# 第4章 演算子 - 演習問題

# 演習問題

## 基礎問題

### 演習4-1: 四則演算計算機

2つの整数を入力として受け取り、すべての算術演算（加算、減算、乗算、除算、剰余）の結果を表示するプログラムを作成してください。

**要件:** - ユーザーから2つの整数を入力 - 各演算の結果を見やすく表示 - ゼロ除算のチェックを含める

**期待される出力例:**

2つの整数を入力してください: 10 3  
10 + 3 = 13  
10 - 3 = 7  
10 \* 3 = 30  
10 / 3 = 3  
10 % 3 = 1

### 演習4-2: 比較と論理演算

2つの整数を入力として受け取り、関係演算子と論理演算子を使って最大値と最小値を求めるプログラムを作成してください。

**要件:** - 2つの整数を入力 - if文を使わず、条件演算子（三項演算子）のみで実装 - 最大値と最小値を表示

### 演習4-3: インクリメント・デクリメント

前置と後置のインクリメント・デクリメント演算子の違いを確認するプログラムを作成してください。

**要件:** - 変数に対して前置・後置の両方を使用 - 各操作後の値を表示 - 式の中での動作も確認

## 応用問題

### 演習4-4: ビット操作

整数を入力として受け取り、その数値の各ビットを表示し、特定のビット操作を実行するプログラムを作成してください。

**要件:** - 2進数表示 - 指定したビット位置のON/OFF確認 - 特定ビットの設定・クリア・トグル機能

### 演習4-5: 条件演算子の活用

3つの数値を入力として受け取り、条件演算子（三項演算子）のみを使って昇順に並び替えるプログラムを作成してください。

**要件:** - if文、switch文を使用しない - 条件演算子のネストを活用 - 並び替えた結果を表示

### 演習4-6: 演算子優先順位

複雑な式を含むプログラムを作成し、演算子の優先順位による計算結果の違いを確認してください。

**要件:** - 同じ数値で異なる式を評価 - 括弧の有無による結果の違いを表示 - 少なくとも3種類以上の演算子を使用

## 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 ex4\_1\_calculator.c, ex4\_2\_comparison.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 ex4\_1\_calculator.c -o ex4\_1\_calculator  
./ex4\_1\_calculator

## ヒント

* 演算子の優先順位に注意
* ゼロ除算やオーバーフローに対する考慮
* 可読性を意識したコードを心がける
* 適切なコメントを追加する

# 解答例

このディレクトリには演習問題の解答例が含まれています。

## 解答ファイル一覧

### 基礎問題

1. **演習4-1: 四則演算計算機**
   * <ex4_1_calculator.c> - C90準拠版
   * <ex4_1_calculator_c99.c> - C99準拠版
   * 2つの整数に対する基本的な算術演算を実行
2. **演習4-2: 比較と論理演算**
   * <ex4_2_comparison.c> - C90準拠版
   * <ex4_2_comparison_c99.c> - C99準拠版
   * 条件演算子のみで最大値・最小値を求める
3. **演習4-3: インクリメント・デクリメント**
   * <ex4_3_increment.c> - C90準拠版
   * <ex4_3_increment_c99.c> - C99準拠版
   * 前置と後置の違いを詳細に確認

### 応用問題

1. **演習4-4: ビット操作**
   * <ex4_4_bitwise.c> - C90準拠版
   * <ex4_4_bitwise_c99.c> - C99準拠版
   * ビット演算子を使った各種操作の実装
2. **演習4-5: 条件演算子の活用**
   * <ex4_5_conditional.c> - C90準拠版
   * <ex4_5_conditional_c99.c> - C99準拠版
   * 3つの数値を条件演算子のみで昇順に並び替え
3. **演習4-6: 演算子優先順位**
   * <ex4_6_precedence.c> - C90準拠版
   * <ex4_6_precedence_c99.c> - C99準拠版
   * 複雑な式での演算子優先順位の確認

## 学習のポイント

### C90版とC99版の主な違い

1. **変数宣言位置**
   * C90: ブロックの先頭でのみ宣言可能
   * C99: 使用箇所で宣言可能
2. **bool型の使用**
   * C90: int型で真偽値を表現
   * C99: stdbool.hによるbool型の使用
3. **inline関数**
   * C90: 未対応
   * C99: inline関数による効率化
4. **コメントスタイル**
   * C90: /\* \*/ のみ
   * C99: // も使用可能
5. **固定幅整数型**
   * C90: 標準整数型のみ
   * C99: stdint.hによるuint8\_t等の使用

### 各演習の重要概念

#### 演習4-1: 四則演算計算機

* ゼロ除算のエラー処理
* 整数除算と実数除算の違い
* 代入演算子の使い方

#### 演習4-2: 比較と論理演算

* 関係演算子の評価結果（0または1）
* 論理演算子による条件の組み合わせ
* 条件演算子によるif文なしの条件分岐

#### 演習4-3: インクリメント・デクリメント

* 前置と後置の評価タイミングの違い
* 式の中での副作用
* 未定義動作の回避

#### 演習4-4: ビット操作

* ビットマスクの作成と使用
* 特定ビットの設定・クリア・反転
* ビットシフトによる乗除算

#### 演習4-5: 条件演算子の活用

* ネストした条件演算子の使い方
* 複数条件の組み合わせ
* 可読性とのバランス

#### 演習4-6: 演算子優先順位

* 演算子の優先順位と結合規則
* 括弧による評価順序の制御
* よくある間違いとその回避方法

## コンパイルと実行

# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic ex4\_1\_calculator.c -o ex4\_1\_calculator  
  
# C99準拠でコンパイル  
gcc -std=c99 -Wall -Wextra -pedantic ex4\_1\_calculator\_c99.c -o ex4\_1\_calculator\_c99  
  
# 実行  
./ex4\_1\_calculator

## 注意事項

* すべての解答例は教育目的で書かれており、実用的なコードよりも理解しやすさを重視しています
* エラー処理は基本的なものに留めており、実際のプロダクションコードではより堅牢な処理が必要です
* 各解答ファイルには詳細なコメントが含まれているので、コードと合わせて読むことを推奨します ## ex4\_1\_calculator.c

