プログラミング言語 I

Programming Language I

－　２０００年度版　－

福岡工業大学
情報工学部 情報工学科
柴田望洋

BohYoh Shibata
Fukuoka Institute of Technology

本資料について

◆ 本資料は、2000 年度・福岡工業大学 情報工学部 情報工学科 1 年生の講義

『プログラミング言語Ⅰ』

の補助テキストとして、福岡工業大学 情報工学部 情報工学科 柴田望洋が編んだものである。

◆ 参考文献・引用文献等は、資料の最後にまとめて示す。

◆ 諸君が本資料をファイルに綴じやすいように、研究室の学生達（卒研生と大学院生）が時間を割いて、わざわざ穴を開けるという作業を行っている（一度のパンチで開けることのできる枚数は限られており、**気の遠くなるような**時間がかかっている）。
必ずＢ５のバインダーを用意して、きちんと綴じていただきたい。

本資料は、Microsoft 社のワープロソフトウェアである Microsoft Word 2000 を用いて作成した。

第1章

1-1 まずは画面に表示

■ コメント

他人が作成したプログラムは、いろいろな意味でとっつきにくいものです。しかし、ソースプログラムに適切なコメントが書かれていれば、読みやすく、理解しやすくなります。また、自分が作ったプログラムの全てを永遠に記憶できるはずはありませんから、コメントを残しておくことは、プログラム作成者自身にとっても重要なことです。

■ 改行文字 ¥n と\n

UNIX などの OS では、円記号である￥の代わりに＼を使うこと。

さて、テキスト **List1-1** を以下のように書きかえる（¥n を削る）と、二つの文字列は、改行されることなく、表示されることになります。確認してみましょう。

■ List1-1 改

```
#include  <iostream.h>

int main(void)
{
    cout << "初めてのC++プログラム。";
    cout << "画面に対して出力を行っています。";

    return (0);
}
```

初めてのC++プログラム。画面に対して出力を行っています。

また、**List1-1** は以下のように実現することもできます。

■ List1-1 別の実現例

```
#include  <iostream.h>

int main(void)
{
    cout << "初めてのC++プログラム。¥n画面に対して出力を行っています。¥n";

    return (0);
}
```

初めてのC++プログラム。
画面に対して出力を行っています。

◆◇◆◇◆◇◆◇◆◇◆◇◆◇◆◇ Column ◇◆◇◆◇◆◇◆◇◆◇◆◇◆◇◆◇◆

Bjarne Stroustrup　…　Ｃ++言語の開発者

Ｃ++言語を開発したのは、AT&T ベル研究所の Dr. Bjarne Stroustup。
彼のホームページの URL は、

```
http://www.research.att.com/~bs/homepage.html
```

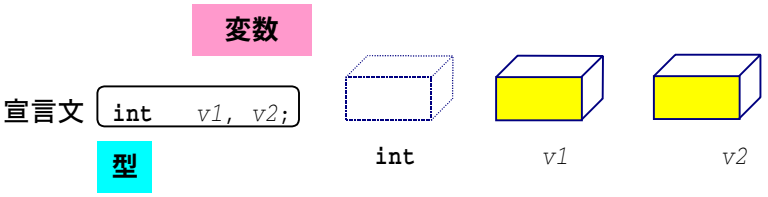
です。

Bjarne Stroustrup 氏のホームページ

1-2　変数とキーボードから入力

■ 変数と宣言

数値などの値を格納するための《箱》が、**変数**（*variable*）です。その《箱》の型には、いくつもの種類があります。テキスト **List1-4** で利用している型は、`int` **型**という**型**（*type*）であり、**整数**を格納することができます。



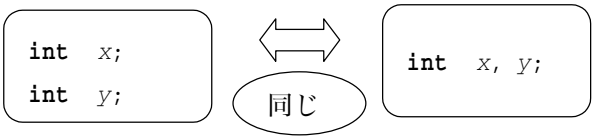
上の図では、型である **int** を点線の箱で、変数 *v1* および *v2* を実線の箱で表しています。点線の箱は、その変数と同じ大きさであり、その型の諸性質を内に秘めた**設計図**（タコ焼きのカタ）であると理解しましょう。その型を持つ変数は、その設計図に基づいて作られた**実体**（カタから作られた本物のタコ焼き）なのです。

