

Chapter 1: Process Control and Flowcharts

BEFORE YOU START

To perform the experiments in this text, you will need to have your Board of Education connected to your computer, the BASIC Stamp Editor software installed, and to have verified the communication between your computer and your BASIC Stamp. For detailed instructions, see *What's a Microcontroller?* - a free download from www.parallax.com. You will also need the parts contained in the Process Control Parts Kit. For a full listing of system, software, and hardware requirements, see Appendix B.

WHAT IS PROCESS CONTROL?

Process control refers to the control of one or more system parameters, such as temperature, flow rate or position. While most systems are a continual process, such as maintaining a temperature, other processes may be a sequence of actions, for example, the assembly of a product.

Control systems can be very simple or very complex. Figure 1-1 is a block diagram of a simple continuous control system. For control of the process, an input (such as a setpoint control or switch) is required into the controller. Based on the input, the controller will drive an actuator to cause the desired effect on the process.

Examples of actuators are heaters for temperature, pumps for flow, and servos for positioning.

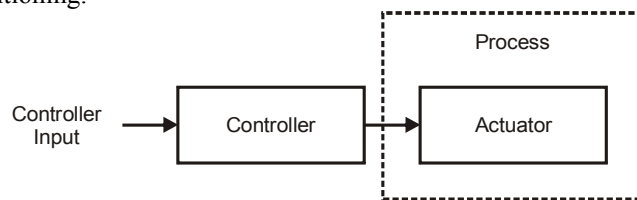


Figure 1-1
Simple Process
Control Block
Diagram

Consider the example of a common car heating system. The driver adjusts a temperature control to change the heat output of the vents. If the driver becomes too warm when weather conditions change, the temperature control must be adjusted to return to a comfortable temperature. This is a very simple system in that most automobiles do not monitor the cabin with temperature sensors to automatically control the heat output of the vents.

A more sophisticated system would have a sensor to monitor temperature and provide feedback to the controller. The controller would automatically adjust the actuator to regulate the controlled parameter - temperature. The controller would drive the heating system to maintain the temperature near the defined set point. An example of this is your home heating system.

Consider the difference between how the cabin temperature of the automobile is controlled versus the temperature in a home. In the automobile, the heat output is variable but has no sensors that directly affect the heat output and maintain temperature. In home heating a sensor is used to monitor the temperature, but the output of the heating system is not variable; it is either on or off and cycles to maintain temperature in a comfortable band. The controller itself may be very simple, such as a metallic coil that expands and contracts, or more complex, such as a microcontroller similar to the BASIC Stamp.

These are two very unique means of controlling a process. First, the types of drive employed may be variable or on/off. Second, whether feedback from the system may or may not be used in the control of the system.

INPUT, DRIVE AND MONITORING

Just as important as the type of control employed are the methods used for the input into the controller. Will the inputs provide a simple on/off input to the controller? If using an analog (variable level) input instead of a digital one (only two levels), how can it be conditioned for on/off input if needed? If analog input is required, what are methods to bring this data into the BASIC Stamp? How is the data represented in the BASIC Stamp and how can it be converted to meaningful information?




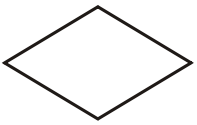
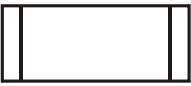


In terms of the drive of a system, there are several questions as well. Do we need to employ on/off control of the actuator, such as turning a heater or pump on or off? Does the process require variable control of the drive such as regulating heat or flow output between on or off? Does the process actuator require higher current or voltage than is provided by the BASIC Stamp? How can the BASIC Stamp outputs be used to control these actuators?

In industry, monitoring of systems is often required in order to ensure proper control action and to determine response in order to adjust this control action. Another monitoring aspect is data logging, or being able to collect real time data from the system for analysis.

This text explores these areas of process control through simple circuits using the BASIC Stamp microcontroller, and illustrates use with much larger systems.

ACTIVITY #1: FLOWCHARTS FOR REPRESENTING PROCESSES

When you hear the word ‘flowchart’, it may bring to mind programming, but a flowchart is often used for more than programming. A flowchart is a graphical representation of steps and decisions used to arrive at a logical outcome. It can be used to arrive at management decisions, system troubleshooting decisions, and other processes that involve well-defined steps and outcomes. Table 1-1 shows the most popular symbols used in flowcharting. These blocks, connected with flow lines, are used to describe the actions and flow of the program.

Table 1-1: Flowcharting Symbols	
	Start/Stop: Indicates the beginning or end of a program or routine.
	Process: Indicates an internal process, such as calculations or delays.
	Input/Output: Indicates an input from an external source or output to an external source.
	Decision: Indicates a decision to continue flow in one of two directions based on a condition.
	Predefined Process: Indicates a predefined process, such as a subroutine, to be performed.
	Matching connectors indicate a connection between two locations in the flowchart.
	Flow lines: Indicates direction of flow between symbols.

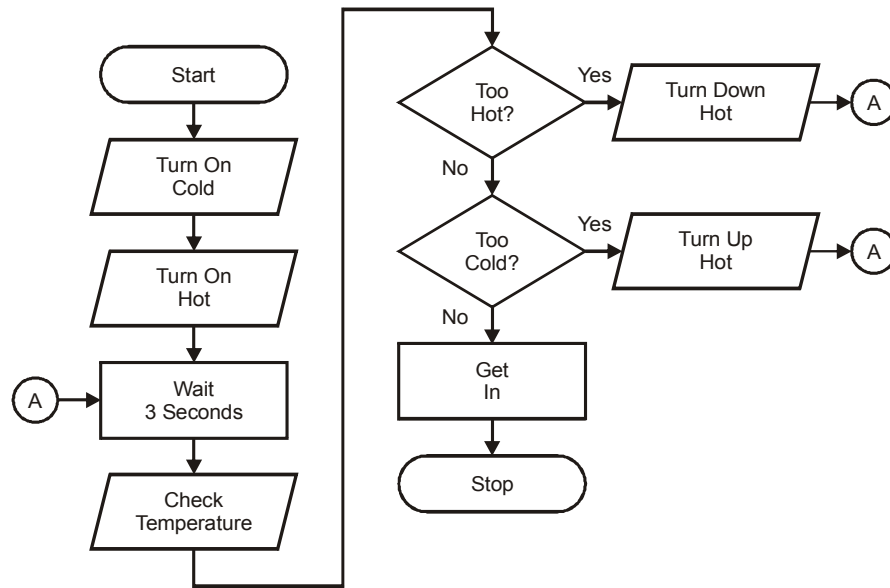
While flowcharting has fallen out of fashion in many programming circles due to the advent of object-oriented programming (PBASIC used by the BASIC Stamp is procedural language), it is still an excellent tool when planning program flow. Flowcharting is particularly useful in process control because it can be used to visually represent the steps and decisions required to perform control of the system.

