# 第3章 データ型と変数 - 演習問題

# 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）  
\*/