記憶クラス

山本昌志[*]

2006年11月17日

概 要

C 言語の記憶クラスにつて説明する．

1 前回の復習と本日の学習内容

1.1 復習

先週は，C 言語の変数のスコープについて学習した．変数のスコープとは，その変数の有効範囲のことで，それは宣言する場所で決まる．以下の箇条書きと図 1 に示すように 3 種類のスコープがある．

- 全ての関数の外側，プログラムの先頭付近で宣言した変数は全ての関数で有効である．これを**グローバル**変数と呼ぶ．
- 関数の先頭で宣言した変数は，その関数内のみで有効である．これを，**ローカル変数**と呼ぶ．また，関数の仮引数もローカル変数である．
- コードブロック—{ }で囲まれた部分[1]—の先頭で宣言した変数は，そのブロック内のみで有効になる．これもローカル変数のひとつであるが，ここでは**ブロック内宣言の変数**と呼ぶことにする．

スコープが異なれば，同じ名前の変数名を使うことができる．その場合，優先度の高い順に並べると，ブロック内宣言の変数 → ローカル変数 → グローバル変数となる．

[*]独立行政法人　秋田工業高等専門学校　電気情報工学科

[1]`if` 文やループの時使った．

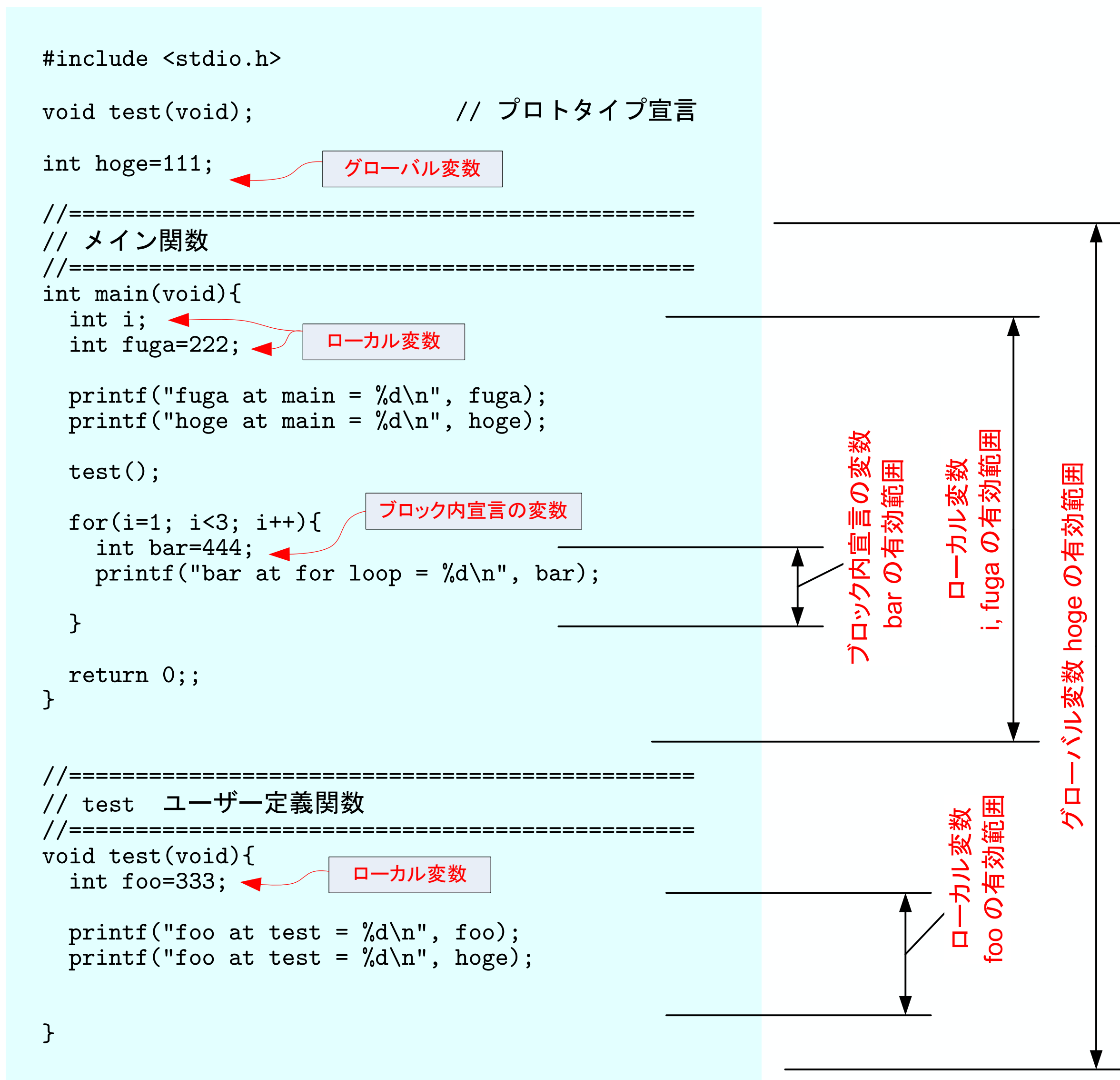

図 1: 変数のスコープ (有効範囲)

1.2 本日の学習内容

本日の学習範囲は教科書 [1] の p.189–202 である．教科書は，変数のスコープ (ローカル変数/グローバル変数) と記憶クラスの区別があいまいなところがある．少し分かり難いので，このプリントに記憶クラスをまとめる．変数のスコープに関しては，先週のプリントのとおり．

本日の学習のゴールは以下の通り．

- 変数のスコープの意味が分かる．
- 自動変数 (auto) と静的変数 (static) の違いがわかる．
- 分割コンパイルする場合のグローバル変数の取扱い方法が分かる．
