МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

**отчет**

по практической работе №1

по дисциплине «Программирование»

Тема: "Текстовые строки как массивы символов"

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0323 |  | Рудницкая Т.С. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

Цель работы.

Знакомство со строками в C++.

Основные теоретические положения.

Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). Поэтому, если вы хотите создать текстовый массив для хранения 10 (𝑁) символов, нужно выделить память под 11(𝑁 + 1) символов. Объявленный таким образом массив может использоваться для хранения текстовых строк, содержащих не более 10 символов. Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки. При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

1. char S1[15] = “This is text”; char S2[] = “Пример текста”;

Последние два элемента переменной 𝑆1 просто не используются, а строка 𝑆2 автоматически подстраивается под длину инициализирующего текста. При работе со строками можно обращаться к отдельным символам строки как в обычном одномерном массиве с помощью индексов:

1. cout << S1[0]; // На экране будет выведен символ ‘T’

Если строка формируется при помощи цикла (или иного способа), то необходимо ее конец обязательно записать нулевой символ '\0'.  
При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Вывод текстовых строк на экран крайне простая задача:

1. char Str[21] = “Это пример текста”;
2. cout << Str << endl; cout << “Это текстовый литерал.” << endl;

Ввод текста с клавиатуры можно осуществлять разными способами, каждый из которых имеет определенные особенности. 7 Непосредственное чтение текстовых строк из потока вывода осуществляется до первого знака пробела. Такой способ чтения обеспечивает ввод символов до первого пробельного символа (не до конца строки). Остальные символы введенного с клавиатуры остаются в потоке ввода и могут быть прочитаны из него следующими операторами >>. Для того чтобы прочесть всю строку полностью, можно воспользоваться одной из функций gets или gets\_s (для этого в программу должен быть включен заголовочный файл). Функция gets имеет один параметр, соответствующий массиву символов, в который осуществляется чтение. Вторая функция (gets\_s) имеет второй параметр, задающий максимальную длину массива символов 𝑆𝑡𝑟. Ввод текста, длина которого (вместе с нулевым символом) превышает значение второго параметра (то есть длины символьного массива 𝑆𝑡𝑟), приводит к возникновению ошибки при выполнении программы Предпочтительно использование функции потока ввода cin.getline:

1. const int N = 21; char Str [N]; cin.getline (Str, N); // Пусть введена строка “Это пример текста”
2. cout << Str << endl; // На экран будет выведено “ Это пример текста”

Если длина введенного с клавиатуры текста превышает максимальную длину массива 𝑆𝑡𝑟, в него будет записано (в нашем примере) 20 символов вводимого текста и нулевой символ. Остальные символы введенного текста остаются во входном потоке и могут быть взяты из него следующими инструкциями ввода. Функция cin.getline может иметь третий параметр, задающий символ, при встрече которого чтение строки из потока прекращается: cin.getline (Str, N, ‘.’); Иногда чтение из потока невозможно (например, попытка считать слишком длинный текст). Для того чтобы продолжить чтение из потока, необходимо восстановить его нормальное состояние. Этого можно достигнуть с помощью функции потока cin.clear(), которая сбрасывает состояние потока в нормальное.

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро. 10 Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим. Убедиться в этом можно по примеру, приведенному на рис. 1.1.

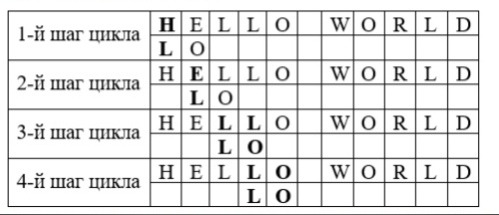


Рисунок 1.1 – Поиск подстроки “LO” в строке “HELLO WORLD”

На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена. Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма. Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке. Суть алгоритма: дана подстрока 𝑆 и строка 𝑇. Требуется определить индекс, начиная с которого образец 𝑆 содержится в строке 𝑇. Если 𝑆 не содержится в 𝑇, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке. Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат). Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку. Правила построения таблицы сдвигов:

Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).

Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.

Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.

Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.

Постановка задачи.

Разработать алгоритм и написать программу, которая позволяет:

1. С клавиатуры или из файла (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.
2. Необходимо отредактировать входной текст:
   * удалить лишние пробелы;
   * удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);
   * исправить регистр букв, если это требуется (пример некорректного использования букв: пРиМЕр);
3. Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм КНМ.
4. Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.
5. Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.

Выполнение работы.

Так как в моей программе используется визуальный интерфейс C++/CLI, ввод строки для работы (и последующих подстрок) будет производится из текстового поля формы. Значение текстовых полей в CLI хранится в переменных типа System::String^. Так что для работы требуется конвертировать эту строку в массив символов Char\*, а потом для вывода снова конвертировать в System::String^.

В работе используются несколько глобальных переменных: вектор ResultArr, в котором хранятся результаты поиска подстроки в строке, и сама строка char\* str.

