Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Звіт
з лабораторної роботи № 3
на тему:

«Стековий процесор» та «Представлення даних у форматі чисел з плаваючою точкою»

Варіант 12

Студента другого курсу
групи К-23(2)
Міщука Романа Андрійовича
Факультету комп’ютерних наук
та кібернетики

Київ – 2022

1. Постановка задачі

1.1. Задача

Необхідно розробити програму представлення введеного у діалозі десяткового числа у експоненціальній формі у визначеного варіантом форматі ЧПТ. Вимоги до програми:

1. ЧПТ має представлятися з побітною розшифровкою і у зручному для аналізу вигляді (окремо знак мантиси, окремо біти характеристики, окремо неявний біт і окремо біти мантиси).
2. ЧПТ має представлятися, якщо це можливо, у нормалізованій формі (для ненормалізованого представлення використовувати рекомендації IEEE 754).
3. При запуску програми вона має зразу вивести ряд стандартних представлень ЧПТ та їх 10-еквівалентів, а потім запропонувати користувачу у діалозі ввести 10-число і вивести його у форматі ЧПТ. Для 10-числа має бути такий синтаксис: ±ц,ц...цЕ±ц...ц. Знак “+” може опускатися та знак експоненти може бути як у верхньому, так і у нижньому регістрах.
4. Стандартними представленнями (у вигляді форматного представлення ЧПТ та його 10-еквіваленту) мають бути такі значення:
    a. мінімальне за абсолютною величиною ненульове представлення;
    b. максимальне додатнє представлення;
    c. мінімальне від’ємне представлення;
    d. число +1,0E0;
    e. значення +∞;
    f. значення -∞;
    g. будь-який варіант для ненормалізованого ЧПТ;
    h. будь-який варіант для NaN-значення.

1.2. Задачі варіанту

Індивідуальний варіант завдання: 12.

1) Для представлення чисел з плаваючою крапкою у пам’яті відводиться 43 біти: 11 для експоненти, 31 для мантиси та 1 для знаку числа.
2) Після написання імітаційної моделі, на її основі необхідно реалізувати програму, яка б рахувала значення функції F(a,b)=a*b/(b+2.4).
3) Стек складається із 8 регістрів, кожен з яких зберігає число з плаваючою точкою.

1.3. Рекомендації щодо виконання роботи

ЧПТ має представлятися згідно рекомендацій IEEE 754, зокрема:

- нормалізована мантиса має представлення 1,bbb...b (де одиниця цілої частини є неявним бітом)
- нульове значення ЧПТ має всі нулі у мантисі та характеристиці;
- якщо у полі мантиси всі нулі, а у характеристиці всі одиниці, то це є +∞ чи -∞ у залежності від знаку числа;
- ненормалізоване представлення має нульову характеристику та ненульову мантису; але справжнє значення порядку при цьому має дорівнювати мінімальному для формату значенню; неявний біт приймається рівним нулю;
- якщо характеристика із одиниць та ненульова мантиса, то це NaN-значення (незвичайна числова величина).

2. Реалізація

2.1. Регістри та пам’ять

Модель використовує 5 регістрів стану виконання програми та стек, що складається із 8 регістрів плаваючої точки. При цьому, стек реалізовано модульним чином: при видаленні якого-небудь елементу, вказівних на вершину просто зрушується назад; якщо при додаванні чергового елементу відбувається вихід за рамки стеку, значення записується в найнижчий регістр, таким чином здійснюючи цикл. Таке рішення прийнято зважаючи на покращення швидкодії та уникнення помилок виконання. Перелік регістрів стану виконання:

- **`IR`** – регістр команди, що наразі виконується (відображається у вигляді рядка команди);
- **`PC`** – лічильник команд (порядковий номер рядка команди) (відображається у десятковому вигляді);
- **`TC`** – лічильник тактів (відображається у десятковому вигляді);
- **`RS`** – регістр стану (відображається у двійковому вигляді).
- **`SL`** – регістр, що вказує на індекс вершини стеку(відображається у двійковому вигляді).

