
LEGO MindStorm の NXC による制御プログラミング

平成 30 年 9 月 26 日版
西井 淳

目 次

1	はじめに	2
2	準備	2
2.1	LEGO MindStorm のドライバインストール	2
2.2	NXC のインストール	2
2.3	LEGO MindStorm の準備	3
3	MindStorm を動かしてみよう	3
3.1	Bricx を起動する	3
3.2	プログラムを書いてみよう	3
3.3	プログラムを実行する	4
3.4	プログラムにエラーがおきた時	4
3.5	モータのスピードを変える	5
4	プログラミング入門	5
4.1	変数を使う	5
4.1.1	演算子と基本的な関数	5
4.1.2	配列	6
4.1.3	変数の使用例	6
4.1.4	乱数	7
4.2	制御構文	7
4.2.1	条件式	7
4.2.2	if 文, while 文	8
4.2.3	do~while 文	8
4.2.4	repeat 文	9
4.2.5	until 文	10
4.3	センサー	10
4.3.1	センサ入力を待つ	11
4.3.2	タッチセンサを使う	11
4.3.3	光センサを使う	11
4.3.4	カラーセンサを使う	12
4.3.5	音センサを使う	13
4.3.6	超音波センサを使う	14
4.4	Task と Functions	14
4.4.1	Tasks	14
4.4.2	関数 (Functions)	15
4.4.3	引数のアドレス渡し	16
4.4.4	並列タスク	16
4.5	LCD ディスプレイ表示	17
4.6	ファイル入出力	18
5	発展的なプログラミング	20
5.1	LED ライトを点灯する	20
5.2	音楽を鳴らす	20
5.3	関数のマクロ定義	21
5.4	inline 関数	22
A	用意されている数学関数	23
B	トラブル対策	23

1 はじめに

LEGO MindStorm は LEGO が MIT と共同開発したロボット制御の学習用キットです。本テキストでは、LEGO MindStorm をプログラミング言語 NXC(Not eXactly C) を使って制御する方法を説明します。NXC は C 言語と似ていますが、簡単に MindStorm を制御できるように C 言語と異なる点もいろいろあります。また、NXC プログラミングのための統合開発環境 Bricx も開発されています。

なお、本テキストは Daniele Benedettelli 著の “Programming LEGO NXT Robots using NXC”, John Hansen 著の ”Not eXactly C (NXC) Programmer’s manual” をもとにして短くまとめたものです。いずれも <http://bricxcc.sourceforge.net/nbc/nxcdoc/> からダウンロードできますし、Bricx にも添付されているので、詳細を知りたい人は見てください。



2 準備

2.1 LEGO MindStorm のドライバインストール

以下の方法で LEGO MindStorm のドライバをダウンロード・インストールしてください。

1. 西井のホームページの「計算モデル論実習 II」のページで”MindStorm ドライバのインストール方法”を参照し、”LEGO MindStorms Update page” をクリックする。
もしくは [Http://mindstorms.lego.com/support/files/](http://mindstorms.lego.com/support/files/) にアクセスする。
2. 表示されたページの、左のメニュー項目 Drivers”をクリック
3. 右に表示される “Fantom Driver” をクリックしてダウンロード
4. ダウンロードしたらファイルを展開後、展開フォルダ内にある “setup.exe” をダブルクリックすれば ドライバのインストーラを起動できます。インストールは表示される指示に従うだけでできます。必要ならば西井のホームページの「計算モデル論実習 II」のページにある” ドライバインストールガイド” を参照してください。

2.2 NXC のインストール

以下のいずれかの方法で BricxCC をダウンロード後、インストールしてください。

方法 1 西井のホームページの「計算モデル論実習 II」のページで”Bricx(Windows 専用) の最新版” をクリック

方法 2 <http://bricxcc.sourceforge.net/> で “latest version” をクリック

ダウンロードしたらインストーラ (brixcc_setup_3389.exe といった名前) を実行して、Yes か Next をひたすら選択すればインストール完了。

Bricx の起動は、スタートメニューの “Bricx Command Center” をクリック。

2.3 LEGO MindStorm の準備

LEGO MindStorm の本体の A ポートに右モータが、 C ポートに左モータが接続されている事を確認してください。

3 MindStorm を動かしてみよう

3.1 Bricx を起動する

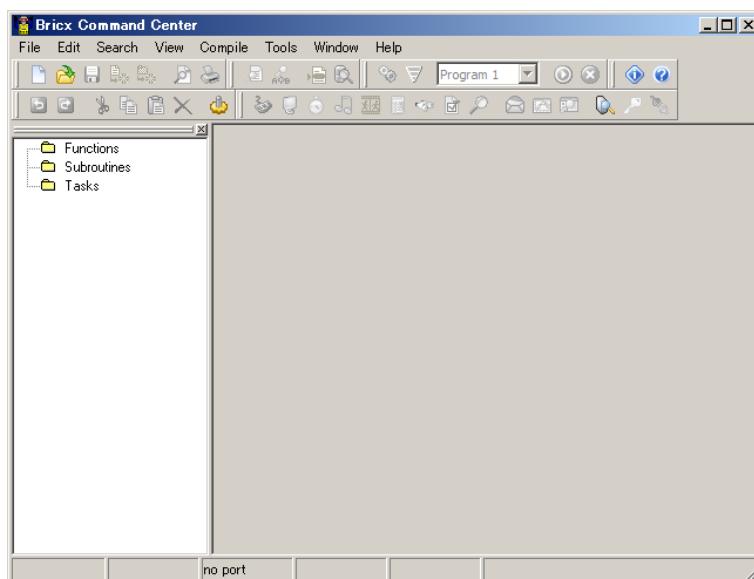


Bricx を初めて起動すると上のような、 LEGO MindStorm 接続先を設定する画面が出てきます。 LEGO MindStorm を USB ケーブルでパソコンへ接続し、以下のように設定して [OK] を押してください。

