

Lab.4.1

#include <iostream>

#include <string>

using namespace std;

int main()

{

cout << "Enter count: ";

int c;

cin >> c;

int \*\* g = new int\*[c];

for (int i = 0; i < c; i++)

{

int n;

cout << "Enter the number of marks: ";

cin >> n;

g[i] = new int[n+1];

g[i][0] = n;

for (int j = 1; j <= n; j++)

{

cin >> g[i][j];

}

}

for (int i = 0; i < c; i++)

{

float finalGrad = 0;

string grades;

for (int j = 1; j <= g[i][0]; j++)

{

finalGrad += g[i][j];

grades += to\_string(g[i][j]) + " ";

}

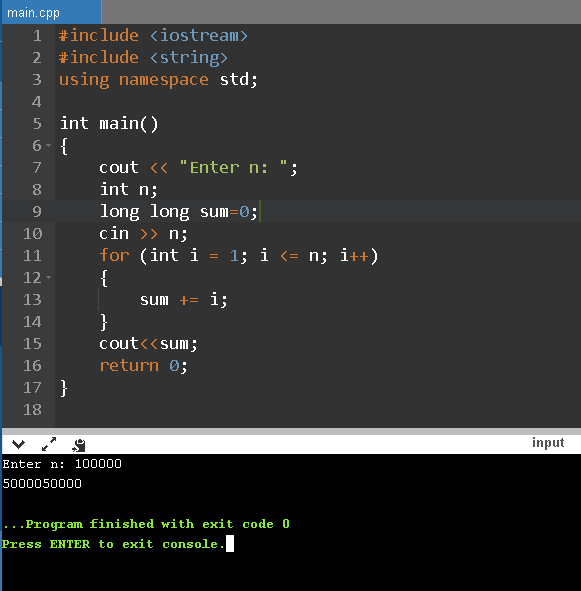
finalGrad = finalGrad / g[i][0];

cout<<"Course "<<i+1<<": final "<<finalGrad<<", grades: "<<grades.c\_str()<<endl;

}

return 0;

}



Lab.4.2(1)

#include <iostream>

#include <string>

using namespace std;

int main()

{

cout << "Enter n: ";

int n;

long long sum=0;

cin >> n;

for (int i = 1; i <= n; i++)

{

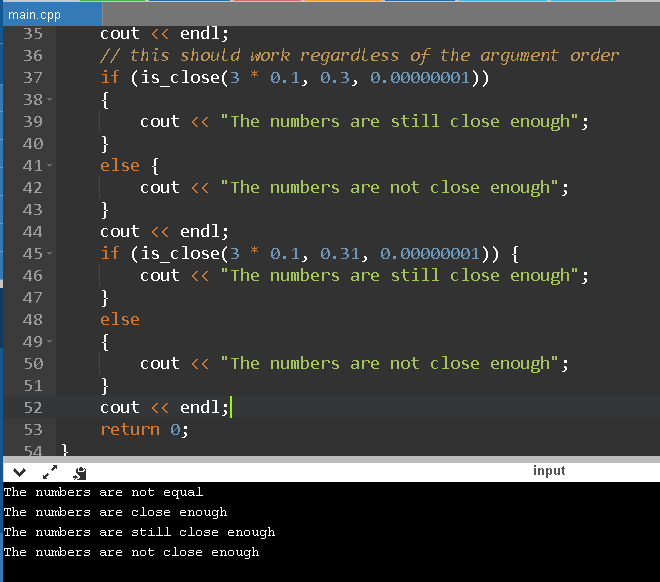
sum += i;

}

cout<<sum;

return 0;

}



Lab.4.2(2)

#include <iostream>

#include <cmath>

using namespace std;

bool is\_close(double a, double b, double tolerance)

{

double diff = a - b;

if (fabs(diff) > tolerance)

return false;

else

return true;

}

int main()

{

if (0.3 == 3 \* 0.1)

