

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 04

Exercise 04

SQUARE ROOT IN C WITH DOUBLES

# Name: Minakov Nikita

PURPOSE

The purpose of this exercise is to give you concrete practice with buffered input and output in C. You will also be creating a small program in C that computes the square root of two.

For this exercise, you are provided with a C program that compares a double value, digit by digit, with a number in a text file. The program displays the position at which the numbers differ. Your task is to add code to the main function of this program to calculate the square root of two.

ACTIVITIES

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

1. Download and unzip (or unarchive) Exercise 04 from the LMS.
2. Change directory to the src directory, where you will find a C program and a make file. The makefile for this exercise will compile a source file named exercise04.c into an executable called exercise04.
3. Edit the C source file and add code where indicated to compute the square root of two, using the following algorithm (based upon Newton’s method for calculating a square root) :
   1. Guess at the square root value (e.g. the value divided by 2)
   2. Compute a result by dividing the value by the guess
   3. Compute a new guess by averaging the result and the guess
   4. Compute the value delta by dividing the guess by 10n where n is the number of significant digits required
   5. If the new guess differs from the guess by more than the delta, set guess to be the value of new guess and go back to step ‘b’
   6. The new guess value is the square root
4. Use 1E+10 for the number of significant digits specified in the algorithm. The loop indicated in the algorithm should be executed a maximum of 20 times (although it may exit sooner based on step ‘e’ in the algorithm).
5. Use “make” to compile the C program and then run it.
6. Your calculated square root of two should differ from the value in the text file at position 17 (your value and the value in the text file should be identical up through position 16, including the decimal position). Copy and paste the output of your program below:
7. When you are ready to submit your work, first remove all intermediate files from your src directory. This includes exercise04.o and the exercise04 executable. You can use the “make” command to do this by typing “make clean” at the command prompt.
8. Save and archive your work, including this document and your finished program, and upload it to the LMS.

My code:

#include <stdio.h>  
#include <math.h>  
#include <stdlib.h>  
#include <string.h>  
#include <stdbool.h>  
#include <ctype.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**)  
{  
 **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);  
  
 // 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;  
}

