プログラミング演習 I 課題 2 テキスト

10 進数と 2 進数

—チェックディジットと全加算器—

http://www.ced.is.utsunomiya-u.ac.jp/lecture/2014/prog/p1/kadai2/

担当：篠田 一馬
しのだ かずま
shinoda@is.utsunomiya-u.ac.jp
028-689-6267
工学部・情報工学科棟 5F
9-510

課題概要

C 言語のプログラムは関数の集まりである．main も関数である．したがって，関数が一つもないプログラムはあり得ない．課題 2 では，関数の定義の仕方や使い方，標準関数を利用する仕組みを学ぶ．そのための題材として，整数から 10 進数や 2 進数の各桁の数を取り出す方法，学籍番号のチェックディジット生成，2 進数での加算を取り上げ，これら基本的なアルゴリズムについても学ぶ．課題は 5 つある．

10進：広辞苑によると「十進」は「じっしん」と読む．他の辞書では「じゅっしん」という読み方も併記されている．

課題 (2-1) 学籍番号(整数)の入力に対し，1 の位～100,000 の位の数（10 進数 1 桁目～6 桁目の数）を求め，出力せよ．

課題 (2-2) 課題(2-1)のプログラムに，各桁の数から学籍番号のチェックディジットを求める関数を加え，学籍番号と共にチェックディジットを出力するようにせよ．

課題 (2-3) 0～255 の整数の入力に対し，その 2 進数表現を求め，出力せよ．

課題 (2-4) 0/1 による 8 桁の 2 進数入力に対して，それが表す整数値を出力せよ．このとき，8 桁の 2 進数からそれが表す整数を求める関数を作成・利用すること．

課題 (2-5) 0/1 による 8 桁の 2 つの 2 進数入力に対して，それらの和を 2 進表現で求め，その 2 進表現を出力せよ．このとき，2 ビットの論理和，論理積，排他的論理和を求める関数，および，これらを用いて 2 進数 1 桁の加算を行う全加算器の機能を関数で実現すること．なお，確認を容易にするため，入力と出力の各 2 進数に対する 10 進表現も出力すること．

各課題のプログラムは保存しておくこと

各課題毎にプログラムを作成すること．たとえば課題(2-2)は，課題(2-1)で作成したプログラムに機能を追加する内容である．しかし，課題(2-1)で作成したソースコードを直接編集してしまうと，課題(2-1)で作成したソースコードが失われてしまう．課題(2-1)で作成したソースコードは変更しないで，その内容を課題(2-2)のソースコードにコピーしてから編集するようにすること．

ソースコード：プログラミング言語を用いて記述したプログラム．ソースプログラムとも言う．

日程

1 日目は，課題(2-2)まで行い，TA が出題した学籍番号について，課題(2-2)のプログラムを用いてそのチェックディジットを解答し，チェックを受けること．
2 日目は，課題(2-4)まで行い，課題(2-3)と課題(2-4)のソースリスト，および，5 個の 0～255 の整数での検証結果を TA に提示し，チェックを受けること．
3 日目は，課題(2-5)を終わらせ，レポート作成に取り組む．

ソースリスト：文書中に示されたひとかたまりのソースコード．

課題 (2-1)，(2-2) の前に

以下の計算をしてみよ．

(1) 下の学籍番号欄にあなたの学籍番号を記入する．
(2) 学籍番号の各桁にその下の数を掛ける．
(3) それらをすべて足し合わせる．
(4) 足し合わせた結果から 0～10 を引いて 11 の倍数（66，77，88 など）になるようにする．
(5) このとき引いた数を変換表で文字に変換する．

その文字があなたの学籍番号のチェックディジットである．合っているかな？
この計算を行うプログラムの作成が課題(2-1)，(2-2)である．

変換表

0→A	
1→A	
2→Z	
3→Y	
4→X	
5→U	
6→M	
7→K	
8→H	
9→C	
10→B	

チェックディジット → □

学籍番号： □ □ □ □ □ □

× 7 × 6 × 5 × 4 × 3 × 2

↓ ↓ ↓ ↓ ↓ ↓

□ + □ + □ + □ + □ + □ ⇒ □ − □ = 11 の倍数

↑

チェックディジット： p. 6 参照．

課題(2-1) 10 進 1 桁目～6 桁目の抽出

学籍番号(整数)の入力に対し，1 の位～100,000 の位の数（10 進数 1 桁目～6 桁目の数）を求め，出力せよ．たとえば，学籍番号として「092900」とタイプして，これを scanf 関数を用いて int 型変数 n に読み込むと，n には整数 92900 が入る．このとき本課題では，整数 92900 の 6 桁目は「なし」ではなく 0 であると考えよ．

