C言語プログラミング教材 - 演習問題集（上級編）

第12章〜第16章の演習問題

C言語学習教材プロジェクト

2024年

Table of Contents

[はじめに 5](#_Toc202361528)

[前提知識 5](#_Toc202361529)

[演習問題の特徴 5](#_Toc202361530)

[チャレンジ問題 5](#_Toc202361531)

[第12章: ビット操作とビットフィールド - 演習問題 6](#_Toc202361532)

[演習問題の目的 6](#_Toc202361533)

[基礎問題 6](#_Toc202361534)

[n 問題12-1: 基本的なビット操作 6](#_Toc202361535)

[n 問題12-2: フラグ管理システム 7](#_Toc202361536)

[応用問題 8](#_Toc202361537)

[n 問題12-3: ビットフィールドでメモリ節約 8](#_Toc202361538)

[n 問題12-4: RGB色操作 9](#_Toc202361539)

[n 問題12-5: ビットベクタで集合演算 9](#_Toc202361540)

[チャレンジ問題 10](#_Toc202361541)

[n 問題12-6: CRC計算の実装 10](#_Toc202361542)

[n 問題12-7: ビットボードでオセロ 11](#_Toc202361543)

[n 問題12-8: ハフマン符号化（上級） 12](#_Toc202361544)

[n 問題12-9: ブルームフィルタ（上級） 12](#_Toc202361545)

[演習のヒント 12](#_Toc202361546)

[n ビット操作の基本パターン 12](#_Toc202361547)

[n ビットマスクの作成 13](#_Toc202361548)

[n ビットカウント 13](#_Toc202361549)

[提出方法 13](#_Toc202361550)

[評価基準 13](#_Toc202361551)

[第13章: 構造体とポインタ - 演習問題 14](#_Toc202361552)

[演習の目的 14](#_Toc202361553)

[基礎問題 14](#_Toc202361554)

[n 問題13-1: 学生情報管理 14](#_Toc202361555)

[n 問題13-2: 座標計算 15](#_Toc202361556)

[n 問題13-3: 商品管理 16](#_Toc202361557)

[応用問題 17](#_Toc202361558)

[n 問題13-4: 従業員データベース 17](#_Toc202361559)

[n 問題13-5: 図書管理システム 18](#_Toc202361560)

[チャレンジ問題 19](#_Toc202361561)

[n 問題13-6: 成績管理システム 19](#_Toc202361562)

[n 問題13-7: 連結リストの実装 20](#_Toc202361563)

[提出方法 21](#_Toc202361564)

[学習のポイント 21](#_Toc202361565)

[第14章: 関数ポインタ - 演習問題 22](#_Toc202361566)

[演習の目的 22](#_Toc202361567)

[基礎問題 22](#_Toc202361568)

[n 問題14-1: 関数ポインタの基本操作 22](#_Toc202361569)

[n 問題14-2: 関数選択システム 23](#_Toc202361570)

[応用問題 24](#_Toc202361571)

[n 問題14-3: 配列処理のコールバック 24](#_Toc202361572)

[n 問題14-4: 関数ポインタ配列を使った計算機 25](#_Toc202361573)

[チャレンジ問題 26](#_Toc202361574)

[n 問題14-5: ソートアルゴリズム選択システム 26](#_Toc202361575)

[n 問題14-6: イベント駆動システム 26](#_Toc202361576)

[提出方法 27](#_Toc202361577)

[コンパイル例 28](#_Toc202361578)

[n C90準拠 28](#_Toc202361579)

[n C99準拠 28](#_Toc202361580)

[学習のポイント 28](#_Toc202361581)

[n 関数ポインタの基本 28](#_Toc202361582)

[n コールバック関数 28](#_Toc202361583)

[n 実用的なテクニック 28](#_Toc202361584)

[注意事項 28](#_Toc202361585)

[実用的な応用 29](#_Toc202361586)

[次のステップ 29](#_Toc202361587)

[第15章: 複数ファイル・発展技術 - 演習問題 30](#_Toc202361588)

[演習の目的 30](#_Toc202361589)

[基礎問題 30](#_Toc202361590)

[n 問題15-1: プリプロセッサマクロの基本 30](#_Toc202361591)

[n 問題15-2: 安全なメモリ操作マクロ 31](#_Toc202361592)

[n 問題15-3: 基本的なメモリプール 32](#_Toc202361593)

[応用問題 33](#_Toc202361594)

[n 問題15-4: 汎用的なプリプロセッサライブラリ 33](#_Toc202361595)

[n 問題15-5: 高性能メモリアロケーター 34](#_Toc202361596)

[チャレンジ問題 35](#_Toc202361597)

[n 問題15-6: キャッシュフレンドリーなデータ構造 35](#_Toc202361598)

[n 問題15-7: プリプロセッサベースのDSL 36](#_Toc202361599)

[提出方法 37](#_Toc202361600)

[学習のポイント 38](#_Toc202361601)

[第16章: C23の新機能 - 演習問題 39](#_Toc202361602)

[演習の目的 39](#_Toc202361603)

[基礎問題 39](#_Toc202361604)

[n 問題16-1: bool型の活用 39](#_Toc202361605)

[n 問題16-2: ビット操作と2進数リテラル 40](#_Toc202361606)

[応用問題 41](#_Toc202361607)

[n 問題16-3: typeof演算子の応用 41](#_Toc202361608)

[n 問題16-4: nullptr安全プログラミング 42](#_Toc202361609)

[n 問題16-5: デジット区切り記号の活用 43](#_Toc202361610)

[チャレンジ問題 44](#_Toc202361611)

[n 問題16-6: C23総合演習 - 設定管理システム 44](#_Toc202361612)

[n 問題16-7: ビット演算デバッガ 45](#_Toc202361613)

[提出方法 46](#_Toc202361614)

[学習のポイント 47](#_Toc202361615)

[上級編を修了して 47](#_Toc202361616)

[習得したスキル 47](#_Toc202361617)

[n システムレベルプログラミング 47](#_Toc202361618)

[n 高度なデータ構造 47](#_Toc202361619)