Так же в работе используется служебная функция delStr, которая удаляет из строки подстроку заданной длины, начиная с указанного индекса.

1. void Coursach::lab4::delStr(int i, int len)
2. {
3. //в цикле сдвигаем строку на столько символов влево, сколько нужно удалить
4. for (i; i <= strlen(str); i++) {
5. str[i] = str[i + len];
6. }
7. }

Для типографирования строки проверяем: если символ является знаком препинания – смотрим соседний символ, повторные символы удаляем. Если это точка – смотрим два соседних символа, если точек три, то оставляем, если две, то лишнюю удаляем. Если символ буква, то проверяем, не является ли она прописной, и при необходимости меняем на строчную.

1. void Coursach::lab4::inputTextCorretion()
2. {
3. //в цикле находим все знаки препинания, кроме точки, с ней отдельно
4. for (int i = 0; i <= strlen(str); i++) {
5. if (str[i] == ' ' ||
6. str[i] == ',' ||
7. str[i] == ':' ||
8. str[i] == ';' ||
9. str[i] == '?' ||
10. str[i] == '!')
11. {
12. //если следующий символ такой же, то удаляем его
13. if (str[i + 1] == str[i]) delStr((i + 1), 1);
14. }
15. //если символ точка и следующий символ тоже точка
16. else if (str[i] == '.' && str[i + 1] == str[i]) {
17. //проверим третий символ последовательности, и если это тоже точка, просто сдвигаем маркер вперёд и идём на следующую итерацию цикла
18. if (str[i + 2] == str[i]) {
19. i++;
20. i++;
21. continue;
22. }
23. //если это две точки, то удаляем лишнюю
24. else
25. delStr((i + 1), 1);
26. }
27. //приводим char к int, чтобы посмотреть код символа
28. //если это прописная латинская буква, то меняем её на соответствующую строчную
29. else {
30. int ch = static\_cast<int>(str[i]);
31. if (ch <= 90 && ch >= 65)
32. str[i] = ch + 32;
33. }
34. }
35. }

Реализация поиска перебором:

1. int Coursach::lab4::bruteForceSearch(char\* subStr)
2. {
3. int strLen = strlen(str);
4. int subStrLen = strlen(subStr);
5. int flag = 0;
6. int counter = 0;
7. //задаем длину вектору, количество совпадений будет точно не больше, чем длина строки
8. resultsArr.resize(strLen);
9. for (int i = 0; i <= strLen - subStrLen; i++) {
10. //если очередной символ совпадает с первым символом подстроки
11. if (str[i] == subStr[0]) {
12. //ставим флаг и запускаем проверку дальше
13. flag = 1;
14. //если есть несовпадение с подстрокой, то прерываем сравнение и идём дальше
15. for (int j = 0; j < subStrLen; j++, i++) {
16. if (str[i] != subStr[j]) {
17. flag = 0;
18. break;
19. }
20. }
21. //или записываем позицию найденной подстроки в результирующий массив и увеличиваем счётчик
22. if (flag == 1) {
23. resultsArr[counter] = (i - subStrLen);
24. counter++;
25. }
26. }
27. }
28. return counter;
29. }

Реализация поиска КНМ:

1. int Coursach::lab4::knmSearch(char \* subStr)
2. {
3. int strLen = strlen(str);
4. int subStrLen = strlen(subStr);
5. int flag = 0;
6. int counter = 0;
7. int resultIndex = 0;
8. std::vector<int> tempArr;
9. //задаем длину временному вектору, количество совпадений будет точно не больше, чем длина строки
10. tempArr.resize(strLen);
11. //находим все вхождения первой буквы искомой подстроки
12. //и записываем во временный массив
13. for (int i = 0; i <= strLen - subStrLen; i++) {
14. if (str[i] == subStr[0]) {
15. tempArr[counter] = i;
16. counter++;
17. }
18. }
19. //если было найдено хотя бы одно вхождение
20. if (counter > 0) {
21. //зададим длину вектору, в который записываются окончательные результаты
22. //их будет не больше, чем найденных вхождений первого символа
23. resultsArr.resize(counter);
24. //если подстрока входит не полностью, то обнуляем переменную-флаг
25. for (int i = 0; i < counter; i++) {
26. flag = 1;
27. int k = tempArr[i];
28. for (int j = 0; j < subStrLen; j++, k++) {
29. if (str[k] != subStr[j]) {
30. flag = 0;
31. break;
32. }
33. }
34. //если флаг не обнулён, записываем вхождение в результирующий массив
35. if (flag == 1) {
36. resultsArr[resultIndex] = tempArr[i];
37. resultIndex++;
38. }
39. }
40. }
41. return resultIndex;
42. }

Чтобы удалить вхождения указанной пользователем подстроки, находим все вхождения с помощью поиска КНМ, а потом с помощью функции удаления подстрок, удаляем в цикле все вхождения.