すなわち：

> **◆ 重 要 ◆**
> **変数とは、諸性質の設計図である型を基に作られた実体である。**

変数は、その利用に先立って**宣言**（*decralation*）が必要です。宣言の際には、変数に対して、それぞれ異なった名前を与えることになります。

■ 変数の命名規則については、テキスト *p.79* で解説します。



■ 変数の値の入出力

画　面　へ　の　出力	`cout << x;`	`<<`　挿入演算子
キーボードからの入力	`cin >> x;`	`<<`　抽出演算子

■ 記号の読み方

C++のプログラム中には、いろいろな記号が登場します。これらの記号の読み方を、俗称も含めて紹介します。

```
! 感嘆符、エクスクラメーション、びっくりマーク、びっくり、ノット
- マイナス符号、負符号、ハイフン、マイナス、ひく
+ プラス符号、正符号、プラス、たす
* アステリスク、アスタリスク、アスター、かけ、こめ、ほし
/ スラッシュ、スラ、わる
\ 逆斜線、バックスラッシュ、バック　　※JISコードでは¥
¥ 円記号、円、円マーク
% パーセント
. ピリオド、小数点文字、ドット、てん
, コンマ、カンマ
: コロン、ダブルドット
; セミコロン
' 一重引用符、引用符、シングルクォーテーション
" 二重引用符、ダブルクォーテーション
( 左括弧、左丸括弧、左小括弧、パーレン
) 右括弧、右丸括弧、右小括弧
{ 左波括弧、左中括弧、ブレイス
} 右波括弧、右中括弧
[ 左角括弧、左大括弧、ブラケット
] 右角括弧、右大括弧
< 小なり
> 大なり
& アンド、アンパサンド
~ チルダ、なみ、にょろ　　※JISコードでは- （オーバライン）
? 疑問符、はてな、クエッション、クエスチョン
^ アクサンシルコンフレックス、ハット
# シャープ、ナンバー
_ 下線、アンダーライン、アンダーバー、アンダースコア
= 等号、イクオール
| 縦線
```

■ 演習問題（1）

◆ 人間が文字の並びとして作成するプログラムを格納したファイルを (1) ファイルと呼ぶ。それを実行するためには、0 と 1 すなわちビットの並びで構成される (2) プログラムへと変換するための (3) や (4) などの手続きが必要である。

◆ 読み手に理解して欲しい注釈は、(5) として、プログラム中に書きこむことができる。

その記述形式としては、// 以降に注釈を書く方法と、/* と */ の間に書く方法の２つがある。前者では、その行の終端までが (5) となるのに対し、後者は、複数の行にまたがることができる。

なお、(5) を書くことや、その内容によって、プログラムの動作が変わることは (6)。

（選択肢）{ a）ある。 b）ない。

◆ 画面に『ABCD』と出力・改行するのは、以下のようなプログラム（部分）となる。

(7) << "ABCD¥n";

ここで << は、(8) 演算子と呼ばれ、画面に対する文字の流れとみなすことのできる《出力 (9) 》に対して文字を挿入する働きをする。なお、"ABCD" のように、二重引用符 " で囲まれた文字の並びのことを (10) と呼ぶ。

◆ 画面に対して、右図のような出力をするには、以下のようなプログラム（部分）となる。

(11) << " (12) ";

```
ABC
DEFG

HIJK
```

◆ 入出力を行うために必要な情報が格納されているヘッダである iostream.h の内容をソースプログラム中に取り込むには、

(13) <iostrem.h>

という指令が必要であり、このように、ヘッダの内容を取り組むことを (14) という。

◆ 整数値などの、値を格納するための“箱”であると考えることのできるオブジェクト（変数）は、それを利用する前に必ず (15) を行わなければならない。

たとえば、整数を格納するためのオブジェクト x を利用するには、以下のような (15) が必要であり、このような文のことを特に (16) と呼ぶ。

(17) x;

(1)	
(2)	
(3)	
(4)	
(5)	
(6)	
(7)	
(8)	
(9)	
(10)	
(11)	
(12)	
(13)	
(14)	
(15)	
(16)	
(17)	

■ 演習問題（2）

◆ キーボードから読み込んだ整数値を、変数 *x* に格納するプログラム（部分）は、以下のようになる。

```
(1) >> x;
```

ここで >> は、(2) 演算子と呼ばれ、キーボードからの文字の流れとみなすことのできる入力ストリームから文字を取り出す働きをする。

◆ 画面に対して、右図のような出力をするには、以下のようなプログラム（部分）となる。

```
(3) << "(4)";
```

温
故
知
新

◆ 画面に対して、右図のような出力をするには、以下のようなプログラム（部分）となる。

```
(5) << "福(6)";
(5) << "工(6)";
(5) << "大(6)";
```

