第１１回(7/2)

４．いくつかのトピック

(1) ビットごとの演算子

C言語には，次のようなビット単位で演算を行う特別な演算子が用意されている。

&	ビットごとの AND
|	ビットごとの OR
^	ビットごとの XOR（排他的論理和）
~	1 の補数

これらの演算子は文字型と整数型で機能し，浮動小数点数型では使用できない。AND, OR, XOR は，それぞれのオペランドの対応するビットを比較して結果を返す。一方，１の補数演算子は，整数型もしくは文字型の各ビットを反転する単項演算子である。

《クイズ》次の演算はどんな意味をもっているだろうか。

```
n = n & 0177;
x = x & ~077;
```

例題４－１

XOR 演算子には面白い特徴がある。a，b という２つの値がある場合，a XOR b の結果に対してもう一度 b で XOR 演算を行うと，結果は a に戻る。例えば，a = 100，b = 21987 とした場合，次のような出力結果が得られるプログラムをつくりなさい。

▼出力結果４－１

```
a の値：100
b の値：21987
1 番目の a = a^b の値：21895
2 番目の a = a^b の値：100
```

▼プログラム４－１

```
01      /* E4-1 */
02      /* XOR 演算子 */
03      #include <stdio.h>
04
05      int main()
06      {
07          int a, b;
08          a = 100; b = 21987;
09          printf("a の値：%d¥n", a);
10          printf("b の値：%d¥n", b);
11          a = a ^ b;
12          printf("1 番目の a = a^b の値：%d¥n", a);
13          a = a ^ b;
14          printf("2 番目の a = a^b の値：%d¥n", a);
15
16          return 0;
17      }
```

練習問題３０

例題４－１を参考にして，ユーザが指定したキー文字（unsigned char 型）と XOR 演算子を組み合わせて，ファイルを暗号化するプログラムをつくりなさい。但し，ファイル名とキー文字はキーボードから入力します。出力例のように動作する main 関数を作成しなさい。

《ヒント》暗号化されたファイルを入力して同じキー文字を用いて再度実行すると暗号が解除できることを確認しなさい。

▼出力例

ファイルを暗号化します！

入力ファイル名は：Q30.c↵
出力ファイル名は：Q30_crypt.c↵
暗号化のキー文字は：%↵

暗号化が完了しました！

(2) シフト演算子

C言語には，左または右方向のビットシフト演算子がある。書式は次のようになる。

値　<<　ビット数

値　>>　ビット数

左シフトは数値に 2 を掛けるのに等しく，シフトされた後の右側のビットは 0 で埋められる。一方，右シフトは数値を 2 で割るのに等しく，シフトされた左側のビットは 0 で埋められる（下記の注を参照)。但し，負の数が指定された場合は，1 で埋められる。いずれもシフトしてはみ出したビットは消失する。

ほぼ全ての CPU の内部演算で，シフト演算は算術演算よりも高速に処理される。

注．右シフトの注意

符号なし数の右シフトでは左から 0 が入るが，符号付きの場合は計算機によって符号桁が左から入ってくる場合（算術シフト）と 0 が入ってくる場合（論理シフト）とがある。

《クイズ》次の関数の働きを述べなさい。但し，仮定として，ビット位置 0 は右端で，n, p は正の整数で(p+1-n)は 0 以上とする。例えば，getbits(x,4,3)は何を返すでしょう。

```
unsigned getbits(unsigned x, int p, int n)
{
    return (x >> (p+1-n)) & ~(~0 << n);
}
```

練習問題３１

Ｃ言語には，バイトデータのローテート（rotate）演算子がない。ローテート処理では，はみ出たビットを逆の端に置く。例えば，10100000 を左に１つローテートすると 01000001 となる。呼び出される毎に１バイトデータの各ビットを１つずつ左にローテートする関数

```
void rotate(unsigned char *)
```

を作成しなさい。この関数を利用して，ユーザが入力した任意の整数（1～255）を合計 16 回ローテートした結果を出力例のように表示するプログラムをつくりなさい。

《ヒント》バイトの端からはみ出たビットにアクセスするには共用体を用いる方法が使えます。あるいは，ローテートする前にビット演算子を用いて最上位ビットが 1 か 0 かを判定して，ローテートした際のビット列を決定しなさい。

ユーザが入力した整数は scanf を用いて unsigned char n で宣言した変数 n で受けて，ローテート関数は rotate(&n)のように呼び出しなさい。また，出力例のようにビット列を表示する部分は，n の各ビットが 0 か 1 をビット演算子で調べればできます。

