第６回

変数の有効範囲

```
void fun()
{
    int loc;                ┐
        …                   │   自動（ローカル）変数の有効範囲
                            ┘       関数内の宣言
}

int glob;                   ┐
main()                      │
{                           │
        …                   │   外部（グローバル）変数の有効範囲
}                           │       全ての関数外の宣言
void fun()                  │
{                           │
        …                   │
                            ┘
}
```

自動（ローカル）変数

関数の内部で宣言される。関数の外側からは全く見えない。
１つの関数の中では，任意のブロックの冒頭で，使用する自動変数を宣言する。

外部（グローバル）変数

全ての関数の外側で宣言することによって作成される。
プログラム全体から認識することができ，プログラムのどこからでもそれを使用することができる。プログラムの実行中はその値が保持される。

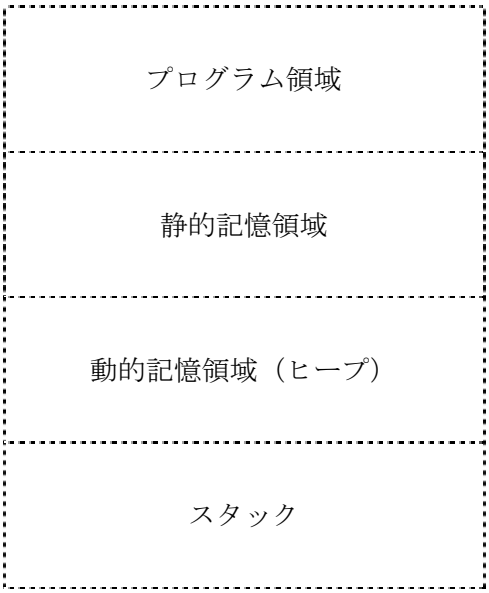
自動変数と外部変数の優先関係

```
#include <stdio.h>
void func(void);

int count; /* 外部変数 */

int main()
{
    count = 10;
    func();
    printf("main()の count の値：%d¥n", count);
    return 0;
}

void func(void)
{
    int count; /* 自動変数 */
    count = 100;
    printf("func()の count の値：%d¥n", count);
}
```

記憶域クラスの種類

変数は，記憶領域の確保される場所によって性質が異なる。したがって，記憶領域のどこに変数を確保するかという記憶域クラスを決定する必要がある。

まず，記憶領域は下図のように構成されている。但し，４つの領域の並び順はコンピュータ，コンパイラによって異なっており，この順番とは限らない。



記憶領域の構成

記憶域クラスの種類

種類	記憶域クラス指定子	記憶場所
自動	auto	スタック
静的	static	静的記憶領域
外部	extern	静的記憶領域 他のファイルで extern 宣言
レジスタ	register	レジスタ

ア）　自動変数

変数を宣言した関数が処理されている間だけスタックに存在し，関数の処理が終わると消滅する。必要となる期間だけ領域が確保されるので記憶装置の有効利用が可能となる。

記憶域クラス指定子 auto は省略することができる。

イ）　静的変数

関数の処理が終わっても消滅せず，値をいつまでも保持している。変数を初期化する場合は静的変数とする。

ウ）　外部変数

ソースプログラムを構成するすべてのファイルの中で，外部変数の定義は一つだけでなければならない。他のファイルからアクセスするためには，そのファイルで extern 宣言を行うことにより，別のファイルで定義されて確保されている外部変数の記憶領域を借りることになる。

エ）　レジスタ

レジスタに領域が割り当てられる。高速で処理することができる。ただし，使用できるレジスタの個数が限定されていることに注意する必要がある。

《初期化》

外部変数と静的変数の初期化は，プログラムの実行を始める前（コンパイル時）に一度だけ行われる（明示しないと 0 に初期化される）。一方，自動変数とレジスタ変数の初期化は，関数やブロックに入るごとに行われる。

