金沢工業大学2009年度前学期 開講科目

コンピュータ工学

配布資料

光永 法明

平成21年8月26日

目 次

第1章　コンピュータ入門

1.1　コンピュータの構成部品

身近にあるコンピュータの例としてパソコンの構成部品を見てみましょう (図 1.1)。大きく 3 つに分けることが出来ます：

- CPU
- メモリ
- 周辺機器（ハードディスク、ディスプレイ、キーボード、マウス、プリンタほか）

CPU は手順（プログラム）に従って動作する部品です。メモリはメモ用紙・ノートのようなもの、周辺機器はそれ以外と考えてください。これは携帯電話やテレビの中のコンピュータでも変わりはありません。

ポイント：

- CPU は手順（プログラム）にしたがって動作する
- プログラムはコンピュータの動作を決める
- メモリはメモ用紙である

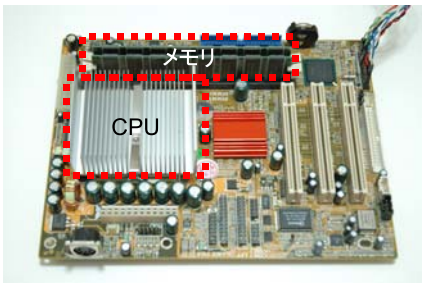

図 1.1: パソコンのマザーボード上には CPU とメモリ、電源や周辺機器のための回路がある。印のないところはほとんどが周辺機器用の回路。

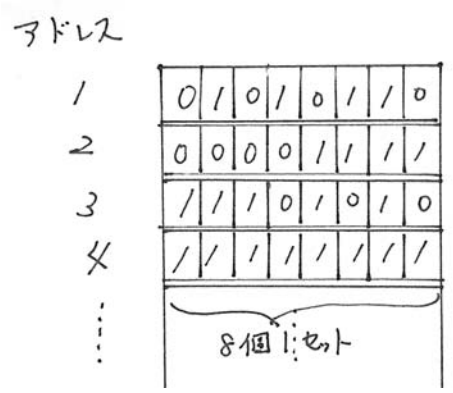

図 1.2: メモリにはアドレスがあり、1 マスに 0/1 が 1 つ記入できる。8 マスが 1 組になっている

1.2　メモリ

メモリは、どのようなメモ用紙でしょうか。図 1.2 を見てください。このように 1 マスに 0 または 1 を記入できる欄が 8 個で 1 セットになって並んでいます。升目には番号がついていてアドレス（address: 番地) と呼ばれます。メモをする/見るときには、本のページのように、アドレスを指定します。

ポイント：

- メモリはアドレスで管理する
- アドレスでメモリ上の位置を指定する

1.3　ビット

情報を表す最小の単位がビットです。1 ビットでは、2 つの状態 0 と 1 を扱います。0 と 1 で何が表現できるか考えて見ましょう (図 1.3)。もちろん、数字の 0 と 1 が表せます。電球の点灯/消灯も表せます。1/-1、○/×、A/B、ご飯がある/ない、たこ/いかと、2 通りの組み合わせのものなら表現できるのです。

ポイント：

- 1 ビットは 2 つの状態を表せる
- 約束を決めれば、1 ビットで、2 つの数字、2 つの記号、2 つの状態、2 つの種類など 2 通りのものが表現できる

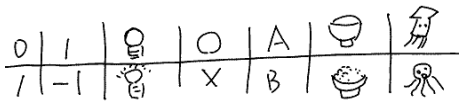

図 1.3: 1 ビットで表せるもの

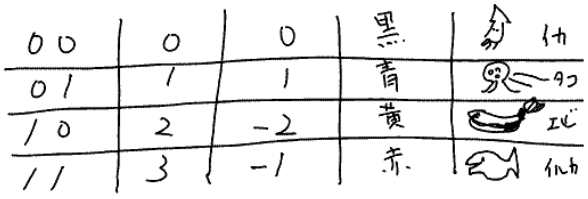

図 1.4: 2 ビットで表す

1.4 2 ビット

では 2 ビットあると何が表現できるでしょうか？2 ビットの数字で表すと、00, 01, 10, 11 の 4 つの組み合わせが出来ます (図 1.4)。

1.5 バイト

1 ビットだけでは不便なので 8 個まとめて、1 バイトと呼ぶ単位でコンピュータ上で扱うことが多くなります。0/1 が 8 個あるので、$2^8 = 256$ 通りの状態が表せます。メモリの 1 マスは 1 ビット、8 個まとめて 1 バイトを単位としてアドレスがついていることになります。

ポイント：

- 8 ビットをまとめて 1 バイトという
- 1 バイトには 8 ビット含まれる
- 1 バイトで 256 通りが表せる
- メモリはバイト単位で扱う
- アドレスが 1 違うと 1 バイト違う

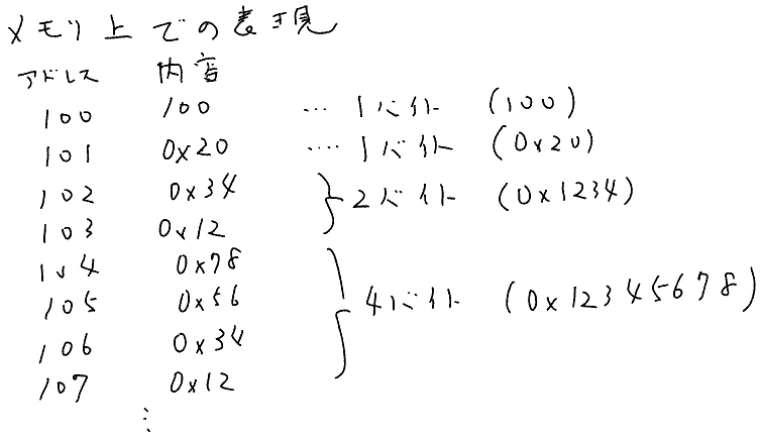

図 1.5: メモリ上での数値の表現

1.6　組み合わせと数字の表現

1 バイトで 256 通りの組み合わせが表現できます。つまり、整数に限るなら 0 から 255 までが表現できます。負の数も表したいときには整数なら-128 から 127 まで表現することも出来ます。

ポイント：

- 1 バイトで正の整数なら 0 から 255 までを表せる
- 1 バイトで正負の整数なら -128 から 127 までを表せる

1.7　256 では足りない

256 通り (-128 から 127) では、困ったことになる場面が多そうです。そこで、コンピュータで数値を表すときには 2 バイト（16 ビット）以上のメモリを使って数値を表すような方法があります。例えば 2 バイトなら、256 × 256 ＝ 65536 通りの組み合わせがあります。また 4 バイト（32 ビット）, 8 バイト（64 ビット）も、よく使われます。メモリ上では、数値を 1 バイト (16 進数の 2 桁) ずつ順に並べて図 1.5 のように表現します[1]。

ポイント：

- 1 バイトで足りないときには 2 バイト以上使う
- 複数バイトの数字はメモリ上で連続して並べる

[1] この図 (リトルエンディアンの CPU) とは逆の順番、0x12, 0x34 や 0x12, 0x34, 0x56, 0x78 と書く CPU(ビッグエンディアンという) もあります。

1.7.1 整数の範囲

バイト数と扱える数字の組み合わせの数は次の通りになります。

- 1 バイト (16 進数 2 桁)：$2^8 = 256$ 通り
- 2 バイト (16 進数 4 桁)：$2^{16} = 65536$ 通り
- 4 バイト (16 進数 8 桁)：$2^{32} = 4294967296$ 通り
- 8 バイト (16 進数 16 桁)：$2^{64} = 18446744073709551616$ 通り