- レジスター変数の利点と使い方が分かる．

2 記憶クラス

2.1 記憶クラスとは

C 記憶クラスは，メモリーにデータを格納する方法のことである．C 言語には，4 個の記憶クラス指定子 (1)auto，(2)extern，(3)register，(4)static がある．これらは，変数宣言の先頭に

```
auto int hogehoge;
extern int fugafuga;
register int foo;
static int bar;
```

と書く．

記憶クラスは 4 種類あるが，この中で static はローカル変数とグローバル変数では，動作が異なる．したがって，5 種類の動作があると考える．それぞれの動作について，次節以降に説明する．

2.2 記憶クラスの使い方

2.2.1 自動変数 (auto)

これは，自動変数と呼ばれるもので，諸君がいままで使ってきた変数である．変数宣言のとき記憶クラス指定子が無い場合，自動変数としてコンパイラーは取り扱う．したがって，プログラム中に

```
auto int hogehoge;
```

というように auto を書くことはまず無い．いままで多くのプログラムを見てきたが，わざわざ auto を記述しているプログラムを見たことがない．C 言語の前身である B 言語との互換性を保つために，現在では使われない auto がある．

自動変数は，それが宣言されている関数が呼び出されたときに記憶領域—メモリー— が確保されて，関数での処理が終了したら廃棄される．例えば，リスト 1 のように関数 hoge() を 3 回呼び出すプログラムの動作を考える．関数 hoge() の中の変数 a は自動変数である．ゆえに，関数 hoge() が呼び出される毎に記憶領域 a が初期化されて作成され，その関数の動作が終了する毎に廃棄が実行される．自動的に記憶領域の処理を行われるので,「自動変数 (auto)」と呼ばれる．