1. void Coursach::lab4::delSubStr(char \*subStr, int counter)
2. {
3. //считаем длину удаляемой последовательности
4. int len = strlen(subStr);
5. //вызываем функцию удаления подстроки столько раз, сколько последовательностей было найдено
6. for (int i = 0; i < counter; i++) {
7. delStr(resultsArr[i] - len\*i, len);
8. }
9. }

Чтобы найти все слова, в которых первая буква повторяется в самом слове, прогоняем всю строку. Считаем, что слово начинается с буквы или цифры и заканчивается перед пробелом. Находим первую латинскую букву или цифру и перебираем всё условное слово до пробела. Если в нём находится похожая буква, то оставляем, если нет – удаляем всё слово.

1. void Coursach::lab4::firstLetterSearch()
2. {
3. char firstLetter;
4. int firstLetterPos = 0;
5. //длина слова не меньше 1, потому что в каждом найденном слове есть как минимум одна буква
6. int wordLen = 1;
7. int flag = 0;
8. int ch;
9. int strLen = strlen(str);
10. for (int i = 0; i < strLen; i++) {
11. //приводим char к int, чтобы посмотреть
12. ch = static\_cast<int>(str[i]);
13. //буквы/цифры или что-то другое. Подразумеваем, что слово начинается
14. //с буквы или цифры
15. if (ch <= 122 && ch >= 97 || ch <= 57 && ch >= 48) {
16. //нашли начало слова, запомнили его первую букву и её положение
17. firstLetter = str[i];
18. firstLetterPos = i;
19. i++;
20. //теперь до первого пробела или до конца слова проверям
21. //нет ли в слове такой же буквы, как первая
22. //если есть совпадение, то прибавляем единичку к переменной-флагу
23. while ((str[i] != ' ') && (i < strLen)) {
24. if (str[i] == firstLetter) {
25. flag++;
26. }
27. i++;
28. wordLen++;
29. }
30. //проверили всё слово, если повторяющихся букв нет
31. if (flag == 0) {
32. //удаляем всё слово из строки
33. delStr((firstLetterPos), ++wordLen);
34. //меняем позицию маркера и уменьшаем длину слова
35. i -= wordLen;
36. strLen -= wordLen;
37. }
38. //обнуляем переменную-флаг и переменную с длиной слова
39. else {
40. flag = 0;
41. }
42. wordLen = 1;
43. }
44. }
45. }

Загрузка файла происходит через системный диалог openFileDialog.

Так же в программе есть обработка ошибок. В случае, если не загружается файл, если пользователь не указал строку или подстроку, внизу окна выводится сообщение об ошибке.

Полный код программы размещён в приложении А.

Выводы.

Поиск подстроки в строке в C++ — это тривиальная задача, которая легко решается, если есть понимание того, как в принципе работать с массивом символов.

Приложение А

Полный код программы

1) файл lab4.h

1. //показ ошибки
2. void showErr(String^ text);
3. //спрятать ошибки
4. void hideErr();
5. //оттипографировать строку
6. void inputTextCorretion();
7. //удалить один или несколько символов из строки
8. void delStr(int i, int len);
9. //удалить символ из строки
10. void showSearchResults(int counter);
11. //поиск перебором
12. int bruteForceSearch(char \* subStr);
13. //поиск КНМ
14. int knmSearch(char \* subStr);
15. //удалить все найденные подстроки
16. void delSubStr(char \*subStr, int counter);
17. //найти все слова, в которых первая буква повторяется
18. void firstLetterSearch();
19. //закрыть форму
20. private: System::Void btnCloseForm\_Click(System::Object^ sender, System::EventArgs^ e);
21. //получить текст для работы
22. private: System::Void btnGetText\_Click(System::Object^ sender, System::EventArgs^ e);
23. //обработка нажатия кнопки "поиск перебором"
24. private: System::Void bruteForceSearchBtn\_Click(System::Object^ sender, System::EventArgs^ e);
25. //обработка нажатия кнопки "поиск КНМ"
26. private: System::Void knmSearchBtn\_Click(System::Object^ sender, System::EventArgs^ e);
27. //обработка нажатия кнопки "удалить"
28. private: System::Void btnDeleteSubstr\_Click(System::Object^ sender, System::EventArgs^ e);
29. //обработка нажатия кнопки "вывести"
30. private: System::Void btnFirstLetterSearch\_Click(System::Object^ sender, System::EventArgs^ e);
31. //обработка нажатия кнопки "Загрузить из файла"
32. private: System::Void loadFromFile\_Click(System::Object^ sender, System::EventArgs^ e);

