Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Новгородский государственный университет
имени Ярослава Мудрого»

П. М. Довгалюк

Практикум на языке C++

Великий Новгород
2021

ББК 32.973.26-018.2
Д58

Печатается по решению
РИС НовГУ

Рецензенты:
рецензент 1
рецензент 2

Довгалюк, П. М.
Д58 Практикум на языке C++ / П. М. Довгалюк; НовГУ им. Ярослава Мудрого. – Великий Новгород, 2021. – xx с.

...

Пособие предназначено для студентов первых курсов технических специальностей.

ББК 32.973.26-018.2

Оглавление

Раздел 1

Строки

1.1 Изограммы

Изограмма — это слово (или фраза), в котором не повторяются буквы.

Напишите программу, которая проверяет, является ли введённая строка изограммой. Строка состоит из множества латинских букв в разном регистре, пробелов и знаков препинания. Причём в изограмме пробелы или знаки препинания могут повторяться.

При реализации используйте следующие конструкции:

- Функцию `std::isalpha`
- Функцию `std::tolower` или `std::toupper`

Контрольные вопросы

В вопросах подразумевается, что для входных данных используется кодировка ASCII.

- Какой фрагмент кода мог бы заменить функцию `std::isalpha`?
- Как можно реализовать функцию `std::tolower`?

1.2 Боб

Боб не очень любит разговаривать, поэтому использует небольшой набор реплик.

Он отвечает «Sure.» на любой вопрос, например «How are you?».

Он говорит «Whoa, chill out!», если вы КРИЧИТЕ НА НЕГО (то есть используете одни заглавные буквы).

Он отвечает «Calm down, I know what I'm doing!» на вопрос, в котором вы кричите.

Он говорит «Fine. Be that way!» если вы обращаетесь к нему, но ничего не говорите (это значит, что входная строка программы состоит из пробельных символов).

Он говорит «Whatever.» во всех остальных случаях.

Напишите диалоговую программу, которая имитирует разговор с Бобом. На каждую входную строку она должна выводить такой ответ, какой давал бы Боб в такой же ситуации. Входные реплики для программы должны подчиняться правилам пунктуации английского языка.

Контрольные вопросы

- Каким способом проверяется, что все буквы во введённой строке заглавные?
- Какую функцию стандартной библиотеки можно использовать для проверки пробельных символов?

1.3 Шифровальный квадрат

Реализуйте описанный ниже алгоритм шифрования текста. На вход ваша программа будет получать текст на английском языке, а на выходе должна быть зашифрованная версия этого текста.

При шифровании удаляются все знаки препинания и пробелы, а буквы приводятся к нижнему регистру.

Затем нормализованный таким образом текст разбивается на строки. Эти строки можно выстроить в виде прямоугольника.

Например, предложение «If man was meant to stay on the ground, god would have given us roots.» нормализуется в виде:

```
ifmanwasmeanttostayonthegroundgodwouldhavegivenusroots
```

Полученный текст перестраивается в виде прямоугольника размером в r строк и c столбцов, причём $c \geq r$ и $c - r \leq 1$.

Таким образом, текст из примера выше должен быть преобразован в следующий прямоугольник из 8 столбцов и 7 строк:

```
"ifmanwas"
"meanttos"
"tayonthe"
"groundgo"
"dwouldha"
"vegivenu"
"sroots  "
```

Закодированное сообщения получается чтением сверху вниз всех столбцов от левого до правого. Для нашего примера получится следующий шифр:

```
imtgdvsfearwermayoogoanouuiontnnlvtwttddesaohghnsseoau
```

Выведите полученный текст фрагментами, разделяя их пробелами. Всего должно быть c фрагментов длиной r. Если для заполнения прямоугольника $r \times c$ не хватает n символов, дополните каждый из n последних фрагментов одним пробелом в конце, вот так:

```
"imtgdvs fearwer mayoogo anouuio ntnnlvt wttddes aohghn  sseoau "
```

Если снова расположить эти фрагменты в прямоугольнике, их можно аналогичным образом расшифровать:

```
"imtgdvs"
"fearwer"
"mayoogo"
"anouuio"
"ntnnlvt"
"wttddes"
"aohghn "
"sseoau "
```

Рекомендации к заданию

При написании программы попробуйте обойтись без дополнительной структуры данных, в которой хранился бы прямоугольник.

