第５章

記憶力トレーニング

本章では、記憶力を鍛えるための《単純記憶トレーニング》や《プラスワントレーニング》などのプログラムを作成します。

この章で学ぶおもなこと

- 整数型の表現範囲
- 処理系に依存しない英字の取扱い
- 記号文字による棒グラフ表示（横方向と縦方向）
- 文字列の比較
- 隣接する文字列リテラル
- 配列要素の循環的利用
- 記憶域の動的な確保と解放

- size_t 型
- calloc 関数
- free 関数
- malloc 関数
- strcmp 関数
- strncmp 関数

5-1 単純記憶トレーニング

本章では、記憶力トレーニングソフトの作成を通じて、配列や文字列の活用法などを学習します。最初に作るのは、瞬間的に表示される数値や文字を記憶するソフトです。

4 桁の数値を記憶するトレーニング

List 5-1 のプログラムを実行しましょう。一瞬（0.5 秒）だけ 4 桁の数値が表示されます。瞬時に記憶して、その数値をキーボードから打ち込みます。このトレーニングを 10 回行うと、正しく解答できた回数と所要時間が表示されます。

整数型の表現範囲

前章までに学習した技術だけを使っていますので、プログラムの理解は容易なはずです。そこで、記憶すべき数値の“桁数”を増やすことにしましょう。プログラムの変更は簡単であるように感じられるかもしれませんが、実はそうではありません。

というのも、**Table 5-1** に示すように、整数型が有限の値しか表せないからです。

▶ ここに示しているのは、**最低限の値**です。処理系によっては、これより広い範囲の数値を表現できます。

Table 5-1 整数型の表現範囲

型	少なくとも表現できる値の範囲
`char`	0 ～ 255 または -127 ～ 127
`signed char`	-127 ～ 127
`signed short int`	-32,767 ～ 32,767
`signed int`	-32,767 ～ 32,767
`signed long int`	-2,147,483,647 ～ 2,147,483,647
`unsigned char`	0 ～ 255
`unsigned short int`	0 ～ 65,535
`unsigned int`	0 ～ 65,535
`unsigned long int`	0 ～ 4,294,967,295

出題する数値を 5 桁以上にするためには、変数 *x* や *no* を `int` 型から `long` 型に変更するだけでよさそうです。

しかし、`int` 型で 32,767 までしか表現できない処理系では、***rand*** 関数が返す値は（その返却値型が `int` 型であるため）最大でも 32,767 であり、たとえ *x* や *no* を `long` 型にしても 5 桁を超える数値の出題は**できません**。

▶ `int` 型で表現できる最大値と、***rand*** 関数が生成する乱数の最大値が一致する保証がないことにも注意しなければなりません。たとえば Visual C++ では、`int` 型は 32 ビットであって、2 の補

List 5-1 chap05/kiokud1.c

```
/* 単純記憶トレーニング（４桁の数値を記憶）*/

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

#define MAX_STAGE   10                          /* ステージ数 */

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t c1 = clock(), c2;

    do {
        if ((c2 = clock()) == (clock_t)-1)  /* エラー */
            return (0);
    } while (1000.0 * (c2 - c1) / CLOCKS_PER_SEC < x);
    return (1);
}

int main(void)
{
    int     stage;
    int     success = 0;                        /* 正解数 */
    clock_t start, end;                         /* 開始時刻・終了時刻 */

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

    printf("４桁の数値を記憶しましょう。\n");

    start = clock();
    for (stage = 0; stage < MAX_STAGE; stage++) {
        int x;                                  /* 読み込んだ値 */
        int no = rand() % 9000 + 1000;          /* 記憶すべき数値 */

        printf("%d", no);
        fflush(stdout);
        sleep(500); /* 問題提示は0.5秒だけ */

        printf("\r入力せよ：");
        scanf("%d", &x);

        if (x != no)
            printf("\a間違いです。\n");
        else {
            printf("正解です。\n");
            success++;
        }
    }
    end = clock();

    printf("%d回中%d回成功しました。\n", MAX_STAGE, success);
    printf("%.1f秒でした。\n", (double)(end - start) / CLOCKS_PER_SEC);

    return (0);
}
```

実行例

```
４桁の数値を記憶しましょう。
1397      …  0.5秒で消えます。
入力せよ：1397↵
正解です。
2468      …  0.5秒で消えます。
入力せよ：2486↵
♪間違いです。

… 中略 …

10回中8回成功しました。
9.2秒でした。
```