2) Файл lab4.cpp

1. #include "lab4.h"
2. using namespace System::Runtime::InteropServices;
3. int err = 0;
4. char\* str;
5. std::vector<int> resultsArr;
6. System::Void Coursach::lab4::btnCloseForm\_Click(System::Object^ sender, System::EventArgs^ e)
7. {
8. this->Close();
9. }
10. void Coursach::lab4::delStr(int i, int len)
11. {
12. //в цикле сдвигаем строку на столько символов влево, сколько нужно удалить
13. for (i; i <= strlen(str); i++) {
14. str[i] = str[i + len];
15. }
16. }
17. void Coursach::lab4::showErr(String^ text)
18. {
19. errorLabel1->Text += text;
20. errorPanel->Visible = true;
21. err++;
22. }
23. void Coursach::lab4::hideErr()
24. {
25. errorPanel->Visible = false;
26. err = 0;
27. deleteSubstrLabel->Visible = false;
28. }
29. void Coursach::lab4::inputTextCorretion()
30. {
31. //в цикле находим все знаки препинания, кроме точки, с ней отдельно
32. for (int i = 0; i <= strlen(str); i++) {
33. if (str[i] == ' ' ||
34. str[i] == ',' ||
35. str[i] == ':' ||
36. str[i] == ';' ||
37. str[i] == '?' ||
38. str[i] == '!')
39. {
40. //если следующий символ такой же, то удаляем его
41. if (str[i + 1] == str[i]) delStr((i + 1), 1);
42. }
43. //если символ точка и следующий символ тоже точка
44. else if (str[i] == '.' && str[i + 1] == str[i]) {
45. //проверим третий символ последовательности, и если это тоже точка, просто сдвигаем маркер вперёд и идём на следующую итерацию цикла
46. if (str[i + 2] == str[i]) {
47. i++;
48. i++;
49. continue;
50. }
51. //если это две точки, то удаляем лишнюю
52. else
53. delStr((i + 1), 1);
54. }
55. //приводим char к int, чтобы посмотреть код символа
56. //если это прописная латинская буква, то меняем её на соответствующую строчную
57. else {
58. int ch = static\_cast<int>(str[i]);
59. if (ch <= 90 && ch >= 65)
60. str[i] = ch + 32;
61. }
62. }
63. }
64. System::Void Coursach::lab4::btnGetText\_Click(System::Object^ sender, System::EventArgs^ e)
65. {
66. str = (char\*)(void\*)Marshal::StringToHGlobalAnsi(textBox1->Text);
67. if (strlen(str) == 0) {
68. showErr("Введите подстроку для поиска");
69. }
70. else {
71. hideErr();
72. if (workPanel->Visible == false) {
73. workPanel->Visible = true;
74. }
75. inputTextCorretion();
76. correctText->Text = gcnew System::String(str);
77. }
79. }
80. void Coursach::lab4::showSearchResults(int counter)
81. {
82. if (counter > 0) {
83. if (counter == 1) searchIndexesLabel->Text = "на позиции ";
84. else searchIndexesLabel->Text = "на позициях ";
85. for (int i = 0; i < counter; i++) {
86. searchIndexesLabel->Text += Convert::ToString(resultsArr[i]) + " ";
87. }
88. }
89. else {
90. searchIndexesLabel->Text = "нигде";
91. }
92. searchResultsPanel->Visible = true;
93. }
94. int Coursach::lab4::bruteForceSearch(char\* subStr)
95. {
96. int strLen = strlen(str);
97. int subStrLen = strlen(subStr);
98. int flag = 0;
99. int counter = 0;
100. //задаем длину вектору, количество совпадений будет точно не больше, чем длина строки
101. resultsArr.resize(strLen);
102. for (int i = 0; i <= strLen - subStrLen; i++) {
103. //если очередной символ совпадает с первым символом подстроки
104. if (str[i] == subStr[0]) {
105. //ставим флаг и запускаем проверку дальше
106. flag = 1;
107. //если есть несовпадение с подстрокой, то прерываем сравнение и идём дальше
108. for (int j = 0; j < subStrLen; j++, i++) {
109. if (str[i] != subStr[j]) {
110. flag = 0;
111. break;
112. }
113. }
114. //или записываем позицию найденной подстроки в результирующий массив и увеличиваем счётчик
115. if (flag == 1) {
116. resultsArr[counter] = (i - subStrLen);
117. counter++;
118. }
119. }
120. }
121. return counter;
122. }
123. System::Void Coursach::lab4::bruteForceSearchBtn\_Click(System::Object^ sender, System::EventArgs^ e)
124. {
125. char\* subStr = (char\*)(void\*)Marshal::StringToHGlobalAnsi(searchBox->Text);
126. if (strlen(subStr) == 0) {
127. showErr("Введите подстроку для поиска");
128. }
129. else {
130. hideErr();
131. int counter = bruteForceSearch(subStr);
132. showSearchResults(counter);
134. }
135. }
136. int Coursach::lab4::knmSearch(char \* subStr)
137. {
138. int strLen = strlen(str);
139. int subStrLen = strlen(subStr);
140. int flag = 0;
141. int counter = 0;
142. int resultIndex = 0;
143. std::vector<int> tempArr;
144. //задаем длину временному вектору, количество совпадений будет точно не больше, чем длина строки
145. tempArr.resize(strLen);
146. //находим все вхождения первой буквы искомой подстроки
147. //и записываем во временный массив
148. for (int i = 0; i <= strLen - subStrLen; i++) {
149. if (str[i] == subStr[0]) {
150. tempArr[counter] = i;
151. counter++;
152. }
153. }
