第 2 章　変数と演算子を使って計算しよう

概要

プログラムは、計算結果などの数値を記録するために変数を使用します。ここでは計算プログラムを例に、変数の使い方と計算で使用する記号（演算子）について学びます。

この章の目標

整数など変数の使用方法について覚える。
変数によって扱える数値の範囲が異なることを覚える。
計算で使用する+や-などの記号（演算子）を覚える。

2.1　変数と宣言

プログラムで複雑な処理を行うためには、計算結果を記録し、必要に応じて読み出すことの出来る変数が必要となります。
まず、次のプログラムを作成して下さい。

例題 2-2[p. 23]、例題 2-3[p. 25]

C 言語では、使用する変数の種類とその名前をまずはじめに記述します。これを「変数の宣言」といいます。

・ポイント（数値を記録する[p. 21]、注意[p. 25]）
(1)変数名は、アルファベットで始まり、英数字のみを使用します。詳細は、資料 A「変数名の付け方」(p. 11)を参考にして下さい。
(2)プログラムの最初に、変数の宣言を必ず行います。
(3)C 言語では、＝は、「等しい」の記号ではありません。変数に数値を記録（代入）するという意味です。

●代入の繰り返し
プログラムにおいて、例えば、「x = x + 1」の意味は、まず、右辺の「x + 1」の計算をし、その計算結果の値を左辺の変数「x」に代入するという意味です。
この処理の理解を深めるために、次のコラムを読んだ後、例題を作成して確認してみましょう。

コラム：プログラムの変数と方程式の変数[p. 26]
例題　：例題 2-4[p. 27]

●変数の型
C 言語で使用する変数の種類のことを「型」と言い、次のような型があります。それぞれ扱える数値の範囲が決まっていることをよく覚えておいて下さい。

・基本的データ型
char　　：1 バイト整数；1 文字を保持
int　　：整数（16 ビットもしくは 32 ビット；機種に依存）
float　：単精度浮動小数点数（32 ビット；厳密には機種に依存）
double ：倍精度浮動小数点数（64 ビット；厳密には機種に依存）

・修飾子
short　　: int に適用 ; 短いビット数で数値を扱う
long　　: int, double に適用 ; 長いビット数で数値を扱う
signed　: char, int に適用 ; 符号あり (数値の範囲に負を含む)
unsigned : char, int に適用 ; 符号なし (数値の範囲は常に正か 0)

・数値の範囲
いろいろな変数[p.29]

補足 :
(1) int 型、double 型、そして、char 型の三つの型がよく使用されます。
(2) long double 型は、サポートしていない処理系があります。
(3) char 型の数値の範囲は、厳密には機種に依存します。
(4) int や signed を省略することが可能です。以下は、その一例です。
　　signed int -> int
　　signed short int -> short int, short
　　signed long int -> long int, long

・プログラム例
(P 例 : 2.1-1)
```
#include<stdio.h>

main()
{
        int x;
        double y;
        char z;

        x = 123;
        y = 123.567;
        z = 'A';
        printf("x = %d¥n",x);
        printf("y = %lf¥n",y);
        printf("z = %c¥n",z);
}
```

●オーバフロー
signed char 型の変数は、-128 から+127 までの整数を扱えます。この変数に対して、次のような処理を行うプログラムを作成し、実行したときの値を見て下さい。

(P 例 : 2.1-2)
```
#include<stdio.h>

main()
{
        signed char x;

        x = 127;
        x = x + 1;
        printf("x = %d¥n",x);
```

```
}
```

表示結果は、正しい値（128）だったでしょうか？画面には「x = -128」と表示されたと思います。
これは、signed char 型の変数が扱える数値の範囲を超えたためです。なぜこのような表示結果となったのか、資料 B「2 の補数表示」(pp.12-13)を参考に考えて見ましょう。
また、皆さんがプログラムを作成する際は、扱える数値の範囲を意識して計算を行って下さい。

●変数の型変換

変数の型によって、扱える数値の範囲が異なることを学びました。では次に、異なる型の間での値の受け渡しについて見てみましょう。
まず、次のような処理を行うプログラムを作成し、実行したときの値を見て下さい。

