# 第9章 関数

## 対応C規格

* **主要対象:** C90
* **学習内容:** 関数の基本、引数と戻り値、関数のスコープ、再帰関数、関数プロトタイプ

## 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：

* 関数の定義と呼び出しができる
* 引数と戻り値を適切に使える
* 関数のスコープを理解する
* 再帰関数を作成できる
* 関数プロトタイプの重要性を理解する

## 理論解説

### 関数の基本概念

関数は特定の処理をまとめたコードブロックで、プログラムの再利用性と保守性を向上させます。

#### 関数を使う理由

1. **コードの再利用性**: 同じ処理を何度も書く必要がない
2. **プログラムの構造化**: 複雑な問題を小さな部分に分割
3. **保守性の向上**: 修正が必要な箇所を特定しやすい
4. **可読性の向上**: プログラムの意図が明確になる
5. **デバッグの容易さ**: 問題のある部分を特定しやすい

#### 関数の構成要素

戻り値の型 関数名(引数リスト)  
{  
 /\* 関数本体 \*/  
 return 戻り値; /\* 戻り値がある場合 \*/  
}

### 基本的な関数の例

#include <stdio.h>  
  
/\* 2つの数の合計を計算する関数 \*/  
int add(int a, int b)  
{  
 int result = a + b;  
 return result;  
}  
  
int main(void)  
{  
 int num1 = 10;  
 int num2 = 20;  
 int sum;  
   
 /\* 関数の呼び出し \*/  
 sum = add(num1, num2);  
   
 printf("%d + %d = %d\n", num1, num2, sum);  
   
 return 0;  
}

### 関数の定義と宣言

#### 関数プロトタイプ（前方宣言）

#include <stdio.h>  
  
/\* 関数プロトタイプ \*/  
int multiply(int x, int y);  
void print\_result(int value);  
double calculate\_average(int \*array, int size);  
  
int main(void)  
{  
 int a = 5, b = 3;  
 int product;  
   
 product = multiply(a, b);  
 print\_result(product);  
   
 return 0;  
}  
  
/\* 関数の実装 \*/  
int multiply(int x, int y)  
{  
 return x \* y;  
}  
  
void print\_result(int value)  
{  
 printf("結果: %d\n", value);  
}  
  
double calculate\_average(int \*array, int size)  
{  
 int i;  
 int sum = 0;  
   
 if (size <= 0)  
 {  
 return 0.0;  
 }  
   
 for (i = 0; i < size; i++)  
 {  
 sum += array[i];  
 }  
   
 return (double)sum / size;  
}

### 引数の渡し方

#### 値渡し（Call by Value）

/\* 値渡しの例 \*/  
void swap\_wrong(int a, int b)  
{  
 int temp = a;  
 a = b;  
 b = temp;  
 /\* 呼び出し元の変数は変更されない \*/  
}  
  
/\* ポインタを使った参照渡し \*/  
void swap\_correct(int \*a, int \*b)  
{  
 int temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
 /\* 呼び出し元の変数が変更される \*/  
}  
  
int main(void)  
{  
 int x = 10, y = 20;  
   
 printf("交換前: x = %d, y = %d\n", x, y);  
   
 swap\_wrong(x, y);  
 printf("値渡し後: x = %d, y = %d\n", x, y);  
   
 swap\_correct(&x, &y);  
 printf("ポインタ渡し後: x = %d, y = %d\n", x, y);  
   
 return 0;  
}

### 様々な関数の種類

#### 戻り値のない関数（void関数）

void print\_header(void)  
{  
 printf("====================\n");  
 printf(" プログラム開始\n");  
 printf("====================\n");  
}  
  
void greet\_user(char \*name)  
{  
 printf("こんにちは、%sさん！\n", name);  
}

#### 配列を扱う関数

/\* 配列の要素数を計算できないため、サイズを別途渡す必要がある \*/  
int array\_sum(int arr[], int size)  
{  
 int i, sum = 0;  
   
 for (i = 0; i < size; i++)  
 {  
 sum += arr[i];  
 }  
   
 return sum;  
}  
  
/\* 配列を初期化する関数 \*/  
void initialize\_array(int arr[], int size, int value)  
{  
 int i;  
   
 for (i = 0; i < size; i++)  
 {  
 arr[i] = value;  
 }  
}

### 再帰関数

再帰関数は自分自身を呼び出す関数です。

/\* 階乗を計算する再帰関数 \*/  
int factorial(int n)  
{  
 /\* 基底条件 \*/  
 if (n <= 1)  
 {  
 return 1;  
 }  
   
 /\* 再帰呼び出し \*/  
 return n \* factorial(n - 1);  
}  
  
/\* フィボナッチ数列を計算する再帰関数 \*/  
int fibonacci(int n)  
{  
 /\* 基底条件 \*/  
 if (n <= 1)  
 {  
 return n;  
 }  
   
