第１章

まずは慣れよう

問題 1-1

整数値15と37の和を計算して、その値である「52」を表示するプログラムを作成せよ。


```
/*
    整数値15と37の和を表示
*/

#include  <stdio.h>

int main(void)
{
    printf("%d\n", 15 + 37);          /* 整数値15と37の和を10進数で表示 */

    return (0);
}
```

実行結果

```
52
```

▶　本書に示すプログラムでは、ある規則にもとづいて、*斜体*、太字、**ゴシック体**、淡い丸文字などを使い分けています。これは、読者のみなさんが読みやすいようにとの配慮によるものです。みなさんがプログラムを打ち込む際には、このような区別は不要です。

ソースプログラム

私たちが文字の並びとして作るプログラムを**ソースプログラム**（*source program*）と呼びます。ソースプログラムを格納・保存したファイルである**ソースファイル**（*source file*）には拡張子.cを与えるという慣習がありますから、本問のソースファイルの名前は、たとえばex0101.cとしましょう。

一般に、文字の並びとして作成したソースプログラムは、コンピュータが理解できる形式である**ビット**の並び、すなわち0と1の並びに変換するための**翻訳**などの作業が必要です。その作業方法は、処理系によって異なりますので、みなさんが利用している処理系のマニュアルなどを参照してください。

注釈（コメント）

プログラム中/*から*/までの淡い丸文字で示した部分が**注釈**（コメント）（*comment*）です。このプログラムのように、作成者を含め、その読み手に伝えたいことを、日本語や英語などの簡潔な言葉で書き込んでおきましょう。

なお、注釈の有無や、その内容によって、プログラムの動作が変わるといったことはありません。

▶　***printf***による表示に関しては、右ページで解説します。

問題 1-2

▶『明解』演習 1-1 (p.6)

整数値15から37を引いた値を計算して「15から37を引いた値は-22です。」と表示するプログラムを作成せよ。

```
/*
    整数値15から37を引いた値を親切に表示
*/

#include  <stdio.h>

int main(void)
{
    printf("15から37を引いた値は%dです。\n", 15 - 37);

    return (0);
}
```

実行結果

```
15から37を引いた値は-22です。
```


関数

C言語には、多くの**関数**（*function*）が用意されており、それらをうまく利用することによって、素早く手短にプログラムを作ることができます。**関数呼出し**（*function call*）は、関数に処理を行ってもらうための**依頼**であると理解しましょう。

その際に必要な**補助的な指示**は、**( )**の中に**実引数**（*argument*）として与えます。なお、実引数が複数ある場合は、コンマ文字 , で区切ります。

printf … 書式化して表示を行う関数

前問および**本問**のプログラムは、表示を行うために用意されている***printf***関数（一般に***printf***はプリントエフなどと呼ばれます。末尾の f は書式＝*format* に由来します）を呼び出すことによって、計算結果を表示しています。

この***printf***関数に対して与える最初の実引数を**書式文字列**（*format string*）と呼びます。その書式文字列中の **%d** は、

続く実引数の値を 10 進数で表示せよ

と、書式を指示するための**変換指定**（*conversion specification*）です。書式文字列中の変換指定でない文字は、基本的にはそのまま出力されます（下図）。

▶ **%d** の d は、10 進数＝ *decimal* に由来します。

```
printf("%d\n", 15 + 37);
52
```

問題 1-1 の出力

```
printf("15から37を引いた値は%dです。\n", 15 - 37);
15から37を引いた値は-22です。
```

問題 1-2 の出力

▶ 書式文字列中の **\n** については、**次問**で解説します。なお、***printf*** 関数は多機能ですから、少しずつ説明していきます（全機能に関しては、『明解』*p.318* ～ *321* にまとめています）。

▶『明解』演習 1-2 （p.9）

問題 1-3

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、プログラム中、***printf***関数の呼出しは、1回限りとする。

```
風
林
火
山
```



```
/*
    “風林火山”を1行に1文字ずつ表示
*/

#include  <stdio.h>

int main(void)
{
    printf("風\n林\n火\n山\n");

    return (0);
}
```

