Not Quite C: A CS 0 Option

LMICSE Workshop
June 14 - 17, 2005
Alma College
NQC

- Well suited for those new to programming
- Often used in non-majors courses
- Allows the demonstration of most non-object oriented aspects of introductory programming
- Easy introduction to concurrent programming issues
NQC Basics

- C-like syntax
- Limited by use of Lego standard firmware
  - 10 threads
  - 32 global variables, 16 local variables per thread
  - Integers only
  - Byte codes are interpreted
  - No call stack, no memory management
- Functions
- Simple event model
- Limited arrays
RCX Basic Motor Control

- Recall that there are three power outputs, and in NQC the motors attached to them are called
  - OUT_A
  - OUT_B
  - OUT_C

- Basic motor control commands
  - `OnFwd( ... )`: run the ... motor(s) forward
  - `OnRev( ... )`: run the ... motor(s) in reverse
  - `Off( ... )`: stop the ... motor(s)

- Another useful command
  - `Wait( ... )`: pause the program execution for ... microseconds
Example 1: Drive Forward

- Drive forward for 4 seconds

```c
task main () {
    OnFwd(OUT_A);
    OnFwd(OUT_C);
    Wait(400);
    Off(OUT_A);
    Off(OUT_C);
}
```
Example 1 Revised: Drive Forward

- Motor specifications can be “added” together.

```cpp
task main () {
    OnFwd(OUT_A + OUT_C);
    Wait(400);
    Off(OUT_A + OUT_C);
}
```
Example 2: Out and Back

- Move forward for 5 seconds, turn around and come back to the starting point

```c
task main () {
  OnFwd(OUT_A + OUT_C); // Move Forward
  Wait(500);

  OnRev(OUT_C); // Turn Around
  Wait(200);

  OnFwd(OUT_C); // Come Back
  Wait(500);

  Off(OUT_A + OUT_C); // Stop
}
```
Setting Motor Power

- You can set an output power level by the `SetPower` command

- Examples
  - `SetPower(OUT_A, 4);`
  - `SetPower(OUT_A + OUT_C, 1);`

- The first parameter is the output(s), the second a power level in the range 0..7

- Affects motor torque more than actual speed
Inline Functions

- NQC provides for functions which are “inline” by the compiler
- Pass by value and pass by reference parameters like in C
- All functions return void.
Example 3: Square Patrol

- Repeatedly drive around in a square shape

```c
task main () {
    while (true) {
        moveForward(500);
        turnLeft(100);
    }
}

void moveForward(int time) {
    OnFwd(OUT_A + OUT_C);
    Wait(time);
    Off(OUT_A + OUT_C);
}

void turnLeft(int time) {
    OnFwd(OUT_C);
    OnRev(OUT_A);
    Wait(time);
    Off(OUT_A + OUT_C);
}
```
Using Sensors (1)

- In the Reactive Control paradigm the bot controls its actions in response to input from its sensors.
- Two types of sensors come with the Mindstorms kit
  - Bump Sensors
  - Light Sensors
- Recall that there are three sensor connections, and in NQC the sensors attached to them are called
  - SENSOR_1
  - SENSOR_2
  - SENSOR_3
- More than one sensor can be attached to a connection
Using Sensors (1)

- To use a sensor we must first define its type using `SetSensor`

- Examples
  - `SetSensor(SENSOR_1, SENSOR_TOUCH);`
  - `SetSensor(SENSOR_2, SENSOR_LIGHT);`

- By default, touch sensors return 0 or 1 (1 if currently pressed)
- By default, light sensors return a value in the range 0..100, with 0 meaning no measured light

- Other return modes are possible, see the NQC manual for details
Example 4: Move till Bump

- Move forward until a bump
- Solution uses sensor polling

```c
task main () {
    SetSensor(SENSOR_2, SENSOR_TOUCH);

    OnFwd(OUT_A + OUT_C); // Move Forward
    while (SENSOR_2 != 1) // Until a Bump
        ;
    Off(OUT_A + OUT_C); // Then Stop
}
```
Example 5: Move till Darker

- Move forward until light reading becomes \textit{tolerance} units less than the starting light level

```plaintext
task main () {
    int startLevel;
    int tolerance = 5;

    SetSensor(SENSOR_2, SENSOR_LIGHT);
    startLevel = SENSOR_2;

    OnFwd(OUT_A + OUT_C); // Move Forward

    while (SENSOR_2 > startLevel - tolerance)
    {
        // Until Darker

    Off(OUT_A + OUT_C); // Then Stop
}
```
What’s Wrong?