154. //если было найдено хотя бы одно вхождение
155. if (counter > 0) {
156. //зададим длину вектору, в который записываются окончательные результаты
157. //их будет не больше, чем найденных вхождений первого символа
158. resultsArr.resize(counter);
159. //если подстрока входит не полностью, то обнуляем переменную-флаг
160. for (int i = 0; i < counter; i++) {
161. flag = 1;
162. int k = tempArr[i];
163. for (int j = 0; j < subStrLen; j++, k++) {
164. if (str[k] != subStr[j]) {
165. flag = 0;
166. break;
167. }
168. }
169. //если флаг не обнулён, записываем вхождение в результирующий массив
170. if (flag == 1) {
171. resultsArr[resultIndex] = tempArr[i];
172. resultIndex++;
173. }
174. }
175. }
176. return resultIndex;
177. }
178. System::Void Coursach::lab4::knmSearchBtn\_Click(System::Object^ sender, System::EventArgs^ e) {
179. char\* subStr = (char\*)(void\*)Marshal::StringToHGlobalAnsi(searchBox->Text);
180. if (strlen(subStr) == 0) {
181. showErr("Введите подстроку для поиска");
182. }
183. else {
184. hideErr();
185. int counter = knmSearch(subStr);
186. showSearchResults(counter);
187. }
188. }
189. void Coursach::lab4::delSubStr(char \*subStr, int counter)
190. {
191. //считаем длину удаляемой последовательности
192. int len = strlen(subStr);
193. //вызываем функцию удаления подстроки столько раз, сколько последовательностей было найдено
194. for (int i = 0; i < counter; i++) {
195. delStr(resultsArr[i] - len\*i, len);
196. }
197. }
198. System::Void Coursach::lab4::btnDeleteSubstr\_Click(System::Object^ sender, System::EventArgs^ e)
199. {
200. char\* subStr = (char\*)(void\*)Marshal::StringToHGlobalAnsi(delBox->Text);
201. if (strlen(subStr) == 0) {
202. showErr("Введите подстроку для удаления");
203. }
204. else {
205. hideErr();
206. int counter = knmSearch(subStr);
207. if (counter > 0) {
208. delSubStr(subStr, counter);
209. correctText->Text = gcnew System::String(str);
210. deleteSubstrLabel->Text = "удалено";
211. deleteSubstrLabel->Visible = true;
212. }
213. else {
214. deleteSubstrLabel->Text = "введённая последовательность не найдена";
215. deleteSubstrLabel->Visible = true;
216. }
217. }
218. }
219. void Coursach::lab4::firstLetterSearch()
220. {
221. char firstLetter;
222. int firstLetterPos = 0;
223. //длина слова не меньше 1, потому что в каждом найденном слове есть как минимум одна буква
224. int wordLen = 1;
225. int flag = 0;
226. int ch;
227. int strLen = strlen(str);
228. for (int i = 0; i < strLen; i++) {
229. //приводим char к int, чтобы посмотреть
230. ch = static\_cast<int>(str[i]);
231. //буквы/цифры или что-то другое. Подразумеваем, что слово начинается
232. //с буквы или цифры
233. if (ch <= 122 && ch >= 97 || ch <= 57 && ch >= 48) {
234. //нашли начало слова, запомнили его первую букву и её положение
235. firstLetter = str[i];
236. firstLetterPos = i;
237. i++;
238. //теперь до первого пробела или до конца слова проверям
239. //нет ли в слове такой же буквы, как первая
240. //если есть совпадение, то прибавляем единичку к переменной-флагу
241. while ((str[i] != ' ') && (i < strLen)) {
242. if (str[i] == firstLetter) {
243. flag++;
244. }
245. i++;
246. wordLen++;
247. }
248. //проверили всё слово, если повторяющихся букв нет
249. if (flag == 0) {
250. //удаляем всё слово из строки
251. delStr((firstLetterPos), ++wordLen);
252. //меняем позицию маркера и уменьшаем длину слова
253. i -= wordLen;
254. strLen -= wordLen;
255. }
256. //обнуляем переменную-флаг и переменную с длиной слова
257. else {
258. flag = 0;
259. }
260. wordLen = 1;
261. }
262. }
263. }
264. System::Void Coursach::lab4::btnFirstLetterSearch\_Click(System::Object^ sender, System::EventArgs^ e)
265. {
266. firstLetterSearch();
267. correctText->Text = gcnew System::String(str);
268. }
269. System::Void Coursach::lab4::loadFromFile\_Click(System::Object^ sender, System::EventArgs^ e)
270. {
271. //тут хранится путь к файлу
272. String^ fileName = "";
273. //когда файл выбран
274. if (openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) {
275. fileName = openFileDialog1->FileName;
276. }
277. //обработка ошибки открытия
278. try {
279. StreamReader^ file = File::OpenText(fileName);
280. str = (char\*)(void\*)Marshal::StringToHGlobalAnsi(file->ReadToEnd());
281. //прячем ошибки
282. hideErr();
283. //показываем рабочй интерфейс, если до этого он был скрыт
284. if (workPanel->Visible == false) {
285. workPanel->Visible = true;
286. }
287. //запускаем функцию типограф
288. inputTextCorretion();
289. //выводим в лейбл с рабочим текстом
290. correctText->Text = gcnew System::String(str);
291. }
292. catch(Exception^ e){
293. showErr("Ошибка открытия файла");
294. }
295. }

