錬成問題

▪ ある事象は、それが自分自身を含んでいたり、自分自身を用いて定義されていたりするときに、 (1) 的であるといわれる。

▪ cout と cin が所属しているのは、 (2) 名前空間である。

▪ 宣言された識別子が、そのソースファイルでのみ通用する性質が (3) であり、そのソースファイルだけでなく、外部のソースファイルにも通用する性質が (4) である。
名前無し名前空間に所属する識別子に与えられるのは、 (5) である。
▶ (5) の選択肢：(a)前者　(b)後者

▪ 右に示すのは、要素数が *n* である配列 *a* の要素中の最小値を返却するように作られた関数の枠組みである。このように、引数の型に依存しないように抽象的に記述された関数の枠組みのことを、 (6) と呼ぶ。

```
(13) <(14) Type>
(15) minof(const Type a[], int n)
{
    (16) min = a[0];
    for (int i = 1; i < n; i++)
        if (a[i] < min)
            min = a[i];
    return (17);
}
```

minof を利用して、要素数が 5 である **int** 型の配列 *z* の最小値を求めて、その値を *c* に代入する処理は、以下のように行う。

```
c = (7)((8), 5);
```

この呼出しによって、**int** 型配列の最小値を求めるための、関数の実体である (9) がコンパイラによって生成される。

実引数の型などに基づいて、 (6) から、 (9) が生成されることを (10) という。 (10) すべき関数をコンパイラが自動的に判断できない文脈や、コンパイラによって自動的に (10) されるものとは異なる関数を呼び出したい文脈では、 (11) な (10) を行う必要がある。また、 (6) の定義をそのまま適用すべきでない型に対しては、 (11) な (12) を行った専用の関数を定義する必要がある。

▪ 右に示すのは、同一型の *a* が指す値と *b* が指す値を交換する (6) である。

```
(13) <(14) Type>
void swap(Type* a, Type* b)
{
    (18) t = *a;
    (19) = *b;
    (20) = t;
}
```

▪ 右に示すのは、1 から *n* までの合計を求めて返却する関数である。

```
unsigned sum(unsigned n)
{
    if (n == 0)
        return (21);
    return n + (22);
}
```

▪ 右に示す 1 と 2 の形式の宣言は、それぞれ、**using** (23) および **using** (24) と呼ばれる。

```
1 using namespace std;
2 using namespace std::cin;
```

▪ 以下に示すのは、二つの整数値 *x* と *y* の最大公約数を求めて返却する関数 *gcd* と、それを利用して、要素数 *n* の **int** 型配列 *a* の全要素の最大公約数を求めて返却する関数である。

```
int gcd(int x, int y)
{
    if (y == (25))
        return x;
    else
        return gcd(y, (26));
}
```

```
int gcdary(int a[], int n)
{
    if (n == 1)
        return a[0];
    else if (n == 2)
        return gcd(a[0], (27));
    else
        return gcd(a[0], (28));
}
```

▪ 以下に示すのは、二つの名前空間と、そのメンバの定義である。

```
namespace English {
    int x = 1;

    void hello()
    {
        cout << "Hello!\n";
    }
}
```

```
namespace Japanese {
    (29) int x;
    void hello();
}
int (30)::x = 2;
void (31)::hello()
{
    cout << "こんにちは。\n";
}
```

名前空間 *English* に所属する *x* と *hello* をアクセスする式は (32) と (33) で、名前空間 *Japanese* に所属する *x* と *hello* をアクセスする式は (34) と (35) である。

▪ 右に示すのは、前問の名前空間 *English* と *Japanese* に対して、*ENG* と *JPN* の別名を与える宣言である。

```
(36) ENG (37);
(36) JPN (38);
```

▪ 右に示すのは、*a*, *b* の最大値を求めるための、名前無し名前空間に所属する関数 *max2* の定義である。

```
(39) {
    int max2(int a, int b) {
        return a < b ? a : b;
    }
}
```

▪ 二つのソースファイルから構成される以下のプログラムの誤りを正せ。 (40)

```
// 指定された範囲の全整数の合計を求める "main.cpp"
#include <iostream>
using namespace std;
int main()
{
    int x, y;
    cout << "整数x：";   cin >> x;
    cout << "整数y：";   cin >> y;

    cout << "1からyまでの合計は"
         << sum(y) << "です。\n";

    cout << "xからyまでの合計は"
         << sum(y, x) << "です。\n";
}
```

```
整数x：3↵
整数y：10↵
1からyまでの合計は55です。
xからyまでの合計は52です。
```

```
// b以上a以下の整数の合計を求める
                              "sum.cpp"
static int sum(int a, int b = 1)
{
    int sum = 0;
    for (int i = b; i <= a; i++)
        sum += i;
    return sum;
}
```

▪ 外部結合が与えられるものにａ、内部結合が与えられるものにｂ、無結合が与えられるものにｃを記入せよ。

- ▫ インライン関数 … (41)
- ▫ **static** 付きで定義された（インラインでない）関数 … (42)
- ▫ **static** 無しで定義された（インラインでない）関数 … (43)
- ▫ 名前無し名前空間の中で定義された（インラインでない）関数 … (44)
- ▫ 定値オブジェクト … (45)
- ▫ 関数の外で **static** 付きで定義された（**const** でない）変数 … (46)
- ▫ 関数の中で **static** 無しで定義された（**const** でない）変数 … (47)
- ▫ 名前無し名前空間の中で定義された（**const** でない）変数 … (48)

▪ あるプログラムが複数のソースファイルから構成されているとする。外部結合をもつ同一名の識別子が、複数のソースファイルで定義されていると、(49) が発生する。

▶ 選択肢：(a)コンパイルエラー　(b)リンク時エラー　(c)実行時エラー

▪ **using** (50) を用いると、単一の識別子を **::** を使うことなく単純名でアクセスできるようになり、**using** (51) を用いると、ある名前空間に属するすべての識別子を **::** を使うことなく単純名でアクセスできる。なお、**::** の名称は (52) 演算子である。

▪ 右に示すのは、*a* が *b* と等しいかどうかを判定して返却する関数と関数テンプレートである（関数版は **int** 型の値を返却し、関数テンプレート版は論理型の値を返却する）。

```
int eq(int a, int b)
{
    return a == b ? 1 : 0;
}

template <class Type>
(57) eq((58) a, (59) b)
{
    return a == b;
}
```

int 型変数 *m*，*n* に対して、関数版 *eq* を呼び出す式は (53) であり、関数テンプレート版 *eq* を呼び出す式は (54) である。また、**double** 型変数 *x*，*y* に対して関数 *eq* を呼び出す式は (55) である。また、*m* と *x* に対して **double** 型の関数テンプレート版 *eq* を呼び出す式は (56) である。

▪ 前問の関数テンプレート版 *eq* は、文字列の等価性の判定を正しく行えない。文字列での判定が正しく行えるように、**const char*** 用に明示的に特殊化した関数テンプレートを作成せよ。

なお、その関数テンプレートを呼び出すプログラムも作成すること。… (60)