プログラム作成上の指針

(1) 変数宣言

次の課題(2-2)で，整数の各桁の数を使って計算を行うので，計算で求める各桁の数は別々の変数に格納するようにせよ．そのための 6 個の変数が必要となる．変数名は自分で決めてよい．以下の説明では，100,000 の位，10,000 の位，1,000 の位，100 の位，10 の位，1 の位の数を入れる変数をそれぞれ，

```
int a5,a4,a3,a2,a1,a0;
```

と宣言したとして説明する．また，入力された学籍番号を格納するための変数を

```
int n;
```

と宣言したとして説明する．

(2) 学籍番号の入力

int 型変数 n に対して，

```
scanf("%d",&n);
```

で整数値を入力する．

(3) 各桁の数の抽出

たとえば，整数 n の 1000 の位の数 a3 を求めるには，整数の割り算「/」と整数の割り算の余り「%」を用いて，

```
a3 = (n / 1000) % 10;
```

という計算（n を 1000 で整数除算し，さらに，それを 10 で割った余りを求める）を行う．100 の位の数 a2 は，

```
a2 = (n / 100) % 10;
```

で計算する．

(4) 出力

求めた 6 個の数を出力するとき，

```
printf("%d %d %d %d %d %d¥n",a5,a4,a3,a2,a1,a0);
```

により，たとえば学籍番号として「092900」を入力した場合，「0 9 2 9 0 0」のように各桁を 1 個の空白で区切って出力する．

課題(2-2) チェックディジット生成関数

課題(2-1)のプログラムに，各桁の数から学籍番号のチェックディジットを求める関数を加え，学籍番号と共にチェックディジットを出力するようにせよ．

チェックディジット生成法

6 桁の学籍番号「a5 a4 a3 a2 a1 a0」からチェックディジット c を求めるには，a5～a0 の重み付き和が 11 で割り切れるように補正する数を求めればよい．具体的な方法は幾つか考えられる．プログラムが比較的簡単なのは以下の方法である．a5～a0 の重み付き和を 11 で割った余り m を

```
m = (7*a5 + 6*a4 + 5*a3 + 4*a2 + 3*a1 + 2*a0) % 11
```

のように求める．そして，以下の対応関係でチェックディジットを決定する．

m = 0 および 1 のとき，チェックディジットは A	m = 6 のとき，チェックディジットは M
m = 2 のとき，チェックディジットは Z	m = 7 のとき，チェックディジットは K
m = 3 のとき，チェックディジットは Y	m = 8 のとき，チェックディジットは H
m = 4 のとき，チェックディジットは X	m = 9 のとき，チェックディジットは C
m = 5 のとき，チェックディジットは U	m =10 のとき，チェックディジットは B

チェックディジット 1 文字を格納する char 型変数を cd とすると，m の値から定まるチェックディジットを cd に格納するプログラム片は，if 文を用いると，

char 型:p.6 参照

```
if ((m == 0) || (m == 1))
    cd = 'A';
else if (m == 2)
```

```
        cd = 'Z';
    else if (m == 3)
    ・・・・・・・・・
    else if (m == 10)
        cd = 'B';
```
のようになる．if・・・else if・・・の代わりに switch 文を使ってもよい．

チェックディジット生成関数

6 桁の学籍番号に対応した整数 a5，a4，a3，a2，a1，a0 からチェックディジットを求める関数は典型的には以下のように定義する．

関数定義:pp. 7-8 参照

```
char check_digit(int a5,int a4,int a3,int a2,int a1,int a0)          リスト 1
{
    char cd;
    int m;
    ・・・・・
    a5～a0 からチェックディジットを求め，それを変数 cd にセットする．
    ・・・・・
    return cd;
}
```

ここで，

・check_digit はプログラマが適当に定める関数名

・a5，a4，a3，a2，a1，a0 は関数に与える int 型パラメータ（仮引数（ひきすう））

・関数は char 型の値を返す

・cd は，char 型のローカル変数

ローカル変数:p. 9 参照

プログラムができたら，まわりの人から学籍番号とチェックディジットを教えてもらい，検証すること．

課題(2-3)，(2-4)，(2-5)の前に

右の 2 進数の足し算を筆算でしてみよ．$(01011)_2=(11)_{10}$，$(00101)_2=5$，よって和は$(16)_{10}$すなわち$(10000)_2$と答を出すのではない．1＋1 は$(10)_2$だから，その桁の答は 0 で，1 繰り上がるというように計算すること．CPU はこのような計算を論理回路で行っている．その手順をプログラムとして記述するのが課題(2-5)である．課題(2-3)は，整数の 2 進表現の各桁を求めるためのプログラム，課題(2-4)は逆に 2 進表現を整数に変換するためのプログラムで，いずれも，課題(2-5)に必要である．CPU が持っている機能の一部をプログラムとして実現し，これを CPU に実行させるのは無駄だと思うかもしれない．しかし，論理回路の動作確認をソフトウェアで行うことは，論理回路のハードウェアを試作することなしにその動作確認ができるため，ハードウェアの設計製作には不可欠な技術である．