実行結果

```
風
林
火
山
```

\n … 改行を表す拡張表記

問題 1-1と**問題 1-2**のプログラムでは、いずれも***printf***関数に与える書式文字列の末尾に**\n**が付いています。これは、**改行**（*new line*）を表すための特別な表記です。改行を出力すると、続く表示は、次の行の先頭から始まります。

▶ 日本で多くのパソコンに採用されているJISコードという文字体系では、逆斜線 \ の代わりに、円記号¥を使います。したがって、みなさんの環境によっては、¥を使わなければならないかもしれません。その場合は、本書のすべての \ を¥と読みかえてください。

\nのように複数の文字を並べて一つの文字を表す特別な表記を**拡張表記**（エスケープシーケンス）（*escape sequence*）と呼びます。

さて、もしも**問題 1-1**のプログラムで、***printf***関数の呼出し部が、

```
printf("%d", 15 + 37);
```

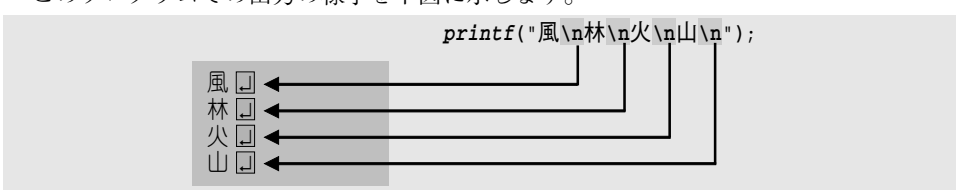
```
▷ex0101↵
52▷
```

となっており、**\n**が無かったとしましょう。プログラムを実行すると、多くの実行環境では、右上に示すように、プログラムの出力結果である52の直後にプロンプトがくっついてしまいます。

したがって、プログラムの最後の出力では、改行を出力した方がよいのです。

▶ この例では、実行プログラム名がex0101であると仮定しています。なお、▷はオペレーティングシステムのプロンプトであり、MS-DOSであればA>などの記号が、UNIXであれば%などの記号が表示されます。

このプログラムでの出力の様子を下図に示します。



▶ 図中の↵は、改行のイメージを表すものであり、実際に目に見えるわけではありません。

▶『明解』演習 1-3 （p.9）

問題 1-4

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、プログラム中、***printf***関数の呼出しは、1回限りとする。

```
もしもし。
こんにちは。

それでは。
```



```c
/*
    挨拶を表示
*/

#include  <stdio.h>

int main(void)
{
    printf("もしもし。\nこんにちは。\n\nそれでは。\n");

    return (0);
}
```

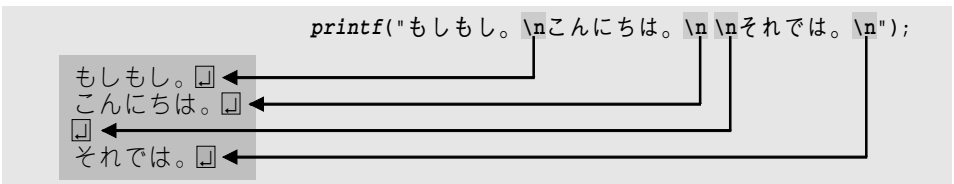
実行結果

```
もしもし。
こんにちは。

それでは。
```

空行の出力

このプログラムは、本質的に**前問**と同様です。空の行を表示するには、**\n**を出力すればよいですね。



なお、**前問**や**本問**のように、書式化して出力する値がない場合は、***printf***関数には書式文字列だけを与えます（その中に変換指定を入れないように）。

printf関数の呼出しを1回に限らないのであれば、表示する部分は、以下のようにしても構いません。

```c
printf("もしもし。\n");
printf("こんにちは。\n\n");
printf("それでは。\n");
```

```c
printf("もしもし。\nこんにちは。\n");
printf("\nそれでは。\n");
```