福
工
大

(1)	
(2)	
(3)	
(4)	
(5)	
(6)	
(7)	
(8)	
(9)	
(10)	
(11)	
(12)	
(13)	
(14)	

◆ 以下に示すのは、キーボードから二つの整数を読み込み、それらの和と、最初の値から２番目の値を引いた値を出力するプログラムである。

```
#include (7)

int (8)(void)
{
    (9)  x;          // 整数 x の宣言
    (9)  y;          // 整数 y の宣言

    (10) << "ｘを入力してください：";     // x の入力をうながす
    (11) >> x;                          // x に整数を読み込む

    (10) << "ｙを入力してください：";     // y の入力をうながす
    (11) >> y;                          // y に整数を読み込む

    (10) << "ｘ＋ｙは" << (12) << "です。¥n";
    (10) << "ｘ－ｙは" << (13) << "です。¥n";

    (14) (0);
}
```

■ プログラム作成演習（1）

(1-1) 右に示すような表示を行うプログラムを作成せよ。ただし、名前は“柴田望洋”でなく、自分の名前とすること。

```
私の名前は柴田望洋。
２年１組の学生です。
```

(1-2) 右に示すような表示を行うプログラムを作成せよ。ただし、名前および学籍番号は、自分のものとすること。

なお、2 行目と 3 行目の間が 1 行空いていることに注意せよ。

```
氏　　名：柴田望洋。
学籍番号：99A1001

趣　　味：パソコン
```

(1-3) 右に示すような表示を行うプログラムを作成せよ。

すなわち、自分の名前を漢字表記したものを、各行に 1 文字ずつ表示すること。なお、プログラム中に cout を使うのは 1 回限りであるものとし、２回以上使わないこと。

```
柴
田
望
洋
```

```
□ <iostream.h>

□ main(□)
{
    cout << "柴□田□望□洋□";          // １行に１文字ずつ表示

    return (0);
}
```

(1-4) 右に示すように、読み込んだ整数値をそのまま表示するプログラムを作成せよ。

```
整数を入力して下さい：123↵
あなたは123と入力しましたね。
```

なお、網掛け部は、キーボードから入力を行う箇所であり、↵は Enter キーの入力を示している。もし、53↵と入力された場合には、『あなたは 53 と入力しましたね。』と表示しなければならない。

```
□ <iostream.h>

□ main(□)
{
    int □;              // x は整数

    cout << "整数を入力してください：";      // 入力をうながす
    cin □ x;                                // x に整数を読み込む

    cout << "あなたは" << □ << "と入力しましたね。¥n";

    return (0);
}
```

(1-5) 右に示すように、キーボードから読み込んだ値に 10 を加えた値を表示するプログラムを作成せよ。

```
整数を入力して下さい：57
その値に10を加えると67です。
```

(1-6) 右に示すように、キーボードから読み込んだ値から 100 を減じた値を表示するプログラムを作成せよ。

```
整数を入力して下さい：57
その値から100を引くと-43です。
```

(1-8) キーボードから三つの整数値を読み込んで、その和を求めて表示するプログラムを作成せよ。なお、整数値を格納する変数名は *a*, *b*, *c* とすること。

```
整数を入力せよ(1)：12
整数を入力せよ(2)：9
整数を入力せよ(3)：30
それらの和は51です。
```

(1-9) キーボードから四つの整数値を読み込んで、その和を求めて表示するプログラムを作成せよ。なお、整数値を格納する変数名は *w*, *x*, *y*, *z* とすること。

1-3 型

■ 型

int 型 … 整数を格納 例： 5

double 型 … 浮動小数点数（実数）を格納 例： 3.14

整数変数に実数値を代入すると、小数点以下の部分は切り捨てられる。

■ 代入演算子

= 演算子を用いることによって、代入を行うことができます。

◆ 単純代入演算子

x = *y*	*y* の値を *x* に代入する。

■ 算術演算子

+, -, *, /, %の演算子を用いることによって、四則演算を行うことができます。

整数どうしの除算例

5 / 3 → 1　　7 / 2 → 3　　12 / 4 → 3

5 % 3 → 2　　7 % 2 → 1　　12 % 4 → 0

◆ 加法演算子

x + *y*	*x* に *y* を加えた値を求める。
x - *y*	*x* から *y* を引いた値を求める。

◆ 乗法演算子

x * *y*	*x* に *y* をかけた値を求める。
x / *y*	*x* を *y* で割った値を求める。
x % *y*	*x* を *y* で割った余りを求める（**整数にのみ適用できる**）。