	0	1	0	1	1
+)	0	0	1	0	1

課題(2-3) 整数 0～255 の 2 進数表現への変換

0～255 の整数の入力に対し，その 2 進数表現を求め，これを 10 進表現とともに出力せよ．
課題(2-1)のプログラムは 10 進での各桁の数を求めるものであった．この課題は，課題(2-1)の 10 進数が 2 進数になったものと考えるだけで十分である．具体的には，整数 N を 2 進数で表現するとき，最下位桁を 0 桁目として，i 桁目の値 a_i は，$a_i = (N / 2^i) \% 2$　で計算できる．

整数 0～255 を扱うには 2 進数で 8 桁必要となることに注意せよ．出力は，例えば入力整数が 129 のとき，「1 0 0 0 0 0 0 1 (129)」のような形式とすること．

課題(2-4) 2 進数表現の整数への変換

0/1 による 8 桁の 2 進数入力に対して，それが表す整数値を出力せよ．このとき，8 桁の 2 進数からそれが表す整数を求める関数を作成・利用すること．

8 桁の 2 進数を格納する変数が

```
    int a7,a6,a5,a4,a3,a2,a1,a0;
```

と宣言されているものとする．8 桁の 2 進数入力では，各桁を空白で区切って入力させ，それを

```
    scanf("%d %d %d %d %d %d %d %d",&a7,&a6,&a5,&a4,&a3,&a2,&a1,&a0);
```

で読み取るようにする．

8 桁の 2 進数からそれが表す整数を求める関数は，典型的には以下のように定義する．

```
int bin_to_int(int a7, int a6, int a5, int a4, int a3, int a2, int a1, int a0)
{
    ・・・・・
}
```
リスト 2

ここで，

・bin_to_int はプログラマが適当に定める関数名

・a7，a6，a5，a4，a3，a2，a1，a0 は関数に与える int 型パラメータ（仮引数（ひきすう））

・関数は int 型の値を返す

この関数で，8 桁の 2 進数(a7 a6 a5 a4 a3 a2 a1 a0)が表す整数 N は以下の式で計算でする．

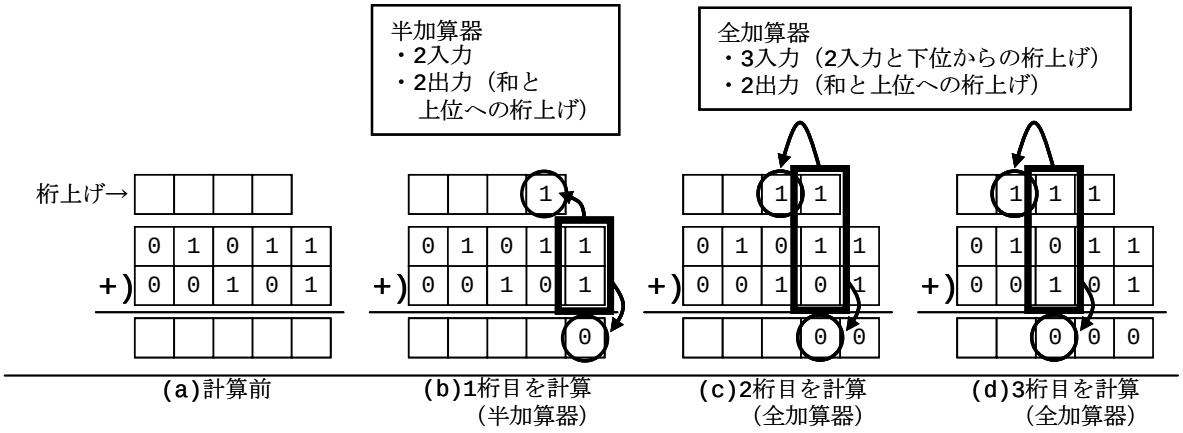
$N = a7*128 + a6*64 + a5*32 + a4*16 + a3*8 + a2*4 + a1*2 + a0$

課題(2-5) 全加算器による 8 桁の 2 進数の和

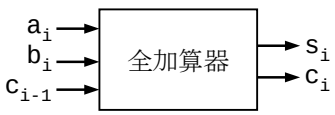
0/1 による 8 桁の 2 つの 2 進数入力に対して，それらの和を 2 進表現で求め，その 2 進表現を出力せよ．このとき，2 ビットの論理和，論理積，排他的論理和を求める関数，および，これらを用いて 2 進数 1 桁の加算を行う全加算器の機能を関数で実現すること．なお，確認を容易にするため，入力と出力の各 2 進数に対する 10 進表現も「1 0 0 0 0 0 0 0 1 (129)」の形式で出力すること．