☆プログラム実行中のアドレスの変化に着目すると次のように分類できる

アドレスが「固定」の変数：

［1］外部変数

［2］外部的に static な変数（他のファイルから参照できない外部変数）

［3］関数内の static な変数

アドレスが「変化」する変数：

［1］関数のパラメータ

［2］関数内の自動変数

例題２－３

次のプログラムを実行して，各記憶域クラスの変数等がどのように記憶領域上にマップされているか調べてみましょう。

▼プログラム２－３

```
01      /* E2-3 */
02      /* メモリマップを調べる */
03      #include <stdio.h>
04      #include <stdlib.h>
05
06      int glob;
07      static int stat;
08
09      void func(int p)
10      {
11          static int s;
12          int loc;
13
14          printf("parameter addr   : %p¥n", &p );
15          printf("func static addr : %p¥n", &s );
16          printf("func local  addr : %p¥n", &loc );
17      }
18
19      main()
20      {
21          char *alc = (char *)malloc(1000);
22
23          func(100);
24          printf("allocated addr   : %p¥n", alc );
25          printf("global var addr  : %p¥n", &glob );
26          printf("static var addr  : %p¥n", &stat );
27          printf("func() addr      : %p¥n", func );
28          printf("main() addr      : %p¥n", main );
29          printf("addr of printf   : %p¥n", printf );
30      }
```

・静的変数の使い方(1)

```
01    #include <stdio.h>
02
03    void f(void);
04
05    int main()
06    {
07        int i;
08        for (i = 0; i < 10; i++) f();
09        return 0;
10    }
11
12    void f(void)
13    {
14        static int count = 0;
15        count++;
16        printf("count : %d¥n", count);
17    }
```

・静的変数の使い方(2)

```
01    #include <stdio.h>
02
03    void f(void);
04
05    int main()
06    {
07        f();
08        return 0;
09    }
10
```

```
11    void f(void)
12    {
13        static int stop = 0;
14        stop++;
15        if (stop == 10) return;
16        printf("%d¥n", stop);
17        f(); /* 再帰呼び出し */
18    }
```

・外部変数の使い方

ファイル１

```
01    #include <stdio.h>
02
03    int count;
04
05    void func(void);
06
07    int main()
08    {
09        int i;
10        func();   /* count の値設定 */
11        for (i = 0; i < count; i++)
12            printf("%d¥n", i);
13        return 0;
14    }
```

ファイル２

```
01    #include <stdlib.h>
02    #include <time.h>
03
04    extern int count;   /* extern 宣言がないとコンパイルエラー */
05
```

```
06    void func(void)
07    {
08        time_t now;
09        time(&now);
10        srand(now);    /* 現在時刻を乱数の種に設定 */
11
12        count = rand()%100;
13    }
```

《分割コンパイルのしかた》

file2.c は変更しないのであれば，file2.c のみコンパイルをしてオブジェクトファイルにしておく。

```
$ gcc -c file2.c
```
← オブジェクトファイル file2.o ができる

一方，file1.c は変更を度々してその都度コンパイル・リンクして実行させるような場合，

```
$ gcc -o file1 file1.c file2.o
```
← 実行ファイル file1 ができる

により，file2.o をリンクして用いる。この場合，file1.c のオブジェクトファイルは作成されない。コンパイル処理が重い場合に，file2.c のコンパイルが省略できるので処理が早くなる。

練習問題２２

２２−１．初期設定された任意の文字列の順序を，逆に並べ換えるプログラムをつくりなさい。ただし，並べ換える部分を関数にして，関数には文字列を引数として渡しなさい。

文字列の例としては，

```
char str[] = "My name is Toru Wakahara";
```

のように自分の名前を用いてみなさい。

《ヒント》次のようなプロトタイプ宣言をもつ関数 reverse を定義しなさい。

```
void reverse(char *);
```

引数で渡された文字列自身が逆順になるようにしてみましょう。

２２−２．初期設定された任意の整数配列内のデータを，大きい順に並べ換えるプログラムをつくりなさい。ただし，並べ換える部分を関数にして，配列とデータ数を引数として渡しなさい。例として，次の 10 個の整数を格納した配列に適用してみなさい。

```
static int dd[10] = { 27, 89, 45, 18, 55, 64, 92, 73, 34, 88 };
```