▶ これらは一例であって、実現のバリエーションは無数です。

文

これまでの全てのプログラムがそうですが、***printf***関数の呼出しだけでなく、

```c
return (0);
```

にもセミコロンが付いています。これは、日本語での句点 。に相当するものです。

最後に句点があって、日本語として正しい文となるように、C言語でも、原則として、セミコロンを与えることによって正しい**文**（*statement*）となるのです。

問題 1-5

右に示すように、「警報!!警報!!」と表示しながら警報を発するプログラムを作成せよ。

```
警報!!警報!!
```



```
/*
    警報を発する
*/

#include  <stdio.h>

int main(void)
{
    printf("警報!!\a警報!!\a\n");          /* 表示とともに警報を発する */

    return (0);
}
```

実行結果

```
警報!!警報!!
```

▶ プログラムを実行する環境によっては、警報（音でなく視覚的なものである場合もありますが、普通はいわゆる《ビープ音》です）が発せられないことや、二つの警報がまとめて1回だけ発せられることもあります。

\a … 警報を表す拡張表記

警報（*alert*）を表す拡張表記が**\a**です。

▶ 拡張表記の一覧表は、**第8章**（*p.189*）に示します。

文字列リテラル

"ABC"や**"こんにちは。"**のように、一連の文字を二重引用符**"**で囲んだものは、ひと続きの文字の並びを表す**文字列リテラル**（*string literal*）です。

▶ 本来、文字列リテラルの中に漢字などの全角文字を使うのは、規則違反です。もっとも、私たちが日本で使う処理系の大部分は、全角文字が利用できるようになっています。読者のみなさんが読みやすいようにとの配慮から、本書では全角文字を使っています。

プログラムの構造

これまでのプログラムは全て、下図のような構造となっています。現在の段階で、全てを理解する必要はありません。少しずつ理解していきましょう。

```
#include  <stdio.h>

int main(void)
{

    return (0);
}
```

- **#include**は、**第6章**で学習します。
- **main**関数は、**第6章**で学習します。
- 複合文{ }は、**第3章**で学習します。
- **return**文は、**第6章**で学習します。

記号文字の読み方

C言語のプログラムでは、英数字以外にも多くの記号文字を使います。C言語で利用する記号文字の読み方を、俗称も含め以下の表に示します。

!	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

問題 1-6

整数を格納するint型の変数に適当な値を代入し、その値を表示するプログラムを作成せよ。


```
/*
    変数に整数値を代入して表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;                                /* 変数noはint型の変数 */

    no = 75;                                /* noに75を代入 */

    printf("noの値は%dです。\n", no);       /* noの値を表示 */

    return (0);
}
```

実行結果

```
noの値は75です。
```

変数

プログラム中に埋め込まれた値である**定数**（*constant*）とは異なり、変数には、自由に値を入れたり取り出したりすることができます。数値などを格納する『箱』であるとも考えられる変数を使うには、

```
int  no;
```

といった**宣言**（*declaration*）を事前に行います（一般にintはイントと呼ばれます）。

この宣言によって、*no*という名前の箱が一つ用意されることになります（下図）。

▶ intは、整数＝*integer*に由来します。

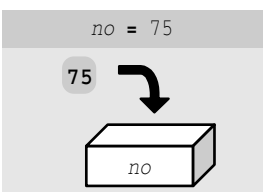
この箱には、整数値のみを格納することができ、*no*は“int型”であると呼ばれます。このプログラムでの変数名は*no*ですが、自由に名前を与えることができます。たとえば*x*のように、1文字だけの名前でも構いません。

