Write a sketch that once you enter ‘g’ for go it turns the LED on and off for the amount of time in an array. Have the program terminate when it reaches a negative value (as time is always positive).

Example:

```
short timeArray[] = {500, 400, 500, 400, 100, 200, 100, 200, -1};
```

This would have the LED on for 500 ms, off for 400 ms, on for 500 ms, off for 400 ms, on for 100 ms, off 200 for ms…

```
#define LED 13

short timeArray[] = {500, 400, 500, 400, 100, 200,
100, 200, -1};

void setup()
{
  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("Ready...");

  //setup LED pin
  pinMode(LED, OUTPUT);
}

void loop()
{
  unsigned char command;
  if (Serial.available() > 0)
  {
    command = Serial.read();
    int i = 0;
    int highLow = 1;

    if(command == 'g')
    {
      while(timeArray[i] >= 0)
      {
        digitalWrite(LED, highLow);
        delay(timeArray[i]);
        highLow = !highLow;
        i++;
      }
    }
  }
}
```

Here is one solution to the problem

First we setup the serial port, print that the program is running, then we setup the LED pin as an output

In the main loop we are more careful with the serial port and make sure there is something to read before trying to read it, then if it is ‘g’ we increment though the array until we reach a negative value flipping the value of the LED by negating highLow.

Write a sketch that allows you to blink both the red and the green LED for different amounts of time simultaneously.

The green LED is on pin 5.

Example:

```
short timeArrayR[] = {500, 400, 500, 400, 100, 200, 100, 200, -1};
short timeArrayG[] = {100, 200, 100, 300, 50, 50, 50, 50, 50, 50, -1};
```

```
#define RedLED 13
#define GreenLED 5
#define True 1
#define False 0

short timeArrayR[] = {500, 400, 500, 400, 100, 200,
100, 200, -1};
short timeArrayG[] = {100, 200, 100, 300, 50, 50,
50, 50, 50, 50, -1};


void setup()
{
  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("Ready...");

  //setup LED pins
  pinMode(RedLED, OUTPUT);
  pinMode(GreenLED, OUTPUT);
}

void loop()
{
  unsigned char command;
  if (Serial.available() > 0)
  {
    command = Serial.read();
    int i = 0;
    int j = 0;
    int notDone = True;
    int notDoneRed = True;
    int notDoneGreen = True;
    long currentTime = 0;
 if(command == 'g')
    {
      while(notDone)
      {

        notDoneRed = setLED(currentTime, timeArrayR, RedLED);
        notDoneGreen = setLED(currentTime, timeArrayG, GreenLED);
        notDone = notDoneGreen || notDoneRed;
        delay(1);
        currentTime++;
      }
    }
  }
}

int setLED(long currentTime, int edges[], int pin)
{
  int i = 0;
  long totalTime = 0;
  while(edges[i] > 0)
  {
    totalTime += edges[i];
    if(currentTime < totalTime)
    {
      if(i%2 == 0)digitalWrite(pin, HIGH);
      else digitalWrite(pin, LOW);
      return True;
    }
    i++;
  }

  return false;

}
```

```
//Sketch Multi
#define rows 6
#define columns 8

int myMatrix[rows][columns];

void setup()
{
  int i, j; // counters

  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  //initialize my array
  for(i = 0; i<rows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      myMatrix[i][j] = j*(1+i);
    }
  }

  //print array
  for(i = 0; i<rows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      Serial.print(myMatrix[i][j]);
      Serial.print("\t");
    }
    Serial.print("\n");
  }

}
void loop()
{

}
```

In your homework you will eventually meet not only arrays but multi-dimensional arrays (or arrays of arrays).

In this example we see a 2-D array which could be thought of as a matrix of numbers.

Though we can type in the numbers using the {} notation, (for a 2x2 matrix that would look like {{1,2}, {3,4}}) here we initialize programmatically with *nested* for loops that index over the elements in the arrays.

Then we print out the matrix using the tab character \t to space things out nicely.

Try it out, then modify the print function so that it prints the sum of each row rather than the individual values.

```
//Sketch Multi
#define rows 6
#define columns 8

int myMatrix[rows][columns];

void setup()
{
  int i, j; // counters

  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  //initialize my array
  for(i = 0; i<rows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      myMatrix[i][j] = j*(1+i);
    }
  }

  //print array
  for(i = 0; i<rows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      Serial.print(myMatrix[i][j]);
      Serial.print("\t");
    }
    Serial.print("\n");
  }

}
void loop()
{

}
```

In your homework you will eventually meet not only arrays but multi-dimensional arrays (or arrays of arrays).