したがって 0 以上の整数を表すときには、各バイト数で以下の範囲を表すことが出来ます。

- 1 バイト (16 進数 2 桁)：0 から 255
- 2 バイト (16 進数 4 桁)：0 から 65535
- 4 バイト (16 進数 8 桁)：0 から 4294967295
- 8 バイト (16 進数 16 桁)：0 から 18446744073709551615

ところで負の数を扱いたいときもあります。そのときには、

- 1 バイト (16 進数 2 桁)：-128 〜 127
- 2 バイト (16 進数 4 桁)：-32768 〜 32767
- 4 バイト (16 進数 8 桁)：-2147483648 〜 2147483647
- 8 バイト (16 進数 16 桁)：-9223372036854775808 〜 9223372036854775807

とします[2]。

ポイント：

- 0 以上の整数を表すときには、扱える数値の範囲は組み合わせの数より 1 少ない
- 負の整数を表すときには、最大の数値が組み合わせの数の約半分になる

[2] 16 進数で 0x80 から 0xff, 0x8000 から 0xffff, 0x80000000 から 0xffffffff, 0x8000000000000000 から 0xffffffffffffffff を負の値とします。また 0xff, 0xffff, 0xffffffff, 0xffffffffffffffff を -1 とします (2 の補数表現と呼ぶ)。

表 1.1: 文字コード表

2進	10進	16進	文字
0010 0000	32	20	
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	−
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

2進	10進	16進	文字
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	¥
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

2進	10進	16進	文字
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

1.8　コンピュータと文字

コンピュータでは直接文字を表せません。そこで、48 は A, 49 は B といったように文字の代わりに番号で表すことにします (表 1.1[3])。コンピュータやプログラムで、同じ文字に同じ番号を割り当てるため、文字コード表というものを使います[4]。

1.9　整数以外も扱いたい

整数と文字では表現できないものがあります。そうです。実数です。コンピュータの中での表現方法として浮動小数点という表現をよく使います。10 進数の指数表現を思い出し

[3]以下を引用：「ASCII」『フリー百科事典　ウィキペディア日本語版』(http://ja.wikipedia.org/)。2009 年 3 月 14 日 19 時（UTC）現在での最新版を取得。

[4]日本語の文字は表にはありませんが、同様に数値で文字を表します。

てください。

$$1234.56 = 1.23456 \times 10^3 \tag{1.1}$$

と表せます。ところで、仮数部の数字 (123456) と、指数部の数字 (3) を規則を決めて並べてみましょう。たとえば、仮数部を 6 桁、指数部を 4 桁で表すとします。そうすると、

$$+12345678| + 0003 \tag{1.2}$$

と表せます。数字は 10 桁 (と正負の記号が 2 つ) で、有効数字 6 桁ながら、$-9.99999 \times 10^{9999}$ から 9.99999×10^{9999} までの大きな範囲の数字が表せることになります。そして $0.00001 \times 10^{-9999} = 1.0 \times 10^{-10004}$ という小さな数字も扱えます。このように同じ桁数で表現できる数字の範囲を広げる方法が使われています。

ところで有効数字の話が出ました。コンピュータの中で、大きな数字と小さな数字を足したり引いたりすると、小さな数字の部分の誤差が出てしまいます。このことに注意してください。また、ここでは 10 進数表記ですが、コンピュータ内では 2 進数表記のため変換に誤差が出ます。詳細は省きますが、このことも頭の片隅に覚えておいてください。

ポイント：

- コンピュータの中では「浮動小数点」という方法で実数を表す
- 大きな数字と小さな数字を、足し引きするときには誤差に注意する
- 10 進数表現と 2 進表現での誤差にも注意する

1.10 変数とアドレス

ここで喫茶店の会計をすることを考えて見ましょう。メニューにはコーヒーとジュースだけなのですが、値段はときどき変わります。注文の数も変わります。このとき会計は、

（コーヒーの値段）×（コーヒーの数）+（ジュースの値段）×（ジュースの数）

で計算できます。

数式で表すなら、コーヒーの値段を x、ジュースの値段を y、コーヒーの注文数を a、ジュースの注文数を b とします。ただし、x, y, a, b は整数とします。このとき

$$ax + by \tag{1.3}$$

と変数で会計の計算式を書くことが出来ます。値段や注文数が変わっても、計算式はそのままです。

さて、喫茶店用のコンピュータで、コーヒーの値段をアドレス 10 に、ジュースの値段をアドレス 11 に書くと決めましょう。注文されたコーヒーとジュースの数は、アドレス 12 と 13 に書きます。そうすると、

（アドレス 10 の値）×（アドレス 12 の値）+（アドレス 11 の値）×（アドレス 13 の値）

という計算をすれば、飲み物の値段や注文された個数に応じて同じ手順で会計が出来ます。でも、これでは、どういう計算をしているかよく分かりません。

そこで、アドレスに分かりやすい名前をつけましょう。そして数学と同じように変数と呼びましょう。例えば、

- 変数の名前：$coffee$, 変数の型:1 バイトの整数, 変数のアドレス：10
- 変数の名前：$coffee\_num$, 変数の型:1 バイトの整数, 変数のアドレス：12
- 変数の名前：$juice$, 変数の型:1 バイトの整数, 変数のアドレス：11
- 変数の名前：$juice\_num$, 変数の型:1 バイトの整数, 変数のアドレス：13

とします。そうすると、

$$(coffee) \times (cofee\_num) + (juice) \times (juice\_num) \tag{1.4}$$

と計算式が書けます。このとき、アドレス 10 の値が 200 であれば、コーヒーの値段は 200 円として計算するのです。

ポイント：

- プログラムには変数を使う
- 変数の種類のことを型という
- 変数には名前とアドレスと型、値がある
- 変数を使うと値が変わっても計算式（プログラム）はそのままでいい
- プログラムで扱う変数の名前は長くてもいい

1.11　まとめ

- コンピュータは CPU、メモリ、周辺機器から構成される
- コンピュータの動作はプログラムで決まる
- メモリはアドレスで、バイトを単位に管理される
- 1 バイトは 8 ビット
- 変数には、名前とアドレスと型、値がある
- プログラムでは変数の使い方が重要になる

第2章　コンパイルしてみよう

2.1　プログラミング言語

コンピュータで動作するプログラムは機械語と呼ばれる表現になっています。この機械語は分かりにくいので、もう少し人に分かりやすい表現の形式でプログラムを書くことが一般的です。その表現形式のことを「プログラミング言語」と呼びます。C 言語はその一種です。

2.2　コンパイラ: C 言語から機械語へ

C 言語から機械語へ変換することを「コンパイル」といいます。「コンパイル」するツール（プログラム）を「コンパイラ」とよびます。この授業では「gcc」というコンパイラを使います。「コンパイル」する前の C 言語で書いたプログラムのことは「ソースプログラム」、ファイルのことを「ソースファイル」と呼びます。

2.3　Code::Blocks

Code::Blocks は

- ソースファイルの編集ができたり、
- コンパイラをボタン 1 つで呼び出したり、
- コンパイルして出来たプログラムを実行したり、
- プログラムを 1 行ずつ確かめながら動かすことが出来たり、

ツールです。このようなツールのことを「統合開発環境」と呼びます。Visual C++ なども、その 1 つです。Code::Blocks と一緒に gcc もインストールできます。

2.4　ソースファイルについての注意

ソースファイルは「テキストファイル形式」で保存します。Windows のメモ帳で扱える形式です。Code::Blocks で編集するときには気にする必要はありませんが、MS-WORD で開いたときなどには注意してください。

