演習問題の解答

第1章

演習 1–1

&*n** … **&n* が指しているオブジェクトであり、*n* そのものです。

&**p* … *p* が指しているオブジェクトへのポインタであり、*p* そのものです。

以下のプログラムで確認できます。

演習 1-1 chap01/ex0101.c

```
/* 演習1-1の解答例 */

#include <stdio.h>

int main(void)
{
    int n  = 100;
    int *p = &n;

    printf("  nの値＝%d\n", n);      /*   nはint型 */
    printf("*&nの値＝%d\n", *&n);    /* *&nはint型 */
    printf("  pの値＝%p\n", p);      /*   pはint *型 */
    printf("&*pの値＝%p\n", &*p);    /* &*pはint *型 */

    printf("sizeof(n)  ＝%u\n", (unsigned)sizeof(n));
    printf("sizeof(*&n)＝%u\n", (unsigned)sizeof(*&n));
    printf("sizeof(p)  ＝%u\n", (unsigned)sizeof(p));
    printf("sizeof(&*p)＝%u\n", (unsigned)sizeof(&*p));

    return 0;
}
```

実行結果一例

```
  nの値＝100
*&nの値＝100
  pの値＝312
&*pの値＝312
sizeof(n)  ＝2
sizeof(*&n)＝2
sizeof(p)  ＝4
sizeof(&*p)＝4
```

演習 1–2

各式の意味は、右ページの表に示しています（処理系によって表示される値が異なるため、プログラムの実行例は省略します）。

演習 1-2 chap01/ex0102.c

```
/* 演習1-2の解答例 */

#include <stdio.h>

int main(void)
{
    int n;
    int *p;

    printf("sizeof*p           = %u\n", (unsigned)sizeof*p);
    printf("sizeof&n           = %u\n", (unsigned)sizeof&n);
    printf("sizeof-1           = %u\n", (unsigned)sizeof-1);
    printf("sizeof(unsigned)-1 = %u\n", (unsigned)sizeof(unsigned)-1);
    printf("sizeof(double)-1   = %u\n", (unsigned)sizeof(double)-1);
    printf("sizeof((double)-1) = %u\n", (unsigned)sizeof((double)-1));
    printf("sizeof n+2         = %u\n", (unsigned)sizeof n+2);
    printf("sizeof(n+2)        = %u\n", (unsigned)sizeof(n+2));
    printf("sizeof(n+2.0)      = %u\n", (unsigned)sizeof(n+2.0));

    return 0;
}
```

▶ “**sizeof** *n*+2” における **sizeof** と *n*+2 のあいだの空白は削除できません。もし削除すると、コンパイラが “sizeofn” を一つの単語とみなすからです。そのような名前の識別子が宣言されていない旨のエラーが発生します。

sizeof*p	*p の大きさですから sizeof(int) と同じ値となります。
sizeof&n	&n の大きさですから sizeof(int *) と同じ値となります。
sizeof-1	-1 は int 型の整数定数ですから sizeof(int) と同じ値となります。
sizeof(unsigned)-1	unsigned 型の大きさから 1 を引いた値となります。
sizeof(double)-1	double 型の大きさから 1 を引いた値となります。
sizeof((double)-1)	整数 -1 を double 型にキャストした式の大きさのことであり、sizeof(double) と同じ値となります。
sizeof n+2	sizeof(int) に 2 を加えた値となります。
sizeof(n+2)	int+int の結果は int ですから、sizeof(int) と同じ値となります。
sizeof(n+2.0)	int+double の結果は double ですから、sizeof(double) と同じ値となります。

演習 1–3

ポインタであろうとなかろうと、二値を交換するには、同じ型の作業用変数をもう一つ用意して、やりくりするのが定石です。

演習 1-3 chap01/ex0103.c

```
/* 演習1-3の解答例 */
#include <stdio.h>

int main(void)
{
    int x = 123,  y = 456;
    int *p1 = &x;            /* p1はxを指す */
    int *p2 = &y;            /* p2はyを指す */
    int *temp;

    temp = p1;
    p1 = p2;
    p2 = temp;

    printf("*p1の値＝%d\n", *p1);        /* p1が指すyの値を表示 */
    printf("*p2の値＝%d\n", *p2);        /* p2が指すxの値を表示 */

    return 0;
}
```

交換

実行結果

```
*p1の値＝456
*p2の値＝123
```

演習 1–4

Fortran での“べき乗”を求める演算子 ** は、C 言語には存在しません。単に 5 と *p とを掛けるだけですから、5 * 55 で 275 と表示されます。

演習 1-4 chap01/ex0104.c

```
/* 演習1-4の解答例 */
#include <stdio.h>

int main(void)
{
    int x = 55;
    int *p = &x;
    printf("%d\n", 5**p);

    return 0;
}
```

