|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Практическое задание № 1 | | |
| по дисциплине «Информатика» | | |
| **Представление данных в ЭВМ** | | |
|  | | |
|  | Бригада 3 | Богданович павел |
| Группа ПМ-01 | чирков артём |
| Вариант 3 | самсонов семён |
|  |  |
|  |  |
|  |  |
| Преподаватель | Тракимус Юрий Викторович |
|  |  |
| Новосибирск, 2021 | | |

1. **Задание**

Написать программу для задания № 6 и №7.

1. **Программа**

Файл **Converters\_binary-hex.h**

#pragma once

#include <string>

#include <map>

namespace converters

{

const std::map<std::string, std::string> radix2\_to\_radix16\_map = {

{"0000", "0"},

{"0001", "1"},

{"0010", "2"},

{"0011", "3"},

{"0100", "4"},

{"0101", "5"},

{"0110", "6"},

{"0111", "7"},

{"1000", "8"},

{"1001", "9"},

{"1010", "A"},

{"1011", "B"},

{"1100", "C"},

{"1101", "D"},

{"1110", "E"},

{"1111", "F"},

};

const std::map<std::string, std::string> radix16\_to\_radix2\_map = {

{"0", "0000"},

{"1", "0001"},

{"2", "0010"},

{"3", "0011"},

{"4", "0100"},

{"5", "0101"},

{"6", "0110"},

{"7", "0111"},

{"8", "1000"},

{"9", "1001"},

{"A", "1010"},

{"B", "1011"},

{"C", "1100"},

{"D", "1101"},

{"E", "1110"},

{"F", "1111"},

};

inline std::string binary\_to\_hex(std::string binary\_number)

{

if (binary\_number.size() % 4 != 0)

binary\_number = std::string(4 - binary\_number.size() % 4, '0') + binary\_number;

std::string hex\_number;

for (int i = 0; i < binary\_number.size(); i += 4)

{

hex\_number += radix2\_to\_radix16\_map.find(binary\_number.substr(i, 4))->second;

}

return hex\_number;

}

inline std::string hex\_to\_binary(const std::string& hex\_number)

{

std::string binary\_number;

for (const auto elem : hex\_number)

{

binary\_number += radix16\_to\_radix2\_map.find(std::string(1, toupper(elem)))->second;

}

return binary\_number;

}

inline bool is\_float\_double\_hex(const std::string& number)

{

if (number.size() % 8 <= 1)

{

return true;

}

return false;

}

}

Файл **Converters\_machinery.h**

#pragma once

#include <string>

#include <map>

#include "Converters\_binary-hex.h"

namespace converters

{

template <typename T = double>

class computer\_radix10\_number

{

private:

std::string hex\_number\_;

T decimal\_number\_ = 0.0;

bool is\_negative\_ = false;

static const std::string inf\_nan\_exponent;

static const std::string zero\_exponent;

static const std::string plus\_inf;

static const std::string minus\_inf;

static const std::string nan;

static const int exponent\_size;

static const int mantissa\_size;

T \_translate\_to\_decimal(std::string hex\_number)

{

if (hex\_number.size() != sizeof(T) \* 2)

{

throw std::exception((std::string("Error: wrong length of computer number: length is ") + std::to\_string(hex\_number.size())).c\_str());

}

std::string binary\_number = hex\_to\_binary(hex\_number);

is\_negative\_ = binary\_number [ 0 ] == '1' ? true : false;

binary\_number = binary\_number.substr(1);

std::string str\_exponent = binary\_number.substr(0, 11);

if (str\_exponent == inf\_nan\_exponent)

{

if (binary\_number.substr(12).find('1') != std::string::npos)

{

decimal\_number\_ = NAN;

}

else

{

decimal\_number\_ = is\_negative\_ ? -INFINITY : INFINITY;

}

}

else

{

// Translate exponent from binary to decimal

int shifted\_int\_exp = 0;

for (int i = str\_exponent.size() - 1; i >= 0; --i)