▼出力例

```
1～255 の整数を入力して下さい（0 で終了）：1↵
  1-th shift:       2   00000010
  2-th shift:       4   00000100
  3-th shift:       8   00001000
  4-th shift:      16   00010000
  5-th shift:      32   00100000
  6-th shift:      64   01000000
  7-th shift:     128   10000000
  8-th shift:       1   00000001
  9-th shift:       2   00000010
 10-th shift:       4   00000100
 11-th shift:       8   00001000
 12-th shift:      16   00010000
 13-th shift:      32   00100000
 14-th shift:      64   01000000
 15-th shift:     128   10000000
 16-th shift:       1   00000001
```

1～255 の整数を入力して下さい（0 で終了）：

(3) 動的なメモリの割り当て

動的なメモリの割り当てとは，プログラムの実行中に必要に応じてメモリを割り当てる処理を指す。コンピュータ上のすべてのメモリをフルに利用する必要のあるアプリケーションで有効である。

Ｃの動的メモリ割り当て関数の核となるのは malloc()と free()関数で,次の書式を取る。

```
void *malloc(size_t n);
void free(void *ptr);
```

malloc()は n バイトのメモリ領域を割り当てて，割り当てたメモリブロックの先頭を指すポインタを返す。この記憶領域は 0 で初期化される。また，メモリ要求に応えられないときは，NULL ポインタを返す。

但し，size_t は sizeof 演算子が返す符号なし整数型である。

free(ptr)は ptr で指し示されるメモリ領域を解放する関数である。但し，ptr は割り当てられたメモリブロックの先頭を指すポインタである。

どちらの関数も stdlib.h を使用するため，これをインクルードしておく。

注）総称的なポインタ型 void＊：void＊ には任意のポインタがキャスト可能であり，情報を失うことなく逆キャストもできる。この汎用ポインタには次の二つの目的がある。

(1)「型が一致しない」というエラーを起こさずに，任意の型のデータを指すポインタとして関数に渡すことができる。

(2) 関数の戻り値として汎用ポインタを返せるようにする。

例題４－２

10 個の要素を持つ動的な整数配列を作るプログラムを作成しなさい。次に，ポインタ演算または配列の添え字を使って，1 から 10 までの値を配列に代入しなさい。最後に，代入した値をすべて表示して，その後でメモリを解放しなさい。

▼プログラム４－２

```
01      /* E4-2 */
02      /* 動的なメモリの割り当て */
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      int main()
07      {
08          int *ptr, i;
09
10          ptr = (int *) malloc(10 * sizeof(int));
11          if (!ptr) {
12              printf("メモリー割り当てエラー");
13              exit(1);
14          }
15
16          for (i = 0; i < 10; i++) ptr[i]= i+1;
17
18          printf("代入した整数配列を表示します¥n");
19          for (i = 0; i < 10; i++) printf("%d ", *(ptr+i));
20          printf("¥n");
21
22          free(ptr);
23
24          return 0;
25      }
```

例題４－３

２次元の整数型配列を動的に生成するプログラムをつくってみましょう。このために，行数 m と列数 n を引数として渡す関数

```
int **get_array(int m, int n)
```

を作成します。この関数 get_array は関数内部で動的に生成した整数型の 2 次元配列 array[m][n]の先頭アドレスを返すものとします。すなわち，main 関数であらかじめ宣言してある

```
int **array;
```

に対して，次のようにして 2 次元配列のメモリを確保できます。

```
array = get_array(m, n);
```

こうしてメモリを確保した array[m][n]に main 関数内で j=0,…,m-1, i=0,…,n-1 として

```
array[j][i] = (j+1)*(i+1);
```

を代入してから，array[m][n]の内容を画面に表示してみなさい。

▼出力例

```
整数の 2 次元配列を動的に生成します。
行数を入力して下さい：4↵
列数を入力して下さい：5↵

配列内のデータを表示します。
  1   2   3   4   5
  2   4   6   8  10
  3   6   9  12  15
  4   8  12  16  20
```

