計測・制御用プログラム言語 NXC入門

Introduction of NXC for Measurements and Controls

伊藤　陽介　*Ito Yosuke*

暫定版

２０１２年５月

鳴門教育大学

ま　え　が　き

みなさんはコンピュータというとどのような形を想像するでしょうか？　ほとんどの人はパソコンというでしょう。しかし，コンピュータは大きくわけて２種類の使い方がなされています。パソコンのように文書を作成したりインターネットを閲覧したりするような様々な情報を処理するコンピュータと，機械の中に部品として組み込まれて温度や音を読み取り，モータやヒーターを制御するコンピュータがあります。

身の回りにある機械をあげてみましょう。携帯電話，自動車，エアコン，電子炊飯器，洗濯機，電子レンジなどたくさんの種類があります。例えば，電子炊飯器は「はじめちょろちょろ中パッパ，赤子泣いてもふた取るな」といわれるような微妙な温度調整を行って，おいしいご飯を炊いています。単純な動作だけではなく，まわりの状態に応じて適切な働き方をさせるためには，コンピュータが必要です。また，ロボットにコンピュータを内蔵すると，自分で考えて行動できるようになります。

複雑な手順をコンピュータにさせるためには，それを表すための言葉を使います。その言葉のことをプログラム言語といいます。パソコンの文書作成用ソフトウェアやゲームなどもすべてプログラム言語で作られています。プログラム言語にはたくさんの種類がありますが，よく使われているものとしてＣ言語があります。プログラム言語を学ぶときに，実際にそれが自分の考えているとおりに動作するかどうか確かめる必要があります。パソコンには，本体にディスプレイやキーボード，マウス，スピーカーが付いていますが，自分自身を移動させることはできません。プログラムの動作結果のほとんどはディスプレイに表示された絵や文字として反応があるだけです。

しかし，コンピュータを組み込んだロボットは，モータなどを使って移動することができます。例えば，「迷路を通り抜ける」というプログラムを実行すると，実際に迷路を探索しながら抜け出すような動作をロボットが行うと面白いと思いませんか？　自分だけのロボットを作り，そして自分の思い通りにロボットを動かすための魂ともいえる計測・制御用プログラム言語を学びましょう。このテキストではＣ言語と似ている NXC という名前のプログラム言語を取り上げています。この言語を使うとブロックで作られたロボットの動きを簡単にプログラムとして作ることができます。

それでは，楽しいプログラミングの世界に入りましょう。

2012 年 5 月

鳴門教育大学　生活・健康系コース（技術・工業・情報）
教　授　伊　藤　　陽　介

本テキストの使い方

本テキストは，主に中学生がロボットを動かすためのプログラム言語を学ぶことができるように作られています。

第１章では，ロボットの頭脳となるコンピュータを内蔵したNXTブロックの使い方とNXCのプログラムを作るためのソフトウェアの設定方法について説明しています。

第２章から第４章までは，NXTブロックを単体で使って基本的なプログラムの作り方について説明でしています。

第５章から第９章までは，移動型三輪ロボットを使っていろいろなプログラムを簡単なものから順番に説明しています。各章を順番に学習することで基本的なロボットの動きをNXCのプログラムとして作ることができるようになります。プログラム例を実際に入力して，ロボットを動かしてみるとプログラムに書かれた命令の意味がよりわかるようになります。

第１０章は，ラインをたどる「ライントレース・ロボット」について説明しています。簡単そうに見えて奥の深い「ライントレース」に取り組んでください。この章の学習が終わったら，ロボットの形を工夫してより速くライントレースできるロボットを作成してみましょう。

チャレンジでは，問題を解決するプログラムを自分で作ることで，今まで学んだことがわかっているかどうか確認できるようにしています。

このテキストに掲載されているプログラムは，次の動作環境で動作を確認しています。

パソコンのＯＳ	：Windows 7 (32bit版または64bit版)
NXCのバージョン	：NXC Version 2.2
NXCの開発環境	：Bricx Command Center Version 3.3
ロボットのキット	：LEGO Mindstroms education 9797
NXTブロックのファームウェアのバージョン	：1.31

計測・制御用プログラム言語 NXC 入門

目　　次

第１章　ロボット教材

１．１　コンピュータを内蔵したブロック

LEGO 製 Mindstorms NXT（レゴ製マインドストーム・ネクスト）は，ロボットを作ることのできる様々なブロックとロボットの頭脳となるコンピュータを内蔵したブロック NXT を含んでいます。NXT ブロック各部の機能を図１－１に示します。

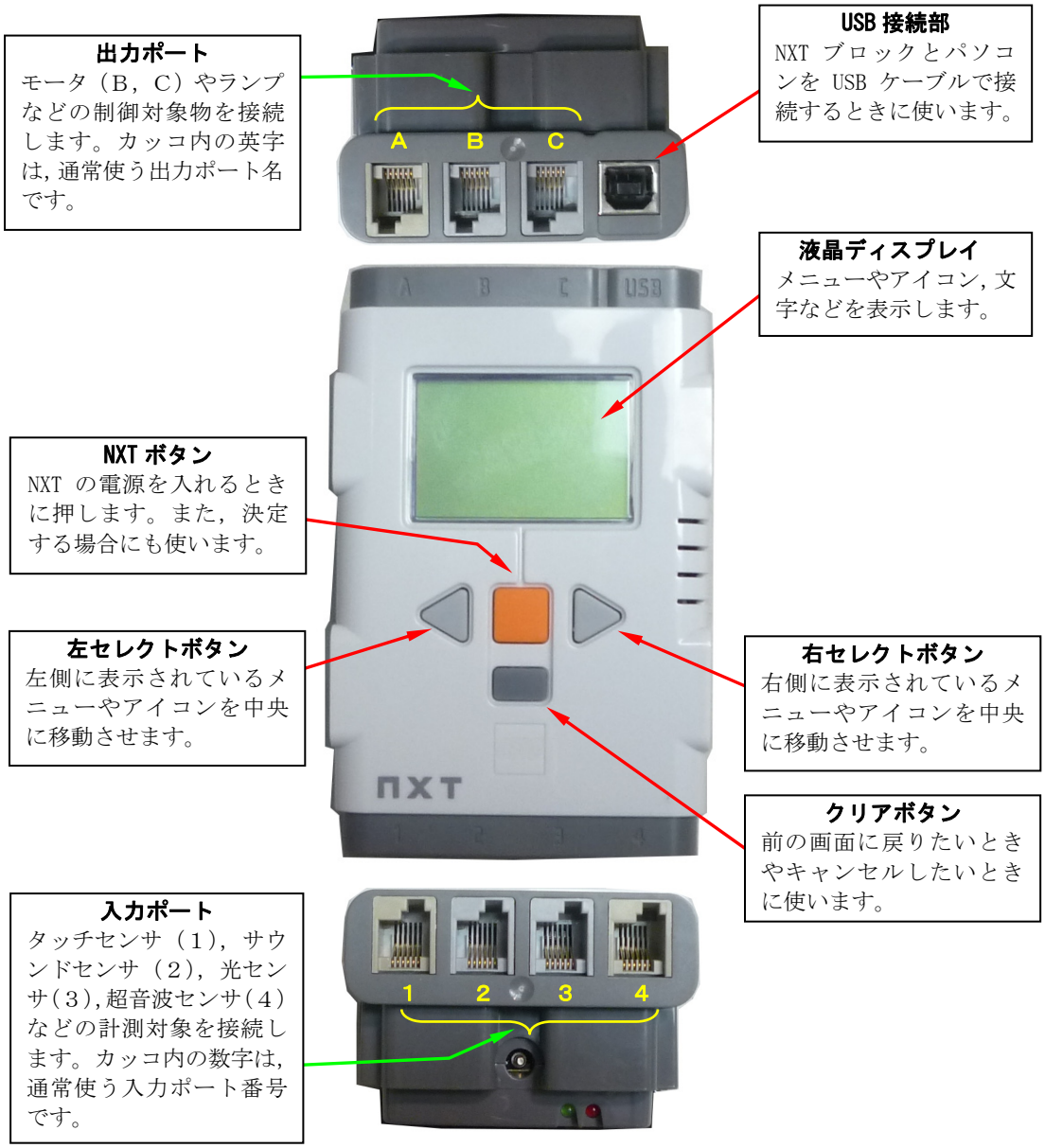


図１－１　NXT ブロックの使い方

１．２　使用するロボット

このテキストは大きく分けて２つの学習内容の構成からできています。そのため，２種類の教材を使用します。

第２章から第４章では，図１－１の NXT ブロックを単体で使用し，基本的な情報処理の流れとプログラムについて学習します。

第５章から第１０章では，図１－２に示す移動型三輪ロボットを実際に動作させてプログラムを使って計測や制御を行う情報処理について学習します。移動型三輪ロボットは，Mindstorms NXT に同梱されている組み立て説明図を見て作りましょう。NXT ブロックと他のブロックを組み合わせて様々な形のロボット作ることができますが，まず移動型三輪ロボットを使って学習を始めましょう。

図１－２　移動型三輪ロボット

１．３　計測・制御用プログラム言語

ロボットを思い通りに動かすためには，NXT ブロックに内蔵されたコンピュータに動きを手順として指示しなければなりません。この手順のことをプログラムといいます。ここでは，ロボットが外部の状態を知ったり，モータを動かしたりできる計測・制御用プログラム言語 NXC を使います。

NXC のプログラムを作る方法は，図１－３のようになります。まず，NXC で書かれたプログラムを BricxCC（Bricx Command Center）というソフトウェアで作成します。そのプログラムを命令コードに変換した後，NXT ブロックに送信します。その後，NXT ブロックを操作してプログラムを実行するとロボットが動作し始めます。

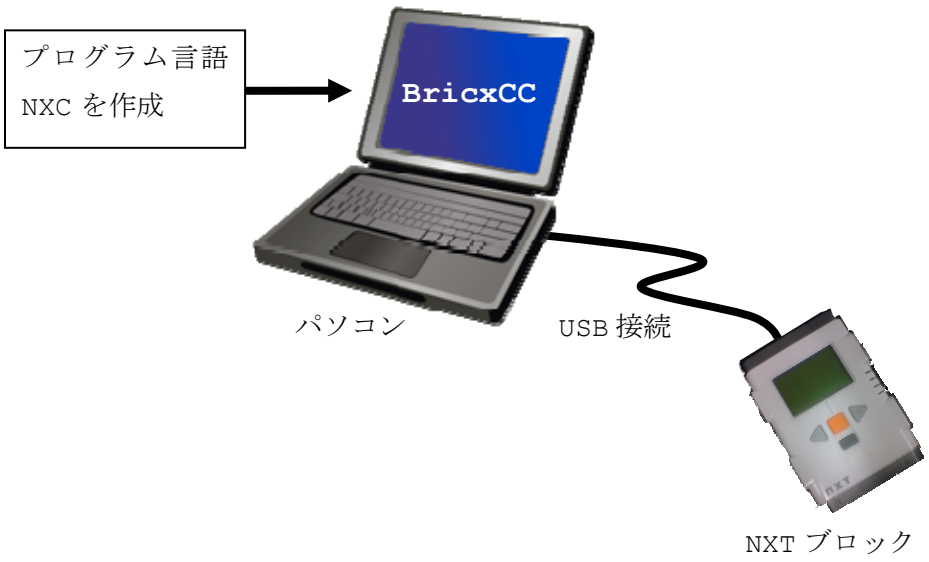


図１－３　プログラム言語 NXC の作り方

１．４　プログラムを作るための準備

NXC のプログラムを Windows パソコンを使って作るための準備は次のとおりです。

① NXT ブロック用ドライバ・ソフトウェアを管理者権限でインストールします。

② NXC プログラムを作るための BricxCC をインストールすると，Windows の Program Files の中に BrixCC というフォルダが作成され，その中に 

というファイルがあります。デスクトップにショートカットを作っておくと便利です。

③ NXT にファームウェアと呼ばれる基本ソフトウェアを BrixCC を使ってダウンロードします。このファームウェアが NXC プログラムを実行し，ロボットを動かします。NXT の電池がなくなるとファームウェアは消えてしまいます。その時は，新しい電池を交換した後，もう一度ファームウェアをダウンロードする必要があります。

④ 

をダブルクリックすると，図１－４のような画面が出てきます。Port（接続元）を「usb」に設定し，Brick Type（ブロックの型）を「NXT」に設定します。Firmware（ファームウェア）は「Standard（標準）」を選択し，OK ボタンをクリックします。

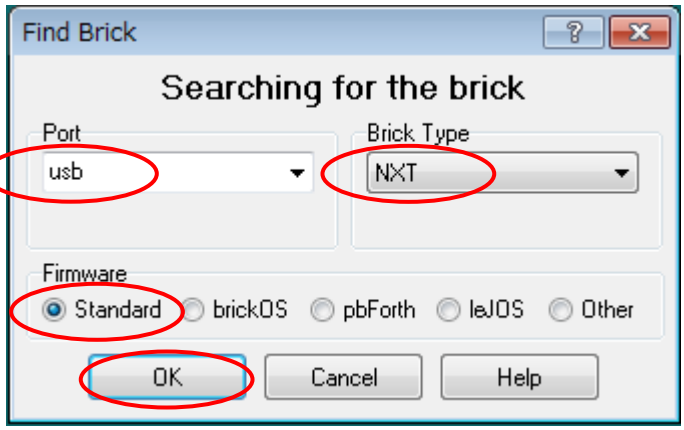


図１－４　Find Brick の画面

⑤　プログラムを打ち込む準備をします。④の操作で図１－５のような２つの画面が出てきます。左側の画面は，この段階では使用しないので触れないようにしましょう。メニューバーにある「File（ファイル）」をクリックし，図１－６のように出てきたら「New（新規）」をクリックします。図１－７に示す画面が表示されてプログラムを入力する環境が整います。この状態では，まだプログラムに名前が付いていないので untitled1 となっています。

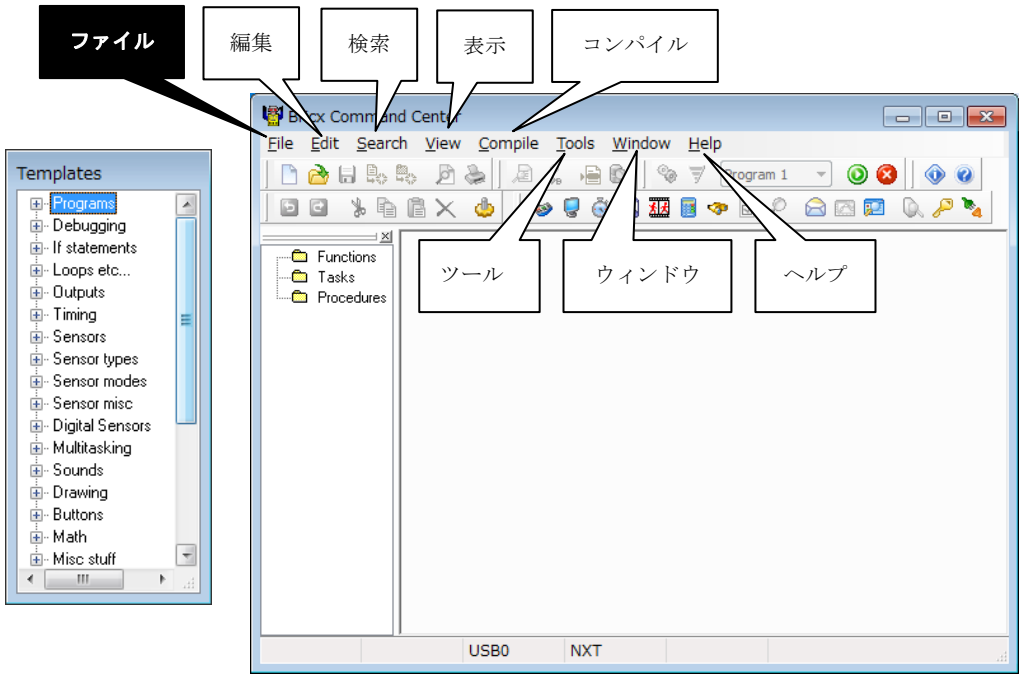


図１－５　BricxCC の基本画面

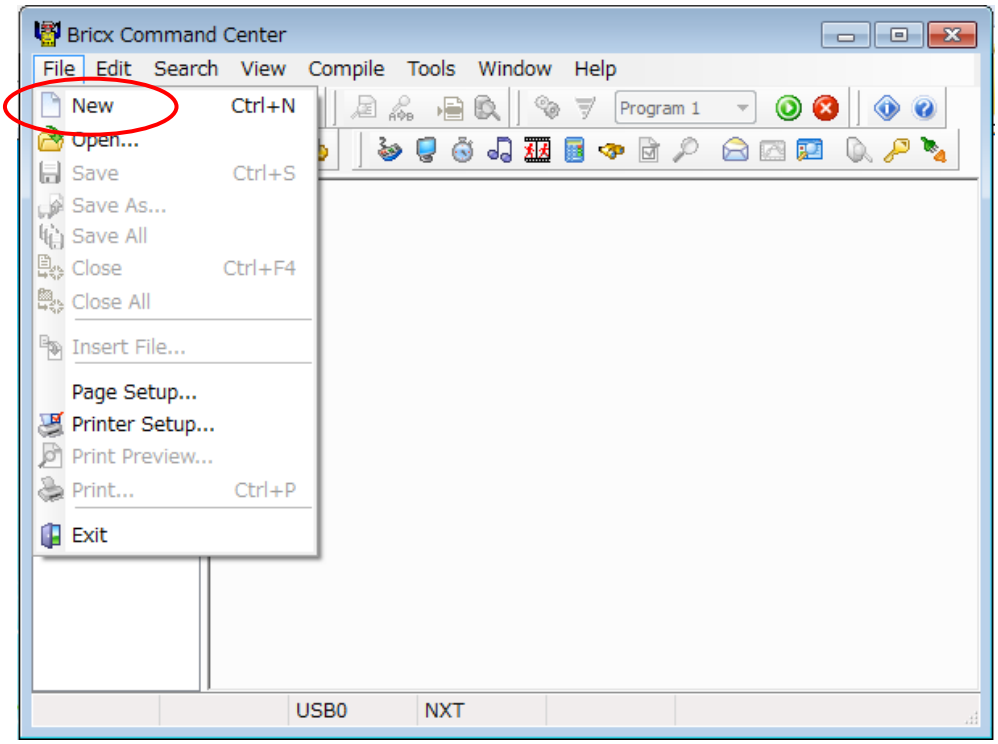


図１－６　ファイルの新規作成のための準備

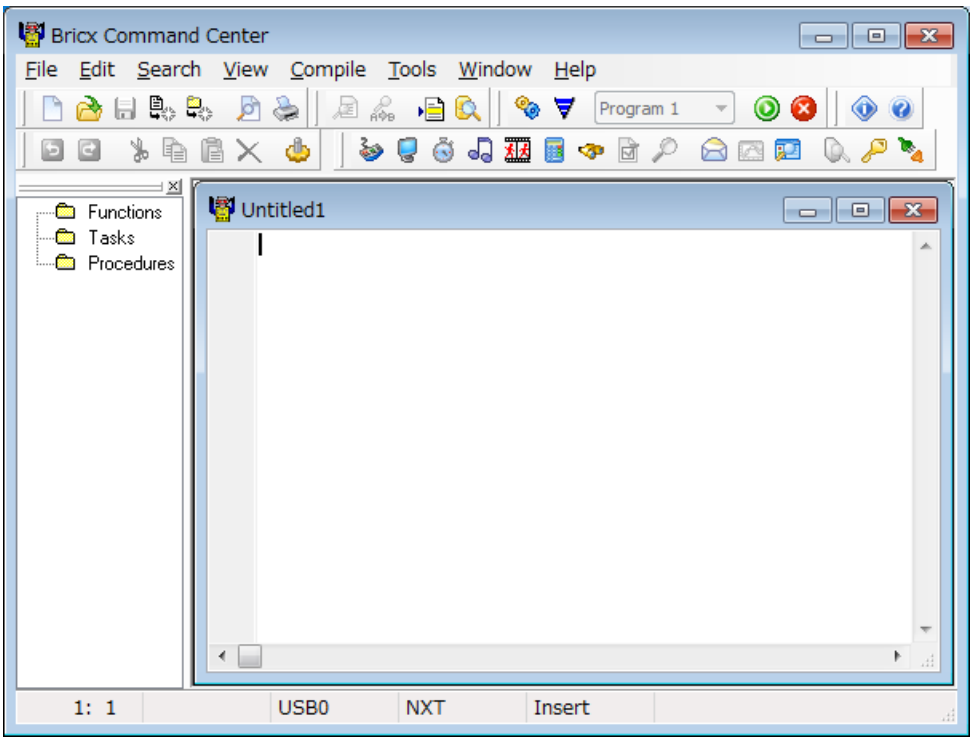


図１－７　Untitled1（名前なし１）のプログラム入力画面

⑥　プログラムの表示や編集のための設定をします。まず，メニューバーの「Edit　(編集)」をクリックし，さらに「Preferences（初期設定)」をクリックします（図１－８)。つぎに，図１－９のように「Editor（編集)」タブを選択します。図１－１０のような画面に切り替わるので，図中で示した項目のチェックを外します。「Color code the program（プログラムのカラーコード)」の項目は，日本語フォントを使わないときはチェックを入れて，プログラムを色づけして見やすくできます。日本語フォントを使うときは，うまく表示されないときがあるので，チェックを外しておきます。また，行と行の間に空白を少し入れて見やすくしたいときは，「Extra line spacing（拡張行間)」の数値を大きくします。

続けて「Edior Font（編集用フォント)」をクリックします。すると，図１－１１の画面が表示されるので，プログラムで日本語を使うときは，図中で指定したように設定し，「OK」ボタンをクリックします。その後，図１－１０に戻るので，再度「OK」ボタンを押します。

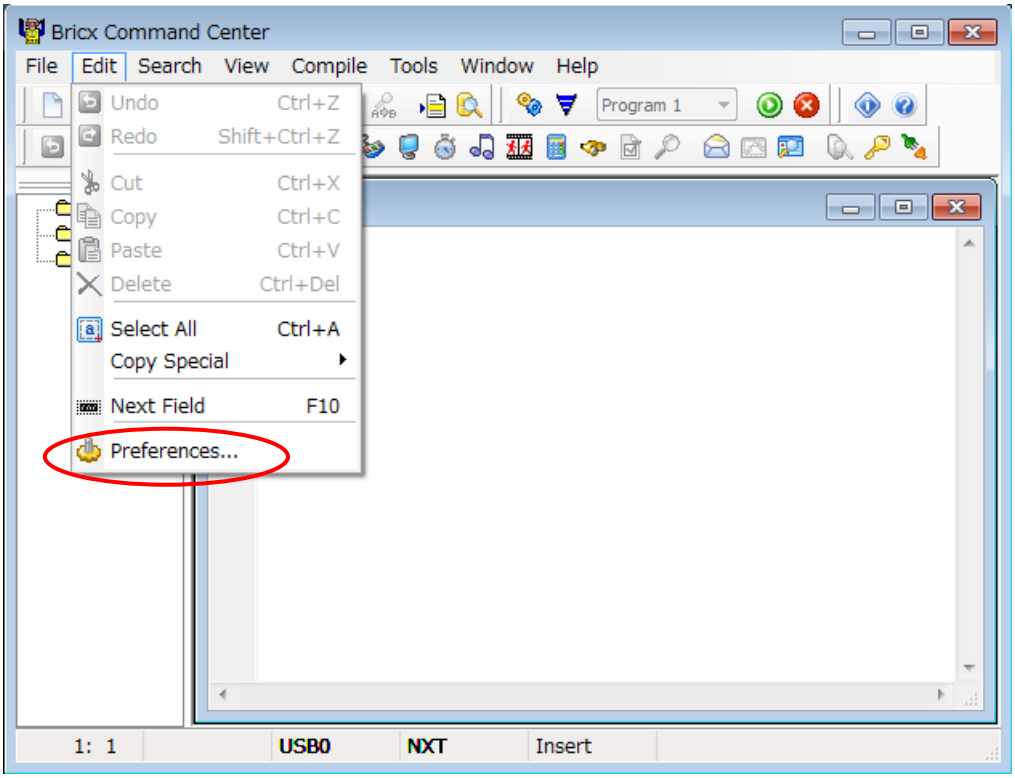


図１－８　設定ウィンドウを開く

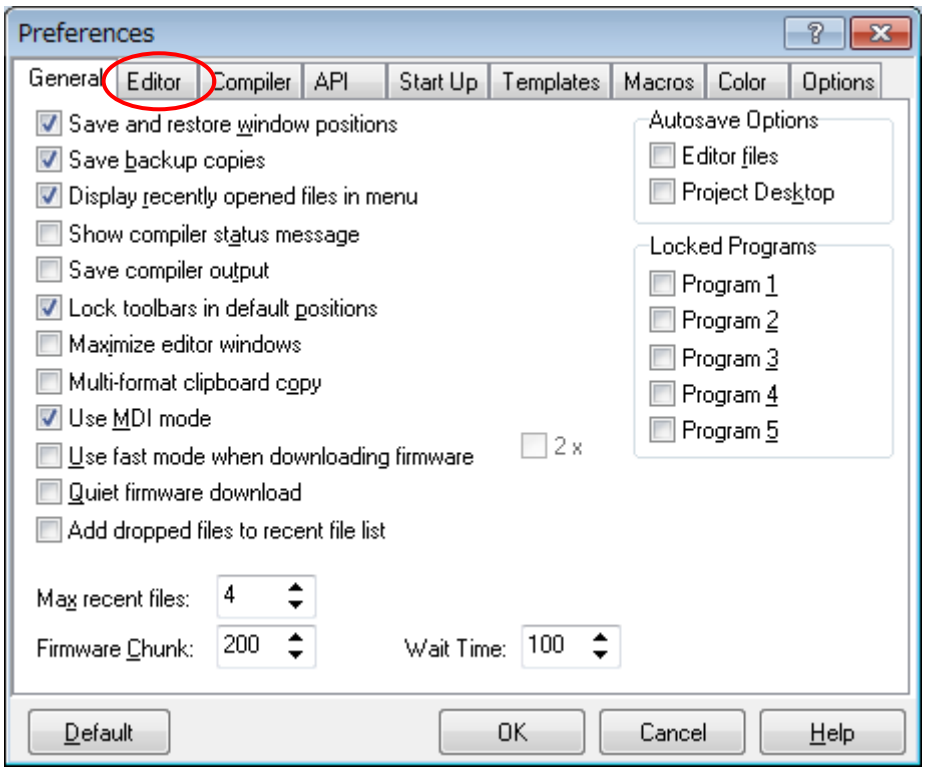


図１－９　「Editor（編集）」タブを選択する

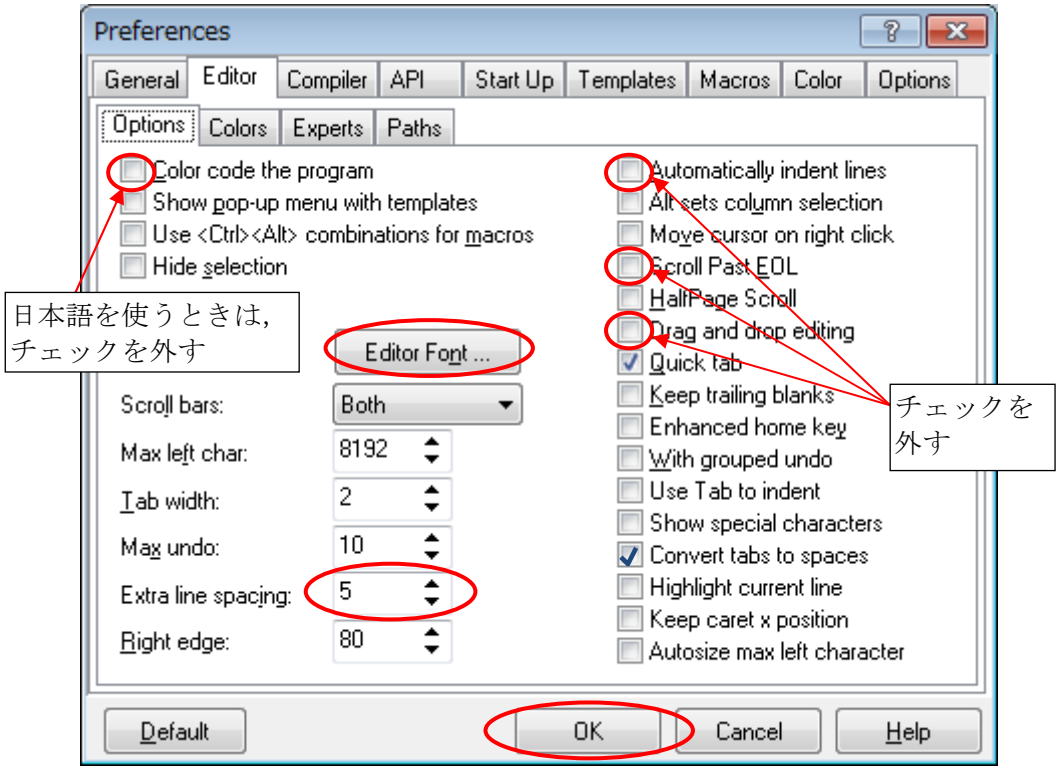


図１－１０　編集の設定をする

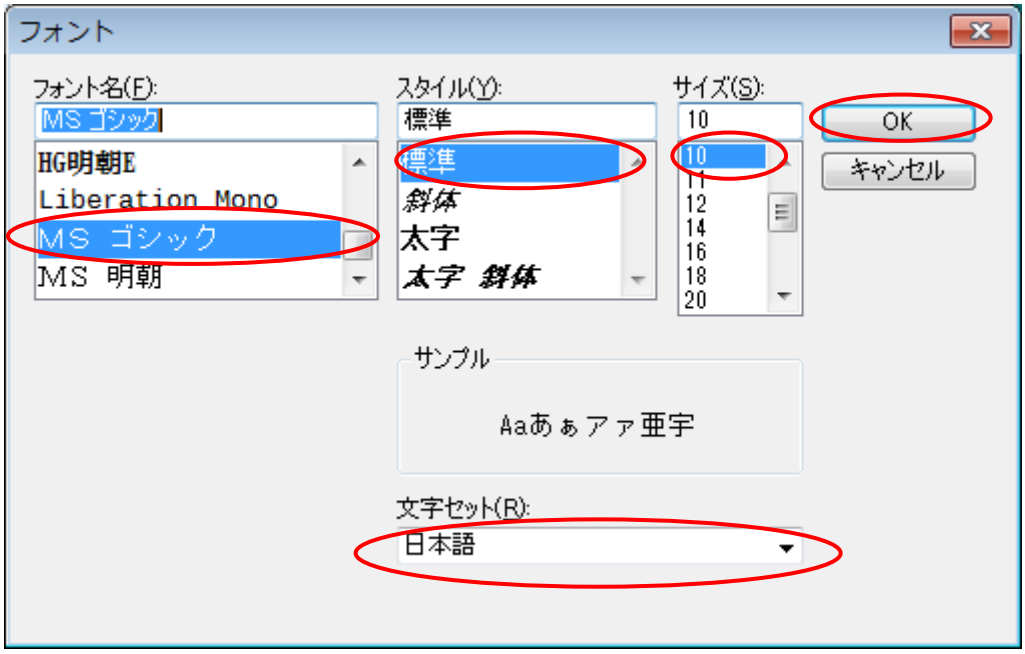


図１－１１　フォントを日本語に指定する

第２章　初めてのプログラム

この章では，NXT ブロックだけを使って動作させるプログラムを取り扱います。

２．１　プログラムの入力

それでは，実際にプログラム（Program2-1）を入力してみましょう。初めてプログラムを入力するときに注意する点は，次の２つです。

※　プログラムを入力するときに注意しなければならないこと　※

① 半角英数で入力する。全角は特別な場合しか使うことができません。

② 大文字と小文字を区別する。

プログラムは「{」と「}」で表される「中カッコ」でくくられている部分をひとまとまりとして書きます。このひとまとまりを分かりやすくするために，半角の空白を４つ入れて段付けします。ここで，半角の空白がわかりやすいように，␣と表記してあります。␣はスペースバーを押して半角の空白を入力します。また，左側にある数字は，説明用の行番号なので，この数字も入力しません。

