МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

ОТЧЕТ

по практической работе №4

по дисциплине «Программирование»

Тема: Текстовые строки как массивы символов

Студентка гр. 0324 Косенко А.Р.

Преподаватель Глущенко А.Г.

Санкт-Петербург

2020

Цель работы.

Изучение способов обработки текстовых данных. Получение практических навыков работы с файлами. Изучение различных алгоритмов поиска подстроки в строке.

Основные теоретические положения.

Работа с файлами

Для работы с файлами в языке C++ используют потоки трех видов:

- поток ввода (класс ifstream);
- поток вывода (класс ofstream);
- поток ввода-вывода (класс fstream);

Класс ifstream используется для выполнения чтения данных из файлов. Поток ofstream – для записи данных в файл. Поток fstream – для чтения и записи данных в файл.

Открыть файл можно двумя способами:

1) сначала создать поток, а потом связать его с файлом:

```
ofstream File;
File.open("D:\\test.txt");
```

2) создать поток и связать поток с файлом:

```
ofstream File ("D:\\test.txt");
```

После того, как поток был связан с файлом, необходимо обязательно проверить открылся ли файл. Если файл открыть не удалось, то переменная потока (File) принимает значение false, если файл открыт – true.

Проверку успешного открытия файла можно выполнить с помощью условного оператора:

```
If (!File) // другой вариант
if (!File.is_open()) cout << "Ошибка при открытии файла!";
```

Функция потока is_open() возвращает логическое значение в зависимости от результата операции открытия файла. Файл закрывается с помощью функции потока close(): File.close(); 5 Каждый поток использует свой вариант функции open. Первый параметр определяет имя открываемого файла (представляет собой массив символов). Второй параметр определяет режим открытия

файла. Этот параметр имеет значение по умолчанию, то есть является необязательным. Возможные значения этого параметра:

- ios::app – при открытии файла на запись (поток ofstream) обеспечивает добавление всех выводимых в файл данных в конец файла;
- ios::ate – обеспечивает начало поиска данных в файле начиная с конца файла;
- ios::in – файл открывается для чтения из него данных;
- ios::out – файл открывается для записи данных в файл;
- ios::binary – открытие файла в двоичном режиме (по умолчанию все файлы открываются в текстовом режиме);
- ios::trunc – содержимое открываемого файла уничтожается (его длина становится равной 0). Эти флаги можно комбинировать с помощью побитовой операции ИЛИ (|). Если файл открывается без использования функции open, эти флаги тоже можно использовать:

```
fstream File ("E:\\test.txt", ios :: out | ios :: app);
```

Файл открывается на вывод с добавлением записываемых данных в конец файла. Запись и чтение данных в текстовых файлах ничем не отличается от способов ввода-вывода данных с помощью потоков cin и cout. Методы форматирования вывода и вводы данных остаются такими же (флаги форматирования, манипуляторы, функции потоков). Необходимо помнить, что при использовании операции >> для чтения данных из текстовых файлов, процесс чтения останавливается при достижении пробельного символа. Поэтому для чтения строки текста, содержащей несколько слов, необходимо использовать, например, функцию getline().

Определение текстовой строки

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:

```
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:

```
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'.

Ввод текстовых строк с клавиатуры

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Вывод текстовых строк на экран крайне простая задача:

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

Ввод текста с клавиатуры можно осуществлять разными способами, каждый из которых имеет определенные особенности.

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:

```
const int N = 21; char Str [N];
cin.getline (Str, N); // Пусть введена строка “Это пример текста”
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.

Если забирать остатки данных из потока ввода не надо, то следует очистить его с помощью функции cin.sync().

Класс string

Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ '\0'). Класс string был введен как альтернативный вариант для работы со строками типа char.

Чтобы использовать возможности класса string, нужно подключить библиотеку и пространство имен std. Объявление же переменной типа string осуществляется схоже с обычной переменной:

```
string S1; // Переменная с именем s1 типа string
string S2 = "Пример"; // объявление с инициализацией
```

Создание нового типа string было обусловлено недостатками работы с строками символов, который показывал тип char. В сравнении с этим типом string имеет ряд основных преимуществ:

• возможность использования для обработки строк стандартные операторы C++ (=, +, , +=, ! =, <=, >=,[])Использование типа char приводило требовало написание чрезмерного программного кода;

• обеспечение лучшей надежности программного кода;

• обеспечение строки, как самостоятельного типа данных.

Класс string обладает широким функционалом:

· функция compare() сравнивает одну часть строки с другой;

• функция length() определяет длину строки;

• функции find() и rfind() служат для поиска подстроки в строке (отличаются функции лишь направлением поиска);

• функция erase() служит для удаления символов;

• функция replace() выполняет замену символов;

• функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;

• Аналогом использования оператора + является функция append();

• Аналогом использования оператора = является функция assign();

Но весь функционал string накладывает и свой негативный отпечаток. Основным недостатком string в сравнении с типом char является замедленная скорость обработки данных.

Поиск подстроки в строке

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.

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.

1-й шаг цикла	**H**	E	L	L	O		W	O	R	L	D
	L	O									
2-й шаг цикла	H	**E**	L	L	O		W	O	R	L	D
		L	O								
3-й шаг цикла	H	E	**L**	**L**	O		W	O	R	L	D
			L	**O**							
4-й шаг цикла	H	E	L	**L**	**O**		W	O	R	L	D
				L	**O**						

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.

ТАБЛИЦА ВЛЮЧЕНИЙ

0	1	**2**	**3**	4	5	6	7	8	**9**	10
H	E	L	L	O		W	O	R	L	D

	0	1	**2**	**3**	4	5	6	7	8	**9**	10
1-й шаг цикла	H	E	**L**	**L**	O		W	O	R	L	D
			L	**O**							
2-й шаг цикла	H	E	L	**L**	**O**		W	O	R	L	D
				L	**O**						

Рисунок 1.2 – пример работы алгоритма КМП

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:

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

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

3) 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.

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

	0	1	**2**	**3**	4	5	6	7	8	**9**	10
1-й шаг цикла	H	E	L	**L**	O		W	O	R	L	D
	E	L	L	**O**							
2-й шаг цикла	H	**E**	**L**	**L**	**O**		W	O	R	L	D
		E	**L**	**L**	**O**						

Таблица отступов

3	1	1	4
E	L	L	O

Длина подстроки: 4

Рисунок 1.3 – Пример работы алгоритма Бойра-Мура

Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

Постановка задачи.

1) С клавиатуры или с 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 регистра букв: пРиМЕр);

3) Вывести на экран только те слова последовательности, в которых встречаются одинаковые буквы.
4) Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.
5) Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КНМ, а нечетные – Бойера-Мура. (*)

Выполнение работы.

1) Вводила текст при помощи **getline(cin, a).** Далее создала фильтр для возможных символов, которые включали в себя : ,.:;?!-()'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ. Проводила проверку на количество слов(для этого создала счётчик слов). А также через size() узнавала длину слова. Далее объединила все эти пункты в одно условие if.

2)Редактировала текст, проходя по строке через цикл: пока встречались одинаковые знаки препинания, удаляла при помощи функции .erase.

3) Создала условие для обнаружения слов и счётчик для него. Далее проходилась по словам и при обнаружении одинаковых букв(за это отвечал другой счётчик) выводила слово.
4) Пользователь вводит последовательность , которую хочет удалить и количество символов , которая содержится в последовательности. Используя функцию erase, удаляю последовательность с позиции до указанной длины.

Вывод.

Изучены способы обработки текстовых данных. Получены навыки работы с файлами. Изучен алгоритм поиска подстроки в строке.

Примеры работы.

```
!!!далее вводите слова на латинице!!!
Введите строку: 123 timp tomp  !!!! nIna
Выберите задание:

1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
4. Выход
>>> 1
Неотредактированный текст: 123 timp tomp  !!!! nIna
Текст после редактирвоания: 123 timp tomp ! nina
```

```
1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
4. Выход
>>> 2
Слова, в которых повторяются буквы: nina

Выберите задание:
```

```
Какую полседоватльность нужно удалить: mp

Теперь последовательность выглядит так: 123 tri tro ! nina
Выберите задание:

1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
```