- Port : usb
- Brick Type : NXT
- Firmware : Standard

接続先を設定せずに Bricx を起動したり、 LEGO MindStorm を接続せずに [OK] を押した場合などには、接続先の再設定が必要な場合があります。この場合 Bricx メニューの “Tools” より “Find Brick” を選択すると、上記の画面が出てきて再設定ができます。

3.2 プログラムを書いてみよう

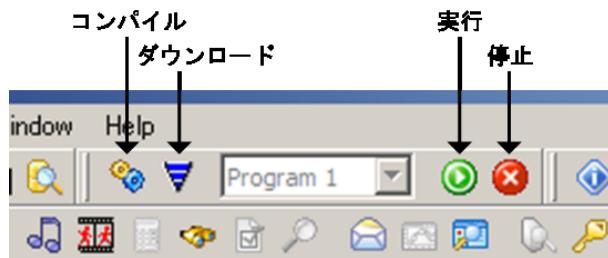


Bricx を起動すると上のような画面がでてきます。“File” メニューから “New” を選択して以下のようなプログラムを書いてみましょう。

```
i
task main()
{
    OnFwd(OUT_A, 75); // A ポート (OUT_A) のモータを 75% の力で前進方向 (Forward) に動かす
    OnFwd(OUT_C, 75); // C ポート (OUT_C) のモータを 75% の力で前進方向 (Forward) に動かす
    Wait(4000); // 4 秒 (4000ms) 待つ
    OnRev(OUT_AC, 75); // A, C ポート (OUT_AC) のモータを 75% の力で逆向き (Reverse) に動かす
    Wait(4000); // 4 秒待つ
    Off(OUT_AC); // A, C ポートのモータを止める
}
```

各行において//より後の文はコメントとみなされます。C 言語と同様に/* */ではさむこともできます。

3.3 プログラムを実行する

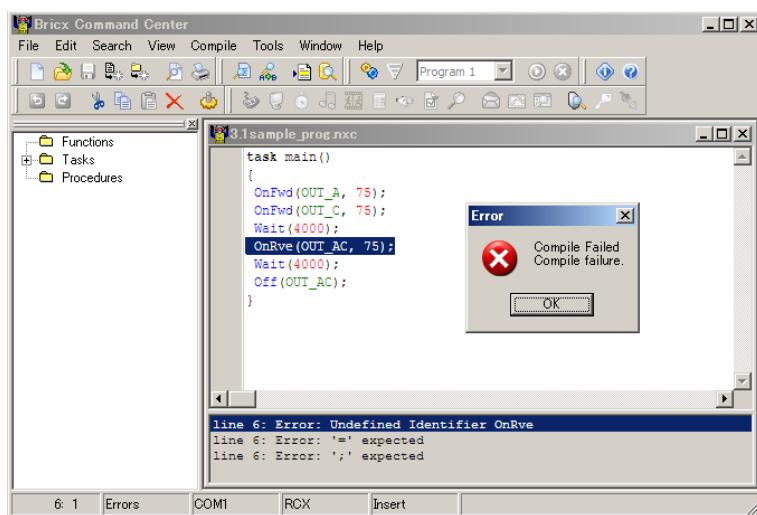


Bricx の上部のメニュー ボタンを左から順に押せば, “compile” (コンパイル), “download” (MindStorm へ ダウンロード), “run” (実行), “stop” (停止) となります。

プログラムを MindStorm にインストールした後は, USB ケーブルをはずして MindStorm のボタンを操作してもプログラムの実行はできます。

3.4 プログラムにエラーがおきた時

コンパイル時にエラー行がハイライト表示されます。



3.5 モータのスピードを変える

モータのスピードを変えるのは簡単です。以下の例ではモータの出力を 30 % にし、また前の例よりもプログラムが短くなるように修正してあります。

```
task main()
{
    OnFwd(OUT_AC, 30);
    Wait(4000);
    OnRev(OUT_AC, 30);
    Wait(4000);
    Off(OUT_AC);
}
```

[問 3.1] 少しバックしてから向きを変えて 3 秒前進するプログラムを作りましょう。

4 プログラミング入門

4.1 変数を使う

NXC では、C 言語のように変数を使うこともできます。NXC には以下のような変数の型があります。

byte	... 整数型 (8bit unsigned, 0~255)
int	... 整数型 (16bit, -32,768~32,767)
unsigned int	... 整数型 (16bit unsigned, 0~65,535)
short	... 整数型 (16bit, int と同じ)
long	... 整数型 (32bit, -2,147,483,648~2,147,483,647)
unsigned long	... 整数型 (32bit unsigned, 0~4,294,967,295)
float	... 浮動小数点型 (32bit)
bool	... true か false の二値をとる変数型
string	... 文字変数型
struct	... 構造体