まず最初に，点と線を使って図２－１のような顔を描いて見ましょう。

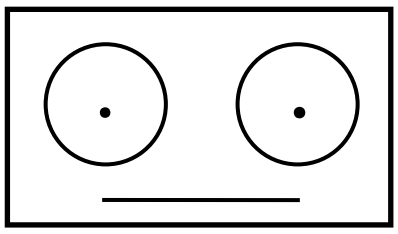


図２－１　顔のサンプル

【Program2-1】

```
1 task␣main()
2 {
3 ␣␣␣␣PointOut(20,40);
4 ␣␣␣␣PointOut(80,40);
5 ␣␣␣␣CircleOut(20,40,15);
6 ␣␣␣␣CircleOut(80,40,15);
7 ␣␣␣␣LineOut(20,10,80,10);
8 ␣␣␣␣while(true);
9 }
```

プログラムを入力すると画面には図２－２のように表示されます。エディッタの設定によっては，プログラムの内容に応じて自動的に文字の色が「青色」や「赤色」になります。また，これから文字を入力できる位置は，画面左下に行番号と列番号（図２－２では 9：2）で表示されています。

このプログラムに「Program2-1」という名前をつけて保存しましょう。メニューバーにある「File（ファイル)」をクリックし，「Save（保存)」をクリックします。図２－３に示すように保存するフォルダの指定とファイルの名前の入力ができるようになります。「ファイル名」の右横にある長方形の枠内に「Program2-1.nxc」と入力します。最後の「.nxc」は，このファイルの内容を NXC のプログラムであることを示す拡張子ですから，消さないように注意しましょう。Windows の設定によっては，ファイルの一覧で「.nxc」が表示されないこともあります。ファイルに名前が付くと，図２－４のようにプログラムのウィンドウの上に「Program2-1」と表示されるようになります」。

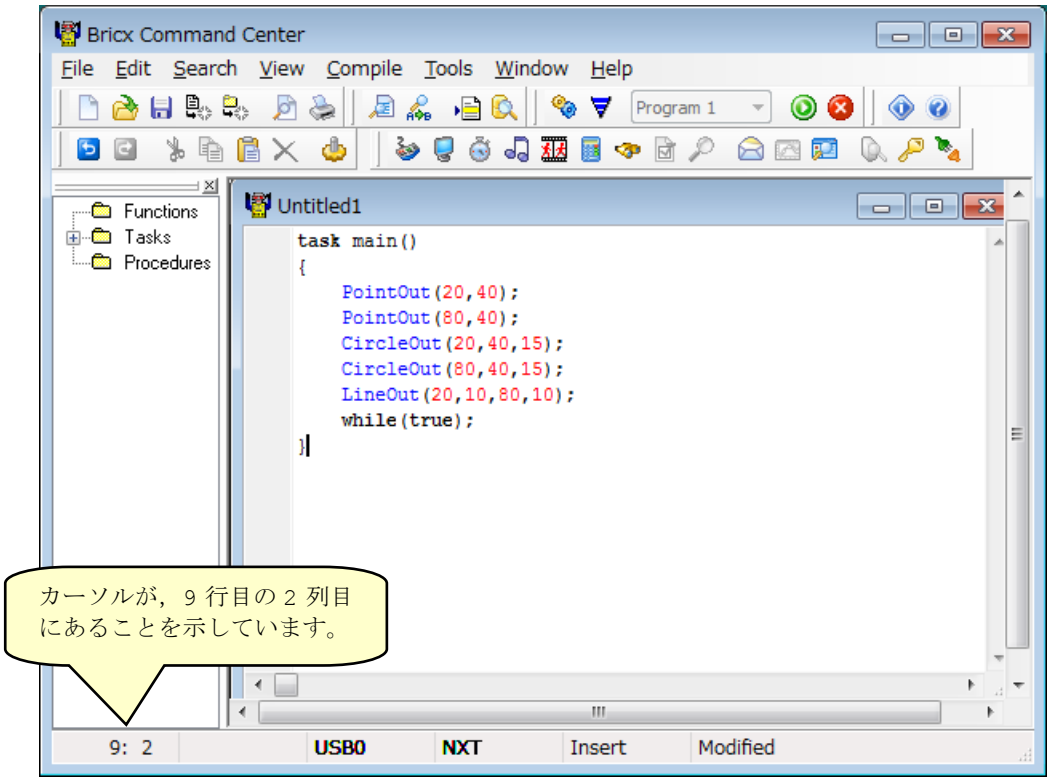


図２－２　プログラムを入力した状態

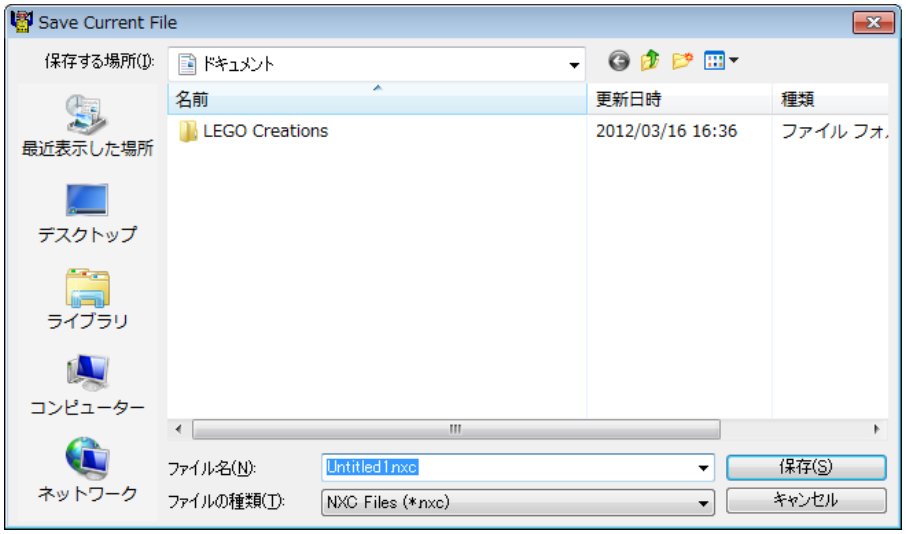


図２－３　保存するフォルダを指定して，プログラムに名前をつけて保存

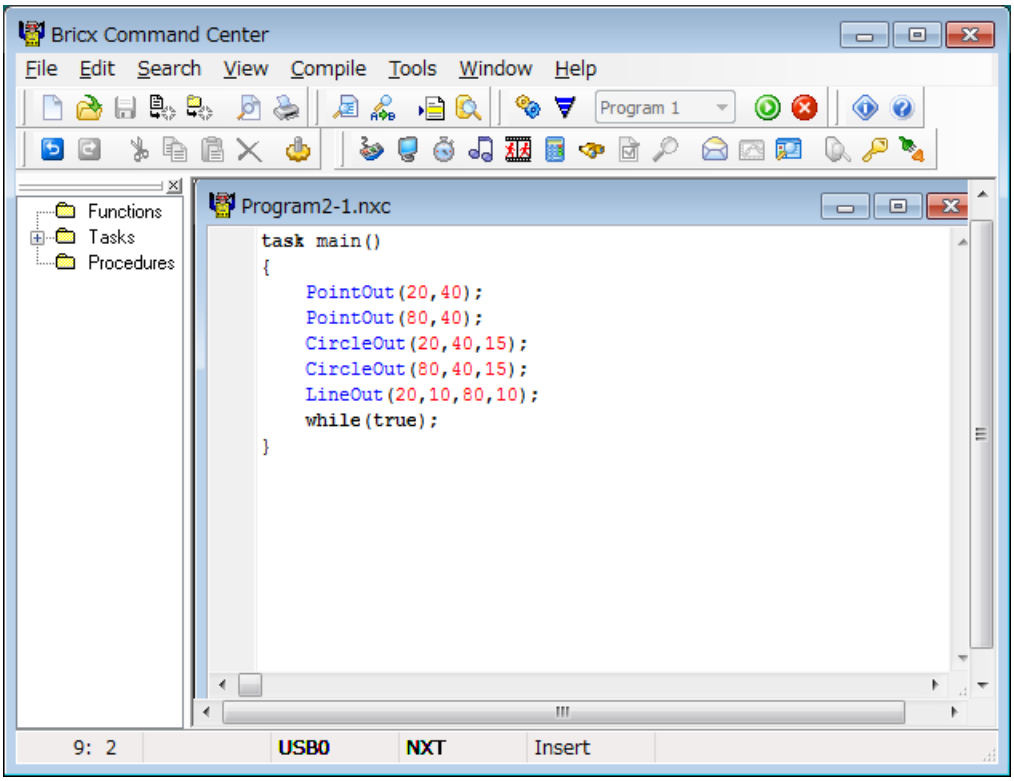


図２－４　Program2-1 という名前のプログラム

２．２　プログラムの説明

Program2-1 は, NXT ブロックの液晶ディスプレイに顔の絵を表示するプログラムです。このプログラム例には，プログラムを作成する上で必ず使う命令がいくつか入っています。

それでは，プログラムの意味を説明していきます。

1 行目　task main()

プログラムを作成するときに必ず使う「書き方」です。プログラムは task という命令の集まりからできています。このプログラムでは main という名前の task を含みます。task の後ろには，さまざまな名前をつけることができますが，プログラムの実行開始を示す main という名前の task はプログラムの中に必ず書いておかなければなりません。

task はいくつかの命令文から構成され，「;（セミコロン）」で区切られています。このプログラムでは main という名前の task から実行されます。main の中身は, 2 行目の「{」から 9 行目の「}」までに書かれた命令です。

ここで，3～8 行目のプログラムを理解するために，知っておかなければならないことを説明します。数学で学習した座標の考え方を思い出しましょう。

図１－１で紹介した液晶ディスプレイは，図２－５のような座標軸と対応します。X 座標の値は，0 から 99 まで，Y 座標の値は 0 から 63 までです。X 座標と Y 座標の値を組み合わせて位置を指定し，点や線，円などを描きます。

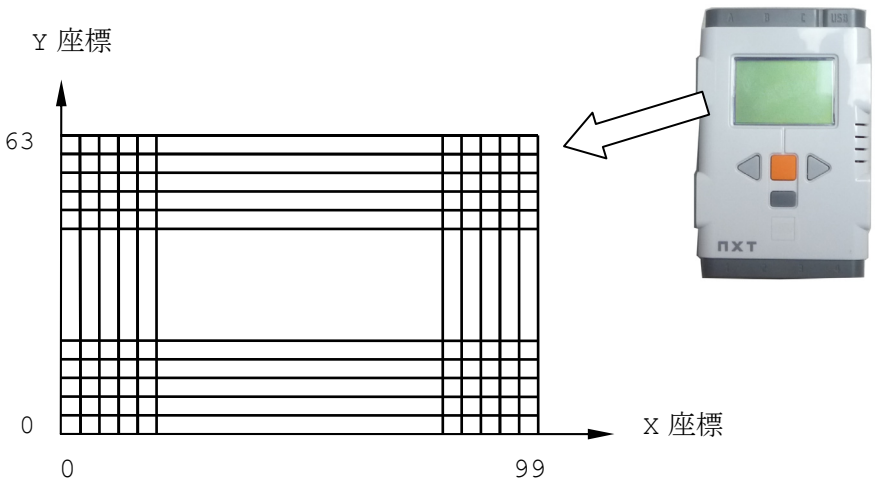


図２－５　液晶ディスプレイの座標

3 行目　PointOut(20,40);

Point（点）を描くという命令です。(20,40)は，(X 座標の値，Y 座標の値)を表し，x=20，Y=40 の位置に点が描かれます。

4 行目　PointOut(80,40);

3 行目と同じように，x=80，y=40 の位置に点を描きます。3 行目と 4 行目で描かれる点の Y 座標は，どちらも 40 と同じ値なので２つの点は横に並ぶことになります(図２－６)。

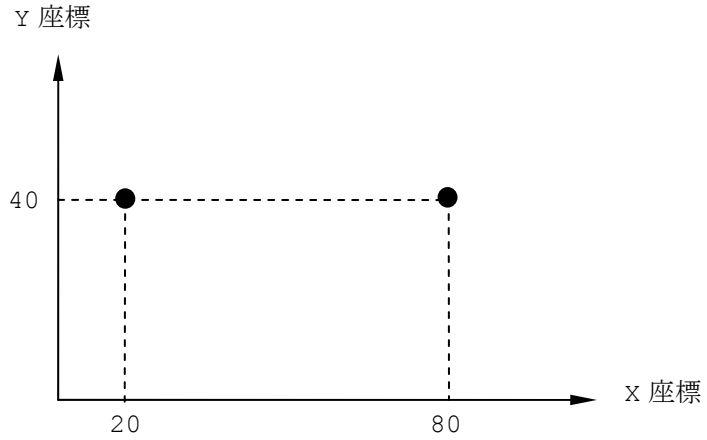


図２－６　液晶ディスプレイの表示（3，4 行目）

5 行目　CircleOut(20,40,15);

Circle（円）を描くという命令です。(20,40,15)は（円の中心の X 座標の値，円の中心の Y 座標の値，円の半径）を示しています。そのため，x=20，Y=40 を中心とした半径 15 の円が描かれます。ここで X，Y 座標の値が 3 行目と同じなことに気がつきましたか？つまり，この円は，3 行目で描いた点を中心とした円であるということになります。

6 行目　CircleOut(80,40,15);

5 行目と同じように，x=80，Y=40 を中心とした半径 15 の円が描かれます。X，Y 座標の値が 4 行目と同じなので，4 行目で描いた点を中心とした円になります（図２－７)。

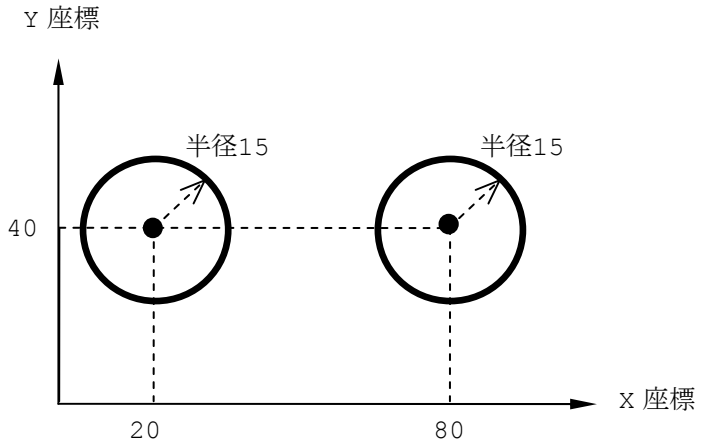


図２－７　液晶ディスプレイの表示（3～6 行目）

7 行目　LineOut(20,10,80,10);

Line（線）を描くという命令です。(20,10,80,10)は（線を開始する X 座標の値，線を開始する Y 座標の値，線を終了する X 座標の値，線を終了する Y 座標の値）を示しています。x=20，Y=10 の点から x=80，Y=10 までの点を結ぶように線を描きます。Y 座標の値が同じなので，図２－８に示すように水平な線が描かれます。

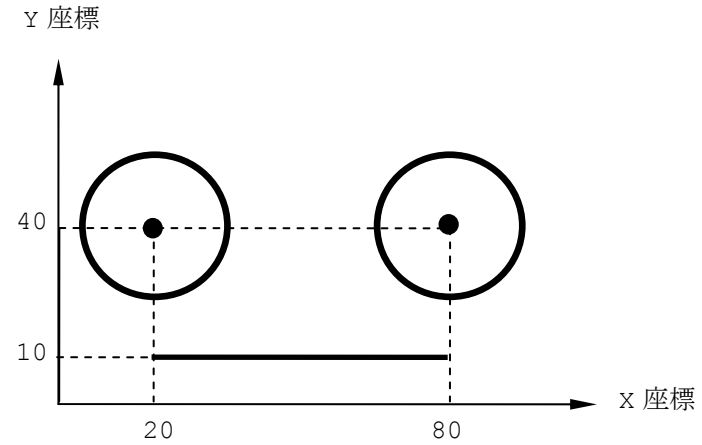


図２－８　水平な線を描いた状態（7 行目）

8 行目　while(true);

プログラムの実行が終了すると自動的に液晶ディスプレイを消してしまうので，終了しないように，プログラムを空回り状態で実行し終わらないようにしています。

このように，プログラムに使われている各命令には意味があり，その内容を簡単な英語で書き表しています。

２．３　プログラムのコンパイル・ダウンロード・実行

Program2-1 を実行してみます。プログラムを実行するには次の手順で行います。

①　コンパイル

NXC のプログラムは，人間にとってわかりやすくなっていますが，NXT 本体にとっては，それを理解することは大変難しいのです。そのため，NXC で書かれたプログラムを NXT が理解できる命令コードを示す数値データに変換する必要があります。このことを「コンパイル」といいます。

NXC のプログラムのコンパイルをするときは，図２－９に示すメニューバーの「Compile（コンパイル）」をクリックし，さらに「Compile（コンパイル）  」をクリックします。

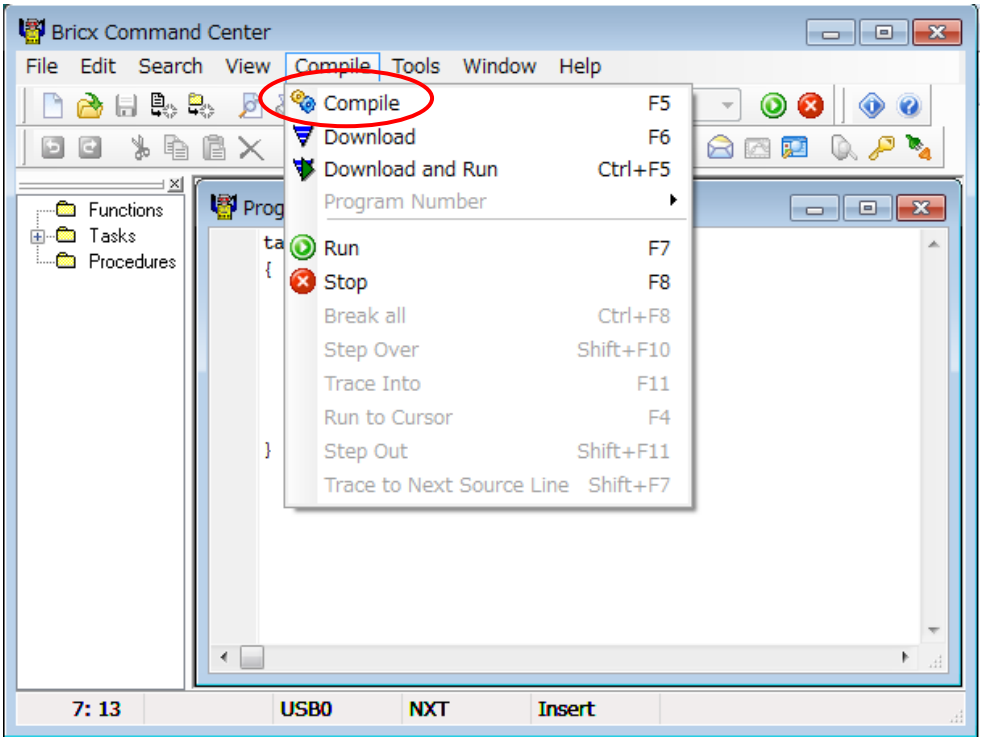


図２－９　プログラムのコンパイル

プログラムに間違い（「エラー」といいます。）がないときは，画面に変化は起こりません。これで，コンパイルは成功です。プログラムにエラーがあると，コンパイルを行うことができません。プログラムの間違いを修正します。「２．４　プログラムにエラーが出たら」を読んで修正しましょう。

② ダウンロード

次に，コンパイルされたデータをパソコンから NXT ブロックにダウンロードします。NXT の電源が入っていることと，パソコンと NXT が USB ケーブルで接続されていることを確認してから，データをダウンロードします。図１－１と図１－３を見て，正しく接続されているか確認しましょう。

図２－１０に示すようにメニューバーの「Compile（コンパイル)」をクリックし，さらに「Download（ダウンロード)，」をクリックします。

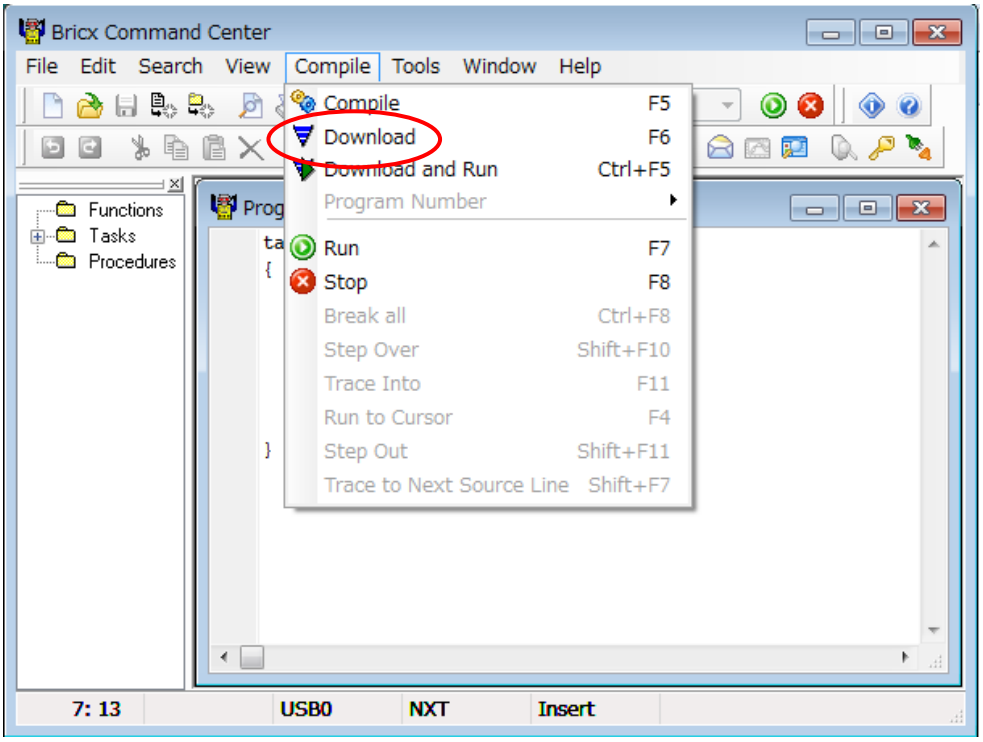


図２－１０　プログラムのダウンロード

もし，図２－１１のように「Download（ダウンロード)」の表示が影のようになっていたり，が白黒だったりして，ダウンロードができない場合は，NXT の電源が入っていることと，パソコンと NXT が USB ケーブルで正しく接続されていることを再度確認しましょう。

それでも，ダウンロードができない場合は，図２－１２に示すように「Tools（ツール)」をクリックし，さらに「Find Brick（NXT ブロックを探す)」をクリックします。すると，図２－１３のようなウィンドウが開くので，「Port（接続元)」が「usb」，「Brick Type（ブロックの型)」が「NXT」，「Firmware（あらかじめ組み込まれたソフトウェア)」が

「Standard（標準)」になっていることを確認し，OK ボタンをクリックします。これで「Download（ダウンロード)」が表示さて，  がカラー表示になって，ダウンロードできるようになります。

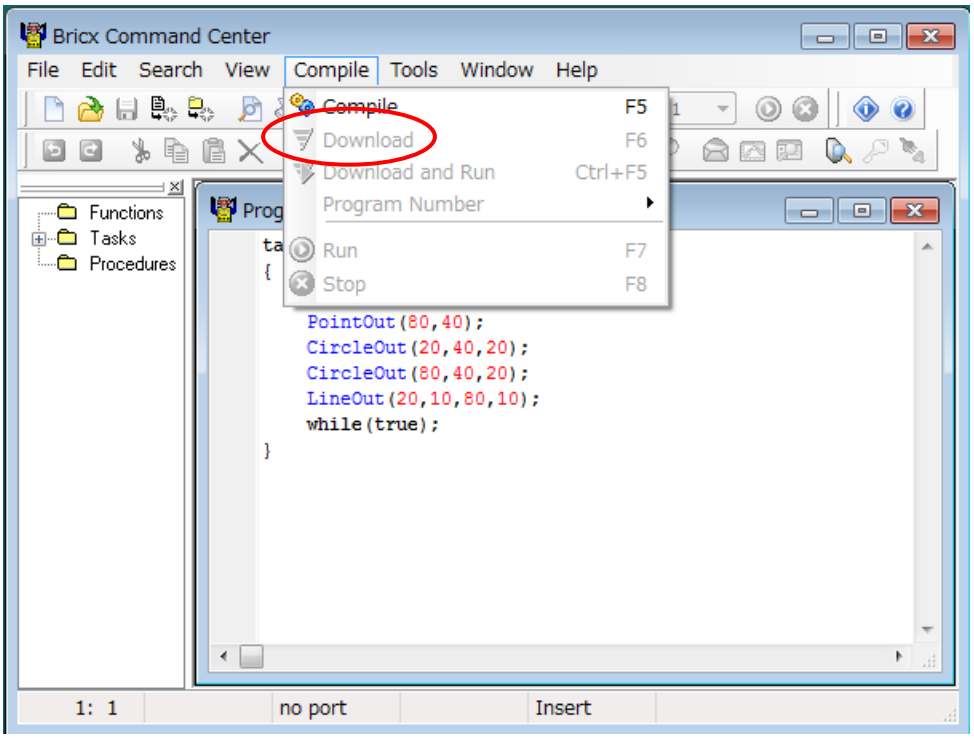


図２－１１　「Download（ダウンロード)」できない表示

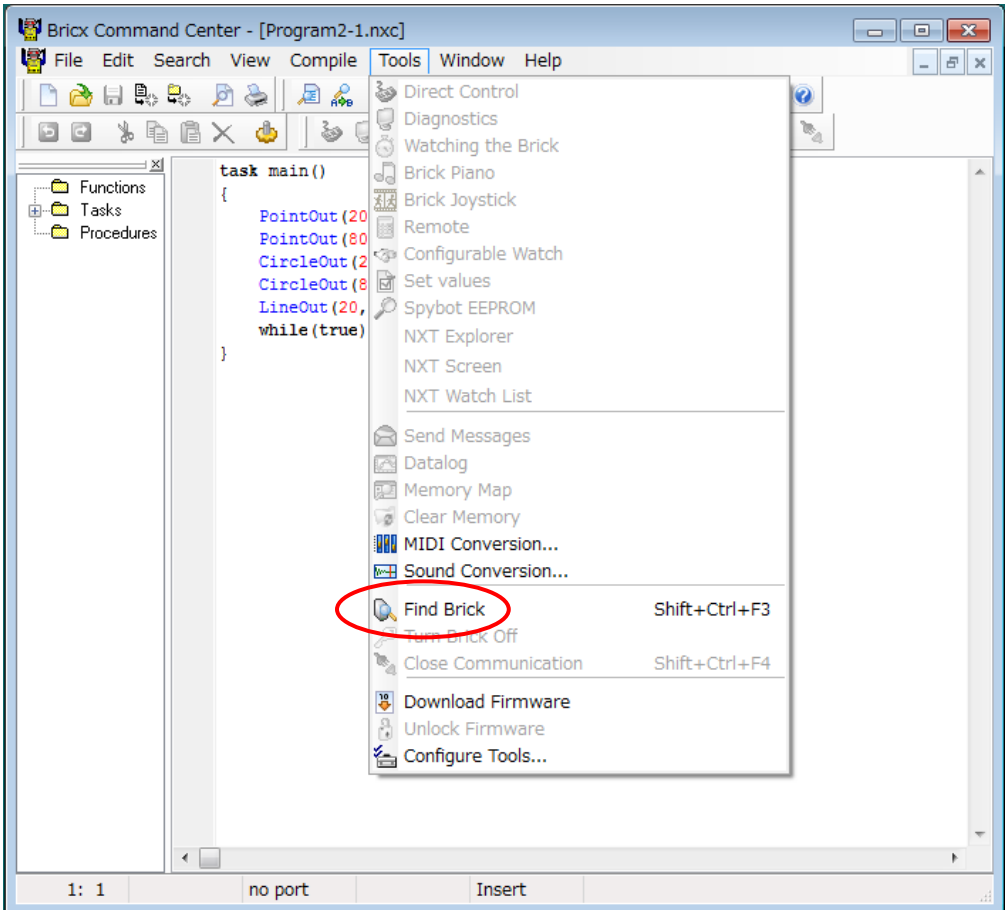


図２－１２　NXT ブロックを探す

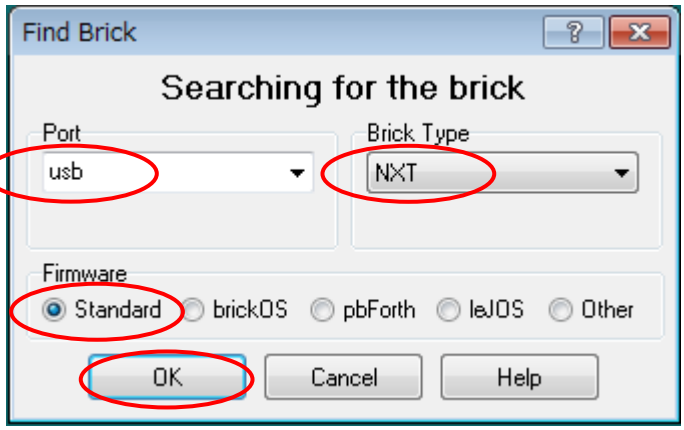


図２－１３　パソコンと NXT ブロックの接続方法

ダウンロードを実行すると，NXT ブロックから「プッ」という音がでて，成功を確認できます。このときに，コンパイルされたプログラムのデータがパソコンから USB ケーブル

を通って NXT ブロックにダウンロードされています。

③　実行

プログラムを実行するには，図１－１に示す NXT ブロックの NXT ボタン（オレンジ色）を押します。NXT ボタンを押すごとに液晶ディスプレイの表示が「My Files（自分のファイル）」→「Software files（ソフトウェアのファイル）」→「Program2-1」→「Program2-1　Run（Program2-1 の実行）」のように表示されていきます。

または，BricxCC のメニューバーの下にある  をクリックすることでも実行することができます。この方法を使うと直前にダウンロードされたプログラムが自動的に選択されるので便利です。

