平成２５年度
成長分野等における中核的専門人材養成の戦略的推進事業

企業提案型プロジェクト学習

実践教材

自動車組込み分野の中核的専門人材養成の実践的教育プログラムの実証

目次

1. はじめに

本書では、以下の知識を有している読者を想定している。

- 基本的な Windows XP の操作
- プログラミングの経験（C 言語または C++）
- Excel でのファイル編集
- エディタでのファイル編集

また、「ユニット積上げ式のモデルカリキュラム」で検討した新規科目の一部を新たに盛り込み、内容の拡充を図っている。

1.1. 本書の目的

本書は、産業技術である複数のシステム統合を行う際に必要なインターフェース・通信規格などの基礎、使用法を学び、その実践及び応用技術についての解説を目的としたテキストである。

ミニカーを題材に、実務に近い形式で実習を行うことで、自動車組込み産業の人材ニーズに対応した実践的な知識・技術を有する技術力を持つ人材の育成を目指す。

1.2. 本書の構成

1 章から4 章までが共通する概論、5 章では赤外線コントローラ、6 章では設計について解説している。7 章は開発環境の構築手順、8 章から10 章についてはプログラム開発の基礎内容の解説、11 章から13 章では実践的使用法の導入実習の解説と製品の品質管理について記載している。

ミニカーのコントロール方法は、赤外線コントローラからのコマンド送信により行う。

対象とするミニカーは、「チョロQハイブリッド！ マリオカート Wii VS タイプ」とする。

1.3. PC 環境

本書で想定するパソコンの構成を以下に示す。

OS	Windows XP Professional/Home SP3
CPU	2GHz 以上
メモリ	512MB 以上推奨
ディスク	2 GB 以上

1.4. 使用機材

使用機材について、以下に示す。

- ミニカー　一式
  - チョロQハイブリッド！ マリオカート Wii VS タイプ（タカラトミー）
    本書では、チョロQハイブリッド！ マリオカート Wii VS タイプを
    例題として記載する。

- 赤外線送信コントローラ　一式

- PIC ライター　一式

2. 論理学

日常言語が人とのコミュニケーションを第一としている以上、論理的であると同時に感情や環境なども含めた様々な情報の伝達ができるということが重要である。

一方、アルゴリズムやプログラムの開発、回路設計などでは、情報は本質的なもの以外は一切取り払われ、その情報の取扱いも日常とは異なった「論理的な思考」や論理の表現手段（言語）が必要となる。

たとえば、すでに回路設計や探索問題などでは論理の代数的表現であるブール代数や述語論理が、また、論理表現手段としては C 言語などが用いられている。情報の取扱い方として「論理的に考える」ことは、プログラム開発を行う上で非常に大切なことである。

「論理」とは、人間の思考が形式化された思考パターンのことであり、論理的思考すなわち物事を論じるときの論じ方そのものの正しさは Aristotle 以来議論されてきた。特に 19 世紀以降、George Boole（ブール）、Augustus de Morgan(ド・モーガン)、GottlobFrege(フリーゲ)などが今日の論理学の基礎を作り現在に至ったとされている。

論理学は大きく次の二つに分けられる。

- 命題論理学
- 述語論理学

単に、このような論理学の知識を得れば論理的な思考を身に着けることができるというものではないが、大事なことはこのような考え方、頭の働かせ方、そのこと自体が論理的なものの考え方を鍛えていくということである。

ここでは、論理的思考にとって必要な最低限の命題論理学の一端を学んでいく。

2.1. 命題論理

2.1.1. 命題と命題論理

命題とは、文章で表された真、偽（True，False）を問うことのできる主張や判断で、真（T）か偽（F）のいずれかの値をとり、両方またはいずれでもない値はとらない言明文をいう。

例えば次の①から⑤の文章を考えてみる。

①今日は天気が良い

②△ABC で∠A = ∠B = 60°なら△ABC は正三角形である

③今日は非常に暑い

④1 組のサイコロを投げた場合、「目の和は最大で 13 である」

⑤片付けなさい

表 2-1

番号	命　題	真　　偽	説　　明
①	素命題	真または偽となる	一つの言明文
②	複合命題	真	～ならば･･･である。真の命題である
③	素命題	真または偽となる	一つの言明文
④	素命題	偽	一つの言明文。偽の命題である
⑤	×	ー	真または偽を問えないので命題ではない

命題論理とは、一つの言明文（素命題または基本命題とも言う）が「～でない」、「そして」、「または」、「ならば」などで結び付けられた複合命題となるとき、素命題と複合命題の論理的関係を論じる体系のことを言う。

命題論理の主題は、素命題から複合命題をつくるとき、素命題の真偽が、複合命題にどのように関係するかを明らかにし、前提となる事実から結論を導き出す推論の有効性を確かめることである。

命題論理では、命題そのものの内的な性格までは立ち入らない形式論理である。そうすることによって、命題は、p，q，r ･･･などの記号を用いることができ、この記号は T（真），F（偽）の二つの値をとる変数として扱うことができるようになる。

また、「～でない」、「そして」、「または」、「ならば」などは表 2-2 に示すような論理記号が用いられ、次項で命題論理の概要について記す。

2.1.2. 複合命題

複合命題はいくつかの素命題を論理記号（接続詞）によって結び付けられて形成される。

表 2-2　論理記号の種類

番号	論理記号	名　　称	用法例　　意味
①	～	否定	～p：p でない
②	∨	選言	p∨q：p または q
③	∧	連言	p∧q：p かつ q
④	→	含意（条件）	p→q：p ならば q
⑤	↔	同値（双条件）	p ↔ q：p ならば q かつその時に限る

2.1.3. 複合命題と基本命題の真理値

いくつかの素命題から論理記号によって、複合命題を作成することができ、素命題と複合命題の真偽は、真理値表で解釈することができる。

上記表の①～⑤の真理値表は次のようになる。以下に p，q の 2 変数の場合を示す。

①否定　～p

p	～p
T	F
F	T

図 2-1

②選言　p∨q

※

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F

図 2-2

※第 1 行目　p と q の両方が真のとき p∨q が真となっている。
このような選言は包含的選言（p または q、または両方）と言われる。

日常使用する「または」とは、例えば「私はこの夏ハワイかグアムに行く」といった場合、いずれか一方が選ばれ、両方を意味しない。このような場合非包含的選言（p または q ただし両方ではない）の意味合いが強く、日常使われる意味とは異なっている。

③連言　p∧q

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

図 2-3

④含意　p→q

※

p	q	p→q
T	T	T
T	F	F
F	T	T
F	F	T

図 2-4

・命題 p は命題 q であるための十分条件
・命題 q は命題 p であるための必要条件

※p は前提条件（または前件部）、q は後件部で、含意は前定条件 p が F なら q の真偽にかかわらず真となっている。

たとえば、p ：宝くじに当たった　q：10 万円あげる　とする。　p→q の上記図 2-4 の真理値表に照らしてみると、※の 3 行目が不可解に感じるが、前提が偽なら後件の真偽に関わらず真とする。即ち、当たった場合については言明しているが、当たらなかった場合については述べられていないいわゆる「善意の解釈」を与える（寛容の原則という）。

⑤同値　p↔q

p	q	p↔q
T	T	T
T	F	F
F	T	F
F	F	T

図 2-5

・論理的には
（p→q）∧（q→p）と同値である

・命題 p は命題 q の必要十分条件
・命題 q は命題 p の必要十分条件

同値は上記図 2-4 で T であった 3 行目が F となっている。

例　論理式　p→(q∨r)

　　p：1cm は 10mm である　q：1m は 10cm である　r：1m は 1000mm である。

として、上記論理式の真偽を示せ。

　上記素命題の真偽は　p：T　　q：F　r：T　であり、図 2 の 3 行目および図 4 の 1 行目から、p→(q∨r)=T→(F∨T)=T→T=T

2.1.4. 命題論理の基本的性質

(1) 結合律

$p \vee (q \vee r) = (p \vee q) \vee r$

$p \wedge (q \wedge r) = (p \wedge q) \wedge r$

(2) 交換律

$p \vee q = q \vee p$

$p \wedge q = q \wedge p$

(3) 吸収律

$p \wedge (p \vee q) = p$

$p \vee (p \wedge q) = p$

(4) 分配律

$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$

$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$

(5) 排中律

$p \vee \sim p = T$

(6) 矛盾律

$p \wedge \sim p = F$

(7) 二重否定

$\sim(\sim p) = p$

(8) ベキ等律

$p \vee p = p$

$p \wedge p = p$

(9) ド･モーガンの法則

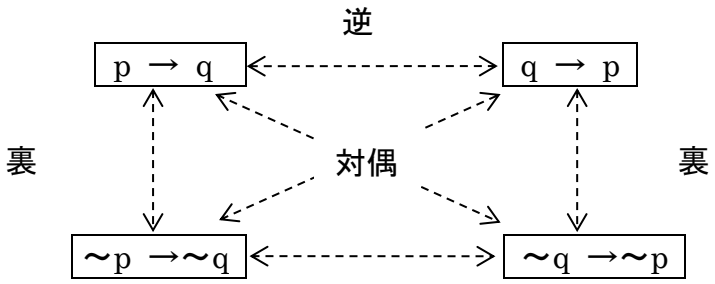
$\sim(p \vee q) = \sim p \wedge \sim q$

$\sim(p \wedge q) = \sim p \vee \sim q$

(10) 対偶

$p \rightarrow q \quad = \sim q \rightarrow \sim p$

なお、命題 p→q があったとき、その「裏」、「逆」の命題は下記の通り。



・逆は必ずしも真ではない　・裏は必ずしも真ではない　・対偶は、真である。

(11) 定数の除去

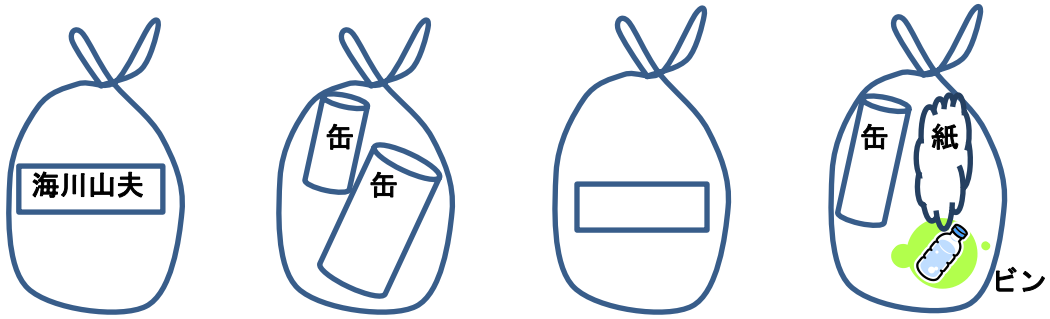
$p \wedge T = p$　　$p \vee F = p$

◆コラム◆

問題

ある団地ではごみ袋に名前を書いてごみの分別収集の徹底を呼び掛けている。収集の係員は「ごみ袋に名前が書いてあるならごみはきちんと分別されている」
という経験則に基づいて、たくさんのごみの整理に当たっている。今日は缶の収集日である。
この経験則が正しいかどうかを確かめるには少なくとも次のどのごみ袋を調べたらよいか。

(1)　(2)　(3)　(4)



（答え　(1)、(4) ）なお、(2)は裏、(3)は逆で必ずしも真ではない

(12)　変数の除去

$p \wedge F = F$

$p \vee T = T$

(13)含意記号→、同値記号↔の除去

$p \rightarrow q = \sim p \vee q$

$p \leftrightarrow q = (p \rightarrow q) \wedge (q \rightarrow p) = (\sim p \vee q) \wedge (\sim q \vee p)$

2.2. 論理式の解釈

ある論理式が、n個の異なった素命題で構成されるとき、2^n個の解釈が存在する。即ち、2^n個の真理値が存在し、論理式は、真理値表で解釈することができる。

ある論理式がすべての解釈において真である場合、そのような論理式（命題）はトートロジー（論理的に真な命題、恒真式）であるという。

また、ある論理式（命題）がすべての解釈において偽であるとき、そのような論理式は恒偽式といわれ、恒真式の否定は恒偽式である。

論理式は真になる解釈が少なくとも一つ存在する充足可能な論理式と真になる解釈は一つも存在しない充足不能な論理式に分けられる。

下記にトートロジーの一例をあげる。

$((p \rightarrow q) \wedge p) \rightarrow q$

表 2-3　　トートロジー

p	q	$((p \rightarrow q) \wedge p) \rightarrow q$
T	T	T
T	F	T
F	T	T
F	F	T

2.3. 有効な推論

推論とは一つの命題の結論が他の命題（前提）から導かれることを言う。

推論の有効性は、前提がすべて真で、結論もまた真であるとき、かつそのときに限って有効である。推論は大きく分けて一般的なものから特殊なものを導くものと特殊なものから一般的なものを導くものと 2 つあり、前者は演繹的推論、後者は帰納的推論と呼ばれている。記号論理は演繹推論で有効な推論法として知られている 3 つを下記に挙げる。

(1) 肯定式

$$\frac{p \to q \quad p}{\therefore q}$$

(2) 否定式

$$\frac{p \to q \quad \sim q}{\therefore \sim p}$$

(3) 三段論法

$$\frac{p \to q \quad q \to r}{\therefore p \to r}$$

表 2-4　真理値表による(1)の証明

p	q	p → q	p	(p → q)∧p	(p → q)∧p→q
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	F	T
F	F	T	F	F	T

例題. 上記（2）否定式の推論の有効性を確かめよ。

(p → q)∧(∼q)→(∼p)が妥当（すべて T になること）であることを示せばよい。

ここでは、命題論理の基本的性質または定理を使って証明すると

(p → q)∧(∼q)→(∼p)＝∼((∼p∨q)∧(∼q)∨(∼p)　・・・・・(13) より
＝((p∧∼q)∨q)∨(∼p)　・・・・・・・(9)
＝((p∨q)∧(∼q∨q))∨(∼p)　・・・・・(4)
＝(p∨q)∨(∼p)　・・・(11)
＝(p∨∼p)∨q　・・・・(1)
＝T∨q　・・・・(5)
＝T　・・・(12)

となり、この推論は有効である。

例題

１.「靴が半額セールならば花子はその靴を買う」この否定はどのような文になるか。

前件を P、後件を Q とすると　与えられた文は　P→Q　この否定は

～（P→Q）＝～（～P∨Q）＝P∧～Q

答え：「花子は半額セールでもその靴は買わない」

２. 前提①：民主的な学級では、学級長はクラスメイトによって直接選挙される

前提②：ある学級 A では、学級長はクラスメイトによって直接選挙されない

結論：したがってある学級 A は民主的でない

この推論は有効か。

前提①の前件部、後件部をそれぞれ P、Q、前提②を～Q とすると結論は～P となり（前記(2)の推論の否定形式）、推論自体は有効である。しかし、結論は真とは言えない。

推論の有効性は、結論が真か偽かということから決定されるものではない。この場合、誤った前提から正しく結論が導き出されたということである。

すなわち、推論の導出は正しいが、論証は誤っているということになる。

2.4. ブール代数と論理演算

ブール代数は、ある命題 X の真偽をそれぞれ 1、0 で表し、論理に代数的表現を与えることによって、複雑な論理や推論を全く代数的計算と同じく機械的に計算できる。そして、命題論理の記号をそれぞれ下記のように置き換えることにより、命題論理の基本的性質もそのまま成立する。

表 2-5　命題論理とブール代数

命題論理	ブール代数		
	論理記号	用法例	意　味
否定　～	否定　－（バー）	$\overline{A}$	A でない
選言　∨	論理和　＋	A＋B	A or B
連言　∧	論理積　・	A・B	A and B
真　T	1		
偽　F	0		

◆コラム◆

問題　3 つのセンサスイッチ P、Q、R がある。このうち二つ以上が感知（ON）した時、ランプ A が点灯する回路は、どのような回路となるか。ただし、命題は次のように定める。

p：スイッチ P がオン　q：スイッチ Q がオン　r：スイッチ R がオン

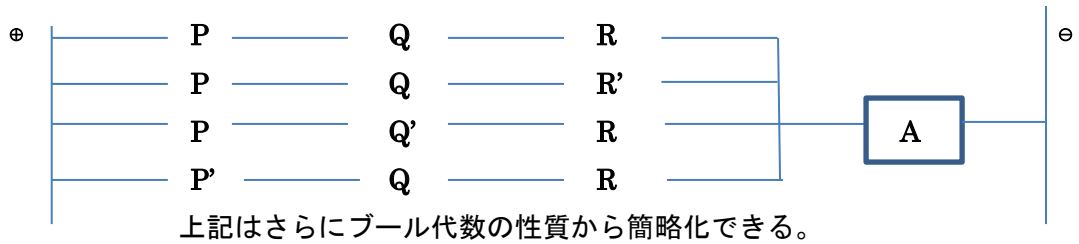
また、それぞれのスイッチの OFF は P’、Q’、R’であらわすものとする。

p	q	r	題意の真理値	対応する論理式
T	T	T	T	p∧q∧r
T	T	F	T	p∧q∧～r
T	F	T	T	p∧～q∧r
T	F	F		
F	T	T	T	～p∧q∧r
F	T	F		
F	F	T		
F	F	F		

題意を満足する真理値表は 1、2、3、および 5 行目である。従って論理式は

(p∧q∧r)∨(p∧q∧～r)∨(p∧～q∧r)∨(～p∧q∧r)

回路は



上記はさらにブール代数の性質から簡略化できる。

3.　アルゴリズム

3.1. アルゴリズムとは

アルゴリズムとは、全てのプログラムの基礎となる物で、品質の高いプログラムを作るため、再利用性の高いプログラムを作るため、さらにはコンピュータの思考を理解するためにも必要な知識である。

アルゴリズムは、単なる処理手順と表現される事もあるが、その優劣によって性能評価には大きな差が出る。特に計算速度にはその差が顕著に現れ、それを端的に示す例としてD.Gale（デイヴィッド・ゲイル）と L.S.Shapley（ロイド・シャプレイ）が提唱した「安定した結婚」と言う問題が有る。（提唱者の名前を取って、ゲイル＝シャプレイアルゴリズムと呼ばれる。）

********** 問題 **********

独身男女各Ｎ名ずつ集団お見合いをし、各相手に対する好感度を出した。
彼らは、お互いが、自分の結婚相手よりも好感度が高いと浮気をしてしまう。
彼らの全員が浮気する心配の無い結婚の組み合わせを見つけなさい。

ルール

男性から女性にプロポーズする事とする。
プロポーズされた女性は、未だ婚約していないなら、このプロポーズを受諾して
婚約しなければならない。婚約しているならば、婚約者とプロポーズしてきた男性と
比較して、婚約者の方が好きならばプロポーズを拒絶し、そうでなければ
婚約を解消して、新しく婚約する。
婚約している男性は、他の女性にプロポーズ出来ない。
また、以前プロポーズして拒絶された女性にプロポーズする事は出来ない。

****

Ｎ＝４名として、男性をＡ，Ｂ，Ｃ，Ｄとし、女性をａ，ｂ，ｃ，ｄとする。
それぞれ、異性への好感度が図の通りだったとする。

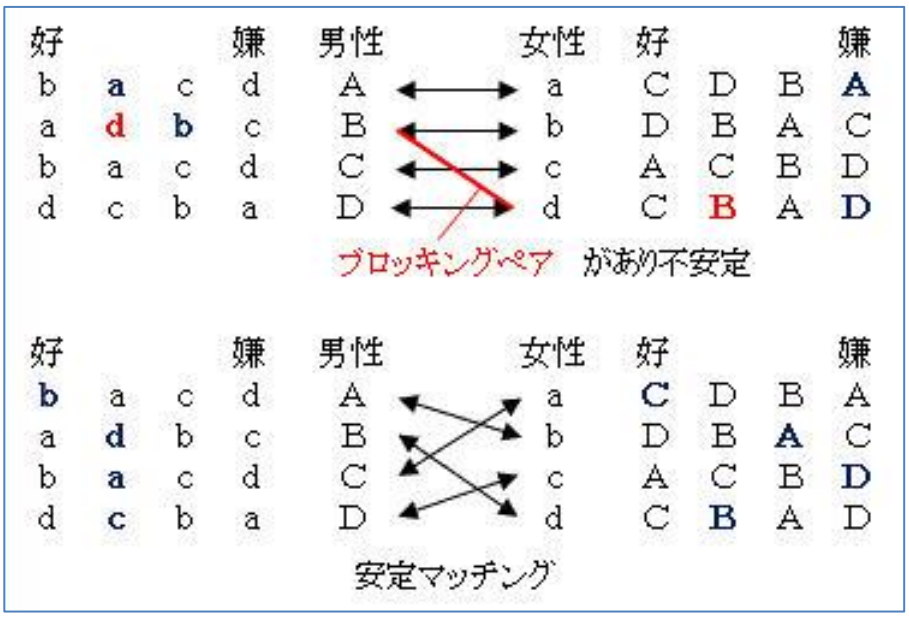


図 3-1　好感度

上の図の場合、Ｂとｄがブロッキングペア（浮気をしたい相手）となっている。
ｄにとっては、ＤよりもＢのほうが好きだし、Ｂはｂよりもｄのほうが好きなので
現在の関係を破壊してＢ－ｄの関係をもちたいと考える。
それに対して、下図（Ａ－ｂ、Ｂ－ｄ、Ｃ－ａ、Ｄ－ｃ）では、一方が現在の婚約者よりも好きな異性がいたとしても、その相手が自分よりも好きな人と婚約している（例えば、ｃは最悪のＤだが、他のＡ～Ｂは、みなｃよりも好きな人と婚約している）ので、不倫の関係が成立しない。すなわち、安定マッチングになっている。

この安定マッチングを探すための最も単純なアルゴリズムは、全ての組み合わせについて浮気相手がいないかを検証していく方法だが、これは人数によっては膨大な計算時間を必要とする。（全ての組み合わせの数は、人数の階乗※1 となるため）さらに浮気確認では、人数の２乗の計算が必要となり、このアルゴリズムでは「人数の階乗×人数の２乗」回の計算が必要となる。

※1…階乗：1x2x3x4x5x6x7x8x9x10..... という計算のこと。

人数	計算回数	
1	1	$1\times1^2=1$
5	3,000	$1\times2\times3\times4\times5\times5^2=3,000$
10	362,880,000	$1\times2\times3\times4\times5\times6\times\cdots10^2=362,880,000$
15	294,226,732,800,000	$1\times2\times3\times4\times\cdots15^2=294,226,732,800,000$
20	973,160,803,300,000,000,000	……

