Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет

ЗВІТ
ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 11
з навчальної дисципліни
“Базові методології та технології програмування”

Реалізація програмних засобів оброблення динамічних структур даних та бінарних файлів

ЗАВДАННЯ ВИДАВ
доцент кафедри кібербезпеки
та програмного забезпечення
Доренський О. П.
https://github.com/odorenskyi/

ВИКОНАВ
студент академічної групи
КІ22-2
Ткаченко О. Ю.

ПЕРЕВІРИВ
ст. викладач
кафедри кібербезпеки
та програмного забезпечення
Собінов О. Г.

Кропивницький – 2023

Мета: набуття ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів C++ для керування динамічною пам’яттю та бінарними файловими потоками.

Варіант №10

Завдання: 1. У складі команди IT-проекта розробити програмні модулі оброблення динамічної структури даних.

2. Реалізувати програмний засіб на основі розроблених командою IT-проекта модулів.

Склад команди IT-проекта:
Карпова Єлизавета(https://github.com/odorenskyi/Karpova-Yelyzaveta-KI222),
Тітарова Анастасія(https://github.com/odorenskyi/Titarova-Anastasiia-KI222),
Ткаченко Олексій(https://github.com/odorenskyi/Tkachenko-Oleksii-KI222)

Додаток А містить лістинг проекту.

Висновок: Виконавши цю лабораторну роботу з теми “ Реалізація програмних засобів оброблення динамічних структур даних та бінарних файлів” я набув ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів C++ для керування динамічною пам’яттю та бінарними файловими потоками.

У процесі вивчення даної теми було розглянуто різні аспекти реалізації програмних засобів для роботи з динамічними структурами даних. Було досліджено основні типи динамічних структур, такі як списки, стеки, черги та дерева, і розглянуто їх внутрішню структуру та принципи роботи. Виявлено, що використання правильної структури даних може значно полегшити роботу з обробкою і збереженням інформації.

Також було проаналізовано важливі аспекти роботи з бінарними файлами. Виявлено, що бінарні файли дозволяють ефективно зберігати дані у вигляді послідовності бітів, що забезпечує компактність та швидкий доступ до інформації. Було розглянуто протоколи читання та запису бінарних файлів, а також показано, як правильно використовувати їх для зберігання структурованих даних.

Загальний висновок полягає в тому, що реалізація програмних засобів для оброблення динамічних структур даних та бінарних файлів є критично важливою для розробки ефективного та оптимізованого програмного забезпечення. Правильний вибір та використання структур даних, а також правильна робота з бінарними файлами можуть суттєво поліпшити продуктивність програми, забезпечити швидкий доступ до даних та ефективне використання ресурсів комп'ютера.

ДОДАТОК А

```
#include <iostream>
#include <fstream>
#include <vector>

struct Product {
    int sectionNumber;
    std::string sectionName;
    std::string groupCode;
    std::string productName;
};

class ProductDirectory {
private:
    std::vector<Product> products;

public:
    void loadDirectoryFromFile(const std::string& filename) {
        std::ifstream file(filename);
        if (!file) {
            std::cout << "Cannot open file: " << filename << std::endl;
            return;
        }

        products.clear();
        Product product;
        std::string line;
        while (std::getline(file, line)) {
            if (line.empty()) {
                continue;
            }

            if (line.find("Розділ") != std::string::npos) {
                product.sectionNumber =
std::stoi(line.substr(line.find_first_of("0123456789")));
                product.sectionName = line.substr(line.find_first_of(".") + 2);
            } else if (line.find("Група") != std::string::npos) {
                product.groupCode = line.substr(line.find_first_of("0123456789"));
                product.productName = line.substr(line.find_first_of(" ") + 1);
                products.push_back(product);
            }
        }

        file.close();
        std::cout << "Directory loaded from file: " << filename << std::endl;
```

```
    }

    void saveDirectoryToFile(const std::string& filename) {
        std::ofstream file(filename);
        if (!file) {
            std::cout << "Cannot open file: " << filename << std::endl;
            return;
        }

        for (const auto& product : products) {
            file << "Розділ " << product.sectionNumber << ". " <<
product.sectionName << std::endl;
            file << "Група " << product.groupCode << " " << product.productName <<
std::endl;
        }

        file.close();
        std::cout << "Directory saved to file: " << filename << std::endl;
    }

    void searchProductByCode(const std::string& productCode) {
        bool found = false;
        for (const auto& product : products) {
            if (product.groupCode == productCode) {
                std::cout << "Product found: " << "Розділ " << product.sectionNumber
<< ". " << product.sectionName << ", "
                          << "Група " << product.groupCode << " " <<
product.productName << std::endl;
                found = true;
            }
        }

        if (!found) {
            std::cout << "Product not found." << std::endl;
        }
    }

    void addProduct(const Product& newProduct) {
        products.push_back(newProduct);
        std::cout << "Product added to the directory." << std::endl;
    }

    void removeProduct(const std::string& productCode) {
        for (auto it = products.begin(); it != products.end(); ++it) {
            if (it->groupCode == productCode) {
```

```
            products.erase(it);
            std::cout << "Product removed from the directory." << std::endl;
            return;
        }
    }

    std::cout << "Product not found." << std::endl;
  }
};

int main() {
  ProductDirectory directory;
  directory.loadDirectoryFromFile("directory.txt");

  int choice;
  std::string productCode, productName;
  Product newProduct;

  do {
    std::cout << "Menu:\n";
    std::cout << "1. Search product by code\n";
    std::cout << "2. Save directory to file\n";
    std::cout << "3. Add new product to directory\n";
    std::cout << "4. Remove product from directory\n";
    std::cout << "5. Exit\n";
    std::cout << "Enter your choice: ";
    std::cin >> choice;

    switch (choice) {
      case 1:
        std::cout << "Enter product code: ";
        std::cin >> productCode;
        directory.searchProductByCode(productCode);
        break;
      case 2:
        directory.saveDirectoryToFile("directory.txt");
        break;
      case 3:
        std::cout << "Enter section number, section name, group code, and
product name: ";
        std::cin >> newProduct.sectionNumber >> newProduct.sectionName >>
newProduct.groupCode >> newProduct.productName;
        directory.addProduct(newProduct);
        break;
      case 4:
```

```
            std::cout << "Enter product code: ";
            std::cin >> productCode;
            directory.removeProduct(productCode);
            break;
        case 5:
            directory.saveDirectoryToFile("directory.txt");
            std::cout << "Exiting the program." << std::endl;
            break;
        default:
            std::cout << "Invalid choice. Please try again." << std::endl;
            break;
    }

    std::cout << std::endl;
} while (choice != 5);

return 0;
}
```