

Міністерство освіти і науки України
Чернівецький національний університет імені Юрія Федьковича
Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Кафедра математичних проблем управління і кібернетики

ЗВІТ
про проходження літньої обчислювальної практики
(термін проходження 17.06 – 01.07.2023)
на базі практики кафедри МПУіК

Виконавець: студент 141Б групи
Ленука Ю. О.

Керівник практики:
доц. Фратавчан В.Г.

Оцінка за національною шкалою

Кількість балів _____

Оцінка за шкалою ECTS _____

Підпис керівника практики

Дата _____

Чернівці – 2023

Частина 1

Завдання №1. Рекурсивні підпрограми.

Варіант 4

Постановка задачі

Задано натуральне число n . Розробити програму для обчислення заданих сум, використовуючи рекурсивні процедури або функції.

$$4. \sum_{k=1}^n \frac{a_k b_k}{(k+1)!}, \quad \begin{aligned} a_1 &= 1, & a_k &= 0.3b_{k-1} + 0.2a_{k-1}, \\ b_1 &= 1, & b_k &= a_{k-1}^2 + b_{k-1}^2. \end{aligned}$$

Програмна реалізація (код програми)

Підключені модулі

```
1  #include <iostream>
2  #include <iomanip>
```

Рекурсивні функції для обчислення факторіалу та обчислення степеня

```
unsigned long long fact(unsigned int x)
{
    return (x == 0 || x == 1) ? 1 : x * fact(x - 1);
}

double pow(double x, unsigned int n)
{
    return (x == 1 || x == 0 || n == 0) ? 1 : x * pow(x, n - 1);
}
```

Рекурсивна функція для знаходження суми

```
double sum(unsigned int k, int n, double& a, double& b)
{
    double rez = (a * b) / fact(k + 1);
    std::cout << "a = " << std::setw(15) << a << " b = " << std::setw(15) << b << " rez = " << std::setw(15) << rez << '\n';
    a = 0.3 * a + 0.2 * b;
    b = pow(a, 2) + pow(b, 2);
    return (k == n) ? rez : rez + sum(++k, n, a, b);
}
```

Функція main()

```
int main()
{
    int n, k = 1;
    double a = 1, b = 1;
    std::cout << "Enter n: "; std::cin >> n;
    std::cout << "\nFinal rez = " << sum(k, n, a, b);
    return 0;
}
```

Скріншоти роботи програми (введені дані та результат виконання програми)

```
Enter n: 5
a =          1  b =          1  rez =          0.5
a =          0.5 b =          1.25 rez =          0.104167
a =          0.4 b =          1.7225 rez =          0.0287083
a =          0.4645 b =          3.18277 rez =          0.01232
a =          0.775903 b =          10.732 rez =          0.0115653
Final rez = 0.65676
```

Висновки

Під час виконання завдання засвоїли принципи організації рекурсивних процесів та отримали практичні навички розроблення і використання рекурсивних процедур та функцій.

Рекурсія – це такий спосіб організації обчислювального процесу, при якому підпрограма під час виконання звертається сама до себе.

Розробили програму для обчислення суми від числа k до числа n . Реалізовано рекурсивні функції для обчислення факторіала, степеня та обчислення суми.

Завдання №2. Динамічні структури даних

Варіант 4

Постановка задачі

4. Розробити програму, яка створює списки $L1$ і $L2$, елементами яких є слова із великих латинських літер. Знаходить всі слова списку $L1$, що не містяться у $L2$, і друкує їх, розділюючи пробілами, в оберненому порядку до їх розміщення.

Програмна реалізація (код програми)

Модулі програми

Код класу list:

```
#pragma once
#include <iostream>
#include <initializer_list>
template <class T>
class list
{
private:
    struct Node
    {
        T data;
        Node* prev;
        Node* next;
        Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
    };
    Node* head;
    Node* tail;
    int size;
public:
    list() : head(nullptr), tail(nullptr), size(0) {}
    list(std::initializer_list<T> values) : list() {
        for (const auto& value : values) {
            push_back(value);
            size++;
        }
    }
};
```

```

    }
}
~list()
{
    clear();
}
list& operator=(const list& other)
{
    if (this != &other)
    {
        clear();
        std::cout << other.size;
    }
    size = other.size;
    return *this;
}
void push_back(const T& value)
{
    Node* newNode = new Node(value);
    if (head == nullptr)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        newNode->prev = tail;
        tail->next = newNode;
        tail = newNode;
    }
    size++;
}
void push_front(const T& value)
{
    Node* newNode = new Node(value);