表 3-1　人数による計算回数（階乗計算の場合）

人数が増えるごとに、急激に計算回数が増える事が分る。

これを高速化するアルゴリズムを考えて行くが、そもそも全ての組み合わせを調べる必要はない。男性は、第一希望の女性に求婚する。この時、女性は別の相手がいない、もしくは居ても好感度が低ければ求婚してきた男性と婚約する。フラれた男性は、次のランクの女性に求婚する。つまり、男性はランクを落としながら、女性はランクを上げながら相手を探していく。

このアルゴリズムの手順を記載すると以下のようになる

１．男性が第一希望の女性に一斉にプロポーズする
２．女性は一番好みの男性をキープし、それ以外を拒絶
３．拒絶された男性は、次の希望の女性にプロポーズ
４．女性は、より好みの男性が現れるたびに、キープ相手を変更し、その他の男性を拒絶

この流れを、拒絶される男性がいなくなるまで続ける。

このアルゴリズムでは、Ｎ人の男性がＮ人の女性に求婚するので、計算回数は人数の２乗となる。先ほどのアルゴリズムと比較すると人数が増えるほど、計算に要する時間に大きな差が出てくる事が分る。

人数	計算回数	
1	1	$1^2=1$
5	25	$5^2=25$
10	100	$10^2=100$
15	225	$15^2=225$
20	400	$20^2=400$

表 3-2　人数による計算回数

アルゴリズムの性能評価では、計算速度が重要視される。
メモリ使用量もアルゴリズムによって異なる。高速なアルゴリズムでは２～３倍の差が出るが、速度ほどの差にはならない。計算の精度も、差が出る場合があるが、ほとんどのアルゴリズムは正確で、速度ほどの差にはならない。

その他に重要なのが、プログラムの作り易さである。高度なアルゴリズムほど時間がかかり、バグも混入しやすくなる。計算速度やメモリ使用量の差が、容認できるレベルであれば簡単な物が使われるケースが多い。

3.2. 様々なアルゴリズム

① **再帰的アルゴリズム**
再帰とは、Ａというものを定義するとき、Ａの定義のなかでＡ自体を含むことを言う。

② **ソート**
ばらばらなデータを、順序に従って並べ直す。様々な手法が有る。

③ **探索**
データの集合の中から、特定のデータを探索する。

④ **文字列検索**
文字列の中から、部分文字列を探しだす。

⑤ **データ圧縮**
データの容量を減らす。

⑥ **暗号化アルゴリズム**
RSA、DES、トリプル DES など。

⑦ **並列・分散アルゴリズム**
マルチプロセッサや OS のマルチタスクを利用して、複数の処理を同時に進める。

⑧ **近似アルゴリズム・ヒューリスティックアルゴリズム**
計算量が多く、現実的な時間内に答えが出せない場合、完全な正解では無く有る程度正解に近い答えを出す。

⑨ **遺伝的アルゴリズム**
生物の進化に見立てたアルゴリズム。解の候補を生成させ、より解に近い物を選択し、交叉・突然変異などの操作を繰り返しながら解を探索していく。

上記以外にも様々なアルゴリズムが存在するが、少なくともこれらは知識として知っておきたいアルゴリズムである。

3.3. アルゴリズムの記述

様々な問題を解決するとき、論理的な思考のもとで問題解決の手順を明らかにできれば、プログラムに落とし込むときには非常に有効である。代表的な手法としてフローチャートがある。フローチャートは問題解決の手法として、手続き型知識表現で、一般的にプログラムのソースを作成する際、その前段階に記述される。アルゴリズムの記述には、フローチャートが用いられる。これによって

(1) 論理を可視化でき、わかりやすい
(2) プログラムを複数の人で分けて開発でき、効率よく作成できる

従って一定の決まりのもとでフローチャートは作成されなければならない。
図記号は JIS 規格で定められている。

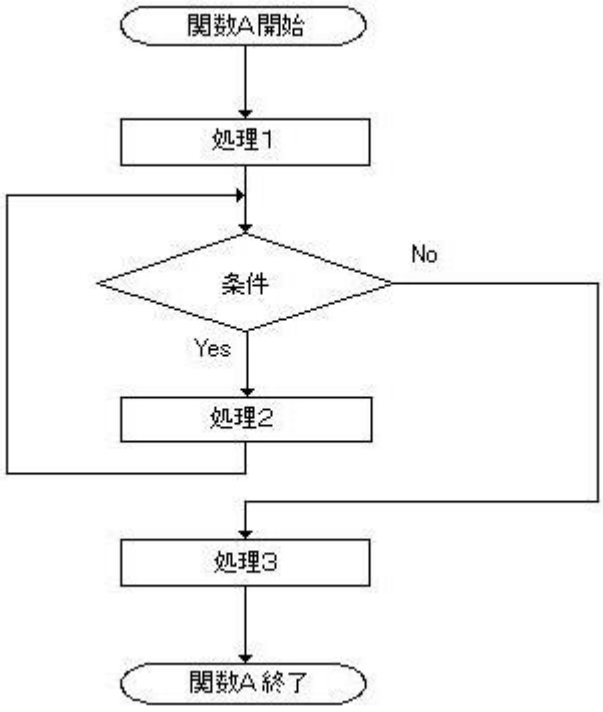
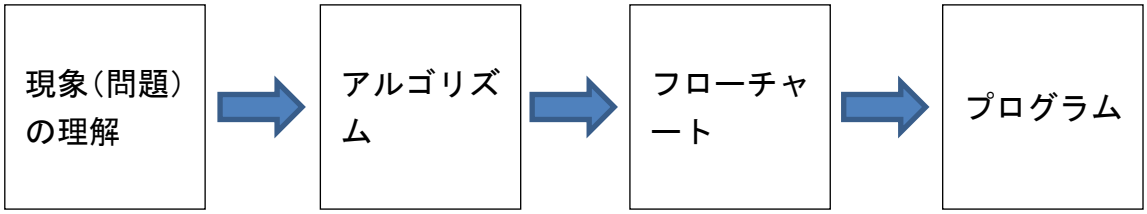


図 3-2　フローチャート

フローチャートの図記号については、設計の章で詳しく触れる。

3.3.1. フローチャートを作成する際の基本的事項

まず、プログラムまでの一連の手順を図に示す。



アルゴリズムやフローチャートは日常の簡単な現象や問題を取り上げ、論理的な視点に立って多くの問題にあたって考えることが重要である。

記述に当たっての基本的事項および注意事項を以下に列記する。

(1) 最初と最後を明記する
(2) 処理の流れは原則として次のようにする
・上から下へ
・左から右へ
・逆行する場合は、矢印をつけて表現する
(3) 交差しないようにする。また、線が不明確になっていないこと
(4) 処理が 2 重に行われていないか（一つの箱に一つの線が繋がっていること）
(5) 条件の確認　Yes　No の分岐に間違いがないかの確認をする

下記フローチャートは、「目玉焼き」の手続きの一例をフローチャートで示したものである。

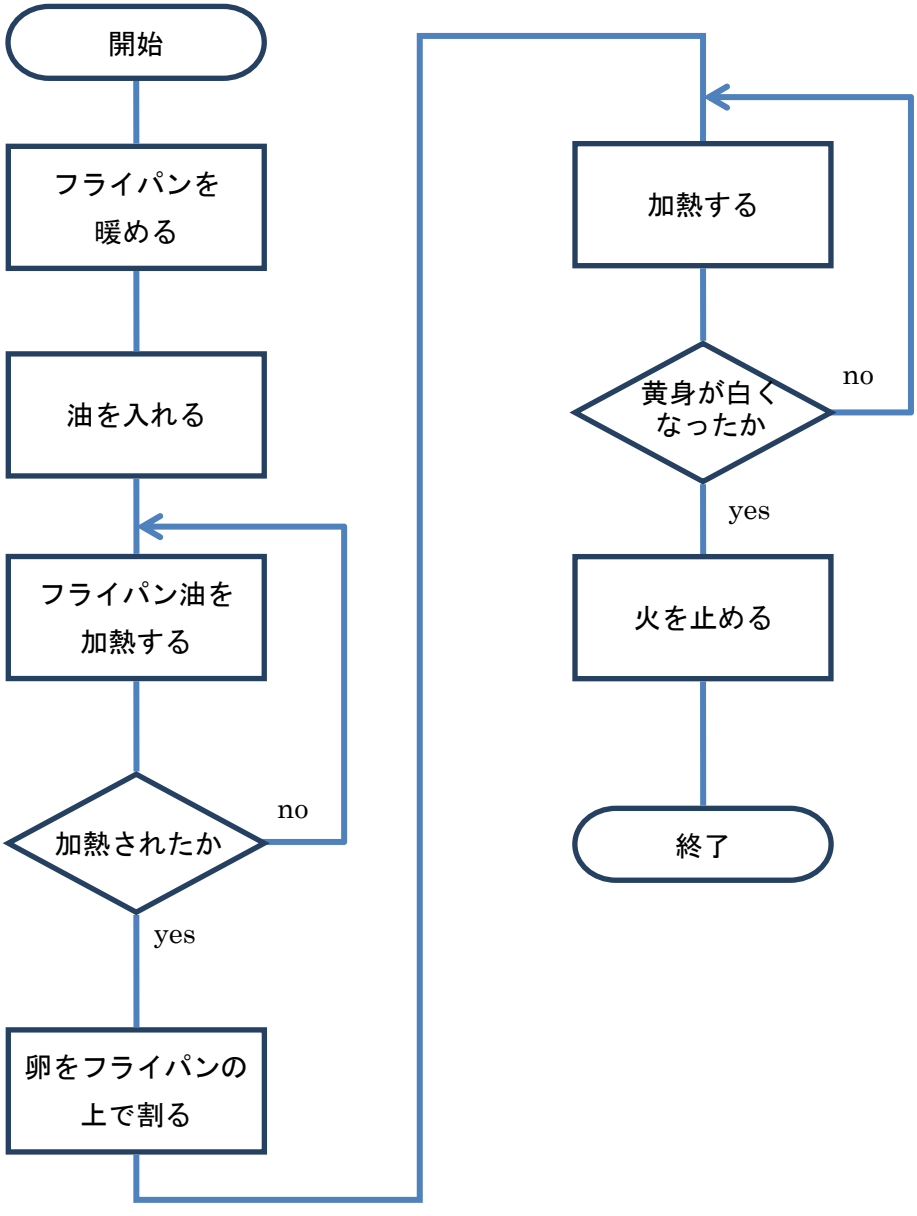


図 3-3　目玉焼きのフローチャート

3.3.2. 処理の構造

フローチャートの基本的な構造は次の 3 つである。

・順次構造　・分岐構造　・反復構造

これらを用いて問題のアルゴリズムの手続き知識をフローチャートで表現する。

(1) 順次（逐次）構造

各ステップで書かれた順番に出現する部分で、上から下へ順番に処理される。

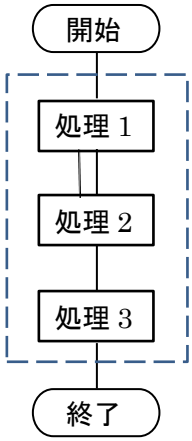


図 3-4

(2) 分岐（判断または選択）構造

条件によって枝分かれして「条件を満たす（yes）」の場合と「満たさない（no）」の場合は別の処理をすることになる。

条件によって処理される部分は 1 つだけで、処理が終わったら元の流れに戻す。

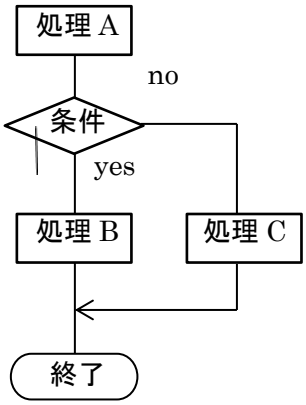


図 3-5

(3) 反復構造

ある条件を満たしているときは繰り返して処理をする構造。

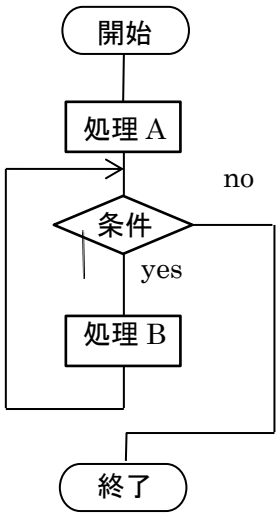


図 3-6

(4) 多重分岐構造

分岐をいくつか組み合わせた構造の一例を下に示す。

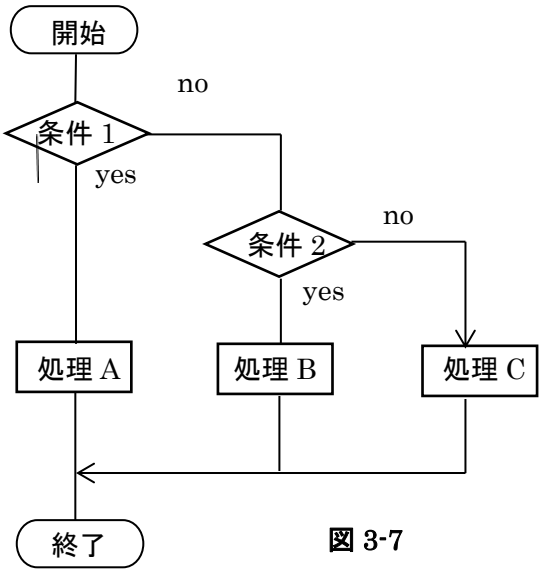


図 3-7

◆コラム◆

階段を上がるときの上がり方として次の二つの上がり方があるとする。

① 1段上がる　　②2段上がる（一段飛ばし）

問題

5段の階段があったら①と②を使って上がるとしたら何通りの上がり方があるか。

なお、1段の場合は①のみで1通り、2段の場合は①と②で2通りです。順次3段、4段の場合と列記していくと次のようになる。

階段の段数	1段	2段	3段	4段	5段	・・・・
上がり方	1	2	3	5	8	・・・・

従って5段の場合、8通りの上り方がある。

では10段の階段では何通りの上り方があるか。

（答え　89通り）

この上り方に出てくる数列はフィボナッチ数と言われ、あるｎ項の数は前の2項の和となっている。

なお、一般的にはフィボナッチ数は、次のように表される。

１、１、２、３、５、８、１３、・・・・・・・・

例. 上記コラムの階段の上がり方で
配列ｆの要素に10段までの上がり方を
格納するフローチャートを考えてみる。

数列は

$f_0 = 1$，$f_1 = 1$ 、

$f_n = f_{n-2} + f_{n-1}$

となる。

従って、図12のような
フローチャートとなる。

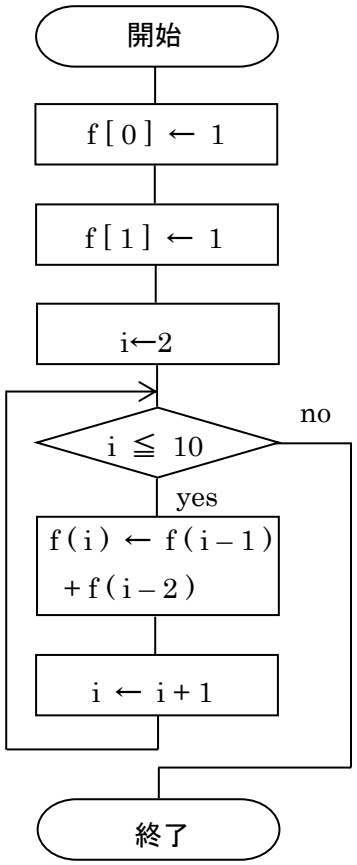


図12

4. 組み込みプログラム

現在、私たちの身の回りには、冷蔵庫、掃除機、洗濯機、炊飯器などといった、たくさんの電化製品がある。これらの電化製品のほとんどに、制御を行うためのコンピュータが入っている。

また、最近の自動車には、その高性能化に伴い、エンジンをはじめとする機器や装備を制御するための電子制御装置（ECU：Electronic Control Unit）が複数搭載されており、それらの制御にも小型のコンピュータが使われている。

これらのコンピュータは、ごくごく小さなもので“**マイコン**”と呼ばれる。このマイコンを動かすためのプログラムこそが「組み込みプログラム」である。

なぜ「組み込み」と呼ぶのかというと、例えば掃除機ならば「掃除機の動作を制御するプログラム」をマイコンに仕込んでおき、マイコンとともに機械に「組み込んで」使うものだからである。

今回の学習ではミニカー（自動車）を例題として取り上げる。

自動車にとってのマイコンは、「走る、曲がる、止まる」という基本機能を制御するだけではなく、安全、安心、快適などの付加価値を高めるための不可欠なパーツである。

4.1. マイコン

“マイコン”とは“**マイクロコントローラ**”(microcontroller)の略称であり、IC チップ一つ分の大きさしかない“小さな”コンピュータである。

機能や性能は一般的なパソコンと比べるまでもなく貧弱であるが、一方で非常に小さく、また消費電力も少ないことから、性能よりも省電力、省スペースであることのほうが重要なものなどによく用いられる。

以前、“マイコン”と言えば“マイクロコンピュータ”(microcomputer)を指す言葉であったが、少なくとも 2000 年代以降ではこの意味では用いられないので、注意する必要がある。

4.2. PIC（ピック）

PIC（Peripheral Interface Controller）とは、Microchip Technology 社が製造しているマイクロコントローラ（制御用 IC）製品群の総称であり、プログラミングが可能なワンチップマイコンの一種である。

ワンチップマイコンとは、一つの IC 上に CPU、メモリ、プログラムメモリなどが含まれていて、これに電源を供給するだけ（ものによってはクロック回路やコンデンサなどを付加する）で動作するマイコンである。

ワンチップマイコンは汎用的な処理を行うことはできないが、小さな IC 回路のみで特定の機能の処理を一手に行うことができる。そのため、コンピュータ制御を必要とする装置の多くに組み込まれている。

自動車や炊飯器の制御システムに採用されていたり、あるいはマウスやキーボードにおける入力情報の制御などにもワンチップマイコンが用いられている。

ワンチップのメリットとしては、省面積、低コスト、回路の設計や配線作業の負担が大きく減る事が挙げられる。

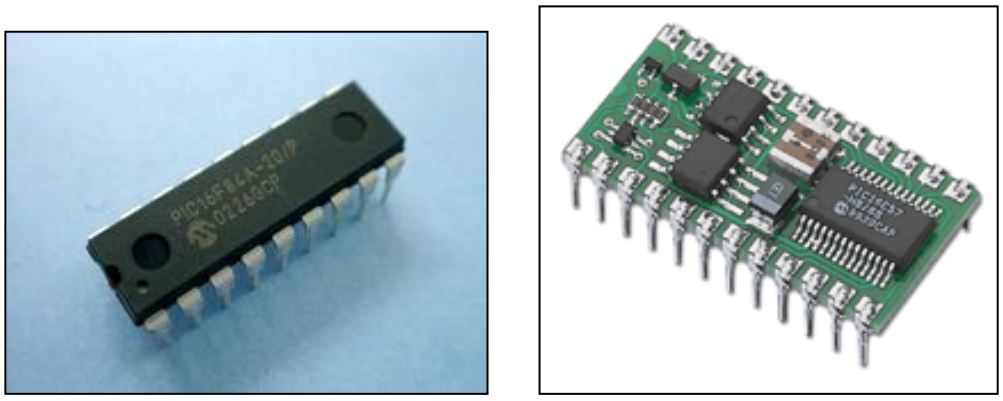


図 4-1　PIC とマイコン

5. 設計

設計とは、物（製品、システム、アプリケーションなど）を造る際に必要となる工程の一つである。

設計は、動作内容や制御内容の関連性を共通のモデル（図）を用いて表現し、関係者間でシステム全体の妥当性を判断するための方法である。仕様の確実な伝達を目的として設計し、設計書を作成する。

設計にはいくつかの種類があるが、大きく“基本設計”と“詳細設計”の２種類に分けられる。

次項は「付録 H.昨年度教材参照」の参照番号 1～3 を参照。(差し替え資料有り)

差し替え資料「図 3-1　設計書の位置づけ」と差し替え

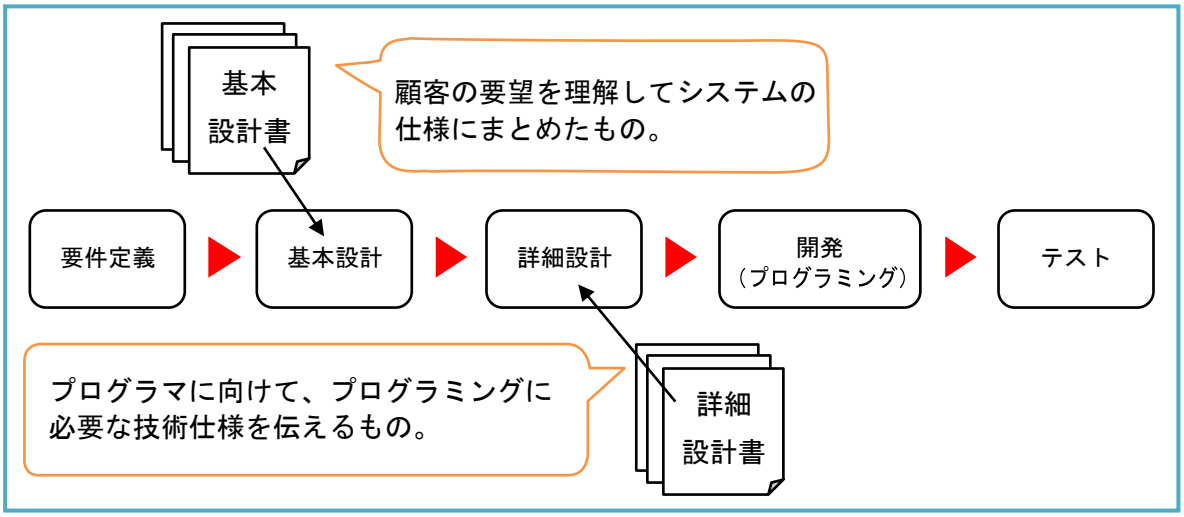


図 3-1　設計書の位置づけ

5.1. モデリング

ここで解説するモデリングとは、「3 次元グラフィックスにおいて、モデル（物体）の形状を作成すること」ではなく、共通のモデルを用いて、関係者間でシステム全体の妥当性を判断するための方法であり、設計の一部として使用する。

5.1.1. モデリングの必要性

自動車システムの開発だけでなく、業務アプリケーションや、クライアント／サーバ、ヒューマンインターフェースに至るシステムの開発では、プログラミング言語（C 言語、Java など)を用いて、システムを構築する。

