

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 06

Exercise 06

Function Libraries in c

# Name: Minakov Nikita Csse1709r

PURPOSE

The purpose of this exercise is to introduce the design concept of libraries in C and how they are implemented.

A library in C is simply a group of related functions and associated program constructs (e.g. special types, constants). Conceptually, libraries in C have public functions and private functions (in the Java sense). The notion of “public” and “private” are *not* enforced or explicitly supported by the C language. The creation and use of libraries in C are geared towards good design and modular programs, which are easier to create and maintain.

Examples of libraries that you have been using are the stdio library (printf, etc.), the time library (for clock, the constant CLOCKS\_PER\_SEC, etc.). Some of the C libraries have man pages, e.g. “man stdio” displays the manual page for the stdio library.

A library typically is comprised of a source file containing the functions that are part of the library, and a header file containing definitions required for a program to use the library. Function prototypes in the header file define the *interface* for the library. These functions can be thought of as “public” functions. The function prototypes defined in the library’s header file make the interface “visible” to source files that include the library header file. Functions that do not have a prototype in the library header file are not accessible from the including source file. It is important to understand that this is merely a convention. Successful application development relies on adherence to this convention.

Header files that you create should adhere to the convention of defining a constant specific to the header file. This constant is used to prevent a header file from being included multiple times in the same source file. The format of your header file should be as follows:

/\* a comment describing the header file’s purpose \*/

#ifndef SQROOT\_LIBRARY // if this library has not been included yet...

#define SQROOT\_LIBRARY // Remember that this library has now been included

<your prototypes and definitions go here>

#endif // end of if body

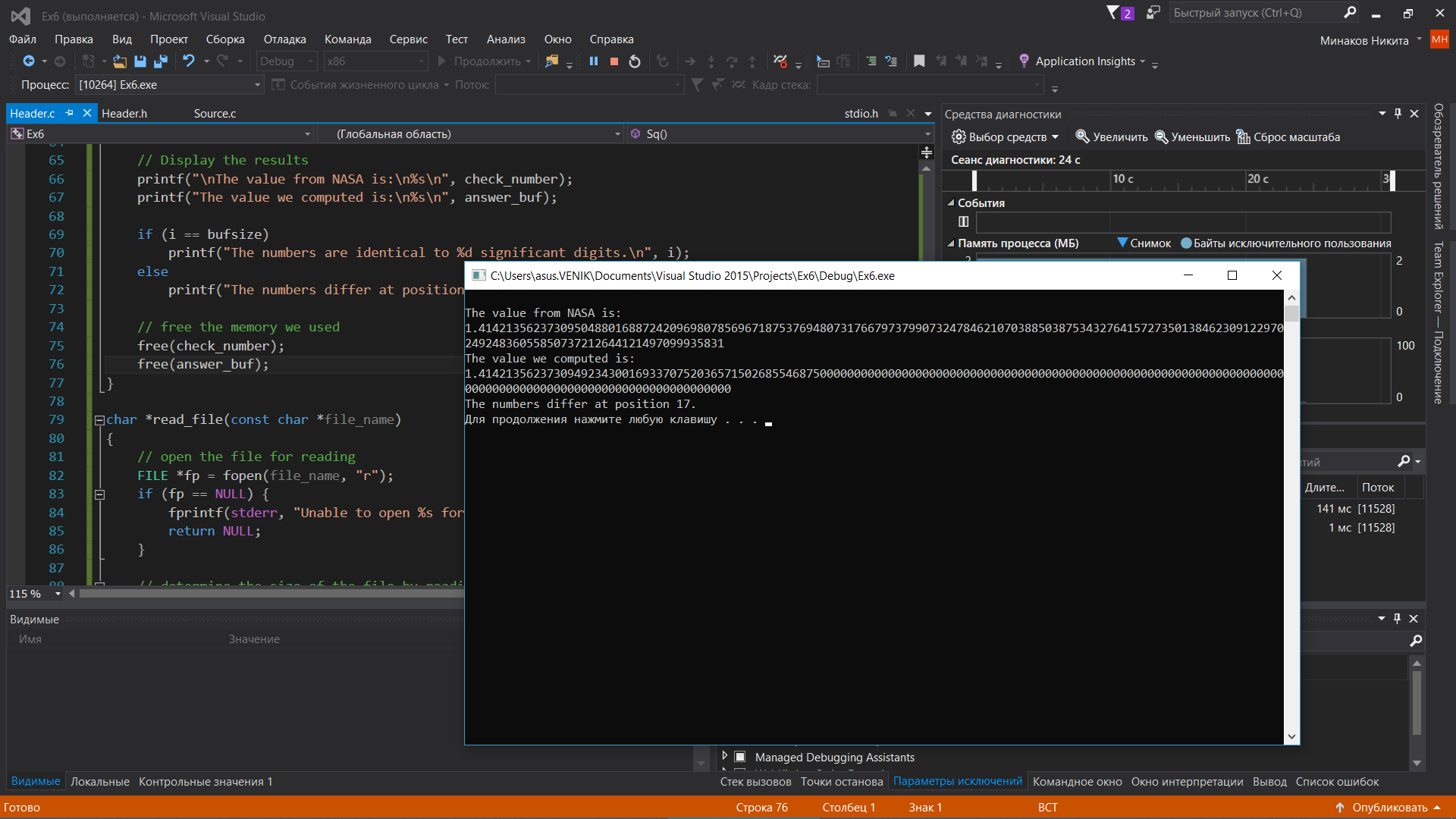
Header files that are part of your program (as opposed to the operating system and development environment) are considered “local”. System header files, such stdio.h, are included in your program using angle brackets (i.e. #include <stdio.h>). Local header files are included using quotes (e.g. #include “your\_header.h”). The angle brackets and quotes tell the compiler where on the computer to look for the header files.