```

```

    if (head == nullptr)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
    size++;
}

void pop_back()
{
    if (tail == nullptr)
    {
        return;
    }
    if (head == tail)
    {
        delete tail;
        head = nullptr;
        tail = nullptr;
    }
    else
    {
        Node* prevNode = tail->prev;
        prevNode->next = nullptr;
        delete tail;
        tail = prevNode;
    }
    size--;
}

void pop_front()

```

```

{
    if (head == nullptr)
    {
        return;
    }
    if (head == tail)
    {
        delete head;
        head = nullptr;
        tail = nullptr;
    }
    else
    {
        Node* nextNode = head->next;
        nextNode->prev = nullptr;
        delete head;
        head = nextNode;
    }
    size--;
}

void clear()
{
    Node* current = head;
    while (current != nullptr) {
        Node* nextNode = current->next;
        delete current;
        current = nextNode;
    }
    head = nullptr;
    tail = nullptr;
    size = 0;
}

int length() const
{
    return size;
}

```

```

}

void print() const
{
    for (const auto& element : *this) {
        std::cout << element << " ";
    }
    std::cout << std::endl;
}

class Iterator {
private:
    Node* currentNode;

public:
    Iterator(Node* node) : currentNode(node) {}
    Iterator& operator++()
    {
        if (currentNode != nullptr) {
            currentNode = currentNode->next;
        }
        return *this;
    }
    T& operator*() const
    {
        return currentNode->data;
    }
    bool operator!=(const Iterator& other) const
    {
        return currentNode != other.currentNode;
    }
};

Iterator begin() const
{
    return Iterator(head);
}

Iterator end() const

```



```

{
    return Iterator(nullptr);
}
};

```

Функція, яка виконує завдання

```

template<typename T>
list<T>& unique_rev(list<T>& L1, list<T>& L2)
{
    list<T>* rez = new list<T>;
    for (const auto& word : L1) {
        bool found = false;
        for (const auto& word2 : L2) {
            if (word == word2) {
                found = true;
                break;
            }
        }
        if (!found) {
            rez->push_front(word);
        }
    }
    return *rez;
}

```

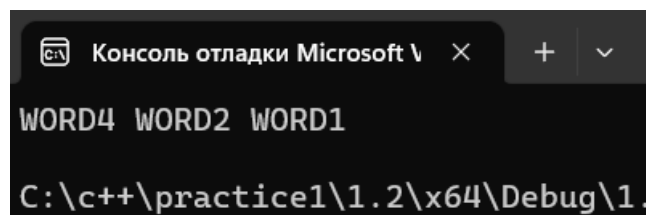
Функція main()

```

int main() {
    list<std::string> L1 = { "WORD1", "WORD2", "WORD3", "WORD4", "WORD5" };
    list<std::string> L2 = { "WORD3", "WORD5" };
    (unique_rev<std::string>(L1, L2)).print();
    return 0;
}

```

Скріншоти роботи програми (введені дані та результат виконання програми)



```

Консоль отладки Microsoft Visual Studio
WORD4 WORD2 WORD1
C:\c++\practice1\1.2\x64\Debug\1.

```

Висновки

Під час виконання завдання засвоїли принципи роботи з динамічними структурами даних та отримали практичні навички розроблення алгоритмів і програм для роботи зі списками.

Було реалізовано двозв'язний список. Програма дозволяє вводити Знаходить всі слова списку L1, що не містяться у L2 , і друкує їх, розділяючи пробілами, в оберненому порядку до їх розміщення.

Зв'язаний список — одна з найважливіших структур даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів, а вказівниками, які входять в склад елементів списку та вказують на наступний за даним елемент або на наступний та попередній елементи.

Завдання №3. Обробка файлових даних

Варіант 4

Постановка задачі

4. Задано два тексти, слова в яких розділені пробілами і розділовими знаками. Розробити програму, яка вилучає із першого тексту всі слова, що містяться у другому тексті.

Програмна реалізація (код програми)

Модулі програми

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

Функція, яка читає текст з файлу і розбиває його на слова

```
void readFile(std::string file, std::string& text, std::vector<std::string>& words)
{
    std::ifstream inputFile(file);
    if (inputFile.is_open())
    {
        std::string word;
        while (inputFile >> word)
        {
            words.push_back(word);
        }
        for (auto& word : words) text += word + ' ';
        inputFile.close();
    }
    else
    {
        std::cerr << "Помилка відкриття файлу для читання" << std::endl;
    }
}
```

Функція для вилучення слів, що містяться у першому і другому тексті одночасно

```
void extractCommonWords(std::string& text1, const std::string& text2, std::vector<std::string>& words2)
{
    std::vector<std::string> commonWords;
    for (std::string word : words2)
    {
        if (text1.find(word) <= text1.size())
        {
            commonWords.push_back(word);
        }
    }
    for (auto& word : commonWords)
    {
        text1.erase(text1.find(word), word.size());
    }
}
```

