

# PERFORMANCE, DATA STRUCTURES AND ALGORITHMS

Exercise 10



Exercise 10

Read and write a unumber

# Name: Minakov Nikita Csse1709R

PURPOSE

The purpose of this exercise is to give you practice at working with structures in C.

Characters in C are encoded using the ASCII table (refer back to chapter 2 in your textbook). You can search the Internet to find the entire table. There are some useful characteristics of the ASCII table. One is that alphanumeric characters are listed in numerical order. Thus, the character ‘0’ is less than ‘1’. In fact, ‘0’ is exactly one less than ‘1’, since the ASCII code for ‘0’ is 48 and the ASCII code for ‘1’ is 49. This allows us to compare characters without converting them to numbers. It also allows an easy way to translate ASCII characters to numeric values. For example, the string “123” is comprised of 4 bytes with the values 49, 50, 51, and 0 (0 is the terminating 0 byte). Conveniently, we can easily translate this into a numeric byte array by subtracting 48 (the ASCII code for ‘0’) from each digit: 49-48, 50-48, 51-48 yields (numeric) 1, 2, and 3. Conversely, if you have a byte array with numeric values, you can add 48 to each value to get the printable ASCII character for that value.

For this exercise you are given a program that reads data from “stdin” (from the terminal). Your task in this exercise is to write a function to populate an unlimited precision number structure from the input values and a second function to print the contents of the structure.

The program repeatedly prompts the user for input until the user enters an empty line (hits return at the prompt). The input has three values separated by commas: <sign>, <decimalpower>, and digits”. <sign> is either + or -, decimalpower is an integer; digits are the digits in the number. All values are required (i.e. the sign is not optional).

Example input: +,1,123

Example output:

sign = positive, decimal power = 1, size = 3, value = 123

You entered the number 1.23

Example input: -,2,123

Example output:

sign = negative, decimal power = 2, size = 3, value = 123

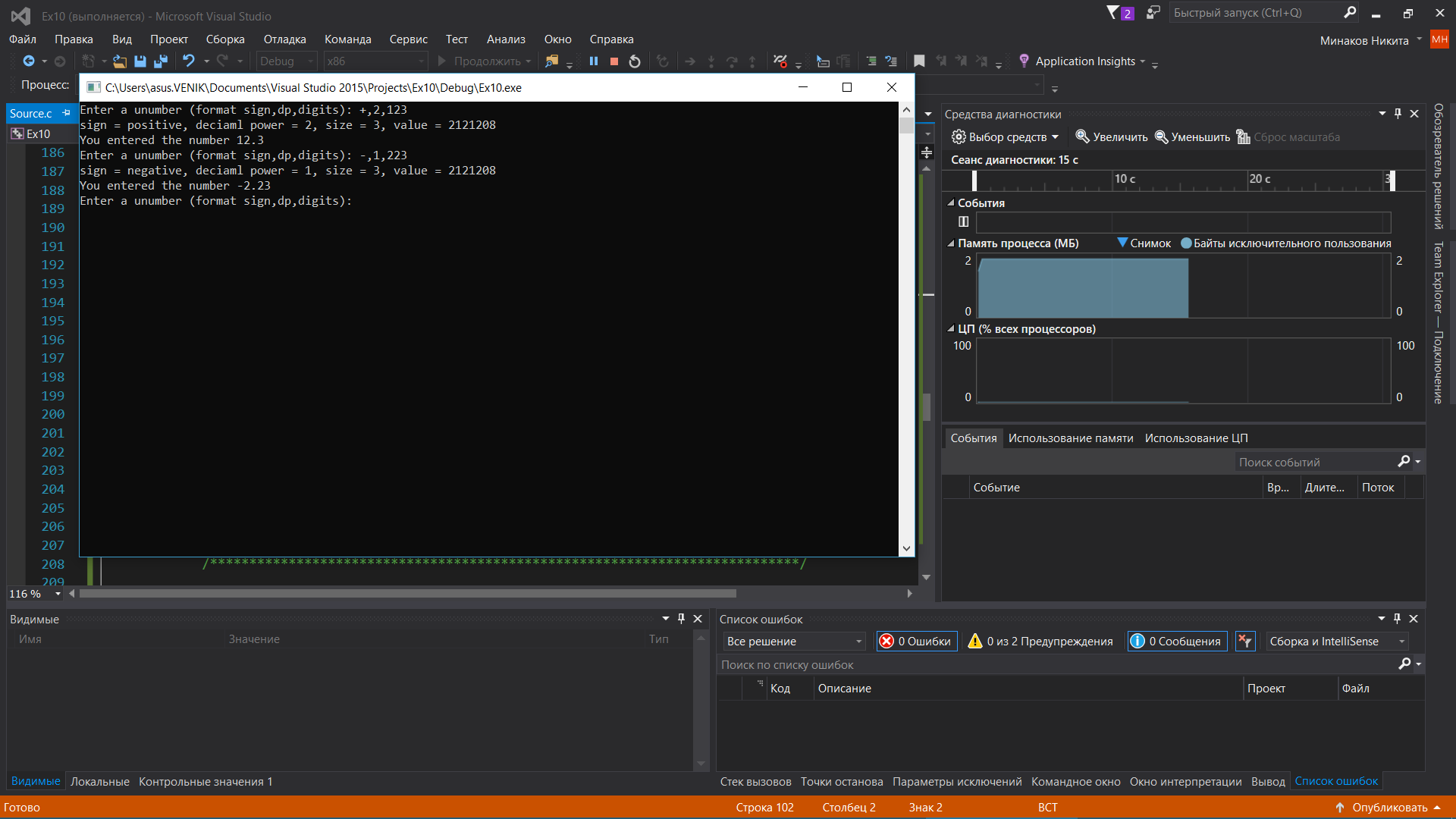
You entered the number -12.3

You will not need to change main() for this exercise.

ACTIVITIES

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

1. Complete the body of the new\_unumber\_from\_string() function so that it correctly stores the different parameter values into the UNumber parameter.
2. Add code to the body of the print\_unum\_struct() function so that the function displays a Unumber structure in the format of the *first* line of output in the examples above.
3. Test your program thoroughly with different values to verify that it works correctly.
4. Document your code carefully.
5. In your Engineering Notebook, explain your approach to creating your code and your testing strategy. Include examples of the test input you used and the corresponding output from your program.
6. When you are ready to submit your work, first remove all intermediate files from your src directory.
7. Save and archive your finished C program and upload it to the LMS.



#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define INPUT\_SIZE 512 // plenty large for one input line

typedef struct unumber\_struct {

char \*unum; // an array containing the unumber in numeric form

int size; // the number of elements in unum

int dp; // the decimal power of the unum

bool sign; // the sign (true = positive)

} UNumber;

#ifdef MEMORY\_TRACE

// a global to count the number of allocated memory blocks

// increment once when memory is allocated, decrement once when memory

// is freed. If this is not zero at the end of the program, there is

// a memory leak somewhere.

//

// This must be initialized to zero at the start of the program.

int allocated\_memory\_blocks;

/\*

\* A utility function for printing debugging output associated with

\* allocating memory. Use paired with debug\_free\_memory to get output

\* for a rudimentary heap trace. This function is only defined if the

\* program is compiled with the -DDEBUG\_TRACE option.

\*

\* Parameters:

\* in: msg - a message to print for debugging

\* in: num - the number of items to be allocated

\* in: size - the size of each item

\* in: zero - if true, calloc is used to zero allocated memory, else malloc

\* (memory not zeroed)

\*

\* Returns: pointer to the allocated memory or NULL if memory could not be

\* allocated. The returned pointer must be cast to the appropriate

\* pointer type by the caller.

\*/

void \*allocate\_memory(const int num, const int size, const bool zero)

{

// if the boolean zero is true, use calloc, which allocates

// memory and writes zeroes to it. Use malloc otherwise, which

// does not initialize the memory. Calloc is much less efficient,

// so we avoid using it when malloc will suffice.

void \*p = (zero ? calloc(num, size) : malloc(num \* size));

if (p)

allocated\_memory\_blocks++;

return p;

}

/\*

\* A utility function for printing debugging output associated with freeing memory.

\* Use paired with debug\_allocate\_memory to get output for a rudimentary heap trace.

\* This function is only defined if the program is compiled with the -DDEBUG\_TRACE option.

\*

\* Parameters:

\* in: msg - a message to print for debugging

\* in: p - pointer to the memory to be freed

\*

\* Returns: n/a

\*/

void free\_memory(void \*p)

{

free(p);

allocated\_memory\_blocks--;

return;

}

#else

// We only want to use the allocate\_memory and free\_memory functions if we are tracing

// memory usage. So if we are not tracing memory usage, define the names of those

// functions to be macros that are replaced at compile time with actual calls to malloc,

// calloc and free.

//

// See function definitions above for parameters.

#define allocate\_memory(n,s,z) (z ? calloc(n,s) : malloc(n\*s));

#define free\_memory(p) free(p)

#endif

/\*

\* Delete a unumber - this frees the memory of a unumber. Allows the programs that use

\* this library to not look into the structure (they still can, but they shouldn't have

\* toa).

\*

\* Parameters:

\* in: msg - a message to print for debugging

\* in: del - pointer to a UNumber structure to delete

\*

\* Returns: n/a

\*/

void free\_unumber(UNumber \*del)

{

// wrapping this up in a function will allow us to include this function in

// a unumber library later so that the program that uses the unumber library

// doesn't have to know what's in the structure (mimicking Java private data)

free\_memory(del->unum);

return;

}

/\*

\* Convert a unumber to a string. Allocates memory for the string, which must later be freed.