実行結果

```
275
```

演習 1–5

関数 *summ_diff* の３番目の引数 *wa* と４番目の引数 *sa* がポインタです。これらの引数には、**main** 関数から渡される **&***sum* と **&***diff* が代入されます。そのため、*wa* は *sum* を指し、*sa* は *diff* を指すことになります。求めた和と差を ***wa* と ***sa* に代入すると、**main** 関数の *sum* と *diff* の値が更新されます。

演習 1-5 chap01/ex0105.c

```
/* 演習1-5の解答例 */
#include <stdio.h>

/*--- xとyの和を*waに差を*saに格納 ---*/
void sum_diff(int x, int y, int *wa, int *sa)
{
    *wa = x + y;                          /* 和 */
    *sa = (x > y) ? x - y : y - x;        /* 差 */
}

int main(void)
{
    int n1, n2;
    int sum, diff;      /* 和と差 */

    printf("整数n1：");   scanf("%d", &n1);
    printf("整数n2：");   scanf("%d", &n2);

    sum_diff(n1, n2, &sum, &diff);

    printf("n1とn2の和＝%d\n", sum);        /* sumの値を表示 */
    printf("n1とn2の差＝%d\n", diff);       /* diffの値を表示 */

    return 0;
}
```

実行例

```
整数n1：54
整数n2：87
n1とn2の和＝141
n1とn2の差＝33
```

演習 1–6

関数 *sort3d* を理解しましょう。

1. ***x1* と ***x2* の値を比べます。もし左側の ***x1* が右側の ***x2* よりも大きければ、それらの値を交換します（大きいほうの値が ***x2* に入ります）。

2. ***x2* と ***x3* の値を比べます。もし左側の ***x2* が右側の ***x3* よりも大きければ、それらの値を交換します（大きいほうの値が ***x3* に入ります）。

   ここまでの２段階の手続きによって、最も大きい値が ***x3* に格納されます。

3. 最大値が ***x3* に格納されたわけですから、次に行うのは、残った二値 ***x1* と ***x2* の最大値を ***x2* に格納することです。これは、第２位を決定するための「敗者復活戦」です。**if** 文を実行すると、***x1* と ***x2* の大きいほうの値が ***x2* に格納されます。

   最大値が ***x3* に格納され、２番目に大きい値が ***x2* に格納されたわけですから、***x1* には当然最小値が格納されています。これでソートは完了です。

２値の値を交換する関数 *swapd* は、**List 1-14** の関数 *swap* と同じ構造です（仮引数の型が **int** * 型から **double** * 型に変更されているだけです）。

演習 1-6 chap01/ex0106.c

```
/* 演習1-6の解答例 */
#include <stdio.h>

/*--- *xと*yの値を交換 ---*/
void swapd(double *x, double *y)
{
    double temp = *x;
    *x = *y;
    *y = temp;
}

/*--- *x1≦*x2≦*x3となるようにソート ---*/
void sort3d(double *x1, double *x2, double *x3)
{
    if (*x1 > *x2) swapd(x1, x2); ←1
    if (*x2 > *x3) swapd(x2, x3); ←2
    if (*x1 > *x2) swapd(x1, x2); ←3
}

int main(void)
{
    double d1, d2, d3;

    printf("実数d1：");   scanf("%lf", &d1);
    printf("実数d2：");   scanf("%lf", &d2);
    printf("実数d3：");   scanf("%lf", &d3);

    sort3d(&d1, &d2, &d3);

    printf("d1≦d2≦d3となるようにソートしました。\n");
    printf("d1の値＝%.3f\n", d1);          /* d1の値を表示 */
    printf("d2の値＝%.3f\n", d2);          /* d2の値を表示 */
    printf("d3の値＝%.3f\n", d3);          /* d3の値を表示 */

    return 0;
}
```

実行例

```
実数d1：3.1416↵
実数d2：0.0↵
実数d2：2.5↵
d1≦d2≦d3となるようにソートしました。
d1の値＝0.000
d2の値＝2.500
d3の値＝3.142
```

第2章

演習 2–1

ポインタ *p* が a[2] を指します。配列 a の要素 a[0] ～ a[4] を指すポインタは、先頭から順に *p* - 2, *p* - 1, *p*, *p* + 1, *p* + 2 です。

演習 2-1 chap02/ex0201.c

```
/* 演習2-1の解答例 */
#include <stdio.h>

int main(void)
{
    int i;
    int a[5];           /* int型の配列 */
    int *p = &a[2];     /* pはa[2]を指す */

    for (i = 0; i < 5; i++)
        printf("&a[%d] = %p  p + (%2d) = %p\n", i, &a[i], i - 2, p + i - 2);

    return 0;
}
```

実行結果一例

```
&a[0] = 100  p + (-2) = 100
&a[1] = 102  p + (-1) = 102
&a[2] = 104  p + ( 0) = 104
&a[3] = 106  p + ( 1) = 106
&a[4] = 108  p + ( 2) = 108
```