{

cout << "The numbers are equal";

}

else

{

cout << "The numbers are not equal";

}

cout << endl;

if (is\_close(0.3, 3 \* 0.1, 0.00000001))

{

cout << "The numbers are close enough";

}

else

{

cout << "The numbers are not close enough";

}

cout << endl;

// this should work regardless of the argument order

if (is\_close(3 \* 0.1, 0.3, 0.00000001))

{

cout << "The numbers are still close enough";

}

else {

cout << "The numbers are not close enough";

}

cout << endl;

if (is\_close(3 \* 0.1, 0.31, 0.00000001)) {

cout << "The numbers are still close enough";

}

else

{

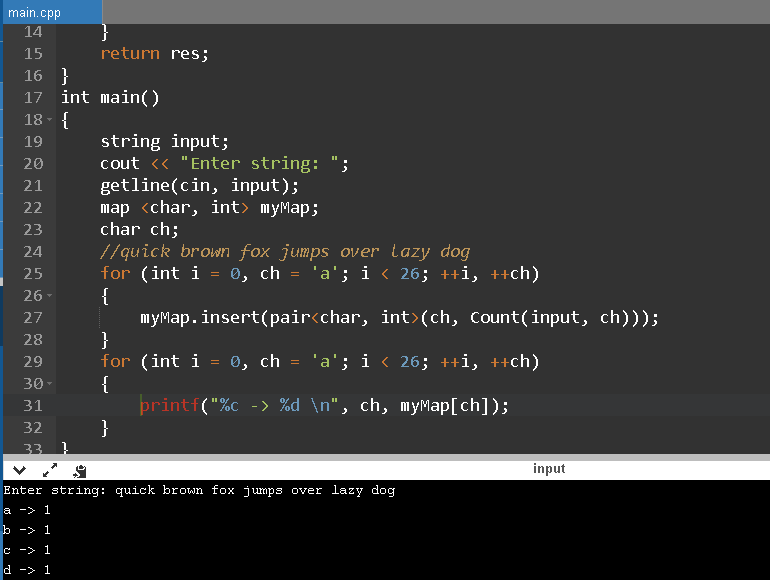
cout << "The numbers are not close enough";

}

cout << endl;

return 0;

}



Lab. 4.4

#include <iostream>

#include <map>

#include <string>

using namespace std;

int Count(string str, char ch)

{

int res = 0;

for (int i = 0; i < str.length(); i++)

{

if (str[i] == ch)

res++;

}

return res;

}

int main()

{

string input;

cout << "Enter string: ";

getline(cin, input);

map <char, int> myMap;

char ch;

//quick brown fox jumps over lazy dog

for (int i = 0, ch = 'a'; i < 26; ++i, ++ch)

{

myMap.insert(pair<char, int>(ch, Count(input, ch)));

}

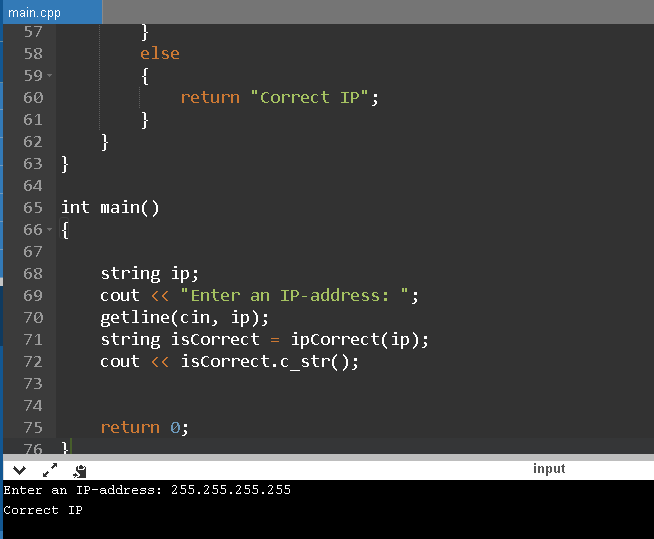
for (int i = 0, ch = 'a'; i < 26; ++i, ++ch)

