

猫でもわかるC++プログラミング 第2版 練習問題解答

第1章「C++ をはじめる前に」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。
(1) C++ とC 言語はまったく別な言語である。
(2) オブジェクト指向プログラミングの3 本柱は、データの公開、継承、多態性である。
(3) コンピュータの5 大機能とは、制御・演算・記憶・入力・出力機能のことである。
(4) ほとんどすべてのコンピュータでは、プログラムはハードディスクから直接読み取られて実行される。
(5) C++ のソースファイルはバイナリファイルである。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ×

■ 解説

- (1) C++ はC 言語を機能拡張した言語です (p.2)。
- (2) オブジェクト指向プログラミングの3 本柱は「カプセル化」、「継承」、「多態性」です (p.3)。
- (3) コンピュータの5 大機能とは、制御・演算・記憶・入力・出力機能です (p.4)。
- (4) プログラムはメモリにコピーされてから実行されます (p.5)。
- (5) C++ のソースファイルはテキストファイルです (p.6)。

第2章「C++ の基本」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。
(1) C++ では、データの出力には出力演算子 (output operator) 「>>」を使う。
(2) 出力演算子は文字列の出力専用である。
(3) スペースを出力するにはendl マニピュレータを使う。
(4) C++ のプログラムはmain 関数から始まる。
(5) using namespace ディレクティブを使うと名前空間の指定を省略できる。

■ 解答

(1) × (2) × (3) × (4) ○ (5) ○

■ 解説

- (1) 出力演算子は「<<」です (p.13)。
- (2) 出力演算子は、文字列だけでなく数値などあらゆるデータを出力できます (2.3節、3.4 節など)。
- (3) endl マニピュレータは改行を出力します (p.14)。
- (4) プログラムはmain 関数から始まります (p.12)。
- (5) 名前空間の指定を省略するには、using namespace ディレクティブを使います (p.14)。

第3 章 「変数とデータ型」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。
 - (1) 変数を使うには、使う前に宣言をしなくてはならない。
 - (2) 変数は宣言と同時に初期化することはできない。
 - (3) hex マニピュレータを利用すると整数値を16 進形式で表示できる。
 - (4) キーボードからの入力を変数に代入するには入力演算子を利用することができます。
 - (5) C++ では printf 関数は利用できない。

■ 解答

- (1)○ (2)× (3)○ (4)○ (5)×

■ 解説

- (1) 変数は使う前に宣言しなくてはなりません (p.26)。
- (2) 変数は、宣言と同時に値を代入（初期化）できます (p.28)。
- (3) hex マニピュレータを使うと16 進形式で表示できます (p.34)。
- (4) 入力演算子「>>」とstd::cin を使うとキーボード入力を取得できます (p.43)。
- (5) C++ でもprintf 関数を利用可能です (p.19 のコラム)。

第4 章 「式と演算子」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。
 - (1) 演算子が複数存在する式では左から右に順番に演算が行われる。
 - (2) 優先順位の同じ演算子は右から左に演算が行われる。
 - (3) インクリメント演算子はオペランドの後に付く場合と前に付く場合があり、それぞれ意味が異なる。

(4) sizeof 演算子の返す型はint 型である。

(5) キャスト演算子を使うと一時的に変数を他の型として使うことができる。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ○

