

# 計算する

三池 克明

プログラム開発の基本中の基本といえる「演算」と「変数」について解説します。

## 目次

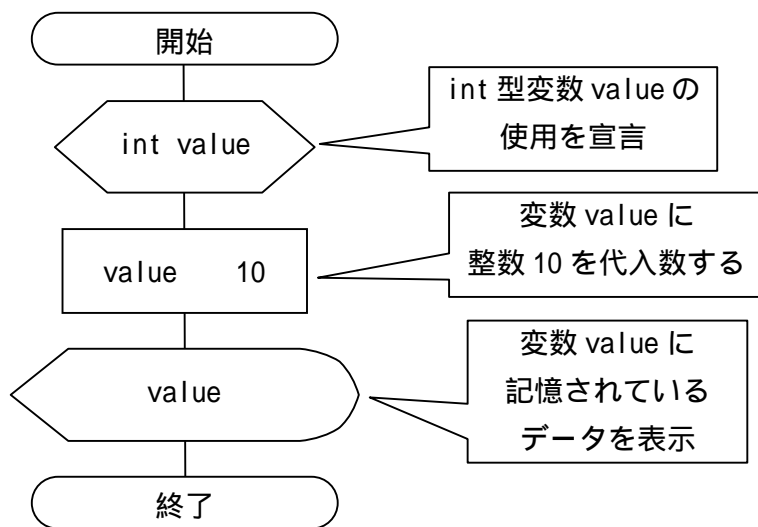
1.	変数を使う	1
1.1.	変数を使ったプログラム	1
1.2.	計算する	4
1.3.	データ型を変える	6
2.	データ型	9
2.1.	int 型	10
2.2.	short 型と long 型	10
2.3.	char 型	11
2.4.	double 型	12
2.5.	float 型と long double 型	12
2.6.	unsigned 型	13
2.7.	void 型	13
3.	演算子	14
3.1.	代入... =	14
3.2.	余り... %	16
3.3.	増加、減少など... +=、 -= など	17
3.4.	インクリメント、デクリメント... ++、 --	17
4.	関数 printf() あれこれ	19
4.1.	定数や式の結果を表示させる	19
4.2.	符号付きで表示する	21
4.3.	桁数を指定する	22
5.	演習問題	25

## 1. 変数を使う

### 1.1. 変数を使ったプログラム

変数とはデータ（数値や文字列など）を記憶するものです。  
データを入れる入れ物、あるいはメモ用紙をイメージすれば理解しやすいと思います。

変数を扱うプログラムとして以下のフローチャートどおりに処理するプログラムを作成します。



コーディングをするとこのようになります。

value1.cpp

```
1: /* 変数を扱う その1 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    value;
7:
8:     value = 10;
9:
10:    printf("value = %d\n", value);
11: }
```

このソースコードを「value1.cpp」(.cpp は自動的に付記されます)というファイル名で保存します。

コンパイル、実行すると、このように表示されます。

実行結果

```
value = 10
```

それではプログラムの動きを一つ一つたどっていきましょう。

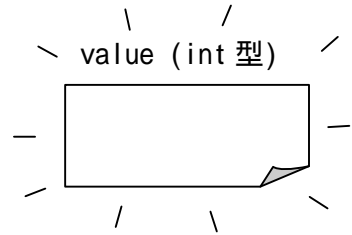
```
5: {
6:     int    value;
7:
```

int 型というデータ型で名前 value の変数を使うことを宣言しています。また int 型は整数のみ扱うことができます。

( int の語源は Integer、つまり整数です )

