; i < num\_ranges; i++)  
 {  
 int start = demo\_ranges[i][0];  
 int end = demo\_ranges[i][1];  
 printf(" 範囲 %d から %d の素数:\n ", start, end);  
 print\_primes\_in\_range(start, end);  
 printf("\n");  
 }  
  
 /\* デモ3: 範囲内の素数の個数をカウント \*/  
 printf("3. 範囲内の素数の個数をカウント\n");  
 for (i = 0; i < num\_ranges; i++)  
 {  
 int start = demo\_ranges[i][0];  
 int end = demo\_ranges[i][1];  
 int count = count\_primes\_in\_range(start, end);  
 printf(" 範囲 %d から %d: %d個\n", start, end, count);  
 }  
 printf("\n");  
  
 /\* デモ4: n番目の素数を求める \*/  
 printf("4. n番目の素数を求める\n");  
 int nth\_tests[] = {1, 5, 10, 15, 20, 25};  
 int nth\_count = sizeof(nth\_tests) / sizeof(nth\_tests[0]);  
   
 for (i = 0; i < nth\_count; i++)  
 {  
 int n = nth\_tests[i];  
 printf(" %d番目の素数: %d\n", n, nth\_prime(n));  
 }  
 printf("\n");  
  
 /\* 追加デモ: 最初の30個の素数を表示 \*/  
 printf("5. 最初の30個の素数\n ");  
 for (i = 1; i <= 30; i++)  
 {  
 printf("%d ", nth\_prime(i));  
 if (i % 10 == 0) printf("\n ");  
 }  
 printf("\n\n");  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}

### ex9\_2\_array\_operations.c

/\*  
 \* 演習9-2: 配列操作関数 - 解答例  
 \*   
 \* 配列に対する各種操作を行う関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <stdlib.h> /\* calloc, free用 \*/  
  
/\* 関数プロトタイプ \*/  
/\* 基本操作 \*/  
int array\_sum(int arr[], int size);  
double array\_average(int arr[], int size);  
int find\_max\_index(int arr[], int size);  
  
/\* 検索・ソート \*/  
int linear\_search(int arr[], int size, int target);  
void sort\_array(int arr[], int size);  
void reverse\_array(int arr[], int size);  
  
/\* 配列統計 \*/  
void find\_min\_max(int arr[], int size, int \*min, int \*max);  
int count\_duplicates(int arr[], int size);  
  
/\* ヘルパー関数 \*/  
void print\_array(int arr[], int size);  
void swap(int \*a, int \*b);  
  
/\* 配列の要素の合計を計算する関数 \*/  
int array\_sum(int arr[], int size)  
{  
 int sum = 0;  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 sum += arr[i];  
 }  
   
 return sum;  
}  
  
/\* 配列の平均値を計算する関数 \*/  
double array\_average(int arr[], int size)  
{  
 if (size == 0)  
 {  
 return 0.0;  
 }  
   
 return (double)array\_sum(arr, size) / size;  
}  
  
/\* 配列内の最大値のインデックスを返す関数 \*/  
int find\_max\_index(int arr[], int size)  
{  
 int max\_index = 0;  
 int i;  
   
 if (size <= 0)  
 {  
 return -1; /\* エラー \*/  
 }  
   
 for (i = 1; i < size; i++)  
 {  
 if (arr[i] > arr[max\_index])  
 {  
 max\_index = i;  
 }  
 }  
   
 return max\_index;  
}  
  
/\* 線形検索関数（見つかったらインデックス、見つからなければ-1を返す） \*/  
int linear\_search(int arr[], int size, int target)  
{  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 if (arr[i] == target)  
 {  
 return i;  
 }  
 }  
   
 return -1; /\* 見つからなかった \*/  
}  
  
/\* 配列を昇順にソートする関数（バブルソート） \*/  
void sort\_array(int arr[], int size)  
{  
 int i, j;  
   
 for (i = 0; i < size - 1; i++)  
 {  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 swap(&arr[j], &arr[j + 1]);  
 }  
 }  
 }  
}  
  
/\* 配列の要素を逆順にする関数 \*/  
void reverse\_array(int arr[], int size)  
{  
 int i;  
   
 for (i = 0; i < size / 2; i++)  
 {  
 swap(&arr[i], &arr[size - 1 - i]);  
 }  
}  
  
/\* 配列の最大値と最小値を同時に求める関数 \*/  
void find\_min\_max(int arr[], int size, int \*min, int \*max)  
{  
 int i;  
   
 if (size <= 0 || min == NULL || max == NULL)  
 {  
 return;  
 }  
   
 \*min = arr[0];  
 \*max = arr[0];  
   
 for (i = 1; i < size; i++)  
 {  
 if (arr[i] < \*min)  
 {  
 \*min = arr[i];  
 }  
 if (arr[i] > \*max)  
 {  
 \*max = arr[i];  
 }  
 }  
}  
  
/\* 配列内の重複する要素の数を数える関数 \*/  
int count\_duplicates(int arr[], int size)  
{  
 int count = 0;  
 int i, j;  
 int \*counted;  
   
 if (size <= 1)  
 {  
 return 0;  
 }  
   
 /\* カウント済みフラグ配列を作成 \*/  
 counted = (int\*)calloc(size, sizeof(int));  
 if (counted == NULL)  
 {  
 return -1; /\* メモリ割り当てエラー \*/  
 }  
   
 for (i = 0; i < size - 1; i++)  
 {  
 if (counted[i])  
 {  
 continue; /\* 既にカウント済み \*/  
 }  
   
 int dup\_count = 0;  
 for (j = i + 1; j < size; j++)  
 {  
 if (arr[i] == arr[j])  
 {  
 dup\_count++;  
 counted[j] = 1;  
 }  
 }  
   
 if (dup\_count > 0)  
 {  
 count += dup\_count;  
 }  
 }  
   
 free(counted);  
 return count;  
}  
  
/\* 配列を表示するヘルパー関数 \*/  
void print\_array(int arr[], int size)  
{  
 int i;  
   
 printf("[");  
 for (i = 0; i < size; i++)  
 {  
 printf("%d", arr[i]);  
 if (i < size - 1)  
 {  
 printf(", ");  
 }  
 }  
 printf("]\n");  
}  
  
/\* 2つの要素を交換するヘルパー関数 \*/  
void swap(int \*a, int \*b)  
{  
 int temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 int test\_array[] = {5, 2, 8, 2, 9, 1, 5, 5, 3, 7};  
 int size = sizeof(test\_array) / sizeof(test\_array[0]);  
 int min, max;  
 int search\_target;  
 int result;  
   
 printf("=== 配列操作関数のテスト ===\n\n");  
   
 /\* 元の配列を表示 \*/  
 printf("元の配列: ");  
 print\_array(test\_array, size);  
 printf("\n");  
   
 /\* 基本操作のテスト \*/  
 printf("=== 基本操作 ===\n");  
 printf("合計: %d\n", array\_sum(test\_array, size));  
 printf("平均: %.2f\n", array\_average(test\_array, size));  
 printf("最大値のインデックス: %d (値: %d)\n",   
 find\_max\_index(test\_array, size),   
 test\_array[find\_max\_index(test\_array, size)]);  
 printf("\n");  
   
 /\* 検索のテスト \*/  
 printf("=== 線形検索 ===\n");  
 search\_target = 8;  
 result = linear\_search(test\_array, size, search\_target);  
 if (result != -1)  
 {  
 printf("%d はインデックス %d で見つかりました\n", search\_target, result);  
 }  
 else  
 {  
 printf("%d は見つかりませんでした\n", search\_target);  
 }  
   
 search\_target = 10;  
 result = linear\_search(test\_array, size, search\_target);  
 if (result != -1)  
 {  
 printf("%d はインデックス %d で見つかりました\n", search\_target, result);  
 }  
 else  
 {  
 printf("%d は見つかりませんでした\n", search\_target);  
 }  
 printf("\n");  
   
 /\* 統計情報のテスト \*/  
 printf("=== 統計情報 ===\n");  
 find\_min\_max(test\_array, size, &min, &max);  
 printf("最小値: %d, 最大値: %d\n", min, max);  
 printf("重複する要素の数: %d\n", count\_duplicates(test\_array, size));  
 printf("\n");  
   
 /\* 配列のコピーを作成（ソート・逆順用） \*/  
 int work\_array[10];  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 work\_array[i] = test\_array[i];  
 }  
   
 /\* ソートのテスト \*/  
 printf("=== ソート ===\n");  
 printf("ソート前: ");  
 print\_array(work\_array, size);  
 sort\_array(work\_array, size);  
 printf("ソート後: ");  
 print\_array(work\_array, size);  
 printf("\n");  
   
 /\* 逆順のテスト \*/  
 printf("=== 逆順 ===\n");  
 for (i = 0; i < size; i++)  
 {  
 work\_array[i] = test\_array[i];  
 }  
 printf("逆順前: ");  
 print\_array(work\_array, size);  
 reverse\_array(work\_array, size);  
 printf("逆順後: ");  
 print\_array(work\_array, size);  
 printf("\n");  
   
 /\* エッジケースのテスト \*/  
 printf("=== エッジケースのテスト ===\n");  
 int empty\_array[1];  
 int single\_element[] = {42};  
   
 printf("空配列の平均: %.2f\n", array\_average(empty\_array, 0));  
 printf("単一要素配列の最大値インデックス: %d\n",   
 find\_max\_index(single\_element, 1));  
 printf("単一要素配列の重複数: %d\n",   
 count\_duplicates(single\_element, 1));  
   
 return 0;  
}```  
  
### ex9\_3\_string\_processing.c  
  
```c  
/\*  
 \* 演習9-3: 文字列処理関数 - 解答例  
 \*   
 \* 文字列に対する各種処理を行う関数群を実装します。  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
/\* 基本処理 \*/  
void my\_strcpy(char dest[], const char src[]);  
void my\_strcat(char dest[], const char src[]);  
int my\_strcmp(const char str1[], const char str2[]);  
  
/\* 変換処理 \*/  
void to\_uppercase(char str[]);  
void to\_lowercase(char str[]);  
void reverse\_string(char str[]);  
  
/\* 解析処理 \*/  
int count\_words(const char str[]);  
int is\_palindrome(const char str[]);  
int count\_vowels(const char str[]);  
  
/\* ヘルパー関数 \*/  
int my\_strlen(const char str[]);  
int is\_space(char c);  
int is\_alpha(char c);  
int is\_vowel(char c);  
  
/\* 文字列の長さを計算する関数 \*/  
int my\_strlen(const char str[])  
{  
 int len = 0;  
   
 while (str[len] != '\0')  
 {  
 len++;  
 }  
   
 return len;  
}  
  
/\* 文字列をコピーする関数 \*/  
void my\_strcpy(char dest[], const char src[])  
{  
 int i = 0;  
   
 while (src[i] != '\0')  
 {  
 dest[i] = src[i];  
 i++;  
 }  
 dest[i] = '\0';  
}  
  
/\* 文字列を連結する関数 \*/  
void my\_strcat(char dest[], const char src[])  
{  
 int dest\_len = my\_strlen(dest);  
 int i = 0;  
   
 while (src[i] != '\0')  
 {  
 dest[dest\_len + i] = src[i];  
 i++;  
 }  
 dest[dest\_len + i] = '\0';  
}  
  
/\* 文字列を比較する関数 \*/  
int my\_strcmp(const char str1[], const char str2[])  
{  
 int i = 0;  
   
 while (str1[i] != '\0' && str2[i] != '\0')  
 {  
 if (str1[i] < str2[i])  
 {  
 return -1;  
 }  
 else if (str1[i] > str2[i])  
 {  
 return 1;  
 }  
 i++;  
 }  
   
 /\* 片方の文字列が終了した場合 \*/  
 if (str1[i] == '\0' && str2[i] == '\0')  
 {  
 return 0; /\* 同じ \*/  
 }  
 else if (str1[i] == '\0')  
 {  
 return -1; /\* str1が短い \*/  
 }  
 else  
 {  
 return 1; /\* str2が短い \*/  
 }  
}  
  
/\* 文字列を大文字に変換する関数 \*/  
void to\_uppercase(char str[])  
{  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (str[i] >= 'a' && str[i] <= 'z')  
 {  
 str[i] = str[i] - 'a' + 'A';  
 }  
 i++;  
 }  
}  
  
/\* 文字列を小文字に変換する関数 \*/  
void to\_lowercase(char str[])  
{  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (str[i] >= 'A' && str[i] <= 'Z')  
 {  
 str[i] = str[i] - 'A' + 'a';  
 }  
 i++;  
 }  
}  
  
/\* 文字列を逆順にする関数 \*/  
void reverse\_string(char str[])  
{  
 int len = my\_strlen(str);  
 int i;  
 char temp;  
   
 for (i = 0; i < len / 2; i++)  
 {  
 temp = str[i];  
 str[i] = str[len - 1 - i];  
 str[len - 1 - i] = temp;  
 }  
}  
  
/\* 文字列内の単語数を数える関数 \*/  
int count\_words(const char str[])  
{  
 int count = 0;  
 int in\_word = 0;  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (is\_space(str[i]))  
 {  
 in\_word = 0;  
 }  
 else  
 {  
 if (!in\_word)  
 {  
 count++;  
 in\_word = 1;  
 }  
 }  
 i++;  
 }  
   
 return count;  
}  
  
/\* 文字列が回文かどうかを判定する関数 \*/  
int is\_palindrome(const char str[])  
{  
 int len = my\_strlen(str);  
 int i, j;  
   
 /\* 空白を無視してチェック \*/  
 i = 0;  
 j = len - 1;  
   
 while (i < j)  
 {  
 /\* 左側の空白をスキップ \*/  
 while (i < j && is\_space(str[i]))  
 {  
 i++;  
 }  
   
 /\* 右側の空白をスキップ \*/  
 while (i < j && is\_space(str[j]))  
 {  
 j--;  
 }  
   
 /\* 大文字小文字を無視して比較 \*/  
 char left = str[i];  
 char right = str[j];  
   
 if (left >= 'A' && left <= 'Z')  
 {  
 left = left - 'A' + 'a';  
 }  
 if (right >= 'A' && right <= 'Z')  
 {  
 right = right - 'A' + 'a';  
 }  
   
 if (left != right)  
 {  
 return 0; /\* 回文ではない \*/  
 }  
   
 i++;  
 j--;  
 }  
   
 return 1; /\* 回文 \*/  
}  
  
/\* 文字列内の母音の数を数える関数 \*/  
int count\_vowels(const char str[])  
{  
 int count = 0;  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 if (is\_vowel(str[i]))  
 {  
 count++;  
 }  
 i++;  
 }  
   
 return count;  
}  
  
/\* 文字が空白かどうかを判定するヘルパー関数 \*/  
int is\_space(char c)  
{  
 return c == ' ' || c == '\t' || c == '\n';  
}  
  
/\* 文字がアルファベットかどうかを判定するヘルパー関数 \*/  
int is\_alpha(char c)  
{  
 return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z');  
}  
  
/\* 文字が母音かどうかを判定するヘルパー関数 \*/  
int is\_vowel(char c)  
{  
 char lower = c;  
   
 if (c >= 'A' && c <= 'Z')  
 {  
 lower = c - 'A' + 'a';  
 }  
   
 return lower == 'a' || lower == 'e' || lower == 'i' ||   
 lower == 'o' || lower == 'u';  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 char str1[100] = "Hello, World!";  
 char str2[100] = "C Programming";  
 char str3[100] = "";  
 char str4[100] = "A man a plan a canal Panama";  
 char str5[100] = "race car";  
 char buffer[200];  
   
 printf("=== 文字列処理関数のテスト ===\n\n");  
   
 /\* 基本処理のテスト \*/  
 printf("=== 基本処理 ===\n");  
 printf("str1: \"%s\"\n", str1);  
 printf("str2: \"%s\"\n", str2);  
 printf("strlen(str1): %d\n", my\_strlen(str1));  
 printf("strlen(str2): %d\n", my\_strlen(str2));  
   
 /\* strcpyのテスト \*/  
 my\_strcpy(str3, str1);  
 printf("\nstrcpy(str3, str1)後: \"%s\"\n", str3);  
   
 /\* strcatのテスト \*/  
 my\_strcpy(buffer, str1);  
 my\_strcat(buffer, " ");  
 my\_strcat(buffer, str2);  
 printf("strcat結果: \"%s\"\n", buffer);  
   
 /\* strcmpのテスト \*/  
 printf("\nstrcmp(str1, str2): %d\n", my\_strcmp(str1, str2));  
 printf("strcmp(str1, str1): %d\n", my\_strcmp(str1, str1));  
 printf("strcmp(\"ABC\", \"ABD\"): %d\n", my\_strcmp("ABC", "ABD"));  
 printf("\n");  
   
 /\* 変換処理のテスト \*/  
 printf("=== 変換処理 ===\n");  
 char test\_str[100];  
   
 /\* 大文字変換 \*/  
 my\_strcpy(test\_str, "Hello World 123!");  
 printf("元の文字列: \"%s\"\n", test\_str);  
 to\_uppercase(test\_str);  
 printf("大文字変換: \"%s\"\n", test\_str);  
   
 /\* 小文字変換 \*/  
 to\_lowercase(test\_str);  
 printf("小文字変換: \"%s\"\n", test\_str);  
   
 /\* 逆順 \*/  
 my\_strcpy(test\_str, "abcdef");  
 printf("\n元の文字列: \"%s\"\n", test\_str);  
 reverse\_string(test\_str);  
 printf("逆順: \"%s\"\n", test\_str);  
 printf("\n");  
   
 /\* 解析処理のテスト \*/  
 printf("=== 解析処理 ===\n");  
   
 /\* 単語数カウント \*/  
 char sentence1[] = "The quick brown fox jumps over the lazy dog";  
 char sentence2[] = " One two three ";  
 char sentence3[] = "";  
   
 printf("\"%s\"\n単語数: %d\n\n", sentence1, count\_words(sentence1));  
 printf("\"%s\"\n単語数: %d\n\n", sentence2, count\_words(sentence2));  
 printf("\"%s\"\n単語数: %d\n\n", sentence3, count\_words(sentence3));  
   
 /\* 回文判定 \*/  
 printf("回文判定:\n");  
 printf("\"%s\": %s\n", str4, is\_palindrome(str4) ? "回文" : "回文ではない");  
 printf("\"%s\": %s\n", str5, is\_palindrome(str5) ? "回文" : "回文ではない");  
 printf("\"%s\": %s\n", str1, is\_palindrome(str1) ? "回文" : "回文ではない");  
   
 /\* 母音カウント \*/  
 printf("\n母音の数:\n");  
 printf("\"%s\": %d個\n", str1, count\_vowels(str1));  
 printf("\"%s\": %d個\n", str2, count\_vowels(str2));  
 printf("\"%s\": %d個\n", "AEIOU aeiou", count\_vowels("AEIOU aeiou"));  
   
 return 0;  
}```  
  
### ex9\_4\_multiple\_returns.c  
  
```c  
/\*  
 \* 演習9-4: 複数戻り値を持つ関数 - 解答例  
 \*   
 \* ポインター引数を使って複数の値を返す関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <math.h>  
  
/\* 関数プロトタイプ \*/  
/\* 時間変換 \*/  
void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds);  
void time\_add(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*result\_h, int \*result\_m, int \*result\_s);  
void time\_diff(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*diff\_h, int \*diff\_m, int \*diff\_s);  
  
/\* 座標計算 \*/  
void polar\_to\_cartesian(double r, double theta, double \*x, double \*y);  
void cartesian\_to\_polar(double x, double y, double \*r, double \*theta);  
void midpoint(double x1, double y1, double x2, double y2, double \*mid\_x, double \*mid\_y);  
  
/\* 統計計算 \*/  
void basic\_statistics(double data[], int size, double \*mean, double \*variance, double \*std\_dev);  
void min\_max\_range(double data[], int size, double \*min, double \*max, double \*range);  
void percentiles(double sorted\_data[], int size, double \*q1, double \*median, double \*q3);  
  
/\* ヘルパー関数 \*/  
void sort\_doubles(double arr[], int size);  
void swap\_double(double \*a, double \*b);  
  
/\* 秒を時・分・秒に変換する関数 \*/  
void seconds\_to\_hms(int total\_seconds, int \*hours, int \*minutes, int \*seconds)  
{  
 if (hours == NULL || minutes == NULL || seconds == NULL)  
 {  
 return;  
 }  
   
 \*hours = total\_seconds / 3600;  
 total\_seconds %= 3600;  
 \*minutes = total\_seconds / 60;  
 \*seconds = total\_seconds % 60;  
}  
  
/\* 2つの時刻を加算する関数 \*/  
void time\_add(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*result\_h, int \*result\_m, int \*result\_s)  
{  
 if (result\_h == NULL || result\_m == NULL || result\_s == NULL)  
 {  
 return;  
 }  
   
 int total\_seconds = (h1 \* 3600 + m1 \* 60 + s1) + (h2 \* 3600 + m2 \* 60 + s2);  
 seconds\_to\_hms(total\_seconds, result\_h, result\_m, result\_s);  
}  
  
/\* 2つの時刻の差を計算する関数 \*/  
void time\_diff(int h1, int m1, int s1, int h2, int m2, int s2,   
 int \*diff\_h, int \*diff\_m, int \*diff\_s)  
{  
 if (diff\_h == NULL || diff\_m == NULL || diff\_s == NULL)  
 {  
 return;  
 }  
   
 int seconds1 = h1 \* 3600 + m1 \* 60 + s1;  
 int seconds2 = h2 \* 3600 + m2 \* 60 + s2;  
 int diff\_seconds = seconds1 - seconds2;  
   
 /\* 負の場合は正に変換 \*/  
 if (diff\_seconds < 0)  
 {  
 diff\_seconds = -diff\_seconds;  
 }  
   
 seconds\_to\_hms(diff\_seconds, diff\_h, diff\_m, diff\_s);  
}  
  
/\* 極座標を直交座標に変換する関数 \*/  
void polar\_to\_cartesian(double r, double theta, double \*x, double \*y)  
{  
 if (x == NULL || y == NULL)  
 {  
 return;  
 }  
   
 \*x = r \* cos(theta);  
 \*y = r \* sin(theta);  
}  
  
/\* 直交座標を極座標に変換する関数 \*/  
void cartesian\_to\_polar(double x, double y, double \*r, double \*theta)  
{  
 if (r == NULL || theta == NULL)  
 {  
 return;  
 }  
   
 \*r = sqrt(x \* x + y \* y);  
 \*theta = atan2(y, x);  
}  
  
/\* 2点の中点を求める関数 \*/  
void midpoint(double x1, double y1, double x2, double y2, double \*mid\_x, double \*mid\_y)  
{  
 if (mid\_x == NULL || mid\_y == NULL)  
 {  
 return;  
 }  
   
 \*mid\_x = (x1 + x2) / 2.0;  
 \*mid\_y = (y1 + y2) / 2.0;  
}  
  
/\* 基本統計量を計算する関数 \*/  
void basic\_statistics(double data[], int size, double \*mean, double \*variance, double \*std\_dev)  
{  
 if (mean == NULL || variance == NULL || std\_dev == NULL || size <= 0)  
 {  
 return;  
 }  
   
 /\* 平均値を計算 \*/  
 double sum = 0.0;  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 sum += data[i];  
 }  
 \*mean = sum / size;  
   
 /\* 分散を計算 \*/  
 double sum\_sq\_diff = 0.0;  
 for (i = 0; i < size; i++)  
 {  
 double diff = data[i] - \*mean;  
 sum\_sq\_diff += diff \* diff;  
 }  
 \*variance = sum\_sq\_diff / size;  
   
 /\* 標準偏差を計算 \*/  
 \*std\_dev = sqrt(\*variance);  
}  
  
/\* 最小値、最大値、範囲を求める関数 \*/  
void min\_max\_range(double data[], int size, double \*min, double \*max, double \*range)  
{  
 if (min == NULL || max == NULL || range == NULL || size <= 0)  
 {  
 return;  
 }  
   
 \*min = data[0];  
 \*max = data[0];  
   
 int i;  
 for (i = 1; i < size; i++)  
 {  
 if (data[i] < \*min)  
 {  
 \*min = data[i];  
 }  
 if (data[i] > \*max)  
 {  
 \*max = data[i];  
 }  
 }  
   
 \*range = \*max - \*min;  
}  
  
/\* パーセンタイル（四分位数）を計算する関数 \*/  
void percentiles(double sorted\_data[], int size, double \*q1, double \*median, double \*q3)  
{  
 if (q1 == NULL || median == NULL || q3 == NULL || size <= 0)  
 {  
 return;  
 }  
   
 /\* 中央値（第2四分位数） \*/  
 if (size % 2 == 0)  
 {  
 \*median = (sorted\_data[size/2 - 1] + sorted\_data[size/2]) / 2.0;  
 }  
 else  
 {  
 \*median = sorted\_data[size/2];  
 }  
   
 /\* 第1四分位数 \*/  
 int q1\_pos = size / 4;  
 if (size % 4 == 0)  
 {  
 \*q1 = (sorted\_data[q1\_pos - 1] + sorted\_data[q1\_pos]) / 2.0;  
 }  
 else  
 {  
 \*q1 = sorted\_data[q1\_pos];  
 }  
   
 /\* 第3四分位数 \*/  
 int q3\_pos = (3 \* size) / 4;  
 if ((3 \* size) % 4 == 0)  
 {  
 \*q3 = (sorted\_data[q3\_pos - 1] + sorted\_data[q3\_pos]) / 2.0;  
 }  
 else  
 {  
 \*q3 = sorted\_data[q3\_pos];  
 }  
}  
  
/\* 配列をソートするヘルパー関数 \*/  
void sort\_doubles(double arr[], int size)  
{  
 int i, j;  
 for (i = 0; i < size - 1; i++)  
 {  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 swap\_double(&arr[j], &arr[j + 1]);  
 }  
 }  
 }  
}  
  
/\* 2つの要素を交換するヘルパー関数 \*/  
void swap\_double(double \*a, double \*b)  
{  
 double temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 printf("=== 複数戻り値を持つ関数のテスト ===\n\n");  
   
 /\* 時間変換のテスト \*/  
 printf("=== 時間変換 ===\n");  
 int hours, minutes, seconds;  
   
 seconds\_to\_hms(7265, &hours, &minutes, &seconds);  
 printf("7265秒 = %d時間 %d分 %d秒\n", hours, minutes, seconds);  
   
 int h1 = 2, m1 = 45, s1 = 30;  
 int h2 = 1, m2 = 50, s2 = 40;  
 int result\_h, result\_m, result\_s;  
   
 time\_add(h1, m1, s1, h2, m2, s2, &result\_h, &result\_m, &result\_s);  
 printf("%d:%02d:%02d + %d:%02d:%02d = %d:%02d:%02d\n",   
 h1, m1, s1, h2, m2, s2, result\_h, result\_m, result\_s);  
   
 time\_diff(h1, m1, s1, h2, m2, s2, &result\_h, &result\_m, &result\_s);  
 printf("%d:%02d:%02d - %d:%02d:%02d = %d:%02d:%02d\n\n",   
 h1, m1, s1, h2, m2, s2, result\_h, result\_m, result\_s);  
   
 /\* 座標変換のテスト \*/  
 printf("=== 座標変換 ===\n");  
 double r = 5.0, theta = M\_PI / 4; /\* 45度 \*/  
 double x, y;  
   
 polar\_to\_cartesian(r, theta, &x, &y);  
 printf("極座標(r=%.2f, θ=%.2f) → 直交座標(x=%.2f, y=%.2f)\n",   
 r, theta, x, y);  
   
 double x2 = 3.0, y2 = 4.0;  
 double r2, theta2;  
 cartesian\_to\_polar(x2, y2, &r2, &theta2);  
 printf("直交座標(x=%.2f, y=%.2f) → 極座標(r=%.2f, θ=%.2f)\n",   
 x2, y2, r2, theta2);  
   
 double mid\_x, mid\_y;  
 midpoint(0, 0, 10, 10, &mid\_x, &mid\_y);  
 printf("(0, 0)と(10, 10)の中点: (%.2f, %.2f)\n\n", mid\_x, mid\_y);  
   
 /\* 統計計算のテスト \*/  
 printf("=== 統計計算 ===\n");  
 double data[] = {85.5, 90.2, 78.3, 92.1, 88.7, 79.5, 91.3, 86.4, 83.2, 87.6};  
 int size = sizeof(data) / sizeof(data[0]);  
 double mean, variance, std\_dev;  
   
 basic\_statistics(data, size, &mean, &variance, &std\_dev);  
 printf("データ: ");  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 printf("%.1f ", data[i]);  
 }  
 printf("\n");  
 printf("平均: %.2f, 分散: %.2f, 標準偏差: %.2f\n", mean, variance, std\_dev);  
   
 double min, max, range;  
 min\_max\_range(data, size, &min, &max, &range);  
 printf("最小値: %.2f, 最大値: %.2f, 範囲: %.2f\n", min, max, range);  
   
 /\* データをソートしてパーセンタイル計算 \*/  
 double sorted\_data[10];  
 for (i = 0; i < size; i++)  
 {  
 sorted\_data[i] = data[i];  
 }  
 sort\_doubles(sorted\_data, size);  
   
 double q1, median, q3;  
 percentiles(sorted\_data, size, &q1, &median, &q3);  
 printf("第1四分位数: %.2f, 中央値: %.2f, 第3四分位数: %.2f\n", q1, median, q3);  
   
 return 0;  
}```  
  
### ex9\_5\_error\_handling.c  
  
```c  
/\*  
 \* 演習9-5: エラーハンドリング付き関数 - 解答例  
 \*   
 \* エラー処理を含む堅牢な関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <limits.h>  
#include <errno.h>  
  
/\* エラーコード定義 \*/  
#define SUCCESS 0  
#define ERROR\_NULL\_POINTER -1  
#define ERROR\_INVALID\_ARGUMENT -2  
#define ERROR\_OVERFLOW -3  
#define ERROR\_UNDERFLOW -4  
#define ERROR\_MEMORY\_ALLOCATION -5  
#define ERROR\_FILE\_NOT\_FOUND -6  
#define ERROR\_PERMISSION\_DENIED -7  
#define ERROR\_DIVIDE\_BY\_ZERO -8  
#define ERROR\_OUT\_OF\_RANGE -9  
  
/\* グローバルエラー状態（オプション） \*/  
static int g\_last\_error = SUCCESS;  
static char g\_error\_message[256] = "";  
  
/\* エラー処理のヘルパー関数 \*/  
void set\_error(int error\_code, const char \*message);  
int get\_last\_error(void);  
const char\* get\_error\_message(void);  
void clear\_error(void);  
const char\* error\_to\_string(int error\_code);  
  
/\* 数値計算関数（エラーチェック付き） \*/  
int safe\_add(int a, int b, int \*result);  
int safe\_multiply(int a, int b, int \*result);  
int safe\_divide(int dividend, int divisor, int \*result, int \*remainder);  
int safe\_power(int base, int exponent, long \*result);  
  
/\* 配列操作関数（エラーチェック付き） \*/  
int safe\_array\_access(int arr[], int size, int index, int \*value);  
int safe\_array\_sum(int arr[], int size, long \*sum);  
int safe\_array\_average(int arr[], int size, double \*average);  
  
/\* 文字列操作関数（エラーチェック付き） \*/  
int safe\_string\_copy(char \*dest, size\_t dest\_size, const char \*src);  
int safe\_string\_concat(char \*dest, size\_t dest\_size, const char \*src);  
int safe\_string\_to\_int(const char \*str, int \*value);  
  
/\* ファイル操作関数（エラーチェック付き） \*/  
int safe\_file\_read\_line(FILE \*file, char \*buffer, size\_t buffer\_size);  
int safe\_file\_write\_int(FILE \*file, int value);  
  
/\* エラーを設定する関数 \*/  
void set\_error(int error\_code, const char \*message)  
{  
 g\_last\_error = error\_code;  
 if (message != NULL)  
 {  
 strncpy(g\_error\_message, message, sizeof(g\_error\_message) - 1);  
 g\_error\_message[sizeof(g\_error\_message) - 1] = '\0';  
 }  
 else  
 {  
 g\_error\_message[0] = '\0';  
 }  
}  
  
/\* 最後のエラーコードを取得する関数 \*/  
int get\_last\_error(void)  
{  
 return g\_last\_error;  
}  
  
/\* エラーメッセージを取得する関数 \*/  
const char\* get\_error\_message(void)  
{  
 return g\_error\_message;  
}  
  
/\* エラー状態をクリアする関数 \*/  
void clear\_error(void)  
{  
 g\_last\_error = SUCCESS;  
 g\_error\_message[0] = '\0';  
}  
  
/\* エラーコードを文字列に変換する関数 \*/  
const char\* error\_to\_string(int error\_code)  
{  
 switch (error\_code)  
 {  
 case SUCCESS:  
 return "成功";  
 case ERROR\_NULL\_POINTER:  
 return "NULLポインターエラー";  
 case ERROR\_INVALID\_ARGUMENT:  
 return "無効な引数";  
 case ERROR\_OVERFLOW:  
 return "オーバーフロー";  
 case ERROR\_UNDERFLOW:  
 return "アンダーフロー";  
 case ERROR\_MEMORY\_ALLOCATION:  
 return "メモリ割り当てエラー";  
 case ERROR\_FILE\_NOT\_FOUND:  
 return "ファイルが見つかりません";  
 case ERROR\_PERMISSION\_DENIED:  
 return "アクセス権限がありません";  
 case ERROR\_DIVIDE\_BY\_ZERO:  
 return "ゼロ除算";  
 case ERROR\_OUT\_OF\_RANGE:  
 return "範囲外";  
 default:  
 return "不明なエラー";  
 }  
}  
  
/\* 安全な加算関数 \*/  
int safe\_add(int a, int b, int \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 /\* オーバーフローチェック \*/  
 if ((b > 0 && a > INT\_MAX - b) || (b < 0 && a < INT\_MIN - b))  
 {  
 set\_error(ERROR\_OVERFLOW, "加算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = a + b;  
 return SUCCESS;  
}  
  
/\* 安全な乗算関数 \*/  
int safe\_multiply(int a, int b, int \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 /\* オーバーフローチェック \*/  
 if (a > 0 && b > 0 && a > INT\_MAX / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a < 0 && b < 0 && a < INT\_MAX / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a > 0 && b < 0 && b < INT\_MIN / a)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (a < 0 && b > 0 && a < INT\_MIN / b)  
 {  
 set\_error(ERROR\_OVERFLOW, "乗算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = a \* b;  
 return SUCCESS;  
}  
  
/\* 安全な除算関数 \*/  
int safe\_divide(int dividend, int divisor, int \*result, int \*remainder)  
{  
 if (result == NULL || remainder == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (divisor == 0)  
 {  
 set\_error(ERROR\_DIVIDE\_BY\_ZERO, "ゼロで除算");  
 return ERROR\_DIVIDE\_BY\_ZERO;  
 }  
   
 /\* INT\_MIN / -1 のオーバーフローチェック \*/  
 if (dividend == INT\_MIN && divisor == -1)  
 {  
 set\_error(ERROR\_OVERFLOW, "除算でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 \*result = dividend / divisor;  
 \*remainder = dividend % divisor;  
 return SUCCESS;  
}  
  
/\* 安全な累乗関数 \*/  
int safe\_power(int base, int exponent, long \*result)  
{  
 if (result == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "結果ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (exponent < 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "負の指数はサポートされていません");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 long power = 1;  
 long temp\_base = base;  
   
 while (exponent > 0)  
 {  
 if (exponent & 1)  
 {  
 /\* オーバーフローチェック \*/  
 if (temp\_base > 0 && power > LONG\_MAX / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 if (temp\_base < 0 && power < LONG\_MIN / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 power \*= temp\_base;  
 }  
   
 exponent >>= 1;  
 if (exponent > 0)  
 {  
 /\* temp\_base \* temp\_base のオーバーフローチェック \*/  
 if (temp\_base > 0 && temp\_base > LONG\_MAX / temp\_base)  
 {  
 set\_error(ERROR\_OVERFLOW, "累乗でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 temp\_base \*= temp\_base;  
 }  
 }  
   
 \*result = power;  
 return SUCCESS;  
}  
  
/\* 安全な配列アクセス関数 \*/  
int safe\_array\_access(int arr[], int size, int index, int \*value)  
{  
 if (arr == NULL || value == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または値ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 if (index < 0 || index >= size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "インデックスが範囲外");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 \*value = arr[index];  
 return SUCCESS;  
}  
  
/\* 安全な配列合計関数 \*/  
int safe\_array\_sum(int arr[], int size, long \*sum)  
{  
 if (arr == NULL || sum == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または合計ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 \*sum = 0;  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 /\* オーバーフローチェック \*/  
 if ((arr[i] > 0 && \*sum > LONG\_MAX - arr[i]) ||  
 (arr[i] < 0 && \*sum < LONG\_MIN - arr[i]))  
 {  
 set\_error(ERROR\_OVERFLOW, "配列合計でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
 \*sum += arr[i];  
 }  
   
 return SUCCESS;  
}  
  
/\* 安全な配列平均関数 \*/  
int safe\_array\_average(int arr[], int size, double \*average)  
{  
 if (arr == NULL || average == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "配列または平均ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (size <= 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "配列サイズが無効");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 long sum;  
 int result = safe\_array\_sum(arr, size, &sum);  
 if (result != SUCCESS)  
 {  
 return result;  
 }  
   
 \*average = (double)sum / size;  
 return SUCCESS;  
}  
  
/\* 安全な文字列コピー関数 \*/  
int safe\_string\_copy(char \*dest, size\_t dest\_size, const char \*src)  
{  
 if (dest == NULL || src == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "送信先または送信元がNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (dest\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 size\_t src\_len = strlen(src);  
 if (src\_len >= dest\_size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "バッファサイズ不足");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 strcpy(dest, src);  
 return SUCCESS;  
}  
  
/\* 安全な文字列連結関数 \*/  
int safe\_string\_concat(char \*dest, size\_t dest\_size, const char \*src)  
{  
 if (dest == NULL || src == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "送信先または送信元がNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (dest\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 size\_t dest\_len = strlen(dest);  
 size\_t src\_len = strlen(src);  
   
 if (dest\_len + src\_len >= dest\_size)  
 {  
 set\_error(ERROR\_OUT\_OF\_RANGE, "バッファサイズ不足");  
 return ERROR\_OUT\_OF\_RANGE;  
 }  
   
 strcat(dest, src);  
 return SUCCESS;  
}  
  
/\* 安全な文字列から整数への変換関数 \*/  
int safe\_string\_to\_int(const char \*str, int \*value)  
{  
 if (str == NULL || value == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "文字列または値ポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 char \*endptr;  
 errno = 0;  
 long result = strtol(str, &endptr, 10);  
   
 /\* 変換エラーチェック \*/  
 if (errno == ERANGE || result > INT\_MAX || result < INT\_MIN)  
 {  
 set\_error(ERROR\_OVERFLOW, "整数変換でオーバーフロー");  
 return ERROR\_OVERFLOW;  
 }  
   
 if (endptr == str || \*endptr != '\0')  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "無効な数値形式");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 \*value = (int)result;  
 return SUCCESS;  
}  
  
/\* 安全なファイル読み込み関数 \*/  
int safe\_file\_read\_line(FILE \*file, char \*buffer, size\_t buffer\_size)  
{  
 if (file == NULL || buffer == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "ファイルまたはバッファがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (buffer\_size == 0)  
 {  
 set\_error(ERROR\_INVALID\_ARGUMENT, "バッファサイズがゼロ");  
 return ERROR\_INVALID\_ARGUMENT;  
 }  
   
 if (fgets(buffer, buffer\_size, file) == NULL)  
 {  
 if (feof(file))  
 {  
 set\_error(ERROR\_UNDERFLOW, "ファイルの終端に達しました");  
 return ERROR\_UNDERFLOW;  
 }  
 else  
 {  
 set\_error(ERROR\_FILE\_NOT\_FOUND, "ファイル読み込みエラー");  
 return ERROR\_FILE\_NOT\_FOUND;  
 }  
 }  
   
 return SUCCESS;  
}  
  
/\* 安全なファイル書き込み関数 \*/  
int safe\_file\_write\_int(FILE \*file, int value)  
{  
 if (file == NULL)  
 {  
 set\_error(ERROR\_NULL\_POINTER, "ファイルポインターがNULL");  
 return ERROR\_NULL\_POINTER;  
 }  
   
 if (fprintf(file, "%d\n", value) < 0)  
 {  
 set\_error(ERROR\_PERMISSION\_DENIED, "ファイル書き込みエラー");  
 return ERROR\_PERMISSION\_DENIED;  
 }  
   
 return SUCCESS;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 int result;  
 int int\_result;  
 long long\_result;  
 double double\_result;  
 char buffer[100];  
   
 printf("=== エラーハンドリング付き関数のテスト ===\n\n");  
   
 /\* 数値計算のテスト \*/  
 printf("=== 数値計算 ===\n");  
   
 /\* 正常な加算 \*/  
 result = safe\_add(100, 200, &int\_result);  
 printf("100 + 200 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* オーバーフローする加算 \*/  
 result = safe\_add(INT\_MAX, 1, &int\_result);  
 printf("INT\_MAX + 1 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 除算のテスト \*/  
 int remainder;  
 result = safe\_divide(17, 5, &int\_result, &remainder);  
 printf("17 ÷ 5 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d 余り %d (成功)\n", int\_result, remainder);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* ゼロ除算 \*/  
 result = safe\_divide(10, 0, &int\_result, &remainder);  
 printf("10 ÷ 0 = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 累乗のテスト \*/  
 result = safe\_power(2, 10, &long\_result);  
 printf("2^10 = ");  
 if (result == SUCCESS)  
 {  
 printf("%ld (成功)\n", long\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
 printf("\n");  
   
 /\* 配列操作のテスト \*/  
 printf("=== 配列操作 ===\n");  
 int test\_array[] = {10, 20, 30, 40, 50};  
 int array\_size = 5;  
   
 /\* 正常な配列アクセス \*/  
 result = safe\_array\_access(test\_array, array\_size, 2, &int\_result);  
 printf("配列[2] = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* 範囲外アクセス \*/  
 result = safe\_array\_access(test\_array, array\_size, 10, &int\_result);  
 printf("配列[10] = ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 配列平均 \*/  
 result = safe\_array\_average(test\_array, array\_size, &double\_result);  
 printf("配列の平均 = ");  
 if (result == SUCCESS)  
 {  
 printf("%.2f (成功)\n", double\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
 printf("\n");  
   
 /\* 文字列操作のテスト \*/  
 printf("=== 文字列操作 ===\n");  
   
 /\* 正常な文字列コピー \*/  
 result = safe\_string\_copy(buffer, sizeof(buffer), "Hello, World!");  
 printf("文字列コピー: ");  
 if (result == SUCCESS)  
 {  
 printf("\"%s\" (成功)\n", buffer);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* バッファオーバーフロー防止 \*/  
 char small\_buffer[5];  
 result = safe\_string\_copy(small\_buffer, sizeof(small\_buffer), "This is too long");  
 printf("小さいバッファへのコピー: ");  
 if (result == SUCCESS)  
 {  
 printf("\"%s\" (成功)\n", small\_buffer);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 /\* 文字列から整数への変換 \*/  
 result = safe\_string\_to\_int("12345", &int\_result);  
 printf("\"12345\" を整数に変換: ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s\n", error\_to\_string(result));  
 }  
   
 /\* 無効な文字列 \*/  
 result = safe\_string\_to\_int("abc123", &int\_result);  
 printf("\"abc123\" を整数に変換: ");  
 if (result == SUCCESS)  
 {  
 printf("%d (成功)\n", int\_result);  
 }  
 else  
 {  
 printf("エラー: %s (%s)\n", error\_to\_string(result), get\_error\_message());  
 }  
   
 printf("\n");  
   
 /\* エラー処理のサマリー \*/  
 printf("=== エラー処理のサマリー ===\n");  
 printf("エラーハンドリングにより、以下の問題を防ぐことができました：\n");  
 printf("- オーバーフロー/アンダーフロー\n");  
 printf("- ゼロ除算\n");  
 printf("- 範囲外アクセス\n");  
 printf("- バッファオーバーフロー\n");  
 printf("- 無効な入力\n");  
 printf("- NULLポインター参照\n");  
   
 return 0;  
}```  
  
### ex9\_6\_struct\_functions.c  
  
```c  
/\*  
 \* 演習9-6: 構造体を使った関数 - 解答例  
 \*   
 \* 構造体を引数や戻り値として使用する関数群を実装します。  
 \*/  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
  
/\* 構造体定義 \*/  
/\* 商品情報 \*/  
typedef struct {  
 int id;  
 char name[50];  
 double price;  
 int stock;  
} Product;  
  
/\* 学生情報 \*/  
typedef struct {  
 int id;  
 char name[50];  
 double score;  
 char grade;  
} Student;  
  
/\* 日付 \*/  
typedef struct {  
 int year;  
 int month;  
 int day;  
} Date;  
  
/\* 座標点 \*/  
typedef struct {  
 double x;  
 double y;  
} Point;  
  
/\* 矩形 \*/  
typedef struct {  
 Point top\_left;  
 Point bottom\_right;  
} Rectangle;  
  
/\* 関数プロトタイプ \*/  
/\* 商品管理関数 \*/  
Product create\_product(int id, const char \*name, double price, int stock);  
void print\_product(Product p);  
double calculate\_total\_value(Product p);  
int compare\_products\_by\_price(Product p1, Product p2);  
void update\_stock(Product \*p, int quantity);  
void apply\_discount(Product \*p, double discount\_percent);  
  
/\* 学生管理関数 \*/  
Student create\_student(int id, const char \*name, double score);  
void print\_student(Student s);  
void update\_score(Student \*s, double new\_score);  
char calculate\_grade(double score);  
Student find\_best\_student(Student students[], int count);  
void sort\_students\_by\_score(Student students[], int count);  
  
/\* 日付操作関数 \*/  
Date create\_date(int year, int month, int day);  
void print\_date(Date d);  
int is\_valid\_date(Date d);  
int is\_leap\_year(int year);  
int days\_in\_month(int year, int month);  
int date\_difference(Date d1, Date d2);  
int compare\_dates(Date d1, Date d2);  
  
/\* 座標・図形関数 \*/  
Point create\_point(double x, double y);  
double point\_distance(Point p1, Point p2);  
Rectangle create\_rectangle(Point top\_left, Point bottom\_right);  
double rectangle\_area(Rectangle r);  
double rectangle\_perimeter(Rectangle r);  
int point\_in\_rectangle(Point p, Rectangle r);  
  
/\* 商品を作成する関数 \*/  
Product create\_product(int id, const char \*name, double price, int stock)  
{  
 Product p;  
 p.id = id;  
 strncpy(p.name, name, sizeof(p.name) - 1);  
 p.name[sizeof(p.name) - 1] = '\0';  
 p.price = price;  
 p.stock = stock;  
 return p;  
}  
  
/\* 商品情報を表示する関数 \*/  
void print\_product(Product p)  
{  
 printf("商品ID: %d\n", p.id);  
 printf("商品名: %s\n", p.name);  
 printf("価格: %.2f円\n", p.price);  
 printf("在庫: %d個\n", p.stock);  
}  
  
/\* 商品の総価値を計算する関数 \*/  
double calculate\_total\_value(Product p)  
{  
 return p.price \* p.stock;  
}  
  
/\* 商品を価格で比較する関数 \*/  
int compare\_products\_by\_price(Product p1, Product p2)  
{  
 if (p1.price < p2.price) return -1;  
 if (p1.price > p2.price) return 1;  
 return 0;  
}  
  
/\* 在庫を更新する関数 \*/  
void update\_stock(Product \*p, int quantity)  
{  
 if (p != NULL)  
 {  
 p->stock += quantity;  
 if (p->stock < 0)  
 {  
 p->stock = 0;  
 }  
 }  
}  
  
/\* 割引を適用する関数 \*/  
void apply\_discount(Product \*p, double discount\_percent)  
{  
 if (p != NULL && discount\_percent >= 0.0 && discount\_percent <= 100.0)  
 {  
 p->price \*= (1.0 - discount\_percent / 100.0);  
 }  
}  
  
/\* 学生を作成する関数 \*/  
Student create\_student(int id, const char \*name, double score)  
{  
 Student s;  
 s.id = id;  
 strncpy(s.name, name, sizeof(s.name) - 1);  
 s.name[sizeof(s.name) - 1] = '\0';  
 s.score = score;  
 s.grade = calculate\_grade(score);  
 return s;  
}  
  
/\* 学生情報を表示する関数 \*/  
void print\_student(Student s)  
{  
 printf("学生ID: %d\n", s.id);  
 printf("氏名: %s\n", s.name);  
 printf("得点: %.1f点\n", s.score);  
 printf("成績: %c\n", s.grade);  
}  
  
/\* 学生の成績を更新する関数 \*/  
void update\_score(Student \*s, double new\_score)  
{  
 if (s != NULL && new\_score >= 0.0 && new\_score <= 100.0)  
 {  
 s->score = new\_score;  
 s->grade = calculate\_grade(new\_score);  
 }  
}  
  
/\* 得点から成績を計算する関数 \*/  
char calculate\_grade(double score)  
{  
 if (score >= 90.0) return 'A';  
 if (score >= 80.0) return 'B';  
 if (score >= 70.0) return 'C';  
 if (score >= 60.0) return 'D';  
 return 'F';  
}  
  
/\* 学生配列から最高得点者を見つける関数 \*/  
Student find\_best\_student(Student students[], int count)  
{  
 Student best = students[0];  
 int i;  
   
 for (i = 1; i < count; i++)  
 {  
 if (students[i].score > best.score)  
 {  
 best = students[i];  
 }  
 }  
   
 return best;  
}  
  
/\* 学生を得点順にソートする関数 \*/  
void sort\_students\_by\_score(Student students[], int count)  
{  
 int i, j;  
 Student temp;  
   
 for (i = 0; i < count - 1; i++)  
 {  
 for (j = 0; j < count - 1 - i; j++)  
 {  
 if (students[j].score < students[j + 1].score)  
 {  
 temp = students[j];  
 students[j] = students[j + 1];  
 students[j + 1] = temp;  
 }  
 }  
 }  
}  
  
/\* 日付を作成する関数 \*/  
Date create\_date(int year, int month, int day)  
{  
 Date d;  
 d.year = year;  
 d.month = month;  
 d.day = day;  
 return d;  
}  
  
/\* 日付を表示する関数 \*/  
void print\_date(Date d)  
{  
 printf("%04d年%02d月%02d日", d.year, d.month, d.day);  
}  
  
/\* 日付の妥当性をチェックする関数 \*/  
int is\_valid\_date(Date d)  
{  
 if (d.year < 1 || d.year > 9999)  
 {  
 return 0;  
 }  
   
 if (d.month < 1 || d.month > 12)  
 {  
 return 0;  
 }  
   
 if (d.day < 1 || d.day > days\_in\_month(d.year, d.month))  
 {  
 return 0;  
 }  
   
 return 1;  
}  
  
/\* うるう年かどうかを判定する関数 \*/  
int is\_leap\_year(int year)  
{  
 return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);  
}  
  
/\* 月の日数を返す関数 \*/  
int days\_in\_month(int year, int month)  
{  
 int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
   
 if (month < 1 || month > 12)  
 {  
 return 0;  
 }  
   
 if (month == 2 && is\_leap\_year(year))  
 {  
 return 29;  
 }  
   
 return days[month - 1];  
}  
  
/\* 2つの日付の差を計算する関数（簡易版） \*/  
int date\_difference(Date d1, Date d2)  
{  
 /\* より正確な実装には、各月の日数を考慮した計算が必要 \*/  
 /\* ここでは簡易的に年と月を日数に変換 \*/  
 int days1 = d1.year \* 365 + d1.month \* 30 + d1.day;  
 int days2 = d2.year \* 365 + d2.month \* 30 + d2.day;  
   
 return days1 - days2;  
}  
  
/\* 日付を比較する関数 \*/  
int compare\_dates(Date d1, Date d2)  
{  
 if (d1.year != d2.year)  
 {  
 return d1.year - d2.year;  
 }  
   
 if (d1.month != d2.month)  
 {  
 return d1.month - d2.month;  
 }  
   
 return d1.day - d2.day;  
}  
  
/\* 点を作成する関数 \*/  
Point create\_point(double x, double y)  
{  
 Point p;  
 p.x = x;  
 p.y = y;  
 return p;  
}  
  
/\* 2点間の距離を計算する関数 \*/  
double point\_distance(Point p1, Point p2)  
{  
 double dx = p2.x - p1.x;  
 double dy = p2.y - p1.y;  
 return sqrt(dx \* dx + dy \* dy);  
}  
  
/\* 矩形を作成する関数 \*/  
Rectangle create\_rectangle(Point top\_left, Point bottom\_right)  
{  
 Rectangle r;  
 r.top\_left = top\_left;  
 r.bottom\_right = bottom\_right;  
 return r;  
}  
  
/\* 矩形の面積を計算する関数 \*/  
double rectangle\_area(Rectangle r)  
{  
 double width = r.bottom\_right.x - r.top\_left.x;  
 double height = r.top\_left.y - r.bottom\_right.y;  
 return fabs(width \* height);  
}  
  
/\* 矩形の周囲長を計算する関数 \*/  
double rectangle\_perimeter(Rectangle r)  
{  
 double width = fabs(r.bottom\_right.x - r.top\_left.x);  
 double height = fabs(r.top\_left.y - r.bottom\_right.y);  
 return 2.0 \* (width + height);  
}  
  
/\* 点が矩形内にあるかを判定する関数 \*/  
int point\_in\_rectangle(Point p, Rectangle r)  
{  
 return p.x >= r.top\_left.x && p.x <= r.bottom\_right.x &&  
 p.y <= r.top\_left.y && p.y >= r.bottom\_right.y;  
}  
  
/\* メイン関数 - テスト用 \*/  
int main(void)  
{  
 printf("=== 構造体を使った関数のテスト ===\n\n");  
   
 /\* 商品管理のテスト \*/  
 printf("=== 商品管理 ===\n");  
 Product p1 = create\_product(1001, "ノートパソコン", 98000.0, 5);  
 Product p2 = create\_product(1002, "マウス", 2980.0, 20);  
   
 print\_product(p1);  
 printf("総価値: %.2f円\n\n", calculate\_total\_value(p1));  
   
 print\_product(p2);  
 printf("総価値: %.2f円\n\n", calculate\_total\_value(p2));  
   
 printf("価格比較: ");  
 int cmp = compare\_products\_by\_price(p1, p2);  
 if (cmp > 0)  
 {  
 printf("商品1の方が高い\n");  
 }  
 else if (cmp < 0)  
 {  
 printf("商品2の方が高い\n");  
 }  
 else  
 {  
 printf("同じ価格\n");  
 }  
   
 printf("\n商品1に20%%割引を適用\n");  
 apply\_discount(&p1, 20.0);  
 print\_product(p1);  
 printf("\n");  
   
 /\* 学生管理のテスト \*/  
 printf("=== 学生管理 ===\n");  
 Student students[] = {  
 create\_student(2001, "田中太郎", 85.5),  
 create\_student(2002, "鈴木花子", 92.0),  
 create\_student(2003, "佐藤次郎", 78.3),  
 create\_student(2004, "山田美咲", 88.5)  
 };  
 int student\_count = 4;  
   
 int i;  
 printf("学生一覧:\n");  
 for (i = 0; i < student\_count; i++)  
 {  
 printf("--- 学生%d ---\n", i + 1);  
 print\_student(students[i]);  
 }  
   
 printf("\n最高得点者:\n");  
 Student best = find\_best\_student(students, student\_count);  
 print\_student(best);  
   
 printf("\n得点順にソート後:\n");  
 sort\_students\_by\_score(students, student\_count);  
 for (i = 0; i < student\_count; i++)  
 {  
 printf("%d位: %s (%.1f点)\n", i + 1, students[i].name, students[i].score);  
 }  
 printf("\n");  
   
 /\* 日付操作のテスト \*/  
 printf("=== 日付操作 ===\n");  
 Date d1 = create\_date(2024, 3, 15);  
 Date d2 = create\_date(2024, 12, 25);  
 Date d3 = create\_date(2024, 2, 29); /\* うるう年 \*/  
 Date d4 = create\_date(2023, 2, 29); /\* 無効な日付 \*/  
   
 printf("日付1: ");  
 print\_date(d1);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d1) ? "OK" : "NG");  
   
 printf("日付2: ");  
 print\_date(d2);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d2) ? "OK" : "NG");  
   
 printf("日付3: ");  
 print\_date(d3);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d3) ? "OK" : "NG");  
   
 printf("日付4: ");  
 print\_date(d4);  
 printf(" (妥当性: %s)\n", is\_valid\_date(d4) ? "OK" : "NG");  
   
 printf("\n2024年はうるう年? %s\n", is\_leap\_year(2024) ? "はい" : "いいえ");  
 printf("2023年はうるう年? %s\n", is\_leap\_year(2023) ? "はい" : "いいえ");  
   
 printf("\n日付の比較:\n");  
 int date\_cmp = compare\_dates(d1, d2);  
 if (date\_cmp < 0)  
 {  
 printf("日付1の方が早い\n");  
 }  
 else if (date\_cmp > 0)  
 {  
 printf("日付2の方が早い\n");  
 }  
 else  
 {  
 printf("同じ日付\n");  
 }  
 printf("\n");  
   
