

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 05

Exercise 05

Compare C to Java with Doubles

# Name: Minakov Nikita

PURPOSE

The purpose of this exercise is to give you practice instrumenting a C program for performance measurement.

For this exercise, you will use both the C and Java implementations of the program that computes the square root of a number using doubles.

Both C and Java provide ways to measure performance in the form of system calls. The method to use in Java is System.nanoTime(). In C, you will use the clock() function. Your program will need to use these twice, once before the code you want to time, and once after. The difference between the two is the time it took to execute that portion of the program.

For information on the C function, refer to the C man page for the clock function by typing “man clock”. In addition to providing information about what the function does, this man page tells you that you need to include the header file for time.h (where clock() and relevant types and constants are defined) in your program by inserting the line “#include <time.h>” at the top of your program.

Note that you can also type “man 3 clock”. The 3 in the second command tells the man command to look in section 3 of the man pages, which contain C language functions, rather than shell commands and programs. It is sometimes necessary to tell the man program which section to refer to when there is a shell command and a C function that share the same name.

If you are using a virtual machine, be sure to obtain performance numbers from both programs running within the virtual machine, as the fact that you are using a virtual machine will impact performance.

ACTIVITIES

Perform each of the following activities. If you have questions, issues, or doubts, please ask for help and do not just guess.

1. Add code to measure the execution time *in milliseconds* of the part of the Java program that computes the square root of a number. Do not include program input and output in the code to be timed.
2. Add corresponding code to C program.
3. Run both programs and compare the execution time.
4. How long did it take to run each program (enter your answer in the space below)?
5. What conclusions can you draw about the performance of computations in Java versus in C based on your results (enter your answer in the space below)?
6. When you are ready to submit your work, first remove all intermediate files from your src directory. This includes exercise05.o and the exercise05 executable. You can use the “make” command to do this by typing “make clean” at the command prompt.
7. Save and archive your work, including this document and your finished C program (you do not need to submit the Java program), and upload it to the LMS.

#include <stdio.h>  
#include <math.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdbool.h>  
#include <ctype.h>  
#include <time.h>  
  
  
  
#define SQR2\_FILE "SquareRootTwo.txt"  
  
**char** \*read\_file(**const char** \*file\_name);  
  
/\*  
 \* Calculate the square root of 2, compare the result to a known value for the  
 \* square root of two, and display both.  
 \*  
 \* Parameters: n/a  
 \*  
 \* Returns:  
 \* 0 on success, else 1  
 \*/  
**int** main(**void**)  
{  
 // time calculating variables  
 clock\_t start\_t, end\_t;  
 **double** total\_t;  
  
 // Start point time  
 start\_t = clock();  
 printf("start\_t = %ld\n", start\_t);  
  
 **int** i;  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Student's code goes here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 // Put the code here that computes the square root of two. The result  
 // of the computation should be stored in the variable new\_guess.  
 **double** guess = 1;  
 **double** new\_guess = 0.0;  
 **const double** delta = 1E+10;  
 **for**(**int** i = 1; i < 20; i++){  
 new\_guess = 0.5\*(guess + 2/guess);  
 **if**(fabs(new\_guess - guess) > delta)  
 **break**;  
 **else** guess = new\_guess;  
 }  
  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Student's code goes here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
 // Display the calculated square root  
 printf("\nThe square root of two is: %lf\n", new\_guess);  
  
 // Read in the square root that NASA has computed from a file  
 // read\_file allocates memory for check\_number, so we will  
 // have to free it later.  
 **char** \*check\_number = read\_file(SQR2\_FILE);  
 **if** (!check\_number) {  
 // there was a problem reading the file, error message already printed  
 **return** 1;  
 }  
  
 **int** bufsize = strlen(check\_number) + 1; // plus the null character  
 **char** \*answer\_buf = (**char** \*)malloc((size\_t)bufsize);  
 **if** (!answer\_buf) {  
 fprintf(stderr, "Unable to allocate %d bytes for answer\n", bufsize);  
 free(check\_number);  
 **return** 1;  
 }  
  
 // convert the answer from a double to a string  
 // the precision (number of digits after the decimal point) is the number  
 // of digits after the decimal point in the data from NASA, which is  
 // assumed to be bufsize-2  
 sprintf(answer\_buf, "%.\*lf", bufsize-2, new\_guess);  
  
 // Compare the two strings. Can't use strcmp here because we want to know  
 // at what character the strings differ.  
 **char** \*a = answer\_buf;  
 **char** \*c = check\_number;  
 i = 0;  
 **while**(\*a != '\0' && \*c != '\0') {  
 **if** (\*a != \*c) {  
 **break**;  
 }  
 i++; a++; c++;  
 }  
  
 // Display the results  
 printf("\nThe value from NASA is:\n%s\n", check\_number);  
 printf("The value we computed is:\n%s\n", answer\_buf);  
  
 **if** (i == bufsize)  
 printf("The numbers are identical to %d significant digits.\n", i);  
 **else** printf("The numbers differ at position %d.\n", i+1);  
  
 // End point  
 end\_t = clock();  
 printf("end\_t = %ld\n", end\_t);  
 // Total compile time calculating  
 total\_t = (**double**)(end\_t - start\_t) / CLOCKS\_PER\_SEC;  
 printf("Total: %lf\n", total\_t );  
  
 // free the memory we used  
 free(check\_number);  
 free(answer\_buf);  
  
