錬成問題

▪ 文字の並びを表現するのが文字列である。文字列の終端は、文字コードが (1) である (2) 文字である。文字列リテラルには、(3) 記憶域期間が与えられ、その末尾には (2) 文字が付加される。

(2) 文字を含めて最大10文字を配列 *s* に読み込むには、cin. (4) とする。

▪ 以下に示すプログラムの実行結果を示せ。

```
cout << sizeof("")          << ' '
     << sizeof("\0")        << ' '
     << sizeof("ABC")       << ' '
     << sizeof("ABC\0012") << '\n';
```

(5)

▪ 以下に示すのは、いずれも **char** 型配列 *s* が文字列 **"FBI"** を表すように初期化する宣言である。いずれも、(6) バイトの記憶域を占有する。

```
char s[4] = {(7), (8), (9), (10)};
char s[4] = {"(11)"};
char s[4] = "(12)";
```

▪ 以下に示すのは、ポインタ *p* が文字列リテラル **"FBI"** の先頭文字を指すように初期化する宣言である。ポインタと文字列リテラルをあせわて (13) バイトの記憶域を占有する。

```
char (14) = "FBI";
```

▪ 以下に示すのは、三つの文字列 **"ABC"**, **"X"**, **"123"** を表す配列の宣言である。

```
char s[3][(15)] = {"ABC", "X", "123"};     // 各列は4文字分の配列
char (16)p[]    = {"ABC", "X", "123"};     // 各要素は文字列を指す
```

ここで、配列 *s* の要素型は (17) で要素数は (18) である。また、配列 *p* の要素型は (19) で要素数は (20) である。

▪ 右に示すのは、文字列 *s* を空にする関数である。

```
void null(char s(21))
{
    *s = (22);
}
```

▪ 右に示すのは、文字列 *s* が **"ABC"** であれば **true** を、そうでなければ **false** を返却する関数である。

```
bool isABC(const char (23)s)
{
    if ((24)++ != 'A') return false;
    if ((24)++ != 'B') return false;
    if ((24)++ != 'C') return false;
    if ((25) != (26)) return false;
    return true;
}
```

▪ 以下に示すプログラムの実行結果を示せ。

```
char str[] = "ABC\nDEF\ØGHI\n";
cout << str;
```

(27)

▪ 以下に示すのは、文字列 s 中の小文字を大文字に、大文字を小文字に変換して表示する関数である。なお、このプログラムには、<(28)> ヘッダのインクルードが必要である。

```
void put_altstr(const char (29) s)
{
    while ((30)) {
        cout << ((31)((30)) ? (32)<char>(toupper((30)))
                            : (33)<char>(tolower((30))));
        (34)++;
    }
}
```

▪ 右に示すのは、二つの文字列 s1 と s2 が等しければ（すべての文字が同じであれば）**true** を、そうでなければ **false** を返却する関数である。

```
bool str_eq(const char* s1, const char* s2)
{
    while (*s1 (35) *s2) {
        if (*s1 == (36))
            return (37);
        s1++;  s2++;
    }
    return (38);
}
```

▪ 以下に示すのは、文字列 s に含まれている全数字文字の個数を返却する関数である。

```
int digit_no(const char s(39))
{
    int count = (40);
    while ((41)) {
        if ((41) >= (42) && (41) <= (43))
            count++;
        (44)++;
    }
    return count;
}
```

▪ 以下に示すプログラムの実行結果を示せ。

```
char* a = "FBI", * b = "CIA";
char* t = a;   a = b;  b = t;
cout << "a = " << a << '\n';
cout << "b = " << b << '\n';
```

```
a =
b =
```

(45)

▪ コンパイルエラーとならないものに○を、なるものに×を埋めよ。

```
        char a[] = "ABC";
        char* p  = "ABC";
(46)    a = "XYZ";
(47)    p = "XYZ";
```

▪ 以下に示す三つの *str_len* は、文字列 *s* のナル文字を含まない長さを求める関数である。

```
int str_len(const char* s)
{
    int len = 0;
    while (s[ (48) ])
        len++;
    return (49);
}
```

```
int str_len(const char* s)
{
    int len = 0;
    while (*(50)++)
        len++;
    return (51);
}
```

```
int str_len(const char* s)
{
    const char* p = (52);
    while (*(53))
        p++;
    return p - (54);
}
```

