# 第4章 演算子

## 対応C規格

* **主要対象:** C90
* **学習内容:** 算術演算子、関係演算子、論理演算子、ビット演算子、演算子の優先順位

## 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：

* さまざまな演算子の種類と使い方を理解する
* 演算子の優先順位と結合規則を把握する
* 複雑な式を正しく記述できる
* インクリメント・デクリメント演算子を適切に使える
* ビット演算の基本を理解する

## 理論解説

### 算術演算子

基本的な数値計算を実行する演算子です。

| 演算子 | 意味 | 例 | 結果 |
| --- | --- | --- | --- |
| + | 加算 | 5 + 3 | 8 |
| - | 減算 | 5 - 3 | 2 |
| \* | 乗算 | 5 \* 3 | 15 |
| / | 除算 | 7 / 3 | 2 (整数除算) |
| % | 剰余 | 7 % 3 | 1 |

#### 除算の注意点

int a = 7, b = 3;  
int result = a / b; /\* 結果: 2 (整数除算) \*/  
double result = a / b; /\* 結果: 2.0 (整数除算後に型変換) \*/  
double result = (double)a / b; /\* 結果: 2.333... (実数除算) \*/

### 代入演算子

値を変数に代入する演算子です。

| 演算子 | 意味 | 例 | 等価な記述 |
| --- | --- | --- | --- |
| = | 代入 | a = 5 | - |
| += | 加算代入 | a += 3 | a = a + 3 |
| -= | 減算代入 | a -= 2 | a = a - 2 |
| \*= | 乗算代入 | a \*= 2 | a = a \* 2 |
| /= | 除算代入 | a /= 3 | a = a / 3 |
| %= | 剰余代入 | a %= 4 | a = a % 4 |

int count = 10;  
count += 5; /\* count は 15 になる \*/  
count \*= 2; /\* count は 30 になる \*/

### インクリメント・デクリメント演算子

変数の値を増減させる演算子です。

| 演算子 | 意味 | 前置 | 後置 |
| --- | --- | --- | --- |
| ++ | インクリメント | ++a | a++ |
| -- | デクリメント | --a | a-- |

#### 前置と後置の違い

int a = 5, b = 5;  
int x, y;  
  
x = ++a; /\* a を先にインクリメント, x = 6, a = 6 \*/  
y = b++; /\* b の値を先に使用, y = 5, b = 6 \*/

**C99版での詳細:** [increment\_decrement\_c99.c](solutions/increment_decrement_c99.c)

### 関係演算子（比較する）

関係演算子は、2つの値を比較して「正しい（真）」か「正しくない（偽）」かを判定します。日常生活での「大きい・小さい」「同じ・違う」の判断をプログラムで行うための演算子です。

#### C言語における真偽の表現

C言語では： - **真（True）** = 0以外の値（通常は1） - **偽（False）** = 0

| 演算子 | 意味 | 例 | 結果 | 日常での例 |
| --- | --- | --- | --- | --- |
| == | 等しい | 5 == 3 | 0 (偽) | パスワードが一致するか |
| != | 等しくない | 5 != 3 | 1 (真) | 在庫が0でないか |
| < | より小さい | 5 < 3 | 0 (偽) | 年齢が18歳未満か |
| <= | 以下 | 5 <= 3 | 0 (偽) | 体重が標準以下か |
| > | より大きい | 5 > 3 | 1 (真) | 得点が合格点より高いか |
| >= | 以上 | 5 >= 3 | 1 (真) | 残高が購入金額以上か |

#### 実用的な例

/\* パスワードチェック \*/  
int password = 1234;  
int input = 1234;  
if (password == input) {  
 printf("ログイン成功\n");  
}  
  
/\* 在庫管理 \*/  
int stock = 5;  
if (stock > 0) {  
 printf("在庫あり（%d個）\n", stock);  
} else {  
 printf("在庫切れ\n");  
}  
  
/\* 年齢制限チェック \*/  
int age = 16;  
if (age >= 18) {  
 printf("成人向けコンテンツを表示\n");  
} else {  
 printf("アクセス制限中\n");  
}

#### よくある間違い：= と == の混同

int x = 5;  
  