[n 関数型プログラミング要素 47](#_Toc202361620)

[n プロジェクト構成 47](#_Toc202361621)

[実務への応用 48](#_Toc202361622)

[継続的な学習 48](#_Toc202361623)

[最後に 48](#_Toc202361624)

# はじめに

本書は、C言語プログラミング教材上級編の演習問題集です。ビット操作、構造体、関数ポインタなど、実務で必要となる高度な技術を実践的に習得するための問題を収録しています。

## 前提知識

上級編の演習問題に取り組む前に、以下の内容を完全に理解していることが必要です：

* ポインタと配列の関係
* 動的メモリ管理
* 関数の設計と実装
* 文字列処理

## 演習問題の特徴

上級編の演習問題は、実務で遭遇する実際の課題に基づいています：

* **システムプログラミング**: ハードウェアに近いレベルの処理
* **データ構造**: 効率的なデータ管理手法
* **設計パターン**: 再利用可能なコード設計

## チャレンジ問題

各章には「チャレンジ問題」が含まれています。これらは発展的な内容で、より深い理解と創造性が求められます。

# 第12章: ビット操作とビットフィールド - 演習問題

## 演習問題の目的

この章では、ビット演算子の使い方、ビットマスクによるフラグ管理、ビットフィールドの活用、そして実践的なビット操作技術を習得します。

## 基礎問題

### 問題12-1: 基本的なビット操作

以下の処理を行う関数を実装してください：

1. 整数の特定ビットをセット/クリア/反転する関数
2. 整数の特定範囲のビットを抽出する関数
3. 整数のビットを左右に循環シフトする関数

/\* 実装すべき関数のプロトタイプ \*/  
void set\_bit(unsigned int \*value, int bit);  
void clear\_bit(unsigned int \*value, int bit);  
void toggle\_bit(unsigned int \*value, int bit);  
int test\_bit(unsigned int value, int bit);  
  
unsigned int extract\_bits(unsigned int value, int start, int count);  
unsigned int rotate\_left(unsigned int value, int shift);  
unsigned int rotate\_right(unsigned int value, int shift);

**要件**

* ビット位置は0から31まで
* 範囲外のビット位置の場合は何もしない
* 循環シフトは32ビットの範囲で行う

**期待される出力例:**

=== 基本的なビット操作 ===  
元の値: 0xA5 (10100101)  
  
ビット3をセット: 0xAD (10101101)  
ビット5をクリア: 0x85 (10000101)  
ビット1を反転: 0xA7 (10100111)  
ビット6のテスト: 1 (セットされている)  
  
ビット2から4ビット抽出: 0x9 (1001)  
左に3ビット循環シフト: 0x2D4 (1010110100)  
右に2ビット循環シフト: 0x40000029 (01000000000000000000000000101001)

### 問題12-2: フラグ管理システム

ゲームキャラクターの状態を管理するフラグシステムを実装してください：

/\* 状態フラグの定義 \*/  
#define STATUS\_ALIVE 0x0001  
#define STATUS\_POISONED 0x0002  
#define STATUS\_PARALYZED 0x0004  
#define STATUS\_CONFUSED 0x0008  
#define STATUS\_SLEEPING 0x0010  
#define STATUS\_BURNING 0x0020  
#define STATUS\_FROZEN 0x0040  
#define STATUS\_INVISIBLE 0x0080  
#define STATUS\_INVINCIBLE 0x0100  
#define STATUS\_BUFFED 0x0200

実装する機能： 1. 複数の状態を同時に設定/解除 2. 特定の状態の組み合わせをチェック 3. 状態の一覧を表示 4. 相反する状態の自動解除（例：凍結と燃焼）

**期待される出力例:**

=== キャラクター状態管理 ===  
初期状態: ALIVE  
  
状態を追加: POISONED, CONFUSED  
現在の状態: ALIVE | POISONED | CONFUSED  
  
BURNINGを追加...  
FROZENが自動的に解除されました  
現在の状態: ALIVE | POISONED | CONFUSED | BURNING  
  
状態チェック:  
- 生存している: はい  
- 毒と混乱の両方: はい  
- 無敵状態: いいえ  
  
全状態一覧:  
[x] ALIVE  
[x] POISONED  
[ ] PARALYZED  
[x] CONFUSED  
[ ] SLEEPING  
[x] BURNING  
[ ] FROZEN  
[ ] INVISIBLE  
[ ] INVINCIBLE  
[ ] BUFFED

## 応用問題

### 問題12-3: ビットフィールドでメモリ節約

学生の成績データを効率的に格納する構造体を設計してください：

/\* 成績データ構造体（ビットフィールド使用） \*/  
struct StudentGrade {  
 unsigned int student\_id : 20; /\* 学生ID（0-1048575） \*/  
 unsigned int subject\_code : 10; /\* 科目コード（0-1023） \*/  
 unsigned int grade : 7; /\* 成績（0-100） \*/  
 unsigned int semester : 2; /\* 学期（1-4） \*/  
 unsigned int year : 7; /\* 年度（0-99, 2000年基準） \*/  
 unsigned int attendance : 1; /\* 出席状況（合格/不合格） \*/  
 unsigned int retake : 1; /\* 再履修フラグ \*/  
};

実装する機能： 1. 成績データの入力と表示 2. 通常の構造体とのサイズ比較 3. 成績の統計情報（平均点、最高点、最低点） 4. 条件に合う学生の検索

**期待される出力例:**

=== 成績データ管理 ===  
構造体サイズ比較:  
- ビットフィールド版: 8 バイト  
- 通常版: 32 バイト  
- 節約率: 75.0%  
  
成績データ入力:  
学生ID: 12345  
科目コード: 101  
成績: 85  
学期: 2  
年度: 24 (2024年)  
出席: 合格  
再履修: いいえ  
  
登録された成績一覧:  
ID 科目 成績 学期 年度 出席 再履修  
12345 101 85 2 2024 ○ -  
12346 101 92 2 2024 ○ -  
12347 101 68 2 2024 × ○  
  
