第１２回

４．いくつかのトピック

(5) main 関数の引数を利用したファイル処理

main 関数は，起動する環境から引数を受け取ることができる。
例えば、次に示すように，main 関数に引数を用いたプログラムを作成する。

```
01    /* sample */
02    /* main 関数の引数 */
03    #include <stdio.h>
04
05    main(int argc, char *argv[])
06    {
07        int i;
08        printf("argc = %d¥n", argc);
09        for (i = 0; i <= argc; i ++)
10            printf("argv[%d] = %s¥n", i, argv[i]);
11    }
```

このプログラムをコンパイルして実行ファイル名を sample とした場合，次のようにコマンドラインで sample の後に（複数の）文字列を並べて入力すると，

```
$ ./sample a b cpy↵
```

実行結果は，次のようになる。

```
argc = 4
argv[0] = ./sample
argv[1] = a
argv[2] = b
argv[3] = cpy
argv[4] = (null)
```

このように，プログラムに"a"，"b"，"cpy"という文字列へのポインタを渡すことができ，

argv[0]にはコマンド名（実行ファイル名），argv[argc]には null が入る。
このようなことを行うためには，main 関数で次のように宣言すればよい。

```
main(int argc, char *argv[])
```

例題４－６

任意のテキストファイルの内容を，次のようにコマンドを入力することによって表示するプログラムをつくりなさい。

```
$ ./e4-6  ファイル名↵
```

考え方

任意のテキストファイルの「ファイル名」は argv[1]に入るので，そのファイル名でオープンし，fgetc 関数でファイルを読み込み，putchar 関数で表示すればよい。

▼プログラム４－６

```
01      * E4-6 */
02      /*テキストファイルの表示*/
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      main(int argc, char *argv[])
07      {
08          FILE *fp1;
09          int d;
10
11          /* コマンドライン引数の数が正しいかどうか確認する */
12          if (argc != 2) {
13              printf("使用法:〈プログラム名〉〈ファイル名〉¥n");
```

```
14              exit(1);
15          }
16
17          if ((fp1 = fopen(argv[1], "r")) == NULL) {
18              printf("ファイルがオープンできません。¥n");
19              exit(1);
20          }
21
22          while ((d = fgetc(fp1)) != EOF)
23              putchar(d);
24          fclose(fp1);
25      }
```

練習問題３４

練習問題３０で，入力ファイル名，暗号化ファイル名，キー文字をコマンドラインから指定できるようにしなさい。例えば，次のようにコマンドラインから入力することになります。

```
$ ./Q34 original.txt cipher.txt %↵
```

練習問題３５

コマンドラインで指定した２つのファイルの内容を交換するプログラムを作りなさい。例えば，次のように c.txt および d.txt と指定した場合は，プログラムを実行すると，もともと d.txt に入っていたデータが c.txt に入り，c.txt の内容が d.txt に入ります。

```
$ ./Q35 c.txt d.txt↵
```

《ヒント》プログラム内で一時的な作業ファイル temp.txt を読み書き用にオープンして，temp.txt を媒介にして２つのファイルの内容を交換するようにしなさい。いずれのファイルもバイナリモードでオープンする必要があります。また、内容を交換する際、前の内容が残らないようにするには、いったんファイルをクローズして新たに書き込み用にオープンすればうまくいきます。

(6) typedef

C には新しいデータ型の名前を作成するために，typedef と呼ばれる機能が用意されています。typedef の一般形式は次の通り。

```
typedef  既存の型名  新しい型名;
```

例えば，次の typedef 宣言は名前 Length を int の同義語にする。これにより，この型 Length は int 型とまったく同様に宣言や型変換で使用できる。

```
typedef int Length;
Length len, maxlen;
Length *length[];
```

同様に，宣言

```
typedef char *String;
```

は String を char *すなわち文字ポインタの同義語にする。以下のように使用できる。

```
String p, lineptr[MAXLINES];
p = (String) malloc(100);
```

例題４－７

ジョセファスの問題を解くプログラムについて，構造体に対する typedef の使い方をマスターしてみましょう。

```
typedef struct node {
    int key;
    struct node *next;
} Listnode;
typedef Listnode *Listptr;
```

▼プログラム４－７

```
01    /* E4-7 */
02    /* ジョセファスの問題 */
03    /* 構造体の typedef */
03    #include <stdio.h>
04    #include <stdlib.h>
05
06    typedef struct node {
```

```
07        int key;
08        struct node *next;
09    } Listnode;
10    typedef Listnode *Listptr;
11
12    main()
13    {
14        int i, j, N, M;
15        Listptr t, x;
16        printf("Enter the total number of people : ");
17        scanf("%d", &N);
18        printf("Who attempts suicide first ? ");
19        scanf("%d", &M);
20        printf("¥nThe sequence of suicide is as follows: ¥n");
21        t = (Listptr) malloc(sizeof(Listnode));
22        t->key = 1;
23        x = t;
24        for (i = 2; i <= N; i++) {
25            t->next = (Listptr) malloc(sizeof(Listnode));
26            t = t->next;
27            t->key = i;
28        }
29        t->next = x;
30        j = 0;
31        while (t != t->next) {
32            for (i = 1; i < M; i++)
33                t = t->next;
34            printf("%3d -->", t->next->key);
35            x = t->next;
36            t->next = t->next->next;
37            free(x);
38            j++;
39            if (j%10 == 0) printf("¥n");
40        }
41        printf("%3d¥n¥n", t->key);
42    }
```

