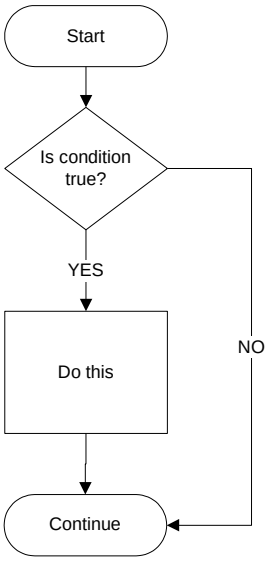
Flow control statements are statements that alter the flow of statement execution in a C program. Four important statements are **if()**, **else**, **while()**, and **for()**.

Conditional execution: if() and else

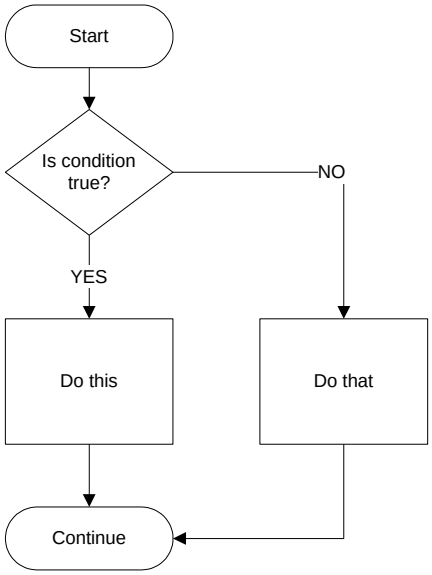
- The if() statement allows a statement of block of statements to be executed, based on a condition. If the condition is true, the statement(s) will be executed. If the condition is false, the statement(s) will be skipped.
- An else statement allows execution of an alternative statement or block of statements if the condition for the prior if() statement is false.

Examples (if and if-else):

```
if(condition)
{
    this;
}
```

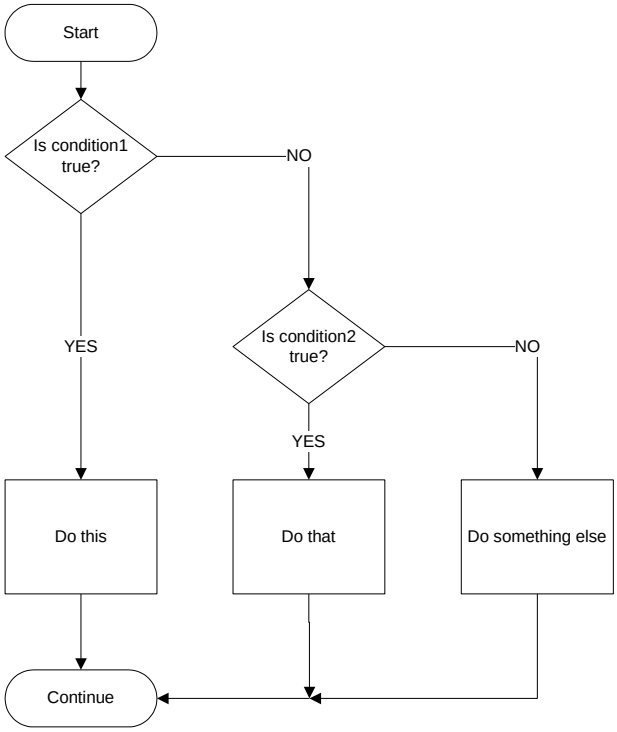



```
if(condition)
{
    this;
}
else
{
    that;
}
```



Example (nested if-else):

```
if(condition1)
{
    this;
}
else
{
    if(condition2)
    {
        that;
    }
    else
    {
        something else;
    }
}
```



- If statements can be "nested". That is, one if() can be inside the block of statements for another if(). This is where it is really important to use good style for indentation and commenting.

Example:

```
if(x<=15)
{
      if{x>7)
      {
            PAOUT = 0x03; //turn on green LED
      }//end if
      else
      {
            PAOUT = 0x05; //turn on yellow LED
      }//end else
}//end if
else
{
      PAOUT = 0x06; //turn on RED LED
}//end else
```

Exercise:
Complete the table with the range of values of x when one LED is turned on, as specified. The first table entry is completed as an example. Refer to the handout on variable types for the range of variables of type **char** and **unsigned char**.