統計情報:  
平均点: 81.7  
最高点: 92 (学生ID: 12346)  
最低点: 68 (学生ID: 12347)

### 問題12-4: RGB色操作

RGB565形式（16ビット）の色を操作するプログラムを作成してください：

/\* RGB565色構造体 \*/  
typedef struct {  
 unsigned int blue : 5;  
 unsigned int green : 6;  
 unsigned int red : 5;  
} RGB565;  
  
/\* または16ビット整数として扱う \*/  
typedef unsigned short RGB565\_int;

実装する機能： 1. RGB888（24ビット）⇔ RGB565の相互変換 2. 色の明度調整（明るく/暗く） 3. グレースケール変換 4. 2つの色の混合（アルファブレンド風）

**期待される出力例:**

=== RGB565色操作 ===  
RGB888 (255, 128, 64) → RGB565: 0xFC20  
RGB565: R=31, G=32, B=0  
  
RGB565 0xFC20 → RGB888: (248, 129, 0)  
（精度損失あり）  
  
明度調整:  
元の色: 0x7BEF (R=15, G=31, B=15)  
50%明るく: 0xBDF7 (R=23, G=47, B=23)  
50%暗く: 0x39E7 (R=7, G=15, B=7)  
  
グレースケール変換:  
カラー: 0xF800 (赤) → グレー: 0x7BEF  
  
色の混合:  
色1: 0xF800 (赤)  
色2: 0x001F (青)  
混合(50:50): 0x780F (紫)

### 問題12-5: ビットベクタで集合演算

ビットベクタを使って集合演算を実装してください：

#define SET\_SIZE 256 /\* 0-255の整数を扱う \*/  
  
typedef struct {  
 unsigned char bits[SET\_SIZE / 8];  
} BitSet;

実装する機能： 1. 集合の初期化、要素の追加/削除 2. 和集合、積集合、差集合、対称差 3. 部分集合の判定 4. 集合の要素数カウント 5. 集合の内容表示

**期待される出力例:**

=== ビットベクタ集合演算 ===  
集合A: {1, 3, 5, 7, 9}  
集合B: {2, 3, 5, 8, 9}  
  
和集合 (A ∪ B): {1, 2, 3, 5, 7, 8, 9}  
積集合 (A ∩ B): {3, 5, 9}  
差集合 (A - B): {1, 7}  
対称差 (A △ B): {1, 2, 7, 8}  
  
集合C: {3, 5}  
C ⊆ A: true (CはAの部分集合)  
C ⊆ B: true (CはBの部分集合)  
  
要素数:  
|A| = 5  
|B| = 5  
|A ∪ B| = 7  
|A ∩ B| = 3

## チャレンジ問題

### 問題12-6: CRC計算の実装

簡単なCRC-8チェックサムを計算する関数を実装してください：

/\* CRC-8計算（多項式: x^8 + x^2 + x + 1） \*/  
unsigned char calculate\_crc8(const unsigned char \*data, int length);

実装内容： 1. ビット単位でCRCを計算する基本実装 2. テーブルを使った高速化実装 3. データの破損検出テスト 4. 性能比較

**期待される出力例:**

=== CRC-8計算 ===  
データ: "Hello"  
CRC-8 (ビット単位): 0x1A  
CRC-8 (テーブル): 0x1A  
  
破損検出テスト:  
元データ: "Hello" (CRC: 0x1A)  
破損データ: "Hallo" (CRC: 0x4D)  
破損が検出されました！  
  
性能比較 (10000回実行):  
ビット単位: 125ms  
テーブル方式: 15ms  
高速化: 8.3倍

### 問題12-7: ビットボードでオセロ

8×8のオセロ盤面をビットボードで表現してください：

typedef struct {  
 unsigned long long black; /\* 黒石の位置 \*/  
 unsigned long long white; /\* 白石の位置 \*/  
} OthelloBoard;

実装する機能： 1. 盤面の初期化と表示 2. 石を置ける位置の判定 3. 石を置いた時の反転処理 4. 勝敗判定

**期待される出力例:**

=== オセロゲーム ===  
 A B C D E F G H  
1 . . . . . . . .  
2 . . . . . . . .  
3 . . . \* . . . .  
4 . . \* O X . . .  
5 . . . X O \* . .  
6 . . . . \* . . .  
7 . . . . . . . .  
8 . . . . . . . .  
  
黒番です  
置ける場所(\*): D3, C4, F5, E6  
  
D3に置きます...  
  
 A B C D E F G H  
1 . . . . . . . .  
2 . . . . . . . .  
3 . . . X . . . .  
4 . . \* X X . . .  
5 . . . X O \* . .  
6 . . . . \* . . .  
7 . . . . . . . .  
8 . . . . . . . .  
  
石の数: 黒=4, 白=1

### 問題12-8: ハフマン符号化（上級）

簡単なハフマン符号化を実装してください：

1. 文字の出現頻度をカウント
2. ハフマン木を構築
3. 各文字のビット列を決定
4. テキストをビット列に圧縮
5. ビット列から元のテキストに復元

### 問題12-9: ブルームフィルタ（上級）

確率的データ構造であるブルームフィルタを実装してください：

1. 複数のハッシュ関数の実装
2. 要素の追加
3. 要素の存在確認（偽陽性あり）
4. 偽陽性率の測定

## 演習のヒント

### ビット操作の基本パターン

/\* nビット目をセット \*/  
value |= (1 << n);  
  
/\* nビット目をクリア \*/  
value &= ~(1 << n);  
  
/\* nビット目を反転 \*/  
value ^= (1 << n);  
  
/\* nビット目をテスト \*/  
if (value & (1 << n)) { /\* ... \*/ }

### ビットマスクの作成

/\* 下位nビットのマスク \*/  
mask = (1 << n) - 1;  
  
/\* startからcountビット分のマスク \*/  
mask = ((1 << count) - 1) << start;

### ビットカウント

/\* Brian Kernighanのアルゴリズム \*/  
int count = 0;  
while (n) {  
 n &= n - 1;  
 count++;  
}

