**Benchmarking a Program**

Benchmarking is typically used to mimic a type of workload on an architecture and evaluate the performance.

Broadly classified into two categories- SYNTHETIC BENCHMARKING AND APPLICATION BENCHMARKING

**Synthetic benchmarking**:Aimed at testing the performance of a specific component

**Application benchmarking**: Run real world programs on the systems

**Reference: https://upcommons.upc.edu/bitstream/handle/2099.1/4483/CAUBET\_Josep\_MasterOfScienceThesis\_EPSC\_KTH.pdf**

**Popular benchmarking techniques:**

**Whetstone Benchmark**: The output is the machine’s speed in kilo Whetstones per second

**Dhrystone Benchmark**: Contains no floating point operations. Output is the machine’s speed in kilo Dhrystones per second

**NOTE**:

1. While benchmarking different architectures , the clocks must be synchronized.

2. Location of code is a prime factor-fetching the code from cache memory is easier and faster than fetching it from RAM or ROM.

**Benchmarking Parameters:**

1. Processing Speed
2. Temperature of the Microprocessor
3. Power Consumption
4. Storage used up

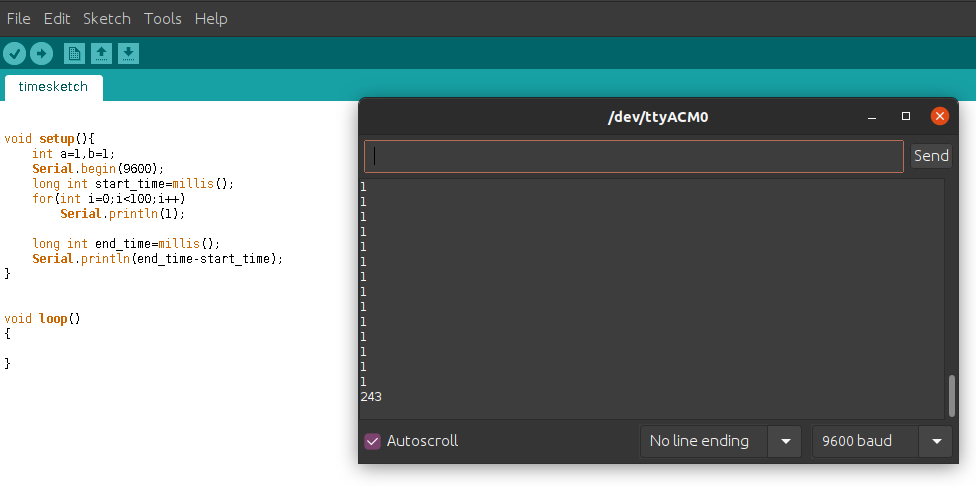
**Execution Time**: Each execution algorithm is run for number of loops in order to make up the execution uptil milli seconds( especially on arduino type microcontrollers

**Some Experiments on benchmarking :the time taken by arduino Mega2560:**

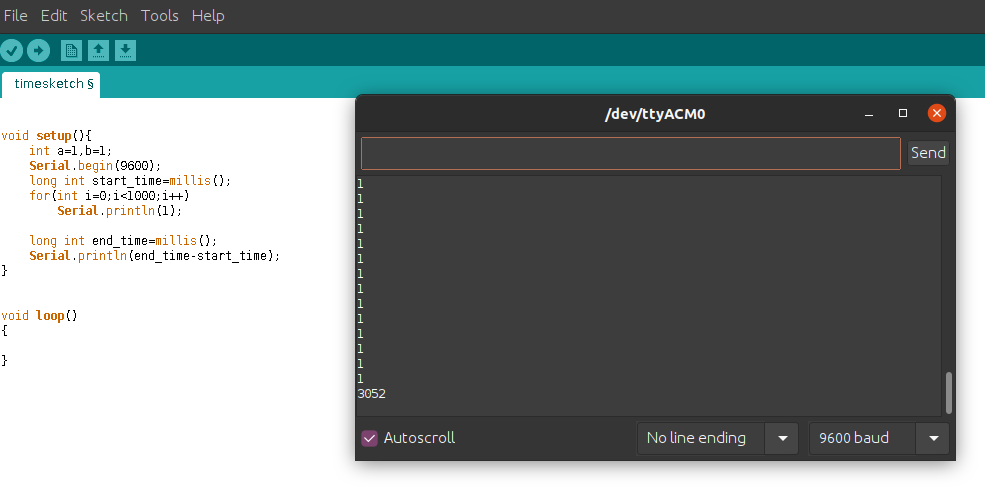
We use millis() and micros() function here to identify the time lapsed while executing the code snippet