/\* 間違い：代入になってしまう \*/  
if (x = 10) { /\* xに10を代入し、10（真）として評価 \*/  
 printf("この部分は必ず実行される\n");  
}  
  
/\* 正しい：比較 \*/  
if (x == 10) { /\* xが10と等しいか比較 \*/  
 printf("xは10です\n");  
}

**覚え方のコツ**： - = は「代入（入れる）」の矢印 → - == は「等しいか確認」の二重チェック

### 論理演算子（条件を組み合わせる）

論理演算子は、複数の条件を組み合わせて複雑な判断を行うための演算子です。

| 演算子 | 意味 | 例 | 説明 | 日常での例 |
| --- | --- | --- | --- | --- |
| && | かつ（AND） | a && b | 両方とも真なら真 | 「晴れ」かつ「暖かい」なら外出 |
| || | または（OR） | a || b | どちらか真なら真 | 「土曜」または「日曜」なら休み |
| ! | でない（NOT） | !a | 真偽を反転 | 「雨でない」なら洗濯 |

#### 真理値表で理解する

**AND（&&）の動作** | A | B | A && B | |—|—|——–| | 真 | 真 | 真 | | 真 | 偽 | 偽 | | 偽 | 真 | 偽 | | 偽 | 偽 | 偽 |

**OR（||）の動作** | A | B | A || B | |—|—|———-| | 真 | 真 | 真 | | 真 | 偽 | 真 | | 偽 | 真 | 真 | | 偽 | 偽 | 偽 |

#### 実用的な例

/\* 営業時間チェック（9時〜17時） \*/  
int hour = 14;  
if (hour >= 9 && hour <= 17) {  
 printf("営業中です\n");  
}  
  
/\* 休日判定 \*/  
int day = 6; /\* 1=月曜...7=日曜 \*/  
if (day == 6 || day == 7) {  
 printf("週末です\n");  
}  
  
/\* 割引条件（学生でない一般客） \*/  
int is\_student = 0;  
if (!is\_student) {  
 printf("一般料金です\n");  
}  
  
/\* 複雑な条件：優待条件 \*/  
int age = 65;  
int is\_member = 1;  
if ((age >= 65 || age < 18) && is\_member) {  
 printf("優待料金が適用されます\n");  
}

#### 短絡評価（ショートサーキット）

論理演算子には「短絡評価」という重要な特性があります。結果が確定した時点で、残りの評価をスキップします：

/\* &&の短絡評価：左が偽なら右は評価しない \*/  
int a = 0, b = 10;  
if (a != 0 && b / a > 5) { /\* a が 0 なので b/a は計算されない \*/  
 /\* ゼロ除算エラーを回避できる \*/  
}  
  
/\* ||の短絡評価：左が真なら右は評価しない \*/  
int is\_admin = 1;  
int has\_permission = 0;  
if (is\_admin || check\_permission()) { /\* is\_adminが真なので関数は呼ばれない \*/  
 printf("アクセス許可\n");  
}

### ビット演算子

ビットレベルで値を操作する演算子です。

| 演算子 | 意味 | 例 | 説明 |
| --- | --- | --- | --- |
| & | ビットAND | a & b | 対応するビットが両方1の場合1 |
| | | ビットOR | a | b | 対応するビットのいずれかが1の場合1 |
| ^ | ビットXOR | a ^ b | 対応するビットが異なる場合1 |
| ~ | ビット反転 | ~a | 各ビットを反転 |
| << | 左シフト | a << 2 | ビットを左に2つシフト |
| >> | 右シフト | a >> 1 | ビットを右に1つシフト |

unsigned char a = 5; /\* 00000101 \*/  
unsigned char b = 3; /\* 00000011 \*/  
  
printf("a & b = %d\n", a & b); /\* 1 (00000001) \*/  
printf("a | b = %d\n", a | b); /\* 7 (00000111) \*/  
printf("a ^ b = %d\n", a ^ b); /\* 6 (00000110) \*/  
printf("~a = %d\n", ~a); /\* 250 (11111010) \*/