/\*  
 \* ファイル名: ex4\_1\_calculator.c  
 \* 演習4-1: 四則演算計算機  
 \* 説明: 2つの整数を入力として受け取り、基本的な算術演算の結果を表示  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int num1, num2;  
 int add\_result, sub\_result, mul\_result;  
   
 printf("===== 四則演算計算機 =====\n");  
 printf("2つの整数を入力してください: ");  
   
 /\* 入力の取得 \*/  
 if (scanf("%d %d", &num1, &num2) != 2) {  
 printf("エラー: 正しい整数を入力してください\n");  
 return 1;  
 }  
   
 /\* 基本的な算術演算 \*/  
 add\_result = num1 + num2;  
 sub\_result = num1 - num2;  
 mul\_result = num1 \* num2;  
   
 /\* 結果の表示 \*/  
 printf("\n=== 計算結果 ===\n");  
 printf("%d + %d = %d\n", num1, num2, add\_result);  
 printf("%d - %d = %d\n", num1, num2, sub\_result);  
 printf("%d \* %d = %d\n", num1, num2, mul\_result);  
   
 /\* 除算と剰余（ゼロ除算チェック付き） \*/  
 if (num2 != 0) {  
 printf("%d / %d = %d (整数除算)\n", num1, num2, num1 / num2);  
 printf("%d %% %d = %d (剰余)\n", num1, num2, num1 % num2);  
   
 /\* 実数除算も表示 \*/  
 printf("%d / %d = %.2f (実数除算)\n",   
 num1, num2, (double)num1 / num2);  
 } else {  
 printf("%d / %d = エラー (ゼロ除算)\n", num1, num2);  
 printf("%d %% %d = エラー (ゼロ除算)\n", num1, num2);  
 }  
   
 /\* 追加情報の表示 \*/  
 printf("\n=== 追加情報 ===\n");  
 printf("入力値の絶対値: |%d| = %d, |%d| = %d\n",   
 num1, (num1 < 0) ? -num1 : num1,  
 num2, (num2 < 0) ? -num2 : num2);  
   
 /\* 代入演算子のデモンストレーション \*/  
 printf("\n=== 代入演算子のデモ ===\n");  
 {  
 int temp = num1;  
 printf("初期値: temp = %d\n", temp);  
   
 temp += num2;  
 printf("temp += %d -> temp = %d\n", num2, temp);  
   
 temp -= num2;  
 printf("temp -= %d -> temp = %d\n", num2, temp);  
   
 if (num2 != 0) {  
 temp \*= num2;  
 printf("temp \*= %d -> temp = %d\n", num2, temp);  
   
 temp /= num2;  
 printf("temp /= %d -> temp = %d\n", num2, temp);  
   
 temp %= num2;  
 printf("temp %%= %d -> temp = %d\n", num2, temp);  
 }  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_1\_calculator  
===== 四則演算計算機 =====  
2つの整数を入力してください: 10 3  
  
=== 計算結果 ===  
10 + 3 = 13  
10 - 3 = 7  
10 \* 3 = 30  
10 / 3 = 3 (整数除算)  
10 % 3 = 1 (剰余)  
10 / 3 = 3.33 (実数除算)  
  
=== 追加情報 ===  
入力値の絶対値: |10| = 10, |3| = 3  
  
=== 代入演算子のデモ ===  
初期値: temp = 10  
temp += 3 -> temp = 13  
temp -= 3 -> temp = 10  
temp \*= 3 -> temp = 30  
temp /= 3 -> temp = 10  
temp %= 3 -> temp = 1  
  
学習ポイント:  
1. 基本的な算術演算子（+, -, \*, /, %）の使用  
2. ゼロ除算のエラーチェック  
3. 整数除算と実数除算の違い  
4. 代入演算子（+=, -=, \*=, /=, %=）の動作  
5. 条件演算子による絶対値の計算  
\*/```  
  