全加算器の解説

詳しくは，「論理設計とスイッチング理論 I」で学ぶ．ここでは，概略のみ述べる．まず，小学校での筆算による足し算を思い出してみよう．下位の桁から順次上の桁へと計算していく方法である．2 つの 2 進数の和は，これと全く同じ原理で行う．下図(b)の 1 桁目では，2 ビットの 2 を法とした加算 1+1=0 を行い，上位の桁に 1 を繰り上げる．すると，下図(c)や(d)のように上位の桁では，3 ビットの和を計算することになる．下図(b)のように，2 ビットを入力し，それらの和と桁上げを出力する回路を半加算器，下図(c)や(d)のように，3 ビットを入力し，それらの和と桁上げを出力する回路を全加算器という．



2 進数($a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$) と ($b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$) の加算を行うことを考えよう．i 桁目で，下位からの桁上げ c_{i-1} とその桁の a_i と b_i の計 3 ビットの入力に対して，2 を法とした和 s_i と上位への桁上げ c_i の計 2 ビットの出力を計算する全加算器の真理値表と各出力の論理式を以下に示す．



全加算器の真理値表

入力 a_i	b_i	c_{i-1}	出力 s_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

和 s_i と上位への桁上げ c_i の論理式

$s_i = a_i \oplus b_i \oplus c_{i-1}$

$c_i = a_i \cdot b_i + b_i \cdot c_{i-1} + c_{i-1} \cdot a_i$

$= a_i \cdot b_i + (a_i \oplus b_i) \cdot c_{i-1}$

演算子の意味

⊕:排他的論理和

+:論理和

・:論理積

この全加算器による計算を 1 桁目から，順次，上位の桁へと行うことにより，2 進数の加算を行う．ただし，最下位桁 i=0 での下位からの桁上げ $c_{0-1} = c_{-1}$ は 0 とする．

プログラム作成上の指針

(1) 論理和，論理積，排他的論理和を求める関数 or，and，xor

2 ビットの論理和を求める関数 or は，典型的には以下のように定義する．ただし，これ以外の方法も考えられる．論理積 and，排他的論理和 xor についても同様に定義せよ．

これ以外の方法：

(1) a+b が 0 のとき 0，それ以外では 1.

(2) a|b（ビット単位の OR）.

他にもあるかな？

リスト 3

```
int or(int a,int b)
{
    if ((a == 1) || (b == 1))
        return 1;
    else
        return 0;
}
```

(2) 全加算器の実装方法

全加算器の機能を関数として実現するとき，全加算器の出力が和と上位への桁上げの計 2 つであることに注意すると，全加算器の関数は典型的には以下のように定義する．

リスト 4

```
void full_adder(int a,int b,int cin,int *s,int *cout)
{
    *s    = .......; /* 和を求める論理式 */
    *cout = .......; /* 上位への桁上げを求める論理式 */
}
```

論理式は，例えば a⊕b⊕c は，xor(xor(a,b),c)あるいは xor(a,xor(b,c))という関数呼び出しで計算できる．

(3) 関数 full_adder の使い方

8 桁の 2 進数と各桁の桁上げを格納する変数として，

```
int a7,a6,a5,a4,a3,a2,a1,a0;
int b7,b6,b5,b4,b3,b2,b1,b0;
int s7,s6,s5,s4,s3,s2,s1,s0;
int c7,c6,c5,c4,c3,c2,c1,c0;
```

のように宣言したとすると，まず，a7～a0 と b7～b0 に課題(2-4)のようにして，2 つの 8 桁の 2 進数を入力した後，下位の桁から以下のようにして和 s0～s7 を求めて行く．

```
full_adder(a0,b0,  0,&s0,&c0);
full_adder(a1,b1,c0,&s1,&c1);
full_adder(a2,b2,c1,&s2,&c2);
・・・・・・・・
```

関数 full_adder の第 3 パラメータ（桁上げ）は，最下位桁のときのみ数値 0 を与え，それ以降は直前の計算で求めた桁上げの結果を与えていることに注意せよ．