▶ 名前（識別子）の命名規則に関しては、**第4章**（*p.76*）を参照してください。

代入

このプログラムで初登場の記号 = は、**右側の値を左側に代入しなさい**という意味です。

したがって、変数*no*には75が代入され、その値は75となります。



問題 1-7

右に示すように、読み込んだ整数値を**int**型の変数に格納し、その値を表示するプログラムを作成せよ。

```
noの値を入力してください：32↵
noの値は32となっています。
```



```
/*
    読み込んだ整数の値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;

    printf("noの値を入力してください：");          /* 整数値の入力を促す */
    scanf("%d", &no);                              /* 整数値を読み込む */

    printf("noの値は%dとなっています。\n", no);    /* noの値を表示 */

    return (0);
}
```

実行例

```
noの値を入力してください：32↵
noの値は32となっています。
```

scanf … 読込みを行う関数

画面への表示を行う***printf***関数と対照的に、キーボードから数値などの読込みを行うのが***scanf***関数（一般に***scanf***はスキャンエフなどと呼ばれます）です。

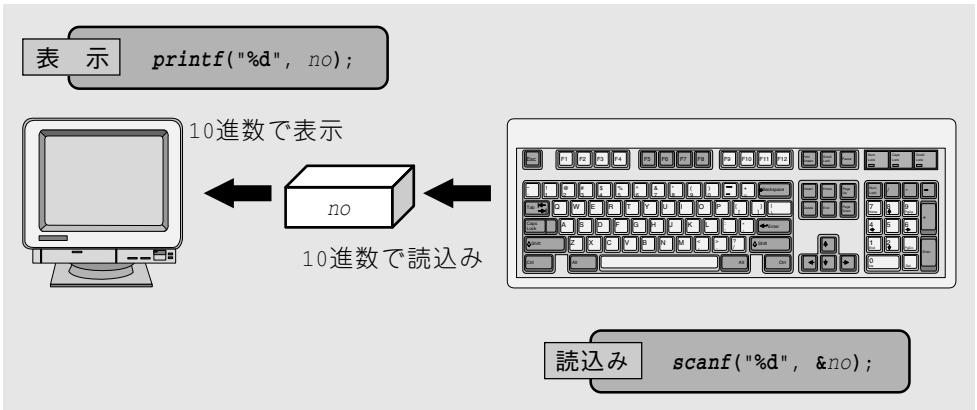
変換指定**%d**は、***printf***の場合と同様、10進数の指定であり、

キーボードから10進数を読み込んで、その値を*no*に格納してください。

と依頼していることになります。

ただし、***printf***の場合とは異なり、変数名の前に記号**&**を付けなければなりません。

▶ **&**については、**第10章**（*p.222*）で解説します。なお、***scanf***関数は多機能ですから、少しずつ説明していきます（全機能に関しては、『明解』*p.322*～*325*にまとめています）。



問題 1-8

▶『明解』演習 1-4 (p.13)

右に示すように、読み込んだ整数値に10を加えた値を表示するプログラムを作成せよ。

```
整数を入力してください：57↵
その数に10を加えると67です。
```



```c
/*
    読み込んだ整数値に10を加えた値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;

    printf("整数を入力してください：");              /* 整数値の入力を促す */
    scanf("%d", &no);                               /* 整数値を読み込む */

    printf("その数に10を加えると%dです。\n", no + 10);

    return (0);
}
```

実行例

```
整数を入力してください：57↵
その数に10を加えると67です。
```

加算

問題1-1からずっと使ってきましたが、+という記号は、加算(足し算)を行う働きをもっています。

▶ このような記号を**演算子**と呼びます。**次章**で解説します。

Column … C言語の歴史

C言語は、1972年頃にDennis M. Ritchieによって開発されました。彼は当時、Ken Thompsonらと共同で、ミニコンピュータのオペレーティングシステムであるUNIXの開発に携わっていました。このOSは、初期の段階ではアセンブリ言語を用いて開発されましたが、その後、C言語で書き直されました。

初期のUNIXを移植するために開発されたのがC言語ですから、ある意味では

「C言語はUNIXの副産物である。」

ということになります。

そのUNIX本体だけでなく、その上で動作する多くのアプリケーションも、C言語で開発されることになります。

そのため、C言語は、まずUNIXの世界で広まりました。しかし、その勢いは全くとどまらず、次第に大型コンピュータやパーソナルコンピュータの世界にも普及していったのです。

