

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №3**

з дисципліни “Основи програмування”

тема “**Бінарне дерево пошуку**”

|  |  |  |
| --- | --- | --- |
| Виконав  студент I курсу  групи КП-92  Мовчан Максим Олександрович  (*прізвище, ім’я, по батькові*)  варіант № 9 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  викладач  Гадиняк Руслан Анатолійович  (*прізвище, ім’я, по батькові*) |

Київ 2020

**Мета роботи**

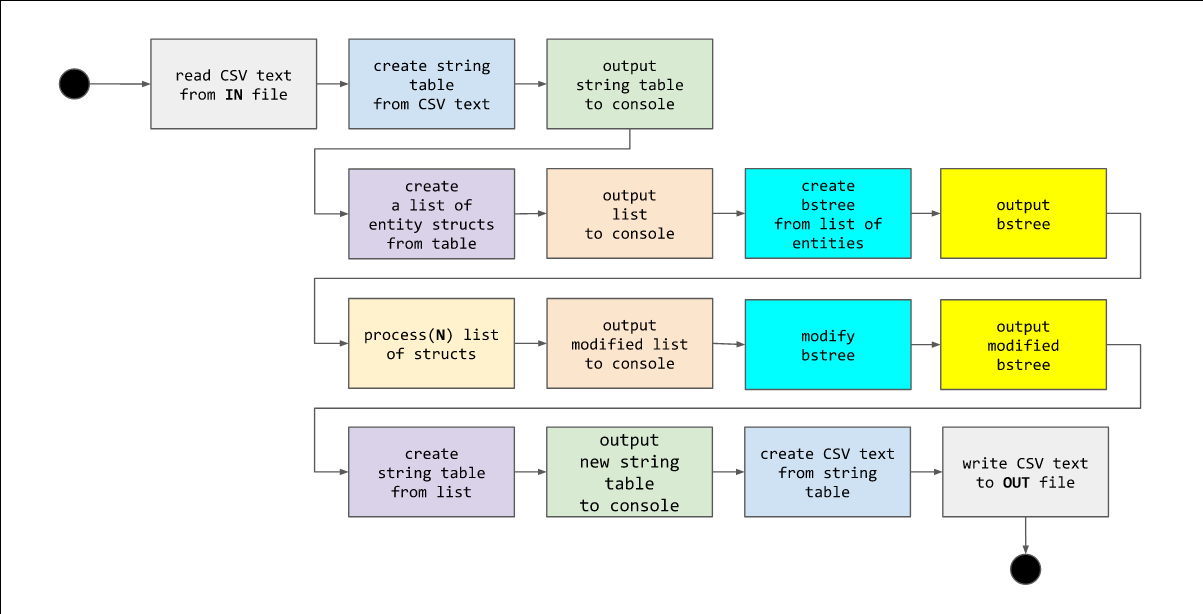
Реалізувати та використати АТД "Бінарне дерево пошуку".  
Навчитись створювати та обходити нелінійні структури даних.  
Використати шаблонні класи для реалізації колекцій.

**Постановка завдання**

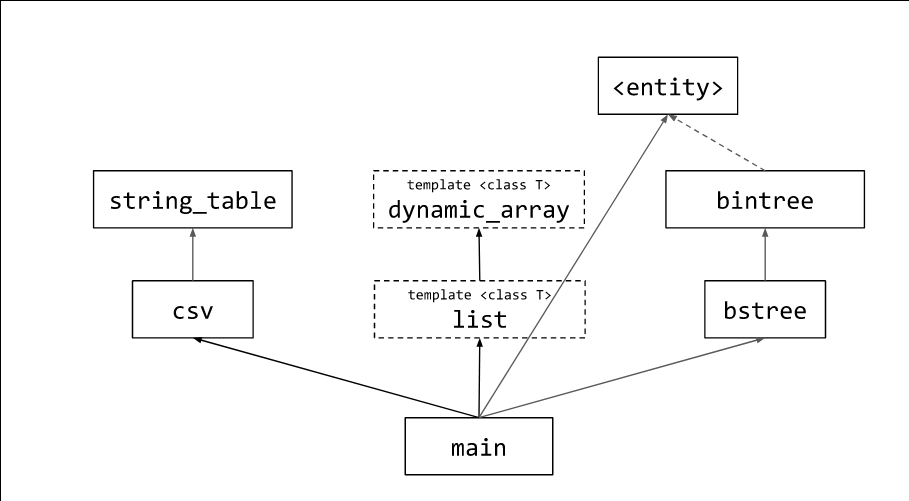
Створити консольну програму, що дозволяє працювати з бінарним деревом пошуку.