2.5　Code::Blocks をインストールしたら使ってみよう

2.5.1　プロジェクトの新規作成

インストールした Code::Blocks を起動すると図 2.1 の画面[1]になります。ここで簡単なプログラムを作ってみましょう。

1. Create a new porject をクリックします。
2. 出てきたウィンドウ (図 2.2) で、Console application を Go をクリックします。
3. 図 2.3 で Next をクリックします。
4. 開発する言語は「C」を選び、Next をクリックします (図 2.4)。
5. 図 2.5 で Project Title に「test」(何でも構いません) を入れ、Folder to create project in にソースファイルなどを入れるディレクトリを指定します。ここでは... をクリックして、c:
   src
   test を作りました。そして Next をクリックします。
6. 図 2.6 で Finish をクリックします。
7. プロジェクトが作成されるので、左の test の中の Sources から main.c をダブルクリックします (図 2.7)。

Code::Blcoks では、プログラムのコンパイル方法などを「プロジェクト」と呼ぶ単位で管理していて、上記の操作では、新しいプロジェクトとソースファイル main.c の雛形を作っています。main.c は list2-1.c のようになっているはずです。

Code::Blocks では、ほかにワークスペースを保存することが出来ます。ワークスペースは、複数のプロジェクトを同時に開いたり、ウィンドウの配置などをまとめたものです。保存しておくことで、前の作業を継続しやすくなります。

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

リスト 2-1 (list2-1.c)