## ex4\_2\_comparison.c  
  
```c  
/\*  
 \* ファイル名: ex4\_2\_comparison.c  
 \* 演習4-2: 比較と論理演算  
 \* 説明: 関係演算子と論理演算子を使って最大値と最小値を求める  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int num1, num2;  
 int max, min;  
 int comparison\_result;  
   
 printf("===== 比較と論理演算 =====\n");  
 printf("2つの整数を入力してください: ");  
   
 /\* 入力の取得 \*/  
 if (scanf("%d %d", &num1, &num2) != 2) {  
 printf("エラー: 正しい整数を入力してください\n");  
 return 1;  
 }  
   
 /\* 条件演算子（三項演算子）のみで最大値と最小値を求める \*/  
 max = (num1 > num2) ? num1 : num2;  
 min = (num1 < num2) ? num1 : num2;  
   
 printf("\n=== 結果 ===\n");  
 printf("入力値: %d, %d\n", num1, num2);  
 printf("最大値: %d\n", max);  
 printf("最小値: %d\n", min);  
   
 /\* 関係演算子の詳細な比較 \*/  
 printf("\n=== 関係演算子による比較 ===\n");  
 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\n", num1, num2, num1 <= num2);  
 printf("%d > %d : %d\n", num1, num2, num1 > num2);  
 printf("%d >= %d : %d\n", num1, num2, num1 >= num2);  
   
 /\* 論理演算子の使用例 \*/  
 printf("\n=== 論理演算子の応用 ===\n");  
   
 /\* 両方の数が正の数かチェック \*/  
 comparison\_result = (num1 > 0) && (num2 > 0);  
 printf("両方とも正の数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 少なくとも一方が正の数かチェック \*/  
 comparison\_result = (num1 > 0) || (num2 > 0);  
 printf("少なくとも一方が正の数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 両方の数が偶数かチェック \*/  
 comparison\_result = !(num1 % 2) && !(num2 % 2);  
 printf("両方とも偶数: %s\n", comparison\_result ? "はい" : "いいえ");  
   
 /\* 条件演算子のネストによる3つの値の比較（拡張例） \*/  
 printf("\n=== 条件演算子の応用 ===\n");  
 {  
 int third\_value = 0;  
 int max\_of\_three;  
 int min\_of\_three;  
   
 printf("3つ目の値として0を使用します\n");  
   
 /\* 3つの値の最大値（ネストした条件演算子） \*/  
 max\_of\_three = (num1 > num2) ?   
 ((num1 > third\_value) ? num1 : third\_value) :  
 ((num2 > third\_value) ? num2 : third\_value);  
   
 /\* 3つの値の最小値（ネストした条件演算子） \*/  
 min\_of\_three = (num1 < num2) ?   
 ((num1 < third\_value) ? num1 : third\_value) :  
 ((num2 < third\_value) ? num2 : third\_value);  
   
 printf("3つの値 (%d, %d, %d) の最大値: %d\n",   
 num1, num2, third\_value, max\_of\_three);  
 printf("3つの値 (%d, %d, %d) の最小値: %d\n",   
 num1, num2, third\_value, min\_of\_three);  
 }  
   
 /\* 値の範囲判定（条件演算子のみ使用） \*/  
 printf("\n=== 範囲判定 ===\n");  
 {  
 const char \*range1 = (num1 >= -10 && num1 <= 10) ?   
 "範囲内（-10〜10）" : "範囲外";  
 const char \*range2 = (num2 >= -10 && num2 <= 10) ?   
 "範囲内（-10〜10）" : "範囲外";  
   
 printf("第1の値（%d）: %s\n", num1, range1);  
 printf("第2の値（%d）: %s\n", num2, range2);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_2\_comparison  
===== 比較と論理演算 =====  
2つの整数を入力してください: 7 12  
  
=== 結果 ===  
入力値: 7, 12  
最大値: 12  
最小値: 7  
  
=== 関係演算子による比較 ===  
7 == 12 : 0  
7 != 12 : 1  
7 < 12 : 1  
7 <= 12 : 1  
7 > 12 : 0  
7 >= 12 : 0  
  
=== 論理演算子の応用 ===  
両方とも正の数: はい  
少なくとも一方が正の数: はい  
両方とも偶数: いいえ  
  
=== 条件演算子の応用 ===  
3つ目の値として0を使用します  
3つの値 (7, 12, 0) の最大値: 12  
3つの値 (7, 12, 0) の最小値: 0  
  
=== 範囲判定 ===  
第1の値（7）: 範囲内（-10〜10）  
第2の値（12）: 範囲外  
  
学習ポイント:  
1. 条件演算子（三項演算子）による最大値・最小値の判定  
2. 関係演算子（==, !=, <, <=, >, >=）の使用  
3. 論理演算子（&&, ||, !）の使用  
4. 条件演算子のネストによる複雑な条件判定  
5. if文を使わない条件分岐の実装  
\*/```  
  
## ex4\_3\_increment.c  
  
```c  
/\*  
 \* ファイル名: ex4\_3\_increment.c  
 \* 演習4-3: インクリメントとデクリメント演算  
 \* 説明: 前置と後置のインクリメント・デクリメント演算子の違いを確認  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
int main(void)  
{  
 int a, b, result;  
 int array[5] = {10, 20, 30, 40, 50};  
 int i;  
   
 printf("===== インクリメント・デクリメント演算子 =====\n");  
   
 /\* 基本的な動作確認 \*/  
 printf("\n=== 基本的な動作 ===\n");  
   
 a = 5;  
 printf("初期値: a = %d\n", a);  
   
 result = a++; /\* 後置インクリメント \*/  
 printf("result = a++ → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = ++a; /\* 前置インクリメント \*/  
 printf("result = ++a → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = a--; /\* 後置デクリメント \*/  
 printf("result = a-- → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 result = --a; /\* 前置デクリメント \*/  
 printf("result = --a → result = %d, a = %d\n", result, a);  
   
 /\* 式の中での動作確認 \*/  
 printf("\n=== 式の中での動作 ===\n");  
   
 a = 5;  
 b = 3;  
 result = a++ \* b;  
 printf("a = 5, b = 3\n");  
 printf("result = a++ \* b → result = %d, a = %d\n", result, a);  
   
 a = 5;  
 b = 3;  
 result = ++a \* b;  
 printf("a = 5, b = 3\n");  
 printf("result = ++a \* b → result = %d, a = %d\n", result, a);  
   
 /\* 複雑な式での使用 \*/  
 printf("\n=== 複雑な式での使用 ===\n");  
   
 a = 10;  
 b = 5;  
 result = a++ + ++b;  
 printf("a = 10, b = 5\n");  
 printf("result = a++ + ++b → result = %d, a = %d, b = %d\n",   
 result, a, b);  
   
 a = 10;  
 b = 5;  
 result = --a - b--;  
 printf("a = 10, b = 5\n");  
 printf("result = --a - b-- → result = %d, a = %d, b = %d\n",   
 result, a, b);  
   
 /\* 配列とポインタでの使用 \*/  
 printf("\n=== 配列での使用 ===\n");  
 printf("配列: ");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n");  
   
 i = 0;  
 int temp\_val = array[i++];  
 printf("array[i++] = %d, i = %d\n", temp\_val, i);  
   
 i = 0;  
 temp\_val = array[++i];  
 printf("array[++i] = %d, i = %d\n", temp\_val, i);  
   
 /\* 実用的な例：ループでの使用 \*/  
 printf("\n=== ループでの実用例 ===\n");  
   
 printf("後置インクリメント（i++）:\n");  
 for (i = 0; i < 5; i++) {  
 printf("i = %d, array[%d] = %d\n", i, i, array[i]);  
 }  
   
 printf("\n前置インクリメント（++i）:\n");  
 for (i = 0; i < 5; ++i) {  
 printf("i = %d, array[%d] = %d\n", i, i, array[i]);  
 }  
   
 /\* 注意事項：未定義動作の例（コメントアウト） \*/  
 printf("\n=== 注意事項 ===\n");  
 printf("以下のような式は未定義動作となるので避けるべきです：\n");  
 printf(" i = i++; （同じ変数への複数の副作用）\n");  
 printf(" a[i] = i++; （評価順序が不定）\n");  
 printf(" func(i++, i++); （引数の評価順序が不定）\n");  
   
