College of

Nanotechnology, Sciences, and Engineering Department of Computer Science

ICSI 333 SYSTEM FUNDAMENTALS CQUPT, SPRING 2025

LAB 2.1 - 2.3

The total grade for three labs is 100 points.

You must follow the programming and documentation guidelines.

**DESCRIPTION**

You must write a C program that generates random numbers in the range of [-100.0, 100.0] and, depending on the value of the number, creates a new node at the beginning of one of the four linked lists: positive numbers with a non-zero whole part, positive fractions, negative fractions, and negative numbers with the non-zero whole part. The program stops when the total amount of randomly generated positive numbers reaches the value specified by the user.

The nodes of the linked lists must keep the generated number, the structure with a sign, an exponent, and a mantissa of IEEE 754 representation of the number, the value restored from the IEEE 754 representation, and a pointer to the next node.

If the restored value is not equal to the value of the generated number, the node should not be added, and an error message should be printed to stderr.

Finally, each linked list must be saved in a separate file, but all four files must have the same extension specified by the user. Special requirements:

* The user preferences must be inputted as command-line arguments.
* You should use the results of Project 1 as a library function that returns the structure with the sign, exponent, and mantissa.
* To calculate the restored value, you should write and add to the library a function that implements the following formula VALUE = -1*S* × 1.*M* × 2*E*-127, where *S* is the sign bit (0 or 1), *M* is the mantissa (000...0002 to 111...1112) and *E* is the biased exponent (000000002 to 1111 11102).
* To get full credit for the project, it must be organized into three files: the main program, the library module with implementations of IEEE 754 conversion functions, and the header file.

Work on the project task by task.

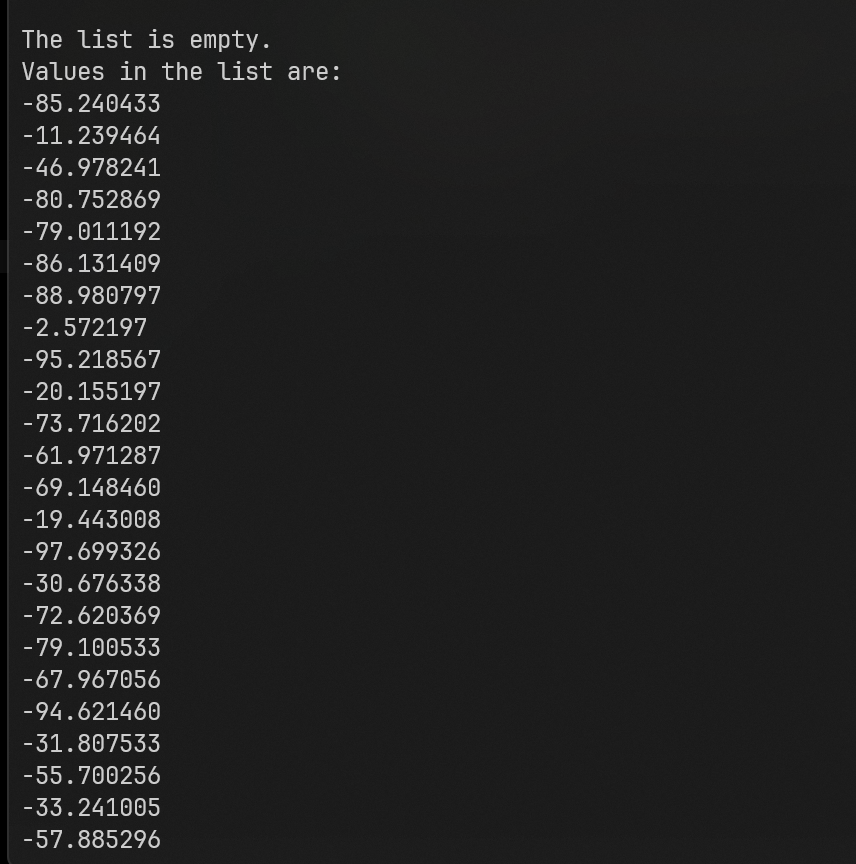
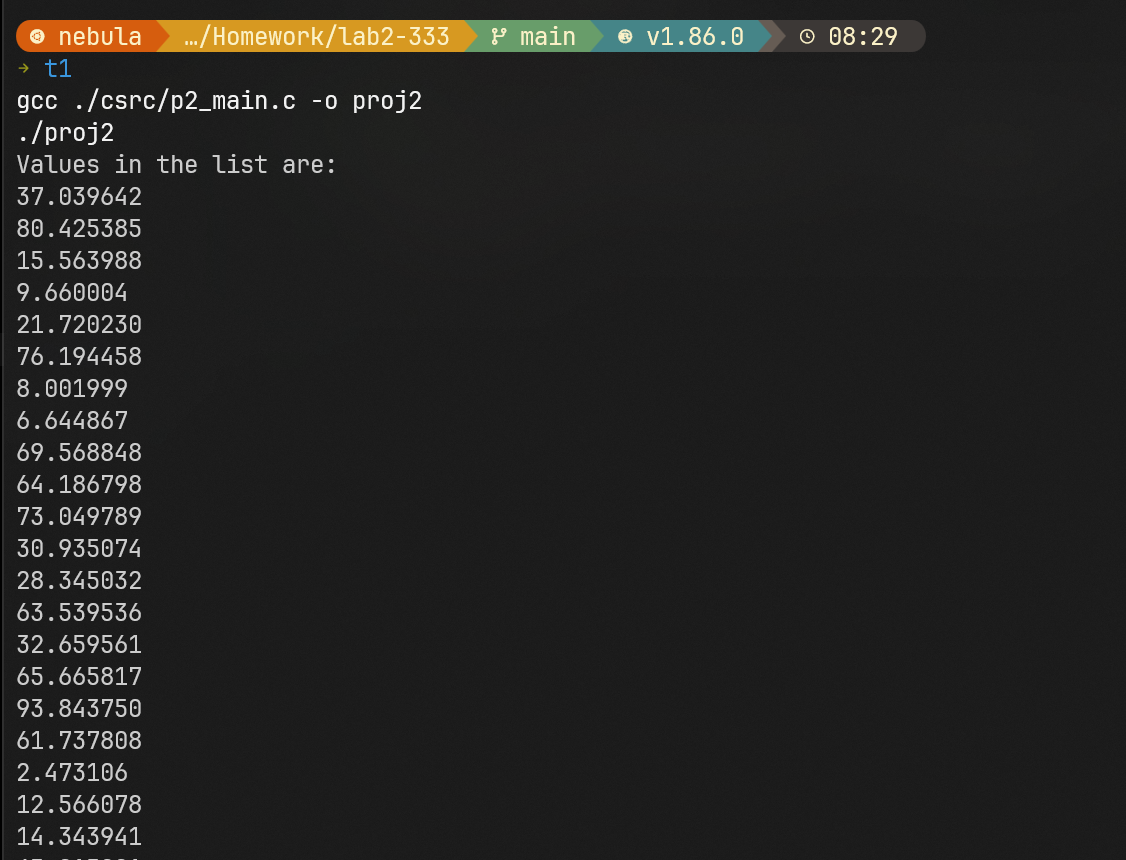
**RECOMMENDED STEPS OF DEVELOPMENT**