Resolution: millis()- 1 millisecond::macros()-4 microseconds

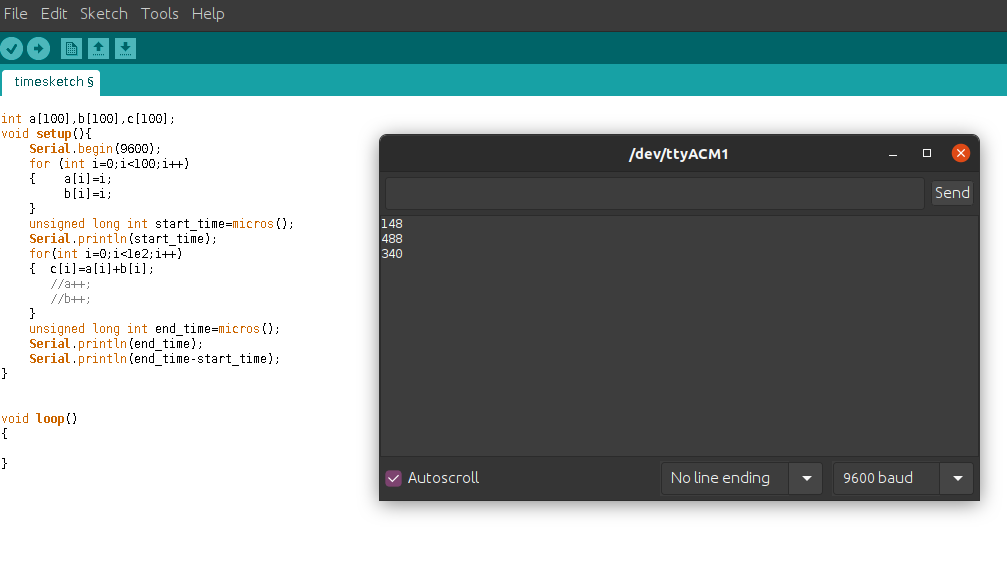
Running a loop to print a number 100 times: 243 milliseconds



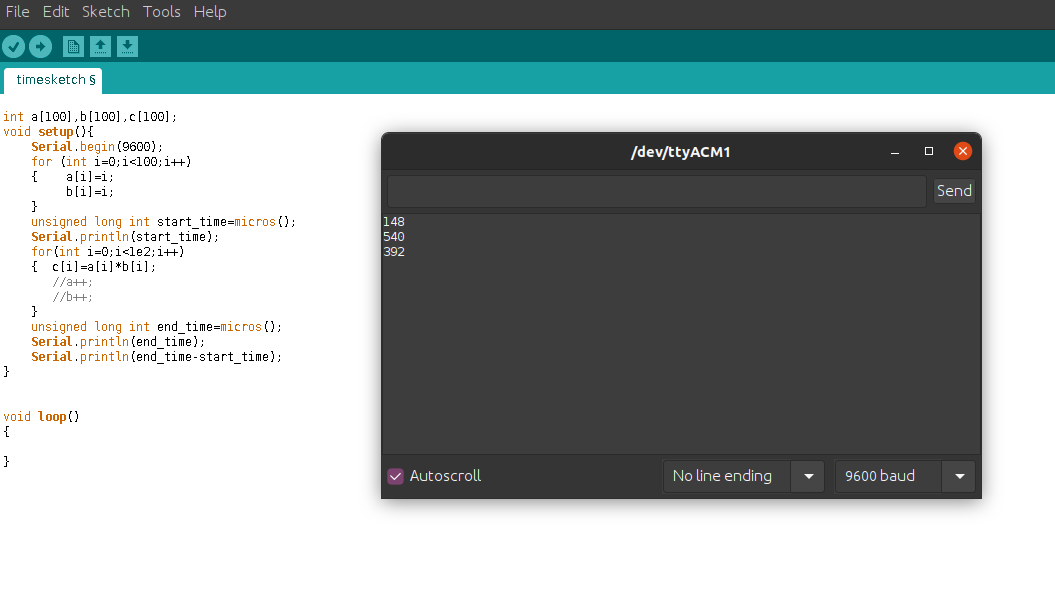
Running a loop to print a number 1000 times: 3052 milliseconds