この文では右図のようにメモ用紙を用意し、

- このメモ用紙を value と呼ぶ
- int 型、つまり整数のデータしか書き込まない



とってください。

なお现阶段では、このメモ用紙には何も書き込まれていません。

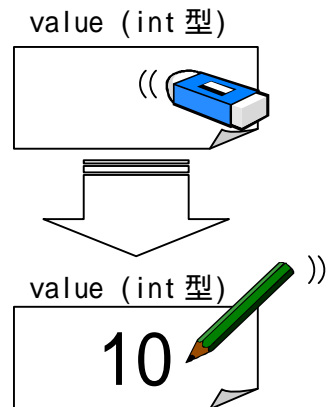
(厳密には何かしらのデータが残っていますが、その説明は省略します)

```
7:
8:     value = 10;
9:
```

int 型変数 value に 10 を代入しています。  
代入とは変数に値を書き込むことで、右図の  
ようにメモ用紙に

- メモに何かが書かれていたら消す
- メモ用紙にデータをメモする

とってください。



```
9:
10:     printf("value = %d\n", value);
11: }
```

関数 printf() は文字列を表示するだけでなく、変数が記憶しているデータを任意の形式で表示することもできます。

文字列の中を見てください。「%d」という部分がありますね。

この部分に value のデータが表示されます。

また%d は「データを 10 進法の整数で表示する」という意味です。

## 1.2. 計算する

次に以下の四則演算を行い、その結果を画面に表示するプログラムを作成します。

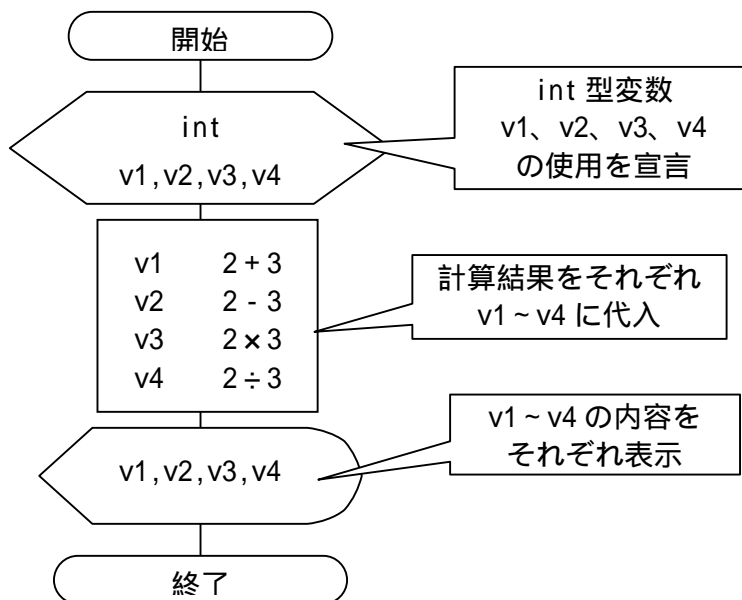
$$v_1 = 2 + 3$$

$$v_2 = 2 - 3$$

$$v_3 = 2 \times 3$$

$$v_4 = 2 \div 3$$

フローチャートに表すとこのようになります。



コーディングをするとこのようになります。

value2.cpp

```
1: /* 変数を扱う その2 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    v1, v2, v3, v4;
7:
8:     v1 = 2 + 3;
9:     v2 = 2 - 3;
10:    v3 = 2 * 3;
11:    v4 = 2 / 3;
12:
13:    printf("(v1, v2, v3, v4) = (%d, %d, %d, %d)\n", v1, v2, v3, v4);
14: }
```

乗算 ( × ) は \*  
除算 ( ÷ ) は /  
で表す

このソースコードを「value2.cpp」というファイル名で保存します。  
そしてコンパイル、実行するとこのように表示されます。

実行結果

(v1, v2, v3, v4) = (5, -1, 6, 0)

実際に計算した結果と、プログラムの計算結果を比べてみましょう。

$$\begin{aligned}v_1 &= 2 + 3 = 5 \\v_2 &= 2 - 3 = -1 \\v_3 &= 2 \times 3 = 6 \\v_4 &= 2 \div 3 = \frac{2}{3}\end{aligned}$$