Take the everyday task of preparing the temperature of the shower before stepping into it. In pseudocode, or English statements outlining the steps to take, this is how we would proceed:

1. Turn on cold water.
2. Turn on hot water.
3. Wait 3 seconds for temperature to stabilize.
4. Test water temperature.
5. If too hot, then:
 - a. Turn hot water down.
 - b. Go back to step 3.
6. If too cold, then:
 - a. Turn hot water up.
 - b. Go back to step 3.
7. If just right then get in shower.

While it's not too difficult to read through these steps to see what actions should be taken, as a program or procedure becomes more complex it becomes more difficult to visualize the flow of the process and what actions and branches are needed. For example, how much more complex would the flow be if the hot water valve becomes fully open before the optimum temperature is reached?

As complexity increases, a flowchart makes it easier to visualize how the process will flow. Take a look at the flowchart in Figure 1-2, which describes the same process as the pseudocode above.

Figure 1-2 Adjusting Shower Temperature Flowchart

Note how each of the symbols is used.

- Typically, an input/output symbol is used when bringing data or information into the controller (in this case the person adjusting the temperature by sensing and adjusting the water actuators).
- The processing symbol is used when the controller is performing internal processing of data or a task, such as waiting or calculations.
- Finally, decision blocks are used to guide the flow of the procedure in one direction or another based on the decision results.

A decision can take one of two forms:

- Questions resulting in Yes or No.
- Statements resulting in True or False.

As humans, we typically work with questions resulting in yes/no. In flowcharting, it is better to use statements that result in true/false due to the logical nature of programming where conditions are checked to be true or false. Take the following example for the shower process:

- Is the water too hot? YES – Turn down the hot.
- The water is too hot. TRUE – Turn down the hot.

In programming, a typical condition may be:

```
IF (Water_Temp > 95) THEN ...
```

In this example, when the condition is checked, the equality will either be true or false. Using true/false statements makes the transition from the flowchart to the programming language easier.

Challenge 1-1: Modify the Flowchart for True/False

- ✓ Modify the flowchart in Figure 1-2 to use true/false statements instead of yes/no questions.

ACTIVITY #2: SEQUENTIAL FLOW AND CODE

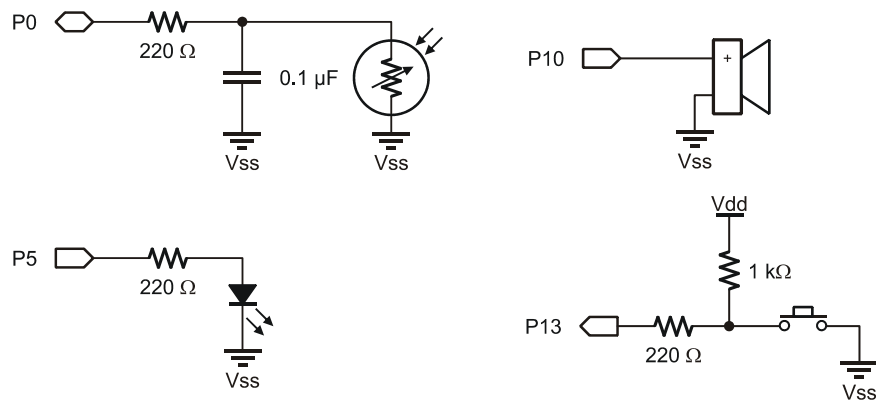
"Sequential flow" means moving from one operation to the next with no branches being made. In this activity a simple circuit will be used to illustrate principles of sequential flow and how the PBASIC language is used in programming the BASIC Stamp.

Parts Required

- (3) Resistors – 220 Ω
- (1) Resistor – 1 k Ω
- (1) Photoresistor
- (1) Pushbutton – Normally Open
- (1) LED – Red
- (1) Piezospeaker
- (1) Capacitor – 0.1 μ F

- ✓ Construct the photoresistor, LED, piezospeaker, and pushbutton circuits shown in Figure 1-3.

Figure 1-3 Test Circuit Schematics



For an introduction to building basic circuits with these components, please see *What's a Microcontroller?*, the recommended starting point for the Stamps in Class series. It is available for free download or purchase from www.parallax.com.

Figure 1-4 is a flowchart to have the circuit continuously perform a sequence of operations. Without knowing any programming, can you determine what should occur when the program is entered and run?

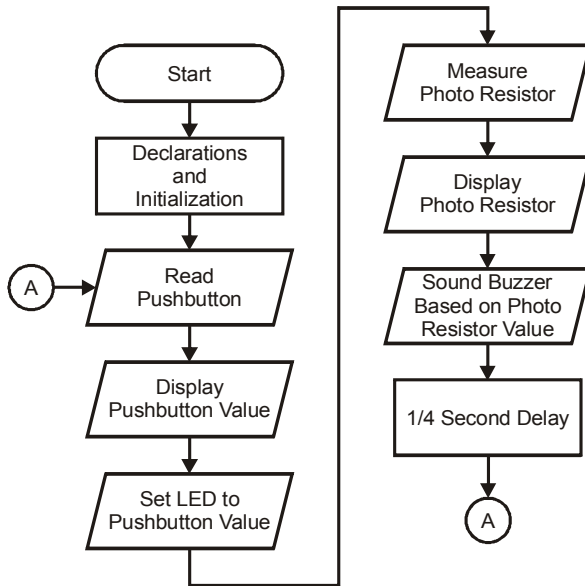


Figure 1-4
Simple Sequential
Operation Flowchart



AVOID TYPOS! All of the BASIC Stamp (.bs2) programs listed in this text are available for free download from the Process Control product page at www.parallax.com.