- We want the bot to patrol back and forth until a bump sensor is pressed.
- Why doesn’t this proposed solution work?

```cpp
task main () {
    SetSensor(SENSOR_2, SENSOR_TOUCH);
    OnFwd(OUT_A);
    while(SENSOR_2 != 1) {
        OnFwd(OUT_C);
        Wait(400);
        OnRev(OUT_C);
        Wait(200);
    }
    Off(OUT_A + OUT_C);
}
```
Multitasking

- The RCX supports up to 10 concurrent threads of execution.
- Use the `start` and `stop` commands to control task execution.
Example 6: Patrol with Bump Stop

- Patrol back and forth until a bump sensor is pressed.

```c
task main () {
    SetSensor(SENSOR_2, SENSOR_TOUCH);
    start patrol;
    while(SENSOR_2 != 1) {
        stop patrol;
        Off(OUT_A + OUT_C);
    }
}

task patrol () {
    OnFwd(OUT_A);
    while(true) {
        OnFwd(OUT_C);
        Wait(400);
        OnRev(OUT_C);
        Wait(200);
    }
}
```
Using Resources

- Be careful how tasks use the RCX’s resources
  - **Shareable resources**: the sensors
    - One task’s reading of them does not affect what other tasks are doing
  - **Non-shareable resources**: the motors
    - If more than one task is trying to control the motors, unpredictable results will occur
    - This is an example of *race conditions*
    - A typical source of error once tasks have been introduced in the course
NQC Links

- NQC was developed by Dave Baum and is now maintained by John Hansen
  - http://bricxcc.sourceforge.net/nqc/

- For the Macintosh the IDE of choice is MacNQC

- For Windows the IDE of choice is Bricx Command Center (bricxCC)
  - http://bricxcc.sourceforge.net/
  - BricxCC can also be used with BrickOS, LejOS, and pForth
An NQC Laboratory Project: Feed ME!

• Idea
  • Imagine that your roverbot needs to feed, and becomes increasingly weakened when it has not fed for some time. Food makes it stronger. For our purposes we will assume that the bot can feed at any of the black circles in its playpen. When it has found a food source, it should stop in order to feed on that source. Once full, it should again begin to wander around its world. For a certain period of time after feeding, it will not want to eat again. Once it becomes hungry, however, it must find a food source in a timely manner, or it will begin to lose energy.

• Materials
  • Robot with two bump sensors and a light sensor pointed at the floor
  • A play area containing black circles of various sizes
The Specifics

1. Try to find a food source within ten seconds.
2. If a food source is not found
   • simulate a loss of energy by adjusting the speed of the robot by a fixed amount.
   • Return to the feeding behavior (i.e., return to the first step above).
3. If a food source is found
   • Stop for 3 seconds in order to eat.
   • If the robot was not already at maximum energy capacity, then give it back some of its energy. Used the same fixed amount you used to simulate the loss of energy.
   • Then begin to wander the world again. For 5 seconds the robot should not want to eat.
   • Return to feeding behavior (i.e., return to the first step above).
4. The robot should stop either when someone presses the "off" button or when the robot runs completely out of energy
Project Notes and Stages

- General Notes:
  - Use five different energy levels.
  - Your bot should be able to wander randomly around its playpen, dealing with any wall or obstruction it may find.

- Project Stages: Demonstrate each stage to the TA
  1. The roverbot randomly moves around until it finds a food source and stops over it.
  2. The roverbot loses energy every 5 seconds until it finds food. It stops over the food, or if it doesn't find food eventually stops due to lack of energy.
  3. The complete project as described.
Hands-on Activity: Taking Damage

• Modify the patrol solution (example 6) so that whenever the bot’s bump sensors are pressed instead of stopping it spins around and then goes back to its patrol.

• More advanced problems:
  • Each time the bot is bumped it spins around one more time.
  • As above, but with a maximum number of times the bot can be bumped before it just stops.
  • As above, but that max number is entered into the bot by pressing its left bumper that many times, followed by a right bumper press. Then the bot begins its patrol.