Функція для запису у файл

```
void writeFile(std::string file, std::string& text)
{
    std::ofstream outputFile(file, std::ios::app);
    if (outputFile.is_open())
    {
        outputFile << text << '\n';
        outputFile.close();
        std::cout << "Результат записано у файл 'result.txt'" << std::endl;
    }
    else
    {
        std::cerr << "Помилка відкриття файлу для запису" << std::endl;
    }
}
```

Функція main()

```
int main()
{
    setlocale(LC_ALL, "ukr");
    std::string text1;
    std::string text2;
    std::vector<std::string> words1;
    std::vector<std::string> words2;
    readFile("text1.txt", text1, words1);
    readFile("text2.txt", text2, words2);
    std::cout << "Text 1: " << text1 << '\n';
    std::cout << "Text 2: " << text2 << '\n';
    extractCommonWords(text1, text2, words2);
    std::cout << "Result: " << text1 << '\n';
    writeFile("result.txt", text1);
    return 0;
}
```

Перший текст: The quick brown fox jumps, over the lazy dog.

Другий текст: The dog jumps in the lake.

Скріншоти роботи програми (введені дані та результат виконання програми)

```
Text 1: The quick brown fox jumps, over the lazy dog.
Text 2: The dog jumps in the lake.
Result: quick brown fox , over lazy .
Результат записано у файл 'result.txt'
```

Запис результату у текстовий файл

```
quick brown fox , over lazy .
```

Висновки

Під час виконання завдання ознайомились з поняттям файла та методами доступу до файлів. Засвоїли принцип організації текстових файлів та отримали практичні навички роботи з текстовими файлами.

Файлом називають спосіб зберігання інформації на фізичному пристрої. Файл - це поняття, яке застосовується до всього – від файлу на диску до терміналу. В C++ відсутні оператори для роботи з файлами. Всі необхідні дії виконуються за допомогою функцій, включених в стандартну бібліотеку. Вони дозволяють працювати з різними пристроями, такими, як диски, принтер, комунікаційні канали і т.д. Ці пристрої сильно відрізняються один від одного. Однак файлова система перетворює їх в єдине абстрактне логічне пристрій, який називається потоком.

Було реалізовано програму, яка знаходить і вилучає всі слова з першого тексту, що входять у другий текст. Створено функції запису(в комбінації з функцією для розбиття тексту на окремі слова) та зчитування з файлу.

Частина 2

Завдання 1

Тема: *Контейнери стандартної бібліотеки шаблонів мови C++.*

Варіант 4

Постановка задачі

Задача_1_4. Дано набір дійсних чисел. Вивести всі елементи цього набору в тому ж порядку. Використовувати ітератори `ptin_iterator`, `ptout_iterator` і алгоритм `copy`.

Програмна реалізація (код програми)

Підключені модулі

```
#include <iostream>
#include <vector>
#include <algorithm>
```

Створення вектору та ітераторів

```
std::vector<double> numbers = { 1.2, 3.4, 5.6, 7.8 };

auto begin = numbers.begin();
auto end = numbers.end();

std::ostream_iterator<double> output(std::cout, " ");
```

Виведення результату через алгоритм `copy`

```
std::copy(begin, end, output);
std::cout << std::endl;
```

Скріншоти роботи програми (введені дані та результат виконання програми)

```
1.2 3.4 5.6 7.8
```

Висновки

Під час виконання завдання задається набір дійсних чисел. Програма вивела числа з вектору в консоль. Було використано ітератори `ptin_iterator` (`auto begin = numbers.begin(); auto end = numbers.end();`),

`ptout_iterator` (`std::ostream_iterator<double> output(std::cout, " ");`) та алгоритм `copy` та контейнер `vector`.

Вектор (англ. *vector*) в C++ є динамічним масивом, який представляє собою послідовність елементів з можливістю швидкого доступу до будь-якого елемента за його індексом. Вектори в C++ забезпечують автоматичне збільшення розміру при додаванні нових елементів.

Алгоритм *copy* у C++ використовується для копіювання елементів з одного діапазону у інший.

Аргументи функції:

- *first, last*: ітератори, що задають початок та кінець вхідного діапазону, з якого будуть копіювані елементи.
- *OutputIterator result*: ітератор, що задає початок вихідного діапазону, в який будуть скопійовані елементи.

Завдання 2

Варіант 12

Постановка задачі

Задача_2_11. Дано список L з парною кількістю елементів. Копіювати в кінець списку всі елементи, розташовані в першій половині, замінивши при цьому негативні елементи на нулі і розташувавши скопійовані елементи в зворотному порядку. Використовувати алгоритм `replace_copy_if`, ітератор вставки та зворотні ітератори, а також функцію `advance`.