■ 解説

(1) 演算子は優先順位の高いものから評価されます (p.50)。

(2) 優先順位が同じ場合は結合規則に従います (p.55)。

(3) 後置か前置かによって評価の順番が違います (p.57)。

(4) size_t 型を返します (p.61)。

(5) キャスト演算子は、変数を指定した型に変換します (p.68)。

第5 章 「制御文」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。

(1) if 文では、制御式が真のときに実行される文を真文という。

(2) switch 文の事例式にはdouble 型の数値が利用できる。

(3) break 文は、switch 文を終了させることはできない。

(4) continue 文は、ブロックの先頭に戻って処理を続けるときに使う。

(5) for 文では継続条件式を省略することはできない。

■ 解答

(1) ○ (2) × (3) × (4) ○ (5) ×

■ 解説

(1) 制御式が真のときに実行される文は真文です (p.82)。

(2) switch 文の事例式は整数もしくは文字の定数でなければなりません (p.92)。

(3) switch 文は、break 文で終了します (p.92)。

(4) continue 文を使うと、ブロック先頭に戻ります (p.113)。

(5) for 文では継続条件式を省略することができます (無限ループとなります) (p.110)。

第6 章 「関数」

■ 問題

1. 関数の説明について、正しいものには○、正しくないものには×を付けてください。

(1) C++ では、関数はデフォルト引数を持つことができる。

(2) C++ では、戻り値の型が異なれば同名の関数が複数存在してもよい。

(3) C++ では、変数の宣言は、変数を使用する前であれば任意の場所でできる。

(4) 自作関数を使用するときは必ずプロトタイプ宣言が必要である。

(5) 関数は必ず戻り値を返さなくてはならない。

■ 解答

(1) ○ (2) × (3) ○ (4) × (5) ×

■ 解説

(1) 関数はデフォルト引数を持つことができます (p.138)。

(2) 引数の型、個数で同名の関数が識別されます。戻り値の型のみが異なっても識別されません (p.142)。

(3) 変数は使用する前であれば、どこででも宣言できます (p.130)。

(4) ポトムアップ方式のときは不要です (p.123 のコラム)。

(5) void を使えば、何も返さない関数を作れます (p.122)。

■ 問題

2. 正しいものには○、正しくないものには×を付けてください。

(1) 一度宣言した変数はプログラムが終了するまで有効である。

(2) static な変数は、それを宣言した関数から制御が離れても値を保持している。

(3) 関数は自分自身を呼び出すことはできない。

(4) 関数のオーバーロードとデフォルト引数は同時に使用できない。

(5) 関数は、戻り値の型が異なればオーバーロードすることができる。

■ 解答

(1) × (2) ○ (3) × (4) × (5) ×

■ 解説

(1) 変数には寿命があります (p.125)。

(2) static 変数はプログラムが終了するまで値を保持します (p.130)。

(3) 関数は自分自身を呼び出せます。これを再帰呼び出しといいます (p.135)。

(4) 関数のオーバーロードとデフォルト引数は同時に使えますが、注意すべき点もあります (p.144)。

(5) 引数の個数や型が違う場合のみ同じ関数名を使えます (p.142)。

第7 章「ポインタと参照」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。

(1) ポインタは、使用される前なら任意の位置で宣言することができる。

(2) 「int *p1, p2;」と宣言したとき、p2 もポインタである。

(3) ポインタに格納したアドレスはあとから変更することはできない。

(4) 参照は宣言時に初期化する必要がある。

(5) 参照を関数に渡しても、呼び出し先の関数で変数の値を変更できないようにすることができます。

■ 解答

(1) ○ (2) × (3) × (4) ○ (5) ○

■ 解説

(1) ポインタは、使用される前ならどこでも宣言できます (p.155)。

(2) p1 のみint 型へのポインタで、p2 はint 型になります (p.160)。

(3) ポインタに格納したアドレスは変更可能です (p.160)。

(4) 参照は宣言時に初期化しなければなりません (p.167)。

(5) const を付けると呼び出し先で値を変更できなくなります (p.169)。

第8 章「配列・文字列とポインタ」

■ 問題

1. 配列の説明について、正しいものには○、正しくないものには×を付けてください。

(1) int a[5]; と宣言するとa[1]、a[2]、a[3]、a[4]、a[5]をint 型の変数として扱うことができる。

(2) int a[3]; と宣言したとき、a はa[0]のアドレスを表す。

(3) int a[2]; と宣言したとき、a+1 はa[1]のアドレスを表す。

(4) 配列とポインタはまったく同じように扱うことができる。

(5) 文字はint 型として扱うこともできる。

■ 解答

(1) × (2) ○ (3) ○ (4) × (5) ○

■ 解説

- (1) `a[0]`、`a[1]`、`a[2]`、`a[3]`、`a[4]`をint型の変数として扱うことができます (p.171)。
- (2) 配列名はその配列の先頭要素のアドレスを表しています (p.177)。
- (3) 配列`a`に対して`a+n`は`a[n]`のアドレスを表します (p.178)。
- (4) 配列名は、ポインタと違って変数ではないので書き換えられません (p.179)。
- (5) 文字をint型にキャストすると、ASCIIコードが得られます (p.187、p.190)。