**Adding two arrays of 100 elements each**: 340 microseconds



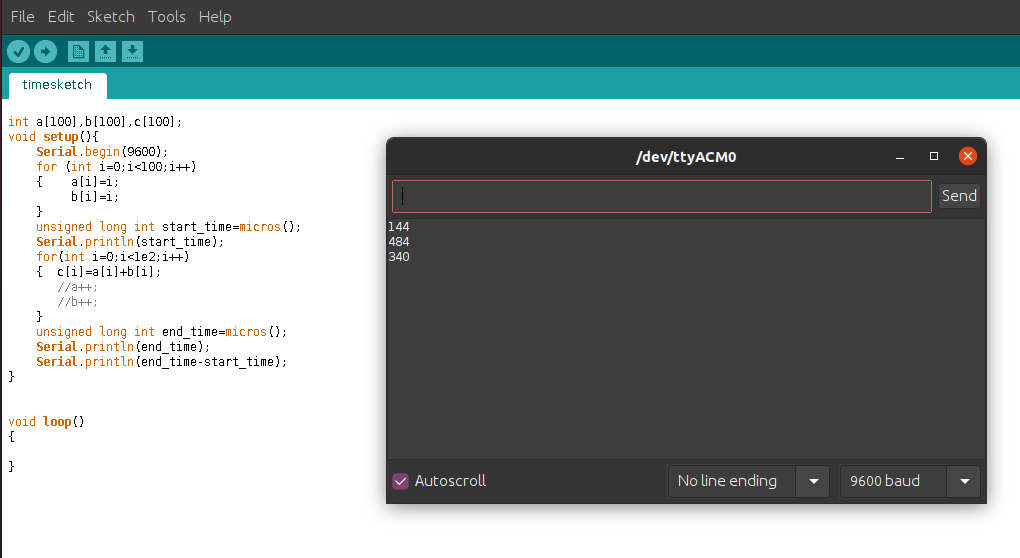
**Multiplying two arrays of 100 elements each**: 392 microseconds



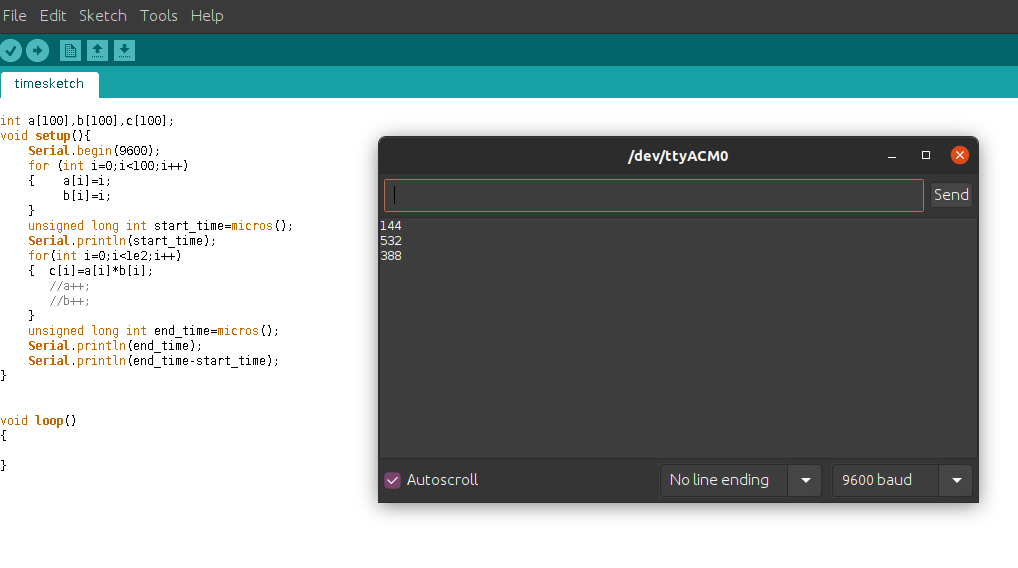
**INTERESTINGLY**: If initialisation of the variable was done inside the loop- compiler optimizes and tries to take only the initialisation during the last iteration(compiler optimization) .