v4 の値が間違っています。  $2 \div 3 = \frac{2}{3}$  なので 0.666...になるはずですが。

では、どうしてこのようになったのでしょうか。

### 1.3. データ型を変える

value2.cpp の 6 行目を見てみましょう。

```
5: {  
6:     int    v1, v2, v3, v4;  
7:
```

となっています。また int 型変数は整数しか扱えないと説明しました。

int 型は右図のように計算結果の小数点以下を切り捨てます。

これが  $2 \div 3 = 0$  になってしまった理由です。

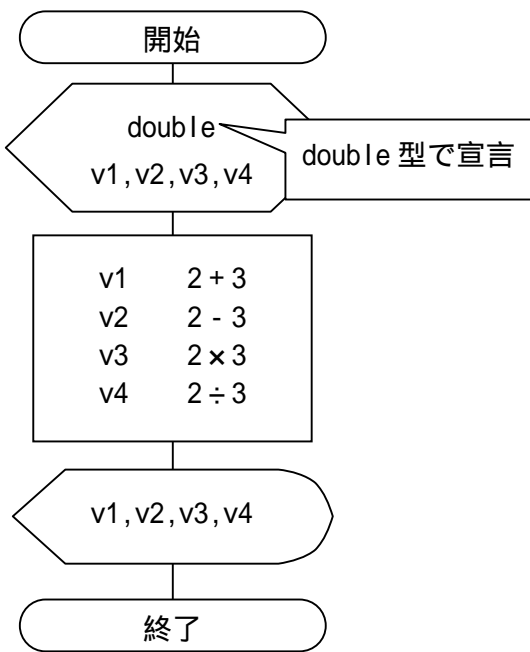
$$2 \div 3 = 0.6666666666\dots$$

= 0 小数点以下は切り捨てる

よって今回のプログラムでは int 型は使えません。

そこで小数を扱える double 型の変数を使いましょう。

フローチャートはこのように変更されます。



ソースコードもこのように変更されます。

このソースコードを「value3.cpp」というファイル名で保存します。

value3.cpp

```
1: /* 変数を扱う その3 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     double v1, v2, v3, v4;
7:
8:     v1 = 2 + 3;
9:     v2 = 2 - 3;
10:    v3 = 2 * 3;
11:    v4 = 2 / 3;
12:
13:    printf("(v1, v2, v3, v4) = (%f, %f, %f, %f)\n", v1, v2, v3, v4);
14: }
```

double に修正

double 型の表示は%f

そしてコンパイル、実行するとこのように表示されます。

実行結果

```
(v1, v2, v3, v4) = (5.000000, -1.000000, 6.000000, 0.000000)
```

おや？ まだ結果が間違っています。

value03.cpp の 11 行目を見てみましょう。

```
10:    v3 = 2 * 3;
11:    v4 = 2 / 3;
12:
```

この「=」がある構文を代入式と呼びますが、この代入式の右辺を「2.0 / 3.0」と修正してみましょう。

ソースコードはこのように変更されます。

このソースコードを「value4.cpp」というファイル名で保存します。

value4.cpp

```
1: /* 変数を扱う その4 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     double v1, v2, v3, v4;
7:
8:     v1 = 2 + 3;
9:     v2 = 2 - 3;
10:    v3 = 2 * 3;
11:    v4 = 2.0 / 3.0;
12:
13:    printf("(v1, v2, v3, v4) = (%f, %f, %f, %f)\n", v1, v2, v3, v4);
14: }
```

この行を修正

そしてコンパイル、実行するとこのように表示されます。

実行結果

```
(v1, v2, v3, v4) = (5.000000, -1.000000, 6.000000, 0.666667)
```

今度うまくいきました。

これは右辺の値を `double` 型にしたためです。

定数(2とか10.5など)を小数の桁を含めて記述するとコンパイラは `double` 型の定数と判断します。

(コンパイラやコンピュータの仕様によっては `float` 型と判断することもあります)