TASK #1. LINKED LISTS

Start with the generating of random numbers with the function rand(). You may find an example in the lecture handouts. But this time the result should be a float number in the range of [-100.0, 100,0].

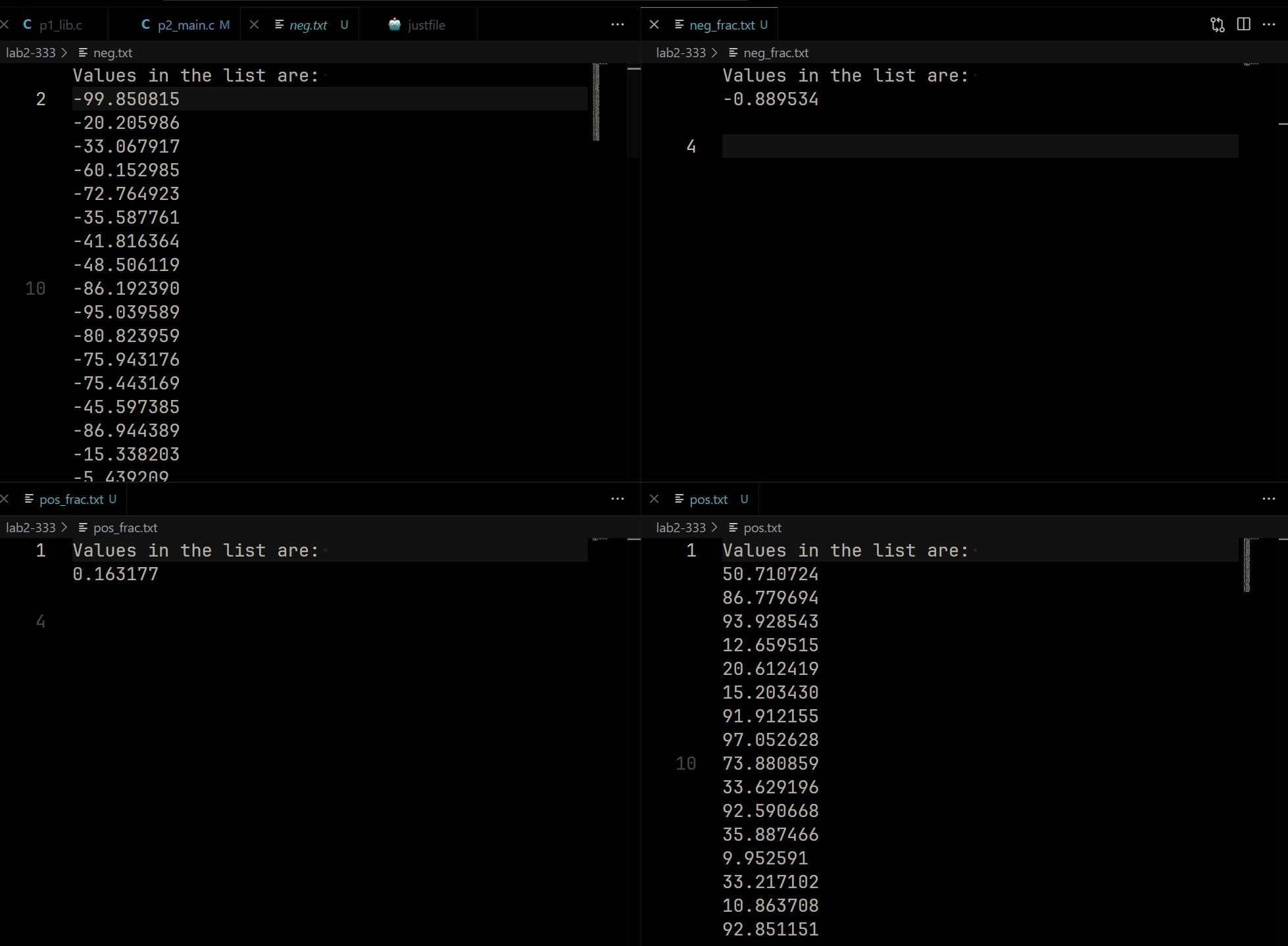
At this stage, you can hardcode the desired quantity of positive numbers. I recommend setting it to at least 100, so your program can generate numbers in each of the four categories given in the description.