(7) 関数へのポインタ

C では，関数それ自身は変数ではないが，関数へのポインタを定義するのは可能であり，このポインタは代入したり，配列に置いたり，関数へ渡したり，関数から返したりすることができる。

関数へのポインタを宣言するには，関数の戻り値と同じ型を持つポインタ変数として宣言し，仮引数も後ろに続けて宣言する。例えば，2 つの整数型の仮引数を持ち，整数を返す関数へのポインタ p を宣言する一般形式は次のようになる。

```
int (*p)(int, int);
```

演算子の優先順位から，*p を括弧で囲む必要がある点に注意すること。

例えば，関数 sum()のプロトタイプ宣言

```
int sum(int, int);
```

があれば，次のような代入ができる。

```
p = sum;
```

ここで，sum が関数 sum のアドレスになっていることに注意する。すなわち，関数名はアドレスを表しており，配列名の場合と同様に，&演算子を前に付ける必要はない。

例題４－８

関数ポインタの配列を使う例を考えてみましょう。ここでは，加減乗除を行う 4 つの関数 sum, subtract, mul, div を宣言しておき，これらの関数へのポインタを配列に格納して使い分けを行うプログラムを作成してみましょう。

▼プログラム４－８

```
01      /* E4-8 */
02      /* 関数ポインタの配列 */
03      #include <stdio.h>
04
05      int sum(int, int);
06      int subtract(int, int);
07      int mul(int, int);
08      int div(int, int);
09
```

```
10      int (*p[4])(int, int);
11
12      main()
13      {
14          int result;
15          int i, j, op;
16          char *name[4] = { "加算", "減算", "乗算", "除算" };
17
18          p[0] = sum;
19          p[1] = subtract;
20          p[2] = mul;
21          p[3] = div;
22
23          printf("2 つの整数を入力して下さい : ");
24          scanf("%d%d", &i, &j);
25          printf("0:加算, 1:減算, 2:乗算, 3:除算¥n");
26          do {
27              printf("演算の番号を入力して下さい : ");
28              scanf("%d", &op);
29          } while (op < 0 || op > 3);
30
31          result = (*p[op])(i, j);
32          printf("%s の結果 = %d¥n", name[op], result);
33      }
34
35      int sum(int a, int b)
36      {
37          return a+b;
38      }
39
40      int subtract(int a, int b)
41      {
42          return a-b;
43      }
44
45      int mul(int a, int b)
```

```
46      {
47          return a*b;
48      }
49
50      int div(int a, int b)
51      {
52          return b ? a/b : 0;
53      }
```

注意：上記プログラムの 10 行目で，次のように関数ポインタの配列を初期化することもできる。この場合は，18～21 行は不要となる。

```
10      int (*p[4])(int, int) = {sum, subtract, mul, div};
```

練習問題３６（難問）

ヘッダファイル<stdlib.h>で定義されているライブラリ関数 qsort()は，クイックソートアルゴリズムを用いて,どんな型の 1 次元配列にも適用できる汎用のソートルーチンです。qsort 関数の一般形式を示します。

```
void qsort(void *array, size_t number, size_t size,
      int (*comp)(const void *a, const void *b));
```

ここで，array はソートの対象となる配列の最初の要素へのポインタです。number で配列の要素数，size で各要素のサイズ（バイト数）を指定します。最後の引数が比較用の関数へのポインタです。この比較用関数はユーザが自分で作成します。但し，int 型の戻り値は次のようにします。

*a > *b　正の値を返します。
*a == *b　ゼロを返します。
*a < *b　負の値を返します。

ここで，次のプロトタイプ宣言をもつ，それぞれ int 型と double 型の引数の比較を行う関数 comp_i と comp_d とを用意してみましょう。

```
int comp_i(const void *m, const void *n);
int comp_d(const void *x, const void *y);
```

最後に，これらを用いて，次の出力例に示すような動作を行う main 関数を作成しなさい。

▼出力例

ライブラリ関数 qsort を使って乱数のソートを行います

整数 1～100 の乱数を 10 個発生しました
　79　50　97　92　94　16　41　13　5　30
ソートすると
　5　13　16　30　41　50　79　92　94　97

実数[0, 1]の乱数を 10 個発生しました
0.697　0.718　0.225　0.045　0.913　0.783　0.894　0.471　0.288　0.854

ソートすると

0.045　0.225　0.288　0.471　0.697　0.718　0.783　0.854　0.894　0.913

《ヒント》

比較関数 comp_i の定義は次のようになります。

```
int comp_i(const void *m, const void *n)
{
    return *(int *)m - *(int *)n;
}
```

これを参考にして，comp_d を作成してみましょう。

また，乱数の発生には現在時刻を利用して乱数の種を初期化しましょう。次のようになります。但し，時間関数を使うので #include <time.h> が必要です。

```
    time_t now;
    …
    time(&now);
    srand(now);
```

乱数発生のライブラリ関数 rand および乱数の種をセットする関数 srand はヘッダファイル<stdlib.h>で定義されています。参考書で調べて，整数 1～100 や実数[0, 1]の範囲の乱数を発生する方法を考えてみましょう。