 /\* 座標・図形のテスト \*/  
 printf("=== 座標・図形 ===\n");  
 Point p\_1 = create\_point(0.0, 0.0);  
 Point p\_2 = create\_point(3.0, 4.0);  
 Point p\_3 = create\_point(5.0, 5.0);  
   
 printf("点1: (%.1f, %.1f)\n", p\_1.x, p\_1.y);  
 printf("点2: (%.1f, %.1f)\n", p\_2.x, p\_2.y);  
 printf("点1と点2の距離: %.2f\n", point\_distance(p\_1, p\_2));  
   
 Rectangle rect = create\_rectangle(create\_point(0.0, 10.0), create\_point(10.0, 0.0));  
 printf("\n矩形: 左上(%.1f, %.1f) - 右下(%.1f, %.1f)\n",  
 rect.top\_left.x, rect.top\_left.y,  
 rect.bottom\_right.x, rect.bottom\_right.y);  
 printf("面積: %.2f\n", rectangle\_area(rect));  
 printf("周囲長: %.2f\n", rectangle\_perimeter(rect));  
   
 printf("\n点の位置判定:\n");  
 printf("点2(%.1f, %.1f)は矩形内? %s\n",   
 p\_2.x, p\_2.y, point\_in\_rectangle(p\_2, rect) ? "はい" : "いいえ");  
 printf("点3(%.1f, %.1f)は矩形内? %s\n",   
 p\_3.x, p\_3.y, point\_in\_rectangle(p\_3, rect) ? "はい" : "いいえ");  
   
 return 0;  
}```  
  
### ex9\_7\_recursive\_functions.c  
  
```c  
/\*  
 \* ファイル名: ex9\_7\_recursive\_functions.c  
 \* 演習9-7: 再帰関数  
 \* 説明: 階乗、フィボナッチ、ユークリッドの互除法などの再帰関数の実装  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
long factorial\_recursive(int n);  
int fibonacci\_recursive(int n);  
int gcd\_recursive(int a, int b);  
int is\_palindrome\_recursive(const char str[], int start, int end);  
void reverse\_string\_recursive(char str[], int start, int end);  
int array\_sum\_recursive(int arr[], int size);  
int find\_max\_recursive(int arr[], int size);  
  
/\* 階乗を計算する再帰関数 \*/  
long factorial\_recursive(int n)  
{  
 /\* ベースケース \*/  
 if (n <= 1)  
 {  
 return 1;  
 }  
   
 /\* 再帰ケース \*/  
 return n \* factorial\_recursive(n - 1);  
}  
  
/\* フィボナッチ数列を計算する再帰関数 \*/  
int fibonacci\_recursive(int n)  
{  
 /\* ベースケース \*/  
 if (n <= 0)  
 {  
 return 0;  
 }  
 if (n == 1)  
 {  
 return 1;  
 }  
   
 /\* 再帰ケース \*/  
 return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2);  
}  
  
/\* ユークリッドの互除法による最大公約数を求める再帰関数 \*/  
int gcd\_recursive(int a, int b)  
{  
 /\* ベースケース \*/  
 if (b == 0)  
 {  
 return a;  
 }  
   
 /\* 再帰ケース \*/  
 return gcd\_recursive(b, a % b);  
}  
  
/\* 文字列が回文かどうか再帰的に判定する関数 \*/  
int is\_palindrome\_recursive(const char str[], int start, int end)  
{  
 /\* ベースケース: 文字が交差したか同じ位置になった場合 \*/  
 if (start >= end)  
 {  
 return 1; /\* 回文 \*/  
 }  
   
 /\* 現在の位置の文字が異なる場合 \*/  
 if (str[start] != str[end])  
 {  
 return 0; /\* 回文ではない \*/  
 }  
   
 /\* 再帰ケース: 内側の文字列をチェック \*/  
 return is\_palindrome\_recursive(str, start + 1, end - 1);  
}  
  
/\* 文字列を再帰的に逆順にする関数 \*/  
void reverse\_string\_recursive(char str[], int start, int end)  
{  
 char temp;  
   
 /\* ベースケース \*/  
 if (start >= end)  
 {  
 return;  
 }  
   
 /\* 文字を交換 \*/  
 temp = str[start];  
 str[start] = str[end];  
 str[end] = temp;  
   
 /\* 再帰ケース \*/  
 reverse\_string\_recursive(str, start + 1, end - 1);  
}  
  
/\* 配列の合計を再帰的に計算する関数 \*/  
int array\_sum\_recursive(int arr[], int size)  
{  
 /\* ベースケース \*/  
 if (size <= 0)  
 {  
 return 0;  
 }  
   
 /\* 再帰ケース \*/  
 return arr[size - 1] + array\_sum\_recursive(arr, size - 1);  
}  
  
/\* 配列の最大値を再帰的に見つける関数 \*/  
int find\_max\_recursive(int arr[], int size)  
{  
 int max\_of\_rest;  
   
 /\* ベースケース \*/  
 if (size == 1)  
 {  
 return arr[0];  
 }  
   
 /\* 再帰ケース \*/  
 max\_of\_rest = find\_max\_recursive(arr, size - 1);  
   
 /\* 現在の要素と残りの最大値を比較 \*/  
 return (arr[size - 1] > max\_of\_rest) ? arr[size - 1] : max\_of\_rest;  
}  
  
/\* 文字列の長さを求める補助関数 \*/  
int my\_strlen(const char str[])  
{  
 int len = 0;  
 while (str[len] != '\0')  
 {  
 len++;  
 }  
 return len;  
}  
  
int main(void)  
{  
 /\* C90: すべての変数を関数先頭で宣言 \*/  
 int factorial\_tests[] = {0, 1, 5, 10, 12};  
 int num\_factorial\_tests = sizeof(factorial\_tests) / sizeof(factorial\_tests[0]);  
 int gcd\_pairs[][2] = {{48, 18}, {60, 48}, {17, 13}, {100, 75}};  
 int num\_gcd\_pairs = sizeof(gcd\_pairs) / sizeof(gcd\_pairs[0]);  
 char\* palindrome\_tests[] = {"racecar", "hello", "madam", "abcba", "test"};  
 int num\_palindrome\_tests = sizeof(palindrome\_tests) / sizeof(palindrome\_tests[0]);  
 char test\_strings[][20] = {"hello", "world", "recursive", "function"};  
 int num\_strings = sizeof(test\_strings) / sizeof(test\_strings[0]);  
 int test\_array1[] = {1, 2, 3, 4, 5};  
 int test\_array2[] = {10, 20, 30, 40};  
 int test\_array3[] = {7, 14, 21, 28, 35, 42};  
 int i;  
   
 printf("=== 再帰関数のデモ ===\n\n");  
  
 /\* デモ1: 階乗計算 \*/  
 printf("1. 階乗計算\n");  
   
 for (i = 0; i < num\_factorial\_tests; i++)  
 {  
 int n = factorial\_tests[i];  
 printf(" %d! = %ld\n", n, factorial\_recursive(n));  
 }  
 printf("\n");  
  
 /\* デモ2: フィボナッチ数列 \*/  
 printf("2. フィボナッチ数列\n");  
 printf(" 最初の15項: ");  
 for (i = 0; i < 15; i++)  
 {  
 printf("%d ", fibonacci\_recursive(i));  
 }  
 printf("\n\n");  
  
 /\* デモ3: 最大公約数 \*/  
 printf("3. 最大公約数（ユークリッドの互除法）\n");  
   
 for (i = 0; i < num\_gcd\_pairs; i++)  
 {  
 int a = gcd\_pairs[i][0];  
 int b = gcd\_pairs[i][1];  
 printf(" gcd(%d, %d) = %d\n", a, b, gcd\_recursive(a, b));  
 }  
 printf("\n");  
  
 /\* デモ4: 回文判定 \*/  
 printf("4. 回文判定\n");  
   
 for (i = 0; i < num\_palindrome\_tests; i++)  
 {  
 char\* str = palindrome\_tests[i];  
 int len = my\_strlen(str);  
 int is\_pal = is\_palindrome\_recursive(str, 0, len - 1);  
 printf(" \"%s\" -> %s\n", str, is\_pal ? "回文です" : "回文ではありません");  
 }  
 printf("\n");  
  
 /\* デモ5: 文字列の逆順 \*/  
 printf("5. 文字列の逆順\n");  
   
 for (i = 0; i < num\_strings; i++)  
 {  
 char original[20];  
 int j;  
 int len;  
   
 /\* 元の文字列をコピー \*/  
 for (j = 0; test\_strings[i][j] != '\0'; j++)  
 {  
 original[j] = test\_strings[i][j];  
 }  
 original[j] = '\0';  
   
 len = my\_strlen(test\_strings[i]);  
 reverse\_string\_recursive(test\_strings[i], 0, len - 1);  
 printf(" \"%s\" -> \"%s\"\n", original, test\_strings[i]);  
 }  
 printf("\n");  
  
 /\* デモ6: 配列の合計 \*/  
 printf("6. 配列の合計（再帰）\n");  
   
 printf(" 配列 {1, 2, 3, 4, 5} の合計: %d\n",   
 array\_sum\_recursive(test\_array1, 5));  
 printf(" 配列 {10, 20, 30, 40} の合計: %d\n",   
 array\_sum\_recursive(test\_array2, 4));  
 printf(" 配列 {7, 14, 21, 28, 35, 42} の合計: %d\n",   
 array\_sum\_recursive(test\_array3, 6));  
 printf("\n");  
  
 /\* デモ7: 配列の最大値 \*/  
 printf("7. 配列の最大値（再帰）\n");  
 printf(" 配列 {1, 2, 3, 4, 5} の最大値: %d\n",   
 find\_max\_recursive(test\_array1, 5));  
 printf(" 配列 {10, 20, 30, 40} の最大値: %d\n",   
 find\_max\_recursive(test\_array2, 4));  
 printf(" 配列 {7, 14, 21, 28, 35, 42} の最大値: %d\n",   
 find\_max\_recursive(test\_array3, 6));  
 printf("\n");  
  
 printf("=== 再帰関数デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 再帰関数の基本構造（ベースケースと再帰ケース）  
2. 数学的計算での再帰の応用  
3. 文字列処理での再帰の活用  
4. 配列処理での再帰的アプローチ  
5. 再帰の効率性と限界の理解  
  
実装のポイント:  
- C90準拠のため、変数は関数先頭で宣言  
- ベースケースの重要性  
- スタックオーバーフローの注意  
- 再帰よりも反復が効率的な場合の判断  
\*/

### ex9\_8\_advanced\_strings.c

/\*  
 \* ファイル名: ex9\_8\_advanced\_strings.c  
 \* 演習9-8: 高度な文字列処理  
 \* 説明: 文字列の解析、パターンマッチング、フォーマット処理などの高度な関数群  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
int tokenize(char str[], char tokens[][50], char delimiter);  
int check\_brackets(const char str[]);  
int extract\_numbers(const char str[], int numbers[], int max\_count);  
int wildcard\_match(const char str[], const char pattern[]);  
int string\_replace(char str[], const char old\_substr[], const char new\_substr[]);  
void center\_string(char result[], const char str[], int width);  
int parse\_csv\_line(const char line[], char fields[][100], int max\_fields);  
  
/\* 補助関数 \*/  
int my\_strlen(const char str[]);  
void my\_strcpy(char dest[], const char src[]);  
int my\_strncmp(const char str1[], const char str2[], int n);  
int is\_digit(char c);  
  
/\* 文字列をトークンに分割する関数 \*/  
int tokenize(char str[], char tokens[][50], char delimiter)  
{  
 int token\_count = 0;  
 int i = 0;  
 int token\_start = 0;  
 int token\_len;  
 int j;  
   
 while (str[i] != '\0' && token\_count < 20) /\* 最大20トークン \*/  
 {  
 if (str[i] == delimiter)  
 {  
 /\* トークンの終了 \*/  
 token\_len = i - token\_start;  
 if (token\_len > 0)  
 {  
 /\* トークンをコピー \*/  
 for (j = 0; j < token\_len && j < 49; j++)  
 {  
 tokens[token\_count][j] = str[token\_start + j];  
 }  
 tokens[token\_count][j] = '\0';  
 token\_count++;  
 }  
 token\_start = i + 1;  
 }  
 i++;  
 }  
   
 /\* 最後のトークン \*/  
 token\_len = i - token\_start;  
 if (token\_len > 0 && token\_count < 20)  
 {  
 for (j = 0; j < token\_len && j < 49; j++)  
 {  
 tokens[token\_count][j] = str[token\_start + j];  
 }  
 tokens[token\_count][j] = '\0';  
 token\_count++;  
 }  
   
 return token\_count;  
}  
  
/\* 文字列内の括弧の対応をチェックする関数 \*/  
int check\_brackets(const char str[])  
{  
 int round\_count = 0; /\* ( ) \*/  
 int square\_count = 0; /\* [ ] \*/  
 int curly\_count = 0; /\* { } \*/  
 int i = 0;  
   
 while (str[i] != '\0')  
 {  
 switch (str[i])  
 {  
 case '(':  
 round\_count++;  
 break;  
 case ')':  
 round\_count--;  
 if (round\_count < 0) return 0; /\* 閉じ括弧が多い \*/  
 break;  
 case '[':  
 square\_count++;  
 break;  
 case ']':  
 square\_count--;  
 if (square\_count < 0) return 0;  
 break;  
 case '{':  
 curly\_count++;  
 break;  
 case '}':  
 curly\_count--;  
 if (curly\_count < 0) return 0;  
 break;  
 }  
 i++;  
 }  
   
 /\* すべての括弧がバランスしているかチェック \*/  
 return (round\_count == 0 && square\_count == 0 && curly\_count == 0);  
}  
  
/\* 文字列から数値を抽出する関数 \*/  
int extract\_numbers(const char str[], int numbers[], int max\_count)  
{  
 int count = 0;  
 int i = 0;  
 int current\_number = 0;  
 int in\_number = 0;  
 int is\_negative = 0;  
   
 while (str[i] != '\0' && count < max\_count)  
 {  
 if (is\_digit(str[i]))  
 {  
 if (!in\_number)  
 {  
 in\_number = 1;  
 current\_number = 0;  
 is\_negative = 0;  
   
 /\* 負の数のチェック \*/  
 if (i > 0 && str[i - 1] == '-')  
 {  
 is\_negative = 1;  
 }  
 }  
 current\_number = current\_number \* 10 + (str[i] - '0');  
 }  
 else  
 {  
 if (in\_number)  
 {  
 /\* 数値の終了 \*/  
 numbers[count] = is\_negative ? -current\_number : current\_number;  
 count++;  
 in\_number = 0;  
 }  
 }  
 i++;  
 }  
   
 /\* 文字列の最後が数値の場合 \*/  
 if (in\_number && count < max\_count)  
 {  
 numbers[count] = is\_negative ? -current\_number : current\_number;  
 count++;  
 }  
   
 return count;  
}  
  
/\* 簡単なワイルドカード（\*、?）パターンマッチング関数 \*/  
int wildcard\_match(const char str[], const char pattern[])  
{  
 int str\_i = 0;  
 int pat\_i = 0;  
 int str\_len = my\_strlen(str);  
 int pat\_len = my\_strlen(pattern);  
   
 while (str\_i < str\_len && pat\_i < pat\_len)  
 {  
 if (pattern[pat\_i] == '\*')  
 {  
 /\* \* は0文字以上にマッチ \*/  
 pat\_i++;  
 if (pat\_i == pat\_len) return 1; /\* パターンの最後が \* \*/  
   
 /\* 次の文字が見つかるまでスキップ \*/  
 while (str\_i < str\_len)  
 {  
 if (wildcard\_match(&str[str\_i], &pattern[pat\_i]))  
 {  
 return 1;  
 }  
 str\_i++;  
 }  
 return 0;  
 }  
 else if (pattern[pat\_i] == '?' || pattern[pat\_i] == str[str\_i])  
 {  
 /\* ? は任意の1文字、または文字が一致 \*/  
 str\_i++;  
 pat\_i++;  
 }  
 else  
 {  
 return 0; /\* 不一致 \*/  
 }  
 }  
   
 /\* 残りのパターンがすべて \* かチェック \*/  
 while (pat\_i < pat\_len && pattern[pat\_i] == '\*')  
 {  
 pat\_i++;  
 }  
   
 return (str\_i == str\_len && pat\_i == pat\_len);  
}  
  
/\* 文字列の置換関数（簡単版） \*/  
int string\_replace(char str[], const char old\_substr[], const char new\_substr[])  
{  
 char temp[1000]; /\* 作業用バッファ \*/  
 int old\_len = my\_strlen(old\_substr);  
 int new\_len = my\_strlen(new\_substr);  
 int str\_len = my\_strlen(str);  
 int i = 0;  
 int temp\_i = 0;  
 int replacements = 0;  
   
 while (i < str\_len)  
 {  
 if (my\_strncmp(&str[i], old\_substr, old\_len) == 0)  
 {  
 /\* 置換対象が見つかった \*/  
 int j;  
 for (j = 0; j < new\_len; j++)  
 {  
 temp[temp\_i++] = new\_substr[j];  
 }  
 i += old\_len;  
 replacements++;  
 }  
 else  
 {  
 temp[temp\_i++] = str[i];  
 i++;  
 }  
 }  
   
 temp[temp\_i] = '\0';  
 my\_strcpy(str, temp);  
   
 return replacements;  
}  
  
/\* 文字列を指定幅で中央揃えする関数 \*/  
void center\_string(char result[], const char str[], int width)  
{  
 int str\_len = my\_strlen(str);  
 int padding = (width - str\_len) / 2;  
 int i;  
   
 if (str\_len >= width)  
 {  
 /\* 文字列が幅より長い場合はそのままコピー \*/  
 my\_strcpy(result, str);  
 return;  
 }  
   
 /\* 左側のパディング \*/  
 for (i = 0; i < padding; i++)  
 {  
 result[i] = ' ';  
 }  
   
 /\* 文字列をコピー \*/  
 for (i = 0; i < str\_len; i++)  
 {  
 result[padding + i] = str[i];  
 }  
   
 /\* 右側のパディング \*/  
 for (i = padding + str\_len; i < width; i++)  
 {  
 result[i] = ' ';  
 }  
   
 result[width] = '\0';  
}  
  
/\* CSV形式の文字列を解析する関数 \*/  
int parse\_csv\_line(const char line[], char fields[][100], int max\_fields)  
{  
 int field\_count = 0;  
 int i = 0;  
 int field\_start = 0;  
 int in\_quotes = 0;  
 int field\_len;  
 int j;  
   
 while (line[i] != '\0' && field\_count < max\_fields)  
 {  
 if (line[i] == '"')  
 {  
 in\_quotes = !in\_quotes;  
 }  
 else if (line[i] == ',' && !in\_quotes)  
 {  
 /\* フィールドの終了 \*/  
 field\_len = i - field\_start;  
 for (j = 0; j < field\_len && j < 99; j++)  
 {  
 fields[field\_count][j] = line[field\_start + j];  
 }  
 fields[field\_count][j] = '\0';  
 field\_count++;  
 field\_start = i + 1;  
 }  
 i++;  
 }  
   
 /\* 最後のフィールド \*/  
 if (field\_count < max\_fields)  
 {  
 field\_len = i - field\_start;  
 for (j = 0; j < field\_len && j < 99; j++)  
 {  
 fields[field\_count][j] = line[field\_start + j];  
 }  
 fields[field\_count][j] = '\0';  
 field\_count++;  
 }  
   
 return field\_count;  
}  
  
/\* 補助関数の実装 \*/  
int my\_strlen(const char str[])  
{  
 int len = 0;  
 while (str[len] != '\0') len++;  
 return len;  
}  
  
void my\_strcpy(char dest[], const char src[])  
{  
 int i = 0;  
 while (src[i] != '\0')  
 {  
 dest[i] = src[i];  
 i++;  
 }  
 dest[i] = '\0';  
}  
  
int my\_strncmp(const char str1[], const char str2[], int n)  
{  
 int i;  
 for (i = 0; i < n; i++)  
 {  
 if (str1[i] != str2[i])  
 {  
 return str1[i] - str2[i];  
 }  
 if (str1[i] == '\0')  
 {  
 return 0;  
 }  
 }  
 return 0;  
}  
  
int is\_digit(char c)  
{  
 return (c >= '0' && c <= '9');  
}  
  
int main(void)  
{  
 printf("=== 高度な文字列処理のデモ ===\n\n");  
  
 /\* デモ1: 文字列の分割 \*/  
 printf("1. 文字列の分割（トークン化）\n");  
 char test\_string1[] = "apple,banana,orange,grape";  
 char tokens[20][50];  
 int token\_count = tokenize(test\_string1, tokens, ',');  
 int i, j;  
   
 printf(" 文字列: \"%s\"\n", "apple,banana,orange,grape");  
 printf(" 分割結果: ");  
 for (i = 0; i < token\_count; i++)  
 {  
 printf("\"%s\"", tokens[i]);  
 if (i < token\_count - 1) printf(", ");  
 }  
 printf("\n\n");  
  
 /\* デモ2: 括弧のバランスチェック \*/  
 printf("2. 括弧のバランスチェック\n");  
 const char\* bracket\_tests[] = {  
 "((()))",  
 "([{}])",  
 "(()",  
 "([)]",  
 "{[()()()]}"  
 };  
 int num\_bracket\_tests = sizeof(bracket\_tests) / sizeof(bracket\_tests[0]);  
   
 for (i = 0; i < num\_bracket\_tests; i++)  
 {  
 int is\_balanced = check\_brackets(bracket\_tests[i]);  
 printf(" \"%s\" -> %s\n", bracket\_tests[i],   
 is\_balanced ? "バランス取れています" : "バランスが取れていません");  
 }  
 printf("\n");  
  
 /\* デモ3: 数値の抽出 \*/  
 printf("3. 文字列からの数値抽出\n");  
 const char\* number\_tests[] = {  
 "価格は1500円、税込み1650円です",  
 "座標: x=10, y=-25, z=100",  
 "電話番号: 03-1234-5678"  
 };  
 int num\_number\_tests = sizeof(number\_tests) / sizeof(number\_tests[0]);  
   
 for (i = 0; i < num\_number\_tests; i++)  
 {  
 int numbers[20];  
 int count = extract\_numbers(number\_tests[i], numbers, 20);  
 printf(" \"%s\"\n", number\_tests[i]);  
 printf(" 抽出された数値: ");  
 for (j = 0; j < count; j++)  
 {  
 printf("%d", numbers[j]);  
 if (j < count - 1) printf(", ");  
 }  
 printf("\n\n");  
 }  
  
 /\* デモ4: ワイルドカードマッチング \*/  
 printf("4. ワイルドカードマッチング\n");  
 const char\* match\_tests[][2] = {  
 {"hello", "h\*o"},  
 {"test123", "test\*"},  
 {"filename.txt", "\*.txt"},  
 {"abc", "a?c"},  
 {"hello", "world"}  
 };  
 int num\_match\_tests = sizeof(match\_tests) / sizeof(match\_tests[0]);  
   
 for (i = 0; i < num\_match\_tests; i++)  
 {  
 int matches = wildcard\_match(match\_tests[i][0], match\_tests[i][1]);  
 printf(" \"%s\" と \"%s\" -> %s\n",   
 match\_tests[i][0], match\_tests[i][1],  
 matches ? "マッチします" : "マッチしません");  
 }  
 printf("\n");  
  
 /\* デモ5: 文字列の置換 \*/  
 printf("5. 文字列の置換\n");  
 char replace\_test[] = "Hello world! This is a test world.";  
 printf(" 元の文字列: \"%s\"\n", replace\_test);  
 int replacements = string\_replace(replace\_test, "world", "universe");  
 printf(" 置換後: \"%s\"\n", replace\_test);  
 printf(" 置換回数: %d回\n\n", replacements);  
  
 /\* デモ6: 文字列の中央揃え \*/  
 printf("6. 文字列の中央揃え\n");  
 const char\* center\_tests[] = {"Hello", "World", "Test"};  
 int widths[] = {10, 15, 20};  
   
 for (i = 0; i < 3; i++)  
 {  
 char centered[50];  
 center\_string(centered, center\_tests[i], widths[i]);  
 printf(" 幅%d: |%s|\n", widths[i], centered);  
 }  
 printf("\n");  
  
 /\* デモ7: CSV解析 \*/  
 printf("7. CSV形式の解析\n");  
 const char\* csv\_tests[] = {  
 "名前,年齢,職業",  
 "田中太郎,25,エンジニア",  
 "\"佐藤,花子\",30,\"営業,マネージャー\""  
 };  
 int num\_csv\_tests = sizeof(csv\_tests) / sizeof(csv\_tests[0]);  
   
 for (i = 0; i < num\_csv\_tests; i++)  
 {  
 char fields[10][100];  
 int field\_count = parse\_csv\_line(csv\_tests[i], fields, 10);  
 printf(" \"%s\"\n", csv\_tests[i]);  
 printf(" フィールド: ");  
 for (j = 0; j < field\_count; j++)  
 {  
 printf("[%s]", fields[j]);  
 if (j < field\_count - 1) printf(" ");  
 }  
 printf("\n\n");  
 }  
  
 printf("=== 高度な文字列処理デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 文字列の動的な解析と分割  
2. 構文解析の基本（括弧のバランス）  
3. パターンマッチングアルゴリズム  
4. 文字列操作の効率的な実装  
5. フォーマット処理とパース処理  
  
実装のポイント:  
- C90準拠のため、変数は関数先頭で宣言  
- 安全な文字列操作（バッファオーバーフロー対策）  
- 効率的なアルゴリズムの選択  
- エラーハンドリングの考慮  
\*/```  
  
### ex9\_9\_sorting\_algorithms.c  
  
```c  
/\*  
 \* ファイル名: ex9\_9\_sorting\_algorithms.c  
 \* 演習9-9: ソートアルゴリズム関数  
 \* 説明: 各種ソートアルゴリズムの実装（バブル、選択、挿入、マージソート等）  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
void bubble\_sort(int arr[], int size);  
void selection\_sort(int arr[], int size);  
void insertion\_sort(int arr[], int size);  
void merge\_sort(int arr[], int left, int right);  
void merge(int arr[], int left, int mid, int right);  
void quick\_sort(int arr[], int left, int right);  
int partition(int arr[], int left, int right);  
void sort\_strings(char strings[][100], int count);  
void print\_array(int arr[], int size);  
void copy\_array(int dest[], int src[], int size);  
  
/\* バブルソート \*/  
void bubble\_sort(int arr[], int size)  
{  
 int i, j, temp;  
 int swapped;  
   
 for (i = 0; i < size - 1; i++)  
 {  
 swapped = 0;  
 for (j = 0; j < size - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 /\* 要素を交換 \*/  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 swapped = 1;  
 }  
 }  
 /\* 交換が発生しなかった場合、ソート完了 \*/  
 if (!swapped) break;  
 }  
}  
  
/\* 選択ソート \*/  
void selection\_sort(int arr[], int size)  
{  
 int i, j, min\_idx, temp;  
   
 for (i = 0; i < size - 1; i++)  
 {  
 min\_idx = i;  
 for (j = i + 1; j < size; j++)  
 {  
 if (arr[j] < arr[min\_idx])  
 {  
 min\_idx = j;  
 }  
 }  
   
 /\* 最小値を先頭に移動 \*/  
 if (min\_idx != i)  
 {  
 temp = arr[i];  
 arr[i] = arr[min\_idx];  
 arr[min\_idx] = temp;  
 }  
 }  
}  
  
/\* 挿入ソート \*/  
void insertion\_sort(int arr[], int size)  
{  
 int i, j, key;  
   
 for (i = 1; i < size; i++)  
 {  
 key = arr[i];  
 j = i - 1;  
   
 /\* key より大きい要素を後ろに移動 \*/  
 while (j >= 0 && arr[j] > key)  
 {  
 arr[j + 1] = arr[j];  
 j--;  
 }  
 arr[j + 1] = key;  
 }  
}  
  
/\* マージソート用のマージ関数 \*/  
void merge(int arr[], int left, int mid, int right)  
{  
 int left\_arr[1000]; /\* 左の部分配列 \*/  
 int right\_arr[1000]; /\* 右の部分配列 \*/  
 int left\_size = mid - left + 1;  
 int right\_size = right - mid;  
 int i, j, k;  
   
 /\* 左の部分配列をコピー \*/  
 for (i = 0; i < left\_size; i++)  
 {  
 left\_arr[i] = arr[left + i];  
 }  
   
 /\* 右の部分配列をコピー \*/  
 for (j = 0; j < right\_size; j++)  
 {  
 right\_arr[j] = arr[mid + 1 + j];  
 }  
   
 /\* マージ処理 \*/  
 i = 0; j = 0; k = left;  
 while (i < left\_size && j < right\_size)  
 {  
 if (left\_arr[i] <= right\_arr[j])  
 {  
 arr[k] = left\_arr[i];  
 i++;  
 }  
 else  
 {  
 arr[k] = right\_arr[j];  
 j++;  
 }  
 k++;  
 }  
   
 /\* 残りの要素をコピー \*/  
 while (i < left\_size)  
 {  
 arr[k] = left\_arr[i];  
 i++;  
 k++;  
 }  
   
 while (j < right\_size)  
 {  
 arr[k] = right\_arr[j];  
 j++;  
 k++;  
 }  
}  
  
/\* マージソート \*/  
void merge\_sort(int arr[], int left, int right)  
{  
 int mid;  
   
 if (left < right)  
 {  
 mid = left + (right - left) / 2;  
   
 /\* 左半分をソート \*/  
 merge\_sort(arr, left, mid);  
   
 /\* 右半分をソート \*/  
 merge\_sort(arr, mid + 1, right);  
   
 /\* マージ \*/  
 merge(arr, left, mid, right);  
 }  
}  
  
/\* クイックソート用の分割関数 \*/  
int partition(int arr[], int left, int right)  
{  
 int pivot = arr[right]; /\* 最後の要素をピボットとする \*/  
 int i = left - 1;  
 int j, temp;  
   
 for (j = left; j <= right - 1; j++)  
 {  
 if (arr[j] < pivot)  
 {  
 i++;  
 temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
   
 temp = arr[i + 1];  
 arr[i + 1] = arr[right];  
 arr[right] = temp;  
   
 return i + 1;  
}  
  
/\* クイックソート \*/  
void quick\_sort(int arr[], int left, int right)  
{  
 int pivot\_index;  
   
 if (left < right)  
 {  
 pivot\_index = partition(arr, left, right);  
   
 /\* ピボットの左側をソート \*/  
 quick\_sort(arr, left, pivot\_index - 1);  
   
 /\* ピボットの右側をソート \*/  
 quick\_sort(arr, pivot\_index + 1, right);  
 }  
}  
  
/\* 文字列配列のソート（バブルソート版） \*/  
void sort\_strings(char strings[][100], int count)  
{  
 int i, j, k;  
 char temp[100];  
 int compare\_result;  
   
 for (i = 0; i < count - 1; i++)  
 {  
 for (j = 0; j < count - 1 - i; j++)  
 {  
 /\* 文字列比較 \*/  
 compare\_result = 0;  
 for (k = 0; strings[j][k] != '\0' && strings[j + 1][k] != '\0'; k++)  
 {  
 if (strings[j][k] < strings[j + 1][k])  
 {  
 compare\_result = -1;  
 break;  
 }  
 else if (strings[j][k] > strings[j + 1][k])  
 {  
 compare\_result = 1;  
 break;  
 }  
 }  
   
 if (compare\_result == 0)  
 {  
 if (strings[j][k] != '\0') compare\_result = 1;  
 else if (strings[j + 1][k] != '\0') compare\_result = -1;  
 }  
   
 if (compare\_result > 0)  
 {  
 /\* 文字列を交換 \*/  
 for (k = 0; strings[j][k] != '\0'; k++)  
 {  
 temp[k] = strings[j][k];  
 }  
 temp[k] = '\0';  
   
 for (k = 0; strings[j + 1][k] != '\0'; k++)  
 {  
 strings[j][k] = strings[j + 1][k];  
 }  
 strings[j][k] = '\0';  
   
 for (k = 0; temp[k] != '\0'; k++)  
 {  
 strings[j + 1][k] = temp[k];  
 }  
 strings[j + 1][k] = '\0';  
 }  
 }  
 }  
}  
  
/\* 配列を表示する関数 \*/  
void print\_array(int arr[], int size)  
{  
 int i;  
 printf("[");  
 for (i = 0; i < size; i++)  
 {  
 printf("%d", arr[i]);  
 if (i < size - 1) printf(", ");  
 }  
 printf("]");  
}  
  
/\* 配列をコピーする関数 \*/  
void copy\_array(int dest[], int src[], int size)  
{  
 int i;  
 for (i = 0; i < size; i++)  
 {  
 dest[i] = src[i];  
 }  
}  
  
int main(void)  
{  
 printf("=== ソートアルゴリズムのデモ ===\n\n");  
  
 /\* テスト用データ \*/  
 int original\_data[] = {64, 34, 25, 12, 22, 11, 90, 88, 76, 50};  
 int data\_size = sizeof(original\_data) / sizeof(original\_data[0]);  
 int test\_data[20];  
 int i;  
  
 printf("元のデータ: ");  
 print\_array(original\_data, data\_size);  
 printf("\n\n");  
  
 /\* 1. バブルソート \*/  
 printf("1. バブルソート\n");  
 copy\_array(test\_data, original\_data, data\_size);  
 printf(" ソート前: ");  
 print\_array(test\_data, data\_size);  
 printf("\n");  
 bubble\_sort(test\_data, data\_size);  
 printf(" ソート後: ");  
 print\_array(test\_data, data\_size);  
 printf("\n\n");  
  
 /\* 2. 選択ソート \*/  
 printf("2. 選択ソート\n");  
 copy\_array(test\_data, original\_data, data\_size);  
 printf(" ソート前: ");  
 print\_array(test\_data, data\_size);  
 printf("\n");  
 selection\_sort(test\_data, data\_size);  
 printf(" ソート後: ");  
 print\_array(test\_data, data\_size);  
 printf("\n\n");  
  
 /\* 3. 挿入ソート \*/  
 printf("3. 挿入ソート\n");  
 copy\_array(test\_data, original\_data, data\_size);  
 printf(" ソート前: ");  
 print\_array(test\_data, data\_size);  
 printf("\n");  
 insertion\_sort(test\_data, data\_size);  
 printf(" ソート後: ");  
 print\_array(test\_data, data\_size);  
 printf("\n\n");  
  
 /\* 4. マージソート \*/  
 printf("4. マージソート\n");  
 copy\_array(test\_data, original\_data, data\_size);  
 printf(" ソート前: ");  
 print\_array(test\_data, data\_size);  
 printf("\n");  
 merge\_sort(test\_data, 0, data\_size - 1);  
 printf(" ソート後: ");  
 print\_array(test\_data, data\_size);  
 printf("\n\n");  
  
 /\* 5. クイックソート \*/  
 printf("5. クイックソート\n");  
 copy\_array(test\_data, original\_data, data\_size);  
 printf(" ソート前: ");  
 print\_array(test\_data, data\_size);  
 printf("\n");  
 quick\_sort(test\_data, 0, data\_size - 1);  
 printf(" ソート後: ");  
 print\_array(test\_data, data\_size);  
 printf("\n\n");  
  
 /\* 6. 文字列ソート \*/  
 printf("6. 文字列ソート\n");  
 char fruits[][100] = {"orange", "apple", "banana", "grape", "cherry"};  
 int fruit\_count = 5;  
   
 printf(" ソート前: ");  
 for (i = 0; i < fruit\_count; i++)  
 {  
 printf("%s", fruits[i]);  
 if (i < fruit\_count - 1) printf(", ");  
 }  
 printf("\n");  
   
 sort\_strings(fruits, fruit\_count);  
   
 printf(" ソート後: ");  
 for (i = 0; i < fruit\_count; i++)  
 {  
 printf("%s", fruits[i]);  
 if (i < fruit\_count - 1) printf(", ");  
 }  
 printf("\n\n");  
  
 /\* 7. 性能比較用のより大きなデータセット \*/  
 printf("7. 大きなデータセットでのテスト\n");  
 int large\_data[50];  
 int large\_size = 50;  
   
 /\* テストデータを生成（擬似ランダム） \*/  
 printf(" 生成されたデータ（最初の20個）: ");  
 for (i = 0; i < large\_size; i++)  
 {  
 large\_data[i] = (i \* 17 + 23) % 100; /\* 簡単な擬似ランダム \*/  
 if (i < 20)  
 {  
 printf("%d", large\_data[i]);  
 if (i < 19) printf(", ");  
 }  
 }  
 printf("...\n");  
   
 /\* 挿入ソートでソート \*/  
 insertion\_sort(large\_data, large\_size);  
   
 printf(" ソート後（最初の20個）: ");  
 for (i = 0; i < 20; i++)  
 {  
 printf("%d", large\_data[i]);  
 if (i < 19) printf(", ");  
 }  
 printf("...\n\n");  
  
 printf("=== ソートアルゴリズムデモ完了 ===\n");  
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 基本的なソートアルゴリズムの理解と実装  
2. 各アルゴリズムの時間計算量の違い  
3. 分割統治法（マージソート、クイックソート）  
4. 文字列ソートの実装方法  
5. アルゴリズムの選択基準  
  
実装のポイント:  
- C90準拠のため、変数は関数先頭で宣言  
- 各ソートの特徴を活かした実装  
- 効率性と可読性のバランス  
- メモリ使用量の考慮  
\*/```  
  