NXC には浮動小数的型の変数がないことに気をつけましょう。

4.1.1 演算子と基本的な関数

NXC では以下のような演算子や関数を使うことができます。

オペレータ	意味	例
abs()	絶対値	abs()
sign()	符号	sign()
++, --	値を 1 増やす, 減らす	$x ++$
!	NOT	$!x$
*, /, %	掛け算, 割り算, 余り	$x * y$
+, -	足し算, 引き算	$x + y$
<, >	比較	$x < y$
==, !=		$x == 1$
&&	AND	$x \& \& y$
	OR	$x y$

4.1.2 配列

配列は以下のように宣言します。配列の各要素は 0 に初期化されます。

```
task main()
{
    int myArray[10][10];      //10 × 10 の配列を宣言
    int myVector[10];         //大きさ 10 の配列を宣言
}
```

4.1.3 変数の使用例

[問 4.1] 下のプログラムを修正して `values[0]～values[3]` の値を MindStorm の LCD ディスプレイに表示するようにしなさい。

```

int aaa;
int bbb,ccc;
int values[];
task main()
{
    aaa = 5;
    bbb = 2 * aaa;
    ccc = bbb;
    ccc /= 2;
    aaa *= (ccc + 3);
    ArrayInit(values, 0, 10);      //配列 values を大きさ 10 として各要素値を 0 にする
    values[0] = aaa;
    values[1] = bbb;
    values[2] = aaa*bbb;
    values[3] = ccc;
    NumOut(10, LCD_LINE1, aaa); //LCD ディスプレイの 1 行目 (LCD_LINE1), 左から 10 画素の位置に数値表示
    NumOut(10, LCD_LINE2, bbb); //LCD ディスプレイの 2 行目 (LCD_LINE2), 左から 10 画素の位置に数値表示
    NumOut(10, LCD_LINE3, ccc); //LCD ディスプレイの 3 行目 (LCD_LINE3), 左から 10 画素の位置に数値表示
    Wait(2000);                  //2 秒経ったら終わり
}

```

4.1.4 亂数

以下は関数 `Random()` を使って乱数を発生したプログラム例です。関数 `Random()` の詳細やその他の数学関数については A 節を参照してください。

```

int move_time, turn_time;
task main()
{
    while(true)          // 無限ループ
    {
        move_time = Random(600); // 0 から 600 の範囲で乱数発生
        turn_time = Random(400); // 0 から 400 の範囲で乱数発生
        OnFwd(OUT_AC, 75);
        Wait(move_time);
        OnRev(OUT_A, 75);
        Wait(turn_time);
    }
}

```

4.2 制御構文

4.2.1 条件式

`if` 文等の条件式には以下の表現が使えます。

条件式	意味
true	常に真
false	常に偽
expr	expr の値が 0 でなければ真
expr1==expr2	以下いずれも C 言語と同様
expr1 != expr2	
expr1 < expr2	
expr1 <= expr2	
expr1 > expr2	
expr1 >= expr2	
! condition	
cond1 && cond2	
cond1 cond2	

4.2.2 if 文, while 文

if 文や while 文も C 言語と同様に使えます。

[問 4.2] 以下のプログラムの動作を説明しなさい。

```
#define MOVE_TIME    500 // C 言語と同様にマクロ定義も使用可（以下の行にある MOVE_TIME は
// 500 に置き換えられる）
#define TURN_TIME    360
task main()
{
    while(true)
    {
        OnFwd(OUT_AC, 75);
        Wait(MOVE_TIME);
        if (Random() >= 0)      // Random() は int 型 (-32,768~32,767) の乱数を返す
        {
            OnRev(OUT_C, 75);
        }
        else
        {
            OnRev(OUT_A, 75);
        }
        Wait(TURN_TIME);
    }
}
```

4.2.3 do~while 文

以下のように do~while 文も使うことができます。

```
do
{
  ...
}
while (condition);
```

[問 4.3] 以下のプログラムの動作を説明しなさい。

```
int move_time, turn_time, total_time;
task main()
{
  total_time = 0;
  do
  {
    move_time = Random(1000);
    turn_time = Random(1000);
    OnFwd(OUT_AC, 75);
    Wait(move_time);
    OnRev(OUT_C, 75);
    Wait(turn_time);
    total_time += move_time;
    total_time += turn_time;
  }
  while (total_time < 20000);
  Off(OUT_AC);
}
```

do 文ではまず do に続く {} 内の命令を一度実行した後に、while() で指定した条件がテストされることに気をつけましょう。

4.2.4 repeat 文

repeat(n){}を使うと、n 回同じことをくり返す命令を作れます。

```
#define MOVE_TIME      500
#define TURN_TIME      500
task main()
{
  repeat(4)
  {
    OnFwd(OUT_AC, 75);
    Wait(MOVE_TIME);
    OnRev(OUT_C, 75);
    Wait(TURN_TIME);
  }
  Off(OUT_AC);
}
```

repeat 文は、多重ループにすることもできます。

```
#define MOVE_TIME 1000
#define TURN_TIME 500
task main()
{
    repeat(10)
    {
        repeat(4)
        {
            OnFwd(OUT_AC, 75);
            Wait(MOVE_TIME);
            OnRev(OUT_C, 75);
            Wait(TURN_TIME);
        }
    }
    Off(OUT_AC);
}
```

4.2.5 until 文

```
until (condition)
{
    ...
}
```

until 文は、条件が満たされている限り{}内の文がくり返し実行されます。

4.3 センサー

MindStorm では様々なセンサーを利用できます。ロボットの入力ポート 1 にタッチセンサを、入力ポート 2 に光センサを、入力ポート 3 にカラーセンサを、入力ポート 4 に超音波センサを接続してください。カラー センサは地面に向けてつけましょう。