さて、RitchieはBrian W. Kernighanと共に、C言語の解説書である

“The C Programming Language”, Prentice-Hall, 1978

を著しました。C言語の設計者が自ら著したこの書は、C言語のバイブルとして多くの人々に読まれることになります。そして、著者のイニシャルに由来して、“K&R”という愛称で親しまれます。

K&Rの巻末には、C言語の言語仕様を規定した“Reference Manual(参照マニュアル)”が付録として採録されています。ここに記された言語仕様が、C言語の標準的な仕様であると考えられることになりました。

問題 1-9

▶『明解』演習 1-5 （p.13）

右に示すように、読み込んだ整数値から10を減じた値を表示するプログラムを作成せよ。

```
整数を入力してください：57↵
その数から10を減じると47です。
```



```
/*
    読み込んだ整数値から10を減じた値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;

    printf("整数を入力してください：");              /* 整数値の入力を促す */
    scanf("%d", &no);                               /* 整数値を読み込む */

    printf("その数から10を減じると%dです。\n", no - 10);

    return (0);
}
```

実行例

```
整数を入力してください：57↵
その数から10を減じると47です。
```

減算

加算を行う + と対照的に、減算（引き算）を行うのが - です。

前問の加算 *no* + 10 は、足す順番を逆にして 10 + *no* としても構いませんが、**本問**の減算を 10 - *no* としてはいけませんね。

Column … 標準規格

K&Rの参照マニュアルに規定されている言語仕様は、曖昧で紛らわしい部分が少なからずありました。したがって、C言語の普及とともに、多くの『方言』が生まれ、独自の拡張機能をもつC言語が氾濫することになります。

本来のC言語は、可搬性が高いこと、すなわち、あるコンピュータ用にC言語で作ったプログラムを、他のコンピュータ用に移植しやすいということを大きな特長としていました。しかし、方言の発生と相まって、満足な可搬性が維持できなくなってきます。

そこで当然の流れとして、C言語の世界標準規格を決めようという動きがおこります。言語の仕様を全世界で共通化しようとするのですから、その作業はとても慎重なものとなりました。国際標準化機構 ISO（International Organization for Standardization）と、米国国内規格協会ANSI（American National Standards Institute）が、協力して作業を行ったのです。

1989年12月には、米国内の規格である

ANSI X3.159-1989

American National Standard for Information Systems - Programming Language-C

が制定され、1990年12月には、世界規格である

INTERNATIONAL STANDARD ISO/IEC 9889 : 1990(E) Programming Languages-C

が制定されました。これらは、体裁は違うものの、内容としては同一のものです。さらに、日本では、やはり同一の内容をもつ規格である

JIS X3010-1993 プログラミング言語C

が1993年に制定されました。

▶『明解』演習 1-6 (p.15)

問題 1-10

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、表示には***printf***関数ではなく***puts***関数を利用すること。

```
風
林
火
山
```



```
/*
    “風林火山”を1行に1文字ずつ表示（puts関数版）
*/

#include  <stdio.h>

int main(void)
{
    puts("風\n林\n火\n山");

    return (0);
}
```

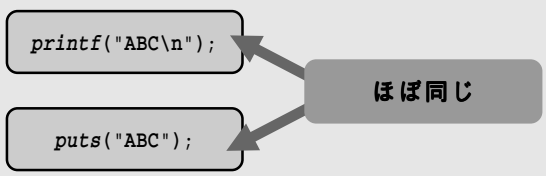
実行結果

```
風
林
火
山
```

puts … 表示を行う関数

本プログラムで初めて使用した***puts***関数（末尾のsは文字列＝*string*に由来し、一般に***puts***はプットエスなどと呼ばれます）は、実引数として与えられた文字の並びを出力して、さらに最後に改行を行います。