**C99版での詳細:** [bitwise\_demo\_c99.c](examples/bitwise_demo_c99.c)

### 条件演算子（三項演算子）

条件に基づいて値を選択する演算子です。

条件 ? 真の場合の値 : 偽の場合の値

int a = 3, b = 7;  
int max = (a > b) ? a : b; /\* b が大きいので max = 7 \*/  
  
printf("大きい方: %d\n", max);

**C99版での詳細:** [conditional\_operator\_c99.c](solutions/conditional_operator_c99.c)

### sizeof演算子

データ型や変数のサイズを取得する演算子です。

printf("int のサイズ: %lu バイト\n", (unsigned long)sizeof(int));  
printf("double のサイズ: %lu バイト\n", (unsigned long)sizeof(double));  
  
int arr[10];  
printf("配列のサイズ: %lu バイト\n", (unsigned long)sizeof(arr));

### 演算子の優先順位

演算子には優先順位があり、計算の順序に影響します。

| 優先順位 | 演算子 | 結合規則 |
| --- | --- | --- |
| 1 | () [] -> . | 左から右 |
| 2 | ! ~ ++ -- + - \* & sizeof (型) | 右から左 |
| 3 | \* / % | 左から右 |
| 4 | + - | 左から右 |
| 5 | << >> | 左から右 |
| 6 | < <= > >= | 左から右 |
| 7 | == != | 左から右 |
| 8 | & | 左から右 |
| 9 | ^ | 左から右 |
| 10 | | | 左から右 |
| 11 | && | 左から右 |
| 12 | || | 左から右 |
| 13 | ?: | 右から左 |
| 14 | = += -= \*= /= %= &= ^= |= <<= >>= | 右から左 |
| 15 | , | 左から右 |

#### 優先順位の例

int result = 2 + 3 \* 4; /\* 結果: 14 (乗算が先) \*/  
int result = (2 + 3) \* 4; /\* 結果: 20 (括弧が先) \*/

**C99版での詳細:** [precedence\_demo\_c99.c](examples/precedence_demo_c99.c)

## サンプルコード

### 演算子の基本使用例

プログラムファイル: [operators\_demo.c](examples/operators_demo.c)  
C99版: [operators\_demo\_c99.c](examples/operators_demo_c99.c)

さまざまな演算子の使用方法を学習します。

### ビット演算の実例

プログラムファイル: [bitwise\_demo.c](examples/bitwise_demo.c)  
C99版: [bitwise\_demo\_c99.c](examples/bitwise_demo_c99.c)

ビット演算の動作を可視化して確認します。

### 演算子優先順位の確認

プログラムファイル: [precedence\_demo.c](examples/precedence_demo.c)  
C99版: [precedence\_demo\_c99.c](examples/precedence_demo_c99.c)

演算子の優先順位による計算結果の違いを確認します。

### コンパイルと実行

# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic examples/operators\_demo.c -o operators\_demo  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic examples/operators\_demo\_c99.c -o operators\_demo\_c99  
  
# 実行  
./operators\_demo

## 演習課題

演習課題の詳細は <exercises/README.md> を参照してください。

### 基礎問題

1. **四則演算計算機**
   * 2つの数値に対してすべての算術演算を実行するプログラムを作成してください
2. **比較と論理演算**
   * 2つの数値の中から最大値を求めるプログラムを作成してください
3. **インクリメント・デクリメント**
   * 前置と後置の違いを確認するプログラムを作成してください

### 応用問題

1. **ビット操作**
   * 整数の各ビットを表示するプログラムを作成してください
2. **条件演算子の活用**
   * 三項演算子を使って複雑な条件分岐を実装してください
3. **演算子優先順位**
   * 複雑な式の計算順序を確認するプログラムを作成してください

## 解答例

各演習問題の解答例は <solutions/> ディレクトリにあります：

* [increment\_decrement.c](solutions/increment_decrement.c) / [C99版](solutions/increment_decrement_c99.c)
* [operator\_precedence.c](solutions/operator_precedence.c) / [C99版](solutions/operator_precedence_c99.c)
* [conditional\_operator.c](solutions/conditional_operator.c) / [C99版](solutions/conditional_operator_c99.c)

