An NQC Primer

1. Program structure

```c
    task main () {
        ... lines of code go here
    }
```

- Every statement ends with a semicolon (;)
- Blocks are defined with `{` and `}`
  (instead of `BEGIN` ... `END` as is done in Karel)

2. Names of built-ins
- Outputs: OUT_A, OUT_B, OUT_C,
- Inputs: SENSOR_1, SENSOR_2, SENSOR_3

3. Robot-like Primitives
- On(OUT_A);
- Off (OUT_A);
- Fwd(OUT_A);
- Rev(OUT_A);
- OnFwd(OUT_A); // convenience command
- OnRev(OUT_A); // convenience command
- Wait(n);       // n is in hundredths of seconds
- Concatenation operator ... On(OUT_A + OUT_B)
4. Variables
   • type and name must be declared ... integer is the most commonly used
     ▪ int a,b,c
   • Operators
     ▪ =    // assignment       c = a;
     ▪ +    // addition         c = a + b;
     ▪ -    // subtraction      c = a – b;
     ▪ *    // multiplication   c = a * b;
     ▪ /    // short division   c = a / b;

5. Constants
   • #define n 100
   • #define left OUT_A

6. Repetition instructions
   • repeat (n) { ... block of instructions ...}
   • while (condition) { ... block of instructions ... } 
   • do { ... block of instructions ... } while (condition)
   • until (condition) { ... block of instructions ...}
7. Conditions
• true // always true
• false // always false
• SENSOR_N ? Z
  ▪ N = 1, 2, 3 and Z is some value
  ▪ where ? can be <, >, ==, <=, >=, !=
• n ? m // n and m are integers previously declared
• compound conditions
  ▪ A || B // A or B
    → eg. ((SENSOR_1 == true) || (SENSOR_2 == true))
  ▪ A && B // A and B
    → ((n > 4) && (SENSOR_3 == false))

8. Control
• if (condition) { … block of instructions … }
• if (condition) { … block of instructions … }
  else { … block of instructions … }
9. Housekeeping details

- comments are started with //
- multi-line comments /* */
- SetSensor(Port, Type)
  - where Port is: SENSOR_1, SENSOR_2, SENSOR_3
  - where Type is: SENSOR_TOUCH, SENSOR_LIGHT, SENSOR_ROTATION

10. Subprograms

- Tasks
  - execute concurrently
  - must be declared
  - can be stopped and started
    - start taskName;
    - stop taskName;

- Functions
  - execute sequentially
  - must be declared
  - are called by using the function name
  - can be passed parameters
An Example Program -- which does nothing but uses tasks and functions

#define bump SENSOR_2
#define left OUT_A
#define right OUT_C
#define myTime 150

void function turn(int turnTime){  Rev(left);  Wait(turnTime);  Fwd(left); }

task checkCollision(){  while (true) {    if (bump) {      Rev(left);      Wait (50);      Fwd(left);    }  }
}

task main(){  SetSensor(SENSOR_2, SENSOR_TOUCH);  Fwd(left+right);  On(left + right);  start checkCollision;  Wait(myTime);  turn(200);  until(false);  }