リスト 1: 自動変数の例.

```
1 #include <stdio.h>
2
3 int hoge(void);
```

```
4
5  //==========================================================
6  // メイン関数
7  //==========================================================
8  int main(void)
9  {
10   int i, foo;
11
12   for(i=1; i<=3; i++){
13     foo = hoge();
14     printf("%d\tfoo=%d\n", i, foo);
15   }
16
17   return 0;
18 }
19
20 //==========================================================
21 // 自動変数を使ったユーザー定義関数 hoge
22 //==========================================================
23 int hoge(void)
24 {
25   int a=0;                // 自動変数 auto int a=0 もOK
26
27   a++;                    // a = a+1 と同じ
28
29   return a;
30 }
```

このプログラムの実行結果は，以下のようになる．図 2 に関数呼出と変数 a の変化を示す．関数 hoge() 内の自動変数 a は関数が呼び出された時に作成され，関数での処理が終わった時点で消滅する．したがって，以下の結果が得られるのである．

実行結果

```
1       foo=1
2       foo=1
3       foo=1
```

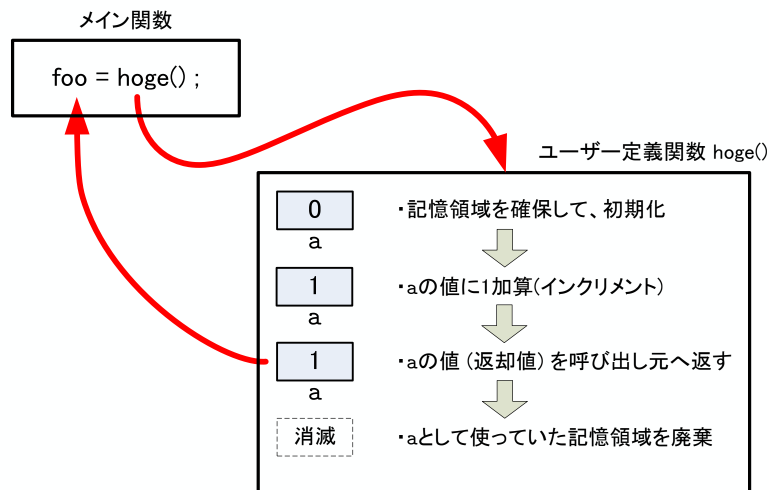

図 2: 関数 hoge() の動作と内部の自動変数 a の変化

2.2.2 静的変数 (static)

記憶クラスに静的変数を用いると，関数の処理が終了しても変数を破棄しないで保存することができる．そうすると，次にその関数を呼び出したとき，保存した変数を利用することが可能となる．次のように

```
static int hogehoge;
```

と書くことにより，変数 hogehoge を静的変数とすることができる．

グローバル変数でも，変数の値を保存することができる．しかし，グローバル変数は他の関数からもアクセスできるので，思わぬことで値が変更されることがある．したがって，変数の値を保存する必要があるが，他の関数がそれを使わない場合，静的変数を使う．

静的変数を使ったプログラムをリスト 2 に示す．このプログラムの 25 行目のみ，リスト 1 と異なる．すなわち，関数 hoge() の変数 a が静的変数になっている．

リスト 2: ローカル変数を静的変数として宣言.

```
1  #include <stdio.h>
2
3  int hoge(void);
4
5  //==========================================================
6  // メイン関数
7  //==========================================================
8  int main(void)
9  {
10   int i, foo;
11
12   for(i=1; i<=3; i++){
13     foo = hoge();
```

```
14       printf("%d\tfoo=%d\n", i, foo);
15     }
16
17     return 0;
18   }
19
20   //============================================================
21   // 静的変数を使ったユーザー定義関数 hoge
22   //============================================================
23   int hoge(void)
24   {
25     static int a=0;          // 静的変数
26
27     a++;                     // a = a+1 と同じ
28
29     return a;
30   }
```

このプログラムの実行結果は，以下のようになる．図 3 に 1 回目関数呼出と変数 a の変化を示す．関数 hoge() 内の静的変数 a はプログラム実行に先立って—メイン関数の実行よりも前に—静的変数 a は作成され，初期化 (a=0) が行われる．そして，プログラムが動作している間，その変数は保存される．したがって，以下の結果が得られるのである．

