# 錬成問題

▪ 単項 **&** 演算子は [ (1) ] 演算子と呼ばれ、オペランドのオブジェクトへの [ (2) ] を生成する演算子である。また、単項 ***** 演算子は [ (3) ] 演算子と呼ばれ、オペランドのアドレスに格納されているオブジェクトをアクセスするための演算子である。

▪ Type* 型のポインタ *ptr* の値が、Type 型オブジェクト *n* のアドレスであるとき、『*ptr* は *n* を [ (4) ] 』と表現する。

▪ *a* が要素数 *n* の Type 型配列であるとき、式 *a* は、*a*[ [ (5) ] ] を指すポインタである。Type* 型ポインタ *p* が、配列 *a* 中の要素 *a*[*k*] を指しているものとする。このとき、*p* + *i* は *a*[ [ (6) ] ] を指し、*p* - *i* は *a*[ [ (7) ] ] を指す。*p* をインクリメントすると、*p* は *a*[ [ (8) ] ] を指すように更新される。

▪ プログラム実行時の任意のタイミングで動的に生成・破棄できるオブジェクトの生存期間は、 [ (9) ] 記憶域期間と呼ばれる。**new** 演算子によるオブジェクトの生成に失敗した場合は、 [ (10) ] 例外が投げられるので、必要に応じて、その例外を [ (11) ] して例外処理を行うことになる。なお、 [ (10) ] 例外は、< [ (12) ] > ヘッダで定義されている。

▪ 以下に示すのは、**int** 型の仮引数 *a* を *b* で割った商と剰余（小数点以下は切捨て）を、*quot* が指す変数と *rem* が指す変数に格納する関数である。

```
void div(int a, int b, [ (13) ] quot, [ (14) ] rem)
{
    [ (15) ] = a / b;      // 商
    [ (16) ] = a % b;      // 剰余
}
```

▪ 以下に示すのは、**int** 型の仮引数 *a* と *b* の和と差を、*sum* が指す変数と *diff* が指す変数に格納する関数である。

```
void sum_diff(int a, int b, [ (17) ] sum, [ (18) ] diff)
{
    [ (19) ] = a + b;                    // 和
    [ (20) ] = a > b ? a - b : b - a;    // 差
}
```

▪ 以下に示すプログラム部分の実行結果を示せ。

```
int x = 55;
int*p = &x;
cout << 5**p;
```

[ (21) ]

▪ 右に示すのは、**double** 型オブジェクトを生成して、3.14 を代入後に表示・破棄するプログラムである。

```
double* a = new [ (22) ];
[ (23) ] = 3.14;
cout << [ (23) ] << '\n';
[ (24) ] a;
```

▪ 以下に示すのは、二つの整数値AとBを対話的に読み込む関数である。実行例に示すように、Bに読み込む値は、Aの値以上でなければならない（そうでない場合は、再入力させる）。

```
void scan2acdint(int* a, int* b)
{
    cout << "二つの整数を昇順に入力せよ。\n";
    cout << "A：";   cin >> [ (25) ];
    do {
        cout << "B：";   cin >> [ (26) ];
    } while ([ (27) ] < [ (28) ]);
}
```

```
二つの整数を昇順に入力せよ。
A：7↵
B：4↵
B：6↵
B：8↵
```

▪ 以下に示すプログラムの実行結果を示せ。

```
int x = 5, y = 7, sw;
cout << "値を変更する変数[0…x / 1…y]：";
cin >> sw;
int* ptr = (sw == 0) ? &x : &y;
*ptr = 999;
cout << "x = " << x << '\n';
cout << "y = " << y << '\n';
```

```
値を変更する変数[0…x / 1…y]：0↵
x =
y =  (29)
```

```
値を変更する変数[0…x / 1…y]：1↵
x =
y =  (30)
```

▪ 右に示すのは、受け取ったint型オブジェクトへのポインタxとyが指すオブジェクトの値を交換する関数である。

```
void swap( (31) x,  (32) y)
{
    (33) t = *x;
    *x = *y;
    *y = t;
}
```

▪ 右に示すのは、前問の関数swapを利用して、受け取ったint型オブジェクトへのポインタxとyが指すオブジェクトの値を昇順にソートする（*x≦*yとなるように並べかえる）関数である。

```
void sort2( (34) x,  (35) y)
{
    if ( (36) > (37) )
        swap( (38) ,  (39) );
}
```