  
```{=openxml}  
<w:p><w:r><w:br w:type="page"/></w:r></w:p>

# 第10章 ポインタ基礎

## 演習問題

## 基礎問題

### 演習10-1: ポインタの基本操作

2つの整数変数の値をポインタを使って交換するプログラムを作成してください。

**要件:** - 2つの整数変数を宣言・初期化 - ポインタを使った値の交換関数を実装 - 交換前後の値とアドレスを表示 - 直接的な値の交換とポインタ経由の交換を比較

**期待される出力例:**

交換前: a = (アドレス: xffffbffac), b = (アドレス: xffffbffa)  
ポインタを使った交換実行...  
交換後: a = (アドレス: xffffbffac), b = (アドレス: xffffbffa)

### 演習10-2: 配列とポインタ

ポインタ演算を使って配列の要素を逆順に表示するプログラムを作成してください。

**要件:** - 整数配列を宣言・初期化 - ポインタ演算で配列の最後から最初に向かってアクセス - インデックス記法とポインタ記法の両方で表示 - 配列のサイズを動的に計算

### 演習10-3: 文字列操作

ポインタを使って文字列の長さを計算し、文字列を逆順にするプログラムを作成してください。

**要件:** - 文字列の長さ計算関数（strlen相当）をポインタで実装 - 文字列を逆順にする関数をポインタで実装 - 元の文字列と逆順文字列を表示 - 文字列リテラルと文字配列の違いを考慮

## 応用問題

### 演習10-4: 配列操作関数

ポインタを使って配列の最大値、最小値、平均値を計算する関数群を作成してください。

**要件:** - 最大値を見つけてそのポインタを返す関数 - 最小値を見つけてそのポインタを返す関数 - 平均値を計算する関数（戻り値はdouble） - 各関数の結果を使ってレポートを作成

**実装する関数:**

int\* find\_max(int \*arr, int size);  
int\* find\_min(int \*arr, int size);  
double calculate\_average(int \*arr, int size);  
void print\_statistics(int \*arr, int size);

### 演習10-5: 文字列処理

ポインタを使って文字列の検索、置換、分割を実行する関数群を作成してください。

**要件:** - 文字列内で特定の文字を検索する関数 - 文字列内の文字を置換する関数 - 文字列を特定の文字で分割する関数 - 大文字・小文字を無視した比較機能

### 演習10-6: データ変換

ポインタを使って配列のデータ型変換を実行するプログラムを作成してください。

**要件:** - int配列をfloat配列に変換 - 文字列を数値配列に変換 - バイト配列を整数として解釈 - エラーハンドリングの実装

## 発展問題

### 演習10-7: メモリ操作

ポインタを使って任意のデータ型の配列をコピーする汎用関数を作成してください。

**要件:** - void\*ポインタを使った汎用コピー関数 - バイト単位でのメモリコピー - 型安全性の考慮 - コピー対象の境界チェック

**関数プロトタイプ:**

void\* generic\_copy(void \*dest, const void \*src, size\_t size);  
int compare\_memory(const void \*ptr, const void \*ptr, size\_t size);

### 演習10-8: アルゴリズム実装

ポインタを使って各種ソートアルゴリズムを実装してください。

**要件:** - バブルソート（ポインタ版） - 選択ソート（ポインタ版） - 挿入ソート（ポインタ版） - 汎用的な比較関数の使用

### 演習10-9: データ構造

ポインタを使って簡単なリンクリスト構造を実装してください。

**要件:** - ノード構造体の定義 - リストへの要素追加・削除 - リストの走査・検索 - メモリリークの防止

**構造体例:**

typedef struct Node {  
 int data;  
 struct Node \*next;  
} Node;

## チャレンジ問題

### 演習10-10: 動的配列シミュレータ

malloc/freeを使わずに、静的配列とポインタを使って動的配列のような動作を実現してください。

**要件:** - 固定サイズの大きな配列をメモリプールとして使用 - 要素の追加・削除・挿入機能 - メモリの断片化管理 - ガベージコレクション機能

## 提出方法

1. 各問題に対して別々のCファイルを作成
2. ファイル名は ex\_pointer\_swap.c, ex\_array\_reverse.c のように命名
3. 各プログラムの冒頭にコメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認

## コンパイル例

gcc -std=c90 -Wall -Wextra -pedantic ex\_pointer\_swap.c -o ex\_pointer\_swap  
./ex\_pointer\_swap

## ヒント

### ポインタの基本

* 宣言: int \*ptr;
* アドレス取得: ptr = &variable;
* 値の取得: value = \*ptr;
* 演算: ptr + 1 は次の要素を指す

### 安全なプログラミング

* NULLポインタのチェック: if (ptr != NULL)
* 配列の境界チェック
* 初期化されていないポインタの使用を避ける

### デバッグのコツ

* アドレスと値を分けて表示
* ポインタ演算の結果を段階的に確認
* コンパイラの警告に注意を払う

### よくある間違い

* \*ptr++ と (\*ptr)++ の違い
* ローカル変数のアドレスを返すこと
* 配列の範囲外アクセス
* NULLポインタの参照

## 解答例

この章の演習問題の解答例です。各解答にはC90版とC99版の両方を用意しています。

## 演習問題の解答

### 演習10-1: ポインタの基本操作

2つの整数変数の値をポインタを使って交換する演習です。

* <ex10_1_pointer_swap.c> - C90準拠版
* <ex10_1_pointer_swap_c99.c> - C99準拠版（//コメント、bool型、インライン関数、restrict修飾子）

**学習ポイント:** - ポインタによる参照渡しの仕組み - 値渡しとの違いの理解 - アドレス演算子（&）と間接参照演算子（\*）の使用 - ポインタの安全な使用方法

### 演習10-2: 配列とポインタ

ポインタ演算を使って配列の要素を逆順に表示する演習です。

* <ex10_2_array_reverse.c> - C90準拠版
* <ex10_2_array_reverse_c99.c> - C99準拠版（可変長配列、複合リテラル、指定初期化子）

**学習ポイント:** - 配列とポインタの等価性 - ポインタ演算による配列要素アクセス - メモリレイアウトの理解 - 安全な境界チェック

### 演習10-3: 文字列操作

ポインタを使った文字列の長さ計算、複写、連結を行う演習です。

* <ex10_3_string_manipulation.c> - C90準拠版
* <ex10_3_string_manipulation_c99.c> - C99準拠版（//コメント、restrict修飾子、bool型）

**学習ポイント:** - 文字列とポインタの関係 - 文字列の終端（ヌル文字）の処理 - ポインタを使った文字列操作関数の実装 - バッファオーバーフローの防止

### 演習10-4: 配列操作関数

ポインタを使って配列の最大値、最小値、平均値を求める演習です。

* <ex10_4_array_functions.c> - C90準拠版
* <ex10_4_array_functions_c99.c> - C99準拠版（//コメント、bool型、restrict修飾子）

**学習ポイント:** - 配列をポインタとして関数に渡す方法 - 配列サイズの適切な管理 - ポインタを使った効率的な配列操作 - 関数の戻り値としてのポインタの活用

## C90とC99の主な違い

### 1. コメントスタイル

* C90: /\* \*/ のみ
* C99: // 単一行コメントも使用可能

### 2. 変数宣言

* C90: ブロックの先頭でのみ宣言可能
* C99: 使用箇所で宣言可能（forループ内での宣言など）

### 3. bool型

* C90: 存在しない（intで代用）
* C99: <stdbool.h>でbool、true、falseが使用可能

### 4. インライン関数

* C90: 存在しない
* C99: inlineキーワードで関数のインライン化が可能

### 5. 可変長配列（VLA）

* C90: 存在しない
* C99: 実行時にサイズが決まる配列を作成可能

### 6. 複合リテラル

* C90: 存在しない
* C99: (Type){初期化子}の形式で一時オブジェクトを作成可能

### 7. 指定初期化子

* C90: 存在しない
* C99: {[index] = value}の形式で配列の特定要素を初期化可能

### 8. restrict修飾子

* C90: 存在しない
* C99: ポインタが重複しないことをコンパイラに伝える最適化ヒント

### 9. 固定幅整数型

* C90: 存在しない
* C99: <stdint.h>でint8\_t、int16\_t、int32\_tなどが使用可能

## 実行方法

# C90版のコンパイルと実行  
gcc -std=c90 -Wall -Wextra -pedantic ex10\_1\_pointer\_swap.c -o ex10\_1\_pointer\_swap  
./ex10\_1\_pointer\_swap  
  
# C99版のコンパイルと実行  
gcc -std=c99 -Wall -Wextra -pedantic ex10\_1\_pointer\_swap\_c99.c -o ex10\_1\_pointer\_swap\_c99  
./ex10\_1\_pointer\_swap\_c99  
  
# その他の演習例  
gcc -std=c90 -Wall -Wextra -pedantic ex10\_2\_array\_reverse.c -o ex10\_2\_array\_reverse  
gcc -std=c90 -Wall -Wextra -pedantic ex10\_3\_string\_manipulation.c -o ex10\_3\_string\_manipulation  
gcc -std=c90 -Wall -Wextra -pedantic ex10\_4\_array\_functions.c -o ex10\_4\_array\_functions  
  
# Makefileを使用した場合  
make run-ex10\_1\_pointer\_swap # C90版  
make run-ex10\_1\_pointer\_swap\_c99 # C99版  
make run-ex10\_2\_array\_reverse # C90版  
make run-ex10\_2\_array\_reverse\_c99 # C99版  
make run-ex10\_3\_string\_manipulation # C90版  
make run-ex10\_3\_string\_manipulation\_c99 # C99版  
make run-ex10\_4\_array\_functions # C90版  
make run-ex10\_4\_array\_functions\_c99 # C99版

## ポインタ使用時の注意事項

### 1. 初期化の重要性

// NG: 未初期化ポインタの使用  
int \*ptr;  
\*ptr = 10; // 危険！未初期化ポインタの参照  
  
// OK: 適切な初期化  
int value = 0;  
int \*ptr = &value;  
\*ptr = 10; // 安全

### 2. NULLポインタのチェック

// 関数でポインタを受け取る際は必ずチェック  
void process\_data(int \*ptr)  
{  
 if (ptr == NULL) {  
 printf("エラー: NULLポインタが渡されました\n");  
 return;  
 }  
 // 処理を続行  
}

### 3. 配列の境界チェック

// 配列のサイズを超えたアクセスは未定義動作  
int arr[5] = {1, 2, 3, 4, 5};  
int \*ptr = arr;  
  
// NG: 境界を超えたアクセス  
printf("%d\n", \*(ptr + 10)); // 危険！  
  
// OK: 境界内でのアクセス  
for (int i = 0; i < 5; i++) {  
 printf("%d ", \*(ptr + i));  
}

### 4. ローカル変数のアドレスを返さない

// NG: ローカル変数のアドレスを返す  
int\* dangerous\_function(void)  
{  
 int local\_var = 100;  
 return &local\_var; // 危険！関数終了後に無効になる  
}  
  
// OK: 静的変数または呼び出し元から受け取ったポインタを返す  
int\* safe\_function(int \*input)  
{  
 static int static\_var = 100;  
 return &static\_var; // 安全（静的変数）  
 // または return input; （引数として受け取ったポインタ）  
}

## デバッグのヒント

1. **アドレスの確認**: printf("%p\n", (void\*)ptr) でポインタの値を確認
2. **値の確認**: printf("%d\n", \*ptr) でポインタが指す値を確認
3. **境界チェック**: 配列アクセス前に必ず範囲をチェック
4. **コンパイラ警告**: -Wall -Wextra -pedantic を使用して警告を有効化

これらの解答例を参考に、ポインタの安全で効果的な使用方法を身につけてください。 ### ex10\_1\_pointer\_swap.c

/\*  
 \* ファイル名: ex10\_1\_pointer\_swap.c  
 \* 演習10-1: ポインタの基本操作  
 \* 説明: 2つの整数変数の値をポインタを使って交換  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\* ポインタを使った値の交換関数 \*/  
void swap\_with\_pointers(int \*ptr\_a, int \*ptr\_b)  
{  
 int temp;  
   
 printf(" swap関数内部:\n");  
 printf(" 交換前: \*ptr\_a = %d, \*ptr\_b = %d\n", \*ptr\_a, \*ptr\_b);  
 printf(" ポインタ: ptr\_a = %p, ptr\_b = %p\n", (void\*)ptr\_a, (void\*)ptr\_b);  
   
 temp = \*ptr\_a;  
 \*ptr\_a = \*ptr\_b;  
 \*ptr\_b = temp;  
   
 printf(" 交換後: \*ptr\_a = %d, \*ptr\_b = %d\n", \*ptr\_a, \*ptr\_b);  
}  
  
/\* 参考：値渡しでは交換できない例 \*/  
void swap\_by\_value(int a, int b)  
{  
 int temp;  
   
 printf(" 値渡し関数内部:\n");  
 printf(" 交換前: a = %d, b = %d\n", a, b);  
   
 temp = a;  
 a = b;  
 b = temp;  
   
 printf(" 交換後: a = %d, b = %d\n", a, b);  
 printf(" (この変更は呼び出し元には反映されません)\n");  
}  
  
int main(void)  
{  
 int x = 10, y = 20;  
 int \*ptr\_x, \*ptr\_y;  
 int numbers[4] = {100, 200, 300, 400};  
 int \*start, \*end;  
 int i; /\* C90: ループ変数をブロックの先頭で宣言 \*/  
   
 printf("===== ポインタによる値の交換デモ =====\n\n");  
   
 /\* 初期状態の表示 \*/  
 printf("1. 初期状態:\n");  
 printf(" x = %d (アドレス: %p)\n", x, (void\*)&x);  
 printf(" y = %d (アドレス: %p)\n", y, (void\*)&y);  
   
 /\* ポインタの初期化 \*/  
 ptr\_x = &x;  
 ptr\_y = &y;  
   
 printf("\n2. ポインタの初期化後:\n");  
 printf(" ptr\_x = %p (指す値: %d)\n", (void\*)ptr\_x, \*ptr\_x);  
 printf(" ptr\_y = %p (指す値: %d)\n", (void\*)ptr\_y, \*ptr\_y);  
   
 /\* ポインタを使った交換の実行 \*/  
 printf("\n3. ポインタを使った交換の実行:\n");  
 swap\_with\_pointers(ptr\_x, ptr\_y);  
   
 printf("\n4. 交換後の状態（main関数内）:\n");  
 printf(" x = %d (アドレス: %p)\n", x, (void\*)&x);  
 printf(" y = %d (アドレス: %p)\n", y, (void\*)&y);  
 printf(" \*ptr\_x = %d, \*ptr\_y = %d\n", \*ptr\_x, \*ptr\_y);  
   
 /\* 元に戻して値渡しとの比較 \*/  
 printf("\n===== 参考：値渡しとの比較 =====\n");  
   
 /\* 元の値に戻す \*/  
 x = 10; y = 20;  
 printf("\n5. 値を元に戻しました: x = %d, y = %d\n", x, y);  
   
 printf("\n6. 値渡しによる交換試行（効果なし）:\n");  
 swap\_by\_value(x, y);  
   
 printf("\n7. 値渡し後の状態（main関数内）:\n");  
 printf(" x = %d, y = %d (変更されていない)\n", x, y);  
   
 /\* アドレス演算子の使用例 \*/  
 printf("\n===== アドレス演算子の直接使用 =====\n");  
   
 printf("\n8. アドレス演算子を直接使った交換:\n");  
 printf(" 交換前: x = %d, y = %d\n", x, y);  
 swap\_with\_pointers(&x, &y);  
 printf(" 交換後: x = %d, y = %d\n", x, y);  
   
 /\* ポインタ演算の応用例 \*/  
 printf("\n===== ポインタ演算の応用 =====\n");  
   
 start = numbers;  
 end = numbers + 3;  
   
 printf("\n9. 配列の両端要素の交換:\n");  
 printf(" 配列: ");  
 for (i = 0; i < 4; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 printf(" start = %p (\*start = %d)\n", (void\*)start, \*start);  
 printf(" end = %p (\*end = %d)\n", (void\*)end, \*end);  
   
 swap\_with\_pointers(start, end);  
   
 printf(" 交換後の配列: ");  
 for (i = 0; i < 4; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. ポインタによる参照渡し:  
 - 関数で元の変数の値を変更可能  
 - アドレスを渡すことで間接アクセス  
   
2. 値渡しとの違い:  
 - 値渡しでは元の変数は変更されない  
 - ポインタ渡しでは元の変数が変更される  
   
3. アドレス演算子(&)の使用:  
 - &variable で変数のアドレスを取得  
 - 関数呼び出し時に直接使用可能  
   
4. 間接参照演算子(\*)の使用:  
 - \*pointer でポインタが指す値にアクセス  
 - 左辺値として使用すると値の変更が可能  
   
5. メモリアドレスの理解:  
 - 変数のアドレスは実行ごとに変わる可能性  
 - ポインタはアドレスを格納する変数  
   
6. 実用的な応用:  
 - 配列要素の交換  
 - 複数の値を同時に変更する関数  
 - データ構造の操作  
  
注意点:  
- ポインタは必ず有効なアドレスを指すよう初期化  
- NULLポインタの参照は避ける  
- ポインタが指すメモリが有効であることを確認  
\*/

### ex10\_2\_array\_reverse.c

/\*  
 \* ファイル名: ex10\_2\_array\_reverse.c  
 \* 演習10-2: 配列とポインタ  
 \* 説明: ポインタ演算を使って配列の要素を逆順に表示  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\* インデックス記法で配列を表示 \*/  
void print\_array\_index(int arr[], int size, char \*label)  
{  
 int i;  
   
 printf("%s (インデックス記法): ", label);  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
}  
  
/\* ポインタ記法で配列を表示 \*/  
void print\_array\_pointer(int \*ptr, int size, char \*label)  
{  
 int i;  
   
 printf("%s (ポインタ記法): ", label);  
 for (i = 0; i < size; i++) {  
 printf("%d ", \*(ptr + i));  
 }  
 printf("\n");  
}  
  
/\* ポインタ演算で配列を逆順表示 \*/  
void print\_array\_reverse\_pointer(int \*ptr, int size)  
{  
 int i;  
 int \*last\_ptr = ptr + size - 1; /\* 最後の要素を指す \*/  
   
 printf("逆順表示 (ポインタ演算): ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", \*last\_ptr);  
 last\_ptr--; /\* ポインタを前の要素に移動 \*/  
 }  
 printf("\n");  
}  
  
/\* ポインタインクリメントで配列を逆順表示 \*/  
void print\_array\_reverse\_increment(int \*ptr, int size)  
{  
 int \*reverse\_ptr = ptr + size - 1; /\* 最後の要素から開始 \*/  
   
 printf("逆順表示 (デクリメント): ");  
 while (reverse\_ptr >= ptr) {  
 printf("%d ", \*reverse\_ptr);  
 reverse\_ptr--;  
 }  
 printf("\n");  
}  
  
/\* 配列のアドレス情報を表示 \*/  
void show\_address\_info(int \*arr, int size)  
{  
 int i;  
   
 printf("\n=== アドレス情報 ===\n");  
 printf("配列名のアドレス: %p\n", (void\*)arr);  
 printf("&arr[0]のアドレス: %p\n", (void\*)&arr[0]);  
 printf("同じアドレス? %s\n", (arr == &arr[0]) ? "はい" : "いいえ");  
   
 printf("\n各要素のアドレスと値:\n");  
 for (i = 0; i < size; i++) {  
 printf(" arr[%d]: アドレス=%p, 値=%d, ポインタ記法=\*(arr+%d)=%d\n",  
 i, (void\*)&arr[i], arr[i], i, \*(arr + i));  
 }  
   
 printf("\nアドレス間隔の確認:\n");  
 for (i = 0; i < size - 1; i++) {  
 long byte\_diff = (char\*)&arr[i+1] - (char\*)&arr[i];  
 printf(" arr[%d]とarr[%d]の差: %ld バイト\n", i, i+1, byte\_diff);  
 }  
 printf(" int型のサイズ: %lu バイト\n", (unsigned long)sizeof(int));  
}  
  
/\* ポインタ演算の詳細確認 \*/  
void demonstrate\_pointer\_arithmetic(int \*arr, int size)  
{  
 int i;  
 int \*ptr = arr;  
   
 printf("\n=== ポインタ演算の詳細 ===\n");  
 printf("基準ポインタ ptr = %p\n", (void\*)ptr);  
   
 for (i = 0; i < size; i++) {  
 printf("ptr + %d = %p, \*(ptr + %d) = %d\n",  
 i, (void\*)(ptr + i), i, \*(ptr + i));  
 }  
   
 printf("\n負のオフセットでのアクセス:\n");  
 ptr = arr + size - 1; /\* 最後の要素を指す \*/  
 printf("基準ポインタ ptr = %p (最後の要素)\n", (void\*)ptr);  
   
 for (i = 0; i < size; i++) {  
 printf("ptr - %d = %p, \*(ptr - %d) = %d\n",  
 i, (void\*)(ptr - i), i, \*(ptr - i));  
 }  
}  
  
/\* 配列を実際に逆順に並び替える関数 \*/  
void reverse\_array\_inplace(int \*arr, int size)  
{  
 int \*start = arr;  
 int \*end = arr + size - 1;  
 int temp;  
   
 printf("\n=== 配列の実際の逆順並び替え ===\n");  
 printf("並び替え前: ");  
 print\_array\_pointer(arr, size, "");  
   
 while (start < end) {  
 printf("交換: \*%p(%d) ↔ \*%p(%d)\n",   
 (void\*)start, \*start, (void\*)end, \*end);  
   
 temp = \*start;  
 \*start = \*end;  
 \*end = temp;  
   
 start++;  
 end--;  
 }  
   
 printf("並び替え後: ");  
 print\_array\_pointer(arr, size, "");  
}  
  
int main(void)  
{  
 int numbers[] = {10, 20, 30, 40, 50, 60};  
 int backup[] = {10, 20, 30, 40, 50, 60}; /\* 元の配列のバックアップ \*/  
 int size = sizeof(numbers) / sizeof(numbers[0]);  
 int \*ptr = numbers;  
   
 printf("===== ポインタ演算による配列の逆順表示 =====\n\n");  
   
 /\* 配列サイズの動的計算 \*/  
 printf("配列サイズの動的計算:\n");  
 printf(" sizeof(numbers) = %lu バイト\n", (unsigned long)sizeof(numbers));  
 printf(" sizeof(numbers[0]) = %lu バイト\n", (unsigned long)sizeof(numbers[0]));  
 printf(" 要素数 = %d\n", size);  
   
 printf("\n1. 通常の表示:\n");  
 print\_array\_index(numbers, size, "元の配列");  
 print\_array\_pointer(ptr, size, "元の配列");  
   
 printf("\n2. 逆順表示の方法:\n");  
 print\_array\_reverse\_pointer(ptr, size);  
 print\_array\_reverse\_increment(ptr, size);  
   
 /\* アドレス情報の表示 \*/  
 show\_address\_info(numbers, size);  
   
 /\* ポインタ演算の詳細 \*/  
 demonstrate\_pointer\_arithmetic(numbers, size);  
   
 /\* 実際の逆順並び替え \*/  
 reverse\_array\_inplace(numbers, size);  
   
 printf("\n=== 元の配列との比較 ===\n");  
 print\_array\_pointer(backup, size, "元の配列");  
 print\_array\_pointer(numbers, size, "逆順配列");  
   
 /\* 配列を元に戻す \*/  
 reverse\_array\_inplace(numbers, size);  
 printf("\n再度逆順にして元に戻しました:\n");  
 print\_array\_pointer(numbers, size, "復元された配列");  
   
 /\* 部分配列の逆順表示 \*/  
 printf("\n=== 部分配列の逆順表示 ===\n");  
 {  
 int start\_index = 1;  
 int end\_index = 4;  
 int partial\_size = end\_index - start\_index + 1;  
 int i; /\* C90: ループ変数をブロックの先頭で宣言 \*/  
 int \*partial\_end;  
   
 printf("部分配列 [%d-%d] の逆順表示:\n", start\_index, end\_index);  
 printf("通常: ");  
 for (i = start\_index; i <= end\_index; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 printf("逆順: ");  
 for (i = end\_index; i >= start\_index; i--) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 printf("ポインタ演算: ");  
 partial\_end = numbers + end\_index;  
 for (i = 0; i < partial\_size; i++) {  
 printf("%d ", \*(partial\_end - i));  
 }  
 printf("\n");  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 配列とポインタの等価性:  
 - 配列名は先頭要素のポインタと同等  
 - arr[i] と \*(arr + i) は同じ意味  
   
2. ポインタ演算:  
 - ptr + n は n個後の要素を指す  
 - ptr - n は n個前の要素を指す  
 - ポインタ同士の差は要素数  
   
3. 逆順アクセスの方法:  
 - 最後の要素から順次デクリメント  
 - 負のオフセットを使用  
 - ポインタの境界チェック  
   
4. メモリレイアウト:  
 - 配列要素は連続したメモリに配置  
 - アドレス差 = 要素サイズ × 要素数の差  
   
5. 実用的な応用:  
 - 配列の逆順並び替え  
 - 部分配列の処理  
 - 動的なサイズ計算  
   
6. 安全なプログラミング:  
 - 配列の境界チェック (start < end)  
 - ポインタの有効性確認  
 - オーバーフロー・アンダーフローの防止  
  
注意点:  
- 配列の範囲外アクセスは未定義動作  
- ポインタ演算は型サイズを自動考慮  
- sizeof演算子は配列全体のサイズを返す  
\*/

### ex10\_3\_string\_manipulation.c

/\*  
 \* ex10\_3\_string\_manipulation.c  
 \*   
 \* 演習10-3: 文字列操作  
 \*   
 \* ポインタを使った文字列操作の実装  
 \* - カスタムstrlen関数  
 \* - 文字列逆順関数  
 \* - 文字列リテラルと文字配列の違い  
 \*   
 \* Author: Claude  
 \* Date: 2025-06-29  
 \*/  
  
#include <stdio.h>  
  
/\* カスタムstrlen関数：ポインタを使って文字列の長さを計算 \*/  
size\_t my\_strlen(const char \*str) {  
 const char \*p = str; /\* ポインタで文字列の先頭を記憶 \*/  
   
 /\* ヌル文字に到達するまでポインタを進める \*/  
 while (\*p != '\0') {  
 p++;  
 }  
   
 /\* ポインタの差分が文字列の長さ \*/  
 return p - str;  
}  
  
/\* 文字列を逆順にする関数：ポインタを使って実装 \*/  
void reverse\_string(char \*str) {  
 char \*start = str; /\* 文字列の先頭ポインタ \*/  
 char \*end = str + my\_strlen(str) - 1; /\* 文字列の末尾ポインタ \*/  
 char temp;  
   
 /\* 先頭と末尾のポインタが交差するまで文字を交換 \*/  
 while (start < end) {  
 /\* ポインタを使って文字を交換 \*/  
 temp = \*start;  
 \*start = \*end;  
 \*end = temp;  
   
 /\* ポインタを内側に移動 \*/  
 start++;  
 end--;  
 }  
}  
  
/\* 文字列情報を表示する関数 \*/  
void print\_string\_info(const char \*label, const char \*str) {  
 printf("%s: \"%s\"\n", label, str);  
 printf(" アドレス: %p\n", (void\*)str);  
 printf(" 長さ: %zu文字\n", my\_strlen(str));  
}  
  
int main(void) {  
 /\* 文字列リテラル（読み取り専用） \*/  
 const char \*literal = "Hello, World!";  
   
 /\* 文字配列（変更可能） \*/  
 char array1[] = "C Programming";  
 char array2[] = "Pointer Practice";  
 char array3[] = "12345";  
   
 printf("=== 文字列操作のデモンストレーション ===\n\n");  
   
 /\* 1. カスタムstrlen関数のテスト \*/  
 printf("1. 文字列の長さ計算（my\_strlen関数）\n");  
 printf("----------------------------------------\n");  
 print\_string\_info("文字列リテラル", literal);  
 print\_string\_info("文字配列1", array1);  
 print\_string\_info("文字配列2", array2);  
 print\_string\_info("文字配列3", array3);  
 printf("\n");  
   
 /\* 2. 文字列の逆順処理 \*/  
 printf("2. 文字列の逆順処理\n");  
 printf("----------------------------------------\n");  
   
 /\* 文字列リテラルは変更できないことを説明 \*/  
 printf("文字列リテラル \"%s\" は読み取り専用のため、\n", literal);  
 printf("逆順にすることはできません。\n\n");  
   
 /\* 文字配列は変更可能 \*/  
 printf("文字配列1の逆順処理:\n");  
 printf(" 元の文字列: \"%s\"\n", array1);  
 reverse\_string(array1);  
 printf(" 逆順文字列: \"%s\"\n\n", array1);  
   
 printf("文字配列2の逆順処理:\n");  
 printf(" 元の文字列: \"%s\"\n", array2);  
 reverse\_string(array2);  
 printf(" 逆順文字列: \"%s\"\n\n", array2);  
   
 printf("文字配列3の逆順処理:\n");  
 printf(" 元の文字列: \"%s\"\n", array3);  
 reverse\_string(array3);  
 printf(" 逆順文字列: \"%s\"\n\n", array3);  
   
 /\* 3. ポインタ操作の詳細説明 \*/  
 printf("3. ポインタ操作の仕組み\n");  
 printf("----------------------------------------\n");  
   
 /\* 文字列をポインタで走査する例 \*/  
 {  
 const char \*p;  
 int count = 0;  
   
 printf("文字列 \"%s\" の各文字:\n", array3);  
 for (p = array3; \*p != '\0'; p++) {  
 printf(" array3[%d] = '%c' (アドレス: %p)\n",   
 count, \*p, (void\*)p);  
 count++;  
 }  
 }  
   
 printf("\n");  
   
 /\* 4. 文字列リテラルと文字配列の違い \*/  
 printf("4. 文字列リテラルと文字配列の違い\n");  
 printf("----------------------------------------\n");  
 printf("文字列リテラル:\n");  
 printf(" - const char \*型で宣言\n");  
 printf(" - 読み取り専用メモリに配置\n");  
 printf(" - 変更しようとするとエラー\n\n");  
   
 printf("文字配列:\n");  
 printf(" - char array[]型で宣言\n");  
 printf(" - スタック上に配置\n");  
 printf(" - 変更可能\n");  
   
 return 0;  
}

### ex10\_4\_array\_functions.c

/\*  
 \* 演習10-4: 配列操作関数  
 \*   
 \* 配列を操作する関数をポインタを使って実装する演習です。  
 \* - 最大値・最小値を見つけてポインタを返す  
 \* - 平均値を計算する  
 \* - 統計情報をレポート形式で表示する  
 \*   
 \* 作成者: C言語学習者  
 \* 作成日: 2024年  
 \*/  
  
#include <stdio.h>  
  
/\* 関数プロトタイプ宣言 \*/  
int\* find\_max(int \*arr, int size);  
int\* find\_min(int \*arr, int size);  
double calculate\_average(int \*arr, int size);  
void print\_statistics(int \*arr, int size);  
  
/\*  
 \* 配列の最大値を見つけてそのポインタを返す  
 \* 引数: arr - 配列の先頭ポインタ, size - 配列のサイズ  
 \* 戻り値: 最大値へのポインタ  
 \*/  
int\* find\_max(int \*arr, int size) {  
 int \*max\_ptr = arr; /\* 最初は先頭要素を最大値と仮定 \*/  
 int i;  
   
 /\* 配列の全要素を走査 \*/  
 for (i = 1; i < size; i++) {  
 /\*  
 \* ポインタ演算を使った要素アクセス  
 \* arr + i は i番目の要素のアドレス  
 \* \*(arr + i) でその値を取得  
 \*/  
 if (\*(arr + i) > \*max\_ptr) {  
 max\_ptr = arr + i; /\* 新しい最大値のアドレスを記録 \*/  
 }  
 }  
   
 return max\_ptr; /\* 最大値のアドレスを返す \*/  
}  
  
/\*  
 \* 配列の最小値を見つけてそのポインタを返す  
 \* 引数: arr - 配列の先頭ポインタ, size - 配列のサイズ  
 \* 戻り値: 最小値へのポインタ  
 \*/  
int\* find\_min(int \*arr, int size) {  
 int \*min\_ptr = arr; /\* 最初は先頭要素を最小値と仮定 \*/  
 int i;  
   
 /\* 配列の全要素を走査 \*/  
 for (i = 1; i < size; i++) {  
 /\* ポインタを使った比較 \*/  
 if (\*(arr + i) < \*min\_ptr) {  
 min\_ptr = arr + i; /\* 新しい最小値のアドレスを記録 \*/  
 }  
 }  
   
 return min\_ptr; /\* 最小値のアドレスを返す \*/  
}  
  
/\*  
 \* 配列の平均値を計算する  
 \* 引数: arr - 配列の先頭ポインタ, size - 配列のサイズ  
 \* 戻り値: 平均値（double型）  
 \*/  
double calculate\_average(int \*arr, int size) {  
 int sum = 0;  
 int i;  
   
 /\* ポインタを使った要素の合計計算 \*/  
 for (i = 0; i < size; i++) {  
 sum += \*(arr + i); /\* ポインタ演算で各要素にアクセス \*/  
 }  
   
 /\* 平均値を計算して返す（double型にキャスト） \*/  
 return (double)sum / size;  
}  
  
/\*  
 \* 配列の統計情報を表示する  
 \* 最大値、最小値、平均値とそれらのポインタ情報を含む  
 \*/  
void print\_statistics(int \*arr, int size) {  
 int \*max\_ptr, \*min\_ptr;  
 double average;  
 int i;  
   
 printf("========== 配列統計レポート ==========\n");  
 printf("配列のサイズ: %d要素\n\n", size);  
   
 /\* 配列の内容を表示 \*/  
 printf("配列の内容:\n");  
 for (i = 0; i < size; i++) {  
 printf(" arr[%d] = %d (アドレス: %p)\n",   
 i, \*(arr + i), (void\*)(arr + i));  
 }  
 printf("\n");  
   
 /\* 最大値の情報 \*/  
 max\_ptr = find\_max(arr, size);  
 printf("最大値:\n");  
 printf(" 値: %d\n", \*max\_ptr);  
 printf(" アドレス: %p\n", (void\*)max\_ptr);  
 printf(" インデックス: %ld\n", (long)(max\_ptr - arr));  
 printf("\n");  
   
 /\* 最小値の情報 \*/  
 min\_ptr = find\_min(arr, size);  
 printf("最小値:\n");  
 printf(" 値: %d\n", \*min\_ptr);  
 printf(" アドレス: %p\n", (void\*)min\_ptr);  
 printf(" インデックス: %ld\n", (long)(min\_ptr - arr));  
 printf("\n");  
   
 /\* 平均値の情報 \*/  
 average = calculate\_average(arr, size);  
 printf("平均値: %.2f\n\n", average);  
   
 /\* ポインタ解析情報 \*/  
 printf("ポインタ解析:\n");  
 printf(" 配列の先頭アドレス: %p\n", (void\*)arr);  
 printf(" 最大値と先頭の差: %ld要素\n", (long)(max\_ptr - arr));  
 printf(" 最小値と先頭の差: %ld要素\n", (long)(min\_ptr - arr));  
 printf(" 最大値と最小値の差: %ld要素\n", (long)(max\_ptr - min\_ptr));  
   
 printf("======================================\n");  
}  
  
int main(void) {  
 /\* デモ用配列データ \*/  
 int numbers[] = {45, 23, 67, 89, 12, 34, 56, 78, 90, 1};  
 int size = sizeof(numbers) / sizeof(numbers[0]);  
   
 printf("配列操作関数のデモンストレーション\n");  
 printf("=====================================\n\n");  
   
 /\* 統計情報を表示 \*/  
 print\_statistics(numbers, size);  
   
 /\* 個別の関数呼び出しデモ \*/  
 printf("\n個別関数の使用例:\n");  
 printf("------------------\n");  
   
 /\* find\_max関数の使用例 \*/  
 {  
 int \*max = find\_max(numbers, size);  
 printf("find\_max()の結果: 値=%d, アドレス=%p\n",   
 \*max, (void\*)max);  
 }  
   
 /\* find\_min関数の使用例 \*/  
 {  
 int \*min = find\_min(numbers, size);  
 printf("find\_min()の結果: 値=%d, アドレス=%p\n",   
 \*min, (void\*)min);  
 }  
   
 /\* calculate\_average関数の使用例 \*/  
 {  
 double avg = calculate\_average(numbers, size);  
 printf("calculate\_average()の結果: %.2f\n", avg);  
 }  
   
 printf("\n");  
   
 /\* ポインタ概念の説明 \*/  
 printf("ポインタの重要な概念:\n");  
 printf("--------------------\n");  
 printf("1. 配列名は配列の先頭要素へのポインタとして扱われる\n");  
 printf("2. 関数に配列を渡すと、実際にはポインタが渡される\n");  
 printf("3. ポインタ演算により配列要素にアクセスできる\n");  
 printf("4. 関数からポインタを返すことで、配列内の特定要素を指示できる\n");  
   
 return 0;  
}

# 第11章 構造体とポインタ

## 演習問題

## 基礎問題

### 演習11-1: 学生情報管理

学生の情報（ID、名前、年齢、成績）を格納する構造体を定義し、ポインタを使って情報を表示・更新するプログラムを作成してください。

**要件:** - 学生情報を表す構造体を定義 - 学生情報を入力する関数を作成 - 学生情報を表示する関数を作成（ポインタ引数） - 成績を更新する関数を作成（ポインタ引数）

**期待される動作例:**

学生情報を入力してください:  
ID: 1001  
名前: 田中太郎  
年齢: 20  
成績: 85.5  
  
=== 学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 85.5点  
  
新しい成績を入力してください: 90.0  
成績を更新しました。  
  
=== 更新後の学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点

### 演習11-2: 座標計算

2D座標を表す構造体を定義し、2点間の距離を計算する関数をポインタを使って実装してください。

**要件:** - 座標を表す構造体（x, y）を定義 - 2点間の距離を計算する関数を作成（ポインタ引数） - 座標を移動する関数を作成（ポインタで更新） - math.hのsqrt関数を使用

**期待される動作例:**

点1の座標を入力 (x y): 0 0  
点2の座標を入力 (x y): 3 4  
  
点1: (0, 0)  
点2: (3, 4)  
2点間の距離: 5.00  
  
点1を移動します。  
移動量を入力 (dx dy): 1 1  
  
移動後の点1: (1, 1)  
新しい距離: 3.61

### 演習11-3: 商品管理

商品情報（コード、名前、価格、在庫）の構造体を作成し、構造体配列で複数商品を管理するプログラムを作成してください。

**要件:** - 商品情報を表す構造体を定義 - 最大10個の商品を管理できる配列を用意 - 商品を追加する関数を作成 - 全商品を表示する関数を作成 - 在庫を更新する関数を作成（商品コードで検索）

**期待される動作例:**

=== 商品管理システム ===  
1. 商品追加  
2. 商品一覧表示  
3. 在庫更新  
0. 終了  
選択: 1  
  
商品コード: 101  
商品名: ノートPC  
価格: 98000  
在庫数: 5  
商品を追加しました。  
  
選択: 2  
=== 商品一覧 ===  
コード: 101  
商品名: ノートPC  
価格: 98000円  
在庫: 5個

## 応用問題

### 演習11-4: 従業員データベース

従業員情報と部署情報をネストした構造体で管理し、部署別の給与統計を算出するプログラムを作成してください。

**要件:** - 部署情報を表す構造体（部署名、部署コード） - 従業員情報を表す構造体（ID、名前、部署情報、給与） - 部署別の平均給与を計算する関数 - 最高給与の従業員を検索する関数 - 構造体ポインタ配列を使用

### 演習11-5: 図書管理システム

本の情報（タイトル、著者、出版年、貸出状況）を管理し、検索・貸出・返却機能を実装してください。

**要件:** - 書籍情報を表す構造体を定義 - 貸出状況を管理（貸出中フラグ、借りた人のID） - タイトルで検索する関数 - 貸出処理を行う関数 - 返却処理を行う関数 - 貸出中の本一覧を表示する関数

### 演習11-6: 成績管理システム

学生と科目の構造体を使って、学生別・科目別の成績統計を管理するプログラムを作成してください。

**要件:** - 科目情報を表す構造体（科目名、科目コード、単位数） - 成績情報を表す構造体（学生ID、科目コード、点数、評価） - 学生別の平均点を計算する関数 - 科目別の平均点を計算する関数 - GPA計算機能（A=4.0, B=3.0, C=2.0, D=1.0, F=0.0）

## 発展問題

### 演習11-7: 動的配列システム

構造体ポインタ配列を動的に拡張できるシステムを実装してください。

**要件:** - 初期容量を持つ構造体ポインタ配列 - 容量が不足したら自動的に拡張する機能 - メモリ管理を適切に行う - 要素の追加・削除機能

### 演習11-8: データソート

構造体ポインタ配列を複数の条件（名前、年齢、成績など）でソートできるプログラムを作成してください。

**要件:** - 複数のソート基準を選択可能 - 昇順・降順の切り替え機能 - ソート用の比較関数を複数実装 - qsort関数を使用

### 演習11-9: 階層データ構造

会社組織（部署→チーム→従業員）のような階層構造を構造体とポインタで表現してください。

**要件:** - 部署、チーム、従業員の構造体を定義 - 階層関係をポインタで表現 - 組織図を表示する関数 - 特定の部署・チームの人数を集計する関数

## 提出方法

1. 各問題に対して別々のCファイルを作成
2. ファイル名は ex11\_1\_student.c, ex11\_2\_coordinate.c のように命名
3. 各プログラムの冒頭にコメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認

## コンパイル例

gcc -std=c90 -Wall -Wextra -pedantic ex11\_1\_student.c -o ex11\_1\_student  
./ex11\_1\_student

## ヒント

* 構造体のサイズに注意（パディング）
* ポインタ経由でのメンバーアクセスには->演算子を使用
* 構造体の比較は直接できないため、メンバーごとに比較
* 文字列のコピーにはstrcpy関数を使用
* 構造体を関数に渡す際は、効率のためポインタを使用することを推奨

## 解答例

このディレクトリには演習問題の解答例が含まれています。

## 解答ファイル一覧

### 基礎問題

1. **演習11-1: 学生情報管理**
   * <ex11_1_student.c> - C90準拠版
   * <ex11_1_student_c99.c> - C99準拠版
   * 構造体とポインタを使った基本的なデータ管理
2. **演習11-2: 座標計算**
   * <ex11_2_coordinate.c> - C90準拠版
   * <ex11_2_coordinate_c99.c> - C99準拠版
   * 構造体で座標を表現し、距離計算を実装
3. **演習11-3: 商品管理**
   * <ex11_3_product.c> - C90準拠版
   * <ex11_3_product_c99.c> - C99準拠版
   * 構造体配列による商品在庫管理システム

### 応用問題

1. **演習11-4: 従業員データベース**
   * ネストした構造体と統計計算
   * （未実装）
2. **演習11-5: 図書管理システム**
   * 検索・貸出・返却機能の実装
   * （未実装）
3. **演習11-6: 成績管理システム**
   * 学生別・科目別の成績統計とGPA計算
   * （未実装）

### 発展問題

1. **演習11-7: 動的配列システム**
   * 動的メモリ管理による拡張可能な配列
   * （未実装）
2. **演習11-8: データソート**
   * 複数条件でのソート機能
   * （未実装）
3. **演習11-9: 階層データ構造**
   * 会社組織の階層構造表現
   * （未実装）

## 学習のポイント

### C90版とC99版の主な違い

1. **構造体の初期化**
   * C90: 順序通りの初期化のみ
   * C99: 指定イニシャライザが使用可能
2. **bool型**
   * C90: intで代用
   * C99: stdbool.hのbool型
3. **変数宣言**
   * C90: ブロックの先頭でのみ
   * C99: 使用箇所で宣言可能
4. **可変長配列（VLA）**
   * C90: 未対応
   * C99: 実行時にサイズ決定可能
5. **inline関数**
   * C90: 未対応
   * C99: inline修飾子で最適化

### 各演習の重要概念

#### 演習11-1〜11-3（基礎）

* 構造体の定義と初期化
* ドット演算子とアロー演算子
* 構造体配列の操作
* ポインタによる効率的なデータ渡し

#### 演習11-4〜11-6（応用）

* ネストした構造体
* 構造体ポインタ配列
* 複雑なデータ構造の管理
* 統計処理とデータ分析

#### 演習11-7〜11-9（発展）

* 動的メモリ管理
* qsortによる汎用ソート
* 階層的なデータ構造
* より高度なポインタ操作

## コンパイルと実行

# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic ex11\_1\_student.c -o ex11\_1\_student  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic ex11\_1\_student\_c99.c -o ex11\_1\_student\_c99  
  
# 座標計算（数学ライブラリをリンク）  
gcc -std=c90 -Wall -Wextra -pedantic ex11\_2\_coordinate.c -o ex11\_2\_coordinate -lm  
  
# 実行  
./ex11\_1\_student

## 注意事項

* 構造体のサイズはパディングの影響を受けることがある
* 構造体の直接比較はできない（メンバーごとに比較）
* 文字列メンバーへの代入にはstrcpyを使用
* 大きな構造体は値渡しより、ポインタ渡しが効率的
* 動的メモリを使用する場合は、必ず解放処理を実装 ### ex11\_1\_student.c

/\*  
 \* ファイル名: ex11\_1\_student.c  
 \* 演習11-1: 学生情報管理  
 \* 説明: 構造体とポインタを使った学生情報の管理  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_NAME\_LEN 50  
  
/\* 学生情報を表す構造体 \*/  
struct Student {  
 int id;  
 char name[MAX\_NAME\_LEN];  
 int age;  
 double score;  
};  
  
/\* 学生情報を入力する関数（デモ版） \*/  
void input\_student(struct Student \*student)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("学生情報を入力してください:\n");  
   
 /\* デモ用: 固定値を設定 \*/  
 student->id = 1001;  
 student->age = 20;  
 student->score = 85.5;  
 strcpy(student->name, "田中太郎");  
   
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d\n", student->age);  
 printf("成績: %.1f\n", student->score);  
}  
  
/\* 学生情報を表示する関数 \*/  
void display\_student(const struct Student \*student)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("\n=== 学生情報 ===\n");  
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d歳\n", student->age);  
 printf("成績: %.1f点\n", student->score);  
}  
  
/\* 成績を更新する関数 \*/  
void update\_score(struct Student \*student, double new\_score)  
{  
 if (student == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 if (new\_score < 0.0 || new\_score > 100.0) {  
 printf("エラー: 成績は0〜100の範囲で入力してください\n");  
 return;  
 }  
   
 double old\_score = student->score;  
 student->score = new\_score;  
   
 printf("\n成績を更新しました。\n");  
 printf("変更前: %.1f点 → 変更後: %.1f点\n", old\_score, new\_score);  
}  
  
/\* 成績の評価を返す関数 \*/  
char get\_grade(double score)  
{  
 if (score >= 90.0) return 'A';  
 if (score >= 80.0) return 'B';  
 if (score >= 70.0) return 'C';  
 if (score >= 60.0) return 'D';  
 return 'F';  
}  
  
/\* 詳細情報を表示する関数 \*/  
void display\_detailed\_info(const struct Student \*student)  
{  
 if (student == NULL) {  
 return;  
 }  
   
 printf("\n=== 詳細な学生情報 ===\n");  
 printf("ID: %d\n", student->id);  
 printf("名前: %s\n", student->name);  
 printf("年齢: %d歳\n", student->age);  
 printf("成績: %.1f点 (評価: %c)\n", student->score, get\_grade(student->score));  
   
 /\* 成績に応じたコメント \*/  
 if (student->score >= 90.0) {  
 printf("コメント: 大変優秀です！\n");  
 } else if (student->score >= 80.0) {  
 printf("コメント: 良い成績です。\n");  
 } else if (student->score >= 70.0) {  
 printf("コメント: まずまずの成績です。\n");  
 } else if (student->score >= 60.0) {  
 printf("コメント: もう少し頑張りましょう。\n");  
 } else {  
 printf("コメント: 努力が必要です。\n");  
 }  
}  
  
int main(void)  
{  
 struct Student student;  
 double new\_score = 90.0; /\* デモ用: 固定値 \*/  
   
 printf("===== 学生情報管理システム =====\n\n");  
   
 /\* 学生情報の入力 \*/  
 input\_student(&student);  
   
 /\* 学生情報の表示 \*/  
 display\_student(&student);  
   
 /\* 成績更新のデモ \*/  
 printf("\n成績を更新しますか？ (y/n): y\n");  
 printf("新しい成績を入力してください: %.1f\n", new\_score);  
   
 /\* 成績の更新 \*/  
 update\_score(&student, new\_score);  
   
 /\* 更新後の情報表示 \*/  
 printf("\n=== 更新後の学生情報 ===\n");  
 printf("ID: %d\n", student.id);  
 printf("名前: %s\n", student.name);  
 printf("年齢: %d歳\n", student.age);  
 printf("成績: %.1f点\n", student.score);  
   
 /\* 詳細情報の表示 \*/  
 display\_detailed\_info(&student);  
   
 /\* 追加機能：複数の学生の比較 \*/  
 printf("\n\n=== 複数学生の管理デモ ===\n");  
   
 struct Student students[3] = {  
 {2001, "田中太郎", 20, 85.5},  
 {2002, "佐藤花子", 21, 92.0},  
 {2003, "山田次郎", 19, 78.0}  
 };  
   
 int i;  
 double total\_score = 0.0;  
 struct Student \*top\_student = &students[0];  
   
 printf("\n全学生の一覧:\n");  
 for (i = 0; i < 3; i++) {  
 printf("%d. %s (%.1f点)\n",   
 students[i].id, students[i].name, students[i].score);  
 total\_score += students[i].score;  
   
 /\* 最高得点の学生を記録 \*/  
 if (students[i].score > top\_student->score) {  
 top\_student = &students[i];  
 }  
 }  
   
 printf("\n統計情報:\n");  
 printf("平均点: %.1f点\n", total\_score / 3);  
 printf("最高得点: %s (%.1f点)\n", top\_student->name, top\_student->score);  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 学生情報管理システム =====  
  
学生情報を入力してください:  
ID: 1001  
名前: 田中太郎  
年齢: 20  
成績: 85.5  
  
=== 学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 85.5点  
  
成績を更新しますか？ (y/n): y  
新しい成績を入力してください: 90.0  
  
成績を更新しました。  
変更前: 85.5点 → 変更後: 90.0点  
  
=== 更新後の学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点  
  
=== 詳細な学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
成績: 90.0点 (評価: A)  
コメント: 大変優秀です！  
  
=== 複数学生の管理デモ ===  
  
全学生の一覧:  
2001. 田中太郎 (85.5点)  
2002. 佐藤花子 (92.0点)  
2003. 山田次郎 (78.0点)  
  
統計情報:  
平均点: 85.2点  
最高得点: 佐藤花子 (92.0点)  
\*/

### ex11\_2\_coordinate.c

/\*  
 \* ファイル名: ex11\_2\_coordinate.c  
 \* 演習11-2: 座標計算  
 \* 説明: 2D座標を表す構造体を定義し、2点間の距離を計算  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <math.h>  
  
/\* 2D座標を表す構造体 \*/  
struct Point2D {  
 double x;  
 double y;  
};  
  
/\* 座標を表示する関数 \*/  
void display\_point(const struct Point2D \*point, const char \*name)  
{  
 if (point == NULL || name == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 printf("%s: (%.2f, %.2f)\n", name, point->x, point->y);  
}  
  
/\* 2点間の距離を計算する関数（ポインタ引数） \*/  
double calculate\_distance(const struct Point2D \*p1, const struct Point2D \*p2)  
{  
 double dx, dy;  
   
 if (p1 == NULL || p2 == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return -1.0;  
 }  
   
 dx = p2->x - p1->x;  
 dy = p2->y - p1->y;  
   
 return sqrt(dx \* dx + dy \* dy);  
}  
  
/\* 座標を移動する関数（ポインタで更新） \*/  
void move\_point(struct Point2D \*point, double dx, double dy)  
{  
 if (point == NULL) {  
 printf("エラー: 無効なポインタです\n");  
 return;  
 }  
   
 point->x += dx;  
 point->y += dy;  
}  
  
/\* 中点を計算する関数 \*/  
struct Point2D calculate\_midpoint(const struct Point2D \*p1, const struct Point2D \*p2)  
{  
 struct Point2D midpoint;  
   
 if (p1 == NULL || p2 == NULL) {  
 midpoint.x = 0.0;  
 midpoint.y = 0.0;  
 return midpoint;  
 }  
   
 midpoint.x = (p1->x + p2->x) / 2.0;  
 midpoint.y = (p1->y + p2->y) / 2.0;  
   
 return midpoint;  
}  
  
/\* 原点からの距離を計算する関数 \*/  
double distance\_from\_origin(const struct Point2D \*point)  
{  
 struct Point2D origin = {0.0, 0.0};  
 return calculate\_distance(&origin, point);  
}  
  
/\* 座標の象限を判定する関数 \*/  
int get\_quadrant(const struct Point2D \*point)  
{  
 if (point == NULL) {  
 return 0;  
 }  
   
 if (point->x > 0 && point->y > 0) {  
 return 1; /\* 第1象限 \*/  
 } else if (point->x < 0 && point->y > 0) {  
 return 2; /\* 第2象限 \*/  
 } else if (point->x < 0 && point->y < 0) {  
 return 3; /\* 第3象限 \*/  
 } else if (point->x > 0 && point->y < 0) {  
 return 4; /\* 第4象限 \*/  
 } else {  
 return 0; /\* 軸上または原点 \*/  
 }  
}  
  
int main(void)  
{  
 struct Point2D point1, point2;  
 struct Point2D midpoint;  
 double distance;  
 double dx, dy;  
 int quadrant;  
   
 printf("===== 2D座標計算プログラム =====\n\n");  
   
 /\* デモ用: 固定値で初期化 \*/  
 printf("点1の座標を入力 (x y): 0 0\n");  
 point1.x = 0.0;  
 point1.y = 0.0;  
   
 printf("点2の座標を入力 (x y): 3 4\n");  
 point2.x = 3.0;  
 point2.y = 4.0;  
   
 /\* 座標の表示 \*/  
 printf("\n");  
 display\_point(&point1, "点1");  
 display\_point(&point2, "点2");  
   
 /\* 2点間の距離を計算 \*/  
 distance = calculate\_distance(&point1, &point2);  
 printf("2点間の距離: %.2f\n", distance);  
   
 /\* 中点の計算 \*/  
 midpoint = calculate\_midpoint(&point1, &point2);  
 display\_point(&midpoint, "\n2点の中点");  
   
 /\* 原点からの距離 \*/  
 printf("\n原点からの距離:\n");  
 printf("点1: %.2f\n", distance\_from\_origin(&point1));  
 printf("点2: %.2f\n", distance\_from\_origin(&point2));  
   
 /\* 座標の移動 \*/  
 printf("\n点1を移動します。\n");  
 printf("移動量を入力 (dx dy): 1 1\n");  
 dx = 1.0;  
 dy = 1.0;  
   
 move\_point(&point1, dx, dy);  
   
 printf("\n移動後の");  
 display\_point(&point1, "点1");  
   
 /\* 新しい距離の計算 \*/  
 distance = calculate\_distance(&point1, &point2);  
 printf("新しい距離: %.2f\n", distance);  
   
 /\* 象限の判定 \*/  
 printf("\n象限の判定:\n");  
 quadrant = get\_quadrant(&point1);  
 printf("点1は");  
 if (quadrant > 0) {  
 printf("第%d象限にあります\n", quadrant);  
 } else {  
 printf("軸上または原点にあります\n");  
 }  
   
 quadrant = get\_quadrant(&point2);  
 printf("点2は");  
 if (quadrant > 0) {  
 printf("第%d象限にあります\n", quadrant);  
 } else {  
 printf("軸上または原点にあります\n");  
 }  
   
 /\* 追加デモ: 複数点の管理 \*/  
 printf("\n\n=== 複数点の管理デモ ===\n");  
   
 struct Point2D points[] = {  
 {1.0, 2.0},  
 {-3.0, 4.0},  
 {-2.0, -5.0},  
 {4.0, -1.0},  
 {0.0, 3.0}  
 };  
 int num\_points = 5;  
 int i, j;  
 double max\_distance = 0.0;  
 int max\_i = 0, max\_j = 1;  
   
 /\* 全ての点を表示 \*/  
 printf("\n登録された点:\n");  
 for (i = 0; i < num\_points; i++) {  
 char name[20];  
 sprintf(name, "点%d", i + 1);  
 display\_point(&points[i], name);  
 }  
   
 /\* 最も離れた2点を見つける \*/  
 for (i = 0; i < num\_points - 1; i++) {  
 for (j = i + 1; j < num\_points; j++) {  
 distance = calculate\_distance(&points[i], &points[j]);  
 if (distance > max\_distance) {  
 max\_distance = distance;  
 max\_i = i;  
 max\_j = j;  
 }  
 }  
 }  
   
 printf("\n最も離れた2点:\n");  
 printf("点%d (%.1f, %.1f) と 点%d (%.1f, %.1f)\n",  
 max\_i + 1, points[max\_i].x, points[max\_i].y,  
 max\_j + 1, points[max\_j].x, points[max\_j].y);  
 printf("距離: %.2f\n", max\_distance);  
   
 /\* 全点の重心を計算 \*/  
 struct Point2D centroid = {0.0, 0.0};  
 for (i = 0; i < num\_points; i++) {  
 centroid.x += points[i].x;  
 centroid.y += points[i].y;  
 }  
 centroid.x /= num\_points;  
 centroid.y /= num\_points;  
   
 printf("\n全点の重心: (%.2f, %.2f)\n", centroid.x, centroid.y);  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 2D座標計算プログラム =====  
  
点1の座標を入力 (x y): 0 0  
点2の座標を入力 (x y): 3 4  
  
点1: (0.00, 0.00)  
点2: (3.00, 4.00)  
2点間の距離: 5.00  
  
2点の中点: (1.50, 2.00)  
  
原点からの距離:  
点1: 0.00  
点2: 5.00  
  
点1を移動します。  
移動量を入力 (dx dy): 1 1  
  
移動後の点1: (1.00, 1.00)  
新しい距離: 3.61  
  
象限の判定:  
点1は第1象限にあります  
点2は第1象限にあります  
  
=== 複数点の管理デモ ===  
  
登録された点:  
点1: (1.00, 2.00)  
点2: (-3.00, 4.00)  
点3: (-2.00, -5.00)  
点4: (4.00, -1.00)  
点5: (0.00, 3.00)  
  
最も離れた2点:  
点2 (-3.0, 4.0) と 点3 (-2.0, -5.0)  
距離: 9.06  
  
全点の重心: (0.00, 0.60)  
\*/

### ex11\_3\_product.c

/\*  
 \* ファイル名: ex11\_3\_product.c  
 \* 演習11-3: 商品管理  
 \* 説明: 商品情報の構造体配列で複数商品を管理  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
#define MAX\_PRODUCTS 10  
#define MAX\_NAME\_LEN 50  
  
/\* 商品情報を表す構造体 \*/  
struct Product {  
 int code;  
 char name[MAX\_NAME\_LEN];  
 double price;  
 int stock;  
};  
  
/\* 商品を追加する関数 \*/  
int add\_product(struct Product products[], int \*count,   
 int code, const char \*name, double price, int stock)  
{  
 int i;  
   
 if (\*count >= MAX\_PRODUCTS) {  
 printf("エラー: これ以上商品を追加できません（最大%d個）\n", MAX\_PRODUCTS);  
 return 0;  
 }  
   
 if (name == NULL || strlen(name) == 0) {  
 printf("エラー: 商品名が無効です\n");  
 return 0;  
 }  
   
 if (price < 0) {  
 printf("エラー: 価格は0以上である必要があります\n");  
 return 0;  
 }  
   
 if (stock < 0) {  
 printf("エラー: 在庫数は0以上である必要があります\n");  
 return 0;  
 }  
   
 /\* 商品コードの重複チェック \*/  
 for (i = 0; i < \*count; i++) {  
 if (products[i].code == code) {  
 printf("エラー: 商品コード %d は既に登録されています\n", code);  
 return 0;  
 }  
 }  
   
 /\* 商品を追加 \*/  
 products[\*count].code = code;  
 strncpy(products[\*count].name, name, MAX\_NAME\_LEN - 1);  
 products[\*count].name[MAX\_NAME\_LEN - 1] = '\0';  
 products[\*count].price = price;  
 products[\*count].stock = stock;  
   
 (\*count)++;  
 printf("商品を追加しました。\n");  
   
 return 1;  
}  
  
/\* 全商品を表示する関数 \*/  
void display\_all\_products(const struct Product products[], int count)  
{  
 int i;  
 double total\_value = 0.0;  
   
 if (count == 0) {  
 printf("登録されている商品はありません。\n");  
 return;  
 }  
   
 printf("\n=== 商品一覧 ===\n");  
 printf("%-8s %-20s %10s %8s\n", "コード", "商品名", "価格", "在庫");  
 printf("--------------------------------------------------------\n");  
   
 for (i = 0; i < count; i++) {  
 printf("%-8d %-20s %10.0f円 %8d個\n",  
 products[i].code,  
 products[i].name,  
 products[i].price,  
 products[i].stock);  
 total\_value += products[i].price \* products[i].stock;  
 }  
   
 printf("--------------------------------------------------------\n");  
 printf("登録商品数: %d個\n", count);  
 printf("在庫総額: %.0f円\n", total\_value);  
}  
  
/\* 商品コードで検索する関数 \*/  
struct Product\* find\_product\_by\_code(struct Product products[], int count, int code)  
{  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (products[i].code == code) {  
 return &products[i];  
 }  
 }  
   
 return NULL;  
}  
  
/\* 在庫を更新する関数 \*/  
int update\_stock(struct Product products[], int count, int code, int new\_stock)  
{  
 struct Product \*product;  
 int old\_stock;  
   
 if (new\_stock < 0) {  
 printf("エラー: 在庫数は0以上である必要があります\n");  
 return 0;  
 }  
   
 product = find\_product\_by\_code(products, count, code);  
   
 if (product == NULL) {  
 printf("エラー: 商品コード %d は見つかりませんでした\n", code);  
 return 0;  
 }  
   
 old\_stock = product->stock;  
 product->stock = new\_stock;  
   
 printf("商品「%s」の在庫を更新しました: %d個 → %d個\n",  
 product->name, old\_stock, new\_stock);  
   
 return 1;  
}  
  
/\* 在庫が少ない商品を表示する関数 \*/  
void display\_low\_stock(const struct Product products[], int count, int threshold)  
{  
 int i;  
 int found = 0;  
   
 printf("\n=== 在庫が%d個以下の商品 ===\n", threshold);  
   
 for (i = 0; i < count; i++) {  
 if (products[i].stock <= threshold) {  
 if (!found) {  
 printf("%-8s %-20s %8s\n", "コード", "商品名", "在庫");  
 printf("----------------------------------------\n");  
 found = 1;  
 }  
 printf("%-8d %-20s %8d個\n",  
 products[i].code,  
 products[i].name,  
 products[i].stock);  
 }  
 }  
   
 if (!found) {  
 printf("該当する商品はありません。\n");  
 }  
}  
  
/\* 価格帯で商品を検索する関数 \*/  
void search\_by\_price\_range(const struct Product products[], int count,  
 double min\_price, double max\_price)  
{  
 int i;  
 int found = 0;  
   
 printf("\n=== 価格 %.0f円 〜 %.0f円 の商品 ===\n", min\_price, max\_price);  
   
 for (i = 0; i < count; i++) {  
 if (products[i].price >= min\_price && products[i].price <= max\_price) {  
 if (!found) {  
 printf("%-8s %-20s %10s\n", "コード", "商品名", "価格");  
 printf("----------------------------------------\n");  
 found = 1;  
 }  
 printf("%-8d %-20s %10.0f円\n",  
 products[i].code,  
 products[i].name,  
 products[i].price);  
 }  
 }  
   
 if (!found) {  
 printf("該当する商品はありません。\n");  
 }  
}  
  
int main(void)  
{  
 struct Product products[MAX\_PRODUCTS];  
 int product\_count = 0;  
 int choice;  
   
 printf("===== 商品管理システム =====\n\n");  
   
 /\* デモ用: 初期データを追加 \*/  
 add\_product(products, &product\_count, 101, "ノートPC", 98000, 5);  
 add\_product(products, &product\_count, 102, "USBメモリ", 2500, 20);  
 add\_product(products, &product\_count, 103, "マウス", 1500, 15);  
 add\_product(products, &product\_count, 104, "キーボード", 3500, 8);  
 add\_product(products, &product\_count, 105, "モニター", 25000, 3);  
   
 /\* メニューのシミュレーション \*/  
 printf("\n=== 商品管理システム ===\n");  
 printf("1. 商品追加\n");  
 printf("2. 商品一覧表示\n");  
 printf("3. 在庫更新\n");  
 printf("0. 終了\n");  
   
 /\* デモ: 商品一覧を表示 \*/  
 printf("選択: 2\n");  
 display\_all\_products(products, product\_count);  
   
 /\* デモ: 新商品を追加 \*/  
 printf("\n選択: 1\n");  
 printf("\n商品コード: 106\n");  
 printf("商品名: ヘッドセット\n");  
 printf("価格: 4800\n");  
 printf("在庫数: 12\n");  
 add\_product(products, &product\_count, 106, "ヘッドセット", 4800, 12);  
   
 /\* デモ: 在庫更新 \*/  
 printf("\n選択: 3\n");  
 printf("\n更新する商品コード: 103\n");  
 printf("新しい在庫数: 25\n");  
 update\_stock(products, product\_count, 103, 25);  
   
 /\* 更新後の一覧表示 \*/  
 printf("\n選択: 2\n");  
 display\_all\_products(products, product\_count);  
   
 /\* 追加機能のデモ \*/  
 printf("\n\n=== 追加機能デモ ===\n");  
   
 /\* 在庫が少ない商品 \*/  
 display\_low\_stock(products, product\_count, 5);  
   
 /\* 価格帯検索 \*/  
 search\_by\_price\_range(products, product\_count, 2000, 5000);  
   
 /\* 商品統計 \*/  
 printf("\n\n=== 商品統計 ===\n");  
 if (product\_count > 0) {  
 int i;  
 double total\_value = 0.0;  
 double max\_price = products[0].price;  
 double min\_price = products[0].price;  
 int total\_stock = 0;  
 const char \*expensive\_product = products[0].name;  
 const char \*cheapest\_product = products[0].name;  
   
 for (i = 0; i < product\_count; i++) {  
 total\_value += products[i].price \* products[i].stock;  
 total\_stock += products[i].stock;  
   
 if (products[i].price > max\_price) {  
 max\_price = products[i].price;  
 expensive\_product = products[i].name;  
 }  
 if (products[i].price < min\_price) {  
 min\_price = products[i].price;  
 cheapest\_product = products[i].name;  
 }  
 }  
   
 printf("登録商品数: %d個\n", product\_count);  
 printf("在庫総数: %d個\n", total\_stock);  
 printf("在庫総額: %.0f円\n", total\_value);  
 printf("平均単価: %.0f円\n", total\_value / total\_stock);  
 printf("最高価格商品: %s (%.0f円)\n", expensive\_product, max\_price);  
 printf("最低価格商品: %s (%.0f円)\n", cheapest\_product, min\_price);  
 }  
   
 printf("\n選択: 0\n");  
 printf("システムを終了します。\n");  
   
 return 0;  
}  
  
/\*  
実行例:  
===== 商品管理システム =====  
  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
商品を追加しました。  
  
=== 商品管理システム ===  
1. 商品追加  
2. 商品一覧表示  
3. 在庫更新  
0. 終了  
選択: 2  
  
=== 商品一覧 ===  
コード 商品名 価格 在庫  
--------------------------------------------------------  
101 ノートPC 98000円 5個  
102 USBメモリ 2500円 20個  
103 マウス 1500円 15個  
104 キーボード 3500円 8個  
105 モニター 25000円 3個  
--------------------------------------------------------  
登録商品数: 5個  
在庫総額: 693500円  
  
選択: 1  
  
商品コード: 106  
商品名: ヘッドセット  
価格: 4800  
在庫数: 12  
商品を追加しました。  
  
選択: 3  
  
更新する商品コード: 103  
新しい在庫数: 25  
商品「マウス」の在庫を更新しました: 15個 → 25個  
  
選択: 2  
  
=== 商品一覧 ===  
コード 商品名 価格 在庫  
--------------------------------------------------------  
101 ノートPC 98000円 5個  
102 USBメモリ 2500円 20個  
103 マウス 1500円 25個  
104 キーボード 3500円 8個  
105 モニター 25000円 3個  
106 ヘッドセット 4800円 12個  
--------------------------------------------------------  
登録商品数: 6個  
在庫総額: 766100円  
  
=== 追加機能デモ ===  
  
=== 在庫が5個以下の商品 ===  
コード 商品名 在庫  
----------------------------------------  
101 ノートPC 5個  
105 モニター 3個  
  
=== 価格 2000円 〜 5000円 の商品 ===  
コード 商品名 価格  
----------------------------------------  
102 USBメモリ 2500円  
104 キーボード 3500円  
106 ヘッドセット 4800円  
  
=== 商品統計 ===  
登録商品数: 6個  
在庫総数: 73個  
在庫総額: 766100円  
平均単価: 10495円  
最高価格商品: ノートPC (98000円)  
最低価格商品: マウス (1500円)  
  
選択: 0  
システムを終了します。  
\*/

# 第12章 関数ポインタ

## 演習問題

## 演習の目的

* 関数ポインタの基本概念を理解する
* コールバック関数の実装方法を習得する
* 関数ポインタ配列を活用した動的関数選択を学ぶ
* 実行時関数切り替えシステムの設計を理解する

## 演習問題

### 演習12-1: 関数ポインタの基本操作

関数ポインタを使って複数の関数を動的に呼び出すプログラムを作成してください。

**要件:** - 2つの整数を受け取り、結果を返す関数を複数定義する - 関数ポインタを使ってこれらの関数を呼び出す - 関数ポインタ配列を使った実装も含める

**実装すべき関数:** - int maximum(int a, int b) - 大きい方の値を返す - int minimum(int a, int b) - 小さい方の値を返す  
- int power(int a, int b) - aのb乗を返す（簡単な実装で可）

**ファイル名:** ex12\_1\_basic\_function\_pointer.c

### 演習12-2: 関数選択システム

文字に基づいて関数を選択し実行するシステムを実装してください。

**要件:** - 文字（‘+’, ‘-’, ’\*‘,’/’など）に基づいて関数を選択 - 選択された関数を実行して結果を表示 - 無効な文字が指定された場合のエラーハンドリング - 構造体を使った関数ポインタ管理

**ファイル名:** ex12\_2\_function\_selector.c

### 演習12-3: 配列処理のコールバック

コールバック関数を使って配列の各要素に異なる処理を適用するプログラムを作成してください。

**要件:** - 整数配列を処理する関数を複数定義 - コールバック関数として配列処理関数に渡す - 処理前後の配列の状態を表示 - 動的な処理選択機能

**実装すべき処理:** - 各要素を倍にする - 各要素から1を引く - 各要素の符号を反転する

**ファイル名:** ex12\_3\_array\_callback.c

### 演習12-4: 関数ポインタ配列を使った計算機

関数ポインタ配列を使用した計算機プログラムを作成してください。

**要件:** - 関数ポインタの配列を定義 - インデックスを指定して演算を選択 - 複数の演算を連続で実行可能 - 演算履歴を表示する機能 - 統計情報の収集と表示

**実装すべき演算:** - 加算、減算、乗算、除算 - ゼロ除算のエラーハンドリング

**ファイル名:** ex12\_4\_calculator\_function\_array.c

## チャレンジ問題

### チャレンジ12-5: ソートアルゴリズム選択システム

異なるソートアルゴリズムを関数ポインタで切り替えるシステムを作成してください。

**要件:** - バブルソート、選択ソート、挿入ソートを実装 - 比較関数も関数ポインタで指定（昇順/降順） - ソート前後の配列状態を表示 - アルゴリズムの性能比較機能

**ファイル名:** ex12\_5\_sort\_algorithms.c

### チャレンジ12-6: イベント駆動システム

コールバック関数を使ったイベント駆動システムを実装してください。

**要件:** - 複数種類のイベント（開始、停止、エラー、警告） - イベントタイプごとに異なるハンドラーを登録 - イベント発生時に適切なハンドラーを呼び出し - イベントの優先度機能

**ファイル名:** ex12\_6\_event\_system.c

## 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 ex12\_1.c, ex12\_2.c のような形式で
3. コメントで問題番号と簡単な説明を記載
4. C90規格でコンパイルできることを確認
5. C99版も作成し、\_c99.c サフィックスを付ける

## コンパイル例

### C90準拠

gcc -std=c90 -Wall -Wextra -pedantic ex12\_1\_basic\_function\_pointer.c -o ex12\_1\_basic\_function\_pointer

### C99準拠

gcc -std=c99 -Wall -Wextra -pedantic ex12\_1\_basic\_function\_pointer\_c99.c -o ex12\_1\_basic\_function\_pointer\_c99

## 学習のポイント

### 関数ポインタの基本

1. **宣言**: int (\*func\_ptr)(int, int);
2. **初期化**: func\_ptr = function\_name;
3. **呼び出し**: result = func\_ptr(a, b);
4. **配列**: int (\*operations[])(int, int) = {add, sub, mul};

### コールバック関数

1. **概念**: 関数を引数として渡す仕組み
2. **活用**: 動的な処理選択、カスタマイズ可能な処理
3. **設計**: 関数ポインタを引数に取る関数の実装

### 実用的なテクニック

1. **関数テーブル**: 構造体による関数ポインタ管理
2. **エラーハンドリング**: 無効な関数ポインタの検出
3. **状態管理**: 関数ポインタによる状態遷移
4. **性能**: 関数ポインタ使用時のオーバーヘッド考慮

## 注意事項

* 関数ポインタは初期化前に使用しない
* 型の一致を厳密に確認する
* NULL ポインタチェックを忘れずに
* デバッグ時は関数名の確認が困難な場合がある

## 実用的な応用

これらの技術は以下のような場面で使用されます：

* **GUI フレームワーク**: イベントハンドラ
* **組み込みシステム**: 割り込みハンドラ
* **ゲーム開発**: 状態管理、AI 行動パターン
* **システムプログラミング**: プラグインシステム

## 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13章: 高度なプログラミング技法](../advanced/) - 関数ポインタを活用したより複雑なシステム設計 - マルチスレッドプログラミングでの関数ポインタ活用

## 解答例

このディレクトリには、第12章の演習問題の解答例が含まれています。各解答例はC90標準準拠版とC99版の両方を提供し、すべて非対話型のデモ版として実装されています。

## 解答例一覧

### 演習12-1: 関数ポインタの基本操作

* **C90版**: <ex12_1_basic_function_pointer.c>
* **C99版**: <ex12_1_basic_function_pointer_c99.c>
* **内容**: 関数ポインタを使った動的関数呼び出しのデモ（maximum, minimum, power関数）
* **学習ポイント**: 関数ポインタの宣言、初期化、呼び出し、配列での管理

### 演習12-2: 関数選択システム

* **C90版**: <ex12_2_function_selector.c>
* **C99版**: <ex12_2_function_selector_c99.c>
* **内容**: 文字に基づく関数選択システム（構造体による関数ポインタ管理）
* **学習ポイント**: 構造体と関数ポインタの組み合わせ、エラーハンドリング

### 演習12-3: 配列処理のコールバック

* **C90版**: <ex12_3_array_callback.c>
* **C99版**: <ex12_3_array_callback_c99.c>
* **内容**: コールバック関数を使った配列要素処理（倍加、減算、符号反転）
* **学習ポイント**: コールバック関数の概念、関数ポインタを引数として渡す方法

### 演習12-4: 関数ポインタ配列を使った計算機

* **C90版**: <ex12_4_calculator_function_array.c>
* **C99版**: <ex12_4_calculator_function_array_c99.c>
* **内容**: 関数ポインタ配列による計算機（履歴管理、統計機能付き）
* **学習ポイント**: 関数ポインタ配列、履歴管理、連続計算

## 実装上の特徴

### 非対話型デモ版について

すべての解答例は、クラッシュを避けるため scanf() による対話型入力を排除し、予め定義された値を使用するデモ版として実装されています。

### 教育的配慮

* 各ファイルには詳細なコメントと学習ポイントを記載
* C90とC99の違いを明確に示すコード例
* 実用的な応用例を含むデモンストレーション

### セキュリティ対策

* ゼロ除算エラーのハンドリング
* 配列境界の適切なチェック
* 無効な関数ポインタの検出

## C90版とC99版の主な違い

### C90版の特徴

* 変数宣言は関数やブロックの先頭で行う
* /\* \*/ 形式のコメントのみ使用
* for文の初期化部で変数宣言不可
* bool型は独自定義または整数型で代替

### C99版の特徴

* 変数宣言を使用箇所で可能
* // 形式のコメントも使用可能
* for文内での変数宣言が可能
* <stdbool.h>のbool型を使用
* 指定初期化子による構造体初期化
* 可変長配列（VLA）のサポート

## コンパイル方法

### C90版のコンパイル

gcc -Wall -Wextra -pedantic -std=c90 -o program\_name source\_file.c

### C99版のコンパイル

gcc -Wall -Wextra -pedantic -std=c99 -o program\_name source\_file\_c99.c

## 学習のポイント

1. **関数ポインターの基本**
   * 関数ポインターの宣言方法
   * 関数アドレスの取得と呼び出し
   * 関数ポインターの型定義（typedef）
2. **関数ポインターの応用**
   * コールバック関数の実装
   * 関数ポインター配列の活用
   * 動的な関数選択
3. **設計パターン**
   * イベント駆動プログラミング
   * プラグインアーキテクチャ
   * 状態マシンパターン
4. **実践的な活用**
   * エラーハンドリング
   * 拡張性の高い設計
   * 保守性を考慮したコード構造

## 注意事項

* これらの解答例は一つの実装方法を示したものです
* 実際の開発では、要件に応じて適切な実装を選択してください
* エラーハンドリングは基本的なレベルに留めています
* メモリ管理が必要な場合は、適切な解放処理を追加してください ### ex12\_1\_basic\_function\_pointer.c

/\*  
 \* ファイル名: ex12\_1\_basic\_function\_pointer.c  
 \* 演習12-1: 関数ポインタの基本操作  
 \* 説明: 関数ポインタを使って複数の関数を動的に呼び出すデモプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 演算関数群 \*/  
int maximum(int a, int b)  
{  
 return (a > b) ? a : b;  
}  
  
int minimum(int a, int b)  
{  
 return (a < b) ? a : b;  
}  
  
int power(int a, int b)  
{  
 /\* 簡単な累乗実装（正の整数のみ） \*/  
 int result = 1;  
 int i;  
   
 if (b < 0) {  
 return 0; /\* 負の指数は0を返す \*/  
 }  
   
 for (i = 0; i < b; i++) {  
 result \*= a;  
 }  
   
 return result;  
}  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int (\*operation)(int, int);  
 int a, b, result;  
   
 printf("=== 関数ポインタの基本操作デモ ===\n");  
   
 /\* デモ用の値（実際のプログラムでは入力を受け取る） \*/  
 a = 8;  
 b = 3;  
   
 printf("対象の値: a = %d, b = %d\n\n", a, b);  
   
 /\* 1. maximum関数のテスト \*/  
 operation = maximum;  
 result = operation(a, b);  
 printf("maximum(%d, %d) = %d\n", a, b, result);  
   
 /\* 2. minimum関数のテスト \*/  
 operation = minimum;  
 result = operation(a, b);  
 printf("minimum(%d, %d) = %d\n", a, b, result);  
   
 /\* 3. power関数のテスト \*/  
 operation = power;  
 result = operation(a, b);  
 printf("power(%d, %d) = %d\n", a, b, result);  
   
 /\* 関数ポインタ配列のデモ \*/  
 {  
 int (\*operations[])(int, int) = {maximum, minimum, power};  
 char \*names[] = {"maximum", "minimum", "power"};  
 int i;  
   
 printf("\n=== 関数ポインタ配列デモ ===\n");  
 for (i = 0; i < 3; i++) {  
 result = operations[i](a, b);  
 printf("%s(%d, %d) = %d\n", names[i], a, b, result);  
 }  
 }  
   
 /\* 異なる値での演算デモ \*/  
 printf("\n=== 異なる値での演算例 ===\n");  
 {  
 int test\_values[][2] = {{10, 5}, {3, 7}, {2, 4}, {15, 15}};  
 int num\_tests = sizeof(test\_values) / sizeof(test\_values[0]);  
 int i;  
   
 for (i = 0; i < num\_tests; i++) {  
 int x = test\_values[i][0];  
 int y = test\_values[i][1];  
   
 printf("値: %d, %d\n", x, y);  
 printf(" max: %d, min: %d, pow: %d\n",   
 maximum(x, y), minimum(x, y), power(x, y));  
 }  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 関数ポインタの宣言と初期化  
2. 関数ポインタを通じた間接関数呼び出し  
3. 関数ポインタ配列の活用  
4. 実行時の動的な関数選択  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 関数ポインタの型宣言: int (\*operation)(int, int)  
- 関数名は関数のアドレスを表す  
- デモ版として固定値を使用（実際は入力を受け取る）  
- 配列を使った複数の関数ポインタ管理  
\*/

### ex12\_2\_function\_selector.c

/\*  
 \* ファイル名: ex12\_2\_function\_selector.c  
 \* 演習12-2: 関数選択システム  
 \* 説明: 文字に基づいて関数を選択し実行するデモプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 演算関数群 \*/  
int add(int a, int b)  
{  
 return a + b;  
}  
  
int subtract(int a, int b)  
{  
 return a - b;  
}  
  
int multiply(int a, int b)  
{  
 return a \* b;  
}  
  
int divide\_safe(int a, int b)  
{  
 if (b == 0) {  
 printf("エラー: ゼロ除算を検出\n");  
 return 0;  
 }  
 return a / b;  
}  
  
/\* 関数ポインタと対応する文字を管理する構造体 \*/  
typedef struct {  
 char op\_char;  
 int (\*func)(int, int);  
 char \*name;  
} OperationMapping;  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 OperationMapping operations[] = {  
 {'+', add, "加算"},  
 {'-', subtract, "減算"},   
 {'\*', multiply, "乗算"},  
 {'/', divide\_safe, "除算"}  
 };  
 int num\_operations = sizeof(operations) / sizeof(operations[0]);  
 int a, b, result;  
 int i, j;  
 char demo\_ops[] = {'+', '-', '\*', '/', 'X'}; /\* X は無効な演算子 \*/  
 int num\_demos = sizeof(demo\_ops) / sizeof(demo\_ops[0]);  
   
 printf("=== 関数選択システムデモ ===\n");  
   
 /\* デモ用の値 \*/  
 a = 12;  
 b = 4;  
   
 printf("対象の値: a = %d, b = %d\n\n", a, b);  
   
 /\* 各演算子のデモ \*/  
 for (i = 0; i < num\_demos; i++) {  
 char op = demo\_ops[i];  
 int found = 0;  
   
 printf("演算子 '%c' を選択:\n", op);  
   
 /\* 対応する関数を検索 \*/  
 for (j = 0; j < num\_operations; j++) {  
 if (operations[j].op\_char == op) {  
 result = operations[j].func(a, b);  
 printf(" %s: %d %c %d = %d\n",   
 operations[j].name, a, op, b, result);  
 found = 1;  
 break;  
 }  
 }  
   
 if (!found) {  
 printf(" エラー: 無効な演算子です\n");  
 }  
 printf("\n");  
 }  
   
 /\* 関数ポインタ配列を使った方法 \*/  
 printf("=== 関数ポインタ配列を使った実装 ===\n");  
 {  
 int (\*operation\_funcs[])(int, int) = {add, subtract, multiply, divide\_safe};  
 char operators[] = {'+', '-', '\*', '/'};  
 char \*names[] = {"加算", "減算", "乗算", "除算"};  
   
 for (i = 0; i < 4; i++) {  
 result = operation\_funcs[i](a, b);  
 printf("%s (%c): %d %c %d = %d\n",   
 names[i], operators[i], a, operators[i], b, result);  
 }  
 }  
   
 /\* 異なる値での演算例 \*/  
 printf("\n=== 異なる値での演算例 ===\n");  
 {  
 int test\_values[][2] = {{20, 5}, {7, 3}, {15, 0}};  
 int num\_tests = sizeof(test\_values) / sizeof(test\_values[0]);  
   
 for (i = 0; i < num\_tests; i++) {  
 int x = test\_values[i][0];  
 int y = test\_values[i][1];  
   
 printf("値: %d, %d\n", x, y);  
 printf(" 加算: %d, 減算: %d, 乗算: %d, 除算: %d\n",  
 add(x, y), subtract(x, y), multiply(x, y), divide\_safe(x, y));  
 }  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 文字による関数選択システム  
2. 構造体を使った関数ポインタ管理  
3. エラーハンドリングの実装  
4. 動的な関数呼び出し  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 構造体配列による関数ポインタの管理  
- ループによる関数検索と実行  
- デモ版として固定値を使用（実際は入力を受け取る）  
- 無効な演算子のエラーハンドリング  
\*/

### ex12\_3\_array\_callback.c

/\*  
 \* ファイル名: ex12\_3\_array\_callback.c  
 \* 演習12-3: 配列処理のコールバック  
 \* 説明: コールバック関数を使って配列の各要素に異なる処理を適用するプログラム  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
#define ARRAY\_SIZE 6  
  
/\* 配列処理用のコールバック関数群 \*/  
void double\_element(int \*element)  
{  
 \*element \*= 2;  
}  
  
void decrement\_element(int \*element)  
{  
 \*element -= 1;  
}  
  
void negate\_element(int \*element)  
{  
 \*element = -(\*element);  
}  
  
/\* 配列を表示する関数 \*/  
void print\_array(int \*arr, int size, char \*description)  
{  
 int i;  
   
 printf("%s: [", description);  
 for (i = 0; i < size; i++) {  
 printf("%d", arr[i]);  
 if (i < size - 1) {  
 printf(", ");  
 }  
 }  
 printf("]\n");  
}  
  
/\* 配列の各要素にコールバック関数を適用する関数 \*/  
void apply\_to\_array(int \*arr, int size, void (\*callback)(int \*), char \*operation\_name)  
{  
 int i;  
   
 printf("\n=== %s を適用 ===\n", operation\_name);  
 print\_array(arr, size, "処理前");  
   
 for (i = 0; i < size; i++) {  
 callback(&arr[i]);  
 }  
   
 print\_array(arr, size, "処理後");  
}  
  
/\* 配列を初期値にリセットする関数 \*/  
void reset\_array(int \*arr, int \*original, int size)  
{  
 int i;  
 for (i = 0; i < size; i++) {  
 arr[i] = original[i];  
 }  
}  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int original\_array[ARRAY\_SIZE] = {1, 2, 3, 4, 5, 6};  
 int work\_array[ARRAY\_SIZE];  
 int i;  
   
 /\* コールバック関数ポインタ配列 \*/  
 void (\*operations[])(int \*) = {double\_element, decrement\_element, negate\_element};  
 char \*operation\_names[] = {"各要素を倍にする", "各要素から1を引く", "各要素の符号を反転"};  
 int num\_operations = sizeof(operations) / sizeof(operations[0]);  
   
 printf("=== 配列処理のコールバックデモ ===\n");  
   
 /\* 初期配列の表示 \*/  
 print\_array(original\_array, ARRAY\_SIZE, "初期配列");  
   
 /\* 各操作を順次実行 \*/  
 for (i = 0; i < num\_operations; i++) {  
 /\* 作業用配列に初期値をコピー \*/  
 reset\_array(work\_array, original\_array, ARRAY\_SIZE);  
   
 /\* コールバック関数を適用 \*/  
 apply\_to\_array(work\_array, ARRAY\_SIZE, operations[i], operation\_names[i]);  
 }  
   
 /\* 複数の操作を連続適用するデモ \*/  
 printf("\n=== 複数操作の連続適用デモ ===\n");  
 reset\_array(work\_array, original\_array, ARRAY\_SIZE);  
 print\_array(work\_array, ARRAY\_SIZE, "開始");  
   
 /\* 1. 倍にする \*/  
 for (i = 0; i < ARRAY\_SIZE; i++) {  
 double\_element(&work\_array[i]);  
 }  
 print\_array(work\_array, ARRAY\_SIZE, "倍にした後");  
   
 /\* 2. 1を引く \*/  
 for (i = 0; i < ARRAY\_SIZE; i++) {  
 decrement\_element(&work\_array[i]);  
 }  
 print\_array(work\_array, ARRAY\_SIZE, "1を引いた後");  
   
 /\* 3. 符号を反転 \*/  
 for (i = 0; i < ARRAY\_SIZE; i++) {  
 negate\_element(&work\_array[i]);  
 }  
 print\_array(work\_array, ARRAY\_SIZE, "符号反転後");  
   
 /\* 関数ポインタを使った動的選択のデモ \*/  
 printf("\n=== 動的関数選択デモ ===\n");  
 {  
 int selection\_pattern[] = {0, 1, 2, 0, 1, 2}; /\* 操作の選択パターン \*/  
   
 reset\_array(work\_array, original\_array, ARRAY\_SIZE);  
 print\_array(work\_array, ARRAY\_SIZE, "開始");  
   
 for (i = 0; i < ARRAY\_SIZE; i++) {  
 int op\_index = selection\_pattern[i];  
 printf("要素[%d]: %s を適用\n", i, operation\_names[op\_index]);  
 operations[op\_index](&work\_array[i]);  
 }  
   
 print\_array(work\_array, ARRAY\_SIZE, "最終結果");  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. コールバック関数の概念と実装  
2. 関数ポインタを引数として渡す方法  
3. 配列処理での関数ポインタ活用  
4. 動的な処理選択システム  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- void (\*callback)(int \*) 型の関数ポインタ  
- コールバック関数は配列要素を直接変更  
- 関数ポインタ配列による複数操作の管理  
- デモ版として予め定義された処理パターンを使用  
\*/

### ex12\_4\_calculator\_function\_array.c

/\*  
 \* ファイル名: ex12\_4\_calculator\_function\_array.c  
 \* 演習12-4: 関数ポインタ配列を使った計算機  
 \* 説明: 関数ポインタ配列を使用した計算機プログラム（デモ版）  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
#define MAX\_OPERATIONS 4  
#define MAX\_HISTORY 10  
  
/\* 計算履歴を記録する構造体 \*/  
typedef struct {  
 int operand1;  
 int operand2;  
 char operation;  
 int result;  
} CalculationHistory;  
  
/\* 演算関数群 \*/  
int add(int a, int b)  
{  
 return a + b;  
}  
  
int subtract(int a, int b)  
{  
 return a - b;  
}  
  
int multiply(int a, int b)  
{  
 return a \* b;  
}  
  
int divide\_safe(int a, int b)  
{  
 if (b == 0) {  
 printf("エラー: ゼロ除算\n");  
 return 0;  
 }  
 return a / b;  
}  
  
/\* 履歴を表示する関数 \*/  
void print\_history(CalculationHistory \*history, int count)  
{  
 int i;  
   
 printf("\n=== 計算履歴 ===\n");  
 if (count == 0) {  
 printf("履歴はありません\n");  
 return;  
 }  
   
 for (i = 0; i < count; i++) {  
 printf("%d. %d %c %d = %d\n",   
 i + 1,   
 history[i].operand1,   
 history[i].operation,  
 history[i].operand2,  
 history[i].result);  
 }  
}  
  
/\* 統計情報を表示する関数 \*/  
void print\_statistics(CalculationHistory \*history, int count)  
{  
 int i;  
 int operation\_count[MAX\_OPERATIONS] = {0}; /\* 各演算の回数 \*/  
 char operations[] = {'+', '-', '\*', '/'};  
 char \*operation\_names[] = {"加算", "減算", "乗算", "除算"};  
   
 printf("\n=== 統計情報 ===\n");  
 printf("総計算回数: %d回\n", count);  
   
 /\* 各演算の回数をカウント \*/  
 for (i = 0; i < count; i++) {  
 int j;  
 for (j = 0; j < MAX\_OPERATIONS; j++) {  
 if (history[i].operation == operations[j]) {  
 operation\_count[j]++;  
 break;  
 }  
 }  
 }  
   
 /\* 演算別統計を表示 \*/  
 for (i = 0; i < MAX\_OPERATIONS; i++) {  
 printf("%s (%c): %d回\n",   
 operation\_names[i], operations[i], operation\_count[i]);  
 }  
}  
  
int main(void)  
{  
 /\* 変数の宣言（C90では先頭で宣言） \*/  
 int (\*operations[])(int, int) = {add, subtract, multiply, divide\_safe};  
 char op\_symbols[] = {'+', '-', '\*', '/'};  
 char \*op\_names[] = {"加算", "減算", "乗算", "除算"};  
   
 CalculationHistory history[MAX\_HISTORY];  
 int history\_count = 0;  
   
 /\* デモ用の計算データ \*/  
 int demo\_data[][3] = {  
 {10, 5, 0}, /\* 10 + 5 \*/  
 {15, 3, 1}, /\* 15 - 3 \*/  
 {4, 6, 2}, /\* 4 \* 6 \*/  
 {20, 4, 3}, /\* 20 / 4 \*/  
 {8, 0, 3}, /\* 8 / 0 (ゼロ除算テスト) \*/  
 {7, 2, 2}, /\* 7 \* 2 \*/  
 {100, 25, 1}, /\* 100 - 25 \*/  
 };  
 int num\_demos = sizeof(demo\_data) / sizeof(demo\_data[0]);  
 int i;  
   
 printf("=== 関数ポインタ配列計算機デモ ===\n");  
   
 /\* デモ計算の実行 \*/  
 for (i = 0; i < num\_demos && history\_count < MAX\_HISTORY; i++) {  
 int a = demo\_data[i][0];  
 int b = demo\_data[i][1];  
 int op\_index = demo\_data[i][2];  
   
 if (op\_index >= 0 && op\_index < MAX\_OPERATIONS) {  
 int result = operations[op\_index](a, b);  
 char op\_char = op\_symbols[op\_index];  
   
 printf("計算 %d: %d %c %d = %d (%s)\n",   
 i + 1, a, op\_char, b, result, op\_names[op\_index]);  
   
 /\* 履歴に記録 \*/  
 history[history\_count].operand1 = a;  
 history[history\_count].operand2 = b;  
 history[history\_count].operation = op\_char;  
 history[history\_count].result = result;  
 history\_count++;  
 }  
 }  
   
 /\* 履歴表示 \*/  
 print\_history(history, history\_count);  
   
 /\* 統計情報表示 \*/  
 print\_statistics(history, history\_count);  
   
 /\* 関数ポインタ配列の活用例 \*/  
 printf("\n=== 関数ポインタ配列の特徴デモ ===\n");  
 {  
 int test\_a = 12;  
 int test\_b = 3;  
   
 printf("値 %d と %d での全演算結果:\n", test\_a, test\_b);  
 for (i = 0; i < MAX\_OPERATIONS; i++) {  
 int result = operations[i](test\_a, test\_b);  
 printf(" %s: %d %c %d = %d\n",   
 op\_names[i], test\_a, op\_symbols[i], test\_b, result);  
 }  
 }  
   
 /\* 連続計算のデモ \*/  
 printf("\n=== 連続計算デモ ===\n");  
 {  
 int value = 10;  
 int operands[] = {2, 3, 4, 2};  
 int ops[] = {0, 2, 1, 3}; /\* +, \*, -, / \*/  
   
 printf("初期値: %d\n", value);  
 for (i = 0; i < 4; i++) {  
 int old\_value = value;  
 value = operations[ops[i]](value, operands[i]);  
 printf("ステップ %d: %d %c %d = %d\n",   
 i + 1, old\_value, op\_symbols[ops[i]], operands[i], value);  
 }  
 printf("最終結果: %d\n", value);  
 }  
   
 return 0;  
}  
  
/\*  
学習ポイント:  
1. 関数ポインタ配列による動的関数選択  
2. 構造体を使った履歴管理  
3. 統計情報の収集と表示  
4. 連続計算での関数ポインタ活用  
  
実装のポイント:  
- C90準拠のため、すべての変数を関数先頭で宣言  
- 関数ポインタ配列の初期化と活用  
- 構造体配列による履歴管理  
- デモ版として予め定義された計算パターンを使用  
- ゼロ除算エラーのハンドリング  
\*/

# 第13章 複数ファイル・発展技術

## 演習問題

## 基礎課題

### 演習13-1: プリプロセッサマクロの基本

以下の仕様を満たすプログラムを作成してください：

**仕様：** - 数学的な定数をマクロで定義する（PI、E、黄金比など） - 基本的な計算マクロを定義する（面積、体積など） - 条件付きコンパイルでデバッグ機能を切り替える - 文字列化マクロと連結マクロを活用する

**実装すべき機能：** - 円の面積・周長計算 - 球の体積・表面積計算 - デバッグ情報の出力（条件付き） - マクロによる型安全な最大値・最小値関数

**ファイル名：** ex13\_1\_macro\_basics.c

### 演習13-2: 安全なメモリ操作マクロ

メモリ操作の安全性を向上させるマクロ群を作成してください：

**仕様：** - NULL チェック付きメモリ割り当て - 配列境界チェック付きアクセス - 自動的なメモリ解放 - メモリリーク検出機能

**実装すべきマクロ：** - SAFE\_MALLOC(size) - NULLチェック付きmalloc - SAFE\_FREE(ptr) - NULLクリア付きfree - ARRAY\_BOUNDS\_CHECK(arr, index, size) - 境界チェック - MEMORY\_LEAK\_TRACKER - 簡単なリーク検出

**ファイル名：** ex13\_2\_safe\_memory.c

### 演習13-3: 基本的なメモリプール

固定サイズオブジェクト用の簡単なメモリプールを実装してください：

**仕様：** - 事前に確保されたメモリ領域からオブジェクトを割り当て - フリーリストによる高速な割り当て・解放 - プールの使用状況を表示する機能 - 初期化・終了処理

**実装すべき機能：** - プールの初期化と終了処理 - オブジェクトの取得と返却 - 使用状況の表示 - エラーハンドリング

**ファイル名：** ex13\_3\_memory\_pool.c

## 応用課題

### 演習13-4: 汎用的なプリプロセッサライブラリ

さまざまな用途に使える汎用的なマクロライブラリを作成してください：

**仕様：** - 型判定マクロ（コンパイル時） - 汎用的なスワップマクロ - ループ展開マクロ - アサーション機能付きマクロ - ベンチマーク計測マクロ

**実装すべき機能：** - TYPE\_CHECK(a, b) - 型の一致確認 - GENERIC\_SWAP(a, b) - 任意の型のスワップ - REPEAT(n, code) - コードの繰り返し展開 - BENCHMARK\_BLOCK(name) - ブロックの実行時間測定 - STATIC\_ASSERT(condition, message) - コンパイル時アサーション

**ファイル名：** ex13\_4\_generic\_macros.c

### 演習13-5: 高性能メモリアロケーター

パフォーマンスを重視したカスタムメモリアロケーターを実装してください：

**仕様：** - サイズ別メモリプール（小・中・大オブジェクト） - メモリの断片化を最小限に抑える仕組み - アロケーション統計情報の収集 - スレッドセーフ対応（簡易版）

**実装すべき機能：** - 複数サイズのメモリプール管理 - First Fit / Best Fit アルゴリズム - メモリ使用統計とレポート機能 - デバッグモードでの詳細トレース

**ファイル名：** ex13\_5\_allocator.c

### 演習13-6: キャッシュフレンドリーなデータ構造

CPU キャッシュ効率を考慮したデータ構造を実装してください：

**仕様：** - 配列ベースの動的配列（vector 風） - キャッシュラインを意識したメモリレイアウト - プリフェッチを活用した高速アクセス - メモリプールとの連携

**実装すべき機能：** - 動的な要素追加・削除 - キャッシュ効率的な反復処理 - バルク操作（一括挿入・削除） - パフォーマンス測定機能

**ファイル名：** ex13\_6\_cache\_vector.c

## 挑戦課題

### 演習13-7: プリプロセッサベースのDSL

プリプロセッサを使ってドメイン固有言語（DSL）を作成してください：

**仕様：** - 状態機械を記述するDSL - イベント駆動システムの記述 - 自動的なコード生成 - コンパイル時検証機能

**実装例：**

STATE\_MACHINE(TrafficLight)  
 STATE(Red) TIMEOUT() -> Yellow  
 STATE(Yellow) TIMEOUT() -> Green   
 STATE(Green) TIMEOUT() -> Red  
END\_STATE\_MACHINE  
  
EVENT\_HANDLER(ButtonPress)   
 // イベント処理コード  
END\_EVENT\_HANDLER

**ファイル名：** ex13\_7\_dsl.c

### 演習13-8: 動的メモリ管理フレームワーク

ガベージコレクション機能付きのメモリ管理フレームワークを実装してください：

**仕様：** - 参照カウント式ガベージコレクション - 循環参照の検出と解決 - 弱参照（weak reference）のサポート - メモリプレッシャー対応

**実装すべき機能：** - オブジェクトの自動管理 - 参照カウントの増減 - 循環参照のマーク&スイープ - メモリ不足時の自動解放

**ファイル名：** ex13\_8\_gc\_framework.c

### 演習13-9: リアルタイムメモリアロケーター

リアルタイムシステム向けの決定的メモリアロケーターを実装してください：

**仕様：** - O(1) での割り当て・解放保証 - メモリ断片化の完全排除 - 予測可能なメモリ使用量 - 割り込み処理からの安全な使用

**実装すべき機能：** - 固定時間でのメモリ操作 - 事前確保によるメモリプール - 優先度別メモリ管理 - リアルタイム統計情報

**ファイル名：** ex13\_9\_realtime.c

## 評価基準

### 基礎課題（各10点）

* 正しく動作する：5点
* コードの可読性：3点
* エラーハンドリング：2点

### 応用課題（各15点）

* 機能の完全性：5点
* パフォーマンス：4点
* 設計の優秀さ：3点
* 拡張性：3点

### 挑戦課題（各20点）

* 実装の完成度：8点
* 創意工夫：5点
* 技術的難易度：4点
* 実用性：3点

## 提出方法

1. 各課題を指定されたファイル名で作成
2. コンパイル用のMakefileを作成
3. 実行例とパフォーマンス結果を含むREADME.mdを作成
4. すべてのファイルをsolutions/フォルダーに配置

## 学習のポイント

### プリプロセッサの活用

* マクロの安全な使用方法
* 条件付きコンパイルの効果的な活用
* コード生成の自動化
* デバッグ支援機能

### メモリ管理の最適化

* メモリプールの設計と実装
* キャッシュ効率を考慮したデータ構造
* パフォーマンス測定と改善
* メモリリークの検出と防止

### 高度なC言語技法

* 型安全なプログラミング
* コンパイル時計算の活用
* ハードウェア特性を考慮した実装
* システムプログラミング技法

## 参考資料

### プリプロセッサ高度技法

// X-マクロパターン  
#define COLORS \  
 X(RED, "赤") \  
 X(GREEN, "緑") \  
 X(BLUE, "青")  
  
typedef enum {  
#define X(name, desc) COLOR\_##name,  
 COLORS  
#undef X  
 COLOR\_COUNT  
} Color;  
  
const char\* color\_names[] = {  
#define X(name, desc) desc,  
 COLORS  
#undef X  
};

### メモリアライメント最適化

// キャッシュライン境界でのアライメント  
#define CACHE\_LINE\_SIZE 64  
#define CACHE\_ALIGNED \_\_attribute\_\_((aligned(CACHE\_LINE\_SIZE)))  
  
typedef struct CACHE\_ALIGNED {  
 int frequently\_accessed\_data;  
 char padding[CACHE\_LINE\_SIZE - sizeof(int)];  
} CacheOptimizedStruct;

### 高性能メモリ操作

// SIMD命令を使った高速メモリコピー  
#include <immintrin.h>  
  
void fast\_memcpy(void\* dst, const void\* src, size\_t size) {  
 // AVXを使った実装例  
 if (size >= 32 && ((intptr\_t)dst & 31) == 0 && ((intptr\_t)src & 31) == 0) {  
 // 32バイト境界でアラインされている場合の最適化  
 }  
}

## よくある間違いとその対策

### 1. マクロの副作用

**間違い：**

#define SQUARE(x) x \* x  
int a = 5;  
int result = SQUARE(++a); // a が2回インクリメントされる

**正解：**

#define SQUARE(x) ({ \  
 typeof(x) \_temp = (x); \  
 \_temp \* \_temp; \  
})

### 2. メモリアライメントの無視

**間違い：**

char buffer[100];  
int\* ptr = (int\*)&buffer[1]; // 不正なアライメント

**正解：**

#include <stdalign.h>  
alignas(int) char buffer[100];  
int\* ptr = (int\*)buffer;

### 3. キャッシュ効率の無視

**間違い：**

// 列優先アクセス（キャッシュミスが多発）  
for (int j = 0; j < cols; j++) {  
 for (int i = 0; i < rows; i++) {  
 matrix[i][j] = value;  
 }  
}

**正解：**

// 行優先アクセス（キャッシュフレンドリー）  
for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 matrix[i][j] = value;  
 }  
}

これらの課題を通じて、C言語の高度な機能を習得し、パフォーマンスを意識したプログラミングスキルを身につけてください。

## 解答例

## 演習13-1: 高度なプリプロセッサマクロ

**ファイル**: ex13\_1\_macro\_basics.c, ex13\_1\_macro\_basics\_c99.c

基本的なプリプロセッサマクロの実装です。 - 数学定数、計算マクロ、ビット操作 - デバッグ支援マクロ（条件付きコンパイル） - C90版：標準準拠のマクロ実装 - C99版：可変引数マクロ、\_\_func\_\_マクロ活用

## 演習13-2: 安全なメモリ操作マクロ

**ファイル**: ex13\_2\_safe\_memory.c, ex13\_2\_safe\_memory\_c99.c

メモリ安全性を高めるマクロセットの実装です。 - NULLチェック付きメモリ操作 - 配列境界チェック - メモリリーク追跡システム - C90版：基本的な安全性チェック - C99版：inline関数、可変引数マクロによる拡張

## 演習13-3: シンプルなメモリプール

**ファイル**: ex13\_3\_memory\_pool.c, ex13\_3\_memory\_pool\_c99.c

固定サイズオブジェクト用メモリプールの実装です。 - O(1)での割り当て・解放 - フリーリスト管理 - フラグメンテーション対策 - C90版：基本的なプール実装 - C99版：型安全性の向上、統計機能追加

## 演習13-4: 汎用的なプリプロセッサライブラリ

**ファイル**: ex13\_4\_generic\_macros.c, ex13\_4\_generic\_macros\_c99.c

高度なマクロプログラミング技法の実装です。 - 型チェック、汎用操作マクロ - ループ展開、ベンチマークマクロ - C90版：memcpyベースの汎用swap - C99版：\_Generic、typeof、複合リテラル活用

## 演習13-5: 高性能メモリアロケーター

**ファイル**: ex13\_5\_allocator.c, ex13\_5\_allocator\_c99.c

マルチプール型カスタムアロケーターの実装です。 - サイズ別プール管理 - 高速割り当て・解放 - 詳細な統計情報 - C90版：基本的なプール選択アルゴリズム - C99版：inline最適化、restrict修飾子活用

## 演習13-6: キャッシュフレンドリーなデータ構造

**ファイル**: ex13\_6\_cache\_vector.c, ex13\_6\_cache\_vector\_c99.c

キャッシュ効率を考慮した動的配列の実装です。 - キャッシュラインアライメント - プリフェッチヒント - バルク操作の最適化 - C90版：基本的なアライメント対応 - C99版：フレキシブル配列メンバー、VLA活用

## 演習13-7: プリプロセッサベースのDSL

**ファイル**: ex13\_7\_dsl.c, ex13\_7\_dsl\_c99.c

ドメイン固有言語の実装例です。 - 状態機械DSL - イベント駆動システムDSL - ワークフローDSL - C90版：基本的なマクロDSL - C99版：可変引数マクロによる表現力向上

## 演習13-8: ガベージコレクション機能付きメモリ管理

**ファイル**: ex13\_8\_gc\_framework.c, ex13\_8\_gc\_framework\_c99.c

3つのGCアルゴリズムの実装です。 - マーク&スイープGC - 参照カウント方式 - 世代別GC - C90版：基本的なGCアルゴリズム - C99版：インクリメンタルGC、クロージャ対応

## 演習13-9: リアルタイムメモリアロケーター

**ファイル**: ex13\_9\_realtime.c, ex13\_9\_realtime\_c99.c

決定的時間保証を持つアロケーターの実装です。 - O(1)時間保証 - デッドライン管理 - WCET（最悪実行時間）追跡 - C90版：基本的なリアルタイム保証 - C99版：キャッシュカラリング、高度な統計

## ビルド方法

# 特定の解答をビルド  
make ex13\_1\_macro\_basics  
  
# C99版をビルド  
make ex13\_1\_macro\_basics\_c99  
  
# すべての解答をビルド  
make all  
  
# 実行  
./ex13\_1\_macro\_basics

## 学習のポイント

1. **プリプロセッサの限界と可能性**
   * マクロの適切な使用場面
   * 型安全性の確保方法
   * デバッグの考慮
2. **メモリ管理の最適化**
   * カスタムアロケーターの設計
   * キャッシュ効率の考慮
   * リアルタイム性の保証
3. **C90とC99の違い**
   * 言語機能の進化
   * より安全で表現力豊かなコード
   * パフォーマンスの改善
4. **実践的な応用**
   * 組み込みシステム
   * ゲームエンジン
   * リアルタイムシステム ### ex13\_1\_macro\_basics.c

/\*  
 \* 演習13-1の解答例: プリプロセッサマクロの基本  
 \* ファイル名: ex13\_1\_macro\_basics.c  
 \* 説明: 数学定数、計算マクロ、デバッグ機能の実装  
 \* C90準拠 - typeof等のGNU拡張は使用しない  
 \*/  
  
#include <stdio.h>  
#include <math.h>  
#include <time.h>  
  
/\* デバッグモードの設定 \*/  
#define DEBUG\_MODE 1  
  
/\* 数学的定数の定義 \*/  
#define PI 3.14159265358979323846  
#define E 2.71828182845904523536  
#define GOLDEN\_RATIO 1.61803398874989484820  
  
/\* 文字列化マクロ \*/  
#define STRINGIFY(x) #x  
#define TOSTRING(x) STRINGIFY(x)  
  
/\* 連結マクロ \*/  
#define CONCAT(a, b) a##b  
#define CONCAT3(a, b, c) a##b##c  
  
/\* 面積計算マクロ \*/  
#define CIRCLE\_AREA(radius) (PI \* (radius) \* (radius))  
#define CIRCLE\_CIRCUMFERENCE(radius) (2.0 \* PI \* (radius))  
#define RECTANGLE\_AREA(width, height) ((width) \* (height))  
#define TRIANGLE\_AREA(base, height) (0.5 \* (base) \* (height))  
  
/\* 体積計算マクロ \*/  
#define SPHERE\_VOLUME(radius) ((4.0 / 3.0) \* PI \* (radius) \* (radius) \* (radius))  
#define SPHERE\_SURFACE\_AREA(radius) (4.0 \* PI \* (radius) \* (radius))  
#define CYLINDER\_VOLUME(radius, height) (PI \* (radius) \* (radius) \* (height))  
#define CUBE\_VOLUME(side) ((side) \* (side) \* (side))  
  
/\* C90準拠の最大値・最小値マクロ \*/  
#define MAX(a, b) ((a) > (b) ? (a) : (b))  
#define MIN(a, b) ((a) < (b) ? (a) : (b))  
  
/\* 絶対値マクロ \*/  
#define ABS(x) ((x) < 0 ? -(x) : (x))  
  
/\* デバッグ出力マクロ（C90準拠 - 可変引数マクロなし） \*/  
#if DEBUG\_MODE  
#define DEBUG\_PRINT(msg) printf("[DEBUG:%s:%d] %s\n", \_\_FILE\_\_, \_\_LINE\_\_, msg)  
#define DEBUG\_VAR\_INT(var) printf("[DEBUG:%s:%d] %s = %d\n", \_\_FILE\_\_, \_\_LINE\_\_, #var, var)  
#define DEBUG\_VAR\_DOUBLE(var) printf("[DEBUG:%s:%d] %s = %g\n", \_\_FILE\_\_, \_\_LINE\_\_, #var, var)  
#else  
#define DEBUG\_PRINT(msg)  
#define DEBUG\_VAR\_INT(var)  
#define DEBUG\_VAR\_DOUBLE(var)  
#endif  
  
/\* アサーションマクロ \*/  
#define ASSERT\_POSITIVE(x) \  
 do { \  
 if ((x) <= 0) { \  
 printf("エラー: %s は正の値である必要があります (値: %g)\n", #x, (double)(x)); \  
 return -1; \  
 } \  
 } while (0)  
  
/\* 単位変換マクロ \*/  
#define DEG\_TO\_RAD(deg) ((deg) \* PI / 180.0)  
#define RAD\_TO\_DEG(rad) ((rad) \* 180.0 / PI)  
#define CELSIUS\_TO\_FAHRENHEIT(c) ((c) \* 9.0 / 5.0 + 32.0)  
#define FAHRENHEIT\_TO\_CELSIUS(f) (((f) - 32.0) \* 5.0 / 9.0)  
  
/\* 範囲チェックマクロ \*/  
#define IN\_RANGE(x, min, max) ((x) >= (min) && (x) <= (max))  
  
/\* 配列サイズ取得マクロ \*/  
#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))  
  
/\* ベンチマーク用変数（グローバル） \*/  
clock\_t bench\_start\_time;  
  
/\* ベンチマーク開始マクロ \*/  
#define BENCHMARK\_START(name) \  
 do { \  
 bench\_start\_time = clock(); \  
 printf("ベンチマーク開始: %s\n", #name); \  
 } while (0)  
  
/\* ベンチマーク終了マクロ \*/  
#define BENCHMARK\_END(name) \  
 do { \  
 clock\_t bench\_end\_time = clock(); \  
 double elapsed = ((double)(bench\_end\_time - bench\_start\_time)) / CLOCKS\_PER\_SEC; \  
 printf("ベンチマーク終了: %s (実行時間: %.6f秒)\n", #name, elapsed); \  
 } while (0)  
  
/\* 条件付きコンパイル \*/  
#ifdef ENABLE\_EXTENDED\_MATH  
#define EXTENDED\_PI\_DIGITS 50  
const char extended\_pi[] = "3.14159265358979323846264338327950288419716939937510";  
#endif  
  
/\* 計算結果の表示関数 \*/  
void print\_calculation(const char\* desc, const char\* formula, double value)  
{  
 printf("%-25s: %s = %.6f\n", desc, formula, value);  
}  
  
/\* テスト関数群 \*/  
void test\_basic\_calculations(void)  
{  
 double radius = 5.0;  
 double width = 10.0, height = 8.0;  
 double base = 6.0, tri\_height = 4.0;  
   
 printf("=== 基本計算テスト ===\n");  
   
 DEBUG\_VAR\_DOUBLE(radius);  
 DEBUG\_VAR\_DOUBLE(width);  
 DEBUG\_VAR\_DOUBLE(height);  
   
 print\_calculation("円の面積", "PI \* 5 \* 5", CIRCLE\_AREA(radius));  
 print\_calculation("円の周長", "2 \* PI \* 5", CIRCLE\_CIRCUMFERENCE(radius));  
 print\_calculation("長方形の面積", "10 \* 8", RECTANGLE\_AREA(width, height));  
 print\_calculation("三角形の面積", "0.5 \* 6 \* 4", TRIANGLE\_AREA(base, tri\_height));  
   
 printf("\n");  
}  
  
void test\_volume\_calculations(void)  
{  
 double radius = 3.0;  
 double height = 10.0;  
 double side = 4.0;  
   
 printf("=== 体積計算テスト ===\n");  
   
 print\_calculation("球の体積", "(4/3) \* PI \* 3^3", SPHERE\_VOLUME(radius));  
 print\_calculation("球の表面積", "4 \* PI \* 3^2", SPHERE\_SURFACE\_AREA(radius));  
 print\_calculation("円柱の体積", "PI \* 3^2 \* 10", CYLINDER\_VOLUME(radius, height));  
 print\_calculation("立方体の体積", "4^3", CUBE\_VOLUME(side));  
   
 printf("\n");  
}  
  
void test\_max\_min\_functions(void)  
{  
 int a = 10, b = 20;  
 double x = 3.14, y = 2.71;  
 int counter = 0;  
 int temp\_counter;  
 int max\_result;  
   
 printf("=== 最大値・最小値テスト ===\n");  
   
 printf("MAX(%d, %d) = %d\n", a, b, MAX(a, b));  
 printf("MIN(%d, %d) = %d\n", a, b, MIN(a, b));  
 printf("MAX(%.2f, %.2f) = %.2f\n", x, y, MAX(x, y));  
 printf("MIN(%.2f, %.2f) = %.2f\n", x, y, MIN(x, y));  
   
 /\* 副作用のテスト（C90準拠版） \*/  
 printf("副作用テスト: counter = %d\n", counter);  
 temp\_counter = counter + 1;  
 max\_result = MAX(temp\_counter, 5);  
 counter = temp\_counter;  
 printf("MAX(++counter, 5) = %d, counter = %d （注意：副作用の例）\n", max\_result, counter);  
   
 printf("\n");  
}  
  
void test\_unit\_conversions(void)  
{  
 double degrees = 90.0;  
 double radians = PI / 4.0;  
 double celsius = 25.0;  
 double fahrenheit = 77.0;  
   
 printf("=== 単位変換テスト ===\n");  
   
 print\_calculation("90度をラジアンに", "90 \* PI / 180", DEG\_TO\_RAD(degrees));  
 print\_calculation("π/4ラジアンを度に", "(PI/4) \* 180 / PI", RAD\_TO\_DEG(radians));  
 print\_calculation("25℃を華氏に", "25 \* 9/5 + 32", CELSIUS\_TO\_FAHRENHEIT(celsius));  
 print\_calculation("77℉を摂氏に", "(77 - 32) \* 5/9", FAHRENHEIT\_TO\_CELSIUS(fahrenheit));  
   
 printf("\n");  
}  
  
void test\_string\_macros(void)  
{  
 printf("=== 文字列マクロテスト ===\n");  
   
 printf("PI定数の文字列化: %s\n", TOSTRING(PI));  
 printf("E定数の文字列化: %s\n", TOSTRING(E));  
 printf("黄金比の文字列化: %s\n", TOSTRING(GOLDEN\_RATIO));  
   
 /\* 連結マクロのテスト \*/  
 printf("連結マクロのテスト: %s\n", TOSTRING(CONCAT(test\_, function)));  
   
 printf("\n");  
}  
  
void test\_range\_and\_array(void)  
{  
 int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
 int value = 5;  
   
 printf("=== 範囲チェックと配列テスト ===\n");  
   
 printf("配列サイズ: %lu\n", (unsigned long)ARRAY\_SIZE(numbers));  
 printf("値 %d は範囲 [1, 10] に含まれる: %s\n",  
 value, IN\_RANGE(value, 1, 10) ? "はい" : "いいえ");  
 printf("値 %d は範囲 [6, 10] に含まれる: %s\n",  
 value, IN\_RANGE(value, 6, 10) ? "はい" : "いいえ");  
   
 printf("\n");  
}  
  
void test\_benchmark(void)  
{  
 int i;  
 double sum = 0.0;  
   
 printf("=== ベンチマークテスト ===\n");  
   
 BENCHMARK\_START(calculation\_test);  
   
 /\* 計算集約的な処理をシミュレート \*/  
 for (i = 0; i < 1000000; i++) {  
 sum += CIRCLE\_AREA(i \* 0.001);  
 }  
   
 BENCHMARK\_END(calculation\_test);  
   
 printf("計算結果（無視される値）: %.6f\n", sum);  
 printf("\n");  
}  
  
/\* コンパイル時情報の表示 \*/  
void show\_compile\_info(void)  
{  
 printf("=== コンパイル時情報 ===\n");  
 printf("ファイル: %s\n", \_\_FILE\_\_);  
 printf("コンパイル日時: %s %s\n", \_\_DATE\_\_, \_\_TIME\_\_);  
 printf("C標準: ");  
   
#if defined(\_\_STDC\_VERSION\_\_)  
 #if \_\_STDC\_VERSION\_\_ >= 201112L  
 printf("C11\n");  
 #elif \_\_STDC\_VERSION\_\_ >= 199901L  
 printf("C99\n");  
 #elif \_\_STDC\_VERSION\_\_ >= 199409L  
 printf("C95\n");  
 #else  
 printf("C90\n");  
 #endif  
#else  
 printf("C90\n");  
#endif  
   
 printf("デバッグモード: %s\n", DEBUG\_MODE ? "有効" : "無効");  
   
#ifdef ENABLE\_EXTENDED\_MATH  
 printf("拡張数学モード: 有効\n");  
 printf("拡張π値: %s\n", extended\_pi);  
#else  
 printf("拡張数学モード: 無効\n");  
#endif  
   
 printf("\n");  
}  
  
/\* エラーハンドリングのテスト \*/  
int test\_error\_handling(void)  
{  
 double negative\_radius = -5.0;  
   
 printf("=== エラーハンドリングテスト ===\n");  
   
 printf("負の半径でのテスト: %.1f\n", negative\_radius);  
 ASSERT\_POSITIVE(negative\_radius);  
   
 /\* この行は実行されない \*/  
 printf("この行は表示されません\n");  
   
 return 0;  
}  
  
int main(void)  
{  
 printf("=== プリプロセッサマクロ基本デモ ===\n\n");  
   
 /\* コンパイル時情報の表示 \*/  
 show\_compile\_info();  
   
 /\* 各テスト関数の実行 \*/  
 test\_basic\_calculations();  
 test\_volume\_calculations();  
 test\_max\_min\_functions();  
 test\_unit\_conversions();  
 test\_string\_macros();  
 test\_range\_and\_array();  
 test\_benchmark();  
   
 /\* エラーハンドリングのテスト（プログラム終了） \*/  
 printf("エラーハンドリングのテストを実行します...\n");  
 test\_error\_handling();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== プリプロセッサマクロ基本デモ ===  
  
=== コンパイル時情報 ===  
ファイル: ex13\_1\_macro\_basics.c  
コンパイル日時: Dec 25 2024 12:00:00  
C標準: C90  
デバッグモード: 有効  
拡張数学モード: 無効  
  
=== 基本計算テスト ===  
[DEBUG:ex13\_1\_macro\_basics.c:108] radius = 5  
[DEBUG:ex13\_1\_macro\_basics.c:109] width = 10  
[DEBUG:ex13\_1\_macro\_basics.c:110] height = 8  
円の面積 : PI \* 5 \* 5 = 78.539816  
円の周長 : 2 \* PI \* 5 = 31.415927  
長方形の面積 : 10 \* 8 = 80.000000  
三角形の面積 : 0.5 \* 6 \* 4 = 12.000000  
  
=== 体積計算テスト ===  
球の体積 : (4/3) \* PI \* 3^3 = 113.097335  
球の表面積 : 4 \* PI \* 3^2 = 113.097335  
円柱の体積 : PI \* 3^2 \* 10 = 282.743339  
立方体の体積 : 4^3 = 64.000000  
  
=== 最大値・最小値テスト ===  
MAX(10, 20) = 20  
MIN(10, 20) = 10  
MAX(3.14, 2.71) = 3.14  
MIN(3.14, 2.71) = 2.71  
副作用テスト: counter = 0  
MAX(++counter, 5) = 5, counter = 1 （注意：副作用の例）  
  
=== 単位変換テスト ===  
90度をラジアンに : 90 \* PI / 180 = 1.570796  
π/4ラジアンを度に : (PI/4) \* 180 / PI = 45.000000  
25℃を華氏に : 25 \* 9/5 + 32 = 77.000000  
77℉を摂氏に : (77 - 32) \* 5/9 = 25.000000  
  
=== 文字列マクロテスト ===  
PI定数の文字列化: PI  
E定数の文字列化: E  
黄金比の文字列化: GOLDEN\_RATIO  
連結マクロのテスト: test\_function  
  
=== 範囲チェックと配列テスト ===  
配列サイズ: 10  
値 5 は範囲 [1, 10] に含まれる: はい  
値 5 は範囲 [6, 10] に含まれる: いいえ  
  
=== ベンチマークテスト ===  
ベンチマーク開始: calculation\_test  
ベンチマーク終了: calculation\_test (実行時間: 0.123456秒)  
計算結果（無視される値）: 523598.775600  
  
エラーハンドリングのテストを実行します...  
=== エラーハンドリングテスト ===  
負の半径でのテスト: -5.0  
エラー: negative\_radius は正の値である必要があります (値: -5)  
\*/```  
  
### ex13\_2\_safe\_memory.c  
  