（P 例：2.1-3）

```
#include<stdio.h>

main()
{
        int x;
        double y;

        x = 10;
        y = x / 4;
        printf("y = %lf¥n",y);
}
```

表示結果は、正しい値だったでしょうか？右辺の「x」と「4」は、共に整数型のため、その計算結果も整数型として、小数点以下を切り捨てて、「2」と表示されたと思います。
従って、正しい計算結果を得るためには、右辺を実数型に変換して計算する必要があります。具体的には、「y = x / 4;」の箇所を「y = (double)x / 4.0;」に修正し、実行します。これで、正しい値（2.5）が表示されたと思います。
この計算例に示すように、整数と実数が混在する計算では、注意して下さい。

（P 例：2.1-4）

```
#include<stdio.h>

main()
{
        int i;
        double d;

        i = d = 100/3; printf("i= %lf, d= %lf¥n",(double)i,d);
        d = i = 100/3; printf("i= %lf, d= %lf¥n",(double)i,d);
        i = d = 100/3.; printf("i= %lf, d= %lf¥n",(double)i,d);
        d = i = (double)100/3; printf("i= %lf, d= %lf¥n",(double)i,d);
}
```

(P 例 : 2.1-5)
```
#include<stdio.h>

main()
{
        const int i=2;
        const double d=4.8;
        int x;
        double y;

        x = (y=d/i); printf("x= %lf, y= %lf¥n",(double)x,y);
        y = (x=d/i); printf("x= %lf, y= %lf¥n",(double)x,y);
        y = d * (x = ((int)2.4 + 2.4)/d);
        printf("x= %lf, y= %lf¥n",(double)x,y);
}
```

●補足

(1)型のサイズを調べよう

char 型は、1 バイトですが、この他の基本的データ型は、機種に依存します。下記のプログラムを実行して、自分が使用している環境における基本的データ型のバイト数を調べてみましょう。

(P 例 : 2.1-6)
```
#include<stdio.h>

main()
{
        printf("size of char   = %d バイト¥n",sizeof(char));
        printf("size of int    = %d バイト¥n",sizeof(int));
        printf("size of float  = %d バイト¥n",sizeof(float));
        printf("size of double = %d バイト¥n",sizeof(double));
}
```

・追加説明

1 バイトは、8 ビットです。

(2)数値の範囲の確認プログラム

自分が使用している環境において、それぞれの変数が扱える範囲を正確に知りたい場合は、次のプログラムを実行して下さい。

printf 関数の中の%の後に続く、d や u などについては、第 3 章で詳しく説明します。

(P 例 : 2.1-7)
```
#include<stdio.h>
#include<limits.h> /* 整数の範囲を定義 */
#include<float.h>  /* 浮動小数点数の範囲を定義 */

main()
{
        printf("整数¥n");
        printf("[ char ]           max:%d, min:%d¥n",CHAR_MAX,CHAR_MIN);
        printf("[ signed char ]    max:%d, min:%d¥n",SCHAR_MAX,SCHAR_MIN);
```

```
        printf("[ unsigned char ]  max:%u, min:0¥n",UCHAR_MAX);
        printf("[ short ]          max:%hd, min:%hd¥n",SHRT_MAX,SHRT_MIN);
        printf("[ unsigned short ] max:%hu, min:0¥n",USHRT_MAX);
        printf("[ int ]            max:%d, min:%d¥n",INT_MAX,INT_MIN);
        printf("[ unsigned int ]   max:%u, min:0¥n",UINT_MAX);
        printf("[ long ]           max:%ld, min:%ld¥n",LONG_MAX,LONG_MIN);
        printf("[ unsigned long ]  max:%lu, min:0¥n",ULONG_MAX);
        printf("¥n 浮動小数点数¥n");
        printf("[ float ]          max:%e, min:%e¥n",FLT_MAX,FLT_MIN);
        printf("[ double ]         max:%le, min:%le¥n",DBL_MAX,DBL_MIN);
}
```

・出力例

整数

```
[ char ]           max:127, min:-128
[ signed char ]    max:127, min:-128
[ unsigned char ]  max:255, min:0
[ short ]          max:32767, min:-32768
[ unsigned short ] max:65535, min:0
[ int ]            max:2147483647, min:-2147483648
[ unsigned int ]   max:4294967295, min:0
[ long ]           max:2147483647, min:-2147483648
[ unsigned long ]  max:4294967295, min:0
```