**Running the same programs on Arduino UNO:**

**Adding two arrays of 100 elements each:** 340 microseconds

****

**Multiplying two arrays of 100 elements each:** 388 microseconds



**RESULT OF OUR SAMPLE BENCHMARKING**:

**ARDUINO UNO SEEMS TO BE FASTER THAN ARDUINO MEGA IN MULTIPLICATION**

**BENCHMARKING PROGRAMS:**

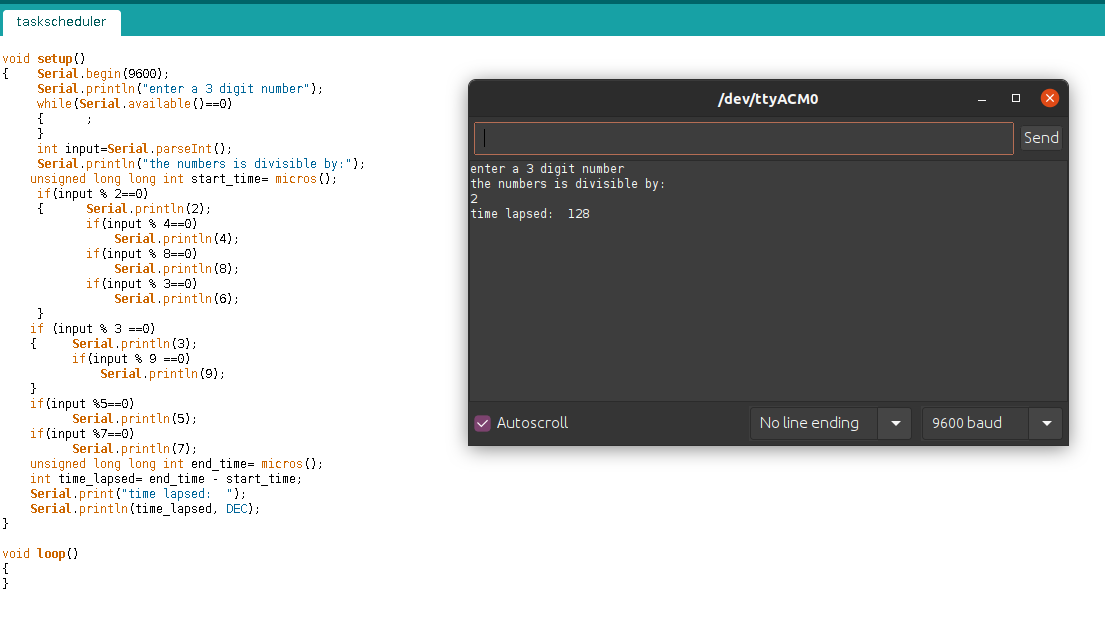
Let’s benchmark the time taken by two programs run on ARDUINO UNO

**Functionality:** the program prints the numbers in range 2 to 9 that divide a given input number

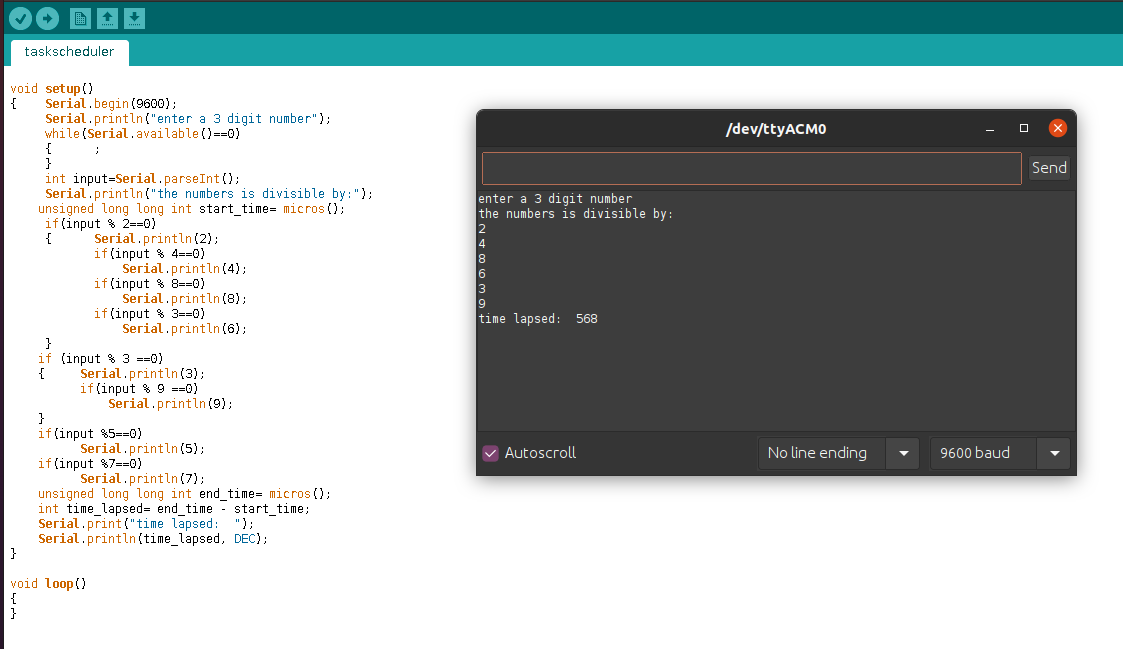
**Program 1: THE TASK SCHEDULER MODEL**

has a logical method checking using if statements ( note comparison is done 8 times[ 9-2+1= 8]