 **return** 0;  
}  
  
/\*  
 \* Read a file. Return the contents in a string. Memory for the string is allocated and must  
 \* be freed by the caller.  
 \*  
 \* This function discards all nondigit characters except for allowing one decimal point, if  
 \* present. (The input file is expected to contain a decimal number, possibly with leading  
 \* and/or trailing spaces and embedded newlines).  
 \*  
 \* Parameters:  
 \* in: file\_name - the name of the file to read  
 \*  
 \* Returns:  
 \* A pointer to a string containing the number in the file. The string is null-terminated.  
 \* Returns NULL if an error is encountered (file not found, memory allocation, etc).  
 \*/  
**char** \*read\_file(**const char** \*file\_name)  
{  
 // open the file for reading  
 FILE \*fp = fopen(file\_name, "r");  
 **if** (fp == NULL) {  
 fprintf(stderr, "Unable to open %s for reading\n", file\_name);  
 **return** NULL;  
 }  
  
 // determine the size of the file by reading each character one by one until we get  
 // to end of file, then rewinding the file pointer back to the beginning  
 size\_t size = 0;  
 **while**(fgetc(fp) != EOF) {  
 size++;  
 }  
 rewind(fp);  
  
 size ++; // add one for the null byte on the end  
  
 // allocate space for the contents of the file, include space for the null at the end  
 // of the string  
 **char** \*buf = (**char** \*)malloc(size);  
 **if**(!buf) {  
 fprintf(stderr, "Unable to allocate %ld bytes for file buffer\n", (**long**) size);  
 **return** NULL;  
 }  
  
 // Read the file one character at a time and store the characters in  
 // the buffer. Skip newlines. In fact, skip all characters that are not digits.  
 // Allow one decimal point. This has the side effect of validing that the input  
 // file actually contains a float number.  
 bool have\_decimal = false;  
 **int** i;  
 **char** \*p = buf;  
 **for** (i=0; i<size; i++) {  
 **char** c = (**char**)fgetc(fp);  
 **if** (c == '.') {  
 **if**(have\_decimal) {  
 // found more than one decimal point  
 fprintf(stderr, "Input from %s is not a valid float in decimal format\n", file\_name);  
 free(buf);  
 fclose(fp);  
 **return** NULL;  
 }  
 have\_decimal = true;  
 \*(p++) = c; // save the decimal point  
 } **else if** (isdigit(c)) {  
 \*(p++) = c; // save the digit  
 }  
 // else do nothing - do not save the character  
 }  
  
 // null-terminate the buffer (so that it becomes a string)  
 \*p = '\0';  
  
 // close the input file  
 fclose(fp);  
  
 // return the data from the file as a null-terminated string  
 **return** buf;  
}

################### Java

/\*\*\*\*\*\*\*\*\*\*

\* <p> Title: Compute Square Root of Two </p>

\*

\* <p> Description: A sample program to show how to compute square root of two using doubles </p>

\*

\* <p> Copyright: Copyright © 2009 </p>

\*

\* @author Lynn Robert Carter

\* @version 1.00 The initial version

\*

\*/

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.lang.Double;

public class Mainline {

public static void main (String [] args) {

long flag1 = System.nanoTime();

double sig = 1E+10; // Used to establish the limit for ending the loop

double value = 2.0; // We want the square root of this value

// This is the initial guess

double oldGuess = value / 2;

// We set up the newGuess to be the same as the guess

double newGuess = oldGuess;

// The variables used to control the loop

double limit; // The limit that determines the end of the loop

// Initialize the loop iteration counter

int iteration = 0;

do { // This is the implementation of Newton's method for finding square root

iteration++; // Keep track of the number of iterations

oldGuess = newGuess; // Establish the current guess based on the newGuess from before

// System.out.println(" oldGuess: " + oldGuess);

newGuess = (oldGuess + value / oldGuess) / 2.0; // compute the new guess based on the oldGuess

limit = newGuess / sig; // Establish the limit to be relative to the size of the guess...

// System.out.println("Interation: " + iteration + " has completed.");

// System.out.println();

// System.out.println(" newGuess: " + newGuess);

// Keep looping as long as the difference between the two guesses is larger than the relative limit and the

} while (Math.abs(oldGuess-newGuess) > limit && iteration < 20); // number of iterations is not too large

// Display the calculated square root

System.out.println();

System.out.println("The square root of two is: " + newGuess);

System.out.println();

// Read in the square root that NASA has computed from a file

try {

Scanner checkReader = new Scanner(new File("SquareRootTwo.txt"));

String checkNumber = "";

// Read in the value from NASA

while (checkReader.hasNextLine())

checkNumber += checkReader.nextLine().trim();

// Convert the computed value to a string

String answer = Double.valueOf(newGuess).toString();

// Compare the two strings

int diff = -1;

for (int i = 0; i < answer.length(); i++)

if (checkNumber.charAt(i) != answer.charAt(i)){

diff = i;

break;

}

// Display the results

System.out.println();

System.out.println("The value from NASA is:");

System.out.println(checkNumber);

System.out.println();

System.out.println("The value we computed is:");

System.out.println(answer);

System.out.println();

if (diff == -1)

System.out.println("The numbers are identical to " + (answer.length()- 1) + " significant digits.");

else

System.out.println("The numbers differ at position " + diff);

} catch (FileNotFoundException e) {

System.out.println("TNot able to find the file: SquareRootTwo.txt");

}

long flag2 = System.nanoTime();

System.out.println("Flag 1(Start time)nanoS: " + flag1);

System.out.println("Flag 2(Finish time)nanoS: " + flag2);

double time = (flag2-flag1) \* 1e-9;

System.out.println("Time of compile: " + time);

}

}