■ 問題

2. アドレスとメモリの説明について、正しいものには○、正しくないものには×を付けてください。

- (1) 文字列"ABC"の式の値は、先頭文字'A'の置かれたアドレスである。
- (2) 「`char *str = "ABC";`」とすると`*str + 1`は'B'を表す。
- (3) `main`関数は引数を持つことができない。
- (4) `int main(int argc, char *argv[]){ ... }`とした場合、`argv[0]`には、このプログラム自身の名前が格納されている。
- (5) C++では、メモリを動的に確保するのに`new`関数を利用できる。

■ 解答

- (1) ○ (2) ○ (3) × (4) ○ (5) ×

■ 解説

- (1) 文字列の式の値は、先頭文字へのアドレスを表します (p.192)。
- (2) `*str + 1`は`str[1]`すなわち'B'を表します (p.193)。
- (3) `main`関数は、コマンドライン引数をとることができます (p.211)。
- (4) コマンドライン引数`argv[0]`にはプログラム自身の名前が格納されます (p.212)。
- (5) `new`は関数ではなく演算子です (p.219)。

第9章「クラスの基本」

■ 問題

1. 次のクラス定義は正しいでしょうか。もし誤りがあるとすれば、それはどこですか。

```
class A {  
    int x, y;  
public:  
    func();  
}
```

■ 解答・解説

「private:」は省略可能なので、その点については問題ありません。「int x, y;」という書き方も間違いではありません。誤りは、メンバ関数func の戻り値の型が示されていない点と、クラス定義の最後付けるべきセミコロンが抜けている点です。

■ 問題

2. 次のクラス定義は正しいでしょうか。もし誤りがあるとすれば、それはどこですか。

```
class A {  
    private:  
        int x;  
    public:  
        int func();  
    private:  
        int y;  
};
```

■ 解答・解説

問題のない正しい定義です。「public:」や「private:」の位置は任意であり、このように分離してもかまいません。

■ 問題

3. クラスの説明について、正しいものには○、正しくないものには×を付けてください。

- (1) クラスの公開部のメンバにアクセスするには「オブジェクト名-> メンバ名」のようにアロー演算子を使う。
- (2) クラスのデータメンバは、int a = 10; のように初期化できる。
- (3) クラスのインスタンスをオブジェクトという。
- (4) クラス定義で、アクセスコントロールを省略するとpublic とみなされる。
- (5) メンバ関数をクラス外で定義することはできない。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ×

■ 解説

(1) メンバへのアクセスにはドット演算子を使用します。ポインタを介してメンバにアクセスする場合にアロー演算子を使います (p.228)。

(2) クラスのデータメンバは宣言と初期化を同時にすることはできません (p.236)。

(3) クラスのインスタンス (instance、実体) はオブジェクトとも呼ばれます (p.228)。

(4) アクセスコントロールを指定しないと、private とみなされます (p.227)。

(5) メンバ関数はクラスの外で定義することができます (p.229)。

■ 問題

4. コンストラクタの説明について、正しいものには○、正しくないものには×を付けてください。

(1) コンストラクタはプログラマが必ず記述しなくてはならない。

(2) コンストラクタはオーバーロードできる。

(3) コンストラクタを記述したら必ずデストラクタも記述しなくてはならない。

(4) コンストラクタは引数をとることができます。

(5) コンストラクタの戻り値の型はvoid である。

■ 解答

(1) × (2) ○ (3) × (4) ○ (5) ×

■ 解説

(1) プログラマが記述しない場合は、デフォルトコンストラクタが自動的に生成されます (p.250)。

(2) コンストラクタも関数の一種なのでオーバーロードできます (p.240)。

(3) コンストラクタだけをプログラマが記述してもかまいません (p.237 のサンプルプログラム等)。

(4) コンストラクタは引数をとることができます (p.237)。

(5) コンストラクタに戻り値はありません (void 型ではない) (p.237)。

■ 問題

5. コピーコンストラクタについて説明してください。

■ 解答

オブジェクトがコピーされるときに自動的に呼び出される特殊なコンストラクタで、クラスに含まれるメンバの値でオブジェクトを初期化します。コピーコンストラクタをプログラマが記述していない場合は、自明なコピーコンストラクタがコンパイラによって自動的に作成されます (p.244)。

