Lesson 5
記憶力トレーニング

本 Lesson では、短期記憶や作業記憶を鍛える《単純記憶力トレーニング》《プラスワントレーニング》を開発します。

この Lesson で学ぶおもなこと

- 整数型の表現範囲
- 特定範囲の数値の読込み
- 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）
- 配列要素の循環的利用

- strcmp 関数

5-1 単純記憶トレーニング

本 Lesson では、記憶力トレーニングソフトを作成しながら配列や文字列の活用法を学習します。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。4 桁の数値が 0.5 秒間だけ表示されるので、それを記憶して、その値をキーボードから打ち込んでください。10 回のトレーニングが終了すると、正しく解答できた回数と所要時間が表示されます。

▶ ここ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でしょう。

整数の表現範囲

記憶すべき数値の桁数を増やすことにします。プログラムの変更は容易と感じられるかもしれませんが、実はそうではありません。

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。

▶ ここに示すのは**最低限の値**であり、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** 型に変更すればよさそうです。

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）高々 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**。

List 5-1

```
/*
    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 c, s = clock();

    do {
        if ((c = clock()) == (clock_t)-1)       /* エラー */
            return (0);
    } while (1000UL * (c - s) / 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;          /* 記憶する数 */

        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。
```

任意の桁の数値を記憶するトレーニング

記憶すべき数値を最大 20 桁までに拡張したプログラムを **List 5-2** に示します。

List 5-2

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

#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 c, s = clock();

    do {
        if ((c = clock()) == (clock_t)-1)       /* エラー */
            return (0);
    } while (1000UL * (c - s) / 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("数字記憶トレーニング\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';
```

レベルの読込み

```
        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);
}
```


レベルの入力

プログラムを実行してみましょう。

まず最初に、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**です。

レベルの最小値 3 と最大値 20 を表すのが、二つのマクロ *LEVEL_MIN* と *LEVEL_MAX* です。

プレーヤにレベルの入力を促して、変数 *level* に読み込むのが、プログラムの網かけ部です。

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。**

実行例

```
数字記憶トレーニング
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。
```

*

もし変数 *level* に読み込まれた値が 6 であれば、

> 6 桁の数値を記憶しましょう。

と表示してトレーニングを開始します。

数値を文字列で表す

5 桁以上の数値を処理系に依存することなく **int** 型や ***rand*** 関数で取り扱うのは困難です。そこで本プログラムでは、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、以下に示す**文字列**で表します。

```
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。

■ プレーヤが入力する数字の並び … x

配列 *x* の要素数は、*LEVEL_MAX* * 2 です（ナル文字を含めて 40 文字分です）。

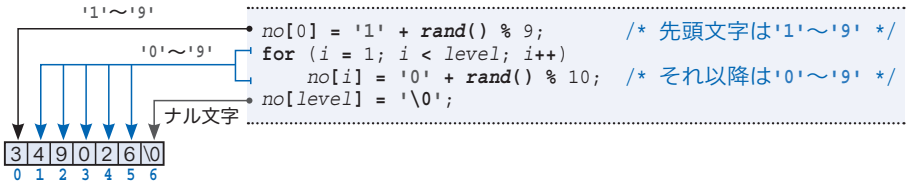
要素数を多めにしているのは、**プレーヤがキーボードから 20 桁以上の数値を打ち込んだときに、他の変数領域を壊さないようにする**ためです。

問題用文字列の作成と表示

たとえば、レベル 6 の問題が 100000 ～ 999999 であることからも分かるように、出題するのは、先頭が 1 から 9 で、2 文字目以降が 0 から 9 の並びです。

Fig.5-1 に示すように、先頭の *no*[0] に '1' ～ '9' を、それ以降の *no*[1], *no*[2], … , *no*[*level* - 1] に '0' ～ '9' 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.103）ため、'1' に 0 ～ 8 を加えると '1' ～ '9' が得られ、'0' に 0 ～ 9 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。

作成した問題用文字列 *no* を表示するのは 125 × *level* ミリ秒です。

▶ たとえばレベルが 6 であれば問題は 0.75 秒だけ表示されます。

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

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

カーソルを行の先頭に戻してから "入力せよ：" と表示すると、次のように、問題の数が消されずに残ってしまいます（レベルが 10 以上の場合）。

入力せよ：538

カーソルを先頭に戻した後に、*level* 個のスペースを表示して問題を消して、再びカーソルを先頭に戻してから『入力せよ：』と表示していることに注意しましょう。

▶ 書式文字列 "%*s" は、Lesson 2 で学習しました（p.51）。

strcmp 関数：文字列の比較

出題した文字列 *no* と、読み込んだ文字列 *x* が等しいかどうかの判断に利用しているのが、文字列を比較する ***strcmp*** 関数です。

	strcmp
ヘッダ	#include <string.h>
形　式	int ***strcmp***(const char *s1, const char *s2);
機　能	*s1* が指す文字列と *s2* が指す文字列の大小関係（先頭から順に 1 文字ずつ比較していき、異なる文字が出現したときに、それらの文字の対に成立する大小関係とする）の比較を行う。
返却値	等しければ 0、*s1* が *s2* より大きければ正の整数値、*s1* が *s2* より小さければ負の整数値を返す。

本プログラムでは、二つの文字列 *no* と *x* が等しいかどうかを、***strcmp*** 関数が返した値が 0 であるかどうかで判断しています。

正解した場合は、変数 *success* の値をインクリメントします。

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

▶ ***strcmp*** 関数による文字列の大小関係の判断の結果は、**文字コード体系に依存する**ことを知っておきましょう。

というのも、各文字の値は、その環境で採用されている文字コード体系に依存しており、その値をもとに比較を行うからです。

したがって、"123" と "ABC" の大小関係や、"abc" と "ABC" の大小関係は、採用されている文字コード体系によって変わることになります。