第2章 画面への入出力

2.1 画面への出力 printf

■ Hello Worldプログラムとprintf関数

リスト2.1は，C言語で書かれたもっとも簡単なプログラムであり，たいていの参考書の冒頭に必ずと言ってよいほど掲載されているプログラムです．これはディスプレイ画面に"Hello World"という文字列を表示するだけの内容です．

リスト2.1 printf関数で文字列を画面出力

```
#include <stdio.h> ……この場所にファイル<stdio.h>を取り込む

main()  ……プログラムはここから始まる
{

    printf("Hello world¥n"); ……プログラムの本文．printf関数で文字列"Hello world"を画面表示

    return 0;  ……プログラムの終了
}
```

● #記号から始まる行はプリ・プロセッサ（第12章参照）と呼ばれ，"**#include**"は，次の**<>**内に囲まれた名のファイル"**stdio.h**"をこの場所に取り込むことを意味します．直接プログラムに記述されておらず，別の場所から取り込むファイルのことを，インクルードファイルなどといいます．**stdio.h**には入出力を行うプログラムが定義されており，ここでは**printf**というプログラムを使用します．詳しくは，第8章で説明します．

● "**main()**"から始まり，3行目の'**{**'と最後の行の'**}**'によって囲まれる単位を関数と呼び，最初に"**main()**"と書かれる関数が**main**関数です．第7章で解説するように，C言語はいろいろな関数の組み合わせによって書かれますが，この**main**関数はプログラムの中に必ずなければなりません．

● **main**関数内部にプログラムの本文が書かれます．ここで書かれているのは，

```
printf("hello world¥n");
```

と，

```
return 0;
```

の2行だけであり，2行目の"**return 0;**"は，**main**関数の最後に必ず置かなければならない1行です．ただし，現在のところはとくに具体的な作業内容を指示するわけではありません．したがって，1行目の"**printf("Hello world¥n");**"がとりあえず問題となります．

● "**printf("Hello world¥n");**"の1行も，実は関数と呼ばれる単位を呼び出すための文章です．

関数とは，ある特定の作業をコンピュータに行わせるための命令形（コマンド）のようなものであり，C言語では標準関数として数百種類もの関数が用意されています（第8章参照）．関数のスタイルは，最初にまず関数名が書かれ，次に丸括弧**()**が続くという形式になっています．たとえば，"**abcd**"という名の関数があるとすると，"**abcd()**"が関数の記述形式となります．**()**内にはいろいろなデータを挿入することが多いです．

● "**printf("Hello world¥n");**"は**printf**という名の関数を呼び出す文章であり，**printf**関数は画面にテキスト（文字データ）の出力を行う機能を持ちます．**()**内に"**hello world¥n**"が挿入されており，出力する文字列が「hello world」です．この**printf**関数を使用するときは，必ずプログラム冒頭で"**stdio.h**"という名のインクルードファイルを取り込まなければなりません．

●「**¥n**」は，ここで改行を行うことを意味します．'**¥**'から始まる2文字はエスケープ・シーケンスと呼ばれ，それ自体を画面に出力するのではなく，ある機能をコンピュータに実行させるための記号になります．

● 最後の'**;**'（セミコロン）は，1行の終わりをあらわす句読点（.）と同じような意味を持ち，命令の区切り記号として使います．

■ エスケープ・シーケンス

次のプログラムのように，改行（ニューライン）記号"**¥n**"を置かないと，**printf**関数を2行続けても改行されません．

リスト2.2　改行（ニューライン）記号"\n"を置かない

```
#include <stdio.h>
main()
{
    printf("Hello world");  ……ニューライン記号（¥n）を置かない
    printf("How are you¥n"); ……2行の文字列を画面に出力

    return 0;
}
```

実行結果

```
Hello worldHow are you
```

きちんと改行するときは，1文の終わりに改行記号**¥n**を置く必要があります．

リスト2.3 改行（ニューライン）記号"\n"を置く

```
#include <stdio.h>
main()
{
    printf("Hello world¥n");    ……ニューライン記号（¥n）を入れる
    printf("How are you¥n");    ……2行の文字列を画面に出力

    return 0;
}
```