■ 問題

6. デストラクタの説明について、正しいものには○、正しくないものには×を付けてください。

(1) デストラクタの名前はプログラマが自由に決めることができる。

(2) デストラクタは引数を持つことができる。

(3) デストラクタの戻り値の型はvoid である。

(4) デストラクタは必ずクラス内部に定義を書かなくてはならない。

(5) デストラクタは、必ずしもプログラマが記述する必要はない。

■ 解答

(1) × (2) × (3) × (4) × (5) ◎

■ 解説

- (1) クラス名にチルダ (`) を付けた名前でなくてはなりません (p.251)。
- (2) デストラクタは引数をとりません (p.251)。
- (3) デストラクタに戻り値はありません (void 型ではない) (p.251)。
- (4) 外部で定義してもかまいません (p.252 のサンプルプログラム)。
- (5) プログラムが記述しない場合はデフォルトのデストラクタが生成されます (p.251)。

第10 章 「クラスの継承」

■ 問題

1. クラスの継承について、正しいものには○、正しくないものには×を付けてください。
- (1) あるクラスを基本クラスとして、新しいクラスを派生することができます。
- (2) 基本クラスから派生クラスを派生させる方法は、public と private の2 種類のみである。
- (3) 基本クラスのすべてのメンバは派生クラスに引き継がれる。
- (4) 派生クラスが存在する場合、基本クラスのコンストラクタが実行されることはない。
- (5) 派生クラスのオブジェクトの生成段階で、基本クラスのデストラクタが呼び出される。

■ 解答

(1) ◎ (2) × (3) ◎ (4) × (5) ×

■ 解説

- (1) あるクラスを基本クラスとして、新しいクラスを派生することができます (p.273)。
- (2) 派生させる方法には、public、private、protected があります (p.274)。
- (3) すべて引き継がれます (p.275 の表10-1)。
- (4) 派生クラスのオブジェクトが生成されるときに、基本クラスのコンストラクタが呼び出されます (p.278)。
- (5) 基本クラスのコンストラクタが呼び出されます (p.278)。

■ 問題

2. 正しいものには○、正しくないものには×を付けてください。

- (1) 派生クラスは、基本クラスと同じ名前、同じ引数のメンバ関数を持つことはできない。
- (2) オーバーライドされた基本クラスのメンバ関数は呼び出すことができない。
- (3) 抽象クラスはオブジェクトを生成できない。
- (4) オーバーライドされたメンバ関数を持つクラスを抽象クラスという。

(5) 仮想関数をメンバに持つクラスを抽象クラスという。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ×

■ 解説

(1) メンバ関数のオーバーライドです (p.281)。

(2) スコープ解決演算子を利用して呼び出すことができます (p.281)。

(3) 抽象クラスはオブジェクトを生成できません (p.289)。

(4) 純粹仮想関数を 1 つ以上メンバに持つクラスを抽象クラスといいます (p.289)。

(5) 仮想関数をメンバに持っていても抽象クラスとはいいません (p.289)。

■ 問題

3. 正しいものには○、正しくないものには×を付けてください。

(1) フренд関数と認められた関数でも、そのクラスの非公開部にアクセスすることはできない。

(2) フренд関数は、C++ のデータの隠蔽性を破壊することはない。

(3) 派生クラスのオブジェクトが生成されるとき、基本クラスのデフォルトコンストラクタが実行される。

(4) 派生クラスのコンストラクタで、基本クラスのコンストラクタを引数付きで呼び出すことはできない。

(5) オーバーライドされた基本クラスのメンバ関数は、アクセスコントロールを利用して呼び出すことができる。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ×