Example Program: SimpleSequentialProgram.bs2

✓ Enter and run SimpleSequentialProgram.bs2.

```

' -----[ Title ]-----
' Process Control - SimpleSequentialProgram.bs2
' Tests and illustrates sequential flow using a simple test circuit.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo  PIN  0          ' Alias for photo resistor circuit on P0
LED     PIN  5          ' Alias for LED on P5
Buzzer  PIN  10         ' Alias for buzzer on P10
PB      PIN  13         ' Alias for pushbutton on P13
PBVal   VAR  Bit       ' Bit variable to hold pushbutton value
PhotoVal VAR  Word     ' Word variable to hold RC Time value
  
```



```

BuzzerDur CON 250          ' Constant for duration of tone for buzzer

' -----[ Initialization ]-----
OUTPUT LED                 ' Set LED pin to be an output
OUTPUT Buzzer              ' Set Buzzer pin to be an output

' -----[ Main Routine ]-----
DO
  ' ***** Read Pushbutton
  PBVal = PB                ' Read Pushbutton value and assign to PBVal
                           ' Display Pushbutton value

  ' ***** Display pushbutton value

  DEBUG CLS, "Pushbutton Value = ", DEC PBVal, CR

  ' ***** Set LED to pushbutton value
  LED = PBVal              ' Set LED based on Pushbutton value

  ' ***** Measure Photoresistor
  HIGH Photo               ' Charge photoresistor's RC network Capacitor
  PAUSE 10                 ' Allow 10 milliseconds to charge fully
  RCTIME Photo, 1, PhotoVal ' Measure discharge time through photoresistor

  ' ***** Display photoresistor value
  DEBUG "Photo RC Time Value = ", DEC PhotoVal, CR

  ' ***** Sound buzzer at set duration at frequency of PhotoVal
  FREQUOT Buzzer, BuzzerDur, PhotoVal

  ' ***** 1/4 seconds delay
  PAUSE 250                ' 1/4 second pause
LOOP                       ' Loop back to DO to repeat continuously

```

- √ Test the circuit by pressing the pushbutton and varying the light falling on the photoresistor.
 - When the button is pressed does the state of the pushbutton change from 1 to 0 in the Debug Terminal?
 - When the button is pressed does the LED change from on to off?
 - When the sensor is darkened does the photoresistor RC time value change in the Debug Terminal?
 - Does the frequency output of the buzzer change in relation to the photoresistor's RC time value? Note that the buzzer has a very limited frequency response range.
- √ If your circuit does not operate properly, verify your circuit connections and code.

Code Discussion

As you read through the program, you can see that the coding that corresponds to the various elements of the flowchart are well highlighted using comments.

The pushbutton switch is active-low, meaning that its value is 0 when pressed. This is because the pushbutton is pulled up to Vdd when not pressed and brought to Vss when pressed. (This will be explored more in Chapter 3.)

Note that the flowchart block for 'Measure Photo Resistor' takes 3 lines of code. The flowchart just describes the process and is not intended to be a line-by-line description. This flowchart could be used for coding or designing any number of devices in any number of languages.



Looking it Up: The PBASIC commands and programming techniques used here were introduced in *What's a Microcontroller?*, the recommended prerequisite to *Process Control*. If you would like a refresher about specific program elements, you can look it up quickly in the BASIC Stamp Editor's Help file. Or, refer to the *BASIC Stamp Syntax and Reference Manual*, available for purchase or free download from www.parallax.com.

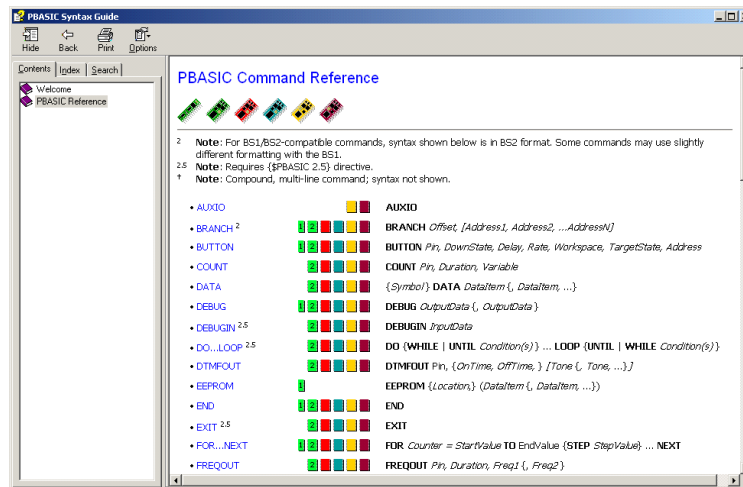


Figure 1-5
BASIC Stamp
Editor's Help
Files

*The PBASIC
Syntax Guide
places
information and
examples for all
commands at
your fingertips.*

Challenge 1-2: Coding from a Flowchart

Figure 1-6 is a flowchart for a different sequence of operations, using the same circuit. Code a program to match this sequence of events. Hints for coding are provided in the flow symbols.

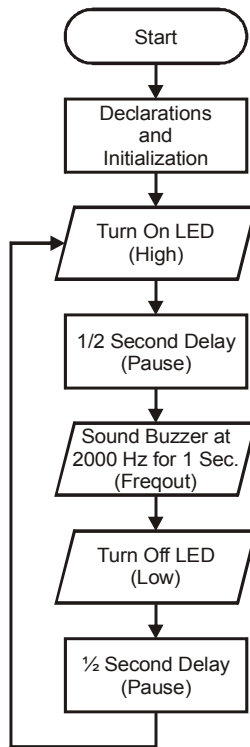


Figure 1-6
Challenge 1-2
Flowchart

ACTIVITY #3: FLOW AND CODING WITH CONDITIONAL BRANCHES

In most processes, measurements are made and decisions are then based on those measurements (such as, in the shower example, whether to turn up or down the hot water based on the current temperature). In the BASIC Stamp, there are multiple ways to code decisions and conditional branches.

Parts Required

Same as Activity #2

Consider the flowchart in Figure 1-7. What should occur when the button is pressed, and when it is not pressed?

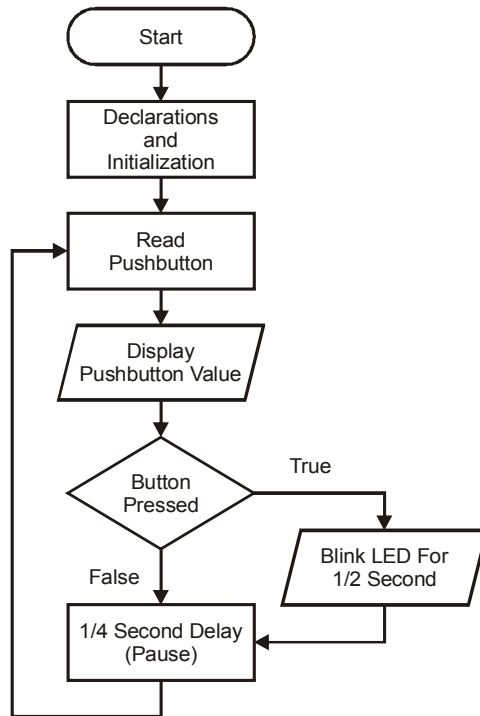


Figure 1-7
Conditional LED Blink
Flowchart

If you said the LED would blink on for ½ second when the button is pressed, and not at all when not pressed, you would be correct.

Example Program: ConditionalLEDBlink.bs2

✓ Enter, save and run ConditionalLEDBlink.bs2.

```
' -----[ Title ]-----
' Process Control - ConditionalLEDBlink.bs2
' Blinks the LED based on state of Pushbutton
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN    10      ' Alias for photo resistor circuit on P0
LED       PIN     5      ' Alias for LED on P5
Buzzer    PIN    10      ' Alias for buzzer on P10
PB         PIN    13      ' Alias for pushbutton on P13
PBVal     VAR    Bit     ' Bit variable to hold pushbutton value
PhotoVal  VAR    Word    ' Word variable to hold RC Time value
BuzzerDur CON  250      ' Constant for duration of tone for buzzer

' -----[ Main Routine ]-----
DO
  ' ***** Read Pushbutton
  PBVal = PB              ' Read Pushbutton Value and assign to PBVal

  ' ***** Display Pushbutton value
  DEBUG CLS,"Pushbutton value = ", DEC PBVal,CR

  ' ***** Button Pressed Conditional and Code
  IF (PBVal = 0) THEN     ' If pushbutton pressed is true then,
    HIGH LED             ' blink the LED
    PAUSE 500
    LOW LED
  ENDIF
  ' ***** 1/4 second pause
  PAUSE 250
LOOP                    ' Loop back to DO to repeat continuously
```

Code Discussion

The **IF...THEN...ENDIF** block is used to test the condition. Based on the result, the program will execute the code within the block if true or skip over it if false.

```

IF (PBVal = 0) THEN      ' If condition is true then,
    HIGH LED             ' blink the LED
    PAUSE 500
    LOW LED
ENDIF
' ***** 1/4 second pause
PAUSE 250

```

When the button is not pressed, the conditional test of **PBVal=0** will result in false because the value of **PBVal** is 1. Execution will branch to after the **ENDIF**, executing the **PAUSE 250**.

When the button is pressed, **PBVal** will in fact equal 0; **PBVal=0** will be true, the code within the block will be executed, and the LED will blink.



Code Formatting Tip: While indents in lines are not required, they do help to visually represent code that is common to sections.

Challenge 1-3: Code for True and False Conditions

Many times, different code must be executed depending on whether a condition is true or false. The **IF...THEN...ELSE...ENDIF** structure can be used to perform this task. If the condition is false, the code in the **ELSE** section will be executed.

```

IF (condition) THEN
    Code to run if true
ELSE
    Code to run if false
ENDIF

```

- ✓ Figure 1-8 is a flowchart that requires different code depending on whether the button is pressed or not. Modify ConditionalLEDBlink.bs2 to match the flowchart's operation.