そして計算結果も `double` 型で算出し、左辺の変数に代入するのです。

つまり変数だけでなく定数にもデータ型があります。

## 2. データ型

以下に頻繁に使われるデータ型を表にまとめました。  
必要に応じてこのページを開く習慣をつけましょう。

型の表記	型名	読み方	範囲	定数の表記例
char	文字型	キャラ チャー	-128 ~ 127	'a' '<'
short	単精度 整数型	ショート	-32,768 ~ 32,767	123 -2212
int	整数型	イント	-2,147,483,648 ~ 2,147,483,647	4567 -23
long	倍精度 整数型	ロング	-2,147,483,648 ~ 2,147,483,647	655381 -1231
float	単精度 実数型	フロート	$1.18 \times 10^{-38} \sim 3.40 \times 10^{38}$ (有効数字 7 桁)	3.14f -3.0f
double	倍精度 実数型	ダブル	$2.23 \times 10^{-308} \sim 1.79 \times 10^{308}$ (有効数字 15 桁)	5.5 -10.21
long double	倍々精度 実数型	ロングダブル	$3.37 \times 10^{-4932} \sim 1.18 \times 10^{4932}$ (有効数字 18 桁)	152.31 2.718281
unsigned	符号なし	アンサインド	---	35u
void	ボイド型	ボイド	---	---

使用するコンピュータやコンパイラの仕様などにより範囲や表記例が異なる場合があります

---

## 2.1. int 型

---

「value1.cpp」で使ったとおり、整数のデータのことです。

整数を扱う場合はこのデータ型を使えばよいでしょう。

扱える範囲は -2,147,483,648 ~ 2,147,483,647 としていますが、環境によってこの範囲は異なります。これはコンピュータや OS などの仕様に合わせているためです。

また、関数 printf() で表示する場合は「%d」を使います。

int1.cpp

```
1: /* int 型変数を扱う */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    a;
7:
8:     a = 10;
9:
10:    printf("変数 a の値は %d です。¥n", a);
11: }
```

実行結果

変数 a の値は 10 です。

---

## 2.2. short 型と long 型

---

これらも整数を扱うデータ型ですが扱える範囲が環境によって変わることはありません。よってコンピュータや OS などに依存しないプログラムを開発する場合はこちらを使えばよいでしょう。

と、いっても一般的なパソコンで動作するプログラムは int 型を使用しても問題ありません。

---

## 2.3. char 型

---

文字を扱うデータ型です。

ただし扱えるのは半角 1 文字です。

また char 型の定数 (文字定数と呼ぶ) は ' (シングルクォーテーション) で囲まなければならない。

これなら大丈夫  
'a'、'N'、'+'

× これはダメ  
'あ'、'ab'

また、関数 printf() で表示する場合は「%c」を使います。

char1.cpp

```
1: /* char 型変数を扱う */
2: #include <stdio.h>
3:
4: main()
5: {
6:     char    a;
7:
8:     a = 'Y';
9:
10:    printf("変数 a の値は %c です。¥n", a);
11: }
```

実行結果

変数 a の値は Y です。

---

## 2.4. double 型

---

実数を扱うデータ型です。

ただし円周率 3.14159...などの無限小数は、ある程度の桁で切り捨てられます。数学的な計算に良く使われるデータ型です。

また、関数 printf で表示する場合は「%f」あるいは「%lf」を使います。

double1.cpp

```
1: /* double 型変数を扱う */
2: #include <stdio.h>
3:
4: main()
5: {
6:     double e;
7:
8:     e = 2.71828;
9:
10:    printf("変数 e の値は %f です。¥n", e);
11: }
```

実行結果

変数 a の値は 2.718280 です。

---

## 2.5. float 型と long double 型

---

これらも実数を扱うデータ型です。

ただし扱える範囲や有効桁数が異なります。

それほど精度が求められない計算の場合は float 型を、細かい精度を求められる計算には long double 型を使えばよいでしょう。

なお、本書では実数を扱う場合は全て double 型とします。

---

## 2.6. unsigned 型

---

これは「unsigned int」などのように他の型と組み合わせて使います。

unsigned をつけると、範囲が正の数だけになります。

例えば short 型は -32,768 ~ 32,767 の整数ですが、unsigned short 型は 0 ~ 65535 の整数になります。扱える正の整数が増えました。