並べ替えた結果は次のようになるはずです。

ソート後 ＝ 92 89 88 73 64 55 45 34 27 18

《ヒント》次のようなプロトタイプ宣言をもつ関数 isort を定義しなさい。

```
void isort(int *, int);
```

引数で渡された整数配列自身が大きい順に並べ換えられます。

２２−３．次の関数および記憶クラスに関する記述ア）～カ）の中で正しいものを選択しなさい。

ア）関数を呼び出す場合は，必ず実引数を与える必要がある。

イ）実引数は，変数名だけでなく配列のアドレスや定数を指定することができる。

ウ）記憶域クラス指定子の extern は，分割コンパイルをするために使用される。

エ）呼び出される関数を，呼び出す関数よりも前に記述することによって，関数のプロトタイプ宣言を省略することができる。

オ）実引数の名前と，仮引数の名前は同じでなければならない。

カ）関数に使用する return 文は，複数個あってもよいし，省略してもよい。

２２−４．（難問）指数関数 e^x の近似値を返す関数 expon(x, n)をつくります。第１引数は浮動小数点数 x で，第２引数は e^x を Taylor 展開で展開する際の最大次数 n になります。n 次項までの Taylor 展開の式は下に示す通りです。
さらに，浮動小数点数 y と正の整数 m を引数とし，y の小数点 m 桁より下を切り捨てて出力する関数 trunc(y, m)をつくりなさい。
次に，これらの２つの関数を用いて，e^x の近似値を小数点以下の桁数を指定して出力するプログラムをつくりなさい。

$$e^x \approx 1 + x + x^2/2! + x^3/3! + \ldots + x^n/n!$$

《ヒント》上式を効率よく計算する工夫をしなさい。
また，関数 trunc は，trunc(-3.141593, 3) と呼び出すと，-3.141 のように画面に出力する。但し，printf("%.*f", m, y); は用いないで定義すること。

▼出力例

指数関数 exp(x)の近似値を計算します！
x を入力してください：2.5↵
Taylor 展開の最大次数 n は：100↵
小数点以下の桁数 m は：3↵
exp(x)の近似値 ＝ 12.182

４．プリプロセッサ（２）

プリプロセッサの機能は，前処理指令を用いて，指示された内容に従ってソースプログラムを書き直すことである。

#include 命令は，外部に用意されたファイルを取り込み，差し替える機能である。これにより，任意のプログラムを外部ファイルとして保存しておき，必要に応じて（再度プログラミングせずに）取り込むことができる。

例題２－４

角度を入力すると，ラジアン角に変換した値を出力するプログラムをつくりなさい。ただし，角度を変換する部分は，関数として登録した外部ファイルをプリプロセッサの#include 命令を使用して読み込み，活用することとしなさい。

▼出力結果２－４

```
角度を整数で入力してください！
60↵
60 度は 1.047197[rad]です
```

考え方

角度を変換する部分は，あらかじめ関数として定義しておく。この関数はヘッダファイルとして取り込めるように，kaku.h というファイル名として書き出しておき，このファイルをメインプログラムと結合して実行プログラムを作成する。

▼プログラム２－４

```
01      /* E2-4 */
02      /* ラジアン角への変換 */
03      #include <stdio.h>
04      #include "kaku.h"
05
06      main()
07      {
08          int deg;
09          float rad;
10          printf("角度を整数で入力してください!¥n");
11          scanf("%d", &deg);
12          rad = kaku(deg);
13          printf("%d 度は%f[rad]です。¥n", deg, rad);
14      }
```

kaku.h の内容

```
float kaku(int deg)
{
    return 3.14159 * deg / 180.0 ;
}
```

５．探索

蓄積された大量のデータの中から，必要なデータをコンピュータによる計算によって取り出すことを探索という。

例題２－５

小さな数字から大きな数字の順に整列している整数型のデータ 15 個を，配列に初期設定する。次に，任意の整数型のデータを入力し，該当するデータが配列内に存在するかどうかを２分探索によって決定するプログラムをつくりなさい。