解説

☆：　基本事項．必ず理解すること．
☆☆：　中級．知っていると自慢できる．
☆☆☆：上級．将来必ず役に立つので，少なくとも一度は読んでおくこと．

チェックディジット　☆☆

チェックディジット（check digit）とは，本学の学籍番号ではその末尾にある A～Z の英字のことである．これを付加することにより，学籍番号の入力誤り（タイプ誤りや手書きの学籍番号の判読誤り）の検出を行うことができる．ただし，自動的に誤りを訂正することはできない．自動車免許証，各種カード，バーコード，図書の ISBN など，多桁の番号にはすべてチェックディジットが付加されている．本学の学籍番号のチェックディジットは，モジュラス 11（modulus 11，modulo 11，mod 11）ウエイト 2～7 と呼ばれる方式である．モジュラスとは数学で「法」という意味で，たとえば，2 を法とした加算は 0+0=0，0+1=1，1+0=1，1+1=0，つまり，整数加算結果を 2 で割って余りを求める(2 での剰余を求める)ということである．チェックディジットの世界でモジュラス 11 とは，各桁の和を 11 での剰余でチェックするということである．ウエイト 2～7 とは，各桁の和を求めるとき，2 から 7 の重み（ウェイト weight）を付けることを意味する．具体的には，6 桁の学籍番号とチェックディジットからなる 7 桁の数列「n5 n4 n3 n2 n1 n0 c」について次のような重み付き和
N = 7*n5 ＋ 6*n4 ＋ 5*n3 ＋ 4*n2 ＋ 3*n1 ＋ 2*n0 ＋ c
を求める．ただし、「c」は，チェックディジットが A，B，C，H，K，M，U，X，Y，Z の場合は，それぞれ，0 または 10，1，2，3，4，5，6，7，8，9 の値をとるとみなす．この計算結果Ｎが 11 で割り切れない場合、当該データは誤りと判定する．ただし，チェックディジットが「A」の場合は，c=0 の場合と c=10 の場合の両方でＮが 11 で割り切れないとき当該データを誤りと判定することになる．

char 型　☆

文字を格納するための変数は，通常，char 型で宣言する．char 型とは，文字のみを対象とする型ではなく，文字コードが格納できる 1 バイト（8 ビット）の整数型のことである．したがって，char 型に格納できる整数は-128～127 となる．char c; と宣言した変数 c に文字 A（正確には，文字 A の文字コード）を代入するには，文　c = 'A'; とする．ここで'A'は，文字 A の文字コードを表す整数である．文字をシングルクォート「'」で囲むことに注意すること．間違えてダブルクォート「"」を使うと，それは文字列と解釈されてしまう．

宣言と定義の違い　☆☆☆

C 言語では一般に，「宣言（declaration)」は利用する対象の性質をコンパイラに指示することを意味し，「定義（definition)」は対象の性質を指示すると同時にメモリへの割り付けもコンパイラに指示するというように使い分けている（Pascal ではちょっと違って，「定義」するのは定数と型のみであった)．簡単に言うと，次のようになる．

コンパイラ:p. 11 参照

　宣言：こういうものを「使うのでよろしく」とコンパイラに知らせる．
　定義：こういうものを「メモリに配置しろ」とコンパイラに命じる．

最初のうちはあまり気にしなくてもよいが，関数の定義，関数の宣言（プロトタイプ宣言)，変数の定義，変数の宣言などという表現が出てきたら，両者の違いを考えてみること．また，自分で文章を書くときも注意して使い分けること．

命名法　☆☆

変数や関数，定数等を定義・宣言するとき，その識別子として適切な名前をつける必要がある．ループカウンタとして使う int 型のローカル変数名には，i，j，k など短い名前を用いるのが一般的である．一方，重要な役割を持つ変数や関数には，その役割が分かるような名前をつける．例えば，チェックディジットを格納する文字型変数であれば，リスト１のように，「cd」と短い名前を付ける以外に，check_digit，checkDigit，CheckDigit，ch_check_digit，chCheckDigit などが考えられる．最後の 2 つは，変数の型である文字型を表す ch を先頭に付けている．いずれにしても，プログラムを作った数ヶ月後に自分で読んで意味がわかるよう，命名してほしい．google で「変数　命名規則」などのキーワードで検索し，どのような命名法が使われているかを調べてみよ．

関数定義を記述する位置　☆

ソースコード中で関数を定義する位置は，
・その関数を呼び出す関数よりも前（リスト 5）
・その関数を呼び出す関数よりも後（リスト 6）
のどちらでもよい．ただし，後者の場合は，プロトタイプ宣言が必要となる．要するに，ソースコード中に関数呼び出しが現れる前に（使う前に），その関数がどんな関数か（そのパラメータと戻り値）を定義あるいは宣言しておく必要がある．これは，コンパイラがソースコードを先頭から順に読みながら機械コードに翻訳していくためである．