[1]以降の画面は、メッセージを日本語にしていない Code::Blocks の表示です。また実行結果は例であり多少違っても問題がない場合があります

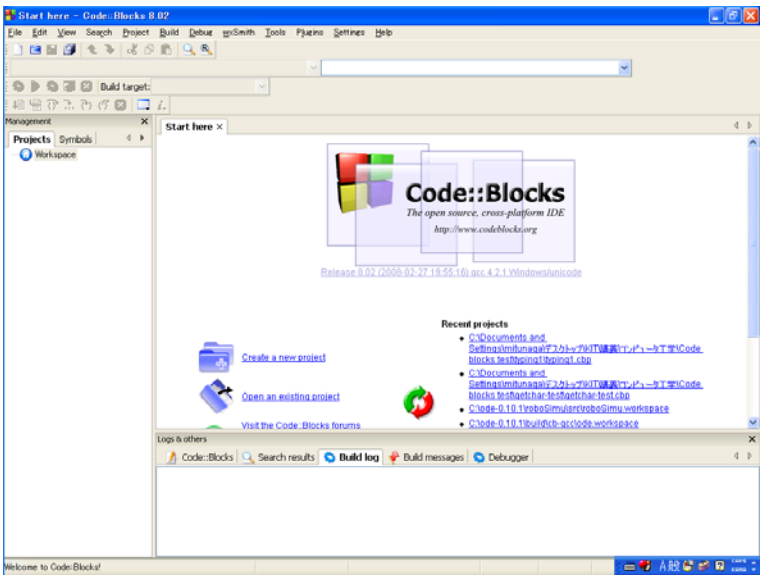

図 2.1: Code::Blocks の起動画面

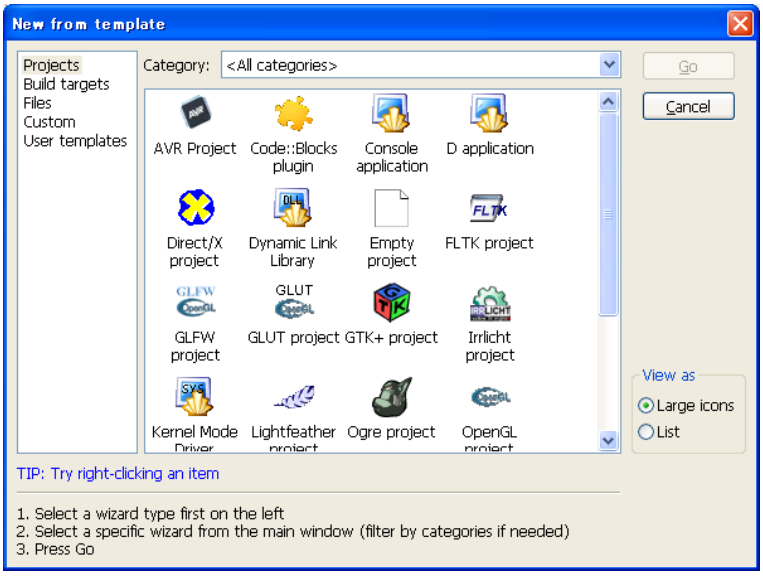

図 2.2: 作成するプロジェクトの種類を選ぶ

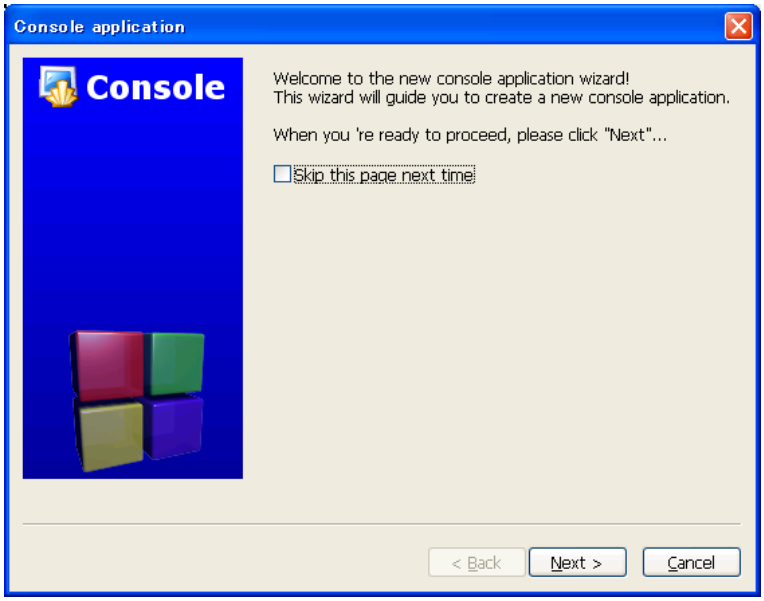

図 2.3: プロジェクト作成ウィザードの最初のウィンドウ

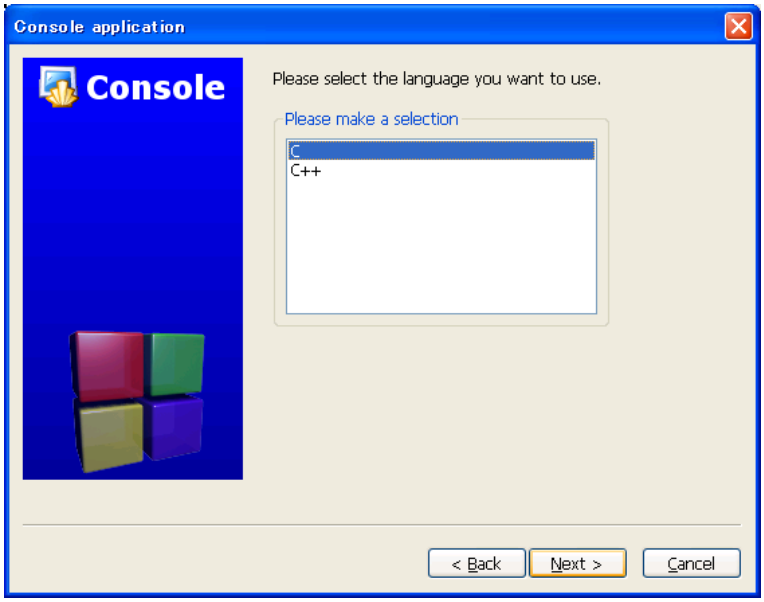

図 2.4: 使用する言語 (ここでは C または C++) を選ぶ

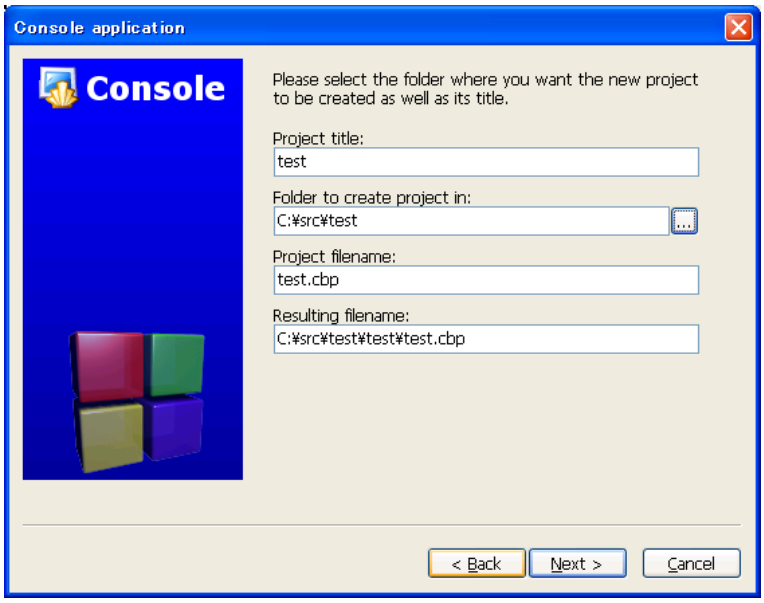

図 2.5: プロジェクトの名前とディレクトリを指定する

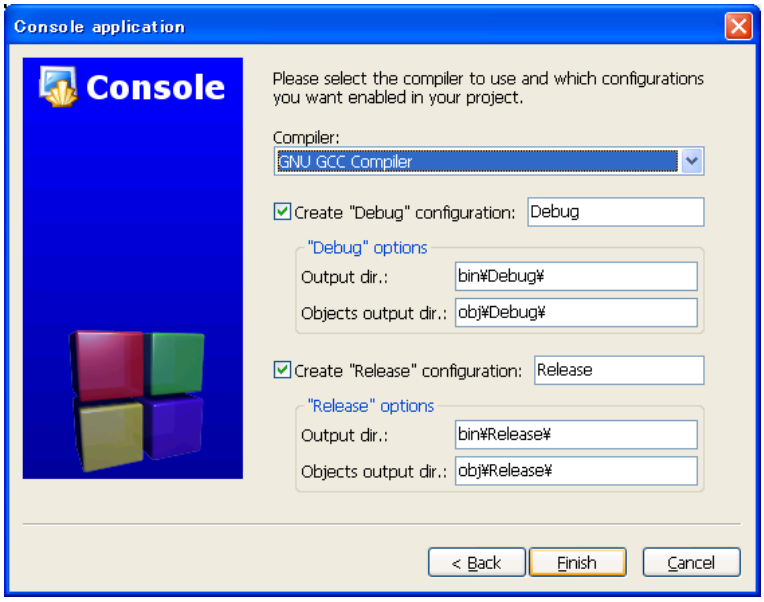

図 2.6: コンパイラ等を指定する。あらかじめ設定されている値でＯＫ。

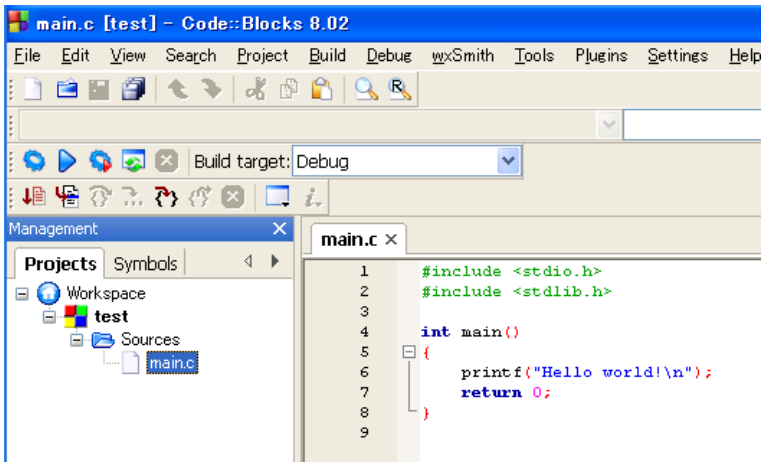

図 2.7: プロジェクトが作成されプログラムの雛形が用意されたところ。

2.5.2　プログラムを実行してみる

それではプログラムを実行してみましょう。左上にボタンが並んでいます (図 2.8)。上の段が実行ボタン、下の段がデバッグボタンです。Build はコンパイル、Run は実行です。Build&Run はコンパイルしたら、すぐに実行します。Rebuild は再コンパイル (Code::Blocks が不要と判断した場合にもコンパイルする、何か不具合があるときに使う)、Abort は、実行中のプログラムを停止します。それでは Build&Run ボタンをクリックしてみましょう。図 2.9 が表示されれば OK です。「Hello world」がプログラムの出したメッセージ、Process ～Press any key to continue. までが Code::Blocks の出すメッセージです。このウィンドウで何かキーを押すとウィンドウは消えます。

Code::Blocks のウィンドウに戻って、下のほうに注目してください (図 2.10)。ここにコンパイルの結果などが表示されます。プログラムのデバッグ (不具合を直すこと) には重要なポイントになるので確認する習慣をつけてください。

Windows のエクスプローラで c:\src\test [2]の下を見ると main.c と bin, obj というディレクトリが作成されています。main.c が先ほど作られたソースファイルの雛形になります。bin\debug のディレクトリには、test.exe というファイルが出来ています。このファイルをダブルクリックすると、さきほどのプログラムが実行できます[3]。

2.5.3　デバッガ機能を使ってみる

図 2.8 の下の段に並んでいるボタンはデバッガ機能のボタンです。ちょっと使ってみましょう。ソースファイルをリスト 2-2 のように変更してください。そしてカーソルを 1 つ