タッチセンサ



音センサ



光センサ



超音波センサ



カラーセンサ

4.3.1 センサ入力を待つ

以下は何かにぶつかるまで直進するプログラム例です。

```
task main()
{
    SetSensor(IN_1,SENSOR_TOUCH); //入力ポート1にタッチセンサが接続
    OnFwd(OUT_AC, 75);
    until (SENSOR_1 == 1); //SENSOR_1は入力ポート1のセンサ値を示す
    Off(OUT_AC);
}
```

関数 `SetSensor()` は MindStorm のどの入力ポートにどのようなセンサが接続されているかを指定します。入力ポートの指定はポート番号に応じて `IN_1` から `IN_4` で行います。

4.3.2 タッチセンサを使う

タッチセンサが物体に接触しているかどうかを 1 (接触), 0 (非接触) のセンサ値で知ることができます。以下は障害物に当たったら少し後ろに下がってから向きを変えて動き続けるプログラム例です。

```
task main()
{
    SetSensorTouch(IN_1); //SetSensor (IN_1, SENSOR_TOUCH)と同じ意味
    OnFwd(OUT_AC, 75);
    while (true)
    {
        if (SENSOR_1 == 1) //センサ値が1(衝突時)かどうかチェック
        {
            OnRev(OUT_AC, 75); Wait(300);
            OnFwd(OUT_A, 75); Wait(300);
            OnFwd(OUT_AC, 75);
        }
    }
}
```

関数 `SetSensorTocuch()` は接触センサーがどのポートに接続されているかを指定する命令です。

4.3.3 光センサを使う

光センサを使うと、明るさを 0 から 100 までの数値で知ることができます。以下は明るい時には前進、暗い時には後向きに回転するプログラムです。光センサの値は MindStorm の LCD ディスプレイに表示されます。LCD ディスプレイへの表示方法の詳細は 4.5 節を参照してください。

```
#define THRESHOLD 40
task main()
{
    int s;
    SetSensorLight(IN_2,false); //入力ポート 2 に光センサを接続.
    while (true)
    {
        s=Sensor(IN_2);           //Sensor(IN_2) のかわりに SENSOR_2 としてもよい
        TextOut(15, LCD_LINE1,"Light:", true); // 1 行目 (LCD_LINE1) の左から 15 画素の位置に文字列表示
        NumOut(60, LCD_LINE1, s);      // 1 行目 (LCD_LINE1) の左から 60 画素の位置にセンサ値表示
        if (s>THRESHOLD) {
            OnFwd(OUT_AC, 75);
        } else {
            OnRev(OUT_C, 75);
        }
    }
}
```

関数 `SetSensorLight()` は光センサーがどのポートに接続されているかを指定する命令です。関数 `Sensor()` は指定した入力ポートからの入力値を返り値とする関数です。

光センサーの横の LED が発光することで反射光の強度を測りたいときには、`SetSensorLight()` の第二変数で `SetSensorLight(IN_2,true)` と指定するか、これを省略して `SetSensorLight(IN_2)` とします。LED は消灯したままでよければ `SetSensor(IN_2, SENSOR_LIGHT)` を代わりに使うことも出来ます。

[問 4.4] 上記のプログラムを改造して、明るいところでは右に、暗いところでは左にロボットが曲がるプログラムを作ってみましょう。THRESHOLD の値は周囲の明るさによって変更が必要な場合があります。

4.3.4 カラーセンサを使う

カラーセンサを使うと、赤、青、緑の色情報を数値情報として知ることができます。単に明るさ情報を取得して、光センサーの代わりに使うこともできます。以下は地面の明るさや色の情報を MindStorm の LCD ディスプレイに表示するプログラムです。LCD ディスプレイへの表示方法の詳細は 4.5 節を参照してください。

```

task main()
{
    unsigned int r,g,b;
    SetSensorColorFull(IN_3); //入力ポート3に光センサを接続。
    while (true)
    {
        r=ColorSensorValue(IN_3,INPUT_RED);
        g=ColorSensorValue(IN_3,INPUT_GREEN);
        b=ColorSensorValue(IN_3,INPUT_BLUE);
        TextOut(15, LCD_LINE1,":", true); //1行目に明るさ表示
        TextOut(15, LCD_LINE2,"Red: ", true); //2行目に赤色の強度
        TextOut(15, LCD_LINE3,"Green:", true); //3行目に緑色の強度
        TextOut(15, LCD_LINE4,"Blue: ", true); //4行目に青色の強度
        NumOut(60, LCD_LINE1, (r+g+b)/3);
        NumOut(60, LCD_LINE2, r);
        NumOut(60, LCD_LINE3, g);
        NumOut(60, LCD_LINE4, b);
        Wait(500);
    }
}

```

関数 `SetSensorLight()` はカラーセンサーがどのポートに接続されているかを指定する命令です。この命令ではカラーセンサーの横の LED が発光することで色情報を取得できますが、LED を消灯したい時にはかわりに `SetSensorColorNone(IN_2)` を使います。

[問 4.5] 上記のプログラムを改造して、カラーセンサを使って地面に書いた黒い線にそってロボットを動かすプログラムを作成してみましょう。黒い線の右端に沿って動くプログラムと、左端に沿って動くプログラムの両方を作成してみてください。THRESHOLD の値やモータ出力は周囲の明るさやコースによって変更が必要な場合があるのでいろいろなコースで試してみましょう。