 /\* ポインタでの使用例 \*/  
 printf("\n=== ポインタでの使用例 ===\n");  
 {  
 int \*ptr = array;  
   
 printf("\*ptr++ = %d （値を取得してからポインタを進める）\n", \*ptr++);  
 printf("現在のptr位置の値: %d\n", \*ptr);  
   
 ptr = array; /\* リセット \*/  
 printf("\*++ptr = %d （ポインタを進めてから値を取得）\n", \*++ptr);  
 printf("現在のptr位置の値: %d\n", \*ptr);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_3\_increment  
===== インクリメント・デクリメント演算子 =====  
  
=== 基本的な動作 ===  
初期値: a = 5  
result = a++ → result = 5, a = 6  
result = ++a → result = 6, a = 6  
result = a-- → result = 5, a = 4  
result = --a → result = 4, a = 4  
  
=== 式の中での動作 ===  
a = 5, b = 3  
result = a++ \* b → result = 15, a = 6  
a = 5, b = 3  
result = ++a \* b → result = 18, a = 6  
  
=== 複雑な式での使用 ===  
a = 10, b = 5  
result = a++ + ++b → result = 16, a = 11, b = 6  
a = 10, b = 5  
result = --a - b-- → result = 4, a = 9, b = 4  
  
=== 配列での使用 ===  
配列: 10 20 30 40 50  
array[i++] = 10, i = 1  
array[++i] = 20, i = 1  
  
=== ループでの実用例 ===  
後置インクリメント（i++）:  
i = 0, array[0] = 10  
i = 1, array[1] = 20  
i = 2, array[2] = 30  
i = 3, array[3] = 40  
i = 4, array[4] = 50  
  
前置インクリメント（++i）:  
i = 0, array[0] = 10  
i = 1, array[1] = 20  
i = 2, array[2] = 30  
i = 3, array[3] = 40  
i = 4, array[4] = 50  
  
=== 注意事項 ===  
以下のような式は未定義動作となるので避けるべきです：  
 i = i++; （同じ変数への複数の副作用）  
 a[i] = i++; （評価順序が不定）  
 func(i++, i++); （引数の評価順序が不定）  
  
=== ポインタでの使用例 ===  
\*ptr++ = 10 （値を取得してからポインタを進める）  
現在のptr位置の値: 20  
\*++ptr = 20 （ポインタを進めてから値を取得）  
現在のptr位置の値: 20  
  
学習ポイント:  
1. 前置と後置の違い（値の返却タイミング）  
2. 式の中での評価順序  
3. 配列やポインタでの実用的な使い方  
4. 未定義動作を避けるための注意点  
5. ループでの効率的な使用方法  
\*/```  
  
