LUND UNIVERSITY

Robot Design Introduction to NQC

Jacek Malec
Department of Computer Science
Lund University

July 19, 2007

AI@CS
Department of Computing Science

LUND UNIVERSITY

Not Quite C

However, rather similar ...
This tutorial is based on Dave Baum's *NQC Programmer's Guide*, V. 2.5 a4 (available together with the software, from Help menu).
Version 3.1 r5 is available from
`http://bricxcc.sourceforge.net/nqc/`

- Lexical rules;
- Program structure;
- Statements;
- Expressions;
- Preprocessor;
- API.

AI@CS
Department of Computing Science

LUND UNIVERSITY

Application Programmer Interface (API)

- Sensors;
- Outputs;
- Sound;
- LCD display;
- Communication;
- Timers;
- Counters;
- Data logging;
- RCX features.

AI@CS
Department of Computing Science

LUND UNIVERSITY

General information

- NQC compiles a program into LEGO bytecode
- RCX interprets the bytecode
- Functionality depends on the firmware (we use RCX2)
- NQC consists of two separate pieces: language and API
- API is a separate part of the compiler

AI@CS
Department of Computing Science

LUND UNIVERSITY

Lexical rules

- comments:

```
/* this is a comment */
// this as well
```

- whitespace (space, tab, newline) does not matter

```
x=2;
x = 2 ;
x = 1 >> 2; // shift right
x = 1 > > 2; // error
```

- numerical constants: decimal (`x = 10;`) or hexadecimal (`x = 0x10;`)
- identifiers begin with a letter or underscore. Beware of reserved keywords!

AI@CS
Department of Computing Science

LUND UNIVERSITY

Program structure

- tasks

```
task mytask() {
// code (statements)
}
```

NQC task = RCX task (max 10)
there must exist at least one task named `main`

- functions (expanded inline!)

```
void myfunction(arg_list) {
// body
}
```

argument types: `int`, `const int`, `int&` and `const int&`: value passing semantics vary

AI@CS
Department of Computing Science

Program structure 2

- subroutines (no args, no nesting, max 8)

```
sub mysubroutine() {
// body
}
```

- variables: always 16 bit integers
  need to be declared (int x; or `int y = 2;`)
  may be global and local
  limited pool: 32 global, 16 local (declare as locally as possible!)
- arrays
  declare: `int my_array[3];`
  use: `my_array[0] = 12; my_array[2] = my_array[1];` (note limitations!)

Statements;

- variable declaration;
- assignment;
- control structures
  - if (condition) consequence
  - if (condition) consequence else alternative
  - compound statement (`{...}`
  - while (condition) body
  - do body while (condition)
  - for (stmt1 ; condition ; stmt2) body
  - repeat (expression) body
  - switch (expression) body
  - goto label:

Statements 2

- access control (prioritisation):

  acquire (resources) body
  acquire (resources) body catch handler
- event monitoring:

  monitor (events) body
  monitor (events) body handler_list

  catch (catch_events) handler
  catch handler

  Events must be numbered (`EVENT_MASK()` macro)

Statements 3, Expressions

- subroutine calls
- task activation (`start`, `stop`
- break, continue — within loops
- return
- expression (only `x++;` or `y- -;` make sense)

Expressions:

- values and their combinations by using operators, yield a value
- conditions, yield a logical value (`true`, `false`, usual C convention)

The preprocessor

- `#include`
- `#define`, `#undef`
- `#ifdef`, `#ifndef`
- `#if`, `#elif`, `#else`, `#endif`

have standard C meaning.

- `#pragma noinit`
  `#pragma init myfunction`
- `#pragma reserve storagelocation`
  `#pragma reserve start end`

API

- Sensors (`SENSOR_1`, `SENSOR_2`, `SENSOR_3`), both arguments and values!
- sensor types: touch, temperature, light, rotation, none (raw)
- sensor modes: raw, bool, edge, pulse, percent, fahrenheit, celsius, rotation
- `SetSensorType`, `SetSensorMode`, `SetSensor`, `ClearSensor` (only for cummulative)
- access: `SensorValue(n)` (note: n = 0, 1, 2!), others

API 2

- Outputs (`OUT_A, OUT_B, OUT_C`), combinations using +
- output attributes: mode (off, on, float), direction (forward, backward, toggle), power level (0-7), appropriate `Set`-functions
- convenience calls (On, Off, Float, OnFwd, ...)

Sound, Display, Communication

- `PlaySound(constant)`, SOUND_CLICK, SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP, SOUND_FAST_UP.
- `PlayTone(int freq, const duration)`
- mute, unmute, clear
- `SelectDisplay(mode)`, `SetUserDisplay(var,const)`
- `Message()` reads the buffer, `ClearMessage()` clears it
- `SendMessage(int)`, `SetTxPower(const)`
- serial communication is possible

Timers, Counters

- Timers: 4 with 100ms or 10ms resolution; `ClearTimer(0), Timer(2), SetTimer(3, x), FastTimer(2)`
- Counters — overlap with memory locations 0–2 (use `#pragma reserve`!)
- `ClearCounter(1), IncCounter(1), DecCounter(1), Counter(1)`

Other Stuff

- `SetPriority(prio)` - sets task's priority to prio. Useful for access control (acquire-statements)
- event monitoring - up to 16 freely configurable events
- logging data: `CreateDatalog(size)`, `AddToDatalog(val)`
- `Wait(hundreths)`
- `StopAllTasks()`
- `Random(const)`
- sleeping, program switching, battery access, firmware version, clock access

Other systems than NQC

- LeJOS (www.lejos.org), works also for NXT (new generation brick): gives you Java Virtual Machine (but without GC)
- BrickOS — replacement operating system, allowing one to use GNU C or C++ for program development (http://brickos.sourceforge.net/)