## 提出方法

1. 各演習問題ごとにex12\_1.c、ex12\_2.c…という名前でファイルを作成
2. 動作確認用のmain関数を含める
3. コメントで実装の工夫や考察を記載

## 評価基準

* **正確性**: 要求された機能が正しく動作するか
* **効率性**: ビット操作の利点を活かした実装か
* **可読性**: コードが理解しやすく整理されているか
* **完成度**: エラー処理やエッジケースへの対応

頑張ってください！

# 第13章: 構造体とポインタ - 演習問題

## 演習の目的

* 構造体の定義と使い方を理解する
* 構造体のメンバーアクセス（ドット演算子とアロー演算子）を習得する
* 構造体配列と構造体ポインタの操作を身につける
* ネストした構造体の設計と実装を経験する

## 基礎問題

### 問題13-1: 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）

**実装のヒント**

* 構造体ポインタのメンバーアクセスには->演算子を使用
* 文字列のコピーにはstrcpy()を使用

**期待される出力例:**

=== 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: 山田太郎  
年齢: 20  
成績: 85.5  
  
入力された学生情報:  
ID: 1001  
名前: 山田太郎  
年齢: 20歳  
成績: 85.50点  
  
成績を更新します...  
新しい成績: 92.0  
  
更新後の学生情報:  
ID: 1001  
名前: 山田太郎  
年齢: 20歳  
成績: 92.00点

### 問題13-2: 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

**実装のヒント**

* 距離の公式: sqrt((x2-x1)^2 + (y2-y1)^2)
* コンパイル時に-lmオプションが必要

**期待される出力例:**

=== 座標計算プログラム ===  
点A の座標:  
x: 0  
y: 0  
  
点B の座標:  
x: 3  
y: 4  
  
点A(0.00, 0.00) と 点B(3.00, 4.00) の距離: 5.00  
  
点Aを移動します:  
x方向の移動量: 2  
y方向の移動量: 1  
  
移動後の点A: (2.00, 1.00)  
新しい距離: 2.24

### 問題13-3: 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）

**実装のヒント**

* 構造体配列の初期化に注意
* 商品数のカウンタを別途管理

**期待される出力例:**

=== 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  
4. 終了  
  
選択: 1  
商品コード: P001  
商品名: ノート  
価格: 150  
在庫数: 50  
商品を追加しました。  
  
選択: 1  
商品コード: P002  
商品名: ペン  
価格: 100  
在庫数: 100  
商品を追加しました。  
  
選択: 2  
--- 商品一覧 ---  
コード: P001  
商品名: ノート  
価格: 150円  
在庫: 50個  
  
コード: P002  
商品名: ペン  
価格: 100円  
在庫: 100個  
  
選択: 3  
更新する商品コード: P001  
新しい在庫数: 45  
在庫を更新しました。

## 応用問題

### 問題13-4: 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

**実装のヒント**

* ネストした構造体のアクセス方法を理解
* 部署コードで従業員をグループ化

**期待される出力例:**

=== 従業員データベース ===  
従業員データ:  
ID: 1001, 名前: 田中太郎  
部署: 営業部 (コード: SALES)  
給与: 350000円  
  
ID: 1002, 名前: 鈴木花子  
部署: 営業部 (コード: SALES)  
給与: 320000円  
  
ID: 1003, 名前: 佐藤次郎  
部署: 開発部 (コード: DEV)  
給与: 450000円  
  
=== 部署別統計 ===  
営業部 (SALES):  
- 人数: 2名  
- 平均給与: 335000円  
  
開発部 (DEV):  
- 人数: 1名  
- 平均給与: 450000円  
  
最高給与の従業員:  
佐藤次郎 (開発部) - 450000円

### 問題13-5: 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

**実装のヒント**

* 部分一致検索にはstrstr()を使用
* 貸出状況は構造体内のフラグで管理

**期待される出力例:**

=== 図書管理システム ===  
1. 図書検索  
2. 貸出処理  
3. 返却処理  
4. 貸出中一覧  
5. 終了  
  
選択: 1  
検索するタイトル: プログラミング  
  
検索結果:  
1. 「C言語プログラミング入門」 - 山田太郎 (2023年)  
 状態: 貸出可能  
2. 「プログラミング言語C」 - カーニハン&リッチー (1988年)  
 状態: 貸出中 (借主ID: U001)  
  
選択: 2  
貸出する本の番号: 1  
借りる人のID: U002  
「C言語プログラミング入門」を貸出しました。  
  
選択: 4  
--- 貸出中の本一覧 ---  
1. 「プログラミング言語C」  
 借主ID: U001  
2. 「C言語プログラミング入門」  
 借主ID: U002

## チャレンジ問題

### 問題13-6: 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）

**追加要件**

* CSVファイルからのデータ読み込み機能
* 成績分布のヒストグラム表示

**期待される出力例:**

=== 成績管理システム ===  
データ読み込み完了: 15件  
  
学生別統計:  
学生ID 1001:  
- 受講科目数: 3  
- 平均点: 83.3  
- GPA: 3.33  
  
科目別統計:  
プログラミング基礎 (CS101):  
- 受講者数: 5  
- 平均点: 78.4  
- 成績分布:  
 A (90-100): \*\* (2名)  
 B (80-89): \* (1名)  
 C (70-79): \* (1名)  
 D (60-69): \* (1名)  
 F (0-59): - (0名)  
  
全体GPA順位:  
1. 学生ID 1003: GPA 3.80  
2. 学生ID 1001: GPA 3.33  
3. 学生ID 1002: GPA 2.95

### 問題13-7: 連結リストの実装

構造体を使って単方向連結リストを実装してください。

**要件**

* ノード構造体（データと次のノードへのポインタ）
* リストの先頭に要素を追加
* リストの末尾に要素を追加
* 特定の値を持つノードを削除
* リスト全体を表示
* メモリリークを防ぐ適切な解放処理