Контрольные вопросы

- Какой максимальный объём текста может зашифровать ваша программа за один раз?
- Как хранится в памяти прямоугольник, который выводится на экран?
- В каком порядке обрабатываются символы нормализованной строки для вывода прямоугольника?

1.4 Подстроки

На входе в программу поступает строка из цифр и число n. Программа должна вывести все непрерывные подстроки исходной строки длины n.

Например, для строки 49142 и $n = 3$ вывод будет таким: 491, 914, 142.

А для $n = 4$ таким: 4914, 9142.

Контрольные вопросы

- Какую стандартную функцию можно использовать для получения подстрок?

Раздел 2

std::array

2.1 Массивы-значения

Сравните две программы ниже. Предположите, какие значения выводятся в каждой из них, а затем попробуйте запустить их и проверить результат.

```
1  #include <iostream>
2
3  typedef int MyArr[5];
4
5  void f(MyArr a)
6  {
7      a[4] = 0;
8  }
9
10 int sum(MyArr a)
11 {
12     int r = 0;
13     for (int i = 0 ; i < 5 ; ++i)
14         r += a[i];
15
16     return r;
17 }
18
19 int main()
20 {
21     MyArr a = {1, 2, 3, 4, 5};
```

```
22      std::cout << "Sum1: " << sum(a) << '\n';
23      f(a);
24      std::cout << "Sum2: " << sum(a) << '\n';
25  }
```

```
 1  #include <iostream>
 2  #include <array>
 3
 4  typedef std::array<int, 5> MyArr;
 5
 6  void f(MyArr a)
 7  {
 8      a[4] = 0;
 9  }
10
11  int sum(MyArr a)
12  {
13      int r = 0;
14      for (int i = 0 ; i < 5 ; ++i)
15          r += a[i];
16
17      return r;
18  }
19
20  int main()
21  {
22      MyArr a = {1, 2, 3, 4, 5};
23      std::cout << "Sum1: " << sum(a) << '\n';
24      f(a);
25      std::cout << "Sum2: " << sum(a) << '\n';
26  }
```

Контрольные вопросы

- Что выведут первая и вторая программы?
- Как можно было бы доработать вторую, чтобы получался такой же результат, как и в первой?
- Какое преимущество даёт использование `std::array`?
- Что нужно изменить, чтобы не переделывать функцию `sum` при изменении размера массива?

Раздел 3

std::vector

3.1 new[]/delete[] vs std::vector

Перепишите следующую программу, используя `std::vector`. Последовательно избавьтесь от следующих конструкций:

- Операторы `new[]/delete[]` (переменные `a` и `b` превратите в объекты `std::vector`.
- Переменная `m` и цикл для вычисления её значения.

```
1  #include <iostream>
2
3  int main()
4  {
5      int n;
6      std::cin >> n;
7      int *a = new int[n];
8      for (int i = 0 ; i < n ; ++i)
9          std::cin >> a[i];
10
11     int k;
12     std::cin >> k;
13     int m = 0;
14     for (int i = 0 ; i < n ; ++i)
15         if (a[i] < k)
16             ++m;
17
```

```
18      int *b = new int[m];
19      m = 0;
20      for (int i = 0 ; i < n ; ++i)
21          if (a[i] < k)
22              b[m++] = a[i];
23
24      delete[] a;
25      for (int i = 0 ; i < m ; ++i)
26          std::cout << b[i] << ' ';
27      delete[] b;
28  }
```

Контрольные вопросы

- Что делает исходная программа?
- Какие ограничения неявно наложены на значение переменной `n`?
- На сколько строк удалось сократить исходный код, используя `std::vector`?
- Когда освобождается память, используемая `std::vector`?

Раздел 4

std::list

4.1 Список вместо массива

Проанализируйте следующую программу. Попробуйте выяснить, как время её работы растёт с увеличением входного параметра n.

```
1  #include <iostream>
2
3  int main()
4  {
5      int n;
6      std::cin >> n;
7      int *a = new int[n];
8      for (int i = 0 ; i < n ; ++i)
9      {
10         int v;
11         std::cin >> v;
12         for (int j = i ; j > 0 ; --j)
13             a[j] = a[j - 1];
14         a[0] = v;
15     }
16
17     for (int i = 0 ; i < n ; ++i)
18         std::cout << a[i] << ' ';
19     std::cout << '\n';
20     delete[] a;
21 }
```

Перепишите программу с использованием контейнера `std::list` вместо массива. Для вставки значений в список используйте функцию `std::list::insert`.