**Time taken by snippet- 128 microseconds : INPUT NUMBER:134**

****

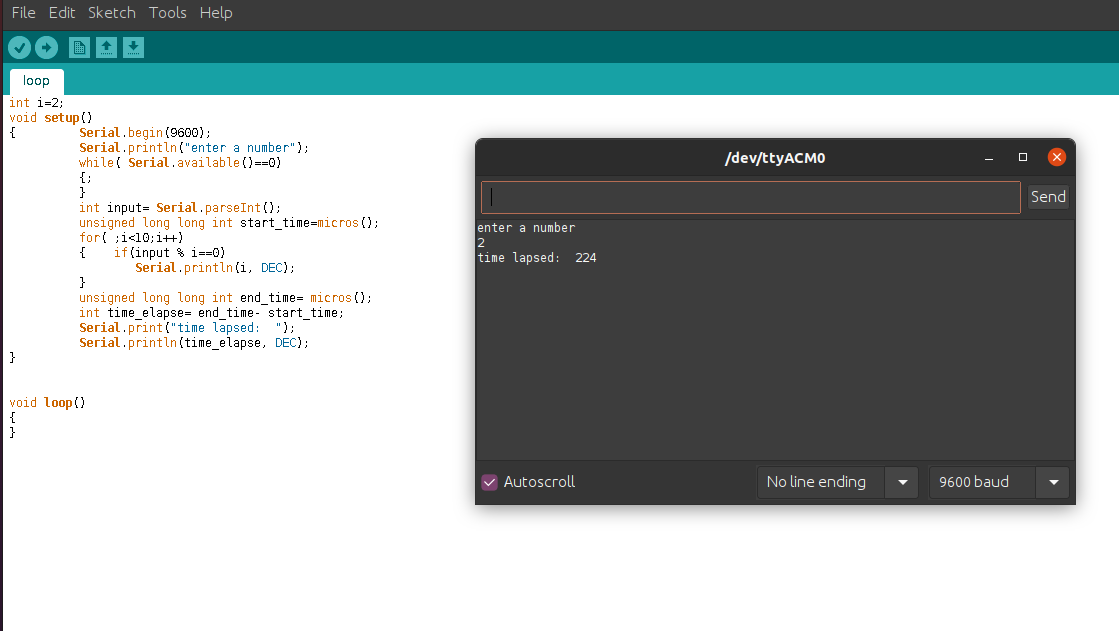
**Time taken by snippet- 568 microseconds- INPUT NUMBER -576**

****

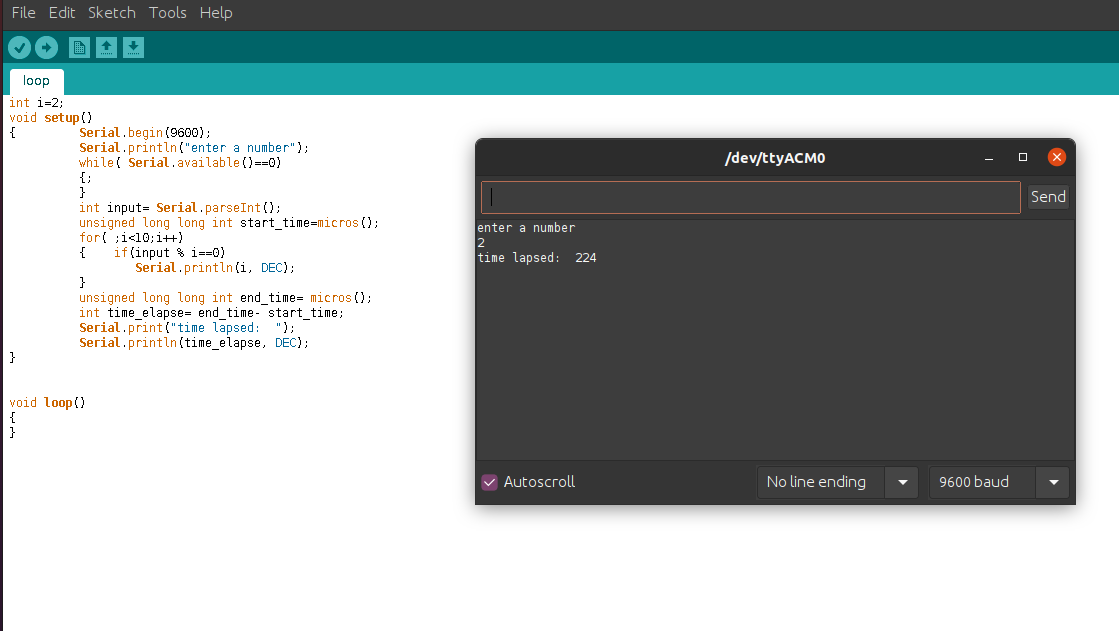
**Program 2: THE LOOP MODEL-**

A basic for loop is done from 2 to 9 [note: 8 times comparison is done]