**実装のヒント**

* 自己参照構造体の定義方法を理解
* malloc/freeを使った動的メモリ管理

**期待される出力例:**

=== 連結リスト操作 ===  
1. 先頭に追加  
2. 末尾に追加  
3. 削除  
4. 表示  
5. 終了  
  
選択: 1  
値: 10  
先頭に10を追加しました。  
  
選択: 2  
値: 20  
末尾に20を追加しました。  
  
選択: 1  
値: 5  
先頭に5を追加しました。  
  
選択: 4  
リスト: 5 -> 10 -> 20 -> NULL  
  
選択: 3  
削除する値: 10  
10を削除しました。  
  
選択: 4  
リスト: 5 -> 20 -> NULL  
  
選択: 5  
メモリを解放しています...  
終了しました。

## 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 ex13\_1.c, ex13\_2.c のような形式で
3. 構造体の設計意図をコメントで説明
4. メモリ管理を適切に行う（特に動的メモリを使用する場合）

## 学習のポイント

* 構造体は関連するデータをまとめる強力な機能
* ポインタ経由のアクセスでは->演算子を使用
* 構造体の引数渡しは通常ポインタで行う（効率のため）
* ネストした構造体で複雑なデータ構造を表現可能

# 第14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

## 基礎問題

### 問題14-1: 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）

**期待される出力例:**

=== 関数ポインタの基本操作 ===  
a = 10, b = 5  
  
関数ポインタで直接呼び出し:  
maximum(10, 5) = 10  
minimum(10, 5) = 5  
power(10, 5) = 100000  
  
関数ポインタ配列で呼び出し:  
操作0: maximum(10, 5) = 10  
操作1: minimum(10, 5) = 5  
操作2: power(10, 5) = 100000  
  
関数アドレス表示:  
maximum: 0x400650  
minimum: 0x400670  
power: 0x400690

**ファイル名:** ex14\_1\_basic\_function\_pointer.c

### 問題14-2: 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

**期待される出力例:**

=== 関数選択システム ===  
利用可能な演算: + - \* / % ^  
  
第1の数: 15  
第2の数: 3  
演算子: +  
結果: 15 + 3 = 18  
  
第1の数: 20  
第2の数: 4  
演算子: /  
結果: 20 / 4 = 5  
  
第1の数: 10  
第2の数: 0  
演算子: /  
エラー: ゼロ除算はできません  
  
第1の数: 8  
第2の数: 3  
演算子: @  
エラー: 無効な演算子です

**ファイル名:** ex14\_2\_function\_selector.c

## 応用問題

### 問題14-3: 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

**期待される出力例:**

=== 配列処理のコールバック ===  
元の配列: [1, 2, 3, 4, 5]  
  
処理選択:  
1. 各要素を倍にする  
2. 各要素から1を引く  
3. 各要素の符号を反転  
選択: 1  
  
処理後: [2, 4, 6, 8, 10]  
  
別の配列: [5, -3, 8, -2, 0]  
処理選択: 3  
  
処理後: [-5, 3, -8, 2, 0]  
  
複数処理の連続実行:  
初期配列: [10, 20, 30]  
→ 倍にする: [20, 40, 60]  
→ 1を引く: [19, 39, 59]  
最終結果: [19, 39, 59]

**ファイル名:** ex14\_3\_array\_callback.c

### 問題14-4: 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

**期待される出力例:**

=== 関数ポインタ配列計算機 ===  
演算メニュー:  
0: 加算 (+)  
1: 減算 (-)  
2: 乗算 (\*)  
3: 除算 (/)  
4: 終了  
  
第1の数: 100  
第2の数: 25  
演算番号: 0  
結果: 100 + 25 = 125  
  
演算番号: 3  
結果: 100 / 25 = 4  
  
=== 演算履歴 ===  
1. 100 + 25 = 125  
2. 100 / 25 = 4  
  
=== 統計情報 ===  
総演算回数: 2  
- 加算: 1回 (50.0%)  
- 減算: 0回 (0.0%)  
- 乗算: 0回 (0.0%)  
- 除算: 1回 (50.0%)

**ファイル名:** ex14\_4\_calculator\_function\_array.c

## チャレンジ問題

### 問題14-5: 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

**期待される出力例:**

=== ソートアルゴリズム選択システム ===  
配列: [34, 7, 23, 32, 5, 62]  
  
ソートアルゴリズム:  
1. バブルソート  
2. 選択ソート  
3. 挿入ソート  
選択: 1  
  
順序:  
1. 昇順  
2. 降順  
選択: 1  
  
バブルソート（昇順）実行中...  
結果: [5, 7, 23, 32, 34, 62]  
比較回数: 15, 交換回数: 8  
  
=== 性能比較（配列サイズ: 100） ===  
アルゴリズム 比較回数 交換回数 時間(ms)  
バブルソート 4950 2475 0.125  
選択ソート 4950 99 0.098  
挿入ソート 2525 2426 0.087

**ファイル名:** ex14\_5\_sort\_algorithms.c

### 問題14-6: 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

**期待される出力例:**

=== イベント駆動システム ===  
イベントハンドラーを登録中...  
  
イベント発生シミュレーション:  
  
[開始] システムが起動しました  
[警告] メモリ使用率が80%を超えました  
[エラー] ファイルが見つかりません: config.txt  
[停止] システムをシャットダウンします  
  
優先度付きイベント処理:  
優先度3: [エラー] 重大なエラーが発生  
優先度1: [警告] 軽微な警告  
優先度2: [開始] サービス開始  
  
=== イベント統計 ===  
開始イベント: 2回  
停止イベント: 1回  
エラーイベント: 2回  
警告イベント: 2回  
総イベント数: 7回

**ファイル名:** ex14\_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 ex14\_1.c, ex14\_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 ex14\_1\_basic\_function\_pointer.c -o ex14\_1\_basic\_function\_pointer

### C99準拠

gcc -std=c99 -Wall -Wextra -pedantic ex14\_1\_basic\_function\_pointer\_c99.c -o ex14\_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

# 第15章: 複数ファイル・発展技術 - 演習問題

## 演習の目的

* プリプロセッサの高度な使い方を習得する
* メモリ管理の最適化技術を学ぶ
* キャッシュ効率を考慮したプログラミングを理解する
* システムプログラミングの基礎を身につける