You will use the solution to Exercise 05 as a starting point for this exercise and create a library that contains the computation part of the program (the calculation for the square root).

ACTIVITIES

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

1. Create a function that calculates the square root of a value. The code you created in Exercise 04 (and that was included in the solution to Exercise 05) will be the main body of your new function. Put this function in a separate source file from that containing main(). Name the file “sqroot.c”.
2. Create a header file containing the function prototype for your new function. This defines the interface to your library. Name the file “sqroot.h”.
3. Modify main() to include your header file and call your new square root function.
4. Using the makefile supplied for this exercise, compile your program to be sure that it has no errors. If the output of make indicates that it does not know how to make a file, make sure that you have correctly named your source files as specified above.
5. Run your program and verify that the output has not changed.
6. Examine the makefile given to you for this exercise and compare it to the makefile from Exercise 05 to see how dependencies for libraries are defined. (You will need this information for Project 02, where you will create your own makefile.)
7. When you are ready to submit your work, first remove all intermediate files from your src directory. This includes exercise06.o and the exercise06 executable as well as the intermediate file for your library (sqroot.o). You can use the “make” command to do this by typing “make clean” at the command prompt.
8. Save and archive your finished C program, including both of your C source files and your header file, and upload it to the LMS.



Main.c:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include"Header.h"

#define SQR2\_FILE "SquareRootTwo.txt"

int main(void)

{

Sq();

system("pause");

return 0;

}

Header.h:

#pragma once

#ifndef SDP4\_MAIN\_H

#define SDP4\_MAIN\_H

#endif //SDP4\_MAIN\_H

#include <stdio.h>

#include <stdbool.h>

#include <corecrt\_malloc.h>

char \*read\_file(const char \*file\_name);

void Sq();

Header.c:

//

// Created by asus on 24.01.2019.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Header.h"

#define SQR2\_FILE "SquareRootTwo.txt"

void Sq() {

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);

}

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;

}