## ex4\_4\_bitwise.c  
  
```c  
/\*  
 \* ファイル名: ex4\_4\_bitwise.c  
 \* 演習4-4: ビット演算  
 \* 説明: ビット演算子を使って数値の特定ビットを操作  
 \* 規格: C90準拠  
 \*/  
  
#include <stdio.h>  
  
/\* ビットを表示する関数 \*/  
void print\_bits(unsigned int value, int bits)  
{  
 int i;  
 printf("0b");  
 for (i = bits - 1; i >= 0; i--) {  
 printf("%d", (value >> i) & 1);  
 if (i % 4 == 0 && i > 0) {  
 printf("\_"); /\* 4ビットごとに区切り \*/  
 }  
 }  
}  
  
int main(void)  
{  
 unsigned int num;  
 int bit\_position;  
 unsigned int result;  
 unsigned int mask;  
   
 printf("===== ビット操作 =====\n");  
 printf("整数を入力してください (0-255): ");  
   
 if (scanf("%u", &num) != 1 || num > 255) {  
 printf("エラー: 0から255の範囲で入力してください\n");  
 return 1;  
 }  
   
 printf("\n元の値: %u = ", num);  
 print\_bits(num, 8);  
 printf("\n\n");  
   
 /\* ビット単位の基本演算 \*/  
 printf("=== ビット単位の基本演算 ===\n");  
   
 /\* AND演算 \*/  
 result = num & 0xF0; /\* 上位4ビットを取得 \*/  
 printf("AND演算 (上位4ビット取得):\n");  
 printf("%u & 0xF0 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* OR演算 \*/  
 result = num | 0x0F; /\* 下位4ビットを1にセット \*/  
 printf("OR演算 (下位4ビットセット):\n");  
 printf("%u | 0x0F = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* XOR演算 \*/  
 result = num ^ 0xFF; /\* 全ビット反転 \*/  
 printf("XOR演算 (全ビット反転):\n");  
 printf("%u ^ 0xFF = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* NOT演算 \*/  
 result = ~num & 0xFF; /\* 8ビットのみ表示 \*/  
 printf("NOT演算 (ビット反転):\n");  
 printf("~%u = %u = ", num, result);  
 print\_bits(result, 8);  
 printf("\n\n");  
   
 /\* ビットシフト演算 \*/  
 printf("=== ビットシフト演算 ===\n");  
   
 /\* 左シフト \*/  
 result = num << 1;  
 printf("左シフト (1ビット):\n");  
 printf("%u << 1 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf(" (2倍)\n\n");  
   
 /\* 右シフト \*/  
 result = num >> 1;  
 printf("右シフト (1ビット):\n");  
 printf("%u >> 1 = %u = ", num, result);  
 print\_bits(result, 8);  
 printf(" (1/2)\n\n");  
   
 /\* 特定ビットの操作 \*/  
 printf("=== 特定ビットの操作 ===\n");  
 printf("操作するビット位置を入力 (0-7): ");  
   
 if (scanf("%d", &bit\_position) != 1 || bit\_position < 0 || bit\_position > 7) {  
 printf("エラー: 0から7の範囲で入力してください\n");  
 return 1;  
 }  
   
 /\* ビットマスクの作成 \*/  
 mask = 1u << bit\_position;  
   
 /\* ビットのテスト \*/  
 printf("\nビット%dのテスト:\n", bit\_position);  
 if (num & mask) {  
 printf("ビット%dは1です\n", bit\_position);  
 } else {  
 printf("ビット%dは0です\n", bit\_position);  
 }  
   
 /\* ビットのセット \*/  
 result = num | mask;  
 printf("\nビット%dをセット:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* ビットのクリア \*/  
 result = num & ~mask;  
 printf("\nビット%dをクリア:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* ビットの反転 \*/  
 result = num ^ mask;  
 printf("\nビット%dを反転:\n", bit\_position);  
 printf("結果: %u = ", result);  
 print\_bits(result, 8);  
 printf("\n");  
   
 /\* 実用的な例 \*/  
 printf("\n=== 実用的なビット操作例 ===\n");  
   
 /\* 偶数・奇数の判定 \*/  
 printf("偶数・奇数判定: ");  
 if (num & 1) {  
 printf("%uは奇数です (最下位ビットが1)\n", num);  
 } else {  
 printf("%uは偶数です (最下位ビットが0)\n", num);  
 }  
   
 /\* 2のべき乗の判定 \*/  
 if (num > 0 && (num & (num - 1)) == 0) {  
 printf("%uは2のべき乗です\n", num);  
 } else {  
 printf("%uは2のべき乗ではありません\n", num);  
 }  
   
 /\* ビットカウント（立っているビットの数） \*/  
 {  
 unsigned int temp = num;  
 int count = 0;  
 while (temp) {  
 count += temp & 1;  
 temp >>= 1;  
 }  
 printf("立っているビットの数: %d個\n", count);  
 }  
   
 /\* 上位ビットと下位ビットの交換 \*/  
 {  
 unsigned int upper = (num & 0xF0) >> 4;  
 unsigned int lower = (num & 0x0F) << 4;  
 result = upper | lower;  
 printf("\n上位4ビットと下位4ビットの交換:\n");  
 printf("元: ");  
 print\_bits(num, 8);  
 printf("\n結果: ");  
 print\_bits(result, 8);  
 printf(" (%u)\n", result);  
 }  
   
 return 0;  
}  
  
/\*  
実行例:  
$ ./ex3\_4\_bitwise  
===== ビット操作 =====  
整数を入力してください (0-255): 170  
  
元の値: 170 = 0b1010\_1010  
  
=== ビット単位の基本演算 ===  
AND演算 (上位4ビット取得):  
170 & 0xF0 = 160 = 0b1010\_0000  
  
OR演算 (下位4ビットセット):  
170 | 0x0F = 175 = 0b1010\_1111  
  
XOR演算 (全ビット反転):  
170 ^ 0xFF = 85 = 0b0101\_0101  
  
NOT演算 (ビット反転):  
~170 = 85 = 0b0101\_0101  
  
=== ビットシフト演算 ===  
左シフト (1ビット):  
170 << 1 = 340 = 0b0101\_0100 (2倍)  
  
右シフト (1ビット):  
170 >> 1 = 85 = 0b0101\_0101 (1/2)  
  
=== 特定ビットの操作 ===  
操作するビット位置を入力 (0-7): 3  
  
ビット3のテスト:  
ビット3は1です  
  
ビット3をセット:  
結果: 170 = 0b1010\_1010  
  
ビット3をクリア:  
結果: 162 = 0b1010\_0010  
  
ビット3を反転:  
結果: 162 = 0b1010\_0010  
  
=== 実用的なビット操作例 ===  
偶数・奇数判定: 170は偶数です (最下位ビットが0)  
170は2のべき乗ではありません  
立っているビットの数: 4個  
  
上位4ビットと下位4ビットの交換:  
元: 0b1010\_1010  
結果: 0b1010\_1010 (170)  
  
学習ポイント:  
1. ビット演算子（&, |, ^, ~, <<, >>）の使い方  
2. ビットマスクによる特定ビットの操作  
3. ビット演算を使った効率的な処理  
4. 2進数表示によるビット操作の可視化  
5. 実用的なビット操作テクニック  
\*/```  
  
## ex4\_5\_conditional.c  
  
```c  
/\*  
 \* ファイル名: ex4\_5\_conditional.c  
 \* 演習4-5: 条件演算子  
 \* 説明: 3つの数値を条件演算子のみで昇順に並び替える  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
  
/\* 時間表示用の構造体 \*/  
typedef struct  
{  
 int hour;  
 int minute;  
} Time;  
  
/\* ユーティリティ関数 \*/  
int max(int a, int b);  
int min(int a, int b);  
int sign(int x);  
double absolute(double x);  
char get\_grade(int score);  
const char \*get\_season(int month);  
  
int main(void)  
{  
 /\* C90準拠のために全ての変数を最初に宣言 \*/  
 int number;  
 int score;  
 char grade;  
 int values[] = {23, 89, 45, 12, 67, 91, 34};  
 int size;  
 int array\_max;  
 int array\_min;  
 int range\_min, range\_max;  
 int in\_range\_count;  
 Time current\_time;  
 int hour\_12;  
 const char \*ampm;  
 int age;  
 char gender;  
 int is\_student;  
 int discount\_rate;  
 const char \*discount\_reason;  
 int x, y;  
 char text[] = "Hello, World!";  
 int months[] = {3, 6, 9, 12};  
 int indices[] = {-1, 0, 3, 5, 7, 10};  
 int array\_size;  
 int safe\_array[5] = {10, 20, 30, 40, 50};  
 int i;  
 char ch;  
 const char \*type;  
 int idx;  
 int safe\_idx;  
 int safe\_value;  
  
 printf("=== 条件演算子の応用 ===\n");  
  
 /\* 基本的な使用 \*/  
 printf("基本的な使用:\n");  
 number = 15;  
 printf(" 数値: %d\n", number);  
 printf(" 絶対値: %d (%s)\n",  
 (number >= 0) ? number : -number,  
 (number >= 0) ? "正数" : "負数");  
 printf(" 符号: %c\n", (number > 0) ? '+' : (number < 0) ? '-'  
 : '0');  
 printf(" 偶数/奇数: %s\n", (number % 2 == 0) ? "偶数" : "奇数");  
 printf("\n");  
  