**Time taken by snippet: 224 microseconds[ INPUT NUMBER 134]**

****

**Time taken by snippet: 636 microseconds [INPUT NUMBER 576]**

****

**RESULTS OF COMPARISON:**

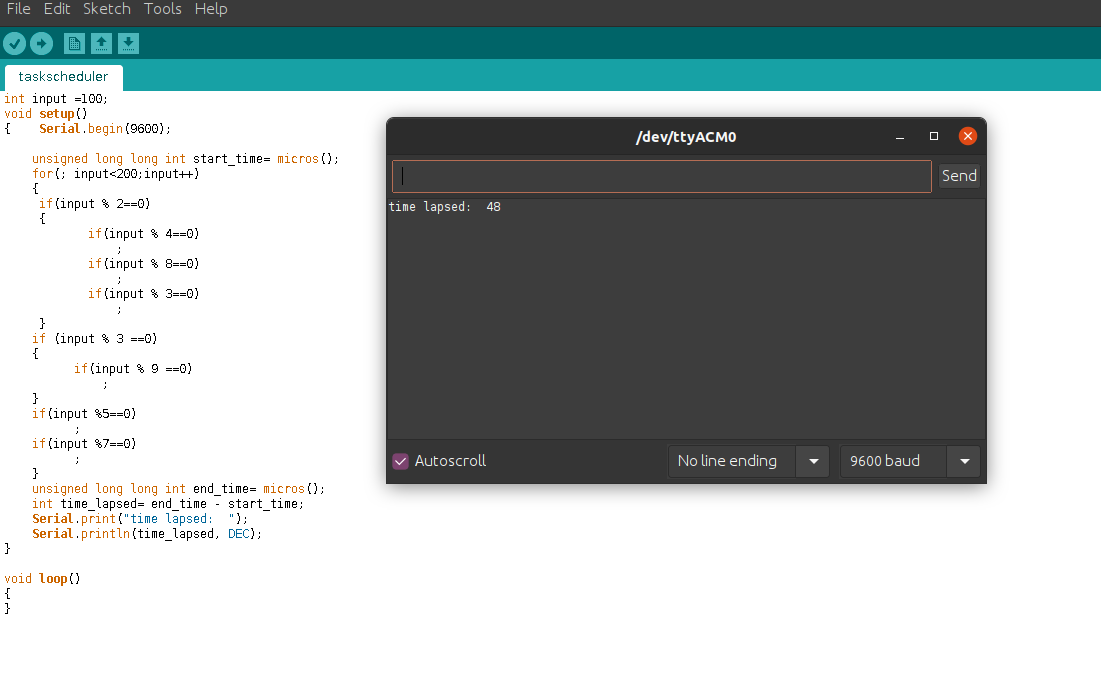
Time ratio=224/128= 1.75

Time ratio= 636/568=1.12

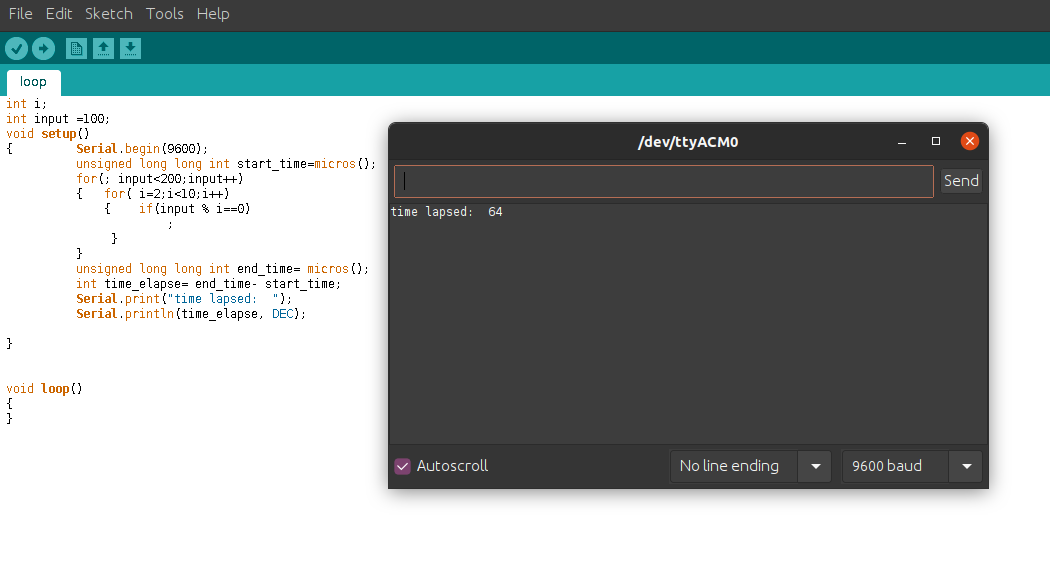
Let’s try to generalise it for 100 numbers in from 100-199