NXT ブロックの液晶ディスプレイに，図２－１４のような顔が表示されましたか？

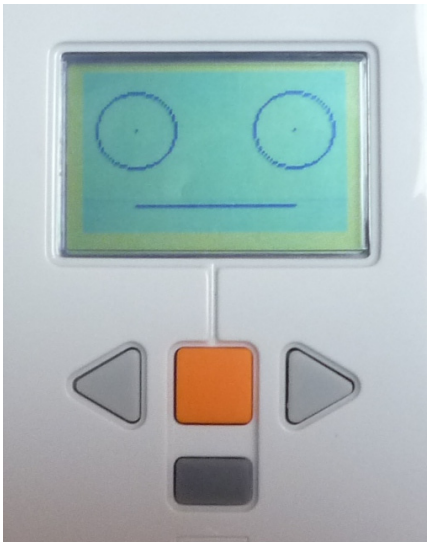


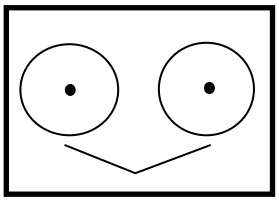
図２―１４　Program2-1 の実行結果

【チャレンジ２－１】

Program2-1 の 5 行目と 6 行目の「CircleOut」の３つ目の数値（円の半径を表す数値）を変えてみましょう。思い通りの目になりましたか？

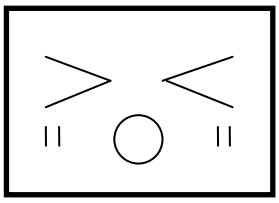
【チャレンジ２－２】

Program2-1 の 7 行目以降を工夫して，下図のような顔を液晶ディスプレイに表示してみましょう。



【チャレンジ２－３】

下図のような顔を液晶ディスプレイに表示してみましょう。



２．４　プログラムにエラーが出たら

コンパイルを実行したときに，図２－１５に示すような「Error（エラー)」メッセージが表示されるとプログラムの中に間違いがあります。この間違いのことを「エラー」といいます。コンパイルを行うと，ウインドウの中央に小ウインドウ（①）が開き，間違いの原因となる行の文字列が青く指定されます。また，ウインドウの下に表示される青い行の文字列は，エラーの原因を解説してくれています。

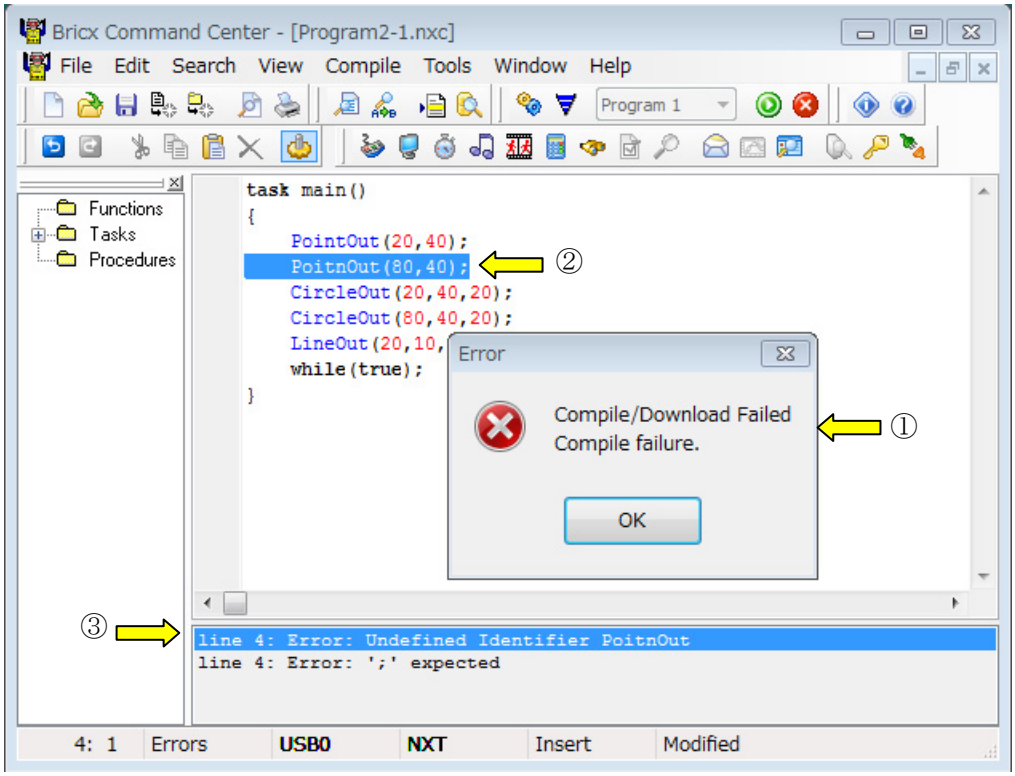


図２－１５　コンパイル中にエラーが発生

この例では，4 行目の英語のつづりが，PoitnOut と間違っています（②)。PoitnOut を PointOut に修正すると正しくコンパイルできます。ウインドウの下（③）に表示された文字列には，「line 4: Error:Undefined Identifier PoitnOut（4 行目：エラー：PoitnOut という単語は認識できません)」と書いてあります。

図２－１６の例では，6 行目の Circleout(80,40,20)のあとに「;（セミコロン)」を付け忘れています（②)。6 行目の最後に「;」を付け加えると正しくコンパイルできます。ウインドウの下（③）に表示された文字列には，「line 7: Error:";"expected（7 行目：エラー：“；”を入力してください)」と書いてあります。このように，指摘された行

の前の行に間違いがあるときもあります。指摘された行だけではなく，その前後にも間違いがないかよく確認してプログラムのエラーを修正しましょう。

　コンパイルしてエラーが出なくなったら，プログラムに間違いがないかもう一度よく確認してからダウンロードして実行しましょう。

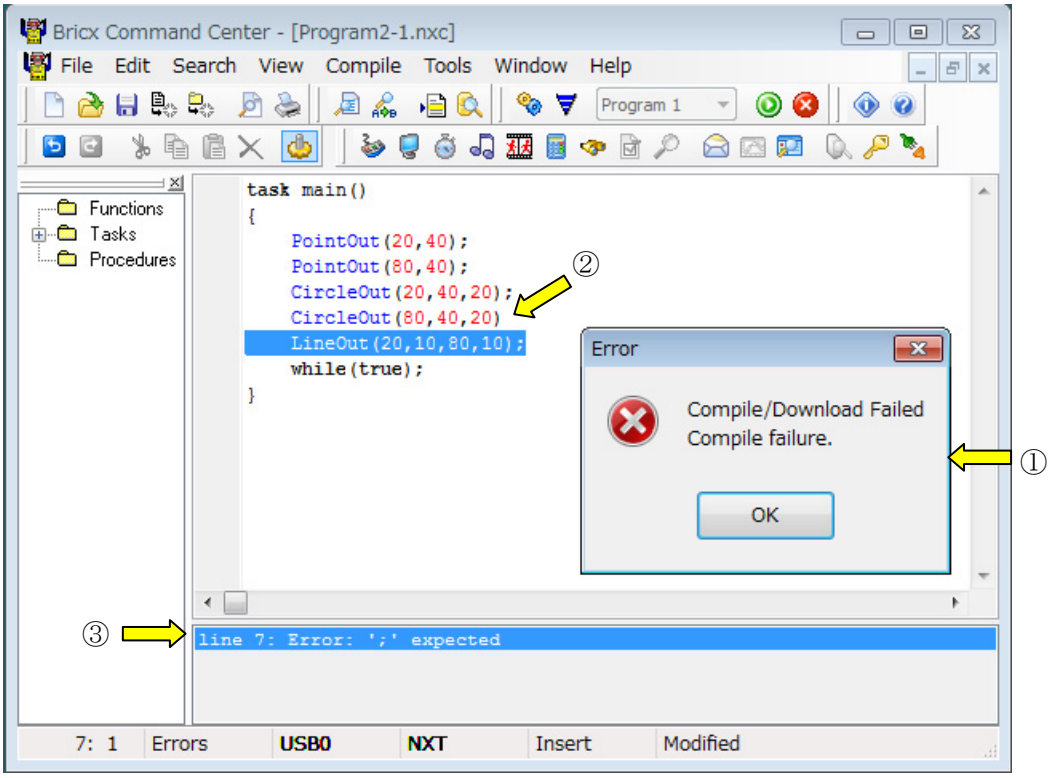


図２－１６　間違ったプログラムの例とエラーメッセージ

２．５　プログラムの保存とプログラムを開く

プログラムを保存するときは，図２－１７に示すようにメニューバーの「File(ファイル)」をクリックし，「Save（保存)」をクリックします。

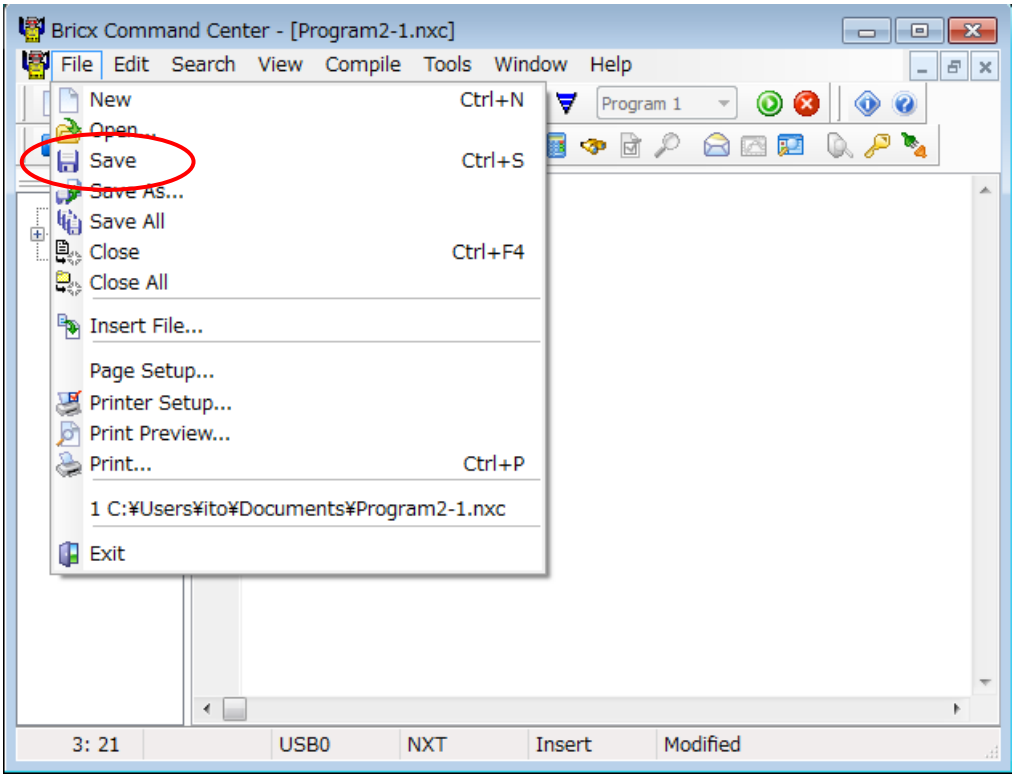


図２－１７　プログラムの保存

すでにプログラムに名前をつけて保存しているときは，上書き保存されます。まだ，名前を付けていないときは，図２－３のような保存するフォルダと，保存するファイルの名前を入力する画面が表示されます。

ファイルの名前は，「半角英数のみ」を使ってつけるようにします。日本語を使うことはできません。

「保存」ボタンをクリックしてファイルを保存すると,「ファイルの名前.nxc」というファイルができます。例えば，Program2-1 という名前で保存すると Program2-1.nxc というファイルで保存されています。

すでに保存したプログラムを開くには，図２－１８に示すようにメニューバーの「File（ファイル)」をクリックし，さらに「Open（ファイルを開く）  」をクリックします。

すると，図２－１９のようなウィンドウが開くので，保存していたフォルダを選択して「ファイルの名前.nxc」というファイルを選択して「開く」をクリックするとファイルを開くことができます（図２－１９）。

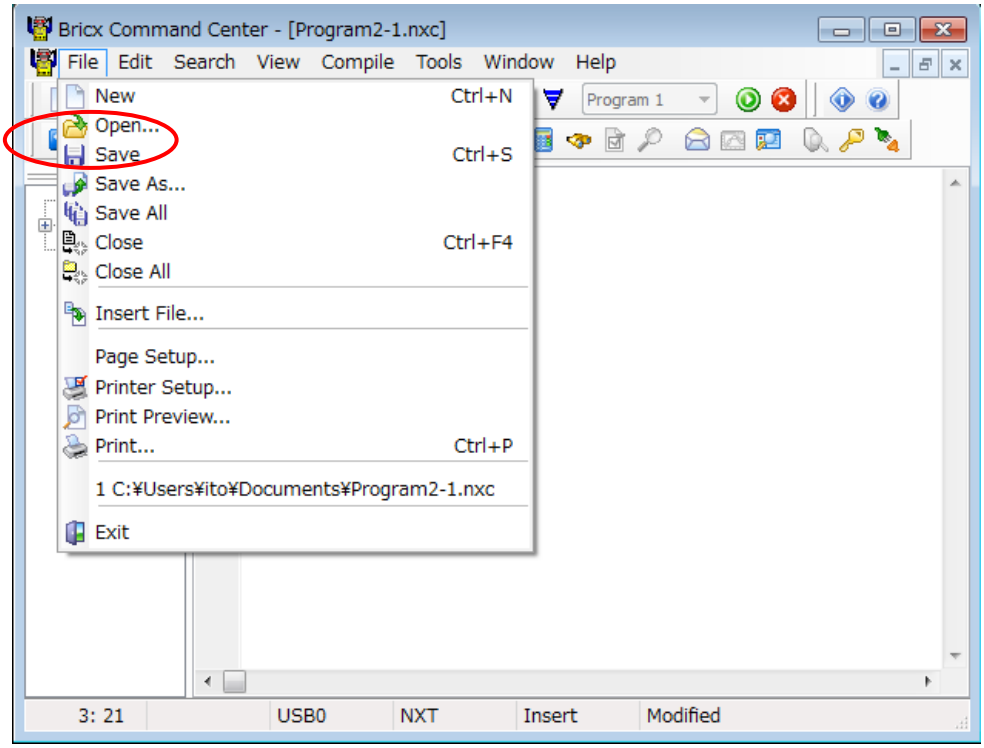


図２－１８　保存していたプログラムを開く

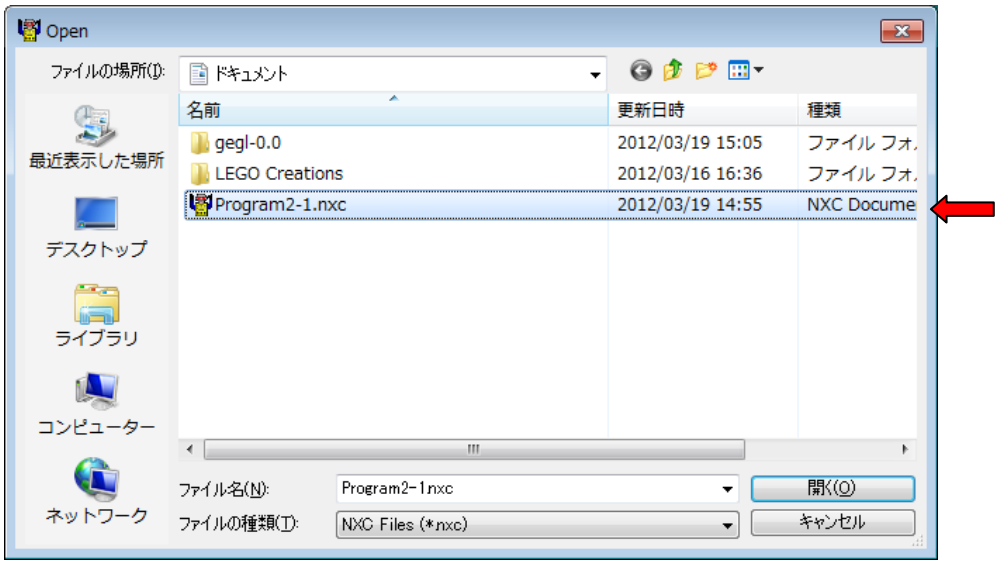


図２－１９　保存していたフォルダと，開くプログラムの名前を選択

第３章　初めてのセンサ

まわりの状態を知るための仕組み（計測機能）をもつものを「センサ」といいます。温度や明るさ，音など様々な状態を知るセンサがありますが，この章では，何かに触れたことを知るためのセンサである「タッチセンサ」について学びます。

３．１　タッチセンサの仕組み

ここでは，NXT ブロック用タッチセンサ（図３－１）を使います。タッチセンサは，小さな押しボタンスイッチで接触を検知します。オレンジ色のスイッチ部分を押したり離したりすることで触れている状態を知ります。

タッチセンサは,NXT ブロックと専用のケーブルを使って図３－２のように接続します。４つある入力ポート（図１－１）のうち，一番左端の入力ポート１番に接続できているか確認しましょう。

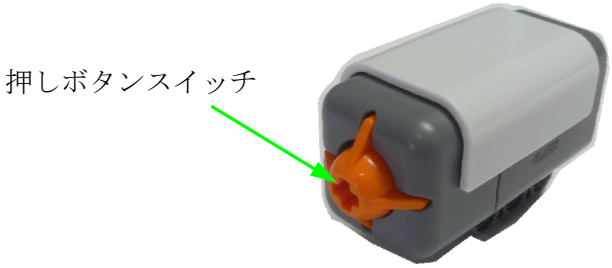


図３－１　NXT ブロック用タッチセンサ

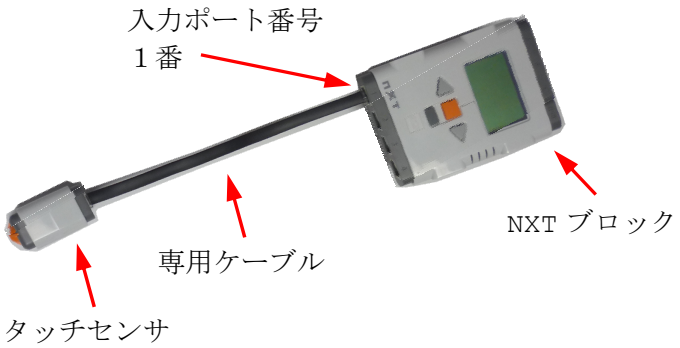


図３－２　タッチセンサと NXT ブロックの接続

３．２　触れたことを感知するプログラム

何かに触れたことを感知するプログラム（Program3-1）を入力しましょう。

【Program3-1】

```
 1 task main()
 2 {
 3     SetSensorTouch(IN_1);
 4     PointOut(20,40);
 5     PointOut(80,40);
 6     CircleOut(20,40,15);
 7     CircleOut(80,40,15);
 8     until(Sensor(IN_1)==1);
 9     LineOut(20,10,80,10);
10     while(true);
11 }
```

コンパイルして，NXT ブロックにダウンロードした後，「Program3-1」を選択して，実行しましょう。液晶ディスプレイに２つの「目」が表示されます。タッチセンサのスイッチを押すと，「口」が表示されて顔が完成します。

Program3-1 は，NXT ブロックの液晶ディスプレイに表示された未完成な顔を，タッチセンサを押すことによって，完成させるプログラムです。それでは新たに加わったプログラムを細かく説明していきます。

3 行目　SetSensorTouch(IN_1);

SetSensorTouch は「入力ポート１番に接続したセンサをタッチセンサである。」ということを NXT ブロックに知らせる命令です。IN_1 は，入力ポート１番を示しています。

なお，入力ポートとして２番を使うときは，IN_2 と書きます。同様に入力ポート３番，４番のときは，IN_3，IN_4 とします。

4 行目から 7 行目までのプログラムは，「２．２ プログラムの説明」で紹介したように「点」と「丸」で表現される２つの「目」を描きます。

8 行目　until(Sensor(IN_1)==1);

入力ポート１番に接続されたタッチセンサのスイッチが押されるまで待つ命令です。タッチセンサのスイッチが押されると，9 行目に実行が移ります。

until は，その次のカッコの中の状態が「正しく」なるまで，待ちなさいという命令です。Sensor(IN_1)は，入力ポート１の状態を示します。「==」は，等しいかどうかを示します。その次の「1」は，タッチセンサのスイッチが押された状態の値を示します。until の次のカッコの中の「Sensor(IN_1)==1」は，「入力ポート１番に接続されたタッチセンサのスイッチが押されたら正しい」という意味となります。

そのため「until(Sensor(IN_1)==1)」は，入力ポート１番に接続されたタッチセンサのスイッチが押されるまで待つ」処理を行います。

9 行目のプログラムは，「２．２　プログラムの説明」で紹介したように「口」を表現するために「線」を描く命令です。

３．３　タッチセンサの使い方

Program3-1 で表示される顔を，まず，左目だけ表示して，タッチセンサのスイッチを押すと，右目が表示され，そのスイッチを離すと，「口」が描かれるようにしてみましょう。

この動作をするプログラムを Program3-2 に示します。6 行目と 10 行目で until 文を使っていますが，スイッチの状態を示す数値が異なることに注意しましょう。

また，プログラムの処理内容のまとまりがよくわかるように，7 行目と 11 行目に「空白行」を入れています。「空白行」はプログラムの実行には無関係ですが，処理のまとまりがよくわかるようになるので，よく使われます。

【Program3-2】

```
 1 task main()
 2 {
 3     SetSensorTouch(IN_1);
 4     PointOut(20,40);
 5     CircleOut(20,40,15);
 6     until(Sensor(IN_1)==1);
 7
 8     PointOut(80,40);
 9     CircleOut(80,40,15);
10     until(Sensor(IN_1)==0);
11
12     LineOut(20,10,80,10);
13     while(true);
14 }
```

【チャレンジ３－１】

入力ポート２番にタッチセンサを接続したときは，Program3-1 をどのように修正すればよいでしょうか？

第４章　サウンド

NXT ブロックはスピーカーを内蔵しています。このスピーカーを使ってサウンドを鳴らしたり簡単な音楽を演奏したりできます。また，サウンドを使うことでプログラムがどのような動作を行っているのかを知らせることができます。

４．１　組み込みサウンド

NXT ブロックには表４－１のような３つのサウンドがあらかじめ組み込まれています。このサウンドは，PlayFileEx（サウンドファイルの再生）という命令を使ってサウンドのファイル名を指定して鳴らすことができます。

表４－１　組み込みサウンド

NXT ブロック内のファイル名（␣は半角スペース）	サ　ウ　ン　ド　の　内　容
!␣Startup.rso	NXT ブロックの電源をオンにしたときの音（タラララララという音）
!␣Click.rso	NXT ブロックのボタンを押したときの音
!␣Attention.rso	ピロロロという機械音

PlayFileEx 命令は，サウンドが鳴り終わるまで待たず，すぐに次の命令を実行してしまいます。そのため，次々にサウンドが再生されてしまうので，PlayFileEx の後，サウンドの再生時間分よりも長い時間だけ「Wait」命令を使って，時間待ちしなければなりません。

PlayFileEx の次のカッコの中は，（サウンドファイル名，音量，繰り返しの有無）の順に指定します。

「サウンドのファイル名」は，二重引用符「"」と「"」でくくります。

「音量」は，「0（無音)」から「4（最大)」の５段階を設定します。

「繰り返しの有無」は，「0（繰り返しなし)」または「1（繰り返しあり)」です。

Program4-1 は，表４－１の組み込みサウンドを順番に鳴らしていくプログラムです。すべて「音量」は「3」，「繰り返しの有無」は「0（繰り返しなし)」になっています。

【Program4-1】

```
1  task main()
2  {
3  ␣␣␣␣PlayFileEx("!␣Startup.rso",3,0);
4  ␣␣␣␣Wait(2000);
5  ␣␣␣␣PlayFileEx("!␣Click.rso",3,0);
6  ␣␣␣␣Wait(2000);
7  ␣␣␣␣PlayFileEx("!␣Attention.rso",3,0);
8  ␣␣␣␣Wait(2000);
9  }
```

ここで，Wait 命令を使った時間待ちについて説明します。

4，6，8 行目の Wait(2000)は，カッコの中に示された数値に対応する時間だけ待ち，次の命令に実行を移します。この数値の単位はティックと呼びます。1 ティックは 0.001 秒です。2000 ティックは，0.001×2000=2 秒間となります。

このプログラムでは，3，5，7 行目に「組み込みサウンドを音量 3，繰り返しなしで鳴らす」という命令を実行した後，それぞれのサウンドが鳴っている間より長く待つように，2000 ティック（2 秒間）待つようになっています。

もし，音の出ないサウンドがある場合は，そのサウンドファイルが NXT ブロックの中に記憶されていません。そのときは，ファームウェアをダウンロードし直してください。

【チャレンジ４－１】

Program4-1 のサウンドファイルを鳴らすときの「音量（0～4)」や「繰り返しの有無（0，1)」の数値をいろいろ変えてみましょう。

４．２　音楽の演奏

音楽は，音の高さや音量，鳴らす長さで決められる単音と休みの長さを組み合わせてできています。単音を鳴らすための命令として PlayTone と PlayToneEx があります。この命令を組み合わせていろいろな音楽を演奏するプログラムを作ってみましょう。

単音は，PlayTone 命令に続く(　)カッコの中に，音の高さ，鳴らす時間の順に指定します。音の高さは周波数という数値で指定します。この数値が大きいほど高い音となり，小さいほど低い音になります。鳴らす時間の単位は Wait 命令と同じように 0.001 秒（1 ティック）単位で指定します。PlayTone 命令では音量を指定できません。あらかじめ NXT ブロックのメニュー画面で設定した音量で鳴ります。

音の高さを指定する数値と音階の対応を表４－２に示します。NXT ブロックでは，８オクターブの範囲で音を鳴らすことができます。聴きやすい音楽を作るときは，音の高さが 440 で指定される「ラ」を基準とします。

表４－２　音の高さ（周波数）を指定する数値

オクターブ	1	2	3	4	5	6	7	8
ド	33	65	131	262	523	1047	2093	4186
ド＃	35	69	139	277	554	1109	2217	4435
レ	37	73	147	294	587	1175	2349	4699
レ＃	39	78	156	311	622	1245	2489	4978
ミ	41	82	165	330	659	1319	2637	5274
ファ	44	87	175	349	698	1397	2794	5588
ファ＃	46	92	185	370	740	1480	2960	5920
ソ	49	98	196	392	784	1568	3136	6272
ソ＃	52	104	208	415	831	1661	3322	6645
ラ	55	110	220	440	880	1760	3520	7040
ラ＃	58	117	233	466	932	1865	3729	7459
シ	62	123	247	494	988	1976	3951	

Program4-2 は,「ド，レ，ミ，ファ，ソ，ラ，シ」と，0.4 秒間（400 ティック）ずつ鳴らし，音と音の間で 0.1 秒間（500-400=100 ティック）休むプログラムです。プログラム中の「//　ド」などは，注釈です。プログラムを分かりやすくするために書いてあるものなので，入力しなくてもかまいません。

【Program4-2】

```
 1 task main()
 2 {
 3     PlayTone(262,400); Wait(500); // ド
 4     PlayTone(294,400); Wait(500); // レ
 5     PlayTone(330,400); Wait(500); // ミ
 6     PlayTone(349,400); Wait(500); // ファ
 7     PlayTone(392,400); Wait(500); // ソ
 8     PlayTone(440,400); Wait(500); // ラ
 9     PlayTone(494,400); Wait(500); // シ
10 }
```

PlayTone 命令は音を鳴らし始めると，すぐに次の命令を実行してしまいます。そのため，PlayTone 命令で指定した鳴らす時間だけ，Wait 命令を入れて待つようにしなければなりません。

PlayTone 命令では音量を設定できませんでした。音量も指定して単音を鳴らすことのできる命令として PlayToneEx があります。

PlayToneEx 命令に続く(　)カッコの中で，音の高さ，鳴らす時間，音量，繰り返しの有無を順に指定します。「音の高さ」と「鳴らす時間」は，PlayTone 命令と同じで，それぞれ「周波数」と「0.001 秒単位の値」で指定します。「音量」は「0（無音)」から「4（最大)」の５段階を設定します。「繰り返しの有無」は「0（繰り返しなし)」または「1（繰り返しあり)」です。

Program4-3 は,「ド」の音量が次第に大きくなっていくプログラムです。これは,PlayToneEx 命令の３番目に指定した音量の値が段々と大きくなっていることから分かります。４番目の数値はいずれも「0（繰り返しなし)」なので，単音になっています。

【Program4-3】

```
 1 task main()
 2 {
 3     PlayToneEx(262,400,0,0); Wait(500);
 4     PlayToneEx(262,400,1,0); Wait(500);
 5     PlayToneEx(262,400,2,0); Wait(500);
 6     PlayToneEx(262,400,3,0); Wait(500);
 7     PlayToneEx(262,400,4,0); Wait(500);
 8 }
```

Program4-4 は,「ドレミファソドレミファソ」という音が出るプログラムですが，よく聴いてみると１回目の「ドレミファソ」と２回目の「ドレミファソ」とでは，違って聞こえてきます。プログラムをよく見ると，１回目の「ドレミファソ」は「0（繰り返しなし)」,２回目の「ドレミファソ」は「1（繰り返しあり)」になっています。音量はいずれも「3」で同じとなっています。

【Program4-4】

```
 1 task main()
 2 {
 3     PlayToneEx(262,400,3,0); Wait(500);
 4     PlayToneEx(294,400,3,0); Wait(500);
 5     PlayToneEx(330,400,3,0); Wait(500);
 6     PlayToneEx(349,400,3,0); Wait(500);
 7     PlayToneEx(392,400,3,0); Wait(500);
 8
 9     PlayToneEx(262,400,3,1); Wait(500);
10     PlayToneEx(294,400,3,1); Wait(500);
11     PlayToneEx(330,400,3,1); Wait(500);
12     PlayToneEx(349,400,3,1); Wait(500);
13     PlayToneEx(392,400,3,1); Wait(500);
14 }
```

Program4-5 は，童謡「きらきら星」の曲の一部です。プログラムを実行して演奏させてみましょう。

【Program4-5】

```
 1 task main()
 2 {
 3     PlayTone(262,400); Wait(500);
 4     PlayTone(262,400); Wait(500);
 5     PlayTone(392,400); Wait(500);
 6     PlayTone(392,400); Wait(500);
 7     PlayTone(440,400); Wait(500);
 8     PlayTone(440,400); Wait(500);
 9     PlayTone(392,400); Wait(500);
10                        Wait(500);
11     PlayTone(349,400); Wait(500);
12     PlayTone(349,400); Wait(500);
13     PlayTone(330,400); Wait(500);
14     PlayTone(330,400); Wait(500);
15     PlayTone(294,400); Wait(500);
16     PlayTone(294,400); Wait(500);
17     PlayTone(262,400); Wait(500);
18 }
```

Program4-5 の音階をよりわかりやすくすることができます。音階の数値を「音名」で書いたプログラムを Program4-6 に示します。

【Program4-6】

```
 1 #define DO 262
 2 #define RE 294
 3 #define MI 330
 4 #define FA 349
 5 #define SO 392
 6 #define LA 440
 7
 8 task main()
 9 {
10     PlayTone(DO,400); Wait(500);
11     PlayTone(DO,400); Wait(500);
12     PlayTone(SO,400); Wait(500);
13     PlayTone(SO,400); Wait(500);
14     PlayTone(LA,400); Wait(500);
15     PlayTone(LA,400); Wait(500);
16     PlayTone(SO,400); Wait(500);
17                       Wait(500);
18     PlayTone(FA,400); Wait(500);
19     PlayTone(FA,400); Wait(500);
20     PlayTone(MI,400); Wait(500);
21     PlayTone(MI,400); Wait(500);
22     PlayTone(RE,400); Wait(500);
23     PlayTone(RE,400); Wait(500);
24     PlayTone(DO,400); Wait(500);
25 }
```

8 行目の task main()の前の 1～6 行目で，使用する音階を定義（define）することで，より簡単にプログラムを入力することができます。#define の次には，「音名」と「周波数」に対応する数値を書きます。プログラム中で「音名」を書くと，コンパイル時にそれが数値に置きかわります。

Program4-7 は，「さくらさくら」の曲を演奏するプログラムです。楽譜に合わせて PlayTone で指定する周波数や鳴らす時間，Wait 命令の時間が少しずつ異なることが分かります。