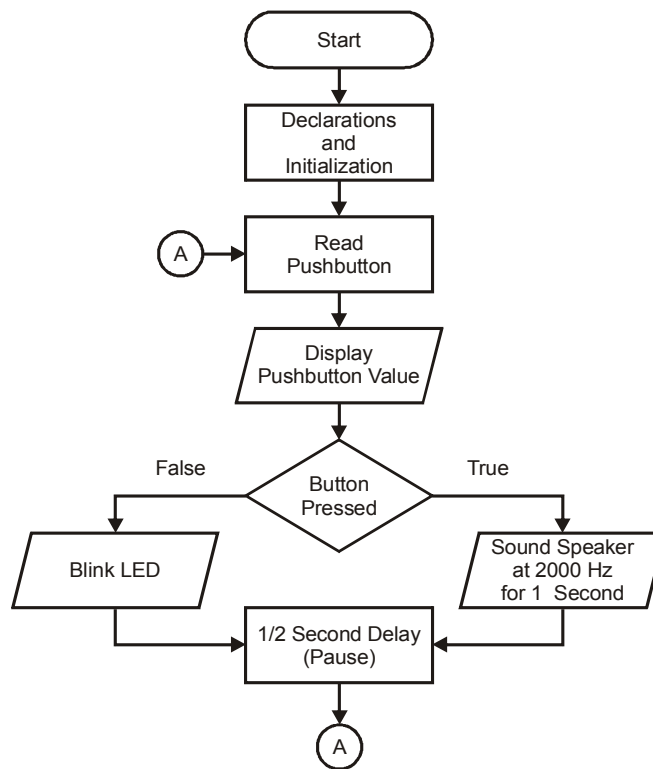


Figure 1-8
Conditional LED
Blink or Tone
Flowchart

ACTIVITY #4: PREDEFINED PROCESSES WITH SUBROUTINES

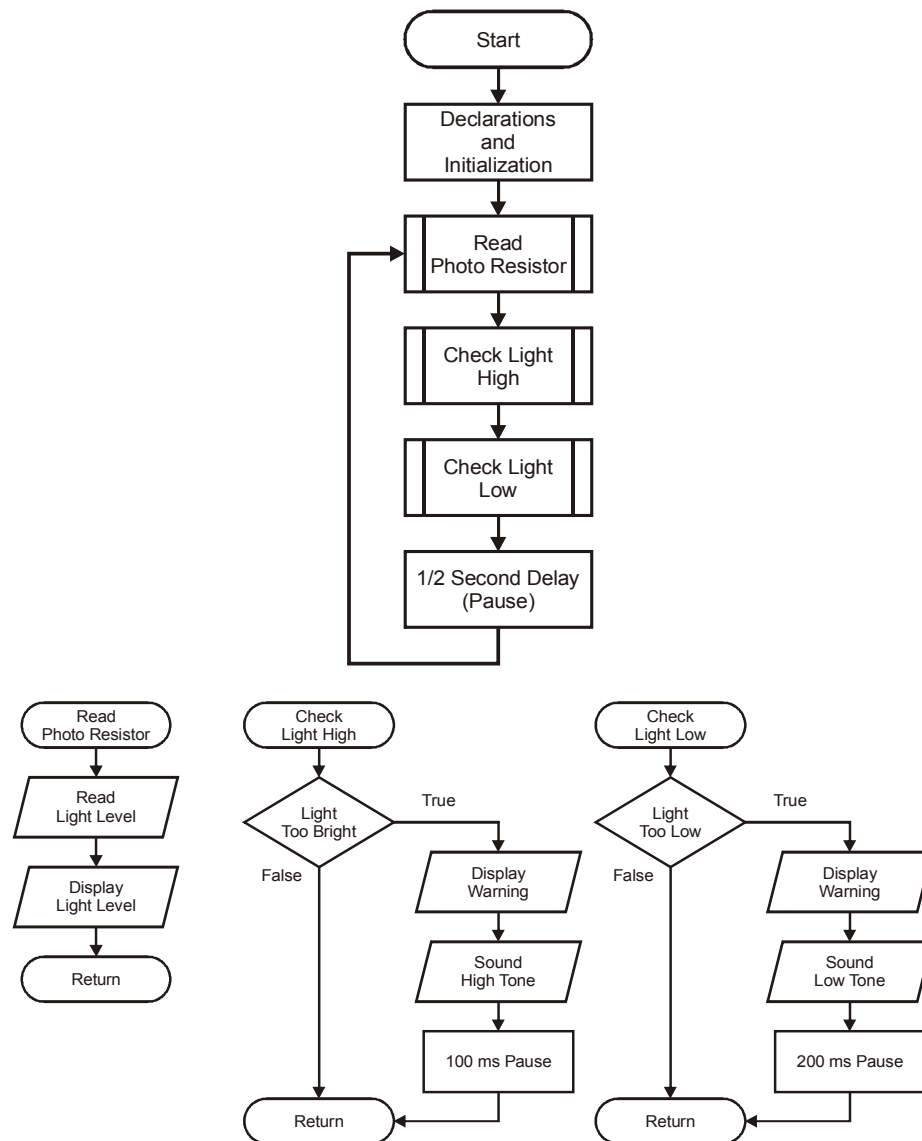
Parts Required

Same as Activity #2

As more operations are added to the flowchart, it can become quite large and complex. The same holds true for programs. In the previous programs, all operations were performed within the main routine, and the same held true in the flowchart.

As the process increases in size and complexity, it is best to break it down into more manageable pieces. By looking at the main loop of the flowchart or the main routine of the code, it is easy to see the overall operation of the program without being overwhelmed by the amount of code. Finally, analyzing or troubleshooting is much easier if it can be performed without having to flip between several pages or continually scroll up or down to different sections of the program. For example, consider the flowchart in Figure 1-9.

Looking at the main loop, it is easy to see the overall operation of the process. The pre-defined processes take the place of specialized code to perform these operations. Each pre-defined process has its own flowchart to define its operation. How will the process operate based on this flowchart? What occurs if the light level is low?

Figure 1-9 Light Alarms using Predefined Processes Flowchart

Example Program: LightAlarmWithSubroutines.bs2

✓ Enter and run LightAlarmWithSubroutines.bs2.

```
' -----[ Title ]-----
' Process Control - LightAlarmWithSubroutines.bs2
' Sounds alarm based on photoresistor readings
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN  0      ' Alias for photo resistor circuit on P0
LED       PIN  5      ' Alias for LED on P5
Buzzer    PIN 10      ' Alias for buzzer on P10
PhotoVal  VAR Word   ' Variable to hold RC Time value
PhotoMin  VAR Word   ' Holds minimum light level value
PhotoMax  VAR Word   ' Hold maximum light level value

' ---[ Initialization ] -----
PhotoMin  = 500       ' Set minimum light value
PhotoMax  = 5000      ' Set maximum light value
PAUSE 1000           ' Allow connection to stabilize -- for Chapter 2

' -----[ Main Routine ]-----
DO
  GOSUB ReadPhoto
  GOSUB CheckLightHigh
  GOSUB CheckLightLow
  PAUSE 500
LOOP

' -----[ Subroutines ]-----

ReadPhoto:           ' Read light level and plot values
  HIGH Photo
  PAUSE 10
  RCTIME Photo,1,PhotoVal
  DEBUG DEC PhotoVal, ",", DEC PhotoMin, ",", DEC PhotoMax,CR
RETURN

CheckLightHigh:      ' Test if high light level
  IF (PhotoVal < PhotoMin) THEN
    DEBUG "LIGHT LEVEL HIGH!",CR
    FREQOUT Buzzer,100,3000
    PAUSE 100
  ENDIF
RETURN

CheckLightLow:       ' Test if low light level
  IF (PhotoVal > PhotoMax) THEN
    DEBUG "LIGHT LEVEL LOW!",CR
```

```

FREQOUT Buzzer,200,1000
PAUSE 200
ENDIF
RETURN

```

- ✓ Move your hand over the photoresistor, and watch the Debug Terminal. What occurs as the light level RC time is
 - Less than 500?
 - Between 500 and 5000?
 - Greater than 5000?