Задача_2_12. Розв'язати завдання Задача_2_11, де замість списку L заданий дек(двобічна черга) D з парною кількістю елементів. Оскільки операція вставки робить усі ітератори дек(двобічна черга)а недійсними, вирішення завдання використовувати допоміжний дек(двобічна черга) D0. Ініціалізувати дек(двобічна черга) D0 першою половиною елементів дек(двобічна черга)а D, після чого

8

Об'єктно – орієнтоване програмування мовою C++

Лазорик В.В.

застосувати алгоритм `replace_copy_if`, використовуючи дек(двобічна черга) D0 як джерело, а дек(двобічна черга) D як приймач даних.

Програмна реалізація (код програми)

Підключені модулі

```
#include <iostream>
#include <deque>
#include <algorithm>
```

Предикат

```
bool isNegative(int num)
{
    return num < 0;
}
```


Функція main() з кодом завдання

```
int main()
{
    std::deque<int> D = { 1, -2, 3, -4, 5, -6 };

    std::deque<int> D0(D.begin(), D.begin() + D.size() / 2);

    std::replace_copy_if(D0.rbegin(), D0.rend(), std::back_inserter(D), isNegative, 0);

    for (const auto& num : D) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Скріншоти роботи програми (введені дані та результат виконання програми)

1 -2 3 -4 5 -6 3 0 1

Висновки

Під час виконання завдання було використано контейнер deque та алгоритм replace_copy_if.

Дека (англ. deque) в C++ - контейнер стандартної бібліотеки, який представляє собою послідовність елементів з можливістю швидкого доступу до початку та кінця послідовності. Ім'я "дека" є скороченням від "double-ended queue" (черга з подвійним кінцем). Дека схожа на вектор (std::vector) та списки (std::list), але має деякі додаткові можливості. Вона дозволяє ефективно вставляти та видаляти елементи як на початку, так і в кінці послідовності. Також, до деки можна швидко звертатися до будь-якого елемента за його індексом.

Завдання 3

Варіант 4

Постановка задачі

Задача_3_3. Даний вектор V , елементами якого є англійські слова, набрані великими літерами. Визначити сумарну довжину слів, що починаються з однієї і тієї ж літери, і вивести всі різні літери, з яких вибираються елементи вектора V разом із сумарною довжиною цих елементів (в алфавітному порядку букв); довжину виводити відразу після відповідної літери. Використовувати допоміжне відображення M , ключами якого є початкові літери елементів вектора V , а значення сумарна довжина цих елементів. При заповненні відображення M не використовувати умовні конструкції (достатньо операцій індексування [], інкремента та функції члена `size` для рядків). Елементи вектора V (при заповненні відображення M) та елементи відображення M (при виведенні отриманих результатів) перебирати в циклі з параметром ітератором відповідного контейнера.

Задача_3_4. Розв'язати задачу Задача_3_3, використовуючи для перебору елементів вектора V і відображення M виклики алгоритму `for_each` або, якщо компілятор підтримує стандарт C++11, цикли `for` елементів контейнера.

Програмна реалізація (код програми)

Підключені модулі

```
#include <iostream>
#include <vector>
#include <map>
```

Функція `main()`

```
int main() {
    std::vector<std::string> V = { "APPLE", "BANANA", "ELEPHANT", "CAT", "DOG", "ATOM" };
    std::map<char, int> M;

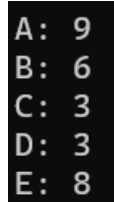
    for (const std::string& word : V) {
        char firstLetter = word[0];
        M[firstLetter] += word.size();
    }

    for (const auto& entry : M) {
        std::cout << entry.first << ": " << entry.second << std::endl;
    }

    return 0;
}
```

Тут через range-based for реалізовано знаходження першої літери, довжини слова і запис у map.

Скріншоти роботи програми (введені дані та результат виконання програми)



```
A: 9
B: 6
C: 3
D: 3
E: 8
```

Висновки

Під час виконання завдання було використано контейнери vector, map і range-based for. результат угруповано у вигляді класу map<std::string, int>.

Map в C++ є контейнером стандартної бібліотеки, який представляє собою асоціативний масив, що містить пари ключ-значення. Ключі у mapі є унікальними, тобто в mapі не може бути декілька елементів з однаковим ключем. Map впорядкований за ключами відповідно до їх сортування, що дає швидкий доступ до значень за ключами. За допомогою оператора [] можна додавати нові елементи, звертатися до значень за ключем або змінювати значення за ключем.