## 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  
  
# 特定のプログラムをコンパイル  
make operators\_demo  
  
# C99版をコンパイル  
make operators\_demo\_c99  
  
# 全て実行  
make run-all  
  
# クリーンアップ  
make clean

## 規格による違い

### C90での制限事項

* ビット演算子は整数型に対してのみ使用可能
* 論理演算子の結果は0または1
* ブロック先頭でのみ変数宣言可能

### C99以降の拡張

* \_Bool型の追加により、論理値の取り扱いが明確化
* 複素数型に対する演算子の追加
* forループ内での変数宣言が可能
* インライン関数の使用が可能

## よくある間違い

### 1. 代入と比較の混同

/\* NG: 代入を条件文で使用 \*/  
if (a = 5) /\* 常に真になる \*/  
 /\* ... \*/  
  
/\* OK: 比較演算子を使用 \*/  
if (a == 5) /\* a が 5 と等しいかチェック \*/  
 /\* ... \*/

### 2. 整数除算の結果

/\* NG: 期待した結果にならない \*/  
double result = 5 / 2; /\* 結果: 2.0 \*/  
  
/\* OK: 実数除算を使用 \*/  
double result = 5.0 / 2.0; /\* 結果: 2.5 \*/

### 3. 演算子の優先順位

/\* NG: 意図しない計算順序 \*/  
int result = a + b \* c + d; /\* b \* c が先に計算される \*/  
  
/\* OK: 括弧で明示 \*/  
int result = (a + b) \* (c + d);

### 4. ビット演算の優先順位

/\* NG: 期待しない結果 \*/  
if (flags & MASK == 1) /\* flags & (MASK == 1) として評価される \*/  
  
/\* OK: 括弧で明示 \*/  
if ((flags & MASK) == 1) /\* 正しい評価順序 \*/

## 次の章へ

演算子と式を理解したら、[制御構造（条件分岐）](../control-if/README.md) に進んでください。

## 参考資料

* [C言語演算子リファレンス](https://ja.cppreference.com/w/c/language/operator_precedence)
* [ビット演算詳細](https://ja.cppreference.com/w/c/language/operator_arithmetic)
* [演算子優先順位表](https://ja.cppreference.com/w/c/language/operator_precedence)

# サンプルコード

## bitwise\_demo.c

/\*  
 \* bitwise\_demo.c  
 \* ビット演算の動作を可視化して確認  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\* ビットパターンを表示する関数 \*/  
void print\_bits(unsigned char value) {  
 int i;  
 for (i = 7; i >= 0; i--) {  
 printf("%d", (value >> i) & 1);  
 if (i == 4) printf(" "); /\* 読みやすさのため4ビットごとに空白 \*/  
 }  
}  
  
int main(void) {  
 unsigned char a = 5; /\* 00000101 \*/  
 unsigned char b = 3; /\* 00000011 \*/  
 unsigned char result;  
   
 printf("===== ビット演算のデモ =====\n");  
 printf("初期値:\n");  
 printf(" a = %3d : ", a);  
 print\_bits(a);  
 printf("\n");  
 printf(" b = %3d : ", b);  
 print\_bits(b);  
 printf("\n\n");  
   
 /\* ビットAND \*/  
 result = a & b;  
 printf("ビットAND (a & b):\n");  
 printf(" ");  
 print\_bits(a);  
 printf("\n& ");  
 print\_bits(b);  
 printf("\n---------------\n ");  
 print\_bits(result);  
 printf(" = %d\n\n", result);  
   
 /\* ビットOR \*/  
 result = a | b;  
 printf("ビットOR (a | b):\n");  
 printf(" ");  
 print\_bits(a);  
 printf("\n| ");  
 print\_bits(b);  
 printf("\n---------------\n ");  
 print\_bits(result);  
 printf(" = %d\n\n", result);  
   
 /\* ビットXOR \*/  
 result = a ^ b;  
 printf("ビットXOR (a ^ b):\n");  
 printf(" ");  
 print\_bits(a);  
 printf("\n^ ");  
 print\_bits(b);  
 printf("\n---------------\n ");  
 print\_bits(result);  
 printf(" = %d\n\n", result);  
   