THIS TIME ONLY COMPARISONS ARE DONE, VALUES ARE NOT PRINTED

Output for TASKSCHEDULER MODEL= 48 microseconds



Output for LOOP MODEL- 64 microseconds

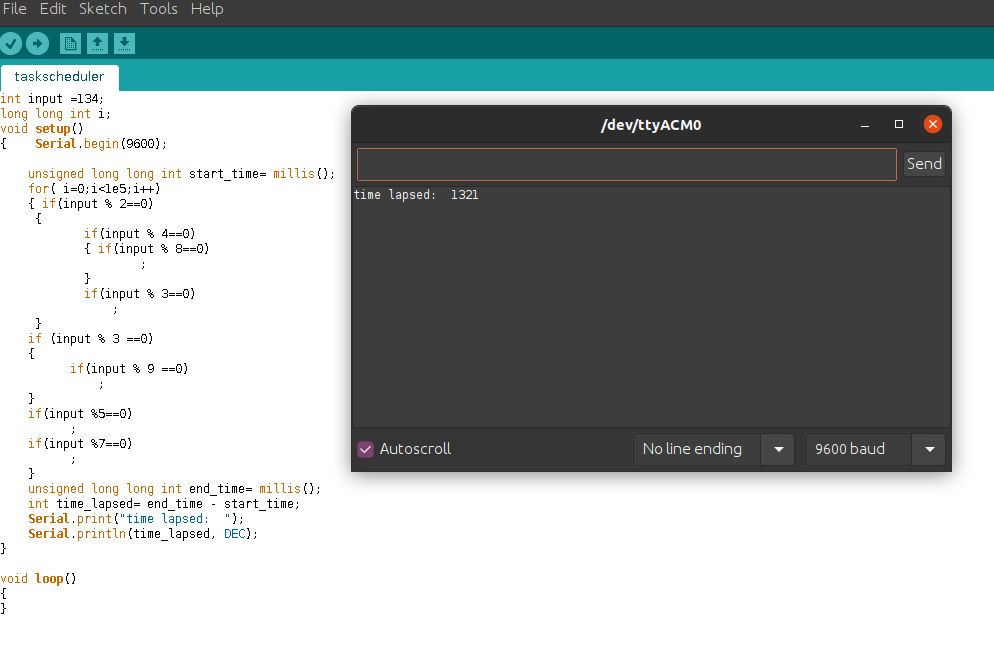


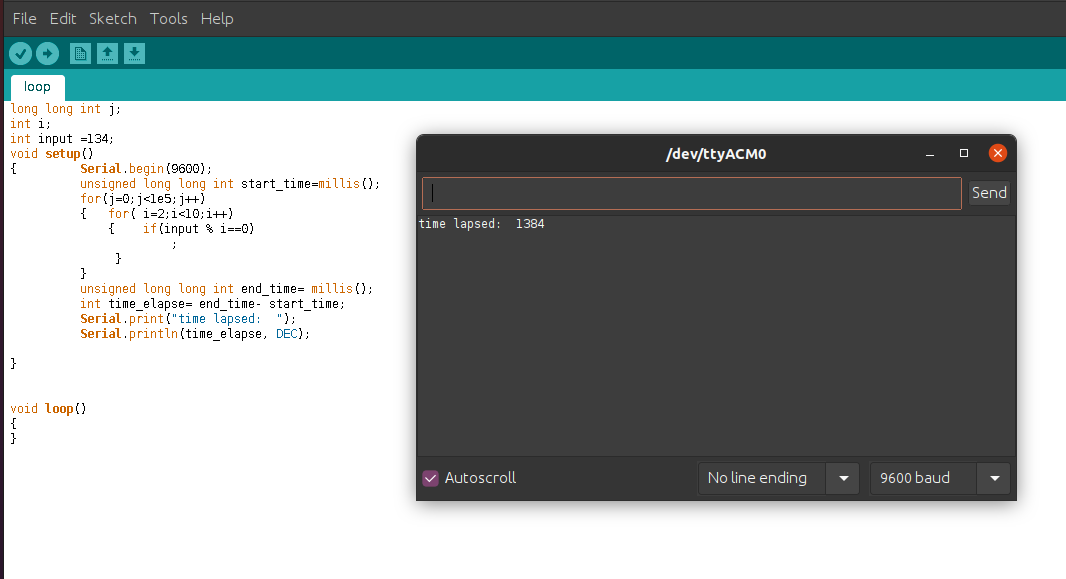
**CLEARLY- task scheduler model is 1.333 times more efficient than loop model on an average**

Let’s increase the number of iterations for the same input to get a better average of the time. The no. of iterations of the same is 100000(1e5)

We are using millis() instaed of micros() because time is too large for the datatype to hold.