```c  
/\*  
 \* 演習13-2の解答例: 安全なメモリ操作マクロ  
 \* ファイル名: ex13\_2\_safe\_memory.c  
 \* 説明: NULLチェック、境界チェック、メモリリーク検出機能の実装  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
/\* メモリリーク追跡用の構造体 \*/  
typedef struct MemoryTracker {  
 void \*ptr;  
 size\_t size;  
 const char \*file;  
 int line;  
 struct MemoryTracker \*next;  
} MemoryTracker;  
  
/\* グローバルなメモリ追跡リスト \*/  
static MemoryTracker \*g\_memory\_list = NULL;  
static size\_t g\_total\_allocated = 0;  
static size\_t g\_total\_freed = 0;  
static int g\_allocation\_count = 0;  
static int g\_free\_count = 0;  
  
/\* メモリ追跡の有効/無効 \*/  
#define MEMORY\_TRACKING\_ENABLED 1  
  
/\* エラー処理用マクロ（C90準拠 - 可変引数マクロなし） \*/  
#define MEMORY\_ERROR(msg) \  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] %s\n", \_\_FILE\_\_, \_\_LINE\_\_, msg)  
  
#define MEMORY\_WARNING(msg) \  
 fprintf(stderr, "[MEMORY WARNING:%s:%d] %s\n", \_\_FILE\_\_, \_\_LINE\_\_, msg)  
  
#define MEMORY\_INFO(msg) \  
 printf("[MEMORY INFO:%s:%d] %s\n", \_\_FILE\_\_, \_\_LINE\_\_, msg)  
  
/\* 安全なメモリ割り当てマクロ \*/  
#define SAFE\_MALLOC(size) safe\_malloc\_impl((size), \_\_FILE\_\_, \_\_LINE\_\_)  
#define SAFE\_CALLOC(count, size) safe\_calloc\_impl((count), (size), \_\_FILE\_\_, \_\_LINE\_\_)  
#define SAFE\_REALLOC(ptr, size) safe\_realloc\_impl((ptr), (size), \_\_FILE\_\_, \_\_LINE\_\_)  
  
/\* 安全なメモリ解放マクロ \*/  
#define SAFE\_FREE(ptr) \  
 do { \  
 if (ptr) { \  
 safe\_free\_impl((ptr), \_\_FILE\_\_, \_\_LINE\_\_); \  
 (ptr) = NULL; \  
 } \  
 } while (0)  
  
/\* 配列境界チェック付きアクセスマクロ \*/  
#define ARRAY\_BOUNDS\_CHECK(arr, index, size) \  
 ((index) >= 0 && (size\_t)(index) < (size\_t)(size))  
  
#define SAFE\_ARRAY\_ACCESS(arr, index, size, default\_val) \  
 (ARRAY\_BOUNDS\_CHECK(arr, index, size) ? (arr)[index] : (default\_val))  
  
#define SAFE\_ARRAY\_SET(arr, index, size, value) \  
 do { \  
 if (ARRAY\_BOUNDS\_CHECK(arr, index, size)) { \  
 (arr)[index] = (value); \  
 } else { \  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] 配列境界エラー: インデックス %d が範囲 [0, %lu) を超えています\n", \  
 \_\_FILE\_\_, \_\_LINE\_\_, (int)(index), (unsigned long)(size)); \  
 } \  
 } while (0)  
  
/\* メモリ初期化マクロ \*/  
#define SAFE\_MEMSET(ptr, value, size) \  
 do { \  
 if (ptr) { \  
 memset((ptr), (value), (size)); \  
 } else { \  
 MEMORY\_ERROR("NULLポインタに対するmemset"); \  
 } \  
 } while (0)  
  
#define SAFE\_MEMCPY(dest, src, size) \  
 do { \  
 if ((dest) && (src)) { \  
 memcpy((dest), (src), (size)); \  
 } else { \  
 MEMORY\_ERROR("NULLポインタに対するmemcpy"); \  
 } \  
 } while (0)  
  
/\* 文字列操作の安全なマクロ \*/  
#define SAFE\_STRNCPY(dest, src, size) \  
 do { \  
 if ((dest) && (src) && (size) > 0) { \  
 strncpy((dest), (src), (size) - 1); \  
 (dest)[(size) - 1] = '\0'; \  
 } else { \  
 MEMORY\_ERROR("不正なstrncpyパラメータ"); \  
 } \  
 } while (0)  
  
#define SAFE\_STRNCAT(dest, src, size) \  
 do { \  
 if ((dest) && (src) && (size) > 0) { \  
 size\_t dest\_len = strlen(dest); \  
 if (dest\_len < (size) - 1) { \  
 strncat((dest), (src), (size) - dest\_len - 1); \  
 } \  
 } else { \  
 MEMORY\_ERROR("不正なstrncatパラメータ"); \  
 } \  
 } while (0)  
  
/\* ポインタ検証マクロ \*/  
#define VALIDATE\_POINTER(ptr, action) \  
 do { \  
 if (!(ptr)) { \  
 MEMORY\_ERROR("NULLポインタが検出されました"); \  
 action; \  
 } \  
 } while (0)  
  
#define VALIDATE\_POINTER\_RETURN(ptr, retval) \  
 VALIDATE\_POINTER(ptr, return (retval))  
  
#define VALIDATE\_POINTER\_RETURN\_VOID(ptr) \  
 VALIDATE\_POINTER(ptr, return)  
  
/\* メモリリーク検出マクロ \*/  
#define MEMORY\_LEAK\_TRACKER\_INIT() init\_memory\_tracker()  
#define MEMORY\_LEAK\_TRACKER\_REPORT() report\_memory\_leaks()  
#define MEMORY\_LEAK\_TRACKER\_CLEANUP() cleanup\_memory\_tracker()  
  
/\* 実装関数群 \*/  
  
/\* メモリ追跡の初期化 \*/  
void init\_memory\_tracker(void)  
{  
 g\_memory\_list = NULL;  
 g\_total\_allocated = 0;  
 g\_total\_freed = 0;  
 g\_allocation\_count = 0;  
 g\_free\_count = 0;  
 MEMORY\_INFO("メモリ追跡システムを初期化しました");  
}  
  
/\* メモリ追跡エントリの追加 \*/  
static void add\_memory\_tracker(void \*ptr, size\_t size, const char \*file, int line)  
{  
#if MEMORY\_TRACKING\_ENABLED  
 MemoryTracker \*tracker = (MemoryTracker \*)malloc(sizeof(MemoryTracker));  
 if (tracker) {  
 tracker->ptr = ptr;  
 tracker->size = size;  
 tracker->file = file;  
 tracker->line = line;  
 tracker->next = g\_memory\_list;  
 g\_memory\_list = tracker;  
   
 g\_total\_allocated += size;  
 g\_allocation\_count++;  
 }  
#endif  
}  
  
/\* メモリ追跡エントリの削除 \*/  
static void remove\_memory\_tracker(void \*ptr, const char \*file, int line)  
{  
#if MEMORY\_TRACKING\_ENABLED  
 MemoryTracker \*\*current = &g\_memory\_list;  
 while (\*current) {  
 if ((\*current)->ptr == ptr) {  
 MemoryTracker \*to\_remove = \*current;  
 g\_total\_freed += to\_remove->size;  
 g\_free\_count++;  
 \*current = to\_remove->next;  
 free(to\_remove);  
 return;  
 }  
 current = &(\*current)->next;  
 }  
 fprintf(stderr, "[MEMORY WARNING:%s:%d] 解放されたポインタ %p が追跡リストに見つかりません\n",  
 file, line, ptr);  
#endif  
}  
  
/\* 安全なmalloc実装 \*/  
void \*safe\_malloc\_impl(size\_t size, const char \*file, int line)  
{  
 void \*ptr;  
   
 if (size == 0) {  
 fprintf(stderr, "[MEMORY WARNING:%s:%d] サイズ0でのmalloc\n", file, line);  
 return NULL;  
 }  
   
 ptr = malloc(size);  
 if (!ptr) {  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] malloc失敗: %luバイト\n",   
 file, line, (unsigned long)size);  
 return NULL;  
 }  
   
 add\_memory\_tracker(ptr, size, file, line);  
 printf("[MEMORY INFO:%s:%d] malloc成功: %luバイト @ %p\n",   
 file, line, (unsigned long)size, ptr);  
   
 return ptr;  
}  
  
/\* 安全なcalloc実装 \*/  
void \*safe\_calloc\_impl(size\_t count, size\_t size, const char \*file, int line)  
{  
 void \*ptr;  
 size\_t total\_size;  
   
 if (count == 0 || size == 0) {  
 fprintf(stderr, "[MEMORY WARNING:%s:%d] サイズ0でのcalloc\n", file, line);  
 return NULL;  
 }  
   
