# 第1章 導入・環境構築

## 対応C規格

* **主要対象:** 全規格共通
* **学習内容:** C言語の基礎知識、開発環境構築、規格解説

## 学習目標

この章を完了すると、以下のことができるようになります：

* C言語の特徴と歴史を理解する
* 開発環境を構築し、基本的なコンパイル手順を覚える
* C言語の規格（C90, C99, C11, C17）の違いを理解する
* 簡単なプログラムをコンパイル・実行できる

## 概要と詳細

### C言語とは

C言語は1972年にデニス・リッチーがベル研究所で開発したプログラミング言語です。現在でも世界中で広く使われている、とても重要なプログラミング言語の一つです。

#### なぜC言語を学ぶのか？

初めてプログラミングを学ぶ方は「なぜC言語から始めるの？」と思うかもしれません。実は、C言語を学ぶことには大きなメリットがあります：

1. **コンピュータの仕組みが理解できる**
   * C言語はコンピュータのハードウェアに近い言語です
   * メモリやCPUがどのように動作するか実感できます
   * 他の言語を学ぶときの基礎知識になります
2. **多くのプログラミング言語の元になっている**
   * C++、Java、C#などはC言語の影響を受けています
   * C言語の文法を知っていれば、他の言語も学びやすくなります
3. **今でも現役で使われている**
   * 50年以上の歴史がありますが、今でも重要な場面で使われています
   * 特に高速性や効率性が求められる場面では欠かせません

#### 主な特徴

C言語には以下のような特徴があります。初心者には少し難しく感じるかもしれませんが、一つずつ理解していけば大丈夫です：

* **システムプログラミング向け**:
  + OS（オペレーティングシステム）の開発に使われます
  + ハードウェアを直接制御できます
  + 例：LinuxカーネルはC言語で書かれています
* **高い移植性**:
  + 一度書いたプログラムが様々なコンピュータで動きます
  + Windows、Mac、Linuxなど、異なるOSでも同じコードが使えます
  + これを「Write Once, Run Anywhere（一度書けば、どこでも動く）」と言います
* **効率性**:
  + 実行速度が非常に速い
  + メモリの使用量を最小限に抑えられる
  + 限られたリソースで動作させる必要がある場面で重宝されます
* **シンプルな文法**:
  + 言語の機能が基本的なものに絞られています
  + 覚えることが比較的少ない
  + ただし、その分プログラマーが考えることは多くなります

#### 使用分野

C言語は以下のような分野で活躍しています：

* **オペレーティングシステム（OS）**
  + Linux：世界中のサーバーで使われているOS
  + Windows：一部のコア部分
  + macOS：カーネル部分
* **組込みシステム**
  + 家電製品（エアコン、洗濯機、電子レンジなど）
  + 自動車の制御システム
  + IoTデバイス（スマートウォッチ、センサーなど）
* **データベースシステム**
  + MySQL、PostgreSQLなどの有名なデータベース
  + 高速なデータ処理が必要なため
* **ゲームエンジン**
  + ゲームの基盤となる部分
  + 3Dグラフィックスの処理など
* **科学技術計算**
  + 物理シミュレーション
  + 数値解析
  + 機械学習の基礎ライブラリ

### C言語の規格

C言語は時代とともに進化し、複数の規格（バージョン）が策定されています。プログラミング言語にも「バージョン」があることを不思議に思うかもしれませんが、これは言語をより使いやすく、より強力にするための改良の歴史です。

#### C言語規格の歴史

| 規格名 | 発表年 | 正式名称 | 主な特徴 |
| --- | --- | --- | --- |
| **C90** | 1990年 | ISO/IEC 9899:1990 | 初の国際標準、ANSI C |
| **C99** | 1999年 | ISO/IEC 9899:1999 | \_Bool型、可変長配列、inline関数 |
| **C11** | 2011年 | ISO/IEC 9899:2011 | 匿名構造体、\_Generic、マルチスレッド |
| **C17** | 2018年 | ISO/IEC 9899:2018 | C11のバグ修正版 |
| **C23** | 2024年 | ISO/IEC 9899:2024 | bool型標準化、typeof演算子、nullptr、2進数リテラル |

#### 各規格の詳しい説明

**C90（ANSI C）** - C言語の最初の国際標準規格です - 最も基本的で、すべてのC言語コンパイラが対応しています - シンプルで学習しやすく、初心者にお勧めです - 古い組込みシステムでも確実に動作します

**C99** - 多くの便利な機能が追加されました - 例：forループの中で変数を宣言できるようになりました c for (int i = 0; i < 10; i++) { /\* C99から可能 \*/ } - 現在最も広く使われている規格です