実行結果

```
Hello world
How are you
```

この'¥'から始まるひとつの機能を実行させるための記号をエスケープ・シーケンスといい，次のようなものがあります．

¥n	改行復帰	¥f	改ページ
¥t	水平タブ	¥a	ベルを鳴らす
¥v	垂直タブ	¥0	ヌル
¥b	バックスペース	¥r	キャリッジリターン

<表2.1>エスケープ・シーケンス

■ フリーフォーマットとインデント

C言語のプログラムはフリーフォーマットであるため，次のように1行にまとめて記述することもできます．

リスト2.4 1行にまとめて記述

```
#include <stdio.h>
main(){printf("Hello world¥n");return 0;} ……1行にまとめて書いてもかまわない
```

しかし，これではプログラムが読みにくいため，**main**関数内部のコードは，左から4文字～8文字分だけ次下げ（インデント）して書くスタイルを習慣付けた方がよいでしょう．第2章以降の条件文や繰り返し文を書く際には，ひとつの実行単位を次々とインデントして書く機会が増えてきます．通常は，tabキーを利用してインデントを行います．

■ コメント文

/*からはじまり，***/***で終わる部分は，コンパイル時にコメントとして読み飛ばします．プログラムの中で簡単な註釈を書き込んでおきたいときには，それを**/*～*/**で囲むことによってコメントとすることができます．

リスト2.5　コメント文/*～*/の使用

```
/* this is HelloWorld program */          ……コメント

#include <stdio.h>
main()
{
    /*  Hello worldと画面出力する  */      ……コメント

    printf("Hello world¥n");

    /*  次はreturn文．プログラムの終了を表す  */ ……コメント

    return 0;
}
```

また，これはC言語ではなくC++言語の規約になりますが，行の先頭にスラッシュ'/'を二つ続けて"//"を置くと，その一行だけがコメント文となります．

リスト2.6　コメント文//～の使用

```
// this is HelloWorld program             ……コメント

#include <stdio.h>
main()
{
    //  Hello worldと画面出力する          ……コメント

    printf("Hello world¥n");

    // 次はreturn文．プログラムの終了を表す    ……コメント

    return 0;
}
```

2.2　変数と定数／数値の出力

■ 変数と定数と変換文字

変数とは，あるデータを保存するための記憶場所であり，具体的なデータのことを定数といいます．たとえば，25という数値データ（定数）を記憶するためには，まず適当な名を付けた変数を次のような型宣言文によって用意して，データを代入文によって記憶します．

`int  a;`…型宣言文

`a = 25;`…代入文

最初の"**int**"は，使用する変数の型を宣言するものであり，ここでは4バイトの整数型を意味しています（後述）．次の'**a**'が自分で適当に決める変数名です．

"a=25;"が代入文であり，これによって'a'という名の記憶場所に25という数値データを保存します．

リスト2.7　変数にデータ入力し，画面表示

```
#include <stdio.h>
main()
{
    int a;          ……aという変数の型宣言をする（4バイトの整数型）
                        変数を使用する場合は，必ず最初に変数と型を宣言しなければならない
    a = 25;         ……aという変数に数値25を代入する
    printf("%d¥n", a);  ……変数aに代入されている数値を変換指示子%dを使って表示する

    return 0;
}
```

実行結果

```
25
```

データを記憶した変数の内容を参照するときには，'%'から始まる記号――変換指示子を用います．"%d"は整数値を参照するための変換指示子であり，これをprintf関数の中で，

```
printf("%d¥n", a);
```

として使用します．変換指示子"%d"は，その文字が直接画面に出力されるのではなく，"%d¥n"の次に置かれる','の後の変数aに代入されている数値を代わりに出力する機能を持ちます．