 /\* ビット反転 \*/  
 result = ~a;  
 printf("ビット反転 (~a):\n");  
 printf("~ ");  
 print\_bits(a);  
 printf("\n---------------\n ");  
 print\_bits(result);  
 printf(" = %d\n\n", result);  
   
 /\* 左シフト \*/  
 printf("===== シフト演算のデモ =====\n");  
 a = 5; /\* リセット \*/  
   
 result = a << 1;  
 printf("左シフト (a << 1):\n");  
 printf(" ");  
 print\_bits(a);  
 printf(" << 1\n= ");  
 print\_bits(result);  
 printf(" = %d (2倍)\n\n", result);  
   
 result = a << 2;  
 printf("左シフト (a << 2):\n");  
 printf(" ");  
 print\_bits(a);  
 printf(" << 2\n= ");  
 print\_bits(result);  
 printf(" = %d (4倍)\n\n", result);  
   
 /\* 右シフト \*/  
 a = 20; /\* 00010100 \*/  
   
 result = a >> 1;  
 printf("右シフト (a >> 1):\n");  
 printf(" ");  
 print\_bits(a);  
 printf(" >> 1\n= ");  
 print\_bits(result);  
 printf(" = %d (1/2)\n\n", result);  
   
 result = a >> 2;  
 printf("右シフト (a >> 2):\n");  
 printf(" ");  
 print\_bits(a);  
 printf(" >> 2\n= ");  
 print\_bits(result);  
 printf(" = %d (1/4)\n\n", result);  
   
 /\* 実用例：フラグ管理 \*/  
 printf("===== ビット演算の実用例（フラグ管理） =====\n");  
 #define FLAG\_A 0x01 /\* 0000 0001 \*/  
 #define FLAG\_B 0x02 /\* 0000 0010 \*/  
 #define FLAG\_C 0x04 /\* 0000 0100 \*/  
 #define FLAG\_D 0x08 /\* 0000 1000 \*/  
   
 unsigned char flags = 0;  
   
 printf("初期状態: ");  
 print\_bits(flags);  
 printf("\n");  
   
 /\* フラグを立てる \*/  
 flags |= FLAG\_A;  
 printf("FLAG\_A を立てる: ");  
 print\_bits(flags);  
 printf("\n");  
   
 flags |= FLAG\_C;  
 printf("FLAG\_C を立てる: ");  
 print\_bits(flags);  
 printf("\n");  
   
 /\* フラグのチェック \*/  
 if (flags & FLAG\_A) {  
 printf("FLAG\_A は ON です\n");  
 }  
 if (flags & FLAG\_B) {  
 printf("FLAG\_B は ON です\n");  
 } else {  
 printf("FLAG\_B は OFF です\n");  
 }  
   
 /\* フラグをクリア \*/  
 flags &= ~FLAG\_A;  
 printf("FLAG\_A をクリア: ");  
 print\_bits(flags);  
 printf("\n");  
   
 /\* フラグをトグル \*/  
 flags ^= FLAG\_B;  
 printf("FLAG\_B をトグル: ");  
 print\_bits(flags);  
 printf("\n");  
   
 return 0;  
}

## operators\_demo.c

/\*  
 \* operators\_demo.c  
 \* さまざまな演算子の使用方法を学習  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void) {  
 int a = 10, b = 3;  
 double x = 10.0, y = 3.0;  
 int result;  
   
 printf("===== 算術演算子のデモ =====\n");  
 printf("整数演算 (a = %d, b = %d):\n", a, b);  
 printf(" a + b = %d\n", a + b);  
 printf(" a - b = %d\n", a - b);  
 printf(" a \* b = %d\n", a \* b);  
 printf(" a / b = %d (整数除算)\n", a / b);  
 printf(" a %% b = %d (剰余)\n", a % b);  
   
 printf("\n実数演算 (x = %.1f, y = %.1f):\n", x, y);  
 printf(" x / y = %.2f (実数除算)\n", x / y);  
   