To add a new node to the proper linked list you may use the lecture handout or textbook examples but note that the new nodes must be added at the beginning of the lists.



TASK #2. FILES

At this stage, you can hardcode the file names and simplify the nodes, so they hold only a number and a pointer. Your program must save four linked lists in four different files.

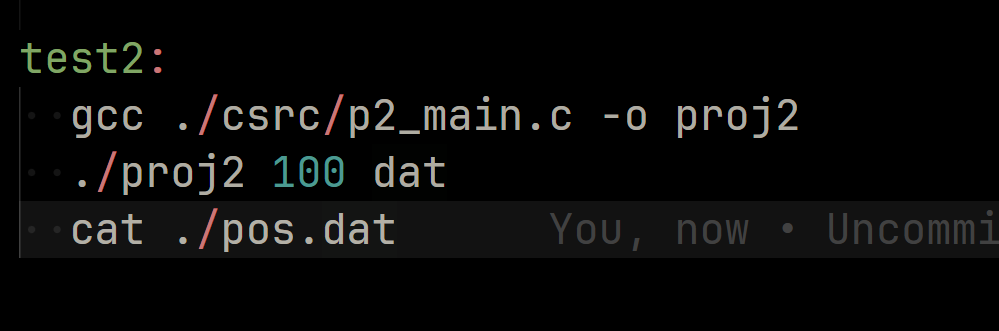


TASK #3. COMMAND LINE ARGUMENTS

Function main() can take string arguments from the command line. Your program must be run with two arguments: the desired quantity of positive numbers to stop random generating and the files’ extension.

You must consider the situation when the user misses the arguments. It will be your choice to use the default values or stop execution with a message to the user.

The example of command line arguments you can find in the textbook or the handouts to the lecture about files.



Output is similar to Task#2 expect for the filename suffix.