{

printf("%c -> %d \n", ch, myMap[ch]);

}

}



Lab.4.5(1)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

void tokenize(string const &str, const char delim, vector<string> &out)

{

size\_t start;

size\_t end = 0;

while ((start = str.find\_first\_not\_of(delim, end)) != string::npos)

{

end = str.find(delim, start);

out.push\_back(str.substr(start, end - start));

}

}

inline bool isInteger(const std::string & s)

{

if (s.empty() || ((!isdigit(s[0])) && (s[0] != '-') && (s[0] != '+'))) return false;

char \* p;

strtol(s.c\_str(), &p, 10);

return (\*p == 0);

}

string ipCorrect(string ip)

{

const char delim = '.';

vector<string> out;

tokenize(ip, delim, out);

if (out.size() <= 3)

{

return "Incorrect parts count";;

}

else if (out.size() >= 5)

{

return "Too many parts";

}

for (auto &ip : out)

{

if (!isInteger(ip))

{

return "Only digits and dots allowed";

}

else if (stoi(ip) > 255 && stoi(ip) <= 999)

{

return "Too big a value of a part";

}

else if (ip.length() >= 4)

{

return "Too many characters in a part";

}

else

{

return "Correct IP";

}

}

}

int main()

{

string ip;

cout << "Enter an IP-address: ";

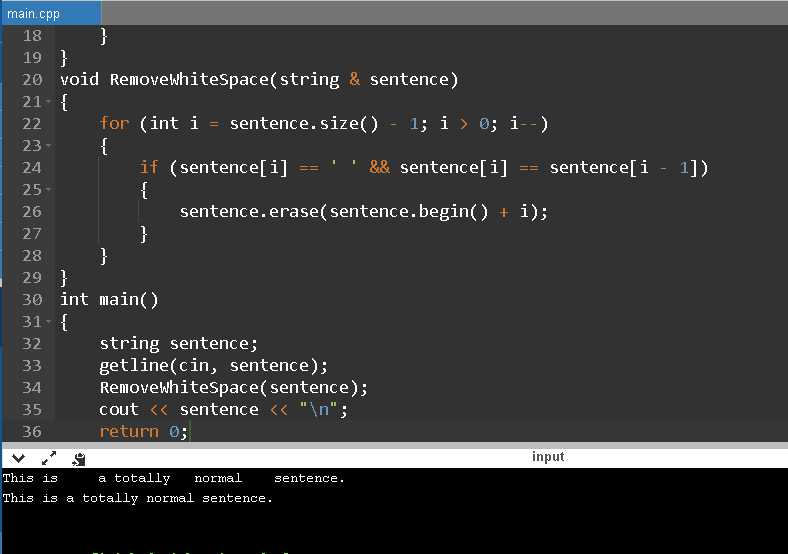
getline(cin, ip);

string isCorrect = ipCorrect(ip);

cout << isCorrect.c\_str();

return 0;

}



Lab. 4.5(2)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

void tokenize(string const &str, const char delim, vector<string> &out)

{

size\_t start;

size\_t end = 0;

while ((start = str.find\_first\_not\_of(delim, end)) != string::npos)

{

end = str.find(delim, start);

out.push\_back(str.substr(start, end - start));

}

}

void RemoveWhiteSpace(string & sentence)

{

for (int i = sentence.size() - 1; i > 0; i--)

{

if (sentence[i] == ' ' && sentence[i] == sentence[i - 1])

{

sentence.erase(sentence.begin() + i);

}

}

}

int main()

{

string sentence;