関数の定義の仕方：p. 8 参照

コンパイラ：p. 11 参照

```
/* リスト 5　使う前に関数を定義 */
#include <stdio.h>

int mul(int a,int b)
{
    return a*b;
}

int main(void)
{
    printf("%d¥n",mul(10,20));
    return 0;
}
```

```
/* リスト 6　後で関数を定義 */
#include <stdio.h>

int mul(int a,int b); /*プロトタイプ宣言*/

int main(void)
{
    printf("%d¥n",mul(10,20));
    return 0;
}

int mul(int a,int b)
{
    return a*b;
}
```

関数のプロトタイプ宣言は，関数定義の先頭の宣言部分をコピーすればよい．但し，末尾のセミコロン；を忘れないように注意すること．

ヘッダーファイル「～.h」の役割　☆☆

関数は使う前に定義／宣言しなければならないことはわかったと思う．では，リスト 5 やリスト 6 で使っている関数 printf はどこで宣言しているのだろうか？

先頭の「#include <stdio.h>」がその答である．これは，その位置で，ファイル stdio.h の内容全体を展開することをコンパイラに指示するものである．つまり，プログラムが 10 行のソースコードであっても，その中に「#include <stdio.h>」があると，コンパイラは stdio.h の内容（738 行）も含めた形で計 748 行のソースコードをコンパイルすることになる．

ファイル stdio.h
　C:¥Program Files¥Microsoft Visual Studio 10.0¥VC¥include¥stdio.h
には，標準入出力ルーチンのための定義と宣言が記述されており，関数 printf のプロトタイプ宣言もある．このような役割を担うファイルをヘッダーファイルと呼び，ヘッダーファイル名の拡張子は「h」とすることとなっている．C 言語の標準ライブラリに含まれる printf のような関数を使うときは，その関数に対応する適切なヘッダーファイルを include しなければならない．たとえば，数学関数 sin を使うときは，プログラムの先頭に「#include <math.h>」と書くことが必要になる．どのヘッダーファイルを include しなければならないかはオンラインマニュアルを参照のこと．

関数定義の仕方　☆

例として，足し算を行う関数 add を示し，関数定義の仕方の簡単な解説をする．

```
/* リスト 7 */
#include <stdio.h>

int add(int a,int b)
{
    return a + b;
}

int main(void)
{
    printf("%d¥n",add(123,456));
    return 0;
}
```

以下は，関数の一般形である．リスト 7 の関数 add や main と比べてみよ．

```
戻り値の型　関数名(パラメータ宣言)
{
    変数宣言等
    文
}
```

リスト 7 は，足し算関数 add を戻り値がある関数として定義した例である．return 文は，処理結果（この場合は足し算）を関数値としてセットし，関数を終える命令である．

```
/* リスト 8 */
#include <stdio.h>

int g_result;

void add(int a,int b)
{
    g_result = a + b;
}

int main(void)
{
    add(123,456);
    printf("%d¥n",g_result);
    return 0;
}
```

リスト 8 は，足し算関数 add を戻り値がない関数として定義した例である．戻り値がない場合，関数定義での戻り値の型は void とする．足し算結果を呼び出し側が知るため，グローバル変数 g_result を定義している．

```
/* リスト 9 */
#include <stdio.h>

int g_operand1,g_operand2,g_result;

void add(void)
{
    g_result = g_operand1 + g_operand2;
}

int main(void)
{
    g_operand1 = 123;
    g_operand2 = 456;
    add();
    printf("%d¥n",g_result);
    return 0;
}
```

リスト 9 は，足し算関数 add を戻り値がなく，パラメータもない関数として定義した例である．パラメータがない場合，パラメータ宣言は void とする．足し算する数を関数に伝え，足し算結果を呼び出し側が知るため，3 つのグローバル変数を定義している．

変数の種類とスコープ，寿命　☆

変数のスコープ（scope，有効範囲）とは，宣言された変数を直接アクセスできるプログラム内の範囲のことである．変数の寿命とは，その変数用のメモリ領域が確保されている期間のことである．スコープと寿命の観点から変数は 3 種類に大別できる．

(1) グローバル変数：main やその他の関数の外側で定義・宣言した変数．スコープは定義・宣言の後の全体となる．すなわち，定義・宣言の後はどこからでもアクセスできる．寿命はプログラムの開始から終了までである．

(2) 動的ローカル変数：自動変数とも言う．関数などのブロック内で宣言した変数．スコープはそのブロック内に限られ，寿命はそのブロックの開始から終了までである．そのブロックの実行が終わると，動的ローカル変数のための領域は破棄（システムに返却）される．単にローカル変数と言うときは，動的ローカル変数をさす場合が多い．

(3) 静的ローカル変数：関数などのブロック内で宣言した変数で，static 記憶クラス指定子をつけたものをさす．スコープはそのブロック内に限られるが，寿命はプログラムの開始から終了までである．何度も呼び出される関数で，課題(2-5)のリスト 4 のように，値を継続して持ち続けたい場合に用いる．