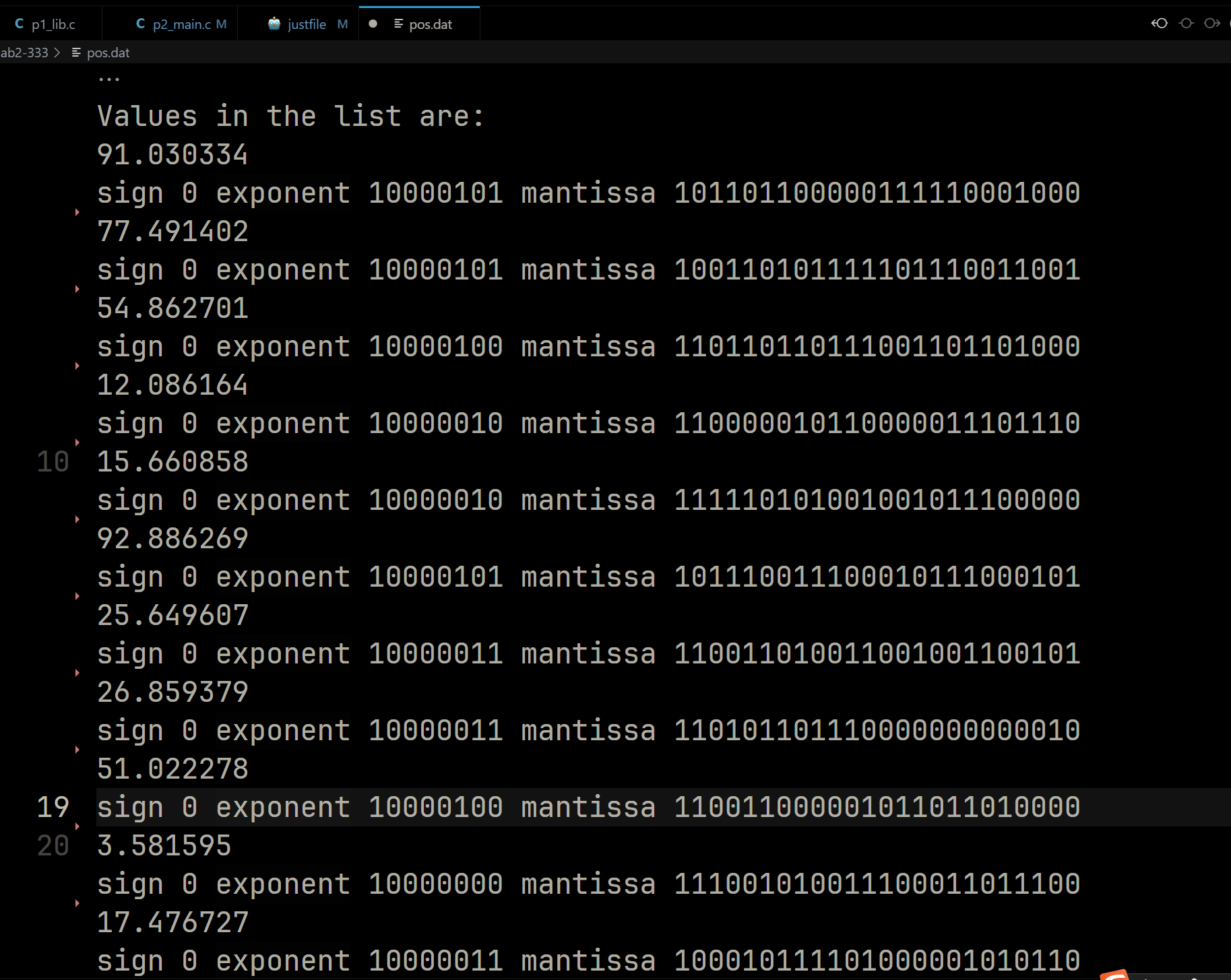
TASK #4. LINKING TO PROJECT 1 FROM LAB 1

Modify your Project 1, so it has no main() function but a collection of functions including one that receives a real number and returns a structure that holds IEEE 754 representation as sequences of 0s and 1s with the fields: sign, exponent, mantissa. Use typedef to define this structure as a new type in a header file. The header file must have the same name as the file with functions from Project 1 and the extension .h. The header must also have the prototypes of all functions that can be used by the third file. For example, if your modified Project 1 file has the name p1\_lib.c, then the header file must have the name p1\_lib.h. Both p1\_lib.c and the main Project 2 file must include the new header file (note double quotes):

#include “p1\_lib.h“

All files must be in the same directory, then you can compile two source code files together:

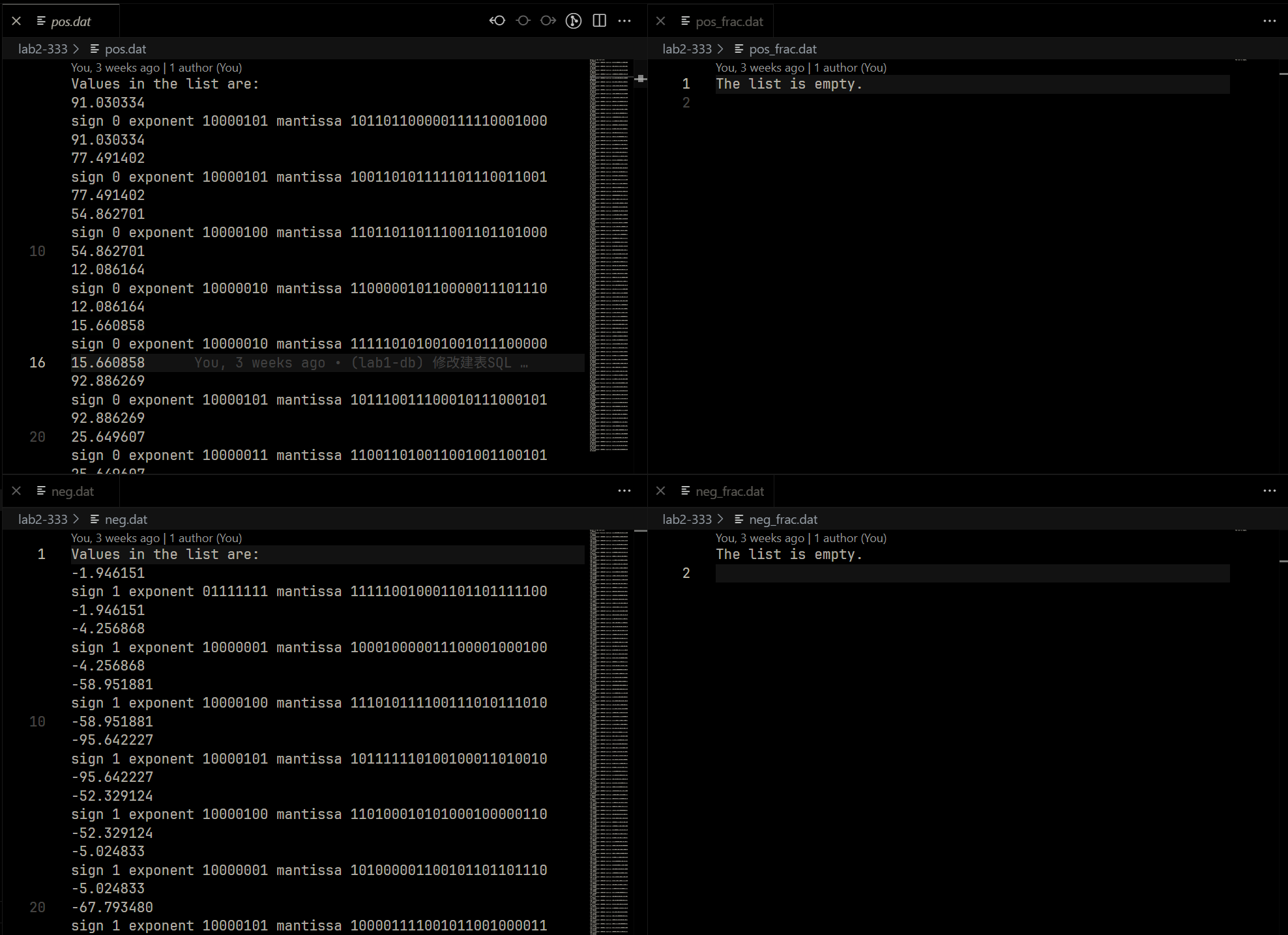
gcc p2\_main.c p1\_lib.c -o proj2



TASK #5. ELABORATING ON THE LINKED LIST NODES

Add a structure with IEEE 754 representation and a float field to your nodes. The new fields must be calculated before the node is added to the list. Use your Project 1 library to calculate the IEEE 754 representation for the random number.

Add a new function to your library that restores the decimal value from the IEEE 754 representation. Use this function to calculate and save the restored number in the field. If the restored number is not equal to the original number, the node should not be added but the error message should be printed to stderr.