Контрольные вопросы

- В чём заключается ошибка программиста, который написал исходную программу?
- Какими способами можно просмотреть все элементы списка, чтобы вывести их на экран?

Раздел 5

Алгоритмы

5.1 std::accumulate

Переделайте функции `sum` в программах из задания 2.1, используя алгоритм `std::accumulate`. Для этого потребуется подключить заголовочный файл `numeric`.

Контрольные вопросы

- Как переделать вызов `std::accumulate`, чтобы вычислялась сумма элементов из первой половины массива?

5.2 Снова std::accumulate

Переделайте программы из предыдущего задания, чтобы вместо суммы функция `std::accumulate` считала произведение элементов.

Контрольные вопросы

- Какое значение должно получиться, если в массиве не будет ни одного элемента?

5.3 Сортировка

Напишите программу, которая вводит с клавиатуры последовательность чисел. Длина последовательности тоже задаётся пользователем. Затем реализуйте свой любимый алгоритм сортировки для того, чтобы упорядочить эту последовательность по возрастанию.

Ваша программа не должна ограничивать длину последовательности размером используемой структуры данных для её хранения.

Контрольные вопросы

- Какой алгоритм сортировки вы применили?
- Можно ли этим алгоритмом отсоритировать за разумное время последовательность из 1000000 элементов?
- Как изменить порядок сортировки в вашей программе?

5.4 std::sort

Замените сортировку из программы для предыдущего задания на функцию `std::sort` из стандартной библиотеки. Для этого понадобится заголовочный файл `algorithm`.

Контрольные вопросы

- Какое хранилище использовалось для исходной последовательности в вашей программе?
- Как можно задать желаемый порядок элементов в результате сортировки?

5.5 Двоичный поиск в массиве

Ещё раз доработайте предыдущую программу. Пусть она запрашивает несколько чисел, которые нужно поискать в массиве. Для каждого нужно вывести нашлось оно или нет.

Сравните скорость работы вашего алгоритма поиска и стандартной функции `std::binary_search`.

Контрольные вопросы

- Сколько итераций цикла может выполнить двоичный поиск для массива из 10 элементов?

Раздел 6

Структуры данных

6.1 Двоичное дерево

Напишите программу, которая выводит на экран графическое представление двоичного дерева. Дерево состоит из N вершин, пронумерованных от 1 до N.

На вход программы кроме числа N поступает также информация о структуре дерева в виде последовательности пар чисел, кодирующих связи. В каждой паре первое число обозначает номер предка, а второе — номер связанного с ним потомка.

Дерево нужно вывести, обозначая вершины соответствующими числами, а связи с помощью символов «прямой слэш» и «обратный слэш».

Пример вывода дерева:

```
      4
    /   \
   /     \
  2       6
 / \     / \
1   3   5   7
```

Это дерево может быть закодировано с помощью следующих входных данных:

```
7
4 2
```

```
4 6
2 1
6 5
2 3
6 7
```

Контрольные вопросы

- Что такое двоичное дерево?
- Как хранится в вашей программе двоичное дерево?

6.2 Двоичное дерево поиска

Доработайте предыдущую программу, чтобы она проверяла, является ли дерево из входных данных двоичным деревом поиска.

В двоичном дереве поиска узлы упорядочены таким образом, что значение любого левого потомка меньше, чем значение корня, а значение любого правого потомка больше, чем корень.

Контрольные вопросы

- Какая может быть максимальная высота двоичного дерева поиска из N вершин?

6.3 Двоичный поиск в дереве

Ещё раз доработайте программу из предыдущего задания. Теперь на вход поступают двоичное дерево поиск и последовательность чисел, которую надо в нём найти.

Для каждого числа во входной последовательности выведите путь до него в двоичном дереве.

Контрольные вопросы

- Сохраняет ли программа в памяти путь, который будет выводиться?