Your Debug Terminal should look similar to Figure 1-10. It displays the current level and the low- and high-level set points.

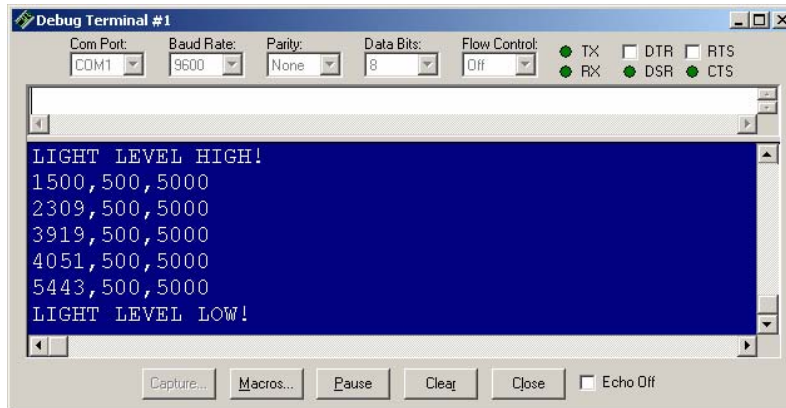


Figure 1-10
Debug
Terminal
Light Level
Alarms



Values or tolerances of the photoresistor and capacitor may vary along with ambient light level where you are. Adjust the high and low level setpoints accordingly in the initialization section of your code.

Code Discussion

Using `GOSUB...RETURN` works well with our flowchart structure. The subroutines are the pre-defined processes. When the `GOSUB` call is run, program execution branches to the named routine. The routine code is executed. When complete, `RETURN` causes execution to branch back to the code after the `GOSUB` call.



Programming Tip: Every routine called with a `GOSUB` must exit with a `RETURN`. Internal pointers keep track of `GOSUBs` and `RETURNS`, and if not matched properly, will result in erroneous behavior of the processor.

Challenge 1-4: Add an Operational Indicator

1. Add a pre-defined process block to the main loop of the flowchart in Figure 1-9.
2. Also add a process flowchart to turn on the LED for 0.25 seconds at every pass through the main loop in order to indicate proper operation of the system.
3. Add code to `LightAlarmWithSubroutines.bs2` to match the flowchart.

ACTIVITY #5: CONDITIONAL LOOPING

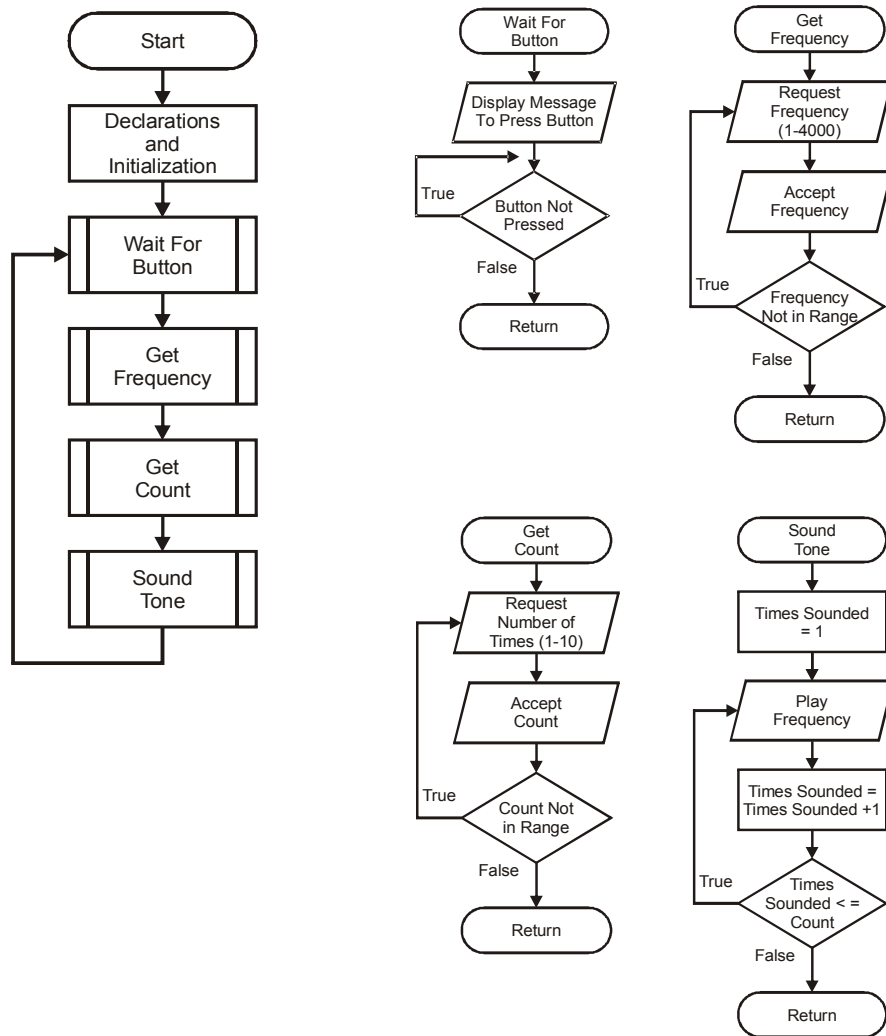
Many times in a process it is necessary to repeat a sequence based on a condition. On the other hand, halting or pausing an execution until a certain condition exists may be required. Consider the process of starting a piece of industrial machinery. Until conditions are met, such as an oil pump running, there may be no need to continue farther into the process. A conditional loop could be used to ensure that a condition exists prior to continuing with the sequence.