演習 2–2

関数 *ary_cpy* は、配列 a と b を同時に走査します。**while** 文による繰返しの回数は、*no* 回です。

演習 2-2 chap02/ex0202.c

```
/* 演習2-2の解答例 */

#include <stdio.h>

/*--- 要素数nの配列bの全要素を配列aにコピー ---*/
void ary_cpy(int a[], const int b[], int no)
{
    while (no-- > 0)
        *a++ = *b++;
}

int main(void)
{
    int i, no;
    int x[5], y[5];
    int x_size = sizeof(x) / sizeof(x[0]);

    for (i = 0; i < x_size; i++) {
        printf("x[%d] : ", i);
        scanf("%d", &x[i]);
    }

    ary_cpy(y, x, x_size);

    printf("配列xの全要素を配列yにコピーしました。\n");
    for (i = 0; i < x_size; i++)
        printf("y[%d]＝%d\n", i, y[i]);

    return 0;
}
```

実行例

```
x[0] : 54
x[1] : 28
x[2] : 89
x[3] : 18
x[4] : 77
配列xの全要素を配列yにコピーしました。
y[0]＝54
y[1]＝28
y[2]＝89
y[3]＝18
y[4]＝77
```

▶ **List4-1**（p.116）のプログラムと、その解説が、本関数の理解の手助けとなります。

演習 2–3

三値のソートを行うのですから、演習 **1-6** と同じ要領です。

演習 2-3 chap02/ex0203.c

```
/* 演習2-3の解答例 */

#include <stdio.h>

#define swap(type, x, y)    do { type temp = x; x = y; y = temp; } while (0)

/*--- *x[0]≦*x[1]≦*x[2]となるようにソート ---*/
void sort_ptr3ary(int *x[])
{
    if (*x[0] > *x[1]) swap(int *, x[0], x[1]);
    if (*x[1] > *x[2]) swap(int *, x[1], x[2]);
    if (*x[0] > *x[1]) swap(int *, x[0], x[1]);
}

int main(void)
{
    int n1, n2, n3;
    int *p[3] = {&n1, &n2, &n3};

    printf("整数n1：");   scanf("%d", &n1);
    printf("整数n2：");   scanf("%d", &n2);
    printf("整数n3：");   scanf("%d", &n3);

    sort_ptr3ary(p);

    printf("ソートしました。\n");
    printf("*p[0]の値＝%d\n", *p[0]);       /* *p[0]の値を表示 */
    printf("*p[1]の値＝%d\n", *p[1]);       /* *p[1]の値を表示 */
    printf("*p[2]の値＝%d\n", *p[2]);       /* *p[2]の値を表示 */

    return 0;
}
```

実行例

```
整数n1：5↵
整数n2：8↵
整数n3：6↵
ソートしました。
*p[0]の値＝5
*p[1]の値＝6
*p[2]の値＝8
```

なお、任意の型の二値を交換する関数形式マクロ *swap* については『解きながら学ぶC言語』『新版 明解C言語 中級編』で学習いただけます。

第３章

演習 3–1

以下に示すプログラムからも、多次元配列の要素は、最も後ろ側の添字が優先的に変化する順に並んでいることが分かります。

演習 3-1 chap03/ex0301.c

```
/* 演習3-1の解答例 */

#include <stdio.h>

int main(void)
{
    int i, j, k;
    int b[3][2][4];

    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            for (k = 0; k < 4; k++)
                printf("&b[%d][%d][%d] = %p\n", i, j, k, &b[i][j][k]);

    return 0;
}
```

実行結果一例

```
&b[0][0][0] = 1000
&b[0][0][1] = 1002
&b[0][0][2] = 1004
&b[0][0][3] = 1006
&b[0][1][0] = 1008
&b[0][1][1] = 1010
… 以下省略 …
```

演習 3–2

詳しい解説は不要でしょう。

演習 3-2 chap03/ex0302.c

```
/* 演習3-2の解答例 */

#include <stdio.h>

int main(void)
{
    int x[3][2][4];

    printf("配列xは%d×%d×%dの３次元配列です。\n",
                (int)(sizeof(x)       / sizeof(x[0])),
                (int)(sizeof(x[0])    / sizeof(x[0][0])),
                (int)(sizeof(x[0][0]) / sizeof(x[0][0][0])));

    return 0;
}
```

実行結果

```
配列xは3×2×4の３次元配列です。
```

ちなみに、要素数を求める式に要素型を埋め込む方法を使うのであれば、要素数を表示する箇所は、以下のようになります。

```
printf("配列xは%d×%d×%dの３次元配列です。\n",
            (int)(sizeof(x)       / sizeof(int[2][4])),
            (int)(sizeof(x[0])    / sizeof(int[4])),
            (int)(sizeof(x[0][0]) / sizeof(int)));
```

sizeof（配列名）/ sizeof（要素型）を使うべきでないことが、この例からもはっきりしますね。

演習 3–3

関数 *fill_avalue* が受け取るのは 3 次元配列です。3 次元の要素数のみが可変であり、2 次元の要素数 2 と、1 次元の要素数 4 は定数です。

