# 第1章 導入・環境構築 - 演習問題

# 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;  
}