 /\* ネストした条件演算子 \*/  
 printf("ネストした条件演算子:\n");  
 score = 85;  
 grade = (score >= 90) ? 'A' : (score >= 80) ? 'B'  
 : (score >= 70) ? 'C'  
 : (score >= 60) ? 'D'  
 : 'F';  
 printf(" 点数: %d\n", score);  
 printf(" 評価: %c (90以上:A, 80以上:B, 70以上:C, 60以上:D, その他:F)\n", grade);  
 printf("\n");  
  
 /\* 配列操作での使用 \*/  
 printf("配列操作での使用:\n");  
 size = sizeof(values) / sizeof(values[0]);  
  
 /\* 最大値・最小値の検索 \*/  
 array\_max = values[0];  
 array\_min = values[0];  
 for (i = 1; i < size; i++)  
 {  
 array\_max = (values[i] > array\_max) ? values[i] : array\_max;  
 array\_min = (values[i] < array\_min) ? values[i] : array\_min;  
 }  
 printf(" 配列の最大値: %d\n", array\_max);  
 printf(" 配列の最小値: %d\n", array\_min);  
  
 /\* 範囲内の要素数 \*/  
 range\_min = 30;  
 range\_max = 80;  
 in\_range\_count = 0;  
 for (i = 0; i < size; i++)  
 {  
 in\_range\_count += (values[i] >= range\_min && values[i] <= range\_max) ? 1 : 0;  
 }  
 printf(" 範囲[%d-%d]内の要素数: %d個\n", range\_min, range\_max, in\_range\_count);  
 printf("\n");  
  
 /\* 時間表示での使用 \*/  
 printf("時間表示での使用:\n");  
 current\_time.hour = 14;  
 current\_time.minute = 30;  
 printf(" 現在時刻: %d時%d分\n", current\_time.hour, current\_time.minute);  
  
 /\* 12時間表記への変換 \*/  
 hour\_12 = (current\_time.hour == 0) ? 12 : (current\_time.hour > 12) ? current\_time.hour - 12  
 : current\_time.hour;  
 ampm = (current\_time.hour < 12) ? "午前" : "午後";  
 printf(" 12時間表記: %s%d時%d分\n", ampm, hour\_12, current\_time.minute);  
 printf(" AM/PM: %s %d:%02d\n",  
 (current\_time.hour < 12) ? "AM" : "PM",  
 hour\_12, current\_time.minute);  
 printf("\n");  
  
 /\* 複雑な条件 \*/  
 printf("複雑な条件:\n");  
 age = 20;  
 gender = 'M';  
 is\_student = 1;  
  
 printf(" 年齢: %d歳, 性別: %c, 学生: %s\n",  
 age, gender, is\_student ? "はい" : "いいえ");  
  
 /\* 割引率の計算 \*/  
 discount\_rate = (is\_student && age < 25) ? 20 : (age >= 65) ? 15  
 : (age < 18) ? 10  
 : 0;  
 discount\_reason = (is\_student && age < 25) ? "学生割引" : (age >= 65) ? "シニア割引"  
 : (age < 18) ? "子供割引"  
 : "割引なし";  
 printf(" 割引率: %d%% (%s)\n", discount\_rate, discount\_reason);  
 printf("\n");  
  
 /\* 計算での使用 \*/  
 printf("計算での使用:\n");  
 x = 5;  
 y = 3;  
 printf(" x=%d, y=%d\n", x, y);  
 printf(" max(x,y) = %d\n", max(x, y));  
 printf(" min(x,y) = %d\n", min(x, y));  
 printf(" sign(x-y) = %+d\n", sign(x - y));  
 printf("\n");  
  
 /\* 文字列処理での使用 \*/  
 printf("文字列処理での使用:\n");  
 printf(" 文字列: \"%s\"\n", text);  
 printf(" 文字数: %d\n", (int)strlen(text));  
  
 /\* 文字の分類 \*/  
 printf(" 文字分類:\n");  
 for (i = 0; text[i] != '\0'; i++)  
 {  
 ch = text[i];  
 type = (ch >= 'A' && ch <= 'Z') ? "大文字" : (ch >= 'a' && ch <= 'z') ? "小文字"  
 : (ch >= '0' && ch <= '9') ? "数字"  
 : (ch == ' ') ? "空白"  
 : "記号";  
 printf(" '%c': %s\n", ch, type);  
 }  
 printf("\n");  
  
 /\* 季節判定 \*/  
 printf("季節判定:\n");  
 for (i = 0; i < 4; i++)  
 {  
 printf(" %d月: %s\n", months[i], get\_season(months[i]));  
 }  
 printf("\n");  
  
 /\* 配列の範囲制限 \*/  
 printf("配列の範囲制限:\n");  
 array\_size = 5;  
  
 for (i = 0; i < 6; i++)  
 {  
 idx = indices[i];  
 safe\_idx = (idx < 0) ? 0 : (idx >= array\_size) ? array\_size - 1  
 : idx;  
 safe\_value = safe\_array[safe\_idx];  
 printf(" インデックス%d: %s (値:%d)\n",  
 idx,  
 (idx >= 0 && idx < array\_size) ? "範囲内" : "範囲外→制限",  
 safe\_value);  
 }  
  
 printf("==================\n");  
  
 return 0;  
}  
  
/\* ユーティリティ関数の実装 \*/  
int max(int a, int b)  
{  
 return (a > b) ? a : b;  
}  
  
int min(int a, int b)  
{  
 return (a < b) ? a : b;  
}  
  
int sign(int x)  
{  
 return (x > 0) ? 1 : (x < 0) ? -1  
 : 0;  
}  
  
double absolute(double x)  
{  
 return (x >= 0.0) ? x : -x;  
}  
  
char get\_grade(int score)  
{  
 return (score >= 90) ? 'A' : (score >= 80) ? 'B'  
 : (score >= 70) ? 'C'  
 : (score >= 60) ? 'D'  
 : 'F';  
}  
  
const char \*get\_season(int month)  
{  
 return (month >= 3 && month <= 5) ? "春" : (month >= 6 && month <= 8) ? "夏"  
 : (month >= 9 && month <= 11) ? "秋"  
 : "冬";  
}  
  