Приложение Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

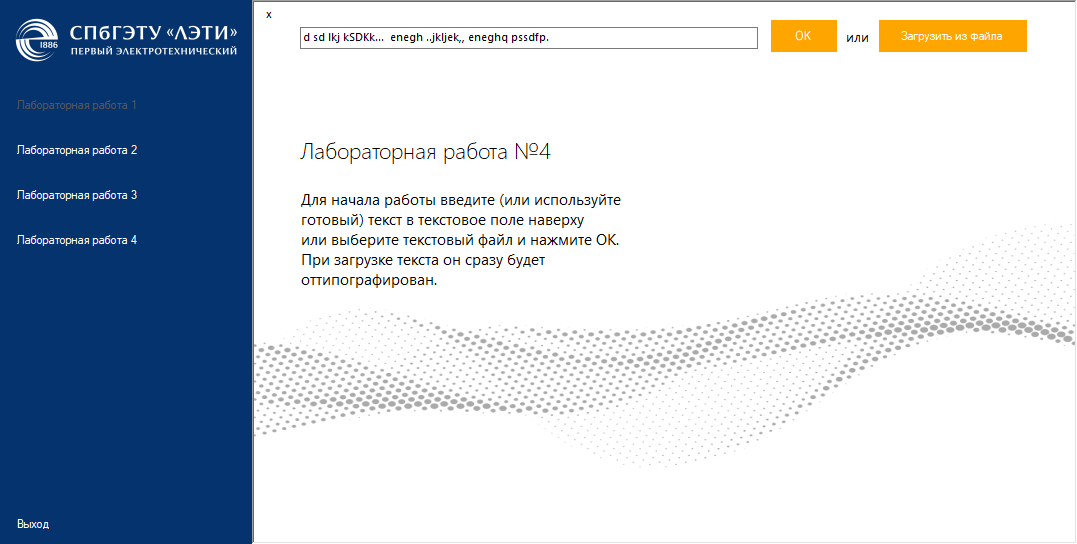


Рисунок 1 – Экран приветствия

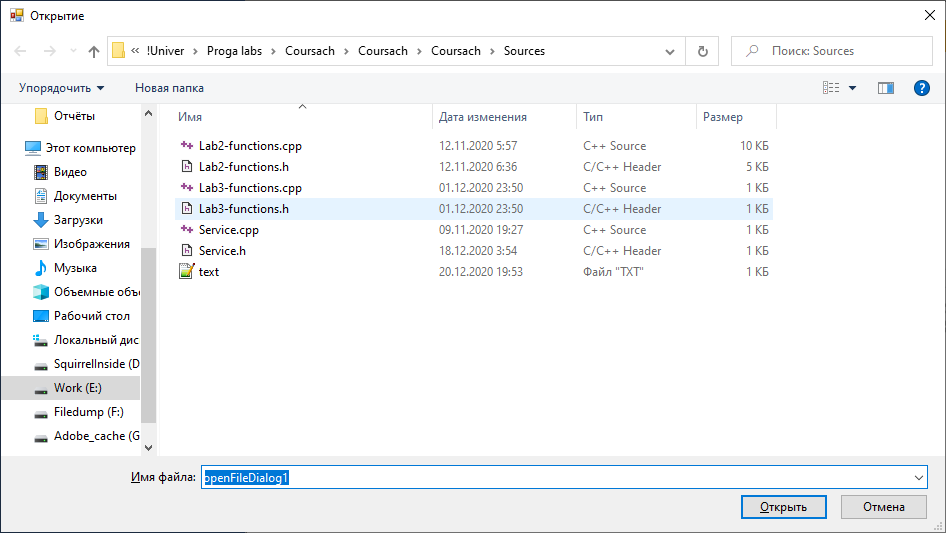


Рисунок 2 – Диалоговое окно загрузки файла