▪ 文字列の長さを返却する標準ライブラリである ***strlen*** 関数の返却値型は (55) である。

strlen 関数や ***strcpy*** 関数などの文字列ライブラリを提供するヘッダは <(56)> である。

以下に示すのは、文字列 *s* の文字列の長さを変数 *len* に求め、文字列 *s2* を *s1* にコピーし、文字列 *c* を文字列 *b* と *a* の両方にコピーし、文字列 *x* の末尾に文字列 *y* を連結するプログラムである。

```
len = strlen( (57) );                    // 文字列sの長さを求めてlenに代入
strcpy( (58) , (59) );                   // 文字列s2をs1にコピー
strcpy( (60) , (61) ( (62) , (63) ));    // 文字列cをbとaにコピー
strcat( (64) , (65) );                   // 文字列xの末尾にyを連結
```

▪ 右に示すのは、文字列 *s2* の文字の並びを反転した文字列を *s1* にコピーする関数である（文字列 *s2* が "ABC" であれば、*s1* に "CBA" をコピーする）。

```
void str_rvcopy(char s1[], char s2[])
{
    int len = strlen( (66) );
    for (int i = 0; i < (67) ; i++)
        s1[ (68) ] = s2[ (69) ];
    s1[len] = 0;
}
```

▪ 右に示すのは、文字列 *s* の文字の並びを反転する関数である。文字列 *s1* が "ABC" であるとき、

```
cout << str_rvs(s1);
```

を実行すると、(70) と表示される。

```
char* str_rvs(char s[])
{
    int len = strlen( (71) );
    for (int i = 0; i < (72) ; i++) {
        char temp = s[i];
        s[i] = (73) ;
        (74) = (75) ;
    }
    return s;
}
```

▪ 右に示すのは、文字列 *s* が回文（先頭から読んでも末尾から読んでも同じ文字列）であれば **true** を、そうでなければ **false** を返却する関数である。

```
bool is_palindrome(const char s[])
{
    int len = 0;
    while (s[len])
        len++;
    for (int i = 0; i < (76) ; i++)
        if (s[i] != (77) )
            return (78) ;
    return (79) ;
}
```

▪ 右に示すのは、文字列 *s* を二重引用符 " で囲んで表示する関数である。

たとえば、*s* に受け取った文字列が "ABC" であれば、「"ABC"」と表示する。

```
void put_str(char *s)
{
    cout << "  (80)  ";
    while (*  (81)  ) {
        cout << *  (81)  ;
          (82)  ++;
    }
    cout << "  (80)  ";
}
```

▪ 右に示すのは、int型整数値 *x* を文字列表現に変換したものを *s* に格納する関数である。なお、返却するのは *s* そのものである。

たとえば、*x* が 1573 であれば *s* に "1573" を格納し、*x* が -328 であれば *s* に "-328" を格納する。

```
char *itoa(int x, char* s)
{
    int len = 0;
    unsigned nx = x >= 0 ? x : -x;

    do {
        s[  (83)  ] = '0' + nx %  (84)  ;
        nx /=  (85)  ;
    } while (nx > 0);
    if (x < 0)
        s[len++] =  (86)  ;
    s[len] =  (87)  ;

    for (int i =  (88)  ; i >= 0; i--) {
        char temp = s[i];
        s[i] =  (89)  ;
          (90)  = temp;
    }
    return s;
}
```

▪ 右に示すのは、文字列 *s2* に含まれないすべての文字を、文字列 *s1* から取り除く関数である。

たとえば、文字列 *s1* が "ABCKCAE" で文字列 *s2* が "ACE" であれば、文字列 *s1* を "ACCAE" に更新する。

```
void strinstr(char s1[], const char s2[])
{
    int i, j, idx =  (91)  ;
    for (i = 0;  (92)  ; i++) {
        for (j = 0;  (93)  ; j++)
            if (s1[i] == s2[j]) {
                s1[  (94)  ] = s1[i];
            break;
        }
    }
    s1[  (95)  ] = '\0';
}
```

▪ 以下に示すのは、文字列 *s2* に含まれるいずれかの文字のうち、文字列 *s1* に含まれる最も先頭に位置する文字を探す関数である。文字を発見した場合は、その文字へのポインタを返し、発見できなかった場合は空ポインタを返却する。

```
const char *str_pbrk(const char* s1, const char* s2)
{
    for ( ;  (96)  ; s1++) {
        const char* t = s2;
        for ( ;  (97)  ; t++)
            if (*t == *s1)
                return s1;
    }
    return  (98)  ;
}
```