数学と同じで、単項の+および-演算子があります。

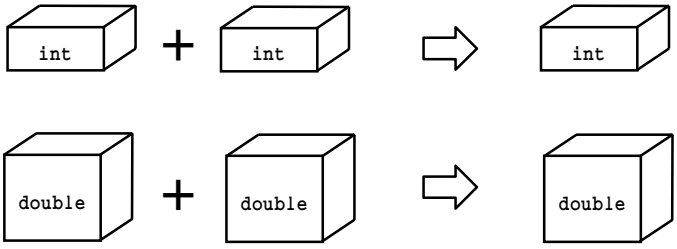
◆ 単項算術演算子

+ *x*	*x* そのものの値。
- *x*	-*x* の値。

■ 算術演算と型

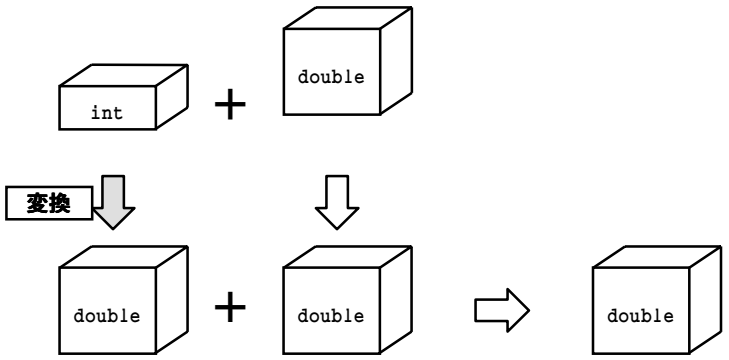
● 同一型のオペランドの演算 ●

int + **int** の演算、**double** + **double** の演算では、演算の結果の型は、演算の対象となるオペランドと同じ型になります（下図)。もちろん、この規則は、原則として他の演算（-, *などの演算）にも適用されます。



● 異なる型のオペランドの演算 ●

int + **double** や、**double** + **int** のように、一方のオペランドが **int** 型で、他方のオペランドが **double** 型である場合は、**int** 型のオペランドの値が **double** 型に“格上げ”されて、**double** 型どうしの演算として行われます。したがって、その演算によって得られる結果も **double** 型となるのです。



したがって、*a*, *b* が **int** 型で、*x*, *y* が **double** 型であるとき、以下のようになります。

sizeof(*a* + *b*) → **sizeof**(**int**)
sizeof(*x* + *y*) → **sizeof**(**double**)
sizeof(*a* + *x*) → **sizeof**(**double**)

■ 演算子とオペランド

+, -, =などのように、いろいろな演算を行う記号のことを**演算子**（*operator*）と呼び、その演算の対象となる式を**オペランド**（*operand*）と呼びます。たとえば、以下の式

```
v1 + v2
```

において、演算子は + であり、そのオペランドは *v1* と *v2* です。一般に、最初のオペランドである *v1* を、**第１オペランド**や**左オペランド**と呼び、演算子の右側のオペランドである *v2* を、**第２オペランド**や**右オペランド**と呼びます。

■ 式と文

式（*expression*）とは、変数や定数、それらを演算子で結合したものです。たとえば、

```
vx + 32              // 加法式
```

では、*vx*, 32, *vx* + 32 のいずれもが式なのです。また、

```
vc = vx + 32         // 代入式：vxと32の和をvcに代入
```

