Purpose: We will take a look at programming this week using a language called Scratch. Scratch is a programming language that was developed at Michigan Institute of Technology to teach programming to people of all ages.

“Scratch is developed by the Lifelong Kindergarten Group at the MIT Media Lab, with financial support from the National Science Foundation, Microsoft, Intel Foundation, MacArthur Foundation, Google, Iomega and MIT Media Lab research consortia.”

https://scratch.mit.edu/about/

Specific concepts we are going to examine throughout this lab include:

- Programs and Algorithms
- Input and Output
- Variables and Data Structures
- Expressions
- Loops
- Conditional statements

Part 0: Programs and Algorithms

Programs are what make computers useful. A program is a finite sequence of instructions written in a particular language (called a programming language) that achieves some task. We’ve seen many examples of programs like Windows, Mac OS X, email clients, web browsers, video games, text editors, and many others. Programs usually achieve one or multiple tasks. For example, an email client program will log you into an email server, send and receive email, and offer many other services like organizing your emails.
Before we can dive into programming, we have to take a step back and define an important concept in computer science called an algorithm. An algorithm is a finite sequence of steps that achieves some task. An algorithm is independent of any programming language, and it can be thought of as “a way to solve a problem” or “a set of directions to perform some task”. The difference between an algorithm and a program can be subtle sometimes, but remember this: an algorithm is just a general solution to a problem or way to achieve a task whereas a program is a solution written in certain programming language that solves a problem or achieves some task.

In this lab, we will write a program using the Scratch programming language that will convert an integer (between 0 and 128) to a binary number. Thus, by the end of the lab you will become a programmer, a person who translates an algorithm into a program that executes on a computer.

**Part 1: The algorithm for converting an integer to a binary number**

Before we write a program to translate an integer to a binary number, we need to recall the algorithm for converting an integer to a binary number. We can't program a computer how to do this conversion unless we know how to do it ourselves.

Here are the steps for an algorithm that converts a nonnegative integer to a binary number:

1. Divide the input integer by 2 (call this value the quotient)
   a. Record the remainder of the quotient
   b. If the quotient is greater than 0, then repeat step 1 using the quotient as the input integer
   c. If the quotient is equal to 0, go to step 2.
2. Read the recorded remainders backwards to get the binary number

Here's an example using the algorithm above to convert 25 to binary:

Step 1:

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>25/2</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>12/2</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>6/2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3/2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(Note: 12 is the quotient here, and it will be used as input to the next step)

(Note: our quotient is 0 here, so we can go to step 2)

Step 2:

Read the remainders backwards (from bottom to top) to get the binary number 11001.

Notice how we used each step in the algorithm to convert 25 to the binary number 11001. Also, notice how we repeated step 1 a total of 5 times before we moved on to step 2.

**Part 2a: Scratch's Website**
From the labs page, download the Scratch file `integerToBinary.sb` to your I: drive. Go to [https://scratch.mit.edu/projects/editor/](https://scratch.mit.edu/projects/editor/) to upload the Scratch file into web Scratch IDE (note: if the IDE does not load in your web browser, then try to use another web browser). IDE stands for integrated development environment, and it is a place where a programmer can program, compile, debug, test, and run their program all in one.

![IDE screenshot](image.png)

Once you upload the Scratch file and click **Control**, in the top right of the IDE, you should see one window split into multiple sections in the web Scratch IDE like the image shown below.

![IDE screenshot](image.png)

The section in the top right of the Scratch IDE is used to pick which programming construct category you want to use like **Control, Operators, Data**, etc, the left section below that shows the programming constructs for the selected category (in pic above, **Control** programming constructs are shown), the middle/right section is for scripts or the program that we will be writing, the top left section is the stage or the part of the program where the user will interact with, and the bottom left section is for sprite animations (we will ignore this window since we won’t be using sprites or animations in the lab).

We have already defined a stage in the top left section, and it gives directions on how a user is supposed to use the program; however, the program itself is not written yet since the scripts window is blank.
If you click the green flag (what starts the program), nothing will happen at this point. We will be building a script or a program that will do the conversion. In this lab we will use the term script and program to mean the same thing.

**Part 2b: Inputs and outputs**

When writing a program, we need to think about the input and outputs. In the Scratch IDE, we have two inputs for our program. The first input is will be the integer that the user inputs by using the slider for input, and the second input is the green flag, that the user will click to start the program. The output will be a string of binary digits (0’s or 1’s) in the box near the bottom of the stage.