4.3.5 音センサを使う

以下はポート2に音センサを接続したプログラム例です。

```

#define THRESHOLD 40
#define MIC SENSOR_2
task main()
{
    SetSensorSound(IN_2); //入力2に音センサ接続。SetSensor(IN_2, SENSOR_SOUND)と同じ意味
    while(true){
        until(MIC > THRESHOLD); //while (MIC<THRESHOLD); でもよい
        OnFwd(OUT_AC, 75);
        Wait(300); //なくても良いが、あると動きが（多分）ややスムーズになる
        until(MIC > THRESHOLD);
        Off(OUT_AC);
        Wait(300);
    }
}

```

関数 `SetSensorSound()` は音センサーを接続したポート番号を指定する命令です。

[問 4.6] 上記のプログラムの動作を説明しなさい。また実際にプログラムをロボットに実行させ、2つのWait

の有無による動作の違いを考察しなさい。

4.3.6 超音波センサを使う

超音波センサは、出した音が返ってくるまでの時間を計ることで前方の物体までの距離を測ることができます。センサです。

[問 4.7] 以下のプログラムの動作を説明し、実際にロボットを使って動作確認をしなさい。

```
#define NEAR 15 //cm
task main(){
    SetSensorLowspeed(IN_4); //入力 4 に超音波センサ接続。SetSensor(IN_4, SENSOR_LOWSPEED) でも良い
    while(true){
        OnFwd(OUT_AC,50);
        while(SensorUS(IN_4)>NEAR);
        Off(OUT_AC);
        OnRev(OUT_C,100);
        Wait(800);
    }
}
```

関数 `SetSensorLowspeed()` は超音波センサがどのポートに接続されているかを指定する命令です。ここで、超音波センサによって対象物まで距離を知るには関数 `SensorUS()` を使う事に気をつけましょう。`SensorUS()` は超音波センサのセンサ値から距離(cm) に変換して返す関数です。

[問 4.8] 超音波センサのセンサ値を LCD ディスプレイに表示するプログラムを作って、超音波センサで測れる距離の範囲を確認しなさい。超音波センサは小さなものには反応しにくいので、大きめの紙や板を使って試すこと。

4.4 Task と Functions

4.4.1 Tasks

NXC には並列計算の機能があり、複数の作業を同時に実行することができます。並列して実行できる作業単位を `task` と呼びます。`task` は 1 つのプログラム中に最大 255 個まで作ることができます。`task` の定義方法は後の一例で示すように C 言語の関数と似ています。プログラム中に `task main` は必ず 1 つだけ作る必要があります。プログラム実行時には `task main` がまず呼び出されます。他の `task` は `task main` から関数 `Precedes()` 使って呼び出します。この場合、`task main` の実行終了後に `Precedes()` で呼び出した関数の実行が始まります。

以下はロボットが障害物に当たったら、方向転換しながら動き回るプログラム例です。ロボットは直進するという `task` である `move_forward()` と、タッチセンサの入力を監視しながら入力があれば方向転換をするという `task` である `check_sensors()` を並列して行っています。

```

mutex mFlag;
task move_forward()
{
    while (true)
    {
        Acquire(mFlag);           //他のタスクが手旗を取ってなければここで取る (mFlag を Acquire)
        OnFwd(OUT_AC, 75);
        Release(mFlag);         //手旗を放す (mutex を解放 (Release))
    }
}
task check_sensors()
{
    while (true)
    {
        if (SENSOR_1 == 1)
        {
            Acquire(mFlag);           //他のタスクが手旗を取ってなければここで取る
            OnRev(OUT_AC, 75); Wait(500);
            OnFwd(OUT_A, 75); Wait(500);
            Release(mFlag);          //手旗を放す
        }
    }
}
task main()
{
    Precedes(move_forward, check_sensors); //2つのタスクを並列処理することを指定
    SetSensorTouch(IN_1);                 //task main の終了後、指定した task の実行がはじまる
}

```

タスク main で呼び出している関数 Precedes は並列計算をする複数の task を呼び出します。必要に応じて 3つ以上の task を引数として指定して呼び出す事もできます。

2つの task を同時に実行していると、その 2つのタスクが 1つのモータに異なる方向に動くように命令を出してしまいます。このような問題を防ぐため、**mutex** (mutual exclusion, 相互排除) という型の変数を使います。上の例では mutex 型の変数 mFlag を 2つの task が関数 Acquire() でとりあいをしています。一方が mFlag の Acquire (獲得) に成功していれば、Release (解放) されるまで、もう一方の task は Acquire できず、その後の命令は実行されません。このような mutex 型の変数は **semaphore** (手旗信号) と呼ばれ、並列計算で一般によく使われます。

4.4.2 関数 (Functions)

C 言語の関数とほぼ同じものとして NXC にも関数 (Function) があります。関数は task から呼び出す事もできますし、関数から他の関数を呼び出す事もできます。