以下に，ローカル変数（動的ローカル変数），グローバル変数の例を示す．

```
/* リスト 10 */
#include <stdio.h>

int g_base;

int base_to_power(int n)
{
    int p;

    p = 1;
    while (n > 0) {
        p = p * g_base;
        n = n - 1;
    }
    return p;
}

int main(void)
{
    int n;

    g_base = 2;
    n = 16;
    printf("%d to power %d = %d¥n",
        g_base,n,base_to_power(n));
    return 0;
}
```

リスト 10 では，グローバル変数 g_base を基底としたべき乗を計算する関数 base_to_power を定義している．このプログラムを実行すると，「2 to power 16 = 65536」と表示される．リスト 10 を例として，各変数のスコープを見ていく．

・グローバル変数 g_base
プログラムのどこからでもアクセス可能．

・関数 base_to_power のローカル変数 p
関数 base_to_power 内からのみ直接*にアクセス可能．

・関数 main のローカル変数 n
関数 main 内からのみ直接*にアクセス可能．

・関数 base_to_power のパラメータ n
関数 base_to_power 内からのみアクセス可能．関数のパラメータは，その関数内では，呼び出し側で初期値を与えたローカル変数と考えることができる．関数のパラメータの内容を関数内で変更しても，その変更は，呼び出し側には影響しない．このパラメータ n の内容も，関数実行時に変更されるが，その変更は関数 main のローカル変数 n には影響しない．

＊ローカル変数の項目で「直接に」と断っているのは，「間接的に」は関数の外からアクセスする方法があるからである．

変数利用の原則として，
・ループ変数（たとえば，for (i = 0;i < 100;i++){...}での変数 i）は必ずローカル変数とする．
・なるべくローカル変数とし，グローバル変数の利用は必要最小限とすること．

変数名の命名法として，グローバル変数は，上の例のように先頭に「g_」を付けるなどして，ローカル変数と区別できるようにすること．

命名法:p.6 参照

main も関数．では誰が呼び出すの？ ☆☆☆

実行ファイル（exe ファイル）は．C runtime（CRT）スタートアップルーチンから実行が開始するようになっている．スタートアップルーチンは各種の初期化を行った後に，関数 main を呼び出すようなプログラムである．そのため，C 言語プログラムは，最初に実行する関数として main を必要とする．

C プログラムのメモリ領域　☆☆☆

ソースコードは，最終的に機械コードに翻訳して，これをメモリ上に置いてから実行する．プログラムを実行しているときのメモリは，典型的なコンピュータでは，大きく 3 つの領域，コード，データ，スタックに分けられる．データ領域は，静的変数領域とヒープ領域に分けられる．

(1) コード領域
翻訳された機械コードが格納される．一般にこの領域をプログラム自身で書き換えることはできない．テキスト領域とも言う．

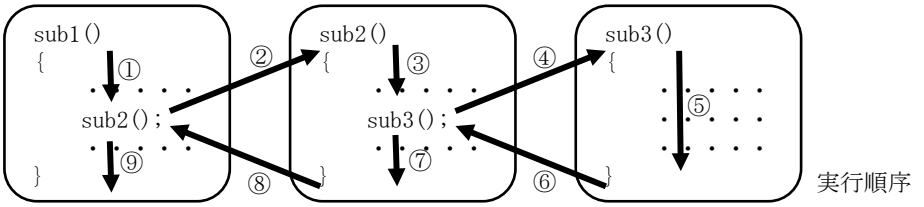
(2) データ領域
(2-1) 静的変数領域
グローバル変数や静的ローカル変数のための領域である．コンパイル時にその大きさが確定し，プログラムの実行開始時から終了時までサイズが変化することはない．より細かく，初期値が指定されている変数，初期値が指定されていない変数，定数のための領域に分ける場合もある．