{

if (str\_exponent [ i ] == '1')

{

shifted\_int\_exp += pow(2, str\_exponent.size() - i - 1);

}

}

// Substract shift (2^(exp\_size-1) + 1)

int int\_exp = shifted\_int\_exp - pow(2, str\_exponent.size() - 1) + 1;

// Get binary mantissa with hidden bit and normalized to number without fraction part

std::string str\_mantissa = (shifted\_int\_exp == 0 ? "0" : "1") + binary\_number.substr(exponent\_size);

for (int i = str\_mantissa.size() - 1; i >= 0; --i)

{

if (str\_mantissa [ i ] == '1')

{

decimal\_number\_ += pow(2, str\_mantissa.size() - i - 1);

}

}

decimal\_number\_ \*= pow(2, int\_exp - 52) \* (is\_negative\_ ? -1 : 1);

}

return decimal\_number\_;

}

std::string \_translate\_to\_hex(T num)

{

std::string binary\_number;

if (isnan(num))

{

binary\_number = nan;

}

else

{

is\_negative\_ = std::signbit(num);

if (isinf(num))

{

if (!is\_negative\_)

{

binary\_number = plus\_inf;

}

else

{

binary\_number = minus\_inf;

}

}

else

{

// Split the num to some parts

num = std::abs(num);

T first\_part = std::floor(num);

T second\_part = num - first\_part;

std::string first\_part\_binary;

std::string second\_part\_binary;

// Get the first part of num in radix2

int i = 0, j = 0;

while (first\_part != 0.0)

{

// If length is more than [mantissa\_size], reduce the string

if (i == mantissa\_size)

{

++j;

first\_part\_binary = first\_part\_binary.substr(1);

}

if (i != mantissa\_size) ++i;

double mod = std::fmod(first\_part, 2.0);

first\_part = (first\_part - mod) / 2.0;

if (mod == 1.0)

{

first\_part\_binary += '1';

}

else

{

first\_part\_binary += '0';

}

}

std::reverse(first\_part\_binary.begin(), first\_part\_binary.end());

// Calc shift of number, and also move the second part, if first part is empty

int shift = pow(2, exponent\_size - 1) - 1;

if (i == 0)

{

while (second\_part < 1 && shift != 0)

{

--shift;

second\_part \*= 2;

}

second\_part\_binary += '1';

second\_part -= 1;

++i;

}

else

{

shift += i - 1 + j;

}

// Translate shift part to binary code

std::string shift\_binary = "";

for (int i = 0; i < exponent\_size; ++i)

{

int mod = shift % 2;

shift /= 2;

if (mod == 1)

{

shift\_binary += '1';

}

// If first part isn't empty, calc it by i (size of first part) and j (shift of first part)

else

{

shift\_binary += '0';

}

}

std::reverse(shift\_binary.begin(), shift\_binary.end());

// Get the second part in radix2, while we got [mantissa\_size]

while (i++ < mantissa\_size)

{

second\_part \*= 2.0;

if (floor(second\_part) == 1.0)

{

second\_part\_binary += '1';

second\_part -= 1;

}

else

{

second\_part\_binary += '0';

}

}

// Finally, radix2 num (and don't forget to delete hidden bit)

binary\_number = (is\_negative\_ ? '1' : '0') + shift\_binary + (first\_part\_binary + second\_part\_binary).substr(1);

}

}

hex\_number\_ = binary\_to\_hex(binary\_number);

return hex\_number\_;

}

public:

/// <summary>

/// Default constructor

/// </summary>

computer\_radix10\_number(void)

{}

/// <summary>

/// Constructor for machinery num from computer format

/// </summary>

/// <param name="computer\_num"> - string with hex or binary formatted number</param>

computer\_radix10\_number(const std::string& computer\_num)

{

\*this = computer\_num;