LED ON	**Range of values of x (assume x is declared as type char)**
Green	$8 \leq x \leq 15$
Yellow	
Red	

LED ON	**Range of values of x (assume x is declared as type unsigned char)**
Green	
Yellow	
Red	

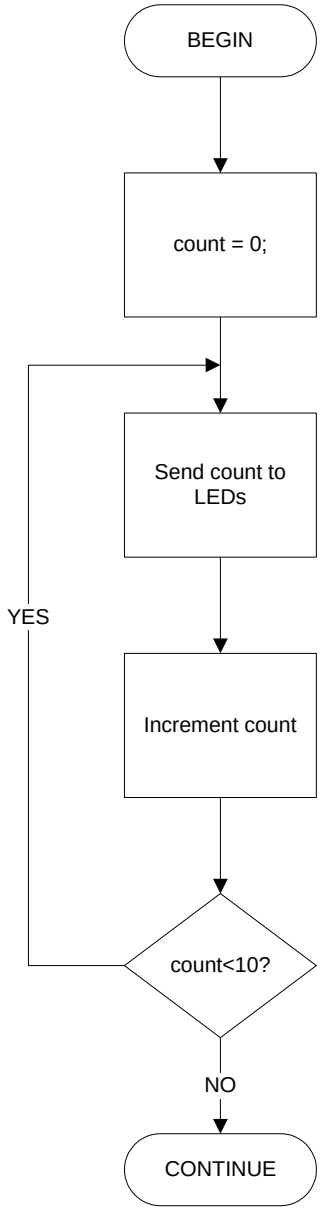
Repetitive execution: while() and for() loops

Two statements in C can be used for repetitive statement execution: **while()** and **for()**. Each of these has conditions in the statement parentheses that determine whether a block of code should execute or not. A while() will execute as long as its condition is true; a for() will execute a specified number of times.

The for() statement syntax is **for(initialize; test; modify){C statement(s);}**. The C statements will execute once for every time through the loop as long as the test condition is true.

Example:
Count from 0 to 9 with a loop and send the value to the LEDs on PORT A of the Z8. Assume the port has been initialized and a variable called **count** has been declared as type **char**.

The loop can be described by the following flowchart.



while() loop code:

```
count=0;
while(count<10)
{
      PAOUT  =  ~count;
      count++;
}
```

for() loop code:

```
for(count=0; count<10; count++)
{
      PAOUT = ~count;
}
```

Since the for loop only executes one statement, the curly brackets can be eliminated:

```
for(count=0; count<10; count++) PAOUT = ~count;
```

A for loop without additional statements can also be used as a rough delay function, or "do nothing" loop. The loop will cause the processor to wait for a defined number of loop cycles until the for is finished its increment cycle.

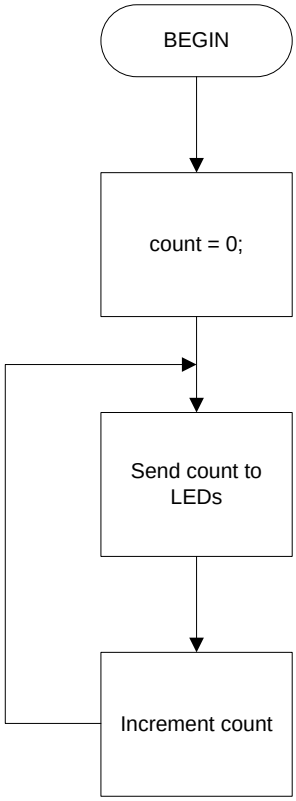
```
for(delay=0; delay<65535; delay++); //delay for a 16-bit count
```

The above statement requires **delay** to be an **unsigned int**. Why?

Perpetual (forever) Loop

If we wish to repeatedly execute one or more C statements forever, we can place the statements inside a while loop that never ends. Such a loop has a test condition that is always true. We can write this as:

```
while(1)
{
      statement(s);
}
```



For example:

- Initialize a count variable
- Send the count value to the LEDs
- Increment the count
- Repeat forever

Part of a C program for the Z8 to do this would be:

```
char count = 0;

while(1)
{
      PAOUT = ~count;
      count++;
} //end while
```

Programming assignment (lab3_2 and lab3_3):

lab3_2: Write a program that does the following:

- Declare a variable of type **unsigned long int** called **i** and initialize it to zero.
- Declare a variable of type **char** and initialize it to zero.
- Configure Z8 Port A to drive three LEDs on PA[2..0].
- loop forever:
    - invert all bits of the char variable, then write it to the LEDs (why invert?)
    - if the char variable is less than or equal to 4, increment the variable
    - otherwise set it to zero
    - The following statement can be used for a delay. Insert it inside the while loop as the last statement in the loop:
        - for(i=0;i<250000;i++);

Instructor's Initials _____

lab3_3: Write a program that starts with an LED output representing the binary value 101 as ON OFF ON, counts down to 000 (OFF OFF OFF), then repeats forever. Use the same for() statement for a delay as you did in lab3_2.c

Instructor's Initials _____