浮動小数点数

```
[ float ]          max:3.402823e+038, min:1.175494e-038
[ double ]         max:1.797693e+308, min:2.225074e-308
```

・追加説明

浮動小数点数の表現の一つである IEEE (Institute of Electrical and Electronics Engineews; 米国電子技術者協会、通称アイトリプルイー) の 754 規格では、浮動小数点数を、32 ビットまたは、64 ビットで表現します。32 ビットを単精度 (Single Precision) と呼び、 64 ビットを倍精度 (Double Precision) と呼びます。

IEEE754

[符号部(sign)][指数部(exponent)][仮数部(mantissa)]

32 ビット表現 (単精度) : 符号部を 1 ビット、仮数部を 23 ビット、指数部を 8 ビット

64 ビット表現 (倍精度) : 符号部を 1 ビット、仮数部を 52 ビット、指数部を 11 ビット

この場合、float 型の有効桁数は、約 6 桁(10 進数)で、double 型の有効桁数は、約 15 桁(10 進数)です。

2.2　定数

●定数の宣言

プログラムで使用する定数は、通常の変数として宣言し、プログラム中で用いることができますが、定数であることを明示するために、次のような宣言を用いるのが一般的です。

・宣言例

変数 MAX を定数 100 として使用するプログラム例です。なお、(P 例 : 2.2-2) は、定数の型を明確に宣言する方

法です。

(P 例 : 2.2-1)

```
#include<stdio.h>

#define MAX 100

main()
{
        int x;

        x = MAX;
        printf("%d¥n",x);
}
```

・補足

「#define MAX 100」は、MAX の変数を使用する前であればどこでも記述することが出来ます。例えば、「int x;」の前後の行に書くことも出来ます。

(P 例 : 2.2-2)

```
#include<stdio.h>

main()
{
        const int MAX = 100;
        int x;

        x = MAX;
        printf("%d¥n",x);
}
```

●定数の種類

定数には、次の三つの種類があります。

・整数定数
・浮動小数点定数
・文字定数

補足 :

特殊な文字定数 (1 文字)

'¥n'	改行
'¥t'	(水平) タブ
'¥b'	後退 (バックスペース)
'¥r'	復帰
'¥a'	ベル (警告文字)
'¥¥'	バックスラッシュ
'¥''	単一引用符
'¥"'	二重引用符
'¥?'	疑問符

表 2.2-1　定数の種類

(a)整数定数

型	説明	
	例	接尾子
int	128 0200 (8進数) 0x80 (16進数)	なし。ただし、intが扱える範囲を超えるとlongとして扱われる。
long	123456789L 123456789l	L、もしくは、l (エル)
unsigned int	1234U 1234u	U、もしくは、u
unsigned long	123456789UL 123456789ul	UL、もしくは、ul

(b)浮動小数点数

型	説明	
	例	接尾子
double	123.4 1.234e+2	なし。
float	123.4F 123.4f	F、もしくは、f
long double	123.4L 123.4l	L、もしくは、l (エル)

(c)文字定数

型	説明	
	例	引用符
char	'a'	1文字；単一引用符 ('')
	"abcd"	文字列；二重引用符 ("")

●補足

#define は、次のように数値以外も定義できます。

(P 例：2.2-3)

```
#include<stdio.h>

#define PRINT printf("%d¥n",x)

main()
{
        int x;

        x = 100;
        PRINT;
}
```

2.3　演算子

その他の演算子[p.30]

計算で使用される四則演算を表す+や-の記号は演算子と呼ばれ、この他にプログラムで使用される演算子には、次のようなものがあります。

1. 算術演算子
   加減演算子
   乗除演算子
   符号演算子
   インクリメント演算子／デクリメント演算子
2. 代入演算子
3. 型演算子
4. 順序演算子
5. 比較演算子
   関係演算子
   等価演算子
   条件演算子
6. 論理演算子
7. アドレス演算子
8. ビット演算子
   論理演算子
   シフト演算子

各演算子の説明については、資料 C「演算子のまとめ」(pp.14-16)を参照して下さい。ここでは、よく使用する代表的な演算子を使用したプログラムを例に、それぞれの演算子の意味について理解を深めていきましょう。