▼プログラム４－３

```
01      /* E4-3 */
02      /* 2 次元配列の動的生成 */
03      #include <stdio.h>
04      #include <stdlib.h>
```

```
05
06      int **get_array(int m, int n);
07      void free_array(int **array, int m);
08
09      main()
10      {
11          int m, n;
12          int **array;
13          int i, j;
14
15          printf("整数の 2 次元配列を動的に生成します。¥n");
16          printf("行数を入力して下さい：");
17          scanf("%d", &m);
18          printf("列数を入力して下さい：");
19          scanf("%d", &n);
20
21          array = get_array(m, n);
22
23          for (j = 0; j < m; j++)
24              for (i = 0; i < n; i++)
25                  array[j][i] = (j+1)*(i+1);
26
27          printf("¥n 配列内のデータを表示します。¥n");
28          for (j = 0; j < m; j++)
29              for (i = 0; i < n; i++) {
30                  printf("%4d", array[j][i]);
31                  if (i == n-1) printf("¥n");
32              }
33          free_array(array, m);
34      }
35
36      int **get_array(int m, int n)
37      {
38          int **array;
39          int i;
40
```

```
41          array = (int **) malloc(m * sizeof(int *));
42          if (array == NULL) {
43              puts("Memory problem¥n");
44              exit(1);
45          }
46          for (i = 0; i < m; i++) {
47              array[i] = (int *) malloc(n * sizeof(int));
48              if (array[i] == NULL) {
49                  puts("Memory problem¥n");
50                  exit(1);
51              }
52          }
53          return array;
54      }
55
56      void free_array(int **array, int m)
57      {
58          int i;
59
60          for (i = 0; i < m; i++)
61              free(array[i]);
62
63          free(array);
64      }
```

(4) 再帰 (Recursion)

Ｃの関数は再帰的に使用できる。つまり，関数は自分自身を直接または間接に呼び出してよい。

また，自己参照的構造体では，構造体の宣言で自分自身をメンバとして“再帰的”に宣言することができる。

いくつかの例題で動作を確認してみる。

例題４－４

整数 n を文字列として印字する関数 void printd(int n)を再帰的に定義して作成し，キーボードから入力した数字を印字するプログラムをつくりなさい。

▼プログラム４－４

```
01  /* E4-4 */
02  /* 再帰 */
03  #include <stdio.h>
04  void printd(int);
05
06  main()
07  {
08      int n;
09
10      while (1) {
11          printf("整数を入力して下さい (exit = 0) : ");
12          scanf("%d", &n);
13          if (!n) break;
14          printd(n);
15          printf("¥n");
16      }
17  }
18
19  void printd(int n)
20  {
21      if (n < 0) {
22          putchar('-');
23          n = -n;
24      }
25      if (n / 10)
26          printd(n / 10);
27      putchar(n%10 + '0');
28  }
```

練習問題３２（難問）

例題４－４で紹介したprintdの考え方を用いて,再帰版の関数itoa_rをつくりなさい。すなわち，次のソースコードを用いて，再帰的なルーチンを呼び出して整数を文字列に変換する関数void itoa_r(int n, char *str)の定義部分を完成しなさい。

```
/* Q32 */
/* int型変数を文字列に変換する再帰版itoa_r */
#include <stdio.h>
#include <stdlib.h>

/* prototype declaration */
void itoa_r(int n, char *str);

enum {
    MAX = 100
};

main()
{
    int n;
    char str[MAX];

    while (1) {
        printf("¥n整数を入力して下さい（終了 = 0）:");
        scanf("%d", &n);
        if (n == 0) break;
        itoa_r(n, str);
```

```
        printf("文字列に変換すると %s です¥n", str);
    }
}

void itoa_r(int n, char *str)
{

    ここに関数定義をすること

}
```

例題４－５

“ジョセファス（Josephus）の問題”を解く。この問題とは，いま N 人が集団自殺をしようとしているとして，まず全員が円陣に並び，その円の中の M 番目の人を順に処刑する。1 人死ぬと取り除かれ，円の大きさが 1 減ることになる。問題は，最後に死ぬ人を見つけることである。一般には，人々の処刑される順番を見つけることが問題である。

例えば，N = 9，M = 5 とすると，

5 --> 1 --> 7 --> 4 --> 3 --> 6 --> 9 --> 2 --> 8

となり，最後に死ぬのは最初 8 番目に並んでいた人になる。この問題を解くプログラムについて自己参照的構造体を用いてつくりなさい。

▼プログラム４－５

```
01    /* E4-5 */
02    /* ジョセファスの問題 */
03    /* 自己参照的構造体 */
04    #include <stdio.h>
05    #include <stdlib.h>
06
07    struct node {
08        int key;
09        struct node *next;
10    };
11
12    main()
13    {
14        int i, j, N, M;
15        struct node *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 = (struct node *) malloc(sizeof *t);
22        t->key = 1;
23        x = t;
24        for (i = 2; i <= N; i++) {
25            t->next = (struct node *) malloc(sizeof *t);
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      }
```

練習問題３３

ジョセファスの問題を解く再帰的プログラムをつくりなさい。

《ヒント》ジョセファスの問題を解く部分（例題４－５のプログラム E4-5 の 30 行～41 行）を再帰関数で実現しなさい。この関数は次のようなプロトタイプ宣言になります。

```
void josephus(struct node *, int);
```