}

/// <summary>

/// Constructor for machinery num from decimal format

/// </summary>

/// <param name="decimal\_num"> - number in decimal format</param>

computer\_radix10\_number(const T decimal\_num)

{

\*this = decimal\_num;

}

/// <summary>

/// Operator for initialise new number from hex or binary format

/// </summary>

/// <param name="computer\_num"> - string with hex or binary formatted number</param>

computer\_radix10\_number<T>& operator= (const std::string& computer\_num)

{

if (is\_float\_double\_hex(computer\_num))

{

hex\_number\_ = computer\_num;

}

else

{

hex\_number\_ = binary\_to\_hex(computer\_num);

}

decimal\_number\_ = \_translate\_to\_decimal(computer\_num);

return \*this;

}

/// <summary>

/// Operator for initialise new number from decimal format

/// </summary>

/// <param name="decimal\_num"> - string in decimal format</param>

computer\_radix10\_number<T>& operator= (const T decimal\_num)

{

decimal\_number\_ = decimal\_num;

\_translate\_to\_hex(decimal\_num);

return \*this;

}

computer\_radix10\_number<T>& operator= (const computer\_radix10\_number<T>& \_right)

{

computer\_radix10\_number<T> new\_num;

new\_num.hex\_number\_ = \_right.hex\_number\_;

new\_num.decimal\_number\_ = \_right.decimal\_number\_;

new\_num.is\_negative\_ = \_right.is\_negative\_;

return \_right;

}

bool operator== (const computer\_radix10\_number<T>& \_right) const

{

return decimal\_number\_ == \_right.decimal\_number\_;

}

bool operator< (const computer\_radix10\_number<T>& \_right) const

{

return decimal\_number\_ < \_right.decimal\_number\_;

}

bool operator> (const computer\_radix10\_number<T>& \_right) const

{

return decimal\_number\_ > \_right.decimal\_number\_;

}

bool operator<= (const computer\_radix10\_number<T>& \_right) const

{

return decimal\_number\_ <= \_right.decimal\_number\_;

}

bool operator>= (const computer\_radix10\_number<T>& \_right) const

{

return decimal\_number\_ >= \_right.decimal\_number\_;

}

computer\_radix10\_number<T> operator+ (const computer\_radix10\_number<T>& \_right) const

{

return computer\_radix10\_number<T>(this->decimal\_number\_ + \_right.decimal\_number\_);

}

computer\_radix10\_number<T> operator- (const computer\_radix10\_number<T>& \_right) const

{

return computer\_radix10\_number<T>(this->decimal\_number\_ - \_right.decimal\_number\_);

}

computer\_radix10\_number<T> operator\* (const computer\_radix10\_number<T>& \_right) const

{

return computer\_radix10\_number<T>(this->decimal\_number\_ \* \_right.decimal\_number\_);

}

computer\_radix10\_number<T> operator/ (const computer\_radix10\_number<T>& \_right) const

{

return computer\_radix10\_number<T>(this->decimal\_number\_ / \_right.decimal\_number\_);

}

/// <summary>

/// Get the binary-formatted number in [std::string]

/// </summary>

std::string to\_binary() const

{

return hex\_to\_binary(hex\_number\_);

}

/// <summary>

/// Get the hex-formatted number in [std::string]

/// </summary>

std::string to\_hex() const

{

return hex\_number\_;

}

/// <summary>

/// Get the decimal-formatted number in [T]

/// </summary>

T to\_decimal() const

{

return decimal\_number\_;

}

/// <summary>

/// Get the sign of number in [bool]

/// </summary>

bool is\_negative() const

{

return is\_negative\_;

}

~computer\_radix10\_number() = default;

};