●色々な演算子を使用したプログラム例
それぞれのプログラムの出力結果は、どのようになるでしょうか？

1. 算術演算子
 (P 例 : 2.3-1)

```
#include<stdio.h>

main()
{
        int x;

        x = 10 - -2 * 5 - 13;         printf("x = %d¥n",x);
        x = -8 + 4 / 2 * 9 - 3;       printf("x = %d¥n",x);
        x = 1 + 3 % 5 + (8 + 7) / 5; printf("x = %d¥n",x);
}
```

 (P 例 : 2.3-2)

```
#include<stdio.h>

main()
{
        int x,y;

        x = 1;
        y = ++x; printf("y = %d,  x = %d¥n",y,x);

        x = 1;
        y = x++; printf("y = %d,  x = %d¥n",y,x);
}
```

2.代入演算子
(P 例 : 2.3-3)

```
#include<stdio.h>

main()
{
        int x;

        x = 5;  printf("x = %d¥n",x);
        x += 2; printf("x = %d¥n",x);
        x *= 3; printf("x = %d¥n",x);
}
```

5.比較演算子
(P 例 : 2.3-4)

```
#include<stdio.h>

main()
{
        int x;

        x = 3 < 5;  printf("x = %d¥n",x);
        x = 3 > 5;  printf("x = %d¥n",x);
        x = 3 <= 5; printf("x = %d¥n",x);
        x = 5 >= 5; printf("x = %d¥n",x);
        x = 3 == 5; printf("x = %d¥n",x);
        x = 3 != 5; printf("x = %d¥n",x);

        x = 3 < 5 ? 10 : 20; printf("x = %d¥n",x);
        x = 3 > 5 ? 10 : 20; printf("x = %d¥n",x);
}
```

6.論理演算子
(P 例 : 2.3-5)

```
#include<stdio.h>

main()
{
        int x,y;

        x = 3 < 5 || 3 > 5; printf("x = %d¥n",x);
        x = 3 < 5 && 3 > 5; printf("x = %d¥n",x);
        x = !(3 < 5);       printf("x = %d¥n",x);

        y = 2000;
        x = y%4==0 && y%100>0 || y%400==0;
        printf("x = %d¥n",x);
}
```

8.ビット演算子
(P 例 : 2.3-6)

```
#include<stdio.h>

main()
{
        signed char x;

        x = ~2;     printf("x = %d¥n",x);
        x = 1 | 3;  printf("x = %d¥n",x);
        x = 1 & 3;  printf("x = %d¥n",x);
        x = 1 ^ 3;  printf("x = %d¥n",x);
        x = 1 << 3; printf("x = %d¥n",x);
}
```

●補足

(1)計算を行う上での各演算子の優先順位は、資料 C「演算子のまとめ」(pp.14-16)の末尾にある優先順位表を参考にして下さい。

(2)演算子を表す記号の読み方については、資料 D「記号の読み方」(p.17)を参照して下さい。

(3)#define を使用した出力結果の工夫
(P 例 : 2.3-7)

```
#include<stdio.h>

#define PRINT(int) printf(#int "  =  %d¥n",int)

main()
{
        PRINT(10 - -2 * 5 - 13);
        PRINT(-8 + 4 / 2 * 9 - 3);
        PRINT(1 + 3 % 5 + (8 + 7) / 5);
}
```

資料 A「変数名の付け方」

C 言語で変数の名前に使用できる文字は、次のとおりです。

a～z　A～Z　0～9　_（下線）

変数名はこれらを組み合わせてつくります。ただし、変数名の先頭に数字を用いることはできません。また長さも 31 文字までで、それ以上長い変数名では 32 文字目以降は無視されます。

さらに次の C の予約語を変数名とすることはできません。

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	voltile
do	if	static	while

補足

C 言語では、大文字と小文字を区別しますので、「tokyo」、「TOKYO」、そして「toKyO」は、異なる変数名として扱われます。

資料 B「2 の補数表示」

●扱える数の範囲

8 ビットのメモリに整数を格納する場合を例に説明します。まず、簡単に全て正の整数とすると、0 から 2^8-1 (255) まで、全部で 256 個の数を表現することができます (図参照)。
そこで、負の数も扱えるように、2 の補数表示では、このうち半分を 0 または正の整数、残り半分を負の整数として表します。
従って、0 または正の整数は、0 から 127 まで表すことができます。また負の整数は-1 から-128 まで表すことができます。