 /\* 再帰呼び出し \*/  
 return fibonacci(n - 1) + fibonacci(n - 2);  
}

### 関数のスコープと生存期間

#### ローカル変数とグローバル変数

#include <stdio.h>  
  
/\* グローバル変数 \*/  
int global\_count = 0;  
  
void increment\_global(void)  
{  
 global\_count++; /\* グローバル変数にアクセス \*/  
}  
  
void local\_example(void)  
{  
 int local\_var = 10; /\* ローカル変数 \*/  
   
 printf("ローカル変数: %d\n", local\_var);  
 /\* 関数を抜けるとlocal\_varは消滅 \*/  
}  
  
/\* static変数の例 \*/  
void counter(void)  
{  
 static int count = 0; /\* static変数は値を保持 \*/  
 count++;  
 printf("呼び出し回数: %d\n", count);  
}  
  
int main(void)  
{  
 int i;  
   
 /\* staticの効果を確認 \*/  
 for (i = 0; i < 3; i++)  
 {  
 counter();  
 }  
   
 return 0;  
}

### 関数ポインタの基礎

関数へのポインタを使うことで、関数を変数のように扱えます。

#include <stdio.h>  
  
/\* 計算用の関数 \*/  
int add(int a, int b) { return a + b; }  
int subtract(int a, int b) { return a - b; }  
int multiply(int a, int b) { return a \* b; }  
  
int main(void)  
{  
 /\* 関数ポインタの宣言 \*/  
 int (\*operation)(int, int);  
 int x = 10, y = 5;  
   
 /\* 関数ポインタに関数を代入 \*/  
 operation = add;  
 printf("%d + %d = %d\n", x, y, operation(x, y));  
   
 operation = subtract;  
 printf("%d - %d = %d\n", x, y, operation(x, y));  
   
 operation = multiply;  
 printf("%d \* %d = %d\n", x, y, operation(x, y));  
   
 return 0;  
}

## 実例コード

完全な実装例は以下のファイルを参照してください：

### 基本的な関数の使い方

* [function\_basics.c](examples/function_basics.c) - C90準拠版
* [function\_basics\_c99.c](examples/function_basics_c99.c) - C99準拠版

### 高度な関数の使い方

* [advanced\_functions.c](examples/advanced_functions.c) - C90準拠版
* [advanced\_functions\_c99.c](examples/advanced_functions_c99.c) - C99準拠版

## コンパイルと実行

# 基本的な関数の例をコンパイル  
gcc -Wall -Wextra -pedantic -std=c90 examples/function\_basics.c -o function\_basics  
  
# 実行  
./function\_basics  
  
# C99版をコンパイル  
gcc -Wall -Wextra -pedantic -std=c99 examples/function\_basics\_c99.c -o function\_basics\_c99  
  
# 数学関数を使う場合は-lmを追加  
gcc -Wall -Wextra -pedantic examples/advanced\_functions.c -lm -o advanced\_functions

## 演習問題

この章の演習問題は<exercises/README.md>にあります。

## 注意事項

1. **関数プロトタイプ**: main関数より後に定義する関数は、必ずプロトタイプ宣言が必要
2. **配列の扱い**: 配列を関数に渡すときは、サイズ情報も一緒に渡す
3. **再帰の深さ**: 再帰関数は深くなりすぎるとスタックオーバーフローを起こす
4. **グローバル変数**: できるだけ使用を避け、必要な場合は慎重に使用する
5. **static変数**: 関数内でstatic変数を使うと、値が保持される

## 次のステップ

関数の基本を理解したら、以下のトピックに進みましょう：

1. より複雑な関数の設計パターン
2. 関数ポインタと高階関数
3. 可変長引数関数（stdarg.h）
4. インライン関数（C99以降）
5. ライブラリ関数の作成

# サンプルコード

## advanced\_functions.c

/\*  
 \* 高度な関数の使い方  
 \*   
 \* このプログラムは、再帰関数、ローカル変数とグローバル変数、  
 \* 配列を引数に取る関数の使い方を示します。  
 \*/  
#include <stdio.h>  
  
/\* グローバル変数 \*/  
int global\_counter = 0;  
  
/\* 関数プロトタイプ \*/  
int factorial(int n);  
int fibonacci(int n);  
void display\_array(int arr[], int size);  
int sum\_array(int arr[], int size);  
double average\_array(int arr[], int size);  
void increment\_counter(void);  
int local\_vs\_global\_demo(int param);  
  
/\* 階乗を計算する再帰関数 \*/  
int factorial(int n)  
{  
 if (n <= 1) {  
 return 1; /\* 基底条件 \*/  
 }  
 return n \* factorial(n - 1); /\* 再帰呼び出し \*/  
}  
  