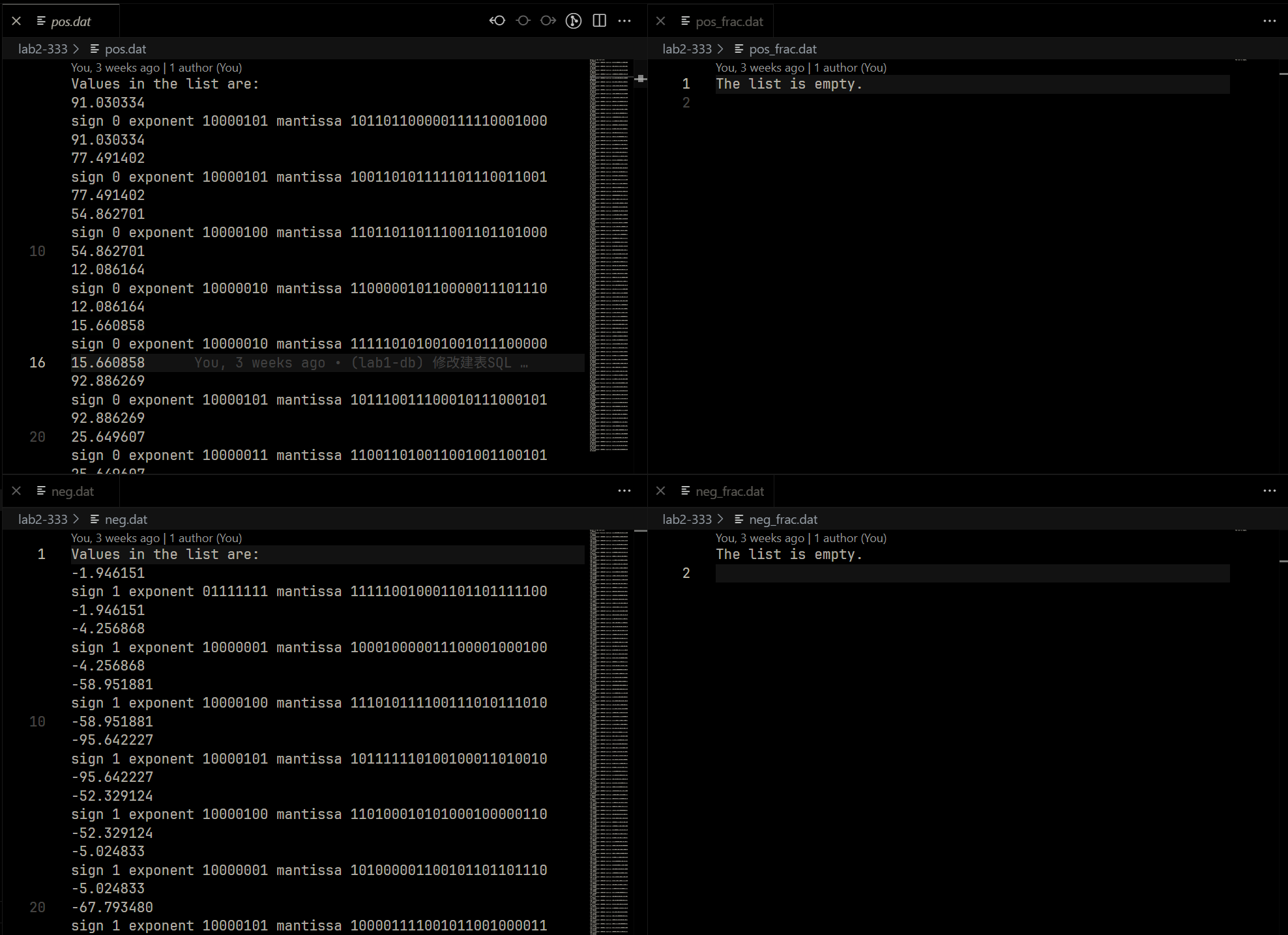


**PROGRAM EXECUTION**

./proj2 100 dat

After running the above, the four files must be created. The examples are attached.

Newly created files:



**SUBMISSION**

You must submit only one, **the most successful**, version. Please review your work carefully before submitting it. Also, check the grading rubric to ensure that your program meets all requirements.

Your lab instructor will provide you with a detailed description of the submission process.

Source Code:

/\*

 \* Author: Gao Junran

 \* Date: 2024-03-26

 \* Description: This program converts a floating-point number into its IEEE 754 single-precision representation.

 \*              The program has been tested against https://baseconvert.com/ieee-754-floating-point for accuracy.

 \*/

 #include "p1\_lib.h"  // Include the library for IEEE 754 representation structures

 #include <stdio.h>

 #include <stdlib.h>

 #include <string.h>

 #include <math.h>

 /\*\*

  \* Pads a binary string with leading or trailing zeros to ensure a fixed size.

  \*

  \* @param source The input binary string.

  \* @param target The output string with padding applied.

  \* @param size The fixed size to pad to.

  \* @param isLeading If 1, pads with leading zeros; if 0, pads with trailing zeros.

  \*/

 void pad\_zero(char \*source, char \*target, int size, int isLeading) {

     int len = strlen(source);

     if (len >= size) {

         strcpy(target, source);

         return;

     }

     int padCount = size - len;

     if (isLeading) {

         // Add leading zeros

         for (int i = 0; i < padCount; i++) {

             target[i] = '0';

         }

         strcpy(target + padCount, source);

     } else {

         // Copy source and add trailing zeros

         strcpy(target, source);

         for (int i = 0; i < padCount; i++) {

             target[len + i] = '0';

         }

         target[size] = '\0';

     }

 }

 /\*\*

  \* Converts an integer to its binary representation.

  \*

  \* @param num The integer to convert.

  \* @param binary The output binary string.

  \*/

 void int\_to\_binary(long long int num, char \*binary) {

     int index = 0;

     char temp[MAX\_SIZE] = {0};

     while (num > 0) {

         temp[index++] = (num % 2) + '0';

         num /= 2;

     }

     int len = index;

     for (int i = 0; i < len; i++) {

         binary[i] = temp[len - i - 1]; // Reverse the binary string

     }

     binary[len] = '\0';

 }

 /\*\*

  \* Converts a fractional decimal to its binary representation.

  \*

  \* @param dec The fractional part of a number.

  \* @param binary The output binary string.

  \*/

 void dec\_to\_binary(long double dec, char \*binary) {

     int index = 0;

     if (dec == 0.0) {

         binary[index++] = '0';

     }

     while (dec > 0.0 && index < 32) { // Limit to 32-bit precision

         dec \*= 2.0;

         if (dec >= 1.0) {

             binary[index++] = '1';

             dec -= 1.0;

         } else {

             binary[index++] = '0';

         }

     }

     binary[index] = '\0';

 }

 /\*\*

  \* Computes the mantissa from integer and fractional binary parts.

  \*

  \* @param int\_binary Binary representation of the integer part.

  \* @param dec\_binary Binary representation of the decimal part.