数形式で負数を表現するため、その表現範囲は -2,147,483,648 ～ 2,147,483,647 です。一方、*rand* 関数が生成する乱数の最大値である **RAND_MAX** は 32,767 です（**int** 型が 16 ビットであった古いバージョンとの互換性を保つための仕様です）。

Visual C++ では、変数 *x* や *no* が **int** 型であっても **long** 型であっても、32,767 以上の乱数を生成することはできないのです。

任意の桁の数値を記憶するトレーニング

記憶すべき数値を 3 桁から 20 桁までの間で、自由に設定できるように拡張したプログラムを **List 5-2** に示します。

List 5-2 chap05/kiokud2.c

```
/* 単純記憶トレーニング（数値を記憶：レベル＝桁数の設定あり）*/

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

#define MAX_STAGE   10                  /* ステージ数 */
#define LEVEL_MIN    3                  /* 最小レベル（桁数）*/
#define LEVEL_MAX   20                  /* 最大レベル（桁数）*/

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t c1 = clock(), c2;

    do {
        if ((c2 = clock()) == (clock_t)-1)  /* エラー */
            return (0);
    } while (1000.0 * (c2 - c1) / CLOCKS_PER_SEC < x);
    return (1);
}

int main(void)
{
    int     i, stage;
    int     level;          /* レベル（数値の桁数）*/
    int     success = 0;    /* 正解数 */
    clock_t start, end;     /* 開始時刻・終了時刻 */

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

    printf("数値記憶トレーニング\n");

    do {
        printf("挑戦するレベル（%d〜%d）：", LEVEL_MIN, LEVEL_MAX);
        scanf("%d", &level);
    } while (level < LEVEL_MIN || level > LEVEL_MAX);

    printf("%d桁の数値を記憶しましょう。\n", level);

    start = clock();
    for (stage = 0; stage < MAX_STAGE; stage++) {
        char no[LEVEL_MAX + 1];             /* 記憶すべき数字の並び */
        char x[LEVEL_MAX * 2];              /* 読み込んだ数字の並び */

        no[0] = '1' + rand() % 9;           /* 先頭文字は'1'〜'9' */
        for (i = 1; i < level; i++)
            no[i] = '0' + rand() % 10;      /* それ以降は'0'〜'9' */
        no[level] = '\0';

        printf("%s", no);
        fflush(stdout);
        sleep(125 * level);                 /* 問題提示は125×levelミリ秒 */
```

```
        printf("\r%*s\r入力せよ：", level, "");
        scanf("%s", x);

        if (strcmp(no, x) != 0)
            printf("\a間違いです。\n");
        else {
            printf("正解です。\n");
            success++;
        }
    }
    end = clock();

    printf("%d回中%d回成功しました。\n", MAX_STAGE, success);
    printf("%.1f秒でした。\n", (double)(end - start) / CLOCKS_PER_SEC);

    return 0;
}
```

トレーニングレベルの入力

まずはプログラムを実行してみましょう。

最初に、トレーニングの“レベル”を3～20の範囲で入力するよう促されます。本トレーニングのレベルとは、**記憶すべき数値の桁数**のことです。右に示す実行例のように、レベルとして6を入力すると、

6桁の数値を記憶しましょう。

と表示され、トレーニングが開始します。

実行例

```
数値記憶トレーニング
挑戦するレベル（3～20）：6
6桁の数値を記憶しましょう。
139237  … 0.75秒で消えます。
入力せよ：139237
正解です。
243568  … 0.75秒で消えます。
入力せよ：243586
♪間違いです。

… 中略 …

10回中8回成功しました。
32.2秒でした。
```

＊

トレーニングレベルを自由に設定できるようになっていることもあり、前のプログラムに比べると、プログラムは複雑です。トレーニングレベルの読込みに関連するのが1と2です。まずは、この部分を理解しましょう。

1 レベルの最小値3と最大値20は、それぞれ*LEVEL_MIN*と*LEVEL_MAX*というオブジェクト形式マクロとして定義されています。

2 プレーヤにレベルの入力を促して、変数*level*に整数値を読み込みます。読み込んだ値が*LEVEL_MIN*以上*LEVEL_MAX*以下（すなわち3以上20以下）でなければ、**do**文を繰り返します。したがって、**do**文が終了した時点での変数*level*の値は、**必ず3以上20以下となります。**

なお、ド・モルガンの法則を利用して、**do**文を以下のように実現することもできます。

```
do {
    /*… 中略 …*/
} while (!(level >= LEVEL_MIN && level <= LEVEL_MAX));
```