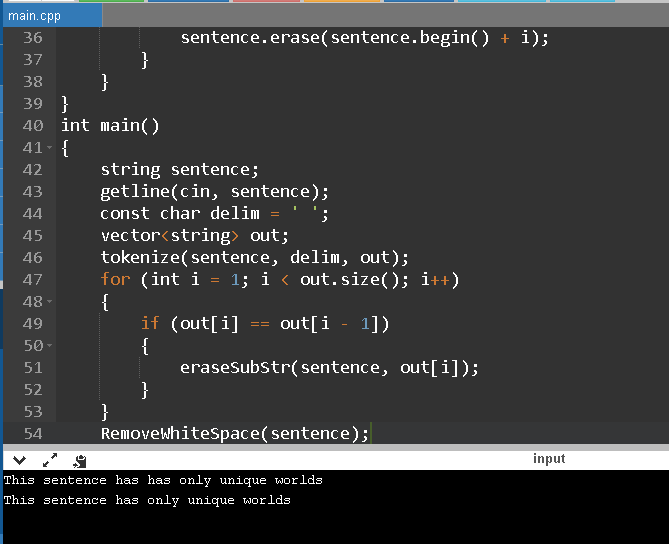
getline(cin, sentence);

RemoveWhiteSpace(sentence);

cout << sentence << "\n";

return 0;

}



Lab. 4.5(3)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

void tokenize(string const &str, const char delim, vector<string> &out)

{

size\_t start;

size\_t end = 0;

while ((start = str.find\_first\_not\_of(delim, end)) != string::npos)

{

end = str.find(delim, start);

out.push\_back(str.substr(start, end - start));

}

}

void eraseSubStr(std::string & mainStr, const std::string & toErase)

{

size\_t pos = mainStr.find(toErase);

if (pos != std::string::npos)

{

mainStr.erase(pos, toErase.length());

}

}

void RemoveWhiteSpace(string & sentence)

{

for (int i = sentence.size() - 1; i > 0; i--)

{

if (sentence[i] == ' ' && sentence[i] == sentence[i - 1])

{

sentence.erase(sentence.begin() + i);

}

}

}

int main()

{

string sentence;

getline(cin, sentence);

const char delim = ' ';

vector<string> out;

tokenize(sentence, delim, out);

for (int i = 1; i < out.size(); i++)

{

if (out[i] == out[i - 1])

{

eraseSubStr(sentence, out[i]);

}

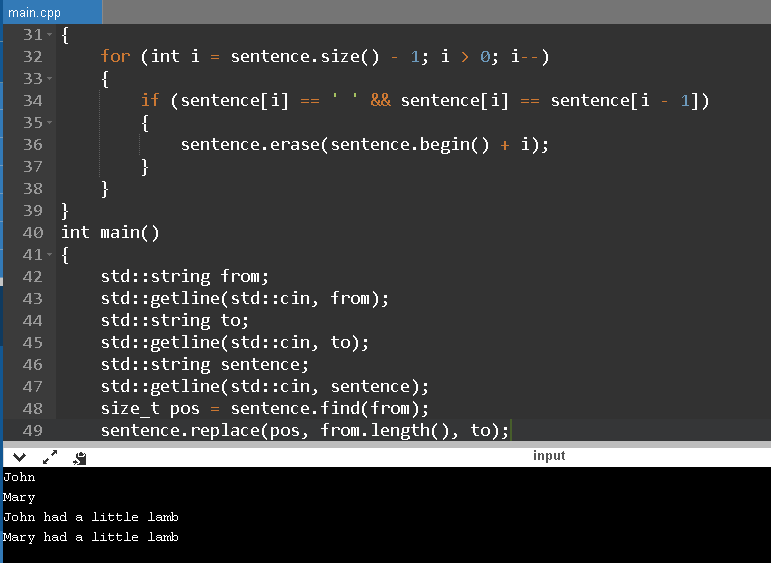
}

RemoveWhiteSpace(sentence);

cout << sentence << "\n";

return 0;

}



Lab.4.5(4)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

void tokenize(string const &str, const char delim, vector<string> &out)