/\*  
学習ポイント:  
1. 条件演算子の基本構文  
 - condition ? expression1 : expression2  
 - 三項演算子とも呼ばれる  
  
2. ネストした条件演算子  
 - 複数の条件を組み合わせ可能  
 - 可読性のため適度に使用  
 - if-else文との使い分け  
  
3. 実用的な応用  
 - 最大値・最小値の選択  
 - 範囲制限（クランプ）  
 - デフォルト値の設定  
 - 文字列の選択  
  
4. パフォーマンス考慮  
 - 式として評価されるため効率的  
 - 関数呼び出しより高速  
 - コンパイラ最適化が効きやすい  
  
5. 使用上の注意点  
 - 複雑な条件は分割する  
 - 副作用のある式は避ける  
 - 型の一致に注意  
 - 可読性を優先  
  
6. 適用場面  
 - 簡単な値の選択  
 - 初期化での条件分岐  
 - マクロ内での条件処理  
 - 関数の引数での条件  
\*/

## ex4\_6\_precedence.c

/\*  
 \* ファイル名: ex4\_6\_precedence.c  
 \* 演習4-6: 演算子の優先順位  
 \* 説明: 複雑な式を含むプログラムで演算子の優先順位を確認  
 \* 規格: C90準拠  
 \*/  
#include <stdio.h>  
  
/\* 複雑な式の評価を段階的に表示する関数 \*/  
void demonstrate\_evaluation\_order(void);  
void demonstrate\_associativity(void);  
void demonstrate\_common\_mistakes(void);  
  
int main(void)  
{  
 printf("=== 演算子の優先順位 ===\n");  
  
 /\* 基本的な優先順位 \*/  
 printf("基本的な優先順位:\n");  
 int result1 = 2 + 3 \* 4;  
 int result2 = (2 + 3) \* 4;  
 printf(" 2 + 3 \* 4 = %d (3\*4が先に計算される)\n", result1);  
 printf(" (2 + 3) \* 4 = %d (括弧で優先順位変更)\n", result2);  
 printf("\n");  
  
 /\* 除算の優先順位 \*/  
 printf("除算の優先順位:\n");  
 int result3 = 10 / 2 \* 3;  
 int result4 = 10 / (2 \* 3);  
 printf(" 10 / 2 \* 3 = %d (左から右へ: (10/2)\*3)\n", result3);  
 printf(" 10 / (2 \* 3) = %d (括弧で変更)\n", result4);  
 printf("\n");  
  
 /\* 比較と論理 \*/  
 printf("比較と論理:\n");  
 int a = 5, b = 3, c = 4;  
 int result5 = a > b && b < c;  
 int result6 = a > b + 1;  
 printf(" %d > %d && %d < %d = %d (比較が先、論理が後)\n", a, b, b, c, result5);  
 printf(" %d > %d + 1 = %d (加算が先: %d > %d)\n", a, b, result6, a, b + 1);  
 printf("\n");  
  
 /\* 代入の右結合 \*/  
 printf("代入の右結合:\n");  
 int x, y, z;  
 x = y = z = 5;  
 printf(" x = y = z = 5\n");  
 printf(" 結果: x=%d, y=%d, z=%d\n", x, y, z);  
 printf("\n");  
  
 /\* ビット演算の優先順位 \*/  
 printf("ビット演算の優先順位:\n");  
 int result7 = 6 & 3 | 4;  
 int result8 = 6 & (3 | 4);  
 printf(" 6 & 3 | 4 = %d (&が先: (6&3)|4 = %d|4)\n", result7, 6 & 3);  
 printf(" 6 & (3 | 4) = %d (括弧で変更: 6&%d)\n", result8, 3 | 4);  
 printf("\n");  
  
 /\* より詳細なデモ \*/  
 demonstrate\_evaluation\_order();  
 demonstrate\_associativity();  
 demonstrate\_common\_mistakes();  
  
 /\* 実用的な例 \*/  
 printf("実用的な優先順位の例:\n");  
  
 /\* 条件式での優先順位 \*/  
 int score = 85;  
 int is\_passing = score >= 60 && score <= 100;  
 printf(" 点数判定: score=%d\n", score);  
 printf(" score >= 60 && score <= 100 = %d\n", is\_passing);  
  
 /\* 計算式での優先順位 \*/  
 double radius = 5.0;  
 double area = 3.14 \* radius \* radius; /\* 左から右へ評価 \*/  
 double circumference = 2 \* 3.14 \* radius;  
 printf(" 円の面積 (r=%.1f): %.2f\n", radius, area);  
 printf(" 円の周長: %.2f\n", circumference);  
  
 /\* 配列インデックスでの優先順位 \*/  
 int array[] = {10, 20, 30, 40, 50};  
 int index = 2;  
 int value = array[index++]; /\* []が先、then ++ \*/  
 printf(" 配列アクセス: array[%d] = %d, その後index = %d\n", index - 1, value, index);  
  
 printf("=======================\n");  
  
 return 0;  
}  
  
/\* 複雑な式の評価順序デモ \*/  
void demonstrate\_evaluation\_order(void)  
{  
 printf("複雑な式の評価順序:\n");  
  
 /\* 段階的に複雑にしていく \*/  
 int p = 2, q = 3, r = 4, s = 5;  
 printf(" 初期値: p=%d, q=%d, r=%d, s=%d\n", p, q, r, s);  
  
 /\* 式1: 算術演算の組み合わせ \*/  
 int expr1 = p + q \* r - s;  
 printf(" p + q \* r - s:\n");  
 printf(" = %d + %d \* %d - %d\n", p, q, r, s);  
 printf(" = %d + %d - %d (乗算先)\n", p, q \* r, s);  
 printf(" = %d - %d (加算次)\n", p + q \* r, s);  
 printf(" = %d (減算最後)\n", expr1);  
 printf("\n");  
  
 /\* 式2: 比較と論理の組み合わせ \*/  
 int expr2 = p < q && r > s || p == q;  
 printf(" p < q && r > s || p == q:\n");  
 printf(" = %d < %d && %d > %d || %d == %d\n", p, q, r, s, p, q);  
 printf(" = %d && %d || %d (比較先)\n", p<q, r> s, p == q);  
 printf(" = %d || %d (&&次)\n", (p < q) && (r > s), p == q);  
 printf(" = %d (||最後)\n", expr2);  
 printf("\n");  
  
