錬成問題

▪ あるクラスのオブジェクトに含まれるメンバとしてのオブジェクトは、(1) と呼ばれる。クラスのデータメンバ型が他のクラス型であるとき、(2) の関係が成立する。

▪ コンストラクタは多重定義 (3) 。メンバ関数は多重定義 (4) 。
クラス型変数の値を等価演算子 == または != によって比較することは (5) 。
関数は、配列を返却することは (6) 。また、クラス型の値は返却 (7) 。
▶ 共通の選択肢：(a)できる　(b)できない

▪ 引数を渡すことなく呼び出せるコンストラクタを、(8) コンストラクタと呼ぶ。

▪ クラス型のオブジェクトが同一型オブジェクトの値で初期化される際は、(9) コンストラクタによって、(10) データメンバの値がコピーされる。(9) コンストラクタは、(11) 。
また、クラス型のオブジェクトに、同一型のオブジェクトの値が代入されるときは、代入演算子の働きによって、(12) データメンバの値がコピーされる。
▶ (10) と (12) の選択肢：(a)すべての　(b)組込み型の　(c)クラス型の
▶ (11) の選択肢：(a)必要に応じてコンパイラによって自動的に作られる
(b)プログラマが定義しなければならない

▪ コンストラクタの明示的な呼出しによって生成される、名前をもたないオブジェクトは、(13) オブジェクトと呼ばれる。(13) オブジェクトは、不要になった時点で、自動的に破棄 (14) 。
▶ (14) の選択肢：(a)される　(b)されない

▪ 定値オブジェクトに対して起動する必要のあるメンバ関数は、(15) メンバ関数として実現しておかなければならない。

▪ クラスの利用者にとっての定値性とは無関係であり、オブジェクトの内部的な状態を表すようなデータメンバは、(16) メンバとして実現しなければならない。

▪ メンバ関数は、自分が所属するオブジェクトを指す (17) ポインタをもっている。そのため、所属するオブジェクトそのものは、(18) によって表せる。

▪ 文字列に対する文字の挿入・抽出を実現する文字列ストリームの利用には、<(19)> ヘッダのインクルードが必要である。なお、文字の挿入が可能であって抽出が不可能な、出力専用の文字列ストリームは、(20) クラスとして提供される。

▪ 以下に示す *Rectangle* は、長方形を表すクラスである。

```
class Rectangle {           // 長方形
    int height, width;      // 高さと幅
public:
    Rectangle(int height, int widht = 1) {
        this->height = height;    this->width = width;
    }
    void put(char ch = '+') {
        for (int i = 1; i <= height; i++) {
            for (int j = 1; j <= width; j++)
                cout << ch;
            cout << '\n';
        }
    }
};
```

▫ 以下に示す各コードについて、コンパイルエラーとならないものに○を、コンパイルエラーとなるものに×を埋めよ。

```
(21)  Rectangle a(5);
(22)  Rectangle b(5, 7);
(23)  Rectangle c = Rectangle(5);
(24)  Rectangle d = Rectangle(5, 7);
(25)  Rectangle e = 5;
(26)  Rectangle f = 5, 7;
(27)  Rectangle g[2];
(28)  Rectangle h[2] = {(5, 7), (2, 6)};
(29)  Rectangle i[2] = {{5, 7}, {2, 6}};
(30)  Rectangle j[2] = {Rectangle(5, 7), Rectangle(2, 6)};
```

▫ コンストラクタから網かけ部の **this->** を削除すると、 (31) 。

▶ 選択肢：(a)コンパイルできなくなる

(b)データメンバ *height* に不定値が代入される

▫ 以下のように改良したクラスを作成せよ。

◇ クラス *Rectangle* のオブジェクトは、どのような方法で構築しても、幅を表すデータメンバ *width* の値が不定値となってしまう。そうならないように改良する。

◇ データメンバ *height* および *width* に対するゲッタ *get_height* と *get_width* およびセッタ *set_height* と *set_width* を追加する。

◇ コンストラクタを、デフォルトコンストラクタとして呼び出せるように変更する。

◇ クラス *Rectangle* 型の **const** オブジェクトに対して、メンバ関数 *put* を呼び出せるように変更する。

```
class Rectangle {           // 長方形
    int height, width;      // 高さと幅
public:

    (32)

};
```

