NQC Tutorial

Introduction

by Derek Schuurman

This tutorial describes a language called NQC (Not Quite C). NQC uses the same syntax, loops, and conditional constructs as C/C++, but lacks some of the more advanced features in the C/C++ programming language. Programs can be compiled using NQC and then downloaded over an infra-red (IR) interface to the RCX computer. The required steps are outlined below.

Part A : Writing, Compiling, and Downloading the robot control program

The steps you should follow when implementing your program should proceed as follow:

- Start by developing your algorithm using pseudocode.
- Once you are confident that your algorithm is ready you can begin coding your program in NQC. Your program may be entered using a simple text editor. Ensure that your program is saved with the suffix '**`.nqc`**' (for example: **`program.nqc`**). Consult the next section for the library commands that can be used to control the robot. Note: use **`#define`** for your constants and use plenty of comments.
- To compile your program, type the following command in a terminal window:

  ```
  nqc -v -s program.nqc
  ```

  Any errors in the source code will result in an error message with the line number of where the error occurs specified. Make any corrections as necessary. A successful compile will produce an output file with a ".rcx" extension.
- Plug the USB Infrared (IR) Tower into the front USB port of the computer. Place the RCX computer by the USB Tower such that the dark plastic panel on the top/front of the unit is facing the front of the tower. Ensure that the RCX unit is turned on.
- To download your program to the RCX computer, type the following command in a terminal window:

  ```
  nqc -v -d program.rcx
  ```

  This download the program to the RCX unit. Once the download completes, the RCX unit will make a sound. If the download fails, reposition the IR tower with respect to the sensor on the RCX computer and try again.
- To run your uploaded program, press the green "Run" button on the RCX unit. Ensure that all the motor and sensor connectors are firmly attached as they should be. The program will run until it completes or until the 'Run' button is pressed again.

Part B: Summary of Some Useful NQC Library Commands

NQC stands of Not-Quite-C and can be used for programming Lego robots. NQC supports the basic C syntax with some small differences. All the basic conditional and iterative control structures are the same as in C including if..else, switch, while, do..while and for loops. However, the only variable type that is supported in NQC is the **int** type and all variables have to be declared at the beginning of a function. NQC also supports functions which may have input arguments, but NQC does not support return values from functions. Consequently, all functions in NQC must be declared with a **`void`** return type. Execution begins in **`main`** which for NQC should be declared in a special way as follows:

```
task main()
{
      // Place your code here...
      :
}
```

The NQC compiler comes with a variety of useful library functions, some of which are described below. Note that no **`#include`** statements are necessary to use these functions.

Reading Sensor Inputs

There are three sensor inputs on the RCX computer numbered 1 to 3. These sensors have been predefined and can be referenced using the names **`SENSOR_1`**, **`SENSOR_2`**, and **`SENSOR_3`**. There are several special function calls that are used to configure and read the sensor inputs.

The sensor inputs can accommodate a wide variety of different sensor types, so at the beginning of your program you should define what kind of sensor is connected to the RCX unit. For example, when using a touch sensor for sensor #1, you should configure the sensor by calling the **`SetSensor`** function as follows:

```
SetSensor(SENSOR_1, SENSOR_TOUCH);
```

To read a sensor value at any time, use the **`SensorValue(n)`** function which returns the sensor reading for sensor n, where n is 0, 1, or 2 (corresponding to sensor 1, 2,or 3). For example, to read sensor #1 into an integer variable called **`x`**:

```
x = SensorValue(0);
```

A 0 or a 1 will be returned depending on the state of the switch. For this assignment, you will only need to use two touch sensor inputs for the left and right bumper switches.

Controlling Motors and Outputs

The names **`OUT_A`**, **`OUT_B`**, and **`OUT_C`** are used to identify the three outputs as labelled on the RCX computer. Each output has three different attributes: *mode*, *direction*, and *power level*. You will need only two outputs for each motor.

To set the mode, use the **`SetOutput(output, mode)`** function where **`output`** can be one of **`OUT_A`**, **`OUT_B`**, or **`OUT_C`** and **`mode`** can be one of **`OUT_OFF`**, **`OUT_ON`** or **`OUT_FLOAT`**. The **`OUT_FLOAT`** option mode can be used to make the motors "coast". For example:

```
SetOutput(OUT_A, OUT_ON);   // Enables the first motor output
```

For many of the commands described here, outputs may also be combined as follows:

```
SetOutput(OUT_A + OUT_B, OUT_ON);   // Enables the motor outputs A and B
```

The direction is set with the **`SetDirection(outputs, direction)`** command where **`output`** can be one of **`OUT_A`**, **`OUT_B`**, or **`OUT_C`** and direction can be one of **`OUT_FWD`**, **`OUT_REV`** or **`OUT_TOGGLE`**. The **`OUT_TOGGLE`** option mode can be used to toggle the motor output direction.

The power attribute can be set using the **`SetPower(outputs, power)`** function where **`output`** can be one of **`OUT_A`**, **`OUT_B`**, or **`OUT_C`** and power can be a value ranging from **`0`** to **`7`**. It is strongly recommended that you use a motor power setting of not higher than 3 or 4 to prevent the robot from moving to quickly and getting damaged.

Some additional convenient functions are also available. In particular the **`OnFwd(output)`** and **`OnRev(output)`** functions can be used to turn an input on and set its direction at the same time. For example:

```
OnFwd(OUT_A);      // Turns ON the motor on output A in forward direction
```

Motor outputs can also be combined as follows:

```
OnRev(OUT_B + OUT_C); // Turn motors on outputs B & C in reverse direction
```

The **`Off(output)`** function can be used to turn motors off. For example:

```
Off(OUT_A);        // Turn off motor attached to output A
```

Other useful functions are provided for timing. The **`Wait(time)`** function makes the computer "sleep" for a specified time in measured in hundredths of a second. For example, to make a program pause for 1 second, use the following:

```
Wait(100);
```

You can also generate "musical" tones at certain times on the computer by using the **`PlayTone(frequency, time)`** function followed by a **`Wait()`** function to wait while the tone is played. For example, to play an 440Hz note for ½ second, use the following function call:

```
PlayTone(440,50); Wait(50);
```

Some references include:

- Dave Baum et al., **"Extreme Mindstorms: An Advanced Guide to LEGO MINDSTORMS",** Apress, 2000
- Jin Sato, "**Jin Sato's Lego Mindstorms",** No Starch Press, 2002.