**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ**

**КАФЕДРА УПРАВЛЕНИЯ И ИНТЕЛЛЕКТУАЛЬНЫХ ТЕХНОЛОГИЙ**

**Лабораторная работа №4**

По дисциплине «Разработка программного обеспечения систем управления»

Выполнил: студент I курса

Группы А-03-19

Синицин Д.И.

Проверил:

Москва 2020

1. Постановка задачи.

Добавить возможность построения гистограммы по данным из файла из сети. Адрес файла задается аргументом командной строки программы. Если адрес не задан, читать данные со стандартного ввода, как раньше.

Работу нужно вести на основе кода общего задания к ЛР № 3 в старом репозитарии в отдельной ветке без создания нового проекта. По этой причине во всех примерах используется lab03.exe.

Пример строки запуска:

lab03.exe http://uii.mpei.ru/study/courses/cs/lab03/marks.txt >marks.svg

Вариант 17:

Устанавливайте опцию CURLOPT\_FAILONERROR и анализируйте результат curl\_easy\_perform(), чтобы проверить результат загрузки файла, а не выполнения запроса. Опишите в отчете способ тестирования.

1. Описание логики решения.

Добавляем к коду ЛР№3 возможность считывания входных данных из сети. Для этого вводим argc (количество аргументов функции main) и argv (массив аргументов). При наличии более одного аргумента считываем данные из сети. Также добавляем вывод версии Windows и имя компьютера.

При добавлении опции CURLOPT\_FAILONERROR мы проверяем результат загрузки файла. То есть, если файл загружен, программа считает данные и отработает. Если же файл не был найден или произошла какая-то ошибка, в окно вывода будет сообщено об этом пользователю.

1. Ссылка на репозитарий (GitHub).

[Ссылка на GitHub](https://github.com/SinitsinDI/lab4/tree/lab4)