 printf("\n===== 代入演算子のデモ =====\n");  
 result = a;  
 printf("result = %d\n", result);  
 result += 5;  
 printf("result += 5 -> %d\n", result);  
 result -= 3;  
 printf("result -= 3 -> %d\n", result);  
 result \*= 2;  
 printf("result \*= 2 -> %d\n", result);  
 result /= 4;  
 printf("result /= 4 -> %d\n", result);  
 result %= 3;  
 printf("result %%= 3 -> %d\n", result);  
   
 printf("\n===== インクリメント・デクリメント演算子 =====\n");  
 a = 5;  
 printf("初期値: a = %d\n", a);  
 printf("a++ = %d (後置インクリメント)\n", a++);  
 printf("現在の a = %d\n", a);  
 printf("++a = %d (前置インクリメント)\n", ++a);  
 printf("現在の a = %d\n", a);  
   
 a = 5;  
 printf("\n初期値: a = %d\n", a);  
 printf("a-- = %d (後置デクリメント)\n", a--);  
 printf("現在の a = %d\n", a);  
 printf("--a = %d (前置デクリメント)\n", --a);  
 printf("現在の a = %d\n", a);  
   
 printf("\n===== 関係演算子のデモ =====\n");  
 a = 5; b = 3;  
 printf("a = %d, b = %d のとき:\n", a, b);  
 printf(" a == b : %d\n", a == b);  
 printf(" a != b : %d\n", a != b);  
 printf(" a < b : %d\n", a < b);  
 printf(" a <= b : %d\n", a <= b);  
 printf(" a > b : %d\n", a > b);  
 printf(" a >= b : %d\n", a >= b);  
   
 printf("\n===== 論理演算子のデモ =====\n");  
 a = 1; b = 0;  
 printf("a = %d (真), b = %d (偽) のとき:\n", a, b);  
 printf(" a && b : %d\n", a && b);  
 printf(" a || b : %d\n", a || b);  
 printf(" !a : %d\n", !a);  
 printf(" !b : %d\n", !b);  
   
 /\* 短絡評価のデモ \*/  
 printf("\n短絡評価のデモ:\n");  
 a = 0; b = 5;  
 printf("a = %d, b = %d のとき:\n", a, b);  
 if (a != 0 && b / a > 2) {  
 printf(" 条件は真です\n");  
 } else {  
 printf(" 条件は偽です (a が 0 なので b/a は評価されない)\n");  
 }  
   
 printf("\n===== 条件演算子（三項演算子）のデモ =====\n");  
 a = 10; b = 20;  
 result = (a > b) ? a : b;  
 printf("a = %d, b = %d のとき:\n", a, b);  
 printf(" (a > b) ? a : b = %d (大きい方の値)\n", result);  
   
 printf("\n===== sizeof演算子のデモ =====\n");  
 printf("基本データ型のサイズ:\n");  
 printf(" sizeof(char) = %lu バイト\n", (unsigned long)sizeof(char));  
 printf(" sizeof(short) = %lu バイト\n", (unsigned long)sizeof(short));  
 printf(" sizeof(int) = %lu バイト\n", (unsigned long)sizeof(int));  
 printf(" sizeof(long) = %lu バイト\n", (unsigned long)sizeof(long));  
 printf(" sizeof(float) = %lu バイト\n", (unsigned long)sizeof(float));  
 printf(" sizeof(double) = %lu バイト\n", (unsigned long)sizeof(double));  
   
 return 0;  
}

## precedence\_demo.c

/\*  
 \* precedence\_demo.c  
 \* 演算子の優先順位による計算結果の違いを確認  
 \* C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void) {  
 int a = 2, b = 3, c = 4, d = 5;  
 int result;  
   
 printf("===== 演算子優先順位のデモ =====\n");  
 printf("初期値: a = %d, b = %d, c = %d, d = %d\n\n", a, b, c, d);  
   