  \* @param mantissa The output mantissa string.

  \*/

 void calculate\_mantissa(char \*int\_binary, char \*dec\_binary, char \*mantissa) {

     char binary[MAX\_SIZE] = {0};

     strcpy(binary, int\_binary);

     strcat(binary, dec\_binary);

     // Normalize by removing leading zeros

     while (binary[0] == '0' && binary[1] != '\0') {

         memmove(binary, binary + 1, strlen(binary));

     }

     if (binary[0] == '\0') {

         strcpy(binary, "1");

     }

     pad\_zero(binary, mantissa, MANTISSA\_SIZE, 0); // Pad with trailing zeros

     mantissa[MANTISSA\_SIZE] = '\0';

 }

 /\*\*

  \* Computes the exponent value for IEEE 754 representation.

  \*

  \* @param real The input floating-point number.

  \* @param exponent The output exponent string.

  \*/

 void calculate\_exponent(long double real, char \*exponent) {

     int exp\_val = 0;

     if (real >= 1.0) {

         while (real >= 2.0) {

             real /= 2.0;

             exp\_val++;

         }

     } else {

         while (real < 1.0) {

             real \*= 2.0;

             exp\_val--;

         }

     }

     exp\_val += 127; // Biasing exponent

     if (exp\_val > 255) {

         exp\_val = 255; // Clamp overflow

     }

     char temp[MAX\_SIZE];

     int\_to\_binary(exp\_val, temp);

     pad\_zero(temp, exponent, EXPONENT\_SIZE, 1); // Pad with leading zeros

 }

 /\*\*

  \* Converts a floating-point number to its IEEE 754 single-precision representation.

  \*

  \* @param real The input floating-point number.

  \* @return The IEEE 754 representation of the input number.

  \*/

 F32Repr float\_to\_IEEE754(long double real) {

     F32Repr repr;

     repr.sign[0] = (real >= 0.0) ? '0' : '1'; // Determine sign bit

     repr.sign[1] = '\0';

     real = fabs(real);

     long long int int\_part = (long long int)real;

     long double dec\_part = real - int\_part;

     char int\_binary[MAX\_SIZE] = {0}, dec\_binary[MAX\_SIZE] = {0};

     int\_to\_binary(int\_part, int\_binary);

     dec\_to\_binary(dec\_part, dec\_binary);

     calculate\_mantissa(int\_binary, dec\_binary, repr.mantissa);

     calculate\_exponent(real, repr.exponent);

     if (strcmp(repr.exponent, "11111111") == 0) { // Check for overflow

         memset(repr.mantissa, '0', MANTISSA\_SIZE);

         repr.mantissa[MANTISSA\_SIZE] = '\0';

     }

     return repr;

 }

 /\*\*

  \* Converts an IEEE 754 single-precision representation back to a floating-point number.

  \*

  \* @param repr The IEEE 754 representation.

  \* @return The floating-point number corresponding to the input representation.

  \*/

 float IEEE754\_to\_float(F32Repr repr) {

     // 1. Parse the sign bit

     int sign = (repr.sign[0] == '1') ? 1 : 0;

     // 2. Parse the exponent (convert binary string to integer)

     int exponent = (int)strtol(repr.exponent, NULL, 2);

     // 3. Parse the mantissa

     double mantissa = 1.0; // Normalized, starts with 1.

     for (int i = 1; i < MANTISSA\_SIZE; i++) {

         if (repr.mantissa[i] == '1') {

             mantissa += pow(2, -i); // Contribution of each bit

         }

     }

     // 4. Calculate the actual floating-point value

     int exp\_adjusted = exponent - 127; // Adjust for bias

     double value = mantissa \* pow(2, exp\_adjusted);

     // 5. Apply the sign

     if (sign) {

         value = -value;

     }

     return (float)value;

 }

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <stdbool.h>

#include <math.h>

#include "p1\_lib.h"

#define MAX\_NUMBERS 100

typedef struct NodeValue {

float value; // The actual floating-point value

F32Repr\* repr; // The IEEE 754 representation of the value

float repr\_value; // The value converted back from the IEEE 754 representation

} NodeValue;

// Define the structure for a linked list node

typedef struct Node {

NodeValue\* value; // Pointer to the value stored in the node

struct Node\* next; // Pointer to the next node in the list

} Node;

/\*\*

\* Generate a random floating-point number in the range [-100, 100].

\*

\* @return A random floating-point number.

\*/

float generate\_random() {

return ((float)rand() / (float)RAND\_MAX) \* 200.0f - 100.0f;

}

/\*\*

\* Insert a new node with the given value at the head of the linked list.

\*

\* @param h Pointer to the head of the linked list.

\* @param t Pointer to the tail of the linked list.

\* @param value Pointer to the value to be stored in the new node.