実行結果

```
1       foo=1
2       foo=2
3       foo=3
```

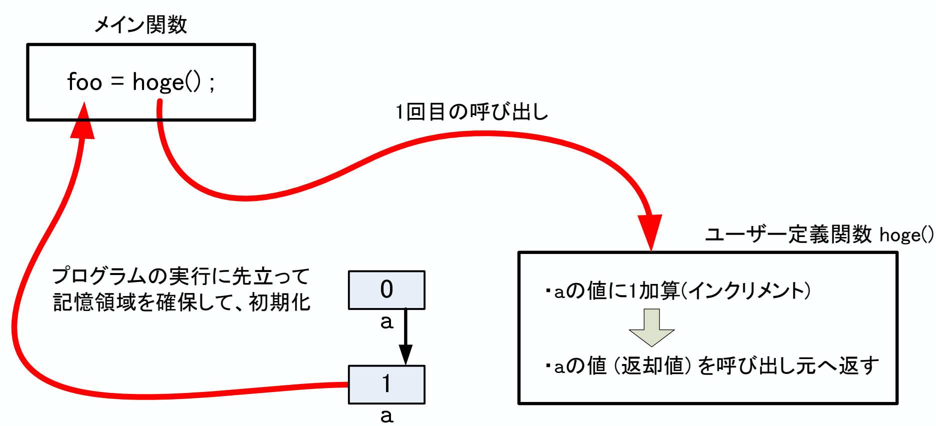

図 3: 関数 hoge() の動作と内部の静的変数 a の変化

2.2.3 複数のソースファイルと外部変数

複数のソースファイルのコンパイル C 言語では，複数のソースファイルからひとつの実行ファイルを作成する仕組みがあり，それを分割コンパイルと言う．これは，複数のプログラマーでひとつのプログラムを作成するときに便利である．例えば，A 君が file1.c を，B 君が file2.c を，C 君が file3.c を作成し，それをまとめてひとつの実行ファイル jikkou を作成することができる (図 4)．この仕組みは，一人で大規模なプログラムを作成するときにも使う．ひとつのファイルだと長すぎて，内容がわかり難くなれば，関連した関数をまとめてひとつにファイルにすることにより，管理が容易になる．