2.2. Команди

Усього було реалізовано 4 команди для роботи із стеком та 4 арифметичні команди:

Команди для роботи із пам’яттю:

- **`push`** `const` – додавання нового елементу на вершину стеку;

- **`pop`** – видалення найвищого елементу стеку;
- **`dup`** – дублювання найвищого елементу стеку;
- **`rev`** – реверсування двох найвищих елементів стеку.

Різновиди арифметичних команд:

- **`add, sub, mul, div`** – виконує відповідну арифметичну операцію, та записує результат у передостанній елемент стеку. При цьому лівим операндом команди виступає передостанній елемент, а правим – останній (найвищий).

3. Демонстрація роботи

3.1. Програма

Для демонстрації роботи була написана невеличка програма, яка рахує значення виразу F(a,b)=a*b/(b+2.4). Вона має ось такий вигляд:

```
# F(a,b)=a*b/(b+2.4)
#
# a = 12.5
# b = -7.81E1

push -7.81E1
dup
push 12.5
mul
rev
push 2.4
add
div
```

При цьому в зазначеному коді використовуються більшість із реалізованих команд.

3.2. Результат роботи в системі

Результат роботи програми під час запуску в головній системі комп’ютера:

```
Double demo:

0111111111100000000000000000000000000000000000 8.98847e+307
1111111111101111111111111111111111Double demo:

0111111111100000000000000000000000000000000000 8.98847e+307
1111111111101111111111111111111111111111111111 -1.79769e+308
0111111111100000000000000000000000000000000000 8.98847e+307
1111111111101111111111111111111111111111111111 -1.79769e+308
0011111111110000000000000000000000000000000000 1
0111111111110000000000000000000000000000000000 inf
1111111111110000000000000000000000000000000000 -inf
0100000000101010000000000000000000000000000000 5
0111111111110000000000000000000000000000000001 nan(snan)

Loading and starting the program:
Total program length: 8 lines

Starting debug:
-------------------------------------
IR = push 1100000001010011100001100110011001100110011 -78.1

Stack state:
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
 0000000000000000000000000000000000000000000000 0
>0000000000000000000000000000000000000000000000 0
```

```
PC = 0
TC = 0
RS = 00000000
-------------------------------------
IR = push 11000000010100111000011001100110011001100110011 -78.1

Stack state:
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
>11000000010100111000011001100110011001100110011 -78.1

PC = 0
TC = 1
RS = 00000001
LS = 0
-------------------------------------

-------------------------------------
IR = dup

Stack state:
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
>11000000010100111000011001100110011001100110011 -78.1

PC = 1
TC = 0
RS = 00000001
LS = 0

-------------------------------------
IR = dup

Stack state:
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
>11000000010100111000011001100110011001100110011 -78.1
 11000000010100111000011001100110011001100110011 -78.1

PC = 1
TC = 1
RS = 00000001
LS = 1
-------------------------------------

-------------------------------------
IR = push 01000000001010010000000000000000000000000000000 12.5

Stack state:
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000 0
```

```
 00000000000000000000000000000000000000000000000000000000000000000 0
>11000000010100111000011001100110011001100110011 -78.1
 11000000010100111000011001100110011001100110011 -78.1

PC = 2
TC = 0
RS = 00000001
LS = 1

-------------------------------------
IR = push 0100000000101001000000000000000000000000000000000000000000000000 12.5

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
>01000000001010010000000000000000000000000000000000000000000000000 12.5
 11000000010100111000011001100110011001100110011 -78.1
 11000000010100111000011001100110011001100110011 -78.1

PC = 2
TC = 1
RS = 00000000
LS = 2
-------------------------------------

-------------------------------------
IR = mul

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
>01000000001010010000000000000000000000000000000000000000000000000 12.5
 11000000010100111000011001100110011001100110011 -78.1
 11000000010100111000011001100110011001100110011 -78.1

PC = 3
TC = 0
RS = 00000000
LS = 2

-------------------------------------
IR = mul

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 01000000001010010000000000000000000000000000000000000000000000000 12.5
>11000000100011101000000111111111111111111111111 -976.25
 11000000010100111000011001100110011001100110011 -78.1

PC = 3
TC = 1
RS = 00000001
LS = 1
-------------------------------------

-------------------------------------
IR = rev
```

```
Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 01000000001010010000000000000000000000000000000000000000000000000 12.5
>11000000100011101000000111111111111111111111111111111111111111111 -976.25
 11000000010100111000011001100110011001100110011 -78.1

PC = 4
TC = 0
RS = 00000001
LS = 1

-----------------------------------
IR = rev

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 01000000001010010000000000000000000000000000000000000000000000000 12.5
>11000000010100111000011001100110011001100110011 -78.1
 11000000100011101000000111111111111111111111111111111111111111111 -976.25

PC = 4
TC = 1
RS = 00000001
LS = 1
-----------------------------------

-----------------------------------
IR = push 0100000000000011001100110011001100110011001100110011001 2.4

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 01000000001010010000000000000000000000000000000000000000000000000 12.5
>11000000010100111000011001100110011001100110011 -78.1
 11000000100011101000000111111111111111111111111111111111111111111 -976.25

PC = 5
TC = 0
RS = 00000001
LS = 1

-----------------------------------
IR = push 0100000000000011001100110011001100110011001100110011001 2.4

Stack state:
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000000000000000000 0
>01000000000000110011001100110011001100110011001100110011001 2.4
 11000000010100111000011001100110011001100110011 -78.1
 11000000100011101000000111111111111111111111111111111111111111111 -976.25

PC = 5
TC = 1
RS = 00000000
```

```
LS = 2
-------------------------------------

-------------------------------------
IR = add

Stack state:
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
>01000000000000110011001100110011001100110011001 2.4
 11000000010100111000011001100110011001100110011 -78.1
 11000000100011101000000111111111111111111111111 -976.25

PC = 6
TC = 0
RS = 00000000
LS = 2

-------------------------------------
IR = add

Stack state:
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 01000000000000110011001100110011001100110011001 2.4
>11000000010100101110110011001100110011001100110 -75.7
 11000000100011101000000111111111111111111111111 -976.25

PC = 6
TC = 1
RS = 00000001
LS = 1
-------------------------------------

-------------------------------------
IR = div

Stack state:
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 01000000000000110011001100110011001100110011001 2.4
>11000000010100101110110011001100110011001100110 -75.7
 11000000100011101000000111111111111111111111111 -976.25

PC = 7
TC = 0
RS = 00000001
LS = 1

-------------------------------------
IR = div

Stack state:
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 00000000000000000000000000000000000000000000000000 0
 01000000000000110011001100110011001100110011001 2.4
```

```
 11000000010100101110110011001100110011001100110 -75.7
>01000000001010011100101011100111111111101010 12.8963

PC = 7
TC = 1
RS = 00000000
LS = 0
-----------------------------------
```