▼出力結果２－５

探索データを入力してください！
65↵
10 番目にあります

考え方

２分探索の対象となるデータは，定まった順序で整列していることが必要である。
探索は次の手順による。
①全体を半分に分け，その位置に目的とするデータがあるかどうかを判定する。目的とするデータがなければ，該当するデータが前半部分にあるか，後半部分にあるかを調べる
②探索するデータがはいっていると思われる部分について，①と同様の処理を半分に分けられなくなるまで繰り返す。

▼プログラム２－５

```
01      /* E2-5 */
02      /* ２分探索 */
03      #include <stdio.h>
04      #include <stdlib.h>
05      #define  KOSU  15
06
07      main()
08      {
09          int key, data[KOSU] =
10               {5, 8, 10, 15, 21, 33, 36, 41,
11                55, 65, 76, 79, 80, 85, 90};
12          int cent, lower = 0, upper = KOSU - 1;
13
14          printf("探索データを入力してください!¥n");
15          scanf("%d", &key);
16
17          while (lower <= upper) {
18              /* 中央の要素を対象とする */
19              cent = (upper + lower) / 2 ;
20              if (*(data + cent) == key) {
21                  printf("%d 番目にあります¥n", cent + 1);
22                  exit(0);      /* プログラム終了 */
23              }
24              if (*(data + cent) < key)
25                  lower = cent + 1;     /* 下限の変更 */
26              else
27                  upper = cent - 1;     /* 上限の変更 */
28          }
29          printf("%d はありません¥n", key);
30      }
```

練習問題２３

２３−１．例題２－５のプログラムで２分探索をする部分を関数にしなさい。ただし，引数として探索すべきデータと，配列のアドレス（配列名）および配列の大きさ（データ数）を渡すようにする。ただし，適合するデータがみつかった場合には何番目かを関数の戻り値として，みつからなかった場合は-1 を関数の戻り値として返すようにする。

２３−２．次のプログラムは，２分探索を応用したもので，郵便番号を入力すると該当する市町村名を出力するものである。①～⑤の空欄部分を埋めて，プログラムを完成しなさい。

```
01      /* 郵便番号→市町村名変換 */
02      #include <stdio.h>
03      #include <string.h>
04      #include <stdlib.h>
05      #define  KOSU   5
06
07      main()
08      {   static char *p_tabl[KOSU] = {"001", "010",
09                  "021", "030-13", "040" };
10          static char *t_tabl[KOSU] = {"北海道札幌市北区", "秋田県秋田市",
11                  "岩手県一関市", "青森県東津軽郡蟹田町", "北海道函館市"};
12          char p_no[10];
13          int  cent, lower = 0, upper = [ ① ] ;
14          printf("¥n 郵便番号を入力してください!¥n");
15          scanf("%s", p_no);
16          while (lower <= upper) {
17              cent = [ ② ] / 2;
18              if (!strcmp(p_tabl[cent], p_no)) {
19                  printf("¥n 郵便番号%s の市町村は%s です¥n", p_no, t_tabl[cent]);
20                  exit(0);
21              }
22              if (strcmp( [ ③ ] , p_no) < 0)
23                  lower = [ ④ ] ;
24              else
25                  upper = [ ⑤ ] ;
26          }
27          printf("¥n その郵便番号は，登録されていません¥n");
```

```
28      }
```

２３−３．前問２．のプログラムを改造して，英語を日本語に翻訳するプログラムをつくりなさい。例として，動物の名前を英語と日本語で 10 種類用意して，例えば dog と入力すると犬と出力するようにしなさい。登録されていない動物であれば，登録されていませんと出力しなさい。但し，動物の英語名はアルファベット順に並べておくこと。