{

size\_t start;

size\_t end = 0;

while ((start = str.find\_first\_not\_of(delim, end)) != string::npos)

{

end = str.find(delim, start);

out.push\_back(str.substr(start, end - start));

}

}

void eraseSubStr(std::string & mainStr, const std::string & toErase)

{

size\_t pos = mainStr.find(toErase);

if (pos != std::string::npos)

{

mainStr.erase(pos, toErase.length());

}

}

void RemoveWhiteSpace(string & sentence)

{

for (int i = sentence.size() - 1; i > 0; i--)

{

if (sentence[i] == ' ' && sentence[i] == sentence[i - 1])

{

sentence.erase(sentence.begin() + i);

}

}

}

int main()

{

std::string from;

std::getline(std::cin, from);

std::string to;

std::getline(std::cin, to);

std::string sentence;

std::getline(std::cin, sentence);

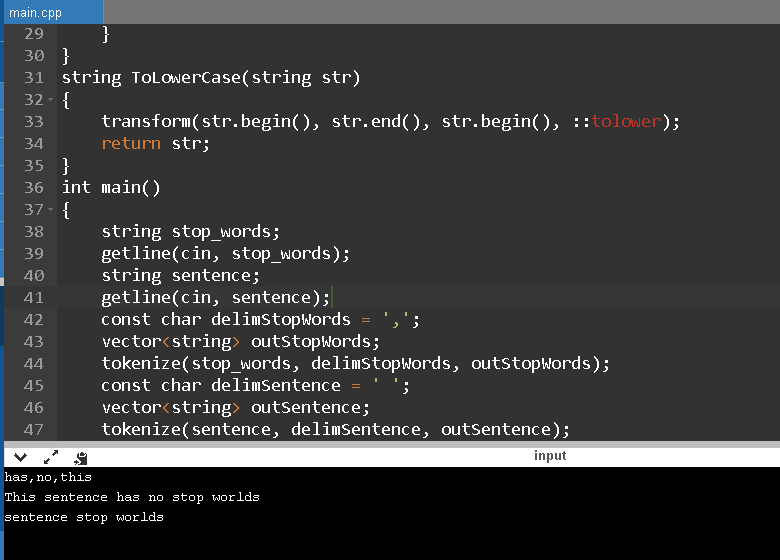
size\_t pos = sentence.find(from);

sentence.replace(pos, from.length(), to);

std::cout << sentence << "\n";

return 0;

}



Lab.4.5(5)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

void tokenize(string const &str, const char delim, vector<string> &out)

{

size\_t start;

size\_t end = 0;

while ((start = str.find\_first\_not\_of(delim, end)) != string::npos)

{

end = str.find(delim, start);

out.push\_back(str.substr(start, end - start));

}

}

void RemoveWhiteSpace(string & sentence)

{

for (int i = sentence.size() - 1; i > 0; i--)

{

if (sentence[i] == ' ' && sentence[i] == sentence[i - 1])

{

sentence.erase(sentence.begin() + i);

}

}

}

string ToLowerCase(string str)

{

transform(str.begin(), str.end(), str.begin(), ::tolower);

return str;

}

int main()

{

string stop\_words;

getline(cin, stop\_words);

string sentence;

getline(cin, sentence);

const char delimStopWords = ',';

vector<string> outStopWords;

tokenize(stop\_words, delimStopWords, outStopWords);

const char delimSentence = ' ';

vector<string> outSentence;

tokenize(sentence, delimSentence, outSentence);

for (int i = 0; i < outSentence.size(); i++)

{

for (int j = 0; j < outStopWords.size(); j++)

{

if (ToLowerCase(outSentence[i]) == ToLowerCase(outStopWords[j]))

{

outSentence.erase(outSentence.begin() + i);

i--;

}

}

}

sentence = "";

for (int i = 0; i < outSentence.size(); i++)