演習 3-3 chap03/ex0303.c

```
/* 演習3-3の解答例 */

#include <stdio.h>

/*--- n×2×4の配列の全構成要素にvを代入 ---*/
void fill_avalue(int a[][2][4], int n, int v)
{
    int i, j, k;

    for (i = 0; i < n; i++)
        for (j = 0; j < 2; j++)
            for (k = 0; k < 4; k++)
                a[i][j][k] = v;
}

int main(void)
{
    int i, j, k, no;
    int mx[3][2][4];

    printf("全構成要素に代入する値：");
    scanf("%d", &no);

    fill_avalue(mx, 3, no);      /* mxの全構成要素にnoを代入 */

    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            for (k = 0; k < 4; k++)
                printf("mx[%d][%d][%d] = %3d\n", i, j, k, mx[i][j][k]);

    return 0;
}
```

実行例

```
全構成要素に代入する値：15
mx[0][0][0] = 15
mx[0][0][1] = 15
mx[0][0][2] = 15
mx[0][0][3] = 15
… 中略 …
mx[2][1][3] = 15
```

演習 3–4

配列名 a は &a[0] と解釈されるため、それに間接演算子を適用した *a は a[0] のことであり、両者の大きさは同じです。このことは、配列の次元数によらず成立します。

演習 3-4 chap03/ex0304.c

```
/* 演習3-4の解答例 */

#include <stdio.h>

int main(void)
{
    int a[3][2][4];

    printf("sizeof(*a)          = %u\n", (unsigned)sizeof(*a));
    printf("sizeof(a[0])        = %u\n", (unsigned)sizeof(a[0]));
    printf("sizeof(a[0][0])     = %u\n", (unsigned)sizeof(a[0][0]));
    printf("sizeof(a[0][0][0]) = %u\n", (unsigned)sizeof(a[0][0][0]));

    return 0;
}
```

実行結果一例

```
sizeof(*a)          = 16
sizeof(a[0])        = 16
sizeof(a[0][0])     = 8
sizeof(a[0][0][0]) = 2
```

第4章

演習 4–1

先頭の文字がナル文字であれば、空の文字列ということになります。

演習 4-1 chap04/ex0401.c

```
/* 演習4-1の解答例 */
#include <stdio.h>

int main(void)
{
    char str[4];

    str[0] = '\0';
    str[1] = 'A';
    str[2] = 'B';
    str[3] = 'C';

    printf("配列strに文字列\"%s\"が格納されています。\n", str);

    return 0;
}
```

実行結果

```
配列strに文字列"ABC"が格納されています。
```

演習 4–2

List4-7 のプログラムにおける文字列の読込みを、以下のように変更した上で、プログラムを実行すると、読み込んだ文字列は正しく *str* に格納されます。

```
scanf("%s", &str);
```

配列名に対してアドレス演算子を適用した式は、配列全体へのポインタと解釈されます。すなわち、**&***str* は、要素型が **char** で要素数が 15 である配列へのポインタです。このポインタは、配列の先頭要素へのポインタ **&***str*[0] と型は異なるものの、値は同じですから、見かけ上はうまくいくのです。

演習 4–3

ポインタ *q* に *p* が代入された結果、*q* と *p* は同じ値となります。そのため、ポインタ *p* と *q* は同一の文字列リテラル "ABCD" の先頭文字 'A' を指すことになります。

文字列のコピーを行っているわけではないことに注意しましょう。

演習 4–4

ポインタ *ptr* にアドレス演算子を適用した **&***ptr* を渡す関数呼出し

```
scanf("%s", &ptr);
```

によって、*ptr* が指す配列 *str* の領域に文字列が読み込まれることはありません。

scanf 関数は、ポインタである *ptr* が格納されている領域に読み込んだ文字列を格納します。たとえば、ポインタ *ptr* が 100 ～ 101 番地に格納されているとして、ナル文字を含め 10 文字を読み込むと、100 ～ 109 番地が書きかえられることになります。

これは非常に危険な行為です。このような間違いを犯さないように注意しましょう。

演習 4–5

本文で学習した、文字列の長さを求める関数や、文字列の表示を行う関数と同じ要領で実現できます。

演習 4-5 chap04/ex0405.c

```
/*--- 文字列sの中に含まれる文字cの個数を調べる ---*/
int str_chnum(const char *s, int c)
{
    int count = 0;

    while (*s)
        if (*s++ == c)
            count++;

    return count;
}
```

演習 4–6

前問とほぼ同様です。

演習 4-6 chap04/ex0406.c

```
/*--- 文字列sの中に含まれる数字文字の個数を調べる ---*/
int str_dignum(const char *s)
{
    int count = 0;

    while (*s) {
        if (*s >= '0' && *s <= '9')
            count++;
        *s++;
    }

    return count;
}
```

