E11 - Autonomous Vehicles
Arduino Programming IV
Given the following 4 binary numbers

00010, 00110, 01001, 00111

1. For each pair of numbers, find their dot product.
2. Output the sum the dot products in the format: “The sum of the dot products is S”
### Let's get started

```c
#define numBinNumbers 4
#define numBitsPerNumber 5

boolean binNumbers[numBinNumbers][numBitsPerNumber] =
  {{0,0,0,0,1}, {0,0,1,1,0}, {0,1,0,0,1}, {0,0,1,1,1}};

void setup()
{
  Serial.begin(9600);
}
```
Outline

- Goal
- Repeated function calls
- Function Calls with Arrays
- Timing
- Operators
Functions & Function Calls

- One or many calls to a function, e.g.

```cpp
void loop()
{
    sayHello();
}
```

- A function definition, e.g.

```cpp
void sayHello()
{
    Serial.println("Hello");
}
```
Repeated Function Calls

- What functions should we add?

```c
int getDotProduct();
```
Repeated Function Calls

- What functions should we add?

```c
int getDotProduct( ? )
```
void setup()
{
    Serial.begin(9600);
    int sum = 0;

    sum = dotProduct(binNumbers[0], binNumbers[1]);

    Serial.print("The sum of dot products is ");
    Serial.println(sum);
}
void setup()
{
  Serial.begin(9600);
  int sum = 0;

  sum = dotProduct(binNumbers[0], binNumbers[1]);
  sum += dotProduct(binNumbers[1], binNumbers[2]);
  sum += dotProduct(binNumbers[2], binNumbers[3]);
  ...

  Serial.print("The sum of dot products is ");
  Serial.println(sum);
}
void setup()
{
    Serial.begin(9600);
    int sum =0;
    for (int i=0; i<numBinNumbers; i++)
    {
        for (int j=0; j<numBinNumbers; j++)
            sum = dotProduct(binNumbers[i], binNumbers[j]);
    }
    Serial.print("The sum of dot products is ");
    Serial.println(sum);
}
void setup()
{
    Serial.begin(9600);
    int sum =0;
    for (int i=0; i<numBinNumbers; i++)
    {
        for (int j=i+1; j<numBinNumbers; j++)
            sum += dotProduct(binNumbers[i], binNumbers[j]);
    }

    Serial.print("The sum of dot products is ");
    Serial.println(sum);
}
Outline

- Goal
- Repeated function calls
- Function Calls with Arrays
- Timing
- Operators
Let's make a function called `dotProduct()`

```c
int dotProduct()
{
    int dp=0;
    return dp;
}
```
Function Calls with Arrays

- Lets make a function called dotProduct()

```java
int dotProduct(boolean n1[numBitsPerNumber],
 boolean n3[numBitsPerNumber])
{
    int dp=0;
    for (int i=0; i<numBitsPerNumber; i++)
    {
        if (n1[i] == n3[i])
            dp = dp + 1;
        else
            dp = dp - 1;
    }
    return dp;
}
```
Let's make a function called dotProduct()

```c
int dotProduct(boolean *n1, boolean *n2)
{
    int dp=0;
    for (int i=0; i<numBitsPerNumber; i++)
    {
        if (n1[i] == n2[i])
            dp = dp + 1;
        else
            dp = dp - 1;
    }
    return dp;
}
```
## Function Calls with Arrays

- Let's make a function called `dotProduct()`

```c
int dotProduct(boolean *n1, boolean *n2) {
    int dp = 0;
    for (int i = 0; i < numBitsPerNumber; i++) {
        if (n1[i] == n2[i])
            dp = dp + 1;
        else
            dp = dp - 1;
    }
    return dp;
}
```

The `*` indicates the function will take a **pointer** to an array.
More on pointers later, but

- The pointer variable stores the address of the memory where the value is stored.
- The value stored there can be accessed.
Outline

- Goal
- Repeated function calls
- Function Calls with Arrays
- Timing
- Operators
We have used:

- The $\text{delay}(t)$ function
  - Delays for $t$ ms before continuing

- The $\text{delayMicroseconds}(t)$ function
  - Delays for $t$ microseconds before continuing
To set up timers