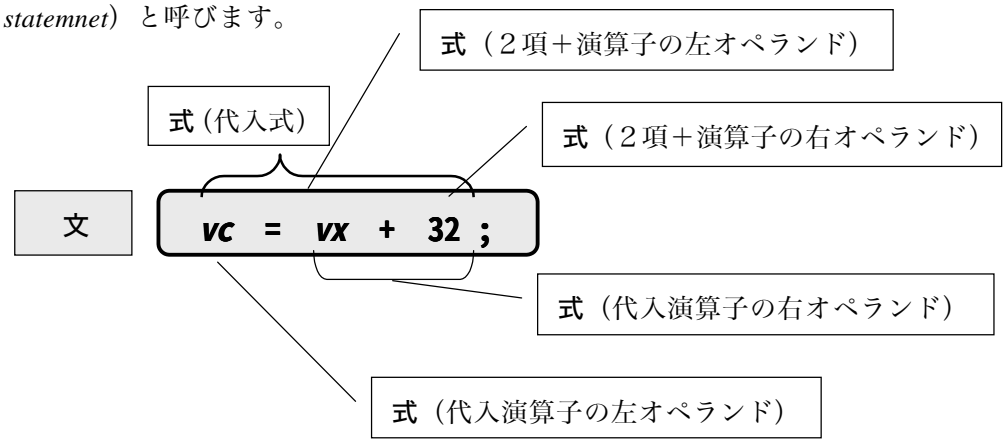
では、*vc*, *vx*, 32, *vx* + 32, *vc* = *vx* + 32 のいずれも式とみなすことができます。

代入 *vc* = *vx* + 32 では、代入演算子のオペランドは、*vc* と *vx* + 32 との 2 つです。代入演算子を用いた、このような式を**代入式**（*assignment expression*）と呼びます。

なお、テキスト p.11 で解説したように、原則として、文の末尾には、セミコロン ; が必要ですから、以下の形となって、はじめて正しい「文」となるのでしたね。

```
vc = vx + 32;        // 式文
```

このように、式にセミコロンを付けることによって文となったものを、**式文**（*expression statemnet*）と呼びます。



■ 演習問題（3）

◆ **int** は整数のみを表現することができ、**double** は少数部分を持つ実数をも取り扱うことができる。このような **int** や **double** が有する諸性質は、(1) という概念で捉えられる。

◆ **int** (1) の変数 *x* に対して整数値 5 を代入するのが、以下のプログラム（部分）である。

```
x = 5;
```

ここで = は (2) 演算子と呼ばれ、右オペランドの値を、左オペランドに代入する働きを持つ。なお、

```
x = 3.45;
```

のように、実数値を代入すると、小数点以下の部分は (3) 。

（選択肢）
- a）切り上げられる。
- b）四捨五入される。
- c）切り捨てられる。

◆ *x* と *y* が **int** (1) の変数であり、それぞれの値が 17 と 4 であるとき、*x* / *y* によって得られる値は (4) 、*x* % *y* によって得られる値は (5) である。

◆ / や % のように、演算を行う記号のことを (6) と呼び、その演算の対象となる式のことを (7) と呼ぶ。

◆ 5 とか 13 とかいった式を (8) 定数と呼び、3.14 とか 2.95 といった式を (9) 定数と呼ぶ。

◆ 任意の (1) や変数、式などの大きさは、(10) 演算子によって求めることができる。この演算子が生成する「大きさ」は、プログラムが実行される環境において“文字”を表すのに必要な大きさの何倍であるかで表される。

したがって、(10) (char) の値は (11) である。

(1)	
(2)	
(3)	
(4)	
(5)	
(6)	
(7)	
(8)	
(9)	
(10)	
(11)	

■ 演習問題（4）

◆ 以下に示すのは、読み込んだ正の整数値の下1桁の数字を表示するプログラムである。

テキスト 演習1-4

```
   (1)        <iostream.h>

(2)  main(  (3)  )
{
    int  (4) ;            // x は整数

    cout << "正の整数値を入力してください：";
    cin  (5)  x;

    cout << "その数の下1桁の数字は"
         << x  (6)  10 << "です。¥n";

    return (0);
}
```

(1)	
(2)	
(3)	
(4)	
(5)	
(6)	
(7)	
(8)	
(9)	
(10)	
(11)	
(12)	
(13)	
(14)	
(15)	
(16)	
(17)	

◆ 以下に示すプログラム部分は、整数オブジェクトに対して、実数値を代入した後に、その値を表示するものである。

```
int  x1, x2;

x1 = 2.95;
x2 = 3.14;

cout << "x1 = " << x1 << '¥n';
cout << "x2 = " << x2 << '¥n';
```

```
x1 =  (7)
x2 =  (8)
```

◆ "x1=" のように二重引用符記号 " で囲んだ文字の並びを (9) と呼ぶのに対し、'¥n'や 'A' のよに、一重引用符記号 ' で囲んだ文字を (10) と呼ぶ。