ロボットに (1) 何事もなければ直進、(2) ぶつかったら方向転換、という動作をさせたい時には (1), (2) に対応する task を書いて並列処理をします。この 2つの task の実現に必要な実行単位（前進する、後進するなど）は関数で作って task から呼び出すことができます。並列処理が不要なら task main 以外の task は作らなくてよいです。

```

void turn_around(int pwr) // 関数の宣言は C 言語と同様
                        // この例では void としてもよい
{
    OnRev(OUT_C, pwr); Wait(900);
    OnFwd(OUT_AC, pwr);
}
task main()
{
    OnFwd(OUT_AC, 75);
    Wait(1000);
    turn_around(75);           //先に定義した関数を呼び出す
    Wait(2000);
    turn_around(75);
    Wait(1000);
    turn_around(75);
    Off(OUT_AC);
}

```

この例では関数の返り値 void ですが、C 言語と同様に int や string を返り値にすることもできます。

4.4.3 引数のアドレス渡し

以下は関数への引数のアドレス渡しの例です。C 言語のようにポイントを使うかわりに、NXC では引数に & をつけることでアドレス渡しを実現できます。(C++ 言語と同じ書式です。)

```

void foo(int &x)      //引数のアドレス渡し
{
    x = 2;
}
task main()
{
    int y = 1;          // y に 1 を代入
    foo(y);            // y は 2 になります
    foo(2);            // アドレス渡しの場合は引数は変数でないとダメ
}

```

4.4.4 並列タスク

[問 4.9] 以下のプログラムの悪い点を指摘し、修正案を書きなさい。

```
task check_sensors()
{
    while (true)
    {
        if (SENSOR_1 == 1)
        {
            OnRev(OUT_AC, 75);
            Wait(500);
            OnFwd(OUT_A, 75);
            Wait(850);
        }
    }
}

task forward()
{
    while (true)
    {
        OnFwd(OUT_AC, 75); Wait(100);
    }
}

task main()
{
    SetSensor(IN_1,SENSOR_TOUCH);
    Precedes(check_sensors, forward);
}
```

[問 4.10] ライントレースをしながら、ライン上にあった障害物にぶつかりそうになつたら、もしくはぶつかつたら少し後ろに下がってからランダムに方向転換してラインを探し、ラインを見つけたらまたライントレースをするプログラムを並列タスクを利用して作りましょう。

4.5 LCD ディスプレイ表示

MindStorm には 100x64 画素の LCD ディスプレイがあります。以下は LCD ディスプレイに文字や数値を出力するプログラム例です。原点座標 (0,0) はディスプレイの左下の位置です。

```

#define X_MAX 99
#define Y_MAX 63
#define X_MID (X_MAX+1)/2
#define Y_MID (Y_MAX+1)/2
task main(){
    int i = 1234;
    TextOut(15, LCD_LINE1,"Display", true); //左から 15 画素,LCD_LINE1(1行目) の位置に文字列表示
    NumOut(60,LCD_LINE1, i); //左から 60 画素,LCD_LINE1(1行目) の位置に数値表示
    PointOut(1,Y_MAX-1);
    PointOut(X_MAX-1,Y_MAX-1);
    PointOut(1,1);
    PointOut(X_MAX-1,1);
    Wait(200);
    RectOut(5,5,90,50);
    Wait(200);
    LineOut(5,5,95,55);
    Wait(200);
    LineOut(5,55,95,5);
    Wait(200);
    CircleOut(X_MID,Y_MID-2,20);
    Wait(800);
    ClearScreen();
    GraphicOut(30,10,"faceclosed.ric"); Wait(500);
    ClearScreen();
    GraphicOut(30,10,"faceopen.ric");
    Wait(1000);
}

```

上記のプログラムで使っている関数の詳細は以下の通りです。

関数名	機能
ClearScreen()	画面を消す
NumOut(x, y, number)	(x,y) の位置に数値 number を表示
TextOut(x, y, string, clear=false)	(x,y) の位置に文字 string を表示。第4引数を true とすると 画面消去後表示
GraphicOut(x, y, filename)	.ric 形式の画像データを表示 (ric 形式)
CircleOut(x, y, radius)	中心 (x,y), 半径 radius の円を描く
LineOut(x1, y1, x2, y2)	(x1,x2) と (x2,y2) を直線で結ぶ
PointOut(x, y)	(x,y) に点を描く
RectOut(x, y, width, height)	四角形を描く。(x,y):左下の座標, width:幅, height:高さ
ResetScreen()	画面をリセットする

[問 4.11] MindStorm の各センサ (超音波センサ, 光センサ) の値の一覧を画面に表示するプログラムを作りましょう。

4.6 ファイル入出力

NXC でファイルにアクセスするには byte 型の変数をファイルアクセス用に用意します。これが C 言語のファイルポインターのような役目をします。

以下はファイルにデータを書き出すプログラム例です。このプログラムではまず、これから作りたいファイル（“Danny.txt”）がすでにあればこれを消してからファイルを作成しています。次に CreateFile 関数でファイルを開きます。この時ファイルの大きさをバイトで指定する必要があることに注意して下さい。ファイルは LEGO の本体のメモリ内に作られます。このためあまり大きなファイルは作れないことによく注意しましょう。また不要になったファイルは必ず直ちに MindStorm 内から消してください。