すなわち、下図にも示すように、***puts("…")***は、***printf("…\n")***とほぼ同じ働きをします。



書式化の必要がなく、改行をしたい場合は、なるべく***printf***関数ではなくて***puts***関数の方を使用しましょう。

▶ ***puts***関数に与えることのできる実引数は一つだけであることに注意しましょう。

もちろん、以下のように***puts***関数の呼出しを複数回に分けても構いません。

```
puts("風\n林");
puts("火\n山");
```

```
puts("風");
puts("林");
puts("火");
puts("山");
```

▶ これらは一例であって、実現のバリエーションは無数です。

問題 1-11

▶『明解』演習 1-7 （p.15）

右に示すように、読み込んだ二つの整数値の積を表示するプログラムを作成せよ。

```
二つの整数を入力してください。
整数１：27↵
整数２：35↵
それらの積は945です。
```



```c
/*
    読み込んだ二つの整数値の積を求めて表示
*/

#include  <stdio.h>

int main(void)
{
    int  n1, n2;
    int  seki;      /* 積 */

    puts("二つの整数を入力してください。");
    printf("整数１：");   scanf("%d", &n1);
    printf("整数２：");   scanf("%d", &n2);

    seki = n1 * n2;                          /* n1とn2の積をsekiに代入 */

    printf("それらの積は%dです。\n", seki);  /* 積を表示 */

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数１：27↵
整数２：35↵
それらの積は945です。
```

乗算

乗算を行うための記号が*です(×ではありません)。なお、求めた積をいったん変数*seki*に代入せず、***printf***関数に直接渡して

```c
printf("それらの積は%dです。\n", n1 * n2);
```

と表示することもできます。この場合、変数 *seki* の宣言も不要です。

求めた積の値を、後で利用するようなプログラムでは、その値を変数に入れておけばよいでしょうが、一回表示するだけの“使い捨て”の運命であることが明確であれば、わざわざ変数を使うまでもないでしょう。ケースバイケースです。

複数の変数を宣言

二つの変数 *n1* と *n2* は、コンマ , で区切って宣言されています。これで、*n1* という名前の変数と *n2* という名前の変数が用意されることになります。

もちろん右に示すように、二つの変数を個別に宣言しても構いません。

```c
int  n1;
int  n2;
```

▶ 各行に一つずつ宣言を書くことによって、その宣言に対する注釈を記入しやすくなりますし、宣言の追加や削除もスムーズに行えるようになります。ただし、プログラムの行数は増えてしまいます。臨機応変に使い分けましょう。

このプログラムには、二つの文がおかれている行があります（網掛け部）。このように、C言語では、自由度の高い自由形式（『明解』*p.84* ～*85* 参照）の表記ができるようになっています。

問題 1-12

▶『明解』演習 1-8 （p.15）

右に示すように、読み込んだ三つの整数値の和を表示するプログラムを作成せよ。

```
三つの整数を入力してください。
整数 1 ：7↵
整数 2 ：15↵
整数 3 ：23↵
それらの和は45です。
```



```
/*
    読み込んだ三つの整数値の和を求めて表示
*/

#include  <stdio.h>

int main(void)
{
    int  n1, n2, n3;

    puts("三つの整数を入力してください。");
    printf("整数 1 ："); scanf("%d", &n1);
    printf("整数 2 ："); scanf("%d", &n2);
    printf("整数 3 ："); scanf("%d", &n3);

    printf("それらの和は%dです。\n", n1 + n2 + n3);      /* 和を表示 */

    return (0);
}
```

実行例

```
三つの整数を入力してください。
整数 1 ：7↵
整数 2 ：15↵
整数 3 ：23↵
それらの和は45です。
```

連続した演算

このプログラムのように、二つ以上の演算を連続して行うことができます。

錬成問題

■ 人間が文字の並びとして作成する（1）プログラムを、コンピュータが理解できる0と1の並び、すなわち（2）の並びへと変換するには、一般に、（3）などの作業が必要である。