**Task scheduler model: Time lapsed= 1321 milliseconds**

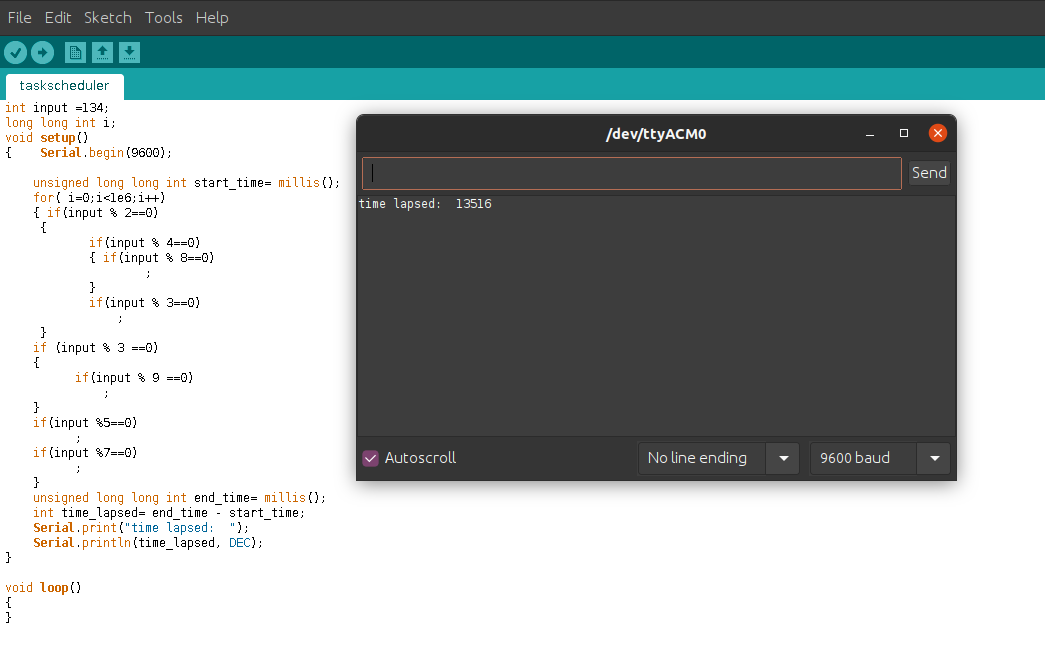
**Loop Model : Time lapsed=1384 milliseconds**

****

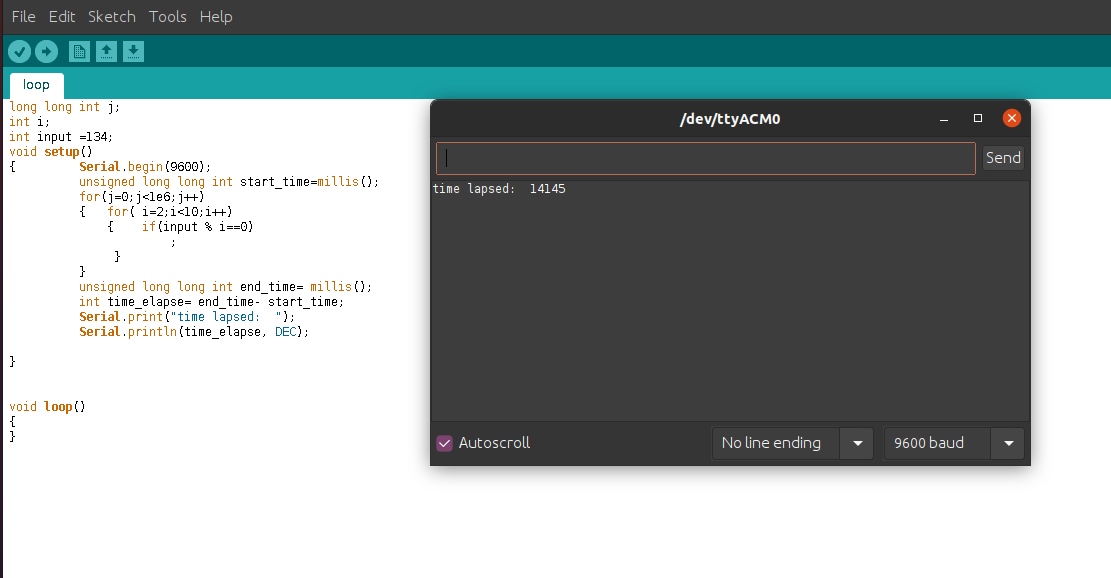
**Relative efficiency= 1384/1321=1.0477**

**Let’s increase the no.of iterations further to 1e6:**

**Task Scheduler model: 13516 milliseconds**

****

**Loop model:14145 milliseocnds**

****

**RELATIVE EFFCIENCY= 1.046**

**CONCLUSION:** Task scheduler model is more efficient than loop model by 1.046 times on an average