## 基礎問題

### 問題15-1: 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。

**要件**

* 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

**期待される出力例:**

=== プリプロセッサマクロの基本 ===  
円の計算（半径 = 5.0）:  
- 面積: 78.54  
- 周長: 31.42  
  
球の計算（半径 = 3.0）:  
- 体積: 113.10  
- 表面積: 113.10  
  
[DEBUG] 計算完了: 円の面積 = 78.54 (半径: 5.0)  
[DEBUG] 計算完了: 球の体積 = 113.10 (半径: 3.0)  
  
型安全な最大値・最小値:  
MAX(10, 20) = 20  
MIN(3.14, 2.71) = 2.71  
  
黄金比を使った計算:  
黄金長方形 (短辺 = 10): 長辺 = 16.18

**ファイル名:** ex15\_1\_macro\_basics.c

### 問題15-2: 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。

**要件**

* 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

**期待される出力例:**

=== 安全なメモリ操作マクロ ===  
  
通常のメモリ割り当て:  
100バイトを割り当て: 成功 (0x7fff5fbff8c0)  
解放完了（ポインタはNULLにリセット）  
  
メモリ不足シミュレーション:  
SAFE\_MALLOC失敗: メモリ割り当てエラー (行: 45)  
  
配列境界チェック:  
配列[0-9]へのアクセステスト:  
index 5: OK (値: 50)  
index 10: エラー - 境界外アクセス検出！ (行: 67)  
  
=== メモリリーク検出レポート ===  
総割り当て: 1024 バイト  
総解放: 768 バイト  
リーク検出: 256 バイト (1ブロック)  
 - ブロック#3: 256バイト (行: 89で割り当て)

**ファイル名:** ex15\_2\_safe\_memory.c

### 問題15-3: 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。

**要件**

* 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

**期待される出力例:**

=== 基本的なメモリプール ===  
プール初期化: 100個のオブジェクト（各64バイト）  
  
10個のオブジェクトを割り当て...  
割り当て成功: 10個  
  
=== プール使用状況 ===  
総容量: 100 オブジェクト  
使用中: 10 オブジェクト (10.0%)  
空き: 90 オブジェクト (90.0%)  
総メモリ: 6400 バイト  
  
5個のオブジェクトを返却...  
  
=== 更新後の使用状況 ===  
使用中: 5 オブジェクト (5.0%)  
空き: 95 オブジェクト (95.0%)  
  
パフォーマンステスト（1000回の割り当て/解放）:  
- メモリプール: 0.15ms  
- 通常のmalloc/free: 2.34ms  
- 高速化: 15.6倍

**ファイル名:** ex15\_3\_memory\_pool.c

## 応用問題

### 問題15-4: 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。

**要件**

* 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

**期待される出力例:**

=== 汎用的なプリプロセッサライブラリ ===  
  
型チェック:  
TYPE\_CHECK(int, int): OK  
TYPE\_CHECK(int, double): 型不一致（コンパイル時警告）  
  
汎用スワップ:  
整数: a=10, b=20 → a=20, b=10  
文字列: s1="Hello", s2="World" → s1="World", s2="Hello"  
構造体: p1=(1,2), p2=(3,4) → p1=(3,4), p2=(1,2)  
  
ループ展開（5回）:  
展開0: i=0, value=0  
展開1: i=1, value=1  
展開2: i=2, value=4  
展開3: i=3, value=9  
展開4: i=4, value=16  
  
=== ベンチマーク結果 ===  
[BENCHMARK] 配列初期化: 0.234ms  
[BENCHMARK] 行列演算: 1.567ms  
[BENCHMARK] ソート処理: 3.890ms  
  
スタティックアサート:  
コンパイル時チェック: すべてOK

**ファイル名:** ex15\_4\_generic\_macros.c

### 問題15-5: 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。

**要件**

* 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

**期待される出力例:**

=== 高性能メモリアロケーター ===  
  
アロケーターを初期化中...  
- 小オブジェクトプール: 0-64 バイト  
- 中オブジェクトプール: 65-512 バイト  
- 大オブジェクトプール: 513+ バイト  
  
割り当てテスト:  
32バイト割り当て: 成功 (小プールから)  
256バイト割り当て: 成功 (中プールから)  
1024バイト割り当て: 成功 (大プールから)  
  
=== メモリ使用統計 ===  
小オブジェクトプール:  
 割り当て回数: 150  
 解放回数: 120  
 現在使用中: 30 (1920 バイト)  
 断片化率: 5.2%  
  
中オブジェクトプール:  
 割り当て回数: 50  
 解放回数: 45  
 現在使用中: 5 (1280 バイト)  
 断片化率: 8.7%  
  
大オブジェクトプール:  
 割り当て回数: 10  
 解放回数: 8  
 現在使用中: 2 (2048 バイト)  
 断片化率: 12.3%  
  
総メモリ使用量: 5248 バイト  
平均割り当て時間: 0.0034ms

**ファイル名:** ex15\_5\_allocator.c

## チャレンジ問題

### 問題15-6: 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。

**要件**

* 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

**期待される出力例:**

=== キャッシュフレンドリーなデータ構造 ===  
  
ベクター初期化（初期容量: 16）  
キャッシュライン境界でアラインメント済み  
  
要素追加テスト:  
1000要素を追加中...  
容量自動拡張: 16 → 32 → 64 → 128 → 256 → 512 → 1024  
  
=== パフォーマンス比較 ===  
順次アクセス（キャッシュ効率的）:  
- キャッシュベクター: 0.089ms  
- 通常の配列: 0.156ms  
- 高速化: 1.75倍  
  