▪ 右のように変数が宣言されている。nの型は (40) で、pの型は (41) で、*pの型は (42) である。

```
int n;
int* p;
```

▪ 以下に示すのは、int型配列vの先頭要素に0を代入する関数である。

```
void set0(int v  (43) )
{
    v[ (44) ] = 0;
}
```

```
void set0(int  (46)  v)
{
     (47) (v +  (48) ) = 0;
}
```

```
void set0(int v  (43) )
{
     (45) v = 0;
}
```

```
void set0(int  (46)  v)
{
    0[ (49) ] = 0;
}
```

いずれの関数においても、仮引数vの型は (50) である。

▪ 右に示す関数 *func* における各文に対して、文法的に正しいものには○を、そうでないものには×を記入せよ。

```
void func(const int *p, int *q)
{
    int a[5], b[5];
    int* ptr;

    ptr = a;          … (51)
    ptr = q;          … (52)
    ptr = &a[0];      … (53)
    ptr = &p[0];      … (54)
    a = b;            … (55)
    a = ptr;          … (56)
    *p = 1;           … (57)
    *q = 1;           … (58)
    *ptr = 1;         … (59)
    p[0] = 1;         … (60)
    q[0] = 1;         … (61)
    ptr[0] = 1;       … (62)
    *p[0] = 1;        … (63)
    *q[0] = 1;        … (64)
    *ptr[0] = 1;      … (65)
}
```

▪ ポインタに単項 * 演算子を適用することによって、ポインタが指すオブジェクトを間接的にアクセスすることを (66) という。ポインタ *p* がオブジェクト *x* を指すとき、*p は *x* の別名すなわち (67) となる。

▪ いかなるオブジェクトも関数も指さないポインタが、(68) である。そのポインタを表す (68) 定数は、< (69) > ヘッダでオブジェクト形式マクロ (70) として定義されている。

▪ 右に示すのは、要素数 *n* の **int** 型配列 *a* の最大値をもつ要素の添字を求めて返却する関数である。最大値をもつ要素が複数存在する場合は、最も先頭の添字を返却する。

たとえば、配列の要素が {1, 3, 5, 4, 5, 3} であれば、最大値 5 をもつ要素の最も先頭要素のインデックスである 2 を返却する。

```
int max_index( (71) int a[], int n)
{
    int max = a[0];
    int idx = (72);

    for (int i = 1; i < n; i++) {
        if (a[i] > (73)) {
            max = (74);
            idx = (75);
        }
    }
    (76);
}
```

▪ 右の関数 *func* に渡す配列の要素数は、4 で (77)。

▶ 選択肢：(a)なければならない　(b)なくてもよい

```
void func(int a[4])
{
    // …中略… //
}
```

▪ 右に示すのは、*n* 行 3 列の **int** 型 2 次元配列の全構成要素に 0 を代入する関数である。

仮引数 *a* の型は、(78) である。

```
void fill0(int a(79), int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < 3; j++)
            a[(80)][(81)] = 0;
}
```

▪ 以下に示すのは、*p* が指すオブジェクトの先頭 *n* バイトの全ビットを 0 にする関数である。

```
void fill_mem(void* p, int n)
{
    unsigned char* q = (82)<unsigned char*>((83));
    for (int i = 0; i < n; i++)
        (84) = 0;
}
```

▪ 右に示すのは、**int** 型配列 *b* の全要素を配列 *a* にコピーする関数である。

```
void ary_cpy(int* a, const int* b, int n)
{
    while (n-- > 0)
        *[ (85) ]++ = *[ (86) ]++;
}
```

▪ 配列 *x* と *y* が右のように宣言されているとする。

```
int x[15];
int y[25];
```

前問の関数 *ary_cpy* の呼出しによって、配列 *b* 内のどの要素が配列 *a* のどの要素にコピーされるのかを示せ。

- *ary_cpy*(*a*, *b*, 15)　　*b*[ (87) ] ～ *b*[ (88) ] の要素が *a*[ (89) ] ～ *a*[ (90) ] にコピーされる。
- *ary_cpy*(&*a*[0], *b*, 8)　　*b*[ (91) ] ～ *b*[ (92) ] の要素が *a*[ (93) ] ～ *a*[ (94) ] にコピーされる。
- *ary_cpy*(&*a*[2], &*b*[4], 5)　　*b*[ (95) ] ～ *b*[ (96) ] の要素が *a*[ (97) ] ～ *a*[ (98) ] にコピーされる。