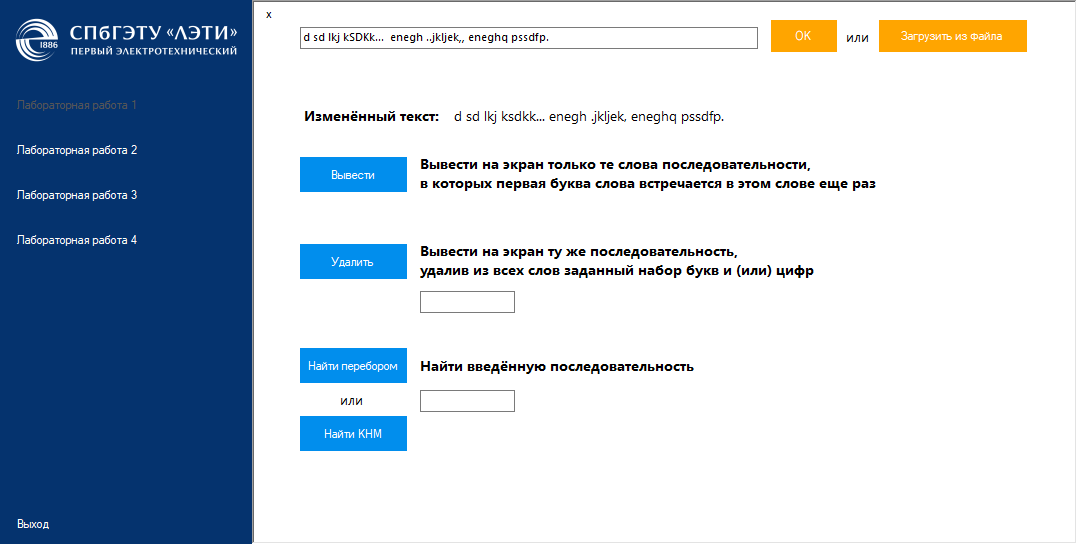


Рисунок 3 – Рабочий интерфейс

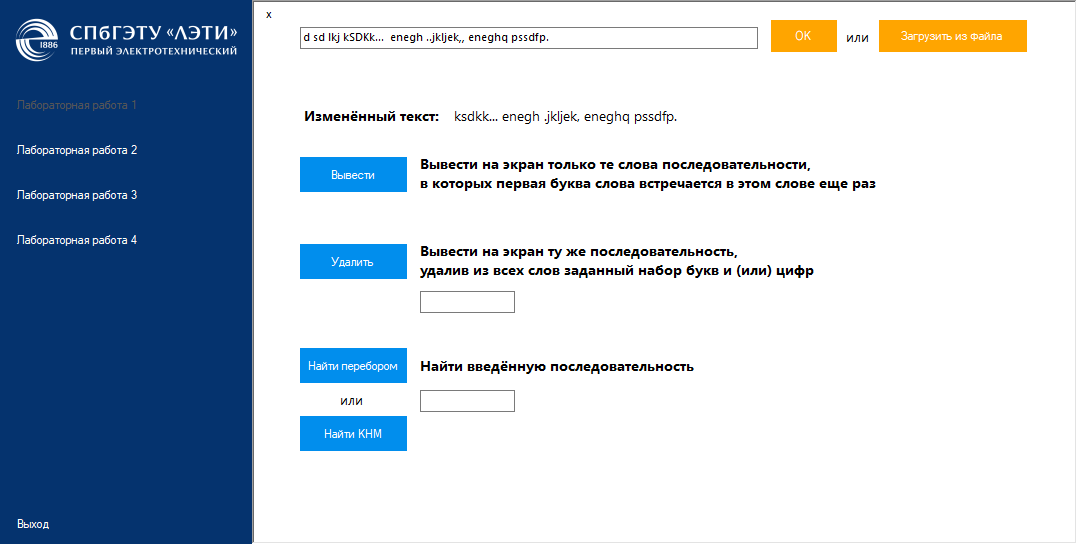


Рисунок 4 – Выполнение задания на вывод только тех слов, в которых первая буква повторяется ещё и в самом слове