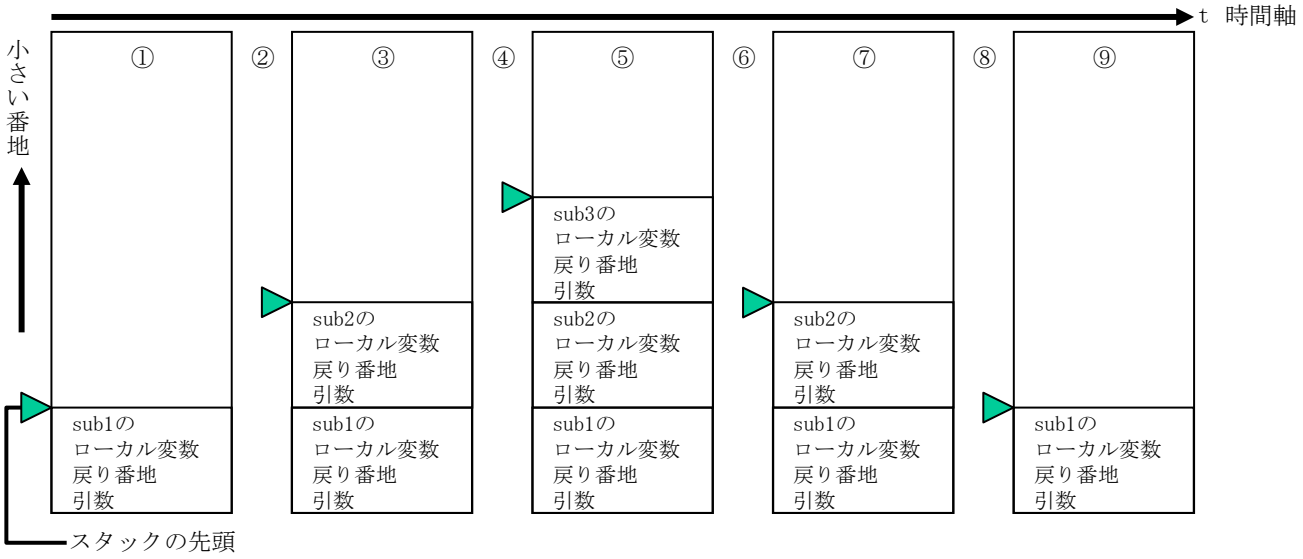
(2-2) ヒープ領域
プログラムで動的に確保した領域は，ここに取られる．C 言語では関数 malloc 等で必要なサイズを確保し，不要になった時点で関数 free でその領域を開放する．ヒープ探索とは無関係である．

(3) スタック領域
関数を呼び出すと，スタックフレームという単位で領域が確保される．これは，関数呼び出し時の引数と戻り番地，関数内でのローカル変数のための領域である．スタック構造となっている．関数呼び出しに伴うスタックの変化を理解すれば，再帰的呼び出しの意味や，ローカル変数の寿命などについて容易に理解できるようになる．

スタック構造: 本を机の上に積み上げるような構造で，データを置くときは一番上に，データを取り出すのも一番上からというような構造.

(4) スタックフレーム
関数を呼び出すと，その関数のスタックフレームがスタックに積まれ，その関数から戻ると，スタックフレームもスタックから取り除かれる．スタックフレームには，関数呼び出し時の引数と戻り番地がセットされ，関数内で宣言されたローカル変数（動的ローカル変数あるいは自動変数）のための領域も確保される．以下に，関数 sub1 内で関数 sub2 を呼び出し，関数 sub2 内で関数 sub3 を呼び出し，そして，関数 sub3 を終えて関数 sub2 に戻り，さらに，関数 sub1 に戻るときのスタックの状態変化を示す．関数内で宣言された動的ローカル変数は，その領域がスタックフレーム内に確保されるので，その関数を終えると，スタックフレームと共に動的ローカル変数の領域もなくなってしまう（別の関数を呼び出すと，そのスタックフレームで上書きされてしまう）ことを理解すること．





コンパイル・リンク　☆

ソースプログラム（例えば hello.c）をコンパイル・リンクする処理の流れを下図に沿って順に説明する．

(1) プリプロセッサ（前処理プログラム）
ソースプログラムを読み込み，主に「#」で始まる行の処理を行う．たとえば，「#include」がある場合は，そこで指定されたファイルをその場所に展開する．「#define KMAX 100」のような定数の定義がある場合は，ソースプログラム中にある文字列「KMAX」を文字列「100」に置き換える処理を行う．

(2) コンパイラ
プリプロセッサにより処理されたソースプログラムを機械コードに翻訳し，オブジェクトプログラム（hello.obj）を生成する．しかし，これだけではプログラムは実行できない．なぜなら，実行時に最初に実行され，main 関数を呼び出すスタートアップルーチンがなく，さらに，printf()などを呼び出している場合は，それらの標準ライブラリ関数もないからである．プリプロセッサとコンパイラによる処理を合わせてコンパイルという．Visual C++の開発環境において，[ビルド]メニューの[コンパイル]により，コンパイルが行われる．

(3) リンカ
コンパイラが生成したオブジェクトプログラムと，main 関数を呼び出すスタートアップルーチンや printf などの標準関数を結び付け，実行可能なファイル（hello.exe）を生成する．この処理をリンクと言う．Visual C++の開発環境においては，[ビルド]メニューの[ビルド]により，ここまでの一連の処理が行われる．但し，既にコンパイル済みの場合はコンパイルは行わない．さらに，リンク済みの場合は何もしない．