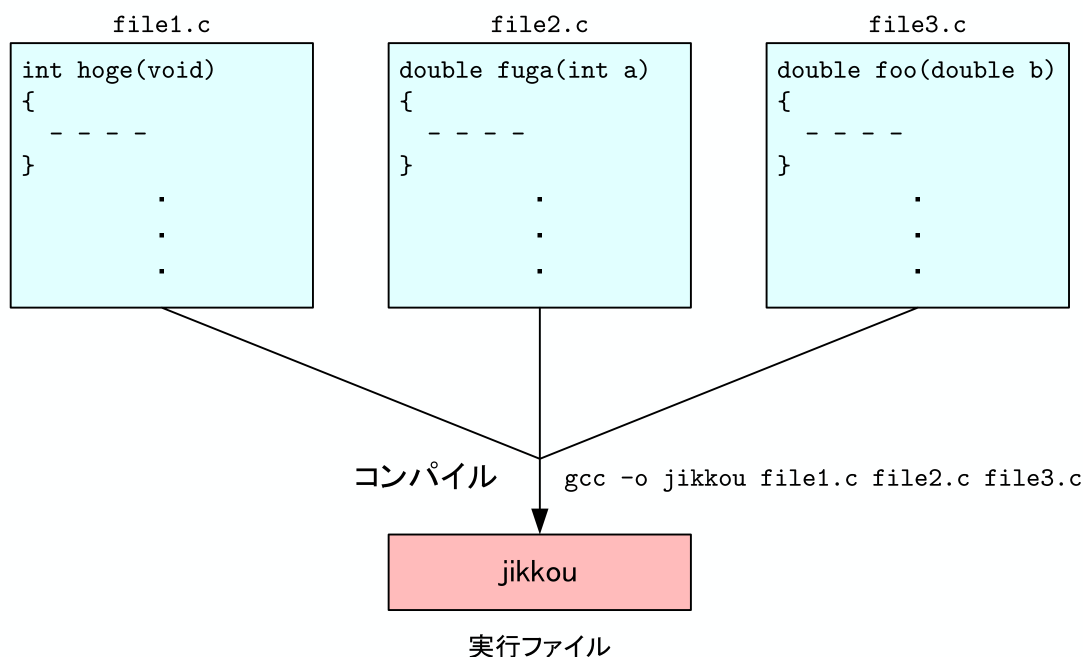

図 4: 複数のソースファイルからひとつの実行ファイルを作る．

グローバル変数の問題 複数のソースファイルからひとつの実行ファイルを作成する場合，全てのファイルで使いたいグローバル変数をどこで宣言するか?—という問題がある[2]．これは，C 言語では次のようにすることになっている．

- どのファイルで宣言しても良い．ただし，宣言はひとつのみ．
- 他のファイルで宣言されたグローバル変数を使う場合，記憶クラス指定子 `extern` を付ける．

リスト 3 と 4 に分割コンパイルで，共通のグローバル変数を使った例を示す．グローバル変数 `foo` は，file1.c でも file2.c でも使用できる同じ変数である．file2.c でグローバル変数の宣言を行い，file1.c では `extern` を付けることにより，どこか他の場所で宣言されていることを示している．

[2]グローバル変数の他に，プロトタイプ宣言も考えなくてはならない．その関数を使う全てのファイルで宣言が必要．

リスト 3: グローバル変数を外部変数として宣言 (file1.c).

```
1  #include <stdio.h>
2
3  extern int foo;         // どこかで宣言されているグローバル変数
4
5  int hoge(void);         // プロトタイプ宣言
6
7  //==========================================================
8  // メイン関数
9  //==========================================================
10 int main(void)
11 {
12   int i;
13
14   for(i=1; i<=3; i++){
15     hoge();
16     printf("%d\tfoo=%d\n", i, foo);
17   }
18
19   return 0;
20 }
```

リスト 4: グローバル変数を使ってデータを渡している (file2.c).

```
1  #include <stdio.h>
2
3  int foo=0;                   //   グローバルな自動変数
4
5  //==========================================================
6  // グローバルな自動変数の値を変えるユーザー定義関数 hoge
7  //==========================================================
8  void hoge(void)
9  {
10
11   foo++;                     // foo = foo+1 と同じ
12
13 }
```

複数のソースファイルから，ひとつの実行ファイルを作成する場合，コンパイルは次のようにする．

コンパイル方法

```
gcc -o jikkou file1.c file2.c
```

関数 `hoge()` 中の `a` は file1.c と file2.c で使えるグローバル変数なので，次の実行結果が得られることが理解できるであろう．

実行結果

```
1       foo=1
2       foo=2
3       foo=3
```

2.2.4 グローバル変数を静的変数と宣言

グローバル変数の仕組みが extern のみだと，名前の衝突の可能性がある．例えば，A 君は，file1.c のみで gvalue と言う変数名を使いたい．しかし，B 君と C 君は file2.c と file3.c で共通につかえるグローバル変数 gvalue を使いたい．同じ名前を使用するが，保存する内容が異なる場合を名前の衝突と言う．

これを解決するには，A 君の file1.c では，

```
static int gvalue;
```

とグローバル変数を静的に宣言する．こうすることにより，file.c でのグローバル変数 gvalue は,file1.c のみのグローバル変数と制限ができる．もちろん，file2.c や file3.c では

```
int gvalue;
extern int gvalue;
```