ただし、プログラミング言語はすべて文字表現であり、その言語の仕組みを知っている人間でないと内容を理解できない。また、複数の機能を持つシステムを製造する場合、各機能間での関連性や、データの受け渡し方法を事前に定義しておかないと、機能間の連携が上手くいかず、システムが全く動作しない事態に陥ることになる。

これらの問題を製造前に払拭するため、システム全体を直感的で解りやすく理解するための手段として、**モデル(図)を作成(モデリング)**する。

モデリングでは、対象となる業務の問題領域で、必要かつ重要な現実世界のオブジェクトを中心に、その構造と関係をまとめる。視覚での表現であるモデルを基に、関係者間で全体概要を理解し、問題領域について共通の認識になっている事を確認する。

モデリングには、システム全体を直感的で解りやすく理解するためだけでなく、以下の利点もある。

- 記憶に残りやすい。
- 自分自身も、他人も概要が素早く理解できる。
- 全体の構造、関係がわかりやすい。
- 一目で要素同士の関連がわかる。
- 関係者間で共通の認識が得られる。（認識のずれを確認できる）
- システム全体を理解した上でコーディングするので、作業のやり直しが少なくなる。

しかし、モデリングも万能では無い。全体像と個々の要素の関係を示したりする事は出来るが、詳細な内容まで含めて表現すると、モデル全体が見えにくくなる。そのため、詳細な情報は文書（言葉）で補足して説明すべき場合もある。

図 5-2　モデリングの必要性

5.2. UML

5.2.1. UML の歴史

UML は“Unified Modeling Language”（統一モデリング言語）の略で、オブジェクトモデリングのために標準化した仕様記述言語である。

1990 年代前半に、オブジェクト指向分析、設計を支援する様々な方法論が出現し、書物などで知られるだけでも 20 種類以上あった。（ブーチ(Grady Booch)による「ブーチ法(Booch Method)」、ランボー(Jim Rumbaugh)による「OMT(Object Modeling Technique)」などが有名）しかし、当初は各方法論で表記法が異なると言う問題を抱えていた。さらには、それぞれの方法論者は、ただその優位性を主張するだけで「方法論戦争」とされ言われるほどであった。

こうした中、**1994 年**に表記法の統一への試みが始まる。前述の**ブーチ**と**ランボー**が中心となり、**Unified Method 統合**への取り組みが開始され、**1995 年 10 月**に統一方法論の草案(**Unified Method V0.8**) が OOPSLA'95 (Object-Oriented Programming, Systems, Languages, and Applications)で発表された。その後、ビジネスや利用者の視点でシステムを捉える「ユースケース」の概念を初めて著した、OOSE(Object Oriented Software Engineering)の**ヤコブソン**(Ivan Jacobson)も **1995 年**にこれに加わり、（この 3 人を「スリーアミーゴス」(the Three Amigos)と呼ぶ）その年、**統一方法論**の **0.9 版**と **0.91 版**が公表される。その後、方法論と記法を分離して、記法を UML として公開する事にした。そして、**1997 年 1 月に UML1.0** を、OMG(Object Management Group：オブジェクト指向技術の様々な標準を決める団体)に提案。**同年 8 月**、さらに改良を加えた **UML1.1** を提案し、**1997 年 11 月**、OMG より標準化認定を受けた。その後も、OMG は専門チーム（UML Revision Task Force）を設け、改訂のための作業を続けた。**1999 年の夏に公開された UML1.3** で多くの問題が解決されて、実用の域に達した。

さらに OMG は改訂を進め、2001 年 2 月に UML1.4、2003 年 5 月に UML1.5 を公開したが、これらはいずれも大きな変更を伴わなかった。その後、MDA(Model Driven Architecture)の実現に向けたモデルの記述力を高めるために、大幅な改定を試み、**2004 年 7 月に UML2.0** を公開。現在の最新バージョンは **UML2.4.1** となっている。

時期	出来事
1994 年	ブーチとランボーが Unified Method の統合開始
1995 年 10 月	Unified Method V0.8 が OOPSLA'95 で発表される
1995 年	ヤコブソンが参加(the Three Amigos)
1995 年	統一方法論 0.9 版と 0.91 版を公表
1997 年 1 月	UML1.0 を公表
1997 年 11 月	UML1.1 が OMG より標準認定を受ける
1999 年	UML1.3 で実用の域に達する
2004 年 7 月	UML2.0

表 5-1　UML 統一の流れ

5.2.2. 組込み開発での UML の利用

組込み開発における開発規模は、増大、複雑化の一途をたどっている。人海戦術による対応は限界に達しており、様々な解決策が模索されている。その解決策の一つとして、UML の利用が挙げられる。

UML はシステム全体の構造を把握するのに便利な表記法である。システムの構造を考えながら設計を行う事で、上流工程におけるシステム設計の品質を向上する事ができる。これによって、上流工程での不具合の混入を減らす事が出来、手戻りによる開発コストの増大を防ぐ事も可能である。また、大規模化する組込み開発において、共通言語である UML を使用して設計を行うと言う事は、メンバ間の意思疎通の円滑化に効果があり、海外リソースを活用する際にも有効である。

UML2.0 は「組込み分野への対応」がなされており、アーキテクチャの概念が強化されている。例えば「コンポジット構造図」では、ポート、インターフェースなど外部との通信を表現する事が可能となり、システムの構成などを容易に表現できるようになっている。また、時間概念の表現ができるよう、「タイミング・チャート図」などが追加されている。そしてシーケンス図においても時間的概念の表現が可能となった。よって、状態を保持する時間、また要求に応答する時間など組込み開発で意識すべき要素の表現が可能となっている。

今回の学習ではフローチャート図を元にモデリングしていくが、モデリングには複数の種類が存在する。以下、モデリングの手法について解説する。

5.2.3. ユースケース図

「Use case(ユースケース)」すなわち“システムの利用例”を図で表現する。
図を構成する要素は以下の 5 項目である。

① 作成するシステムには（サブジェクト）
② どんな利用者がいて（アクター）
③ どんな機能があり（ユースケース）
④ 利用者は機能を直接使えるのか（関連）
⑤ 機能はどの機能を使用するのか（包含（ほうがん））

ユースケース図の特徴として、利用者や使用する他システム、動作するモノは全て“アクター”として扱われ、人型で表される。

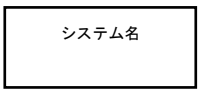
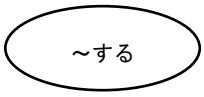
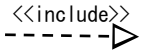
No	名前	意味	図
①	サブジェクト	システム全体	
②	アクター	利用者や使用する他システム	
③	ユースケース	作成する機能(RT ｺﾝﾎﾟｰﾈﾝﾄ)	
④	関連	利用者と機能の関係 (利用者が認識できる機能)	(ｱｸﾀｰとﾕｰｽｹｰｽを結ぶ)
⑤	包含	ある機能（ユースケース）が別の機能を使用する	(ﾕｰｽｹｰｽとﾕｰｽｹｰｽを結ぶ) 

表 5-2　ユースケース図の主な記号

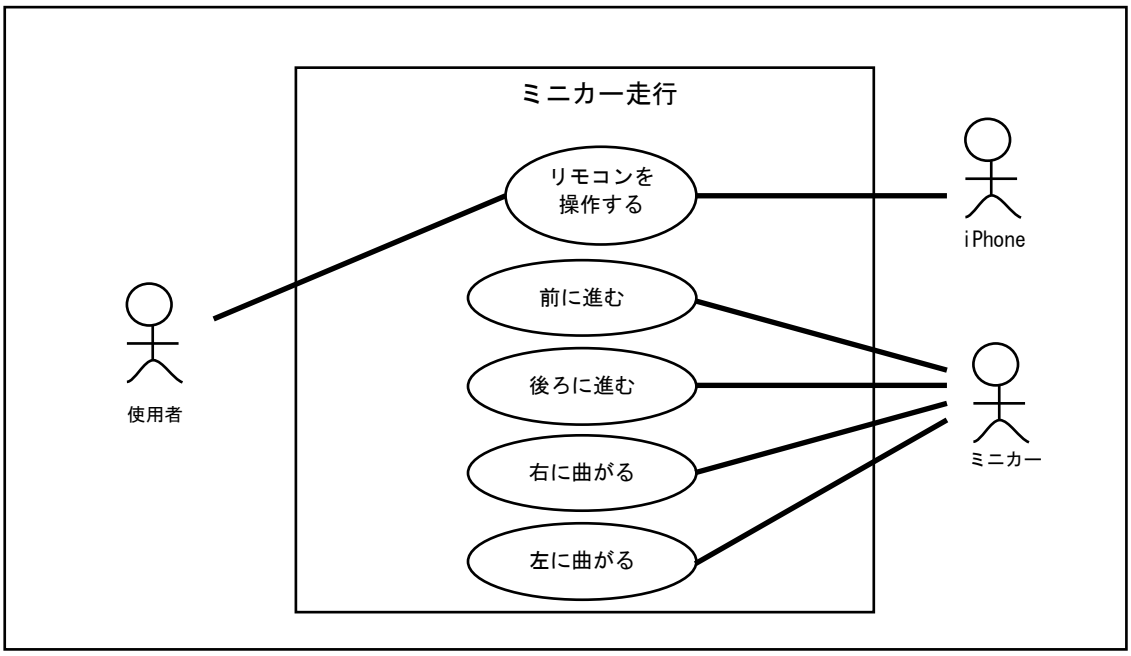


図 5-3　ユースケース図（例:iPhone で動くミニカー）

5.2.4. ステートマシン図

各機能の状態（ステート）の振り分けと、その状態へ移るための条件および効果を図で表現する。

No	名前	意味	図
①	開始状態	開始を示す	●
②	終了状態	終了を示す	◉
③	ステート	機能の状態	状態名
④	遷移	現在の状態から別の状態へ遷移	契機[条件]/効果 →
⑤	選択仮状態	条件により遷移が分岐する	◇

表 5-3　ステートマシン図の主な記号

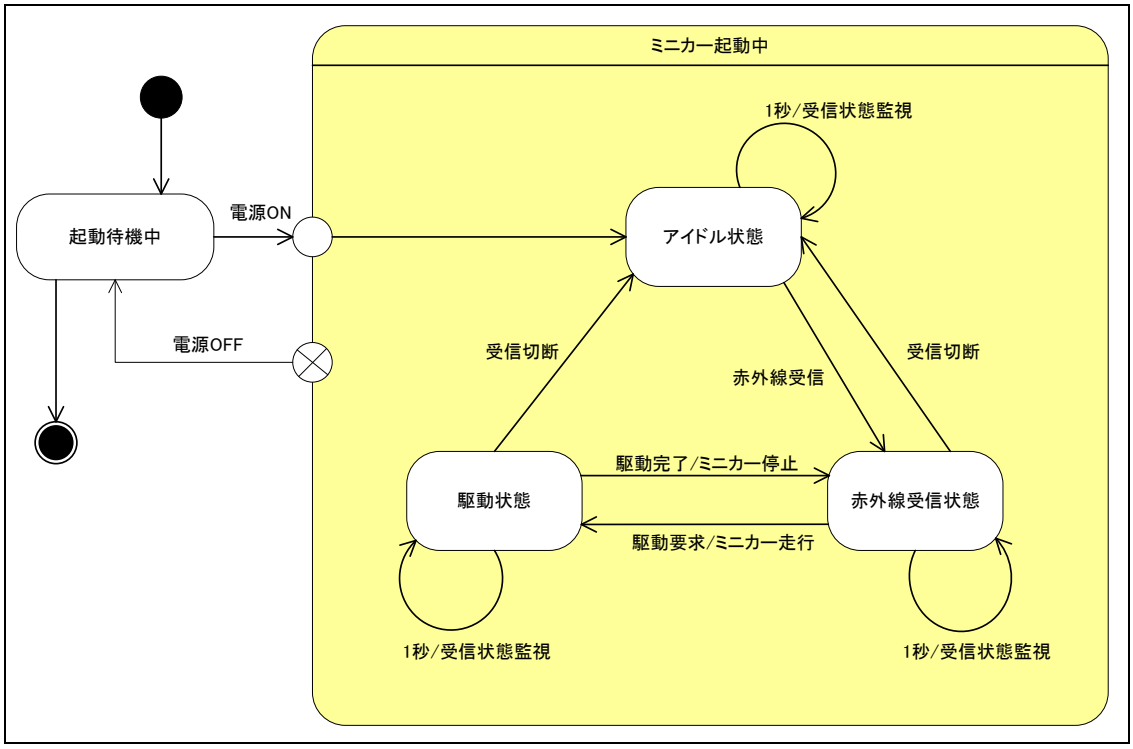


図 5-4　ステートマシン図（例:iPhone で動くミニカー）

5.2.5. クラス図

1 つ 1 つの機能についての関連性を図に表記する。各機能が行う動作（関数：メソッド）や、取り扱うデータ（変数：属性）を、クラス（クラス名の記述された四角）内に記入することで、どの動作がどの機能で使用されるのか、どのデータがどの機能に受け渡されるかを判断する。

オブジェクト指向のプログラム（C++など）の機能を表すことに向いており、クラス図を詳細に詰めていくことで、ソースコードに変換することができる。そのため、クラス図とソースコードの同期が可能な UML モデリングツールも存在する。

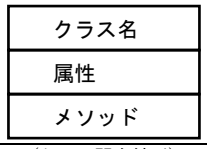
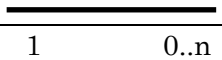
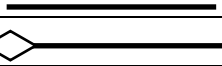
No	名前	意味	図
①	クラス	1 つの機能	
②	メソッド	1 つの機能が行う動作	
③	属性	1 つの機能が保持するデータ	
④	関連	クラス間の関係性を表現	
⑤	多重度	クラス間の数的関係を示す	
⑥	集約	全体とその一部分を示す	
⑦	public	他のｸﾗｽでも使用可な属性・ﾒｿｯﾄﾞ	＋
⑧	private	自ｸﾗｽのみ使用可な属性・ﾒｿｯﾄﾞ	－

表 5-4　クラス図の主な記号

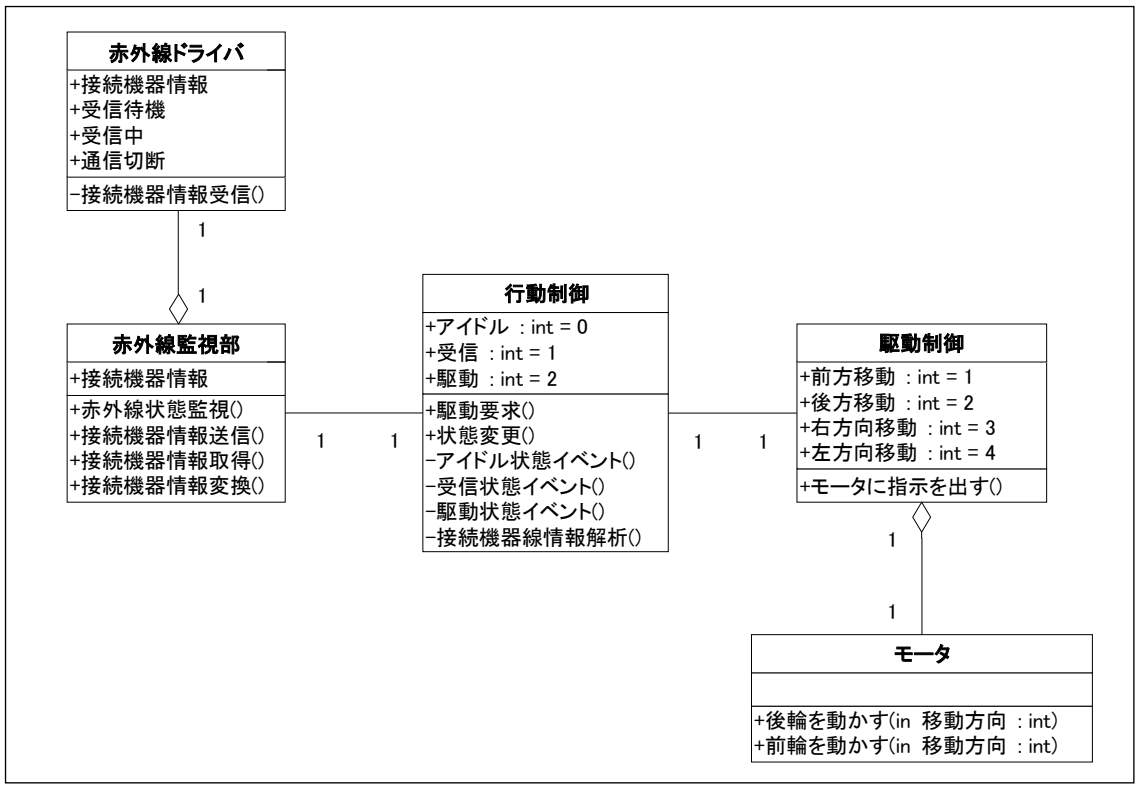


図 5-5　クラス図（例: iPhone で動くミニカー）

5.2.6. シーケンス図

各機能間のやり取りを時系列に沿って表現する。複数の機能のやり取りと、その順番が明確になる。

各要素については、クラス図で作成した名称を使用する。

No	名前	意味	図
①	オブジェクト	1 つの機能(クラス)	:クラス名
②	ライフライン	オブジェクトが生存している期間 点線で表す	
③	メッセージ	別のオブジェクトに働きかける	メソッド名
④	リターン	メッセージに対する返り値	返り値
⑤	活性区間	オブジェクトが活動中	

表 5-5　シーケンス図の主な記号

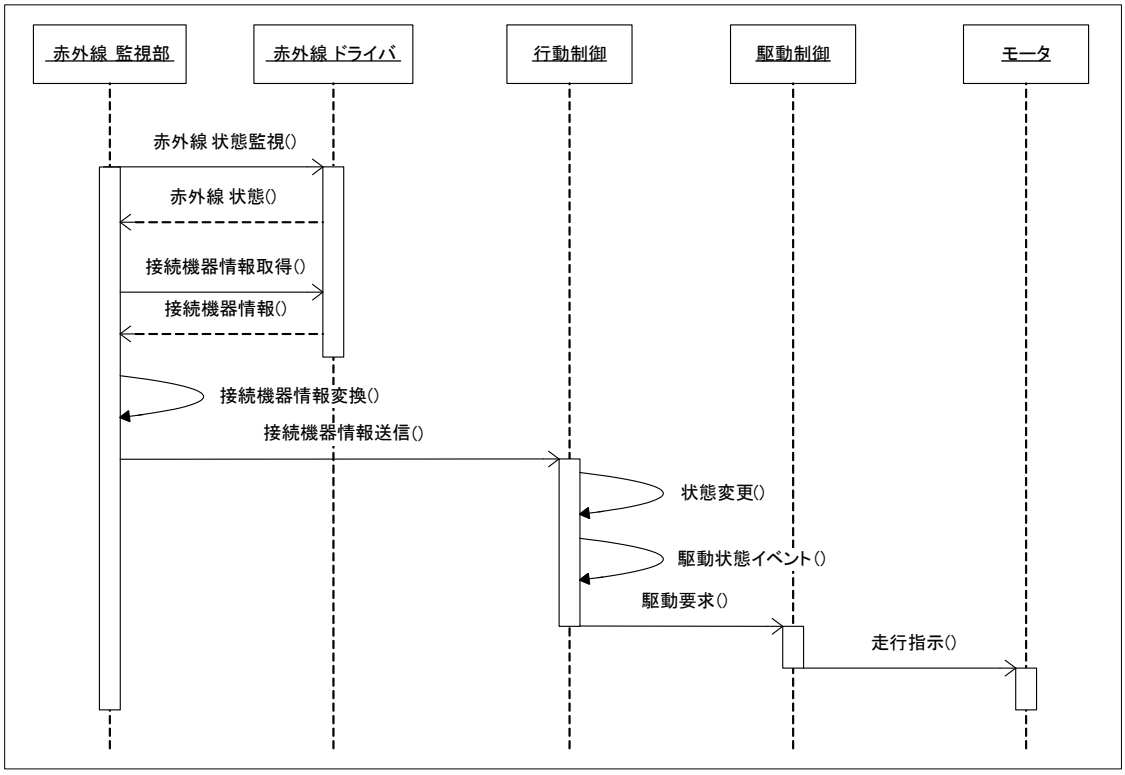


図 5-6　シーケンス図（例: iPhone で動くミニカー）

5.2.7. フローチャート図

“flowchart”とは、「流れ図」という意味であり、“処理の流れを図にしたもの”である。
フローチャート図を元にプログラミングを行うことができる。

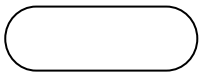
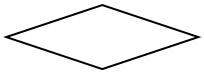
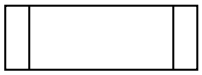
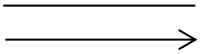
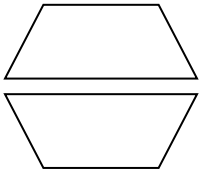
No	名前	意味	図
①	端子	フローの開始と終了を記述するための記号	
②	処理	処理内容を記述するための記号（計算、代入など）	
③	判断	条件分岐を記述するための記号	
④	定義済み処理	関数・別の場所で定義された処理を表す記号	
⑤	線	データの制御や流れを示すための記号 流れの向きを明示する必要があるときは、矢印を付ける	
⑥	ループ端	繰り返しの開始と終了を表す記号	
⑦	結合子	フローチャートを分割する場合や、処理の流れを違う場所へ移動させたい場合に用いる記号 丸の中に数字や文字を入れ、対になるように同じ数字または文字の結合子を配置する	