- **The millis() function**
  - Returns the time since the program started as an unsigned long in units of milliseconds

- **The micros() function**
  - Returns the time since the program started as an unsigned long long in units of microseconds
Outline

- Goal
- Repeated function calls
- Function Calls with Arrays
- Timing
- Operators
Operators

- We can **operate** on variables in more ways than just arithmetic.
  - E.g. $x = 2*y$
- We can compare
- We can test for truth
- We can manipulate bits
- We can combine operations
Operators

- Arithmetic

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>y = a + 2;</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>y = a - 2;</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>y = x * 12;</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>z = x / 3;</td>
</tr>
<tr>
<td>%</td>
<td>modulo</td>
<td>z = 5 % 2;</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
<td>x = 22;</td>
</tr>
</tbody>
</table>
Operators

- Comparisons

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>==</td>
<td>equals</td>
<td>(y == 2)</td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td>not equals</td>
<td>(x != 7)</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>less than</td>
<td>(y &lt; 12)</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td>greater than</td>
<td>(val &gt; max)</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>less than or equal</td>
<td>(z &lt;= 2)</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>(y &gt;= 10)</td>
</tr>
</tbody>
</table>

- E.g.

```cpp
if (z >= 2)
    Serial.println("Bigger than or equal to 2");
```
Combining Comparisons with Boolean Logic

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>(x &amp;&amp; y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>NOT</td>
<td>!x</td>
</tr>
</tbody>
</table>

E.g.

\[
\text{if (z < 2 && z > 0)} \\
\text{Serial.println(“Less than 2 and greater than 0”);}\
\]
Bitwise Comparisons

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>bitwise AND</td>
<td>y = a &amp; 15;</td>
</tr>
<tr>
<td></td>
<td>bitwise OR</td>
<td>y = a</td>
</tr>
<tr>
<td>^</td>
<td>bitwise XOR</td>
<td>y = a ^ b;</td>
</tr>
<tr>
<td>~</td>
<td>bitwise NOT</td>
<td>z = ~x;</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>bitshift left</td>
<td>z = 4 &lt;&lt; 2;</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>bitshift right</td>
<td>x = x &gt;&gt; 8;</td>
</tr>
</tbody>
</table>

Examples:

```c
int a = 5; // binary: 0000000000000101
int b = a << 3; // binary: 0000000000101000, or 40 in decimal
int c = b >> 3; // binary: 0000000000000101, or back to 5
```
Operators

- **Compound Operations**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>increment</td>
<td>a++; // a = a+1</td>
</tr>
<tr>
<td>--</td>
<td>decrement</td>
<td>x--; // x = x - 1</td>
</tr>
<tr>
<td>+=</td>
<td>addition and assignment</td>
<td>y += 3; // y = y + 3</td>
</tr>
<tr>
<td>-=</td>
<td>subtraction and assignment</td>
<td>z -= 10; // z = z - 10</td>
</tr>
<tr>
<td>*=</td>
<td>multiplication and assignment</td>
<td>x *= 4; // x = x * 4</td>
</tr>
<tr>
<td>/=</td>
<td>division and assignment</td>
<td>y /= 10; // y = y / 10</td>
</tr>
<tr>
<td>&amp;=</td>
<td>bitwise AND and assignment</td>
<td>y &amp;= 15; // y = y &amp; 15</td>
</tr>
<tr>
<td></td>
<td>=</td>
<td>bitwise OR and assignment</td>
</tr>
</tbody>
</table>

- **E.g.**

```
i *= 10;
```
There are many other useful Math functions:

- \text{min}(a,b)
- \text{max}(a,b)
- abs()
- ...

Operators

- There are many other useful casting functions:
  - `int(a)`
  - `float(b)`
  - ...

- Example

  ```
  float a = 6.2;
  int b = int(a);
  ```
1. Determine the amount of time in seconds it takes to iterate through 2 for loops, one nested within another. Each loop should iterate from 0 to 4, such that each iteration results in the following output to the serial port:

   “The iteration number is X”

   In this case $X = i \times 5 + j$

2. Finally, the program should output

   “The total time in seconds is Y”