1. Зчитати CSV дані з текстового файлу і сформувати список сутностей, кожна з яких має унікальний цілочисельний ідентифікатор.
2. Побудувати на основі списку бінарне дерево пошуку (БДП). У якості ключів нод БДП використати ідентифікатори сутностей.
3. Ввести N та модифікувати БДП відповідно до умови з попередньої лабораторної роботи.
4. Вивести модифіковане БДП у консоль.
5. Записати модифікований список сутностей у текстовий файл в форматі CSV.

**Блок-схема алгоритму**



**Діаграма залежностей**

****

**Інтерфейси модулів**

|  |
| --- |
| **Dynamic\_array.h** |
| template<typename T> class DynArray {  T\* items\_;  size\_t capacity\_;  public:  explicit DynArray(size\_t cap): capacity\_ {cap} { items\_ = new T[cap]; }  ~DynArray() { delete[] items\_; }   size\_t size() { return capacity\_; }  T& operator[](size\_t i) { return items\_[i]; }   void resize(size\_t cap) {  T\* newArr = new T[cap];  for (size\_t i = 0; i < fmin(capacity\_, cap); i++)  newArr[i] = items\_[i];  delete[] items\_;  items\_ = newArr;  capacity\_ = cap;  } }; |
| **csv.h** |
| #include "string\_table.h" using namespace std; // parse csv, init and fill string table StringTable Csv\_parse(string& csvStr);  // create string and fill it with csv text string Csv\_toString(StringTable& table); |
| **BinTree.h** |
| #include "Organisation.h" struct BinTree {  Organisation value; // set in ctor  BinTree\* left;  BinTree\* right;   explicit BinTree(Organisation v): left(nullptr), right(nullptr) { value = v; } }; |

|  |
| --- |
| **List.h** |
| #include "dynamic\_array.h" template<typename T> class List {  DynArray<T> \_array = new DynArray<T>[size\_];  int size\_;  public:  List(): \_array {16}, size\_ {0} {}  ~List() = default;   int size() { return size\_; }   T& operator[](size\_t i) { return \_array[i]; }  void insert(size\_t index, T value) {  \_array.resize(\_array.size() + 1);  for (size\_t i = 0; size\_ - i > index; i++)  \_array[size\_ - i] = \_array[size\_ - i - 1];  \_array[index] = value;  size\_++;  }  void remove\_at(size\_t index) {  for (size\_t i = index; i < size\_ - 1; i++) {  \_array[i] = \_array[i + 1];  }  size\_--;  }  void push\_back(T value) {  \_array[size\_] = value;  size\_++;  if (size\_ == \_array.size()) \_array.resize(\_array.size() + 1);  }  void remove(T value) {  for (size\_t i = 0; i < size\_; i++) {  if (\_array[i] == value) { remove\_at(i); }  }  }  int index\_of(T value) {  for (size\_t i = 0; i < size\_; i++) {  if (\_array[i] == value) return i;  }  return -1;  }  bool contains(T value) {  for (size\_t i = 0; i < size\_; i++) {  if (\_array[i] == value) return true;  }  return false;  }  bool empty() { return size\_ == 0; }  void clear() { size\_ = 0; } }; bool operator==(const Organisation& o1, const Organisation& o2) {  return o1.country == o2.country && o1.id == o2.id && o1.label == o2.label &&  o1.founded\_date == o2.founded\_date && o1.founders == o2.founders &&  o1.type == o2.type; } |
| **BsTree.h** |
| #include "BinTree.h”  using std::endl, std::cout; class BSTree {  BinTree\* root\_ = nullptr;  size\_t size\_ = 0;  public:  [[nodiscard]] size\_t size() const;  bool lookup(int key);  Organisation search(int key);  Organisation remove(int key);  Organisation remove(int key, BinTree\* node, BinTree\* parent);  void clear();  void insert(Organisation val);  void print();  void print(BinTree\* root, const string& prefix); };  int getKey(Organisation value); |
| **Organisations.h** |
| struct Organisation {  int id;  string country;  string label;  string type;  int founded\_date;  string founders; }; |
| **string\_table.h** |
| class StringTable {   string\* cells\_ = nullptr;  size\_t rows\_ = 0;  size\_t cols\_ = 0;  public:  StringTable(size\_t rows, size\_t cols);  ~StringTable();   size\_t size\_rows() const;  size\_t size\_columns() const;   string& at(size\_t rowIndex, size\_t colIndex);   void print(); }; |

**Вхідні/вихідні дані**

|  |
| --- |
| **data.csv** |
| id,country,label,type,founded,founder(s) 50,US,"Google inc.",Public Company,1998,"Larry Page, Sergey Brin" 30,UK,"EPAM Systems",Public Company,1993,"Arkadiy Dobkin, Leo Lozner" 23,US,"Microsoft Corporation",Public Company,1975,"Bill Gates, Paul Allen" 91,UK,"Apple Inc.",Public Company,1976,"Steve Jobs, Steve Wozniak, Ronald Wayne" 69,UG,"FlexAir",Volunteer Organisation,1488,"Ricardo Milos" |
| **Аргументи командного рядка** |
| -o ../CSV/out.csv -n US -b ../CSV/data.csv |
| **Термінал** |
| [id][country][label][type][founded][founder(s)]  [50][US][Google inc.][Public Company][1998][Larry Page, Sergey Brin]  [30][UK][EPAM Systems][Public Company][1993][Arkadiy Dobkin, Leo Lozner]  [23][US][Microsoft Corporation][Public Company][1975][Bill Gates, Paul Allen]  [91][UK][Apple Inc.][Public Company][1976][Steve Jobs, Steve Wozniak, Ronald Wayne]  [69][UG][FlexAir][Volunteer Organisation][1488][Ricardo Milos]  id Country Label Type Founded Founders  50 US "Google inc." Public Company 1998 "Larry Page, Sergey Brin"  30 UK "EPAM Systems" Public Company 1993 "Arkadiy Dobkin, Leo Lozner"  23 US "Microsoft Corporation"Public Company 1975 "Bill Gates, Paul Allen"  91 UK "Apple Inc." Public Company 1976 "Steve Jobs, Steve Wozniak, Ronald Wayne"  69 UG "FlexAir" Volunteer Organisation 1488 "Ricardo Milos"  ------  SIZE: 5  50 "Google inc."  ├── 91 "Apple Inc."  │ └── 69 "FlexAir"  └── 30 "EPAM Systems"  └── 23 "Microsoft Corporation"  ------  id Country Label Type Founded Founders  50 US "Google inc." Public Company 1998 "Larry Page, Sergey Brin"  23 US "Microsoft Corporation"Public Company 1975 "Bill Gates, Paul Allen"  ------  SIZE: 2  50 "Google inc."  └── 23 "Microsoft Corporation"  ------  [id][country][label][type][founded\_date][founders]  [50][US][Google inc.][Public Company][1998][Larry Page, Sergey Brin]  [23][US][Microsoft Corporation][Public Company][1975][Bill Gates, Paul Allen] |
| **out.csv** |
| id,country,label,type,founded\_date,founders50,US,"Google inc.",Public Company,1998,"Larry Page, Sergey Brin"23,US,"Microsoft Corporation",Public Company,1975,"Bill Gates, Paul Allen" |

**Тексти коду програм**

|  |
| --- |
| **main.cpp** |
| // // Created by xon on 3/24/20. //  #include "BsTree.h" #include "Organisation.h" #include "csv.h" #include "list.h"  #include <cstdlib> #include <fstream> #include <iomanip> #include <iostream> #include <unistd.h> using namespace std;  StringTable processEntities(List<Organisation>& list, Organisation n); void PrintList(List<Organisation>& list); StringTable Table\_fromFile(char\* file\_name); void File\_fromTable(StringTable& table, char\* file\_name); List<Organisation> createEntityListFromTable(StringTable& csvTable);  int main(int argc, char\* argv[]) {  char\* in = nullptr;  char\* out = nullptr;  char\* n = nullptr;  size\_t b = 0, c;  bool isN = 0;   // reading operands  while ((c = getopt(argc, argv, "i:o:n:b")) != -1) {  switch (c) {  case 'i': in = optarg; break;  case 'o': out = optarg; break;  case 'n':  n = optarg;  isN = 1;  break;  case 'b': b = 1; break;  }  }  // reading operands, that not assigned with '-' key  for (size\_t index = optind; index < argc; index++) {  if (!in) {  in = argv[index];  } else if (!out) {  out = argv[index];  } else if (!n) {  n = argv[index];  isN = 1;  }  }  // Check required files  if (!in) {  fprintf(**stderr**, " missing input file\n");  exit(2);  } else if (!out) {  fprintf(**stderr**, " missing output file\n");  exit(2);  } else if (!n) {  n = "";  }  // TABLE > FILE  StringTable table = Table\_fromFile(in);  // COUT < TABLE  table.print();  // TABLE > LISTS  List list = createEntityListFromTable(table);  List list2 = createEntityListFromTable(table);  // COUT < LIST  PrintList(list);  BSTree myTree {};  if (b) {  for (int i = 0; i < list.size(); i++) {  // TREE < LIST  myTree.insert(list[i]);  }  // COUT < TREE  myTree.print();  }  Organisation org;  org.country = n;  // LIST = N || TABLE < LIST  StringTable table2 = processEntities(list, org);  if (isN) {  // COUT < LIST  PrintList(list);  if (b) {  for (int i = 0; i < list2.size(); i++) {  // TREE = N  if (!list.contains(list2[i])) { myTree.remove(list2[i].id); }  }  // COUT < TREE  myTree.print();  }  // COUT < TABLE  table2.print();  }  // CLEAR TREE  myTree.clear();  // OUT <<<<<<<  if (!isN) {  File\_fromTable(table, out);  return 0;  }  File\_fromTable(table2, out); }  StringTable Table\_fromFile(char\* file\_name) {  ifstream file(file\_name);  if (!file.is\_open()) {  perror("\033[;;41mCAN'T OPEN FILE\033[0m\n");  exit(**EXIT\_FAILURE**);  }  string from\_file, temp;  while (getline(file, temp)) {  from\_file += temp + "\n";  }  file.close();  if (file.is\_open()) {  perror("\033[;;41mCAN'T CLOSE FILE\033[0m\n");  exit(**EXIT\_FAILURE**);  }  return Csv\_parse(from\_file); } void File\_fromTable(StringTable& table, char\* file\_name) {  ofstream file(file\_name);  if (!file.is\_open()) {  perror("\033[;;41mCAN'T OPEN FILE\033[0m\n");  exit(**EXIT\_FAILURE**);  }  string out = Csv\_toString(table);  file << out << endl;  file.close();  if (file.is\_open()) {  perror("\033[;;41mCAN'T CLOSE FILE\033[0m\n");  exit(**EXIT\_FAILURE**);  } } int checknum(string str) {  if (isdigit(str[0]))  return stoi(str);  else  cout << "\nDate is not num\n";  exit(1); }  List<Organisation> createEntityListFromTable(StringTable& table) {  List<Organisation> list;  Organisation org;  for (size\_t i = 1; i < table.size\_rows(); i++) {  int col = -1;  org.id = checknum(table.at(i, ++col));  org.country = table.at(i, ++col);  org.label = table.at(i, ++col);  org.type = table.at(i, ++col);  org.founded\_date = checknum(table.at(i, ++col));  org.founders = table.at(i, ++col);  list.push\_back(org);  }  return list; } void PrintList(List<Organisation>& list) {  cout << left << setw(4) << "id";  cout << left << setw(9) << "Country";  cout << left << setw(23) << "Label";  cout << left << setw(23) << "Type";  cout << left << setw(9) << "Founded";  cout << left << setw(23) << "Founders";  cout << endl;  for (int i = 0; i < list.size(); i++) {  cout << left << setw(4) << list[i].id;  cout << left << setw(9) << list[i].country;  cout << left << setw(23) << list[i].label;  cout << left << setw(25) << list[i].type;  