**Part 3a: Declaring and initializing variables and data structures**

We have inputs and outputs for a program, but we also need something to store values. Variables, yes the same variables from Math, are what programs use to store values. We give variables names, initialize them, and change their values as the program progresses. A variable stores a value in the computer’s memory, and a program can access or change the variable’s value by accessing its name. A variable holds only one value, but when we want something to hold multiple values in memory, we use a data structure. In Scratch, there is a simple data structure called a list, and think of it as storing a list of variables or values as the program runs. We will need this list to store the binaryDigits or remainders from step 1 of the algorithm.

![Image of Scratch IDE](image)

From the data category, you can see that the file already contains variables for input, output, quotient, and a list data structure called binaryDigits. The variables should look like the picture to the left. If they don’t, then just use the Make a variable or Make a list buttons to create them. Note: although we already provided you with the necessary variables and lists, you can easily make or delete variables or lists here which may be quite useful to do for a lab quiz (hint hint 😊) or if you use Scratch to write your own program.

What we did here was declare the variables and data structure that our program is going to use. By declaring variables and a data structure, we are telling the program to associate each name with what they store in memory. We don’t care how the program keeps up with values in memory addresses since we can access these values by simply using the variable and data structure names.

Please be sure that variable names and the list name are spelled correctly and match the names in the staging area (upper left window in Scratch). Also, you can click the
checkbox next to the variable and list names to make them appear or disappear in the staging area.

Declaring the variables is a start, but we also need to **initialize** the variables. **Initializing** variables involves **assigning** starting value to variables so each time the program runs, the values stored in memory will be reset to their starting values. **Assigning a variable** means setting it to some value.

So let's initialize the variables by adding set statements to the script. To add something to the script, all you need to do is to drag the programming component in the right hand window over to the scripts window. First, click on the Events category, and drag the **when [flag] clicked** over to the scripts windows to the right. This tells the program to start whenever a person clicks the green flag. Then, click on the Data category, and drag over the **set input to 0** (Note: you have to click on the tab to change it from quotient to input) and connect it to the first piece of the program to get a program with two steps.

Since we connected the pieces, whenever someone clicks the green flag, the connected piece will run in order from top to bottom. What we've done so far is just to initialize our input to 0, which isn't what we want to do since users should be able to input any integer between 0 and 128. So click on the drop down menu by the word input, and change it to quotient. We need to initialize quotient to the input the user selects. So drag from the variables category into the space where the 0 is occupying in **set input to 0**. It should look like when you are finished. Also, go ahead and initialize output to a blank box, and add another statement to make sure the list is empty when we start the program. This is can be done by dragging programming pieces to get the following program. Thus, we initialized all of our variables and our list. Note: You can right click on a piece in the scripts window to get help, delete, or duplicate a line or section of Scratch code.
Save your work, by clicking download to your computer.

Part 3b: Expressions

We learned in class that a CPU can handle many types of mathematical and Boolean expressions. An expression can be something like “1 + 1”, “3 / 2”, “A or B”, etc. An expression can be thought of as combination of operands and operators that evaluates to some value. For example, in the expression “1 + 3”, the operands are 1 and 3, the operator is +, and we evaluate the expression “1 + 3” to get 4. Expressions can become complex like “1 + 3 + 4 + 1 - 2”, but computers have no problem dealing with this; they just do one evaluation at a time, store the values each time in an accumulator until they reach the end of the expression, and then return the final value of the expression.

Now back to the program. To get an expression, click on the Operators category, and drag over an operator you want to use. We will need to use the mod operator; so drag it over to the scripts area. The mod operator is short for modulus operator, and it is mathematical operator that returns the remainder of a number divided by another number. Go ahead and fill in the mod operator with \( \text{quotient mod 2} \). What this does is evaluates the remainder of quotient/2, which is part of the algorithm for converting an integer to binary. Now we need to store this expression in the \text{binaryDigits} list, since we will need it later. Add \( \text{add thing to binaryDigits} \) to the program. But the thing we are adding is the expression. To do this, make sure \( \text{quotient mod 2} \) and \( \text{add thing to binaryDigits} \) are both in the scripts window, and drag the \( \text{add quotient mod 2 to binaryDigits} \) over the \( \text{thing} \), and let go. This creates the statement \( \text{add quotient mod 2 to binaryDigits} \), which will store the remainder of quotient/2 in the \text{binaryDigits} list.

