錬成問題

▪ クラスを構成する主要な各要素の名称を示せ。

```
//--- 会員クラス（ヘッダ部）---//
#include <string>
#include <iostream>

class Member {
    std::string full_name;  // 氏名
    int         no;         // 会員番号                          ……(1)
    int         rank;       // 会員ランク

public:
    Member(std::string name, int number, int grade) {
        full_name = name;  no = number;  rank = grade;           ……(2)
    }

    void print() {
        std::cout << "No." << no << "：" << full_name
                  << "［ランク：" << rank << "］\n";
    }                                                            ……(3)

    int get_rank() { return rank; }

    void set_rank(int grade) { rank = grade; }
};
```

オブジェクトの状態を表す［(1)］は、カタカナ４文字で［(4)］とも呼ばれる。一般に、オブジェクト *x* のメンバ *m* をアクセスする式は［(5)］である。

▪ **int** や **double** などのプログラミング言語 C++ によって提供される型は［(6)］型と呼ばれるのに対し、*Member* のようにプログラム上で作成する型は［(7)］型と呼ばれる。

▪ 以下に示すのは、*Member* 型のオブジェクト *takeda* の宣言である。なお、氏名は "武田和宏"、会員番号は 15、会員ランクは 1 であるとする。

```
Member [(8)]([(9)]);
```

▪ 以下に示すのは、*takeda* に所属する *print* を呼び出す式である。このように、オブジェクトに所属する［(3)］を呼び出すことを、『オブジェクトに［(10)］を送る』と表現する。

```
takeda[(11)];
```

▪ 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。

```
[(12)]  cout << takeda::full_name();
[(13)]  cout << takeda:no();
[(14)]  cout << takeda.rank();
[(15)]  takeda::full_name = "ABC";
[(16)]  takeda:no()  = 123;
[(17)]  takeda.rank = 3;
[(18)]  cout << takeda::get_rank();
[(19)]  cout << takeda:get_rank();
[(20)]  cout << takeda.get_rank();
```

▪ クラス *Member* の定義から (2) を丸ごと削除すると、(21)。

▶ 選択肢：(a)コンパイルできなくなる　(b)オブジェクトを生成できなくなる
(c)オブジェクト生成時に引数を渡せなくなる

▪ クラス *Member* はヘッダとして実現されている。クラス定義の前に“**using namespace** std;”の **using** 指令がないため、***string*** と cout を、std::***string*** および std::cout として表している。原則として、ヘッダ内に **using** 指令は (22)。

▶ 選択肢：(a)置くべきである　(b)置かないほうがよい

その理由を示せ。… (23)。

▪ クラス *Member* に所属する三つの (3) には、(24) 結合が与えられる。また、これらの関数は、インライン関数で (25)。

▶ (25) の選択肢：(a)ある　(b)はない　(c)あるかどうかは状況に依存する

▪ クラス *Member* における (2) と (3) の定義を、クラス定義の外に出して実現したプログラムを作成せよ。ヘッダとして利用できるように実現すること。

```
//--- 会員クラス（ヘッダ部）---//
#include <string>
#include <iostream>
class Member {
    (26)
};
(27)
```

▪ クラス *Member* における (2) と (3) の定義を、クラス定義の外に出して実現したプログラムを作成せよ。なお、定義が取り除かれた部分はヘッダとして利用できるようにした上で、(2) と (3) の定義は、独立した別のソースファイルとして実現すること。

```
//--- 会員クラス（ヘッダ部）---//
#include <iostream>
class Member {
    (28)
};
```

```
//--- 会員クラス（ソース部）---//
#include <string>
#include <iostream>
#include "Member.h"
(29)
```

▪ 一般に、ポインタ *p* が指すオブジェクトのメンバ *m* をアクセスする式は (30) *.m* であるが、式 (31) によってもアクセスできる。なお、前者の式で用いられる演算子 **.** は、(32) 演算子と呼ばれ、後者の式で用いられる演算子は、(33) 演算子と呼ばれる。

▪ データメンバを非公開として外部から保護した上で、メンバ関数とうまく連携させてクラスとしてまとめることを (34) という。

▪ データメンバの値を取得して返却するメンバ関数を (35) と呼び、データメンバに値を設定するメンバ関数を (36) と呼ぶ。両者の総称が (37) である。

▪ クラス定義の外で定義されたメンバ関数には、特に指定されない限り、(38) 結合が与えられてインライン関数と (39)。

▶ (39) の選択肢：(a)なる　(b)ならない　(c)となるかどうかは状況に依存する

▪ コンストラクタの返却値型は (40)。生成済みのオブジェクトに対してコンストラクタを呼び出すことは (41)。オブジェクトに対して、コンストラクタが 1 個も定義されていないクラスには、コンパイラによって、(42) コンストラクタが自動的に定義される。

▶ (40) の選択肢：(a) **void** である　(b)宣言できない　(c)自由に指定できる
(d)自分自身の(そのコンストラクタが所属する) クラス型である

▶ (41) の選択肢：(a)できる　(b)できない

▶ (42) の選択肢：(a)自分自身のクラス型の仮引数を 1 個だけ受け取る
(b)任意の型と個数の引数を受け取る
(c)仮引数を受け取らない

▪ クラスのメンバは、公開することも非公開とすることもできる。公開を指定するアクセス指定子は (43) で、非公開を指定するアクセス指定子は (44) である。データを外部から隠して不正なアクセスから守ることは、データ (45) と呼ばれる。