▶ 数字文字かどうかを判定する網かけ部は、以下のようにも実現できます。

```
if (isdigit(*s))
```

なお、その場合は、<ctype.h> ヘッダのインクルードが必要です。

演習 4–7

二つの文字列を入れかえるためには、配列中の全要素を交換しなければなりません。

演習 4-7 chap04/ex0407.c

```
/*--- 二つの文字列s1とs2を交換する ---*/
void swap_str(char s1[], char s2[])
{
    char *temp;
    while (*s1 && *s2) {    /* 短いほうの末尾まで文字列を交換 */
        char t = *s1; *s1++ = *s2; *s2++ = t;
    }
    if (*s1) {                          /* s1のほうが長ければ */
        temp = s1;
        while (*s1) { *s2++ = *s1++; }  /* s1の残りをs2にコピー */
        *temp = *s2 = '\0';
    } else if (*s2) {                   /* s2のほうが長ければ */
        temp = s2;
        while (*s2) { *s1++ = *s2++; }  /* s2の残りをs1にコピー */
        *temp = *s1 = '\0';
    } else {
        *s1 = *s2 = '\0';
    }
}
```

第5章

演習 5–1

本プログラムは、日本語あるいは英語の単語を表示して、それに対応する英語あるいは日本語の単語を四つの選択肢から選ばせる形式です。問題の言語（日本語なのか英語なのか）、問題の単語、選択肢などはすべて乱数を発生させて決定します。

▶ いろいろなテクニックを使っていますので、少々難しいかもしれません。でも、これくらいのプログラムが理解できて作れるように学習が進むといいですね。

演習 5-1 chap05/ex0501.c

```
/* 演習5-1の解答例 */

#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#define QNO     12      /* 単語の数 */
#define CNO     4       /* 選択肢の数 */

#define swap(type, x, y) do { type t = x; x = y; y = t; } while (0)

/*--- 日本語 ---*/
char *jptr[] = {
    "動物", "車",   "花",   "家",   "机",   "本",
    "椅子", "父",   "母",   "愛",   "平和", "雑誌",
};

/*--- 英語 ---*/
char *eptr[] = {
    "animal", "car",    "flower", "house", "desk",  "book",
    "chair",  "father", "mother", "love",  "peace", "magazine",
};

/*--- 選択肢を表示 ---*/
void print_cand(const int c[], int sw)
{
    int i;
    for (i = 0; i < CNO; i++)
        printf("(%d) %s  ", i, sw ? jptr[c[i]] : eptr[c[i]]);
    printf(" : ");
}

/*--- 選択肢を作成し正解の添字を返す ---*/
int make_cand(int c[], int n)
{
    int i, j, x;

    c[0] = n;                           /* 先頭要素に正解を入れる */
    for (i = 1; i < CNO; i++) {
        do {
            x = rand() % QNO;
            for (j = 0; j < i; j++)
                if (c[j] == x)
                    break;
        } while (i != j);
        c[i] = x;
    }

    j = rand() % CNO;
    if (j != 0)
        swap(int, c[0], c[j]);          /* 正解を移動 */

    return j;
}
```

実行例

```
bookはどれですか？
(0) 本  (1) 平和  (2) 家  (3) 動物 : 0↵
正解です。
もう一度？ 0-いいえ／1-はい : 1↵
家はどれですか？
(0) love  (1) house  (2) car  (3) desk : 1↵
正解です。
もう一度？ 0-いいえ／1-はい : 0↵
```

```
int main(void)
{
    int nq, pq;      /* 問題番号・前回の問題番号 */
    int na;          /* 正解の番号 */
    int sw;          /* 問題の言語（0：日本語／1：英語）*/
    int retry;       /* 再挑戦するか？ */
    int cand[CNO];   /* 選択肢の番号 */

    srand(time(NULL));  /* 乱数の種を初期化 */

    pq = QNO;           /* 前回の問題番号（存在しない番号）*/

    do {
        int no;

        do {                        /* 同じ単語を連続して出題しない */
            nq = rand() % QNO;
        } while (nq == pq);

        na = make_cand(cand, nq);   /* 選択肢を作成 */
        sw = rand() % 2;

        printf("%sはどれですか？\n", sw ? eptr[nq] : jptr[nq]);

        do {
            print_cand(cand, sw);   /* 選択肢を表示 */
            scanf("%d", &no);
            if (no != na)
                puts("違います。");
        } while (no != na);
        puts("正解です。");

        pq = nq;

        printf("もう一度？ 0-いいえ／1-はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return 0;
}
```

演習 5–2

プログラムの構造は **List5-9** と同じです。異なるのは、変数 *argc* のデクリメントのタイミングと、変数 *i* と *argv* のインクリメントのタイミングです。

演習 5-2 chap05/ex0502.c

```
/* 演習5-2の解答例 */
#include <stdio.h>

int main(int argc, char **argv)
{
    int i = 0;

    while (--argc > 0)
        printf("argv[%d] = \"%s\"\n", ++i, *++argv);

    return 0;
}
```