Parts Required

Same as Activity #2

Examine the Conditional Looping flowchart in Figure 1-11.

Figure 1-11 Conditional Looping Flowchart



Example Program: ConditionalLooping.bs2

- ✓ Enter, save and run ConditionalLooping.bs2.
- ✓ To begin, press the pushbutton as directed by the Debug Terminal.
- ✓ Enter a frequency to play and the number of times to play it by typing a value into the white text box at the top of the Debug Terminal, and then pressing Return or Enter.
- ✓ Test using valid and invalid values.

```

' -----[ Title ]-----
' Process Control - ConditionalLooping.bs2
' Sounds tone using conditional loops
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN  0      ' Alias for photo resistor circuit on P0
LED       PIN  5      ' Alias for LED on P5
Buzzer    PIN 10      ' Alias for buzzer on P10
PB        PIN 13      ' Alias for pushbutton on P13
PBVal     VAR Bit     ' Bit variable to hold pushbutton value
PhotoVal  VAR Word    ' Variable to hold RC Time value
FreqVal   VAR Word    ' Frequency to sound
CountVal  VAR Byte    ' Number of tones to sound
X         VAR Byte    ' General Counting variable

' -----[ Main Routine ]-----
DO
  GOSUB WaitForButton
  GOSUB GetFreq
  GOSUB GetCount
  GOSUB SoundTone
  PAUSE 1000
LOOP

' -----[ Subroutines ]-----
WaitForButton:
  DEBUG CLS, "Press the pushbutton to begin",CR
  DO
    LOOP WHILE (PB=1)
  RETURN

GetFreq:
  DO
    DEBUG CR,"Enter the frequency to play (1 to 4000)",CR
    DEBUGIN DEC FreqVal
    LOOP UNTIL (FreqVal <= 4000) ' loop until within range
  RETURN

```

```

GetCount:
DO
    DEBUG CR,"Enter the number of times to play (1 to 10)",CR
    DEBUGIN DEC CountVal
    LOOP WHILE (CountVal > 10)      ' loop while out of range
RETURN

SoundTone:
FOR X = 1 TO CountVal              ' Start X at 1 for counting up to CountVal
    FREQOUT Buzzer,500,FreqVal
    DEBUG "Buzzing ", DEC X,CR
NEXT                               ' Add 1 to X and loop if X <= CountVal
RETURN

```

Program Discussion

The ConditionalLooping.bs2 program uses conditional loops in a variety of ways. The **DO...LOOP WHILE** within the **WaitForButton** routine will repeat while the condition is true. This occurs while the value of the pushbutton input is 1 or not pressed. Once the pushbutton is pressed, the condition will be false and the loop will end.

In the **GetFreq** routine, the **DO...LOOP UNTIL** will repeat until a value within range has been entered. **DEBUGIN** accepts data from the Debug Terminal and stores it as a decimal in the **FreqVal**.

In **GetCount**, a **DO...LOOP WHILE** is used to request the number of times to play the tone and will repeat while the value is outside the appropriate range.

In **SoundTone**, a **FOR...NEXT** loop is used. This is a special conditional loop used for repeating a sequence a set number of times:

```
FOR variable = Start_Value TO End_Value
```

The loop begins with the defined value set to the **Start_Value**. The code within the loop is performed. When **NEXT** is encountered, the variable is incremented and checked against the **End_value**. If the variable is not greater than the **End_value**, the loop repeats. **x** is started at 1 and the loop continues until **x** exceeds the value entered by the user for **CountVal**.

Compare the flowcharts to the code for each routine. The use of either **WHILE** or **UNTIL** is at the programmer's discretion as long as it performs the task intended.

Challenge 1-5

- ✓ Save ConditionalLooping.bs2 under a new name, then add variables and code required to allow the user to enter the duration the tone should be played (entered in milliseconds). Limit the maximum allowable duration to 1000 milliseconds.

CONCLUSION

Process control refers to the control of one or more system parameters. Typically, some form of input is used to adjust this process. A simple process, such as controlling temperature, may be performed in multiple ways. The control and complexity of the system is based on need. For process control the BASIC Stamp is ideally suited for many systems.

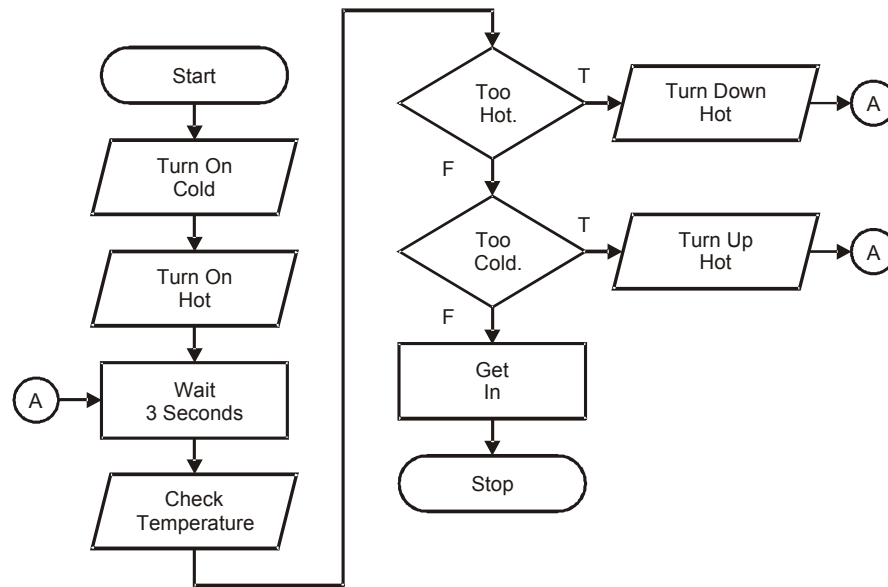
Flowcharts are a visual representation of a program or a process. The flowchart represents the necessary steps to perform the desired actions. Through the use of symbols the actions of the program or process are graphically depicted. With knowledge of PBASIC, the programming language of the BASIC Stamp, the process represented by the flowchart may be programmed into the BASIC Stamp.