【Program4-7】

```
 1 #define SIL 247
 2 #define DO  262
 3 #define RE  294
 4 #define MI  330
 5 #define FA  349
 6 #define SO  392
 7 #define LA  440
 8 #define SI  494
 9 #define DOH 523
10
11 task main()
12 {
13     PlayTone(LA,400);  Wait( 500);
14     PlayTone(LA,400);  Wait( 500);
15     PlayTone(SI,800);  Wait(1000);
16
17     PlayTone(LA,400);  Wait( 500);
18     PlayTone(LA,400);  Wait( 500);
19     PlayTone(SI,800);  Wait(1000);
20
21     PlayTone(LA,400);  Wait( 500);
22     PlayTone(SI,400);  Wait( 500);
23     PlayTone(DOH,400); Wait( 500);
24     PlayTone(SI,400);  Wait( 500);
25     PlayTone(LA,400);  Wait( 500);
26     PlayTone(SI,200);  Wait( 250);
27     PlayTone(LA,200);  Wait( 250);
28     PlayTone(FA,800);  Wait(1000);
29
30     PlayTone(MI,400);  Wait( 500);
31     PlayTone(DO,400);  Wait( 500);
32     PlayTone(MI,400);  Wait( 500);
33     PlayTone(FA,400);  Wait( 500);
34     PlayTone(MI,400);  Wait( 500);
35     PlayTone(MI,200);  Wait( 250);
36     PlayTone(DO,200);  Wait( 250);
37     PlayTone(SIL,800); Wait(1000);
38
39     PlayTone(LA,400);  Wait( 500);
40     PlayTone(SI,400);  Wait( 500);
41     PlayTone(DOH,400); Wait( 500);
42     PlayTone(SI,400);  Wait( 500);
```

```
43      PlayTone(LA,400);  Wait( 500);
44      PlayTone(SI,200);  Wait( 250);
45      PlayTone(LA,200);  Wait( 250);
46      PlayTone(FA,800);  Wait(1000);
47
48      PlayTone(MI,400);  Wait( 500);
49      PlayTone(DO,400);  Wait( 500);
50      PlayTone(MI,400);  Wait( 500);
51      PlayTone(FA,400);  Wait( 500);
52      PlayTone(MI,400);  Wait( 500);
53      PlayTone(MI,200);  Wait( 250);
54      PlayTone(DO,200);  Wait( 250);
55      PlayTone(SIL,800); Wait(1000);
56
57      PlayTone(LA,400);  Wait( 500);
58      PlayTone(LA,400);  Wait( 500);
59      PlayTone(SI,800);  Wait(1000);
60
61      PlayTone(LA,400);  Wait( 500);
62      PlayTone(LA,400);  Wait( 500);
63      PlayTone(SI,800);  Wait(1000);
64
65      PlayTone(MI,400);  Wait( 500);
66      PlayTone(FA,400);  Wait( 500);
67      PlayTone(SI,200);  Wait( 250);
68      PlayTone(LA,200);  Wait( 250);
69      PlayTone(FA,400);  Wait( 500);
70      PlayTone(MI,1200); Wait(1500);
71  }
```

【チャレンジ４－２】

Program4-6 の童謡「きらきら星」の続きのメロディを入力して，曲を完成させましょう。

＜ヒント＞

ドドソソララソ　ファファミミレレド　ソソファファミミレ　ソソファファミミレ

ドドソソララソ　ファファミミレレド

第５章　モータ

この章からは，図１－２の移動型三輪ロボットを動かします。まず，NXT ブロックの正面から見て向かって左側のモータを出力ポートＢに接続し，右側のモータを出力ポートＣに接続していることを確認してください。

５．１　ロボットの前後移動

いよいよロボットを動かすところまでやってきました。ロボットを動かすためには，モータを駆動させる必要があります。早速，プログラム（Program5-1）を入力しましょう。

【Program5-1】

```
 1  task main()
 2  {
 3      OnFwd(OUT_B,30);
 4      OnFwd(OUT_C,30);
 5      Wait(4000);
 6      OnRev(OUT_BC,30);
 7      Wait(4000);
 8      Off(OUT_BC);
 9  }
```

Program5-1 はロボットを 4 秒間前進させ，その後 4 秒間後退させるプログラムです。それでは，新たに加わったプログラムを説明していきます。

3 行目　OnFwd(OUT_B,30);

OnFwd は On Forward（前進）を省略したもので，「モータを前進回転させなさい」という命令です。OnFwd の次にある(　)カッコの中で指定した「OUT_B」は出力ポートＢを示し，そこに接続されたモータが回転しロボットが前進します。「30」は，モータのパワー値を指定しています。モータのパワー値として 0～100 を指定でき，0 で停止状態，100 で最も大きなパワー値となります。

4 行目　OnFwd(OUT_C,30);

3 行目の OnFwd 命令と同じですが，「OUT_C」となっているので出力ポートＣに接続されたモータを前進回転させます。パワー値は「30」でロボットが前進します。

5 行目　Wait(4000);

Wait 命令はカッコ( )の中に示された数値だけ時間稼ぎをします。ここでは，直前に「出力ポートＢのモータを前進回転」と「出力ポートＣのモータを前進回転」という命令が実行されていますので，その状態を 4000 ティック (4 秒間) だけ保ちます。

6 行目　OnRev(OUT_BC,30);

OnRev は On Reverse（後退）を省略したもので，「モータを後進回転させなさい」という命令です。OnRev の次にある( )カッコの中で指定した「OUT_BC」は，出力ポートＢとＣに接続された２つのモータが同時に後進回転し，ロボットが後退することになります。このように，１つの命令で２つのモータを同時に回転させることもできます。また，モータのパワー値は，3，4 行目と同じ「30」です。

7 行目　Wait(4000);

ここでの Wait(4000)は，出力ポートＢとＣのモータを後進回転させる状態を 4 秒間続けることになります。

8 行目　Off(OUT_BC);

Off 命令は，指定した出力ポートに接続されたモータの回転を停止します。「OUT_BC」は，出力ポートＢとＣに接続されたモータを指定しています。

ここで，Program5-1 と次の２つのプログラム（Program5-2 と Program5-3）を見比べてみましょう。３つのプログラムの実行すると，それぞれのロボットの動作はほとんど同じように見えます。

【Program5-2】

```
 1 task main()
 2 {
 3     OnFwd(OUT_B,30);
 4     OnFwd(OUT_C,30);
 5     Wait(4000);
 6     OnRev(OUT_B,30);
 7     OnRev(OUT_C,30);
 8     Wait(4000);
 9     Off(OUT_B);
10     Off(OUT_C);
11 }
```

【Program5-3】

```
 1  task main()
 2  {
 3      OnFwd(OUT_BC,30);
 4      Wait(4000);
 5      OnRev(OUT_BC,30);
 6      Wait(4000);
 7      Off(OUT_BC);
 8  }
```

Program5-2 と Program5-3 とも，出力ポートＢとＣに接続されたモータはほぼ同時に回転し，ロボットを前進させた後，後進させています。

表５－１に示すようにモータの回転についての設定方法では，Program5-2 のように出力ポート名を OUT_B と OUT_C に分けて２つの OnFwd 命令を使っても，Program5-3 のように出力ポート名を OUT_BC と１つにまとめてもほぼ同じ動きとなります。また，Program5-1 のように使っても同様な動きになります。厳密にいえば，各命令を実行するために少しだけ時間がかかりますから，モータ毎に回転の設定を行うと，ほんの少しだけ時間遅れが発生します。

表５－１　モータの設定方法の違い

Program5-2 の一部	Program5-3 の一部
OnFwd(OUT_B,30); OnFwd(OUT_C,30);	OnFwd(OUT_BC,30);

【チャレンジ５－１】

Program5-3 の前進や後退時のパワー値「30」の数値をいろいろ変えて，ロボットの移動する速度を毎秒 cm の単位で計測しましょう。

５．２　ロボットの方向転換

前進と後退，それに移動する速度の変更までができるようになりました。次に，ロボットの移動方向を変える方向転換に挑戦しましょう。三輪型移動ロボットでは，自動車と違いハンドルがないので車輪の角度を変えることによって移動方向を変えることができません。どのようにして方向転換するプログラムを作ればよいのでしょうか？

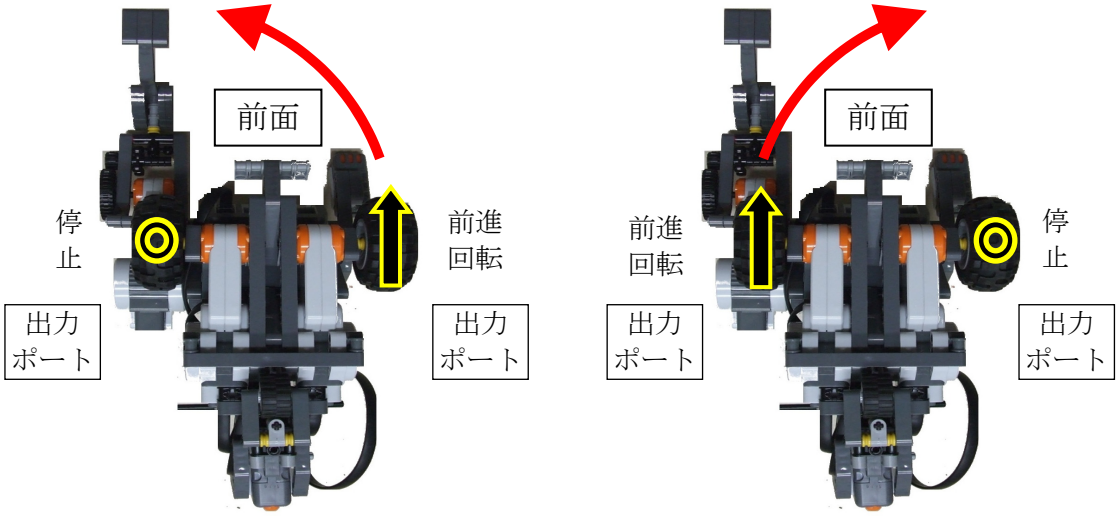
移動型三輪ロボットの左右の車輪は，それぞれ独立に回転方向を設定できます。左右の車輪の回転方向を変えることで，方向転換を実現します。この方法は，ブルドーザーなどのキャタピラーで移動する機械でも使われています。

ここでは，方向転換する方法として図５－１と図５－２の２種類を説明します。両図とも，移動型三輪ロボットを下から見た図なので，移動する方向に注意しましょう。矢印は車輪が進む方向を示し，◎は方向転換するときの中心となる位置を示しています。

図５－１は，片方の車輪だけを回転させて，もう一方の車輪を停止していますから，ロボットの本体は，停止した車輪を中心として，回転する車輪の方向に転換します。このテキストでは，このような方向転換のことを「カーブ」と呼ぶことにします。

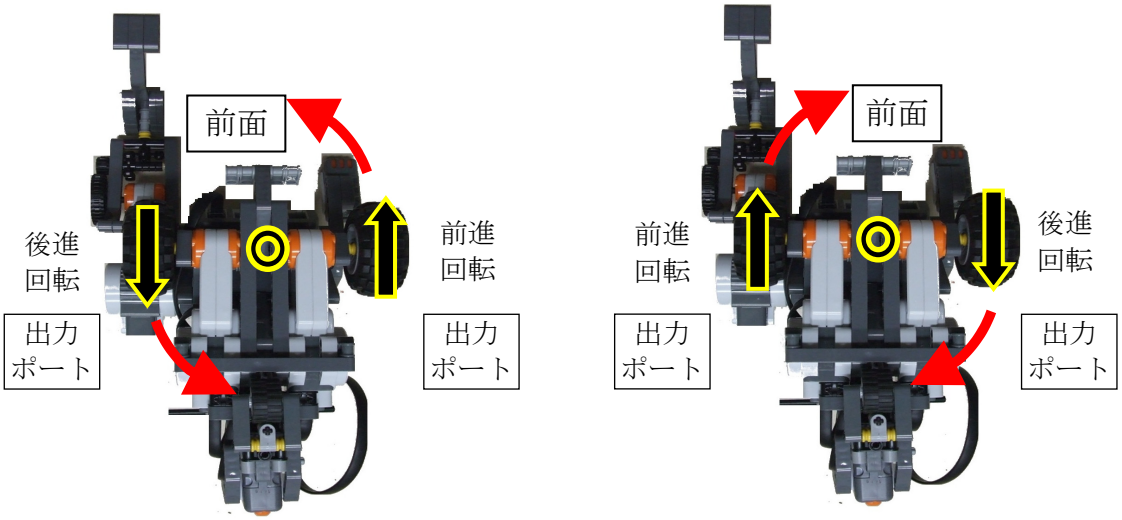
図５－２は，両方の車輪を互いに逆方向に回転させていますから，ロボットの本体は，両方車輪を結ぶ線の真ん中を中心として，前進する車輪の方向に転換します。このテキストでは，このような方向転換のことを「ターン」と呼ぶことにします。［ターン」は，本体を移動することなく方向だけ転換する場合に便利です。

また，左右の車輪の回転方向とパワー値を変化させることで「カーブ」と「ターン」の中間のような方向転換もできます。



（a）右にカーブ　　（b）左にカーブ

図５－１　方向転換「カーブ」するときの車輪の回転方向と方向転換の中心位置（◎）。
ロボットの底面から見た図となっているので，左右が逆になっています。



（a）右にターン　　（b）左にターン

図５－２　方向転換「ターン」するときの車輪の回転方向と方向転換の中心位置（◎）。
ロボットの底面から見た図となっているので，左右が逆になっています。

それでは，次の２つのプログラム（Program5-4 と Program5-5）を入力し，それぞれのロボットの動きを確認しましょう。

【Program5-4】

```
 1  task main()
 2  {
 3      OnFwd(OUT_BC,30);
 4      Wait(4000);
 5      OnFwd(OUT_B,30);
 6      Off(OUT_C);
 7      Wait(3000);
 8      Off(OUT_BC);
 9  }
```

【Program5-5】

```
 1  task main()
 2  {
 3      OnFwd(OUT_BC,30);
 4      Wait(4000);
 5      OnFwd(OUT_B,30);
 6      OnRev(OUT_C,30);
 7      Wait(3000);
 8      Off(OUT_BC);
 9  }
```

Program5-4 と Program5-5 の違いについて説明します。

Program5-4

5 行目　OnFwd(OUT_B,30);
　出力ポートＢのモータを前進回転させる命令です。

6 行目　Off(OUT_C);
　出力ポートＣのモータを停止させる命令です。

出力ポートＢのモータを前進させ，出力ポートＣのモータを停止させています。そのため，ロボットは左に「カーブ」するように移動します。

Program5-5

5 行目　OnFwd(OUT_B,30);
　出力ポートＢのモータを前進回転させる命令です。

6 行目　OnRev(OUT_C,30);
　出力ポートＣのモータを後進回転させる命令です。

出力ポートＢのモータを前進させ，出力ポートＣのモータを後退させています。そのため，ロボットは左に「ターン」するように移動します。

Program5-4 と Program5-5

7 行目　Wait(3000);

左にカーブ，左にターンする時間を，3000 ティック（3 秒間）に設定します。ここでは，カーブやターンする角度を直接指定できないことに注意しましょう。

このように，ロボットの片方のモータを止めたり，回転方向を変えたりすることで，カーブしたりターンしたりすることができます。

【チャレンジ５－２】

右にカーブするようにロボットを動かしましょう。

【チャレンジ５－３】

右にターンするようにロボットを動かしましょう。

第６章　分かりやすいプログラムと繰り返し処理

６．１　数値に名前をつける（記号定数）

Program4-6 でも少し説明しましたが，数値に名前をつけて記号定数で表すと，その数値の意味がよく分かるようになります。また，プログラム中に何度も入力しなければならない同じ意味の数値を記号定数に置きかえると，それぞれの数値を変更しなくても記号定数の値を変更するだけですみます。

記号定数を使うには,#define という命令で内容を決めます。プログラム(Program6-1)を入力しましょう。

※　記号定数を設定するときに注意しなければならないこと　※

① #define，記号定数，設定する文字列の間にスペースを１つ以上入れる。
② 記号定数は，英字から始まり，英数字と_（下線）にする。
③ 習慣的に，記号定数の英字は大文字を使う。
④ 設定する文字列の間にスペースを入れない。

【Program6-1】

```
 1 #define MOVE_T 4000
 2 #define TURN_T 2000
 3
 4 task main()
 5 {
 6     OnFwd(OUT_BC,30);
 7     Wait(MOVE_T);
 8     OnRev(OUT_B,30);
 9     Wait(TURN_T);
10     Off(OUT_BC);
11 }
```

Program6-1 は直進した後に左にターンさせるプログラムです。それでは,新たに加わったプログラムの書き方について説明します。

1 行目　#define MOVE_T 4000

MOVE_T という記号定数に 4000 という文字列を定義しています。この後のプログラムでは，MOVE_T という文字列を 4000 という文字列に置きかえます。

2 行目　#define TURN_T 2000

TURN_T という記号定数に 2000 という文字列を定義しています。この後のプログラムでは，TURN_T という文字列を 2000 という文字列に置きかえます。

7 行目　Wait(MOVE_T);

今まで，Wait の()の中には数値が入っていましたが，ここでは MOVE_T という記号定数が入っています。1 行目で記号定数 MOVE_T は 4000 という文字列に定義されています。そのため，Wait(4000);と同じ命令になります。

8 行目　OnRev(OUT_B,30);

この行では出力ポートＢのモータを後進回転させています。すでにポートＣのモータは，まだ前進している（回転が続いている状態）ので，ロボットは右にターンすることになります。モータの設定は，その状態を変更するまで同じ状態が保たれます。

9 行目　Wait(TURN_T);

7 行目と同じように,記号定数 TURN_T は 2000 という文字列に定義されていますから，Wait(2000);と同じ扱いになります。

記号定数をプログラムの中で利用すると，数値に意味のある名前をつけることができるので，ロボットの動作がわかりやすくなります。Program6-1 のように前進する時間や曲がる時間などを文字列として記号定数に設定します。また，その数値を修正するときは，記号定数に設定する文字列だけを直せばよいので，必要に応じて使うようにしましょう。

【チャレンジ６－１】

Program6-1 の記号定数を自分の分かりやすい名前に変えてみましょう。

６．２　同じ動きの繰り返し

同じ動きをロボットにさせたいときには repeat という命令を使います。繰り返しを行いたい命令のまとまりは repeat に続く「{」から「}」の間に並べられます。このようにたくさんの命令を１つにまとめるときは，中カッコの組を使います。

このまとまりを分かりやすくするために，スペース４個を入れて段付けしましょう。プログラムに含まれる「{」と「}」は，必ず「対」にしておかなければなりません。

例えば，時計回りに四角形を描く動きをロボットにさせたいときどのような手順を考えればよいでしょうか？

図６－１のように四角形を描くためには，「①前進する動き」と「②右に 90° 曲がる」という２種類の動きを４回実行します。

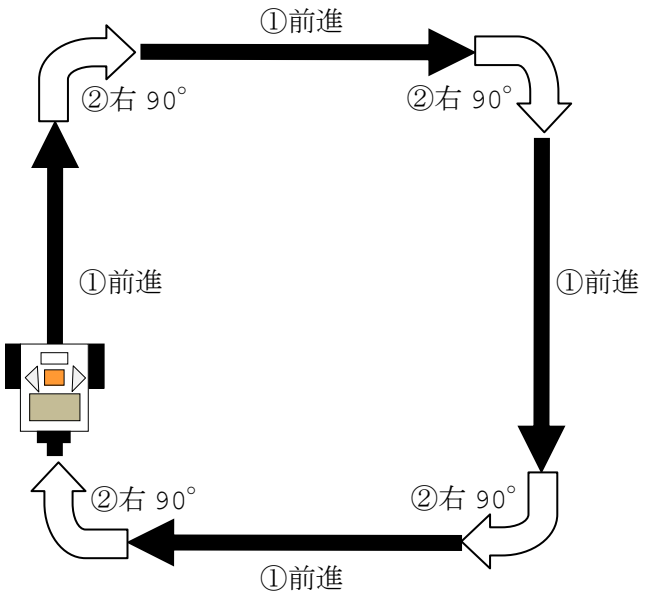


図６－１　時計回りに四角形を描くように移動

まず，右にターンする時間を決めましょう。プログラム(Program6-2)を入力して，実行し，ロボットが約 90° 右にターンする時間を決めて 2 行目の「1000」を修正しましょう。この時間は，ロボットの電池の消耗具合やタイヤと床面の状態で変化します。

【Program6-2】

```
#define TURN_T 1000

task main()
{
    OnRev(OUT_B,30);
    OnFwd(OUT_C,30);
    Wait(TURN_T);
    Off(OUT_BC);
}
```

つぎに，四角形を描く動きをさせるプログラム（Program6-3）を入力しましょう。

【Program6-3】

```
#define MOVE_T 4000
#define TURN_T プログラム(Program6-2)で決めた値

task main()
{
    OnFwd(OUT_BC,30);
    Wait(MOVE_T);
    OnRev(OUT_B,30);
    Wait(TURN_T);

    OnFwd(OUT_BC,30);
    Wait(MOVE_T);
    OnRev(OUT_B,30);
    Wait(TURN_T);

    OnFwd(OUT_BC,30);
    Wait(MOVE_T);
    OnRev(OUT_B,30);
    Wait(TURN_T);

    OnFwd(OUT_BC,30);
    Wait(MOVE_T);
    OnRev(OUT_B,30);
    Wait(TURN_T);

    Off(OUT_BC);
}
```

Program6-3 では，6 行目から 24 行目まで，「①前進」と「②右に曲がる」という動作の命令を４回繰り返して書いていることに気がつきましたか？ Program6-3 のように，同じ命令を繰り返して書くことは，手間がかかり入力ミスも増えますし，プログラムが必要以上に長くなってしまいます。

このような場合には，指定された回数だけ命令を繰り返し実行する repeat 文を使うと便利です。プログラム（Program6-4）を入力しましょう。

【Program6-4】

```
 1 #define MOVE_T 4000
 2 #define TURN_T プログラム(Program6-2)で決めた値
 3
 4 task main()
 5 {
 6     repeat(4)
 7     {
 8         OnFwd(OUT_BC,30);
 9         Wait(MOVE_T);
10         OnRev(OUT_B,30);
11         Wait(TURN_T);
12     }
13     Off(OUT_BC);
14 }
```

Program6-3 と比べて，プログラムの長さが短くなっていることが分かるでしょう。それでは，新たに加わったプログラムを説明していきます。

6 行目　repeat(4)

repeat の(　)内に書いた数値 4 の回数だけ「{」と「}」で囲まれた命令のまとまりを繰り返し実行します。「{」と「}」の間は，スペース４個を入れて段付けすると見やすくなります。

なお，repeat(4)の直後に「;（セミコロン)」を付けてはいけません。「repeat(4);」のようにするとコンパイル時にエラーは出ませんが，7 行目から 12 行目のまとまりを繰り返して実行することにはなりません。よく注意してください。

8 行目～11 行目

repeat によって囲まれた命令になります。ここでは，前進し右に曲がる動作を命令しています。

TURN_T を 90° 曲がるように調整しておき，Program6-3 を実行すると，図６－１に示すような時計回りに四角形を描くようにロボットが動きます。

【チャレンジ６－２】

Program6-4 は，時計回りに四角形を描くプログラムです。このプログラムを修正して反時計回りに四角形を描くようなプログラムを考えましょう。

【チャレンジ６－３】

三角形を描くような動きをさせるにはどのようなプログラムにすればよいでしょうか？

６．３　入れ子を使った繰り返し

みなさんは「入れ子」という言葉を知っていますか？　台所で使うザルは，大きいザルに少し小さいザルを重ねて置くことができるようになっていて，このような特徴をもつ道具を入れ子といいます。同じようなものに，ロシアの伝統的なおもちゃのマトリョーシカがあり，かわいい人形の中に，少し小さい人形が入っています。

ロボットの動きにも，ある動作の中に別の動作を入れ子のように組み合わせることができます。図６－２のように大きな四角形の中に小さな四角形の動きを反時計まわりにロボットにさせるには，どのようにすればよいでしょうか？

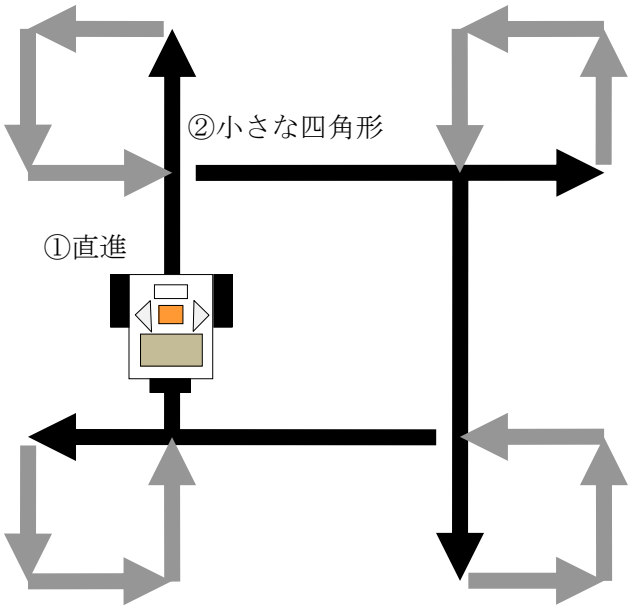


図６－２　大きな四角形の中に小さな四角形を含む動き

図６－２に示す動きをさせるためには，大きな四角形を描くための①直進と，②小さな四角形を描く動作，を４回実行すればよいことがわかります。②の動きは，Program6-4 のように repeat 文を使って１つの動きとしてまとめることができます。

大きな四角形の動きを実現するために，４回行う repeat 命令の中に，②の動きを行う repeat 命令を入れ込むことによって，このような動きをするプログラムを作成できます。Program6-5 を見てみましょう。

【Program6-5】

```
 1 #define LMOVE_T 3000
 2 #define SMOVE_T 1000
 3 #define TURN_T  1200
 4
 5 task main()
 6 {
 7     repeat(4)
 8     {
 9         OnFwd(OUT_BC,30);
10         Wait(LMOVE_T);
11         repeat(3)
12         {
13             OnRev(OUT_C,30);
14             Wait(TURN_T);
15             OnFwd(OUT_C,30);
17             Wait(SMOVE_T);
18         }
19     }
20     Off(OUT_BC);
21 }
```

1 行目　#define LMOVE_T 3000
　大きな四角形を描くために前進する時間を記号定数で設定しています。

2 行目　#define SMOVE_T 1000
　小さな四角形を描くために前進する時間を記号定数で設定しています。

3 行目　#define TURN_T 1000
　小さな四角形を描くために左に 90°ターンする時間を設定しています。この時間は，状況によって変わります。

7 行目　repeat(4)
　この行での repeat は，次の 8 行目から 19 行目のまとまりを４回繰り返す命令になっています。

9 行目～10 行目
　①の前進を実行する部分です。

11 行目～18 行目

②の小さな四角形を描くように実行する部分です。四角形の最初の一辺は，①の動きで行っているので，左に 90° 曲がる動きと直進する動きはそれぞれ３回ずつ行うことになります。そのため，11 行目の repeat で指定する回数は３回となっています。

Program6-5 のように repeat の中に repeat があるプログラムを入れ子構造といいます。

【チャレンジ６－４】

図６－３に示すように大きな三角形と小さな三角形の動きを組み合わせた動きをするプログラムを作りましょう。

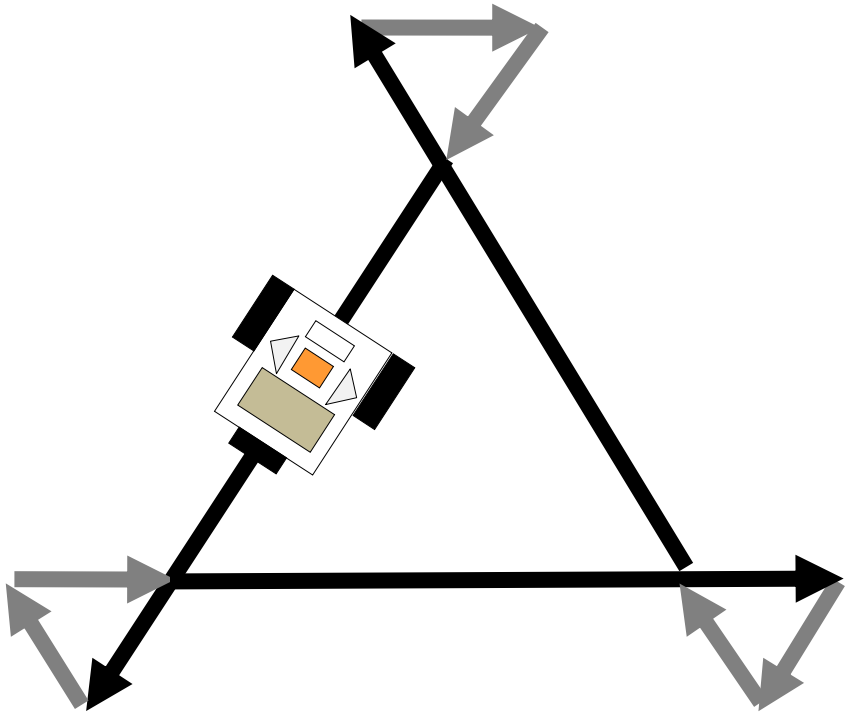


図６－３　大きな三角形の中に小さな三角形を含む動き

６．４　コメントを使ってプログラムを説明

プログラムを分かりやすくするために，プログラム中にロボットの動きや命令の意味を説明するコメント（注釈）を書きます。コメントには日本語を利用できます。コメントはプログラムの実行にまったく影響しません。コメントを書く方法は次の２種類があります。

（１）　//（連続する２つのスラッシュ）
　　//の右側から行の最後までがコメントになります。

【例】

```
    OnFwd(OUT_B);  // この部分はコメント
```

（２）　/* と */
　　いくつかの行にまたがってコメントをするときには，/* から */ の間にコメントを書きます。

【例】

```
    /*
                この部分は何行でもコメント
                    ……
    */
```

ロボットの動きをうまくコメントとしてプログラムに書いておくと，後でプログラムを見直すときに参考になります。

Program6-5 にコメントを追加すると，Program6-6 のようになります。Program6-6 を入力しましょう。コメント部分は自分の分かりやすいように工夫して入力しましょう。

【Program6-6】

```
 1 /*
 2   大きな四角形に小さな四角形を組み合わせた動作をするプログラム
 3       作成日：2012 年 5 月 2 日
 4       作成者：鳴門　渦美
 5 */
 6
 7 #define LMOVE_T 3000  // 大きな四角形の辺を前進する時間
 8 #define SMOVE_T 1000  // 小さな四角形の辺を前進する時間
 9 #define TURN_T  1000  // 90°ターンする時間
10
11 task main()
12 {
13     repeat(4)  // 大きな四角形
14     {
15         OnFwd(OUT_BC,30);  // 大きな四角形の辺をパワー値 30 で前進
17         Wait(LMOVE_T);
18         repeat(3)  // 小さな四角形
19         {
20             OnRev(OUT_C,30);      // 小さな四角形の角をターン
21             Wait(TURN_T);
22             OnFwd(OUT_C,30);      // 小さな四角形の辺を前進
23             Wait(SMOVE_T);
24         }
25     }
26     Off(OUT_BC);       // 停止
27 }
```

コメントの日本語が正しく表示されない場合は，図１－８から図１－１１を参考にして，エディッタのフォントを日本語に設定しましょう。

第７章　少し進んだプログラム

７．１　変数の使い方（うずまきを描くロボット）

より動きに変化のあるロボットを作るためには，プログラムを実行している途中で，動きを決めている値を変える必要があります。このような値を入れておくところとして，変数を使います。変数とは，数値を入れておける箱のようなものと考えればよいでしょう。

例えば，Wait(4000)というプログラムでは，4000 という数値を実行中に変更することはできません。プログラム中に書かれた数値は，プログラムの実行中に変更できませんが，変数は何回でもその値を変更できます。

図７－１のようにうずまき状に四角を描いていく動きをするロボットを考えましょう。うずまき状ですから，前進する時間が少しずつ長くなっていく必要があります。前進する時間に変数を使って，段々と長くなるようにプログラムを作ります。