起動・実行例

```
>ex0502 Sort BinTree
argv[1] = "Sort"
argv[2] = "BinTree"
```

演習 5–3

argv が指す文字列を ***strtod*** 関数によって浮動小数点数に変換して加算します。なお、先頭の文字列に格納されているプログラム名をスキップする点は、前問と同じです。

演習 5-3 chap05/sum.c

```
/* 演習5-3の解答例 */

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    char   str[100];
    char   *ptr = str;
    double sum = 0.0;

    while (--argc > 0) {
        double x = strtod(*++argv, &ptr);
        if (errno != ERANGE && ptr != str)
            sum += x;
    }
    printf("%f\n", sum);

    return 0;
}
```

実行例

```
>sum 15 3.14 1.35E1↵
31.640000
```

演習 5–4

演習 5-4 chap05/detab.c

```
/* 演習5-4の解答例 */

#include <stdio.h>
#include <stdlib.h>

/*--- srcからの入力をタブを展開してdstへ出力 ---*/
void detab(FILE *src, FILE *dst, int width)
{
    int ch, pos = 1;

    while ((ch = fgetc(src)) != EOF) {
        int num;
        switch (ch) {
         case '\t' : num = width - (pos - 1) % width;
                     for ( ; num > 0; num--, pos++) fputc(' ', dst);
                     break;
         case '\n' : fputc(ch, dst); pos=1; break;
         default :   fputc(ch, dst); pos++; break;
        }
    }
}

int main(int argc, char *argv[])
{
    int  width = 8;
    FILE *fp;

    if (argc < 2)
        detab(stdin, stdout, width);          /* 標準入力 → 標準出力 */
    else {
        while (--argc > 0) {
            if (**(++argv) == '-') {
                if (***(*argv) == 't')
                    width = atoi(+++argv);
                else {
                    fputs("パラメータが不正です。\n", stderr);
                    return 1;
                }
            } else if ((fp = fopen(*argv, "r")) == NULL) {
                fprintf(stderr, "\"%s\"はオープンできません。\n", *argv);
                return 1;
            } else {
                detab(fp, stdout, width);     /* ストリームfp → 標準出力 */
                fclose(fp);
            }
```

```
        }
    }
    return 0;
}
```

演習 5–5

演習 5-5 chap05/entab.c

```
/* 演習5-5の解答 */

#include <stdio.h>
#include <stdlib.h>

/*--- srcからの入力をタブ化してdstへ出力 ---*/
void entab(FILE *src, FILE *dst, int width)
{
    int ch, count = 0, ntab = 0, pos = 1;

    for ( ; (ch = fgetc(src)) != EOF; pos++)
        if (ch == ' ') {
            if (pos % width != 0)
                count++;
            else {
                count = 0;  ntab++;
            }
        } else {
            for ( ; ntab > 0; ntab--)
                fputc('\t', dst);
            if (ch == '\t')
                count = 0;
            else
                for ( ; count > 0; count--)
                    fputc(' ', dst);
            fputc(ch, dst);
            if (ch == '\n')
                pos = 0;
            else if (ch == '\t')
                pos += width - (pos - 1) % width - 1;
        }
}

int main(int argc, char *argv[])
{
    int  width = 8;
    FILE *fp;

    if (argc < 2)
        entab(stdin, stdout, width);          /* 標準入力 → 標準出力 */
    else {
        while (--argc > 0) {
            if (**(++argv) == '-') {
                if (*++(*argv) == 't')
                    width = atoi(++*argv);
                else {
                    fputs("パラメータが不正です。\n", stderr);
                    return 1;
                }
            } else if ((fp = fopen(*argv, "r")) == NULL) {
                fprintf(stderr, "\"%s\"はオープンできません。\n", *argv);
                return 1;
            } else {
                entab(fp, stdout, width);    /* ストリームfp → 標準出力 */
                fclose(fp);
            }
        }
    }
    return 0;
}
```

第6章

演習 6–1

&z.a は struct xy * 型、&z.a.x は int * 型、&z.a.y は double * 型、&z.b は int * 型です。

演習 6–2

a が指すオブジェクトのメンバ x の値が、b が指すオブジェクトのメンバ x の値よりも大きいときに、*a と *b を交換します。

演習 6-2 chap06/ex0602.c

```
/*--- メンバxの昇順となるようにa,bを並べかえる ---*/
void sortXYZ(struct xyz *a, struct xyz *b)
{
    if (a->x > b->x) {
        struct xyz temp = *a;
        *a = *b;
        *b = temp;
    }
}
```

演習 6–3

メンバ name は文字列ですから、アドレス演算子 & は不要です。

演習 6-3 chap06/ex0603.c

```
/*--- pが指すMember型オブジェクトの各メンバに値を読み込む ---*/
void scanMember(Member *p)
{
    printf("会員番号：");   scanf("%d", &p->no);        /* &は必要 */
    printf("氏　　名：");   scanf("%s", p->name);       /* &は不要 */
}
```