Means to control output devices include using the **HIGH** and **LOW** commands; **FREQOUT** is used to sound tones; data may be sent to the computer using the **DEBUG** instruction. Conditions may be checked and simple true/false decisions may be made using the **IF...THEN** instructions. Looping is performed using the **DO...LOOP**, and adding **WHILE** or **UNTIL** looping may be performed conditionally. Programs may be broken down into smaller processes that are called with the **GOSUB** command and exited with the **RETURN** command.

SOLUTIONS TO CHAPTER 1 CHALLENGES

Challenge 1-1 Solution

Figure 1-12 Shower Temperature Flowchart with True/False



Note that the yes-no questions became true-false statements.

Challenge 1-2 Solution

Your program might look like this:

```

' -----[ Title ]-----
' Process Control - SimpleFlowchartChallenge.bs2
' Code from flowchart
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN    0           ' Alias for photoresistor circuit on P0
LED       PIN    5           ' Alias for LED on P5
Buzzer    PIN   10          ' Alias for buzzer on P10

```

```

' -----[ Main Routine ]-----
DO
  HIGH LED          ' Turn ON LED
  PAUSE 500          ' 1/2 second delay
  FREQOUT Buzzer, 1000, 2000 ' Sound buzzer at 2000Hz for 1 second
  LOW LED           ' Turn OFF LED
  PAUSE 500         ' 1/2 second delay
LOOP                ' Loop back to DO to repeat continuously

```

Challenge 1-3 Solution

```

' -----[ Title ]-----
' Process Control - ConditionalLEDBlinkChallenge.bs2
' Modify ConditionalLEDBlink for If-Else
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN  0      ' Alias for photo resistor circuit on P0
LED       PIN  5      ' Alias for LED on P5
Buzzer    PIN 10      ' Alias for buzzer on P10
PB        PIN 13      ' Alias for pushbutton on P13
PBVal     VAR Bit     ' Bit variable to hold pushbutton value
PhotoVal  VAR Word    ' Word variable to hold RC Time value
BuzzerDur CON 250     ' Constant for duration of tone for buzzer

' -----[ Initialization ]-----

' -----[ Main Routine ]-----
DO
  ' ***** Read Pushbutton
  PBVal = PB          ' Read Pushbutton Value and assign to PBVal

  ' ***** Display Pushbutton value
  DEBUG CLS,"Pushbutton value = ", DEC PBVal,CR

  ' ***** Button Pressed Conditional and Code
  IF (PBVal = 0) THEN ' If pushbutton pressed is true then,
    FREQOUT Buzzer, 1000,2000 ' True - Sound buzzer
  ELSE
    HIGH LED          ' False - Blink the LED
    PAUSE 500
    LOW LED
  ENDIF
  ' ***** 1/4 second pause
  PAUSE 250
LOOP                ' Loop back to DO to repeat continuously

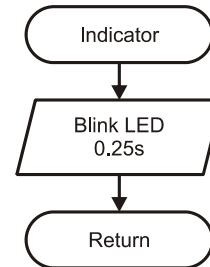
```

Challenge 1-4 Solution

1. To the main flowchart's loop, add another predefined process:



2. Make a flowchart for the predefined process (names should match):



3. To the programs Main Routine `DO...LOOP` add a subroutine call for your predefined process:

```
GOSUB Indicator
```

Under the Subroutines section, add the subroutine for the predefined process:

```
Indicator:
  HIGH LED
  PAUSE 250
  LOW LED
  RETURN
```

Challenge 1-5 Solution

```
' -----[ Title ]-----
' Process Control - ConditionalLoopingChallenge.bs2
' Sounds tone using conditional loops
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----
Photo    PIN    0      ' Alias for photo resistor circuit on P0
LED       PIN    5      ' Alias for LED on P5
Buzzer    PIN    10     ' Alias for buzzer on P10
PB        PIN    13     ' Alias for pushbutton on P13
PBVal     VAR    Bit    ' Bit variable to hold pushbutton value
PhotoVal  VAR    Word   ' Variable to hold RC Time value
FreqVal   VAR    Word   ' Frequency to sound
CountVal  VAR    Byte   ' Number of tones to sound
DurVal    VAR    Word   ' Duration to sound tone
X         VAR    Byte   ' General Counting variable
```

```

' -----[ Main Routine ]-----
DO
  GOSUB WaitForButton
  GOSUB GetFreq
  GOSUB GetCount
  GOSUB GetDuration
  GOSUB SoundTone
LOOP

' -----[ Subroutines ]-----
WaitForButton:
  DEBUG CLS, "Press the pushbutton to begin",CR
  DO
    LOOP WHILE (PBVal=1)
  RETURN

GetFreq:
  DO
    DEBUG CR,"Enter the frequency to play (1 to 4000)",CR
    DEBUGIN DEC FreqVal
    LOOP UNTIL (FreqVal <= 4000)    ' loop until within range
  RETURN

GetCount:
  DO
    DEBUG CR,"Enter the number of times to play (1 to 10)",CR
    DEBUGIN DEC CountVal
    LOOP WHILE (CountVal > 10)      ' loop while out of range
  RETURN

GetDuration:
  DO
    DEBUG CR,"Enter the duration to play tone in milliseconds (0 to 1000)",CR
    DEBUGIN DEC DurVal
    LOOP WHILE (DurVal > 1000)      ' loop while out of range
  RETURN

SoundTone:
  FOR X = 1 TO CountVal             ' Start X at 1 for counting up to CountVal
    FREQOUT Buzzer,DurVal,FreqVal
    DEBUG "Buzzing ", DEC X,CR
  NEXT                               ' Add 1 to X and loop if X <= CountVal
  RETURN

```