変数に用いる記号は，最大32字の英数字と_（アンダーバー）で表します（ただし，数字を最初の一文字に使用してはいけない）．

また，英字における大文字と小文字の区別を行います．通常，変数名，関数名は小文字を使用します．

リスト2.8　いろいろな変数名を使用

```
#include <stdio.h>

main()
{
    int abcdefg;    ……アルファベット
    int abc123;     ……アルファベットと数字の組み合わせ
    int abc_456;    ……アルファベットと数字とアンダーバー'_'の組み合わせ
    return 0;
}
```

プログラムは原則的に上から下に向かって実行されるので，次のように新しく置かれた代入文が，先に書かれた代入文より優先されます．

リスト2.9　後の代入文が優先される

```
#include <stdio.h>

main()
{
    int  a;
```

```
    a = 5;    ……変数aに5を代入する
    a = 8;    ……変数aに8を代入する
    return 0;
}
```

原則的に変数名は自由に名前を付けることができますが，例外的に次に示す単語は予約語と呼ばれ，使用することはできません．

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

<表2.2>　いろいろな予約語

リスト2.10　予約語は使用不可

```
#include <stdio.h>

main()
{
    int  case;    ……予約語は変数名に使用できない

    case = 5;     ……予約語は変数名に使用できない
    return 0;
}
```

printf関数の中で，変数データ参照を行う変換指示子と任意の文字列とを組み合わせてみます．

リスト2.11　文字列"a = "と変換文字%dとの組み合わせ

```
#include <stdio.h>

main()
{
    int  a;
    a = 25;

    printf("a = %d¥n", a);    ……文字列"a = "と変換文字%dとの組み合わせ

    return 0;
}
```

実行結果

```
a = 25
```

結局，**printf**関数の"～"内に置くデータ（引数＝ひきすう）のうち，'¥'からはじまるエスケープ・シーケンス（**¥n**）と'**%**'からはじまる変換指示子（**%d**）以外は，そこに記述されたとおりに画面に出力されると考えてよいのです．スペース（空白）などもそのまま出力されます．

また，**printf**関数1行で複数の変数を参照するためには，複数の変換指示子と，右のコンマで区

切られた複数の変数名を置くことができます．このとき変換指示子の数と，右の変数の数とは合致していなければなりません．

リスト2.12　1行のprintfで複数の変数を参照する

```
#include <stdio.h>

main()
{
    int  a, b, c;                          ……複数の変数宣言

    a = 10;
    b = 20;
    c = 30;

    printf("a = %d b = %d c = %d¥n", a, b, c);  ……複数の変数を複数の変換文字で出力

    return 0;
}
```

実行結果

```
a = 10 b = 20 c = 30
```

■ いろいろな変数の型

変数の型には，代入するデータの種類によって**表2.3**のようなものが用意されています．ただ，使用しているコンピュータやコンパイラによって，各データ型についてどれだけの大きさのメモリ領域を確保するかどうかは若干異なってきます．かつてのMS-DOS時代には，**int**型変数には2バイト分のメモリサイズを確保するのが通例でしたが，現在の32ビットアプリケーションでは4バイト分のメモリサイズを取得していることが多いと思われます．なお，各データ型のサイズは，**sizeof**演算子を使ってを調べることができます（第11章「C言語の演算子」参照）．

<表2.3>　いろいろな変数の型

型	サイズ	値の範囲	代入される数値／文字の例
char	1バイト	−128〜127	'A'
int	4バイト	−2,147,483,648〜2,147,483,647	−150
short	2バイト	−32,768〜32,767	2450
long	4バイト	−2,147,483,648〜2,147,483,647	1234567890
unsigned char	1バイト	0〜255（0x00〜0xff）	255
unsigned int	4バイト	0〜4,294,967,295（0x00000000〜0xffffffff）	0xffff
unsigned short	2バイト	0〜65,535（0x0000〜0xffff）	65535
unsigned long	4バイト	0〜4,294,967,295（0x00000000〜0xffffffff）	2000000000
float	4バイト	3.4×10^{-38} 〜 3.4×10^{38}	98.7654321
double	8バイト	1.7×10^{-308} 〜 1.7×10^{308}	1.2345e100

変数の中のデータを参照するための変換指示子の種類には，次のようなものがあります．