数値を文字列で表す

処理系に依存することなく、**int** 型や ***rand*** 関数で 5 桁以上の数値を取り扱うのは困難です（p.124）。本プログラムでは、記憶すべき数値とプレーヤが入力する数値を、整数ではなく**文字列**で表しています。それらの文字列は、以下のように宣言されています。

```
char no[LEVEL_MAX + 1];      /* 記憶すべき数字の並び */
char x[LEVEL_MAX * 2];       /* 読み込んだ数字の並び */
```

- 記憶すべき数字の並び … no

記憶すべき数値は最大で *LEVEL_MAX* 桁（20 桁）です。文字列の末尾にはナル文字が必要ですから、配列 *no* の要素数は *LEVEL_MAX* + 1 としています（ナル文字を含めて 21 文字分です）。

- 読み込んだ（プレーヤが入力した）数字の並び … x

読み込んだ数値を格納するための配列が *x* です。その要素数は、*no* と同じ *LEVEL_MAX* + 1 でよいはずです。しかし、本プログラムでは *LEVEL_MAX* * 2 としています（ナル文字を含めて 40 文字分です）。要素数を大きくしているのは、**プレーヤがキーボードから 20 桁以上の数値を打ち込んだ場合を考慮しているためです。**

問題用文字列の作成

問題用文字列の先頭は '1' 〜 '9' のいずれかの文字で、2 文字目以降は '0' 〜 '9' のいずれかの文字です。もちろん、文字数はレベルと同じです。

▶ レベルが 6 であれば、"100000" 〜 "999999" の範囲の文字列となります。

Fig.5-1 に示すように、先頭の *no*[0] に '1' 〜 '9' のいずれかの文字を、それ以降の *no*[1], *no*[2], … , *no*[*level* - 1] に '0' 〜 '9' のいずれかの文字をランダムに生成して格納します。

▶ 数字文字 '0', '1', …, '9' のコードは一つずつ増える（p.115）ため、文字 '1' に 0 〜 8 の数値を加えると '1' 〜 '9' が得られ、文字 '0' に 0 〜 9 の数値を加えると '0' 〜 '9' が得られます。

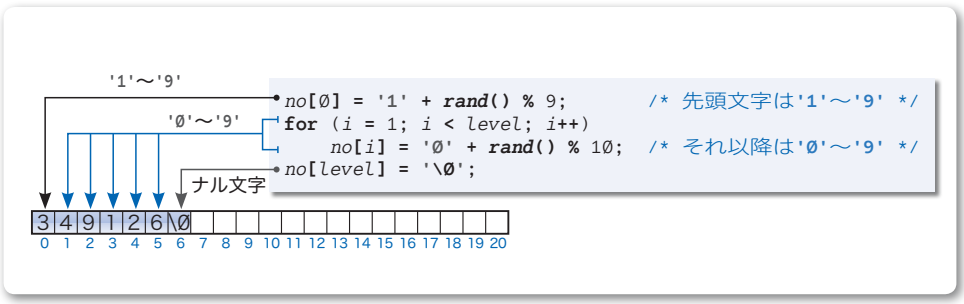

Fig.5-1 問題用文字列の作成（レベルが6の場合）

最後に文字列の終端を表すナル文字を no[level] に格納すると、問題用文字列の作成が完了です。

問題用文字列の表示

問題用文字列 no が完成すると、それを画面に表示します。表示する時間は、トレーニングレベルに比例した 125 × level ミリ秒です。

```
printf("%s", no);
fflush(stdout);
sleep(125 * level);

printf("\r%*s\r入力せよ：", level, "");
```

▶ すなわち、数値の桁数が多くなるほど長く表示されることになります。たとえば、レベルが 6 であれば表示時間は 0.75 秒となり、レベルが 8 であれば 1.0 秒となります。

『入力せよ：』と表示を行う部分は複雑です（**Fig.5-2** a）。まず、復帰 \r によってカーソルを先頭に戻した後に、level 個の空白文字を表示して問題を消去します。それから、再び復帰 \r によってカーソルを先頭に戻して『入力せよ：』と表示します。

図 b に示すように、カーソルを行の先頭に戻した状態から『入力せよ：』と表示してはいけません。問題の一部が残ってしまうからです。

▶ 書式文字列 "%*s" を用いて、任意の個数の空白文字を表示する手法は、第 2 章で学習しました（p.59）。

Fig.5-2 問題消去のための復帰と空白文字の表示