このように明らかに負の数を扱わない場合や、扱える範囲をずらしたい場合に言えばよいでしょう。

なお unsigned が使えるのは char、short、int、long です。

また、関数 printf で表示する場合は「%u」を使います。

unsigned1.cpp

```
1: /* unsigned short 型変数を扱う */
2: #include <stdio.h>
3:
4: main()
5: {
6:     unsigned short  a;
7:
8:     a = 50000;
9:
10:    printf("変数 a の値は %u です。¥n", a);
11: }
```

実行結果

変数 a の値は 50000 です。

---

## 2.7. void 型

---

これは特殊なデータ型です。

変数のデータ型として使うことは極めて稀で、後述する「関数」で使うことが多いです。

今のところは、「こういうものがある」程度の理解で構いません。

### 3. 演算子

演算子とは+ や ÷ など計算を行わせる記号のことです。

C 言語では四則演算の演算子はそれぞれ +、-、\*、/ であることは既に説明したとおりです。

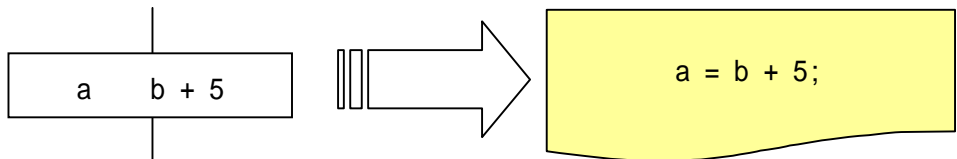
ここではC 言語ならではの演算子について解説します。

#### 3.1. 代入... =

この演算子を使ったプログラムを作りましたが、ここでもう一度説明します。

C 言語での = 演算子は、数学の「等しい」という意味ではありません。右辺の計算結果を左辺に代入するという意味です。

フローチャート上で = と表記しなかったのはそのためです。



これにより以下の記述が可能になります。

```
a = a + 10;
```

もちろん数学的にはありえない式です。

しかしこれを C 言語では、

(右辺の)  $a + 10$  の式を計算し、その結果を (左辺の)  $a$  に代入する。

と解釈します。よって問題ありません。

またこの代入式は、

変数 a の値を 10 増加させる。

と解釈すると理解しやすいでしょう。

## 代入演算子の応用

代入演算子を使用してこのようなコードを記述することも可能です。

例 1)

```
a = 10;
```

```
b = 10;
```

```
c = 10;
```

```
a = b = c = 10;
```

例 2)

```
a = 3;
```

```
printf("a = %d\n", a);
```

```
printf("a = %d\n", a = 3);
```

ただし多用すると見づらいソースコードになるので気をつけましょう。

---

## 3.2. 余り... %

---

整数の割り算の式で、

$$[\text{割られる数}] \div [\text{割る数}] = [\text{商}] \dots [\text{余り}]$$

というのを覚えていますか？

%演算子はこの式の「余り」を算出してくれます。

例えば、

```
b = 10 % 3;
```

の場合、

$$10 \div 3 = 3 \dots 1$$

ですので、変数 b には 1 が代入されます。

---

### 3.3. 増加、減少など... +=、 -= など

---

このような代入式が記述できるのは既に説明しました。

```
a = a + 10;
```

この式を += 演算子を使って記述するとこのようになります。

```
a += 10;
```

同じ考え方で、このように代入式を短縮させることも可能です

```
a = a - 10;      a -= 10;
a = a * 10;     a *= 10;
a = a / 10;     a /= 10;
a = a % 10;     a %= 10;
```

---

### 3.4. インクリメント、デクリメント... ++、 --

---

1 だけの増加あるいは減少はこのように記述することが可能です。

```
a += 1;          a++; または ++a;
a -= 1;          a--; または --a;
```