// Float static constants

const std::string computer\_radix10\_number<float>::inf\_nan\_exponent = "11111111";

const std::string computer\_radix10\_number<float>::zero\_exponent = "00000000";

const std::string computer\_radix10\_number<float>::plus\_inf { "01111111100000000000000000000000" };

const std::string computer\_radix10\_number<float>::minus\_inf { "11111111100000000000000000000000" };

const std::string computer\_radix10\_number<float>::nan { "10000000000000000000000000000000" };

constexpr int computer\_radix10\_number<float>::exponent\_size = 8;

constexpr int computer\_radix10\_number<float>::mantissa\_size = 24;

// end Float static constants

// Double static constants

const std::string computer\_radix10\_number<double>::inf\_nan\_exponent = "11111111111";

const std::string computer\_radix10\_number<double>::zero\_exponent = "00000000000";

const std::string computer\_radix10\_number<double>::plus\_inf { "0111111111110000000000000000000000000000000000000000000000000000" };

const std::string computer\_radix10\_number<double>::minus\_inf { "1111111111110000000000000000000011111111111111111111111111111111" };

const std::string computer\_radix10\_number<double>::nan { "0111111110000000000000000000000000000000000000000000000000000001" };

constexpr int computer\_radix10\_number<double>::exponent\_size = 11;

constexpr int computer\_radix10\_number<double>::mantissa\_size = 53;

//end Double static constants

} // namespace Converters

Файл **Main.cpp**

#include <iostream>

#include <string>

#include "Converters\_machinery.h"

using namespace std;

double accurate\_double\_parcer(string input)

{

for (auto& elem : input)

{

if (elem == ',')

{

elem = '.';

}

else if (elem != '.' && elem != '-' && elem != '+' && elem != 'e' && (elem < '0' || elem > '9'))

{

return NAN;

}

}

return atof(input.c\_str());

}

// Enumeration for state of menu

enum class stats

{

complete,

choose,

to\_hex,

from\_hex,

};

stats \_menuChoose()

{

cout << "What you want to do:" << endl

<< " float/double to computer hexadecimal (enter [1])" << endl

<< " or computer hexadecimal to float/double (enter [2])?" << endl

<< " (Enter [0] for exit)" << endl << "> ";

int input = 0;

cin >> input;

cout << endl;

switch (input)

{

case 0:

return stats::complete;

case 1:

return stats::to\_hex;

case 2:

return stats::from\_hex;

default:

cout << "Incorrect input" << endl;

return stats::choose;

}

}

void \_toHex()

{

int type { 0 };

string num;

double d\_num;

cout << "Input number:" << endl

<< "> ";

cin >> num;

cout << endl;

d\_num = accurate\_double\_parcer(num);

cout << "Enter type of translation ([1] to float, [2] to double):" << endl

<< "> ";

cin >> type;

switch (type)

{

case 1: // float case

{

converters::computer\_radix10\_number<float> result(d\_num);

cout << "Number in radix2-system is: " << result.to\_binary() << endl

<< "Number in radix16-system is: " << result.to\_hex() << endl << endl;

}

break;

case 2: // Double case

{

converters::computer\_radix10\_number<double> result(d\_num);

cout << "Number in radix2-system is: " << result.to\_binary() << endl

<< "Number in radix16-system is: " << result.to\_hex() << endl << endl;

}

break;

default:

{

cout << "Wrong input." << endl << endl;

}

break;

}

}

void \_fromHex()

{

string inp;

cout << "Input the value in computer form:" << endl

<< "> ";

cin >> inp;

if (inp.length() == 16 /\*Length of double\*/)

{

converters::computer\_radix10\_number<double> value(inp);

cout << "Final value: " << std::scientific << value.to\_decimal() << endl;

}

else

{

converters::computer\_radix10\_number<float> value(inp);

cout << "Final value: " << std::scientific << value.to\_decimal() << endl;

}

}

int menu()

{

auto stat = stats::choose;

while (stat != stats::complete)

{

switch (stat)

{

case stats::choose:

stat = \_menuChoose();

break;

case stats::to\_hex:

\_toHex();

stat = stats::choose;

break;

case stats::from\_hex:

\_fromHex();

stat = stats::choose;

break;

}

}

return 0;

}

int main()

{

try

{

menu();

}

catch (std::exception e)

{

cout << e.what() << endl;

}

return 0;

}

1. **Тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Выходные данные |
| 1 | –187.25  –146.836  C067680000000000  C312D604 | C067680000000000  C312D604  -1.872500E+02  -1.468360E+02 | C067680000000000  C312D604  -1.872500E+02  -1.468360E+02 |
| 2 | 465.625  407.555  407D1A0000000000 43CBC70A | 407D1A0000000000 43CBC70A  4.656250E+02  4.075550E+02 | 407D1A0000000000 43CBC70A  4.656250E+02  4.075550E+02 |
| 3 | -310.375  -169.3125  C073660000000000  C3295000 | C073660000000000  C3295000  -3.103750e+02  -1.693125e+02 | C073660000000000  C3295000  -3.103750e+02  -1.693125e+02 |
| 4 | 126.609375  29.6875  405FA70000000000  41ED8000 | 405FA70000000000  41ED8000  1.266094e+02  2.968750e+01 | 405FA70000000000  41ED8000  1.266094e+02  2.968750e+01 |

Таблица краевых значений

Для float:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Выходные данные |
| 1 | FFFFFFFF | -nan | -nan |
| 2 | FF800001 | -nan | -nan |
| 3 | FF800000 | -inf | -inf |
| 4 | FF7FFFFF | -3,40282347e+38 | -3.402823e+38 |
| 5 | 80800000 | -1,17549435e-38 | -1.175494e-38 |
| 6 | 807FFFFF | -1,17549421e-38 | -1.175494e-38 |
| 7 | 80000001 | -1,40129846e-45 | -1.401298e-45 |
| 8 | 80000000 | -0 | -0.000000e+00 |
| 9 | 00000000 | +0 | 0.000000e+00 |
| 10 | 00000001 | 1,40129846e-45 | 1.401298e-45 |
| 11 | 007FFFFF | 1,17549421e-38 | 1.175494e-38 |
| 12 | 00800000 | 1,17549435e-38 | 1.175494e-38 |
| 13 | 7F7FFFFF | 3,40282347e+38 | 3.402823e+38 |
| 14 | 7F800000 | inf | inf |
| 15 | 7F800001 | nan | nan |
| 16 | 7FFFFFFF | nan | nan |

Для double:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Выходные данные |
| 1 | FFFFFFFFFFFFFFFF | -nan | -nan |
| 2 | FFF0000000000001 | -nan | -nan |
| 3 | FFF0000000000000 | -inf | -inf |
| 4 | FFEFFFFFFFFFFFFF | -1,79769313486231570814527423732E308 | -1.797693e+308 |
| 5 | 8010000000000000 | -2,22507385850720138309023271733E-308 | -2.225074e-308 |
| 6 | 800FFFFFFFFFFFFF | -2,22507385850720088902458687609E-308 | -2.225074e-308 |
| 7 | 8000000000000001 | -4.94065645841246544176568792868E-324 | -4.940656e-324 |
| 8 | 8000000000000000 | -0 | -0.000000e+00 |
| 9 | 0000000000000000 | +0 | 0.000000e+00 |
| 10 | 0000000000000001 | 4.94065645841246544176568792868E-324 | 4.940656e-324 |
| 11 | 000FFFFFFFFFFFFF | 2.22507385850720088902458687609E-308 | 2.225074e-308 |
| 12 | 0010000000000000 | 2.22507385850720138309023271733E-308 | 2.225074e-308 |
| 13 | 7FEFFFFFFFFFFFFF | 1,79769313486231570814527423732E308 | 1.797693e+308 |
| 14 | 7FF0000000000000 | inf | inf |
| 15 | 7FF0000000000001 | nan | nan |
| 16 | 7FFFFFFFFFFFFFFF | nan | nan |