表 5-6　フローチャート図の主な記号

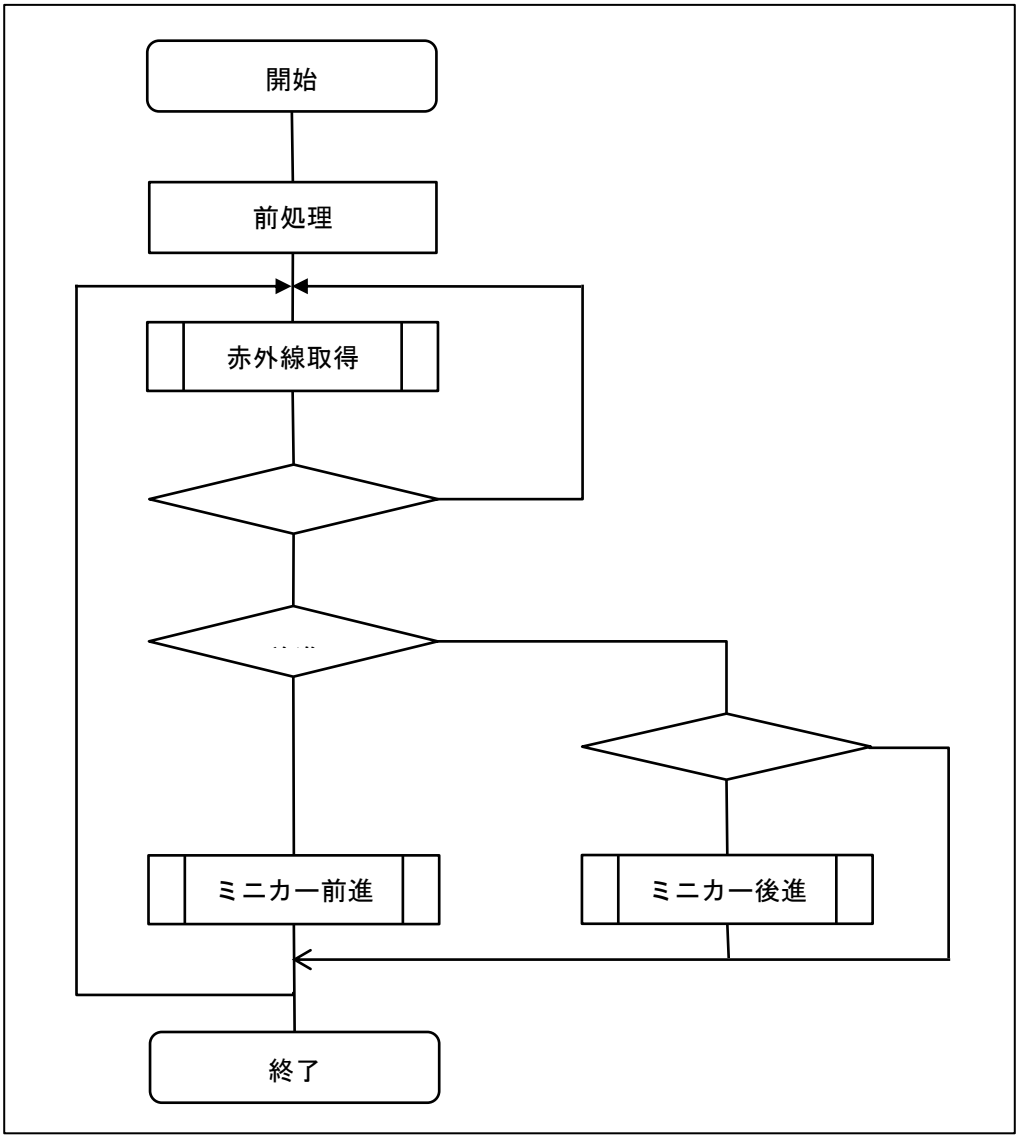


図 5-7　フローチャート図（例: iPhone で動くミニカー）

5.2.8. PAD 図

PAD は“Problem Analysis Diagram”の略で、直訳すると「問題分析図」となる。C 言語のような構造化プログラミングを行うことができる言語について、処理の流れと構造を「順次処理」「反復処理」「分岐処理」とサブルーチン（＝関数）の組み合わせで記述することができる。

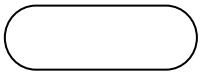
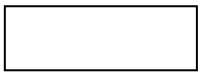
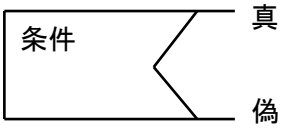
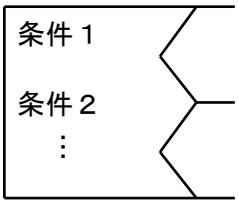
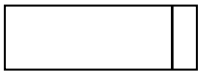
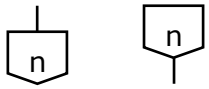
No	要素	意味	図
①	端子	開始と終了を表す 場合によっては省略される	
②	処理	処理や行動を表す	
③	選択	条件を判断し、条件によって処理を選ぶ	
④	多数から選択	条件によって多数の選択から１つ選ぶ	
⑤	前判定ループ	ループ終了条件をループに入る前に判定するタイプのループ	
⑥	後判定ループ	ループ処理の後にループ終了条件を判定するタイプのループ	
⑦	サブルーチン （定義済処理）	関数・別の場所で定義された処理	
⑧	線	記号を結ぶ	
⑨	結合子	対となる同じ数字・文字同士で処理がつながる	

表 5-7　PAD 図の主な記号

(1) 順次処理

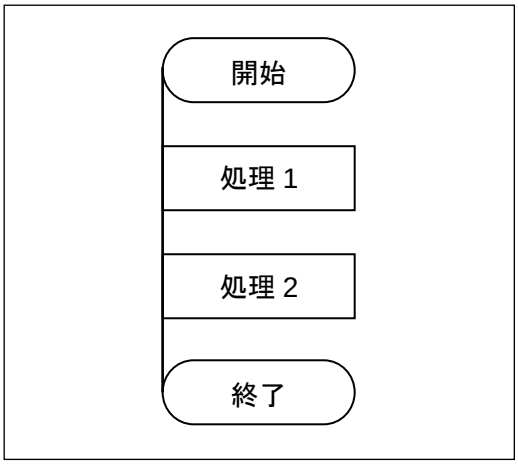


図 5-8　PAD 図（順次処理）

(2) 反復処理

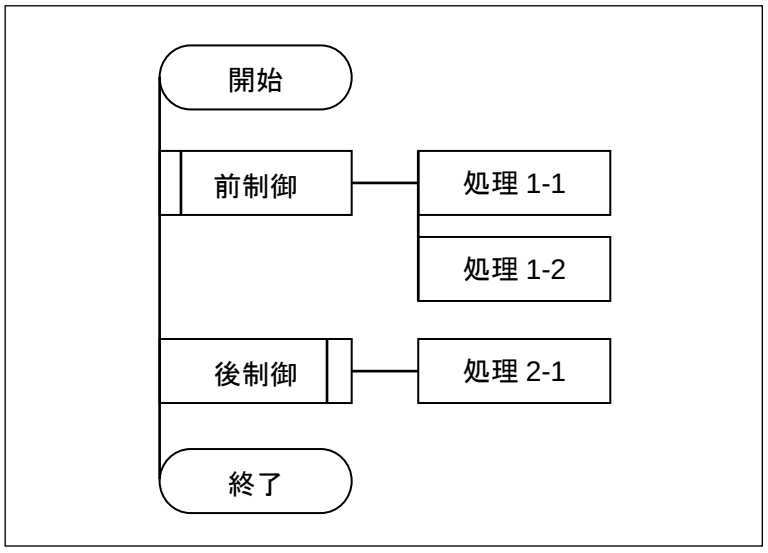


図 5-9　PAD 図（反復処理）

(3) 分岐処理

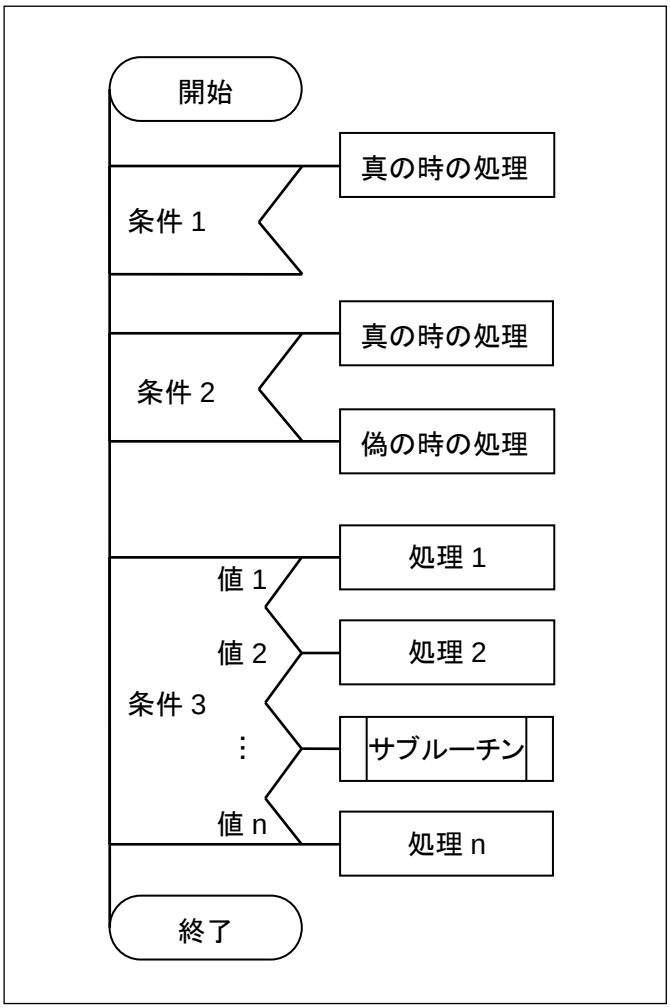


図 5-10　PAD 図（分岐処理）

(4) サブルーチン

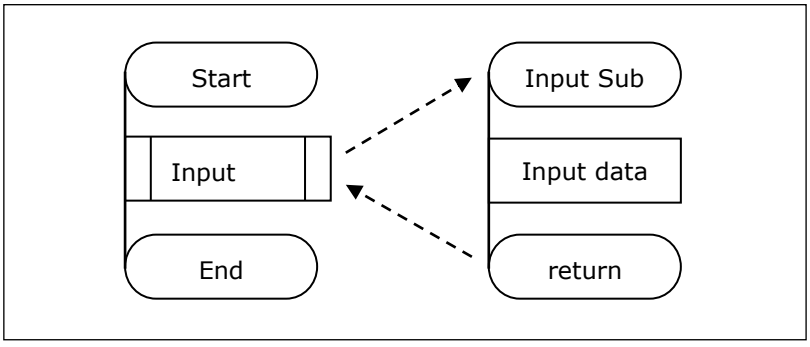


図 5-11　PAD 図（サブルーチン）

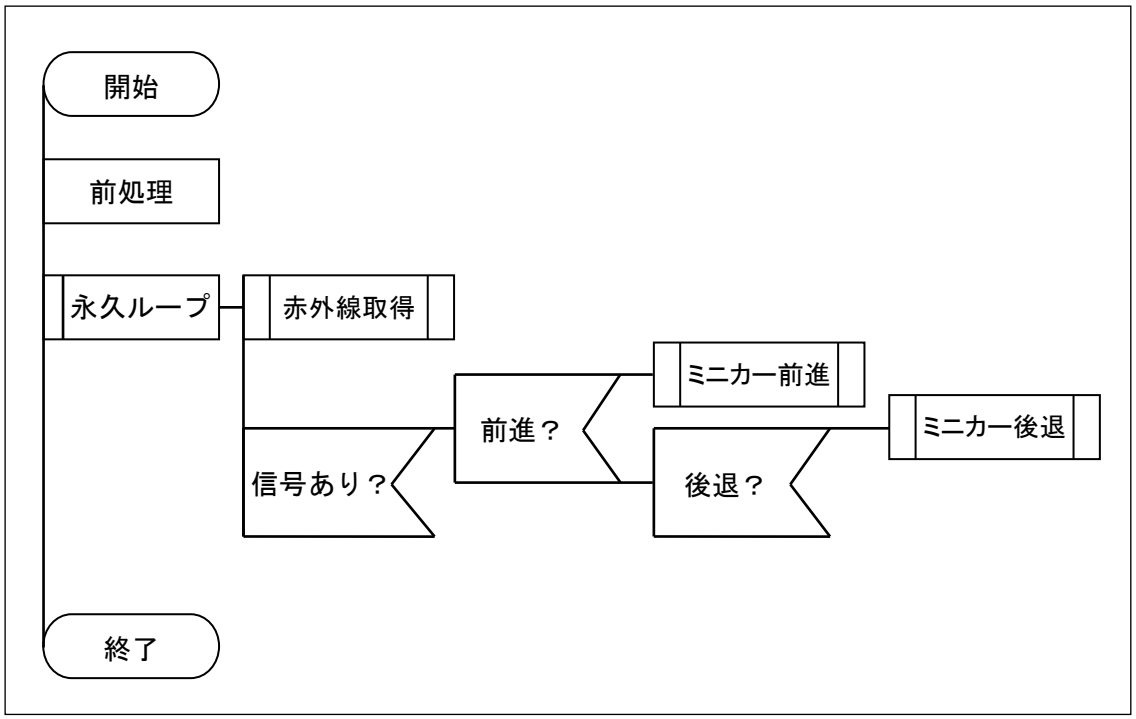


図 5-12　PAD 図（例: iPhone で動くミニカー①）

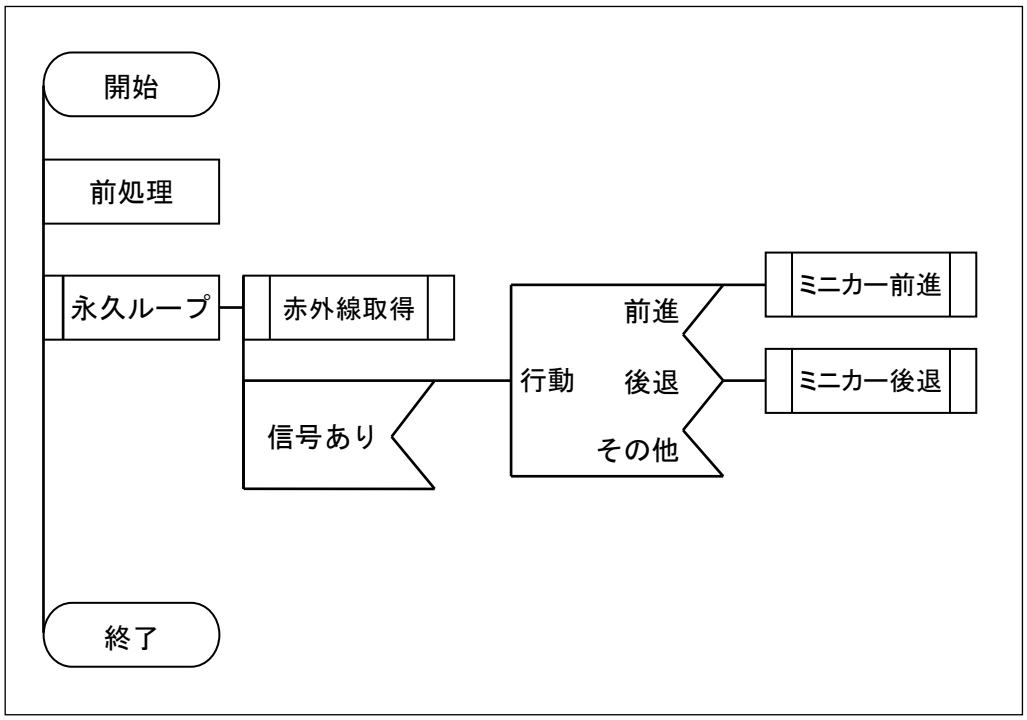


図 5-13　PAD 図（例: iPhone で動くミニカー②）

5.3. 開発の流れ

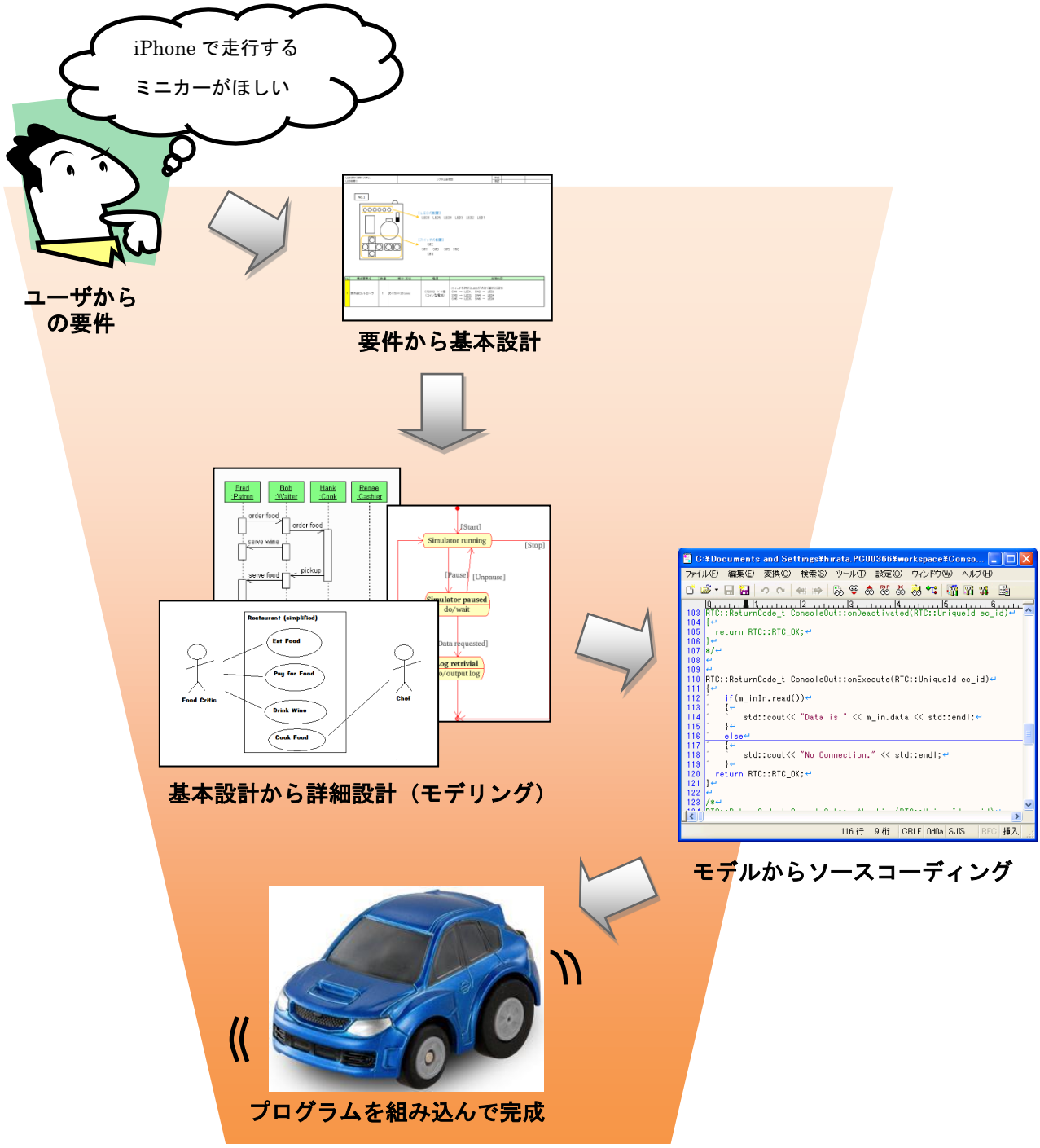


図 5-14　要件から完成までの流れ

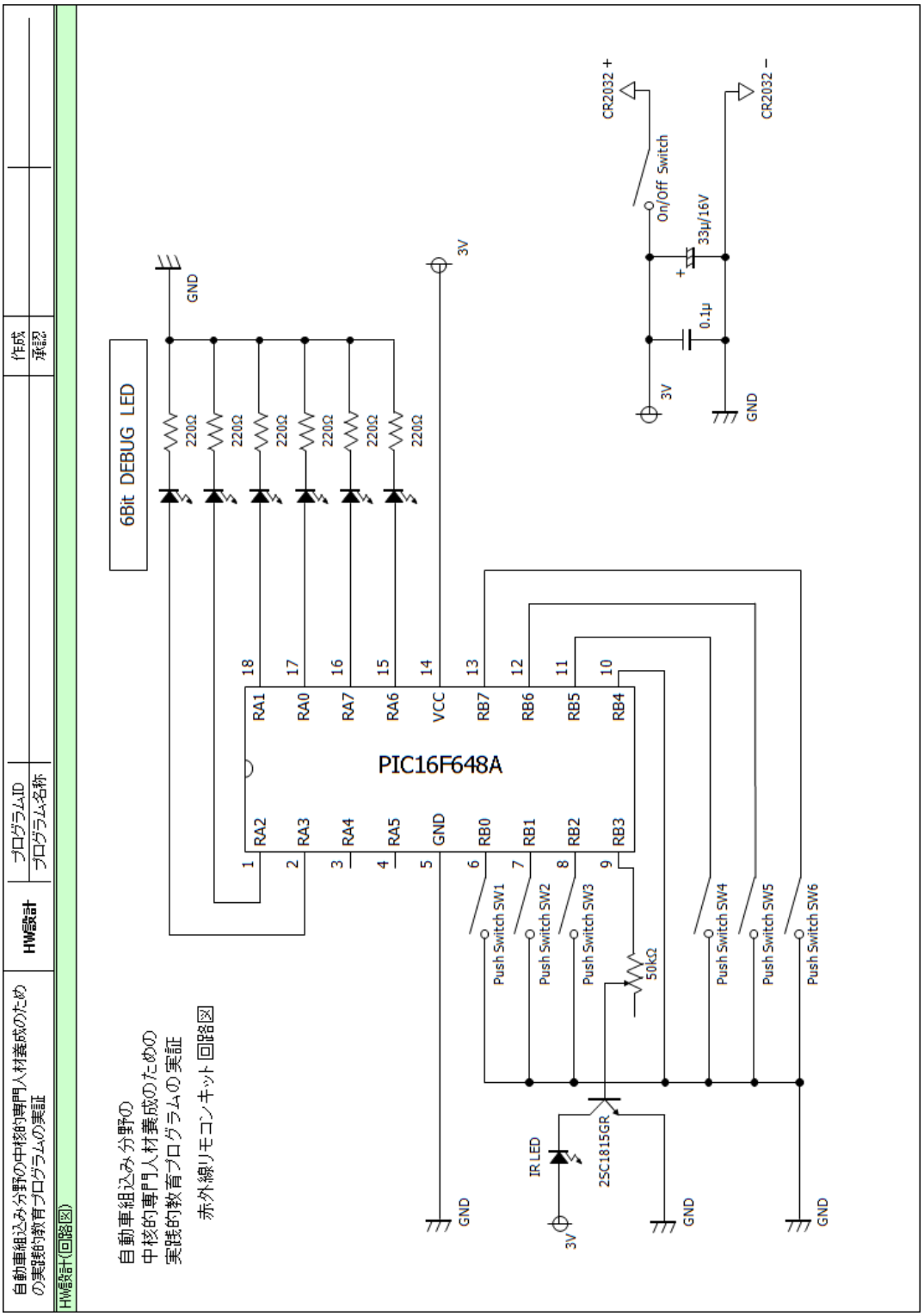
6. 赤外線コントローラ

この章では、部品の配置図・スズメッキ配線図・被服配線図・回路図を参照し、赤外線コントローラについて学習する。このコントローラを使用しミニカーを制御する。

次頁以降の部品表・配置図、配線図、回路図は、CD-ROM ドライブの“教材¥部品表・配置図”フォルダの“赤外線コントローラ_部品表・部品配置図.xls”、“教材¥配線図・回路図”フォルダの“赤外線コントローラ_配線図・回路図.xls”を参照する。