Notice that the shapes of programming constructs in Scratch are setup to match places where syntax would be valid. So if a shape doesn’t fit into a programming construct, then those programming constructs are not syntactically compatible.
Note, the rest of this lab will have you create expressions and constructs by dragging and dropping programming constructs just as you did, and you can create complex statements by adding numerous variables, expressions, and other programming constructs together. Think of Scratch as being Lego blocks used to build a program, where each block is a programming construct. Your job will be to find the programming blocks to create a statement, a line of a program’s code, and understand why building each statement achieves a small step to converting an integer to a binary number.

As a recap, your program should now look like .

Download another copy to your I: drive so you can save your work

Part 3c: Loops

Many algorithms have parts that need to repeat a certain number of times, and in programming, we use constructs called loops to perform repeated calculations. Loops will execute lines of codes a certain number of times until a condition is met. We have to be careful to program loops to stop; otherwise, the program will go into an infinite loop, a loop that never stops on its own. When a program is stuck in an infinite loop it won’t stop until the operating system or something external shuts it down. There are many types of loops in programming languages and in Scratch, but we will stick to the repeat looping construct. To add a repeat loop, click on the Control category, and drag over a repeat until to the scripts window. Notice that this loop will repeat until a Boolean expression is evaluated to be true. This Boolean expression is the condition that stops the loop. From our algorithm, we want to stop step 1 of the algorithm when the quotient is equal to 0. So we need to construct the expression quotient is equal to 0 by dragging the appropriate operators and variables over to the script window to get Now we can drag to get Now any lines of code we put inside the repeat loop will get executed until the quotient is 0, and after the loop is finished, the code connected to the bottom of it will execute. Connect this loop with the rest of your program, and it should look like
Note that this repeat loop should contain the information needed for step 1 in the algorithm to convert an integer to binary. So we will need to set the value for quotient and store the remainder in `binaryDigits`. Let’s add and to the scripts window by creating these complex statements consisting of dragging over operators, variables, and programming constructs. Put both of those statements inside the repeat loop to get , and this instructs the program to set the quotient to whatever it was before divided by 2 and to add the remainder to `binaryDigits`. Notice that order matters here, we need to set the quotient first and then add the remainder to the `binaryDigits`. There are other ways of doing this, but this order is needed since we already initialized quotient before the loop started. Now your program should look like . Add two wait blocks before the loop and at the end of loop inside , so we can see what’s happening step by step.
Input any **odd integer** in the staging area (upper right window), click the green flag, and watch the program run.

**Download another copy to your I: drive so you can save your work**

YIKES!!! Something is going wrong. Our loop may never terminate, so click the **stop** button to stop the program. Notice that the quotient is changing from an integer to a decimal number with decimal points in it. This is a problem. Scratch’s variables change based on how they are assigned; so if an integer is assigned to a decimal number, then the variables becomes a decimal number, and quotient may never be equal to 0 to stop our loop. We want our quotient to always be an integer for our program to work. Our original algorithm is still correct, but to get our program to work in Scratch, we have to figure a workaround to this problem. Often programmers have to find workarounds when translating an algorithm to a specific programming language. In order to program a workaround, we will need a conditional statement.

**Part 3d: Conditional Statements**

When we look at what is going wrong with our program, we notice that if quotient is odd and we divide by 2, then we get a decimal back, which is what we don’t want, but if the quotient is even and we divide by 2, then we get an integer back since the even numbers divided by 2 are always integers. So we would like to handle both of these conditions in our program. To handle multiple conditions in a program we use **conditional statements**.

Statements in the form **if <Boolean Expression>, else <Boolean Expression>** are called **conditional statements**. **Conditional statements** are fundamental programming constructs because they allow a program to do different things based on certain input. We
will need to use an if-else conditional statement inside our loop to account for the cases when the quotient is odd or even. First put the lines of code inside of the repeat loop to the side of it for now (we will use them later so don’t delete them), and drag over an

and put it inside the repeat loop to get

Now let’s add a Boolean expression to the if statement; what this expression means mathematically is that the quotient is odd (when you divide by 2 and you get a remainder of 1, the quotient has to be odd, otherwise it would be even). Now the “if” part represents what the program should do if quotient is odd, and the else part represents what the program should do if the quotient is even.