In this example we see a 2-D array which could be thought of as a matrix of numbers.

Though we can type in the numbers using the {} notation, (for a 2x2 matrix that would look like {{1,2}, {3,4}}) here we initialize programmatically with *nested* for loops that index over the elements in the arrays.

Then we print out the matrix using the tab character \t to space things out nicely.

Try it out, then modify the print function so that it prints the sum of each row rather than the individual values.

```
//Sketch Multi
#define rows 6
#define columns 8

int myMatrix[rows][columns];

void setup()
{
  int i, j; // counters
  int sumValue;

  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  //initialize my array
  for(i = 0; i<rows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      myMatrix[i][j] = j*(1+i);
    }
  }

  //print array
  for(i = 0; i<rows; i++)
  {
    sumValue = 0;
    for(j = 0; j<columns; j++)
    {
      sumValue = sumValue + myMatrix[i][j];
    }
    Serial.println(sumValue);
  }
}
void loop()
{

}
```

Here is one possible solution

This program is the whole algorithm in just the “main” (setup) function.

As your program grows this will become completely in practical. Its often better to think of sub-procedures (functions) to create functions for.

Good candidates for functions are things you have to do more than once that are *testable*.

In the code here the ideas that are distinct are initializing the matrix and printing it.

These should be functions.

```
#define rows1 2
#define rows2 10
#define columns 8

int matrix1[rows1][columns];
int matrix2[rows2][columns];

//function prototypes
void initalizeMatrix(int aMatrix[][columns],  int
numberOfRows);
void printMatrix(int aMatrix[][columns], int
numberOfRows);

void setup()
{
  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  initalizeMatrix(matrix1, rows1);
  initalizeMatrix(matrix2, rows2);

  printMatrix(matrix1, rows1);
  printMatrix(matrix2, rows2);

}

void loop()
{

}
```

Once we use functions we can write down our core idea first compactly.

Here we create two 2-D arrays as global variables, and our main routine just wants to initialize them and print them out.

```
#define rows1 2
#define rows2 10
#define columns 8

int matrix1[rows1][columns];
int matrix2[rows2][columns];

//function prototypes
void initalizeMatrix(int aMatrix[][columns],  int
numberOfRows);
void printMatrix(int aMatrix[][columns], int
numberOfRows);

void setup()
{
  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  initalizeMatrix(matrix1, rows1);
  initalizeMatrix(matrix2, rows2);

  printMatrix(matrix1, rows1);
  printMatrix(matrix2, rows2);

}

void loop()
{

}
```

Once we use functions we can write down our core idea first compactly.

Here we create two 2-D arrays as global variables, and our main routine just wants to initialize them and print them out.

To do this we specify how these functions will work via prototypes, they need a reference to the array, and how many rows it has. [In C you have to provide all the sizes of the array but the last one when passing the array to a function.]

Our function needs the number of rows to avoid going into memory we do not own.

With this done we can write the functions that accomplish these sub-tasks.

```
#define rows1 2
#define rows2 10
#define columns 8

int matrix1[rows1][columns];
int matrix2[rows2][columns];

//function prototypes
void initalizeMatrix(int aMatrix[][columns],  int
numberOfRows);
void printMatrix(int aMatrix[][columns], int
numberOfRows);

void setup()
{
  //setup the serial terminal
  Serial.begin(9600);
  Serial.println("I'm alive!");

  initalizeMatrix(matrix1, rows1);
  initalizeMatrix(matrix2, rows2);

  printMatrix(matrix1, rows1);
  printMatrix(matrix2, rows2);

}

void loop()
{

}
```

The code is very similar to what was in main but now reusable; try it out

```
// C requires the size of all the indexes of multidimensional arrays
at compile time
// Thus the int aMatrix[][columns]

void initalizeMatrix(int aMatrix[][columns],  int numberOfRows)
{
  int i, j; // counters
  //initialize my array
  for(i = 0; i<numberOfRows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      aMatrix[i][j] = j*(1+i);
    }
  }
}


void printMatrix(int aMatrix[][columns], int numberOfRows)
{
  int i, j; // counters

  //print array
  for(i = 0; i<numberOfRows; i++)
  {
    for(j = 0; j<columns; j++)
    {
      Serial.print(aMatrix[i][j]);
      Serial.print("\t");
    }
    Serial.print("\n");
  }
  Serial.print("\n");
}
```