 /\* オーバーフローチェック \*/  
 total\_size = count \* size;  
 if (count != 0 && total\_size / count != size) {  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] calloc整数オーバーフロー: %lu \* %lu\n",  
 file, line, (unsigned long)count, (unsigned long)size);  
 return NULL;  
 }  
   
 ptr = calloc(count, size);  
 if (!ptr) {  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] calloc失敗: %lu \* %luバイト\n",  
 file, line, (unsigned long)count, (unsigned long)size);  
 return NULL;  
 }  
   
 add\_memory\_tracker(ptr, total\_size, file, line);  
 printf("[MEMORY INFO:%s:%d] calloc成功: %lu \* %luバイト @ %p\n",  
 file, line, (unsigned long)count, (unsigned long)size, ptr);  
   
 return ptr;  
}  
  
/\* 安全なrealloc実装 \*/  
void \*safe\_realloc\_impl(void \*ptr, size\_t size, const char \*file, int line)  
{  
 void \*new\_ptr;  
   
 if (!ptr) {  
 return safe\_malloc\_impl(size, file, line);  
 }  
   
 if (size == 0) {  
 safe\_free\_impl(ptr, file, line);  
 return NULL;  
 }  
   
 new\_ptr = realloc(ptr, size);  
 if (!new\_ptr) {  
 fprintf(stderr, "[MEMORY ERROR:%s:%d] realloc失敗: %luバイト\n",   
 file, line, (unsigned long)size);  
 return NULL;  
 }  
   
 /\* 追跡情報の更新 \*/  
 remove\_memory\_tracker(ptr, file, line);  
 add\_memory\_tracker(new\_ptr, size, file, line);  
   
 printf("[MEMORY INFO:%s:%d] realloc成功: %p -> %p (%luバイト)\n",  
 file, line, ptr, new\_ptr, (unsigned long)size);  
   
 return new\_ptr;  
}  
  
/\* 安全なfree実装 \*/  
void safe\_free\_impl(void \*ptr, const char \*file, int line)  
{  
 if (!ptr) {  
 fprintf(stderr, "[MEMORY WARNING:%s:%d] NULLポインタのfree\n", file, line);  
 return;  
 }  
   
 remove\_memory\_tracker(ptr, file, line);  
 free(ptr);  
 /\* 注意: デバッグ目的でポインタ値を表示。実際のコードでは解放後のポインタアクセスは避ける \*/  
 printf("[MEMORY INFO:%s:%d] free成功: アドレス %p を解放しました\n", file, line, (void\*)ptr);  
}  
  
/\* メモリリークレポート \*/  
void report\_memory\_leaks(void)  
{  
 MemoryTracker \*current;  
 size\_t leak\_total = 0;  
 int leak\_count = 0;  
   
 printf("\n=== メモリリークレポート ===\n");  
 printf("総割り当て: %lu バイト (%d 回)\n",  
 (unsigned long)g\_total\_allocated, g\_allocation\_count);  
 printf("総解放: %lu バイト (%d 回)\n",  
 (unsigned long)g\_total\_freed, g\_free\_count);  
   
 if (g\_memory\_list) {  
 printf("\n検出されたリーク:\n");  
 current = g\_memory\_list;  
   
 while (current) {  
 printf(" %p: %lu バイト (%s:%d)\n",  
 current->ptr, (unsigned long)current->size,  
 current->file, current->line);  
 leak\_total += current->size;  
 leak\_count++;  
 current = current->next;  
 }  
   
 printf("\nリーク合計: %lu バイト (%d ブロック)\n",  
 (unsigned long)leak\_total, leak\_count);  
 } else {  
 printf("メモリリークは検出されませんでした。\n");  
 }  
 printf("========================\n\n");  
}  
  
/\* メモリ追跡システムのクリーンアップ \*/  
void cleanup\_memory\_tracker(void)  
{  
 while (g\_memory\_list) {  
 MemoryTracker \*to\_remove = g\_memory\_list;  
 g\_memory\_list = g\_memory\_list->next;  
 free(to\_remove);  
 }  
 MEMORY\_INFO("メモリ追跡システムをクリーンアップしました");  
}  
  
/\* テスト関数群 \*/  
  
void test\_safe\_allocation(void)  
{  
 int \*numbers;  
 int i;  
   
 printf("=== 安全なメモリ割り当てテスト ===\n");  
   
 /\* 通常の割り当て \*/  
 numbers = (int \*)SAFE\_MALLOC(sizeof(int) \* 10);  
 VALIDATE\_POINTER\_RETURN\_VOID(numbers);  
   
 for (i = 0; i < 10; i++) {  
 numbers[i] = i \* i;  
 }  
   
 printf("割り当てた配列: ");  
 for (i = 0; i < 10; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 /\* 配列のサイズ変更 \*/  
 numbers = (int \*)SAFE\_REALLOC(numbers, sizeof(int) \* 20);  
 VALIDATE\_POINTER\_RETURN\_VOID(numbers);  
   
 for (i = 10; i < 20; i++) {  
 numbers[i] = i \* i;  
 }  
   
 printf("拡張後の配列: ");  
 for (i = 0; i < 20; i++) {  
 printf("%d ", numbers[i]);  
 }  
 printf("\n");  
   
 SAFE\_FREE(numbers);  
 printf("メモリを正常に解放しました\n\n");  
}  
  
void test\_array\_bounds\_checking(void)  
{  
 int \*array;  
 int i;  
 int safe\_value;  
   
 printf("=== 配列境界チェックテスト ===\n");  
   
 array = (int \*)SAFE\_MALLOC(sizeof(int) \* 5);  
 VALIDATE\_POINTER\_RETURN\_VOID(array);  
   
 /\* 配列の初期化 \*/  
 for (i = 0; i < 5; i++) {  
 array[i] = (i + 1) \* 10;  
 }  
   
 printf("配列内容: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n");  
   
 /\* 安全なアクセステスト \*/  
 printf("安全なアクセステスト:\n");  
   
 for (i = -1; i <= 6; i++) {  
 if (ARRAY\_BOUNDS\_CHECK(array, i, 5)) {  
 printf(" インデックス %d: %d (有効)\n", i, array[i]);  
 } else {  
 printf(" インデックス %d: 境界外アクセス\n", i);  
 safe\_value = SAFE\_ARRAY\_ACCESS(array, i, 5, -1);  
 printf(" -> デフォルト値: %d\n", safe\_value);  
 }  
 }  
   
 /\* 安全な設定テスト \*/  
 printf("\n安全な設定テスト:\n");  
 SAFE\_ARRAY\_SET(array, 2, 5, 999); /\* 有効 \*/  
 SAFE\_ARRAY\_SET(array, 10, 5, 888); /\* 無効 \*/  
 SAFE\_ARRAY\_SET(array, -1, 5, 777); /\* 無効 \*/  
   
 printf("設定後の配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n");  
   
 SAFE\_FREE(array);  
 printf("\n");  
}  
  
void test\_string\_operations(void)  
{  
 char \*buffer;  
 char long\_string[100];  
 char short\_buffer[20];  
   
 printf("=== 安全な文字列操作テスト ===\n");  
   
 buffer = (char \*)SAFE\_MALLOC(64);  
 VALIDATE\_POINTER\_RETURN\_VOID(buffer);  
   
 /\* 安全な文字列コピー \*/  
 SAFE\_STRNCPY(buffer, "Hello, World!", 64);  
 printf("初期文字列: %s\n", buffer);  
   
 /\* 安全な文字列連結 \*/  
 SAFE\_STRNCAT(buffer, " 追加テキスト", 64);  
 printf("連結後: %s\n", buffer);  
   
 /\* 長すぎる文字列のテスト \*/  
 memset(long\_string, 'A', 99);  
 long\_string[99] = '\0';  
   
 SAFE\_STRNCPY(short\_buffer, long\_string, sizeof(short\_buffer));  
 printf("切り詰められた文字列: %s\n", short\_buffer);  
   
 SAFE\_FREE(buffer);  
 printf("\n");  
}  
  
void test\_memory\_leak\_detection(void)  
{  
 void \*leak1;  
 void \*leak2;  
 void \*no\_leak;  
   
 printf("=== メモリリーク検出テスト ===\n");  
   
 /\* 意図的にリークを作成 \*/  
 leak1 = SAFE\_MALLOC(100);  
 leak2 = SAFE\_MALLOC(200);  
 no\_leak = SAFE\_MALLOC(50);  
   
 /\* 一つだけ解放 \*/  
 SAFE\_FREE(no\_leak);  
   
 printf("意図的にリークを作成しました\n");  
 printf("leak1 = %p (100バイト)\n", leak1);  
 printf("leak2 = %p (200バイト)\n", leak2);  
 printf("no\_leakは解放済み\n\n");  
   
 /\* leak1とleak2は意図的に解放しない \*/  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== 安全なメモリ操作マクロデモ ===\n\n");  
   
 /\* メモリ追跡システムの初期化 \*/  
 MEMORY\_LEAK\_TRACKER\_INIT();  
   
 /\* 各種テストの実行 \*/  
 test\_safe\_allocation();  
 test\_array\_bounds\_checking();  
 test\_string\_operations();  
 test\_memory\_leak\_detection();  
   
 /\* メモリリークレポートの表示 \*/  
 MEMORY\_LEAK\_TRACKER\_REPORT();  
   
 /\* メモリ追跡システムのクリーンアップ \*/  
 MEMORY\_LEAK\_TRACKER\_CLEANUP();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== 安全なメモリ操作マクロデモ ===  
  
[MEMORY INFO:ex13\_2\_safe\_memory.c:135] メモリ追跡システムを初期化しました  
  
=== 安全なメモリ割り当てテスト ===  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 40バイト @ 0x1234560  
割り当てた配列: 0 1 4 9 16 25 36 49 64 81   
[MEMORY INFO:ex13\_2\_safe\_memory.c:279] realloc成功: 0x1234560 -> 0x1234580 (80バイト)  
拡張後の配列: 0 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361   
[MEMORY INFO:ex13\_2\_safe\_memory.c:295] free成功: 0x1234580  
メモリを正常に解放しました  
  
=== 配列境界チェックテスト ===  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 20バイト @ 0x12345a0  
配列内容: 10 20 30 40 50   
安全なアクセステスト:  
 インデックス -1: 境界外アクセス  
 -> デフォルト値: -1  
 インデックス 0: 10 (有効)  
 インデックス 1: 20 (有効)  
 インデックス 2: 30 (有効)  
 インデックス 3: 40 (有効)  
 インデックス 4: 50 (有効)  
 インデックス 5: 境界外アクセス  
 -> デフォルト値: -1  
 インデックス 6: 境界外アクセス  
 -> デフォルト値: -1  
  
安全な設定テスト:  
[MEMORY ERROR:ex13\_2\_safe\_memory.c:390] 配列境界エラー: インデックス 10 が範囲 [0, 5) を超えています  
[MEMORY ERROR:ex13\_2\_safe\_memory.c:391] 配列境界エラー: インデックス -1 が範囲 [0, 5) を超えています  
設定後の配列: 10 20 999 40 50   
[MEMORY INFO:ex13\_2\_safe\_memory.c:295] free成功: 0x12345a0  
  
=== 安全な文字列操作テスト ===  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 64バイト @ 0x12345c0  
初期文字列: Hello, World!  
連結後: Hello, World! 追加テキスト  
切り詰められた文字列: AAAAAAAAAAAAAAAAAAA  
[MEMORY INFO:ex13\_2\_safe\_memory.c:295] free成功: 0x12345c0  
  
=== メモリリーク検出テスト ===  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 100バイト @ 0x1234600  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 200バイト @ 0x1234680  
[MEMORY INFO:ex13\_2\_safe\_memory.c:207] malloc成功: 50バイト @ 0x1234780  
[MEMORY INFO:ex13\_2\_safe\_memory.c:295] free成功: 0x1234780  
意図的にリークを作成しました  
leak1 = 0x1234600 (100バイト)  
leak2 = 0x1234680 (200バイト)  
no\_leakは解放済み  
  
=== メモリリークレポート ===  
総割り当て: 474 バイト (5 回)  
総解放: 174 バイト (3 回)  
  
検出されたリーク:  
 0x1234680: 200 バイト (ex13\_2\_safe\_memory.c:448)  
 0x1234600: 100 バイト (ex13\_2\_safe\_memory.c:447)  
  
リーク合計: 300 バイト (2 ブロック)  
========================  
  
[MEMORY INFO:ex13\_2\_safe\_memory.c:319] メモリ追跡システムをクリーンアップしました  
=== デモ完了 ===  
\*/```  
  
### ex13\_3\_memory\_pool.c  
  
```c  
/\*  
 \* 演習13-3の解答例: 基本的なメモリプール  
 \* ファイル名: ex13\_3\_memory\_pool.c  
 \* 説明: 固定サイズオブジェクト用のメモリプール実装  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <assert.h>  
#include <time.h>  
  
/\* デバッグ出力制御 \*/  
#define DEBUG\_POOL 1  
  
#if DEBUG\_POOL  
#define POOL\_DEBUG(msg) printf("[POOL DEBUG] %s\n", msg)  
#else  
#define POOL\_DEBUG(msg)  
#endif  
  
/\* メモリプールの設定 \*/  
#define DEFAULT\_POOL\_SIZE 16  
#define POOL\_ALIGNMENT 8  
#define POOL\_MAGIC\_NUMBER 0xDEADBEEF  
  
/\* アライメント調整マクロ \*/  
#define ALIGN\_SIZE(size, alignment) \  
 (((size) + (alignment) - 1) & ~((alignment) - 1))  
  
/\* フリーリストノード構造体 \*/  
typedef struct FreeNode {  
 struct FreeNode \*next;  
} FreeNode;  
  
/\* メモリプール構造体 \*/  
typedef struct MemoryPool {  
 void \*memory\_chunk; /\* プール全体のメモリ領域 \*/  
 FreeNode \*free\_list; /\* フリーリストの先頭 \*/  
 size\_t object\_size; /\* オブジェクトサイズ \*/  
 size\_t pool\_capacity; /\* プールの容量 \*/  
 size\_t objects\_in\_use; /\* 使用中オブジェクト数 \*/  
 size\_t total\_allocated; /\* 総割り当て数 \*/  
 size\_t total\_freed; /\* 総解放数 \*/  
 unsigned int magic; /\* 破損検出用マジックナンバー \*/  
 char name[32]; /\* プール名 \*/  
} MemoryPool;  
  
/\* プール検証マクロ \*/  
#define VALIDATE\_POOL(pool) \  
 do { \  
 if (!(pool) || (pool)->magic != POOL\_MAGIC\_NUMBER) { \  
 fprintf(stderr, "無効なメモリプール: %p\n", (void \*)(pool)); \  
 return NULL; \  
 } \  
 } while (0)  
  
#define VALIDATE\_POOL\_VOID(pool) \  
 do { \  
 if (!(pool) || (pool)->magic != POOL\_MAGIC\_NUMBER) { \  
 fprintf(stderr, "無効なメモリプール: %p\n", (void \*)(pool)); \  
 return; \  
 } \  
 } while (0)  
  
/\* 関数プロトタイプ \*/  
MemoryPool \*pool\_create(const char \*name, size\_t object\_size, size\_t capacity);  
void \*pool\_alloc(MemoryPool \*pool);  
void pool\_free(MemoryPool \*pool, void \*ptr);  
void pool\_print\_status(const MemoryPool \*pool);  
void pool\_destroy(MemoryPool \*pool);  
  
/\* メモリプールの初期化 \*/  
MemoryPool \*pool\_create(const char \*name, size\_t object\_size, size\_t capacity)  
{  
 size\_t min\_size;  
 size\_t total\_size;  
 MemoryPool \*pool;  
 void \*memory;  
 char \*ptr;  
 size\_t i;  
   
 if (!name || object\_size == 0 || capacity == 0) {  
 fprintf(stderr, "無効なプール作成パラメータ\n");  
 return NULL;  
 }  
   
 /\* オブジェクトサイズの調整（最小サイズ・アライメント確保） \*/  
 min\_size = sizeof(FreeNode);  
 if (object\_size < min\_size) {  
 object\_size = min\_size;  
 }  
 object\_size = ALIGN\_SIZE(object\_size, POOL\_ALIGNMENT);  
   
 /\* プール構造体の割り当て \*/  
 pool = (MemoryPool \*)malloc(sizeof(MemoryPool));  
 if (!pool) {  
 fprintf(stderr, "プール構造体の割り当てに失敗\n");  
 return NULL;  
 }  
   
 /\* メモリ領域の割り当て \*/  
 total\_size = object\_size \* capacity;  
 memory = malloc(total\_size);  
 if (!memory) {  
 fprintf(stderr, "プールメモリの割り当てに失敗: %lu バイト\n",   
 (unsigned long)total\_size);  
 free(pool);  
 return NULL;  
 }  
   
 /\* プール構造体の初期化 \*/  
 pool->memory\_chunk = memory;  
 pool->object\_size = object\_size;  
 pool->pool\_capacity = capacity;  
 pool->objects\_in\_use = 0;  
 pool->total\_allocated = 0;  
 pool->total\_freed = 0;  
 pool->magic = POOL\_MAGIC\_NUMBER;  
 strncpy(pool->name, name, sizeof(pool->name) - 1);  
 pool->name[sizeof(pool->name) - 1] = '\0';  
   
 /\* フリーリストの構築 \*/  
 pool->free\_list = NULL;  
 ptr = (char \*)memory;  
   
 for (i = 0; i < capacity; i++) {  
 FreeNode \*node = (FreeNode \*)(ptr + i \* object\_size);  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
 }  
   
 printf("[POOL DEBUG] プール '%s' を作成: オブジェクトサイズ=%lu, 容量=%lu\n",  
 pool->name, (unsigned long)pool->object\_size,   
 (unsigned long)pool->pool\_capacity);  
   
 return pool;  
}  
  
/\* オブジェクトの割り当て \*/  
void \*pool\_alloc(MemoryPool \*pool)  
{  
 FreeNode \*node;  
   
 VALIDATE\_POOL(pool);  
   
 if (!pool->free\_list) {  
 fprintf(stderr, "プール '%s' にオブジェクトが残っていません\n", pool->name);  
 return NULL;  
 }  
   
 /\* フリーリストから先頭を取得 \*/  
 node = pool->free\_list;  
 pool->free\_list = node->next;  
   
 /\* 統計情報の更新 \*/  
 pool->objects\_in\_use++;  
 pool->total\_allocated++;  
   
 /\* メモリをクリア（セキュリティ向上） \*/  
 memset(node, 0, pool->object\_size);  
   
 printf("[POOL DEBUG] オブジェクト割り当て: %p (使用中: %lu/%lu)\n",  
 (void \*)node, (unsigned long)pool->objects\_in\_use,   
 (unsigned long)pool->pool\_capacity);  
   
 return (void \*)node;  
}  
  
/\* オブジェクトの解放 \*/  
void pool\_free(MemoryPool \*pool, void \*ptr)  
{  
 char \*pool\_start;  
 char \*pool\_end;  
 char \*obj\_ptr;  
 size\_t offset;  
 FreeNode \*node;  
   
 VALIDATE\_POOL\_VOID(pool);  
   
 if (!ptr) {  
 fprintf(stderr, "NULL ポインタの解放試行\n");  
 return;  
 }  
   
 /\* ポインタがプール内にあるかチェック \*/  
 pool\_start = (char \*)pool->memory\_chunk;  
 pool\_end = pool\_start + (pool->object\_size \* pool->pool\_capacity);  
 obj\_ptr = (char \*)ptr;  
   
 if (obj\_ptr < pool\_start || obj\_ptr >= pool\_end) {  
 fprintf(stderr, "プール外のポインタの解放試行: %p\n", ptr);  
 return;  
 }  
   
 /\* アライメントチェック \*/  
 offset = obj\_ptr - pool\_start;  
 if (offset % pool->object\_size != 0) {  
 fprintf(stderr, "不正にアラインされたポインタ: %p\n", ptr);  
 return;  
 }  
   
 /\* フリーリストに追加 \*/  
 node = (FreeNode \*)ptr;  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
   
 /\* 統計情報の更新 \*/  
 pool->objects\_in\_use--;  
 pool->total\_freed++;  
   
 printf("[POOL DEBUG] オブジェクト解放: %p (使用中: %lu/%lu)\n",  
 ptr, (unsigned long)pool->objects\_in\_use,   
 (unsigned long)pool->pool\_capacity);  
}  
  
/\* プールの使用状況表示 \*/  
void pool\_print\_status(const MemoryPool \*pool)  
{  
 size\_t free\_count = 0;  
 FreeNode \*current;  
 double usage\_ratio;  
 double memory\_efficiency;  
   
 if (!pool || pool->magic != POOL\_MAGIC\_NUMBER) {  
 printf("無効なプール\n");  
 return;  
 }  
   
 printf("\n=== プール '%s' の状況 ===\n", pool->name);  
 printf("オブジェクトサイズ: %lu バイト\n", (unsigned long)pool->object\_size);  
 printf("プール容量: %lu オブジェクト\n", (unsigned long)pool->pool\_capacity);  
 printf("使用中: %lu オブジェクト\n", (unsigned long)pool->objects\_in\_use);  
 printf("空き: %lu オブジェクト\n",   
 (unsigned long)(pool->pool\_capacity - pool->objects\_in\_use));  
   
 usage\_ratio = (double)pool->objects\_in\_use / pool->pool\_capacity \* 100.0;  
 printf("使用率: %.1f%%\n", usage\_ratio);  
   
 printf("総割り当て: %lu 回\n", (unsigned long)pool->total\_allocated);  
 printf("総解放: %lu 回\n", (unsigned long)pool->total\_freed);  
   
 memory\_efficiency = pool->pool\_capacity > 0 ?   
 (double)(pool->pool\_capacity - pool->objects\_in\_use) / pool->pool\_capacity \* 100.0 : 0.0;  
 printf("メモリ効率: %.1f%%\n", memory\_efficiency);  
   
 /\* フリーリストの長さをカウント \*/  
 current = pool->free\_list;  
 while (current) {  
 free\_count++;  
 current = current->next;  
 }  
 printf("フリーリスト長: %lu\n", (unsigned long)free\_count);  
 printf("========================\n\n");  
}  
  
/\* プールの破棄 \*/  
void pool\_destroy(MemoryPool \*pool)  
{  
 if (!pool) {  
 return;  
 }  
   
 if (pool->magic != POOL\_MAGIC\_NUMBER) {  
 fprintf(stderr, "破損したプールの破棄試行\n");  
 return;  
 }  
   
 printf("[POOL DEBUG] プール '%s' を破棄 (リーク: %lu オブジェクト)\n",  
 pool->name, (unsigned long)pool->objects\_in\_use);  
   
 if (pool->objects\_in\_use > 0) {  
 fprintf(stderr, "警告: プール '%s' に %lu 個の未解放オブジェクトがあります\n",  
 pool->name, (unsigned long)pool->objects\_in\_use);  
 }  
   
 /\* メモリ解放 \*/  
 if (pool->memory\_chunk) {  
 free(pool->memory\_chunk);  
 }  
   
 /\* マジックナンバーをクリアして破損をマーク \*/  
 pool->magic = 0;  
 free(pool);  
}  
  
/\* テスト用構造体 \*/  
typedef struct TestObject {  
 int id;  
 char name[16];  
 double value;  
} TestObject;  
  
typedef struct Point {  
 double x, y, z;  
} Point;  
  
/\* プールの基本機能テスト \*/  
void test\_basic\_pool\_operations(void)  
{  
 MemoryPool \*int\_pool;  
 int \*numbers[5];  
 int i;  
 int \*new\_num;  
   
 printf("=== 基本プール操作テスト ===\n");  
   
 /\* 小さなオブジェクト用プール \*/  
 int\_pool = pool\_create("IntPool", sizeof(int), 8);  
 if (!int\_pool) {  
 printf("プール作成に失敗\n");  
 return;  
 }  
   
 pool\_print\_status(int\_pool);  
   
 /\* オブジェクトの割り当て \*/  
 for (i = 0; i < 5; i++) {  
 numbers[i] = (int \*)pool\_alloc(int\_pool);  
 if (numbers[i]) {  
 \*numbers[i] = (i + 1) \* 10;  
 printf("割り当て [%d]: %p = %d\n", i, (void \*)numbers[i], \*numbers[i]);  
 }  
 }  
   
 pool\_print\_status(int\_pool);  
   
 /\* 一部解放 \*/  
 printf("\n一部のオブジェクトを解放...\n");  
 pool\_free(int\_pool, numbers[1]);  
 pool\_free(int\_pool, numbers[3]);  
 numbers[1] = numbers[3] = NULL;  
   
 pool\_print\_status(int\_pool);  
   
 /\* 再割り当て \*/  
 printf("再割り当てテスト...\n");  
 new\_num = (int \*)pool\_alloc(int\_pool);  
 if (new\_num) {  
 \*new\_num = 999;  
 printf("再割り当て: %p = %d\n", (void \*)new\_num, \*new\_num);  
 }  
   
 /\* 残りを解放 \*/  
 for (i = 0; i < 5; i++) {  
 if (numbers[i]) {  
 pool\_free(int\_pool, numbers[i]);  
 }  
 }  
 pool\_free(int\_pool, new\_num);  
   
 pool\_print\_status(int\_pool);  
 pool\_destroy(int\_pool);  
}  
  
/\* 構造体プールのテスト \*/  
void test\_struct\_pool(void)  
{  
 MemoryPool \*obj\_pool;  
 TestObject \*objects[4];  
 int i;  
   
 printf("\n=== 構造体プールテスト ===\n");  
   
 obj\_pool = pool\_create("ObjectPool", sizeof(TestObject), 6);  
 if (!obj\_pool) {  
 return;  
 }  
   
 /\* オブジェクトの作成と初期化 \*/  
 for (i = 0; i < 4; i++) {  
 objects[i] = (TestObject \*)pool\_alloc(obj\_pool);  
 if (objects[i]) {  
 objects[i]->id = i + 100;  
 sprintf(objects[i]->name, "Obj\_%d", i + 1);  
 objects[i]->value = (i + 1) \* 3.14;  
   
 printf("オブジェクト [%d]: ID=%d, Name=%s, Value=%.2f\n",  
 i, objects[i]->id, objects[i]->name, objects[i]->value);  
 }  
 }  
   
 pool\_print\_status(obj\_pool);  
   
 /\* 全て解放 \*/  
 for (i = 0; i < 4; i++) {  
 pool\_free(obj\_pool, objects[i]);  
 }  
   
 pool\_destroy(obj\_pool);  
}  
  
/\* プール枯渇テスト \*/  
void test\_pool\_exhaustion(void)  
{  
 MemoryPool \*small\_pool;  
 Point \*points[5]; /\* プール容量より多く要求 \*/  
 int i;  
 Point \*new\_point;  
   
 printf("\n=== プール枯渇テスト ===\n");  
   
 small\_pool = pool\_create("SmallPool", sizeof(Point), 3);  
 if (!small\_pool) {  
 return;  
 }  
   
 for (i = 0; i < 5; i++) {  
 points[i] = (Point \*)pool\_alloc(small\_pool);  
 if (points[i]) {  
 points[i]->x = i \* 1.0;  
 points[i]->y = i \* 2.0;  
 points[i]->z = i \* 3.0;  
 printf("Point [%d]: (%.1f, %.1f, %.1f)\n",  
 i, points[i]->x, points[i]->y, points[i]->z);  
 } else {  
 printf("Point [%d]: 割り当て失敗（プール枯渇）\n", i);  
 }  
 }  
   
 pool\_print\_status(small\_pool);  
   
 /\* 一つ解放して再試行 \*/  
 printf("\n一つ解放して再試行...\n");  
 pool\_free(small\_pool, points[1]);  
 points[1] = NULL;  
   
 new\_point = (Point \*)pool\_alloc(small\_pool);  
 if (new\_point) {  
 new\_point->x = 99.0;  
 new\_point->y = 88.0;  
 new\_point->z = 77.0;  
 printf("新しいPoint: (%.1f, %.1f, %.1f)\n",  
 new\_point->x, new\_point->y, new\_point->z);  
 }  
   
 /\* クリーンアップ \*/  
 for (i = 0; i < 5; i++) {  
 if (points[i]) {  
 pool\_free(small\_pool, points[i]);  
 }  
 }  
 pool\_free(small\_pool, new\_point);  
   
 pool\_destroy(small\_pool);  
}  
  
/\* エラー処理テスト \*/  
void test\_error\_handling(void)  
{  
 MemoryPool \*pool;  
 int \*valid\_ptr;  
 int external\_var = 42;  
 char \*misaligned;  
   
 printf("\n=== エラー処理テスト ===\n");  
   
 pool = pool\_create("ErrorTestPool", sizeof(int), 4);  
 if (!pool) {  
 return;  
 }  
   
 /\* 正常なポインタ \*/  
 valid\_ptr = (int \*)pool\_alloc(pool);  
   
 /\* 不正なポインタでの解放テスト \*/  
 printf("不正なポインタ解放テスト:\n");  
   
 /\* NULL ポインタ \*/  
 pool\_free(pool, NULL);  
   
 /\* プール外のポインタ \*/  
 pool\_free(pool, &external\_var);  
   
 /\* 不正にアラインされたポインタ \*/  
 misaligned = (char \*)pool->memory\_chunk + 1;  
 pool\_free(pool, misaligned);  
   
 /\* 正常な解放 \*/  
 pool\_free(pool, valid\_ptr);  
   
 printf("エラー処理テスト完了\n");  
   
 pool\_destroy(pool);  
}  
  
/\* パフォーマンステスト \*/  
void test\_performance(void)  
{  
 const size\_t iterations = 100000;  
 const size\_t pool\_size = 1000;  
 MemoryPool \*pool;  
 clock\_t start, end;  
 double elapsed;  
 size\_t i;  
 double \*ptr;  
   
 printf("\n=== パフォーマンステスト ===\n");  
   
 pool = pool\_create("PerfPool", sizeof(double), pool\_size);  
 if (!pool) {  
 return;  
 }  
   
 printf("テスト設定: %lu 回の割り当て・解放 (プールサイズ: %lu)\n",  
 (unsigned long)iterations, (unsigned long)pool\_size);  
   
 start = clock();  
   
 /\* 割り当て・即座解放を繰り返し \*/  
 for (i = 0; i < iterations; i++) {  
 ptr = (double \*)pool\_alloc(pool);  
 if (ptr) {  
 \*ptr = i \* 0.001;  
 pool\_free(pool, ptr);  
 } else {  
 /\* プール枯渇（通常は発生しない） \*/  
 break;  
 }  
 }  
   
 end = clock();  
 elapsed = ((double)(end - start)) / CLOCKS\_PER\_SEC;  
   
 printf("実行時間: %.6f 秒\n", elapsed);  
 printf("1回あたりの平均時間: %.2f ナノ秒\n",  
 elapsed \* 1000000000.0 / iterations);  
   
 pool\_print\_status(pool);  
 pool\_destroy(pool);  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== 基本的なメモリプールデモ ===\n\n");  
   
 /\* 各種テストの実行 \*/  
 test\_basic\_pool\_operations();  
 test\_struct\_pool();  
 test\_pool\_exhaustion();  
 test\_error\_handling();  
 test\_performance();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== 基本的なメモリプールデモ ===  
  
=== 基本プール操作テスト ===  
[POOL DEBUG] プール 'IntPool' を作成: オブジェクトサイズ=8, 容量=8  
  
=== プール 'IntPool' の状況 ===  
オブジェクトサイズ: 8 バイト  
プール容量: 8 オブジェクト  
使用中: 0 オブジェクト  
空き: 8 オブジェクト  
使用率: 0.0%  
総割り当て: 0 回  
総解放: 0 回  
メモリ効率: 100.0%  
フリーリスト長: 8  
========================  
  
[POOL DEBUG] オブジェクト割り当て: 0x1234560 (使用中: 1/8)  
割り当て [0]: 0x1234560 = 10  
[POOL DEBUG] オブジェクト割り当て: 0x1234568 (使用中: 2/8)  
割り当て [1]: 0x1234568 = 20  
[POOL DEBUG] オブジェクト割り当て: 0x1234570 (使用中: 3/8)  
割り当て [2]: 0x1234570 = 30  
[POOL DEBUG] オブジェクト割り当て: 0x1234578 (使用中: 4/8)  
割り当て [3]: 0x1234578 = 40  
[POOL DEBUG] オブジェクト割り当て: 0x1234580 (使用中: 5/8)  
割り当て [4]: 0x1234580 = 50  
  
=== プール 'IntPool' の状況 ===  
オブジェクトサイズ: 8 バイト  
プール容量: 8 オブジェクト  
使用中: 5 オブジェクト  
空き: 3 オブジェクト  
使用率: 62.5%  
総割り当て: 5 回  
総解放: 0 回  
メモリ効率: 37.5%  
フリーリスト長: 3  
========================  
  
一部のオブジェクトを解放...  
[POOL DEBUG] オブジェクト解放: 0x1234568 (使用中: 4/8)  
[POOL DEBUG] オブジェクト解放: 0x1234578 (使用中: 3/8)  
  
=== プール 'IntPool' の状況 ===  
オブジェクトサイズ: 8 バイト  
プール容量: 8 オブジェクト  
使用中: 3 オブジェクト  
空き: 5 オブジェクト  
使用率: 37.5%  
総割り当て: 5 回  
総解放: 2 回  
メモリ効率: 62.5%  
フリーリスト長: 5  
========================  
  
再割り当てテスト...  
[POOL DEBUG] オブジェクト割り当て: 0x1234578 (使用中: 4/8)  
再割り当て: 0x1234578 = 999  
[POOL DEBUG] オブジェクト解放: 0x1234560 (使用中: 3/8)  
[POOL DEBUG] オブジェクト解放: 0x1234570 (使用中: 2/8)  
[POOL DEBUG] オブジェクト解放: 0x1234580 (使用中: 1/8)  
[POOL DEBUG] オブジェクト解放: 0x1234578 (使用中: 0/8)  
  
=== プール 'IntPool' の状況 ===  
オブジェクトサイズ: 8 バイト  
プール容量: 8 オブジェクト  
使用中: 0 オブジェクト  
空き: 8 オブジェクト  
使用率: 0.0%  
総割り当て: 6 回  
総解放: 6 回  
メモリ効率: 100.0%  
フリーリスト長: 8  
========================  
  
[POOL DEBUG] プール 'IntPool' を破棄 (リーク: 0 オブジェクト)  
  
=== 構造体プールテスト ===  
[POOL DEBUG] プール 'ObjectPool' を作成: オブジェクトサイズ=32, 容量=6  
[POOL DEBUG] オブジェクト割り当て: 0x12345a0 (使用中: 1/6)  
オブジェクト [0]: ID=100, Name=Obj\_1, Value=3.14  
[POOL DEBUG] オブジェクト割り当て: 0x12345c0 (使用中: 2/6)  
オブジェクト [1]: ID=101, Name=Obj\_2, Value=6.28  
[POOL DEBUG] オブジェクト割り当て: 0x12345e0 (使用中: 3/6)  
オブジェクト [2]: ID=102, Name=Obj\_3, Value=9.42  
[POOL DEBUG] オブジェクト割り当て: 0x1234600 (使用中: 4/6)  
オブジェクト [3]: ID=103, Name=Obj\_4, Value=12.56  
  
=== プール 'ObjectPool' の状況 ===  
オブジェクトサイズ: 32 バイト  
プール容量: 6 オブジェクト  
使用中: 4 オブジェクト  
空き: 2 オブジェクト  
使用率: 66.7%  
総割り当て: 4 回  
総解放: 0 回  
メモリ効率: 33.3%  
フリーリスト長: 2  
========================  
  
[POOL DEBUG] オブジェクト解放: 0x12345a0 (使用中: 3/6)  
[POOL DEBUG] オブジェクト解放: 0x12345c0 (使用中: 2/6)  
[POOL DEBUG] オブジェクト解放: 0x12345e0 (使用中: 1/6)  
[POOL DEBUG] オブジェクト解放: 0x1234600 (使用中: 0/6)  
[POOL DEBUG] プール 'ObjectPool' を破棄 (リーク: 0 オブジェクト)  
  
=== プール枯渇テスト ===  
[POOL DEBUG] プール 'SmallPool' を作成: オブジェクトサイズ=24, 容量=3  
[POOL DEBUG] オブジェクト割り当て: 0x1234620 (使用中: 1/3)  
Point [0]: (0.0, 0.0, 0.0)  
[POOL DEBUG] オブジェクト割り当て: 0x1234638 (使用中: 2/3)  
Point [1]: (1.0, 2.0, 3.0)  
[POOL DEBUG] オブジェクト割り当て: 0x1234650 (使用中: 3/3)  
Point [2]: (2.0, 4.0, 6.0)  
プール 'SmallPool' にオブジェクトが残っていません  
Point [3]: 割り当て失敗（プール枯渇）  
プール 'SmallPool' にオブジェクトが残っていません  
Point [4]: 割り当て失敗（プール枯渇）  
  
=== プール 'SmallPool' の状況 ===  
オブジェクトサイズ: 24 バイト  
プール容量: 3 オブジェクト  
使用中: 3 オブジェクト  
空き: 0 オブジェクト  
使用率: 100.0%  
総割り当て: 3 回  
総解放: 0 回  
メモリ効率: 0.0%  
フリーリスト長: 0  
========================  
  
一つ解放して再試行...  
[POOL DEBUG] オブジェクト解放: 0x1234638 (使用中: 2/3)  
[POOL DEBUG] オブジェクト割り当て: 0x1234638 (使用中: 3/3)  
新しいPoint: (99.0, 88.0, 77.0)  
[POOL DEBUG] オブジェクト解放: 0x1234620 (使用中: 2/3)  
[POOL DEBUG] オブジェクト解放: 0x1234650 (使用中: 1/3)  
[POOL DEBUG] オブジェクト解放: 0x1234638 (使用中: 0/3)  
[POOL DEBUG] プール 'SmallPool' を破棄 (リーク: 0 オブジェクト)  
  
=== エラー処理テスト ===  
[POOL DEBUG] プール 'ErrorTestPool' を作成: オブジェクトサイズ=8, 容量=4  
[POOL DEBUG] オブジェクト割り当て: 0x1234668 (使用中: 1/4)  
不正なポインタ解放テスト:  
NULL ポインタの解放試行  
プール外のポインタの解放試行: 0x7ffd1234abcd  
不正にアラインされたポインタ: 0x1234669  
[POOL DEBUG] オブジェクト解放: 0x1234668 (使用中: 0/4)  
エラー処理テスト完了  
[POOL DEBUG] プール 'ErrorTestPool' を破棄 (リーク: 0 オブジェクト)  
  
=== パフォーマンステスト ===  
[POOL DEBUG] プール 'PerfPool' を作成: オブジェクトサイズ=8, 容量=1000  
テスト設定: 100000 回の割り当て・解放 (プールサイズ: 1000)  
実行時間: 0.012345 秒  
1回あたりの平均時間: 123.45 ナノ秒  
  
=== プール 'PerfPool' の状況 ===  
オブジェクトサイズ: 8 バイト  
プール容量: 1000 オブジェクト  
使用中: 0 オブジェクト  
空き: 1000 オブジェクト  
使用率: 0.0%  
総割り当て: 100000 回  
総解放: 100000 回  
メモリ効率: 100.0%  
フリーリスト長: 1000  
========================  
  
[POOL DEBUG] プール 'PerfPool' を破棄 (リーク: 0 オブジェクト)  
=== デモ完了 ===  
\*/```  
  
### ex13\_4\_generic\_macros.c  
  
```c  
/\*  
 \* 演習13-4の解答例: 汎用的なプリプロセッサライブラリ  
 \* ファイル名: ex13\_4\_generic\_macros.c  
 \* 説明: 型判定、汎用スワップ、ループ展開、ベンチマーク計測マクロ  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <time.h>  
#include <assert.h>  
  
/\* 型サイズによる型判定マクロ（簡易版） \*/  
#define TYPE\_SIZE\_MATCH(a, b) (sizeof(a) == sizeof(b))  
  
/\* 型の互換性チェック（C90では限定的） \*/  
#define TYPE\_CHECK(a, b) \  
 (TYPE\_SIZE\_MATCH(a, b) && \  
 TYPE\_SIZE\_MATCH(&(a), &(b)))  
  
/\* 汎用的なスワップマクロ（型非依存） \*/  
#define GENERIC\_SWAP(a, b) \  
 do { \  
 char \_temp[sizeof(a)]; \  
 assert(sizeof(a) == sizeof(b)); \  
 memcpy(\_temp, &(a), sizeof(a)); \  
 memcpy(&(a), &(b), sizeof(a)); \  
 memcpy(&(b), \_temp, sizeof(a)); \  
 } while (0)  
  
/\* 高速スワップマクロ（小さい型用） \*/  
#define FAST\_SWAP(type, a, b) \  
 do { \  
 type \_temp = (a); \  
 (a) = (b); \  
 (b) = \_temp; \  
 } while (0)  
  
/\* ループ展開マクロ \*/  
#define REPEAT\_2(code) code code  
#define REPEAT\_4(code) REPEAT\_2(code) REPEAT\_2(code)  
#define REPEAT\_8(code) REPEAT\_4(code) REPEAT\_4(code)  
#define REPEAT\_16(code) REPEAT\_8(code) REPEAT\_8(code)  
  
/\* 可変回数ループマクロ（最大16回） \*/  
#define REPEAT(n, code) \  
 do { \  
 if ((n) >= 16) { REPEAT\_16(code) } \  
 else if ((n) >= 8) { REPEAT\_8(code) } \  
 else if ((n) >= 4) { REPEAT\_4(code) } \  
 else if ((n) >= 2) { REPEAT\_2(code) } \  
 else if ((n) >= 1) { code } \  
 } while (0)  
  
/\* インデックス付きループマクロ \*/  
#define REPEAT\_WITH\_INDEX(n, i, code) \  
 do { \  
 int i; \  
 for (i = 0; i < (n); i++) { \  
 code \  
 } \  
 } while (0)  
  
/\* ベンチマーク計測マクロ \*/  
static clock\_t \_bench\_start\_time;  
static double \_bench\_elapsed\_time;  
  
#define BENCHMARK\_START(name) \  
 do { \  
 printf("ベンチマーク開始: %s\n", name); \  
 \_bench\_start\_time = clock(); \  
 } while (0)  
  
#define BENCHMARK\_END(name) \  
 do { \  
 clock\_t \_bench\_end\_time = clock(); \  
 \_bench\_elapsed\_time = ((double)(\_bench\_end\_time - \_bench\_start\_time)) / CLOCKS\_PER\_SEC; \  
 printf("ベンチマーク終了: %s (実行時間: %.6f秒)\n", name, \_bench\_elapsed\_time); \  
 } while (0)  
  
#define BENCHMARK\_BLOCK(name, code) \  
 do { \  
 BENCHMARK\_START(name); \  
 code \  
 BENCHMARK\_END(name); \  
 } while (0)  
  
/\* コンパイル時アサーション（C90版） \*/  
#define COMPILE\_TIME\_ASSERT(condition) \  
 typedef char \_compile\_time\_assert\_##\_\_LINE\_\_[(condition) ? 1 : -1]  
  
/\* 静的アサーションマクロ \*/  
#define STATIC\_ASSERT(condition, message) \  
 COMPILE\_TIME\_ASSERT(condition)  
  
/\* 配列サイズマクロ \*/  
#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))  
  
/\* 最大・最小マクロ（型安全版） \*/  
#define MAX(a, b) ((a) > (b) ? (a) : (b))  
#define MIN(a, b) ((a) < (b) ? (a) : (b))  
  
/\* 絶対値マクロ \*/  
#define ABS(x) ((x) < 0 ? -(x) : (x))  
  
/\* ビット操作マクロ \*/  
#define BIT\_SET(value, bit) ((value) |= (1U << (bit)))  
#define BIT\_CLEAR(value, bit) ((value) &= ~(1U << (bit)))  
#define BIT\_TOGGLE(value, bit) ((value) ^= (1U << (bit)))  
#define BIT\_CHECK(value, bit) (((value) >> (bit)) & 1U)  
  
/\* 文字列化・連結マクロ \*/  
#define STRINGIFY(x) #x  
#define TOSTRING(x) STRINGIFY(x)  
#define CONCAT(a, b) a##b  
#define CONCAT3(a, b, c) a##b##c  
  
/\* デバッグ出力マクロ \*/  
#ifdef DEBUG  
#define DEBUG\_PRINT(msg) printf("[DEBUG:%s:%d] %s\n", \_\_FILE\_\_, \_\_LINE\_\_, msg)  
#define DEBUG\_VAR(var, fmt) printf("[DEBUG:%s:%d] %s = " fmt "\n", \_\_FILE\_\_, \_\_LINE\_\_, #var, var)  
#else  
#define DEBUG\_PRINT(msg)  
#define DEBUG\_VAR(var, fmt)  
#endif  
  
/\* エラーチェックマクロ \*/  
#define CHECK\_NULL(ptr) \  
 do { \  
 if ((ptr) == NULL) { \  
 fprintf(stderr, "エラー: NULLポインタ検出 (%s:%d)\n", \_\_FILE\_\_, \_\_LINE\_\_); \  
 return -1; \  
 } \  
 } while (0)  
  
/\* テスト関数群 \*/  
  
/\* 型チェックのテスト \*/  
void test\_type\_checking(void)  
{  
 int a = 10, b = 20;  
 double x = 3.14, y = 2.71;  
 char c = 'A';  
   
 printf("=== 型チェックテスト ===\n");  
   
 printf("TYPE\_CHECK(int, int): %s\n",   
 TYPE\_CHECK(a, b) ? "一致" : "不一致");  
   
 printf("TYPE\_CHECK(double, double): %s\n",   
 TYPE\_CHECK(x, y) ? "一致" : "不一致");  
   
 printf("TYPE\_CHECK(int, char): %s\n",   
 TYPE\_CHECK(a, c) ? "一致" : "不一致");  
   
 printf("TYPE\_SIZE\_MATCH(int, float): %s\n",   
 TYPE\_SIZE\_MATCH(a, (float)x) ? "同じサイズ" : "異なるサイズ");  
   
 printf("\n");  
}  
  
/\* 汎用スワップのテスト \*/  
void test\_generic\_swap(void)  
{  
 int a = 100, b = 200;  
 double x = 1.23, y = 4.56;  
 char s1[] = "Hello";  
 char s2[] = "World";  
   
 struct Point {  
 int x, y;  
 } p1 = {10, 20}, p2 = {30, 40};  
   
 printf("=== 汎用スワップテスト ===\n");  
   
 /\* 整数のスワップ \*/  
 printf("整数: 交換前 a=%d, b=%d\n", a, b);  
 GENERIC\_SWAP(a, b);  
 printf("整数: 交換後 a=%d, b=%d\n", a, b);  
   
 /\* 浮動小数点のスワップ \*/  
 printf("浮動小数点: 交換前 x=%.2f, y=%.2f\n", x, y);  
 GENERIC\_SWAP(x, y);  
 printf("浮動小数点: 交換後 x=%.2f, y=%.2f\n", x, y);  
   
 /\* 構造体のスワップ \*/  
 printf("構造体: 交換前 p1=(%d,%d), p2=(%d,%d)\n",   
 p1.x, p1.y, p2.x, p2.y);  
 GENERIC\_SWAP(p1, p2);  
 printf("構造体: 交換後 p1=(%d,%d), p2=(%d,%d)\n",   
 p1.x, p1.y, p2.x, p2.y);  
   
 /\* 高速スワップとの比較 \*/  
 BENCHMARK\_BLOCK("高速スワップ(100万回)", {  
 int i;  
 for (i = 0; i < 1000000; i++) {  
 FAST\_SWAP(int, a, b);  
 }  
 });  
   
 BENCHMARK\_BLOCK("汎用スワップ(100万回)", {  
 int i;  
 for (i = 0; i < 1000000; i++) {  
 GENERIC\_SWAP(a, b);  
 }  
 });  
   
 printf("\n");  
}  
  
/\* ループ展開のテスト \*/  
void test\_loop\_unrolling(void)  
{  
 int sum1 = 0, sum2 = 0;  
 int array[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};  
 int i;  
   
 printf("=== ループ展開テスト ===\n");  
   
 /\* 通常のループ \*/  
 BENCHMARK\_START("通常ループ(1600万回)");  
 for (i = 0; i < 1000000; i++) {  
 int j;  
 for (j = 0; j < 16; j++) {  
 sum1 += array[j];  
 }  
 }  
 BENCHMARK\_END("通常ループ(1600万回)");  
   
 /\* 展開されたループ \*/  
 BENCHMARK\_START("展開ループ(1600万回)");  
 for (i = 0; i < 1000000; i++) {  
 REPEAT\_16(sum2 += array[0];);  
 }  
 BENCHMARK\_END("展開ループ(1600万回)");  
   
 printf("結果確認: sum1=%d, sum2=%d\n", sum1, sum2/16);  
   
 /\* REPEATマクロのテスト \*/  
 printf("\nREPEATマクロテスト:\n");  
 printf("REPEAT(5): ");  
 i = 0;  
 REPEAT(5, { printf("%d ", i++); });  
 printf("\n");  
   
 /\* インデックス付きループ \*/  
 printf("REPEAT\_WITH\_INDEX(10): ");  
 REPEAT\_WITH\_INDEX(10, idx, {  
 printf("%d ", idx);  
 });  
 printf("\n\n");  
}  
  
/\* ビット操作マクロのテスト \*/  
void test\_bit\_operations(void)  
{  
 unsigned int value = 0;  
 int i;  
   
 printf("=== ビット操作テスト ===\n");  
   
 /\* ビットセット \*/  
 BIT\_SET(value, 0);  
 BIT\_SET(value, 2);  
 BIT\_SET(value, 4);  
 BIT\_SET(value, 7);  
   
 printf("ビットセット後: 0x%02X (", value);  
 for (i = 7; i >= 0; i--) {  
 printf("%d", BIT\_CHECK(value, i));  
 }  
 printf(")\n");  
   
 /\* ビットクリア \*/  
 BIT\_CLEAR(value, 2);  
 printf("ビット2クリア後: 0x%02X\n", value);  
   
 /\* ビットトグル \*/  
 BIT\_TOGGLE(value, 3);  
 BIT\_TOGGLE(value, 4);  
 printf("ビット3,4トグル後: 0x%02X\n", value);  
   
 printf("\n");  
}  
  
/\* 静的アサーションのテスト \*/  
void test\_static\_assertions(void)  
{  
 printf("=== 静的アサーションテスト ===\n");  
   
 /\* コンパイル時のサイズチェック \*/  
 STATIC\_ASSERT(sizeof(int) >= 4, "intは少なくとも4バイト必要");  
 STATIC\_ASSERT(sizeof(void\*) >= sizeof(int), "ポインタはint以上のサイズ必要");  
   
 /\* 配列サイズの確認 \*/  
 int array[10];  
 STATIC\_ASSERT(ARRAY\_SIZE(array) == 10, "配列サイズが一致しない");  
   
 printf("すべての静的アサーションをパス\n");  
 printf("sizeof(int) = %lu\n", (unsigned long)sizeof(int));  
 printf("sizeof(void\*) = %lu\n", (unsigned long)sizeof(void\*));  
   
 printf("\n");  
}  
  
/\* ベンチマークマクロの包括的テスト \*/  
void test\_benchmark\_macros(void)  
{  
 int i, j;  
 volatile int dummy = 0;  
   
 printf("=== ベンチマークマクロテスト ===\n");  
   
 /\* ネストしたループのベンチマーク \*/  
 BENCHMARK\_BLOCK("二重ループ(1000x1000)", {  
 for (i = 0; i < 1000; i++) {  
 for (j = 0; j < 1000; j++) {  
 dummy = i \* j;  
 }  
 }  
 });  
   
 /\* 関数呼び出しのオーバーヘッド測定 \*/  
 BENCHMARK\_BLOCK("空関数呼び出し(100万回)", {  
 for (i = 0; i < 1000000; i++) {  
 /\* 空の処理 \*/  
 }  
 });  
   
 printf("\n");  
}  
  
/\* マクロ合成のデモ \*/  
void test\_macro\_composition(void)  
{  
 int array[] = {5, 2, 8, 1, 9, 3, 7, 4, 6};  
 int size = ARRAY\_SIZE(array);  
 int i, j;  
   
 printf("=== マクロ合成デモ ===\n");  
   
 /\* バブルソートをマクロで実装 \*/  
 BENCHMARK\_BLOCK("マクロベースのバブルソート", {  
 for (i = 0; i < size - 1; i++) {  
 for (j = 0; j < size - i - 1; j++) {  
 if (array[j] > array[j + 1]) {  
 GENERIC\_SWAP(array[j], array[j + 1]);  
 }  
 }  
 }  
 });  
   
 printf("ソート結果: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n\n");  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== 汎用的なプリプロセッサライブラリデモ ===\n\n");  
   
 /\* 各種テストの実行 \*/  
 test\_type\_checking();  
 test\_generic\_swap();  
 test\_loop\_unrolling();  
 test\_bit\_operations();  
 test\_static\_assertions();  
 test\_benchmark\_macros();  
 test\_macro\_composition();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== 汎用的なプリプロセッサライブラリデモ ===  
  
=== 型チェックテスト ===  
TYPE\_CHECK(int, int): 一致  
TYPE\_CHECK(double, double): 一致  
TYPE\_CHECK(int, char): 不一致  
TYPE\_SIZE\_MATCH(int, float): 同じサイズ  
  
=== 汎用スワップテスト ===  
整数: 交換前 a=100, b=200  
整数: 交換後 a=200, b=100  
浮動小数点: 交換前 x=1.23, y=4.56  
浮動小数点: 交換後 x=4.56, y=1.23  
構造体: 交換前 p1=(10,20), p2=(30,40)  
構造体: 交換後 p1=(30,40), p2=(10,20)  
ベンチマーク開始: 高速スワップ(100万回)  
ベンチマーク終了: 高速スワップ(100万回) (実行時間: 0.002345秒)  
ベンチマーク開始: 汎用スワップ(100万回)  
ベンチマーク終了: 汎用スワップ(100万回) (実行時間: 0.015678秒)  
  
=== ループ展開テスト ===  
ベンチマーク開始: 通常ループ(1600万回)  
ベンチマーク終了: 通常ループ(1600万回) (実行時間: 0.045678秒)  
ベンチマーク開始: 展開ループ(1600万回)  
ベンチマーク終了: 展開ループ(1600万回) (実行時間: 0.023456秒)  
結果確認: sum1=136000000, sum2=136000000  
  
REPEATマクロテスト:  
REPEAT(5): 0 1 2 3 4   
REPEAT\_WITH\_INDEX(10): 0 1 2 3 4 5 6 7 8 9   
  
=== ビット操作テスト ===  
ビットセット後: 0x95 (10010101)  
ビット2クリア後: 0x91  
ビット3,4トグル後: 0x89  
  
=== 静的アサーションテスト ===  
すべての静的アサーションをパス  
sizeof(int) = 4  
sizeof(void\*) = 8  
  
=== ベンチマークマクロテスト ===  
ベンチマーク開始: 二重ループ(1000x1000)  
ベンチマーク終了: 二重ループ(1000x1000) (実行時間: 0.003456秒)  
ベンチマーク開始: 空関数呼び出し(100万回)  
ベンチマーク終了: 空関数呼び出し(100万回) (実行時間: 0.001234秒)  
  
=== マクロ合成デモ ===  
ベンチマーク開始: マクロベースのバブルソート  
ベンチマーク終了: マクロベースのバブルソート (実行時間: 0.000012秒)  
ソート結果: 1 2 3 4 5 6 7 8 9   
  