4. Код програми

4.1. program.txt

```
# Computing (a % c) % (c % b)
#
# a = 32048
# b = 287
# c = 7298

# store a, b, c
set_c 32048
dump_ca 0
set_c 287
dump_ca 1
set_c 7298
dump_ca 2

# c % b
load_ca 2
mod_ca 1
dump_ca 3

# a % c
load_ca 0
mod_ca 2

# result
mod_ca 3
```

4.2. LAB3.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

#include "bit_type.h"
#include "processor.h"

using namespace std;

vector<string> read_file(string filename = "program.txt")
{
    ifstream file(filename, ios_base::in);
    vector<string> ret{};
    string buff;
    while (getline(file, buff))
    {
        if (buff.empty() || buff[0] == '#' || buff[0] == '\r') continue;
        ret.push_back(buff);
        //cout << buff << "\n";
    }
    file.close();
    return ret;
}

int main()
{
    bit_type buff;
    cout << "Double demo:\n";

    buff.set_abs_min();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_max();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_min();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_double(1);
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_infty();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_minfty();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_double(5);
    bit_type non_norm = buff.to_non_normalized();
    cout << non_norm.get_bit_string() << " " << buff.get_double() << "\n";

    buff.set_nan();
    cout << buff.get_bit_string() << " " << buff.get_double() << "\n";


    cout << "\nLoading and starting the program:\n";
    auto program = read_file();
    processor p(program);
    cout << p.get_program_info() << "\nStarting debug: \n";

    while (p.do_tick())
    {
        cout << p.get_state();
        p.end_tick();
```