と宣言する．

グローバル変数を静的に宣言した例をリスト 6 の 3 行目に示す．この変数 foo とリスト 5 の 3 行目の foo は名前が同一でも全く異なるものである．リストリスト 5 と 6 からひとつの実行ファイルを作成しても，2 つの foo が保存する内容は別物である．

リスト 5: グローバル変数を静的変数と宣言 (file1.c).

```
1  #include <stdio.h>
2
3  static int foo=5;        // 静的なグローバル変数
4
5  int hoge(void);          // プロトタイプ宣言
6
7  //=========================================================
8  // メイン関数
9  //=========================================================
10 int main(void)
11 {
12   int i;
13
14   for(i=1; i<=3; i++){
15     hoge();
16     printf("%d\tfoo=%d\n", i, foo);
17   }
18
19   return 0;
20 }
```

リスト 6: グローバル変数を静的変数と宣言 (file2.c).

```
1  #include <stdio.h>
2
3  static int foo=0;                  //  グローバルな静的変数
4
5  //=========================================================
6  // グローバルな自動変数の値を変えるユーザー定義関数 hoge
7  //=========================================================
8  void hoge(void)
9  {
10
11   foo++;                    // foo = foo+1 と同じ
```

```
12 
13 }
```

静的なグローバル変数が理解できたならば，次の実行結果は納得できるであろう．リスト 5 の 16 行目の `foo` は，file1.c のグローバル変数の `foo` のことである．

実行結果

```
1	foo=5
2	foo=5
3	foo=5
```

2.2.5　レジスター変数

コンピューターの仕組み　コンピューターは，図 5 のようになっている．メモリはデータやプログラムを記憶する装置である．そして，CPU はデータを処理—計算 (演算)—する装置である．このメモリーと CPU の間で猛烈な勢いで，データの受渡しを行うことにより，情報を処理する．CPU の中には，演算装置とレジスターがある．CPU がメモリーから受け取るデータは，いったんレジスターに格納する．レジスターに格納したデータを演算装置が処理するのである．レジスターとメモリーは，どちらもデータを格納するが，以下のような違いがある．

- メモリーは大量のデータを記憶できるが，演算装置とのアクセスが遅い．
- レジスターは記憶容量は小さいが，演算装置とのアクセスが早い．

レジスターは CPU の中にある小さいメモリーのことである．

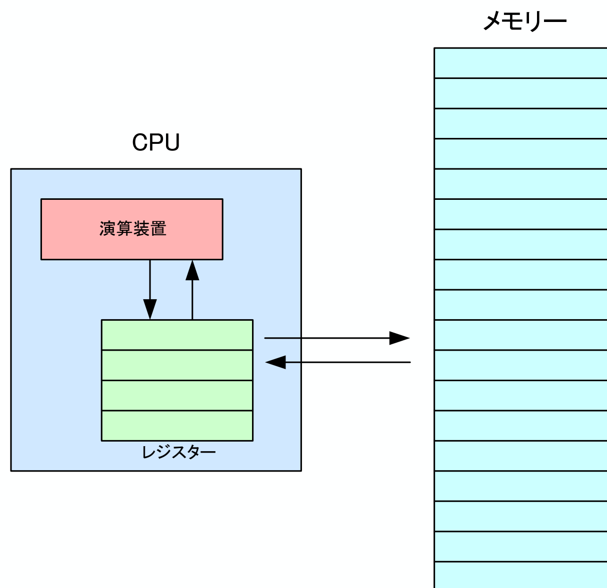

図 5: コンピューターの原理的な仕組み．矢印はデータの流れを表す．

レジスター　レジスター変数を使えば，メモリーを使うより高速に計算ができる．レジスター変数は

```
register int rgsval;
```

と宣言する．これにより，変数 regval はメモリーではなく，レジスターに確保できる．ただ，レジスター変数は多くないので，それによる計算の高速化には限界がある．もし，使用できる以上のレジスター変数を宣言した場合，メモリーに変数を割り当てるようにコンパイラーが勝手に実行ファイルを作成する．