6.4 Кольцевой буфер

Реализуйте кольцевой буфер с использованием статического массива или контейнеров `std::array` и `std::vector`.

Контрольные вопросы

- Можно ли использовать все возможные ячейки в памяти, используемой под буфер?
- Какую абстрактную структуру данных можно реализовать с помощью кольцевого буфера?

Раздел 7

Хеширование

7.1 Анаграммы

Анаграмма — это слово, полученное из исходного перестановкой букв. Напишите программу, которая определяет, какие слова из списка являются анаграммами заданного слова.

Например, для слова «listen» и списка «enlists», «google», «inlets», «banana» программа должна вывести список из одного элемента: «inlets».

Предварительный отсев слов-кандидатов выполняйте с помощью вычисления хеш-функции от этих строк.

Контрольные вопросы

- Какую хеш-функцию вы выбрали?
- Можно ли для тех же целей использовать стандартную реализацию `std::hash<std::string>`?
- Что кроме хеш-функции нужно использовать, чтобы убедиться, что буквы в строках совпадают?

7.2 Коллизии

Найти все анаграммы из слов-кандидатов для всех подмножеств букв из исходного слова.

Получить подмножества букв можно несколькими способами:

- Рекурсивный перебор. На каждом шаге обрабатывается одна буква входного слова. Для каждого варианта (когда буква включается в результирующее слово и когда не включается) рекурсивный поиск продолжается со следующей буквой.
- Последовательно увеличивающиеся целые числа можно раскладывать на двоичные разряды. Каждый разряд сопоставляется одной букве. Если разряд 1, то буква включается в результирующее подмножество.

Для хранения списка слов-кандидатов используйте контейнер `std::unordered_set`. Этот контейнер использует хеширование для поиска и упорядочивания элементов. Используйте второй параметр шаблона контейнера для указания собственной хеш-функции, чтобы сравнивать строки без учёта порядка букв. После генерации очередного подмножества букв можно проверять, если ли уже слово с такими буквами в контейнере.

Проанализируйте, как часто возникают коллизии, если использовать только хеширование, без сложной функции сравнения строк.

Контрольные вопросы

- С чем связано, что строки из разных букв могут получать одни и те же значения хеш-функции?
- Приведите пример двух строк из разных букв, но с одинаковым значением хеш-функции.

Раздел 8

std::map

8.1 Поиск

Перепишите следующую программу, используя `std::map`.
Выполните следующие действия:

- Замените массивы `word` и `def` на переменную-контейнер `std::map`.
- Замените алгоритм поиска на функцию `find` контейнера `std::map`.

```
1  #include <iostream>
2  #include <string>
3
4  int main()
5  {
6      int n;
7      std::cin >> n;
8      std::string *word = new std::string[n];
9      std::string *def = new std::string[n];
10     for (int i = 0 ; i < n ; ++i)
11         std::cin >> word[i] >> def[i];
12
13     std::string s;
14     while (std::cin >> s)
15     {
16         bool found = false;
17         for (int i = 0 ; !found && i < n ; ++i)
18         {
```

```
19                  if (word[i] == s)
20                  {
21                      found = true;
22                      std::cout << def[i] << '\n';
23                  }
24              }
25              if (!found)
26                  std::cout << "Not found\n";
27          }
28
29          delete[] word;
30          delete[] def;
31      }
```

Контрольные вопросы

- Что делает исходная программа?
- Какой тип ключа применяется в использованном контейнере `std::map`?
- Что возвращает функция `find`, если нужный элемент не найден?

8.2 Подсчёт слов

Напишите программу, которая выводит сколько раз каждое слово из входной строки повторяется в ней.

Слова могут состоять из цифр и латинских букв. Слова с разным регистром букв считаются одинаковыми. Знаки препинания, пробелы, переносы строк и символы табуляции нужно игнорировать.

Подзадачи

- Используйте `std::map<std::string, int>` для хранения информации о количестве повторов.
- Попробуйте не считывать весь текст целиком, а читать по одному символу из входного потока.

- Напишите два варианта программы: с преобразованием хранимых в `std::map` слов и без него. Для второго варианта замените функцию сравнения (третий параметр шаблона) на собственную.

Контрольные вопросы

- Определён ли порядок элементов в `std::map`? Можно ли его изменить?