第7章

演習 7–1

List 7-1 のプログラムから *p = 15; を取り除いてプログラムを実行すると、『*p = 0』と表示されます。これで、*calloc* 関数によって確保される領域の全ビットが 0 で埋められることが確認できます。

演習 7–2

新たに確保された部分にのみ 0 を埋める必要がありますので、変更前の大きさを受け取る仕様となっています。

演習 7-2 chap07/ex0702.c

```
/*--- ptrの指すold_sizeバイトの確保済み領域をsizeバイトに変更
      ※ 新たに確保した領域の全ビットを0で埋め尽くす--- */
void *recalloc(void *ptr, size_t size, size_t old_size)
{
    void *tmp;

    if (size == 0)
        return NULL;

    tmp = realloc(ptr, size);
    if (tmp != NULL && size > old_size)
        memset((char *)tmp + old_size, 0, size - old_size);

    return tmp;
}
```

演習 7-3 chap07/ex0703.c

```
/* 演習7-3の解答例 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*--- 文字列sの複製を作る ---*/
char *str_dup(const char *s)
{
    char *p = malloc(strlen(s) + 1);
    return (p == NULL) ? NULL : strcpy(p, s);
}

int main(void)
{
    char s[128];
    char *p;

    printf("文字列sを入力してください：");
    scanf("%s", s);

    if ((p = str_dup(s)) != NULL) {      /* 文字列を複製 */
        printf("その文字列のクローンpを作りました。\n");
        printf("s = \"%s\"\n", s);
        printf("p = \"%s\"\n", p);
        free(p);                         /* 記憶域を解放 */
    }
}
```

演習 7–3

ここでは、***sprintf*** 関数を用いた解答例を示します。

演習 7-4 chap07/ex0704.c

```
/* 演習7-4の解答例 */

#include <stdio.h>
#include <stdlib.h>

#define LENGTH  10      /* 文字列の長さ */

int main(void)
{
    int  num;           /* 文字列の個数 */
    char (*p)[LENGTH];  /* 文字数は定数10 */

    printf("文字列は何個：");
    scanf("%d", &num);

    p = malloc(num * LENGTH);

    if (p == NULL)
        puts("記憶域の確保に失敗しました。");
    else {
        int  i;
        char temp[100];

        for (i = 0; i < num; i++) {          /* 文字列を読み込む */
            printf("p[%d] : ", i);
            scanf("%s", temp);
            sprintf(p[i], "%.9s", temp);
        }

        for (i = 0; i < num; i++)            /* 文字列を表示 */
            printf("p[%d] = %s\n", i, p[i]);

        free(p);                             /* 記憶域を解放 */
    }

    return 0;
}
```

実行結果

```
文字列は何個：5
p[0] : 123456789012
p[1] : 12345678901
p[2] : 1234567890
p[3] : 123456789
p[4] : 12345678
p[0] = 123456789
p[1] = 123456789
p[2] = 123456789
p[3] = 123456789
p[4] = 12345678
```

演習 7–4

argv の複製を作るのですから、**main** 関数で宣言している複製の *pt* は、*argv* と同じ型である **char** ** 型となります。

複製を作成するのが、関数 *dup_argv* です。**main** 関数からの呼出しでは、“**char** へのポインタへのポインタ”である *pt* の値を変更してもらう必要がありますので、アドレス演算子 **&** を適用して、《“**char** へのポインタへのポインタ”へのポインタ》として渡します。

この *pt* へのポインタを、仮引数 *ptr* に受け取った関数 *dup_argv* では、まず *argv* が指している **char** へのポインタの配列（各コマンドライン引数文字列の先頭文字を指すポインタの配列）の複製を行います。ここで、配列の要素数は *argc* ではなくて、*argc* + 1 であることに注意しましょう（番兵として利用できる *argv*[*argc*] には、空ポインタ **NULL** が格納されているのでしたね）。

その配列の全要素に **NULL** を代入し終わったら、コマンドラインの各文字列と同じ長さ

演習 7-5 chap07/ex0705.c

```
/* 演習7-5の解答例 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*--- argvの複製を作る ---*/
int dup_argv(char ***ptr, int argc, char **argv)
{
    int i;

    if ((*ptr = calloc(argc + 1, sizeof(char *))) == NULL)
        return 0;

    for (i = 0; i < argc + 1; i++)
        (*ptr)[i] = NULL;

    for (i = 0; i < argc; i++) {
        if (((*ptr)[i] = malloc(strlen(argv[i]) + 1)) == NULL)
            return 0;
        strcpy((*ptr)[i], argv[i]);
    }
    return 1;
}

/*--- argvが指す文字列の配列を表示 ---*/
void print_argv(int argc, char **argv)
{
    int i = 0;

    while (argc-- > 0)
        printf("argv[%d] = \"%s\"\n", i++, *argv++);
}

int main(int argc, char **argv)
{
    int  i;
    char **pt;

    if (!dup_argv(&pt, argc, argv))
        puts("記憶域の確保に失敗しました。");
    else
        print_argv(argc, pt);

    if (pt != NULL) {
        for (i = 0; i < argc + 1; i++)
            free(pt[i]);                    /* 記憶域を解放 */
        free(pt);
    }

    return 0;
}
```