図７－１の動きをするプログラムを Program7-1 に示します。

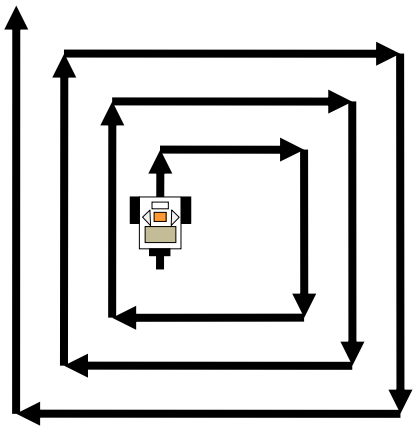


図７－１　うずまき状の四角形を描きながら移動

【Program7-1】

```
 1 #define TURN_T 1100
 2
 3 int move_t;
 4
 5 task main()
 6 {
 7     move_t = 1000;
 8     repeat(10)
 9     {
10         OnFwd(OUT_BC,30);
11         Wait(move_t);
12         OnRev(OUT_B,30);
13         Wait(TURN_T);
14         move_t += 500;
15     }
16     Off(OUT_BC);
17 }
```

それでは新たに加わったプログラムの書き方について説明します。

1 行目　#define TURN_T 1100

90°ターンする時間を表す記号定数 TURN_T を 1100 にします。この値は，状態によって変わります。

3 行目　int move_t;

int move_t で move_t という名前の変数を使うことを宣言しています。int は integer の略で「整数型」という意味です。

変数 move_t を宣言すると表７－１の番号１にあるように，整数値を１つだけ覚えることのできる「move_T という名前の箱」を NXT ブロックの中に作ります。ロボットが前進する時間を変数 move_t に入れておきます。うずまき状に四角形を描くようにするためには，前進する時間を段々と大きくする必要があります。

※　変数を宣言するときに注意しなければならないこと　※

① 変数の名前は英字から始まり，英数字と_（下線）にする。
② 習慣的に，変数の名前は英字の小文字を使う。
③ 変数の名前にスペースを入れない。

7 行目　move_t = 1000;

move_t に初期値として 1000 を代入しています。初期値とは変数の最初に入れておく数値をいいます。変数には初期値を入れておく必要があります（表７－１の番号２）。

14 行目　move_t += 500;

表７－１の番号３と４のように，move_t += 500 を実行するたびに move_t という変数の値に 500 を加えます。

8 行目にある repeat 文では，繰り返し回数が 10 となっているので，move_t += 500 は 10 回実行されます。move_t の値は，1000，1500，2000，2500，3000，･･･と 500 を 10 回加えていくと，6000 まで増加していきます。

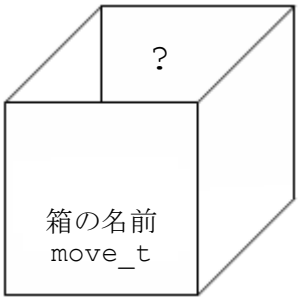
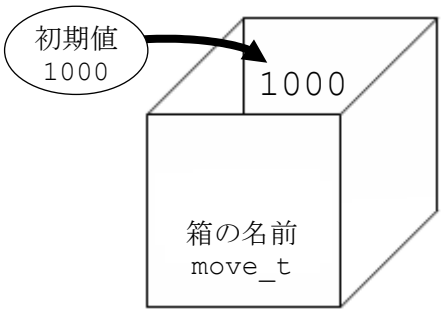
move_t の値が 500 ずつ増加していくと，OnFwd(OUT_BC,30)と Wait(move_t)で設定した前進する時間がだんだんと長くなることになります。ターンするたびに前進する時間が長くなり，うずまきを描くようにロボットが移動します。

ここでは，move_t += 500 として，500 ずつ加えていく方法を説明しました。そのほかにも，変数には，*=で値をかけたり，-=で値を引いたり，/=で値を割ったり（小数点以下は切り捨てられます）することもできます。

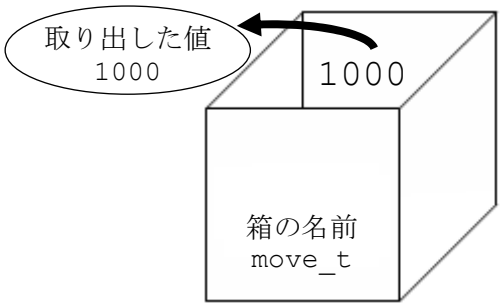
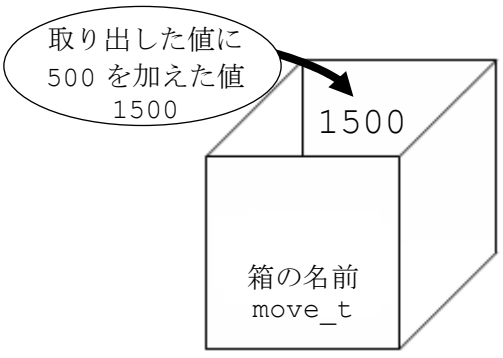
【チャレンジ７－１】

渦巻状の三角形を描くような動きをさせるにはどのようなプログラムにすればよいでしょうか？

表７－１　Program7-1 における変数の値の変化

<table>
<tr><th>番号</th><th colspan="2">内　　容</th><th>意　　味</th></tr>
<tr><td>1</td><td colspan="2">変数は数値が１つだけ入ることのできる「箱」と考える。箱の名前は「変数名」に対応する。箱を準備するために
int move_t;
を使う。準備したときには，箱の中に入っている値は不明である。</td><td> 
</td></tr>
<tr><td>2</td><td colspan="2">最初の値1000をmove_tという箱に入れる。最初の値を「初期値」という。</td><td> 
</td></tr>
<tr><td>3</td><td rowspan="2">move_t += 500
の意味

move_t + 500
↓
move_t

move_t の値は，500 だけ増加することになる。</td><td>move_t の箱から値 1000 を取り出す。</td><td> 
</td></tr>
<tr><td>4</td><td>取り出した値 1000 に 500 を加えた値 1500 を move_t の箱に再び入れる。</td><td> 
</td></tr>
</table>

７．２　乱数の使い方

ロボットが虫のようにランダムな動きをしたら面白いと思いませんか。ここでは，このような動きを乱数を使って作ります。例えば，サイコロをふったときの目の数のように，予測できない値を乱数といいます。

これまでのプログラムに含まれる数値や変数の値は，あらかじめ決まった値しかとりませんでした。予測できない値（乱数）を得るためには，Random 命令を使います。この命令は，指定された範囲の乱数を返します。ここでは，Random 命令で得られた乱数を動く時間に使って，虫のようなロボットを作ります。

Progarm7-2 を実行すると，予測できない前進やターンを繰り返します。そのため同じ動きを繰り返すことはありません。

【Program7-2】

```
 1 int move_t, turn_t;
 2
 3 task main()
 4 {
 5     while(true)
 6     {
 7         move_t = Random(600);
 8         turn_t = Random(400);
 9         OnFwd(OUT_BC,30);
10         Wait(move_t);
11         OnRev(OUT_B,30);
12         Wait(turn_t);
13     }
14 }
```

Program7-2 は，ランダムな時間だけ前進したりカーブしたりするプログラムです。それでは新たに加わったプログラムを詳しく説明していきます。

1 行目　int move_t, turn_t;

Program7-2 では２つの変数 move_t と turn_t を使います。変数 move_t は前進する時間，turn_t はターンする時間を入れます。この時間は，Random という乱数によって値を決めます。変数は「,(コンマ)」を使って，宣言文の中に一度に複数の変数を定義できます。

5 行目　while(true)

repeat 文ではなく while(true)を使っています。while の条件がいつも true（真）なので,「{」と「}」で囲まれている命令を無限に繰り返し実行します。

7 行目　move_t = Random(600);

Random の次にある数値 600 は「0 から 599 までの値をランダムにとる」ということを示しています。「600」を含まないことに注意しましょう。この値を変数 move_t に入れます。前進する時間は 0 秒間から 0.599 秒間（0 から 599 ティック）までのどれかになります。

8 行目　turn_t = Random(400);

Random の次にある数値 400 は「0 から 399 までの値をランダムにとる」ということを示しています。「400」を含まないことに注意しましょう。この値を変数 turn_t に入れます。ターンする時間は 0 秒間から 0.399 秒間（0 から 399 ティック）までのどれかになります。

【チャレンジ７－２】

乱数を使って前進，後進，右ターン，左ターンをいろいろと繰り返すプログラムを作ってみましょう。

第８章　処理の順番を変えるプログラム

今まで，繰り返し命令を実行するために repeat 文を紹介しました。ここでは処理の順番を変えるプログラムで使う「制御文」について説明をします。

８．１　ｉｆ文

ある条件が成り立つときに実行したい命令があるときに使う文です。身近な例をあげると交差点の信号による動作があります。交差点の信号が青のとき進みますが，赤のときには止まります。この事例を if 文で表現すると図８－１のようになります。

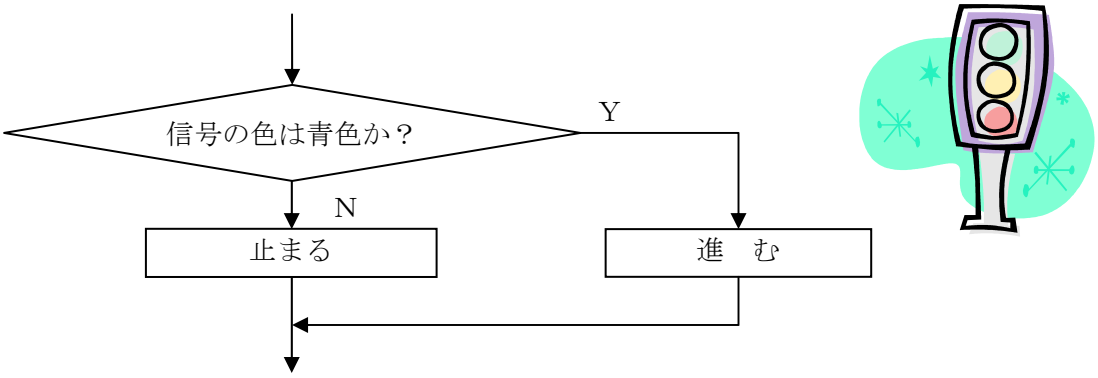


図８－１　交差点の信号による動作

このような図のことを「フローチャート」といいます。フローチャートは，矢印を使って順番に実行する内容を示します。実行する内容は長方形の枠の中に書きます。条件によって実行する内容が変わる場合は，ひし形を使ってその中に条件を書きます。ひし形の横と下にある「Y」と「N」は，それぞれ「ＹＥＳ」，「ＮＯ」を意味します。「Y」は，ひし形の中に書いた条件「信号の色が青色である」が正しいとき，実行する方向を示します。「N」は，条件が正しくないとき，実行する方向を示します。

条件が正しい（成り立つ）ときを「真(true)」，正しくない（成り立たない）ときを「偽(false)」ともいいます。

図８－１のフローチャートを NXC で書くと Program8-1 のようになります。ただし，Program8-1 は説明用ですから，実行することはできません。

【Program8-1】

```
 1  task main()
 2  {
 3      if(信号の色は青か？)
 4      {
 5          進む
 6      }
 7      else
 8      {
 9          止まる
10      }
11  }
```

3 行目 if(信号の色は青か？)

if の直後にある( )には，if 文の条件が入ります。ここでは，「信号の色は青か？」が条件です。

5 行目 進む

if 文の条件が正しいときに実行する命令です。信号の色が青のとき,「信号の色は青か？」の条件は正しいと判断されるので，この命令が実行されます。このように if 文の条件が正しいとき，条件は「真（true)」または「成り立つ」といいます。

7 行目 else

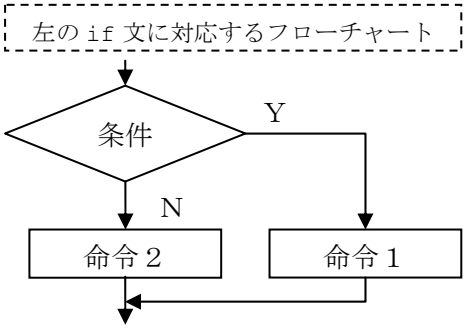
if 文の条件が正しくないときに実行する命令を書くために必要な文です。

9 行目 止まる

if 文の条件が正しくないときに実行する命令です。例えば，信号の色が赤のとき，「信号の色は青か？」の条件は正しくないと判断されます。このように if 文の条件が正しくないとき，条件は「偽（false)」または「成り立たない」といいます。

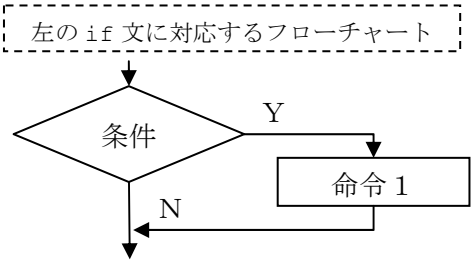
if 文の条件が正しい（Y，真，成り立つ）のとき命令１を実行し，正しくない（N，偽，成り立たない）のとき命令２を実行する if 文は，次のように書きます。

```
if(条件)
{
    命令１
}
else
{
    命令２
}
```



とくに，実行する命令２がないときは，else 以降を省略して，

```
if(条件)
{
    命令１
}
```



と書くこともできます。

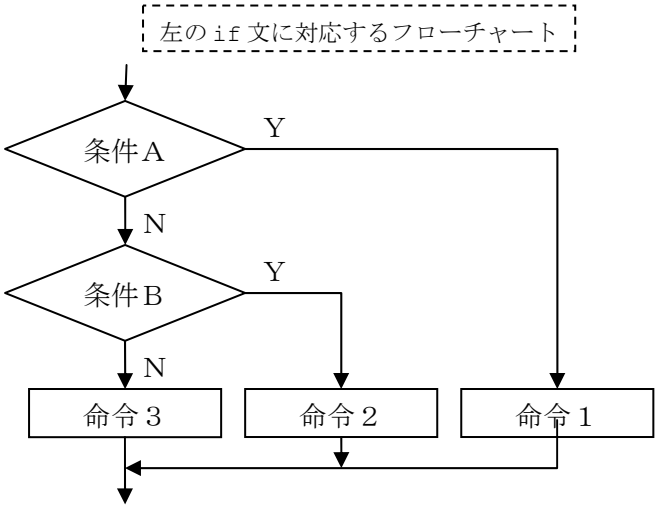
さらに，命令１が１つだけの命令のときは，{　}を省略して，

```
if(条件)命令１;
```

と短く書くこともできます。

また，２つの条件Ａと条件Ｂがあり３通りに分けたいとき，if 文を次のように書きます。

```
if(条件Ａ)
{
    命令１
}
else if(条件Ｂ)
{
    命令２
}
else
{
    命令３
}
```



この場合，条件Ａが正しい（Y，真，成り立つ）のとき，命令１を実行します。条件Ａが（N，偽，成り立たない）であり，条件Ｂが正しい（Y，真，成り立つ）のとき，命令

２を実行し，条件Ｂが（Ｎ，偽，成り立たない）のとき命令３を実行します。

Program8-2 を実行すると，3 秒間前進した後，右または左にターンする動きを繰り返します。この動きを実現するために乱数を用います。ここで使う乱数は，0 か 1 をとり，0 が選ばれたときにロボットが右にターンし，1 が選ばれたときに左にターンするように制御します。

【Program8-2】

```
 1 #define MOVE_T 3000
 2 #define TURN_T 1000
 3
 4 task main()
 5 {
 6     while(true)
 7     {
 8         OnFwd(OUT_BC,30);
 9         Wait(MOVE_T);
10         if(Random(2)==0)
11         {
12             OnRev(OUT_C,30);
13         }
14         else
15         {
16             OnRev(OUT_B,30);
17         }
18         Wait(TURN_T);
19     }
20 }
```

6 行目　while(true)

7 行目から 19 行目にかけてある「{」と「}」で囲まれた範囲を無限に繰り返します。

10 行目　if(Random(2)==0)

if 文の条件 Random(2)==0 に含まれる「==（等号が 2 つ)」は「Random で決まった値（0 または 1）と 0 が等しいか？」という条件になっています。等号が２つあることに注意しましょう。２つの数値の比較には，表８－１に示す記号を使います。

Random(2)は，0 から 1 までの値をランダムにとる命令です。2 は含まれませんから注意しましょう。ここでは，0 または 1 のどちらかの値となります。if 文の条件

「Random(2)==0」は，Random(2)の値が 0 であれば正しい（Y，真，成り立つ），そうでなければ正しくない（N，偽，成り立たない）となります。

12 行目　OnRev(OUT_C,30);
　10 行目で Random(2)の値が 0 のとき実行され，右にターンします。

14 行目　OnFwd(OUT_B,30);
　10 行目で Random(2)の値が 0 でないとき（1 のとき）実行され，左にターンします。

【チャレンジ８－１】

Program8-2 では，ターンして方向転換していました。カーブして方向転換するプログラムに変更しましょう。

８．２　条件の使い方

if 文などで使われる条件を表８－１にまとめました。２つの数値が等しいことや異なることを条件として使えます。また，２つの数値の大小関係も判断することができます。

表８－１　条件の書き方

条件	意　　味	例	例　の　意　味
==	等しい	a == b	a と b が等しいか？
!=	異なる	a != b	a と b が異なるか？
<	小さい	a < b	a が b より小さいか？
<=	小さいか等しい	a <= b	a が b より小さいか，または，等しいか？
>	大きい	a > b	a が b より大きいか？
>=	大きいか等しい	a >= b	a が b より大きいか，または，等しいか？

また，条件Ａと条件Ｂの関係を組み合わせて，１つの条件にまとめることもできます。

条件Ａと条件Ｂの両方ともが正しい（Ｙ，真，成り立つ）ときだけ，正しい（Ｙ，真，成り立つ）と判断するための条件は，

「条件Ａ　&&　条件Ｂ」

と書きます。読み方は「条件Ａ　かつ　条件Ｂ」といいます。

例えば，変数 a の値が 5 から 10 の間にあるかどうか判断する条件は，図８－２のように

「条件Ａ　5 <= a」（a は 5 以上か？）

「条件Ｂ　a <= 10」（a は 10 以下か？）

を組み合わせて，

「5 <= a && a <= 10」

とします。5 <= a <= 10 と書くと間違いになるので，注意しましょう。

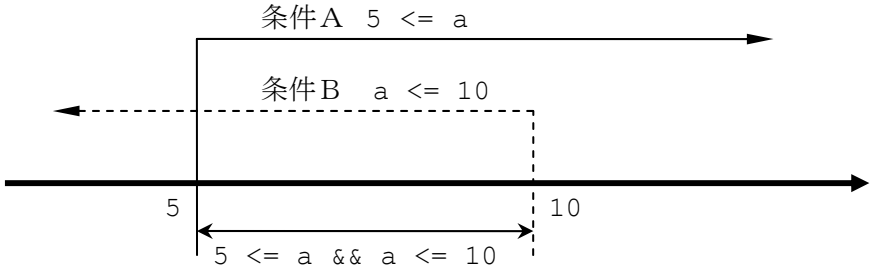


図８－２　数直線を使った条件「5 <= a && a <= 10」の考え方

条件Ａ，または，条件Ｂが正しい（Ｙ，真，成り立つ）ときに，正しい（Ｙ，真，成り立つ）とする判断するための条件は，

「条件Ａ || 条件Ｂ」

と書きます。読み方は「条件Ａ　または　条件Ｂ」といいます。「|（縦棒)」は，キーボードの「¥」のキーをシフトキーを押しながら入力します。「l（小文字のＬ)」や「1（数字の１)」とよく似ているので注意しましょう。

例えば，変数 a の値が 1 か 5 かどうかを判断する条件は，

「条件Ａ　a == 1」(a は 1 と等しいか？)

「条件Ｂ　a == 5」(a は 5 と等しいか？)

を組み合わせて，

「a == 1 || a == 5」

とします。

８．３　while文

ある条件が成り立っている間，命令を繰り返し実行するための制御文として，while 文があります。

while 文は，次のように書きます。

```
while(条件)
{
     命令
}
次の命令
```

while 文のフローチャートは図８－３のようになり，条件が成り立つ（Y）の間，{　} に書かれた命令を繰り返し，条件が成り立たない（N）となると次の命令を実行します。

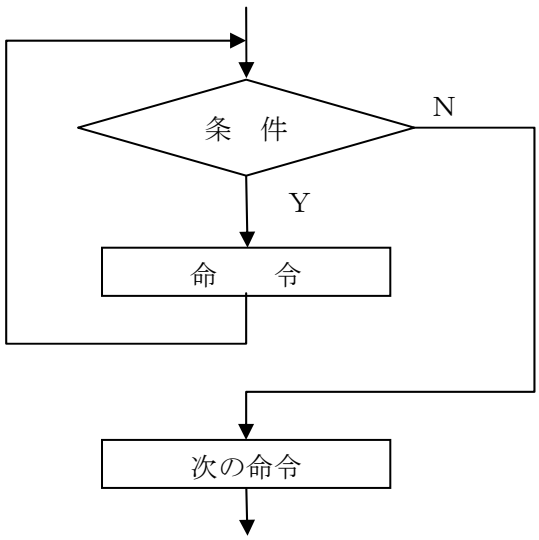


図８－３　while 文のフローチャート

無限に処理を繰り返したいときは，while 文の条件を true とし次のように書きます。

```
while(true)
{
     命令
}
```

とくに，プログラムを終了させたくないときには，プログラムの最後に

```
while(true);
```

と書きます。この場合，繰り返す命令は何もありませんが，while 文を実行し続けます。

10 秒間前進とターンを繰り返すプログラムを while 文を使って作ると Program8-3 のようになります。

【Program8-3】

```
 1 task main()
 2 {
 3     int w, x;
 4     w = 0;
 5     while(w < 10000)
 6     {
 7         x = Random(500) + 500;
 8         OnFwd(OUT_BC,30);
 9         Wait(x);
10         w += x;
11         OnRev(OUT_C,30);
12         Wait(x);
13         w += x;
14     }
15     Off(OUT_BC);
16 }
```

3 行目の int w, x で，変数 w と x を使うことを宣言しています。変数 w は，4 行目で 0 に初期化して，10000 まで数えるために使います。変数 x は，Wait 命令で指定する待ち時間です。

5 行目の while(w < 10000)は，変数 w の値が 10000 より小さい間，7 行目から 13 行目までの命令を繰り返し実行します。条件に含まれる値 10000 は，9 行目と 12 行目の Wait 命令で待つ時間をティックの単位 0.001 秒で表したもので，10 秒間となります。

7 行目の x = Random(500) + 500 は，500 から 999 までの値を x に入れます。

条件「w < 10000」が成り立たなくなると，while 文の次の命令となる 15 行目の Off 命令を実行してロボットは停止します。

８．４　do－while文

while 文とよく似た制御文に do-while 文があります。この文は，最初に指定した命令を実行した後，条件を判断し，その条件が成り立っている間命令を繰り返します。

do-while 文は，次のように書きます。とくに，while(条件)の後ろに「;(セミコロン)」をつけることを忘れないようにしましょう。

```
do
{
     命令
} while(条件);
次の命令
```

do-while 文のフローチャートは図８－４のようになります。まず，命令を実行します。その後，条件が成り立つ（Y）の間，｛　｝に書かれた命令を繰り返し，条件が成り立たない（N）となると次の命令を実行します。

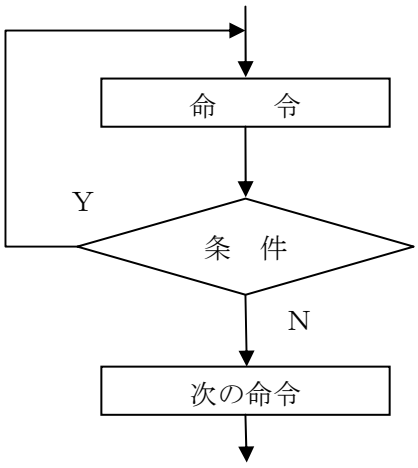


図８－４　do-while 文のフローチャート

do-while 文を使ったプログラム例を Program8-4 に示します。このプログラムを実行するとロボットは 1 秒間前進した後，0.5 秒間停止する動きを実行するたびに変数 a に 1 ずつ足していき，a の値が 5 になると 1 秒間後退する動きになっています。

【Program8-4】

```
 1 int a;
 2
 3 task main()
 4 {
 5     a = 0;
 6     do
 7     {
 8         OnFwd(OUT_BC,30);
 9         Wait(1000);
10         Off(OUT_BC);
11         Wait(500);
12         a += 1;
13     }
14     while(a <= 5);
15     OnRev(OUT_BC,30);
16     Wait(1000);
17     Off(OUT_BC);
18 }
```

5 行目　a = 0;

変数 a の初期値を決めています。ここでは，a の初期値を 0 にしています。

12 行目　a =+ 1;

変数 a に 1 を追加していきます。この命令は a++と書くこともできます。

14 行目　while(a < 5);

12 行目の a += 1 が実行されると変数 a の値は 1 ずつ増加していきます。while の条件は a <= 5 なので, a の値が 5 以下とき, 8 行目から 12 行目までを繰り返し実行します。a の値が 6 になると，while 文の次にある 15 行目の命令を実行します。

【チャレンジ８－２】

Program8-4 を参考にして，2 秒間後退した後，1 秒間停止する動きを実行するたびに変数 a に 1 ずつ足していき，a の値が 5 になると 3 秒間前進する動きのプログラムを考えましょう。

８．５　until文

ある条件が成り立つまで，命令を繰り返し実行するための制御文として，until 文があります。

until 文は，次のように書きます。

```
until(条件)
{
     命令
}
次の命令
```

until 文のフローチャートは図８－５のようになり，条件が成り立っていない（N）とき命令を繰り返し実行し，条件が成り立つ（Y）と次の命令に実行を移します。while 文や do-while 文の繰り返しの条件判断と逆になっているので注意しましょう。

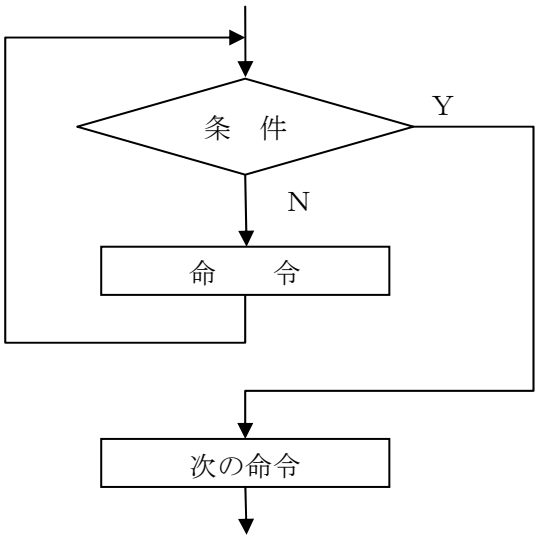


図８－５　until 文のフローチャート

また，実行する命令がなく，単純に条件が成り立つまで待つ場合は，次のように書きます。

```
until(条件);
```

このような使い方は，センサが「ある状態」になるまで待つときに利用されます。

８．６　for 文

ある処理を繰り返し実行するための制御文として for 文があります。

for 文は，次のように書きます。

```
for(命令１； 条件; 命令２)
{
     処理
}
```

for 文のフローチャートは図８－６のようになり，まず，命令１を実行します。条件が成り立っている（Y）とき処理と命令２を実行し，条件が成り立たなく（N）なると次の命令に実行を移します。

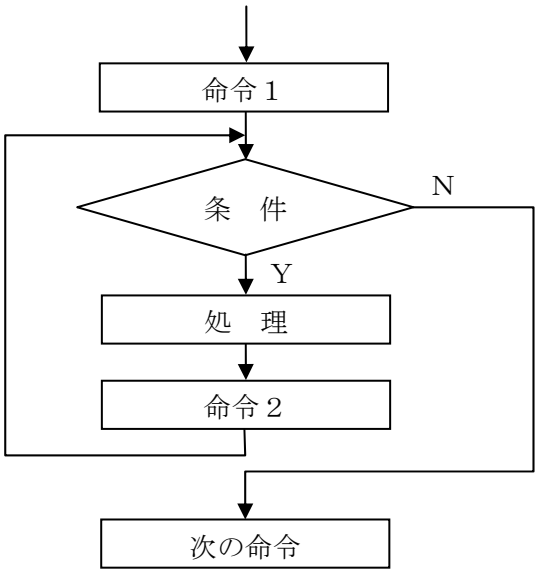


図８－６　for 文のフローチャート

for 文は，指定した回数だけ繰り返したり，値を増加（減少）させながら処理をしたいときに便利な制御文です。

次のプログラムは，NXT ブロックの液晶ディスプレイに格子を描きます。

【Program8-5】

```
 1 task main()
 2 {
 3     int x, y;
 4     for(x=0; x < 100; x+=10) {
 5         LineOut(x,0,x,63);
 6     }
 7
 8     for(y=0; y < 64; y+=10) {
 9         LineOut(0,y,99,y);
10     }
11     while(true);
12 }
```

4 行目の for 文では，変数 x の値が 0 から 10 ずつ増加していき，90 まで，5 行目を繰り返し実行します。LineOut によって垂直な線が順番に描かれます。

8 行目の for 文では，変数 y の値が 0 から 10 ずつ増加していき，60 まで，9 行目を繰り返し実行します。LineOut によって水平な線が順番に描かれます。

８．７　break 文

repeat，while，do-while，until，for の途中で，繰り返しを終了し，次の命令文に実行を移したいときは，break を使います。通常，if 文と一緒に使って，ある条件を満たしたら繰り返しの途中で中断するような処理にします。

次のプログラムは，入力ポート１番に接続したタッチセンサがオンになると終了するように Program8-3 を変更したものです。

【Program8-6】

```
 1 task main()
 2 {
 3     SetSensorTouch(IN_1);
 4     int w, x;
 5     w = 0;
 6     while(w < 10000)
 7     {
 8         if(Sensor(IN_1)==1) break;
 9         x = Random(500) + 500;
10         OnFwd(OUT_BC,30);
11         Wait(x);
12         w += x;
13         OnRev(OUT_C,30);
14         Wait(x);
15         w += x;
16     }
17     Off(OUT_BC);
18 }
```

8 行目の if 文によって，入力ポート１番に接続されたタッチセンサがオンになると break 文が実行され，while 文の繰り返しを抜け出し，17 行目に実行が移りモータが停止します。

第９章　いろいろなセンサ

NXT ブロックの入力ポートには，第３章で説明したタッチセンサに加えて，光センサ，超音波センサ，サウンドセンサなどを接続して利用できます。この章では，いろいろなセンサの使い方を説明します。

９．１　タッチセンサ

タッチセンサ（図３－１）を使うと，何かに触れたら動きをかえるロボットを作ることができます。タッチセンサを図１－２の移動型三輪ロボットの前方に取り付け，入力ポート１番にケーブルで接続します。接続した様子を図９－１に示します。タッチセンサは，移動型三輪ロボットの組み立て説明図の 32～33 ページに掲載されている光センサをタッチセンサに読み替えて，取り付けてください。

図９－１　タッチセンサを前方に取り付けた移動型三輪ロボット

プログラムを作成する前にタッチセンサが正常の動作しているか確認しましょう。

タッチセンサの接続ができたら，NXT ブロックの左・右セレクトボタンを使って「View（見る）」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Touch（接触）」を表示して，NXT ボタン（オレンジ色）を押します。続けて，タッチセンサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，現在のタッチセンサの状態が「0（接触していない）」または「1（接触中）」で表示されます。元の状態に戻りたいときはクリアボタンを押していきます。