**C11** - マルチスレッド（並列処理）のサポートが追加されました - より安全なプログラミングのための機能が強化されました - 比較的新しいため、すべての環境で使えるとは限りません

**C17** - C11の細かいバグを修正したマイナーアップデート - 新機能の追加はほとんどありません

**C23** - C99以来の大規模な改訂となり、現代的なプログラミングのニーズに対応 - bool型の標準化、typeof演算子、nullptr、2進数リテラルなど多数の新機能 - 詳細は[第14章「C23の新機能」](../c23-features/README.md)で解説します

#### どの規格を選ぶべきか？

初学者の方への推奨：

* **学習目的なら**: **C90から始める**
  + 最も基本的な機能を確実に習得できます
  + どこでも動く汎用的なコードが書けるようになります
  + 段階的に新しい機能を学んでいけます
* **実用的なプログラムを書くなら**: **C99を使う**
  + 便利な機能が使えて、生産性が向上します
  + ほとんどの環境で問題なく動作します
  + 現在の業界標準と言えます
* **組込みシステムを作るなら**: **C90またはC99**
  + 使用するマイコンやコンパイラの対応状況を確認してください
  + 古い環境ではC90しか使えない場合があります

#### 規格による違いの例

/\* C90スタイル \*/  
int main(void) {  
 int i; /\* 変数宣言は関数の先頭で \*/  
   
 for (i = 0; i < 10; i++) {  
 printf("%d\n", i);  
 }  
 return 0;  
}  
  
/\* C99以降のスタイル \*/  
int main(void) {  
 for (int i = 0; i < 10; i++) { /\* forループ内で宣言可能 \*/  
 printf("%d\n", i);  
 }  
 return 0;  
}

この教材では、基本的にC90準拠のコードを示し、必要に応じてC99版も提供します。これにより、どんな環境でも動作するコードの書き方を学びながら、新しい機能も理解できるようになっています。

### 開発環境

C言語でプログラミングを始めるには、いくつかのツール（ソフトウェア）が必要です。最初は難しく感じるかもしれませんが、一度セットアップすればずっと使えます。

#### 必須ツール

プログラミングには以下の4つのツールが必要です：

1. **テキストエディター**: プログラムを書くためのソフト
   * メモ帳でも書けますが、専用エディターが便利です
   * 推奨：Visual Studio Code（無料）、Sublime Text、Atom
   * プログラミング用エディターは色分け表示などの便利機能があります
2. **コンパイラ**: 人間が書いたコードをコンピュータが理解できる形に変換
   * C言語で書いたテキストを実行可能なプログラムに変換します
   * 主なコンパイラ：GCC（無料）、Clang（無料）、Visual C++（Windows）
   * この教材ではGCCを使用します
3. **リンカー**: プログラムの部品を組み合わせる
   * 通常はコンパイラに含まれているので、別途インストールは不要です
   * 複数のファイルからなるプログラムを一つにまとめます
4. **デバッガー**: プログラムの誤り（バグ）を見つけるツール
   * プログラムを一行ずつ実行して動作を確認できます
   * GDB（GCCに付属）が一般的です

#### OS別の環境構築手順

お使いのOSに応じて、以下の手順で環境を構築してください：

**Windows:**

最も簡単な方法はMinGW-w64をインストールすることです：

1. MinGW-w64のインストーラーをダウンロード
2. インストール時の設定：
   * Architecture: x86\_64（64ビット版）
   * Threads: posix
   * Exception: seh
3. 環境変数PATHにMinGWのbinフォルダを追加
4. コマンドプロンプトで gcc --version と入力して確認