この演算子は頻繁に使いますので使い慣れておきましょう。

また、a++と++a は細かい機能の違いがありますが、それについては割愛します。

## べき乗や 根を計算する演算子

C 言語には 2 乗などのべき乗や 根などの根を計算するための演算子はありません。

ですが、計算できないわけではありません。

例えば  $A^3$  であれば  $A * A * A$  のように掛け算の連続で記述すれば問題ありません。

また `pow()` や `sqrt()` などの関数を利用することで算出できますが、これについての解説は別の機会に譲ります。

## 4. 関数 printf()あれこれ

### 4.1. 定数や式の結果を表示させる

これまでは関数 printf() で変数の値を表示させてましたが、定数や式の結果を表示させることも可能です。

変数の値を表示させるのに関数 printf() を使いましたが、以下のようなこともできます。

printf1.cpp

```
1: /* 定数を表示 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     printf("%d\n", 12);
7:     printf("%f\n", 3.141592);
8:     printf("%c\n", 'Z');
9: }
```

実行結果

```
12
3.141592
Z
```

このように定数をそのまま表示させることもできます。  
あまり使う場面はなさそうですが...

続いて以下のソースプログラムを入力し、コンパイル、実行してみましょう。

printf2.cpp

```
1: /* 式の結果を表示 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    a;
7:
8:     a = 10;
9:
10:    printf("a = %d\n", a);
11:    printf("a + 5 = %d\n", a + 5);
12:    printf("3 + 10 = %d\n", 3 + 10);
13: }
```

実行結果

```
a = 10
a + 5 = 15
3 + 10 = 13
```

また関数 `printf()` ではこのように式の結果を表示させることもできます。これを活用すると、使用する変数を減らしたり、プログラムを短くすることができます。

その一方でプログラムが見難くなる可能性もあるので注意しましょう。

---

## 4.2. 符号付きで表示する

---

以下のソースプログラムを入力し、コンパイル、実行してみましょう。

printf3.cpp

```
1: /* 符号付きで表示させる */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    a = -10, b = 3;
7:
8:     printf("普通に表示させます。¥n");
9:     printf("(a, b) = (%d, %d)¥n", a, b);
10:
11:     printf("符号付きで表示させます。¥n");
12:     printf("(a, b) = (%+d, %+d)¥n", a, b);
13: }
```

実行結果

普通に表示させます。

(a, b) = ( -10, 3)

符号付きで表示させます。

(a, b) = ( -10, +3)

このように %+d や %+f と+を入れると符号付きで表示します。

---

### 4.3. 桁数を指定する

---

関数 `printf()` では表示する桁数（文字数）を指定することができます  
この機能は今後も頻繁に使用するでマスターしておきましょう。

まずは以下のソースプログラムを入力し、コンパイル、実行してみましょう。

printf4.cpp

```
1: /* 表示桁数を指定する / しない(整数型の場合) */
2: #include <stdio.h>
3:
4: main()
5: {
6:     printf("表示桁数を指定しないで表示します\n");
7:     printf("%d\n", 12);
8:     printf("%d\n", 123);
9:     printf("%d\n", 1234);
10:    printf("%d\n", 12345);
11:
12:    printf("表示桁数を 4 桁にして表示します\n");
13:    printf("%4d\n", 12);
14:    printf("%4d\n", 123);
15:    printf("%4d\n", 1234);
16:    printf("%4d\n", 12345);
17:
18:    printf("表示桁数を 5 桁にして空白部を 0 で埋めます\n");
19:    printf("%05d\n", 12);
20:    printf("%05d\n", 123);
21:    printf("%05d\n", 1234);
22:    printf("%05d\n", 12345);
23: }
```

表示桁数を指定しないで表示します

12

123

1234

12345

表示桁数を 4 桁にして表示します

  12

 123

1234

12345

表示桁数を 5 桁にして空白部を 0 で埋めます

00012

00123

01234

12345

このように桁数の指定や空白桁を 0 で埋めることもできます。

次は以下のソースプログラムです。  
入力し、コンパイル、実行してみましょう。

printf5.cpp

```
1: /* 表示桁数を指定する / しない(実数型の場合) */
2: #include <stdio.h>
3:
4: main()
5: {
6:     printf("表示桁数を指定しないで表示します\n");
7:     printf("%f\n", 3.14);
8:
9:     printf("表示桁数を全部 (小数点含む) で 5 桁、うち小数部を 3 桁で表示します\n");
10:    printf("%5.3f\n", 3.14);
11:
12:    printf("表示桁数を全部 (小数点含む) で 11 桁、\n");
13:    printf("うち小数部を 5 桁で表示し、空白部を 0 で埋めます\n");
14:    printf("%011.5f\n", 3.14);
15: }
```