ここでは，ロボットが何かにぶつかるまで前進し続けるプログラム Program9-1 を示します。

【Program9-1】

```
 1 task main()
 2 {
 3     SetSensorTouch(IN_1);
 4     OnFwd(OUT_BC,30);
 5     until(Sensor(IN_1)==1);
 6     Off(OUT_BC);
 7 }
```

3 行目　SetSensorTouch(IN_1);

どのようなセンサを接続しているか NXT ブロックに伝える命令です。SetSensorTouch に続いて(IN_1)とあります。これは，「入力ポート 1 番はタッチセンサです。」ということを設定する命令になっています。入力ポートの番号を変更するときは，IN_1 の数字を変えます。入力ポートの番号は 1 から 4 までです。

5 行目　until(Sensor(IN_1)==1);

until 文は，指定された条件が成り立つまで命令を繰り返し実行します。ここでは，unitl 文に続く命令は何もありませんから条件「Sensor(IN_1)==1」が成り立つまで何もせず待ち続けます。Sensor(IN_1)の値が，タッチセンサに何も触れていないときは 0，何か触れているときは 1 をとるようになっています。この文では Sensor(IN_1)の値が 1 になるまで待ち続けます。

すでに 4 行目の OnFwd(OUT_BC,30)という命令が実行されロボットは，パワー値 30 で前進していますから，その状態が保たれ，until 文を実行中していてもロボットは前進し続けます。

6 行目 Off(OUT_BC);

Sensor(IN_1)の値が 1 になると，6 行目に実行が移り，モータは停止します。

Program9-1 では，壁などの障害物が前方の取り付けられたタッチセンサにぶつかるとその場で止まるプログラムでした。次に，壁などの障害物に当たると後退して止まるというプログラムを作るにはどうすればよいでしょうか？

Program9-1 の 6 行目の Off(OUT_BC)の前に後退する命令を追加したプログラムは，Program9-2 のようになります。

【Program9-2】

```
 1  task main()
 2  {
 3      SetSensorTouch(IN_1);
 4      OnFwd(OUT_BC,30);
 5      until(Sensor(IN_1)==1);
 6      OnRev(OUT_BC,30);
 7      Wait(1000);
 8      Off(OUT_BC);
 9  }
```

6 行目～8 行目

Program9-1 と比べると 6 行目から 8 行目が変更されています。タッチセンサが反応すると，until 文の次の命令が実行され，モータが OnRev(OUT_BC,30)になり 1 秒間（1000 ティック）後退し止まります。

【チャレンジ９－１】

program9-2 と while(true)を使って，障害物を避けて前進するプログラムを作ってみましょう。図９－２のフローチャートを参考にしてみましょう。

また，カーブやターンの違いをうまく使って，スムーズに障害物を避けられるように工夫しましょう。

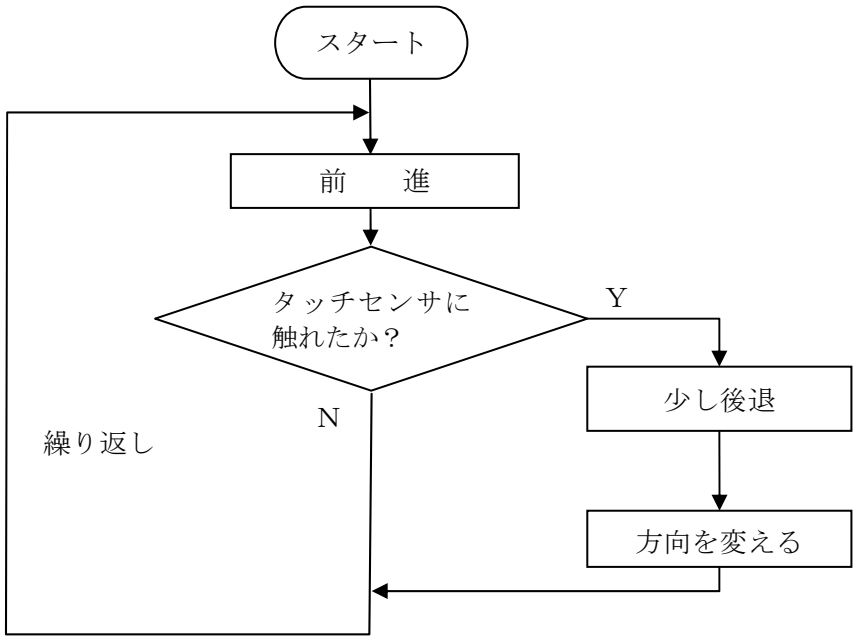


図９－２　障害物を避けて前進する動きのフローチャート

９．２　超音波センサ

次に，コウモリのように超音波を使って距離を測る超音波センサを使ってみましょう。

ここで使う超音波センサは，図９－３に示すように「超音波を送信するスピーカ」と「超音波を受信するマイク」で構成されています。NXT ブロック用超音波センサには，スピーカとマイクが並んで配置されています。ここで使っている超音波は，人間の耳では聞こえない高い周波数(40000Hz)の音です。「スピーカ」から発せられた超音波が，物体に当たり反射し，元に戻ってきた超音波を「マイク」で受信するまでの時間差を測り，その時間差から往復距離を求めます（図９－４）。

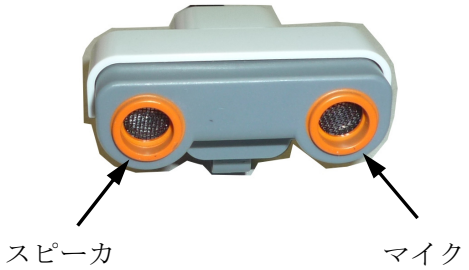


図９－３　超音波センサ

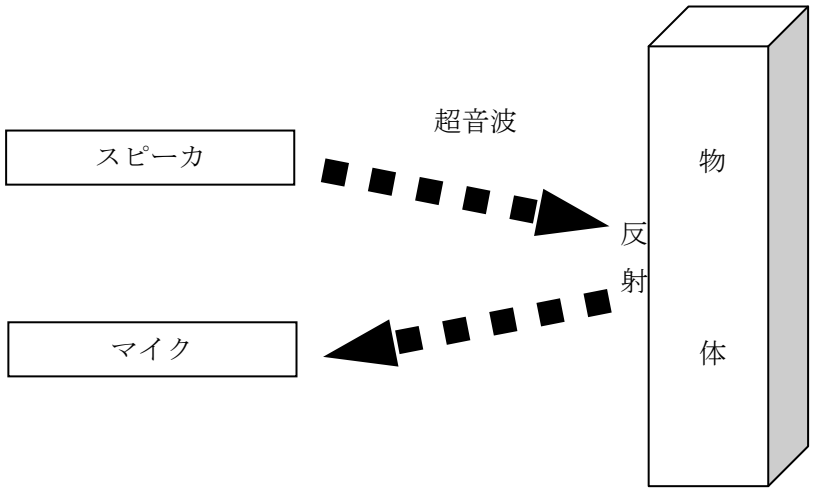


図９－４　超音波を使った距離計測

それでは，超音波センサを使って，何かに近づいたら逃げるロボットを作りましょう。移動型三輪ロボットの組み立て説明図の 28～30 ページを参考にして，超音波センサを図１－２の移動型三輪ロボットの上方に取り付け，入力ポート４番にケーブルで接続します。接続した様子を図９－５に示します。

図９－５　超音波センサを取り付けた移動型三輪ロボット

プログラムを作成する前に超音波センサが正常に動作しているか確認しましょう。

超音波センサの接続ができたら，NXT ブロックの左・右セレクトボタンを使って「View（見る）」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Ultrasonic cm（超音波 cm）」を表示して，NXT ボタン（オレンジ色）を押します。続けて，超音波センサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，超音波センサで計測した距離が「0　cm（最も近い）」から「255 cm（最も遠い）」で表示されます。測定できないときは「?????（不明）」と表示されます。この超音波センサの測定精度は，約 3cm です。元の状態に戻りたいときはクリアボタンを押していきます。

ここでは，ロボットが壁などの手前 35cm まで前進し続けるプログラム Program9-3 を示します。

【Program9-3】

```
 1 int x;
 2
 3 task main()
 4 {
 5     SetSensorUltrasonic(IN_4);
 6
 7     while(true) {
 8         x = SensorUS(IN_4);
 9         NumOut(16,16,x,true);
10
11         if(x > 35) {
12             OnFwd(OUT_BC,30);
13         } else {
14            Off(OUT_BC);
15         }
16     }
17 }
```

5 行目　SetSensorUltrasonic(IN_4);

どのようなセンサを接続しているか NXT ブロックに伝える命令です。SetSensorUltrasonic に続いて(IN_4)とあります。これは,「入力ポート４番は超音波センサです。」ということを設定する命令になっています。入力ポートの番号を変更するときは，IN_4 の数字を変えます。入力ポートの番号は 1 から 4 までです。

7 行目～16 行目　while(true) {・・・}

この while 文は条件が常に成り立つ(true)なので，無限に実行を繰り返します。

8 行目　x=SensorUS(IN_4);

1 行目で宣言した整数型の変数 x に SensorUS(IN_4)の値を入れます。SensorUS(IN_4)の値は，入力ポート４番に接続された超音波センサによって計測された距離を cm の単位で示しています。その範囲は 0～255 です。

9 行目　NumOut(16,16,x,true);

図２－５に示した液晶ディスプレイの座標(16, 16)から x の値を数字で表示します。最後にある「true」は，数字を表示する前に液晶ディスプレイを消去することを指示してい

ます。

11 行目～14 行目　if(x > 35) {・・・}

このif 文の条件は「x > 35」です。x には，計測した距離の値が cm の単位で入っていますから，35cm より離れているとき条件が成り立ち 12 行目の OnFwd(OUT_BC,30)を実行し，パワー値 30 で前進します。さもなければ，35cm 以下の距離となり，else 文以降に書かれている 14 行目の Off(OUT_BC)が実行され，ロボットは停止します。

Program9-3 では，超音波センサと壁などの障害物までの距離が 35cm 以下になるとロボットが停止するプログラムでした。次に，人や壁などの障害物にある程度近づくと後退して逃げるというプログラムを作るにはどうすればよいでしょうか？

このような動きをするロボットのプログラムは，Program9-3 の 14 行目だけ変更し，いままで停止させていた命令を後退させるようにすればできます。このプログラム（Program9-4）は次のようになります。

【Program9-4】

```
 1 int x;
 2
 3 task main()
 4 {
 5     SetSensorUltrasonic(IN_4);
 6
 7     while(true) {
 8         x = SensorUS(IN_4);
 9         NumOut(16,16,x,true);
10
11         if(x > 35) {
12             OnFwd(OUT_BC,30);
13         } else {
14             OnRev(OUT_BC,30);
15         }
16     }
17 }
```

【チャレンジ９－２】

program9-4 を参考にして，障害物を避けて前進するプログラムを作ってみましょう。図９－６のフローチャートを参考にしてみましょう。

また，カーブやターンの違いをうまく使って，スムーズに障害物を避けられるように工夫しましょう。

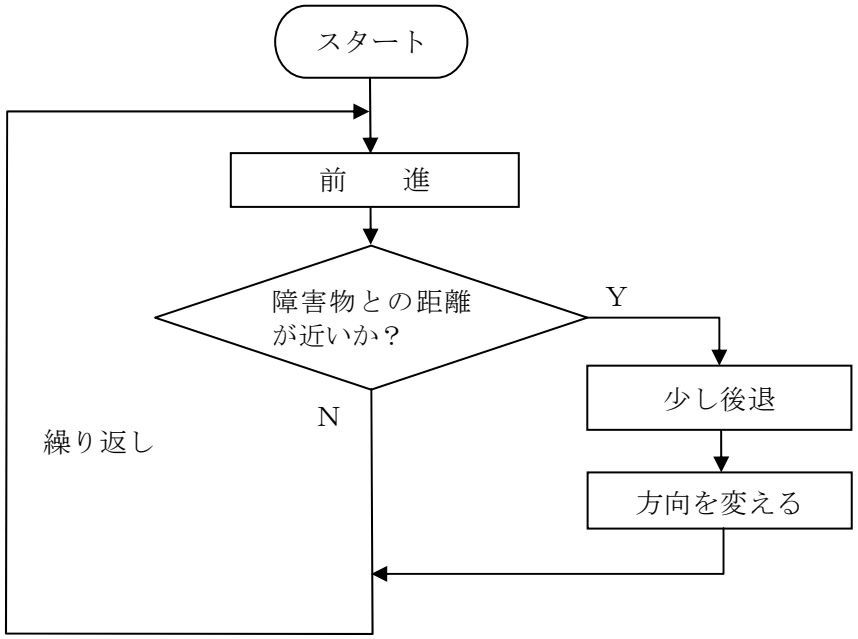


図９－６　障害物を避けて前進する動きのフローチャート

９．３　回転センサ

これまでのプログラムでは，ロボットを直進させたりカーブやターンさせたりする場合，動作時間を設定していました。そのため，移動距離や方向転換の角度は，あまり正確とはいえませんでした。もっと，正確にロボットを動作させる方法を考えてみましょう。

指定した角度だけモータが回転すると，それに応じてタイヤが回転し，ロボットが移動します。正確にロボットを動作させるためには，「動作する時間」ではなく，「モータを回転させる角度」を指定すればよいことがわかります。

図９－７に NXT に接続できるモータの部品を示します。この部品では，オレンジ色の部分が回転します。オレンジ色の部分には，回転角度を計測できる「ロータリエンコーダ」という回転センサが組み込まれています。

図９－７　回転センサを内蔵したモータ部品

それでは，移動型三輪ロボットの左右のモータに内蔵された回転センサを使って回転角度を読み取り，NXT ブロックの液晶ディスプレイにその値を表示してみましょう。Program9-5 を入力してください。このプログラムを実行すると，液晶ディプレイに２つの数値が表示されます。両側のタイヤを手で回転させてみてください。左側の数値は，左側のタイヤの回転角度，右側の数値は，右側のタイヤの回転角度にそれぞれ対応しています。

ロボットを前進させるようにタイヤを回転させると，表示されている値は増加します。反対に後進させるようにタイヤを回転させると値は減少します。タイヤをほぼ一周させると約 360 変化します。同じ方向に回転させていくと，その値はどんどん増加または減少していきます。

【Program9-5】

```
 1 task main()
 2 {
 3     int b,c;
 4     ResetRotationCount(OUT_BC);
 5     while(true) {
 6         b = MotorRotationCount(OUT_B);
 7         c = MotorRotationCount(OUT_C);
 8         NumOut(16,16,b,true);
 9         NumOut(64,16,c,false);
10     }
11 }
```

3 行目　int b,c;

出力ポートＢに接続されたモータに内蔵されている回転センサの値を記憶するための変数 b を宣言しています。同様に，出力ポートＣに接続されたモータに内蔵されている回転センサの値を記憶するための変数 c も宣言しています。

4 行目　ResetRotationCount(OUT_BC);

ResetRotationCount 命令に OUT_BC を指定し，出力ポートＢとＣに接続された２つのモータに内蔵されている回転センサの値をそれぞれ 0 に初期化しています。

5 行目～10 行目　while(true) {・・・}

回転センサの値を読み込み，その値を液晶ディスプレイに表示する処理を無限に繰り返します。

6 行目　b = MotorRotationCount(OUT_B);

MotorRotationCount に OUT_B を指定すると，出力ポートＢに接続されたモータに内蔵されている回転センサの値を示します。この値は左側のタイヤの回転角度に対応します。ここでは，変数 b に代入しています。

7 行目　c = MotorRotationCount(OUT_C);

MotorRotationCount に OUT_C を指定すると，出力ポートＣに接続されたモータに内蔵されている回転センサの値を示します。この値は右側のタイヤの回転角度に対応します。ここでは，変数 c に代入しています。

8 行目　NumOut(16,16,b,true);

NumOut 命令の３番目に true を指定しているので，一旦液晶ディスプレイの表示内容を

すべて消した後，変数 b の値を座標(16, 16)に表示します。

9 行目　NumOut(64,16,c,false);
NumOut 命令の３番目に false を指定しているので，液晶ディスプレイの表示内容を保持したまま，変数 c の値を座標(64,16)に表示します。

次は，ロボットを指定した距離だけ正確に移動するように，モータを回転させる角度を求めることについて考えましょう。

そのためには，直径と円周，おうぎ形の弧の長さと中心角の関係を考えます。図９－８に示すように直径 d，円周率をπとすると，円周 L は，

L ＝ π×d　　……　①

です。また，中心角 t° のおうぎ形の孤の長さ x は，

x ＝ L×t÷360　　……　②

です。①式を②式に代入すると

x ＝(π×d)×t÷360　　……　③

です。ロボットが移動する距離は，タイヤの「弧の長さ」に対応します。③式を t で解くと，

t ＝ x×360÷(π×d)　　……　④

となります。タイヤの直径を d(mm)とし，移動させたい距離 x(mm)が決まると，④式を使って，中心角 t° を求めることができます。

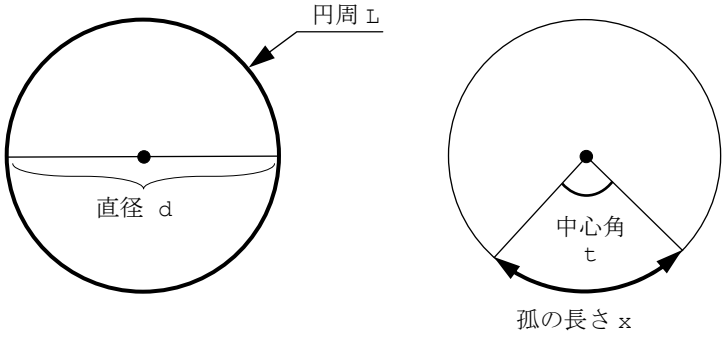


図９－８　円周と直径，孤の長さの関係

移動型三輪ロボットのタイヤの直径は d=56mm です。タイヤはゴムでできているので多少変形します。自分でタイヤの直径を測った値を使いましょう。

それでは，ロボットを x=300mm だけ移動させるために必要な中心角を計算しましょう。円周率πを 3.14 とし，④式にそれぞれの値を代入すると

t = 300×360÷(3.14×56)

となり，

t = 614.19 ……

を得ます。614.19 の少数第１位を四捨五入して整数にし，指定する回転角度を 614°とします。

Program9-6 は，ロボットを 300mm だけ前進させるプログラムです。7 行目の while 文の条件に 614 が記述されています。このプログラムを実行すると，ロボットは約 300mm 前進し，停止します。液晶ディスプレイには，左側と右側の回転角度が表示されています。左側（出力ポートＢ）の回転センサの回転角度は，ほぼ 614 になっていますが，右側（出力ポートＣ）の回転センサの回転角度は，614 から少しずれた値になっています。この理由はタイヤと床面の状態が左右で異なるためです。

【Program9-6】

```
 1 task main()
 2 {
 3     int b,c;
 4     b = 0;
 5     ResetRotationCount(OUT_BC);
 6     OnFwd(OUT_BC,30);
 7     while(b < 614) {
 8         b = MotorRotationCount(OUT_B);
 9         c = MotorRotationCount(OUT_C);
10         NumOut(16,16,b,true);
11         NumOut(64,16,c,false);
12     }
13     Off(OUT_BC);
14     while(true);
15 }
```

6 行目の OnFwd 命令で，出力ポートＢとＣに接続されているモータをパワー値 30 で前進回転させます。

7 行目にある while(b < 614)の条件「b < 614」を満たしている間，回転センサの値を読み取り，液晶ディスプレイに回転角度を表示します。左側のタイヤの回転角度が 614°より小さい間，while につづく 8 行目から 11 行目の処理を繰り返します。条件「b < 614」

を満たさなくなると，13 行目に実行が移り Off 命令でモータが停止します。

14 行目の while(true)は，液晶ディスプレイに表示されている回転角度が消されないように無限に繰り返しを行いプログラムを終了しないようにしています。

回転角度を指定して前進させるプログラムは，Program9-7 のように RotateMotor 命令を使うと簡単になります。出力ポートＢとＣのモータをパワー値 30 で 614°回転させるプログラムです。

【Program9-7】

```
1  task main()
2  {
3      RotateMotor(OUT_BC,30,614);
4  }
```

3 行目の RotateMotor の(　)内の意味は（動かすモータの出力ポート名，モータのパワー値，モータの回転角度）です。回転角度をマイナスにすると後進します。

なお，RotateMotor はモータが指定された回転角度となるまで，次の命令に実行は移りません。

今度は，ロボットを左にターンさせてみましょう。左にターンさせるためには，右側のタイヤを前進，左側のタイヤを後進させます。このときのタイヤの動く道のりを上から見た様子を図９－９に示します。

左右のタイヤの間隔を s(mm)，ターンする角度を u°とすると，左右のタイヤは，直径 s(mm)の円を描くように移動していきます。その道のり x(mm)は，中心角 u°のおうぎ形の弧の長さになりますから，③式と同様に

$$x = (\pi \times s) \times u \div 360 \quad \cdots\cdots\cdots\cdots\cdots\cdots \quad ⑤$$

で求められます。直径 d(mm)のタイヤが x(mm)だけ移動する回転角度 t°は，④式を使って求められます。⑤式を④式に代入すると

$$t = x$$
$$= (\pi \times s) \times u \div 360 \times 360 \div (\pi \times d)$$
$$= s \times u \div d \quad \cdots\cdots\cdots\cdots\cdots\cdots \quad ⑥$$

となり，t は簡単な式で求められます。

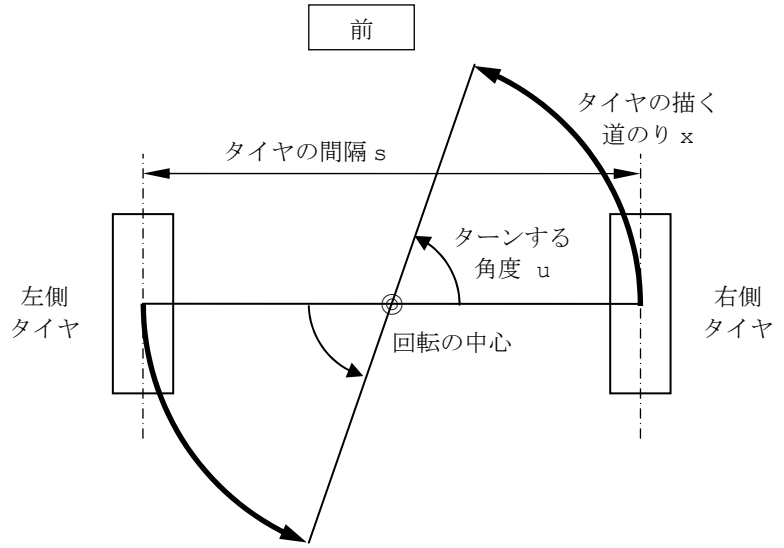


図９－９　ターンするときにタイヤが描く道のり（ロボットの上から見た図）

それでは，⑥式を使ってちょうど u=90°左にターンさせるための回転角度 t°を求めてみましょう。移動型三輪ロボットの左右のタイヤの中心から中心までの間隔を測ったところ，約 106mm でした。タイヤの直径 d=56mm とタイヤの間隔 s=106mm と u=90°を⑥式に代入して

```
t = 106×90÷56
  = 170.35 ·····
```

を得ます。170.35 の少数第１位を四捨五入して整数にし，指定する回転角度を 170°とします。

次に，Program9-6 の 6 行目にある OnFwd の OUT_BC を OUT_B に修正し，OnRev を追加したプログラムを Program9-8 に示します。8 行目の while の条件を「b < 170」として，約 90°だけ左にターンするようにしています。

Program9-8 を実行すると，ロボットが左にターンし停止します。液晶ディスプレイに表示された値は，約 170 と-170 に近い値になっています。

【Program9-8】

```
 1 task main()
 2 {
 3     int b,c;
 4     b = 0;
 5     ResetRotationCount(OUT_BC);
 6     OnFwd(OUT_B,30);
 7     OnRev(OUT_C,30);
 8     while(b < 170) {
 9         b = MotorRotationCount(OUT_B);
10         c = MotorRotationCount(OUT_C);
11         NumOut(16,16,b,true);
12         NumOut(64,16,c,false);
13     }
14     Off(OUT_BC);
15     while(true);
16 }
```

回転角度を指定してターンさせるプログラムは，Program9-9 のようにRotateMotorEx 命令を使うと簡単になります。出力ポートＢとＣのモータをパワー値 30 で互いに逆方向に 170°回転させるプログラムです。

【Program9-9】

```
1 task main()
2 {
3     RotateMotorEx(OUT_BC,30,170,-100,true,true);
4 }
```

3 行目　RotateMotorEx(OUT_BC,30,170,-100,true,true);

3 行目の RotateMotorEx の(　)内の意味は（動かすモータの出力ポート名，モータのパワー値，モータの回転角度，同期値，協調動作，即時停止）です。出力ポートＢとＣに接続された２つのモータをパワー値 30 で回転させます。同期値を-100，協調動作を true にすることで，出力ポートＢのモータを+170°回転させ，同時に，出力ポートＣに接続されたモータを-170°回転させます。即時停止を true に設定しているので，結果として，この命令を実行すると左にターンし停止します。

なお，右にターンさせたい場合は，同期値を+100 に設定します。

RotateMotorEx はモータが指定された回転角度となるまで，次の命令に実行は移りません。

【チャレンジ９－３】

Program9-7 を参考にして，ロボットが 50cm 前進し，30cm 後退するプログラムを作ってみましょう。

【チャレンジ９－４】

ちょうど 120°右にターンするためのモータの回転角度を求めてみましょう。求めた回転角度を使って 120°右にターンし，10cm 前進する動きを繰り返すプログラムを作ってみましょう。

９．４　サウンドセンサ

ロボットに耳をつけて，音に反応するロボットを作ってみましょう。耳のように音を検知するためのセンサをサウンドセンサと呼びます。

ここで使うサウンドセンサは，図９－１０に示すようにマイクが内蔵されています。

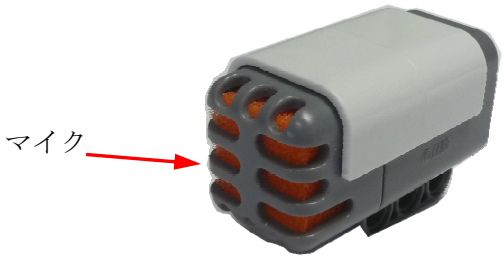


図９－１０　サウンドセンサ

サウンドセンサを使用して，周囲の音によって動きをかえるロボットを作りましょう。移動型三輪ロボットの組み立て説明図の 24～26 ページを参考にして，サウンドセンサを図１－２の移動型三輪ロボットに取り付け，入力ポート２番にケーブルで接続します。

プログラムを作成する前にサウンドセンサが正常に動作しているか確認しましょう。

NXT ブロックの左・右セレクトボタンを使って「View（見る）」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Sound dB（音，単位 dB）」を表示して，NXT ボタン（オレンジ色）を押します。続けて，サウンドセンサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，サウンドセンサで検知した周囲の音の強さが「0%（最も小さい音）」から「100%（最も大きな音）」で表示されます。元の状態に戻りたいときはクリアボタンを押していきます。

ここでは，ロボットが大きな音を検知するまで前進し続けるプログラム Program9-10 を示します。

【Program9-10】

```
 1 task main()
 2 {
 3     SetSensorSound(IN_2);
 4     OnFwd(OUT_BC,30);
 5     until(Sensor(IN_2) > 50);
 6     Off(OUT_BC);
 7 }
```

3 行目　SetSensorSound(IN_2);

どのようなセンサを接続しているか NXT ブロックに伝える命令です。SetSensorSound に続いて(IN_2)とあります。これは，「入力ポート２番はサウンドセンサです。」ということを設定する命令になっています。入力ポートの番号を変更するときは，IN_2 の数字を変えます。入力ポートの番号は 1 から 4 までです。

5 行目　until(Sensor(IN_2) > 50);

until 文は，指定された条件が成り立つまで命令を繰り返し実行します。ここでは，unitl 文に続く命令は何もありませんから条件「Sensor(IN_2) > 50」が成り立つまで何もせず待ち続けます。Sensor(IN_2)の値は，検知した音の大きさに応じて 0 から 100 となります。この文では Sensor(IN_2)の値が 50 を超えるまで待ち続けます。

すでに 4 行目の OnFwd(OUT_BC,30)という命令が実行されロボットは，パワー値 30 で前進していますから，その状態が保たれ，until 文を実行中していてもロボットは前進し続けます。

6 行目　Off(OUT_BC);

Sensor(IN_2)の値が 50 より大きくなると，6 行目に実行が移り，モータは停止します。

【チャレンジ９－５】

サウンドセンサの前で「手をパチンとたたくと」ごとに，約 20cm ずつ前進するロボットを作ってみましょう。

９．５　光センサ

ロボットに眼をつけて，外の光に反応したり，線の上をたどっていくロボットを作ってみましょう。眼のように光の強さを検知するためのセンサを光センサと呼びます。

ここで使う光センサは，図９－１１に示すように光を発する「発光素子」と光の強さを検知する「受光素子」で構成されています。NXT ブロック用光センサは，発光素子と受光素子が並んで配置されています。発光素子を利用しているときは，赤く光ります。受光素子に光が当たると，その強さに応じて電流が流れ，光の強さを検知できます。

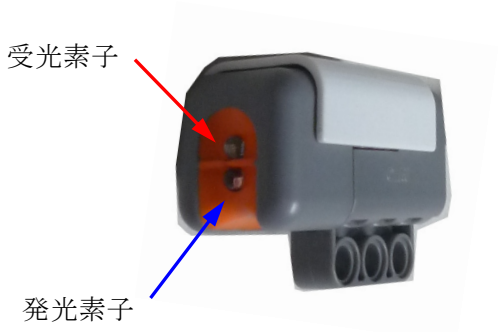


図９－１１　光センサ

光センサを使用して，周囲の光や色によって動きをかえるロボットを作りましょう。移動型三輪ロボットの組み立て説明図の 32～33 ページを参考にして，光センサを図１－２の移動型三輪ロボットの前方に取り付け，入力ポート３番にケーブルで接続します。接続した様子を図９－１２に示します。

（ａ）水平に取り付けた例　　（ｂ）下向きに取り付けた例

図９－１２　光センサを取り付けた移動型三輪ロボット

光センサの接続ができたら，プログラムを作成する前に光センサが次の二通りの方法で正常に動作しているか確認しましょう。

（１）周囲の光の状態を検知できるかを確認します。

NXT ブロックの左・右セレクトボタンを使って「View（見る)」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Ambient light（周囲の光)」を表示して，NXT ボタン（オレンジ色）を押します。続けて，光センサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，光センサで検知した周囲の光の強さが「0%（最も弱い光)」から「100%（最も強い光)」で表示されます。元の状態に戻りたいときはクリアボタンを押していきます。

（２）反射してきた光の状態を検知できるかを確認します。

NXT ブロックの左・右セレクトボタンを使って「View（見る)」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Reflected light（反射光)」を表示して，NXT ボタン（オレンジ色）を押します。続けて，光センサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，光センサの発光素子が赤く光ります。

この赤い光はとても強いので，直接眼で見ないようにしてください。

この光を当てて，そこから反射してきた光の強さが「0%（対象が黒で最も弱い反射光の場合)」から「100%（対象が白や鏡で最も強い反射光の場合)」で表示されます。元の状態に戻りたいときはクリアボタンを押していきます。