=== デモ完了 ===  
\*/```  
  
### ex13\_5\_allocator.c  
  
```c  
/\*  
 \* 演習13-5の解答例: 高性能メモリアロケーター  
 \* ファイル名: ex13\_5\_allocator.c  
 \* 説明: サイズ別メモリプール、断片化最小化、統計情報収集  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <assert.h>  
  
/\* アロケーターの設定 \*/  
#define SMALL\_OBJECT\_SIZE 64  
#define MEDIUM\_OBJECT\_SIZE 256  
#define LARGE\_OBJECT\_SIZE 1024  
  
#define SMALL\_POOL\_CAPACITY 1024  
#define MEDIUM\_POOL\_CAPACITY 256  
#define LARGE\_POOL\_CAPACITY 64  
  
#define ALLOCATOR\_MAGIC 0xAL0CAT0R  
#define BLOCK\_MAGIC 0xB10CK5  
  
/\* デバッグモード \*/  
#define DEBUG\_ALLOCATOR 1  
  
#if DEBUG\_ALLOCATOR  
#define ALLOC\_DEBUG(msg) printf("[ALLOCATOR] %s\n", msg)  
#else  
#define ALLOC\_DEBUG(msg)  
#endif  
  
/\* メモリブロックヘッダー \*/  
typedef struct BlockHeader {  
 size\_t size;  
 unsigned int magic;  
 struct BlockHeader \*next;  
 struct BlockHeader \*prev;  
 int pool\_type; /\* 0: small, 1: medium, 2: large, 3: system \*/  
 int is\_free;  
} BlockHeader;  
  
/\* フリーリストノード \*/  
typedef struct FreeNode {  
 struct FreeNode \*next;  
 size\_t size;  
} FreeNode;  
  
/\* メモリプール構造体 \*/  
typedef struct MemoryPool {  
 void \*memory\_chunk;  
 FreeNode \*free\_list;  
 size\_t object\_size;  
 size\_t capacity;  
 size\_t used\_count;  
 size\_t free\_count;  
} MemoryPool;  
  
/\* アロケーター統計情報 \*/  
typedef struct AllocStats {  
 size\_t total\_allocated;  
 size\_t total\_freed;  
 size\_t current\_usage;  
 size\_t peak\_usage;  
 size\_t small\_allocs;  
 size\_t medium\_allocs;  
 size\_t large\_allocs;  
 size\_t system\_allocs;  
 size\_t failed\_allocs;  
 double total\_alloc\_time;  
 double total\_free\_time;  
} AllocStats;  
  
/\* カスタムアロケーター構造体 \*/  
typedef struct CustomAllocator {  
 MemoryPool small\_pool;  
 MemoryPool medium\_pool;  
 MemoryPool large\_pool;  
 BlockHeader \*system\_blocks;  
 AllocStats stats;  
 unsigned int magic;  
 int initialized;  
} CustomAllocator;  
  
/\* グローバルアロケーターインスタンス \*/  
static CustomAllocator g\_allocator = {0};  
  
/\* ヘルパー関数のプロトタイプ \*/  
static int init\_pool(MemoryPool \*pool, size\_t object\_size, size\_t capacity);  
static void \*pool\_alloc(MemoryPool \*pool);  
static int pool\_free(MemoryPool \*pool, void \*ptr);  
static void destroy\_pool(MemoryPool \*pool);  
static double get\_time\_sec(void);  
  
/\* 時間計測関数 \*/  
static double get\_time\_sec(void)  
{  
 return (double)clock() / CLOCKS\_PER\_SEC;  
}  
  
/\* メモリプールの初期化 \*/  
static int init\_pool(MemoryPool \*pool, size\_t object\_size, size\_t capacity)  
{  
 size\_t total\_size;  
 char \*ptr;  
 size\_t i;  
   
 pool->object\_size = object\_size;  
 pool->capacity = capacity;  
 pool->used\_count = 0;  
 pool->free\_count = capacity;  
   
 total\_size = object\_size \* capacity;  
 pool->memory\_chunk = malloc(total\_size);  
 if (!pool->memory\_chunk) {  
 return -1;  
 }  
   
 /\* フリーリストの構築 \*/  
 pool->free\_list = NULL;  
 ptr = (char \*)pool->memory\_chunk;  
   
 for (i = 0; i < capacity; i++) {  
 FreeNode \*node = (FreeNode \*)(ptr + i \* object\_size);  
 node->size = object\_size;  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
 }  
   
 return 0;  
}  
  
/\* プールからメモリ割り当て \*/  
static void \*pool\_alloc(MemoryPool \*pool)  
{  
 FreeNode \*node;  
   
 if (!pool->free\_list) {  
 return NULL;  
 }  
   
 node = pool->free\_list;  
 pool->free\_list = node->next;  
 pool->used\_count++;  
 pool->free\_count--;  
   
 return (void \*)node;  
}  
  
/\* プールへメモリ解放 \*/  
static int pool\_free(MemoryPool \*pool, void \*ptr)  
{  
 FreeNode \*node;  
 char \*pool\_start;  
 char \*pool\_end;  
 char \*obj\_ptr;  
   
 /\* プール範囲チェック \*/  
 pool\_start = (char \*)pool->memory\_chunk;  
 pool\_end = pool\_start + (pool->object\_size \* pool->capacity);  
 obj\_ptr = (char \*)ptr;  
   
 if (obj\_ptr < pool\_start || obj\_ptr >= pool\_end) {  
 return -1;  
 }  
   
 node = (FreeNode \*)ptr;  
 node->size = pool->object\_size;  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
   
 pool->used\_count--;  
 pool->free\_count++;  
   
 return 0;  
}  
  
/\* プールの破棄 \*/  
static void destroy\_pool(MemoryPool \*pool)  
{  
 if (pool->memory\_chunk) {  
 free(pool->memory\_chunk);  
 pool->memory\_chunk = NULL;  
 }  
}  
  
/\* アロケーターの初期化 \*/  
int allocator\_init(void)  
{  
 if (g\_allocator.initialized) {  
 ALLOC\_DEBUG("アロケーターは既に初期化されています");  
 return 0;  
 }  
   
 memset(&g\_allocator, 0, sizeof(g\_allocator));  
   
 /\* 各プールの初期化 \*/  
 if (init\_pool(&g\_allocator.small\_pool, SMALL\_OBJECT\_SIZE, SMALL\_POOL\_CAPACITY) < 0) {  
 fprintf(stderr, "小オブジェクトプールの初期化に失敗\n");  
 return -1;  
 }  
   
 if (init\_pool(&g\_allocator.medium\_pool, MEDIUM\_OBJECT\_SIZE, MEDIUM\_POOL\_CAPACITY) < 0) {  
 fprintf(stderr, "中オブジェクトプールの初期化に失敗\n");  
 destroy\_pool(&g\_allocator.small\_pool);  
 return -1;  
 }  
   
 if (init\_pool(&g\_allocator.large\_pool, LARGE\_OBJECT\_SIZE, LARGE\_POOL\_CAPACITY) < 0) {  
 fprintf(stderr, "大オブジェクトプールの初期化に失敗\n");  
 destroy\_pool(&g\_allocator.small\_pool);  
 destroy\_pool(&g\_allocator.medium\_pool);  
 return -1;  
 }  
   
 g\_allocator.magic = ALLOCATOR\_MAGIC;  
 g\_allocator.initialized = 1;  
   
 ALLOC\_DEBUG("カスタムアロケーターを初期化しました");  
 printf(" 小プール: %d x %dバイト\n", SMALL\_POOL\_CAPACITY, SMALL\_OBJECT\_SIZE);  
 printf(" 中プール: %d x %dバイト\n", MEDIUM\_POOL\_CAPACITY, MEDIUM\_OBJECT\_SIZE);  
 printf(" 大プール: %d x %dバイト\n", LARGE\_POOL\_CAPACITY, LARGE\_OBJECT\_SIZE);  
   
 return 0;  
}  
  
/\* カスタムmalloc実装 \*/  
void \*custom\_malloc(size\_t size)  
{  
 void \*ptr = NULL;  
 BlockHeader \*header;  
 double start\_time;  
 int pool\_type = -1;  
   
 if (!g\_allocator.initialized) {  
 fprintf(stderr, "アロケーターが初期化されていません\n");  
 return NULL;  
 }  
   
 start\_time = get\_time\_sec();  
   
 /\* サイズに応じたプール選択 \*/  
 if (size <= SMALL\_OBJECT\_SIZE - sizeof(BlockHeader)) {  
 ptr = pool\_alloc(&g\_allocator.small\_pool);  
 pool\_type = 0;  
 if (ptr) g\_allocator.stats.small\_allocs++;  
 } else if (size <= MEDIUM\_OBJECT\_SIZE - sizeof(BlockHeader)) {  
 ptr = pool\_alloc(&g\_allocator.medium\_pool);  
 pool\_type = 1;  
 if (ptr) g\_allocator.stats.medium\_allocs++;  
 } else if (size <= LARGE\_OBJECT\_SIZE - sizeof(BlockHeader)) {  
 ptr = pool\_alloc(&g\_allocator.large\_pool);  
 pool\_type = 2;  
 if (ptr) g\_allocator.stats.large\_allocs++;  
 }  
   
 /\* プールから割り当てできない場合はシステムmalloc \*/  
 if (!ptr) {  
 size\_t total\_size = size + sizeof(BlockHeader);  
 header = (BlockHeader \*)malloc(total\_size);  
 if (!header) {  
 g\_allocator.stats.failed\_allocs++;  
 return NULL;  
 }  
   
 header->size = size;  
 header->magic = BLOCK\_MAGIC;  
 header->pool\_type = 3; /\* system \*/  
 header->is\_free = 0;  
 header->next = g\_allocator.system\_blocks;  
 header->prev = NULL;  
   
 if (g\_allocator.system\_blocks) {  
 g\_allocator.system\_blocks->prev = header;  
 }  
 g\_allocator.system\_blocks = header;  
   
 ptr = (char \*)header + sizeof(BlockHeader);  
 g\_allocator.stats.system\_allocs++;  
 } else {  
 /\* プールから割り当てた場合もヘッダーを設定 \*/  
 header = (BlockHeader \*)ptr;  
 header->size = size;  
 header->magic = BLOCK\_MAGIC;  
 header->pool\_type = pool\_type;  
 header->is\_free = 0;  
 header->next = NULL;  
 header->prev = NULL;  
   
 ptr = (char \*)header + sizeof(BlockHeader);  
 }  
   
 /\* 統計更新 \*/  
 g\_allocator.stats.total\_allocated += size;  
 g\_allocator.stats.current\_usage += size;  
 if (g\_allocator.stats.current\_usage > g\_allocator.stats.peak\_usage) {  
 g\_allocator.stats.peak\_usage = g\_allocator.stats.current\_usage;  
 }  
   
 g\_allocator.stats.total\_alloc\_time += get\_time\_sec() - start\_time;  
   
 return ptr;  
}  
  
/\* カスタムfree実装 \*/  
void custom\_free(void \*ptr)  
{  
 BlockHeader \*header;  
 double start\_time;  
   
 if (!ptr) return;  
   
 if (!g\_allocator.initialized) {  
 fprintf(stderr, "アロケーターが初期化されていません\n");  
 return;  
 }  
   
 start\_time = get\_time\_sec();  
   
 /\* ヘッダーの取得と検証 \*/  
 header = (BlockHeader \*)((char \*)ptr - sizeof(BlockHeader));  
 if (header->magic != BLOCK\_MAGIC) {  
 fprintf(stderr, "不正なメモリブロック\n");  
 return;  
 }  
   
 if (header->is\_free) {  
 fprintf(stderr, "二重解放の検出\n");  
 return;  
 }  
   
 /\* プールタイプに応じた解放 \*/  
 switch (header->pool\_type) {  
 case 0: /\* small pool \*/  
 pool\_free(&g\_allocator.small\_pool, header);  
 break;  
 case 1: /\* medium pool \*/  
 pool\_free(&g\_allocator.medium\_pool, header);  
 break;  
 case 2: /\* large pool \*/  
 pool\_free(&g\_allocator.large\_pool, header);  
 break;  
 case 3: /\* system allocation \*/  
 /\* リストから削除 \*/  
 if (header->prev) {  
 header->prev->next = header->next;  
 } else {  
 g\_allocator.system\_blocks = header->next;  
 }  
 if (header->next) {  
 header->next->prev = header->prev;  
 }  
 free(header);  
 break;  
 default:  
 fprintf(stderr, "不明なプールタイプ\n");  
 return;  
 }  
   
 /\* 統計更新 \*/  
 g\_allocator.stats.total\_freed += header->size;  
 g\_allocator.stats.current\_usage -= header->size;  
 g\_allocator.stats.total\_free\_time += get\_time\_sec() - start\_time;  
   
 header->is\_free = 1;  
}  
  
/\* 統計情報の表示 \*/  
void allocator\_print\_stats(void)  
{  
 AllocStats \*stats = &g\_allocator.stats;  
   
 printf("\n=== アロケーター統計情報 ===\n");  
 printf("総割り当て量: %lu バイト\n", (unsigned long)stats->total\_allocated);  
 printf("総解放量: %lu バイト\n", (unsigned long)stats->total\_freed);  
 printf("現在使用量: %lu バイト\n", (unsigned long)stats->current\_usage);  
 printf("ピーク使用量: %lu バイト\n", (unsigned long)stats->peak\_usage);  
 printf("\n");  
   
 printf("割り当て回数:\n");  
 printf(" 小 (<%d): %lu 回\n", SMALL\_OBJECT\_SIZE, (unsigned long)stats->small\_allocs);  
 printf(" 中 (<%d): %lu 回\n", MEDIUM\_OBJECT\_SIZE, (unsigned long)stats->medium\_allocs);  
 printf(" 大 (<%d): %lu 回\n", LARGE\_OBJECT\_SIZE, (unsigned long)stats->large\_allocs);  
 printf(" システム: %lu 回\n", (unsigned long)stats->system\_allocs);  
 printf(" 失敗: %lu 回\n", (unsigned long)stats->failed\_allocs);  
 printf("\n");  
   
 printf("プール使用状況:\n");  
 printf(" 小: %lu/%d (%.1f%%)\n",   
 (unsigned long)g\_allocator.small\_pool.used\_count,  
 SMALL\_POOL\_CAPACITY,  
 (double)g\_allocator.small\_pool.used\_count / SMALL\_POOL\_CAPACITY \* 100);  
 printf(" 中: %lu/%d (%.1f%%)\n",  
 (unsigned long)g\_allocator.medium\_pool.used\_count,  
 MEDIUM\_POOL\_CAPACITY,  
 (double)g\_allocator.medium\_pool.used\_count / MEDIUM\_POOL\_CAPACITY \* 100);  
 printf(" 大: %lu/%d (%.1f%%)\n",  
 (unsigned long)g\_allocator.large\_pool.used\_count,  
 LARGE\_POOL\_CAPACITY,  
 (double)g\_allocator.large\_pool.used\_count / LARGE\_POOL\_CAPACITY \* 100);  
 printf("\n");  
   
 if (stats->small\_allocs + stats->medium\_allocs + stats->large\_allocs + stats->system\_allocs > 0) {  
 printf("平均割り当て時間: %.6f 秒\n",  
 stats->total\_alloc\_time /   
 (stats->small\_allocs + stats->medium\_allocs + stats->large\_allocs + stats->system\_allocs));  
 }  
   
 if (stats->total\_freed > 0) {  
 printf("平均解放時間: %.6f 秒\n",  
 stats->total\_free\_time /   
 (stats->total\_allocated > 0 ? stats->total\_freed / sizeof(BlockHeader) : 1));  
 }  
   
 printf("=============================\n");  
}  
  
/\* アロケーターの終了処理 \*/  
void allocator\_shutdown(void)  
{  
 BlockHeader \*block, \*next;  
   
 if (!g\_allocator.initialized) {  
 return;  
 }  
   
 /\* システムブロックの解放 \*/  
 block = g\_allocator.system\_blocks;  
 while (block) {  
 next = block->next;  
 free(block);  
 block = next;  
 }  
   
 /\* プールの破棄 \*/  
 destroy\_pool(&g\_allocator.small\_pool);  
 destroy\_pool(&g\_allocator.medium\_pool);  
 destroy\_pool(&g\_allocator.large\_pool);  
   
 g\_allocator.initialized = 0;  
 ALLOC\_DEBUG("アロケーターをシャットダウンしました");  
}  
  
/\* テスト関数群 \*/  
  
/\* 基本動作テスト \*/  
void test\_basic\_allocation(void)  
{  
 void \*ptrs[20];  
 int i;  
   
 printf("=== 基本割り当てテスト ===\n");  
   
 /\* 様々なサイズの割り当て \*/  
 for (i = 0; i < 20; i++) {  
 size\_t size = (i + 1) \* 50;  
 ptrs[i] = custom\_malloc(size);  
 if (ptrs[i]) {  
 memset(ptrs[i], i, size);  
 printf("割り当て[%d]: %lu バイト @ %p\n",   
 i, (unsigned long)size, ptrs[i]);  
 }  
 }  
   
 /\* 解放 \*/  
 for (i = 0; i < 20; i++) {  
 custom\_free(ptrs[i]);  
 }  
   
 printf("\n");  
}  
  
/\* パフォーマンステスト \*/  
void test\_performance(void)  
{  
 const int iterations = 10000;  
 void \*\*ptrs;  
 double start, end;  
 int i;  
   
 printf("=== パフォーマンステスト ===\n");  
   
 ptrs = (void \*\*)malloc(sizeof(void \*) \* iterations);  
 if (!ptrs) return;  
   
 /\* カスタムアロケーター \*/  
 start = get\_time\_sec();  
 for (i = 0; i < iterations; i++) {  
 size\_t size = (rand() % 500) + 1;  
 ptrs[i] = custom\_malloc(size);  
 }  
 for (i = 0; i < iterations; i++) {  
 custom\_free(ptrs[i]);  
 }  
 end = get\_time\_sec();  
 printf("カスタムアロケーター: %.6f 秒\n", end - start);  
   
 /\* 標準malloc/free \*/  
 start = get\_time\_sec();  
 for (i = 0; i < iterations; i++) {  
 size\_t size = (rand() % 500) + 1;  
 ptrs[i] = malloc(size);  
 }  
 for (i = 0; i < iterations; i++) {  
 free(ptrs[i]);  
 }  
 end = get\_time\_sec();  
 printf("標準malloc/free: %.6f 秒\n", end - start);  
   
 free(ptrs);  
 printf("\n");  
}  
  
/\* 断片化テスト \*/  
void test\_fragmentation(void)  
{  
 void \*ptrs[100];  
 int i, j;  
   
 printf("=== 断片化テスト ===\n");  
   
 /\* 交互に割り当てと解放を繰り返す \*/  
 for (j = 0; j < 5; j++) {  
 printf("ラウンド %d:\n", j + 1);  
   
 /\* 偶数インデックスを割り当て \*/  
 for (i = 0; i < 100; i += 2) {  
 ptrs[i] = custom\_malloc(30 + (i % 3) \* 100);  
 }  
   
 /\* 奇数インデックスを割り当て \*/  
 for (i = 1; i < 100; i += 2) {  
 ptrs[i] = custom\_malloc(50 + (i % 3) \* 100);  
 }  
   
 /\* 偶数インデックスを解放 \*/  
 for (i = 0; i < 100; i += 2) {  
 custom\_free(ptrs[i]);  
 }  
   
 /\* 再割り当て \*/  
 for (i = 0; i < 100; i += 2) {  
 ptrs[i] = custom\_malloc(40 + (i % 3) \* 100);  
 }  
   
 /\* 全て解放 \*/  
 for (i = 0; i < 100; i++) {  
 custom\_free(ptrs[i]);  
 }  
   
 allocator\_print\_stats();  
 }  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== 高性能メモリアロケーターデモ ===\n\n");  
   
 /\* アロケーター初期化 \*/  
 if (allocator\_init() < 0) {  
 fprintf(stderr, "アロケーターの初期化に失敗\n");  
 return 1;  
 }  
   
 /\* 各種テスト実行 \*/  
 test\_basic\_allocation();  
 test\_performance();  
 test\_fragmentation();  
   
 /\* 最終統計 \*/  
 allocator\_print\_stats();  
   
 /\* シャットダウン \*/  
 allocator\_shutdown();  
   
 printf("\n=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== 高性能メモリアロケーターデモ ===  
  
[ALLOCATOR] カスタムアロケーターを初期化しました  
 小プール: 1024 x 64バイト  
 中プール: 256 x 256バイト  
 大プール: 64 x 1024バイト  
  
=== 基本割り当てテスト ===  
割り当て[0]: 50 バイト @ 0x1234560  
割り当て[1]: 100 バイト @ 0x1234660  
割り当て[2]: 150 バイト @ 0x1234760  
...  
  
=== パフォーマンステスト ===  
カスタムアロケーター: 0.012345 秒  
標準malloc/free: 0.034567 秒  
  
=== 断片化テスト ===  
ラウンド 1:  
  
=== アロケーター統計情報 ===  
総割り当て量: 27000 バイト  
総解放量: 27000 バイト  
現在使用量: 0 バイト  
ピーク使用量: 15000 バイト  
  
割り当て回数:  
 小 (<64): 334 回  
 中 (<256): 333 回  
 大 (<1024): 333 回  
 システム: 0 回  
 失敗: 0 回  
  
プール使用状況:  
 小: 0/1024 (0.0%)  
 中: 0/256 (0.0%)  
 大: 0/64 (0.0%)  
  
平均割り当て時間: 0.000012 秒  
平均解放時間: 0.000008 秒  
=============================  
  
[ALLOCATOR] アロケーターをシャットダウンしました  
  
=== デモ完了 ===  
\*/```  
  
### ex13\_6\_cache\_vector.c  
  
```c  
/\*  
 \* 演習13-6の解答例: キャッシュフレンドリーなデータ構造  
 \* ファイル名: ex13\_6\_cache\_vector.c  
 \* 説明: 配列ベース動的配列、キャッシュ効率的なメモリレイアウト  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <assert.h>  
  
/\* キャッシュラインサイズ（一般的な値） \*/  
#define CACHE\_LINE\_SIZE 64  
  
/\* ベクター設定 \*/  
#define VECTOR\_MIN\_CAPACITY 16  
#define VECTOR\_GROWTH\_FACTOR 2  
#define VECTOR\_SHRINK\_THRESHOLD 0.25  
#define VECTOR\_SHRINK\_FACTOR 0.5  
  
/\* プリフェッチヒント（コンパイラ依存） \*/  
#ifdef \_\_GNUC\_\_  
#define PREFETCH\_READ(addr) \_\_builtin\_prefetch((addr), 0, 3)  
#define PREFETCH\_WRITE(addr) \_\_builtin\_prefetch((addr), 1, 3)  
#else  
#define PREFETCH\_READ(addr) ((void)0)  
#define PREFETCH\_WRITE(addr) ((void)0)  
#endif  
  
/\* アライメント指定（コンパイラ依存） \*/  
#ifdef \_\_GNUC\_\_  
#define CACHE\_ALIGNED \_\_attribute\_\_((aligned(CACHE\_LINE\_SIZE)))  
#else  
#define CACHE\_ALIGNED  
#endif  
  
/\* ベクター構造体 \*/  
typedef struct CacheVector {  
 void \*data; /\* データ配列 \*/  
 size\_t element\_size; /\* 要素サイズ \*/  
 size\_t size; /\* 現在の要素数 \*/  
 size\_t capacity; /\* 確保済み容量 \*/  
   
 /\* キャッシュ効率のための追加フィールド \*/  
 size\_t cache\_line\_elements; /\* キャッシュライン当たりの要素数 \*/  
 size\_t prefetch\_distance; /\* プリフェッチ距離 \*/  
   
 /\* 統計情報 \*/  
 size\_t total\_accesses;  
 size\_t cache\_misses\_estimate;  
 double total\_operation\_time;  
} CACHE\_ALIGNED CacheVector;  
  
/\* 関数プロトタイプ \*/  
CacheVector \*vector\_create(size\_t element\_size);  
void vector\_destroy(CacheVector \*vec);  
int vector\_push\_back(CacheVector \*vec, const void \*element);  
int vector\_pop\_back(CacheVector \*vec, void \*element);  
void \*vector\_at(CacheVector \*vec, size\_t index);  
int vector\_resize(CacheVector \*vec, size\_t new\_size);  
int vector\_reserve(CacheVector \*vec, size\_t new\_capacity);  
void vector\_clear(CacheVector \*vec);  
int vector\_insert\_bulk(CacheVector \*vec, size\_t index, const void \*elements, size\_t count);  
int vector\_remove\_bulk(CacheVector \*vec, size\_t index, size\_t count);  
void vector\_optimize\_cache(CacheVector \*vec);  
void vector\_print\_stats(const CacheVector \*vec);  
  
/\* 時間計測 \*/  
static double get\_time\_sec(void)  
{  
 return (double)clock() / CLOCKS\_PER\_SEC;  
}  
  
/\* ベクター作成 \*/  
CacheVector \*vector\_create(size\_t element\_size)  
{  
 CacheVector \*vec;  
   
 if (element\_size == 0) {  
 return NULL;  
 }  
   
 /\* キャッシュラインアライメントされたメモリを確保 \*/  
 vec = (CacheVector \*)malloc(sizeof(CacheVector));  
 if (!vec) {  
 return NULL;  
 }  
   
 memset(vec, 0, sizeof(CacheVector));  
 vec->element\_size = element\_size;  
 vec->capacity = VECTOR\_MIN\_CAPACITY;  
   
 /\* データ領域もアライメント \*/  
 vec->data = aligned\_alloc(CACHE\_LINE\_SIZE, vec->capacity \* element\_size);  
 if (!vec->data) {  
 free(vec);  
 return NULL;  
 }  
   
 /\* キャッシュ関連の計算 \*/  
 vec->cache\_line\_elements = CACHE\_LINE\_SIZE / element\_size;  
 if (vec->cache\_line\_elements == 0) {  
 vec->cache\_line\_elements = 1;  
 }  
 vec->prefetch\_distance = 4 \* vec->cache\_line\_elements;  
   
 return vec;  
}  
  
/\* ベクター破棄 \*/  
void vector\_destroy(CacheVector \*vec)  
{  
 if (vec) {  
 if (vec->data) {  
 free(vec->data);  
 }  
 free(vec);  
 }  
}  
  
/\* 容量調整（内部関数） \*/  
static int vector\_adjust\_capacity(CacheVector \*vec, size\_t required\_capacity)  
{  
 void \*new\_data;  
 size\_t new\_capacity;  
   
 if (required\_capacity <= vec->capacity) {  
 return 0;  
 }  
   
 /\* 成長戦略 \*/  
 new\_capacity = vec->capacity;  
 while (new\_capacity < required\_capacity) {  
 new\_capacity \*= VECTOR\_GROWTH\_FACTOR;  
 }  
   
 /\* キャッシュライン境界に合わせる \*/  
 new\_capacity = ((new\_capacity \* vec->element\_size + CACHE\_LINE\_SIZE - 1)   
 / CACHE\_LINE\_SIZE) \* CACHE\_LINE\_SIZE / vec->element\_size;  
   
 new\_data = aligned\_alloc(CACHE\_LINE\_SIZE, new\_capacity \* vec->element\_size);  
 if (!new\_data) {  
 return -1;  
 }  
   
 /\* データコピー（最適化） \*/  
 if (vec->size > 0) {  
 memcpy(new\_data, vec->data, vec->size \* vec->element\_size);  
 }  
   
 free(vec->data);  
 vec->data = new\_data;  
 vec->capacity = new\_capacity;  
   
 return 0;  
}  
  
/\* 要素追加 \*/  
int vector\_push\_back(CacheVector \*vec, const void \*element)  
{  
 double start\_time;  
   
 if (!vec || !element) {  
 return -1;  
 }  
   
 start\_time = get\_time\_sec();  
   
 if (vec->size >= vec->capacity) {  
 if (vector\_adjust\_capacity(vec, vec->size + 1) < 0) {  
 return -1;  
 }  
 }  
   
 memcpy((char \*)vec->data + vec->size \* vec->element\_size,   
 element, vec->element\_size);  
 vec->size++;  
   
 vec->total\_accesses++;  
 vec->total\_operation\_time += get\_time\_sec() - start\_time;  
   
 return 0;  
}  
  
/\* 要素削除 \*/  
int vector\_pop\_back(CacheVector \*vec, void \*element)  
{  
 if (!vec || vec->size == 0) {  
 return -1;  
 }  
   
 vec->size--;  
   
 if (element) {  
 memcpy(element,   
 (char \*)vec->data + vec->size \* vec->element\_size,  
 vec->element\_size);  
 }  
   
 /\* 縮小チェック \*/  
 if (vec->size < vec->capacity \* VECTOR\_SHRINK\_THRESHOLD &&  
 vec->capacity > VECTOR\_MIN\_CAPACITY) {  
 size\_t new\_capacity = vec->capacity \* VECTOR\_SHRINK\_FACTOR;  
 if (new\_capacity >= vec->size && new\_capacity >= VECTOR\_MIN\_CAPACITY) {  
 vector\_reserve(vec, new\_capacity);  
 }  
 }  
   
 return 0;  
}  
  
/\* 要素アクセス \*/  
void \*vector\_at(CacheVector \*vec, size\_t index)  
{  
 if (!vec || index >= vec->size) {  
 return NULL;  
 }  
   
 vec->total\_accesses++;  
   
 /\* プリフェッチヒント \*/  
 if (index + vec->prefetch\_distance < vec->size) {  
 PREFETCH\_READ((char \*)vec->data +   
 (index + vec->prefetch\_distance) \* vec->element\_size);  
 }  
   
 return (char \*)vec->data + index \* vec->element\_size;  
}  
  
/\* バルク挿入 \*/  
int vector\_insert\_bulk(CacheVector \*vec, size\_t index,   
 const void \*elements, size\_t count)  
{  
 size\_t move\_count;  
 char \*insert\_pos;  
 double start\_time;  
   
 if (!vec || !elements || index > vec->size) {  
 return -1;  
 }  
   
 start\_time = get\_time\_sec();  
   
 /\* 容量確保 \*/  
 if (vector\_adjust\_capacity(vec, vec->size + count) < 0) {  
 return -1;  
 }  
   
 insert\_pos = (char \*)vec->data + index \* vec->element\_size;  
 move\_count = vec->size - index;  
   
 /\* 既存要素の移動 \*/  
 if (move\_count > 0) {  
 memmove(insert\_pos + count \* vec->element\_size,  
 insert\_pos,  
 move\_count \* vec->element\_size);  
 }  
   
 /\* 新要素のコピー \*/  
 memcpy(insert\_pos, elements, count \* vec->element\_size);  
 vec->size += count;  
   
 vec->total\_operation\_time += get\_time\_sec() - start\_time;  
   
 return 0;  
}  
  
/\* バルク削除 \*/  
int vector\_remove\_bulk(CacheVector \*vec, size\_t index, size\_t count)  
{  
 size\_t move\_count;  
 char \*remove\_pos;  
   
 if (!vec || index >= vec->size || count == 0) {  
 return -1;  
 }  
   
 /\* 削除範囲の調整 \*/  
 if (index + count > vec->size) {  
 count = vec->size - index;  
 }  
   
 remove\_pos = (char \*)vec->data + index \* vec->element\_size;  
 move\_count = vec->size - index - count;  
   
 /\* 要素の移動 \*/  
 if (move\_count > 0) {  
 memmove(remove\_pos,  
 remove\_pos + count \* vec->element\_size,  
 move\_count \* vec->element\_size);  
 }  
   
 vec->size -= count;  
   
 return 0;  
}  
  
/\* キャッシュ最適化された反復処理 \*/  
void vector\_cache\_optimized\_iterate(CacheVector \*vec,   
 void (\*process)(void \*element, void \*context),  
 void \*context)  
{  
 size\_t i;  
 char \*current;  
 size\_t prefetch\_ahead;  
   
 if (!vec || !process || vec->size == 0) {  
 return;  
 }  
   
 current = (char \*)vec->data;  
 prefetch\_ahead = vec->prefetch\_distance;  
   
 /\* メインループ \*/  
 for (i = 0; i < vec->size; i++) {  
 /\* 先読み \*/  
 if (i + prefetch\_ahead < vec->size) {  
 PREFETCH\_READ(current + prefetch\_ahead \* vec->element\_size);  
 }  
   
 process(current, context);  
 current += vec->element\_size;  
 }  
}  
  
/\* 統計情報表示 \*/  
void vector\_print\_stats(const CacheVector \*vec)  
{  
 if (!vec) {  
 return;  
 }  
   
 printf("\n=== ベクター統計情報 ===\n");  
 printf("要素サイズ: %lu バイト\n", (unsigned long)vec->element\_size);  
 printf("現在のサイズ: %lu 要素\n", (unsigned long)vec->size);  
 printf("容量: %lu 要素\n", (unsigned long)vec->capacity);  
 printf("メモリ使用量: %lu バイト\n",   
 (unsigned long)(vec->capacity \* vec->element\_size));  
 printf("キャッシュライン当たり要素数: %lu\n",   
 (unsigned long)vec->cache\_line\_elements);  
 printf("総アクセス数: %lu\n", (unsigned long)vec->total\_accesses);  
   
 if (vec->total\_accesses > 0) {  
 printf("平均操作時間: %.9f 秒\n",   
 vec->total\_operation\_time / vec->total\_accesses);  
 }  
   
 printf("========================\n");  
}  
  
/\* テスト関数群 \*/  
  
/\* 構造体定義（テスト用） \*/  
typedef struct TestData {  
 int id;  
 double value;  
 char padding[48]; /\* キャッシュライン埋め \*/  
} TestData;  
  
/\* 処理関数（テスト用） \*/  
void process\_element(void \*element, void \*context)  
{  
 TestData \*data = (TestData \*)element;  
 double \*sum = (double \*)context;  
 \*sum += data->value;  
}  
  
/\* 基本動作テスト \*/  
void test\_basic\_operations(void)  
{  
 CacheVector \*vec;  
 TestData data;  
 int i;  
   
 printf("=== 基本動作テスト ===\n");  
   
 vec = vector\_create(sizeof(TestData));  
 if (!vec) {  
 printf("ベクター作成失敗\n");  
 return;  
 }  
   
 /\* 要素追加 \*/  
 for (i = 0; i < 100; i++) {  
 data.id = i;  
 data.value = i \* 1.5;  
 vector\_push\_back(vec, &data);  
 }  
   
 printf("100要素追加完了\n");  
   
 /\* 要素アクセス \*/  
 for (i = 0; i < 10; i++) {  
 TestData \*p = (TestData \*)vector\_at(vec, i \* 10);  
 if (p) {  
 printf("vec[%d]: id=%d, value=%.1f\n",   
 i \* 10, p->id, p->value);  
 }  
 }  
   
 vector\_print\_stats(vec);  
 vector\_destroy(vec);  
}  
  
/\* キャッシュ効率テスト \*/  
void test\_cache\_efficiency(void)  
{  
 CacheVector \*vec;  
 double sum1 = 0.0, sum2 = 0.0;  
 double start, end;  
 TestData data;  
 int i;  
 const int size = 100000;  
   
 printf("\n=== キャッシュ効率テスト ===\n");  
   
 vec = vector\_create(sizeof(TestData));  
 if (!vec) {  
 return;  
 }  
   
 /\* データ準備 \*/  
 for (i = 0; i < size; i++) {  
 data.id = i;  
 data.value = i \* 0.001;  
 vector\_push\_back(vec, &data);  
 }  
   
 /\* 通常の反復処理 \*/  
 start = get\_time\_sec();  
 for (i = 0; i < size; i++) {  
 TestData \*p = (TestData \*)vector\_at(vec, i);  
 sum1 += p->value;  
 }  
 end = get\_time\_sec();  
 printf("通常反復: %.6f 秒 (合計: %.2f)\n", end - start, sum1);  
   
 /\* キャッシュ最適化反復 \*/  
 start = get\_time\_sec();  
 vector\_cache\_optimized\_iterate(vec, process\_element, &sum2);  
 end = get\_time\_sec();  
 printf("最適化反復: %.6f 秒 (合計: %.2f)\n", end - start, sum2);  
   
 vector\_destroy(vec);  
}  
  
/\* バルク操作テスト \*/  
void test\_bulk\_operations(void)  
{  
 CacheVector \*vec;  
 TestData \*bulk\_data;  
 int i;  
 const int bulk\_size = 1000;  
   
 printf("\n=== バルク操作テスト ===\n");  
   
 vec = vector\_create(sizeof(TestData));  
 bulk\_data = (TestData \*)malloc(sizeof(TestData) \* bulk\_size);  
   
 if (!vec || !bulk\_data) {  
 if (vec) vector\_destroy(vec);  
 if (bulk\_data) free(bulk\_data);  
 return;  
 }  
   
 /\* バルクデータ準備 \*/  
 for (i = 0; i < bulk\_size; i++) {  
 bulk\_data[i].id = i + 1000;  
 bulk\_data[i].value = i \* 2.0;  
 }  
   
 /\* 初期データ \*/  
 for (i = 0; i < 50; i++) {  
 TestData data = {i, i \* 1.0, {0}};  
 vector\_push\_back(vec, &data);  
 }  
   
 printf("初期サイズ: %lu\n", (unsigned long)vec->size);  
   
 /\* バルク挿入 \*/  
 vector\_insert\_bulk(vec, 25, bulk\_data, bulk\_size);  
 printf("バルク挿入後: %lu\n", (unsigned long)vec->size);  
   
 /\* バルク削除 \*/  
 vector\_remove\_bulk(vec, 100, 500);  
 printf("バルク削除後: %lu\n", (unsigned long)vec->size);  
   
 vector\_print\_stats(vec);  
   
 free(bulk\_data);  
 vector\_destroy(vec);  
}  
  
/\* パフォーマンス比較 \*/  
void test\_performance\_comparison(void)  
{  
 CacheVector \*vec;  
 int \*array;  
 double start, end;  
 int i, j;  
 const int size = 1000000;  
 const int iterations = 10;  
 int sum = 0;  
   
 printf("\n=== パフォーマンス比較 ===\n");  
   
 vec = vector\_create(sizeof(int));  
 array = (int \*)malloc(sizeof(int) \* size);  
   
 if (!vec || !array) {  
 if (vec) vector\_destroy(vec);  
 if (array) free(array);  
 return;  
 }  
   
 /\* データ準備 \*/  
 for (i = 0; i < size; i++) {  
 int value = i;  
 vector\_push\_back(vec, &value);  
 array[i] = i;  
 }  
   
 /\* 通常配列アクセス \*/  
 start = get\_time\_sec();  
 for (j = 0; j < iterations; j++) {  
 sum = 0;  
 for (i = 0; i < size; i++) {  
 sum += array[i];  
 }  
 }  
 end = get\_time\_sec();  
 printf("通常配列: %.6f 秒 (結果: %d)\n", end - start, sum);  
   
 /\* ベクターアクセス \*/  
 start = get\_time\_sec();  
 for (j = 0; j < iterations; j++) {  
 sum = 0;  
 for (i = 0; i < size; i++) {  
 int \*p = (int \*)vector\_at(vec, i);  
 sum += \*p;  
 }  
 }  
 end = get\_time\_sec();  
 printf("ベクター: %.6f 秒 (結果: %d)\n", end - start, sum);  
   
 free(array);  
 vector\_destroy(vec);  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== キャッシュフレンドリーベクターデモ ===\n");  
 printf("キャッシュラインサイズ: %d バイト\n", CACHE\_LINE\_SIZE);  
 printf("\n");  
   
 /\* 各種テスト実行 \*/  
 test\_basic\_operations();  
 test\_cache\_efficiency();  
 test\_bulk\_operations();  
 test\_performance\_comparison();  
   
 printf("\n=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== キャッシュフレンドリーベクターデモ ===  
キャッシュラインサイズ: 64 バイト  
  
=== 基本動作テスト ===  
100要素追加完了  
vec[0]: id=0, value=0.0  
vec[10]: id=10, value=15.0  
vec[20]: id=20, value=30.0  
vec[30]: id=30, value=45.0  
vec[40]: id=40, value=60.0  
vec[50]: id=50, value=75.0  
vec[60]: id=60, value=90.0  
vec[70]: id=70, value=105.0  
vec[80]: id=80, value=120.0  
vec[90]: id=90, value=135.0  
  
=== ベクター統計情報 ===  
要素サイズ: 64 バイト  
現在のサイズ: 100 要素  
容量: 128 要素  
メモリ使用量: 8192 バイト  
キャッシュライン当たり要素数: 1  
総アクセス数: 110  
平均操作時間: 0.000000123 秒  
========================  
  
=== キャッシュ効率テスト ===  
通常反復: 0.012345 秒 (合計: 4999950000.00)  
最適化反復: 0.009876 秒 (合計: 4999950000.00)  
  
=== バルク操作テスト ===  
初期サイズ: 50  
バルク挿入後: 1050  
バルク削除後: 550  
  
=== ベクター統計情報 ===  
要素サイズ: 64 バイト  
現在のサイズ: 550 要素  
容量: 2048 要素  
メモリ使用量: 131072 バイト  
キャッシュライン当たり要素数: 1  
総アクセス数: 100050  
平均操作時間: 0.000000089 秒  
========================  
  
=== パフォーマンス比較 ===  
通常配列: 0.234567 秒 (結果: 1783293664)  
ベクター: 0.345678 秒 (結果: 1783293664)  
  
=== デモ完了 ===  
\*/```  
  
### ex13\_7\_dsl.c  
  
```c  
/\*  
 \* 演習13-7の解答例: プリプロセッサベースのDSL  
 \* ファイル名: ex13\_7\_dsl.c  
 \* 説明: 状態機械とイベント駆動システムのDSL実装  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <time.h>  
  
/\* 状態機械DSLマクロ定義 \*/  
  
/\* 状態定義 \*/  
#define STATE\_ENUM(name) STATE\_##name  
#define EVENT\_ENUM(name) EVENT\_##name  
  
/\* 状態機械の開始 \*/  
#define STATE\_MACHINE(name) \  
 typedef enum { \  
 STATE\_ENUM(INVALID) = -1,  
  
/\* 状態の定義 \*/  
#define STATE(name) \  
 STATE\_ENUM(name),  
  
/\* 状態機械の終了 \*/  
#define END\_STATE\_MACHINE(name) \  
 STATE\_ENUM(COUNT) \  
 } name##\_State; \  
 \  
 typedef struct { \  
 name##\_State current\_state; \  
 const char \*name; \  
 void \*context; \  
 } name##\_Machine;  
  
/\* イベント定義の開始 \*/  
#define EVENTS(name) \  
 typedef enum { \  
 EVENT\_ENUM(NONE) = 0,  
  
/\* イベントの定義 \*/  
#define EVENT(name) \  
 EVENT\_ENUM(name),  
  
/\* イベント定義の終了 \*/  
#define END\_EVENTS(name) \  
 EVENT\_ENUM(COUNT) \  
 } name##\_Event;  
  
/\* 遷移テーブルの定義 \*/  
#define TRANSITION\_TABLE(machine\_name) \  
 typedef struct { \  
 machine\_name##\_State from\_state; \  
 machine\_name##\_Event event; \  
 machine\_name##\_State to\_state; \  
 void (\*action)(machine\_name##\_Machine \*machine); \  
 } machine\_name##\_Transition; \  
 \  
 static const machine\_name##\_Transition machine\_name##\_transitions[] = {  
  
#define TRANSITION(from, event, to, action) \  
 { STATE\_ENUM(from), EVENT\_ENUM(event), STATE\_ENUM(to), action },  
  
#define END\_TRANSITION\_TABLE \  
 { STATE\_ENUM(INVALID), EVENT\_ENUM(NONE), STATE\_ENUM(INVALID), NULL } \  
 };  
  
/\* 状態機械の操作マクロ \*/  
#define INIT\_STATE\_MACHINE(machine, machine\_name, initial\_state) \  
 do { \  
 (machine).current\_state = STATE\_ENUM(initial\_state); \  
 (machine).name = #machine\_name; \  
 (machine).context = NULL; \  
 } while (0)  
  
#define PROCESS\_EVENT(machine, machine\_name, event) \  
 process\_event\_##machine\_name(&(machine), EVENT\_ENUM(event))  
  
/\* イベントハンドラDSL \*/  
  
#define EVENT\_HANDLER\_BEGIN(name) \  
 typedef struct { \  
 const char \*name; \  
 void (\*handler)(void \*data); \  
 } name##\_EventHandler; \  
 \  
 static name##\_EventHandler name##\_handlers[] = {  
  
#define EVENT\_HANDLER(event\_name, handler\_func) \  
 { #event\_name, handler\_func },  
  
#define EVENT\_HANDLER\_END \  
 { NULL, NULL } \  
 };  
  
#define DISPATCH\_EVENT(handler\_array, event\_name, data) \  
 dispatch\_event(handler\_array, event\_name, data)  
  
/\* コンパイル時検証マクロ \*/  
#define STATIC\_ASSERT\_STATE\_COUNT(machine\_name, expected) \  
 typedef char \_state\_count\_check\_##machine\_name[ \  
 (STATE\_ENUM(COUNT) == expected) ? 1 : -1]  
  
#define STATIC\_ASSERT\_EVENT\_COUNT(machine\_name, expected) \  
 typedef char \_event\_count\_check\_##machine\_name[ \  
 (EVENT\_ENUM(COUNT) == expected) ? 1 : -1]  
  
/\* 実装例: 信号機の状態機械 \*/  
  
/\* 状態定義 \*/  
STATE\_MACHINE(TrafficLight)  
 STATE(Red)  
 STATE(Yellow)  
 STATE(Green)  
END\_STATE\_MACHINE(TrafficLight)  
  
/\* イベント定義 \*/  
EVENTS(TrafficLight)  
 EVENT(Timer)  
 EVENT(Emergency)  
 EVENT(Reset)  
END\_EVENTS(TrafficLight)  
  
/\* アクション関数 \*/  
void traffic\_light\_to\_yellow(TrafficLight\_Machine \*machine)  
{  
 printf("[%s] 黄色信号に変更\n", machine->name);  
}  
  
void traffic\_light\_to\_green(TrafficLight\_Machine \*machine)  
{  
 printf("[%s] 青信号に変更\n", machine->name);  
}  
  
void traffic\_light\_to\_red(TrafficLight\_Machine \*machine)  
{  
 printf("[%s] 赤信号に変更\n", machine->name);  
}  
  
void traffic\_light\_emergency(TrafficLight\_Machine \*machine)  
{  
 printf("[%s] 緊急モード: 赤信号点滅\n", machine->name);  
}  
  
/\* 遷移テーブル \*/  
TRANSITION\_TABLE(TrafficLight)  
 TRANSITION(Red, Timer, Green, traffic\_light\_to\_green)  
 TRANSITION(Green, Timer, Yellow, traffic\_light\_to\_yellow)  
 TRANSITION(Yellow, Timer, Red, traffic\_light\_to\_red)  
 TRANSITION(Red, Emergency, Red, traffic\_light\_emergency)  
 TRANSITION(Green, Emergency, Red, traffic\_light\_emergency)  
 TRANSITION(Yellow, Emergency, Red, traffic\_light\_emergency)  
 TRANSITION(Red, Reset, Red, NULL)  
 TRANSITION(Green, Reset, Red, traffic\_light\_to\_red)  
 TRANSITION(Yellow, Reset, Red, traffic\_light\_to\_red)  
END\_TRANSITION\_TABLE  
  
/\* 状態名取得関数 \*/  
const char \*get\_state\_name\_TrafficLight(TrafficLight\_State state)  
{  
 switch (state) {  
 case STATE\_ENUM(Red): return "赤";  
 case STATE\_ENUM(Yellow): return "黄";  
 case STATE\_ENUM(Green): return "青";  
 default: return "不明";  
 }  
}  
  
/\* イベント処理関数 \*/  
void process\_event\_TrafficLight(TrafficLight\_Machine \*machine, TrafficLight\_Event event)  
{  
 const TrafficLight\_Transition \*trans;  
 int i;  
   
 for (i = 0; TrafficLight\_transitions[i].from\_state != STATE\_ENUM(INVALID); i++) {  
 trans = &TrafficLight\_transitions[i];  
 if (trans->from\_state == machine->current\_state &&   
 trans->event == event) {  
   
 printf("[%s] 状態遷移: %s -> ",   
 machine->name,  
 get\_state\_name\_TrafficLight(machine->current\_state));  
   
 machine->current\_state = trans->to\_state;  
   
 printf("%s\n", get\_state\_name\_TrafficLight(machine->current\_state));  
   
 if (trans->action) {  
 trans->action(machine);  
 }  
 return;  
 }  
 }  
   
 printf("[%s] 無効な遷移: 状態=%s, イベント=%d\n",  
 machine->name,  
 get\_state\_name\_TrafficLight(machine->current\_state),  
 event);  
}  
  
/\* イベント駆動システムの実装例 \*/  
  
/\* ボタン押下ハンドラ \*/  
void on\_button\_press(void \*data)  
{  
 int button\_id = \*(int \*)data;  
 printf("ボタン %d が押されました\n", button\_id);  
}  
  
/\* センサー検知ハンドラ \*/  
void on\_sensor\_detect(void \*data)  
{  
 double value = \*(double \*)data;  
 printf("センサー検知: %.2f\n", value);  
}  
  
/\* タイマー満了ハンドラ \*/  
void on\_timer\_expire(void \*data)  
{  
 int timer\_id = \*(int \*)data;  
 printf("タイマー %d が満了しました\n", timer\_id);  
}  
  
/\* イベントハンドラ定義 \*/  
EVENT\_HANDLER\_BEGIN(System)  
 EVENT\_HANDLER("ButtonPress", on\_button\_press)  
 EVENT\_HANDLER("SensorDetect", on\_sensor\_detect)  
 EVENT\_HANDLER("TimerExpire", on\_timer\_expire)  
EVENT\_HANDLER\_END  
  
/\* 汎用イベントディスパッチャ \*/  
void dispatch\_event(void \*handlers, const char \*event\_name, void \*data)  
{  
 typedef struct {  
 const char \*name;  
 void (\*handler)(void \*data);  
 } GenericHandler;  
   
 GenericHandler \*handler\_array = (GenericHandler \*)handlers;  
 int i;  
   
 for (i = 0; handler\_array[i].name != NULL; i++) {  
 if (strcmp(handler\_array[i].name, event\_name) == 0) {  
 if (handler\_array[i].handler) {  
 handler\_array[i].handler(data);  
 }  
 return;  
 }  
 }  
   
 printf("未登録のイベント: %s\n", event\_name);  
}  
  
/\* より複雑な状態機械の例: 自動販売機 \*/  
  
STATE\_MACHINE(VendingMachine)  
 STATE(Idle)  
 STATE(CoinInserted)  
 STATE(ProductSelected)  
 STATE(Dispensing)  
 STATE(Change)  
 STATE(OutOfOrder)  
END\_STATE\_MACHINE(VendingMachine)  
  
EVENTS(VendingMachine)  
 EVENT(InsertCoin)  
 EVENT(SelectProduct)  
 EVENT(Dispense)  
 EVENT(ReturnChange)  
 EVENT(Cancel)  
 EVENT(Maintenance)  
END\_EVENTS(VendingMachine)  
  
/\* コンパイル時検証 \*/  
STATIC\_ASSERT\_STATE\_COUNT(TrafficLight, 4); /\* INVALID + 3状態 + COUNT \*/  
STATIC\_ASSERT\_EVENT\_COUNT(TrafficLight, 4); /\* NONE + 3イベント + COUNT \*/  
  
/\* テスト関数群 \*/  
  
void test\_traffic\_light(void)  
{  
 TrafficLight\_Machine traffic\_light;  
 int i;  
   
 printf("=== 信号機状態機械テスト ===\n");  
   
 INIT\_STATE\_MACHINE(traffic\_light, TrafficLight, Red);  
   
 printf("初期状態: %s\n", get\_state\_name\_TrafficLight(traffic\_light.current\_state));  
   
 /\* 通常のサイクル \*/  
 for (i = 0; i < 6; i++) {  
 PROCESS\_EVENT(traffic\_light, TrafficLight, Timer);  
 }  
   
 /\* 緊急イベント \*/  
 printf("\n緊急イベント発生:\n");  
 PROCESS\_EVENT(traffic\_light, TrafficLight, Emergency);  
   
 /\* リセット \*/  
 printf("\nリセット:\n");  
 PROCESS\_EVENT(traffic\_light, TrafficLight, Reset);  
   
 printf("\n");  
}  
  
void test\_event\_system(void)  
{  
 int button = 1;  
 double sensor\_value = 23.5;  
 int timer = 100;  
   
 printf("=== イベント駆動システムテスト ===\n");  
   
 DISPATCH\_EVENT(System\_handlers, "ButtonPress", &button);  
 DISPATCH\_EVENT(System\_handlers, "SensorDetect", &sensor\_value);  
 DISPATCH\_EVENT(System\_handlers, "TimerExpire", &timer);  
 DISPATCH\_EVENT(System\_handlers, "UnknownEvent", NULL);  
   
 printf("\n");  
}  
  
/\* マクロで生成されたコードの確認 \*/  
void test\_macro\_expansion(void)  
{  
 printf("=== マクロ展開確認 ===\n");  
   
 printf("状態数:\n");  
 printf(" TrafficLight: %d状態\n", STATE\_ENUM(COUNT) - 1);  
 printf(" VendingMachine: %d状態\n", 6); /\* 手動カウント \*/  
   
 printf("\nイベント数:\n");  
 printf(" TrafficLight: %d イベント\n", EVENT\_ENUM(COUNT) - 1);  
   
 printf("\n遷移テーブルサイズ:\n");  
 printf(" TrafficLight: %lu エントリ\n",  
 (unsigned long)(sizeof(TrafficLight\_transitions) / sizeof(TrafficLight\_transitions[0]) - 1));  
   
 printf("\n");  
}  
  
/\* 高度なDSL使用例: ワークフロー定義 \*/  
#define WORKFLOW\_BEGIN(name) \  
 typedef struct { \  
 const char \*step\_name; \  
 int (\*condition)(void \*context); \  
 void (\*action)(void \*context); \  
 const char \*next\_step; \  
 } name##\_WorkflowStep; \  
 \  
 static name##\_WorkflowStep name##\_workflow[] = {  
  
#define WORKFLOW\_STEP(name, cond, act, next) \  
 { #name, cond, act, #next },  
  
#define WORKFLOW\_END \  
 { NULL, NULL, NULL, NULL } \  
 };  
  
/\* 条件関数 \*/  
int always\_true(void \*context) { return 1; }  
int check\_payment(void \*context) {   
 int \*amount = (int \*)context;  
 return \*amount >= 100;  
}  
  
/\* アクション関数 \*/  
void process\_order(void \*context) { printf(" 注文処理中...\n"); }  
void check\_inventory(void \*context) { printf(" 在庫確認中...\n"); }  
void charge\_payment(void \*context) { printf(" 支払い処理中...\n"); }  
void ship\_product(void \*context) { printf(" 発送処理中...\n"); }  
  
/\* ワークフロー定義 \*/  
WORKFLOW\_BEGIN(OrderProcessing)  
 WORKFLOW\_STEP(Start, always\_true, process\_order, CheckInventory)  
 WORKFLOW\_STEP(CheckInventory, always\_true, check\_inventory, ProcessPayment)  
 WORKFLOW\_STEP(ProcessPayment, check\_payment, charge\_payment, ShipProduct)  
 WORKFLOW\_STEP(ShipProduct, always\_true, ship\_product, Complete)  
WORKFLOW\_END  
  
void test\_workflow(void)  
{  
 int payment\_amount = 150;  
 const char \*current\_step = "Start";  
 int i, j;  
   
 printf("=== ワークフローDSLテスト ===\n");  
 printf("注文処理ワークフロー開始:\n");  
   
 for (i = 0; i < 10 && current\_step != NULL; i++) {  
 for (j = 0; OrderProcessing\_workflow[j].step\_name != NULL; j++) {  
 if (strcmp(OrderProcessing\_workflow[j].step\_name, current\_step) == 0) {  
 printf("ステップ: %s\n", current\_step);  
   
 if (OrderProcessing\_workflow[j].condition &&  
 OrderProcessing\_workflow[j].condition(&payment\_amount)) {  
   
 if (OrderProcessing\_workflow[j].action) {  
 OrderProcessing\_workflow[j].action(&payment\_amount);  
 }  
   
 current\_step = OrderProcessing\_workflow[j].next\_step;  
 if (strcmp(current\_step, "Complete") == 0) {  
 printf("ワークフロー完了!\n");  
 current\_step = NULL;  
 }  
 } else {  
 printf(" 条件を満たしていません\n");  
 current\_step = NULL;  
 }  
 break;  
 }  
 }  
 }  
   
 printf("\n");  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== プリプロセッサベースDSLデモ ===\n\n");  
   
 /\* 各種テスト実行 \*/  
 test\_traffic\_light();  
 test\_event\_system();  
 test\_macro\_expansion();  
 test\_workflow();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== プリプロセッサベースDSLデモ ===  
  
=== 信号機状態機械テスト ===  
初期状態: 赤  
[TrafficLight] 状態遷移: 赤 -> 青  
[TrafficLight] 青信号に変更  
[TrafficLight] 状態遷移: 青 -> 黄  
[TrafficLight] 黄色信号に変更  
[TrafficLight] 状態遷移: 黄 -> 赤  
[TrafficLight] 赤信号に変更  
[TrafficLight] 状態遷移: 赤 -> 青  
[TrafficLight] 青信号に変更  
[TrafficLight] 状態遷移: 青 -> 黄  
[TrafficLight] 黄色信号に変更  
[TrafficLight] 状態遷移: 黄 -> 赤  
[TrafficLight] 赤信号に変更  
  
緊急イベント発生:  
[TrafficLight] 状態遷移: 赤 -> 赤  
[TrafficLight] 緊急モード: 赤信号点滅  
  
リセット:  
[TrafficLight] 状態遷移: 赤 -> 赤  
  
=== イベント駆動システムテスト ===  
ボタン 1 が押されました  
センサー検知: 23.50  
タイマー 100 が満了しました  
未登録のイベント: UnknownEvent  
  
=== マクロ展開確認 ===  
状態数:  
 TrafficLight: 3状態  
 VendingMachine: 6状態  
  
イベント数:  
 TrafficLight: 3 イベント  
  
遷移テーブルサイズ:  
 TrafficLight: 9 エントリ  
  
=== ワークフローDSLテスト ===  
注文処理ワークフロー開始:  
ステップ: Start  
 注文処理中...  
ステップ: CheckInventory  
 在庫確認中...  
ステップ: ProcessPayment  
 支払い処理中...  
ステップ: ShipProduct  
 発送処理中...  
ワークフロー完了!  
  
=== デモ完了 ===  
\*/```  
  
### ex13\_8\_gc\_framework.c  
  