◆ 以下に示すのは、文字の大きさを確認するプログラムである。

```
cout << "sizeof('X') ：" << sizeof('X')  << '¥n';
cout << "sizeof(char)：" << sizeof(char) << '¥n';
```

```
sizeof('X')：  (11)
sizeof(char)：  (12)
```

◆ いま sizeof(**int**)が2で、sizeof(**double**)が8であるとする。変数aが **int** 型で、変数bがdouble型であるとき、sizeof(a)の値は (13) であり、sizeof(b)の値は (14) であり、sizeof(a+3)の値は (15) であり、sizeof(a+3.5)の値は (16) であり、sizeof(x+3.5)の値は (17) である。

■ プログラム作成演習（2）

(1-10) 右に示すようなやり取りによって、キーボードから読み込んだ値を 3 で割った余りを表示するプログラムを作成せよ。

```
整数を入力して下さい：14
その数を3で割った余りは2です。
```

(1-11) 右に示すようなやり取りによって、キーボードから読み込んだ、cm で表された長さをインチに換算した長さを表示するプログラムを作成せよ。ただし、1 インチは 2.54cm であるとする。

```
長さをcmで入力して下さい：8.89
その長さは3.5インチです。
```

(1-12) 右に示すようなやり取りによって、キーボードから読み込んだ、cm で表された身長に対する標準体重を表示するプログラムを作成せよ。ただし、標準体重は(身長 - 100) * 0.9 で求めるものとする。

```
身長を入力して下さい：178.4
標準体重は70.56kgです。
```

(1-13) 右に示すようなやり取りによって、キーボードから読み込んだ整数値の 2 乗値および 3 乗値を表示するプログラムを作成せよ。

```
整数を入力して下さい：5
その数の２乗は25です。
その数の３乗は125です。
```

(1-14) 右に示すようなやり取りによって、キーボードから読み込んだ２つの**実数値**の平均値を表示するプログラムを作成せよ。

```
実数を入力して下さい：5.2
実数を入力して下さい：3.8
それらの平均は4.5です。
```

(1-15) 右に示すようなやり取りによって、キーボードから読み込んだ２つの**整数値**の平均値を表示するプログラムを作成せよ。

※ 整数 / 整数の結果は整数となってしまうことに注意。

```
整数を入力して下さい：5
整数を入力して下さい：4
それらの平均は4.5です。
```

■　間違い探し

以下に示すのは、キーボードから２つの実数値を読み込み、その和、差、積、商を表示するプログラムです。

たくさん間違いがあるけど、何個所見つけることができるかな？

テキスト 演習 1-5 間違い付

```
1  #inculde   <instream.h>
2
3  int main(void)
4  {
5      Double  x. y;      /* 浮動小数点型 /*
6
7      cout << "ｘを入力してください：" ;       // xの入力をうながす
8      cin << x;                                // xに実数を読み込む
9
10     cout << "ｙを入力してください：”  ;      // yの入力をうながす
11     cin << y;                                // yに実数を読み込む
12
13     cout << "x + y = " << x + y << "/n":      // 和
14     cout << "x - y = " << x - y << "/n":      // 差
15     cout << "x * y = " << x * y << "/n":      // 積
16     cout << "x % y = " << x % y << "/n":      // 商
17
18     retrun (0)
19 }
```

【解答】

1 行目… **include** と iostream.h の綴りが間違っている。

5 行目… 型名は Double でなく double でなければならない。
x と *y* を区切る記号は ． でなく ， でなければならない。
コメントの終了を表す記号は /* でなく */ でなければならない。

7 行目… 文の終わりを示す記号 ;は全角文字ではなく半角文字でなければならない。

8 行目… cin からの読み込みは、<< でなくて抽出演算子 >> でなければならない。

10 行目… 文字列リテラルを結ぶ記号 〃 は全角ではなく半角文字でなければならない。

11 行目… cin からの読み込みは、<< でなくて抽出演算子 >> でなければならない。

13〜16 … 文の終わりを示す記号は、コロン:ではなくセミコロン;でなければならない。
改行文字は、/n ではなくて、\n（あるいは¥n）でなければならない。

16 行目… 商を求める演算子は % ではなくて / である。

18 行目… **return** の綴りが間違っている。
文の終わりを示す記号 ； が抜けている。