次項は「付録 H.昨年度教材参照」の参照番号 4～6 を参照。

6.1. 回路図


自動車組込み分野の中核的専門人材養成のための実践的教育プログラムの実証
HW設計
プログラムID
プログラム名称
作成
承認
HW設計(回路図)
自動車組込み分野の
中核的専門人材養成のための
実践的教育プログラムの実証
赤外線リモコンキット回路図
6Bit DEBUG LED
220Ω
GND
3V
PIC16F648A
RA1
RA0
RA7
RA6
VCC
RB7
RB6
RB5
RB4
RA2
RA3
RA4
RA5
GND
RB0
RB1
RB2
RB3
Push Switch SW1
Push Switch SW2
Push Switch SW3
Push Switch SW4
Push Switch SW5
Push Switch SW6
50kΩ
IR LED
2SC1815GR
CR2032 +
CR2032 −
On/Off Switch
33μ/16V
0.1μ

7. 設計書の作成

この章では、5 章で述べた基本設計書と詳細設計書について学習する。
設計書を作成する際に、「設計書は何のために存在しているのか」を意識し、設計書の必要性を確認することが必要である。

次項は「付録 H.昨年度教材参照」の参照番号 7～9 を参照。

8. 開発環境構築

この章では、実習で必要な環境構築を行う。
C 言語プログラムのコンパイラと統合開発環境、PIC ライターのセットアップを行う。
既に別の開発環境を持っている場合はそれを使用してもよいが、開発環境が日本語に対応していない場合は注意する必要がある。（詳しくは付録 E.④を参照のこと）

8.1. PICC LITE（コンパイラ）

C 言語プログラムのコンパイラは、HI-TECH 社が提供しているフリーのコンパイラツール PICC LITE を使用する。
フリーツールのため、対応している PIC は 12F627, 12F629, 16F627, 16F648A, 16C84, 16F84, 16F877(A)など制限がある。また、プログラミング領域として使える ROM サイズや使用できる RAM 領域の制限などもある。しかし、コンパイラとしては評価が高いため、通常使用する場合に特に問題はない。

次項は「付録 H.昨年度教材参照」の参照番号 10 を参照。

8.2. HI-TIDE（統合開発環境）

HI-TIDE は HI-TECH 社が配布している IDE(Integrated Development Environment：統合開発環境)で、PICC LITE や HI-TECH C などを利用できる。
IDE はエディタ、コンパイラ、デバッガなど、プログラミングに必要なツールが一つのインターフェースで統合して扱えるようになっている環境のことである。

次項は「付録 H.昨年度教材参照」の参照番号 11 を参照。

8.3. 開発環境の日本語化

この項は「付録 H.昨年度教材参照」の参照番号 12 を参照。

8.4. IC-Prog（PIC ライター）

この項は「付録 H.昨年度教材参照」の参照番号 13 を参照。

8.5. プロジェクト設定

開発環境の構築が完了したら今回の実習で使用する“プロジェクト”というものを設定する。プロジェクトとは、プログラムを作るときに、必要となるソースファイルやライブラリ、ヘッダファイルなどを一括して管理するための集合体である。

次項は「付録 H.昨年度教材参照」の参照番号 14～17 を参照。

9. プログラムの作成

この項は「付録 H.昨年度教材参照」の参照番号 18 を参照。

10. C 言語の基礎

この章では、C 言語の基礎について説明する。

10.1. C 言語の歴史

C 言語は、1972 年に AT&T 社のベル研究所のデニス・**リッチー**(Dennis Ritchie)が主体となって作ったプログラミング言語である。英語圏では単に“C”と呼称されており、日本でも文書や文脈によっては同様に“C”と呼称される。

UNIX の移植性を高めるために開発された経緯から、オペレーティングシステムカーネル向けの低レベルな記述ができることを特徴としている。

10.1.1. C 言語の誕生

C 言語は、DEC 社のミニコンピュータ PDP-11 で動く UNIX の汎用言語として、1972 年に AT&T ベル研究所の**リッチー**が開発した。ベースとなったのは Algol68 という言語で、さらにそのベースは 1950 年代の終わり頃に生まれた Algol という言語である。

1963 年、Algol60 をベースに、研究用の言語仕様として CPL(Combined Programming Language)という言語が考案された。4 年後の 1967 年、マーティン・リチャーズ(Martin Richards)は CPL の“言語的な優美さ”を受け継ぎつつ、コンパイラを作るための言語 BCPL(Basic CPL)を開発した。

1970 年、ケン・トンプソン(Ken Thompson)は BCPL を元にして、DEC 社のミニコンピュータ PDP-7 用の言語“B”を開発した。この“B”に Algol68 の制御構造とデータ型の概念を加えた言語が C 言語である。

“C”という名称については、アルファベットで B の次は C なので C と命名されたとも、B 言語の元となった BCPL の 2 番目の文字を採って命名されたとも言われている。

10.1.2. 規格の標準化

1978 年に出版された、ブライアン・**カーニハン**(Brian Kernighan)と**リッチー**の共著である「The C Programming Language」（邦題『プログラミング言語 C』石田晴久訳 1981 年）は、その後、標準化がなされるまで実質的な C 言語の標準として使用された。この本は著者名の頭文字から“**K&R**”と呼ばれている。

しかし、この本の記述にはいくつか曖昧な部分が存在していた。そのため、C 言語が普及するに従い、互換性のない処理系が数多く誕生することとなった。

そこで、ISO（国際標準化機構）と ANSI（米国国家規格協会）は協同で C 言語の規格の標準化を進め、1989 年 12 月に ANSI が、1990 年 12 月に ISO が、それぞれ C 言語の規格を発行した。ISO 規格の方には章立てが追加されており、その後 ANSI も ISO 規格にならって章立てを追加した。それぞれ“C89”（ANSI C89）及び“ISO C90”という通称で呼ばれることがある。

最大の特徴は、C++と同様の**関数プロトタイプ**を導入して引数の型チェックを強化したことと、“**void**”や“**enum**”などの新しい型を導入したことである。一方、処理系に依存するとするに留めた部分も幾つかある。例えば、int 型のビット幅、char 型の符号、ビットフィールドのエンディアン、シフト演算の挙動、構造体などへのパディング等である。

また、型の大きさは厳密に決められてはおらず、バイト数は sizeof 演算子で取得し、最大最小値は“limits.h”で参照することとされている。多くの処理系では、char 型は 8 ビット、short 型は 16 ビットであるが、int や long は環境によってまちまちである。また、API などの呼び出しには、ヘッダで“BYTE”や“WORD”などといった typedef で定義した型を使用して回避するのが一般的になっている。基本的に、符号を明示しない場合は signed（符号付き）になる。

日本では、“ISO C90”を翻訳したものが日本工業規格『JIS X 3010:1993 プログラム言語 C』として 1993 年 10 月に制定された。

なお、ANSI での標準化に合わせて「The C Programming Language」の第 2 版が 1988 年に出版された。初版と比較すると、ANSI での標準化(C89)を反映したため、内容が大幅に変更されている。（邦題『プログラミング言語 C　第 2 版』石田晴久訳 1989 年）

1995 年には、主として英語圏での利用を想定して制定された C90 に対して、主に国際化のためにワイド文字版ライブラリを追加した Amendment1 が発行された。これは“C95”と呼ばれることもある。日本では 1996 年に発行された『JIS X 3010:1996 プログラム言語 C』で、この規格に対応した追補・訂正が行われている。

1999 年 12 月には、ISO で規格の改定が行われ、C++の機能のいくつかを取り込むことを含めた機能拡張がなされた C 言語(Second Edition)が制定された。この版の C 言語の規格は“C99”という通称で呼ばれる。日本では、2003 年 12 月に改正された日本工業規格『JIS X 3010:2003 プログラム言語 C』がこの規格に対応している。

現在の最新規格は、2011 年 12 月に改定された“ISO/IEC 9899:2011”（通称“C11”）である。C11 は Unicode 文字列（UTF-32、UTF-16、UTF-8 の各符号化方式）に標準で対応している。そのほか、type-generic 式、C++と同様の無名構造体・無名共用体、排他的アクセスによるファイルオープン方法、quick_exit などのいくつかの標準関数などが追加されたが、その一方で gets 関数は廃止された。

また、C99 で規格上必須要件とされていた機能のうち、複素数型と可変長配列が C11 では省略可能なものに変更された。これらの省略可能な機能は C11 規格合致の必須要件ではないため、仮に完全に規格合致と謳っている処理系であっても、サポートしていない可能性がある。C11 規格では、省略可能な機能のうちコンパイラがどれを提供しているかを判別するために利用できる、テスト用のマクロを用意している。

日本では 2003 年の改正以降、プログラミング言語 C に関する日本工業規格が改正されていないため、C11 に対応した規格はまだ発行されていない。

10.1.3. 組み込みプログラムと C 言語

C 言語は組込み系で最も多く使用されているプログラミング言語である。経済産業省が 2012 年度に行った「ソフトウェア産業の実態把握に関する調査」によると、組込み系で使用されるプログラミング言語の約 6 割が C 言語であり、これに C++や C#を加えると、C 言語系の使用比率は 8 割を超える。

では、C 言語がプログラミング言語の主流を占めているのはなぜだろうか？　それは、以下に挙げる C 言語の特徴によるものである。

（ア） 構造化プログラミングに適している

構造化プログラミングとは、プログラムを処理単位ごとに分割（ブロック化あるいはモジュール化）することで、プログラムの可読性や保守性を向上させようとする手法で、C 言語は言語仕様として構造化をサポートしている。

（イ） 豊富なデータ型をサポートしている

よく、大は小を兼ねると言うが、組み込みの世界にこれは当てはまらない。組み込みシステムでは、可能な限り無駄の無いようにプログラミングするため、データを格納する容器（変数）の容量（サイズ）にも気を使う必要がある。

例えば、何かの真偽を判定するために変数を使う場合、その変数の値が 0 か 1 か分かれば、真偽の判定には必要かつ十分である。この場合、1 ビットのデータ型があれば便利である。逆に、経済の動向とか天体の動きなどをシミュレーションするようなプログラムの場合では、大きな桁を扱えるデータ型が必要となる。また、絶対アドレスの参照やテーブルへ高速にアクセスするためには、ポインタの概念が必要である。これらのデータ型を C 言語はサポートしている。

（ウ） I/O 処理が可能

組み込みシステムの構成要素の 1 つに I/O がある。高級言語で I/O へ簡単にアクセスできれば、プログラミングが楽になる。C 言語を使うと、メモリマップド I/O の場合はポインタ、I/O マップド I/O の場合は専用命令でアクセスできる。

（エ） 移植性が高い

高級言語でプログラミングすれば、プラットフォームが変わっても、同じ記述で同じ処理が可能である。C 言語は高級言語に属し、対応しているプラットフォームも多いため、移植性に優れている。

もし、アセンブリ言語でプログラミングしていると、マイコンが変わるたびに命令セットを覚え直さなければならず、移植のためには多大な工数が必要となる。しかし、C 言語を使えば、マイコンのアーキテクチャの違いはコンパイラが吸収してくれるため、移植時の負担は大幅に軽減される。

（オ） 開発効率が高い

一般的に新規にプログラムをコーディングする場合、プログラマが書けるステップ数は 1 日当たり 1000 行程度と言われている。C 言語の 1 行は、アセンブリ言語の数十行に匹敵するので、単純計算で考えると、C 言語はアセンブリ言語と比較して数倍のスピードでコーディングができることになる。

また、人間はどんなに細心の注意を払っていても、必ずどこかでミスをしてしまうものである。複雑なアセンブリ言語の場合、ほんの些細なミスでもプログラムは動作しない。細かい部分はコンパイラに任せ、ステップ数をできるだけ減らすことによって不具合の発生率を下げることも、C 言語を使うメリットである

（カ） 学習、習得のしやすさ

C 言語は簡単な英語表記で記述する。C 言語で使われる英語は限られているので、英語が苦手でも比較的なじみやすい。また、本屋に行けば C 言語に関する書籍が簡単に入手できるし、インターネット上にも C 言語に関する情報は溢れている。このようなことから、C 言語は独学で習得するのに最適な開発言語と言えるだろう。

（キ） 現実的問題

プログラミング言語には C 言語以外にも多くの優れた言語がある。しかし、それらの言語を使うために、特別な開発環境が必要であるとか、入手するために多大な出費を強いられる場合がある。これに対して、C 言語の場合は適当なスペックを満たすパソコンがあれば動作する。また、“GNU ツール”に代表される、フリーの開発環境もある。

これらを総合的に判断すると、C 言語は、ハードウェアに近い部分からアプリケーションに近い部分までオールラウンドに使え、かつ、CPU の種類にほとんど依存しない言語だからこそ普及したと言えるだろう。

10.2. 変数

この項は「付録 H.昨年度教材参照」の参照番号 19 を参照。

10.3. 計算式

この項は「付録 H.昨年度教材参照」の参照番号 20 を参照。

10.4. for 文（繰り返しと条件分岐）

C 言語では同じ処理を何度も繰り返す場合に、“for”文を使用する。
“for”は英語で、「…の間、ずっと」と言う意味である。“for”文は分岐条件を満たしている間は処理を繰り返す。

なお、初期化、条件、増分を全て省略（“for(;;)”）することで無限ループにすることもできる。

この項は「付録 H.昨年度教材参照」の参照番号 21～22 を参照。

10.5. while 文（繰り返し）

C 言語では同じ処理を何度も繰り返す場合に、“while”を使用する。
“while”文は条件式を評価し、真の間は処理を繰り返す。
“for”文とは異なり、条件式で使われる変数の値をどのように変化させるかは指定できない為、実行される処理の中で別途記述する必要がある。条件値を変化させなければ無限ループとなるので注意が必要である。

逆に条件式を「1」にすることで強制的に無限ループにすることもできる。

この項は「付録 H.昨年度教材参照」の参照番号 23～24 を参照。

10.6. if 文・else 文

この項は「付録 H.昨年度教材参照」の参照番号 25 を参照。

10.7. C 言語の配列

この項は「付録 H.昨年度教材参照」の参照番号 26 を参照。

10.8. C 言語の関数

この項は「付録 H.昨年度教材参照」の参照番号 27 を参照。

10.9. コーディング作法

C 言語は記述の自由度が高く、何もルールを設けずにプログラミングした場合、技術者の技量や経験の差によってソースコードの出来不出来に大きな差が生じてしまう。こうした事態を防ぐため、組織あるいはプロジェクト単位で守るべきコーディング規約を定め、ソースコードの標準化を進めているケースが多い。

通常、コーディング規約とは『品質を保つために守るべきコードの書き方（ルール）』を整理したものであるが、これには下記のような問題点が存在する。

(1) ルールの必要性が理解されていない。または、ルール違反に対する正しい対処方法が理解されていない。
(2) ルールが多すぎて覚えきれない。あるいは、ルールが少なくてカバー範囲が不足している。
(3) ルールの遵守状況を確認するための高精度なツールが無く、確認を技術者が目視で行うレビューに頼っており、負担が大きい。

また、この結果として、既にコーディング規約がある組織や部門においても、それらが形骸化して守られていないというケースもある。

ここで紹介する「コーディング作法」とは、様々なコーディングのシーンで守るべき基本的な考え方を、「ソフトウェアの品質」という視点を考慮して「作法」としてまとめたものである。

10.9.1. 組込みソフトウェア開発の現状

経済産業省が 2012 年度に行った「ソフトウェア産業の実態把握に関する調査」によると、組込み製品出荷後に発生した不具合の原因のうち、ソフトウェア設計によるものが 30.4%、ソフトウェア実装やデバッグによるものが 30.0%であり、実に 60.4%がソフトウェアの不具合によるものだったという調査結果であった。

要因としてはテスト工程での工数不足やテスト項目漏れ等もあるが、そもそも設計の段階で不具合の潜在化を防ぐように先手を打つことができていれば、ソフトウェアの品質を保つことは可能である。

しかし、設計工程で不具合の潜在化を防ぐためのノウハウ（定石）は、各開発の場面での文書化が行われておらず、伝承の域で閉じてしまっているものが多い。

このような事から、コードの品質を十分に保つことができていないのが現状である。

10.9.2. コード品質向上へのアプローチ

コードの品質向上を阻害するものとして、以下の要因が挙げられる。

（ア） コーディングを個人の技量に任せている。

（イ） 第三者のコードレビューが充分に行われない。（形式的なところに集中）

（ウ） 単体テスト工程での品質確保に頼っている。（テスト重視）

（エ） 流用開発が多く、新規開発部分と合わせたコード品質の確保が難しい。

これらは、いずれもコード品質を作り込む課程が効率的よく機能していないことを示している。

コードの品質問題には、大きく分けて 2 つの要因がある。

⑴ コーディングでの問題

・言語知識の不足
・詳細設計書の理解不足
・タイプミス
・インターフェースやアーキテクチャ理解不足

⑵ ソースレビュー／修正の問題

・他人が読みにくい／誤解し易い
・追加／変更／削除等の履歴が無い
・動作しない／使用しない行がある
・コメントが無い／コメント誤り
・確認／テストがしづらい

このような問題を最小限に抑えるためには、適切な時期にコードレビューを行う必要があるが、実装時の現状として、『適切な時期に、的確なコードレビュー』というものは行われていないことが多い。（実装時にレビューしないで、テストで賄っている。）

この問題を解決し、適切・的確なレビューを実施してコード品質を向上させるには、コーディング規約に沿って実装を行うことが 1 つの解決策である。

しかし、コーディング規約の実態はどうかというと、コーディング規約はあるが運用が形骸化していたり、ルールに問題があったりして、必ずしも遵守されているとは限らない。また、未だにコーディング規約を設けていない企業も存在する。

10.9.3. コーディング作法の特徴

コードの品質は、ソフトウェアの品質と同様に「信頼性」「保守性」「移植性」「効率性」といった品質特性で分類することができる。そこで、ルールの基本概念を作法としてまとめ、『JIS X 0129-1　ソフトウェア製品の品質』に準拠した品質概念を基に、ソースコードの品質を保つための慣習や実装の考え方で、個々のルールの基本的な概念を示す。

このことは、意義や必要性を理解することができる実用的な「コーディング規約」の作成に役立つものである。

10.9.4. 信頼性

ソースコードレベルで、ソフトウェアの信頼性について気をつけるべきこととしては、誤動作を引き起こすような記述を極力避けるといった工夫が求められる。

(1) 領域は初期化し、大きさに気を付けて使用する。

① 領域は、初期化してから使用する。
・初期化されていない変数の値は、何が入っているかわからないので不定と考えるべきである。

② 初期化は過不足が無いことがわかるように記述する。
・特に文字列の場合は実体の後ろにある NULL 文字を忘れがちなので、十分に注意する必要がある。

③ ポインタの指す範囲に気を付ける。
・ポインタに対する演算は、ポインタの指している先をわかりにくくする原因となる。すなわち、確保していない領域を参照したり、領域に書き込んだりするバグを埋め込む可能性が高くなるため、注意する必要がある。

⑵ データは範囲、大きさ、内部表現に気を付けて使用する。

① 内部表現に依存しない比較を行う。
・特に浮動小数点型は、ソースコード上に書かれた値と実装された値は完全に一致していないので、比較は許容誤差を考慮して判定する必要がある。

② 真の値と等しいかどうかを調べてはならない。
・C 言語では、真は『0 ではない値』で示され、1 とは限らない。

③ データ型を揃えた演算や比較を行う。
・C 言語の符号無し整数演算は、オーバーフローせずに表現可能な最大数の剰余となる。このため、演算結果が意図と異なっていることに気付かない場合がある。例えば、同じ定数式でも int のビット数が異なる環境では、演算結果がその型で表現できる範囲を超えた場合と超えない場合で結果が異なる。

④ 演算精度を考慮して記述する。
・演算の型は演算に使用する式（オペランド）の型によって決まり、代入先の型は考慮されない。演算の型と代入先の型が異なる場合、誤って代入先の型での演算を期待していることがある。オペランドの型とは異なる型の演算を行いたい場合は、期待する型にキャストしてから演算する必要がある。

⑤ 情報損失の危険のある演算は使用しない。
・値をその型と異なる型の変数に代入すると、値が変わる（情報損失する）可能性がある。情報損失を起こす可能性のあるデータ型への代入や演算を行う場合は、問題が無いことを確認し、そのことを明示するためにキャストを記述する。ただし、機械的にキャストするのでなく、使用するデータ型を変更した方がよい場合が多いので、まず、データ型の変更を検討する。

⑥ 対象データが表現可能な型を使用する。
・ビットフィールドに使用する型は signed int と unsigned int だけとし、1 ビット幅のビットフィールドが必要な場合は signed int 型でなく、unsigned int 型を使用する。コンパイラがサポートしていても、char、short、long 等の型をビットフィールドに使用することは C 言語規格外であるので、移植性を考慮する場合は使用しない。

⑦　ポインタの型に気を付ける。

・ポインタで指し示された型から const 修飾や volatile 修飾を取り除くキャストを行ってはならない。const 修飾や volatile 修飾された領域は、参照しかされない領域であったり、最適化をしてはならなかったりする領域なので、その領域に対するアクセスに注意しなければならない。これらの領域を指すポインタに対し、const や volatile を取り除くキャストを行ってしまうと、せっかく記述した注意項目が見えなくなり、コンパイラは、プログラムの誤った記述に対し、何もチェックできなくなる。

⑧　宣言、使用、定義に矛盾がないことをコンパイラがチェックできる書き方にする。

・引数を持たない関数は、引数の型を void として宣言する。

・関数呼出し、および関数定義の前にプロトタイプ宣言を行う。さらに、同じ宣言が関数呼出しと定義で参照されるようにする。

(3)　動作が保証された書き方にする。

①　領域の大きさを意識した書き方にする。

・配列の大きさを省略して extern 宣言しても、エラーにはならない。
しかし、大きさが省略されていると、配列の範囲外のチェックに支障が生じる場合がある。このため、配列の大きさは明示して宣言した方がよい。
ただし、初期値の個数で配列の大きさを決定し、宣言時に大きさが決まらない場合などは、宣言時の配列の大きさを省略した方がよい場合もある。