\*/

void insert\_node(Node\*\* h, Node\*\* t, NodeValue\* value) {

// Create a new node

Node\* temp = (Node\*)malloc(sizeof(Node));

temp->value = value;

temp->next = NULL;

// If the list is empty

if (\*h == NULL) {

\*h = temp;

\*t = temp;

} else {

// If the list is not empty, insert the new node at the head

temp->next = \*h;

\*h = temp;

}

}

/\*\*

\* Print the linked list and write its contents to a file or standard output.

\*

\* @param h Pointer to the head of the linked list.

\* @param filename The name of the file to write to, or NULL for standard output.

\*/

void fprint\_list(Node\* h, const char\* filename) {

FILE\* output;

if (filename != NULL) {

output = fopen(filename, "w");

if (output == NULL) {

perror("Failed to create file");

exit(EXIT\_FAILURE);

}

} else {

output = stdout;

}

if (h == NULL) {

fprintf(output, "The list is empty.\n");

} else {

fprintf(output, "Values in the list are:\n");

Node\* current = h;

while (current != NULL) {

fprintf(output, "%.6f\nsign %s exponent %s mantissa %s\n%.6f\n",

current->value->value,

current->value->repr->sign,

current->value->repr->exponent,

current->value->repr->mantissa,

current->value->repr\_value

);

current = current->next;

}

fprintf(output, "\n");

}

if (filename != NULL) {

fclose(output);

}

}

/\*\*

\* Free the memory allocated for the linked list.

\*

\* @param head Pointer to the head of the linked list.

\*/

void free\_list(Node\* head) {

Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp->value->repr); // Free the IEEE 754 representation

free(temp->value); // Free the NodeValue structure

free(temp); // Free the Node structure

}

}

/\*\*

\* Main function to generate random floating-point numbers, convert them to IEEE 754 format,

\* and store them in linked lists based on their values.

\*

\* @param argc Number of command-line arguments.

\* @param argv Command-line arguments.

\* @return Exit status.

\*/

int main(int argc, char\* argv[]) {

if (argc != 3) {

fprintf(stderr, "Usage: %s <number\_of\_positive> <file\_ext>\n", argv[0]);

return EXIT\_FAILURE;

}

// Parse command-line arguments

unsigned int num\_positive = atoi(argv[1]);

char file\_ext[100];

strcpy(file\_ext, argv[2]);

// Initialize the random number generator

srand((unsigned int)time(NULL));

// Initialize pointers for the linked lists

Node\* pos = NULL; // List for positive numbers >= 1

Node\* pos\_tail = NULL;

Node\* pos\_frac = NULL; // List for positive fractions < 1

Node\* pos\_frac\_tail = NULL;

Node\* neg = NULL; // List for negative numbers <= -1

Node\* neg\_tail = NULL;

Node\* neg\_frac = NULL; // List for negative fractions > -1

Node\* neg\_frac\_tail = NULL;

unsigned int count = 0;

while (count < num\_positive) {

float random\_value = generate\_random();

F32Repr\* repr = (F32Repr\*)malloc(sizeof(F32Repr));

\*repr = float\_to\_IEEE754((long double) random\_value); // Convert to IEEE 754 format

NodeValue\* node\_value = (NodeValue\*)malloc(sizeof(NodeValue));

node\_value->value = random\_value;

node\_value->repr = repr;

node\_value->repr\_value = IEEE754\_to\_float(\*repr); // Convert back to float

// Check for conversion error

if (fabsf(node\_value->repr\_value - random\_value) > 0.0001) {

fprintf(stderr, "Error: Wrong conversion from ieee754 to float.\n");

return EXIT\_FAILURE;

}

// Insert the value into the appropriate list based on its range

if (random\_value >= 1.0) {

insert\_node(&pos, &pos\_tail, node\_value);

count++;

} else if (random\_value > 0.0 && random\_value < 1.0) {

insert\_node(&pos\_frac, &pos\_frac\_tail, node\_value);

} else if (random\_value > -1.0 && random\_value < 0.0) {

insert\_node(&neg\_frac, &neg\_frac\_tail, node\_value);

} else if (random\_value <= -1.0) {

insert\_node(&neg, &neg\_tail, node\_value);

} else {

fprintf(stderr, "Unexpected value generated: %.2f\n", random\_value);

return EXIT\_FAILURE;

}

}

// Print and save the linked lists to files

fprint\_list(pos, strcat(strcpy((char[100]){}, "pos."), file\_ext)); // Positive numbers >= 1

fprint\_list(pos\_frac, strcat(strcpy((char[100]){}, "pos\_frac."), file\_ext)); // Positive fractions < 1

fprint\_list(neg, strcat(strcpy((char[100]){}, "neg."), file\_ext)); // Negative numbers <= -1

fprint\_list(neg\_frac, strcat(strcpy((char[100]){}, "neg\_frac."), file\_ext)); // Negative fractions > -1

// Free the memory allocated for the linked lists

free\_list(pos);

free\_list(pos\_frac);

free\_list(neg);

free\_list(neg\_frac);

return EXIT\_SUCCESS;

}