■ 解説

(1) フренд関数はそのクラスの非公開部にアクセスできます (p.292)。

(2) フренд関数は C++ のデータ隠蔽性を破壊する危険があります (p.294 のコラム)。

(3) 正しい (p.279)。

(4) 基本クラスのコンストラクタを引数付きで呼び出すこともできます (p.280)。

(5) スコープ解決演算子を利用して呼び出すことができます (p.281)。

■ 問題

4. 正しいものには○、正しくないものには×を付けてください。

(1) 基本クラスのポインタに、派生クラスのアドレスを代入することはできない。

(2) 仮想関数とは中身のない関数である。

(3) 抽象クラスは、少なくとも 1 つの純粹仮想関数をメンバとして持っている。

(4) クラスA からクラスB を派生させ、クラスB からクラスC を派生させた。このような派生の仕方を多重派生とい

う。

(5) 多重継承をする場合、基本クラス間で同名のメンバ関数が存在してもよい。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ○

■ 解説

(1) 可能です (p.286)。

(2) オーバーライドされた基本クラスのメンバ関数に `virtual` を冠したものを仮想関数といいます (p.286)。

(3) 正しい (p.289)

(4) 複数の基本クラスから派生クラスを派生させることを多重継承といいます (p.303)。

(5) 呼び出すときにスコープ解決演算子を使えば、同名のメンバ関数があつてもかまいません (p.305)。

第11章「演算子のオーバーロード」

■ 問題

1. 演算子オーバーロードの説明について、正しいものには○、正しくないものには×を付けてください。

(1) C++ では演算子をオーバーロードすることができる。

(2) C++ では、独自の演算子を定義できる。

(3) 演算子のオーバーロードをフレンド関数で実現することはできない。

(4) 単項演算子はオーバーロードすることはできない。

(5) 後置型インクリメント演算子はオーバーロードすることができない。

■ 解答

(1) ○ (2) × (3) × (4) × (5) ×

■ 解説

(1) 正しい (p.313)。

(2) 新しい記号を導入して独自の演算子を定義することはできません。

(3) 可能です (p.321)。

(4) 可能です (p.322)。

(5) 古いC++ では後置型インクリメント演算子はオーバーロードできませんでしたが、現在はオーバーロード可能です (p.324)。

■ 問題

2. 正しいものには○、正しくないものには×を付けてください。
- (1) 関係演算子のオーバーロード関数はint型の値を返さなくてはならない。
- (2) オーバーロードされた演算子の優先順位を定義することができる。
- (3) 演算子のオーバーロードは、必ずクラスのメンバ関数で実現しなくてはならない。
- (4) 複合代入演算子はオーバーロードすることはできない。
- (5) スコープ解決演算子もオーバーロードすることができる。

■ 解答

(1) × (2) × (3) × (4) × (5) ×

■ 解説

- (1) bool型の値を返さなくてはなりません (p.327)。
- (2) 優先順位は元の演算子の優先順位に従います (p.316)。
- (3) フренд関数で実現してもかまいません (p.321)。
- (4) 可能です。
- (5) スコープ解決演算子はオーバーロードできません。

第12章「ファイル入出力」

■ 問題

1. 正しいものには○、正しくないものには×を付けてください。
- (1) ifstreamクラスを利用するには、iostreamをインクルードする必要がある。
- (2) ofstreamクラスのオブジェクトが破棄されるとき、オープンしていたファイルは自動的にクローズされる。
- (3) ofstreamクラスのオブジェクトを生成するときに、コンストラクタを利用してファイルをオープンすることもできる。
- (4) C++では、ファイルをバイナリモードでオープンすることはできない。
- (5) C++では、C言語のようにFILE構造体を利用することはできない。