 /\* 式3: 混合演算 \*/  
 int expr3 = p++ + ++q \* r-- > s && q != r;  
 printf(" 複雑な混合演算の結果: %d\n", expr3);  
 printf(" 演算後の値: p=%d, q=%d, r=%d, s=%d\n", p, q, r, s);  
 printf("\n");  
}  
  
/\* 結合性のデモ \*/  
void demonstrate\_associativity(void)  
{  
 printf("結合性のデモ:\n");  
  
 /\* 左結合の例 \*/  
 printf("左結合の例:\n");  
 int result1 = 100 / 5 / 2; /\* (100 / 5) / 2 \*/  
 int result2 = 100 / (5 / 2); /\* 100 / (5 / 2) \*/  
 printf(" 100 / 5 / 2 = %d (左結合: (100/5)/2)\n", result1);  
 printf(" 100 / (5 / 2) = %d (括弧で右結合)\n", result2);  
  
 int result3 = 20 - 10 - 5; /\* (20 - 10) - 5 \*/  
 int result4 = 20 - (10 - 5); /\* 20 - (10 - 5) \*/  
 printf(" 20 - 10 - 5 = %d (左結合: (20-10)-5)\n", result3);  
 printf(" 20 - (10 - 5) = %d (括弧で右結合)\n", result4);  
 printf("\n");  
  
 /\* 右結合の例 \*/  
 printf("右結合の例:\n");  
 int a, b, c;  
 a = b = c = 10; /\* a = (b = (c = 10)) \*/  
 printf(" a = b = c = 10 → a=%d, b=%d, c=%d (右結合)\n", a, b, c);  
  
 /\* 条件演算子の右結合 \*/  
 int x = 5;  
 int result5 = x > 0 ? x > 10 ? 10 : x : 0;  
 printf(" 条件演算子: x > 0 ? x > 10 ? 10 : x : 0\n");  
 printf(" x=%d の場合: %d (右結合で評価)\n", x, result5);  
 printf("\n");  
}  
  
/\* よくある間違いの例 \*/  
void demonstrate\_common\_mistakes(void)  
{  
 printf("よくある間違いの例:\n");  
  
 /\* 間違い1: ビット演算と比較の優先順位 \*/  
 printf("間違い1: ビット演算の優先順位\n");  
 int flags = 5; /\* 0101 \*/  
 int mask = 1; /\* 0001 \*/  
  
 /\* 間違った書き方 \*/  
 int wrong = flags & mask == 1; /\* flags & (mask == 1) \*/  
 int correct = (flags & mask) == 1; /\* 正しい \*/  
  
 printf(" flags=%d, mask=%d\n", flags, mask);  
 printf(" flags & mask == 1 = %d (間違い: flags & (mask == 1))\n", wrong);  
 printf(" (flags & mask) == 1 = %d (正しい)\n", correct);  
 printf("\n");  
  
 /\* 間違い2: シフトと算術演算 \*/  
 printf("間違い2: シフトと算術演算\n");  
 int val = 4;  
 int wrong2 = val << 2 + 1; /\* val << (2 + 1) \*/  
 int correct2 = (val << 2) + 1; /\* 正しい \*/  
  
 printf(" val=%d\n", val);  
 printf(" val << 2 + 1 = %d (間違い: val << (2 + 1))\n", wrong2);  
 printf(" (val << 2) + 1 = %d (正しい)\n", correct2);  
 printf("\n");  
  
 /\* 間違い3: ポインタと配列の優先順位 \*/  
 printf("間違い3: ポインタ演算\n");  
 int array[] = {10, 20, 30, 40};  
 int \*ptr = array;  
  
 printf(" \*ptr++ = %d (ポインタを進めてから前の値)\n", \*ptr++);  
 printf(" 現在のptr: %d\n", \*ptr);  
  
 ptr = array; /\* リセット \*/  
 printf(" (\*ptr)++ = %d (値をインクリメント)\n", (\*ptr)++);  
 printf(" array[0] = %d (配列の値が変更された)\n", array[0]);  
  
 array[0] = 10; /\* 復元 \*/  
 printf("\n");  
  
 /\* 間違い4: 条件演算子の優先順位 \*/  
 printf("間違い4: 条件演算子\n");  
 int score = 85;  
 char grade1 = score >= 90 ? 'A' : score >= 80 ? 'B'  
 : 'C';  
 char grade2 = (score >= 90) ? 'A' : ((score >= 80) ? 'B' : 'C');  
  
 printf(" score=%d\n", score);  
 printf(" 評価1: %c (右結合で正しく動作)\n", grade1);  
 printf(" 評価2: %c (括弧で明確化)\n", grade2);  
 printf("\n");  
}  
  
/\*  
学習ポイント:  
1. 演算子の優先順位  
 - 算術 > 関係 > 論理 > 代入  
 - 乗除 > 加減  
 - ビット演算は関係演算より低い  
  
2. 結合性  
 - 左結合: 算術、関係、論理演算  
 - 右結合: 代入、条件演算子  
  
3. よくある間違い  
 - ビット演算と比較の混同  
 - シフトと算術演算の優先順位  
 - ポインタ演算の理解不足  
  
4. 対策  
 - 複雑な式では括弧を使う  
 - 一行に複数の演算子を書かない  
 - 可読性を優先する  
  
5. 実用的なガイドライン  
 - 疑問があれば括弧で明確にする  
 - コンパイラ警告を確認する  
 - 式を分割して可読性を向上させる  
  
重要な優先順位:  
1. 後置演算子（++, --）  
2. 単項演算子（前置++, --, !, ~, \*, &）  
3. 乗除（\*, /, %）  
4. 加減（+, -）  
5. シフト（<<, >>）  
6. 関係（<, <=, >, >=）  
7. 等価（==, !=）  
8. ビット演算（&, ^, |）  
9. 論理演算（&&, ||）  
10. 条件演算子（?:）  
11. 代入（=, +=, -=, ...）  
\*/