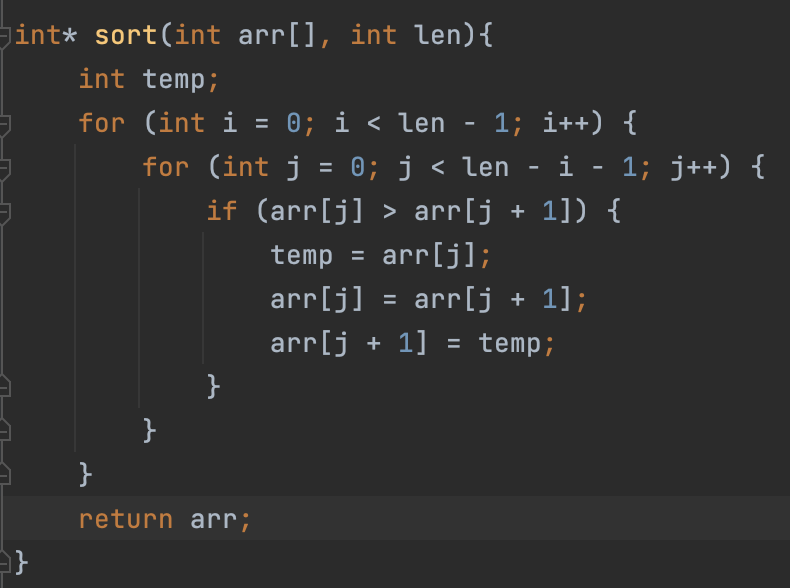
Документация к Лабораторной работе №5.

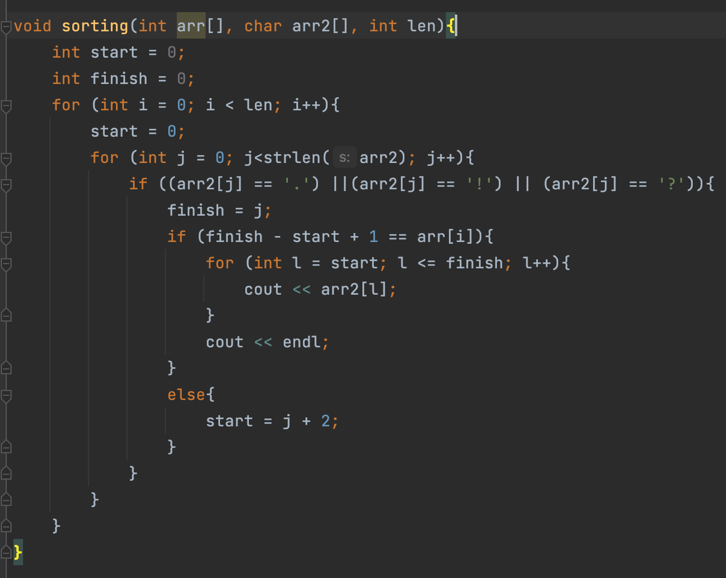
Сорочайкин А.Я. СКБ223.

- Функция Sort. Sortint.cpp

Алгоритм сортировки массива целых чисел. Также известный как «метод пузырька». Суть в следующем. Последовательно перебираются все элементы массива и сравниваются значения двух соседних элементов. Если значение элемента слева больше значения элемента справа, то они «обмениваются» значениями. В результате одного прохода по массиву, наибольшее значение окажется в самой правой ячейке. После следующего перебора, второе по величине значение окажется в предпоследней ячейке. В результате, после n-1 перебора мы получим массив, отсортированный по возрастанию.