９．６　周辺の明るさを検知

周囲の明るさによって動きを変えるロボットを作ってみましょう。図９－１３のように光センサは，受光素子に当たる光の強さを検知します。

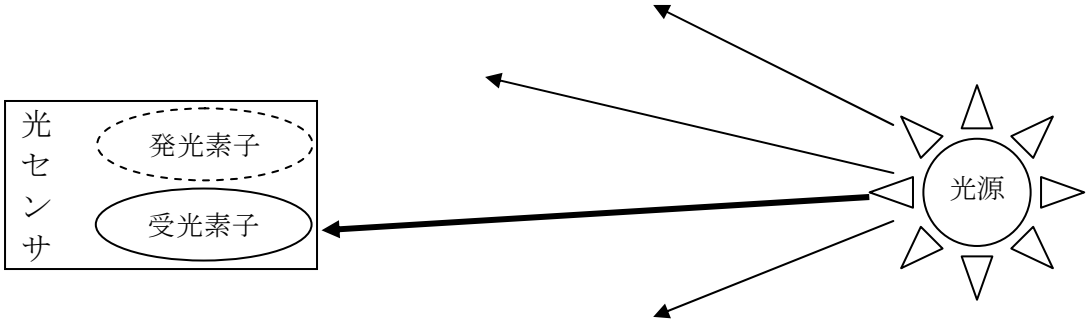


図９－１３　受光素子に当たる光の強さを検知

この機能を利用してロボットが明るいところから暗いところに移動したら，後退するプログラムを作ってみます。光センサは，図９－１２（ａ）のように水平に取り付けます。

Program9-11 は，光センサで検知された光の強さが 25%より小さくなる（暗くなる）と後退するように作られています。

【Program9-11】

```
 1 task main()
 2 {
 3     SetSensorLight(IN_3,false);
 4     OnFwd(OUT_BC,30);
 5     until(Sensor(IN_3) < 25);
 6     OnRev(OUT_BC,30);
 7     Wait(1000);
 8     Off(OUT_BC);
 9 }
```

3 行目　SetSensorLight(IN_3,false);

どのようなセンサを接続しているか NXT ブロックに伝える命令です。SetSensorLight に続いて(IN_3,false)とあります。これは，「入力ポート３番は光センサで，発光素子を発光させない」ということを設定する命令になっています。入力ポートの番号を変更するときは，IN_3 の数字を変えます。入力ポートの番号は 1 から 4 までです。

5 行目　until(Sensor(IN_3) < 25);

until 文は，指定された条件が成り立つまで命令を繰り返し実行します。ここでは，unitl 文に続く命令は何もありませんから条件「Sensor(IN_3) < 25」が成り立つまで何もせず待ち続けます。Sensor(IN_3)の値は，検知した光の強さに応じて 0 から 100 までとなります。この文は「光センサで感知された光の強さを示す Sensor(IN_3)が 25 よりも小さくなるまで待つ」という命令となります。ロボットが暗いところに移動して，光が弱くなるまで前進を続けます。

6 行目　OnRev(OUT_BC);

5 行目の until 文の次に実行する命令です。ロボットが暗いところに移動したら，後退します。

Program9-11 の 5 行目の until 文の条件に指定した 25 という光センサの値は，一例です。この値は周りの光の状態によって変える必要があります。周りがもともと明るかったら数値を大きくする必要があります。反対に，暗ければ数値を小さくしなければなりません。

NXT ブロックの「View」を使って，明るいところと暗いところにロボットを置いて，それぞれの光センサの値を記録して，その平均値とします。

この値の変更をしやすくするために Program9-12 のように記号定数を使うと，わかりやすいプログラムになります。

【Program9-12】

```
 1 #define THRESHOLD 25
 2
 3 task main()
 4 {
 5     SetSensorLight(IN_3,false);
 6     OnFwd(OUT_BC,30);
 7     until(Sensor(IN_3) < THRESHOLD);
 8     OnRev(OUT_BC,30);
 9     Wait(1000);
10     Off(OUT_BC);
11 }
```

1 行目　#define THRESHOLD 25

ここで使われている記号定数 THRESHOLD は「境界値」または「しきい値」という意味です。ある状態を二通りに判断するときに使われる値に使われる用語です。

7 行目　until(Sensor(IN_3) < THRESHOLD);

光センサの値 Sensor(IN_3)が THRESHOLD より小さくなるまで待ちます。

【チャレンジ９－６】

ロボットの光センサに懐中電灯を当てて，その方向にロボットが進むプログラムを作ってみましょう。

第１０章　ライントレース・ロボット

ラインをたどっていくロボットを作ってみましょう。このような動きをするロボットのことを「ライントレース・ロボット」といいます。この章では,「ライントレース・ロボット」の作り方を説明します。

１０．１　反射した光を検出

図１０－１に示すように小判状の黒いラインを印刷したテスト用紙を使って，ラインをたどって動くロボットを作ってみましょう。

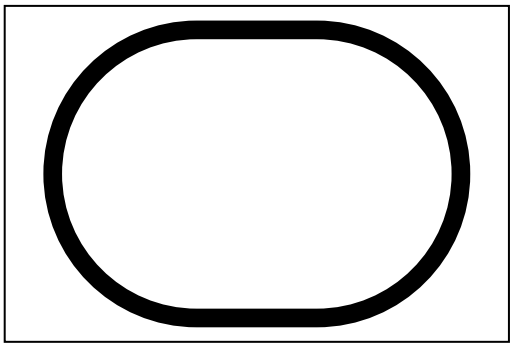


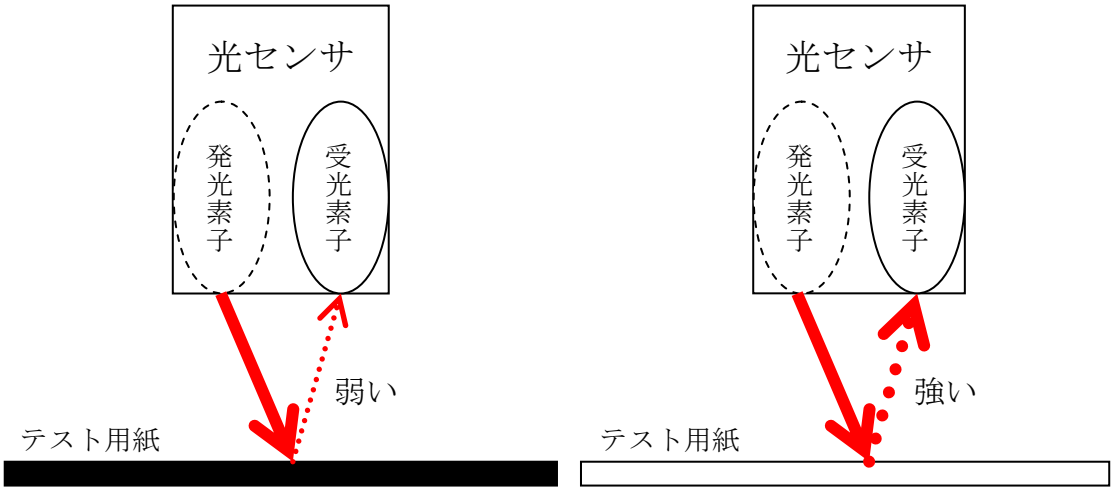
図１０－１　小判状の黒いラインを印刷したテスト用紙

テスト用紙に印刷された黒いラインを検知するために光センサを下向きに取り付けます（図９－１２（ｂ））。

光センサは発光素子と受光素子を使って，テスト用紙の黒いラインと白い部分を識別します。発光素子が出した光がテスト用紙に当たり，反射した光の強さを受光素子で検知し，その変化からテスト用紙の色を識別します。

図１０－２を見てみましょう。この図の➔は，発光素子の発した光を示し，⇢は，テスト用紙から反射した光を示しています。図１０－２（ａ）に示すように，黒いラインの上では黒色の性質から，発光素子の光が吸収されてしまい反射する光の量が少なくなります。そのため，受光素子が感知する光は弱くなり，光センサの値は小さくなります。一方，図１０－２（ｂ）に示すように，白い部分の上では白色の性質から，発光素子の光は強く反射されます。その結果，黒いラインの場合と比較して，受光素子が感知する光は強くなり，光センサの値は大きくなります。

この性質を利用して，光センサを使って，黒いラインの上か白い部分の上かを判断します。



（ａ）　黒いライン　　（ｂ）　白い部分

図１０－２　反射する光の強さの変化

１０．２　ラインをたどる方法

図１０－１のテスト用紙に描かれた黒いラインに沿って動くライントレース・ロボットのプログラムを考えましょう。ライントレース・ロボットは，ロボットがラインの上にあることをいつも検知しながら，ラインからはずれると移動する方向を修正しながら，ラインの片側をなぞるように動きます。

ここでは，テスト用紙のラインに沿って時計回りにロボットが移動していくプログラムを作ります（図１０－３）。始めにロボットはライン上にあるとします。ロボットが前進していくとラインから離れ，はずれてしまうことがあります。そのときは，ラインに戻すためにロボットの方向を変えて，ラインのあるところに移動させなければなりません。

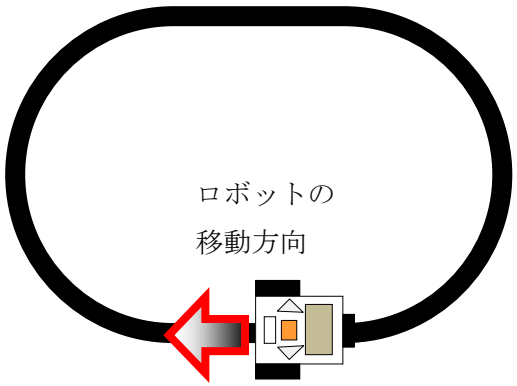


図１０－３　時計回りにライントレース

ラインをたどっていく方法のひとつとして，「ラインの左側をなぞっていく」考え方があります。最初に，ロボットは，ライン上にあるとします。図１０－４のように，ロボットがライン上にあるときは，ラインの左側をなぞるために「左にカーブ」させます。その後，ロボットが白い部分にきたら逆に「右にカーブ」させます。この動作を繰り返すことで，図１０－４のようにロボットは，ラインの左側をジグザグになぞっていくように移動し，結局ラインをたどることになります。

このようにしてライントレースするロボットの動きをフローチャートで表すと図１０－５のようになります。

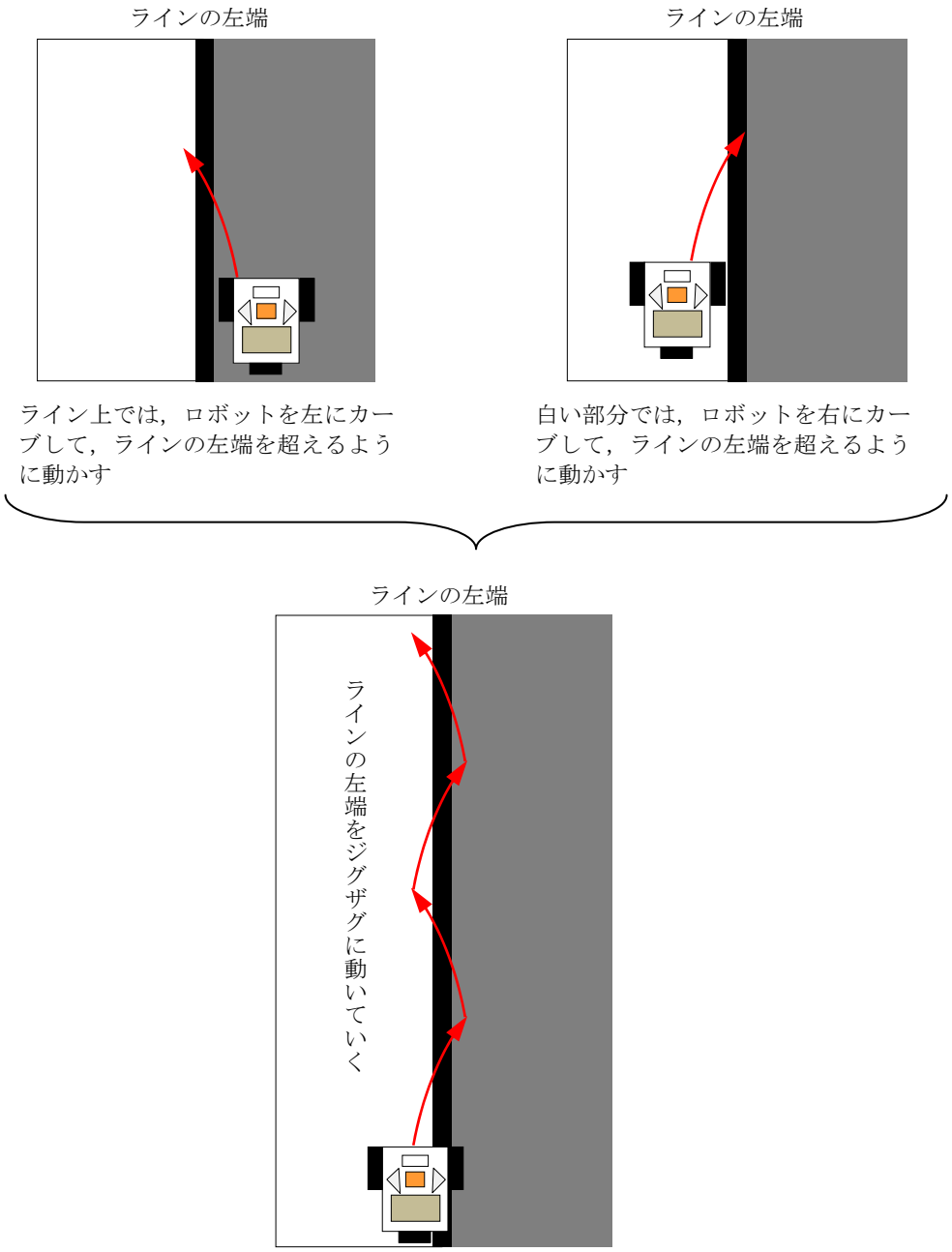


図１０－４　ラインの左端をなぞっていくライントレースの考え方

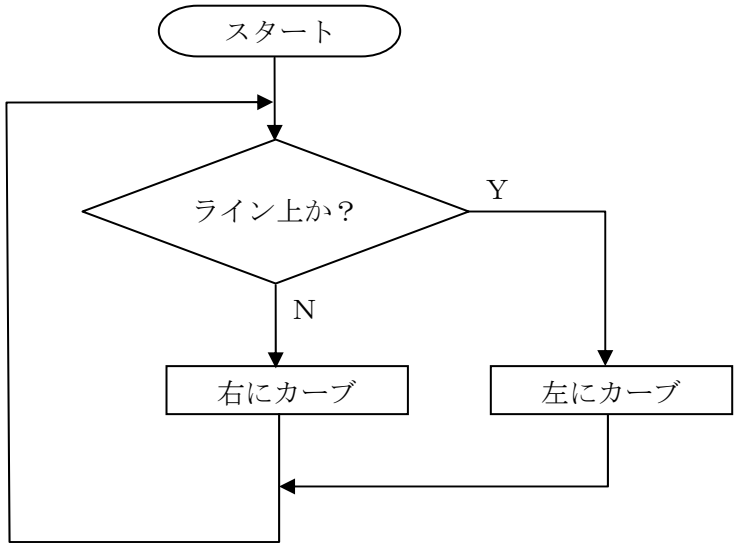


図１０－５　ラインの左端をなぞるライントレースのフローチャート

ロボットがラインの上にあるのか，それとも，ラインからはずれているのか判別するために光センサを使います。図１０－２を使って説明したように，ラインの上にあるときは，ラインが黒いことから反射した光が弱くなり光センサの値が小さくなります。逆に，ラインからはずれているときは，テスト用紙の色が白いことから反射した光が強くなり光センサの値が大きくなります。

光センサの値によって，ライン上にあるかどうか判断することができます。しかし，それを判断するための境界値はどのように決めればよいのでしょうか？　部屋の明るさやテスト用紙への照明方法，人の影などによって光センサの値は，かなり変化します。そこで，ライン上と白い部分での光センサの値を実際に実験して決めます。

NXT ブロックの左・右セレクトボタンを使って「View（見る)」を表示して NXT ボタン（オレンジ色）を押し，再び左・右セレクトボタンを使って「Reflected light（反射光)」を表示して，NXT ボタン（オレンジ色）を押します。続けて，光センサを接続した入力ポートの番号を左・右セレクトボタンを使って表示した後，NXT ボタン（オレンジ色）を押します。すると，光センサの発光素子が赤く光ります。

この赤い光はとても強いので，直接眼で見ないようにしてください。

この光を当てて，そこから反射してきた光の強さが「0%（対象が黒で最も弱い反射光)」から「100%（対象が白や鏡で最も強い反射校)」で表示されます。この状態で，光センサをライン上と白い部分に当てる実験をします。

この実験の結果を表１０－１にまとめてみましょう。適切な境界値を決めるためには，３回ぐらい実験して光センサの値を読んで，平均した値を使います。

表１０－１　ライン上と白い部分の光センサの値

回	ライン上	白い部分
1		
2		
3		
平均		

次に，ライン上と白い部分の境界値を，

（ライン上の光センサの平均値　＋　白い部分の光センサの平均値）÷２

で求め，小数第１位を四捨五入して整数値にしましょう。

この境界値は，ちょうど，２つの値の中間となります。境界値よりも小さいとき，ライン上にあり，逆に大きいとき，ラインからはずれ白い部分にあることがわかります。

図１０－５のフローチャートを参考にして，ここで求めた境界値を記号定数 THRESHOLD とするプログラムを Program10-1 に示します。ここでは，例として境界値を 55 としています。この値は，実験の結果から求めたものに修正しましょう。

【Program10-1】

```
 1 #define THRESHOLD 55
 2 #define SP        30
 3
 4 task main()
 5 {
 6     SetSensorLight(IN_3,true);
 7
 8     while(true) {
 9         if(Sensor(IN_3) < THRESHOLD) {
10             OnFwd(OUT_B,SP);        // ライン上は，左カーブ
11             Off(OUT_C);
12         } else {
13             Off(OUT_B);             // 白い部分は，右カーブ
14             OnFwd(OUT_C,SP);
15         }
16     }
17 }
```

1 行目　#define THRESHOLD 55

光センサがラインの上にあるかどうかを判別するための境界値 55 を記号定数 TRESHOLD とします。この境界値は実験して決めます。

2 行目　#define SP　　30

モータのパワー値 30 を記号定数 SP とします。モータのパワー値の最大値は 100 ですが，あまりモータを速く回転させるとうまくラインをトレースできなくなるので，実験しながらちょうどよい値にしましょう。

6 行目　SetSensorLight(IN_3,true);

どのようなセンサを接続しているか NXT ブロックに伝える命令です。SetSensorLight に続いて(IN_3,true)とあります。これは，「入力ポート３番は光センサで，発光素子を発光する」ということを設定する命令になっています。入力ポートの番号を変更するときは，IN_3 の数字を変えます。入力ポートの番号は 1 から 4 までです。

8 行目～16 行目　while(true) {・・・}

while 文の条件が true になっているので，無限に実行を繰り返すように命令されています。

9 行目　if(Sensor(IN_3)<THRESHOLD) {

もし，入力ポート３番に接続した光センサで検出した値 Sensor(IN_3)が THRESHOLD で定められた境界値よりも小さいとき，この if 文に続く 10，11 行目を実行して左にカーブさせます。この条件は，「光センサがラインを検出したとき」にあたります。さもなければ，Sensor(IN_3)が THRESHOLD 以上の場合となり，12 行目の else より下の 13，14 行目を実行して右にカーブさせます。この条件は，「光センサが白い部分を検出したとき」にあたります。

【チャレンジ１０－１】

Program10-1 では，モータのパワー値が 30 でやや遅くなっています。モータのパワー値をもう少し大きくして，ライントレースさせてみましょう。そのときに，どんな問題がおきるでしょうか？

【チャレンジ１０－２】

Program10-1 では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

図１０－５のフローチャートによるライントレースは，ロボットが常にラインの境界部分を境にしてカーブして移動するので，「ジグザグ」と進み，あまり速くライントレースすることができません。モータのパワー値を 30 から 100 にしても，ロボットの移動速度は速くなりますがジグザグする動作は変わりません。何かよいアイディアはないでしょうか？

Program10-1 では，左右にカーブしながらライントレースしていました。ラインがゆるい曲線であれば，もっとゆるい角度でターンすれば，なめらかに動作するでしょう。ターンする角度をゆるくするためには，停止していたモータを遅い速度で回転させる方法があります。

例えば，左にカーブする命令

```
OnFwd(OUT_B,60);
Off(OUT_C);
```

を

```
OnFwd(OUT_B,60);
OnFwd(OUT_C,25);
```

にするとカーブする角度がゆるくなります。このとき，速く回転する方のモータを「主」遅く回転する方のモータを「副」と呼ぶことにします。

Program10-1 を修正して，カーブする角度をゆるくしたプログラム（Program10-2）を入力しましょう。

【Program10-2】

```
 1 #define THRESHOLD 55
 2 #define SP_MA     60
 3 #define SP_SV     25
 4
 5 task main()
 6 {
 7     SetSensorLight(IN_3,true);
 8
 9     while(true) {
10         if(Sensor(IN_3) < THRESHOLD) {
11             OnFwd(OUT_B,SP_MA);   // ライン上は，左にゆるくカーブ
12             OnFwd(OUT_C,SP_SV);
13         } else {
14             OnFwd(OUT_B,SP_SV);   // 白い部分は，右にゆるくカーブ
15             OnFwd(OUT_C,SP_MA);
16         }
17     }
18 }
```

2 行目　#define SP_MA　　60

3 行目　#define SP_SV　　25

ゆるくカーブするときの主となるモータのパワー値 60 を記号定数 SP_MA に，副となるモータのパワー値 25 を記号定数 SP_SV にします。SP_SV を 0 に定義すると「カーブ」と同じ動作になります。モータのパワー値の最大値は 100 ですが，あまりモータを速く回転させるとうまくラインをトレースできなくなるので，実験しながらちょうどよい値にしま

しょう。

11 行目　OnFwd(OUT_B,SP_MA);
12 行目　OnFwd(OUT_C,SP_SV);
　出力ポートＢに接続されているモータを「主」にして前進回転させるので，左にゆるくカーブしていきます。

14 行目　OnFwd(OUT_B,SP_SV);
15 行目　OnFwd(OUT_C,SP_MA);
　出力ポートＣに接続されているモータを「主」にして前進回転させるので，右にゆるくカーブしていきます。

【チャレンジ１０－３】

Program10-2 では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

　もっと速くラインをたどる方法はないのでしょうか？　次は，ロボットの過去の状態も考慮してみます。光センサがライン上にあるときは，できるだけ前進させて，白い部分にあるときは，できるだけ速くライン上に戻すように考えてみましょう。

　そのために，図１０－６のようにロボットの過去の状態（①）と現在の状態（②）に応じた４通りのパターンで考えます。

　まず，ライン上からライン上に移動する【パターン１】では，すでにロボットはライン上にあるので，とてもゆるく左にカーブさせてできるだけ前進させます。直進してしまうとラインの右端を超えることがあるので，とてもゆるく左にカーブさせます。

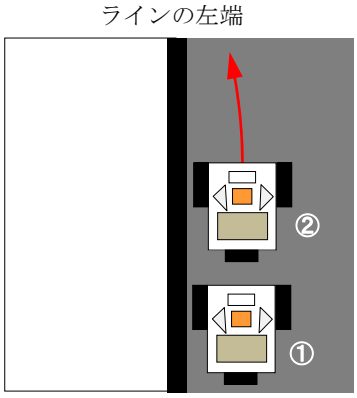
　ライン上から白い部分に入った【パターン２】では，ライン上にあったロボットが少しだけ，ラインを外れたことになるので，ゆるく右にカーブさせて，ライン上に戻すようにします。

白い部分からライン上に入った【パターン３】では，ライン上に入ったロボットの向きを少し修正するために，ゆるく左にカーブさせて，ライン上を移動するようにします。

白い部分から白い部分を移動する【パターン４】では，ロボットがラインを外れて移動しているので，できるだけ速くライン上に戻るように右にカーブさせています。

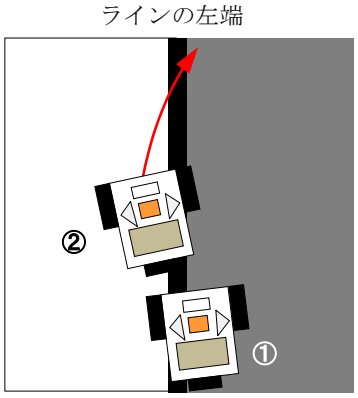
この考え方をフローチャートに描いた図を図１０－７に示します。

【パターン１】①ライン上　⇒　②ライン上



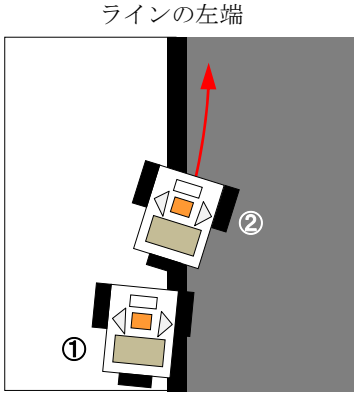
とてもゆるく左にカーブさせ，できるだけ前進させる

【パターン２】①ライン上　⇒　②白い部分



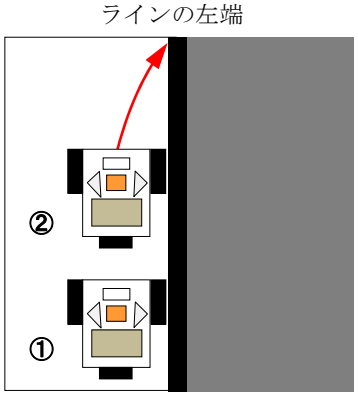
ゆるく右カーブさせ，ライン上に戻るようにさせる

【パターン３】①白い部分　⇒　②ライン上



ゆるく左にカーブさせ，ライン上を前進させる

【パターン４】①白い部分　⇒　②白い部分



右にカーブさせ，できるだけ速くライン上に戻るようにさせる

図１０－６　ロボットの過去の状態（①）と現在の状態（②）を考慮したライントレース

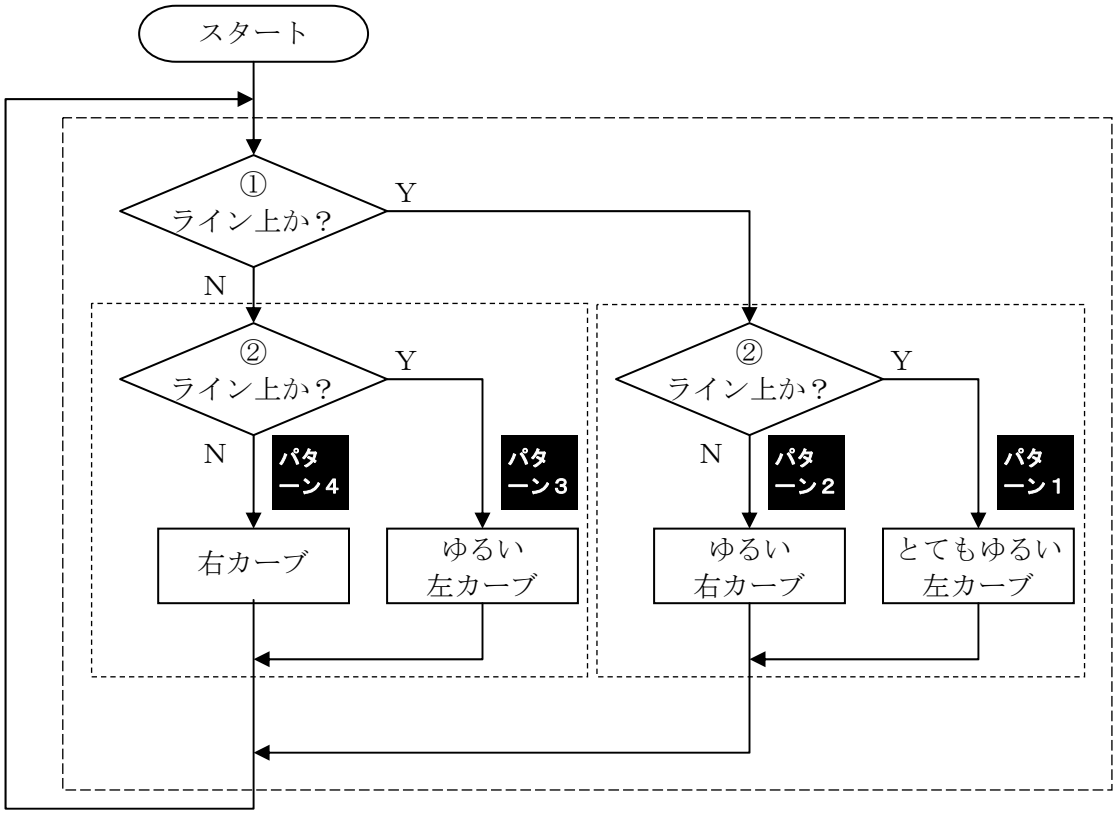


図１０－７　ロボットの過去の状態（①）と現在の状態（②）を考慮したライントレースのフローチャート

それでは，このフローチャートに従った動きをするためのプログラム（Program10-3）を入力しましょう。このプログラムでは，if 文の中にさらに if 文を入れて，４通りのパターンを判断しています。

【Program10-3】

```
 1 #define THRESHOLD 55
 2 #define SP_MA     60
 3 #define SP_SV     35
 4
 5 int s1,s2;
 6
 7 task main()
 8 {
 9     SetSensorLight(IN_3,true);
10
```

```
11      s1 = 1;
12
13      while(true) {
14          s2 = 0;
15          if(Sensor(IN_3) < THRESHOLD) {
16              s2 = 1;
17          }
18
19          if (s1==1) {
20              if (s2==1) { // ライン上→ライン上：とてもゆるい左カーブ
21                  OnFwd(OUT_B,SP_MA);
22                  OnFwd(OUT_C,SP_MA-5);
23              } else {     // ライン上→白い部分：ゆるい右カーブ
24                  OnFwd(OUT_B,SP_SV);
25                  OnFwd(OUT_C,SP_MA);
26              }
27          } else {
28              if (s2==1) { // 白い部分→ライン上：ゆるい左カーブ
29                  OnFwd(OUT_B,SP_MA);
30                  OnFwd(OUT_C,SP_SV);
31              } else {     // 白い部分→白い部分：右カーブ
32                  Off(OUT_B);
33                  OnFwd(OUT_C,SP_MA);
34              }
35          }
36          s1 = s2;
37      }
38  }
```

5 行目　int s1,s2;

状態のパターンを判別するための整数型の変数 s1 と s2 を宣言しています。s1 が過去の状態（①）を示し，この値が 1 だと「ライン上」，0 だと「白い部分」とします。s2 は現在の状態（②）を示します。

11 行目　s1 = 1;

最初にロボットはライン上にありますので,「ライン上」を意味する 0 を s1 に入れています。

13 行目～37 行目　while(true) {・・・}

ライントレース処理の全体を無限に実行するための while 文です。

```
14 行目  s2 = 0;
15 行目  if(Sensor(IN_3) < THRESHOLD) {
16 行目      s2 = 1;
17 行目  }
```

まず，現在の状態を示す変数 S2 の値を 0（「白い部分」）に一旦します。その後，現在の光センサの値 Sensor(IN_3)で読み取り，THRESHOLD より小さいときは，変数 S2 の値を 1 にして「ライン上」であることを記録します。

19 行目～27 行目～35 行目　if (s1==1){・・・} else {・・・}

この if 文で「パターン１と２」,「パターン３と４」の判断します。

20 行目～23 行目～26 行目　if (s1==1){・・・} else {・・・}

この if 文で,「パターン１」と「パターン２」を判断し，図１０－６の処理を行います。

28 行目～31 行目～35 行目　if (s1==1){・・・} else {・・・}

この if 文で,「パターン３」と「パターン４」を判断し，図１０－６の処理を行います。

```
36 行目  s1 = s2;
```

現在の状態を示す変数 s2 の値を，過去の状態を示す変数 s1 に代入して，状態を変化させます。

【チャレンジ１０－４】

Program10-3 では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

１０．３　ラインの両端を使ってたどる方法

ここまでは，ラインの片方の端だけをなぞってラインをトレースする方法でした。誤ってなぞっている端と異なる方を検知すると，ライントレースする方向が逆転してしまうという問題もありました。