/\* フィボナッチ数列を計算する再帰関数 \*/  
int fibonacci(int n)  
{  
 if (n <= 0) {  
 return 0;  
 }  
 if (n == 1) {  
 return 1;  
 }  
 return fibonacci(n - 1) + fibonacci(n - 2);  
}  
  
/\* 配列を表示する関数 \*/  
void display\_array(int arr[], int size)  
{  
 int i;  
 printf("配列の内容: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
}  
  
/\* 配列の合計を計算する関数 \*/  
int sum\_array(int arr[], int size)  
{  
 int sum = 0;  
 int i;  
   
 for (i = 0; i < size; i++) {  
 sum += arr[i];  
 }  
 return sum;  
}  
  
/\* 配列の平均を計算する関数 \*/  
double average\_array(int arr[], int size)  
{  
 if (size == 0) {  
 return 0.0;  
 }  
 return (double)sum\_array(arr, size) / size;  
}  
  
/\* グローバル変数を操作する関数 \*/  
void increment\_counter(void)  
{  
 global\_counter++;  
 printf("グローバルカウンター: %d\n", global\_counter);  
}  
  
/\* ローカル変数とグローバル変数のデモ \*/  
int local\_vs\_global\_demo(int param)  
{  
 int local\_var = 10; /\* ローカル変数 \*/  
   
 printf(" 関数内 - ローカル変数: %d\n", local\_var);  
 printf(" 関数内 - グローバル変数: %d\n", global\_counter);  
 printf(" 関数内 - 引数: %d\n", param);  
   
 local\_var += param;  
 global\_counter += param;  
   
 return local\_var;  
}  
  
int main(void)  
{  
 int i;  
   
 printf("=== 高度な関数の使用例 ===\n\n");  
   
 /\* 1. 再帰関数（階乗） \*/  
 printf("=== 再帰関数: 階乗 ===\n");  
 for (i = 0; i <= 7; i++) {  
 printf("%d! = %d\n", i, factorial(i));  
 }  
 printf("\n");  
   
 /\* 2. 再帰関数（フィボナッチ数列） \*/  
 printf("=== 再帰関数: フィボナッチ数列 ===\n");  
 printf("最初の10個のフィボナッチ数: ");  
 for (i = 0; i < 10; i++) {  
 printf("%d ", fibonacci(i));  
 }  
 printf("\n\n");  
   
 /\* 3. 配列を引数に取る関数 \*/  
 printf("=== 配列を引数に取る関数 ===\n");  
 int numbers[] = {5, 12, 8, 23, 16, 9, 14, 7};  
 int size = sizeof(numbers) / sizeof(numbers[0]);  
   
 display\_array(numbers, size);  
 printf("合計: %d\n", sum\_array(numbers, size));  
 printf("平均: %.2f\n", average\_array(numbers, size));  
 printf("\n");  
   
 /\* 4. ローカル変数とグローバル変数 \*/  
 printf("=== ローカル変数とグローバル変数 ===\n");  
 printf("初期状態のグローバル変数: %d\n", global\_counter);  
   
 printf("\n1回目の関数呼び出し:\n");  
 int result1 = local\_vs\_global\_demo(5);  
 printf(" 戻り値: %d\n", result1);  
   
 printf("\n2回目の関数呼び出し:\n");  
 int result2 = local\_vs\_global\_demo(3);  
 printf(" 戻り値: %d\n", result2);  
   
 printf("\n最終的なグローバル変数: %d\n\n", global\_counter);  
   
 /\* 5. グローバル変数を操作する関数 \*/  
 printf("=== グローバル変数の操作 ===\n");  
 for (i = 0; i < 5; i++) {  
 increment\_counter();  
 }  
 printf("\n");  
   
 /\* 6. 関数の組み合わせ使用 \*/  
 printf("=== 関数の組み合わせ使用 ===\n");  
   
 /\* 配列の各要素の階乗を計算 \*/  
 int sumall\_numbers[] = {1, 2, 3, 4, 5};  
 int sumall\_size = sizeof(sumall\_numbers) / sizeof(sumall\_numbers[0]);  
   
 printf("元の配列: ");  
 display\_array(sumall\_numbers, sumall\_size);  
   
 printf("各要素の階乗:\n");  
 for (i = 0; i < sumall\_size; i++) {  
 printf("%d! = %d\n", sumall\_numbers[i], factorial(sumall\_numbers[i]));  
 }  
 printf("\n");  
   