\*

\* Parameters:

\* in: num - the number to convert to a string

\*

\* Returns: NULL if unable to allocate memory, else a pointer to the new string

\*/

char \*get\_number\_as\_string(const UNumber \*num)

{

// size of the byte array plus decimal point, sign, leading zero, null terminator,

// and space for trailing zeroes, if needed (e.g. 12000000)

int end\_size = num->size + 4 + num->dp;

if (num->dp < 0)

end\_size -= num->dp; // need room for leading zeros

char \*p = (char \*)allocate\_memory(end\_size, 1, false);

if (!p)

return NULL; // unable to allocated memory

int i;

int pi = 0;

int added\_characters = 0;

if (!num->sign) {

p[pi++] = '-';

added\_characters += 1;

}

if (num->dp <= 0) {

p[pi++] = '0';

added\_characters += 1;

}

if (num->dp < 0) {

p[pi++] = '.';

for (i = 0; i>num->dp; i--) {

p[pi++] = '0';

}

for (i = 0; i<num->size; i++) {

p[pi++] = num->unum[i] + 48;

}

}

else {

for (i = 0; i< num->size; i++) {

if (pi - added\_characters == num->dp) {

p[pi++] = '.';

}

p[pi++] = num->unum[i] + 48; // e.g. 0 + 48 = '0' (ascii)

}

for (i = pi; i< num->dp; i++)

p[pi++] = '0';

p[pi] = '\0';

}

return p;

}

/\*

\* Create a new UNumber with the given member values. The specified input number

\* is in the form of a string of ascii digits ('1', '2', etc.). The ascii digits are

\* converted to numeric values by substracting 48. This function allocates memory

\* which must later be freed.

\*

\* Parameters:

\* in: num - pointer to a UNumber structure to hold the return values

\* in: number - string containing the number to create

\* in: dP - the decimal power to be used for the new number

\* in: sign - the sign for the new number

\*

\* Returns: false if there were no errors, else true

\*/

bool new\_unumber\_from\_string(UNumber \*num, const char \*number, const int dp, const char sign)

{

if (sign != '-' && sign != '+') {

fprintf(stderr, "invalid sign '%c'\n", sign);

return true;

}

// Allocate space for the new number

num->unum = (char \*)allocate\_memory(strlen(number), 1, true);

if (!num->unum)

return true; // unable to allocate memory

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Student's Code Goes Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int i = 0;

for (; i < strlen(number); i++)

{

num->unum[i] = number[i] - 48;

}

// Put your code here

num->dp = dp;

if (sign == '+')

{

num->sign = true;

}

else

{

num->sign = false;

}

num->size = strlen(number);

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

return false;

}

/\*

\* A utility function for printing the contents of a UNumber

\* struct to stdout.

\*

\* Parameters:

\* in: us - a pointer to the UNumber structure to be printed

\*

\* Returns: n/a

\*/

void print\_unum\_struct(const UNumber \*us)

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Student's Code Goes Here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (us->sign == true)

{

printf("sign = positive, ");

}

else

{

printf("sign = negative, ");

}

printf("deciaml power = %d, ", us->dp);

printf("size = %d, ", us->size);

printf("value = %d", us->unum);

printf("\n");

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

return;

}

/\*

\* Read information about a UNumber from stdin, store that information in a UNumber

\* structure, print the components of the structure, and display the number as a

\* floating point number.

\*

\* Parameters: n/a

\*

\* Returns:

\* 0 on success, else 1

\*/

int main(void)

{

#ifdef MEMORY\_TRACE

allocated\_memory\_blocks = 0;

#endif

const char prompt[] = "Enter a unumber (format sign,dp,digits): ";

char input[INPUT\_SIZE]; // for the user's input

UNumber unum; // A UNUmber structure for the input values

char sign; // the sign entered by the user

int dp; // the decimal power entered by the user

char digits[INPUT\_SIZE];// The digits entered by the user

// prompt for input

printf(prompt);

// loop until the user hits return at the prompt (1st character will be a newline)

while (fgets(input, INPUT\_SIZE, stdin) && input[0] != '\n') {

// get the components of the user's input, which is comprised of one character, one integer,

// and one string separated by commas. The string is simply the remainder of the input on

// the line after the last comma.

if (sscanf(input, "%c,%d,%s", &sign, &dp, digits) != 3) {

// Did not get the expected input

fprintf(stderr, "invalid input: %s\n", input);

}

else {

// Input has the expected format

// populate a UNumber structure with the input values

if (new\_unumber\_from\_string(&unum, digits, dp, sign)) {

fprintf(stderr, "Error getting unumber from string\n");

}

else {

// Print the components of the unumber structure that the user entered

print\_unum\_struct(&unum);

// Convert the UNumber to a string (e.g. -1.123) for display

char \*p = get\_number\_as\_string(&unum);

printf("You entered the number %s\n", p);

// free memory allocated by get\_number\_as\_string

free\_memory(p);

// free the UNumber structure

free\_unumber(&unum);

}

}

// prompt the user for input again

printf(prompt);

}

#ifdef MEMORY\_TRACE

if (allocated\_memory\_blocks)

printf("There was a memory leak!! %d memory blocks not freed\n", allocated\_memory\_blocks);

else

printf("Congratulations! All memory that was allocated was freed!\n");

#endif

return 0;

}