Hands-on Lab

BricxCC – Strings, Motors and Touch Sensor

Previously, one learned how to install BricxCC as well as write, compile and execute basic NXC programs. One learned about variable types (e.g. int, and float), conditions (e.g. if-then), and loops (e.g. do-while), as well as displaying both text (i.e. TextOut) and numbers (i.e. FormatNum).

This lab introduces strings as well as output hardware like motors and inputs like NXT buttons and touch sensors.

Concept 1 – Strings and Buttons

A `string` variable is one that is assigned one or more alphanumeric characters. Often, `string` variables are confusing to those who are new to the C language. One must understand the difference between numeric characters "7028951331" and numeric values like 7028951331. The former represents the UNLV Mechanical Engineering phone number; one would not perform math on this `string` variable. The later represents a very large number e.g. the population of New York City; one could perform math (e.g. additions, subtractions, etc) on this `long int` variable.

`String` variables will become more important when the topic of files is introduced in later labs. For now, several examples and functions are given to appreciate the `string` variable.

Step 1: Type and Save the program in **Figure 1-1** as **`strcat1_0a.nxc`**

```
task main ()
// FILE: strcat1_0a.nxc
// DATE: 09/09/22 12:53
// AUTH: P.Oh
// DESC: Demonstrate different string functions
// VERS: 1.0a: strcat, ClearScreen, ClearLine, and Buttons

task main() {
  string firstName, lastName, nickName; // person's first names
  string fullName; // stores result of strcat
  bool rightButtonPushed, leftButtonPushed; // Right and left arrow buttons
  bool orangeButtonPushed; // Center button (to start program)

  firstName = "James" ;
  lastName = "Bond" ;
  nickName = "007" ;

  TextOut(0, LCD_LINE1, "Orange Btn starts" );
  do{
    orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while(!orangeButtonPushed);

  ClearScreen(); // clears anything that's on the NXT brick's screen

  TextOut(0, LCD_LINE7, "-> Next" );

  do {
     rightButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
     TextOut(0, LCD_LINE2, firstName);
     TextOut(0, LCD_LINE3, lastName);
     TextOut(0, LCD_LINE4, nickName);
  } while(!rightButtonPushed) // end do-loop
```

```
  ClearScreen();
  TextOut(0, LCD_LINE7, "<- Quit" );

  fullName = StrCat(firstName, " (", nickName, ") ", lastName);

  do {
     leftButtonPushed = ButtonPressed(BTNLEFT, FALSE);
     TextOut(0, LCD_LINE2, fullName);
  } while(!leftButtonPushed) // end do-loop

  PlaySound(SOUND_DOUBLE_BEEP);

} // end of main
```

Figure 1-1: Program **strcat1_0a.nxc**

Code Description: Highlighted in yellow are new functions. The first is **ButtonPressed,** which is a function that uses the pre-defined variables like BTNCENTER, BTNLEFT, BTNRIGHT and BTNEXIT. These variables reflect the value of the NXT Brick's Orange (center), Left Arrow, Right Arrow, and Grey (bottom) buttons. If a button is pushed, its variable's value becomes TRUE.

ButtonPressed is thus useful for waiting for a user's input. One sees this in the first do-while loop. The loop cycles forever (doing nothing but polling ButtonPressed) and exits when the Orange (center) button is pushed.

The second highlighted line introduced **ClearScreen**. This function simply clears anything that was displayed previously. There is a similar function called **ClearLine**. As its name suggests, it will only clear anything displayed on the specified line e.g. ClearLine(LCD_LINE4) would only clear what was displayed previously in LINE 4 of the Brick's display.

The second do-while loop simply displays the values of string variables firstName, lastName and nickName. This is continuously displayed on the NXT Brick, until the user pushes the Right Arrow button.

The third (and last) highlighted line introduced the **StrCat** function. This function takes as inputs, different strings, that are separated by commas, and concatenates (sticks together) them and assigns the result to the string variable named fullName. The third (and last) do-while loop simply displays fullName, until the user pushes the Left Arrow button. Once pushed, the loop exits, a double-beep is played, and the program ends.

Exercise 1: In NXC create programs for the following:

1-1 Look up the StrLen function. Modify strcat1_0a.nxc to additionally display the number of characters in firstName, lastName, and nickName

Concept 2 – Motors: An NXC program to command NXT motors to move

Step 1: Open a new file and save as "`helloMotor.nxc`". Type the NXC code in **Figure 2-1**, save, compile and run

```
// FILE: helloMotor1_0.nxc
// AUTH: P.Oh
// DATE: 03/16/11
// DESC: Motors connected to Ports A and C.  Command to rotate,
// and counter-rotate fixed amount

task main() {

  OnFwd(OUT_AC, 75);
    // NXC Guide P. 294 (6.36.2.27): OnFwd(byte outputs, char pwr)
    // outputs is OUT_X where X = [A, B, C, AB, AC, BC, ABC]
    // pwr is from [0, 100]
  Wait(5000); // continues for specified milliseconds
  OnRev(OUT_AC, 25);
  // NXC Guide P. 300 (6.36.2.37)
  Wait(2000);

  Off(OUT_AC); // stop and end program gracefully
  StopAllTasks();

}
```