または、Microsoft Visual Studio Community（無料）も使えます。

**macOS:**

Xcode Command Line Toolsをインストールします：

# ターミナルで以下のコマンドを実行  
xcode-select --install

**Linux（Ubuntu/Debian）:**

build-essentialパッケージをインストールします：

sudo apt update  
sudo apt install build-essential

### コンパイル手順

C言語プログラムを実行するまでの過程を、料理に例えて説明します：

1. **ソースコード作成**（レシピを書く）
   * 人間が読めるC言語のコードを書きます
   * ファイルの拡張子は .c です
   * 例：hello.c
2. **プリプロセッサ処理**（材料の準備）
   * #include で指定したファイルを取り込みます
   * #define で定義した値を置き換えます
   * まだプログラムではなく、準備段階です
3. **コンパイル**（調理）
   * C言語のコードをアセンブリ言語に変換します
   * 文法エラーがあればここで検出されます
   * まだ実行はできません
4. **アセンブル**（盛り付け）
   * アセンブリ言語を機械語（0と1）に変換します
   * オブジェクトファイル（.o または .obj）が作成されます
5. **リンク**（完成）
   * 必要なライブラリと結合します
   * 実行可能ファイルが生成されます
   * Windowsでは .exe ファイル、Unix系では拡張子なし
6. **実行**（食べる）
   * 作成したプログラムを実行します

#### 基本的なコンパイルコマンド

実際のコンパイル方法を見てみましょう：

# シンプルな方法（すべての処理を一度に）  
gcc hello.c -o hello  
# hello.c をコンパイルして hello という実行ファイルを作成  
  
# 実行  
./hello # Unix/Linux/macOS  
hello # Windows  
  
# より詳細なオプション付き  
gcc -Wall -Wextra -o hello hello.c  
# -Wall: すべての警告を表示  
# -Wextra: 追加の警告も表示  
# -o hello: 出力ファイル名を指定  
  
# 段階的なコンパイル（大きなプログラムで使用）  
gcc -c hello.c # hello.o を作成  
gcc -c another.c # another.o を作成  
gcc hello.o another.o -o program # リンクして実行ファイル作成

#### よくあるエラーと対処法

初心者がよく遭遇するエラー：

1. **“gcc: command not found”**
   * コンパイラがインストールされていません
   * PATHが設定されていません
2. **“hello.c: No such file or directory”**
   * ファイル名が間違っています
   * 現在のディレクトリにファイルがありません
3. **“undefined reference to `main’”**
   * main関数が定義されていません
   * C言語のプログラムには必ずmain関数が必要です
4. **アセンブル**（オブジェクトファイルに変換）
5. **リンク**（実行ファイルの生成）
6. **実行**

#### 基本的なコンパイルコマンド

# 一段階でコンパイル  
gcc program.c -o program  
  
# 段階的なコンパイル  
gcc -c program.c # オブジェクトファイル作成  
gcc program.o -o program # 実行ファイル作成

## サンプルコード

### 環境確認プログラム

プログラムファイル: examples/environment\_check.c

このプログラムは開発環境が正しく設定されているかを確認します。

### コンパイルと実行

# examples ディレクトリに移動  
cd examples  
  
# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic environment\_check.c -o env\_check\_c90  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic environment\_check.c -o env\_check\_c99  
  
# C11準拠でコンパイル  
gcc -std=c11 -Wall -Wextra -pedantic environment\_check.c -o env\_check\_c11  
  
# C17準拠でコンパイル  
gcc -std=c17 -Wall -Wextra -pedantic environment\_check.c -o env\_check\_c17  
  
# 実行  
./env\_check\_c90

## 演習課題

### 基礎問題

1. **環境構築確認**
   * サンプルプログラムを正常にコンパイル・実行できることを確認してください
   * C90、C99、C11、C17の各規格でコンパイルを試してください
2. **コンパイラ情報の確認**
   * 使用しているコンパイラの名前とバージョンを調べてください
   * 対応している規格を確認してください

### 応用問題

1. **規格比較**
   * C90とC99の主な違いを3つ以上挙げてください
   * 組込み開発でC90が今でも使われる理由を考えてください
2. **コンパイル手順の理解**
   * プリプロセッサ、コンパイラ、リンカーの役割を説明してください
   * -E, -S, -c オプションの効果を確認してください

## コンパイル方法

この章では以下のMakefileを使用してコンパイルができます：

# 全ての例題をコンパイル  
make all  
  
# C90準拠でコンパイル  
make STANDARD=c90  
  
# C99準拠でコンパイル   
make STANDARD=c99  
  
# C11準拠でコンパイル  
make STANDARD=c11  
  
# C17準拠でコンパイル  
make STANDARD=c17  
  
# クリーンアップ  
make clean

## 規格による違い

この章では基本的な環境確認のため、規格による大きな違いはありません。ただし、コンパイラの警告やエラーメッセージに違いが出る場合があります。

## 次の章へ

環境構築が完了したら、[基本文法・Hello World](../basics-syntax/README.md) に進んでください。

## 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 (ISO/IEC 9899:2018)](https://www.iso.org/standard/74528.html)
* [GNU Make マニュアル](https://www.gnu.org/software/make/manual/)