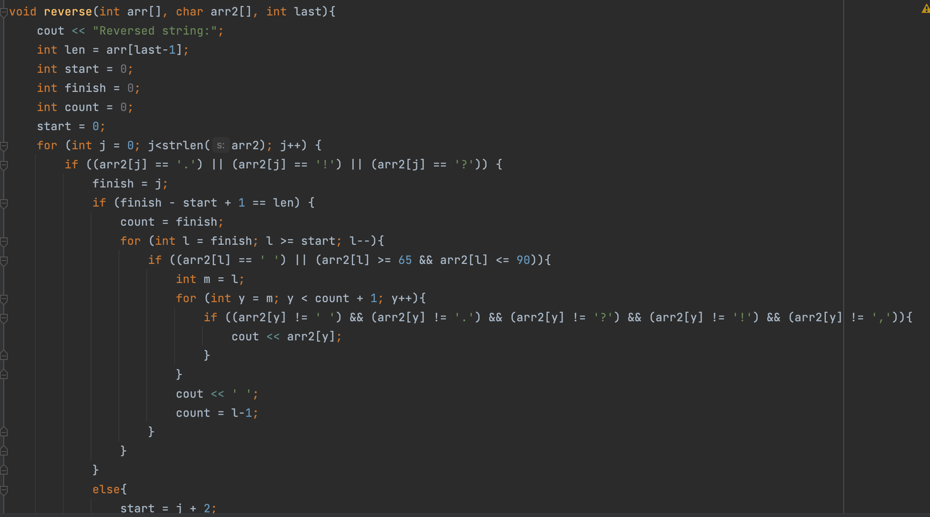
- Функция sorting. sortchar.cpp

Суть алгоритма заключается в следующем. На вход мы получаем массив char’ов, с текстом, массив целых чисел, в котором хранятся отсортированные по возрастанию длины предложений из текста. Последовательно проходясь по длинам предложений начиная с 0ой ячейки, мы ищем в тексте предложения с длиной равной значению длины из массива целых чисел, и выводим найденное предложение. В результате работы алгоритма мы получаем выведенные на экран предложения из текста, отсортированные по возрастанию.

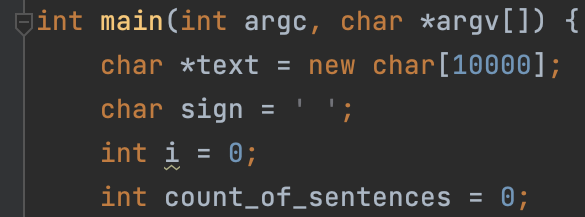


- Функция reverse. reverse.cpp

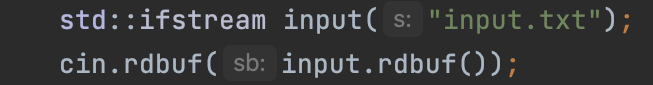
На вход получаем длину наибольшего предложения и массив со всем текстом. Аналогично предыдущей функции, находим начало и конец нужного нам предложения. Далее, начиная с конца предложения, проходимся по массиву, в поисках пробела или заглавной буквы. Найдя нужную нам ячейку выводим подряд все символы от нее, до последнего. Выведя последнее слово предложения, меняем значение окончания перебора, на данное слово. Т.е. пробежали с конца, нашли первое целое слово. Вывели его. Теперь для алгоритма, данное слово отсутствует, вместо него окончание предложения. Таким образом выводим все слова в нужном нам предложении с конца, пока предложение не закончится.



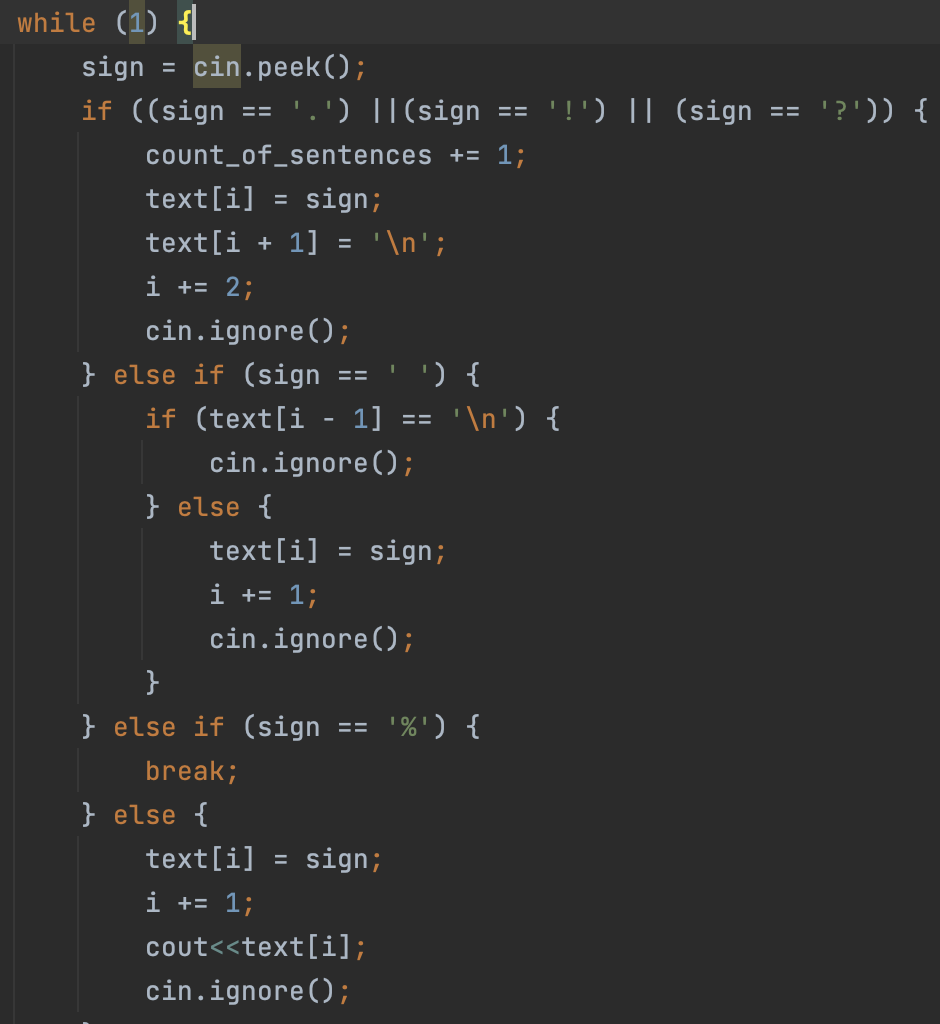
- Функция main. Maim.cpp.