一般的には、n ビットの 2 の補数表示による符号付き整数は、-2^{n-1} から $2^{n-1}-1$ までの整数を表すことができます。

	2進数							
	8ビット	7ビット	6ビット	5ビット	4ビット	3ビット	2ビット	1ビット
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1
値								
7	0	0	0	0	0	1	1	1
127	0	1	1	1	1	1	1	1

図　各ビットと 2 進数の対応

●メモリ上での数の表現

2 の補数表示で実際に整数をメモリに格納する方法は、次のとおりです。
1)　0 から 127 までの数は符号なし整数と同じ方法でメモリに格納する。
2)　-128 から-1 までの数は、256 を加えて、128 から 255 までの数にして、それを符号なし整数と同じ方法でメモリに格納する。
補足 : この規則に従うと負の整数は、最上記のビット、図の 8 ビットの場合は、8 ビット目の値が 1 となります。

ここで 2)について、一般的には、n ビットの 2 の補数表示による場合は、-2^{n-1} から-1 までの数に 2^n を加えて、2^{n-1} から 2^n-1 までの数にし、それを符号なし整数と同じ方法でメモリに格納します。

この規則より、負の整数の各ビットの値を次のように求めることが出来ます。
まず、負符号を付けて正の整数とし、この場合の各ビットの値を求め、次に、各ビットを反転 (1 は 0 に、0 は 1 に) したものに 1 を加えることで得られます。
補足 : 各ビットを反転することを、1 の補数を求めるといいます。X の 1 の補数は、~X と記号では表します。
(証明)
今、負の整数を A とし、規則 2)に従ってメモリに格納した整数を A*とします。規則の定義より、二つの整数には、次のような関係が成り立ちます。

A* = A + 256

ここで、(-A)とこの 1 の補数である~(-A)の和は、各ビットの値が 1 となることから次の式が成り立ちます。

(-A) + ~(-A) = 255

従って、二つの式をあわせると次のように展開できます。

A* = A + 255 + 1
A* = A + (-A) + ~(-A) + 1

よって、次のとおり、負符号を付けて正の整数にし、この各ビットを反転 (1 は 0 に、0 は 1 に) したものに 1

を加えることで、求める A*が得られます。

A* = ~(-A) + 1

●便利な性質

2 の補数表示を用いることによって、整数の加減乗の計算が便利になります。例えば、二つの正の整数 A と B の引き算を行う場合、次のように足し算として計算することができます。

A - B = A + (-B)

計算例

(A - B); A = 6, B = 3

```
 6 :      0 0 0 0 0 1 1 0
+-3 :     1 1 1 1 1 1 0 1
 3 : (1) 0 0 0 0 0 0 1 1
```

8 ビットの場合、8 ビットを超えたものは無視されます。

掛け算についても同様に、正の整数の場合と同じように計算できます。従って、符号付き整数での加減乗の計算は、符号なし整数での加法、乗法、そして、符号の反転で計算することができることになります。

資料C「演算子のまとめ」

●演算子

1. 算術演算子

加減演算子

演算子	例	意味
+	x + y	xとyの和
−	x − y	xからyの差

乗除演算子

演算子	例	意味
*	x * y	xとyの積
/	x / y	xをyで割った商
%	x % y	xをyで割った余り

符号演算子

演算子	例	意味
+	+x	xそのもの
−	−x	xの持つ符号の反転

インクリメント演算子／デクリメント演算子

演算子	例	意味
++	x++　(x−−)	x xは使用された後でインクリメント(デクリメント)される
−−	++x　(−−x)	x+1　(x−1) xは使用される前にインクリメント(デクリメント)される

2. 代入演算子

演算子	例	意味
=	x = y	yはxの型に変換され、xにはyの値が代入される
op=	x += yなど	*op*は、+, −, *, /など（例：x = x + y） x *op* yは、xの型に変換され、xにはx *op* yの値が代入される