{

sentence += outSentence[i] + " ";

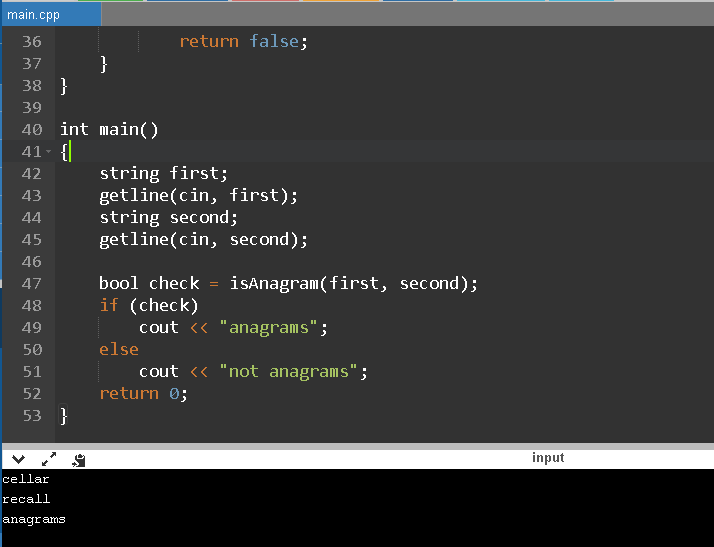
}

RemoveWhiteSpace(sentence);

std::cout << sentence << "\n";

return 0;

}



Lab.4.5(6)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

bool isAnagram(string first, string second)

{

if (first.length() != second.length())

{

return false;

}

else

{

int count = first.length();

for (int i = 0; i < first.length(); i++)

{

for (int j = 0; j < second.length(); j++)

{

if (first[i] == second[j])

{

first.erase(first.begin() + i);

second.erase(second.begin() + j);

i--; j--;

count--;

break;

}

}

}

if (count == 0)

return true;

else

return false;

}

}

int main()

{

string first;

getline(cin, first);

string second;

getline(cin, second);

bool check = isAnagram(first, second);

if (check)

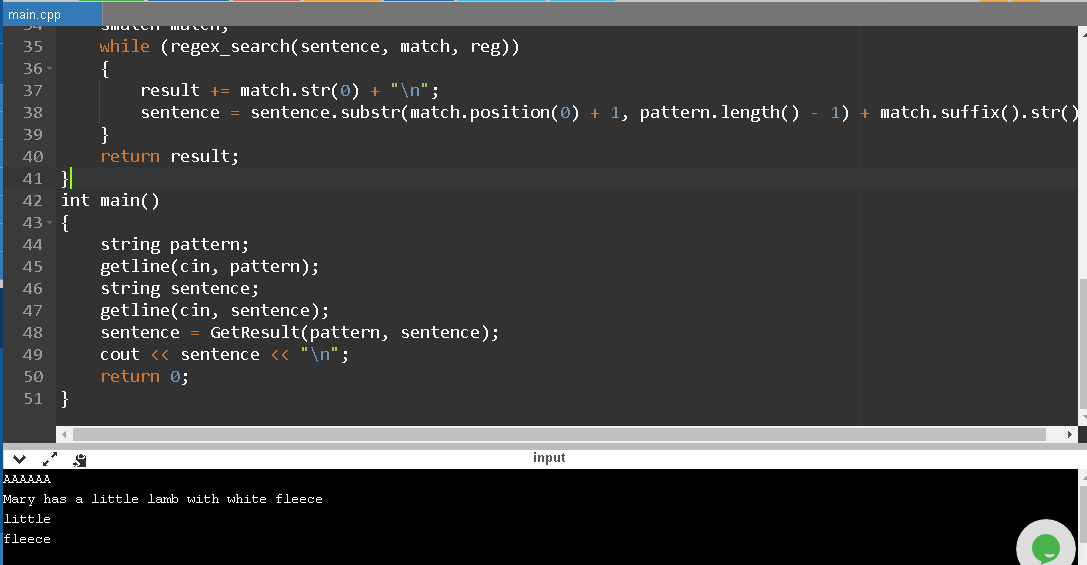
cout << "anagrams";

else

cout << "not anagrams";

return 0;