起動・実行例

```
>ex0705 Sort BinTree↵
argv[0] = "ex0705"
argv[1] = "Sort"
argv[2] = "BinTree"
```

の分だけ記憶域を確保してコピーします。なお、*argv*[*argc*] には NULL が代入済みであり、文字列をコピーする必要はありませんから、この複製作業は、*argv*[0] から *argv*[*argc* - 1] までとなります。

関数 *print_argv* は、*argv* が指す文字列の配列を順に表示する関数です。このプログラムでは、複製した *pt* を渡して表示していますが、オリジナルである *argv* を渡して表示することもできます。

第８章

演習 8–1

関数 *daikei* が台形公式によって積分を行う関数です。

演習 8-1 chap08/ex0801.c

```
/* 演習8-1の解答例 */

#include <stdio.h>

/*--- 関数f(x) ---*/
double f(double x)
{
    return x * x;
}

/*--- 関数g(x) ---*/
double g(double x)
{
    return (x * x * x) + (x * x);
}

/*--- 台形の面積を求める ---*/
double trapezoid(double w1, double w2, double h)
{
    return (w1 + w2) * h / 2.0;
}

/*--- fpが指す関数をx1からx2までn分割で台形公式を用いて積分 ---*/
double daikei(double x1, double x2, int n, double fp(double))
{
    int    i;
    double s = 0.0;
    double step = (x2 - x1) / n;

    for (i = 0; i < n; i++)
        s += trapezoid(fp(x1 + step * i), fp(x1 + step * (i + 1)), step);
    return s;
}

int main(void)
{
    int    n;
    double x1, x2;

    printf("開　始：");   scanf("%lf", &x1);
    printf("終　了：");   scanf("%lf", &x2);
    printf("分割数：");   scanf("%d",  &n);

    printf("関数fの積分値＝%.4f\n", daikei(x1, x2, n, f));

    printf("関数gの積分値＝%.4f\n", daikei(x1, x2, n, g));

    return 0;
}
```

実行例

```
開　始：1.0↵
終　了：5.0↵
分割数：100↵
関数fの積分値＝41.3344
関数gの積分値＝197.3440
```

積分の対象となる関数へのポインタを受け取るのは第４引数です。本プログラムでは、関数 *f* へのポインタと関数 *g* へのポインタを **main** 関数から受け取っています。

▶ 関数 *trapezoid* は、台形の面積を求める関数です。

演習 8–2

関数 *sort_2dstr* が 2 次元配列による文字列の配列をソートする関数で、関数 *sort_pvstr* がポインタの配列による文字列の配列をソートする関数です。

演習 8-2 chap08/ex0802.c

```
/* 演習8-2の解答例 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*--- 文字列の配列（n1×n2の２次元配列）を昇順にソート ---*/
void sort_2dstr(char *p, int n1, int n2)
{
    qsort(p, n1, n2, (int(*)(const void *, const void *))strcmp);
}

/*--- xおよびyが指す文字列の比較関数 ---*/
static int pstrcmp(const void *x, const void *y)
{
    return strcmp(*(const char **)x, *(const char **)y);
}

/*--- 文字列を指すポインタの配列pを昇順にソート ---*/
void sort_pvstr(char *p[], int n)
{
    qsort(p, n, sizeof(char *), pstrcmp);
}

int main(void)
{
    int  i;
    char s[][7] = {"LISP", "C", "Ada", "Pascal"};
    char *p[]   = {"LISP", "C", "Ada", "Pascal"};

    sort_2dstr(&s[0][0], 4, 7);

    sort_pvstr(p, 4);

    puts("昇順にソートしました。");

    for (i = 0; i < 4; i++)
        printf("s[%d] = %s\n", i, s[i]);

    for (i = 0; i < 4; i++)
        printf("p[%d] = %s\n", i, p[i]);

    return 0;
}
```

実行結果

```
昇順にソートしました。
s[0] = Ada
s[1] = C
s[2] = LISP
s[3] = Pascal
p[0] = Ada
p[1] = C
p[2] = LISP
p[3] = Pascal
```

関数 *sort_2dstr* は、行数と列数を可変とするために、**List 3-6** のプログラムと同じ要領で配列を受け取っています（2 次元配列を 1 次元配列とみなします）。

関数 *sort_pvstr* でのソートにおける比較関数が、関数 *pstrcmp* です。ポインタが指す先の文字列を比較する必要があるため、複雑な構造となっています。