[2] 半角の \ と半角の ¥ マークは同じ意味です
[3] このプログラムの場合、一瞬で終わってしまうので、よくわからないかもしれません。

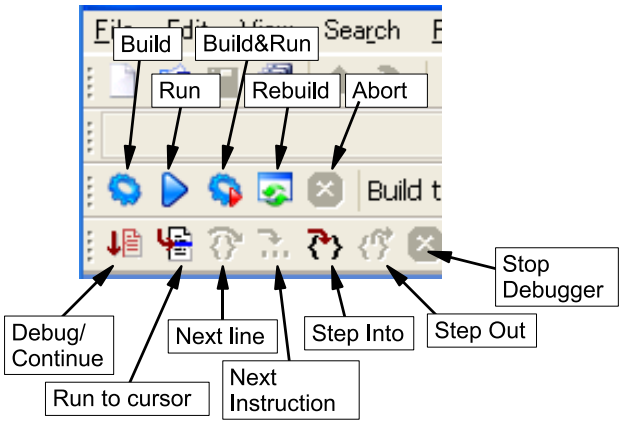

図 2.8: 実行とデバッグのボタン。

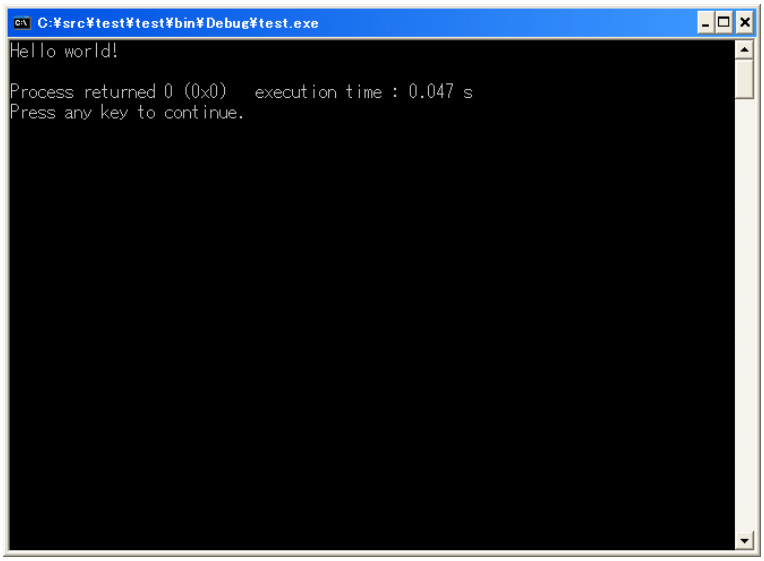

図 2.9: プログラムを実行して終了したところ。

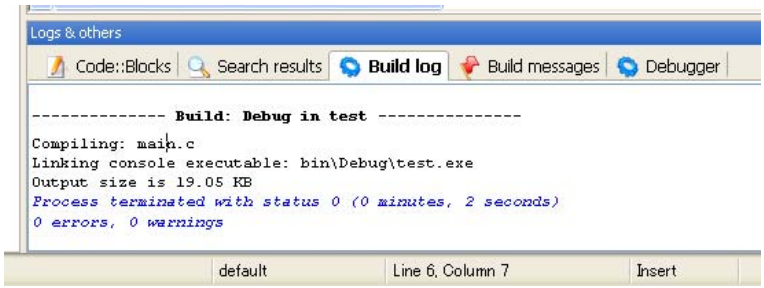

図 2.10: コンパイル結果のメッセージ。

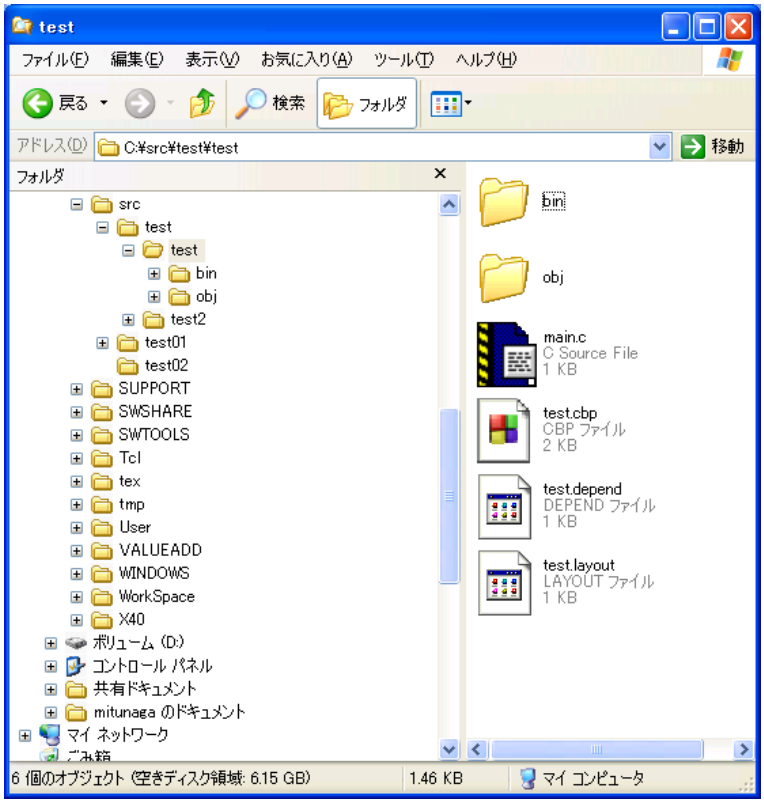

図 2.11: プロジェクトが作られた様子をエクスプローラで見る。

目の printf() の行にしてから、「Run to cursor」のボタンを押してください。「Next line」ボタンを押すと 1 行ずつ順に実行されていく様子が分かります。ほかのボタンも、いろいろ押して動作を試してみてください。

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world(1)!\n");
    printf("Hello world(2)!\n");
    printf("Hello ");
    printf("world(3)!\n");
    printf("Hello world(4)!\n");
    printf("Hello world(5)!\n");
    return 0;
}
```

リスト 2-2 (list2-2.c)

2.5.4 ダウンロードしたファイルを動かす

list2-3.c などダウンロードしたファイルを Code::Blocks で扱うには 2 つの方法があります。1 つは、ファイルを適当なディレクトリにおいておき、Code::Blocks でプロジェクトを開いていない状態で、Code::Blocks にドラッグ&ドロップする方法です。簡単に実行できますが、デバッグ機能は使えません。

もう 1 つはプロジェクトを用意する方法です。

1. まず、第 2.5.1 章のように、新規プロジェクトを作ります。
2. その新規プロジェクトのディレクトリ (図 2.11) に、ダウンロードしたファイルをコピーします。
3. 自動で作成された main.c を右クリックし、「Remove file project」をクリックします (図 2.12)。
4. プロジェクトの名前を右クリックし、「Add files」をクリックし、ダウンロードしたファイルを指定します (図 2.13)。
5. ダイアログが開くので Debug にチェックを入れ、OK をクリックします (図 2.14)。

3, 4 は右クリックでなく、上の「Project」のところをクリックしても構いません。

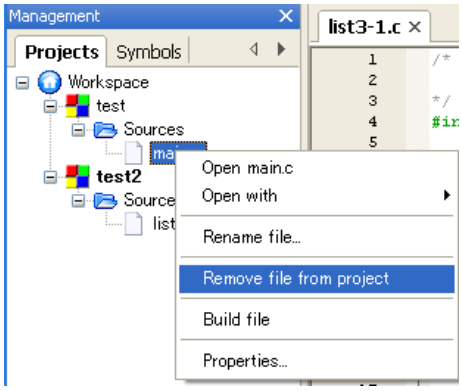

図 2.12: main.c をプロジェクトから外す。

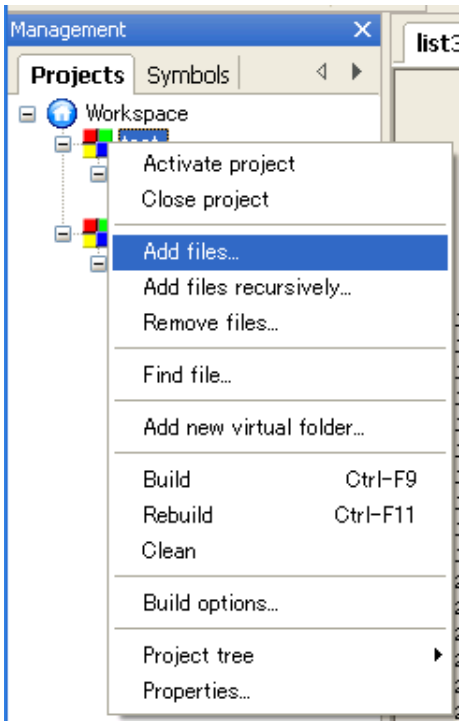

図 2.13: ダウンロードしたファイルをプロジェクトに加える。

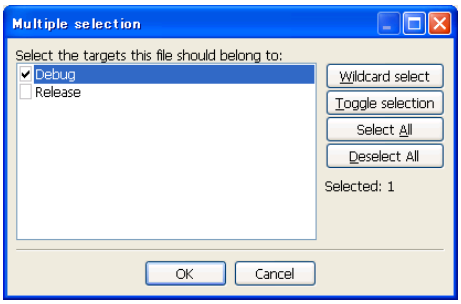

図 2.14: ダイアログが開くので Debug にチェックを入れる。

2.6 演習問題

問題 1 Code::Blocks をインストールし、新規プロジェクトを作成しなさい。

問題 2 Code::Blocks のデバッグ機能を確かめなさい (第 2.5.3 章)。

問題 3 ダウンロードしたファイル list2-3.c を実行してみなさい。プロジェクトも作成してみること。

第3章　C言語早分かり(基礎編)

本章ではC言語のプログラムを理解しする上で必要になる知識をまとめて紹介します。最初から、すべてを覚えてしまう必要はありません。プログラムを書きながら覚えていきましょう。

3.1　式と文

C言語では1つの文の区切りは「;」(セミコロン) です。文には、計算式や関数の呼び出しを書くことが出来ます。

文の例)