Ⅲ．2 次元配列とファイルの応用

１．2 次元配列

・配列の宣言

型指定子　配列名[寸法][寸法]…[寸法]；

ex. int array[5][3]； 5×3 個の整数要素を格納

int table[2][3][2]; 2×3×2 個の整数要素を格納

・配列要素の格納順と添字の関係

1 次元配列 a[N]で要素 a[k]は何番目？　k+1 番目

2 次元配列 b[M][N]で要素 b[j][k]は何番目？　j*N+k+1 番目

3 次元配列 c[L][M][N]で要素 c[i][j][k]は何番目？　i*M*N+j*N+k+1 番目

・2 次元配列の初期化

OK と NO という文字列の初期値を設定する場合

```
static char c[2][3] = {'O', 'K', '¥0', 'N', 'O', '¥0'};
static char c[2][3] = {"OK", "NO"};
```

サイズのない配列　←　最初の寸法のみ省略可能

```
static int pwr[] = { 1, 2, 4, 8, 16 };
                        5 個の要素の長さで自動的に初期化
static char prompt[] = "Enter your name:";
static int sqr[][3] = {
                            1, 2, 3,
                            4, 5, 6,
                            7, 8, 9
                          };
```

《クイズ》2 次元配列 b[][N]の p 番目の要素 b[j][k]の添字は？　j = (p-1)/N, k = (p-1)%N

さらに，3 次元配列 c[][M][N]の p 番目の要素 c[i][j][k]の添字は？

i = (p-1)/M/N, j = (p-1)/N%M, k = (p-1)%N

《補足例題》次のような 2 次元配列を twod[4][5]に for ループで格納して表示せよ。

```
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
```

```
#include <stdio.h>

main()
{
    int i, j, twod[4][5];

    for (i = 0; i < 4; i++)
        for (j = 0; j < 5; j++)
            twod[i][j] = i*j;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 5; j++) {
            printf("%3d", twod[i][j]);
            if (j == 4) printf("¥n");
        }
}
```

・文字列配列の作成

```
char names[10][40];  ←  10 個の文字列を持ち，それぞれの文字列が
                            40 文字の長さ（ナル文字¥0 を含む）
gets(names[2]);      ←  3 番目の文字列 names[2]にキーボードから入力
printf(names[0]);    ←  最初の文字列 names[0]を画面に出力
```

《クイズ》上記をポインタ配列 pnames を用いて同様に書きなさい。

```
char *pnames[10];
gets(pnames[2]);
printf(pnames[0]);
```

《補足例題》英単語(zero, one, two, …, nine)を含む文字列配列を作成し，数字を文字として入力して，その数字に対応する英単語を出力するプログラムを作りなさい。
入力した文字から'0'を引くと添字が得られることを用いる。

```
#include <stdio.h>

main()
{
    char *digits[10] = {
      "zero", "one", "two", "three",
      "four", "five", "six", "seven",
      "eight", "nine"
    };
    int num, sub;

    printf("数字(0-9)を入力してください: ");
    num = getchar();

    printf("¥n");

    sub = num - '0';
    if (sub >= 0 && sub < 10) {
        printf("%c は英語では %s です¥n", num, digits[sub]);
    }
}
```

例題３－１

整数 n を入力すると，寸法 n の魔方陣を作成するプログラムをつくりなさい。ただし，n は 15 位内の奇数とする。

魔方陣は，次に示すド・ラ・ルーブルの方法によって作成することができる。ここでは，n=3 の場合について示す。

①第 1 行の中央の要素に 1 を入れる。

②次の数(1 回目は 2)を，右斜め上方の要素に入れる。このとき，該当する位置に要素がなければ，その行または列の一番遠方の要素に入れる。

③ただし，数を n で割った余りが 1 であればすぐ下の要素に入れる。

④次の数(2 回目は 3)を②と同じように考える。i=2 の右斜め上に要素がないので，その行の最も遠方の d に 3 を入れる。

⑤次の数(3 回目は 4)は n=3 で割った余りが 1 であるので, d=3 のすぐ下の g に入れる。

⑥上の②と③の手順を全部の要素に対して繰り返す。

a 8	b 1	c 6
d 3	e 5	f 7
g 4	h 9	i 2

▼出力結果３－１

15 以下の奇数を入力して下さい：5↵

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

考え方

魔方陣とは，1 から n^2(n は寸法)までの整数を n×n の行列の形に並べたもので，それぞれの行の和，列の和，および対角線上の和が等しいものをいう。

プログラムでは，問題に魔方陣の寸法は 15 以内と指定されているので，15×15 の 2 次元配列を確保する。その後，前述の方法に従って値を割り当てることによって完成する。