②　実行時にエラーになる可能性のある演算に対しては、エラーケースを迂回させる。

・除算や剰余算の右辺式は、0 でないことを確認してから演算を行う。そうしない場合、実行時に 0 除算のエラーが発生する可能性がある。

③　関数呼出しではインターフェースの制約をチェックする。

・関数がエラー情報を戻す場合、エラー情報をテストしなければならない。

・関数に渡す引数に制限がある場合、関数呼び出しする前に、制限値でないことを確認してから関数呼び出しする。

④　再帰呼出しは行わない。

・再起呼出しは実行時の利用スタックサイズが予測できないため、スタックオーバーフローの危険がある。

⑤　分岐の条件に気を付け、所定の条件以外が発生した場合の処理を記述する。

・ループカウンタの比較に等式、不等式は使用しない。

（「<=、>=、<、>」を使用する）

ループカウンタの変化量が 1 でない場合、無限ループになる可能性があるので、ループ回数を判定する比較では、等式（==）、不等式（!=）は使用しない。

⑥　評価順序に気を付ける。

・複数の引数を持つ関数の各実引数の実行（評価）の順序は、コンパイラは保証していない。引数は右から実行されたり、左から実行されたりする。また、＋演算のような 2 項演算の左式と右式の実行の順序も、コンパイラは保証していない。このため、引数並びや 2 項演算式内で、副作用をもつ関数呼出しや volatile 変数を複数記述すると、その実行結果が保証されない場合がある。このような危険な記述は避けるべきである。

10.9.5. 保守性

リリースしたソフトウェアの一部に不具合などが見つかり修正をする場合や、製品に対する市場からの要求などに応じて既存ソフトウェアをベースに新たな機能を追加する場合など、作成したソフトウェアに何らかの手を加える場合、その作業をできるだけ誤りなく効率的に行えるかどうかが重要な特質になる。

(1) 他人が読むことを意識する。

① 使用しない記述を残さない。

・使用しない関数、変数、引数、ラベルなどの宣言（定義）は、削除し忘れたのか、記述を誤っているかの判断が難しいため、保守性を損なう。

② 紛らわしい書き方をしない。

・適切な型を示す接尾語が使用できる定数記述には、接尾語をつけて記述する。

・long 型整数定数を示す接尾語は大文字の“L” のみ使用する。
（小文字の“l”は数字の“1”と混同するため。）

③ 特殊な書き方はしない。

・switch(式)の式には、真偽結果を求める式を記述しない。真偽結果を求める式を switch 文に使用すると、分岐数は 2 つになり、多分岐命令である switch 文を使用する意味が無い。switch 文は、default 節の誤記や、break 文の記述漏れなど、if 文と比較して間違いが発生する可能性が高いので、3 分岐以上にならない場合は、if 文を使用するべきである。

④ 演算の優先順位がわかりやすいように記述する。

・&&や||演算の右式と左式は単純な変数か( )で囲まれた式を記述する。ただし、&&演算が連続して結合している場合や、||演算が連続して結合している場合は、&&式や||式を( )で囲む必要はない。

⑤　関数のアドレス取得の演算や比較演算を省略しない。

・C 言語では、関数名を単独で記述すると関数呼び出しではなく、関数アドレス取得となる。すなわち、関数アドレスの取得に“&”を付ける必要はない。しかしながら、&を付けない場合、関数呼び出しと勘違いする可能性がある。関数のアドレスを求める場合に&を付ける規則を守ることで、&が付かず、( )も続かない関数名の出現をチェックでき、勘違い（ミス）を見つけられる。

・条件判定では、式の結果が 0 の場合は偽、0 以外は真と判断される。このため、条件を判定する式では“!= 0”を省略可能である。ただし、プログラムを明示的にするためには、省略しない方がよい。

⑥　領域は 1 つの利用目的に使用する。

・変数の再利用は可読性を損ない、修正時に正しく修正されない危険性が増すので行わない方がよい。

⑦　名前を再使用しない。

・名前は、自動変数など有効範囲が限られている場合を除き、できる限りプログラムで一意とすることで、プログラムを読みやすくすることができる。

・標準ライブラリで定義されている関数名や変数名やマクロ名を独自に定義すると、プログラムの可読性を低下させる。従って、標準ライブラリの関数名、変数名およびマクロ名は再定義・再利用してはならない。また、定義を解除してはならない。

⑧　勘違いしやすい言語仕様を使用しない。

・論理演算子 && または || の右側のオペランドには、副作用があってはならない。&&や||演算子の右式は、左式の条件結果により、実行されない場合がある。インクリメントなどの副作用のある式を右式に記述すると、左式の条件によりインクリメントされる場合とされない場合が生じ、分かり難くなるため、&&や||演算子の右式には副作用のある式を記述しないようにする。

⑨　特殊な書き方は意図を明示する。

・意図的に何もしない文を記述しなければいけない場合はコメント、空になるマクロなどを利用し、目立たせる。

・無限ループの書き方は、以下のような書き方に統一する。

例 1）for(;;) で統一する。

例 2）while(1) で統一する。

例 3）マクロ化した無限ループを使用する。

⑩　マジックナンバーを埋め込まない。

・意味のある定数は#define でマクロとして定義して使用する。マクロ化することにより、定数の意味を明確に示すことができ、定数が複数箇所で使われているプログラムの変更時も 1 つのマクロを変更すれば済むため、変更ミスを防ぐことができる。

⑪　領域の属性は明示する。

・参照しかしない領域は const であることを示す宣言を行う。

・他の実行単位により更新される可能性のある領域は volatile であることを示す宣言を行う。

⑫　コンパイルされない文でも正しい記述を行う。

・プリプロセッサが削除する部分でも正しい記述を行う。

(2)　修正し間違えないような書き方にする。

①　構造化されたデータやブロックは、まとまりを明確化する。

・配列や構造体を 0 以外で初期化する場合は、構造を示し、それに合わせるために波括弧 '{}' を使用しなければならない。また、すべて 0 以外の場合を除き、初期化データは漏れなく記述する。

②　アクセス範囲や関連するデータは局所化する。

・同一ファイル内で定義された複数の関数からアクセスされる変数は、ファイルスコープで static 変数宣言する。

・同じファイルで定義した関数からのみ呼ばれる関数は、static 関数とする。

・関連する定数を定義するときは、#define より enum を使用する。

(3) プログラムはシンプルに書く。

① 構造化プログラミングを行う。

・C 言語の switch 文における break 忘れは、コーディングミスしやすい代表例の 1 つであり、break 無しの case 文を使用することは避けるべきである。break 無しで次の case に処理を継続させる場合は、break 文がなくても問題の無いことを明示するコメントを、必ず入れるようにする。どのようなコメントを入れるかは、コーディング規約で定める。

② 1 つの文で 1 つの副作用とする。

・1 つの文に代入を複数記述しない。ただし、同じ値を複数の変数に代入する場合を除く。

③ 目的の違う式は、分離して記述する。

・for ループの中で繰返しカウンタとして用いる数値変数は、ループの本体内で変更してはならない。

④ 複雑なポインタ演算は使用しない。

・3 段階以上のポインタ指定は使用しない。3 段階以上のポインタの値の変化を理解することは難しいため、保守性を損なう。

(4) 統一した書き方にする。

① コーディングスタイルを統一する。

・波括弧“{}”や字下げ、空白の入れ方などのスタイルに関する規約を規定する。スタイル規約の決定において、最も重要なことは「決定して統一する」ことであり、「どのようなスタイルに決定するか」ではないことに注意する。決めるべき項目は以下の通り。

a) 波括弧（{}）の位置
b) 字下げ（インデンテーション）
c) 空白の入れ方
d) 継続行における改行の位置（演算子が行末か行頭かのいずれかで統一）

② コメントの書き方を統一する。

・ファイルヘッダコメント、関数ヘッダコメント、行末コメント、ブロックコメント、コピーライトなどの書き方に関する規約を規定する。

③　名前の付け方を統一する。

・プログラムの読みやすさは、名前の付け方に大きく左右される。名前の付け方にも、様々な方式があるが、重要なことは統一性であり、分かりやすさである。

名前の付け方では、次のような項目を規定する。

a）名前全体に対する指針

b）ファイル名（フォルダ名やディレクトリ名を含む）の付け方

c）グローバルな名前とローカルな名前の付け方

d）マクロ名の付け方、など

④　ファイル内の記述内容と記述順序を統一する。

・複数の箇所に記述すると変更ミスの危険があるため、共通に使用するものはヘッダファイルに記述する。

・ソースファイルには、変数および関数の定義と、個々のソースファイルでのみ使用するマクロ、タグ、型（typedef 型）の定義や宣言を記述する。

・ヘッダファイルは重複取り込みに耐えうる作りとする。

⑤　宣言の書き方を統一する。

・関数プロトタイプ宣言では引数の名前を省略できるが、適切な引数名の記述は、関数インターフェース情報として価値がある。引数名を記述する場合、定義と同じ名前を利用し、無用な混乱を避けるべきである。型についても、関数定義やリテラルと同じにした方が理解が容易であるため、同じにすることが推奨される。

⑥　ナル（ヌル）ポインタの書き方を統一する。

・“NULL”はナルポインタとして従来使用されてきた表現だが、実行環境によりナルポインタの表現は異なる。このため、0 を使うほうが安全だと考える人もいる。

⑦　前処理指令の書き方を統一する。

・演算子を含むマクロは、マクロ本体とマクロ引数を括弧で囲む。マクロ本体、マクロ引数を括弧で囲っていない場合、マクロ展開後にマクロに隣接する演算子と、マクロ内の演算子の優先順位によって、期待する演算順序にならず、バグになる可能性がある。

・#ifdef、#ifndef、#if に対応する#else や#endif は同一ファイル内に記述し、プロジェクトで規定したコメントを入れ、対応関係を明確にする。#ifdef などマクロによる処理の切り分けにおいて、#else や#endif が離れた場所に記述されたりネストすると、対応が分かりにくくなる。#ifdef などと対応する#else、#endif などにコメントを付け、対応を分かりやすくする。

(5)　試験しやすい書き方にする。

①　問題発生時の原因を調査しやすい書き方にする。

・デバッグオプション設定時のコーディング方法と、リリースモジュールにログを残すためのコーディング方法を規定する。

②　動的なメモリ割り当ての使用に気を付ける。

・動的メモリを使用すると、不当なメモリをアクセスしたり、メモリをシステムに返却し忘れたりすることにより、システム資源がなくなるメモリリークの危険がある。動的メモリを使用する場合は、使用するメモリ量の上限、メモリ不足の場合の処理、およびデバッグ方法などを規定する

10.9.6. 移植性

組込みソフトウェアの特徴の 1 つは、それが動作するプラットフォームの選択肢が多様である点が挙げられる。即ち、ハードウェア・プラットフォームとしてのマイコンの選択やソフトウェア･プラットフォームである OS の選択など様々な組み合わせが考えられる。

そして、組込みソフトウェアで実現する機能の増大とともに、1 つのソフトウェアを様々なプラットフォームに対応させる形で、既存のソフトウェアを別のプラットフォームに移植する機会が増えてきている。

こうした中で、ソフトウェアの移植性は、ソースコードレベルでも極めて重要な要素である。特に、利用するコンパイラなどに依存する書き方などは日常的にも犯しやすい誤りの 1 つである。

(1) コンパイラに依存しない書き方にする。

① 拡張機能や処理系定義の機能は使用しない。

② 言語規格で定義されている文字や拡張表記のみを使用する。

③ データ型の表現、動作仕様の拡張機能、および処理系依存部分を確認し、文書化する。

・ビットフィールドは、次の動作がコンパイラによって異なる。

a) 符号指定のない int 型のビットフィールドが符号付きと扱われるかどうか。

b) 単位内のビットフィールドの割り付け順序

c) ビットフィールドを記憶域単位の境界

例えば I/O ポートへのアクセスのように、ビット位置が意味をもつデータのアクセスに使用すると、(b),(c)の点から移植性に問題がある。そのため、そのような場合にはビットフィールドは利用せず、& や | などのビット単位の演算を使用する。

④ ソースファイル取り込みについて、処理系依存部分を確認し、依存しない書き方にする。

⑤　コンパイル環境に依存しない書き方にする。

・#include のファイル指定では、絶対パスは記述しない。絶対パスで記述すると、ディレクトリを変更してコンパイルするときに修正が必要となる。

(2)　移植性に問題のあるコードは局所化する。

10.9.7.　効率性

組込みソフトウェアでは、メモリなどの資源効率性や、時間性能、リアルタイム性の要求を考慮した時間効率性に注意しながらコーディングする必要がある。

(1)　資源や時間の効率を考慮した書き方にする。

・繰り返し処理内で、変化のない処理を行わない。

・関数の引数として構造体ではなく構造体ポインタを使用する。
関数の引数として構造体を渡すと、関数呼出し時に構造体のデータをすべて実引数のための領域にコピーする処理が行われ、構造体のサイズが大きいと、速度性能を劣化させる原因となる。参照しかしない構造体を渡す場合は、単に構造体ポインタにするだけでなく、const 修飾を行う。

10.9.8.　コーディングミスの回避

組込みソフトウェアにありがちなコーディングミスには、以下のようなものがある。

(1)　意味のない式や文

・実行されない文を記述

・実行されない式を記述

・実行結果が使用されない文を記述

・実行結果が使用されない式を記述

・実引数で渡した値が使用されない

⑵ 誤った式や文

・誤った範囲指定
・範囲外の比較
・文字列の比較は==演算では行えない
・関数の型と return 文の不整合

⑶ 誤ったメモリの使用

・配列の範囲外の参照・更新
・自動変数の領域のアドレスを呼出し元に渡してしまう誤り
・動的メモリ解放後のメモリ参照
・文字列リテラルを書き込む誤り
・複写サイズの指定誤り

⑷ 論理演算の勘違いによる誤り

・論理和とするところを論理積とした誤り
・論理積とするところを論理和とした誤り
・論理演算とするところをビット演算とした誤り

⑸ タイプミスによる誤り

・“==”演算子を記述するべきところに“=”演算子を記述

⑹ コンパイラによってはエラーにならないケースがある記述

・同名マクロの多重定義
・const 領域に書き込む誤り

コーディング段階でこれらのミスを回避することにより、後工程作業の工数を削減することが期待できる。

11. 赤外線

この項は「付録 H.昨年度教材参照」の参照番号 28 を参照。(差し替え資料有り)

差し替え資料「図 9-1　赤外線①」と差し替え

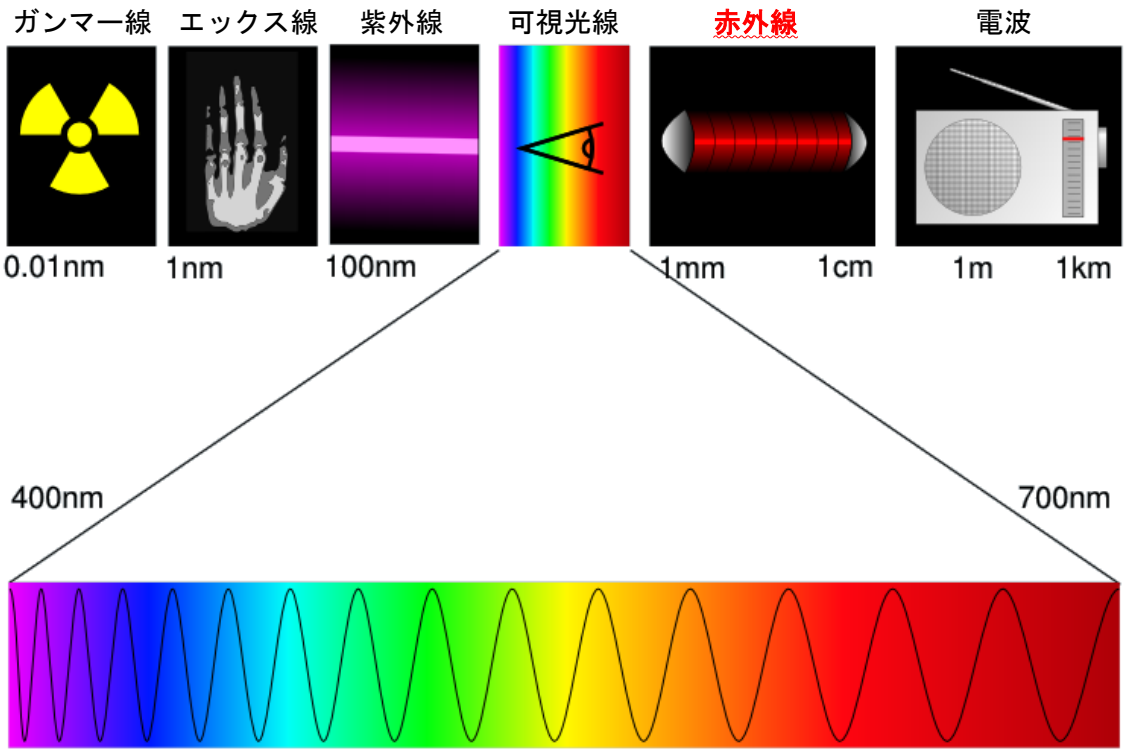


図 11-1　赤外線

12. 実践プログラム（LED 編）

この項は「付録 H.昨年度教材参照」の参照番号 29 を参照。

13. 自動車組込み製品の品質管理

この項は「付録 H.昨年度教材参照」の参照番号 30 を参照。

14. 実践プログラム（ミニカー編）

この項は「付録 H.昨年度教材参照」の参照番号 31 を参照。

14.1. ミニカー操作Ⅶ

ミニカーに装備されている赤外線送信機能を制御し、他のミニカーにアクションを起こさせる。
（この機能を確認するためには、異なる 2 つのバンドのミニカーを用意する必要がある。）

※ マリオカートのアイテム使用に相当するアクションを発生させる機能を使用する。
自動車に本来求められる「安全性」とはかけ離れた機能であるが、今回は制御の学習のためにこの機能を使用する。

14.1.1. 仕様

・「12.4.ミニカー操作Ⅴ」のプログラムに機能追加する。
・SW6 を押しながら同時に SW2,SW4,SW5 のいずれかを押した場合に、アイテム使用アクションを発生させる。
スイッチの組み合わせとアイテムの名称、アクションは以下の通り。

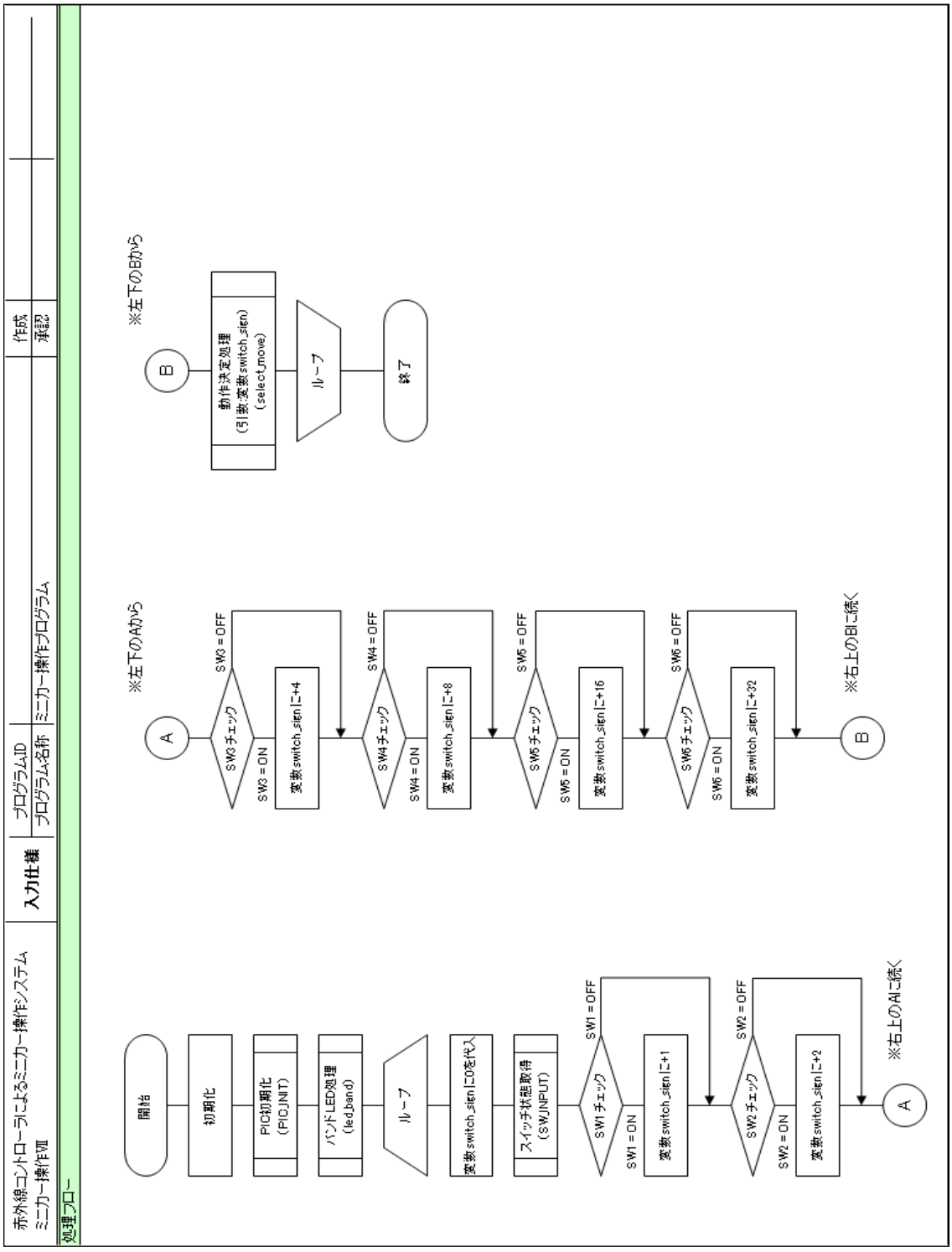
スイッチの組み合わせ	アイテム名称	アクション
SW6 ＋ SW2	ミドリこうら	前方に赤外線を発射し、当たったミニカーは前後に振動する。
SW6 ＋ SW4	バナナ	後方に赤外線を発射し、当たったミニカーは後方にはじかれる。
SW6 ＋ SW5	スーパースター	前後に赤外線を発射し、一定時間無敵状態となる。