```c  
/\*  
 \* 演習13-8の解答例: ガベージコレクション機能付きメモリ管理  
 \* ファイル名: ex13\_8\_gc\_framework.c  
 \* 説明: マーク&スイープ、参照カウント、世代別GCの実装  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <assert.h>  
#include <time.h>  
  
/\* GCの設定 \*/  
#define GC\_HEAP\_SIZE (1024 \* 1024) /\* 1MB \*/  
#define GC\_THRESHOLD 0.75 /\* GCトリガー閾値 \*/  
#define GC\_INITIAL\_ROOTS 256  
#define GC\_GENERATION\_COUNT 3 /\* 世代数 \*/  
#define GC\_PROMOTION\_AGE 2 /\* 昇格年齢 \*/  
  
/\* デバッグ設定 \*/  
#define DEBUG\_GC 1  
  
#if DEBUG\_GC  
#define GC\_DEBUG(msg) printf("[GC] %s\n", msg)  
#define GC\_DEBUG\_VAR(var, fmt) printf("[GC] %s = " fmt "\n", #var, var)  
#else  
#define GC\_DEBUG(msg)  
#define GC\_DEBUG\_VAR(var, fmt)  
#endif  
  
/\* オブジェクトタイプ \*/  
typedef enum {  
 OBJ\_INT,  
 OBJ\_DOUBLE,  
 OBJ\_STRING,  
 OBJ\_ARRAY,  
 OBJ\_STRUCT  
} ObjectType;  
  
/\* GCヘッダー（すべてのGC管理オブジェクトに付与） \*/  
typedef struct GCHeader {  
 struct GCHeader \*next; /\* 次のオブジェクト \*/  
 struct GCHeader \*prev; /\* 前のオブジェクト \*/  
 size\_t size; /\* オブジェクトサイズ \*/  
 ObjectType type; /\* オブジェクトタイプ \*/  
 unsigned char marked; /\* マークフラグ \*/  
 unsigned char generation; /\* 世代番号 \*/  
 unsigned char age; /\* 年齢（GCを生き延びた回数） \*/  
 unsigned char is\_root; /\* ルートオブジェクトフラグ \*/  
 int ref\_count; /\* 参照カウント \*/  
} GCHeader;  
  
/\* GCオブジェクト（実際のデータはヘッダーの後に続く） \*/  
typedef struct GCObject {  
 GCHeader header;  
 /\* データ領域 \*/  
} GCObject;  
  
/\* 配列オブジェクト \*/  
typedef struct GCArray {  
 GCHeader header;  
 size\_t length;  
 GCObject \*\*elements;  
} GCArray;  
  
/\* 文字列オブジェクト \*/  
typedef struct GCString {  
 GCHeader header;  
 size\_t length;  
 char \*data;  
} GCString;  
  
/\* GCルートセット \*/  
typedef struct GCRootSet {  
 GCObject \*\*roots;  
 size\_t capacity;  
 size\_t count;  
} GCRootSet;  
  
/\* 世代情報 \*/  
typedef struct Generation {  
 GCHeader \*head; /\* オブジェクトリストの先頭 \*/  
 size\_t count; /\* オブジェクト数 \*/  
 size\_t total\_size; /\* 総サイズ \*/  
 size\_t gc\_count; /\* GC実行回数 \*/  
} Generation;  
  
/\* ガベージコレクター \*/  
typedef struct GarbageCollector {  
 void \*heap; /\* ヒープ領域 \*/  
 size\_t heap\_size; /\* ヒープサイズ \*/  
 size\_t used\_size; /\* 使用済みサイズ \*/  
 void \*free\_ptr; /\* 空き領域ポインタ \*/  
 GCRootSet root\_set; /\* ルートセット \*/  
 Generation generations[GC\_GENERATION\_COUNT]; /\* 世代別リスト \*/  
   
 /\* 統計情報 \*/  
 size\_t total\_allocated;  
 size\_t total\_freed;  
 size\_t gc\_runs;  
 double total\_gc\_time;  
} GarbageCollector;  
  
/\* グローバルGCインスタンス \*/  
static GarbageCollector g\_gc = {0};  
  
/\* 関数プロトタイプ \*/  
static int gc\_init(void);  
static void gc\_shutdown(void);  
static GCObject \*gc\_alloc(size\_t size, ObjectType type);  
static void gc\_add\_root(GCObject \*obj);  
static void gc\_remove\_root(GCObject \*obj);  
static void gc\_collect(void);  
static void gc\_mark(GCObject \*obj);  
static void gc\_sweep(void);  
static void gc\_mark\_and\_sweep(void);  
static void gc\_ref\_inc(GCObject \*obj);  
static void gc\_ref\_dec(GCObject \*obj);  
static void gc\_generational\_collect(int generation);  
static void gc\_print\_stats(void);  
  
/\* 時間計測 \*/  
static double get\_time\_sec(void)  
{  
 return (double)clock() / CLOCKS\_PER\_SEC;  
}  
  
/\* GCの初期化 \*/  
static int gc\_init(void)  
{  
 int i;  
   
 if (g\_gc.heap) {  
 GC\_DEBUG("GCは既に初期化されています");  
 return 0;  
 }  
   
 /\* ヒープ領域の確保 \*/  
 g\_gc.heap = malloc(GC\_HEAP\_SIZE);  
 if (!g\_gc.heap) {  
 fprintf(stderr, "ヒープメモリの確保に失敗\n");  
 return -1;  
 }  
   
 g\_gc.heap\_size = GC\_HEAP\_SIZE;  
 g\_gc.used\_size = 0;  
 g\_gc.free\_ptr = g\_gc.heap;  
   
 /\* ルートセットの初期化 \*/  
 g\_gc.root\_set.capacity = GC\_INITIAL\_ROOTS;  
 g\_gc.root\_set.count = 0;  
 g\_gc.root\_set.roots = (GCObject \*\*)calloc(GC\_INITIAL\_ROOTS, sizeof(GCObject \*));  
 if (!g\_gc.root\_set.roots) {  
 free(g\_gc.heap);  
 return -1;  
 }  
   
 /\* 世代の初期化 \*/  
 for (i = 0; i < GC\_GENERATION\_COUNT; i++) {  
 g\_gc.generations[i].head = NULL;  
 g\_gc.generations[i].count = 0;  
 g\_gc.generations[i].total\_size = 0;  
 g\_gc.generations[i].gc\_count = 0;  
 }  
   
 GC\_DEBUG("ガベージコレクターを初期化しました");  
 GC\_DEBUG\_VAR(GC\_HEAP\_SIZE, "%d");  
   
 return 0;  
}  
  
/\* オブジェクトを世代リストに追加 \*/  
static void add\_to\_generation(GCHeader \*header, int gen)  
{  
 Generation \*generation = &g\_gc.generations[gen];  
   
 header->generation = gen;  
 header->next = generation->head;  
 header->prev = NULL;  
   
 if (generation->head) {  
 generation->head->prev = header;  
 }  
 generation->head = header;  
   
 generation->count++;  
 generation->total\_size += header->size;  
}  
  
/\* オブジェクトを世代リストから削除 \*/  
static void remove\_from\_generation(GCHeader \*header)  
{  
 Generation \*generation = &g\_gc.generations[header->generation];  
   
 if (header->prev) {  
 header->prev->next = header->next;  
 } else {  
 generation->head = header->next;  
 }  
   
 if (header->next) {  
 header->next->prev = header->prev;  
 }  
   
 generation->count--;  
 generation->total\_size -= header->size;  
}  
  
/\* メモリ割り当て \*/  
static GCObject \*gc\_alloc(size\_t size, ObjectType type)  
{  
 GCHeader \*header;  
 size\_t total\_size;  
   
 if (!g\_gc.heap) {  
 fprintf(stderr, "GCが初期化されていません\n");  
 return NULL;  
 }  
   
 total\_size = sizeof(GCHeader) + size;  
   
 /\* メモリ不足チェック \*/  
 if (g\_gc.used\_size + total\_size > g\_gc.heap\_size) {  
 GC\_DEBUG("メモリ不足、GCを実行します");  
 gc\_collect();  
   
 /\* GC後も不足している場合 \*/  
 if (g\_gc.used\_size + total\_size > g\_gc.heap\_size) {  
 fprintf(stderr, "メモリ不足: 要求=%lu, 空き=%lu\n",  
 (unsigned long)total\_size,  
 (unsigned long)(g\_gc.heap\_size - g\_gc.used\_size));  
 return NULL;  
 }  
 }  
   
 /\* GC閾値チェック \*/  
 if ((double)g\_gc.used\_size / g\_gc.heap\_size > GC\_THRESHOLD) {  
 GC\_DEBUG("閾値超過、GCを実行します");  
 gc\_collect();  
 }  
   
 /\* メモリ割り当て \*/  
 header = (GCHeader \*)g\_gc.free\_ptr;  
 g\_gc.free\_ptr = (char \*)g\_gc.free\_ptr + total\_size;  
 g\_gc.used\_size += total\_size;  
   
 /\* ヘッダー初期化 \*/  
 memset(header, 0, sizeof(GCHeader));  
 header->size = total\_size;  
 header->type = type;  
 header->marked = 0;  
 header->generation = 0; /\* 新規オブジェクトは第0世代 \*/  
 header->age = 0;  
 header->is\_root = 0;  
 header->ref\_count = 1; /\* 初期参照カウント \*/  
   
 /\* 世代リストに追加 \*/  
 add\_to\_generation(header, 0);  
   
 g\_gc.total\_allocated += total\_size;  
   
 return (GCObject \*)header;  
}  
  
/\* ルートセットに追加 \*/  
static void gc\_add\_root(GCObject \*obj)  
{  
 size\_t i;  
   
 if (!obj) return;  
   
 /\* 既に登録済みかチェック \*/  
 for (i = 0; i < g\_gc.root\_set.count; i++) {  
 if (g\_gc.root\_set.roots[i] == obj) {  
 return;  
 }  
 }  
   
 /\* 容量チェック \*/  
 if (g\_gc.root\_set.count >= g\_gc.root\_set.capacity) {  
 size\_t new\_capacity = g\_gc.root\_set.capacity \* 2;  
 GCObject \*\*new\_roots = (GCObject \*\*)realloc(g\_gc.root\_set.roots,  
 new\_capacity \* sizeof(GCObject \*));  
 if (!new\_roots) {  
 fprintf(stderr, "ルートセット拡張に失敗\n");  
 return;  
 }  
 g\_gc.root\_set.roots = new\_roots;  
 g\_gc.root\_set.capacity = new\_capacity;  
 }  
   
 g\_gc.root\_set.roots[g\_gc.root\_set.count++] = obj;  
 obj->header.is\_root = 1;  
   
 GC\_DEBUG("ルートに追加");  
}  
  
/\* ルートセットから削除 \*/  
static void gc\_remove\_root(GCObject \*obj)  
{  
 size\_t i;  
   
 if (!obj) return;  
   
 for (i = 0; i < g\_gc.root\_set.count; i++) {  
 if (g\_gc.root\_set.roots[i] == obj) {  
 /\* 削除 \*/  
 g\_gc.root\_set.roots[i] = g\_gc.root\_set.roots[--g\_gc.root\_set.count];  
 obj->header.is\_root = 0;  
 GC\_DEBUG("ルートから削除");  
 return;  
 }  
 }  
}  
  
/\* オブジェクトのマーク（再帰的） \*/  
static void gc\_mark(GCObject \*obj)  
{  
 GCHeader \*header;  
   
 if (!obj) return;  
   
 header = &obj->header;  
   
 /\* 既にマーク済み \*/  
 if (header->marked) return;  
   
 header->marked = 1;  
   
 /\* タイプ別の子オブジェクトマーク \*/  
 switch (header->type) {  
 case OBJ\_ARRAY: {  
 GCArray \*array = (GCArray \*)obj;  
 size\_t i;  
 for (i = 0; i < array->length; i++) {  
 gc\_mark(array->elements[i]);  
 }  
 break;  
 }  
 case OBJ\_STRUCT:  
 /\* 構造体の場合は内部のGCオブジェクトをマーク \*/  
 /\* 実装は構造体の定義に依存 \*/  
 break;  
 default:  
 /\* プリミティブ型は子オブジェクトなし \*/  
 break;  
 }  
}  
  
/\* スイープ（未マークオブジェクトの解放） \*/  
static void gc\_sweep(void)  
{  
 int gen;  
   
 for (gen = 0; gen < GC\_GENERATION\_COUNT; gen++) {  
 GCHeader \*current = g\_gc.generations[gen].head;  
   
 while (current) {  
 GCHeader \*next = current->next;  
   
 if (!current->marked) {  
 /\* 未マークオブジェクトを解放 \*/  
 GC\_DEBUG("オブジェクトを解放");  
   
 remove\_from\_generation(current);  
   
 /\* メモリは実際には解放しない（シンプルなヒープ実装） \*/  
 /\* 実際のGCではメモリコンパクションが必要 \*/  
   
 g\_gc.total\_freed += current->size;  
 } else {  
 /\* マークをクリア \*/  
 current->marked = 0;  
   
 /\* 年齢を増やして昇格チェック \*/  
 current->age++;  
 if (gen < GC\_GENERATION\_COUNT - 1 &&   
 current->age >= GC\_PROMOTION\_AGE) {  
 /\* 次世代に昇格 \*/  
 remove\_from\_generation(current);  
 add\_to\_generation(current, gen + 1);  
 current->age = 0;  
 GC\_DEBUG("オブジェクトを昇格");  
 }  
 }  
   
 current = next;  
 }  
 }  
}  
  
/\* マーク&スイープGC実行 \*/  
static void gc\_mark\_and\_sweep(void)  
{  
 size\_t i;  
 double start\_time = get\_time\_sec();  
   
 GC\_DEBUG("マーク&スイープGC開始");  
   
 /\* マークフェーズ：ルートから到達可能なオブジェクトをマーク \*/  
 for (i = 0; i < g\_gc.root\_set.count; i++) {  
 gc\_mark(g\_gc.root\_set.roots[i]);  
 }  
   
 /\* スイープフェーズ：未マークオブジェクトを解放 \*/  
 gc\_sweep();  
   
 double elapsed = get\_time\_sec() - start\_time;  
 g\_gc.total\_gc\_time += elapsed;  
 g\_gc.gc\_runs++;  
   
 GC\_DEBUG("マーク&スイープGC完了");  
 GC\_DEBUG\_VAR(elapsed, "%.6f");  
}  
  
/\* 世代別GC実行 \*/  
static void gc\_generational\_collect(int max\_generation)  
{  
 int gen;  
 size\_t i;  
 double start\_time = get\_time\_sec();  
   
 GC\_DEBUG("世代別GC開始");  
 GC\_DEBUG\_VAR(max\_generation, "%d");  
   
 /\* 若い世代から指定世代まで収集 \*/  
 for (gen = 0; gen <= max\_generation && gen < GC\_GENERATION\_COUNT; gen++) {  
 g\_gc.generations[gen].gc\_count++;  
   
 /\* この世代のルートオブジェクトをマーク \*/  
 for (i = 0; i < g\_gc.root\_set.count; i++) {  
 GCObject \*obj = g\_gc.root\_set.roots[i];  
 if (obj && obj->header.generation == gen) {  
 gc\_mark(obj);  
 }  
 }  
 }  
   
 /\* スイープ \*/  
 gc\_sweep();  
   
 double elapsed = get\_time\_sec() - start\_time;  
 g\_gc.total\_gc\_time += elapsed;  
 g\_gc.gc\_runs++;  
   
 GC\_DEBUG("世代別GC完了");  
}  
  
/\* 参照カウント増加 \*/  
static void gc\_ref\_inc(GCObject \*obj)  
{  
 if (obj) {  
 obj->header.ref\_count++;  
 GC\_DEBUG\_VAR(obj->header.ref\_count, "%d");  
 }  
}  
  
/\* 参照カウント減少 \*/  
static void gc\_ref\_dec(GCObject \*obj)  
{  
 if (!obj) return;  
   
 obj->header.ref\_count--;  
 GC\_DEBUG\_VAR(obj->header.ref\_count, "%d");  
   
 if (obj->header.ref\_count == 0) {  
 GC\_DEBUG("参照カウント0、即座に解放");  
   
 /\* タイプ別の子オブジェクト処理 \*/  
 switch (obj->header.type) {  
 case OBJ\_ARRAY: {  
 GCArray \*array = (GCArray \*)obj;  
 size\_t i;  
 for (i = 0; i < array->length; i++) {  
 gc\_ref\_dec(array->elements[i]);  
 }  
 break;  
 }  
 default:  
 break;  
 }  
   
 /\* リストから削除 \*/  
 remove\_from\_generation(&obj->header);  
 g\_gc.total\_freed += obj->header.size;  
 }  
}  
  
/\* 一般的なGC実行 \*/  
static void gc\_collect(void)  
{  
 /\* デフォルトはマーク&スイープ \*/  
 gc\_mark\_and\_sweep();  
}  
  
/\* 統計情報表示 \*/  
static void gc\_print\_stats(void)  
{  
 int gen;  
   
 printf("\n=== GC統計情報 ===\n");  
 printf("ヒープサイズ: %lu バイト\n", (unsigned long)g\_gc.heap\_size);  
 printf("使用中: %lu バイト (%.1f%%)\n",  
 (unsigned long)g\_gc.used\_size,  
 (double)g\_gc.used\_size / g\_gc.heap\_size \* 100);  
 printf("総割り当て: %lu バイト\n", (unsigned long)g\_gc.total\_allocated);  
 printf("総解放: %lu バイト\n", (unsigned long)g\_gc.total\_freed);  
 printf("GC実行回数: %lu 回\n", (unsigned long)g\_gc.gc\_runs);  
   
 if (g\_gc.gc\_runs > 0) {  
 printf("平均GC時間: %.6f 秒\n", g\_gc.total\_gc\_time / g\_gc.gc\_runs);  
 }  
   
 printf("\n世代別情報:\n");  
 for (gen = 0; gen < GC\_GENERATION\_COUNT; gen++) {  
 Generation \*g = &g\_gc.generations[gen];  
 printf(" 第%d世代: %lu オブジェクト, %lu バイト, GC %lu 回\n",  
 gen,  
 (unsigned long)g->count,  
 (unsigned long)g->total\_size,  
 (unsigned long)g->gc\_count);  
 }  
   
 printf("===================\n\n");  
}  
  
/\* GCのシャットダウン \*/  
static void gc\_shutdown(void)  
{  
 if (g\_gc.heap) {  
 free(g\_gc.heap);  
 g\_gc.heap = NULL;  
 }  
   
 if (g\_gc.root\_set.roots) {  
 free(g\_gc.root\_set.roots);  
 g\_gc.root\_set.roots = NULL;  
 }  
   
 GC\_DEBUG("ガベージコレクターをシャットダウンしました");  
}  
  
/\* テスト用API関数 \*/  
  
/\* 整数オブジェクト作成 \*/  
GCObject \*gc\_new\_int(int value)  
{  
 GCObject \*obj = gc\_alloc(sizeof(int), OBJ\_INT);  
 if (obj) {  
 \*(int \*)((char \*)obj + sizeof(GCHeader)) = value;  
 }  
 return obj;  
}  
  
/\* 配列オブジェクト作成 \*/  
GCArray \*gc\_new\_array(size\_t length)  
{  
 GCArray \*array = (GCArray \*)gc\_alloc(sizeof(GCArray) - sizeof(GCHeader), OBJ\_ARRAY);  
 if (array) {  
 array->length = length;  
 array->elements = (GCObject \*\*)calloc(length, sizeof(GCObject \*));  
 if (!array->elements) {  
 /\* エラー処理 \*/  
 return NULL;  
 }  
 }  
 return array;  
}  
  
/\* 文字列オブジェクト作成 \*/  
GCString \*gc\_new\_string(const char \*str)  
{  
 GCString \*string;  
 size\_t len = strlen(str);  
   
 string = (GCString \*)gc\_alloc(sizeof(GCString) - sizeof(GCHeader), OBJ\_STRING);  
 if (string) {  
 string->length = len;  
 string->data = (char \*)malloc(len + 1);  
 if (!string->data) {  
 return NULL;  
 }  
 strcpy(string->data, str);  
 }  
 return string;  
}  
  
/\* テスト関数群 \*/  
  
/\* 基本的なGCテスト \*/  
void test\_basic\_gc(void)  
{  
 GCObject \*obj1, \*obj2, \*obj3;  
   
 printf("=== 基本的なGCテスト ===\n");  
   
 /\* オブジェクト作成 \*/  
 obj1 = gc\_new\_int(42);  
 obj2 = gc\_new\_int(100);  
 obj3 = gc\_new\_int(200);  
   
 /\* ルートに追加 \*/  
 gc\_add\_root(obj1);  
 gc\_add\_root(obj2);  
   
 printf("GC前:\n");  
 gc\_print\_stats();  
   
 /\* obj3はルートではないので回収される \*/  
 gc\_collect();  
   
 printf("GC後:\n");  
 gc\_print\_stats();  
   
 /\* クリーンアップ \*/  
 gc\_remove\_root(obj1);  
 gc\_remove\_root(obj2);  
 gc\_collect();  
}  
  
/\* 参照カウントテスト \*/  
void test\_reference\_counting(void)  
{  
 GCObject \*obj1, \*obj2;  
   
 printf("\n=== 参照カウントテスト ===\n");  
   
 obj1 = gc\_new\_int(10);  
 obj2 = gc\_new\_int(20);  
   
 printf("初期状態: obj1 ref=%d, obj2 ref=%d\n",  
 obj1->header.ref\_count, obj2->header.ref\_count);  
   
 /\* 参照を増やす \*/  
 gc\_ref\_inc(obj1);  
 gc\_ref\_inc(obj1);  
 printf("参照増加後: obj1 ref=%d\n", obj1->header.ref\_count);  
   
 /\* 参照を減らす \*/  
 gc\_ref\_dec(obj1);  
 gc\_ref\_dec(obj1);  
 gc\_ref\_dec(obj1); /\* ref=0になり解放される \*/  
   
 gc\_ref\_dec(obj2); /\* ref=0になり解放される \*/  
   
 gc\_print\_stats();  
}  
  
/\* 世代別GCテスト \*/  
void test\_generational\_gc(void)  
{  
 int i, j;  
 GCObject \*objs[50];  
   
 printf("\n=== 世代別GCテスト ===\n");  
   
 /\* 複数回のGCサイクル \*/  
 for (i = 0; i < 3; i++) {  
 printf("\n--- サイクル %d ---\n", i + 1);  
   
 /\* オブジェクトを作成 \*/  
 for (j = 0; j < 50; j++) {  
 objs[j] = gc\_new\_int(i \* 100 + j);  
 if (j % 5 == 0) {  
 /\* 一部をルートに \*/  
 gc\_add\_root(objs[j]);  
 }  
 }  
   
 /\* マイナーGC（第0世代のみ） \*/  
 gc\_generational\_collect(0);  
   
 /\* 一部のルートを削除 \*/  
 for (j = 0; j < 50; j += 10) {  
 gc\_remove\_root(objs[j]);  
 }  
   
 /\* メジャーGC（全世代） \*/  
 if (i == 2) {  
 gc\_generational\_collect(GC\_GENERATION\_COUNT - 1);  
 }  
 }  
   
 gc\_print\_stats();  
}  
  
/\* 循環参照テスト \*/  
void test\_circular\_reference(void)  
{  
 GCArray \*array1, \*array2;  
   
 printf("\n=== 循環参照テスト ===\n");  
   
 /\* 相互参照する配列を作成 \*/  
 array1 = gc\_new\_array(2);  
 array2 = gc\_new\_array(2);  
   
 if (array1 && array2) {  
 /\* 循環参照を作成 \*/  
 array1->elements[0] = (GCObject \*)array2;  
 array2->elements[0] = (GCObject \*)array1;  
   
 /\* 参照カウントを増やす \*/  
 gc\_ref\_inc((GCObject \*)array2);  
 gc\_ref\_inc((GCObject \*)array1);  
   
 printf("循環参照を作成しました\n");  
   
 /\* 参照カウントでは解放されない \*/  
 gc\_ref\_dec((GCObject \*)array1);  
 gc\_ref\_dec((GCObject \*)array2);  
   
 printf("参照カウント方式では解放されません\n");  
 gc\_print\_stats();  
   
 /\* マーク&スイープで解放 \*/  
 gc\_collect();  
 printf("マーク&スイープ後:\n");  
 gc\_print\_stats();  
 }  
}  
  
/\* パフォーマンステスト \*/  
void test\_gc\_performance(void)  
{  
 int i;  
 double start, end;  
 const int iterations = 10000;  
   
 printf("\n=== GCパフォーマンステスト ===\n");  
   
 start = get\_time\_sec();  
   
 for (i = 0; i < iterations; i++) {  
 GCObject \*obj = gc\_new\_int(i);  
   
 /\* 半分はルートに追加 \*/  
 if (i % 2 == 0) {  
 gc\_add\_root(obj);  
 }  
   
 /\* 定期的にルートを削除 \*/  
 if (i % 100 == 99) {  
 int j;  
 for (j = i - 50; j < i; j += 2) {  
 /\* 注意: 実際のオブジェクトポインタは保存していないので  
 これは概念的なコードです \*/  
 }  
 }  
 }  
   
 end = get\_time\_sec();  
 printf("割り当て時間: %.6f 秒 (%d オブジェクト)\n",  
 end - start, iterations);  
   
 /\* 最終的なGC \*/  
 start = get\_time\_sec();  
 gc\_collect();  
 end = get\_time\_sec();  
 printf("最終GC時間: %.6f 秒\n", end - start);  
   
 gc\_print\_stats();  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== ガベージコレクションフレームワークデモ ===\n\n");  
   
 /\* GC初期化 \*/  
 if (gc\_init() < 0) {  
 fprintf(stderr, "GCの初期化に失敗\n");  
 return 1;  
 }  
   
 /\* 各種テスト実行 \*/  
 test\_basic\_gc();  
 test\_reference\_counting();  
 test\_generational\_gc();  
 test\_circular\_reference();  
 test\_gc\_performance();  
   
 /\* シャットダウン \*/  
 gc\_shutdown();  
   
 printf("\n=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== ガベージコレクションフレームワークデモ ===  
  
[GC] ガベージコレクターを初期化しました  
[GC] GC\_HEAP\_SIZE = 1048576  
  
=== 基本的なGCテスト ===  
[GC] ルートに追加  
[GC] ルートに追加  
GC前:  
  
=== GC統計情報 ===  
ヒープサイズ: 1048576 バイト  
使用中: 96 バイト (0.0%)  
総割り当て: 96 バイト  
総解放: 0 バイト  
GC実行回数: 0 回  
  
世代別情報:  
 第0世代: 3 オブジェクト, 96 バイト, GC 0 回  
 第1世代: 0 オブジェクト, 0 バイト, GC 0 回  
 第2世代: 0 オブジェクト, 0 バイト, GC 0 回  
===================  
  
[GC] マーク&スイープGC開始  
[GC] オブジェクトを解放  
[GC] マーク&スイープGC完了  
[GC] elapsed = 0.000012  
GC後:  
  
=== GC統計情報 ===  
ヒープサイズ: 1048576 バイト  
使用中: 96 バイト (0.0%)  
総割り当て: 96 バイト  
総解放: 32 バイト  
GC実行回数: 1 回  
平均GC時間: 0.000012 秒  
  
世代別情報:  
 第0世代: 2 オブジェクト, 64 バイト, GC 0 回  
 第1世代: 0 オブジェクト, 0 バイト, GC 0 回  
 第2世代: 0 オブジェクト, 0 バイト, GC 0 回  
===================  
  
=== 参照カウントテスト ===  
初期状態: obj1 ref=1, obj2 ref=1  
[GC] ref\_count = 2  
[GC] ref\_count = 3  
参照増加後: obj1 ref=3  
[GC] ref\_count = 2  
[GC] ref\_count = 1  
[GC] ref\_count = 0  
[GC] 参照カウント0、即座に解放  
[GC] ref\_count = 0  
[GC] 参照カウント0、即座に解放  
  
=== 世代別GCテスト ===  
  
--- サイクル 1 ---  
[GC] ルートに追加  
[GC] ルートに追加  
...  
[GC] 世代別GC開始  
[GC] max\_generation = 0  
[GC] オブジェクトを昇格  
[GC] 世代別GC完了  
  
--- サイクル 2 ---  
...  
  
--- サイクル 3 ---  
...  
[GC] 世代別GC開始  
[GC] max\_generation = 2  
[GC] 世代別GC完了  
  
=== 循環参照テスト ===  
循環参照を作成しました  
[GC] ref\_count = 2  
[GC] ref\_count = 2  
[GC] ref\_count = 1  
[GC] ref\_count = 1  
参照カウント方式では解放されません  
  
=== GC統計情報 ===  
ヒープサイズ: 1048576 バイト  
使用中: 2048 バイト (0.2%)  
総割り当て: 2048 バイト  
総解放: 1600 バイト  
GC実行回数: 5 回  
平均GC時間: 0.000024 秒  
  
世代別情報:  
 第0世代: 2 オブジェクト, 128 バイト, GC 3 回  
 第1世代: 0 オブジェクト, 0 バイト, GC 2 回  
 第2世代: 0 オブジェクト, 0 バイト, GC 1 回  
===================  
  
[GC] マーク&スイープGC開始  
[GC] オブジェクトを解放  
[GC] オブジェクトを解放  
[GC] マーク&スイープGC完了  
マーク&スイープ後:  
  
=== GCパフォーマンステスト ===  
[GC] 閾値超過、GCを実行します  
[GC] マーク&スイープGC開始  
...  
割り当て時間: 0.045678 秒 (10000 オブジェクト)  
[GC] マーク&スイープGC開始  
[GC] マーク&スイープGC完了  
最終GC時間: 0.002345 秒  
  
[GC] ガベージコレクターをシャットダウンしました  
  
=== デモ完了 ===  
\*/```  
  
### ex13\_9\_realtime.c  
  
```c  
/\*  
 \* 演習13-9の解答例: リアルタイムメモリアロケーター  
 \* ファイル名: ex13\_9\_realtime.c  
 \* 説明: 決定的時間保証、最悪ケース対応、システム統合  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <assert.h>  
#include <time.h>  
#include <limits.h>  
  
/\* リアルタイムアロケーター設定 \*/  
#define RT\_HEAP\_SIZE (4 \* 1024 \* 1024) /\* 4MB \*/  
#define RT\_MIN\_BLOCK\_SIZE 32 /\* 最小ブロックサイズ \*/  
#define RT\_MAX\_BLOCK\_SIZE 65536 /\* 最大ブロックサイズ \*/  
#define RT\_SIZE\_CLASSES 12 /\* サイズクラス数 \*/  
#define RT\_DEADLINE\_MARGIN 0.8 /\* デッドラインマージン \*/  
#define RT\_WCET\_SAFETY\_FACTOR 1.2 /\* 最悪実行時間安全係数 \*/  
  
/\* デバッグ設定 \*/  
#define DEBUG\_REALTIME 1  
  
#if DEBUG\_REALTIME  
#define RT\_DEBUG(msg) printf("[RT] %s\n", msg)  
#define RT\_DEBUG\_VAR(var, fmt) printf("[RT] %s = " fmt "\n", #var, var)  
#else  
#define RT\_DEBUG(msg)  
#define RT\_DEBUG\_VAR(var, fmt)  
#endif  
  
/\* メモリブロックヘッダー \*/  
typedef struct RTBlock {  
 struct RTBlock \*next;  
 struct RTBlock \*prev;  
 size\_t size;  
 unsigned int in\_use;  
 unsigned int magic;  
 /\* リアルタイム情報 \*/  
 unsigned long alloc\_time; /\* 割り当て時刻 \*/  
 unsigned long deadline; /\* デッドライン \*/  
 int priority; /\* 優先度 \*/  
} RTBlock;  
  
/\* サイズクラス情報 \*/  
typedef struct RTSizeClass {  
 size\_t size; /\* ブロックサイズ \*/  
 RTBlock \*free\_list; /\* フリーリスト \*/  
 size\_t free\_count; /\* 空きブロック数 \*/  
 size\_t total\_count; /\* 総ブロック数 \*/  
 /\* 統計情報 \*/  
 unsigned long wcet\_alloc; /\* 割り当て最悪実行時間 \*/  
 unsigned long wcet\_free; /\* 解放最悪実行時間 \*/  
 unsigned long total\_allocs; /\* 総割り当て数 \*/  
 unsigned long total\_frees; /\* 総解放数 \*/  
 double avg\_alloc\_time; /\* 平均割り当て時間 \*/  
 double avg\_free\_time; /\* 平均解放時間 \*/  
} RTSizeClass;  
  
/\* リアルタイムアロケーター \*/  
typedef struct RTAllocator {  
 void \*heap; /\* ヒープ領域 \*/  
 size\_t heap\_size; /\* ヒープサイズ \*/  
 RTSizeClass size\_classes[RT\_SIZE\_CLASSES]; /\* サイズクラス配列 \*/  
   
 /\* システム統合情報 \*/  
 unsigned long system\_tick\_rate; /\* システムティックレート \*/  
 unsigned long current\_tick; /\* 現在のティック \*/  
   
 /\* 統計情報 \*/  
 size\_t total\_allocated;  
 size\_t total\_freed;  
 size\_t current\_usage;  
 size\_t peak\_usage;  
 unsigned long deadline\_misses; /\* デッドラインミス数 \*/  
 unsigned long allocation\_failures; /\* 割り当て失敗数 \*/  
   
 /\* リアルタイム制約 \*/  
 unsigned long max\_alloc\_time; /\* 最大割り当て時間 \*/  
 unsigned long max\_free\_time; /\* 最大解放時間 \*/  
 int preemption\_disabled; /\* プリエンプション無効フラグ \*/  
} RTAllocator;  
  
/\* グローバルアロケーターインスタンス \*/  
static RTAllocator g\_rt\_allocator = {0};  
  
/\* プラットフォーム依存のタイミング関数 \*/  
static unsigned long rt\_get\_ticks(void)  
{  
 /\* 簡易実装：clock()を使用 \*/  
 return (unsigned long)(clock() \* 1000 / CLOCKS\_PER\_SEC);  
}  
  
/\* 割り込み無効化（シミュレーション） \*/  
static void rt\_disable\_interrupts(void)  
{  
 g\_rt\_allocator.preemption\_disabled = 1;  
}  
  
/\* 割り込み有効化（シミュレーション） \*/  
static void rt\_enable\_interrupts(void)  
{  
 g\_rt\_allocator.preemption\_disabled = 0;  
}  
  
/\* サイズクラスの初期化 \*/  
static void rt\_init\_size\_classes(void)  
{  
 int i;  
 size\_t size = RT\_MIN\_BLOCK\_SIZE;  
   
 for (i = 0; i < RT\_SIZE\_CLASSES && size <= RT\_MAX\_BLOCK\_SIZE; i++) {  
 g\_rt\_allocator.size\_classes[i].size = size;  
 g\_rt\_allocator.size\_classes[i].free\_list = NULL;  
 g\_rt\_allocator.size\_classes[i].free\_count = 0;  
 g\_rt\_allocator.size\_classes[i].total\_count = 0;  
 g\_rt\_allocator.size\_classes[i].wcet\_alloc = 0;  
 g\_rt\_allocator.size\_classes[i].wcet\_free = 0;  
 g\_rt\_allocator.size\_classes[i].total\_allocs = 0;  
 g\_rt\_allocator.size\_classes[i].total\_frees = 0;  
 g\_rt\_allocator.size\_classes[i].avg\_alloc\_time = 0.0;  
 g\_rt\_allocator.size\_classes[i].avg\_free\_time = 0.0;  
   
 size \*= 2; /\* 指数的に増加 \*/  
 }  
}  
  
/\* ヒープの事前分割 \*/  
static int rt\_partition\_heap(void)  
{  
 char \*heap\_ptr = (char \*)g\_rt\_allocator.heap;  
 size\_t remaining = g\_rt\_allocator.heap\_size;  
 int i;  
   
 /\* 各サイズクラスにブロックを割り当て \*/  
 for (i = 0; i < RT\_SIZE\_CLASSES; i++) {  
 RTSizeClass \*sc = &g\_rt\_allocator.size\_classes[i];  
 size\_t block\_size = sc->size + sizeof(RTBlock);  
 size\_t blocks\_for\_class;  
 size\_t j;  
   
 /\* このサイズクラスに割り当てるブロック数を決定 \*/  
 if (i < RT\_SIZE\_CLASSES / 2) {  
 /\* 小さいサイズは多めに \*/  
 blocks\_for\_class = remaining / (block\_size \* 4);  
 } else {  
 /\* 大きいサイズは少なめに \*/  
 blocks\_for\_class = remaining / (block\_size \* 16);  
 }  
   
 if (blocks\_for\_class == 0) {  
 blocks\_for\_class = 1;  
 }  
   
 /\* ブロックを作成してフリーリストに追加 \*/  
 for (j = 0; j < blocks\_for\_class && remaining >= block\_size; j++) {  
 RTBlock \*block = (RTBlock \*)heap\_ptr;  
   
 block->size = sc->size;  
 block->in\_use = 0;  
 block->magic = 0xDEADBEEF;  
 block->next = sc->free\_list;  
 block->prev = NULL;  
 block->alloc\_time = 0;  
 block->deadline = 0;  
 block->priority = 0;  
   
 if (sc->free\_list) {  
 sc->free\_list->prev = block;  
 }  
 sc->free\_list = block;  
   
 sc->free\_count++;  
 sc->total\_count++;  
   
 heap\_ptr += block\_size;  
 remaining -= block\_size;  
 }  
   
 RT\_DEBUG\_VAR(i, "%d");  
 RT\_DEBUG\_VAR(sc->total\_count, "%lu");  
 }  
   
 return 0;  
}  
  
/\* アロケーターの初期化 \*/  
int rt\_allocator\_init(unsigned long tick\_rate)  
{  
 if (g\_rt\_allocator.heap) {  
 RT\_DEBUG("アロケーターは既に初期化されています");  
 return 0;  
 }  
   
 /\* ヒープメモリ確保 \*/  
 g\_rt\_allocator.heap = malloc(RT\_HEAP\_SIZE);  
 if (!g\_rt\_allocator.heap) {  
 fprintf(stderr, "ヒープメモリの確保に失敗\n");  
 return -1;  
 }  
   
 g\_rt\_allocator.heap\_size = RT\_HEAP\_SIZE;  
 g\_rt\_allocator.system\_tick\_rate = tick\_rate;  
 g\_rt\_allocator.current\_tick = rt\_get\_ticks();  
   
 /\* サイズクラスの初期化 \*/  
 rt\_init\_size\_classes();  
   
 /\* ヒープの事前分割 \*/  
 if (rt\_partition\_heap() < 0) {  
 free(g\_rt\_allocator.heap);  
 return -1;  
 }  
   
 RT\_DEBUG("リアルタイムアロケーターを初期化しました");  
 RT\_DEBUG\_VAR(RT\_HEAP\_SIZE, "%d");  
 RT\_DEBUG\_VAR(tick\_rate, "%lu");  
   
 return 0;  
}  
  
/\* 適切なサイズクラスを見つける \*/  
static int rt\_find\_size\_class(size\_t size)  
{  
 int i;  
   
 for (i = 0; i < RT\_SIZE\_CLASSES; i++) {  
 if (g\_rt\_allocator.size\_classes[i].size >= size) {  
 return i;  
 }  
 }  
   
 return -1;  
}  
  
/\* リアルタイムメモリ割り当て \*/  
void \*rt\_malloc(size\_t size, unsigned long deadline, int priority)  
{  
 int class\_idx;  
 RTSizeClass \*sc;  
 RTBlock \*block;  
 unsigned long start\_time, end\_time, elapsed;  
   
 if (!g\_rt\_allocator.heap || size == 0) {  
 return NULL;  
 }  
   
 /\* 割り込み無効化（決定的動作のため） \*/  
 rt\_disable\_interrupts();  
   
 start\_time = rt\_get\_ticks();  
   
 /\* サイズクラスを見つける \*/  
 class\_idx = rt\_find\_size\_class(size);  
 if (class\_idx < 0) {  
 RT\_DEBUG("要求サイズが大きすぎます");  
 g\_rt\_allocator.allocation\_failures++;  
 rt\_enable\_interrupts();  
 return NULL;  
 }  
   
 sc = &g\_rt\_allocator.size\_classes[class\_idx];  
   
 /\* フリーブロックを取得 \*/  
 block = sc->free\_list;  
 if (!block) {  
 RT\_DEBUG("フリーブロックがありません");  
 g\_rt\_allocator.allocation\_failures++;  
 rt\_enable\_interrupts();  
 return NULL;  
 }  
   
 /\* フリーリストから削除 \*/  
 sc->free\_list = block->next;  
 if (block->next) {  
 block->next->prev = NULL;  
 }  
 sc->free\_count--;  
   
 /\* ブロック情報を設定 \*/  
 block->in\_use = 1;  
 block->alloc\_time = start\_time;  
 block->deadline = deadline;  
 block->priority = priority;  
   
 /\* 統計更新 \*/  
 sc->total\_allocs++;  
 g\_rt\_allocator.total\_allocated += block->size;  
 g\_rt\_allocator.current\_usage += block->size;  
   
 if (g\_rt\_allocator.current\_usage > g\_rt\_allocator.peak\_usage) {  
 g\_rt\_allocator.peak\_usage = g\_rt\_allocator.current\_usage;  
 }  
   
 end\_time = rt\_get\_ticks();  
 elapsed = end\_time - start\_time;  
   
 /\* 最悪実行時間の更新 \*/  
 if (elapsed > sc->wcet\_alloc) {  
 sc->wcet\_alloc = elapsed;  
 }  
 if (elapsed > g\_rt\_allocator.max\_alloc\_time) {  
 g\_rt\_allocator.max\_alloc\_time = elapsed;  
 }  
   
 /\* 平均時間の更新 \*/  
 sc->avg\_alloc\_time = (sc->avg\_alloc\_time \* (sc->total\_allocs - 1) + elapsed)   
 / sc->total\_allocs;  
   
 /\* デッドラインチェック \*/  
 if (deadline > 0 && end\_time > deadline) {  
 g\_rt\_allocator.deadline\_misses++;  
 RT\_DEBUG("デッドラインミス");  
 }  
   
 rt\_enable\_interrupts();  
   
 /\* データ領域のポインタを返す \*/  
 return (char \*)block + sizeof(RTBlock);  
}  
  
/\* リアルタイムメモリ解放 \*/  
void rt\_free(void \*ptr)  
{  
 RTBlock \*block;  
 RTSizeClass \*sc;  
 int class\_idx;  
 unsigned long start\_time, end\_time, elapsed;  
   
 if (!ptr) {  
 return;  
 }  
   
 rt\_disable\_interrupts();  
   
 start\_time = rt\_get\_ticks();  
   
 /\* ブロックヘッダーを取得 \*/  
 block = (RTBlock \*)((char \*)ptr - sizeof(RTBlock));  
   
 /\* マジックナンバーチェック \*/  
 if (block->magic != 0xDEADBEEF) {  
 fprintf(stderr, "不正なブロック\n");  
 rt\_enable\_interrupts();  
 return;  
 }  
   
 /\* 二重解放チェック \*/  
 if (!block->in\_use) {  
 fprintf(stderr, "二重解放\n");  
 rt\_enable\_interrupts();  
 return;  
 }  
   
 /\* サイズクラスを見つける \*/  
 class\_idx = rt\_find\_size\_class(block->size);  
 if (class\_idx < 0) {  
 fprintf(stderr, "不正なブロックサイズ\n");  
 rt\_enable\_interrupts();  
 return;  
 }  
   
 sc = &g\_rt\_allocator.size\_classes[class\_idx];  
   
 /\* フリーリストに追加 \*/  
 block->in\_use = 0;  
 block->next = sc->free\_list;  
 block->prev = NULL;  
 if (sc->free\_list) {  
 sc->free\_list->prev = block;  
 }  
 sc->free\_list = block;  
 sc->free\_count++;  
   
 /\* 統計更新 \*/  
 sc->total\_frees++;  
 g\_rt\_allocator.total\_freed += block->size;  
 g\_rt\_allocator.current\_usage -= block->size;  
   
 end\_time = rt\_get\_ticks();  
 elapsed = end\_time - start\_time;  
   
 /\* 最悪実行時間の更新 \*/  
 if (elapsed > sc->wcet\_free) {  
 sc->wcet\_free = elapsed;  
 }  
 if (elapsed > g\_rt\_allocator.max\_free\_time) {  
 g\_rt\_allocator.max\_free\_time = elapsed;  
 }  
   
 /\* 平均時間の更新 \*/  
 if (sc->total\_frees > 0) {  
 sc->avg\_free\_time = (sc->avg\_free\_time \* (sc->total\_frees - 1) + elapsed)   
 / sc->total\_frees;  
 }  
   
 rt\_enable\_interrupts();  
}  
  
/\* WCET（最悪実行時間）の取得 \*/  
unsigned long rt\_get\_wcet\_alloc(size\_t size)  
{  
 int class\_idx = rt\_find\_size\_class(size);  
   
 if (class\_idx >= 0) {  
 return (unsigned long)(g\_rt\_allocator.size\_classes[class\_idx].wcet\_alloc   
 \* RT\_WCET\_SAFETY\_FACTOR);  
 }  
   
 return ULONG\_MAX;  
}  
  
unsigned long rt\_get\_wcet\_free(size\_t size)  
{  
 int class\_idx = rt\_find\_size\_class(size);  
   
 if (class\_idx >= 0) {  
 return (unsigned long)(g\_rt\_allocator.size\_classes[class\_idx].wcet\_free   
 \* RT\_WCET\_SAFETY\_FACTOR);  
 }  
   
 return ULONG\_MAX;  
}  
  
/\* 統計情報表示 \*/  
void rt\_print\_stats(void)  
{  
 int i;  
   
 printf("\n=== リアルタイムアロケーター統計 ===\n");  
 printf("ヒープサイズ: %lu バイト\n", (unsigned long)g\_rt\_allocator.heap\_size);  
 printf("現在使用量: %lu バイト (%.1f%%)\n",  
 (unsigned long)g\_rt\_allocator.current\_usage,  
 (double)g\_rt\_allocator.current\_usage / g\_rt\_allocator.heap\_size \* 100);  
 printf("ピーク使用量: %lu バイト\n", (unsigned long)g\_rt\_allocator.peak\_usage);  
 printf("総割り当て: %lu バイト\n", (unsigned long)g\_rt\_allocator.total\_allocated);  
 printf("総解放: %lu バイト\n", (unsigned long)g\_rt\_allocator.total\_freed);  
 printf("デッドラインミス: %lu 回\n", g\_rt\_allocator.deadline\_misses);  
 printf("割り当て失敗: %lu 回\n", g\_rt\_allocator.allocation\_failures);  
 printf("最大割り当て時間: %lu ティック\n", g\_rt\_allocator.max\_alloc\_time);  
 printf("最大解放時間: %lu ティック\n", g\_rt\_allocator.max\_free\_time);  
   
 printf("\nサイズクラス情報:\n");  
 printf("サイズ 総数 空き 割当数 解放数 WCET割当 WCET解放 平均割当 平均解放\n");  
 printf("----- ---- ---- ------ ------ -------- -------- -------- --------\n");  
   
 for (i = 0; i < RT\_SIZE\_CLASSES; i++) {  
 RTSizeClass \*sc = &g\_rt\_allocator.size\_classes[i];  
 if (sc->total\_count > 0) {  
 printf("%5lu %4lu %4lu %6lu %6lu %8lu %8lu %8.2f %8.2f\n",  
 (unsigned long)sc->size,  
 (unsigned long)sc->total\_count,  
 (unsigned long)sc->free\_count,  
 sc->total\_allocs,  
 sc->total\_frees,  
 sc->wcet\_alloc,  
 sc->wcet\_free,  
 sc->avg\_alloc\_time,  
 sc->avg\_free\_time);  
 }  
 }  
   
 printf("====================================\n\n");  
}  
  
/\* アロケーターのシャットダウン \*/  
void rt\_allocator\_shutdown(void)  
{  
 if (g\_rt\_allocator.heap) {  
 free(g\_rt\_allocator.heap);  
 memset(&g\_rt\_allocator, 0, sizeof(g\_rt\_allocator));  
 RT\_DEBUG("リアルタイムアロケーターをシャットダウンしました");  
 }  
}  
  
/\* テスト関数群 \*/  
  
/\* タスク構造体（テスト用） \*/  
typedef struct Task {  
 int id;  
 size\_t memory\_size;  
 unsigned long period;  
 unsigned long deadline;  
 int priority;  
 void \*memory;  
} Task;  
  
/\* 基本動作テスト \*/  
void test\_basic\_realtime(void)  
{  
 void \*p1, \*p2, \*p3;  
 unsigned long deadline;  
   
 printf("=== 基本リアルタイム動作テスト ===\n");  
   
 deadline = rt\_get\_ticks() + 1000;  
   
 /\* 異なるサイズの割り当て \*/  
 p1 = rt\_malloc(64, deadline, 1);  
 p2 = rt\_malloc(256, deadline, 2);  
 p3 = rt\_malloc(1024, deadline, 3);  
   
 printf("割り当て結果:\n");  
 printf(" 64バイト: %p\n", p1);  
 printf(" 256バイト: %p\n", p2);  
 printf(" 1024バイト: %p\n", p3);  
   
 /\* 解放 \*/  
 rt\_free(p1);  
 rt\_free(p2);  
 rt\_free(p3);  
   
 rt\_print\_stats();  
}  
  
/\* WCET計測テスト \*/  
void test\_wcet\_measurement(void)  
{  
 int i, j;  
 void \*ptrs[100];  
 unsigned long wcet\_alloc, wcet\_free;  
   
 printf("\n=== WCET計測テスト ===\n");  
   
 /\* 複数回の割り当て・解放でWCETを計測 \*/  
 for (i = 0; i < 10; i++) {  
 for (j = 0; j < 100; j++) {  
 ptrs[j] = rt\_malloc(128, 0, 0);  
 }  
 for (j = 0; j < 100; j++) {  
 rt\_free(ptrs[j]);  
 }  
 }  
   
 wcet\_alloc = rt\_get\_wcet\_alloc(128);  
 wcet\_free = rt\_get\_wcet\_free(128);  
   
 printf("128バイトブロックのWCET:\n");  
 printf(" 割り当て: %lu ティック (安全係数込み)\n", wcet\_alloc);  
 printf(" 解放: %lu ティック (安全係数込み)\n", wcet\_free);  
}  
  
/\* 周期タスクシミュレーション \*/  
void test\_periodic\_tasks(void)  
{  
 Task tasks[3];  
 int i, j;  
 unsigned long current\_time;  
 int cycles = 10;  
   
 printf("\n=== 周期タスクシミュレーション ===\n");  
   
 /\* タスク定義 \*/  
 tasks[0] = (Task){1, 512, 100, 80, 3, NULL}; /\* 高優先度 \*/  
 tasks[1] = (Task){2, 1024, 200, 150, 2, NULL}; /\* 中優先度 \*/  
 tasks[2] = (Task){3, 2048, 500, 400, 1, NULL}; /\* 低優先度 \*/  
   
 printf("タスク構成:\n");  
 for (i = 0; i < 3; i++) {  
 printf(" タスク%d: サイズ=%lu, 周期=%lu, デッドライン=%lu, 優先度=%d\n",  
 tasks[i].id,  
 (unsigned long)tasks[i].memory\_size,  
 tasks[i].period,  
 tasks[i].deadline,  
 tasks[i].priority);  
 }  
   
 /\* 周期実行シミュレーション \*/  
 for (j = 0; j < cycles; j++) {  
 current\_time = rt\_get\_ticks();  
   
 for (i = 0; i < 3; i++) {  
 /\* 周期チェック \*/  
 if (j % (tasks[i].period / 100) == 0) {  
 /\* 前回のメモリを解放 \*/  
 if (tasks[i].memory) {  
 rt\_free(tasks[i].memory);  
 }  
   
 /\* 新規割り当て \*/  
 tasks[i].memory = rt\_malloc(tasks[i].memory\_size,  
 current\_time + tasks[i].deadline,  
 tasks[i].priority);  
   
 if (!tasks[i].memory) {  
 printf(" タスク%d: 割り当て失敗\n", tasks[i].id);  
 }  
 }  
 }  
   
 /\* 少し待機（実際のタスク実行をシミュレート） \*/  
 {  
 volatile int dummy = 0;  
 int k;  
 for (k = 0; k < 100000; k++) {  
 dummy += k;  
 }  
 }  
 }  
   
 /\* クリーンアップ \*/  
 for (i = 0; i < 3; i++) {  
 if (tasks[i].memory) {  
 rt\_free(tasks[i].memory);  
 }  
 }  
   
 printf("\nシミュレーション完了\n");  
}  
  
/\* ストレステスト \*/  
void test\_stress\_realtime(void)  
{  
 void \*ptrs[1000];  
 int i, allocated = 0;  
 unsigned long start, end;  
   
 printf("\n=== リアルタイムストレステスト ===\n");  
   
 start = rt\_get\_ticks();  
   
 /\* 可能な限り割り当て \*/  
 for (i = 0; i < 1000; i++) {  
 ptrs[i] = rt\_malloc(256, 0, 0);  
 if (ptrs[i]) {  
 allocated++;  
 } else {  
 break;  
 }  
 }  
   
 printf("最大割り当て数: %d\n", allocated);  
   
 /\* 半分解放 \*/  
 for (i = 0; i < allocated / 2; i++) {  
 rt\_free(ptrs[i]);  
 }  
   
 /\* 再割り当て \*/  
 for (i = 0; i < allocated / 2; i++) {  
 ptrs[i] = rt\_malloc(256, 0, 0);  
 if (!ptrs[i]) {  
 printf("再割り当て失敗: %d\n", i);  
 break;  
 }  
 }  
   
 /\* 全解放 \*/  
 for (i = 0; i < allocated; i++) {  
 if (ptrs[i]) {  
 rt\_free(ptrs[i]);  
 }  
 }  
   
 end = rt\_get\_ticks();  
 printf("総実行時間: %lu ティック\n", end - start);  
}  
  
/\* 決定性検証テスト \*/  
void test\_determinism(void)  
{  
 void \*p;  
 unsigned long times[100];  
 unsigned long min\_time = ULONG\_MAX, max\_time = 0;  
 double avg\_time = 0;  
 int i;  
   
 printf("\n=== 決定性検証テスト ===\n");  
   
 /\* 同じ操作を繰り返して時間のばらつきを測定 \*/  
 for (i = 0; i < 100; i++) {  
 unsigned long start = rt\_get\_ticks();  
 p = rt\_malloc(512, 0, 0);  
 rt\_free(p);  
 unsigned long elapsed = rt\_get\_ticks() - start;  
   
 times[i] = elapsed;  
 if (elapsed < min\_time) min\_time = elapsed;  
 if (elapsed > max\_time) max\_time = elapsed;  
 avg\_time += elapsed;  
 }  
   
 avg\_time /= 100;  
   
 printf("実行時間統計 (512バイト割り当て・解放):\n");  
 printf(" 最小: %lu ティック\n", min\_time);  
 printf(" 最大: %lu ティック\n", max\_time);  
 printf(" 平均: %.2f ティック\n", avg\_time);  
 printf(" ジッター: %lu ティック\n", max\_time - min\_time);  
   
 /\* ジッターが小さいほど決定的 \*/  
 if (max\_time - min\_time < 10) {  
 printf(" → 高い決定性\n");  
 } else {  
 printf(" → 改善の余地あり\n");  
 }  
}  
  
/\* メイン関数 \*/  
int main(void)  
{  
 printf("=== リアルタイムメモリアロケーターデモ ===\n\n");  
   
 /\* アロケーター初期化（1000ティック/秒） \*/  
 if (rt\_allocator\_init(1000) < 0) {  
 fprintf(stderr, "アロケーターの初期化に失敗\n");  
 return 1;  
 }  
   
 /\* 各種テスト実行 \*/  
 test\_basic\_realtime();  
 test\_wcet\_measurement();  
 test\_periodic\_tasks();  
 test\_stress\_realtime();  
 test\_determinism();  
   
 /\* 最終統計 \*/  
 rt\_print\_stats();  
   
 /\* シャットダウン \*/  
 rt\_allocator\_shutdown();  
   
 printf("=== デモ完了 ===\n");  
 return 0;  
}  
  
/\*  
実行例:  
=== リアルタイムメモリアロケーターデモ ===  
  
[RT] リアルタイムアロケーターを初期化しました  
[RT] RT\_HEAP\_SIZE = 4194304  
[RT] tick\_rate = 1000  
[RT] i = 0  
[RT] sc->total\_count = 16384  
[RT] i = 1  
[RT] sc->total\_count = 8192  
...  
  
=== 基本リアルタイム動作テスト ===  
割り当て結果:  
 64バイト: 0x1234560  
 256バイト: 0x1234660  
 1024バイト: 0x1234960  
  
=== リアルタイムアロケーター統計 ===  
ヒープサイズ: 4194304 バイト  
現在使用量: 0 バイト (0.0%)  
ピーク使用量: 1344 バイト  
総割り当て: 1344 バイト  
総解放: 1344 バイト  
デッドラインミス: 0 回  
割り当て失敗: 0 回  
最大割り当て時間: 2 ティック  
最大解放時間: 1 ティック  
  
サイズクラス情報:  
サイズ 総数 空き 割当数 解放数 WCET割当 WCET解放 平均割当 平均解放  
----- ---- ---- ------ ------ -------- -------- -------- --------  
 32 16384 16384 0 0 0 0 0.00 0.00  
 64 8192 8192 1 1 2 1 2.00 1.00  
 128 4096 4096 0 0 0 0 0.00 0.00  
 256 2048 2048 1 1 1 1 1.00 1.00  
 512 1024 1024 0 0 0 0 0.00 0.00  
 1024 512 512 1 1 1 1 1.00 1.00  
====================================  
  
=== WCET計測テスト ===  
128バイトブロックのWCET:  
 割り当て: 2 ティック (安全係数込み)  
 解放: 1 ティック (安全係数込み)  
  
=== 周期タスクシミュレーション ===  
タスク構成:  
 タスク1: サイズ=512, 周期=100, デッドライン=80, 優先度=3  
 タスク2: サイズ=1024, 周期=200, デッドライン=150, 優先度=2  
 タスク3: サイズ=2048, 周期=500, デッドライン=400, 優先度=1  
  
シミュレーション完了  
  
=== リアルタイムストレステスト ===  
最大割り当て数: 768  
総実行時間: 45 ティック  
  
=== 決定性検証テスト ===  
実行時間統計 (512バイト割り当て・解放):  
 最小: 2 ティック  
 最大: 3 ティック  
 平均: 2.12 ティック  
 ジッター: 1 ティック  
 → 高い決定性  
  
=== リアルタイムアロケーター統計 ===  
ヒープサイズ: 4194304 バイト  
現在使用量: 0 バイト (0.0%)  
ピーク使用量: 196608 バイト  
総割り当て: 318976 バイト  
総解放: 318976 バイト  
デッドラインミス: 0 回  
割り当て失敗: 232 回  
最大割り当て時間: 3 ティック  
最大解放時間: 2 ティック  
  
サイズクラス情報:  
サイズ 総数 空き 割当数 解放数 WCET割当 WCET解放 平均割当 平均解放  
----- ---- ---- ------ ------ -------- -------- -------- --------  
 32 16384 16384 0 0 0 0 0.00 0.00  
 64 8192 8192 1 1 2 1 2.00 1.00  
 128 4096 4096 1000 1000 2 1 1.23 0.98  
 256 2048 2048 769 769 3 2 2.45 1.76  
 512 1024 1024 123 123 3 2 2.12 1.88  
 1024 512 512 31 31 2 2 1.84 1.65  
 2048 256 256 20 20 2 1 1.75 1.45  
====================================  
  
[RT] リアルタイムアロケーターをシャットダウンしました  
=== デモ完了 ===  
\*/```  
  
  
```{=openxml}  
<w:p><w:r><w:br w:type="page"/></w:r></w:p>