ランダムアクセス:  
- キャッシュベクター: 0.234ms  
- 通常の配列: 0.312ms  
- 高速化: 1.33倍  
  
バルク操作（100要素一括挿入）:  
- キャッシュベクター: 0.012ms  
- 個別挿入: 0.089ms  
- 高速化: 7.42倍  
  
=== キャッシュ統計 ===  
L1キャッシュヒット率: 94.3%  
L2キャッシュヒット率: 98.7%  
プリフェッチ効果: 23.4%の改善

**ファイル名:** ex15\_6\_cache\_vector.c

### 問題15-7: 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。

**要件**

* 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

**期待される出力例:**

=== プリプロセッサベースのDSL ===  
  
信号機状態機械の定義:  
状態数: 3  
遷移数: 3  
  
状態機械を実行中...  
[00:00] 状態: Red  
[00:05] タイムアウト → Yellow へ遷移  
[00:06] 状態: Yellow  
[00:08] タイムアウト → Green へ遷移  
[00:09] 状態: Green  
[00:14] タイムアウト → Red へ遷移  
[00:15] 状態: Red  
  
=== コード生成結果 ===  
生成された関数:  
- void TrafficLight\_init()  
- void TrafficLight\_process\_event(Event)  
- State TrafficLight\_get\_state()  
- const char\* TrafficLight\_state\_name(State)  
  
生成されたコード行数: 87行  
マクロ展開前: 12行  
圧縮率: 7.25倍  
  
=== 検証結果 ===  
✓ すべての状態から遷移可能  
✓ デッドロックなし  
✓ 未定義遷移なし

**ファイル名:** ex15\_7\_dsl.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 ex15\_1.c, ex15\_2.c のような形式で
3. 高度な機能を使う場合は、その意図をコメントで説明
4. パフォーマンス測定結果も含める

## 学習のポイント

* プリプロセッサは強力だが、使いすぎると可読性が低下する
* メモリ管理は性能に直結する重要な要素
* キャッシュ効率を考慮すると大幅な性能向上が可能
* システムレベルの知識が高度なC言語プログラミングには不可欠

# 第16章: C23の新機能 - 演習問題

## 演習の目的

* C23標準で導入された新機能を理解する
* bool型、2進数リテラル、typeof演算子、nullptrを活用する
* モダンなCプログラミングスタイルを習得する
* より安全で表現力豊かなコードを書く

## 基礎問題

### 問題16-1: 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
* 状態遷移の履歴記録

**実装のヒント:**

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

**期待される出力例:**

=== bool型を使った状態管理システム ===  
  
初期状態:  
接続: false  
認証: false  
権限: false  
ビジー: false  
  
システム接続中...  
接続: true  
認証: false (認証が必要です)  
  
ユーザー認証...  
接続: true  
認証: true  
権限: false (権限の付与待ち)  
  
管理者権限を付与...  
接続: true  
認証: true  
権限: true  
ビジー: false  
  
操作可能状態: Yes  
  
=== 状態遷移履歴 ===  
[00:00:00] システム起動  
[00:00:01] 接続確立  
[00:00:03] 認証成功  
[00:00:05] 権限付与  
[00:00:10] 処理開始 (ビジー: true)  
[00:00:15] 処理完了 (ビジー: false)

### 問題16-2: 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
* 複数フラグの一括操作

**実装例:**

#define FLAG\_READ 0b00000001  
#define FLAG\_WRITE 0b00000010  
#define FLAG\_EXECUTE 0b00000100

**期待される出力例:**

=== 2進数リテラルを使ったフラグ管理 ===  
  
権限フラグの定義:  
FLAG\_READ = 0b00000001 (0x01)  
FLAG\_WRITE = 0b00000010 (0x02)  
FLAG\_EXECUTE = 0b00000100 (0x04)  
FLAG\_DELETE = 0b00001000 (0x08)  
  
初期フラグ: 0b00000000  
  
読み取り権限を設定:  
現在のフラグ: 0b00000001 (READ)  
  
書き込み権限を追加:  
現在のフラグ: 0b00000011 (READ | WRITE)  
  
すべての権限を設定:  
現在のフラグ: 0b00001111 (READ | WRITE | EXECUTE | DELETE)  
  
実行権限をトグル:  
現在のフラグ: 0b00001011 (READ | WRITE | DELETE)  
  
権限チェック:  
- 読み取り: ○  
- 書き込み: ○  
- 実行: ×  
- 削除: ○  
  
一括操作 (読み取り専用に設定):  
現在のフラグ: 0b00000001 (READ)

## 応用問題

### 問題16-3: 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
* 型安全な比較関数マクロ

**実装例:**

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

**期待される出力例:**

=== typeof演算子の汎用マクロ ===  
  
整数配列のテスト:  
配列: [34, 67, 12, 89, 45, 23, 78, 56]  
最大値: 89  
最小値: 12  
合計: 404  
  
浮動小数点配列のテスト:  
配列: [3.14, 2.71, 1.41, 1.73, 2.23]  
最大値: 3.14  
最小値: 1.41  
合計: 11.22  
  
安全な変数交換:  
交換前: a=10, b=20  
交換後: a=20, b=10  
  
文字列の交換:  
交換前: s1="Hello", s2="World"  
交換後: s1="World", s2="Hello"  
  
型安全な比較:  
compare(10, 20) = -1  
compare(3.14, 3.14) = 0  
compare('Z', 'A') = 1

### 問題16-4: 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
* メモリリークの防止

**実装のヒント**

* 従来のNULLの代わりにnullptrを使用
* 戻り値でのエラー通知

**期待される出力例:**

=== nullptr安全リンクリスト ===  
  
要素を追加: 10  
要素を追加: 20  
要素を追加: 30  
  
リストの内容: 10 -> 20 -> 30 -> nullptr  
  
要素20を検索... 見つかりました  
要素40を検索... 見つかりません  
  
要素20を削除...  
リストの内容: 10 -> 30 -> nullptr  
  
空のリストに対する操作:  
削除試行: エラー - リストが空です  
検索試行: エラー - リストが空です  
  
nullptrチェック統計:  
- nullptrチェック総数: 42  
- nullptr検出回数: 7  
- 安全に処理された操作: 100%  
  
メモリリーク検査: OK (すべてのメモリが解放されました)

### 問題16-5: デジット区切り記号の活用

デジット区切り記号を使って、大きな数値を扱うプログラムを作成してください。

**要件**

* 10進数、16進数、2進数での表記
* 可読性の高い定数定義
* ビットマスクの定義
* メモリサイズの計算

**実装例:**

#define MEMORY\_SIZE 16'777'216 // 16MB  
#define MASK\_32BIT 0xFFFF'FFFF  
#define PATTERN 0b1010'1010'1010'1010