・アイテムを使用する場合の伝送仕様は、
① 開始合図を送信
② バンドを表すデータを 1 ビット送信
③ ミニカーを移動させるデータを 4 ビット送信
④ **アイテムを表すデータを 3 ビット送信**
であるが、今回は簡単にするため、③の移動データは“停止”(0b1111)とする。
※ 送信データについては「付録 A. リモコン通信仕様」を確認する。

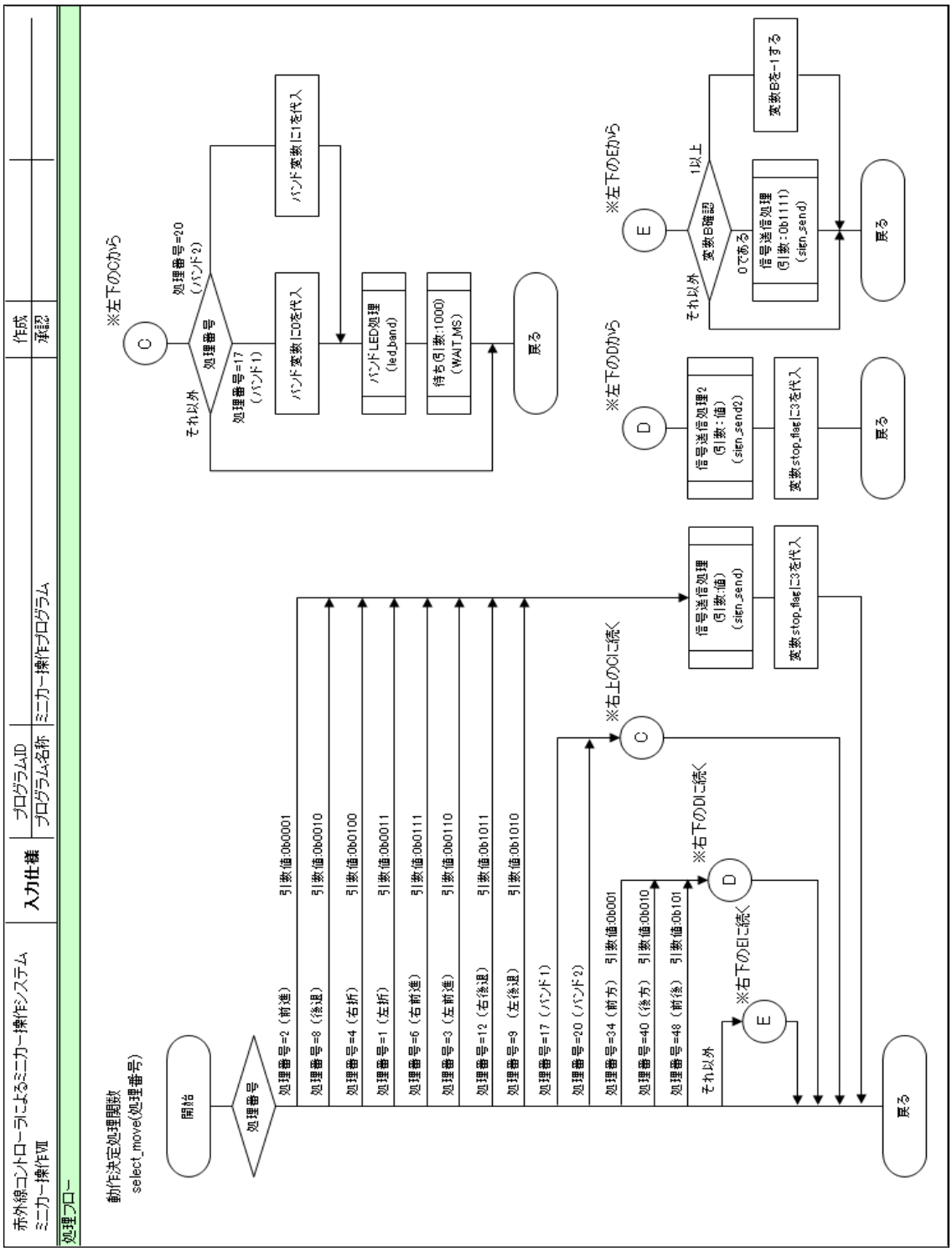
[作業]
・詳細説明書のフローチャート図を作成し、プログラムを作成する。
・プログラム作成後に動作確認を行う。

14.1.2. フローチャート図の確認

-- ミニカー操作Ⅶ 処理フロー（メイン） --

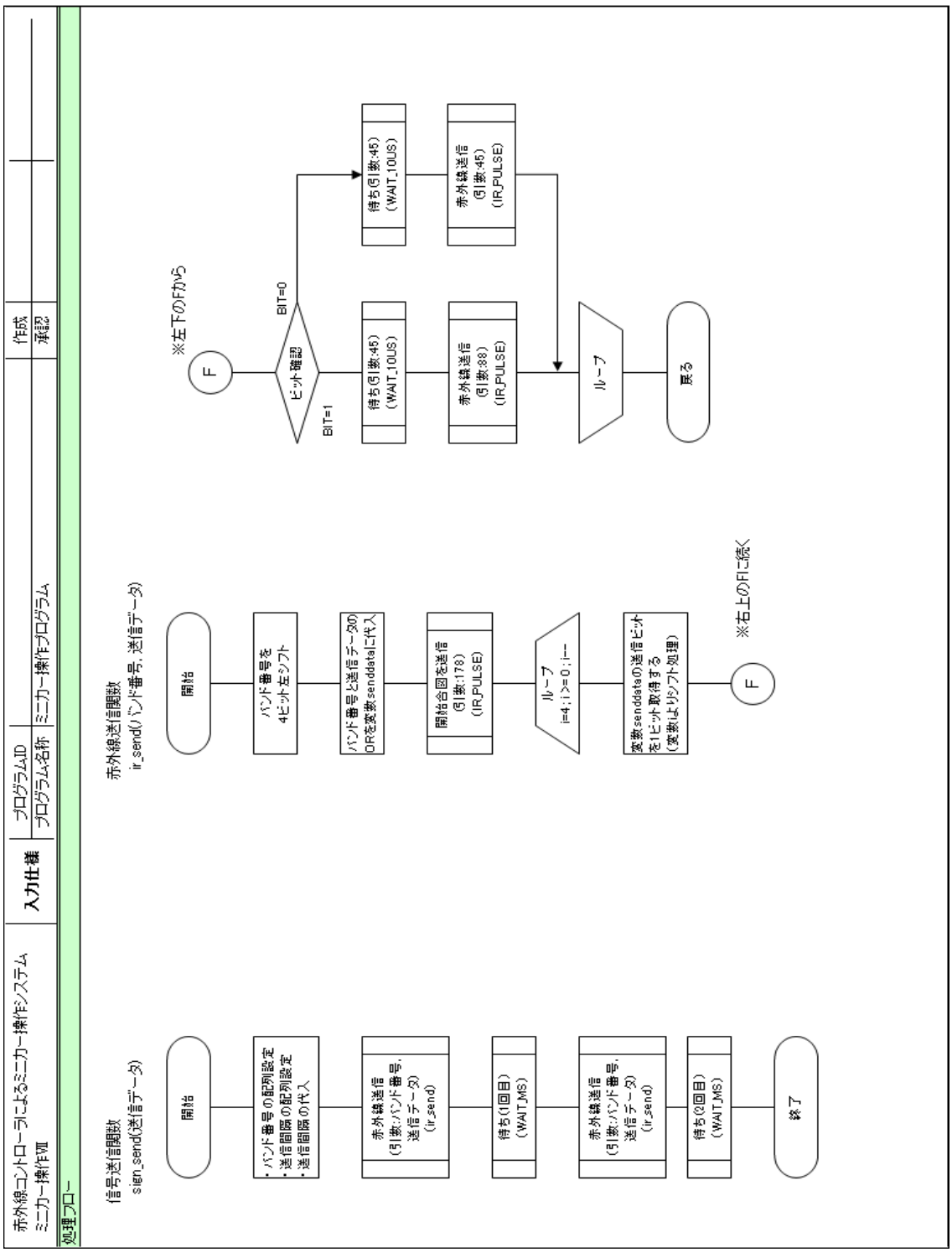


-- ミニカー操作Ⅶ　処理フロー（動作決定処理関数） --

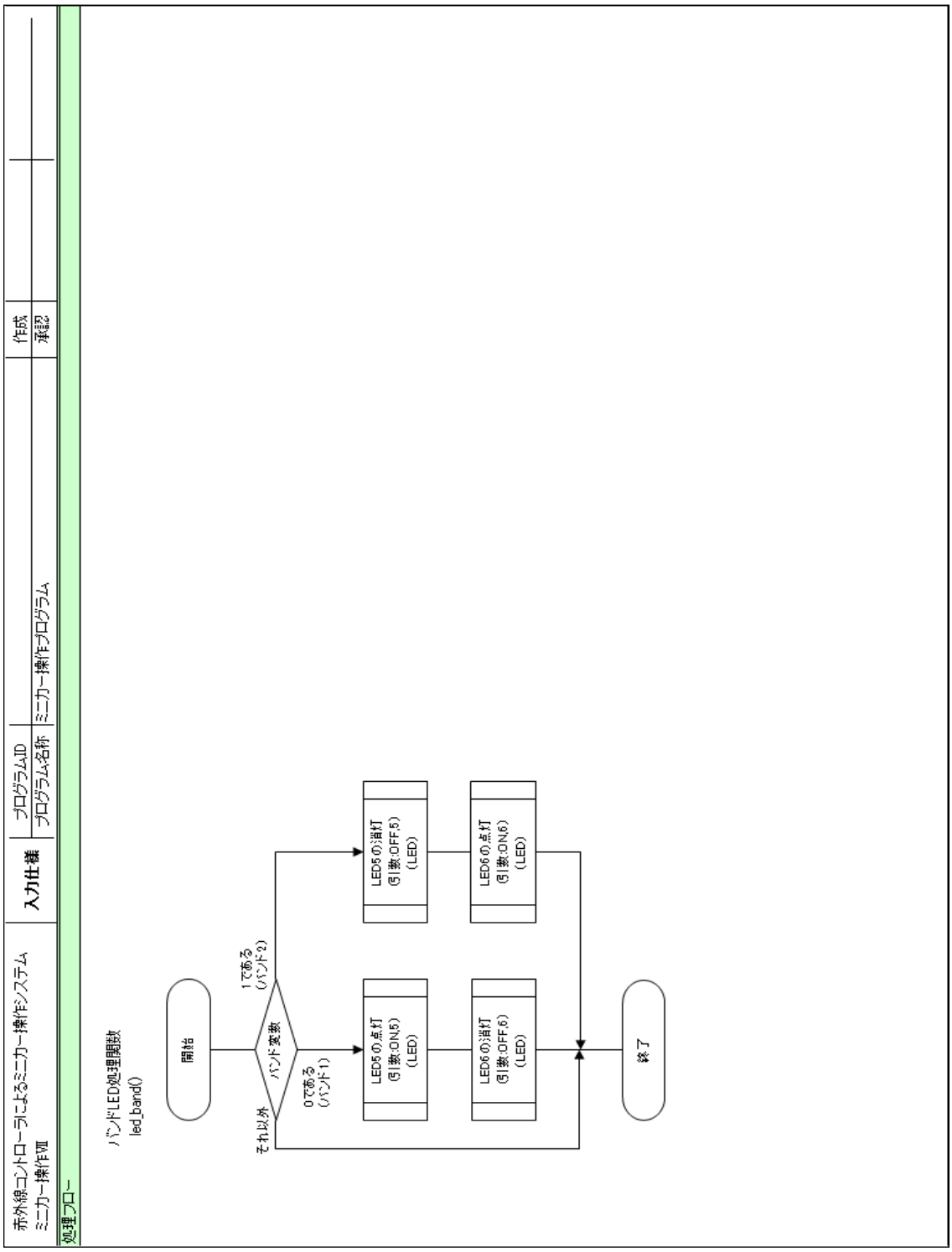


-- 〃ミニカー操作Ⅶ　処理フロー（信号送信関数）　--

-- 〃ミニカー操作Ⅶ　処理フロー（赤外線送信関数）　--

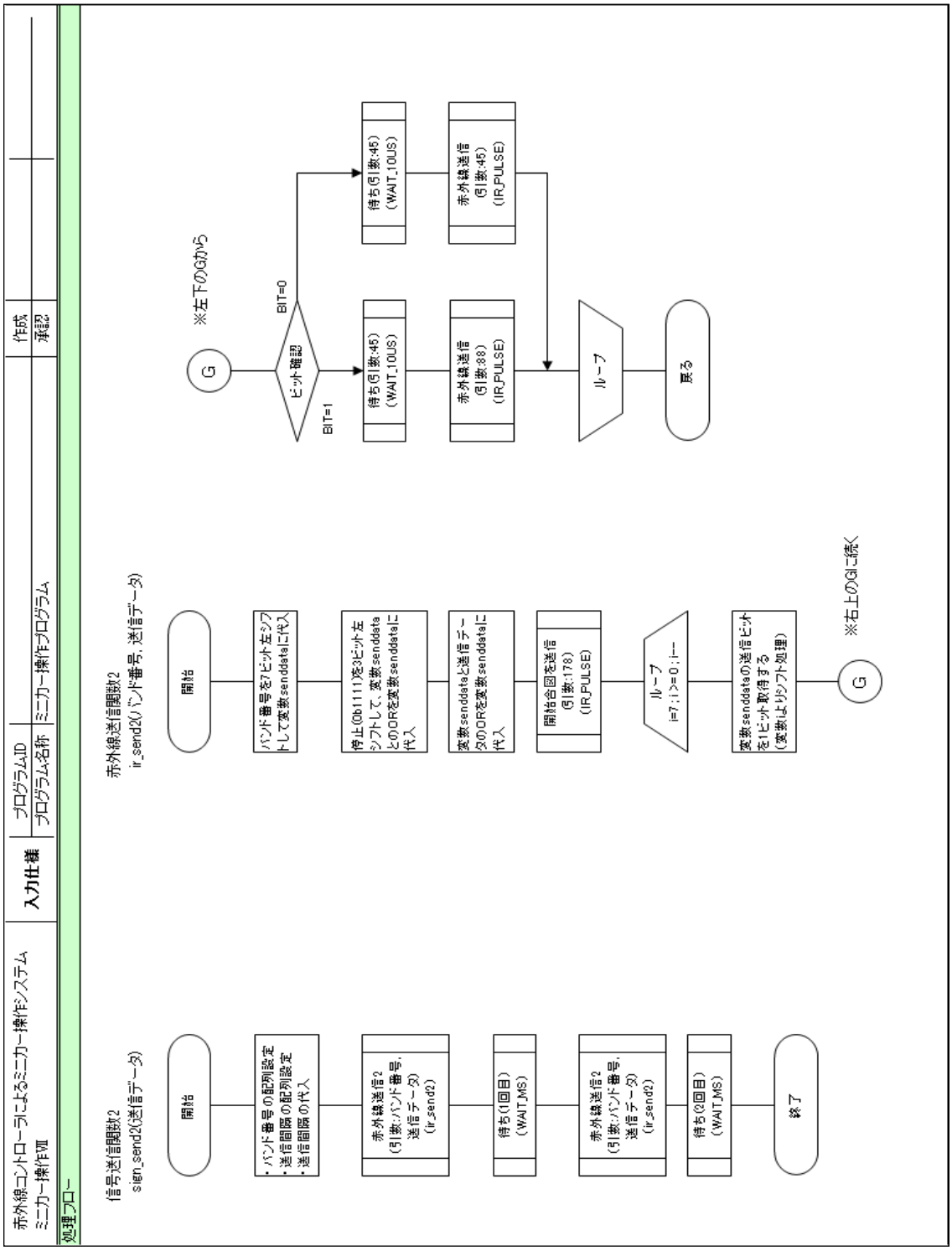


-- ミニカー操作Ⅶ　処理フロー（バンド LED 処理関数） --



-- 〃ミニカー操作Ⅶ　処理フロー（信号送信関数 2）　--

-- 〃ミニカー操作Ⅶ　処理フロー（赤外線送信関数 2）　--



14.1.3. プログラムの確認

-- ミニカー操作Ⅶ プログラム（宣言） --

```
12//インクルード
13#include "RC_Header.h"
14
15// 赤外線送信関数宣言
16void ir_send( unsigned char, unsigned char );
17void ir_send2( unsigned char, unsigned char );
18
19// 信号送信関数
20void sign_send( unsigned int );
21void sign_send2( unsigned int );
22
23// 動作決定処理関数
24void select_move( unsigned int );
25
26// バンドのLED処理関数
27void led_band( void );
28
29
30// バンド変数
31unsigned int now_band = 0;
32
33// 停止フラグ
34int stop_flag = -1;
```

(1)

(2)

(1) 関数宣言（17 行目）

・赤外線送信関数 2 を宣言する。

(2) 関数宣言（21 行目）

・信号送信関数 2 を宣言する。

-- ミニカー操作Ⅶ　プログラム（動作決定処理関数） --

```
// 動作決定処理関数
void select_move( unsigned int switch_sign )
{
    switch( switch_sign )
    {
        // 前進(SW2)
        case 2:
            sign_send( 0b0001 );
            stop_flag = 1;
            break;
        // 後退(SW4)
        case 8:
            sign_send( 0b0010 );
            stop_flag = 1;
            break;
        // 右折(SW3)
        case 4:
            sign_send( 0b0100 );
            stop_flag = 1;
            break;
        // 左折(SW1)
        case 1:
            sign_send( 0b0011 );
            stop_flag = 1;
            break;
        // 右前進(SW2＋SW3)
        case 6:
            sign_send( 0b0111 );
            stop_flag = 3;
            break;
        // 左前進(SW1＋SW2)
        case 3:
            sign_send( 0b0101 );
            stop_flag = 3;
            break;
        // 右後退(SW3＋SW4)
        case 12:
            sign_send( 0b0110 );
            stop_flag = 3;
            break;
        // 左後退(SW1＋SW4)
        case 9:
            sign_send( 0b0000 );
            stop_flag = 3;
            break;
        // バンド切り替え(SW5)
        case 17:
            //バンド1を代入
            now_band = 0;
            //バンドのLED処理
            led_band();
            // バンドを切り替えたら1秒待つ
            WAIT_MS( 1000 );
            break;
        // バンド切り替え(SW5＋SW6)
        case 20:
            //バンド2を代入
            now_band = 1;
            //バンドのLED処理
            led_band();
            // バンドを切り替えたら1秒待つ
            WAIT_MS( 1000 );
            break;
        // 赤外線送信:前方[ミドリこうら](SW2＋SW6)
        case 34:
            sign_send2( 0b001 );
            stop_flag = 3;
            break;
        // 赤外線送信:後方[バナナ](SW4＋SW6)
        case 40:
            sign_send2( 0b010 );
            stop_flag = 3;
            break;
        // 赤外線送信:前後[スーパースター](SW5＋SW6)
        case 48:
            sign_send2( 0b101 );
            stop_flag = 3;
            break;
        // その他は停止処理
        default:
            if( 0 <= stop_flag )
            {
                // 停止フラグが0の時信号送信
                if( 0 == stop_flag )
                {
                    // 停止信号の送信
                    sign_send( 0b1111 );
                }
                // 停止フラグを-1減らす
                stop_flag--;
            }
            break;
    }
}
```

(3)

(4)

(3) 赤外線送信コマンド追加（212-226 行目）

(4) 信号送信関数 2 の呼び出し（214,219,224 行目）

・赤外線を送信する。引数に、使用アイテムの送信データを渡す。

-- ミニカー操作Ⅶ プログラム（赤外線送信関数 2） --

```
266// 赤外線送信関数2
267void ir_send2( unsigned char band, unsigned char data )
268{
269     int i;
270     unsigned char senddata;
271
272     // ビットシフトとORを使ってデータ作成
273     senddata = band << 7;
274     senddata |= 0b1111 << 3;
275     senddata |= data;
276
277     // データ送信開始
278     // 開始合図
279     IR_PULSE( 178 );
280
281     // バンドとデータの8ビットデータを作成
282     for( i = 7; i >= 0; i-- )
283     {
284         if( 1 == (0b00000001 & (senddata >> i)) )
285         {
286             // 1の信号を送付
287             WAIT_10US( 45 );
288             IR_PULSE( 88 );
289         }
290         else
291         {
292             WAIT_10US( 45 );
293             IR_PULSE( 45 );
294         }
295     }
296}
```

(5)

(5) 送信データの作成（272-275 行目）

・バンド(1 ビット)＋動作(4 ビット)**＋アイテム(3 ビット)**を合わせた 8 ビット。

ただし、動作は「停止」(0b1111)で固定とする。

-- ミニカー操作Ⅶ　プログラム（信号送信関数 2） --

```
299// 信号送信関数2
300void sign_send2( unsigned int data )
301{
302    // ﾊﾞﾝﾄﾞ1/2
303    const unsigned char band[] = { 0b0, 0b1 };
304
305    // 1回目と2回目のﾊﾞﾝﾄﾞごとの送信間隔
306    const unsigned int wait[] = { 13, 39 };
307
308    // 送信間隔
309    const unsigned int roop_wait = 100;
310
311    // 1回目の送信
312    ir_send2( band[now_band], data );
313
314    // ﾊﾞﾝﾄﾞに合わせた時間停止
315    WAIT_MS( wait[now_band] );
316
317    // 2回目の送信
318    ir_send2( band[now_band], data );
319
320    // 残り送信間隔分待つ
321    // 送信間隔 - ﾊﾞﾝﾄﾞごとの送信間隔 - ir_send(約10ms)×2
322    WAIT_MS( roop_wait - wait[now_band] - 20 );
323}
```

(6)

(6) 赤外線送信関数 2 の呼び出し（311-312,317-318 行目）

・赤外線を送信する。引数に、バンド番号,使用アイテムの送信データを渡す。

15. 講習後のテスト

下記の説明文の空白部分を回答欄に記入せよ。

１.組み込みプログラムについて

自動車や身近な電化製品（冷蔵庫、掃除機）などには、機能の制御を行うためのコンピュータが入っている。このコンピュータは、“マイコン”と呼ばれる。

このマイコンを動かすためのプログラムを［ ① ］プログラムと呼ぶ。

“マイコン”とは、［ ② ］の略称である。

２.設計について

設計とは、物（製品・システム・アプリ等）を作る際に必要となる工程の一つである。設計にはいくつかの種類があるが、大きく［ ③ ］設計書と［ ④ ］設計書の２種類に分けられる。