```
            cin.get();
        }
    }
```

4.3. processor.h

```
#pragma once
#include <vector>
#include <stack>
#include <string>
#include <sstream>
#include <map>

#include "bit_type.h"
#include "mod_stack.h"

/// <summary>
/// Stack-processor model
/// </summary>
class processor
{
private:
    static constexpr short stack_size = 8;
    mod_stack<bit_type> stack;
    static const unsigned int tc_num = 2;
    unsigned int PC, TC, RS, SL;

    struct command
    {
        enum name_t : unsigned int
        {
            push, pop,
              add, sub, mul, div,
            dup, rev,
            COUNT
        };
        static std::map<name_t, const char*> name_map;

        name_t name;
        bit_type val;

        std::string str() const;

        bool has_val() const;
    } IR;

    std::vector<std::string> program;

public:
    processor(std::vector<std::string> program, size_t ram_size = 10);

    bool do_tick();
    void end_tick();

    std::string get_state() const;

    std::string get_program_info() const;
};
```

4.4. processor.cpp

```
#include "processor.h"

#include <iomanip>
#include <algorithm>

using namespace std;

std::map<processor::command::name_t, const char*> processor::command::name_map{
	{push, "push"}, {pop, "pop"}, {add, "add"}, {sub, "sub"}, {mul, "mul"}, {div, "div"},
{dup, "dup"}, {rev, "rev"}
};

std::string processor::command::str() const
{
	stringstream ret;
	ret << name_map[name];
	if(has_val()) ret << " " << val.get_bit_string() << " " << val.get_double();
	return ret.str();
}

bool processor::command::has_val() const
{
	return name == push;
}

processor::processor(vector<string> program, const size_t ram_size):
	PC(0), TC(0), RS(0), IR(), SL(), stack(stack_size)
{
	this->program = move(program);
}

bool processor::do_tick()
{
	if (PC >= program.size()) return false;

	if (TC == 0)
	{
		// Parsing command data from string
		stringstream s;
		s << program[PC];

		string str_name;
		s >> str_name;
		unsigned int name = 0;
		while (
			strcmp(str_name.c_str(), command::name_map[(command::name_t)name]) != 0
&&
			name < command::name_t::COUNT
			) name++;

		IR.name = (command::name_t)name;
		if (IR.has_val())
		{
			double buff;
			s >> buff;
			IR.val.set_double(buff);
		}
	}
	else if (TC == 1)
	{
		switch (IR.name)
		{
		case command::push:
			stack.push(IR.val);
			break;
```

```
                case command::pop:
                        stack.pop();
                        break;
                case command::add:
                {
                        bit_type a = stack.pop();
                        auto b = &stack.top();
                        b->set_double(b->get_double() + a.get_double());
                        break;
                }
                case command::sub:
                {
                        bit_type a = stack.pop();
                        auto b = &stack.top();
                        b->set_double(b->get_double() - a.get_double());
                        break;
                }
                case command::mul:
                {
                        bit_type a = stack.pop();
                        auto b = &stack.top();
                        b->set_double(b->get_double() * a.get_double());
                        break;
                }
                case command::div:
                {
                        bit_type a = stack.pop();
                        auto b = &stack.top();
                        b->set_double(b->get_double() / a.get_double());
                        break;
                }
                case command::dup:
                        stack.duplicate();
                        break;
                case command::rev:
                        stack.swap();
                        break;
                }
                RS = stack.top().get_sign();
                SL = stack.get_head();
        }

        return PC < program.size();
}
void processor::end_tick()
{
        if (TC == tc_num - 1) PC++;
        TC++;
        TC %= tc_num;
}

string to_binary(unsigned int n, size_t size)
{
        stringstream s;
        while (n > 0)
        {
                s << n % 2;
                n >>= 1;
        }
        string ret = s.str();
        reverse(ret.begin(), ret.end());
        ret = ((size - ret.size() > 0) ? string(size - ret.size(), '0') : "") + ret;
        /*for (int i = bitmem::item_size - 8; i > 0; i -= 8)
                ret.insert(i, 1, '.');*/
        return ret;
}

std::string processor::get_state() const
```

```
{
	static constexpr auto delim = "-----------------------------------\n";

	stringstream ss;
	ss << delim << "IR = " << IR.str() << "\n\n";

	ss << "Stack state:\n";
	const auto stack_iter = stack.get_vector();
	for (short i = stack_size - 1; i >= 0; i--) {
		ss << (((short )SL == i) ? ">" : " ");
		ss << (*stack_iter)[i].get_bit_string() << " ";
		ss << (*stack_iter)[i].get_double() << "\n";
	}
	ss << "\n";
	ss << "PC = " << PC << "\n";
	ss << "TC = " << TC << "\n";
	ss << "RS = " << to_binary(RS, 8) << "\n";
	ss << "LS = " << SL << "\n";

	if (TC == tc_num - 1) ss << delim;

	return ss.str();
}

std::string processor::get_program_info() const
{
	stringstream ret;
	ret << "Total program length: " << program.size() << " lines\n";
	return ret.str();
}
```

4.5. bit_type.h

```
#pragma once
#include <string>

/// <summary>
/// The memory structure:
///
///    0..0sHHHH..HHHMMMMM..MMMM
///    Direction of indexing<---
///                          3210
///
///    where
///    0 - zero bits
///    s - sign bit
///    H - exponent bit
/// M - digits bit
///
/// Special values reference:
///    s H    M       label
///    0 00h  000000h Positive zero
///    1 00h  000000h Negative zero
///    0 FFh  000000h Positive infinity
///    1 FFh  000000h Negative infinity
///    0 FFh >000000h NaN
/// </summary>
class bit_type
{
public:
       static constexpr short digits_len = 31;
       static constexpr short exponent_len = 11;
       static constexpr short exponent_digits_len = exponent_len + digits_len;
       typedef unsigned long long mem_t;

private:
       mem_t data;

       static inline mem_t get_mask(short start, short stop);

public:
       bit_type() : data(0){}

       mem_t get_range(short stop) const;
       mem_t get_range(short start, short stop) const;

       void set_range(short stop, mem_t value);
       void set_range(short start, short stop, mem_t value);

       mem_t get_sign() const;
       mem_t get_exponent() const;
       mem_t get_digits() const;

       void set_sign(mem_t value);
       void set_exponent(mem_t value);
       void set_digits(mem_t value);

       double get_double() const;
       void set_double(double value);
       void set_double_from_str(std::string str);

       void set_max();
       void set_min();
       void set_abs_min();

       void set_infty();
       void set_minfty();
       void set_nan();
```

```
        std::string get_bit_string() const;

        bit_type to_non_normalized() const;
};
```

4.6. bit_type.cpp

```
#include "bit_type.h"
#include <sstream>
#include <algorithm>

bit_type::mem_t bit_type::get_mask(short start, short stop)
{
	return ((mem_t)1 << stop) - ((mem_t)1 << start);
}

bit_type::mem_t bit_type::get_range(short stop) const
{
	return get_range(0, stop);
}

bit_type::mem_t bit_type::get_range(short start, short stop) const
{
	return (data & get_mask(start, stop)) >> start;
}

void bit_type::set_range(short stop, mem_t value)
{
	set_range(0, stop, value);
}

void bit_type::set_range(short start, short stop, mem_t value)
{
	mem_t mask = get_mask(start, stop);
	data = (data & ~mask) | ((value << start) & mask);
}

bit_type::mem_t bit_type::get_sign() const
{
	return get_range(exponent_digits_len, exponent_digits_len + 1);
}

bit_type::mem_t bit_type::get_digits() const
{
	return get_range(digits_len);
}

void bit_type::set_sign(mem_t value)
{
	set_range(exponent_digits_len, exponent_digits_len + 1, value);
}

void bit_type::set_exponent(mem_t value)
{
	set_range(digits_len, exponent_digits_len, value);
}

void bit_type::set_digits(mem_t value)
{
	set_range(digits_len, value);
}

bit_type::mem_t bit_type::get_exponent() const
{
	return get_range(digits_len, exponent_digits_len);
}


union mem_t_to_double
{
	bit_type::mem_t bits;
	double floating;
```

```
};

double bit_type::get_double() const
{
	mem_t digits = get_digits() & get_mask(0, 52); // 52
	mem_t exponent = get_exponent() & get_mask(0, 11); // 11
	mem_t sign = get_sign(); // 1