■ 解答

(1) × (2) ○ (3) ○ (4) × (5) ×

■ 解説

- (1) fstreamをインクルードする必要があります (p.336)。
- (2) 正しい (p.336)。
- (3) 正しい (p.336)。

(4) バイナリモードでもオーブンできます (p.336 の表12-1)。

(5) 可能です。

第13 章「テンプレート」

■ 問題

1. テンプレートの説明について、正しいものには○、正しくないものには×を付けてください。

(1) 関数テンプレートは、関数のオーバーロードと同義である。

(2) 関数テンプレートは、引数や戻り値の型に対応して処理内容が異なる場合にその威力を発揮する。

(3) 関数テンプレートはオーバーロードすることができる。

(4) 関数テンプレートで置き換わる型は1つのみに制限されている。

(5) テンプレートは、C++ の多態性（ポリモーフィズム）を実現するための1つの手段である。

■ 解答

(1) × (2) × (3) ○ (4) × (5) ○

■ 解説

(1) オーバーロードをさらに一般化したもので、同義ではありません (p.355)。

(2) 引数や戻り値の型が異なるだけで処理内容がまったく同じ場合に威力を発揮します (p.355)。

(3) 正しい (p.359)。

(4) 複数の型を置き換えることができます (p.357)。

(5) 正しい (p.3)。

第14 章「標準テンプレートライブラリ」

■ 問題

1. 標準テンプレートライブラリの説明について、正しいものには○、正しくないものには×を付けてください。

(1) 標準テンプレートライブラリの中核をなすのは、コンテナ、アルゴリズム、反復子の3つである。

(2) vector は、キーと値をペアで格納することができます。

(3) sort アルゴリズム関数は map コンテナに対応している。

(4) 反復子に&を付けると、反復子が指している要素の内容を表す。

(5) set コンテナにキーを格納すると、自動的にソートされる。

■ 解答

(1) ○ (2) × (3) × (4) × (5) ○

■ 解説

- (1) 正しい (p.365)。
- (2) vector はキーと値をペアで格納することはできません (p.366)。
- (3) sort 関数は、vector クラスとdeque クラスのみに対応しています (p.372)。
- (4) 反復子に* を付けると、それが指している要素の内容を表します (p.369)。
- (5) set コンテナにキーを格納すると、自動的にソートされます (p.376)。

第15 章「例外処理」

■ 問題

1. 例外処理の説明について、正しいものには○、正しくないものには×を付けてください。

- (1) 1 つのtry ブロックに対して対応するcatch ブロックは1 つとは限らない。
- (2) catch ブロックからthrow することはできない。
- (3) try ブロック内の関数から例外をthrow することはできない。
- (4) catch ブロックで受け取れる例外の型は、int やdouble など既存のデータ型に限られる。
- (5) try ブロックの中にさらにtry ブロックを作ることができる。

■ 解答

- (1)○ (2)× (3)× (4)× (5)○

■ 解説

- (1) catch ブロックは、引数の型が異なれば複数あってもかまいません (p.383)。
- (2) catch ブロックからthrow することもできます (p.391)。
- (3) 可能です (p.386)。
- (4) 自分で定義したクラスをthrow することもできます (p.389)。
- (5) ネストすることも可能です (p.391)。

第16 章「C++11の新機能」

■ 問題

1. C++11に関する説明について、正しいものには○、正しくないものには×を付けてください。

- (1) C++11で導入されたauto を使うと、関数の戻り値の型をコンパイラが推論してくれる。
- (2) 範囲ベースのfor 文を使うと、配列のすべての要素に順にアクセスできる。
- (3) ラムダ関数を使う場合は、lambda ヘッダをインクルードする。
- (4) ラムダ関数は、ラムダ関数の外部にある変数にはアクセスできない。
- (5) ラムダ関数は引数をとることも、戻り値を返すこともできる。

■ 解答

(1) ○ (2) ○ (3) ✗ (4) ○ (5) ○

■ 解説

(1) C++11で導入されたauto を使うと、関数の戻り値の型をコンパイラが推論してくれます (p.396)。

(2) 範囲ベースのfor 文を使うと、配列のすべての要素に順にアクセスできます (p.398)。

(3) 誤り。

(4) ラムダ関数は、ラムダ関数の外部にある変数にはアクセスできます (p.401)。

(5) ラムダ関数は引数をとることも、戻り値を返すこともできます (p.401)。