Федеральное государственное бюджетное образовательное учреждение высшего образования «Национальный исследовательский университет «МЭИ»

Институт информационных и вычислительных технологий

Кафедра Управления и интеллектуальных технологий

**Отчёт по лабораторной работе № 4**

# По курсу «Разработка ПО систем управления»

# «Библиотеки. Низкоуровневые операции»

# Выполнил студент группы А-02-20

# Гребенева Ю.И.

# Проверили

# Мохов А. С

# Василькова П.Д.

Москва 2021

**Задание.**

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.

Код, который будет написан в этой части работы, должен в итоге формировать строку, которую нужно отобразить снизу итогового SVG в формате:

Windows v5.1 (build 1234)Computer name: My-Comp

#### Вариант 18

С помощью функции curl\_easy\_getinfo() печатайте на стандартный вывод ошибок время, затраченное на получение IP-адреса сервера по его имени (name lookup).

КОД ПРОГРАММЫ

***main.cpp***

|  |
| --- |
|  |
|  | using namespace std;    int  main(int argc, char\* argv[])  {    Input input;      if (argc > 1)  {    input = download(argv[1]);    }    else  {  input = read\_input(cin,true);  }      const auto bins = make\_histogram(input);    show\_histogram\_svg(bins);    return 1;  } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

***histogram.h***

|  |
| --- |
|  |
|  | #pragma once  #include <curl/curl.h>  #include <vector>  #include <iostream>  #include <math.h>  #include <string>  #include <sstream>  #include<iomanip>  #include <fstream>      using namespace std;    struct Input  {  vector<double> numbers;  size\_t bin\_count;  };  Input read\_input(istream& in,bool prompt);  Input download(const string& address);  size\_t write\_data(void\* items, size\_t item\_size, size\_t item\_count, void\* ctx);  void find\_minmax(const vector<double>& numbers, double& min, double& max);  vector<double> input\_numbers(istream& in, size\_t count);  vector<double> make\_histogram(Input data);  string make\_info\_text();  void show\_histogram\_text(vector<double>bins);  void svg\_begin(double width, double height);  void svg\_end();  void svg\_text(double left, double baseline, string text);  void svg\_rect(double x, double y, double width, double height, string stroke = "black", string fill = "blue");  void show\_histogram\_svg(const vector<double>& bins); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