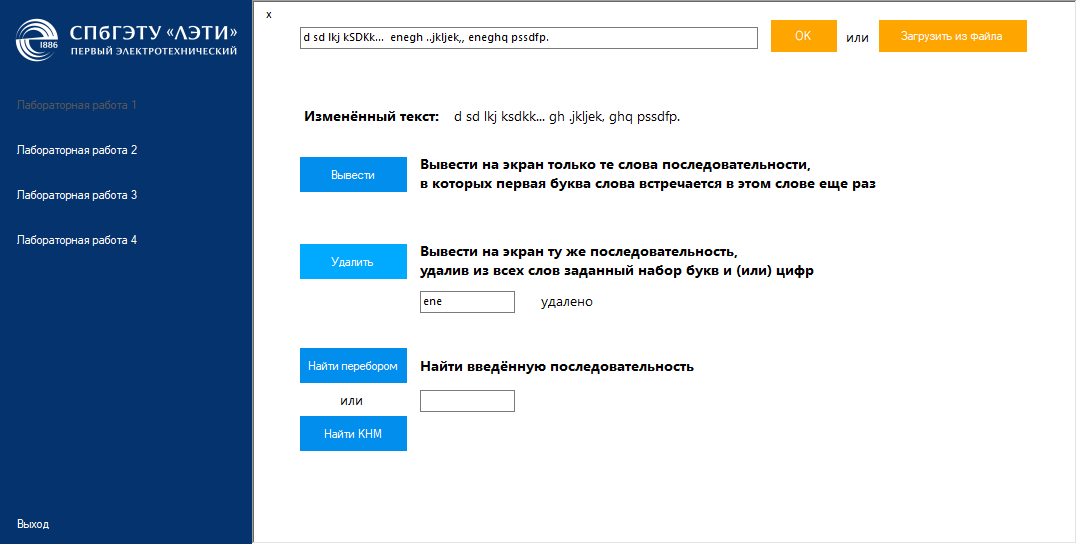


Рисунок 5 – Удаление указанной пользователем подстроки

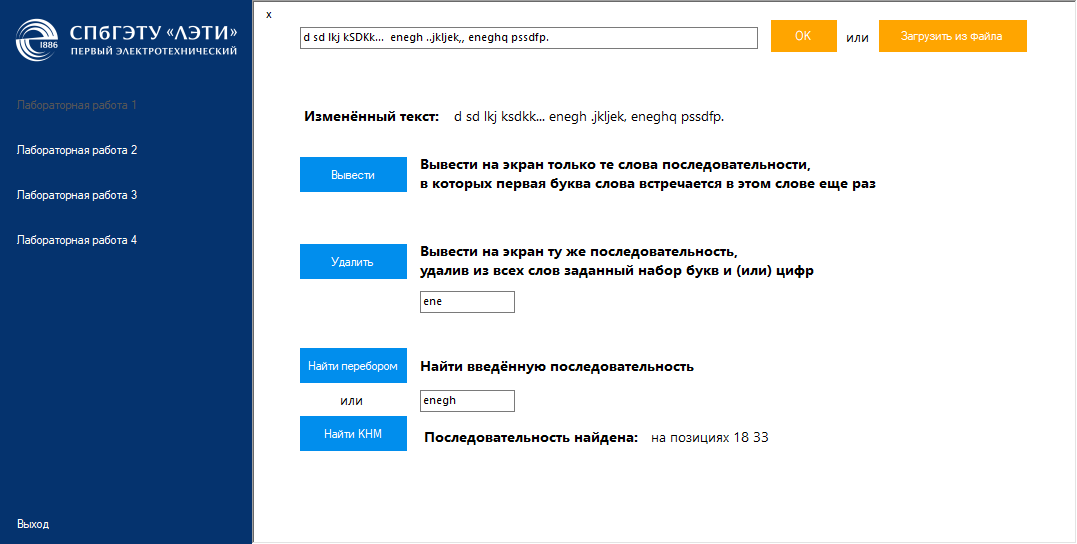


Рисунок 6 – результат работы поиска

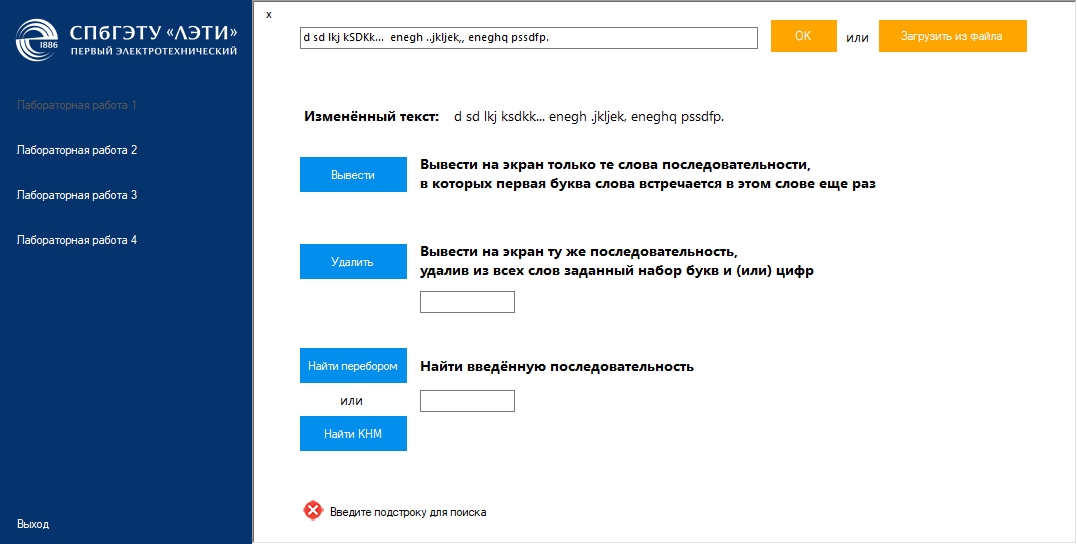


Рисунок 7 – вывод ошибки