```
a = 3;
a = b + c;
y = f(x);
z = f(x, y);
f(x);
```

「=」は「代入演算子」と呼ばれ、左辺の変数に右辺の計算結果を代入します。計算式には表 3.1 の記号が使えます。数式と同じように () も使えますが、「=」は方程式の等号ではなく、代入の意味に使います。

表 3.1: C言語で使える算術演算子

記号	計算
+	＋
-	－
	×
/	÷
%	a % b と書くと a ÷ b の余りが計算される
=	a ＝ b と書くと a に b の値が代入される

3.2　関数

数学の関数は、円上の点 (x, y) について半径を求める関数は、

$$f(x, y) = \sqrt{x^2 + y^2} \tag{3.1}$$

と書けます。ここで変数 x, y の具体的な値が決まると、関数 $f(x, y)$ の値を求められます。

C 言語では、コーヒーの数 coffee_num とジュースの数 juice_num が決まったときに会計をする関数 price() を作ってみます。

```
int
price(int coffee_num, int juice_num)
{
  int sum;

  sum = coffee_num*200 + juice_num*300;

  return sum;
}
```

なんとなく、何をしているか分かりますね。コーヒーの数× 200 とジュースの数× 300 の合計を sum 変数に代入して、結果を返します。ここで関数を呼び出すときに渡す値のことを「引数」（ひきすう）と呼びます。結果を返すには「return」を使います。会計を求めるには、例えば

```
  a = price(2, 3);
```

とします。そうすると、coffee_num に 2、juice_num に 3 が代入されて、関数 price() が「呼び出され」、計算結果 1300 が「返され」ます。返された 1300 のことを「戻り値」（もどりち）または「返り値」（かえりち）と呼びます。戻り値 1300 は変数 a に代入されます。

ここで、

```
  a = price(2, 3);
  price(4, 3);
```

と書いたとしても、a の値は 1300 のままで、1700 にはならないことに注意してください。C 言語の式は方程式ではないからです。また、関数の戻り値を変数に代入しなくても構いません。

また、

1. 引数は関数内で変数として扱えること
2. 変数の値が変わっても、呼び出した側の変数に影響はない

ということを覚えておいてください。1 から同じ関数を、次のように書くことが出来ます。

```
int
price(int coffee, int juice)
{
  coffee = coffee * 200;
  juice = juice * 300

  return coffee + juice;
}
```

このとき呼び出した側の数値 (上の例では定数 2, 3, 4, 一般では変数) が変わることはありません。

3.3 プログラムの流れ：main 関数から始まる

C 言語ではプログラムは main() 関数から始まり[1]、上から順に式を実行していきます[2]。次のプログラムでは、先頭に書いた (1) から (9) の順で式が実行されます。実行に関係がない行も、ありますね。それらについては、以下で説明していきます。

```
      /*
       喫茶の会計を求めるプログラム
       */
      #include <stdio.h>

      int price(int coffee_num, int juice_num);

      int
      main(int argc, char **argv)
      {
        int coffee, juice, sum;

(1)     printf("コーヒーの個数は？");
(2)     scanf("%d", &coffee);
(3)     printf("コーヒーの個数は？");
(4)     scanf("%d", &juice);
```

[1]組み込み向けなどでは main() 関数から始まらないことがあります。

[2]最適化といって、プログラムの動作を速くするようコンパイラに指示をすると順序が入れ替わる場合があります

```
(5)(8)  sum = price(coffee, juice);
(9)     printf("お会計は%d円です。\n", sum);

(10)     reutrn 0;
      }

      /*
        price() 関数は、喫茶の会計を返す
        coffee_num はコーヒーの数, juice_num はジュースの数
                    (それぞれ単位は注文数)
        戻り値は会計の金額で単位は円
       */
      int
      price(int coffee_num, int juice_num)
      {
        int sum;

        // コーヒーの単価は200円, ジュースの単価は300円
(6)     sum = coffee_num*200 + juice_num*300;    // 計算式

(7)     return sum;
      }
```

3.4　空白文字

C言語では、空白文字（スペース、タブ）と改行を書いて、プログラムを見やすくすることができます。たとえば、

```
#include <stdio.h>
main(){printf("Hello world!\n");}
```

と

```
#include <stdio.h>

main()
{
  printf("Hello world!\n");
}
```

の 2 つは同じ動作をしますが、適度な空白のある後者の方が見やすいでしょう。ただし、全角のスペースは使えません。

また文が長くなったら、2 行以上に分けても構いません。たとえば、

```
sum = var_a + var_b + var_c + var_d
    + var_e + var_f + var_g + var_h
    + var_i + var_j;
```

と書くことができます。

3.5 インデント（字下げ）

慣習的に、{}(中括弧) で囲まれた範囲を字下げ（インデント）します。たとえば、

```
int
foo(int bar)          // ここはインデントなし
{
     int boo;         // インデント 1 段

     boo = bar + 1;   // インデント 1 段
     if (boo > 10) { // インデント 1 段
          boo = 10;  // インデント 2 段
     }               // インデント 1 段

     return boo;      // インデント 1 段
}
```

とします。エディタによっては自動で字下げ（オートインデント）の機能があるので利用すると便利です。1 つの { で何文字、字下げするかは自由に決めて構いませんが、オートインデント機能に任せるのも 1 つの方法です。複数の人で共同でプログラムを書くときには、あらかじめ統一しましょう。

3.6 コメント

空白文字以外に、プログラムの中に説明を書きたいときがあります。この説明（メモ）のことを「コメント」と呼びます。コメントの書き方は 2 通りです。1 つの書き方は「/*」と「*/」でコメントを囲みます。もう 1 つの書き方は「//」でコメントの開始を表します。「//」の場合は行末までがコメントになります。たとえば、この関数は何かを示しておくことを考えましょう。

表 3.2: 数値の書き方

	書き方	例
10 進数 (整数)	そのまま書く	123
10 進数 (実数, 浮動小数点)	そのまま書く	123.456
16 進数 (整数)	あたまに 0x をつける	0x123
8 進数 (整数)	あたまに 0 をつける	0123

```
/*
  price() 関数は、喫茶の会計を返す
  coffee_num はコーヒーの数, juice_num はジュースの数 (それぞれ単位は注文数)
  戻り値は会計の金額で単位は円
 */
int
price(int coffee_num, int juice_num)
{
  int sum;

