Жуйков Артем, 23501/4

main.cpp

```cpp
#include <iostream>
#include "converter.h"

int main()
{
    Converter converter;
    converter.convertWithConlose();

    return 0;
}
```

converter.h

```cpp
#ifndef CONVERTER_H
#define CONVERTER_H
#include <string>
#include <map>

class Converter
{
    const std::map< char, int > literals =
    {
        { '0',  0 },
        { '1',  1 },
        { '2',  2 },
        { '3',  3 },
        { '4',  4 },
        { '5',  5 },
        { '6',  6 },
        { '7',  7 },
        { '8',  8 },
        { '9',  9 },
        { 'A', 10 },
        { 'a', 10 },
        { 'B', 11 },
        { 'b', 11 },
        { 'C', 12 },
        { 'c', 12 },
        { 'D', 13 },
        { 'd', 13 },
        { 'E', 14 },
        { 'e', 14 },
        { 'F', 15 },
        { 'f', 15 }
    };
    static const unsigned int accurate = 10;

    short int radixFrom;
    short int radixTo;
    std::string integerPart;
    std::string fractionalPart;

    std::string convertIntegerPart() const;
    std::string convertFractionalPart() const;
    char getKey(int value) const;
```

```
43
44 public:
45     Converter():
46         radixFrom(0),
47         radixTo(0),
48         integerPart(""),
49         fractionalPart("")
50     {}
51     void convertWithConlose();
52     std::string convert(const std::string initialNumber, const int radixFrom
           , const int radixTo);
53 };
54
55 #endif // CONVERTER_H
```

converter.cpp

```
1  #include "converter.h"
2  #include <iostream>
3  #include <math.h>
4
5  void Converter::convertWithConlose()
6  {
7      std::string inputNumber = "";
8      std::string inputFrom = "";
9      std::string inputTo = "";
10
11     std::cout << "number: ";
12     std::cin  >> inputNumber;
13     std::cout << std::endl << "from: ";
14     std::cin  >> inputFrom;
15     std::cout << std::endl << "to: ";
16     std::cin  >> inputTo;
17
18     for (char item : inputNumber) {
19         if ((item < '0' || item > '9') && (item < 'A' || item > 'F') &&
20                 (item < 'a' || item > 'f') && item != ',' && item != '.') {
21             std::cout << "input error" << std::endl;
22             return;
23         }
24     }
25
26     for (char item : inputFrom) {
27         if (item < '0' || item > '9') {
28             std::cout << "input error" << std::endl;
29             return;
30         }
31     }
32
33     for (char item : inputTo) {
34         if (item < '0' || item > '9') {
35             std::cout << "input error" << std::endl;
36             return;
37         }
38     }
39
40     int from = atoi(inputFrom.c_str());
41     int to = atoi(inputTo.c_str());
42
```

```
43      for (char item : inputNumber) {
44          if (item != ',' && item != '.' && literals.at(item) >= from) {
45              std::cout << "wrong numeric base" << std::endl;
46              return;
47          }
48      }
49
50      std::cout << std::endl << inputNumber << " (" << inputFrom << ") = " <<
51                   convert(inputNumber, from, to) << " (" << inputTo << ")" <<
                         std::endl;
52  }
53
54  std::string Converter::convert(const std::string number, const int radixFrom
        , const int radixTo)
55  {
56      this->radixFrom = radixFrom;
57      this->radixTo = radixTo;
58      bool isInteger = true;
59      unsigned int i = 0;
60
61      while (i < number.size() && number[i] != ',' && number[i] != '.') {
62          integerPart.push_back(number[i++]);
63      }
64      if (number[i] == ',' || number[i] == '.') {
65          isInteger = false;
66          i++;
67          while (i < number.size()) {
68              fractionalPart.push_back(number[i++]);
69          }
70      }
71
72      if (isInteger) {
73          return convertIntegerPart();
74      }
75      return convertIntegerPart() + '.' + convertFractionalPart();
76  }
77
78  std::string Converter::convertIntegerPart() const
79  {
80      long integerRadix10 = 0;
81      std::string reverseNumber = "";
82
83      for (unsigned int i = 0; i < integerPart.size(); i++) {
84          integerRadix10 += literals.at(integerPart[i]) * pow(radixFrom,
               integerPart.size() - 1 - i);
85      }
86
87      while (integerRadix10 >= radixTo) {
88          reverseNumber.push_back(getKey(integerRadix10 % radixTo));
89          integerRadix10 /= radixTo;
90      }
91      reverseNumber.push_back(getKey(integerRadix10));
92
93      std::string result = "";
94      for (int i = reverseNumber.length() - 1; i >= 0; i--){
95          result.push_back(reverseNumber[i]);
96      }
97
98      return result;
99  }
```

```
100
101 std::string Converter::convertFractionalPart() const
102 {
103     double fractionalRadix10 = 0;
104     std::string result = "";
105
106     for (unsigned int i = 0; i < fractionalPart.size(); i++) {
107         fractionalRadix10 += literals.at(fractionalPart[i]) * pow(radixFrom,
                ((int)i + 1) * (-1));
108     }
109
110     while (fractionalRadix10 >= 10e-7 && result.length() < accurate) {
111         fractionalRadix10 *= radixTo;
112         result.push_back(getKey((int)fractionalRadix10));
113         fractionalRadix10 -= (int)fractionalRadix10;
114     }
115
116     return result;
117 }
118
119 char Converter::getKey(int value) const
120 {
121     for (std::map< char, int >::const_iterator it = literals.begin(); it !=
           literals.end(); ++it) {
122         if ((*it).second == value) {
123             return (*it).first;
124         }
125     }
126     return '_';
127 }
```