 /\* 7. 再帰の深さの制限例 \*/  
 printf("=== 再帰の制限例 ===\n");  
 printf("大きな数での階乗計算:\n");  
 for (i = 10; i <= 12; i++) {  
 printf("%d! = %d\n", i, factorial(i));  
 }  
 printf("注意: 大きな数では整数オーバーフローが発生する可能性があります\n");  
   
 return 0;  
}```  
  
## function\_basics.c  
  
```c  
/\*  
 \* 関数の基本的な使い方  
 \*   
 \* このプログラムは、関数の定義、呼び出し、  
 \* 引数と戻り値の基本的な使い方を示します。  
 \*/  
#include <stdio.h>  
  
/\* 関数プロトタイプ宣言 \*/  
int add(int a, int b);  
int subtract(int a, int b);  
int multiply(int a, int b);  
double divide(int a, int b);  
void print\_message(void);  
void print\_number(int num);  
int square(int n);  
int max(int a, int b);  
  
/\* 2つの数の加算 \*/  
int add(int a, int b)  
{  
 return a + b;  
}  
  
/\* 2つの数の減算 \*/  
int subtract(int a, int b)  
{  
 return a - b;  
}  
  
/\* 2つの数の乗算 \*/  
int multiply(int a, int b)  
{  
 return a \* b;  
}  
  
/\* 2つの数の除算（浮動小数点で返す） \*/  
double divide(int a, int b)  
{  
 if (b == 0) {  
 printf("エラー: ゼロで割ることはできません\n");  
 return 0.0;  
 }  
 return (double)a / b;  
}  
  
/\* 戻り値なし、引数なしの関数 \*/  
void print\_message(void)  
{  
 printf("これは関数から出力されたメッセージです\n");  
}  
  
/\* 戻り値なし、引数ありの関数 \*/  
void print\_number(int num)  
{  
 printf("数値: %d\n", num);  
}  
  
/\* 数の平方を計算 \*/  
int square(int n)  
{  
 return n \* n;  
}  
  
/\* 2つの数の最大値を返す \*/  
int max(int a, int b)  
{  
 if (a > b) {  
 return a;  
 } else {  
 return b;  
 }  
}  
  
int main(void)  
{  
 int x = 15;  
 int y = 7;  
 int result;  
 double dresult;  
   
 printf("=== 関数の基本操作 ===\n");  
 printf("x = %d, y = %d\n\n", x, y);  
   
 /\* 1. 基本的な算術関数 \*/  
 printf("=== 基本的な算術関数 ===\n");  
 result = add(x, y);  
 printf("%d + %d = %d\n", x, y, result);  
   
 result = subtract(x, y);  
 printf("%d - %d = %d\n", x, y, result);  
   
 result = multiply(x, y);  
 printf("%d x %d = %d\n", x, y, result);  
   
 dresult = divide(x, y);  
 printf("%d / %d = %.2f\n", x, y, dresult);  
 printf("\n");  
   
 /\* 2. void関数の使用 \*/  
 printf("=== void関数の使用 ===\n");  
 print\_message();  
 print\_number(42);  
 print\_number(x + y);  
 printf("\n");  
   
 /\* 3. 関数の戻り値を直接使用 \*/  
 printf("=== 関数の戻り値を直接使用 ===\n");  
 printf("%d の平方は %d です\n", x, square(x));  
 printf("%d と %d の最大値は %d です\n", x, y, max(x, y));  
   
 /\* 関数呼び出しをネスト \*/  
 printf("(%d + %d) の平方は %d です\n", x, y, square(add(x, y)));  
 printf("\n");  
   
 /\* 4. 関数を使った計算の例 \*/  
 printf("=== 複雑な計算例 ===\n");  
 {  
 /\* 三角形の面積計算（底辺 x 高さ / 2） \*/  
 int base = 10;  
 int height = 8;  
 double area = divide(multiply(base, height), 2);  
 printf("底辺%d、高さ%dの三角形の面積: %.1f\n", base, height, area);  
   
 /\* 円の面積の近似計算 \*/  
 int radius = 5;  
 double pi = 3.14159;  
 double circle\_area = pi \* square(radius);  
 printf("半径%dの円の面積（近似）: %.2f\n", radius, circle\_area);  
 printf("\n");  
 }  
   
 /\* 5. デモ用の固定値を使った関数 \*/  
 printf("=== デモ用固定値での関数使用 ===\n");  
 int num1 = 15, num2 = 4;  
   
 printf("使用する値: %d と %d\n", num1, num2);  
   
 printf("\n計算結果:\n");  
 printf("%d + %d = %d\n", num1, num2, add(num1, num2));  
 printf("%d - %d = %d\n", num1, num2, subtract(num1, num2));  
 printf("%d x %d = %d\n", num1, num2, multiply(num1, num2));  
   
 if (num2 != 0) {  
 printf("%d / %d = %.2f\n", num1, num2, divide(num1, num2));  
 }  
   
 printf("最大値: %d\n", max(num1, num2));  
   
 return 0;  
}