▪ 以下に示すのは、２次元座標クラス *Point2D* および円クラス *Circle* と、それらのクラスを利用するプログラムである。

```
#(33) ___Point2D
#(34) ___Point2D
#include <iostream>
//--- ２次元座標クラス ---//
class Point2D {
    int xp, yp; // Ｘ座標とＹ座標                                    1
public:                                                              2
    Point2D(int x = 0, int y = 0) : (35)(x), (36)(y) { }
    int x() (37) { return xp; }        // Ｘ座標xpのゲッタ
    int y() (37) { return yp; }        // Ｙ座標ypのゲッタ
    void print() (37) { std::cout << "(" << xp << "," << yp << ")"; }
};
#(38)
```

```
#(33) ___Circle
#(34) ___Circle
#include <iostream>
#include "Point2D.h"
//--- 円クラス ---//
class Circle {
    (39) center;       // 中心座標
    int radius;        // 半径
public:
    Circle(const (40)& c, int r) : (41)(c), (42)(r) { }
    (43) get_center() (37) { return center; } // 中心座標centerのゲッタ
    int get_radius() (37) { return radius; }    // 半径radiusのゲッタ
    void print() (37) {                         // 表示
        std::cout << "半径[" << radius << "] 中心座標";  center.(44);
    }
};
#(38)
```

```
#include <iostream>
#include "Point2D.h"
#include "Circle.h"
                                                    3
using namespace std;

int main()
{
    Point2D origin(0, 0);              // 原点
    Circle c1(Point2D(3, 5), 7);       // 中心座標(3, 5) 半径7の円
    Circle c2(Point2D(), 8);           // 中心座標(0, 0) 半径8の円
    const Circle c3(origin, 9);        // 中心座標(0, 0) 半径9の円

    cout << "c1 = ";   c1.(45);   cout << '\n';
    cout << "c2 = ";   c2.(45);   cout << '\n';
    cout << "c3 = ";   c3.(45);   cout << '\n';
}
```

```
c1 = 半径[7] 中心座標(3,5)
c2 = 半径[8] 中心座標(0,0)
c3 = 半径[9] 中心座標(0,0)
```

▫ ヘッダ "Point2D.h" と "Circle.h" は、何度インクルードされても多重定義のコンパイルエラーとならないようにするための“インクルード (46) ”が施されている。

なお、3のインクルードの順序を交換すると、コンパイルエラーと (47) 。

▶ (47) の選択肢： (a)なる　(b)ならない

▫ 円クラス *Circle* と 2 次元座標クラス *Point2D* とのあいだには、has-A の関係が成立して (48) 。

▶ 選択肢：(a)いる　(b)いない

▫ クラス *Point2D* 中の❶は (49) と呼ばれ、❷は (50) と呼ばれる。このコンストラクタは以下のように定義することもできる。

```
Point2D(int x = 0, int y = 0) { xp = x;  yp = y; }      // 別解
```

解答プログラムと別解との違いを述べよ。… (51)

▫ 二つの❷の順序を交換すると、データメンバ *xp* と *yp* の初期化の順序が入れ (52) 。

▶ 選択肢：(a)かわる　(b)かわらない　(c)かわるかどうかは処理系に依存する

▫ クラス *Point2D* のコンストラクタは、デフォルトコンストラクタとして機能 (53) 。クラス *Circle* のコンストラクタは、デフォルトコンストラクタとして機能 (54) 。

▶ 共通の選択肢：(a)する　(b)しない

▫ クラス *Point2D* とクラス *Circle* のメンバ関数 *print* を削除して、同等の表示を行う挿入子 **<<** を追加せよ。各クラスの変更にあわせて、クラスを利用するプログラムも変更すること。… (55)

▪ 以下に示すプログラムの実行結果を示せ。

```
class X {
    string m;
public:
    X(string c) { cout << "m = " << m << '\n';  m = c;
                  cout << "m = " << m << '\n';          }
};
int main()
{
    X x1("ABC");
    X x2 = "XYZ";
    X x3 = X("FBI");
}
```

(56)

▪ 以下に示す *Y* は、前問のクラス *X* をメンバとしてもつクラスである。

```
class Y {
    X a;
public:
    Y(string c) { a = Y(c); }
};
```

このプログラムをコンパイルすると、コンパイルエラーが発生する。その理由を示すとともに、エラーとならないように書きかえたプログラムを示せ。… (57)

▪ **const** メンバ関数は、 (58) に対して呼び出すことができ、**const** でないメンバ関数は (59) に対して呼び出すことができる。

▶ 共通の選択肢：(a) **const** オブジェクト　(b)非 **const** オブジェクト
(c) **const** オブジェクトおよび非 **const** オブジェクト

▪ **const** メンバ関数の中で値を変更できるのは、 (60) のデータメンバである。

▶ 選択肢：(a) **mutable**　(b)非 **mutable**　(c) **mutable** および非 **mutable**

▪ あるクラス *X* のデータメンバ *d* が、デフォルトコンストラクタをもたないクラス *C* 型であるとする。*d* に対する値の設定は (61) 。なお、クラス *C* 型がデフォルトコンストラクタをもつのであれば、*d* に対する値の設定は (62) 。

▶ 共通の選択肢：
(a)コンストラクタ初期化子で行わなければならない
(b)コンストラクタ本体で行わなければならない
(c)コンストラクタ初期化子とコンストラクタ本体のいずれで行ってもよい

▪ データメンバの初期化の順序は (63) 。

▶ 選択肢：(a)データメンバ宣言の順序と同じである
(b)コンストラクタ初期化子の順序と同じである
(c)処理系に依存する