Создаем символьный массив, в который будет заноситься текст. Переменная sign будет отображать знак, который в данный момент считывается с консоли. Переменная i обозначает индекс массива text в который будет занесен следующий символ. Переменная amount\_of\_sentences отвечает за количество предложений в нашем тексте.

Далее, проверяем введенные флаги. Если среди считанных флагов присутствует флаг –fromfile, тогда поток ввода с файла меняем на косноль. Т.е. программа воспринимает ввод с файла, как ввод с консоли, что позволяет нам работать с данными, подаваемыми с файла, аналогично данным с консоли.

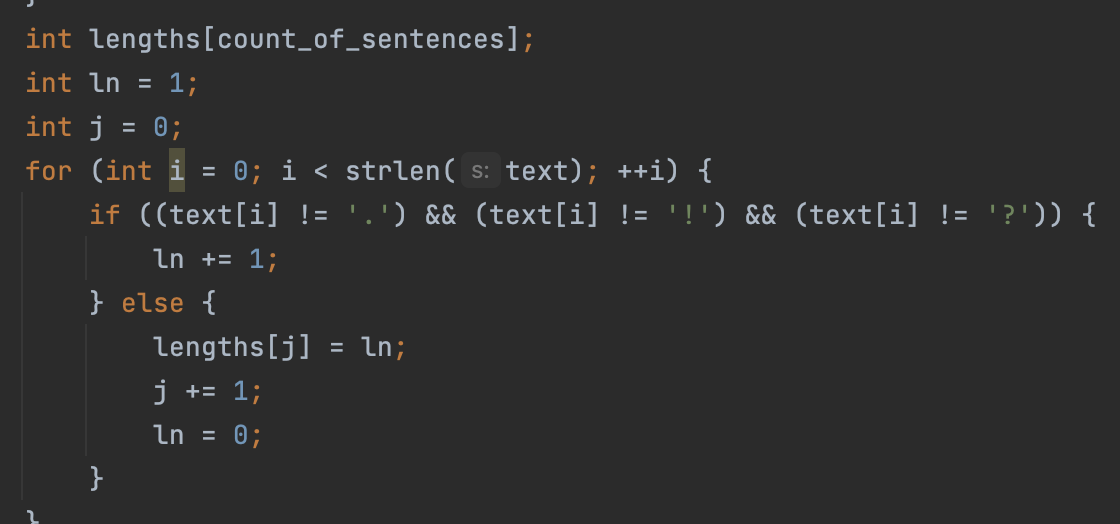


После проверки флагов, начинаем считывать данные с консоли в массив с текстом. В процессе считывания, получаем информацию о количестве предложений в введенном тексте.

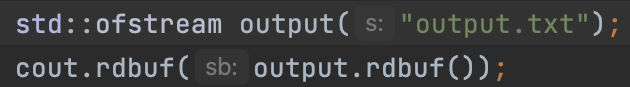


С помощью функции cin.peek, передаем переменной рассматриваемый знак. Далее, в зависимости от его значения, производим добавление в массив с различными условиями. Так как цикл считывания бесконечный, нам нужно условие, которое позволит его закончить. В моем случае, условием окончания считывания является знак «%». Т.е. пользователь, после окончания ввода должен поставить данный символ.

Далее, одним пробегом по массиву считаем длину каждого предложения и заносим полученные длины в массив. После получения массива с длинами, вызываем функцию sort, которая сортирует наш массив по возрастанию.



Следующим шагом проверяем наличие флага –tofile. Если данный флаг присутсвует, то аналогично вводу, меняем поток вывода с файла на консоль. Т.е. текст, выводимый далее и находящийся в той же области видимости будет выводиться в указанный файл, тем же способом, каким он выводится на консоль.



Далее, вызывается функция sorting, которая выводит полученные предложения по возрастанию их длины.

Дальше, выполняется задание по выведению слов из наибольшего предложения в обратном порядке. Для этого, опять меняем поток вывода, и вызываем функцию reverse, которая выводит в файл Nine.txt наибольшее предложение с обратным порядком слов.