Figure 2-1: `helloMotor.nxc` – Rotate Motors A and C then Counter-rotate them

Step 2: Attached 2 NXT motors (to Ports A and C) on the Brick

Step 3: Save All, Compile, then Download and Run

Code Explanation: The NxC manual describes `OnFwd` and `OnRev` statements. These are specific to NXC and not part of the ANSI-C standard. Often, specific hardware (like NXT motors and sensors) dictate using non-ANSI standard statements. The `OnFwd` statement uses the defined constant `OUT_AC` to reference Brick ports A and C and commands motors connected to these ports to run at 75% of maximum power. Similarly, the `OnRev` statement commands the motors to rotate in the opposite direction at 25% of maximum power. Lastly, `Off` and `StopAllTasks` are additional non-ANSI statements, to stop the motors and exit the program gracefully.

Exercise 2: In NxC create programs for the following:

2-1 Modify helloMotor.nxc by adding a do-while loop. This loop should repeat the OnFwd(OUT_AC, 75), Wait(5000), OnRev(OUT_AC, 25); Wait(2000) cycle 3-times.

2-2 Look up the `RotateMotor` statement (page 308 Section 6.36.2.255). Set Motor A to run at a power level of 75 and rotate to -180 degrees.

Concept 3 – Touch Sensor: An NXC program to detect state of NXT Touch Sensor

Step 1: Connect a motor to Port A of the NXT. Connect a Touch Sensor to Port 1 of the NXT. Then type and save the NXC program "`helloMotor.nxc`" shown in **Figure 3-1**.

```
// FILE: touch1_0a.nxc
// DATE: 09/09/22 15:36
// AUTH: P.Oh
// DESC: Touch sensor (on Port 1) to rotate motor (on Port A) CW or CCW
// VERS: 1.0a: simple rotation CW or CCW

task main() {

  byte touchSensorValue; // value is 0 or 1
  bool orangeButtonPushed, rightButtonPushed, leftButtonPushed, greyButtonPushed;
  byte motorPower; // a value [0, 100] with 100 being highest motor power

  motorPower = 50; // set motor's power to 50% of max speed
  SetSensorTouch(IN_1); // touch sensor connected into Port 1

  TextOut(0, LCD_LINE1, "Orange Btn starts" );
  do{
    orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while(!orangeButtonPushed);

  ClearScreen();
  TextOut(0, LCD_LINE1, "-> Quits" );

  do {
    rightButtonPushed = ButtonPressed(BTNRIGHT, FALSE);
    OnFwd(OUT_A, motorPower); // start rotating CW
    touchSensorValue = Sensor(IN_1); // read touch sensor
    // value of 0 = not pushed and 1 = pushed
    NumOut(0, LCD_LINE4, touchSensorValue);
    if(touchSensorValue == 1) { // touch sensor button was pushed
      do{ // reverse rotation while touch sensor button held down
        OnRev(OUT_A, motorPower);
        touchSensorValue = Sensor(IN_1);
      } while(touchSensorValue == 1); // end do-while
    }; // end if
    // touch sensor button is not longer pushed and/or held down
  } while(!rightButtonPushed); // end do-while

  // Right Arrow button was pushed, so exit gracefully
  Off(OUT_A);
  TextOut(0, LCD_LINE7, "Bye!" );
  PlaySound(SOUND_DOUBLE_BEEP);
  Wait(SEC_5);
  StopAllTasks();
} // end of main
```

Figure 3: File **`touch1_0a.nxc`** – motor rotates CW or CCW depending on state of Touch sensor

Code Description: The first yellow highlight introduces the function **`SetSensorTouch`**. It takes as input the pre-defined variable `IN_1`, `IN_2`, `IN_3`, or `IN_4`. The variable is whichever port the Touch Sensor is plugged into. In this case, it is `IN_1`.

The second yellow highlight uses **`Sensor(IN_1)`** to read the value of the Touch Sensor and assign it to the variable named `touchSensorValue`. Whenever the Touch Sensor is pushed, then `touchSensorValue` will have a value of `1`. When it's not pushed, then the value of `touchSensorValue` will be `0`.

The third yellow highlighted lines of code uses a conditional `if` statement. Here, if the Touch Sensor is pushed (i.e. `touchSensorValue` is 1), a `do-while` loop is executed. The `do-while` loop reverses the motor's rotation using an `OnRev` statement. The `do-while` loop also polls the Touch Sensor's value. It exits the loop if the Touch Sensor is released, and returns to rotating the motor forward with the `OnFwd` statement.

Exercise 3: In NxC create programs for the following:

3-1 In touch1_0a.nxc one observes that the Touch Sensor must be held down in order to rotate in the reverse direction. Modify the program so that if the Touch Sensor is only pushed (and not held down), the motor reverses direction. When the Touch Sensor is pushed again, the motor moves in the forward direction.