▪ 右に示すのは、要素数 *n* の **int** 型配列 *b* の全要素を配列 *a* に逆順にコピーする関数である。

```
void rvs_cpy(int* a, const int* b, int n)
{
    const int* p = &b[n - 1];
    while (n-- > 0)
        *[ (99) ]++ = *[ (100) ]--;
}
```

▪ 右に示すのは、要素数 *n* の **int** 型配列 *b* の非負の要素を先頭から順に配列 *a* にコピーする関数である。返却するのは、*a* にコピーされた要素数である。

たとえば、配列 *b* の要素が {1, 0, -1, -2, 5} であれば、配列 *a* の先頭 3 要素に {1, 0, 5} を格納するとともに 3 を返却する。

```
int ary_cpy2(int* a, const int* b, int n)
{
    int count = [ (101) ];
    while (n-- > 0) {
        if (*b >= 0) {
            *[ (102) ]++ = *[ (103) ];
            count++;
        }
        [ (104) ]++;
    }
    return count;
}
```

▪ 以下に示すのは、要素数 *n* の配列 *a* に含まれる値が *key* である要素を探索する関数である。*key* が存在する場合は、その要素の添字（複数ある場合は最も先頭の要素の添字）を返却し、存在しない場合は -1 を返却する。

```
int search(int* a, int n, int key)
{
    int* p = a;

    while (n-- > 0) {
        if (*[ (105) ] == key)
            return p - [ (106) ];
        else
            p++;
    }
    return -1;
}
```

```
int search(int* a, int n, int key)
{
    for (int i = 0; i < n; i++)
        if (*[ (107) ]++ == key)
            return [ (108) ];
    return -1;
}
```

▪ あらゆる型のオブジェクトを指すことのできる特殊なポインタが (109) へのポインタである。任意の型のポインタを (109) へのポインタに代入する際は (110) であり、その逆の代入では (111) である。

▶ (110) と (111) の選択肢：(a)明示的なキャストが必要　(b)明示的なキャストは不要

▪ 右に示すのは、要素数 *n* の **int** 型配列 *a* の非負の要素の添字を先頭から順に配列 *idx* に格納する関数である。返却するのは、*idx* に格納された添字の個数である。

たとえば、配列 *a* の要素が {1, 0, -1, -2, 5} であれば、配列 *idx* の先頭 3 要素に {0, 1, 4} を格納するとともに 3 を返却する。

```
int search_idx(int* a, int* idx, int n)
{
    int* p = idx;
    for (int i = 0; i < n; i++)
        if (a[i] >= 0)
            (112) = i;
    return p - (113);
}
```

▪ 以下に示すプログラムは、ポインタ *p1* と *p2* が *x* と *y* を指すように宣言し、その後二つのポインタの値を入れかえる（*p1* が *y* を指して、*p2* が *x* を指すように更新する）プログラムである。

```
int x = 123,  y = 456;
int* p1 = &x;            // p1はxを指す
int* p2 = &y;            // p2はyを指す
int (114);

temp = p1;
p1 = (115);
(116) = temp;

cout << "*p1の値＝" << *p1 << '\n';
cout << "*p2の値＝" << *p2 << '\n';
```

```
*p1の値＝456
*p2の値＝123
```

▪ 以下に示すのは、要素数 *n* の **long** 型配列を生成・破棄するプログラムである。

なお、左のプログラムでは、生成に失敗した場合に、(117) のメカニズムを用いて、『生成失敗!!』と表示してプログラムを終了する。

```
#include <iostream>
#include <(118)>

using namespace std;

int main()
{
    int n;
    long* a;
    cout << "要素数：";  cin >> n;
    (119) {
        a = (120) long[(121)];
    }
    (122) ((123)) {
        cout << "生成失敗!!\n";
        return 1;
    }
    (124) a;    // 解放
}
```

```
#include <iostream>
#include <(125)>

using namespace std;

int main()
{
    int n;
    long* a;
    cout << "要素数：";  cin >> n;
    a = (126) long[(127)];
    if (a == (128)) {
        cout << "生成失敗!!\n";
        return 1;
    }
    (124) a;    // 解放
}
```