1. Код программы.
2. main.cpp
3. #include <iostream>
4. #include <vector>
5. #include <sstream>
6. #include <string>
7. #include "histogram.h"
8. #include "svg.h"
9. #include <curl/curl.h>
10. using namespace std;
11. Input read\_input(istream& in, bool prompt)
12. {
13. Input data;
14. if (prompt)
15. cerr << "Enter number count: ";
16. size\_t number\_count;
17. in >> number\_count;
18. if (prompt)
19. cerr << "Enter numbers: ";
20. data.numbers = input\_numbers(in, number\_count);
21. if (prompt)
22. cerr << "Enter column count: ";
23. in >> data.bin\_count;
24. return data;
25. }
26. size\_t
27. write\_data(void\* items, size\_t item\_size, size\_t item\_count, void\* ctx)
28. {
29. size\_t data\_size = item\_size \* item\_count;
30. stringstream\* buffer = reinterpret\_cast<stringstream\*>(ctx);
31. (\*buffer).write(reinterpret\_cast<const char\*>(items), data\_size);
32. return data\_size;
33. return 0;
34. }
35. Input download(const string& address) {
36. stringstream buffer;
37. curl\_global\_init(CURL\_GLOBAL\_ALL);
38. CURL\* curl = curl\_easy\_init();
39. if(curl) {
40. CURLcode res;
41. curl\_easy\_setopt(curl, CURLOPT\_URL, address.c\_str());
42. curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_data);
43. curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &buffer);
44. curl\_easy\_setopt(curl, CURLOPT\_FAILONERROR, 1L);//variant 17
45. res = curl\_easy\_perform(curl);
46. //if (res) { //without a variant
47. if (res == CURLE\_HTTP\_RETURNED\_ERROR) { //variant 17
48. cerr << curl\_easy\_strerror(res) << endl;
49. exit(1);
50. }
51. }
52. curl\_easy\_cleanup(curl);
53. return read\_input(buffer, false);
54. }
55. int main(int argc, char\* argv[])
56. {
57. Input input;
58. if (argc > 1)
59. {
60. input = download(argv[1]);
61. }
62. else
63. {
64. input = read\_input(cin, true);
65. }
66. const auto bins = make\_histogram(input);
67. show\_histogram\_svg(bins);
68. return 0;
69. }
70. Svg.cpp
71. #include "svg.h"
72. #include "WinVersion.h"
73. #include <iostream>
74. double bin\_opacity17(double bin, size\_t max\_count)
75. {
76. if(max\_count==0)
77. return 0;
78. return bin/max\_count;
79. }
80. void svg\_begin(double width, double height)
81. {
82. cout << "<?xml version='1.0' encoding='UTF-8'?>\n";
83. cout << "<svg ";
84. cout << "width='" << width << "' ";
85. cout << "height='" << height << "' ";
86. cout << "viewBox='0 0 " << width << " " << height << "' ";
87. cout << "xmlns='http://www.w3.org/2000/svg'>\n";
88. }
89. void svg\_end()
90. {
91. cout << "</svg>\n";
92. }
93. void svg\_rect(double x, double y, double width, double height, string stroke = "black", string fill = "black", double fill\_opacity=1.0)
94. {
95. cout << "<rect x='"<<x<<"' y='"<<y<<"' width='"<<width<<"' height='"<<height<<"' stroke='"<<stroke<<"' fill='"<<fill<<"' fill-opacity='"<<fill\_opacity<<"' />";
96. }
97. void svg\_text(double left, double baseline, string text)
98. {
99. cout << "<text x='" << left << "' y='"<<baseline<<"'>"<<text<<"</text>";
100. }
101. void show\_histogram\_svg(const vector<size\_t>& bins)
102. {
103. const size\_t screen\_width=80;
104. const size\_t width=screen\_width-4;
105. const auto IMAGE\_WIDTH = 400;
106. const auto IMAGE\_HEIGHT = 300;
107. const auto TEXT\_LEFT = 20;
108. const auto TEXT\_BASELINE = 20;
109. const auto TEXT\_WIDTH = 50;
110. const auto BIN\_HEIGHT = 30;
111. const auto BLOCK\_WIDTH = 10;
112. const auto GRAPH\_WIDTH = IMAGE\_WIDTH-TEXT\_WIDTH;
113. svg\_begin(IMAGE\_WIDTH, IMAGE\_HEIGHT);
114. double top = 0;
115. string stroke="black";
116. string fill="red";
117. size\_t max\_count = 0;
118. for (size\_t count : bins)
119. {
120. if (count > max\_count)
121. {
122. max\_count = count;
123. }
124. }
125. const bool scaling\_needed = max\_count > width;
126. if (scaling\_needed)
127. {
128. const double scaling\_multiplier = (double)width / max\_count;
129. if(max\_count\*BLOCK\_WIDTH/GRAPH\_WIDTH>1)
130. {
131. for (size\_t bin : bins)
132. {
133. const double new\_scaling\_multiplier=(double)GRAPH\_WIDTH/(max\_count\*BLOCK\_WIDTH\*scaling\_multiplier);
134. const size\_t new\_bin\_width = (size\_t)BLOCK\_WIDTH\*bin\*scaling\_multiplier\*new\_scaling\_multiplier;
135. svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));
136. svg\_rect(TEXT\_WIDTH, top, new\_bin\_width, BIN\_HEIGHT,stroke,fill,bin\_opacity17(bin,max\_count));
137. top += BIN\_HEIGHT;
138. }
139. }
140. else
141. {
142. for (size\_t bin : bins)
143. {
144. const double bin\_width = BLOCK\_WIDTH\*bin\*scaling\_multiplier;
145. svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));
146. svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,stroke,fill,bin\_opacity17(bin,max\_count));
147. top += BIN\_HEIGHT;
148. }
149. }
150. }
151. else
152. {
153. if(max\_count\*BLOCK\_WIDTH/GRAPH\_WIDTH>1)
154. {
155. for (size\_t bin : bins)
156. {
157. const double new\_scaling\_multiplier=(double)GRAPH\_WIDTH/(max\_count\*BLOCK\_WIDTH);
158. const size\_t new\_bin\_width = (size\_t)BLOCK\_WIDTH\*bin\*new\_scaling\_multiplier;
159. svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));
160. svg\_rect(TEXT\_WIDTH, top, new\_bin\_width, BIN\_HEIGHT,stroke,fill,bin\_opacity17(bin,max\_count));
161. top += BIN\_HEIGHT;
162. }
163. }
164. else
165. {
166. for (size\_t bin : bins)
167. {
168. const double bin\_width = BLOCK\_WIDTH \* bin;
169. svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));
170. svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT,stroke,fill,bin\_opacity17(bin,max\_count));
171. top += BIN\_HEIGHT;
172. }
173. }
174. }
175. cout << "<text x='" << TEXT\_LEFT << "' y='"<<top+BIN\_HEIGHT<<"'>"<<make\_info\_text()<<"</text>";
176. svg\_end();
177. }
178. Histogram.cpp
179. #include "histogram.h"
180. #include <iostream>
181. vector<double> input\_numbers(istream& in, size\_t count) {
182. vector<double> result(count);
183. for (size\_t i = 0; i < count; i++) {
184. in >> result[i];
185. }
186. return result;
187. }
188. void find\_minmax(const vector<double>& numbers, double& min, double& max)
189. {
190. if (numbers.size()==0) return;
191. min = numbers[0];
192. max = numbers[0];
193. for (double number : numbers)
194. {
195. if (number < min)
196. {
197. min = number;
198. }
199. if (number > max)
200. {
201. max = number;
202. }
203. }
204. }
205. vector <size\_t> make\_histogram(Input data)
206. {
207. double min;
208. double max;
209. find\_minmax(data.numbers,min,max);
210. vector<size\_t> bins(data.bin\_count);
211. for (double number : data.numbers)
212. {
213. size\_t bin = (size\_t)((number - min) / (max - min) \* data.bin\_count);
214. if (bin == data.bin\_count)
215. {
216. bin--;
217. }
218. bins[bin]++;
219. }
220. return(bins);
221. }
222. void show\_histogram\_text(const vector<size\_t>& bins)
223. {
224. const size\_t screen\_width=80;
225. const size\_t width=screen\_width-4;
226. size\_t max\_count = 0;
227. for (size\_t count : bins)
228. {
229. if (count > max\_count)
230. {
231. max\_count = count;
232. }
233. }
234. const bool scaling\_needed = max\_count > width;
235. for (size\_t bin : bins)
236. {
237. if (bin < 100)
238. {
239. cout << ' ';
240. }
241. if (bin < 10)
242. {
243. cout << ' ';
244. }
245. cout << bin << "|";
246. size\_t height = bin;
247. if (scaling\_needed)
248. {
249. const double scaling\_multiplier = (double)width / max\_count;
250. height = (size\_t)(bin \* scaling\_multiplier);
251. }
252. for (size\_t i = 0; i < height; i++)
253. {
254. cout << '\*';
255. }
256. cout<<' ';
257. cout << endl;
258. }
259. }
260. WinVersion.cpp
261. #include <windows.h>
262. #include <iostream>
263. #include <string>
264. #include <sstream>
265. using namespace std;
266. string make\_info\_text() {
267. stringstream buffer;
268. DWORD mask = 0x0000ffff;
269. DWORD mask\_major = 0b00000000'00000000'00000000'11111111;
270. DWORD info = GetVersion();
271. DWORD platform = info >> 16;
272. DWORD version = info & mask;
273. DWORD version\_major = version & mask\_major;
274. DWORD version\_minor = version >> 8;
275. if ((info & 0x40000000) == 0) {
276. DWORD build = platform;
277. buffer << "Windows v" << version\_major << "." << version\_minor << "(build " << build << ")\n";
278. }
279. char computer\_name[MAX\_COMPUTERNAME\_LENGTH + 1];
280. DWORD size = sizeof(computer\_name);
281. GetComputerNameA(computer\_name, &size);
282. buffer << "Computer name: " << computer\_name << "\n";
283. return buffer.str();
284. }