cout << left << setw(9) << list[i].founded\_date;  cout << left << setw(44) << list[i].founders;  cout << endl;  } } StringTable processEntities(List<Organisation>& list, Organisation n) {  for (int i = list.size() - 1; i >= 0; i--)  if (list[i].country != n.country) list.remove\_at(i);  StringTable table(list.size() + 1, 6);  for (int i = 0; i < list.size(); i++) {  int col = -1;  table.at(i + 1, ++col) = to\_string(list[i].id);  table.at(i + 1, ++col) = list[i].country;  table.at(i + 1, ++col) = list[i].label;  table.at(i + 1, ++col) = list[i].type;  table.at(i + 1, ++col) = to\_string(list[i].founded\_date);  table.at(i + 1, ++col) = list[i].founders;  }  int col = -1;  table.at(0, ++col) = "id";  table.at(0, ++col) = "country";  table.at(0, ++col) = "label";  table.at(0, ++col) = "type";  table.at(0, ++col) = "founded\_date";  table.at(0, ++col) = "founders";  return table; } |
| **Csv.cpp** |
| #include "csv.h" StringTable Csv\_parse(string& csvStr) {  const char\* begin = csvStr.c\_str();  bool quotes = 0;  size\_t max\_cols = 0, rows = 0, cols = 0;  for (const char\* p = begin; \*p; p++) {  if (\*p == '"')  quotes = !quotes;  else if (\*p == ',' and quotes == 0) {  cols++;  begin = p + 1;  } else if (\*p == '\n') {  if (cols > max\_cols) max\_cols = cols;  cols = 0;  rows++;  begin = p + 1;  }  }  max\_cols++;  StringTable table {rows, max\_cols};  begin = csvStr.c\_str();  quotes = 0, cols = 0, rows = 0;  for (const char\* p = begin; \*p; p++) {  if (\*p == '"')  quotes = !quotes;  else if (\*p == ',' and quotes == 0) {  table.at(rows, cols) = string(begin, p - begin);  cols++;  begin = p + 1;  } else if (\*p == '\n') {  table.at(rows, cols) = string(begin, p - begin);  for (cols++; cols < max\_cols; cols++) {  table.at(rows, cols) = "";  }  cols = 0;  rows++;  begin = p + 1;  }  }  return table; } string Csv\_toString(StringTable& table) {  string tmp;  for (size\_t i = 0; i < table.size\_rows(); i++) {  for (size\_t j = 0; j < table.size\_columns(); j++) {  tmp += table.at(i, j) + ",";  }  tmp[tmp.length() - 1] = '\n';  }  return tmp; } |
| **BsTree.cpp** |
| // // Created by manjaro on 3/28/20. // #include "BsTree.h"  size\_t BSTree::size() const { return size\_; }  int getKey(Organisation value) { return value.id; }  void BSTree::insert(Organisation val) {  if (root\_ == nullptr) {  root\_ = new BinTree(val);  } else {  BinTree\* tmp = root\_;  while (true) {  if (val.id == tmp->value.id) {  cout << "Warning: Value already exists, so nothing will be done."  << endl;  return;  }  if (val.id < tmp->value.id) {  if (tmp->left == nullptr) {  tmp->left = new BinTree(val);  break;  } else {  tmp = tmp->left;  }  } else {  if (tmp->right == nullptr) {  tmp->right = new BinTree(val);  break;  } else {  tmp = tmp->right;  }  }  }  }  size\_++; } void BSTree::print() {  if (root\_ == nullptr) {  cout << "Empty" << endl;  return;  }   cout << "------" << endl << "SIZE: " << size() << endl;  cout << root\_->value.id << " " << root\_->value.label << endl;  print(root\_, "");  cout << "------" << endl; } void BSTree::print(BinTree\* root, const string& prefix) {  if (root == nullptr) { return; }   bool buf\_left = (root->left != nullptr);  bool buf\_right = (root->right != nullptr);   if (!buf\_left && !buf\_right) { return; }   cout << prefix;  cout << ((buf\_left && buf\_right) ? "├── " : "");  cout << ((!buf\_left && buf\_right) ? "└── " : "");   if (buf\_right) {  bool printStrand =  (buf\_left && buf\_right &&  (root->right->right != nullptr || root->right->left != nullptr));  string newPrefix = prefix + (printStrand ? "│ " : " ");  cout << root->right->value.id << " " << root->right->value.label << endl;  print(root->right, newPrefix);  }   if (buf\_left) {  cout << (buf\_right ? prefix : "") << "└── " << root->left->value.id  << " "  ""  << root->left->value.label << endl;  print(root->left, prefix + " ");  } }  bool BSTree::lookup(int key) {  BinTree\* tmp = root\_;  while (true) {  if (tmp == nullptr) {  return false;  } else if (key == tmp->value.id) {  return true;  } else if (key < tmp->value.id) {  tmp = tmp->left;  } else {  tmp = tmp->right;  }  } }  void BSTree::clear() {  if (root\_ == nullptr) return;  std::queue<BinTree\*> queue;  queue.push(root\_);  while (!queue.empty()) {  BinTree\* tmp = queue.front();  queue.pop();  if (tmp->left != nullptr) { queue.push(tmp->left); }  if (tmp->right != nullptr) { queue.push(tmp->right); }  delete tmp;  }  size\_ = 0; }  Organisation BSTree::remove(int key, BinTree\* node, BinTree\* parent) {  if (node->value.id == key) {  int tkey = node->value.id;  BinTree\* root\_right = node->right;  BinTree\* root\_left = node->left;  if (parent == nullptr) {  if (root\_right == nullptr) {  root\_ = root\_left;  } else {  BinTree\* right\_left = root\_right->left;  if (right\_left == nullptr) {  root\_ = root\_right;  root\_right->left = root\_left;  } else {  BinTree\* right\_pointer = root\_right;  while (right\_left->left != nullptr) {  right\_pointer = right\_left;  right\_left = right\_left->left;  }  right\_pointer->left = right\_left->right;  root\_ = right\_left;  right\_left->right = root\_right;  while (right\_left->left != nullptr) {  right\_left = right\_left->left;  }  right\_left->left = root\_left;  }  }  } else {  if (root\_right == nullptr) {  if (parent->left != nullptr) {  if (parent->left->value.id == key) {  parent->left = root\_left;  } else {  parent->right = root\_left;  }  } else {  parent->right = root\_left;  }  } else {  BinTree\* right\_left = root\_right->left;  if (right\_left == nullptr) {  if (parent->left == nullptr) {  if (parent->left->value.id == key) {  parent->left = root\_right;  } else {  parent->right = root\_right;  }  } else {  parent->right = root\_right;  root\_right->left = root\_left;  }  } else {  BinTree\* right = root\_right;  while (right\_left->left != nullptr) {  right = right\_left;  right\_left = right\_left->left;  }  right->left = right\_left->right;  if (parent->left != nullptr) {  if (parent->left->value.id == key) {  parent->left = right\_left;  } else {  parent->right = right\_left;  }  } else {  parent->right = right\_left;  }  right\_left->right = root\_right;  right\_left->left = root\_left;  }  }  }  Organisation ret = node->value;  delete node;  size\_--;  return ret;  }  if (node->value.id < key) {  return remove(key, node->right, node);  } else {  return remove(key, node->left, node);  } } Organisation BSTree::remove(int key) {  if (!lookup(key)) {  perror("Key not exist");  exit(1);  } else {  return remove(key, root\_, nullptr);  } } Organisation BSTree::search(int key) {  BinTree\* tmp = root\_;  while (true) {  if (tmp == nullptr) {  perror("Not found");  exit(1);  } else if (key == tmp->value.id) {  return tmp->value;  } else if (key < tmp->value.id) {  tmp = tmp->left;  } else {  tmp = tmp->right;  }  } } |
| **string\_table.cpp** |
| #include "../HEADERS/string\_table.h" StringTable::StringTable(size\_t rows, size\_t cols) {  cells\_ = new string[rows \* cols];  rows\_ = rows;  cols\_ = cols; } StringTable::~StringTable() { delete[] cells\_; } size\_t StringTable::size\_rows() const { return rows\_; } size\_t StringTable::size\_columns() const { return cols\_; } std::string& StringTable::at(size\_t rowIndex, size\_t colIndex) {  return cells\_[cols\_ \* rowIndex + colIndex]; } void StringTable::print() {  for (size\_t i = 0; i < size\_rows(); i++) {  for (size\_t j = 0; j < size\_columns(); j++) {  string to\_filter = at(i, j);  for (size\_t k = 0; k < to\_filter.length(); k++) {  if (to\_filter[k] == '"') { to\_filter.erase(k, 1); }  }  cout << '[' << to\_filter << ']';  usleep(10000);  }  cout << endl;  } } |

**Висновки**

У лабораторній роботі були використані структури даних на основі динамічного масиву. Виділення пам’яті було виконано за допомогою оператора new та звільнена за допомогою оператор **delete[].**

Для виводу у консоль був використаний заголовочний файл **<iostream>,** об’єкти **cout**, **endl** з простору імен std та оператор вставки у потік **<<**.. Для зчитування аргументів командного рядка був використаний **getOpt**

Для зчитування файлу був використаний getline. Компіляція коду відбувалася за допомогою **CMake**.