**期待される出力例:**

=== デジット区切り記号を使った数値表現 ===  
  
大きな数値の定義:  
人口: 7'825'000'000 人  
メモリサイズ: 16'777'216 バイト (16MB)  
国家予算: 106'600'000'000'000 円  
  
16進数表記:  
アドレス空間: 0xFFFF'FFFF'FFFF'FFFF  
カラーコード: 0x00FF'00FF (緑とマゼンタ)  
  
2進数表記:  
ビットパターン: 0b1010'1010'1010'1010  
マスク: 0b1111'1111'0000'0000  
  
計算例:  
16MB = 16'777'216 バイト  
 = 16'384 KB  
 = 16 MB  
  
ビット演算:  
0b1111'0000'1111'0000 AND 0b1010'1010'1010'1010  
= 0b1010'0000'1010'0000

## チャレンジ問題

### 問題16-6: 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作成してください。

**要件**

* bool型で各種設定のON/OFF管理
* 2進数リテラルでフラグ管理
* nullptrで安全なポインタ操作
* typeof演算子でジェネリックな操作
* 設定の保存と読み込み機能

**機能要件**

* 設定項目の動的追加・削除
* 設定のグループ化
* デフォルト値の管理
* 設定変更の通知機能

**期待される出力例:**

=== C23総合設定管理システム ===  
  
設定グループ: Display  
- brightness: true (bool)  
- dark\_mode: false (bool)  
- resolution: 1920x1080  
- refresh\_rate: 60  
  
設定グループ: Network  
- wifi\_enabled: true (bool)  
- bluetooth\_enabled: false (bool)  
- airplane\_mode: false (bool)  
  
設定フラグ (2進数表示):  
現在: 0b0000'0101 (WiFi | Brightness)  
  
設定変更: dark\_mode = true  
通知: DisplaySettingsChanged -> dark\_mode  
  
設定の保存...  
ファイル: settings.conf  
保存完了 (チェックサム: 0xA5C3)  
  
=== typeof演算子による汎用操作 ===  
設定値の型安全な更新:  
- 整数値: 60 -> 144 (成功)  
- bool値: false -> true (成功)  
- 文字列: "1920x1080" -> "2560x1440" (成功)

### 問題16-7: ビット演算デバッガ

教育的なビット演算デバッガを作成してください。

**要件**

* 2進数リテラルでの入力サポート
* ビット演算の可視化
* ステップ実行機能
* 演算結果の詳細説明

**実装すべき演算**

* AND, OR, XOR, NOT
* ビットシフト（左右）
* ビット回転
* ビットカウント

**期待される出力例:**

=== ビット演算デバッガ ===  
  
入力方式を選択:  
1. 2進数 (0b...)  
2. 16進数 (0x...)  
3. 10進数  
  
選択: 1  
値A: 0b1010'1100  
値B: 0b1100'1010  
  
=== ビット演算の可視化 ===  
  
AND演算:  
 1010 1100  
& 1100 1010  
-----------  
 1000 1000 (0x88)  
  
OR演算:  
 1010 1100  
| 1100 1010  
-----------  
 1110 1110 (0xEE)  
  
XOR演算:  
 1010 1100  
^ 1100 1010  
-----------  
 0110 0110 (0x66)  
  
NOT演算:  
~ 1010 1100  
-----------  
 0101 0011 (0x53)  
  
左シフト (2ビット):  
1010 1100 << 2 = 1011 0000 (0xB0)  
 ↑↑ 消失  
  
右シフト (2ビット):  
1010 1100 >> 2 = 0010 1011 (0x2B)  
 消失 ↑↑  
  
ビット数: A=5個, B=5個

## 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 ex16\_1.c, ex16\_2.c のような形式で
3. C23機能を使用する箇所には必ずコメントを記載
4. コンパイラのC23サポートを確認（gcc 13以降推奨）

## 学習のポイント

* C23の新機能は従来のCコードをより安全で読みやすくする
* bool型により意図が明確なコードが書ける
* 2進数リテラルはビット操作を直感的にする
* nullptrはポインタの安全性を向上させる
* typeof演算子により型安全なマクロが作成可能

# 上級編を修了して

## 習得したスキル

上級編の演習問題を通じて、以下の高度なスキルを習得しました：

### システムレベルプログラミング

* ビット操作による効率的なデータ処理
* メモリレイアウトの理解と最適化
* ハードウェア制御の基礎

### 高度なデータ構造

* 構造体を使った複雑なデータモデリング
* リンクリストやツリー構造の実装
* メモリ効率を考慮した設計

### 関数型プログラミング要素

* 関数ポインタによる高階関数
* コールバックパターンの実装
* 汎用的なアルゴリズムの設計

### プロジェクト構成

* 複数ファイルによるモジュール設計
* ヘッダファイルの適切な管理
* 再利用可能なライブラリの作成

## 実務への応用

これらのスキルは、以下のような分野で活用できます：

1. **組み込みシステム開発**
   * マイコンプログラミング
   * リアルタイムシステム
2. **システムプログラミング**
   * OS開発
   * デバイスドライバ
3. **高性能コンピューティング**
   * 数値計算
   * ゲームエンジン

## 継続的な学習

プログラミングスキルを維持・向上させるために：

1. **コードリーディング**
   * オープンソースプロジェクトのソースコードを読む
   * 優れたC言語プロジェクトから学ぶ
2. **プロジェクト実践**
   * 自分でツールやライブラリを作成
   * 既存プロジェクトへの貢献
3. **新技術の探求**
   * C標準の最新動向をフォロー
   * 関連技術（Rust、Zig等）の学習

## 最後に

C言語は、コンピュータサイエンスの基礎を学ぶ上で最適な言語です。本教材で学んだ知識は、他のプログラミング言語を学ぶ際にも必ず役立ちます。

これからも挑戦を続け、素晴らしいソフトウェアを作り出してください！