一般に，アクセスの多い変数をレジスターに割り付ける．例えば，ループ文の制御変数である．リスト 7 の例では，制御変数の i と j，計算結果を格納する変数 k にレジスター変数を割り当てている．

リスト 7: ローカル変数をレジスター変数として宣言．

```
1  #include <stdio.h>
2
3  //============================================================
4  // メイン関数
5  //============================================================
6  int main(void)
7  {
8    register int i, j, k=0;
9
10   for(i=1; i<=10000; i++){
11     for(j=1; j<=10000; j++){
12       k=i+j;
13     }
14   }
15
16   return 0;
```

```
17 }
```

次のようにすると，プログラムの実行時間が分かる．ただし，リスト 7 をコンパイルした実行ファイルを test_reg とする．

実行時間を計測する方法

```
time ./test_reg
```

実行結果は，次のようになる．最初の real が実際にプログラムの実行に要した時間である．

実行結果

```
real    0m0.110s
user    0m0.108s
sys     0m0.000s
```

レジスター変数を使わない場合，real の時間は 0.227 秒であった．レジスター変数を使うと倍の速度で計算できることが分かる．

3 プログラム作成の練習

[**練習 1**] 次のプロトタイプをもつユーザー定義関数を作成する．

```
void sum_arg(int a);
```

これは仮引数の値を加算し，表示する関数とする．たとえば，この関数を引数，2, 9, 5 を伴って三回呼び出したとすると，2, 11, 16 を表示する．

メイン関数のループ文で，キーボードから値を読み取り，この関数を呼出し，これまでの合計を計算するプログラムを作成せよ．ループの回数は 5 回とする．プログラムは，次の 2 種類のものを作成する．

- 静的なローカル変数を使う．
- グローバル変数を使う．

[**練習 2**] 2 個のソースファイルから 1 つの実行ファイルを作成する練習を行う．file1.c は，メイン関数で次の動作を行う．

- キーボードから 2 つの数値 a，b(倍精度実数) を読み込む．
- file2.c に書かれた関数を

  ```
  kansu(a, b);
  ```

  と呼び出す．
- グローバル変数 x，y の値を表示する．
- a の値が正の場合，この処理を繰り返す．

file2.c には，`void kansu(double a, double b)` の処理を書く．この関数は，グローバル変数 x と y の計算を行う．

$$\mathrm{x} = \mathrm{a}^{\mathrm{b}} \qquad\qquad \mathrm{y} = \log_{\mathrm{a}} \mathrm{x} = \frac{\log_{10} \mathrm{x}}{\log_{10} \mathrm{a}} \tag{1}$$

C 言語でこの数学関数については，教科書の p.127 を見よ．`#include math.h` の記述とコンパイルの時`-lm` を忘れないと．

［**練習 3**］ レジスター変数を使っているリスト 7 のプログラムの実行時間を測定せよ．そして，レジスター変数を使わない場合の実行時間と比較せよ．

4 課題

次回の講義の日 (11 月 24 日) の AM8:45 までに，以下の課題をレポートとして提出すること．表紙等は，いつもの通り．表紙のタイトルは「記憶クラス」とすること．

［**問 1**］ (復) 教科書の p.162–202 を 3 回読め．重要なところには，赤線あるいは蛍光ペンで印を付けよ．不明な点があれば，クラスの者に聞け．それでも分からない場合は，Office hours を利用して私 (山本) に質問しろ．

［**問 2**］ 本日，配布したプリントを 3 回読め．

［**問 3**］ (復)4 つの記憶クラス (`auto`，`static`，`extern`，`register`) について，その内容を表にまとめよ．

参考文献

[1] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　基礎編　第 2 版. (株) オーム社, 2006.