［ ③ ］設計書は、顧客や外部システム担当者と調整しながらシステムの全体像がつかめるような内容を決める。

［ ④ ］設計書は、システムの各機能をより詳細化し、どのような処理を行ってプログラムを動作させるかを決める。

３.設計書の必要性について

設計書は、顧客が要望している内容を［ ⑤ ］しているかを判断する基準となり、作業を行う場合の［ ⑥ ］となる。

また、複数人で作業を行う場合に［ ⑦ ］を統一するために必要となる。

設計書を作成しておくことで、発展型のものを造るときに手間を減らすことになる。

４.製品の品質

製造に携わる企業では、製品を販売する前に、信頼性、［ ⑧ ］性、［ ⑨ ］性が問題ないかどうかを厳重に確認して、より良い品質の製品を提供しなければならない。

そのためには、評価テストという作業が必要となる。

信 頼 性･･･製品の故障が起こりにくいこと

［ ⑧ ］性･･･製品を扱うことによって生じる事故や、けがなどの被害が少ないこと

［ ⑨ ］性･･･製品の使いやすさ

５.評価テスト

評価テストの目的は、プログラム中の［⑩］をできる限り多く発見することである。

あらゆる視点でテストを行い、発見された［⑩］を早期に修正することによって、最終的に誰の目から見ても品質の高い製品であることを証明することができる。

そのためには、開発者、販売者、ユーザのそれぞれの視点からテストを行う必要があり、品質の良い評価を得るには、一定の評価基準が必要である。

開発分野では、その基準を［⑪］と呼んでいる。

テストは大きく分けて、段階的に［⑫］テスト、［⑬］テスト、システムテストがある。

［⑫］テスト･･･プログラムの構造単位（命令、分岐、反復）で確認する。

［⑬］テスト･･･複数の部品を組み合わせて初めて実現できる動作を確認する。

システムテスト･･･目的の機能や性能が実現しているかを確認する。

６.テストの重要性

テストは物作りで必要不可欠であり、仕様通りに動作し、安全でなければならない。

特に［⑭］系のプログラムは、業務系のプログラムより多くのテストを行う。

実社会でテストを十分に行わなかった状態で製品を出荷して、［⑩］が発覚した場合に考えられることとして、以下の項目が挙げられる。

１）［⑮］への影響度

２）損失

３）［⑯］への迷惑

下記のプログラムの空白行を教材の「付録 A. リモコン通信仕様」と「付録 B. 関数仕様」を参照し回答欄に記入せよ。

【仕様】

・電源を入れると前進するプログラムを作成する。（バンドは 1 とする）

・赤外線送信の関数を作成する。

関数名：ir_send()、引数：バンド番号, 送信データ

```
12 //インクルード
13 #include "RC_Header.h"
14
15 //関数プロトタイプ宣言
16 void ir_send(unsigned char, unsigned char);
17
18 /* 関数名 :main */
19 void main(void)
20 {
21     // バンド1/2
22     unsigned char band[] = { 0b0, (1) };                    //(1)
23
24     // 1回目と2回目のバンドごとの送信間隔
25     unsigned int wait[] = {(2)a, (2)b};                     //(2)
26
27     // 初期値バンド（1=0, 2=1）
28     unsigned int now_band = (3);                            //(3)
29
30     // 送信間隔
31     unsigned int roop_wait = 100;
32
33     // 送信データ
34     unsigned char data;
35
36
37     // PIC初期化
38     PIC_INIT();
39
40     // メインループ
41     while(1)
42     {
43         // 前進するデータを設定する
44         data = 0b0001;
45
46         // 1回目の送信
47         ir_send( (4)a , (4)b );                             //(4)
48
49         // バンドに合わせた時間停止
50         WAIT_MS( (5) );                                     //(5)
51
52         // 2回目の送信
53         (6)                                                 //(6)
54
55         // 残り送信間隔分待つ
56         //（送信間隔 - バンドごとの送信間隔 - ir_send(約10ms)×2）
57         WAIT_MS( roop_wait - (7) - 20);                     //(7)
58
59     }
60
61     //暴走防止無限ループ
62     while(1){}
63 }
```

```
/* 関数名 :ir_send：赤外線送信関数 */
void ir_send(unsigned char band, unsigned char data)
{
    int i;
    unsigned char senddata;

    // ビットシフトとORを使ってデータ作成
    band = band <<(8);                                  //(8)
    senddata = (9);                                     //(9)

    // データ送信開始
    // 開始合図
    IR_PULSE( (10) );                                   //(10)

    // バンドとデータの5ビットデータを作成
    for ( i = 4; i >= 0; i-- )
    {
        //ANDを取りビットのデータを送信
        if ( (0b00000001 & ( senddata >> i) ) == 1)
        {
            // 1の信号を送付
            WAIT_10US( (11) );                          //(11)
            IR_PULSE( (12) );                           //(12)
        }
        else
        {
            WAIT_10US( (13) );                          //(13)
            IR_PULSE( (14) );                           //(14)
        }
    }
}
```

回答用紙

点

組	
名前	

［説明文回答欄］

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

［プログラム回答欄］

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

付録 A.　リモコン通信仕様

この項は「付録 H.昨年度教材参照」の参照番号 32 を参照。

付録 B.　関数仕様

この項は「付録 H.昨年度教材参照」の参照番号 33 を参照。

付録 C.　CD-ROM 構成

この項は「付録 H.昨年度教材参照」の参照番号 28 を参照。(差し替え資料有り)

添付 CD-ROM の“教材”のディレクトリ構成のうち、
今年度の追加・修正分を以下に示す。

ディレクトリ	ファイル	内容
	README.txt	注意事項など
PC_Program	ミニカー操作Ⅱ（フォルダ）	ミニカー操作Ⅱの穴埋め用
設計書	記入用_ミニカー操作Ⅵ_詳細設計書.xls	ミニカー操作Ⅵの詳細設計書
	記入用_ミニカー操作Ⅶ_詳細設計書.xls	ミニカー操作Ⅶの詳細設計書
資料	ミニカー操作Ⅶ（フォルダ）	ミニカー操作Ⅶの参考資料
講習後のテスト	確認テスト.xls	講習後のテスト問題と解答

付録 D.　講習後のテストの模範解答

回答用紙

点

組	
名前	

［説明文回答欄］

①	組み込み	⑨	操作
②	マイクロコントローラ	⑩	欠陥　（「バグ」または「不具合」でも正解）
③	基本	⑪	評価指標
④	詳細	⑫	単体
⑤	理解　（「把握」でも正解）	⑬	結合
⑥	仕様書	⑭	組み込み
⑦	仕様	⑮	社会
⑧	安全	⑯	ユーザ

［プログラム回答欄］

(1)	0b1	(7)	wait[now_band]
(2)a	13	(8)	4
(2)b	39	(9)	band | data
(3)	0	(10)	178
(4)a	band[now_band]	(11)	45
(4)b	data	(12)	88
(5)	wait[now_band]	(13)	45
(6)	ir_send(band[now_band], data);	(14)	45

付録 E.　Windows 7 で使用する際の注意点

添付 CD-ROM に収録した開発環境構築用のプログラムは古いバージョンであり、そのままでは Windows 7 で使用することができない。これらのプログラムを Windows 7 で使用するためには、以下の手順が必要となる。

① PICC LITE

システムに“MSVCP71.dll”が存在しない場合、PICC LITE のインストールで不具合が発生する。そのため、先に“MSVCP71.dll”を入手しておき、“C:¥Windows¥System32”の直下に置いておく必要がある。（既存の場合は、この手順は不要）

② HI-TIDE

開発環境の日本語化で“日本語化.bat”を使用しても日本語化できない。この場合、エクスプローラーを用いて、“Install”フォルダにある“eclipse”フォルダをコピーし、“C:¥Program Files¥HI-THECH Software¥HI-TIDE¥3.12”にある“eclipse”フォルダに上書きする。

③ IC-Prog

- メニューバーの[設定]-[オプション]で、[各種]タブにある『NT/2000/XP ドライバを使用』のチェックを外す。
- メニューバーの[設定]-[ハードウェア]で、「インターフェース」を『Windows API』とし、「ポート」を使用する Com ポートに設定する。

④ HI-TIDE 以外の統合開発環境を使用する場合

例えば MPLAB を使用する場合、ファイル名やフォルダ名に全角文字を使用することができない。ファイルパスに全角文字を含んでいると、目的のファイルを見つけることができない。

これらを回避するためには、教材内で日本語になっているフォルダ名やファイル名を全て英数字でリネームする必要がある。

付録 F.　組込み OS としての Linux

組込み OS のシェアは､ITRON 系 API と T-Kernel 系の API で 50%以上を占めている。しかし近年になって、デジタル家電や携帯電話、PDA など、要求性能の高い製品向けの 32bit 組み込み OS 市場で、Linux が急速にシェアを伸ばしている。

オープンソースでカーネル部分や実装機能を自在にカスタマイズできることから､Linux はエンベデッドに向いていると早くから言われてきた。組み込み Linux 導入の機運が本格的に高まったのは 2000 年前後。その頃には、MontaVista Software、Lynx Real-Time Systems（現在は Lynx Works)、Lineo（後の Embedix、Metrowerks に買収される）などの組込み Linux に特化したディストリビュータも台頭してきた。

企業名	製品名
MontaVista Software	MontaVista Linux
TimeSys	TimeStorm Linux Development Suite
Lynux Works	BlueCat Linux
リネオソリューションズ	uLinux
アックス	AxLinux
FSMLabs	RTLinux

表 F-1　主なディストリビューション

組込み Linux には、以下のメリットがある。多くは、PC で発展してきた Linux システムの長所を受け継いだものである。

・オープンソース
・豊富なミドルウェア、アプリケーション
・ネットワーク、マルチメディア対応
・標準 API によるプログラミング
・充実した開発環境
・豊富なプロセッサおよび開発ボードのサポート
・信頼性、堅牢性
・豊富な開発者
・豊富な資料
・ロイヤリティーフリー

オープンソースのメリットについては、詳細は割愛するが、組込み OS の観点では以下の 2 点が重要である。

・ソフトウェアの内部処理を詳細に把握する事が可能で、自分自身で品質確認やメンテナンスを行う事ができる。

・ライセンス条件にもよるが、事由に改変して製品に搭載できるので、問題発生時にソフトウェアベンダの修正納期に影響されず即座に対応可能。

また、従来の組込み OS では実装されるまでに時間がかかったり、調達に多くの費用がかかっていたネットワーク、マルチメディア、最新機器に対応するソフトウェアは、Linux の場合は短期間で開発されてカーネルまたはミドルウェアに取り込まれる。そして、これらは自由に利用できるものが多い。

このように、多くのメリットを持つ組込み Linux にも、克服しなくてはならない課題もある。これらの多くは、Linux が PC またはワークステーション向けに開発された汎用 OS である事に起因している。

・リアルタイム性能の欠如

・ハードウェア資源の大量消費

・高速立ち上げ、シャットダウン

・カーネル技術者の不足

これらの課題を解決するために、各種団体が改善に取り組んでいる。

団体名	設立	HomePage
Embedded Linux Consortium	2000 年 3 月	http://www.embedded-linux.org/
日本エンベデッド・リナックス・コンソーシアム	2000 年 7 月	~~http://www.emblix.org/~~ （2013 年 12 月現在、該当するホームページは存在しない）
Consumer Electronics Linux Forum	2003 年 7 月	http://www.celinuxforum.org/

表 F-2　各種コンソーシアム

最も早く設立された ELC(Embedded Linux Consortium)は、組込み機器に必要な機能を実現するための仕様を策定した。(Embedded Linux Consortium Platform Specification)

同時期に設立された日本エンベデッド・リナックス・コンソーシアム(Emblix)は産学協同による普及活動を行っている。CE Linux Forum はソニーや松下電器と言ったメーカーが中心となって旗揚げされた。そして、カーネルバージョンが 2004 年に 2.6x 系に進化し、リアルタイム性も RTOS と同等の性能が確保されるようになると、一気に普及の足を早めた。

付録 G.　車載ネットワーク

本文中で触れた自動車の電子制御装置（**ECU**：Electronic Control Unit）は、普通自動車 1 台に数十個搭載されており、高級車になると 100 個以上にもなる。これらの ECU は相互にデータのやり取りを行っている。このデータのやり取りに使用されるのが車載ネットワーク（車載 LAN）である。

かつての自動車の通信システムは、操作系など集中配置されたスイッチ群と、そのスイッチ信号を受け取って集中制御を行う ECU 間のローカルな“point to point システム”であった。スイッチ群と ECU 間にはハーネス結線部が LAN（Local Area Network）化され、低速のシリアル通信やカスタム通信プロトコルなどが使用されていた。ECU により集中制御されたローカルな通信システムでは、制御範囲がシステム内で完結していた。そのため、1 つの ECU の故障がシステム全体に影響することは無かった。

それらのローカルな通信システムは、次第に複数の ECU が 1 つの通信回線（通信バス）に接続される電装系ネットワーク・システムに発展していった。これらの車載 LAN 通信プロトコルは、自動車メーカーごとにオリジナルのプロトコルとして開発され、大規模な電子システムが実現化されていった。その後、2000 年頃から、主に基幹車載 LAN に業界標準の **CAN**（Controller Area Network）を使用した制御システムにとって代わられるようになってきた。この変化により、自動車の走行性・安全性・快適性の向上に向けて、増大する ECU をネットワーク化し、分散協調制御することが可能になってきたのである。

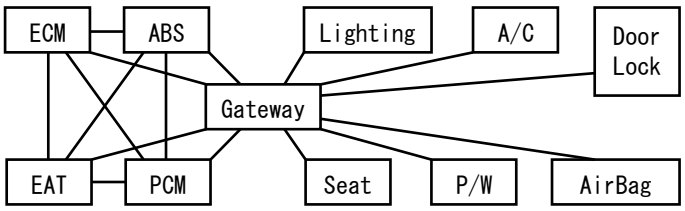


図 G-1　従来の電装部品制御の配線

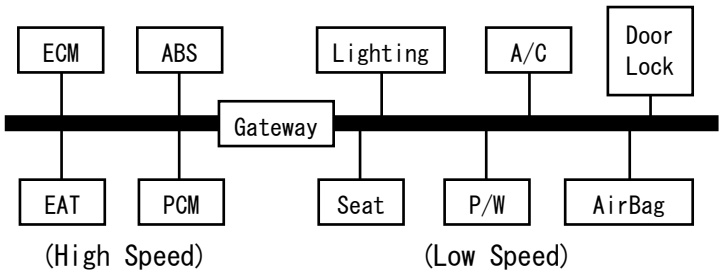


図 G-2　CAN に代表される車載 LAN の場合

現在の車載 LAN は、ボディ系・パワートレイン系・シャーシ系・安全系・情報通信系・故障診断系など、通信速度や通信方式などそれぞれの制御対象に合わせて、最適なネットワーク・システムを敷設するという車載 LAN の構成になっている。

ドアや電動シート、インテリジェント・キー、エアコンなど、ボディ系システムは CAN で、その下位システムは **LIN**（Local Interconnect Network）で接続する。エンジンやパワーステアリング、ブレーキ、自動変速機など高速かつ信頼性の高い通信システムが求められるパワートレイン系システムについては FlexRay で接続する。また、エアバックシステムに代表される安全系システムでは **ASRB**（Safe-by-Wire Plus）などのプロトコルを、情報通信系システムでは **MOST** や **IDB1394** などを用い、それぞれの制御システム間はゲートウェイを介して相互接続され、自動車全体での分散協調制御の実現が目指されることになる。

エレクトロニクス化に伴う自動車の制御用途は、大きくボディ系（車載機器）制御、情報通信系制御、パワートレイン系（駆動走行）制御、シャーシ系（保安装置：安全系・エアバック系）制御などに分類階層構造化され、それぞれの階層が車載 LAN プロトコルを使ってネットワーク化されている。なぜ階層化されているかというと、要求レベルの異なるものを同一のプロトコルで扱うことが非効率だからである。

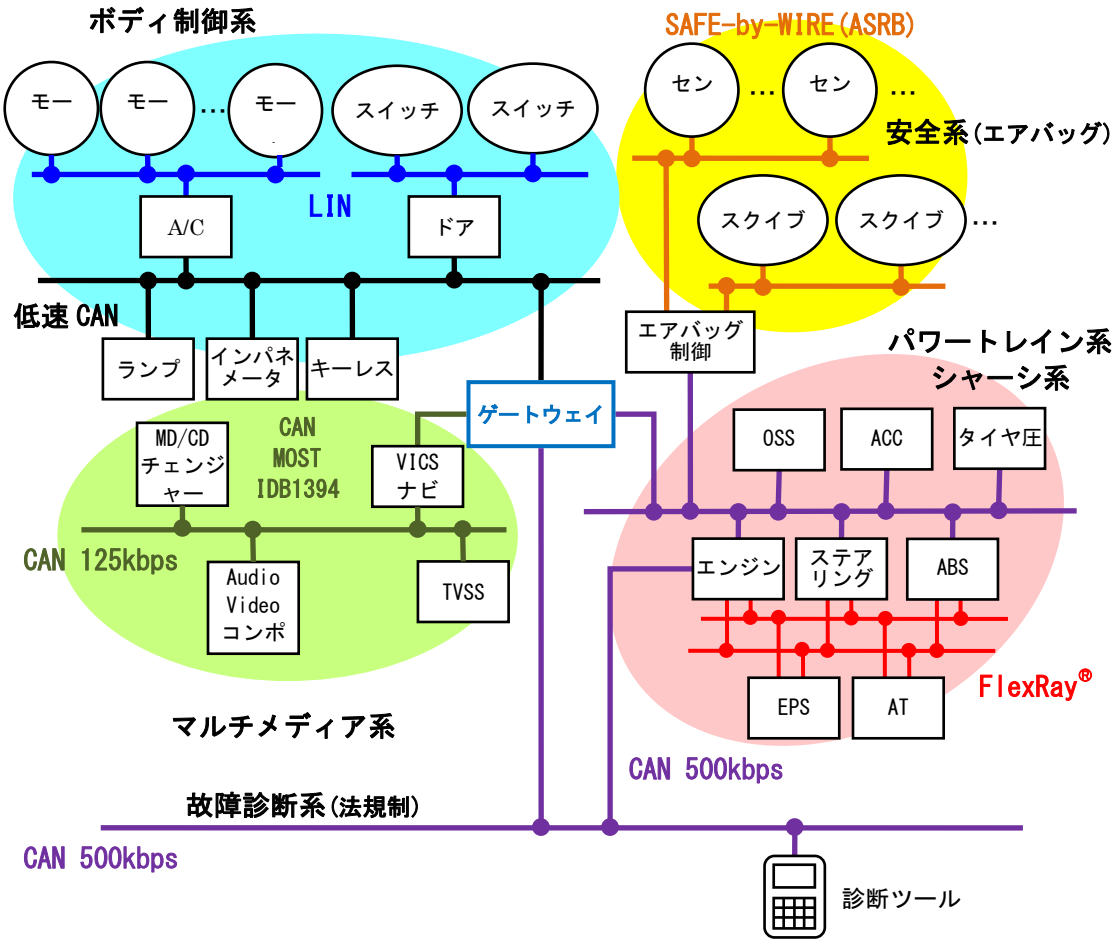


図 G-3　現在の車載 LAN の主な構成

用途	ボディ系	安全系	パワートレイン系	情報系
主なアプリケーション	ドア、シート、エアコン、照明	エアバッグ衝突センサ	エンジン、ブレーキ、ABS、トランスミッション	カーナビ、カーオーディオ
通信速度	低速 LAN (125kbps 以下)	中速 LAN (数十 kbps～ 500kbps)	中速～高速 LAN (500kbps～ 10Mbps 程度)	高速 LAN (数 Mbps～ 数百 Mbps)
特徴	･低コスト ･銅線通信	･タイムスロット通信 ･高信頼性 ･2 重系	･タイムスロット通信 ･高信頼性 ･2 重系 ･光通信(高速 LAN の場合)	･リアルタイムデータ通信 映像情報通信は光通信必須
車載 LAN プロトコル	･CAN(低速) ･BEAN ･LIN	･CAN(中･高速) ･Safe-by Wire ･BST	･CAN(高速) ･FlexRay	･CAN(中速) ･D2B/Optical ･IEBus ･MOST ･IDB1394 ･MOSTⅡ

表 G-1　各用途に適した車載 LAN プロトコルの特徴

- チョロＱ、チョロＱハイブリッド！ は、株式会社タカラトミーの登録商標です。
- PIC は、Microchip Technology Inc.の登録商標です。
- その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

付録 H.　昨年度教材参照

参照番号	参照項目番号	参照項目名	参照ページ	差替資料
1	3.1.	設計書の必要性	13	有
2	3.2.	基本設計	14	
3	3.3.	詳細設計	14	
4	4.1.	制作手順	23	
5	4.2.	部品表・配置図	24	
6	4.3.	はんだ付け	25	
7	5.1.	設計書の書き方	28	
8	5.2.	LED 処理 I	29	
9	5.3.	ミニカー操作 I	35	
10	6.1.1	セットアップ	40	
11	6.2.1.	セットアップ	44	
12	6.3.1.	セットアップ	48	
13	6.4.1.	セットアップ	49	
14	6.5.1.	プロジェクトの設定方法	53	
15	6.5.2.	プロジェクトの設定確認方法	58	
16	6.5.3.	HI-TIDE の起動方法	59	
17	6.5.4.	HI-TIDE の起動時エラーについて	59	
18	7.	プログラムの作成	61	
19	8.1.1.	変数	67	
20	8.1.2.	計算式	68	
21	8.1.3.1.	文法	70	
22	8.1.3.2.	例題プログラム	70	
23	8.1.4.1.	文法	71	
24	8.1.4.2.	例題プログラム	72	
25	8.1.5	if 文・else 文	73	
26	8.2.	C 言語の配列	74	
27	8.3.	C 言語の関数	77	
28	9.	赤外線	82	有
29	10.	実践プログラム（LED 編）	86	

参照番号	参照項目番号	参照項目名	参照ページ	差替資料
30	11.	自動車組込み製品の品質管理	96	
31	12.	実践プログラム（ミニカー編）	108	
32	付録Ａ.	リモコン通信仕様	151	
33	付録Ｂ.	関数仕様	152	
34	付録Ｃ	ＣＤ－ＲＯＭ構成	153	有

平成 25 年度「成長分野等における中核的専門人材養成の戦略的推進事業」
自動車組込み分野の中核的専門人材養成の実践的教育プログラムの実証

企業提案型プロジェクト学習　実践教材

平成 26 年 2 月

学校法人電波学園　名古屋工学院専門学校
〒456－0031 愛知県名古屋市熱田区神宮 4-7-21
Tel：052-681-1311

●本書の内容を無断で転記､掲載することは禁じます。