***histogram.cpp***

|  |
| --- |
|  |
|  | #include "histogram.h"        void find\_minmax(const vector<double>& numbers, double& min, double& max)  {  min = numbers[0];  max = numbers[0];    for (size\_t i = 0; i < numbers.size(); i++)  {  if (numbers[i] < min)  {  min = numbers[i];  }  if (numbers[i] > max)  {  max = numbers[i];  }  }    return;  }  vector<double> input\_numbers(istream& in, size\_t count)  {  vector<double> result(count);  for (size\_t i = 0; i < count; i++)  {  in >> result[i];  }  return result;  }    Input read\_input(istream& in, bool prompt)  {    Input data;  if (prompt)  {  cerr << "Enter number count: ";  }  size\_t number\_count;  in >> number\_count;    if(prompt)  {  cerr << "Enter numbers: ";  }  data.numbers = input\_numbers(in, number\_count);    if(prompt)  {  cerr <<"Enter bin count:";  }  in>>data.bin\_count;    return data;    }  Input download(const string& address)  {  stringstream buffer;  curl\_global\_init(CURL\_GLOBAL\_ALL);  CURL \*curl = curl\_easy\_init();    if(curl)  {  CURLcode res;  curl\_easy\_setopt(curl, CURLOPT\_URL, address.c\_str());  curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, write\_data);  curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &buffer);  string namelookup;  double t=curl\_easy\_getinfo(curl,CURLINFO\_NAMELOOKUP\_TIME\_T,&namelookup);    res = curl\_easy\_perform(curl);  if(res)  {  cerr<< curl\_easy\_strerror(res)<<"("<<res<<")"<<" NAMELOOKUP:"<<t<<endl;  exit(1);  }  else  {  cerr<<" NAMELOOKUP:"<<t<<endl;  }  curl\_easy\_cleanup(curl);  }  return read\_input(buffer,false);  }  size\_t write\_data(void\* items, size\_t item\_size, size\_t item\_count, void\* ctx)  {  size\_t data\_size = item\_size \* item\_count;  stringstream\* buffer = reinterpret\_cast<stringstream\*>(ctx);  buffer->write(reinterpret\_cast<const char\*>(items), data\_size);  return data\_size;  }  string make\_info\_text()  {  stringstream buffer;  DWORD info = GetVersion();  DWORD mask = 0x0000ffff;  DWORD version = info & mask;  DWORD platform = info >> 16;  DWORD mask\_2 = 0x0000ff;  if ((info & 0x80000000) == 0)  {  DWORD version\_major = version & mask\_2;  DWORD version\_minor = version >> 8;  DWORD build = platform;  buffer << "Windows v"<<version\_major<<"."<<version\_minor<<"(build "<<build<<")"<<'\n';  //  }  char comp\_name[MAX\_COMPUTERNAME\_LENGTH + 1];  GetSystemDirectory(comp\_name, MAX\_PATH);  printf("System directory: %s", comp\_name);  DWORD size = MAX\_COMPUTERNAME\_LENGTH+1;  GetComputerNameA(comp\_name, &size);  buffer<<"Computer name:" <<comp\_name;  return buffer.str();  }  vector<double> make\_histogram(Input data)  {    double min, max;  vector<double> bins(data.bin\_count);  find\_minmax(data.numbers, min, max);  size\_t number\_count = data.numbers.size();  double bin\_size = (max - min) / data.bin\_count;  for (size\_t i = 0; i < number\_count; i++)  {  bool flag = false;  for (size\_t j = 0; (j < data.bin\_count - 1) && !flag; j++)  {  auto lo = min + j \* bin\_size;  auto hi = min + (j + 1) \* bin\_size;  if ((lo <= data.numbers[i]) && (hi > data.numbers[i]))  {  bins[j]++;  flag = true;  }    }  if (!flag)  {  bins[data.bin\_count - 1]++;  }  }    return bins;  }    void show\_histogram\_text(vector<size\_t>bins)  {  size\_t bin\_count = bins.size();  size\_t max\_count = 0;  for (size\_t i = 0; i < bin\_count; i++)  {  size\_t count = bins[i];    if (count > max\_count)  {  max\_count = count;  }  }  double height;  const size\_t SCREEN\_WIDTH = 80;  const size\_t MAX\_ASTERISK = SCREEN\_WIDTH - 3 - 1;  if (max\_count > MAX\_ASTERISK)  {  height = (static\_cast<double>(MAX\_ASTERISK) / max\_count);;  }  else  {  height = 1;  }  for (size\_t i = 0; i < bin\_count; i++)  {  size\_t k = bins[i];  cout << " ";  if (k < 100)  {  cout << " ";  }  if (k < 10)  {  cout << " ";  }  cout << k << "|";  for (size\_t j = 0; j < k \* height; j++)  {  cout << "\*";  }  cout << "\n";    }  return;  }  void  svg\_begin(double width, double height)  {  cout << "<?xml version='1.0' encoding='UTF-8'?>\n";  cout << "<svg ";  cout << "width='" << width << "' ";  cout << "height='" << height << "' ";  cout << "viewBox='0 0 " << width << " " << height << "' ";  cout << "xmlns='http://www.w3.org/2000/svg'>\n";  }    void  svg\_end()  {  cout << "</svg>\n";  }  void  svg\_text(double left, double baseline, string text)  {    cout << "<text x='" << left << "' y='" << baseline << "'>" << text << "</text>";  }  void svg\_rect(double x, double y, double width, double height, string stroke, string fill )  {  cout << "<rect x='" << x << "' y='" << y << "' width='" << width << "' height='" << height  << "' stroke='" << stroke << "' fill='" << fill << "'/>";  }    void show\_histogram\_svg( const vector<double>& bins)  {      const auto IMAGE\_WIDTH = 400;  const auto IMAGE\_HEIGHT = 300;  const auto TEXT\_LEFT = 20;  const auto TEXT\_BASELINE = 20;  const auto TEXT\_WIDTH = 50;  const auto BIN\_HEIGHT = 30;  const auto BLOCK\_WIDTH = 10;  const auto MAX\_COUNT = (IMAGE\_WIDTH-TEXT\_WIDTH)/ BLOCK\_WIDTH;    double top = 0;  svg\_begin(IMAGE\_WIDTH, IMAGE\_HEIGHT);  size\_t count;  size\_t max\_count = bins[0];  for (size\_t bin : bins)  {  if (bin > max\_count)  {  max\_count = bin;  }  }    if (max\_count > MAX\_COUNT)  {    const double h = (double)MAX\_COUNT / max\_count;    for (size\_t bin : bins)  {  auto height = (size\_t)(bin \* h);  const double bin\_width = BLOCK\_WIDTH \* height;  svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));  svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "black", "red");  top += BIN\_HEIGHT;    }  svg\_text(0,top + TEXT\_BASELINE, make\_info\_text());  svg\_end();  }  else  {  for (size\_t bin : bins)  {    const double bin\_width = BLOCK\_WIDTH \* bin;  svg\_text(TEXT\_LEFT, top + TEXT\_BASELINE, to\_string(bin));  svg\_rect(TEXT\_WIDTH, top, bin\_width, BIN\_HEIGHT, "black", "red");  top += BIN\_HEIGHT;    }  svg\_text(0,top + TEXT\_BASELINE, make\_info\_text());  svg\_end();  }      } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |