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

# Постановка задачі

## 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:
   1. мінімальне за абсолютною величиною ненульове представлення;
   2. максимальне додатнє представлення;
   3. мінімальне від’ємне представлення;
   4. число +1,0Е0;
   5. значення +∞;
   6. значення -∞;
   7. будь-який варіант для ненормалізованого ЧПТ;
   8. будь-який варіант для NaN-значення.

## 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.

## 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).

# Реалізація

## 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).

## 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).

# Демонстрація роботи

## 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.

## Результат роботи в системі

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

Double demo:

0111111111100000000000000000000000000000000 8.98847e+307

1111111111101111111111111111111111Double demo:

0111111111100000000000000000000000000000000 8.98847e+307

1111111111101111111111111111111111111111111 -1.79769e+308 0111111111100000000000000000000000000000000 8.98847e+307

1111111111101111111111111111111111111111111 -1.79769e+308

0011111111110000000000000000000000000000000 1

0111111111110000000000000000000000000000000 inf

1111111111110000000000000000000000000000000 -inf

0100000000101010000000000000000000000000000 5

0111111111110000000000000000000000000000001 nan(snan)

Loading and starting the program:

Total program length: 8 lines

Starting debug:

------------------------------------

IR = push 1100000001010011100001100110011001100110011 -78.1

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>0000000000000000000000000000000000000000000 0

PC = 0

TC = 0

RS = 00000000

------------------------------------

IR = push 1100000001010011100001100110011001100110011 -78.1

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>1100000001010011100001100110011001100110011 -78.1

PC = 0

TC = 1

RS = 00000001

LS = 0

------------------------------------

------------------------------------

IR = dup

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>1100000001010011100001100110011001100110011 -78.1

PC = 1

TC = 0

RS = 00000001

LS = 0

------------------------------------

IR = dup

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>1100000001010011100001100110011001100110011 -78.1

1100000001010011100001100110011001100110011 -78.1

PC = 1

TC = 1

RS = 00000001

LS = 1

------------------------------------

------------------------------------

IR = push 0100000000101001000000000000000000000000000 12.5

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>1100000001010011100001100110011001100110011 -78.1

1100000001010011100001100110011001100110011 -78.1

PC = 2

TC = 0

RS = 00000001

LS = 1

------------------------------------

IR = push 0100000000101001000000000000000000000000000 12.5

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>0100000000101001000000000000000000000000000 12.5

1100000001010011100001100110011001100110011 -78.1

1100000001010011100001100110011001100110011 -78.1

PC = 2

TC = 1

RS = 00000000

LS = 2

------------------------------------

------------------------------------

IR = mul

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>0100000000101001000000000000000000000000000 12.5

1100000001010011100001100110011001100110011 -78.1

1100000001010011100001100110011001100110011 -78.1

PC = 3

TC = 0

RS = 00000000

LS = 2

------------------------------------

IR = mul

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000101001000000000000000000000000000 12.5

>1100000010001110100000011111111111111111111 -976.25

1100000001010011100001100110011001100110011 -78.1

PC = 3

TC = 1

RS = 00000001

LS = 1

------------------------------------

------------------------------------

IR = rev

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000101001000000000000000000000000000 12.5

>1100000010001110100000011111111111111111111 -976.25

1100000001010011100001100110011001100110011 -78.1

PC = 4

TC = 0

RS = 00000001

LS = 1

------------------------------------

IR = rev

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000101001000000000000000000000000000 12.5

>1100000001010011100001100110011001100110011 -78.1

1100000010001110100000011111111111111111111 -976.25

PC = 4

TC = 1

RS = 00000001

LS = 1

------------------------------------

------------------------------------

IR = push 0100000000000011001100110011001100110011001 2.4

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000101001000000000000000000000000000 12.5

>1100000001010011100001100110011001100110011 -78.1

1100000010001110100000011111111111111111111 -976.25

PC = 5

TC = 0

RS = 00000001

LS = 1

------------------------------------

IR = push 0100000000000011001100110011001100110011001 2.4

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>0100000000000011001100110011001100110011001 2.4

1100000001010011100001100110011001100110011 -78.1

1100000010001110100000011111111111111111111 -976.25

PC = 5

TC = 1

RS = 00000000

LS = 2

------------------------------------

------------------------------------

IR = add

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

>0100000000000011001100110011001100110011001 2.4

1100000001010011100001100110011001100110011 -78.1

1100000010001110100000011111111111111111111 -976.25

PC = 6

TC = 0

RS = 00000000

LS = 2

------------------------------------

IR = add

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000000011001100110011001100110011001 2.4

>1100000001010010111011001100110011001100110 -75.7

1100000010001110100000011111111111111111111 -976.25

PC = 6

TC = 1

RS = 00000001

LS = 1

------------------------------------

------------------------------------

IR = div

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000000011001100110011001100110011001 2.4

>1100000001010010111011001100110011001100110 -75.7

1100000010001110100000011111111111111111111 -976.25

PC = 7

TC = 0

RS = 00000001

LS = 1

------------------------------------

IR = div

Stack state:

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0000000000000000000000000000000000000000000 0

0100000000000011001100110011001100110011001 2.4

1100000001010010111011001100110011001100110 -75.7

>0100000000101001110010101110011111111101010 12.8963

PC = 7

TC = 1

RS = 00000000

LS = 0

------------------------------------

# Код програми

## 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

## LAB3.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();

}

}

## 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;

};

## 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();

}

## 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;

};

## 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;

}

## 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;

}