If the quotient is odd, we must do a fancy mathematical workaround to get Scratch to work. What we will do is round(quotient/2) – 1 to ensure we get an integer back. For example, if the quotient is 5, and we take 5/2 = 2.5. But round(2.5) = 3, since it rounds up. Now, we just subtract 1 off of this to get 2. So the integer portion of 5/2 is 2 remainder 1 which is what we want. Build the expression to handle this, and it should look like

This is actually three expressions in one statement (quotient/2, round, and subtract 1). Now add this expression to a new set quotient expression to get

and place it into the if statement to get

That’s the hard part of the workaround.

Let’s move on to the else statement, and remember that the else statement is when the quotient is even. So we just need to set the quotient to quotient/2 since there is no problem dividing an even number by 2. So place
else statement to get

Also, move the

after the else statement but inside the
repeat loop. We put this after the if-else statement because no matter which part of the conditional statement you take you will always need to record the remainder. Your

program should look like

Download another copy to your I: drive so you can save your work

Notice that the program isn’t completed, but you should see that the **binaryDigits** and quotient are working correctly. You will see that an extra **binaryDigit** is added at the end, but that is no big deal, we’ll handle that shortly. Run the program with 29 as an input, and you should see something like below (please note that you may need to slide the input slider to 128, and then back to 29 to get 29 to show up as an input; the slider is kind of quirky in Scratch).
Part 3e: Output by reading the binaryDigits backwards

The final part of program is to read the binaryDigits (the remainders) backwards. We’ve been storing them so far, now we need to use them in our output, which is a binary string. We will need another repeat loop for go through the binaryDigits list, so go ahead and attach a repeat until to the bottom of your program. What we want to do is read the binaryDigits list backwards, so we want to look at the last thing in the binaryDigits first, add it to the output, then remove the last item, and repeat. Since we have an extra item in the binaryDigits list from the previous loop we can start by adding a to our second repeat loop to get

This removes the last entry in the binaryDigits list. Now, we want to build a complex expression to take the last item out of binaryDigits and add it to the output. For instance if output is 101 and the last binaryDigit is 0, this join operator will return 1010. It is joining the the two operands together. Place this expression inside a set output expression to get

This will set the output variable
to whatever it was joined with the last digit. Place this expression in the second repeat loop to get We left out one thing, the condition that will stop the loop. If we keep removing the last digit of binaryDigits and adding it to the output, then it makes sense to stop when the binaryDigits list is empty. An empty list has a length of 0 in the Scratch programming language. Create the expression, and add it to the condition of the repeat loop to get Add another wait block at the end of the loop.

Download another copy to your I: drive so you can save your work
We are almost finished. Attach a to the end of your program. Your program should look like . Download another copy to your I: drive so you can save your work

**Part 4: Testing your program**

At this point, we think the program is complete, but we need to test it with various inputs to verify that it is working correctly for all inputted integers from 0 to 128. First, save your work to your I: drive. After saving, make sure to test your program by inputting lots of integers to verify your output is the correct binary number. You can verify your program’s output by hand or go online and search for an integer to binary converter program. You can also go back to the edit menu and change the single stepping speed to make your program run faster for testing or run slower when trying to go through it step-by-step.

What happens when you input a 0 to the program? Are you seeing 0 in the output when you input a 0? With the program we’ve made so far, you shouldn’t see anything in the output when a 0 is inputted, and this is incorrect. This is called a software bug. Our program is supposed to work for all inputted integers from 0 to 128, but it seems to working fine for all inputs except 0. Step through the program slowly to see what is
happening when you input a 0. As a good programmer, we need to fix our program’s bug. An easy fix is to add a special case to our program to handle when 0 is inputted, and we can do this with a conditional statement. Take a look at the logic of the conditional statement, and think about what it is doing in English. You also need to think about a good place to put this conditional statement in your program, and put it into your program. Now, try out your code again for an input of 0 and various other inputs to verify that is working. If your program isn’t working, try to insert the conditional statement somewhere else in your program. Save your final program to your I: drive when you are finished.

Compare the program written in Scratch to the algorithm for converting an integer to a binary number, and notice how the Scratch program needed certain workarounds in order to accomplish the same task as the algorithm. Depending on the programming language, the algorithm may be implemented with less workarounds. The important take home point of this lab is to understand how programmers take an algorithm and break it down into many small steps in a programming language. Programming is not an easy discipline, and we’ve merely “scratched” the surface in this lab.

If everything works fine, then you have finished the lab. Congratulations you are now a programmer!!!

**Download another copy to your I: drive so you can save your work**