  // コーヒーの単価は 200 円, ジュースの単価は 300 円
  sum = coffee_num*200 + juice_num*300;    // 計算式

  return sum;
}
```

「/*」「*/」は入れ子に出来ないことに注意してください。たとえば、

```
/* /* コメント */ */
```

とすると、2 つ目の「/*」はコメントとして無視されます。そのため、2 つ目の「*/」は対応がとれなくなってしまうのです。

3.7　定数：数値の書き方

プログラムにあらかじめ書いておく数値や文字などのことを「定数」といいます。数値の書き方は表 3.7 の通りです。8 進数を使うことは少ないかもしれませんが、10 進数のとき、うっかり 0 をつけないようにします。また小数点があるかないかで意味が変わります。1 は整数、1.0 は浮動小数点です。

3.8 定数：文字を表す数値

C 言語では文字も数値で表します。「2」という文字を表すとき、10 進数で 50、16 進数で 0x32 と書くことが出来ます。もっと簡便な方法として'2' と書くことが出来ます。「'」はシングルクオーテーションです。「"」と間違えないようにします。全角文字は表せません。

3.9 特殊な文字を表す記号

「＼」(バックスラッシュ)「¥」(半角の¥マーク) で、特殊な記号を表します。よく使うのは、改行「＼ n(¥n)」、復帰「＼ r(¥r)」、タブ「＼ t(¥t)」です。

3.10 文字列とメモリ

単語や文、文章を扱う場合のために、文字を並べた「文字列」を扱います。メモリ上では図 x のように隣り合う文字を順に並べて表現し、先頭の文字のアドレス（ポインタ）で文字列を代表させます。最後に指定した文字以外に 0 が加わえます。したがって、メモリ上では 5 文字の文字列は 6 バイトになります。

定数の文字列をソースコード上で書くのは簡単です。文字列を「"」（ダブルクオーテーション）で囲みます。たとえば

```
"characters"
"文字列"
"1 行目\n2 行目\n3 行目"
"1 カラム\t2 カラム\n10\t20\n1\t3\n"
```

とします。「\n」が改行を表すので、3 つ目の例は、

```
1 行目
2 行目
3 行目
```

を表す文字列です。「\t」がタブを表すので、4 つ目の例は、

```
1 カラム    2 カラム
10          20
1           3
```

を表します。ただしタブの文字数は環境によって変わるので見た目が違うことがあります。また「"」でくくった文字列は「char 型のポインタ」になることを覚えておいてください。

図 x

表 3.3: C 言語の基本型

型の名前	必要なメモリ	表せる数字の範囲
signed char	1 バイト	-128 〜 127
char	1 バイト	0 〜 255 (または -128 〜 127 コンパイラによる)
unsigned char	1 バイト	0 〜 255
short	2 バイト	-32768 〜 32767
unsigned short	2 バイト	0 〜 65535
long	4 バイト	-2147483648 〜 2147483647
unsigned long	4 バイト	0 〜 4294967295
int	4 バイト	-2147483648 〜 2147483647
unsigned int	4 バイト	0 〜 4294967295
long long	8 バイト	-9223372036854775808 〜 9223372036854775807
unsigned long long	4 バイト	0 〜 18446744073709551615
float	4 バイト	浮動小数点
double	8 バイト	浮動小数点

3.11　基本型

変数の種類のことを「型」と呼びます。C 言語には表 3.3 に示す「基本型」があります。基本型のほかに「配列」「ポインタ」と「構造体」「共有体」があります。扱う値が整数の場合には int [3]をよく使います。浮動小数点では double を使うといいでしょう。char は文字と、組み込みプログラミングでメモリを節約したい場合に、よく使います。

負の整数を扱う signed char では、16 進数で 0x80 から 0xff, short では 0x8000 から 0xffff, 0x80000000 から 0xffffffff, long (int) では 0x8000000000000000 から 0xffffffffffffffff を負の値とします。また 0xff, 0xffff, 0xffffffff, 0xffffffffffffffff を -1 とします (2 の補数表現と呼ぶ)。

3.12　変数は宣言してから使う

C 言語では、メモリ上のアドレスと変数の対応関係 (1.10 章) を人が指定する必要はありません。コンパイラが対応関係をつくってくれます。そのかわり、コンパイラにプログラムで扱う変数名と型の対応関係を知らせる必要があります。それが「宣言」です。宣言の方法は簡単です：

[3]32 ビット OS の場合, マイコンでは signed char または short、16 ビット OS では short , 64 ビット OS では long long と同じであることが多い

＜型の名前＞　＜変数の名前＞;

とします。それと宣言は「ブロック」の先頭、つまり「{」の次に書くというルールを守ってください。同じ型であれば複数の宣言を 1 行にまとめたり、代入を指示する（「初期化」と呼びます）ことも出来ます。例を次に示します。

```
int
functionA(int foo)
{
  int x, y, z;
  double a, b, c;
  int i = 0;
  int k = 0, j = 0;

  ＜ほかの処理＞
}
```

3.13 関数のプロトタイプ宣言

関数も宣言してから使います。関数は「プロトタイプ宣言」を書きましょう。プロトタイプ宣言は、

<戻り値の型> <関数名>(<引数 1 の型> <引数 1 の名前>, <引数 2 の型> <引数 2 の名前>, ...);

です。引数の名前は省略して構いません。最後に「;」を忘れずにつけます。たとえば

```
int price(int, int);
int price(int coffee_num, int juice_num);
```

と書きます。プロトタイプ宣言のあとであれば、関数の内容はソースコードのどこに書いても構いません。

3.14 変数名や関数名

数学の変数名と違って一文字という制限はありません。変数名や関数名には英数字 (A-Z, a-z, 0-9, _) を使って分かりやすい名前をつけます。大文字と小文字は区別します。先頭の文字は英文字か_です。ただし_は特別な意味で使われることがあるので、避けたほうがいいでしょう。数字で始めてはいけません。-(マイナス) や+(プラス) といった記号も使えません。

3.15　予約語:変数名や関数名に使えない名前

C 言語で特別な意味があり、変数名や関数目に使えない英文字の組み合わせを「予約語」と呼びます。予約語は、以下の通りです[4]。C 言語でなんらかの役割を持っている語なので、興味があれば調べてみるとよいでしょう。

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

3.16　printf() 関数 (簡単な使い方)

printf() 関数はメッセージ (文字) を表示するのに使います。使い方は、

```
printf("書式文字列"[, <変数 1> [, <変数 2>[,... ]]]);
```

です。書式文字列には、表示したいメッセージを書きます。たとえば、

```
printf("Hello world\n");
printf("こ\n ん\n に\n ち\n は\n");
```

とします。上の例は「Hello world」と表示され、下の例では「こんにちは」が縦書きのように表示されます。\n, \t は、改行とタブを表します (第 3.10 章)。

書式文字列に表 3.4 に示す%で始まる記号を書くと、順に後ろに並べた変数の値 (または定数の値) と置き換えて表示がされます。% を表示したい場合には%%とします。list3-3.c を実行すると、図 3.1 となります。%で始まる記号 (%%除く) の個数と、変数の個数は一致させる必要があります。