■ 作成者を含め、その読み手に伝えたいことを、簡潔な言葉でプログラムに書き込まれたものは、（4）と呼ばれ、（5）と（6）で囲んで記述する。なお、（4）の有無や、その内容によって、プログラムの動作が変わることは（7）。

▶ （7）の選択肢 … (a)ある (b)ない

■ C言語には、多くの（8）が用意されており、それらをうまく利用することによって、手短にプログラムを作ることができるようになっている。（8）に処理を行ってもらうための依頼ともいえる（8）呼出しにおいて、必要な補助的な指示は、（9）として与える。

■ 表示を行うための（8）として、***printf*** や ***puts*** などがある。

ここで、***printf*** に渡す最初の（9）は、（10）と呼ばれ、その中には、続く（9）の出力書式を指定するための（11）を含むことができる。なお、整数値を10進数で出力するための（11）は、%（12）である。

また、***puts*** による表示では、出力の最後に自動的に改行が（13）。

▶ （13）の選択肢 … (a)行われる (b)行われない

■ **\n** は（14）を表す拡張表記であり、**\a** は（15）を表す拡張表記である。

■ **"ABC"**や**"Hello!!"**のように、文字の並びを二重引用符で囲んだものを（16）と呼ぶ。

■ 整数を格納する *ax* という名前の変数を使うためには、

（17） *ax*;

といった宣言が必要である。

■ 加算（足し算）を行うための記号は **+** であり、減算（引き算）を行うための記号は **-** であるが、乗算（掛け算）を行うための記号は（18）である。

（次ページへ続く）

■　以下に示すのは、右に示すように、「C言語」と表示するプログラムである。

C言語

```
#include  <stdio.h>

int main(void)
{
    printf("    (19)    ");

    return (0);
}
```

■　以下に示すのは、いずれも、右に示すように、「C言語」と縦に表示する（1行に1文字ずつ表示する）プログラムである。

C
言
語

```
#include  <stdio.h>

int main(void)
{
    printf("C  (20)  ");
    printf("言  (20)  ");
    printf("語  (20)  ");
    return (0);
}
```

```
#include  <stdio.h>

int main(void)
{
    printf("    (21)    ");

    return (0);
}
```

■　右に示すのは、警報を2回発するプログラムである。

```
#include  <stdio.h>

int main(void)
{
    printf("    (22)    ");

    return (0);
}
```

■　以下に示すのは、いずれも、右に示すように、「こんにちは。」と「はじめまして。」を3行にわたって（真ん中の行は空白とする）表示するプログラムである。

こんにちは。

はじめまして。

```
#include  <stdio.h>

int main(void)
{
    puts("        (23)        ");

    return (0);
}
```

```
#include  <stdio.h>

int main(void)
{
    puts("        (24)        ");
    puts("        (25)        ");
    puts("        (26)        ");

    return (0);
}
```

■ 以下に示すのは、右に示すように、整数値を読み込んで、その 5 倍の値を表示するプログラムである。

```
整数を入力してください：57↵
その数の5倍は285です。
```

```
#include <stdio.h>

int main(void)
{
    int  (27) ;

    printf("整数を入力してください：");          /* 整数値の入力を促す */
    scanf("  (28)  ",  (29) no);               /* 整数値を読み込む */

    printf("その数の5倍は (30) です。\n", 5  (31)  no);

    return (0);
}
```

■ 以下に示すのは、三つの整数値を読み込んで、それらを掛け合わせた値を表示するプログラムである。

```
#include <stdio.h>

int main(void)
{
    int  (32) ;

    printf("三つの整数を入力してください：");

    printf("一番目：");
    scanf("  (28)  ",  (29) c1);

    printf("二番目：");
    scanf("  (28)  ",  (29) c2);

    printf("三番目：");
    scanf("  (28)  ",  (29) c3);

    printf("それらを掛け合わせた値は (33) です。\n",  (34) );

    return (0);
}
```