	// TODO: in case, exponent size changes, we need to shift sign bit

	mem_t_to_double ret{ (sign << 63) | (exponent << 52) | (digits << (52 - digits_len))
};
	return ret.floating;
}

void bit_type::set_double(double value)
{
	mem_t_to_double buff{};
	buff.floating = value;

	// TODO: this thing is adjusted only to the variant's exponent size
	set_range(digits_len, exponent_digits_len + 1, buff.bits >> 52);
	set_range(digits_len, (buff.bits & get_mask(0, 52)) >> (52 - digits_len));
}

void bit_type::set_double_from_str(std::string str)
{ set_double(std::stod(str)); }

void bit_type::set_max()
{
	set_digits((1 << digits_len) - 1);
	set_exponent((1 << exponent_len) - 2);
	set_digits(0);
}

void bit_type::set_min()
{
	set_digits((1 << digits_len) - 1);
	set_exponent((1 << exponent_len) - 2);
	set_sign(1);
}

void bit_type::set_abs_min()
{
	set_digits(1);
	set_exponent(0);
	set_sign(0);
}

void bit_type::set_infty()
{
	set_digits(0);
	set_exponent((1 << exponent_len) - 1);
	set_sign(0);
}

void bit_type::set_minfty()
{
	set_digits(0);
	set_exponent((1 << exponent_len) - 1);
	set_sign(1);
}

void bit_type::set_nan()
{
	set_digits(1);
	set_exponent((1 << exponent_len) - 1);
	set_sign(0);
```

```
}

std::string to_bite_string(bit_type::mem_t n, short length)
{
	std::stringstream s;
	for (short i = 0; i < length; i++)
	{
		s << n % 2;
		n >>= 1;
	}
	std::string ret = s.str();
	std::reverse(ret.begin(), ret.end());
	return ret;
}

std::string bit_type::get_bit_string() const
{
	return to_bite_string(data, exponent_digits_len + 1);
}

bit_type bit_type::to_non_normalized() const
{
	bit_type ret = *this;
	ret.set_digits((1 << (digits_len - 1)) + (get_digits() >> 1));
	ret.set_exponent(((unsigned) get_exponent()) + 1);
	return ret;
}
```

4.7. mod_stack.h

```
#pragma once
#include <vector>

using namespace std;

template<typename T>
class mod_stack: vector<T>
{
private:
	size_t size, head;

public:
	mod_stack(size_t size);

	void push(T el);
	T pop();

	T& top();

	void swap();
	void duplicate();

	size_t get_head() const;

	const vector<T>* get_vector() const;
};

template <typename T>
mod_stack<T>::mod_stack(size_t size) : vector<T>(size)
{
	this->size = size;
	this->head = size - 1;
}

template <typename T>
void mod_stack<T>::push(T el)
{
	head = (head + 1) % size;
	(*this)[head] = el;
}

template <typename T>
T mod_stack<T>::pop()
{
	size_t buff_head = head;
	head = (head - 1) % size;
	return (*this)[buff_head];
}

template <typename T>
T& mod_stack<T>::top()
{
	return (*this)[head];
}

template <typename T>
void mod_stack<T>::swap()
{
	std::swap((*this)[head], (*this)[(head - 1) % size]);
}

template <typename T>
void mod_stack<T>::duplicate()
{
	push(top());
```

```
}

template <typename T>
size_t mod_stack<T>::get_head() const
{
	return head;
}

template <typename T>
const vector<T>* mod_stack<T>::get_vector() const
{
	return this;
}
```