```
#include <stdio.h>

int
main(int argc, char **argv)
{
  int i = 123;
  double f = 123.456;
  double g = 123.0;
  char c = 'A';
  char s[] = "test";
```

[4] 処理系 (コンパイラ) によって一部異なります

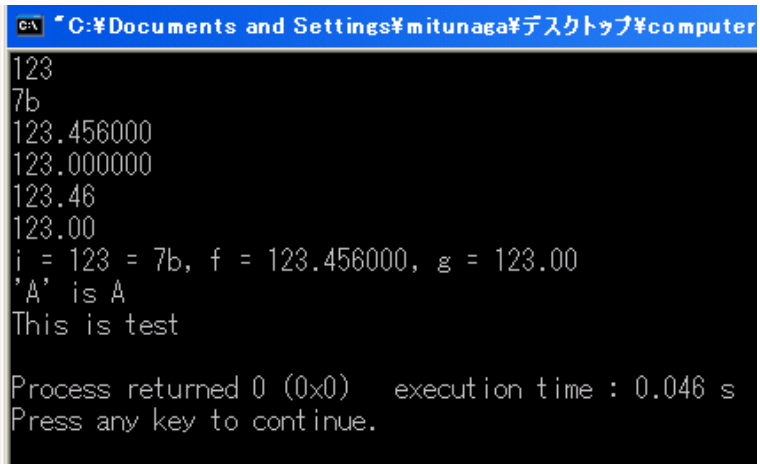

図 3.1: list3-3.c を実行した結果。

```
  printf("%d\n", i);
  printf("%x\n", i);

  printf("%f\n", f);
  printf("%f\n", g);
  printf("%.2f\n", f);
  printf("%.2f\n", g);

  printf("i = %d = %x, f = %f, g = %.2f\n", i, i, f, g);

  printf("'A' is %c\n", c);
  printf("This is %s\n", s);

  return 0;
}
```

リスト 3-3 printf() の使用例 (list3-3.c)

3.17 scanf() 関数 (簡単な使い方)

scanf() 関数は、数字の入力などに使います。使い方は、

```
scanf("書式文字列"[, &<変数 1> [, &<変数 2>[,... ]]]);
```

表 3.4: printf() の書式文字列 (抜粋)

書式文字列に書く記号	置き換える変数の型	使用例	表示例
%d	整数 (10 進数表示)	int i = 123; printf("%d\n", i);	 123
%x	整数 (16 進数表示)	int i = 123; printf("%x\n", i);	 7b

%f	浮動小数点	double f = 123.456;
		double g = 123.0;
		printf("%f\n", f);
		printf("%f\n", g);
			
			123.456000
			123.000000

%.< 桁 >f	浮動小数点	double f = 123.456;
		double g = 123.0;
		printf("%.2f\n", f);
		printf("%.2f\n", g);
			
			123.46
			123.00

%c	文字	char c = 'A'; printf("'A' is %c\n", c);	 'A' is A
%s	文字列	char s[] = "test"; printf("This is %s\n", s);	 This is test
%%	なし	printf("100%%\n");	100%

表 3.5: scanf() の書式文字列 (抜粋)

書式文字列に書く記号	置き換える変数の型	使用例	
%d	整数	int i; scanf("%d",	i);
%f	浮動小数点 (float)	float f; scanf("%f",	f);
%lf	浮動小数点 (double)	double f; scanf("%lf",	f);

です。書式文字列には、表 3.5 の記号を書きます。たとえば、

```
scanf("%d", &i);          // int 型の整数
scanf("%f", &f);          // float 型の浮動小数点
scanf("%lf", &g);         // double 型の浮動小数点
```

とします。%で始まる記号の表す型、個数と、変数の型、個数は一致させる必要があります。変数名の前に&をつけるのを忘れないでください。

3.18 scanf() 関数とアドレス (ポインタ)

scanf() 関数では、変数名の前に&をつけます。実は、この&をつけることで、変数のアドレスを scanf() 関数には渡しています。関数の引数は scanf() 関数で変えることが出来ないのですが、アドレスが分かっていれば第 1.10 章の例のように変数の値を変えることが出来ます。このような関数間の値の渡し方をポインタ渡しと呼びます (第??章)。

3.19 演習問題

3.19.1 プログラムを改造する

問題 1

list2-1.c の出力メッセージ「Hello world!\n」を書き換えなさい。以下をすること：

- 「\n」を 3 回使ってメッセージを書く
- 「\n」と「\t」をそれぞれ 2 回以上使ってメッセージを書く

メッセージ例 1)
```
Hello
world
!
```

メッセージ例 2)
```
こんにちは
Hello
まいど
```

メッセージ例 3)

名前	所属	学年
鈴木	ロボティクス	3 年
田中	ロボティクス	1 年
矢沢	ロボティクス	2 年

問題 2

list3-1.c の会計をコーヒーとジュースではなく、別のもの（たとえば紅茶とココア）にしなさい。金額は好きに決めてよい。以下をすること：

- メッセージを変える
- 変数名を変える
- 単価を変える

問題 3

list3-1.c を書き換えて、メニューを増やしなさい。以下をするとよい。

- 変数を追加する（宣言を忘れない）
- scanf() の行を追加
- price() 関数の引数を増やす

3.19.2　コンパイラのメッセージ

問題 1

以下の間違いのあるプログラムを直して、コンパイラの出すメッセージを確認しなさい。そしてコンパイラのメッセージに合わせて、プログラムをどう直したらよいか、自分用のメモを作りなさい。

1. list3ex1.c には、「;」が抜けている行が 5 行ある。

(a) そのままコンパイルして、コンパイラの出すメッセージをメモしなさい。
(b) 「;」が抜けている行を確認し、メッセージと比較しなさい。
(c) 1 行ずつ修正して、コンパイル、実行できることを確認しなさい。

2. list3ex2.c は、コメントがよくないためコンパイルできない。
3. list3ex3.c には、全角空白の入った行がある。
   (a) そのままコンパイルして、コンパイラの出すメッセージをメモしなさい。
   (b) 全角空白の入った行を確認し、メッセージと比較しなさい。
   (c) 修正して、コンパイル、実行できることを確認しなさい。
4. list3ex4.c は、変数の宣言が抜けている。
5. list3ex5.c は、一部変数名が間違っている。
6. list3ex6.c は、関数定義のところで関数名が間違っている。
7. list3ex7.c は、関数呼び出しのところで関数名が間違っている。
8. list3ex8.c は、プロトタイプ宣言の関数名が間違っている。
9. list3ex9.c は、プロトタイプ宣言で型を間違っている。
10. list3ex10.c は、関数呼び出しのところで引数の数が間違っている。
11. list3ex11.c は、予約語を変数に使っている。

3.19.3 printf() と scanf() の間違い探し

問題 1

list3ex12.c は printf() を呼び出すときの書式文字列と変数の対応関係に間違いがある。

1. そのまま実行して何が起こるかみなさい。
2. 変数に合わせて書式化文字列を直しなさい。

問題 2

list3ex13.c は scanf() を呼び出すときの書式文字列と変数の対応関係に間違いがある。

1. そのまま実行して何が起こるかみなさい。
2. 変数に合わせて書式化文字列を直しなさい。

3.19.4　プログラムを改造する (その 2)

問題 1

ドルやユーロを使う場合には、小数点以下の計算が必要になる。list3-1.c を書き換えて、コーヒーが 1.5 ドル、ジュースが 2.3 ドルのときに会計が出来るように直しなさい。以下が必要になる。

- int を double に変える
- scanf() の引数内の %d を %lf にする
- printf() の引数内の %d を %f にする
- メッセージの中身を円からドルにする

エディタの置換機能は使わずタイプすること。タイプミスがあれば上の演習を思い出して直しなさい。

3.19.5　プログラムを読む

問題 1

list3ex14.c のプログラムについて以下をしなさい。

1. 実行される順番を書きなさい
2. c) コメント, f) 関数の実体, p) 関数のプロトタイプ宣言, v) 変数の宣言の記号をつけなさい
3. 各 sum 変数が代入されている行で、sum 変数に代入される値を書きなさい
4. 実行結果を書きなさい