 /\* 算術演算子の優先順位 \*/  
 printf("【算術演算子の優先順位】\n");  
   
 result = a + b \* c;  
 printf("a + b \* c = %d + %d \* %d = %d\n", a, b, c, result);  
 printf(" → 乗算が先: %d + (%d \* %d) = %d + %d = %d\n\n", a, b, c, b \* c, b \* c, result);  
   
 result = (a + b) \* c;  
 printf("(a + b) \* c = (%d + %d) \* %d = %d\n", a, b, c, result);  
 printf(" → 括弧が先: (%d + %d) \* %d = %d \* %d = %d\n\n", a, b, c, a + b, a + b, result);  
   
 result = a \* b + c \* d;  
 printf("a \* b + c \* d = %d \* %d + %d \* %d = %d\n", a, b, c, d, result);  
 printf(" → 乗算が先: (%d \* %d) + (%d \* %d) = %d + %d = %d\n\n",   
 a, b, c, d, a \* b, c \* d, result);  
   
 /\* 複雑な式の例 \*/  
 printf("【複雑な式の評価順序】\n");  
   
 result = a + b \* c - d;  
 printf("a + b \* c - d = %d + %d \* %d - %d = %d\n", a, b, c, d, result);  
 printf(" 評価順序:\n");  
 printf(" 1) b \* c = %d \* %d = %d\n", b, c, b \* c);  
 printf(" 2) a + (b \* c) = %d + %d = %d\n", a, b \* c, a + b \* c);  
 printf(" 3) (a + b \* c) - d = %d - %d = %d\n\n", a + b \* c, d, result);  
   
 /\* 関係演算子と論理演算子 \*/  
 printf("【関係演算子と論理演算子の優先順位】\n");  
   
 result = a < b && c > d;  
 printf("a < b && c > d = %d < %d && %d > %d = %d\n", a, b, c, d, result);  
 printf(" 評価順序:\n");  
 printf(" 1) a < b = %d < %d = %d (真)\n", a, b, a < b);  
 printf(" 2) c > d = %d > %d = %d (偽)\n", c, d, c > d);  
 printf(" 3) (a < b) && (c > d) = %d && %d = %d\n\n", a < b, c > d, result);  
   
 /\* ビット演算子と比較演算子 \*/  
 printf("【ビット演算子と比較演算子の優先順位】\n");  
   
 /\* 注意：この例は優先順位を示すためのもの \*/  
 result = (a & b) == 2; /\* 括弧を使って意図を明確にする \*/  
 printf("(a & b) == 2 = (%d & %d) == 2 = %d\n", a, b, result);  
 printf(" 評価順序:\n");  
 printf(" 1) a & b = %d & %d = %d\n", a, b, a & b);  
 printf(" 2) (a & b) == 2 = %d == 2 = %d\n\n", a & b, result);  
   
 /\* 代入演算子の結合性 \*/  
 printf("【代入演算子の結合性（右結合）】\n");  
   
 a = b = c = 10;  
 printf("a = b = c = 10 の後: a = %d, b = %d, c = %d\n", a, b, c);  
 printf(" → 右から左に評価: a = (b = (c = 10))\n\n");  
   
 /\* インクリメントと他の演算子 \*/  
 printf("【インクリメント演算子の優先順位】\n");  
   
 a = 5;  
 result = ++a \* 2;  
 printf("a = 5; result = ++a \* 2;\n");  
 printf(" result = %d (前置++が先: 6 \* 2)\n", result);  
 printf(" a = %d\n\n", a);  
   
 a = 5;  
 result = a++ \* 2;  
 printf("a = 5; result = a++ \* 2;\n");  
 printf(" result = %d (後置++は後: 5 \* 2)\n", result);  
 printf(" a = %d\n\n", a);  
   
 /\* 括弧を使った明確な表現 \*/  
 printf("【推奨：括弧で意図を明確にする】\n");  
   
 a = 2; b = 3; c = 4; d = 1;  
   
 /\* 曖昧な式 \*/  
 result = a + b << 2 & c;  
 printf("曖昧: a + b << 2 & c = %d\n", result);  
   
 /\* 明確な式 \*/  
 result = ((a + b) << 2) & c;  
 printf("明確: ((a + b) << 2) & c = %d\n", result);  
 printf(" 評価順序:\n");  
 printf(" 1) a + b = %d + %d = %d\n", a, b, a + b);  
 printf(" 2) (a + b) << 2 = %d << 2 = %d\n", a + b, (a + b) << 2);  
 printf(" 3) ((a + b) << 2) & c = %d & %d = %d\n", (a + b) << 2, c, result);  
   
 return 0;  
}