# サンプルコード

## environment\_check.c

/\*  
 \* environment\_check.c - C言語開発環境確認プログラム  
 \*  
 \* このプログラムは開発環境が正しく設定されているかを確認します。  
 \* 各種C規格でコンパイルして動作を確認してください。  
 \*/  
  
#include <stdio.h>  
#include <limits.h>  
  
/\*  
 \* 環境情報を表示する関数  
 \*/  
void print\_environment\_info(void)  
{  
 printf("=== C言語開発環境確認 ===\n\n");  
  
 /\* コンパイラ情報 \*/  
 printf("コンパイラ情報:\n");  
  
#ifdef \_\_GNUC\_\_  
 printf("- GCC version: %d.%d.%d\n", \_\_GNUC\_\_, \_\_GNUC\_MINOR\_\_, \_\_GNUC\_PATCHLEVEL\_\_);  
#endif  
  
#ifdef \_MSC\_VER  
 printf("- Microsoft Visual C++ version: %d\n", \_MSC\_VER);  
#endif  
  
#ifdef \_\_clang\_\_  
 printf("- Clang version: %d.%d.%d\n", \_\_clang\_major\_\_, \_\_clang\_minor\_\_, \_\_clang\_patchlevel\_\_);  
#endif  
  
 /\* C規格情報 \*/  
 printf("\nC規格情報:\n");  
  
#ifdef \_\_STDC\_VERSION\_\_  
 printf("- \_\_STDC\_VERSION\_\_: %ldL\n", \_\_STDC\_VERSION\_\_);  
  
 if (\_\_STDC\_VERSION\_\_ >= 202311L)  
 {  
 printf("- 対応規格: C23\n");  
 }  
 else if (\_\_STDC\_VERSION\_\_ >= 201710L)  
 {  
 printf("- 対応規格: C17 (C18)\n");  
 }  
 else if (\_\_STDC\_VERSION\_\_ >= 201112L)  
 {  
 printf("- 対応規格: C11\n");  
 }  
 else if (\_\_STDC\_VERSION\_\_ >= 199901L)  
 {  
 printf("- 対応規格: C99\n");  
 }  
 else  
 {  
 printf("- 対応規格: C90 (ANSI C)\n");  
 }  
#else  
 printf("- 対応規格: C90 (ANSI C) または未定義\n");  
#endif  
  
 /\* システム情報 \*/  
 printf("\nシステム情報:\n");  
  
 printf("- int型サイズ: %lu bytes\n", (unsigned long)sizeof(int));  
 printf("- long型サイズ: %lu bytes\n", (unsigned long)sizeof(long));  
 printf("- ポインタサイズ: %lu bytes\n", (unsigned long)sizeof(void \*));  
 printf("- char型範囲: %d 〜 %d\n", CHAR\_MIN, CHAR\_MAX);  
 printf("- int型範囲: %d 〜 %d\n", INT\_MIN, INT\_MAX);  
}  
  
/\*  
 \* 基本的な計算テスト  
 \*/  
void test\_basic\_operations(void)  
{  
 int a = 10;  
 int b = 3;  
 int result;  
  
 printf("\n=== 基本演算テスト ===\n");  
  
 result = a + b;  
 printf("%d + %d = %d\n", a, b, result);  
  
 result = a - b;  
 printf("%d - %d = %d\n", a, b, result);  
  
 result = a \* b;  
 printf("%d \* %d = %d\n", a, b, result);  
  
 result = a / b;  
 printf("%d / %d = %d\n", a, b, result);  
  
 result = a % b;  
 printf("%d %% %d = %d\n", a, b, result);  
  
 printf("基本演算テスト: 正常完了\n");  
}  
  
/\*  
 \* メイン関数  
 \*/  
int main(void)  
{  
 printf("C言語開発環境確認プログラム\n");  
 printf("コンパイル日時: %s %s\n\n", \_\_DATE\_\_, \_\_TIME\_\_);  
  
 /\* 環境情報の表示 \*/  
 print\_environment\_info();  
  
 /\* 基本演算のテスト \*/  
 test\_basic\_operations();  
  
 printf("\n=== 環境確認完了 ===\n");  
 printf("この出力が正常に表示されれば、開発環境は正しく設定されています。\n");  
  
 return 0;  
}