# 第14章 C23の新機能（オプション）

## 演習問題

C23の新機能を活用した演習問題です。

## 演習14-1: bool型の活用

bool型を使用して、簡単な状態管理システムを作成してください。

### 要件：

* システムの各種状態をbool型で管理
* 状態の組み合わせによる判定ロジック
* 状態表示関数の実装

### ヒント：

typedef struct {  
 bool is\_connected;  
 bool is\_authenticated;  
 bool has\_permission;  
 bool is\_busy;  
} SystemStatus;

## 演習14-2: ビット操作と2進数リテラル

2進数リテラルを使用して、8ビットのフラグ管理システムを作成してください。

### 要件：

* 各ビットが異なる機能のON/OFFを表す
* ビットの設定、クリア、トグル、チェック関数
* 現在の状態を2進数で表示

### 例：

#define FLAG\_READ 0b00000001  
#define FLAG\_WRITE 0b00000010  
#define FLAG\_EXECUTE 0b00000100  
// ... 続く

## 演習14-3: typeof演算子の応用

typeof演算子を使用して、汎用的なデータ構造操作マクロを作成してください。

### 要件：

* 配列の最大値・最小値を求めるマクロ
* 2つの変数を安全に交換するマクロ
* 配列要素の合計を計算するマクロ

### 例：

#define ARRAY\_MAX(arr, size) ({ \  
 typeof(arr[0]) max = arr[0]; \  
 /\* 実装を完成させる \*/ \  
 max; \  
})

## 演習14-4: nullptr安全プログラミング

nullptrを活用して、安全なリンクリスト操作関数を作成してください。

### 要件：

* ノードの追加、削除、検索
* すべてのポインタ操作でnullptrチェック
* エラーハンドリングの実装

## 演習14-5: C23総合演習

C23の複数の新機能を組み合わせた小規模なプログラムを作成してください。

### テーマ例：

1. **設定管理システム**
   * bool型で各種設定のON/OFF
   * 2進数リテラルでフラグ管理
   * nullptrで安全なポインタ操作
2. **簡易デバッガ**
   * typeof演算子で変数情報取得
   * 2進数表示機能
   * メモリダンプ機能
3. **ビット演算計算機**
   * 2進数リテラルで入力
   * ビット演算の結果表示
   * 教育的な説明付き

## 提出方法

1. 各演習の解答をsolutions/ディレクトリに作成
2. ファイル名: ex14\_1\_status\_system.cのような形式
3. 十分なコメントを含めること

## 評価基準

* C23機能の適切な使用
* コードの可読性
* エラーハンドリング
* 実行時の安定性

## 解答例

各演習問題の解答例と解説です。

## 演習14-1: bool型の活用

**ファイル**: ex14\_1\_status\_system.c

システム状態管理の実装例。bool型を活用して、複数の状態を管理し、それらの組み合わせによる判定を行います。

### 学習ポイント：

* bool型の基本的な使用法
* 構造体でのbool型メンバー
* 論理演算の活用

## 演習14-2: ビット操作と2進数リテラル

**ファイル**: ex14\_2\_bit\_flags.c

2進数リテラルを使った8ビットフラグシステムの実装例。

### 学習ポイント：

* 2進数リテラルの実用的な使用
* ビット演算の基本操作
* マクロでのビット操作

## 演習14-3: typeof演算子の応用

**ファイル**: ex14\_3\_generic\_macros.c

typeof演算子を使った汎用マクロの実装例。

### 学習ポイント：

* typeof演算子による型の取得
* 型安全なマクロの作成
* Statement Expression（GCC拡張）の活用

## 演習14-4: nullptr安全プログラミング

**ファイル**: ex14\_4\_safe\_linkedlist.c

nullptrを使った安全なリンクリスト実装例。

### 学習ポイント：

* nullptrによる明示的なポインタチェック
* エラーハンドリングのベストプラクティス
* メモリ管理の安全性

## 演習14-5: C23総合演習

**ファイル**: ex14\_5\_config\_manager.c

C23の複数機能を組み合わせた設定管理システムの実装例。

### 学習ポイント：

* 複数のC23機能の統合
* 実用的なプログラム設計
* 可読性と保守性の両立

## 解答例の実行方法

# 個別の解答例をコンパイル・実行  
make ex14\_1\_status\_system  
./solutions/ex14\_1\_status\_system  
  
# すべての解答例を実行  
make run-solutions

## 注意事項

* これらは解答例の一つです。他にも様々な実装方法があります
* C23のサポート状況により、一部の機能が動作しない可能性があります
* エラーハンドリングは教育目的で詳細に実装しています ### ex14\_1\_status\_system.c

/\*\*  
 \* 演習14-1: bool型の活用  
 \* システム状態管理プログラム  
 \*   
 \* C23のbool型を使用して、システムの各種状態を管理し、  
 \* 状態の組み合わせによる判定ロジックを実装します。  
 \*/  
  
#include <stdio.h>  
  
/\* システム状態を管理する構造体 \*/  
typedef struct {  
 bool is\_connected; /\* ネットワーク接続状態 \*/  
 bool is\_authenticated; /\* 認証状態 \*/  
 bool has\_permission; /\* アクセス権限 \*/  
 bool is\_busy; /\* ビジー状態 \*/  
} SystemStatus;  
  
/\* 状態を初期化 \*/  
void init\_status(SystemStatus \*status)  
{  
 status->is\_connected = false;  
 status->is\_authenticated = false;  
 status->has\_permission = false;  
 status->is\_busy = false;  
}  
  
/\* 状態を表示 \*/  
void display\_status(const SystemStatus \*status)  
{  
 printf("=== システム状態 ===\n");  
 printf("接続状態: %s\n", status->is\_connected ? "接続中" : "未接続");  
 printf("認証状態: %s\n", status->is\_authenticated ? "認証済み" : "未認証");  
 printf("権限状態: %s\n", status->has\_permission ? "権限あり" : "権限なし");  
 printf("動作状態: %s\n", status->is\_busy ? "ビジー" : "アイドル");  
 printf("==================\n");  
}  
  
/\* システムが利用可能かチェック \*/  
bool is\_system\_available(const SystemStatus \*status)  
{  
 /\* 接続済み、認証済み、権限あり、かつビジーでない場合に利用可能 \*/  
 return status->is\_connected &&   
 status->is\_authenticated &&   
 status->has\_permission &&   
 !status->is\_busy;  
}  
  
/\* セキュリティチェック \*/  
bool check\_security(const SystemStatus \*status)  
{  
 /\* 接続済みかつ認証済みの場合のみセキュア \*/  
 return status->is\_connected && status->is\_authenticated;  
}  
  
/\* 接続を試行 \*/  
bool connect\_system(SystemStatus \*status)  
{  
 if (status->is\_connected) {  
 printf("既に接続されています\n");  
 return false;  
 }  
   
 printf("システムに接続中...\n");  
 status->is\_connected = true;  
 printf("接続成功\n");  
 return true;  
}  
  
/\* 認証を試行 \*/  
bool authenticate\_system(SystemStatus \*status, const char \*username)  
{  
 if (!status->is\_connected) {  
 printf("エラー: 先に接続してください\n");  
 return false;  
 }  
   
 if (status->is\_authenticated) {  
 printf("既に認証済みです\n");  
 return false;  
 }  
   
 printf("ユーザー '%s' で認証中...\n", username);  
 status->is\_authenticated = true;  
   
 /\* 特定のユーザーには自動的に権限を付与 \*/  
 if (username[0] == 'a' && username[1] == 'd' &&   
 username[2] == 'm' && username[3] == 'i' &&   
 username[4] == 'n' && username[5] == '\0') {  
 status->has\_permission = true;  
 printf("管理者権限を付与しました\n");  
 }  
   
 printf("認証成功\n");  
 return true;  
}  
  
/\* タスクを実行 \*/  
bool execute\_task(SystemStatus \*status, const char \*task\_name)  
{  
 if (!is\_system\_available(status)) {  
 printf("エラー: システムが利用できません\n");  
 display\_status(status);  
 return false;  
 }  
   
 status->is\_busy = true;  
 printf("タスク '%s' を実行中...\n", task\_name);  
   
 /\* タスク実行のシミュレーション \*/  
 for (int i = 0; i < 3; i++) {  
 printf(".");  
 }  
 printf(" 完了\n");  
   
 status->is\_busy = false;  
 return true;  
}  
  
/\* 切断 \*/  
void disconnect\_system(SystemStatus \*status)  
{  
 printf("システムから切断中...\n");  
 init\_status(status); /\* すべての状態をリセット \*/  
 printf("切断完了\n");  
}  
  
int main(void)  
{  
 SystemStatus system;  
   
 /\* システム初期化 \*/  
 init\_status(&system);  
   
 printf("=== bool型を使ったシステム状態管理 ===\n\n");  
   
 /\* 初期状態を表示 \*/  
 display\_status(&system);  
   
 /\* セキュリティチェック（失敗するはず） \*/  
 printf("\nセキュリティチェック: %s\n",   
 check\_security(&system) ? "OK" : "NG");  
   
 /\* 接続試行 \*/  
 printf("\n");  
 connect\_system(&system);  
   
 /\* 再度セキュリティチェック（まだ失敗） \*/  
 printf("\nセキュリティチェック: %s\n",   
 check\_security(&system) ? "OK" : "NG");  
   
 /\* 一般ユーザーで認証 \*/  
 printf("\n");  
 authenticate\_system(&system, "user");  
   
 /\* セキュリティチェック（成功） \*/  
 printf("\nセキュリティチェック: %s\n",   
 check\_security(&system) ? "OK" : "NG");  
   
 /\* タスク実行（権限なしで失敗） \*/  
 printf("\n");  
 execute\_task(&system, "重要なタスク");  
   
 /\* 切断して再接続 \*/  
 printf("\n");  
 disconnect\_system(&system);  
   
 /\* 管理者で再接続 \*/  
 printf("\n");  
 connect\_system(&system);  
 authenticate\_system(&system, "admin");  
   
 /\* 現在の状態 \*/  
 printf("\n");  
 display\_status(&system);  
   
 /\* タスク実行（成功） \*/  
 printf("\n");  
 execute\_task(&system, "重要なタスク");  
   
 /\* 別のタスクも実行 \*/  
 execute\_task(&system, "データベース更新");  
 execute\_task(&system, "ログファイル圧縮");  
   
 /\* 最終的な切断 \*/  
 printf("\n");  
 disconnect\_system(&system);  
   
 return 0;  
}

### ex14\_2\_bit\_flags.c

/\*\*  
 \* 演習14-2: ビット操作と2進数リテラル  
 \* 8ビットフラグ管理システム  
 \*   
 \* C23の2進数リテラルを使用して、ビットフラグの管理システムを実装します。  
 \*/  
  
#include <stdio.h>  
  
/\* ビットフラグの定義（2進数リテラル使用） \*/  
#define FLAG\_READ 0b00000001 /\* 読み取り権限 \*/  
#define FLAG\_WRITE 0b00000010 /\* 書き込み権限 \*/  
#define FLAG\_EXECUTE 0b00000100 /\* 実行権限 \*/  
#define FLAG\_DELETE 0b00001000 /\* 削除権限 \*/  
#define FLAG\_SHARE 0b00010000 /\* 共有権限 \*/  
#define FLAG\_ARCHIVE 0b00100000 /\* アーカイブフラグ \*/  
#define FLAG\_HIDDEN 0b01000000 /\* 隠しファイルフラグ \*/  
#define FLAG\_SYSTEM 0b10000000 /\* システムファイルフラグ \*/  
  
/\* ビット操作マクロ \*/  
#define SET\_BIT(flags, bit) ((flags) |= (bit))  
#define CLEAR\_BIT(flags, bit) ((flags) &= ~(bit))  
#define TOGGLE\_BIT(flags, bit) ((flags) ^= (bit))  
#define CHECK\_BIT(flags, bit) ((flags) & (bit))  
  
/\* フラグの型定義 \*/  
typedef unsigned char FileFlags;  
  
/\* フラグを2進数形式で表示 \*/  
void display\_binary(FileFlags flags)  
{  
 printf("0b");  
 for (int i = 7; i >= 0; i--) {  
 printf("%d", (flags >> i) & 1);  
 }  
}  
  
/\* フラグの状態を詳細表示 \*/  
void display\_flags(FileFlags flags)  
{  
 printf("現在のフラグ: ");  
 display\_binary(flags);  
 printf(" (0x%02X)\n", flags);  
   
 printf("権限設定:\n");  
 printf(" 読み取り: %s\n", CHECK\_BIT(flags, FLAG\_READ) ? "○" : "×");  
 printf(" 書き込み: %s\n", CHECK\_BIT(flags, FLAG\_WRITE) ? "○" : "×");  
 printf(" 実行: %s\n", CHECK\_BIT(flags, FLAG\_EXECUTE) ? "○" : "×");  
 printf(" 削除: %s\n", CHECK\_BIT(flags, FLAG\_DELETE) ? "○" : "×");  
 printf(" 共有: %s\n", CHECK\_BIT(flags, FLAG\_SHARE) ? "○" : "×");  
   
 printf("属性:\n");  
 printf(" アーカイブ: %s\n", CHECK\_BIT(flags, FLAG\_ARCHIVE) ? "○" : "×");  
 printf(" 隠し: %s\n", CHECK\_BIT(flags, FLAG\_HIDDEN) ? "○" : "×");  
 printf(" システム: %s\n", CHECK\_BIT(flags, FLAG\_SYSTEM) ? "○" : "×");  
}  
  
/\* 基本権限の設定（読み取り、書き込み、実行） \*/  
void set\_basic\_permissions(FileFlags \*flags)  
{  
 \*flags = FLAG\_READ | FLAG\_WRITE | FLAG\_EXECUTE;  
 printf("基本権限を設定しました: ");  
 display\_binary(\*flags);  
 printf("\n");  
}  
  
/\* フルアクセス権限の設定 \*/  
void set\_full\_access(FileFlags \*flags)  
{  
 \*flags = FLAG\_READ | FLAG\_WRITE | FLAG\_EXECUTE | FLAG\_DELETE | FLAG\_SHARE;  
 printf("フルアクセス権限を設定しました: ");  
 display\_binary(\*flags);  
 printf("\n");  
}  
  
/\* 読み取り専用に設定 \*/  
void set\_readonly(FileFlags \*flags)  
{  
 \*flags = FLAG\_READ;  
 printf("読み取り専用に設定しました: ");  
 display\_binary(\*flags);  
 printf("\n");  
}  
  
/\* アクセスレベルをチェック \*/  
void check\_access\_level(FileFlags flags)  
{  
 printf("\nアクセスレベル解析:\n");  
   
 /\* 基本的なアクセスパターンをチェック \*/  
 if ((flags & (FLAG\_READ | FLAG\_WRITE | FLAG\_EXECUTE)) ==   
 (FLAG\_READ | FLAG\_WRITE | FLAG\_EXECUTE)) {  
 printf("- フルコントロール可能\n");  
 } else if ((flags & (FLAG\_READ | FLAG\_WRITE)) == (FLAG\_READ | FLAG\_WRITE)) {  
 printf("- 読み書き可能\n");  
 } else if (flags & FLAG\_READ) {  
 printf("- 読み取り専用\n");  
 } else {  
 printf("- アクセス権限なし\n");  
 }  
   
 /\* 特殊な組み合わせをチェック \*/  
 if ((flags & (FLAG\_SYSTEM | FLAG\_HIDDEN)) == (FLAG\_SYSTEM | FLAG\_HIDDEN)) {  
 printf("- 保護されたシステムファイル\n");  
 }  
   
 if (!(flags & FLAG\_SHARE) && (flags & FLAG\_WRITE)) {  
 printf("- 排他的書き込みモード\n");  
 }  
}  
  
/\* ビット演算のデモンストレーション \*/  
void demonstrate\_bit\_operations(void)  
{  
 FileFlags flags = 0;  
   
 printf("\n=== ビット演算デモ ===\n");  
   
 /\* 個別ビットの設定 \*/  
 printf("\n1. ビットを1つずつ設定:\n");  
 SET\_BIT(flags, FLAG\_READ);  
 printf(" READ設定後: ");  
 display\_binary(flags);  
 printf("\n");  
   
 SET\_BIT(flags, FLAG\_WRITE);  
 printf(" WRITE設定後: ");  
 display\_binary(flags);  
 printf("\n");  
   
 SET\_BIT(flags, FLAG\_HIDDEN);  
 printf(" HIDDEN設定後: ");  
 display\_binary(flags);  
 printf("\n");  
   
 /\* ビットのクリア \*/  
 printf("\n2. ビットをクリア:\n");  
 CLEAR\_BIT(flags, FLAG\_WRITE);  
 printf(" WRITEクリア後: ");  
 display\_binary(flags);  
 printf("\n");  
   
 /\* ビットのトグル \*/  
 printf("\n3. ビットをトグル:\n");  
 printf(" EXECUTE トグル前: ");  
 display\_binary(flags);  
 printf("\n");  
 TOGGLE\_BIT(flags, FLAG\_EXECUTE);  
 printf(" EXECUTE トグル後: ");  
 display\_binary(flags);  
 printf("\n");  
 TOGGLE\_BIT(flags, FLAG\_EXECUTE);  
 printf(" 再度トグル後: ");  
 display\_binary(flags);  
 printf("\n");  
   
 /\* 複数ビットの一括操作 \*/  
 printf("\n4. 複数ビットの一括設定:\n");  
 flags |= (FLAG\_ARCHIVE | FLAG\_SYSTEM);  
 printf(" ARCHIVE|SYSTEM: ");  
 display\_binary(flags);  
 printf("\n");  
}  
  
int main(void)  
{  
 FileFlags file\_flags = 0;  
   
 printf("=== 2進数リテラルを使ったビットフラグ管理 ===\n\n");  
   
 /\* 初期状態 \*/  
 printf("初期状態:\n");  
 display\_flags(file\_flags);  
   
 /\* 基本権限の設定 \*/  
 printf("\n--- 基本権限の設定 ---\n");  
 set\_basic\_permissions(&file\_flags);  
 display\_flags(file\_flags);  
 check\_access\_level(file\_flags);  
   
 /\* 属性の追加 \*/  
 printf("\n--- 属性の追加 ---\n");  
 SET\_BIT(file\_flags, FLAG\_ARCHIVE);  
 printf("アーカイブ属性を追加\n");  
 display\_flags(file\_flags);  
   
 /\* フルアクセスに変更 \*/  
 printf("\n--- フルアクセスに変更 ---\n");  
 set\_full\_access(&file\_flags);  
 display\_flags(file\_flags);  
 check\_access\_level(file\_flags);  
   
 /\* システムファイルとして設定 \*/  
 printf("\n--- システムファイル設定 ---\n");  
 SET\_BIT(file\_flags, FLAG\_SYSTEM);  
 SET\_BIT(file\_flags, FLAG\_HIDDEN);  
 CLEAR\_BIT(file\_flags, FLAG\_DELETE); /\* 削除不可に \*/  
 display\_flags(file\_flags);  
 check\_access\_level(file\_flags);  
   
 /\* 読み取り専用に変更 \*/  
 printf("\n--- 読み取り専用に変更 ---\n");  
 set\_readonly(&file\_flags);  
 display\_flags(file\_flags);  
 check\_access\_level(file\_flags);  
   
 /\* ビット演算のデモ \*/  
 demonstrate\_bit\_operations();  
   
 /\* 実用例: 権限のマスク処理 \*/  
 printf("\n\n=== 権限マスクの例 ===\n");  
 FileFlags user\_request = FLAG\_READ | FLAG\_WRITE | FLAG\_DELETE;  
 FileFlags allowed\_mask = FLAG\_READ | FLAG\_WRITE; /\* 削除は許可しない \*/  
 FileFlags granted = user\_request & allowed\_mask;  
   
 printf("要求された権限: ");  
 display\_binary(user\_request);  
 printf("\n許可マスク: ");  
 display\_binary(allowed\_mask);  
 printf("\n付与された権限: ");  
 display\_binary(granted);  
 printf("\n");  
   
 return 0;  
}

### ex14\_3\_generic\_macros.c

/\*\*  
 \* 演習14-3: typeof演算子の応用  
 \* 汎用マクロの実装  
 \*   
 \* C23のtypeof演算子を使用して、型安全な汎用マクロを実装します。  
 \*/  
  
#include <stdio.h>  
  
/\* 配列の要素数を取得 \*/  
#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))  
  
/\* 型安全な最大値マクロ \*/  
#define ARRAY\_MAX(arr, size) ({ \  
 typeof(arr[0]) \_max = arr[0]; \  
 for (size\_t \_i = 1; \_i < (size); \_i++) { \  
 if (arr[\_i] > \_max) { \  
 \_max = arr[\_i]; \  
 } \  
 } \  
 \_max; \  
})  
  
/\* 型安全な最小値マクロ \*/  
#define ARRAY\_MIN(arr, size) ({ \  
 typeof(arr[0]) \_min = arr[0]; \  
 for (size\_t \_i = 1; \_i < (size); \_i++) { \  
 if (arr[\_i] < \_min) { \  
 \_min = arr[\_i]; \  
 } \  
 } \  
 \_min; \  
})  
  
/\* 型安全な配列要素の合計マクロ \*/  
#define ARRAY\_SUM(arr, size) ({ \  
 typeof(arr[0]) \_sum = 0; \  
 for (size\_t \_i = 0; \_i < (size); \_i++) { \  
 \_sum += arr[\_i]; \  
 } \  
 \_sum; \  
})  
  
/\* 型安全な変数交換マクロ \*/  
#define SWAP\_SAFE(a, b) do { \  
 typeof(a) \_temp = (a); \  
 (a) = (b); \  
 (b) = \_temp; \  
} while(0)  
  
/\* 型安全な絶対値マクロ \*/  
#define ABS\_SAFE(x) ({ \  
 typeof(x) \_x = (x); \  
 \_x < 0 ? -\_x : \_x; \  
})  
  
/\* 型安全なクランプ（範囲制限）マクロ \*/  
#define CLAMP(value, min, max) ({ \  
 typeof(value) \_val = (value); \  
 typeof(value) \_min = (min); \  
 typeof(value) \_max = (max); \  
 \_val < \_min ? \_min : (\_val > \_max ? \_max : \_val); \  
})  
  
/\* 配列の平均を計算（整数版と浮動小数点版を自動選択） \*/  
#define ARRAY\_AVG(arr, size) ({ \  
 typeof(arr[0]) \_sum = ARRAY\_SUM(arr, size); \  
 \_sum / (typeof(arr[0]))(size); \  
})  
  
/\* 配列要素を表示する汎用関数マクロ \*/  
#define PRINT\_ARRAY(arr, size, format) do { \  
 printf("[ "); \  
 for (size\_t \_i = 0; \_i < (size); \_i++) { \  
 printf(format, arr[\_i]); \  
 if (\_i < (size) - 1) printf(", "); \  
 } \  
 printf(" ]\n"); \  
} while(0)  
  
/\* デモンストレーション関数 \*/  
void demonstrate\_int\_arrays(void)  
{  
 printf("=== 整数配列での動作 ===\n");  
   
 int numbers[] = {45, 12, 78, 23, 56, 89, 34, 67};  
 size\_t size = ARRAY\_SIZE(numbers);  
   
 printf("配列: ");  
 PRINT\_ARRAY(numbers, size, "%d");  
   
 int max = ARRAY\_MAX(numbers, size);  
 int min = ARRAY\_MIN(numbers, size);  
 int sum = ARRAY\_SUM(numbers, size);  
 int avg = ARRAY\_AVG(numbers, size);  
   
 printf("最大値: %d\n", max);  
 printf("最小値: %d\n", min);  
 printf("合計: %d\n", sum);  
 printf("平均: %d\n", avg);  
   
 /\* 変数の交換 \*/  
 printf("\n変数の交換:\n");  
 int a = 100, b = 200;  
 printf("交換前: a = %d, b = %d\n", a, b);  
 SWAP\_SAFE(a, b);  
 printf("交換後: a = %d, b = %d\n", a, b);  
   
 /\* 絶対値 \*/  
 int negative = -42;  
 printf("\n絶対値: ABS(%d) = %d\n", negative, ABS\_SAFE(negative));  
   
 /\* クランプ \*/  
 printf("\nクランプ（範囲制限）:\n");  
 for (int val = -5; val <= 15; val += 5) {  
 printf("CLAMP(%d, 0, 10) = %d\n", val, CLAMP(val, 0, 10));  
 }  
}  
  
void demonstrate\_double\_arrays(void)  
{  
 printf("\n=== 浮動小数点配列での動作 ===\n");  
   
 double values[] = {3.14, 2.71, 1.41, 1.73, 2.23, 0.57};  
 size\_t size = ARRAY\_SIZE(values);  
   
 printf("配列: ");  
 PRINT\_ARRAY(values, size, "%.2f");  
   
 double max = ARRAY\_MAX(values, size);  
 double min = ARRAY\_MIN(values, size);  
 double sum = ARRAY\_SUM(values, size);  
 double avg = ARRAY\_AVG(values, size);  
   
 printf("最大値: %.2f\n", max);  
 printf("最小値: %.2f\n", min);  
 printf("合計: %.2f\n", sum);  
 printf("平均: %.2f\n", avg);  
   
 /\* 変数の交換 \*/  
 printf("\n変数の交換:\n");  
 double x = 3.14, y = 2.71;  
 printf("交換前: x = %.2f, y = %.2f\n", x, y);  
 SWAP\_SAFE(x, y);  
 printf("交換後: x = %.2f, y = %.2f\n", x, y);  
   
 /\* 絶対値 \*/  
 double neg\_val = -3.14159;  
 printf("\n絶対値: ABS(%.5f) = %.5f\n", neg\_val, ABS\_SAFE(neg\_val));  
}  
  
/\* 複雑な型での動作確認 \*/  
struct Point {  
 int x;  
 int y;  
};  
  
void demonstrate\_complex\_types(void)  
{  
 printf("\n=== 複雑な型での動作 ===\n");  
   
 /\* ポインタの交換 \*/  
 int val1 = 100, val2 = 200;  
 int \*ptr1 = &val1;  
 int \*ptr2 = &val2;  
   
 printf("ポインタの交換:\n");  
 printf("交換前: \*ptr1 = %d, \*ptr2 = %d\n", \*ptr1, \*ptr2);  
 SWAP\_SAFE(ptr1, ptr2);  
 printf("交換後: \*ptr1 = %d, \*ptr2 = %d\n", \*ptr1, \*ptr2);  
   
 /\* 構造体の交換 \*/  
 struct Point p1 = {10, 20};  
 struct Point p2 = {30, 40};  
   
 printf("\n構造体の交換:\n");  
 printf("交換前: p1=(%d,%d), p2=(%d,%d)\n", p1.x, p1.y, p2.x, p2.y);  
 SWAP\_SAFE(p1, p2);  
 printf("交換後: p1=(%d,%d), p2=(%d,%d)\n", p1.x, p1.y, p2.x, p2.y);  
}  
  
/\* 実用的な例: ソート関数での活用 \*/  
#define BUBBLE\_SORT(arr, size) do { \  
 for (size\_t i = 0; i < (size) - 1; i++) { \  
 for (size\_t j = 0; j < (size) - i - 1; j++) { \  
 if (arr[j] > arr[j + 1]) { \  
 SWAP\_SAFE(arr[j], arr[j + 1]); \  
 } \  
 } \  
 } \  
} while(0)  
  
void demonstrate\_sorting(void)  
{  
 printf("\n=== 型安全なソート ===\n");  
   
 /\* 整数配列のソート \*/  
 int int\_arr[] = {64, 34, 25, 12, 22, 11, 90};  
 size\_t int\_size = ARRAY\_SIZE(int\_arr);  
   
 printf("整数配列（ソート前）: ");  
 PRINT\_ARRAY(int\_arr, int\_size, "%d");  
   
 BUBBLE\_SORT(int\_arr, int\_size);  
   
 printf("整数配列（ソート後）: ");  
 PRINT\_ARRAY(int\_arr, int\_size, "%d");  
   
 /\* 浮動小数点配列のソート \*/  
 double double\_arr[] = {64.5, 34.2, 25.8, 12.1, 22.7};  
 size\_t double\_size = ARRAY\_SIZE(double\_arr);  
   
 printf("\n浮動小数点配列（ソート前）: ");  
 PRINT\_ARRAY(double\_arr, double\_size, "%.1f");  
   
 BUBBLE\_SORT(double\_arr, double\_size);  
   
 printf("浮動小数点配列（ソート後）: ");  
 PRINT\_ARRAY(double\_arr, double\_size, "%.1f");  
}  
  
int main(void)  
{  
 printf("=== typeof演算子を使った汎用マクロ ===\n\n");  
   
 /\* 整数配列でのデモ \*/  
 demonstrate\_int\_arrays();  
   
 /\* 浮動小数点配列でのデモ \*/  
 demonstrate\_double\_arrays();  
   
 /\* 複雑な型でのデモ \*/  
 demonstrate\_complex\_types();  
   
 /\* ソートのデモ \*/  
 demonstrate\_sorting();  
   
 return 0;  
}

### ex14\_4\_safe\_linkedlist.c

/\*\*  
 \* 演習14-4: nullptr安全プログラミング  
 \* 安全なリンクリスト実装  
 \*   
 \* C23のnullptrを使用して、NULL安全性を重視したリンクリストを実装します。  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
/\* ノード構造体 \*/  
typedef struct Node {  
 int data;  
 struct Node \*next;  
} Node;  
  
/\* リンクリスト構造体 \*/  
typedef struct {  
 Node \*head;  
 size\_t size;  
} LinkedList;  
  
/\* エラーコード \*/  
typedef enum {  
 LIST\_SUCCESS = 0,  
 LIST\_ERROR\_NULL\_POINTER,  
 LIST\_ERROR\_MEMORY\_ALLOCATION,  
 LIST\_ERROR\_NOT\_FOUND,  
 LIST\_ERROR\_EMPTY\_LIST,  
 LIST\_ERROR\_INVALID\_INDEX  
} ListError;  
  
/\* エラーメッセージを取得 \*/  
const char\* get\_error\_message(ListError error)  
{  
 switch (error) {  
 case LIST\_SUCCESS:  
 return "操作成功";  
 case LIST\_ERROR\_NULL\_POINTER:  
 return "NULLポインタエラー";  
 case LIST\_ERROR\_MEMORY\_ALLOCATION:  
 return "メモリ割り当てエラー";  
 case LIST\_ERROR\_NOT\_FOUND:  
 return "要素が見つかりません";  
 case LIST\_ERROR\_EMPTY\_LIST:  
 return "リストが空です";  
 case LIST\_ERROR\_INVALID\_INDEX:  
 return "無効なインデックス";  
 default:  
 return "未知のエラー";  
 }  
}  
  
/\* リストを初期化 \*/  
ListError list\_init(LinkedList \*\*list)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 \*list = (LinkedList \*)malloc(sizeof(LinkedList));  
 if (\*list == nullptr) {  
 return LIST\_ERROR\_MEMORY\_ALLOCATION;  
 }  
   
 (\*list)->head = nullptr;  
 (\*list)->size = 0;  
   
 return LIST\_SUCCESS;  
}  
  
/\* ノードを作成 \*/  
static Node\* create\_node(int data)  
{  
 Node \*new\_node = (Node \*)malloc(sizeof(Node));  
 if (new\_node != nullptr) {  
 new\_node->data = data;  
 new\_node->next = nullptr;  
 }  
 return new\_node;  
}  
  
/\* リストの先頭に追加 \*/  
ListError list\_push\_front(LinkedList \*list, int data)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 Node \*new\_node = create\_node(data);  
 if (new\_node == nullptr) {  
 return LIST\_ERROR\_MEMORY\_ALLOCATION;  
 }  
   
 new\_node->next = list->head;  
 list->head = new\_node;  
 list->size++;  
   
 return LIST\_SUCCESS;  
}  
  
/\* リストの末尾に追加 \*/  
ListError list\_push\_back(LinkedList \*list, int data)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 Node \*new\_node = create\_node(data);  
 if (new\_node == nullptr) {  
 return LIST\_ERROR\_MEMORY\_ALLOCATION;  
 }  
   
 if (list->head == nullptr) {  
 list->head = new\_node;  
 } else {  
 Node \*current = list->head;  
 while (current->next != nullptr) {  
 current = current->next;  
 }  
 current->next = new\_node;  
 }  
   
 list->size++;  
 return LIST\_SUCCESS;  
}  
  
/\* 指定位置に挿入 \*/  
ListError list\_insert(LinkedList \*list, size\_t index, int data)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 if (index > list->size) {  
 return LIST\_ERROR\_INVALID\_INDEX;  
 }  
   
 if (index == 0) {  
 return list\_push\_front(list, data);  
 }  
   
 Node \*new\_node = create\_node(data);  
 if (new\_node == nullptr) {  
 return LIST\_ERROR\_MEMORY\_ALLOCATION;  
 }  
   
 Node \*current = list->head;  
 for (size\_t i = 0; i < index - 1; i++) {  
 current = current->next;  
 }  
   
 new\_node->next = current->next;  
 current->next = new\_node;  
 list->size++;  
   
 return LIST\_SUCCESS;  
}  
  
/\* 値を検索 \*/  
ListError list\_find(const LinkedList \*list, int data, size\_t \*index)  
{  
 if (list == nullptr || index == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 if (list->head == nullptr) {  
 return LIST\_ERROR\_EMPTY\_LIST;  
 }  
   
 Node \*current = list->head;  
 size\_t pos = 0;  
   
 while (current != nullptr) {  
 if (current->data == data) {  
 \*index = pos;  
 return LIST\_SUCCESS;  
 }  
 current = current->next;  
 pos++;  
 }  
   
 return LIST\_ERROR\_NOT\_FOUND;  
}  
  
/\* 先頭から削除 \*/  
ListError list\_pop\_front(LinkedList \*list, int \*data)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 if (list->head == nullptr) {  
 return LIST\_ERROR\_EMPTY\_LIST;  
 }  
   
 Node \*temp = list->head;  
 if (data != nullptr) {  
 \*data = temp->data;  
 }  
   
 list->head = list->head->next;  
 free(temp);  
 list->size--;  
   
 return LIST\_SUCCESS;  
}  
  
/\* 値で削除 \*/  
ListError list\_remove\_value(LinkedList \*list, int data)  
{  
 if (list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 if (list->head == nullptr) {  
 return LIST\_ERROR\_EMPTY\_LIST;  
 }  
   
 /\* 先頭要素の場合 \*/  
 if (list->head->data == data) {  
 return list\_pop\_front(list, nullptr);  
 }  
   
 /\* それ以外の要素 \*/  
 Node \*current = list->head;  
 while (current->next != nullptr && current->next->data != data) {  
 current = current->next;  
 }  
   
 if (current->next == nullptr) {  
 return LIST\_ERROR\_NOT\_FOUND;  
 }  
   
 Node \*temp = current->next;  
 current->next = temp->next;  
 free(temp);  
 list->size--;  
   
 return LIST\_SUCCESS;  
}  
  
/\* リストを表示 \*/  
void list\_display(const LinkedList \*list)  
{  
 if (list == nullptr) {  
 printf("Error: リストがnullptrです\n");  
 return;  
 }  
   
 printf("リスト（要素数: %zu）: ", list->size);  
   
 if (list->head == nullptr) {  
 printf("空\n");  
 return;  
 }  
   
 printf("[ ");  
 Node \*current = list->head;  
 while (current != nullptr) {  
 printf("%d", current->data);  
 if (current->next != nullptr) {  
 printf(" -> ");  
 }  
 current = current->next;  
 }  
 printf(" ]\n");  
}  
  
/\* リストを解放 \*/  
ListError list\_destroy(LinkedList \*\*list)  
{  
 if (list == nullptr || \*list == nullptr) {  
 return LIST\_ERROR\_NULL\_POINTER;  
 }  
   
 Node \*current = (\*list)->head;  
 while (current != nullptr) {  
 Node \*temp = current;  
 current = current->next;  
 free(temp);  
 }  
   
 free(\*list);  
 \*list = nullptr;  
   
 return LIST\_SUCCESS;  
}  
  
/\* エラーハンドリングのデモ \*/  
void demonstrate\_error\_handling(void)  
{  
 printf("=== エラーハンドリングのデモ ===\n");  
   
 LinkedList \*list = nullptr;  
 ListError error;  
   
 /\* NULLポインタでの操作を試行 \*/  
 error = list\_push\_front(nullptr, 10);  
 printf("NULLリストへの追加: %s\n", get\_error\_message(error));  
   
 /\* リストを初期化 \*/  
 error = list\_init(&list);  
 printf("リスト初期化: %s\n", get\_error\_message(error));  
   
 /\* 空のリストから削除を試行 \*/  
 int value;  
 error = list\_pop\_front(list, &value);  
 printf("空リストからの削除: %s\n", get\_error\_message(error));  
   
 /\* 存在しない要素の検索 \*/  
 list\_push\_back(list, 10);  
 size\_t index;  
 error = list\_find(list, 20, &index);  
 printf("存在しない要素の検索: %s\n", get\_error\_message(error));  
   
 /\* 無効なインデックスへの挿入 \*/  
 error = list\_insert(list, 10, 30);  
 printf("無効なインデックスへの挿入: %s\n", get\_error\_message(error));  
   
 list\_destroy(&list);  
}  
  
/\* 通常操作のデモ \*/  
void demonstrate\_normal\_operations(void)  
{  
 printf("\n=== 通常操作のデモ ===\n");  
   
 LinkedList \*list = nullptr;  
 ListError error;  
   
 /\* リストの作成と初期化 \*/  
 error = list\_init(&list);  
 if (error != LIST\_SUCCESS) {  
 printf("初期化エラー: %s\n", get\_error\_message(error));  
 return;  
 }  
   
 printf("空のリスト: ");  
 list\_display(list);  
   
 /\* 要素の追加 \*/  
 printf("\n要素を追加:\n");  
 list\_push\_front(list, 10);  
 list\_display(list);  
   
 list\_push\_back(list, 20);  
 list\_display(list);  
   
 list\_push\_front(list, 5);  
 list\_display(list);  
   
 list\_insert(list, 2, 15);  
 list\_display(list);  
   
 /\* 検索 \*/  
 printf("\n検索テスト:\n");  
 size\_t index;  
 error = list\_find(list, 15, &index);  
 if (error == LIST\_SUCCESS) {  
 printf("値15はインデックス%zuにあります\n", index);  
 }  
   
 /\* 削除 \*/  
 printf("\n削除テスト:\n");  
 int removed;  
 list\_pop\_front(list, &removed);  
 printf("先頭から%dを削除\n", removed);  
 list\_display(list);  
   
 list\_remove\_value(list, 15);  
 printf("値15を削除\n");  
 list\_display(list);  
   
 /\* メモリ解放 \*/  
 list\_destroy(&list);  
   
 /\* 解放後のポインタチェック \*/  
 if (list == nullptr) {  
 printf("\nリストは正常に解放されました\n");  
 }  
}  
  
/\* ストレステスト \*/  
void stress\_test(void)  
{  
 printf("\n=== ストレステスト ===\n");  
   
 LinkedList \*list = nullptr;  
 list\_init(&list);  
   
 /\* 大量のデータを追加 \*/  
 printf("1000個の要素を追加...\n");  
 for (int i = 0; i < 1000; i++) {  
 if (i % 2 == 0) {  
 list\_push\_front(list, i);  
 } else {  
 list\_push\_back(list, i);  
 }  
 }  
 printf("要素数: %zu\n", list->size);  
   
 /\* ランダムに削除 \*/  
 printf("500個の要素を削除...\n");  
 for (int i = 0; i < 500; i++) {  
 int value;  
 list\_pop\_front(list, &value);  
 }  
 printf("残り要素数: %zu\n", list->size);  
   
 /\* メモリリークチェック \*/  
 list\_destroy(&list);  
 printf("メモリを解放しました\n");  
}  
  
int main(void)  
{  
 printf("=== nullptr を使った安全なリンクリスト ===\n\n");  
   
 /\* エラーハンドリングのデモ \*/  
 demonstrate\_error\_handling();  
   
 /\* 通常操作のデモ \*/  
 demonstrate\_normal\_operations();  
   
 /\* ストレステスト \*/  
 stress\_test();  
   
 printf("\nすべてのテストが完了しました\n");  
   
 return 0;  
}

### ex14\_5\_config\_manager.c

/\*\*  
 \* 演習14-5: C23総合演習  
 \* 設定管理システム  
 \*   
 \* C23の複数の新機能を組み合わせた設定管理システムを実装します。  
 \* - bool型で各種設定のON/OFF  
 \* - 2進数リテラルでフラグ管理  
 \* - typeof演算子で型安全な操作  
 \* - nullptrで安全なポインタ操作  
 \*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
/\* 設定フラグ（2進数リテラル使用） \*/  
#define CONFIG\_DEBUG 0b00000001  
#define CONFIG\_VERBOSE 0b00000010  
#define CONFIG\_LOGGING 0b00000100  
#define CONFIG\_NETWORKING 0b00001000  
#define CONFIG\_ENCRYPTION 0b00010000  
#define CONFIG\_COMPRESSION 0b00100000  
#define CONFIG\_BACKUP 0b01000000  
#define CONFIG\_READONLY 0b10000000  
  
/\* 設定値の型定義 \*/  
typedef enum {  
 TYPE\_BOOL,  
 TYPE\_INT,  
 TYPE\_DOUBLE,  
 TYPE\_STRING  
} ConfigType;  
  
/\* 設定項目構造体 \*/  
typedef struct ConfigItem {  
 char key[64];  
 ConfigType type;  
 union {  
 bool bool\_value;  
 int int\_value;  
 double double\_value;  
 char \*string\_value;  
 } value;  
 struct ConfigItem \*next;  
} ConfigItem;  
  
/\* 設定マネージャー構造体 \*/  
typedef struct {  
 ConfigItem \*items;  
 unsigned char flags;  
 bool is\_modified;  
 char \*filename;  
} ConfigManager;  
  
/\* 型安全な値の設定マクロ \*/  
#define SET\_CONFIG\_VALUE(item, val) do { \  
 typeof(val) \_val = (val); \  
 if (item != nullptr) { \  
 item->value.int\_value = (int)\_val; \  
 } \  
} while(0)  
  
/\* 型安全な最大値取得マクロ \*/  
#define GET\_MAX\_CONFIG(a, b) ({ \  
 typeof(a) \_a = (a); \  
 typeof(b) \_b = (b); \  
 \_a > \_b ? \_a : \_b; \  
})  
  
/\* 設定マネージャーを作成 \*/  
ConfigManager\* config\_create(const char \*filename)  
{  
 ConfigManager \*config = (ConfigManager \*)malloc(sizeof(ConfigManager));  
 if (config == nullptr) {  
 return nullptr;  
 }  
   
 config->items = nullptr;  
 config->flags = 0;  
 config->is\_modified = false;  
   
 if (filename != nullptr) {  
 config->filename = (char \*)malloc(strlen(filename) + 1);  
 if (config->filename != nullptr) {  
 strcpy(config->filename, filename);  
 }  
 } else {  
 config->filename = nullptr;  
 }  
   
 return config;  
}  
  
/\* 設定項目を追加（bool型） \*/  
bool config\_add\_bool(ConfigManager \*config, const char \*key, bool value)  
{  
 if (config == nullptr || key == nullptr) {  
 return false;  
 }  
   
 ConfigItem \*item = (ConfigItem \*)malloc(sizeof(ConfigItem));  
 if (item == nullptr) {  
 return false;  
 }  
   
 strncpy(item->key, key, sizeof(item->key) - 1);  
 item->key[sizeof(item->key) - 1] = '\0';  
 item->type = TYPE\_BOOL;  
 item->value.bool\_value = value;  
 item->next = config->items;  
 config->items = item;  
 config->is\_modified = true;  
   
 return true;  
}  
  
/\* 設定項目を追加（int型） \*/  
bool config\_add\_int(ConfigManager \*config, const char \*key, int value)  
{  
 if (config == nullptr || key == nullptr) {  
 return false;  
 }  
   
 ConfigItem \*item = (ConfigItem \*)malloc(sizeof(ConfigItem));  
 if (item == nullptr) {  
 return false;  
 }  
   
 strncpy(item->key, key, sizeof(item->key) - 1);  
 item->key[sizeof(item->key) - 1] = '\0';  
 item->type = TYPE\_INT;  
 item->value.int\_value = value;  
 item->next = config->items;  
 config->items = item;  
 config->is\_modified = true;  
   
 return true;  
}  
  
/\* 設定項目を追加（double型） \*/  
bool config\_add\_double(ConfigManager \*config, const char \*key, double value)  
{  
 if (config == nullptr || key == nullptr) {  
 return false;  
 }  
   
 ConfigItem \*item = (ConfigItem \*)malloc(sizeof(ConfigItem));  
 if (item == nullptr) {  
 return false;  
 }  
   
 strncpy(item->key, key, sizeof(item->key) - 1);  
 item->key[sizeof(item->key) - 1] = '\0';  
 item->type = TYPE\_DOUBLE;  
 item->value.double\_value = value;  
 item->next = config->items;  
 config->items = item;  
 config->is\_modified = true;  
   
 return true;  
}  
  
/\* 設定項目を追加（文字列） \*/  
bool config\_add\_string(ConfigManager \*config, const char \*key, const char \*value)  
{  
 if (config == nullptr || key == nullptr || value == nullptr) {  
 return false;  
 }  
   
 ConfigItem \*item = (ConfigItem \*)malloc(sizeof(ConfigItem));  
 if (item == nullptr) {  
 return false;  
 }  
   
 strncpy(item->key, key, sizeof(item->key) - 1);  
 item->key[sizeof(item->key) - 1] = '\0';  
 item->type = TYPE\_STRING;  
 item->value.string\_value = (char \*)malloc(strlen(value) + 1);  
 if (item->value.string\_value == nullptr) {  
 free(item);  
 return false;  
 }  
 strcpy(item->value.string\_value, value);  
 item->next = config->items;  
 config->items = item;  
 config->is\_modified = true;  
   
 return true;  
}  
  
/\* 設定項目を検索 \*/  
ConfigItem\* config\_find(ConfigManager \*config, const char \*key)  
{  
 if (config == nullptr || key == nullptr) {  
 return nullptr;  
 }  
   
 ConfigItem \*current = config->items;  
 while (current != nullptr) {  
 if (strcmp(current->key, key) == 0) {  
 return current;  
 }  
 current = current->next;  
 }  
   
 return nullptr;  
}  
  
/\* フラグを設定 \*/  
void config\_set\_flag(ConfigManager \*config, unsigned char flag, bool enable)  
{  
 if (config == nullptr) {  
 return;  
 }  
   
 if (enable) {  
 config->flags |= flag;  
 } else {  
 config->flags &= ~flag;  
 }  
 config->is\_modified = true;  
}  
  
/\* フラグをチェック \*/  
bool config\_check\_flag(ConfigManager \*config, unsigned char flag)  
{  
 if (config == nullptr) {  
 return false;  
 }  
   
 return (config->flags & flag) != 0;  
}  
  
/\* 設定を表示 \*/  
void config\_display(ConfigManager \*config)  
{  
 if (config == nullptr) {  
 printf("Error: 設定マネージャーがnullptrです\n");  
 return;  
 }  
   
 printf("=== 設定情報 ===\n");  
 printf("ファイル: %s\n", config->filename != nullptr ? config->filename : "(なし)");  
 printf("変更状態: %s\n", config->is\_modified ? "変更あり" : "変更なし");  
   
 /\* フラグ状態を2進数で表示 \*/  
 printf("フラグ: 0b");  
 for (int i = 7; i >= 0; i--) {  
 printf("%d", (config->flags >> i) & 1);  
 }  
 printf("\n");  
   
 /\* 個別フラグの状態 \*/  
 printf(" デバッグ: %s\n", config\_check\_flag(config, CONFIG\_DEBUG) ? "ON" : "OFF");  
 printf(" 詳細出力: %s\n", config\_check\_flag(config, CONFIG\_VERBOSE) ? "ON" : "OFF");  
 printf(" ログ記録: %s\n", config\_check\_flag(config, CONFIG\_LOGGING) ? "ON" : "OFF");  
 printf(" ネットワーク: %s\n", config\_check\_flag(config, CONFIG\_NETWORKING) ? "ON" : "OFF");  
 printf(" 暗号化: %s\n", config\_check\_flag(config, CONFIG\_ENCRYPTION) ? "ON" : "OFF");  
 printf(" 圧縮: %s\n", config\_check\_flag(config, CONFIG\_COMPRESSION) ? "ON" : "OFF");  
 printf(" バックアップ: %s\n", config\_check\_flag(config, CONFIG\_BACKUP) ? "ON" : "OFF");  
 printf(" 読み取り専用: %s\n", config\_check\_flag(config, CONFIG\_READONLY) ? "ON" : "OFF");  
   
 /\* 設定項目 \*/  
 printf("\n設定項目:\n");  
 ConfigItem \*current = config->items;  
 while (current != nullptr) {  
 printf(" %s = ", current->key);  
 switch (current->type) {  
 case TYPE\_BOOL:  
 printf("%s", current->value.bool\_value ? "true" : "false");  
 break;  
 case TYPE\_INT:  
 printf("%d", current->value.int\_value);  
 break;  
 case TYPE\_DOUBLE:  
 printf("%.2f", current->value.double\_value);  
 break;  
 case TYPE\_STRING:  
 printf("\"%s\"", current->value.string\_value);  
 break;  
 }  
 printf(" (%s型)\n",   
 current->type == TYPE\_BOOL ? "bool" :  
 current->type == TYPE\_INT ? "int" :  
 current->type == TYPE\_DOUBLE ? "double" : "string");  
 current = current->next;  
 }  
}  
  
/\* 設定を破棄 \*/  
void config\_destroy(ConfigManager \*\*config)  
{  
 if (config == nullptr || \*config == nullptr) {  
 return;  
 }  
   
 /\* 設定項目を解放 \*/  
 ConfigItem \*current = (\*config)->items;  
 while (current != nullptr) {  
 ConfigItem \*temp = current;  
 current = current->next;  
   
 if (temp->type == TYPE\_STRING && temp->value.string\_value != nullptr) {  
 free(temp->value.string\_value);  
 }  
 free(temp);  
 }  
   
 /\* ファイル名を解放 \*/  
 if ((\*config)->filename != nullptr) {  
 free((\*config)->filename);  
 }  
   
 /\* 設定マネージャーを解放 \*/  
 free(\*config);  
 \*config = nullptr;  
}  
  
/\* 実用的な使用例 \*/  
void demonstrate\_practical\_use(void)  
{  
 printf("=== 実用的な設定管理の例 ===\n\n");  
   
 /\* 設定マネージャーを作成 \*/  
 ConfigManager \*app\_config = config\_create("app\_config.ini");  
 if (app\_config == nullptr) {  
 printf("設定マネージャーの作成に失敗\n");  
 return;  
 }  
   
 /\* アプリケーション設定を追加 \*/  
 config\_add\_string(app\_config, "app\_name", "C23 Demo Application");  
 config\_add\_string(app\_config, "version", "1.0.0");  
 config\_add\_int(app\_config, "max\_connections", 100);  
 config\_add\_int(app\_config, "timeout", 30);  
 config\_add\_double(app\_config, "cache\_size\_mb", 256.5);  
 config\_add\_bool(app\_config, "auto\_save", true);  
 config\_add\_bool(app\_config, "show\_tips", false);  
   
 /\* フラグを設定 \*/  
 config\_set\_flag(app\_config, CONFIG\_DEBUG | CONFIG\_VERBOSE, true);  
 config\_set\_flag(app\_config, CONFIG\_LOGGING | CONFIG\_BACKUP, true);  
 config\_set\_flag(app\_config, CONFIG\_ENCRYPTION, false);  
   
 /\* 設定を表示 \*/  
 config\_display(app\_config);  
   
 /\* 設定の変更 \*/  
 printf("\n\n--- 設定を変更 ---\n");  
 ConfigItem \*item = config\_find(app\_config, "max\_connections");  
 if (item != nullptr) {  
 item->value.int\_value = 200;  
 printf("max\_connectionsを200に変更しました\n");  
 }  
   
 /\* セキュリティモードを有効化 \*/  
 printf("\nセキュリティモードを有効化...\n");  
 config\_set\_flag(app\_config, CONFIG\_ENCRYPTION, true);  
 config\_set\_flag(app\_config, CONFIG\_READONLY, true);  
   
 /\* 最終状態を表示 \*/  
 printf("\n");  
 config\_display(app\_config);  
   
 /\* クリーンアップ \*/  
 config\_destroy(&app\_config);  
   
 if (app\_config == nullptr) {  
 printf("\n\n設定マネージャーは正常に破棄されました\n");  
 }  
}  
  
/\* デバッグモードのシミュレーション \*/  
void simulate\_debug\_mode(void)  
{  
 printf("\n\n=== デバッグモードのシミュレーション ===\n");  
   
 ConfigManager \*debug\_config = config\_create("debug.conf");  
   
 /\* デバッグ設定 \*/  
 config\_add\_bool(debug\_config, "enable\_assertions", true);  
 config\_add\_int(debug\_config, "log\_level", 3); /\* 0:ERROR, 1:WARN, 2:INFO, 3:DEBUG \*/  
 config\_add\_string(debug\_config, "log\_file", "/tmp/app\_debug.log");  
   
 /\* すべてのデバッグフラグをON \*/  
 config\_set\_flag(debug\_config, CONFIG\_DEBUG | CONFIG\_VERBOSE | CONFIG\_LOGGING, true);  
   
 /\* デバッグ情報を表示 \*/  
 if (config\_check\_flag(debug\_config, CONFIG\_DEBUG)) {  
 printf("デバッグモードが有効です\n");  
   
 ConfigItem \*log\_level = config\_find(debug\_config, "log\_level");  
 if (log\_level != nullptr) {  
 printf("ログレベル: %d\n", log\_level->value.int\_value);  
 }  
   
 ConfigItem \*log\_file = config\_find(debug\_config, "log\_file");  
 if (log\_file != nullptr) {  
 printf("ログファイル: %s\n", log\_file->value.string\_value);  
 }  
 }  
   
 config\_destroy(&debug\_config);  
}  
  
int main(void)  
{  
 printf("=== C23総合演習: 設定管理システム ===\n\n");  
   
 /\* 実用的な使用例 \*/  
 demonstrate\_practical\_use();  
   
 /\* デバッグモードのシミュレーション \*/  
 simulate\_debug\_mode();  
   
 printf("\n\nプログラムを終了します\n");  
   
 return 0;  
}