WriteLnString (fileHandle, string, bytesWritten) は fileHandle の指すファイルに文字列 string を書き出します。その後何バイト書き出されたかが変数 bytesWritten に代入されます。

```
task main(){
    byte fileHandle;           // ファイルアクセスのための変数を用意
    short fileSize;
    short bytesWritten;
    string read;              // 文字列 read を宣言
    string write;              // 文字列 Write を宣言
    DeleteFile("Danny.txt");   // Danny.txt を消す
    CreateFile("Danny.txt", 512, fileHandle); // 512 バイトの大きさのファイルを開く
    for(int i=2; i<=10; i++){
        write = "NXT is cool ";
        string tmp = NumToStr(i);           // 数値を文字列に変換して tmp に代入
        write = StrCat(write,tmp," times!"); // 引数で与えた 3 つの文字列を Strict で
                                              // つなげて変数 write に代入
        WriteLnString(fileHandle,write, bytesWritten); // 文字列 write を書き出す
    }
    CloseFile(fileHandle);         // ファイルを閉じる
}
```

結果を見るには、BricxCC→Tools→NXT Explorer をクリックして、upload Danny.txt でファイルを PC にアップロードしてみて下さい。

以下は数値データをファイルに書き出し、その後ファイルのデータを読み込んで MindStorm の LCD ディスプレイに表示するプログラム例です。

```
task main () {
    byte handle, time = 0;
    int n, fsize, len, i;
    int in;
    DeleteFile("int.txt");
    CreateFile("int.txt",4096,handle); // 4096 バイトまで書けるファイルを作る
    for (int i = 1000; i<=10000; i+=1000){
        WriteLn(handle,i);          // ; の値を書き出し
    }
    CloseFile(handle);             // ファイルを閉じる
    OpenFileRead("int.txt",fsize,handle); // ファイルを読み出しモードで開く
    until (ReadLn(handle,in)!=NO_ERR){ // 1 行読んで結果を in に代入
        ClearScreen();            // 画面消去
        NumOut(30,LCD_LINE5,in);   // 5 行目の行頭から 30 画素の位置に in の値を書く
        Wait(500);
    }
    CloseFile(handle);             // ファイルを閉じる
}
```

5 発展的なプログラミング

5.1 LED ライトを点灯する

接続した LED ライトの点灯や消灯はモータの制御と同様に行ないます。たとえば B ポートにライトをついている場合は以下のようにします。

```
OnFwd(OUT_B, 70); //点灯
Off(OUT_B); //消灯
```

5.2 音楽を鳴らす

NXT はいろいろな高さの音や音源ファイルを鳴らすこともできます。音を鳴らすには関数 PlayToneEx(frequency, duration, volume, loop?) を使います。frequency は周波数 (Hz) , duration は長さ (ms) , volume は強さ (0 から 4) , そして loop? には一度だけ鳴らすなら FALSE もしくは 0 を, くり返すなら TRUE もしくは 1 を指定します。

PlayTone(frequency, duration) を使うこともできます。この場合の音の強さは一定 (NXT のメニューで設定) になります。以下は周波数の一覧表です。

Sound	3	4	5	6	7	8	9
シ B	247	494	988	1976	3951	7902	
ラ# A#	233	466	932	1865	3729	7458	
ラ A	220	440	880	1760	3520	7040	14080
ソ# G#		415	831	1661	3322	6644	13288
ソ G		392	784	1568	3136	6272	12544
ファ# F#		370	740	1480	2960	5920	11840
ファ F		349	698	1397	2794	5588	11176
ミ E		330	659	1319	2637	5274	10548
レ# D#		311	622	1245	2489	4978	9956
レ D		294	587	1175	2349	4699	9398
ド# C#		277	554	1109	2217	4435	8870
ド C		262	523	1047	2093	4186	8372

PlayFileEx を使うと 1 つの音が終わる前に次のコマンドが実行されるので, Wait コマンドで音と音の間隔を調節する必要があります。

```
#define VOL 3
task main()
{
    PlayToneEx(262,400,VOL,FALSE); Wait(500);
    PlayToneEx(294,400,VOL,FALSE); Wait(500);
    PlayToneEx(330,400,VOL,FALSE); Wait(500);
    PlayToneEx(294,400,VOL,FALSE); Wait(500);
    PlayToneEx(262,1600,VOL,FALSE); Wait(2000);
}
```

BricxCC を使っている方は BrickPiano で簡単に曲を作ることもできます。MindStorm で曲を鳴らしたい時は曲を Task にしてプログラムを組むことをおすすめします。以下は例です。

```

task music()
{
    while (true)
    {
        PlayTone(262,400); Wait(500);
        PlayTone(294,400); Wait(500);
        PlayTone(330,400); Wait(500);
        PlayTone(294,400); Wait(500);
    }
}

task movement()
{
    while(true)
    {
        OnFwd(OUT_AC, 75); Wait(3000);
        OnRev(OUT_AC, 75); Wait(3000);
    }
}

task main()
{
    Precedes(music, movement);
}

```

5.3 関数のマクロ定義

C 言語のように、`#define` を使って定数だけでなく関数のマクロ定義をすることもできます。

```

#define turn_around \
    OnRev(OUT_B, 75); Wait(3400);OnFwd(OUT_AB, 75); // 以下の Turn_around という文字列は、
                                                       // コンパイル前に OnRev …, という文字列におきかえられる

task main()
{
    OnFwd(OUT_AB, 75);
    Wait(1000);
    turn_around;
    Wait(2000);
    turn_around;
    Wait(1000);
    turn_around;
    Off(OUT_AB);
}