3. 型演算子

演算子	例	意味
(*type*)x	(int)x	xが*type*という型に変換される
sizeof x	sizeof x	xが占めるバイト数
sizeof(*type*)	sizeof(int)	*type*という型が占めるバイト数

4. 順序演算子

演算子	例	意味
,	x, y	y xがyの前に評価される

5.比較演算子
関係演算子

演算子	例	意味
<	x < y	xがyよりも小さければ1になり、それ以外は0になる
>	x > y	xがyよりも大きければ1になり、それ以外は0になる
<=	x <= y	xがy以下ならば1になり、それ以外は0になる
>=	x >= y	xがy以上ならば1になり、それ以外は0になる

等価演算子

演算子	例	意味
==	x == y	xがyと等しければ1になり、それ以外は0になる
!=	x != y	xがyと等しくなければ1になり、それ以外は0になる

条件演算子

演算子	例	意味
?:	x ? y : z	xが0でなければyになり、それ以外はzになる

6.論理演算子

演算子	例	意味
&&	x && y	xとyのANDをとる。xとyがどちらも0でなければ1になり、それ以外は0になる
||	x || y	xとyのORをとる。xとyがどちらも0ならば0になり、それ以外は1になる
!	!x	xの論理否定をとる。xが0でなければ0になり、それ以外は1になる

7.アドレス演算子

演算子	例	意味
*	*x	xの基本データ型として、xのアドレスに格納されている値
&	&x	xのアドレス
.	x.y	構造体xのフィールドyの値
->	x->y	xで指し示される構造体のフィールドyの値

8.ビット演算子
論理演算子

演算子	例	意味
&	x & y	ビットごとのxとyのANDをとる。ANDはxとyの両方が1であれば1になり、それ以外は0になる
|	x | y	ビットごとのxとyのORをとる。ORはxとyの両方が0であれば0になり、それ以外は1になる
^	x ^ y	ビットごとのxとyの排他的ORをとる。排他的ORはxとyが同じ値であれば0になり、それ以外は1になる
~	~x	xの、1の補数を求める。1は0になり、0は1になる

シフト演算子

演算子	例	意味
<<	x << y	xをyの数だけ左シフトする。最下位ビットには0が入る
>>	x >> y	xをyの数だけ右シフトする。最上位ビットには、正の値の場合は0が入り、負の値の場合はコンパイラに依存して1または0が入る

●優先順位表

演算子		結合性
一次子	( ) [ ] -> .	左から右
単項	! ~ ++ -- + - (データ型) * & sizeof	右から左
乗法	* / %	左から右
加減	+ -	左から右
シフト	<< >>	左から右
関係	< <= > >=	左から右
等価	== !=	左から右
ビット	&	左から右
ビット	^	左から右
ビット	|	左から右
論理	&&	左から右
論理	||	左から右
条件	? :	右から左
代入	= += -= 等	右から左
カンマ	,	左から右

資料 D「記号の読み方」

記号	読み方	記号	読み方
!	エクスクラメーションマーク、感嘆符、雨だれ	?	クエスチョンマーク、疑問符
″	ウムラウト、引用符、ダブルクオーテーション	@	アットマーク
’	アポストロフィ、アクサンテギュ、シングルクォート、ダッシュ	(	左小括弧、始めパーレン
#	シャープ、番号符、ナンバー	)	右小括弧、終わりパーレン
$	ドル	{	左中括弧、始めカーリーブレース
&	アンパーサンド、アンド	}	右中括弧、終わりカーリーブレース
+	プラス	[	左大括弧、始めブラケット
−	ハイフン、マイナス、連字符	]	右大括弧、終わりブラケット
*	アスタリスク	¥	円、円マーク
/	スラッシュ	^	ハット、キャラット、アクサンシルコンフレックス、サーカムフレックス
%	パーセント	~	チルダ、チルド、オーバーライン
,	カンマ、コンマ、セディユ	_	アンダーバー、アンダライン
.	ドット、ピリオド	`	逆クォート、バッククォート、アクサングラーブ
:	コロン	|	縦棒、パイプ、オア
;	セミコロン	・	中点
<	小なり、不等号(より小)、左アングルブラケット		
=	イコール、等号		
>	大なり、不等号(より大)、右アングルブラケット		