# 第6章 制御構造（ループ） - 演習問題

# 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  
\*/