実行結果

表示桁数を指定しないで表示します

3.140000

表示桁数を全部 (小数点含む) で 5 桁、うち小数部を 3 桁で表示します

3.140

表示桁数を全部 (小数点含む) で 11 桁、

うち小数部を 5 桁で表示し、空白部を 0 で埋めます

00003.14000

%f 表示では更に少数桁の指定もできます。

## 5. 演習問題

このプログラム「ensyu2 -0.cpp」を基に、以下の演習問題に取り組みなさい。

ensyu2 -0.cpp

```
1: /* 2+3 の計算する */
2: #include <stdio.h>
3:
4: main()
5: {
6:     int    ans;
7:
8:     ans = 2 + 3;
9:
10:    printf("2 + 3 = %d\n", a);
11: }
```

2-1. 「ensyu2 -0.cpp」にはミスがあります。そのミスを修正したプログラム「ensyu2 -1.cpp」を作りなさい。

ヒント：

まずは、どういうエラーが表示されるのかを確かめましょう。

2-2. 「ensyu2 -1.cpp」を改造して、 $5 - 8$  の計算をするプログラム「ensyu2 -2.cpp」を作りなさい。

2-3. 「ensyu2 -2.cpp」を改造して、 $3 \times 1 \div 4 \times 6$  の計算をするプログラム「ensyu2 -3.cpp」を作りなさい。

ヒント：

データ型に注意しましょう。

- 2 4 . 「ensyu2 -3.cpp」を改造して、半径 5 [cm]の円の面積を求めるプログラム「ensyu2 -4.cpp」を作りなさい。なお円周率 = 3 . 1 4 とし、表示は以下の形式に従うこと。

実行結果

半径 5[cm]の円の面積は、  
[cm^2]です。

ヒント :

[円の面積] =  $\pi$  × [半径]<sup>2</sup> =  $\pi$  × [半径] × [半径]  
ですよね。憶えていますか？

- 2 5 . 「ensyu2 -4.cpp」を改造して、半径 1 2 [cm]の半円の面積を求めるプログラム「ensyu2 -4.cpp」を作りなさい。なお円周率 = 3 . 1 4 1 6 とし、表示は以下の形式に従うこと。

実行結果

半径 12[cm]の半円の面積は、  
 $\pi$  × [cm^2]です。

ヒント :

円の面積を半分にすればよいだけです。

- 2 6 . 以下のプログラム「ensyu2 -6.cpp」を実行したら想像とは違い「~aの値は'66'です。」と表示された。「~aの値は'B'です。」と表示されるように「ensyu2 -6.cpp」を修正しなさい。

ensyu2 -6.cpp

```
1: /* char 型変数の表示 */
2: #include <stdio.h>
3:
4: main()
5: {
6:     char    a;
7:
8:     a = 'B';
9:
```

```
10:         printf("char 型変数 a の値は '%d' です\n", a);
11:     }
```

2-7. 以下のプログラムを参考に、「'D' + = 'M'」と表示されるようにこの値を探し出し、修正したプログラム「ensyu2-7.cpp」をつくりなさい。

```
1:  /* char 型変数を計算してみた */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:      char    c;
7:
8:      c = 'D' + 1;
9:
10:     printf("'D' + 1 = '%c'\n", c);
11: }
```

ヒント：

以下のソースコードの を 1 にしてコンパイル・実行する、次に を 2 にしてコンパイル・実行する...と一つ一つ試してみましょう。

いずれ解答が出てきます。

```
1:  /* char 型変数を計算してみた */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:      char    c;
7:
8:      c = 'D' +   ;
9:
10:     printf("'D' +   = '%c'\n", c);
11: }
```