▼プログラム３－１

```
01      /* E3-1 */
02      /* 魔法陣 */
03      #include <stdio.h>
04
05      main()
06      {
07          static int mg[16][16], ii=1, jj, nn, kk;
08
09          printf("15以下の奇数を入力してください：");
10          scanf("%d", &nn);
11          jj = (nn + 1) / 2;
12          mg[1][jj] = 1;
13          for (kk = 2; kk <= nn*nn; kk++) {
14              if (kk%nn == 1) {
15                  ii++;
16              } else if (ii == 1) {
17                  ii = nn;
18                  jj++;
19              } else if (jj == nn) {
20                  ii--;
21                  jj = 1;
22              } else {
23                  ii--;
24                  jj++;
25              }
26              mg[ii][jj] = kk;
27          }
28          for (ii = 1; ii <= nn; ii++) {
29              printf("¥n");
30              for (jj = 1; jj <= nn; jj++)
31                  printf("%5d", mg[ii][jj]);
32          }
33      printf("¥n");
34      }
```

練習問題２４

２４−１．下図に示したプログラムは，ある 24 時間営業のコンビニエンスストアの 1 日分の売上データを利用して，時間帯別および売上げ金額別の客数を出力するものである。①，②の空欄部分を埋めて，プログラムを完成しなさい。なお，入力データは，次に示すような形式であるとする。

時刻		金額
時	分	
2 桁	2 桁	6 桁

出力形式の例は，次のとおり。

	− 999	1000−1999	2000−2999	3000−3999	4000−4999	5000−
00:00−00:59	2	5	1	0	1	0
01:00−01:59	4	1	3	0	0	1
02:00−02:59	1	0	0	2	0	0
・						
・						
23:00−23:59	3	4	1	0	0	1

```
/* Q24-1 */
/* コンビニエンスストアの売上分析 */
#include <stdio.h>

main()
{
    static int table[24][6];
    int hour, min, prank, price;

    printf("時刻と金額を入力してください (Ctrl-d で終了) !¥n");
    while ((scanf("%2d%2d%6d", &hour, &min, &price)) != EOF) {
        prank = price/1000 > 5 ? 5 : price/1000;
        table [ ① ] ++;
    }
    printf("%21s- 999  1000-1999  2000-2999  ", " ");
    printf("3000-3999  4000-4999  5000-¥n");
    for (hour = 0; hour <= 23; hour++) {
        printf("%02d:00-%02d:59", hour, hour);
        for (prank = 0; prank <= 5; prank++)
            printf("%11d", [ ② ] );
        printf("¥n");
    }
}
```

２４−２．（超難問）番兵を使った線形探索という面白い技法があります。例えば，

```
int a[N], x, i;
```

が与えられたとき，配列 a の中に x の値があるかどうかを探索する場合，普通のコードは次のようになるでしょう。

```
i = 0;
while (i < N && a[i] != x)
  i++;
if (i != N)
  printf("%d は%d 番目にあります¥n", x, i+1);
else
  printf("%d は見つかりませんでした！¥n", x);
```

ここで番兵を使う線形探索という技法では，配列のサイズを形式的に１増やして，次のように書きます。

```
int a[N+1], x, i;
…
a[N] = x;
i = 0;
while (x != a[i])
  i++;
…
```

こうすると，while 文の判定式が２つから１つに減ります。これによりプログラムは短くかつ高速になります。

では，問題です。

配列 a[N]の中の最大の要素を見つけるのに，配列のサイズを形式的に１増やして，要素 a[N]を番兵に使うプログラムを考えて下さい。