ラインの幅が太いときは，図１０－８のようにライントレースを速めるために，できるだけライン上のロボットを前進させるようにし，ラインを外れたときだけ，元のラインに戻るような動きにすることを考えてみましょう。

この例では，ライン上ではいつも前進するようにします（①）。ラインから外れて白い部分になったら，ロボットをターンさせてラインを探索します（②）。ラインを見つけたら再び前進します（③）。再度，ラインから外れて白い部分になったら，ロボットターンさせてラインを探索します（④）。ラインが見つかったら前進に戻ります（⑤）。この動きを繰り返すことで，ラインの両端を使ってすばやくライントレースができるはずです。

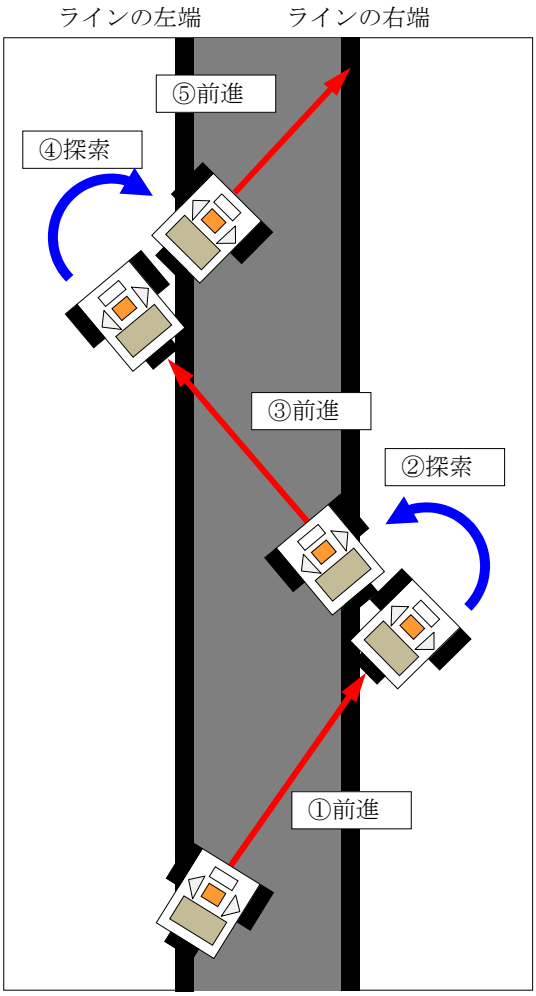


図１０－８　ライン上を前進していくライントレースの考え方

ところが，図１０－８で示した②と④のラインの探索処理は，どちら側にターンすればよいかわかりません。光センサが１つしかないので，ラインの左端と右端を区別できないからです。

そこで，図１０－９のようなラインの探索方法を考えてみましょう。まず最初にある一定時間だけ「左ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ａ）に示します。この状態でラインを見つけることができなかったら，今度は，ターンする時間を増やして「右ターン」しながらラインを探索します。ターンする時間を増やすことで，探索する範囲を広げています。この時間内でラインが見つかった場合を図１０－９（ｂ）に示します。

それでも，ラインが見つからないときは，さらにターンする時間を増やして再び「左ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ｃ）に示します。まだ，ラインが見つからないときは，さらにターンする時間を増やして再び「右ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ｄ）に示します。

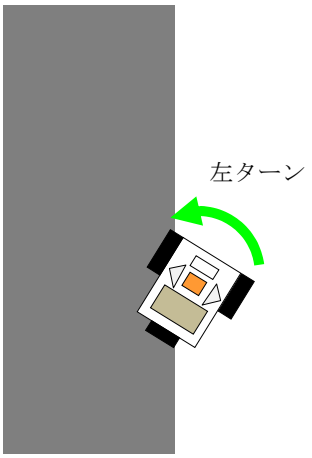
このように，ある一定時間だけ「右ターン」と「左ターン」を交互に行いながら，ラインを探索していくことで，ラインの左端でも右端でも見つけることができます。

光センサが１つのときは，ラインの探索は難しくなりますが，プログラムの工夫にやりがいはあります。

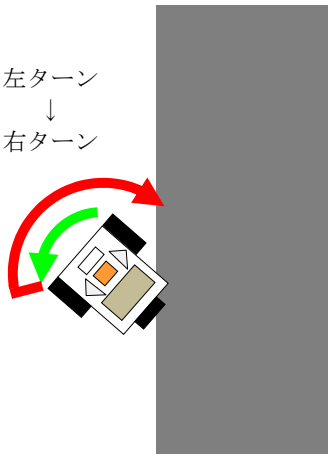
ライン上を前進するライントレース・ロボットのフローチャートを図１０－１０に示します。ロボットは，最初ライン上にあるので，前進します（①）。はじめに探索するターンの方向を一旦「左」にしておきます。「右」にしてもライントレースはできますが，ここでは左にしています。

つぎに，ロボットがラインを外れて白い部分に来ているかどうかを判断します（②）。もし，白い部分になったら，ラインの探索を始めます（③）。探索時間を 0 に初期化しておいてから，探索するターンの方向に従って，「左ターン」または「右ターン」を開始します（④）。探索時間を増加させた後，探索時間内で，光センサを使ってライン上にあるかどうかを検知します。探索時間内でライン上になれば，前進します（⑤）。探索時間を超えたら，探索するターンの方向を「左ターン」ならば「右ターン」，「右ターン」ならば「左ターン」のように交換します（⑥）。ラインを探索できなかったかどうか判断し（⑦），探索できなかったら，再び探索のためのターンを開始します。このとき，ターンの方向は逆になっています。

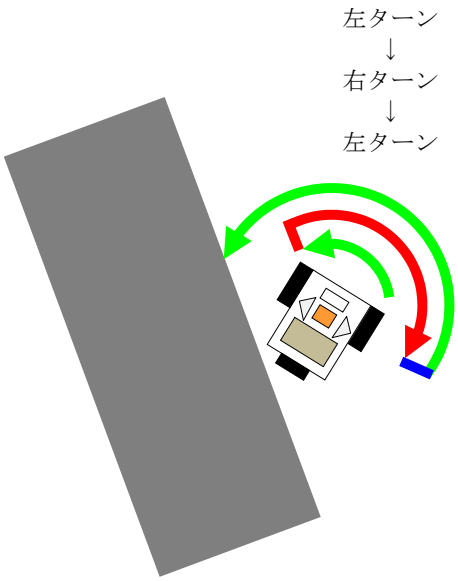
まっすぐなライン上を右端⇒左端⇒右端⇒左端をたどりながら前進する場合，探索のためのターンの方向が，ちょうど交互になり，無駄な探索のためのターンが減ります。



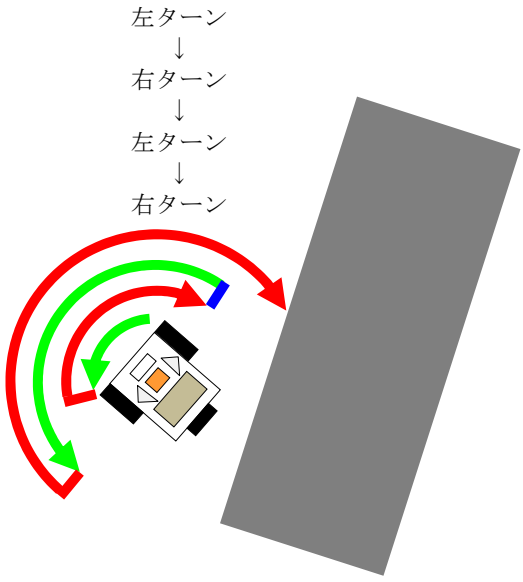
（ａ）左ターンのみでラインを見つける場合



（ｂ）左ターンでラインが見つからないので，途中で右ターンしてラインを見つける場合



（ｃ）左ターンでラインが見つからず，途中で右ターンに変更しても，まだ見つからないときは，再度，探索範囲を広げて左ターンして，ラインを見つける場合



（ｄ）左ターンでラインが見つからず，途中で右ターンに変更しても，まだ見つからないときは，再度，探索範囲を広げて左ターンする。それでもラインが見つからないときはさらに探索範囲を広げて右ターンしてラインを見つける場合

図１０－９　白い部分からラインを探索する方法

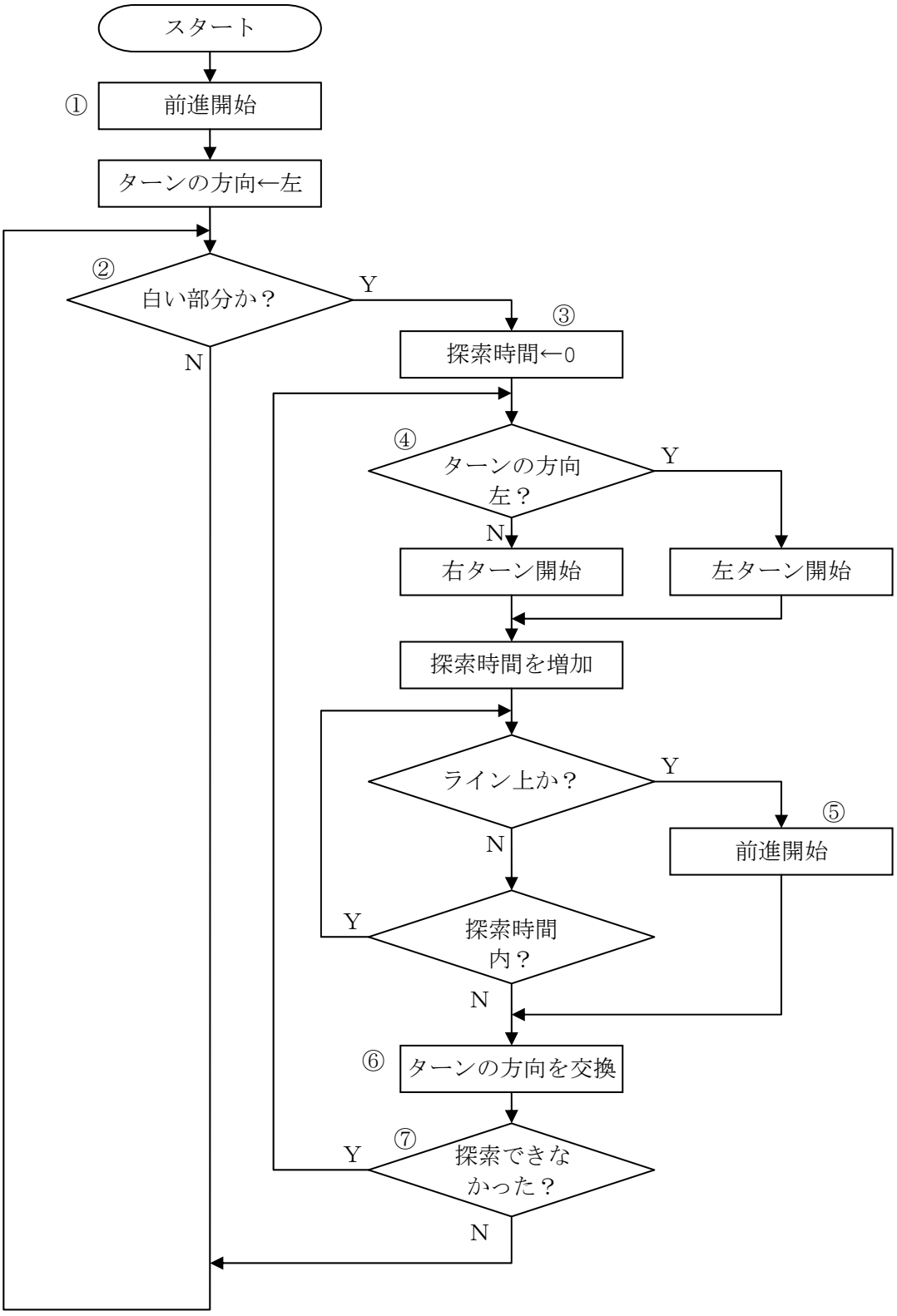


図１０－１０　ライン上を前進するライントレースのフローチャート

それでは，このフローチャートに従った動きをするためのプログラム（Program10-4）を入力しましょう。このプログラムは繰り返しや判断が何重にもなっているので，段付けするためのスペースは２個にしています。

NXT ブロックは，内部に時計をもっていて，その時刻を知ることができます。この時刻は実際の時刻とは異なり，NXT ブロックがオンになってから計時されはじめ，0.001 秒間隔でカウントアップされています。CurrentTick()は，NXT 時刻を示しています。この時刻は大きな数値となり，また，マイナスはありませんから，その変数の型は「unsigned long」とします。NXT 時刻を使って探索のためのターンする時間を制限することができます。

【Program10-4】

```
 1 #define THRESHOLD 55
 2 #define SP         50
 3 #define S_TICK   100
 4
 5 task main()
 6 {
 7   SetSensorLight(IN_3,true);
 8   int flag;
 9   int dir = 1;        // 一番はじめだけ左ターンして探索
10   unsigned long ct,ti;
11
12   OnFwd(OUT_BC,SP);  // 最初はライン上にあるので前進
13   while(true) {
14     if (Sensor(IN_3)>=THRESHOLD) { // ラインから外れて白い部分に来た
15       ti = 0;
16       do {
17         if (dir==1) {
18           OnFwd(OUT_B,SP);  // 左ターンしてラインを探索
19           OnRev(OUT_C,SP);
20         } else {
21           OnRev(OUT_B,SP);  // 右ターンしてラインを探索
22           OnFwd(OUT_C,SP);
23         }
24         flag = 0;            // まだ，ラインは見つかっていない
25         ti += S_TICK;        // 探索時間を長くしていく
26         ct = CurrentTick(); // 現在の NXT 時刻
```

```
27          do {
28            if (Sensor(IN_3) < THRESHOLD) {
29              OnFwd(OUT_BC,SP);     // ライン上は，前進
30              Wait(200);
31              flag = 1;
32              break;
33            }
34          } while(CurrentTick() - ct < ti);  // 探索時間中
35          dir = -dir;  // ターンの方向を交換
36        } while(flag==0);
37      }
38    }
39  }
```

3 行目　#define S_TICK 100

探索のためターンする時間（0.001 秒の単位）を記号定数 S_TICK に定義しています。ここでは，100 ティック（0.1 秒間）としていますが，実験してちょうどよい値に変更しましょう。図１０－９で説明したように，探索するための時間は，段々と増えていきます。この増分も S_TICK で示しています。

8 行目 int flag;

整数型の変数 flag は，探索してラインが見つかったら 1 を，見つからなかったら 0 を入れて使います。

9 行目 int dir = 1;

整数型の変数 dir は，探索するためのターンの方向を示します。「左ターン」は 1，「右ターン」は-1 とします。1 と-1 を使う理由は，左と右を交互に変化させるためにプラスとマイナスの符号を変えることに対応させるためです。変数の宣言時に「dir = 1」とすることで，dir を 1 で初期化しています。

10 行目　unsigned long ct,ti;

「unsigned long」の「unsigned long」は符号を考えない大きな整数型を示しています。ct は，現在の NXT 時刻を 0.001 秒の単位で記憶し，ti は，探索する時間を 0.001 秒の単位で記憶しています。

12 行目　OnFwd(OUT_BC,SP);

まず最初にロボットを前進させます。

13 行目～38 行目　while(true) {・・・}

この while 文は無限に実行するようになっています。この中に，ラインを外れたときの処理を書いています。

14 行目の if 文は，白い部分であるかどうかを判断しています。

15 行目　ti = 0;

一旦，探索する時間を 0 にしておきます。25 行目の ti += S_TICK;で，探索する時間を増加させていきます。

16 行目～36 行目　do {・・・} while(flag==0)

この do-while 文を使って，ラインの探索処理を行っています。while の条件は，flag==0 となっています。ラインの探索に成功したとき flag は 1 となっている（31 行目）いるので，ラインが見つからない限り繰り返し探索のためのターンを実行します。

17 行目～20 行目～23 行目　if (dir==1){・・・} else {・・・}

dir の値が 1 ならば，左ターンを開始し，さもなければ，右ターンを開始します。

24 行目　flag = 0;

ラインの探索の前に，ラインが見つかっていないという意味の 0 を変数 flag に入れます。

25 行目　ti += S_TICK;

ti + S_TICK の値を求めた後，改めて ti に代入ことで，探索する時間を S_TICK だけ増加させていきます。

26 行目　ct = CurrentTick();

現在の NXT ブロック内の時刻（0.001 秒単位）を ct に入れます。この値は，34 行目の while 文の条件の中で使います。

27 行目～34 行目　do {・・・} while(CurrentTick() - ct < ti)

この do-while 文を使って，決められた探索時間（ti）中にラインを見つけます。while 文の条件 CurrentTick() - ct < ti は，現在の NXT ブロック内の時刻 CurrentTick() から探索を開始したときの NXT ブロックの時刻を引き算し，探索に要している時間 CurrentTick()-ct を求め，その値が決められた探索時間（ti）よりも小さい間，探索

をします。もし，決められた探索時間を過ぎてしまったら，ラインを見つけていなくても，この while 文を終わり 35 行目に実行が移ります。

28 行目～33 行目の if 文は，ラインを見つけたことを判断します。

29 行目　OnFwd(OUT_BC,SP);
30 行目　Wait(200);
　ラインが見つかると，前進を再開し，200 ティック（0.2 秒間）だけ待ちます。この時間だけロボットは必ず前に移動します。

31 行目　flag = 1;
　ラインが見つかったことを示す 1 を変数 flag に入れます。

32 行目　break;
　この break 文によって，一番内側の繰り返し処理（ここでは，27 行目～34 行目 do-while 文に対応します）を中断し，35 行目に実行が移ります。このように，ある条件を満たしていたら，繰り返しをやめたいときに break 文を使います。

35 行目　dir = -dir;
　変数 dir の値が 1 のとき，-1 となり，逆に，-1 のとき 1 になります。この計算結果を使って，探索するターンの方向の左右を交換しています。

【チャレンジ１０－５】

Program10-4 では，まっすぐな太いラインの左端と右端で交互にターンしながら前進していくと，かなり速くライントレースできます。しかし，小判型のテスト用紙のようなラインでは，曲線部における探索のためのターンに無駄な動きが目立ちます。このような無駄を減らし，いろいろな形のラインでももっと速くライントレースするためには，どのような工夫をしたらよいでしょうか？

＜ヒント＞

「左ターンでラインを見つけた回数をカウントする変数 n_left」と「右ターンでラインを見つけた回数をカウントする変数 n_right」を使って，探索を開始するときの方向を決める方法があります。この方法を使うときは，n_left+n_right が指定した回数（例えば 16）を超えたら，n_left と n_right を 0 にして状態をたまにリセットするとよいでしょう。

チャレンジ１０－５のプログラム例を Program10-5 に示します。網掛け部分によく注意してプログラムを読みましょう。また，このプログラムからフローチャートを書いてみましょう。

【Program10-5】

```
#define THRESHOLD 55
#define SP         50
#define S_TICK   100
#define N_LIM     12

task main()
{
  SetSensorLight(IN_3,true);
  int flag;
  int dir;
  unsigned long ct,ti;
  int n_left = 0;       // 左ターンでライン探査できた回数
  int n_right = 0;      // 右ターンでライン探査できた回数

  OnFwd(OUT_BC,SP);  // 最初はライン上にあるので前進
  while(true) {
    if (Sensor(IN_3)>=THRESHOLD) { // ラインから外れて白い部分に来た
      dir = 1;
      if (n_left < n_right) { // 左右のターンでライン探査できた回数
        dir = -1;              // を比較して，探査する方向を決める
      }
      if (n_left + n_right > N_LIM) { // ライン探査カウンタのリセット
        n_left = 0;
        n_right = 0;
      }

      ti = 0;
      do {
        if (dir==1) {
          OnFwd(OUT_B,SP);  // 左ターンしてラインを探索
          OnRev(OUT_C,SP);
        } else {
          OnRev(OUT_B,SP);  // 右ターンしてラインを探索
          OnFwd(OUT_C,SP);
        }
        flag = 0;         // まだ，ラインは見つかっていない
        ti += S_TICK;  // 探索時間を長くしていく
```

```
38          ct = CurrentTick(); // 現在の NXT 時刻
39          do {
40            if (Sensor(IN_3) < THRESHOLD) {
41              OnFwd(OUT_BC,SP);     // ライン上は，前進
42              Wait(200);
43              flag = 1;
44              if (dir==1) {
45                n_left++;  // 左ターンでライン探査できた回数を 1 増加
46              } else {
47                n_right++; // 右ターンでライン探査できた回数を 1 増加
48              }
49              break;
50            }
51          } while(CurrentTick() - ct < ti);  // 探索時間中
52          dir = -dir;  // ターンの方向を交換
53        } while(flag==0);
54      }
55    }
56  }
```

【チャレンジ１０－６】

大きな白い厚紙の上に黒のテープなどを使って，いろいろな太さで直線や曲線のラインを作って，「ライントレース・ロボット」を使ったタイムレースをしてみましょう。

付　　録

計測・制御用
プログラム言語 NXC

リファレンスガイド

計測・制御用プログラム言語 NXC　リファレンスガイド

ここでは，プログラム言語 NXC で使える制御文や定数，変数，命令などの一部をまとめました。本テキストで取り扱っていない内容もありますので，その使い方はインターネットや他の NXC に関する解説書を参考にしましょう。

（1）制御文

制　　御　　文	説　　明
while(条件式) { }	条件式が真の間{ }の内容を繰り返す
do{ } while(条件式)	先に{ }を実行し，条件式が真の間{ }の内容を繰り返す
for(文 1; 条件式; 文 2) { }	文 1 を実行し，条件式が真の間{ }の内容と文 2 を繰り返す
until(条件式) { }	条件式が偽の間{ }の内容を繰り返す
break	無条件で繰り返し処理，switch 文から抜け出る
continue	無条件で次の繰り返し処理を実行する
repeat(回数) { }	回数だけ{ }の内容を繰り返す
if(条件式) { }	条件式が真であれば{ }の内容を実行する
if(条件式) {文 1} else {文 2}	条件式が真であれば{文 1}を実行し，偽であれば{文 2}を実行する

switch(式){
case 定数 1: 文 1
…
case 定数 n: 文 n
default: 文 d
}
	式の値が定数 1 のとき文 1 を実行し，同様に，式の値が定数 n のとき文 n を実行する。どの定数とも等しくないときは，文 d を実行する

start タスク名	指定したタスクを開始させる
stop タスク名	指定したタスクを停止する
インライン関数名(引数)	指定された引数でインライン関数を呼び出す

（2）条件式

条　件　式	説　　明
true	いつも真
false	いつも偽
式1 == 式2	式1と式2が等しいとき真
式1 != 式2	式1と式2が異なるとき真
式1 < 式2	式1が式2より小さいとき真
式1 <= 式2	式1が式2以下のとき真
式1 > 式2	式1が式2より大きいとき真
式1 >= 式2	式1が式2以上のとき真
! 条件式	条件式を否定する。条件式が真ならば偽となり，偽ならば真となる
条件式A && 条件式B	条件式Aと条件式Bが両方とも真のときだけ，全体の条件式が真となる
条件式A || 条件式B	条件式Aが真，または，条件Bが真のとき，全体の条件式が真となる

（3）定数

定　　数	説　　明
整数	1や2などの10進数表記の値,0xで始まる場合は16進数表記の値となる
実数	0.5や-0.04などの10進数表記の実数値
文字	'A'などのように，半角英数字を一重引用符'で囲む
文字列	"abc"などのように，文字列を二重引用符"で囲む

（4）変数の型

変数の型	説　　明
bool, byte, unsigned char	符号なし8ビット整数（0～255）
char	符号付き8ビット整数（-128～127）
unsigned int	符号なし16ビット整数（0～65535）
short, int	符号付き16ビット整数（-32768～32767）
unsigned long	符号なし32ビット整数（0～4294967296）
long	符号付き32ビット整数（-2147483648～2147483647）
float	約7桁の精度をもつ実数
string	文字列

（5）演算式

演　算　式	説　　明	例
+	加算	a + b
-	減算	a - b
*	乗算	a * b
/	除算	a / b
%	剰余（あまり）	a % b
++	1 だけ増加（変数のみ）	a++
--	1 だけ減少（変数のみ）	a--
-	-1 をかける	-a
(条件式) ? 式 1:式 2	条件式が真のとき式 1 となり，さもなければ式 2 の値	(a==b) ? 1 : -1
abs( )	絶対値	abs(a)
sign( )	符号値（正の値のとき+1，負の値のとき-1）	sign(a)

（6）演算子

演　算　子	説　　明	例
=	代入	c = a
+=	加算した後，代入	c += a c = c + a と同じ
-=	減算した後，代入	c -= a c = c - a と同じ
*=	乗算した後，代入	c *= a c = c * a と同じ
/=	除算した後，代入	c /= a c = c / a と同じ
%=	剰余（余り）を求めた後，代入	c %= a c = c % a と同じ

（7）プリプロセッサ

プリプロセッサ	説　　明
#include "ファイル名"	指定されたファイル名の内容を読み込む
#define 記号定数 文字列	記号定数を文字列に置換
#define マクロ定義(引数) マクロ文字列	マクロ定義を引数を含めたマクロ文字列に置換

（8）命令（API 関数，インライン関数）

命令（API 関数，インライン関数）		説　　明
センサ		
SetSensorTouch(入力ポート)		入力ポートをタッチセンサに設定する
SetSensorLight(入力ポート,モード)		入力ポートを光センサに設定する。発光素子はモードが true のときオン，false のときオフ
SetSensorUltrasonic(入力ポート)		入力ポートを超音波センサに設定する
SetSensorSound(入力ポート)		入力ポートをサウンドセンサに設定する
Sensor(入力ポート)	タッチセンサ	接触していないとき 0，接触しているとき 1 を返す
	光センサ	明るさに応じた値 0～100 を返す
	サウンドセンサ	音の大きさに応じた値 0～100 (dB) を返す
SensorUS(入力ポート)	超音波センサ	計測距離の値 0～255 (cm) を返す
ResetRotationCount(出力ポート)		出力ポートに接続された回転センサの回転角度を初期化
MotorRotationCount(出力ポート)		出力ポートに接続された回転センサの回転角度を返す
モータ		
OnFwd(出力ポート,パワー値)		出力ポートに接続されたモータをパワー値(0～100)で前進させる。
OnRev(出力ポート,パワー値)		出力ポートに接続されたモータをパワー値(0～100)で後退させる。
Off(出力ポート)		出力ポートに接続されたモータの出力をオフにする
Float(出力ポート)		出力ポートに接続されたモータの出力を滑走状態にする
MotorPower(出力ポート)		出力ポートに設定したパワー値を返す
RotateMotor(出力ポート,パワー値,回転角度)		出力ポートに接続されたモータをパワー値(0～100)で，回転角度だけ回転する
RotateMotorEx(出力ポート,パワー値,回転角度,同期値,協調動作,即時停止)		２つの出力ポートに接続されたモータをパワー値(0～100)で，回転角度だけ回転する。同期値が 0 のとき同じ方向，+100 または-100 のとき逆方向に回転する。協調動作と即時停止は true に設定する

命令（API 関数）	説　　明
サウンド	
PlaySound(サウンド番号)	指定されたサウンド番号の音を鳴らす
PlayTone(周波数,時間)	指定された周波数の音を指定時間(0.001 秒単位)だけ鳴らす。
PlayTone(周波数,時間,音量,繰り返し)	指定された周波数の音を指定時間(0.001 秒単位)，音量(0～4)で鳴らす。繰返が true のとき繰り返し音を鳴らす
PlayFile(ファイル名)	サウンドファイルを鳴らす
PlayFile(ファイル名,音量,繰り返し)	音量(0～4)でサウンドファイルを鳴らす。繰返が true のとき繰り返し音を鳴らす
SetSoundVolume(音量)	音量(0～4)を設定する
SoundVolume()	設定した音量を返す
StopSound()	音を鳴らすことを停止する
液晶ディスプレイ	
NumOut(x,y,値,オプション)	座標(x,y)に値を数字で表示する。オプションとして false の場合上書き，true の場合表示内容をすべて消去してから表示する。その他多くの表示オプションがある
TextOut(x,y,文字列,オプション)	座標(x,y)に文字列を表示する。オプションは NumOut を参照
PointOut(x,y,オプション)	座標(x,y)に点を描画する。オプションは NumOut を参照
LineOut(x1,y1,x2,y2,オプション)	座標(x1,y1)から座標(x2,y2)まで直線を描画する。オプションは NumOut を参照
RectOut(x,y,w,h,オプション)	左上座標(x,y)の幅 w と高さ h の長方形を描画する。オプションは NumOut を参照
CircleOut(x,y,r,オプション)	中心座標(x,y)の半径 r の円を描画する。オプションは NumOut を参照
EllipseOut(x,y,rx,ry,オプション)	中心座標(x,y)の横半径 rx と縦半径 ry のだ円を描画する。オプションは NumOut を参照
ResetScreen()	液晶ディスプレイを消去する

命令（API 関数）	説　　明
その他	
Wait(時間)	0.001 秒(ティック)単位で指定された時間だけ待機する
CurrentTick()	0.001 秒(ティック)単位のシステム時刻を返す
Precedes(タスク 1,...,タスク n)	タスク 1 から n までを同時に実行する
StopAllTasks()	実行しているすべてのタスクを停止する
sqrt(x)	float 型 x の平方根を返す
Random(n)	0～n-1 までの一様乱数
システム	
BatteryLevel()	電池の電圧を 0.001[V]の単位で返す

（9）命令（API 関数，インライン関数）用定数

設定項目	定　数　名	説　　明
入力ポート	IN_1	入力ポート１番
	IN_2	入力ポート２番
	IN_3	入力ポート３番
	IN_4	入力ポート４番
出力ポート	OUT_A	出力ポートＡ
	OUT_B	出力ポートＢ
	OUT_C	出力ポートＣ
	OUT_AB	出力ポートＡとＢ
	OUT_AC	出力ポートＡとＣ
	OUT_BC	出力ポートＢとＣ
	OUT_ABC	出力ポートＡとＢとＣ

（10）予約語

予約語とは，あらかじめ NXC で使うことが決められている単語です。予約語と同じ名前の記号定数や変数，タスクなどを使うことはできません。

予　　約　　語
asm, bool, break, byte, case, char, const, continue, default, do, if, else, enum, false, float, for, goto, inline, int, long, mutex, priority, repeat, return, safecall, short, start, static, stop, string, struct, sub, switch, tasks, true, typedef, unsigned, until, void, while,（__で始まる asm 用キーワード）

著者紹介

伊 藤　陽 介（いとう　ようすけ）

1987年　徳島大学大学院工学研究科修士課程修了，博士（工学）

機械製造業，高等専門学校勤務を経て，現在，鳴門教育大学大学院学校教育研究科生活・健康系コース（技術・工業・情報）教授。情報工学と情報技術教育に関する研究に従事。「『プログラムと計測・制御』のためのロボット学習材の開発と実践」等，日本産業技術教育学会誌に論文発表するとともに，学術講演会等において研究成果を多数発表。主な著書：初心者のためのＣ言語プログラミング（共立出版），情報処理（森北出版）。

謝　辞

本テキストを制作するにあたり，石塚仁志さん（2005年度修了，2003年度卒業），森誉範さん（2006年度修了），船橋知里さん（2008年度卒業），中山詩衣奈さん（2010年度卒業）にご協力いただきました。

計測・制御用プログラム言語NXC入門

2012年 5月 6日　暫定版

著　者　　伊 藤　陽 介

発行者　　鳴門教育大学
技術・工業・情報コース
伊藤研究室

鳴門教育大学

学校名					
学　年		組		番号	
氏　名					