```

`#define` の定義は 1 行で書く必要があります。1 行で入らない時は行末に"\\"を書いて改行していることを明示して下さい。

以下は引数を使ったマクロ関数の定義の例です。

```
#define turn_right(s,t) \
    OnFwd(OUT_A, s);OnRev(OUT_B, s);Wait(t);
#define turn_left(s,t) \
    OnRev(OUT_A, s);OnFwd(OUT_B, s);Wait(t);
#define forwards(s,t)    OnFwd(OUT_AB, s);Wait(t);
#define backwards(s,t)   OnRev(OUT_AB, s);Wait(t);
task main()
{
    backwards(50,10000);
    forwards(50,10000);
    turn_left(75,750);
    forwards(75,1000);
    backwards(75,2000);
    forwards(75,1000);
    turn_right(75,750);
    forwards(30,2000);
    Off(OUT_AB);
}
```

5.4 inline 関数

短い関数を作るときには以下のように inline 宣言をして下さい。この場合、関数を呼び出した場所に関数の実行内容が展開されます。その結果メモリは多く必要になりますが、処理速度は速くなります。

```
inline int Name( Args ) {
    return x*y;
}
```

以下は例です。

```
inline void turn_around()
{
    OnRev(OUT_C, 75); Wait(900);
    OnFwd(OUT_AC, 75);
}
task main()
{
    OnFwd(OUT_AC, 75);
    Wait(1000);
    turn_around();
    Wait(2000);
    turn_around();
    Wait(1000);
    turn_around();
    Off(OUT_AC);
}
```

上記の inline 関数に引数を作った例が以下です。

```

inline void turn_around(int pwr, int turntime)
{
    OnRev(OUT_C, pwr);
    Wait(turntime);
    OnFwd(OUT_AC, pwr);
}
task main()
{
    OnFwd(OUT_AC, 75);
    Wait(1000);
    turn_around(75, 2000);
    Wait(2000);
    turn_around(75, 500);
    Wait(1000);
    turn_around(75, 3000);
    Off(OUT_AC);
}

```

A 用意されている数学関数

Random(n)

0 から n-1 までの値の乱数（整数）を返す

x=Random(10); // 0 から 9 までの値を返す

Random()

int 型 (-32,768～32,767) の乱数を返す

x=Random();

Sqrt(x)

x の平方根（整数）を返す

x=Sqrt(x);

Sin(degrees)

指定した角度に対する Sin 関数の値を 100 倍した整数値 (-100 から 100) を返す

x=Sin(theta);

Cos(degrees)

指定した角度に対する Cos 関数の値を 100 倍した整数値 (-100 から 100) を返す

x=Cos(y);

B トラブル対策

- MindStorm を接続した USB が認識されない。

- MindStorm のドライバを管理者権限でインストールしましょう。一般ユーザ権限でインストールしていると MindStorm とは接続できないので気をつけてください。
- プログラムをコンパイルできない。
 - bricx(zip ファイル) をダウンロードしたまま解凍しないで bricx を起動していませんか？
 - bricx の実行形式のみを (アイコンをドラッグして持っていく) デスクトップ等にコピーして使ってませんか。コピーの代わりにショートカットを使うか、Bricx のフォルダの中の実行形式を直接実行しましょう
 - プログラムのエラーメッセージが出てないか確認しましょう。
- プログラムを MindStorm にダウンロードが出来ない。
 - 3.1 節を参照して Port, Brick Type, Firmware の設定を確認してください。
 - MindStorm に古いプログラムが多数残っていると、ダウンロードのためのメモリが足りなくなることがあります。この場合は MindStorm を操作して不要なプログラムを消してください。
 - MindStorm のドライバを一般ユーザ権限でインストールしていると MindStorm と接続できません。管理者権限でインストールしてください。

索引

Symbols

#define 8, 21

A

Acquire() 15

ArrayInit() 6

B

bool 5

BrickPiano 20

Bricx 2, 3

byte 5

C

CircleOut() 18

ClearScreen() 18, 19

CloseFile() 19

compile 4

Cos() 23

CreateFile() 19

D

DeleteFile() 19

do~while 文 8

download 4

F

fileHandle 19

float 5

function 15

G

GraphicOut() 18

I

if 文 7, 8

inline 22

int 5

L

LEGO MindStorm 2

LineOut() 18

long 5

M

MindStorm 2

mutex 15

N

NumOut() 18, 19

NXC 2

O

Off 4

Off() 4

OnFwd 4

OnFwd() 4

OnRev 4

OnRev() 4

OpenFileRead() 19

P

PlayFileEx() 20

PlayTone() 20

PlayToneEx() 20

PointOut() 18

Precedes 15

R

Random() 7, 23

ReadLn() 19

RectOut() 18

Release() 15

repeat 文 9

ResetScreen() 18

run 4

S

semaphore 15

SensorUS() 14

SetSensor() 11

SetSensorColorFull() 13

SetSensorLight() 12

SetSensorLowspeed() 14

SetSensorSound() 13

SetSensorTouch() 11

short 5

Sin() 23

Sqrt() 23

stop 4

StrCat()	19
string	5
struct	5

T

task	14
TextOut()	18

U

unsigned int	5
unsigned long	5
until 文	10

W

Wait	4
Wait()	4
while 文	8
WriteLnString()	19

あ

アドレス渡し	16
演算子	5

か

カラーセンサ	10
コメント	4
コンパイル	4

さ

実行	4
条件式	7

た

ダウンロード	4
タッチセンサ	10
超音波センサ	10
停止	4

な

入力ポート	11
-------	----

は

配列	6
光センサ	10
並列計算	14
変数	5

ま

モータの出力	5
--------	---

ら

乱数	7
----	---