}



Lab.4.5(7)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

string GetResult(string pattern, string sentence)

{

string re = "";

for (int i = 0; i < pattern.length(); i++)

{

if (pattern[i] == 'D')

{

re += "\\d";

}

else if (pattern[i] == 'A')

{

re += "([a-z]|[A-Z])";

}

else if (pattern[i] == '-')

{

re += "-";

}

else

{

re += ".";

}

}

const regex reg(re);

string result = "";

smatch match;

while (regex\_search(sentence, match, reg))

{

result += match.str(0) + "\n";

sentence = sentence.substr(match.position(0) + 1, pattern.length() - 1) + match.suffix().str();

}

return result;

}

int main()

{

string pattern;

getline(cin, pattern);

string sentence;

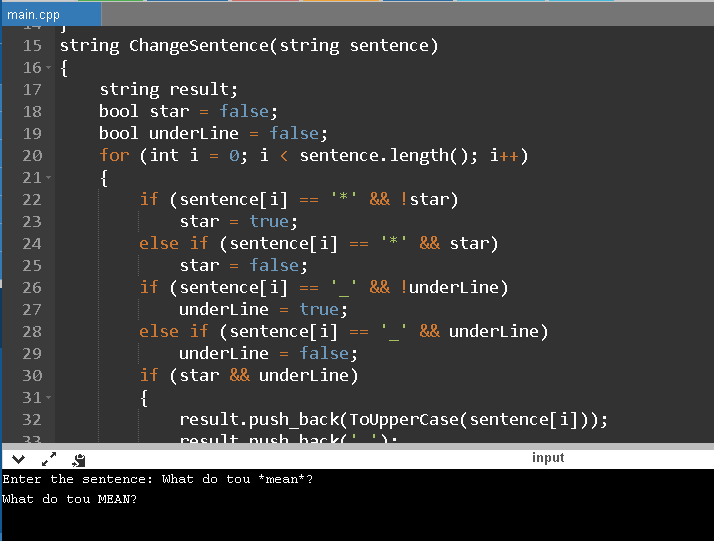
getline(cin, sentence);

sentence = GetResult(pattern, sentence);

cout << sentence << "\n";

return 0;

}



Lab. 4.5(8)

#include <iostream>

#include <string>

#include <vector>

#include <regex>

#include <algorithm>

#include <cctype>

#include <map>

#include <iterator>

using namespace std;

char ToUpperCase(char ch)

{

ch = toupper(ch);

return ch;

}

string ChangeSentence(string sentence)

{

string result;

bool star = false;

bool underLine = false;

for (int i = 0; i < sentence.length(); i++)

{

if (sentence[i] == '\*' && !star)

star = true;

else if (sentence[i] == '\*' && star)

star = false;

if (sentence[i] == '\_' && !underLine)

underLine = true;

else if (sentence[i] == '\_' && underLine)

underLine = false;

if (star && underLine)

{

result.push\_back(ToUpperCase(sentence[i]));

result.push\_back(' ');

}

else if (star)

{

result.push\_back(ToUpperCase(sentence[i]));

}

else if (underLine)

{

result.push\_back(sentence[i]);

result.push\_back(' ');

}

if (!star && !underLine)

{

result.push\_back(sentence[i]);

}

}

for (int i = 0; i < result.length(); i++)

{

if (result[i] == '\*' || result[i] == '\_')

result.erase(i, 1);

}

return result;

}

int main()

{

cout << "Enter the sentence: ";

string sentence;

getline(cin, sentence);

sentence = ChangeSentence(sentence);

cout << sentence << "\n";

return 0;

}