Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Розрахунково-графічна робота

з дисципліни
«Об’єктно-орієнтоване програмування»

Виконала:

студентка групи ІМ-43
Хубеджева Єлизавета Павлівна
номер у списку групи: 26

Перевірив:

Порєв В. М.

Київ 2025

Завдання

Завдання 6. Створити програму для автоматичної побудови #include-ієрархій файлів – залежностей файлів сторонніх проектів C++.

Обгрунтування проектного рішення

Розробка програми для візуалізації #include-ієрархії файлів C++ потребувала ретельного обгрунтування ключових архітектурних рішень.

Перш за все, було обрано мову програмування Python через її високу продуктивність розробки, багату стандартну бібліотеку та крос-платформеність. Python дозволяє швидко реалізовувати складні алгоритми обробки тексту та графів, що є критичним для аналізу залежностей між файлами.

Графічний інтерфейс було реалізовано за допомогою бібліотеки Tkinter, яка є стандартною для Python і не вимагає додаткових встановлень, що спрощує розповсюдження програми. Архітектура програми побудована за модульним принципом, що дозволяє чітко розділити відповідальність: окремий модуль для аналізу файлів, окремий для візуалізації та головний модуль для керування інтерфейсом користувача. Такий підхід підвищує зрозумілість коду, полегшує тестування та можливість майбутніх розширень. Алгоритм аналізу залежностей базується на модифікованому пошуку в глибину (DFS) з механізмом виявлення циклічних залежностей, що є оптимальним для роботи з графами, які можуть містити цикли.

Аналіз можливих варіантів рішення завдання

Перед початком розробки було розглянуто кілька альтернативних шляхів реалізації. Серед мов програмування розглядалися C++ та Python. C++ пропонував найвищу продуктивність та природню сумісність з аналізованою мовою, проте розробка графічного інтерфейсу на C++ є значно більш трудомісткою. Python був обраний як оптимальний баланс між швидкістю розробки, доступністю бібліотек та продуктивністю, достатньою для даної задачі.

Щодо графічних бібліотек, розглядалися Tkinter, PyQt та веб-інтерфейс. PyQt надає більш сучасні та потужні інструменти, але має складніше ліцензування та збільшує розмір дистрибутиву. Веб-інтерфейс на основі HTML/CSS/JavaScript дозволив би створювати дуже сучасний інтерфейс, але вимагав би додаткової серверної частини та ускладнив би архітектуру. Tkinter, незважаючи на дещо застарілий вигляд, є простою, надійною та стандартною бібліотекою, що повністю покриває потреби програми.

Для аналізу залежностей розглядалися алгоритми пошуку в ширину (BFS), пошуку в глибину (DFS) та топологічного сортування. BFS не підходить для визначення рівнів ієрархії. Топологічне сортування працює лише з ациклічними графами, що не завжди відповідає реальним проєктам. DFS з мемоізацією та перевіркою на цикли виявився найбільш підходящим, оскільки ефективно обробляє цикли та природно визначає рівні вкладеності файлів.

Пояснювальна записка

Сучасні програмні проєкти на мові C++ часто складаються з сотень або навіть тисяч файлів, між якими існують складні залежності через директиви препроцесора #include. Розуміння цих залежностей є критично важливим для ефективної розробки, відлагодження та рефакторингу коду. Вручну відстежувати ці залежності практично неможливо, особливо у великих проєктах. Існуючі інструменти часто є або занадто складними, або платними, або не надають наочної візуалізації.

Метою даної роботи є розробка доступного, зручного та ефективного інструменту для автоматизованого аналізу та візуалізації #include-ієрархії файлів у проєктах C++. Програма має дозволити розробникам швидко отримувати уявлення про структуру проєкту, виявляти потенційні проблеми (такі як циклічні залежності) та покращувати архітектуру коду.

Основні завдання, які було поставлено перед розробкою, включають: створення механізму аналізу директив #include у файлах проєкту, побудову графа залежностей між файлами, реалізацію алгоритму визначення рівнів ієрархії, розробку інтуїтивного графічного інтерфейсу для відображення отриманого графа, забезпечення можливостей масштабування та навігації по великим графам, а також обробку особливих випадків, таких як циклічні залежності та відсутні файли.

Актуальність роботи полягає в тому, що незважаючи на появу модулів у C++20, переважна більшість існуючих проєктів продовжує використовувати традиційну модель заголовочних файлів та директив #include. Інструмент для аналізу цих залежностей залишається потрібним для підтримки та модернізації великої кодової бази.

Теоретичні положення

Директива #include є однією з фундаментальних інструкцій препроцесора мов C та C++. Вона виконує текстовий підстановку вмісту одного файлу в інший на етапі препроцесингу. Ця механіка створює мережу залежностей між файлами, яка формує ієрархічну структуру. У найнижчому рівні цієї ієрархії розташовуються файли, які не включають жодних інших файлів проєкту. Наступний рівень складається з файлів, які включають файли найнижчого рівня, і так далі. Така схема ілюструє залежності елементів проєкту та дозволяє зрозуміти його архітектурну будову.

Залежності між файлами природно моделюються орієнтованим графом, де вершини відповідають файлам проєкту, а ребра представляють відношення включення. Орієнтація ребра йде від файлу, що містить директиву #include, до файлу, який включається. Такий граф може бути як ациклічним, так і містити цикли, особливо в разі взаємного включення заголовочних файлів.

Алгоритмічно задача визначення рівнів ієрархії зводиться до аналізу орієнтованого графа. Для ациклічних графів можна застосувати топологічне сортування, однак для графів з циклами потрібні спеціальні підходи. У даній роботі використано модифікований алгоритм пошуку в глибину (DFS), який включає механізм виявлення циклів та мемоізацію проміжних результатів для підвищення ефективності.

Візуалізація графа є окремою важливою задачею. Потрібно ефективно розташувати вершини (файли) на площині таким чином, щоб ієрархія була наочною, а стрілки залежностей не перетиналися без потреби. У програмі реалізовано шарне розташування, де файли одного рівня розміщуються на одній горизонтальній лінії, а рівні розділені вертикальними проміжками. Це забезпечує зрозумілість структури навіть для графів з великою кількістю вершин.

Опис реалізації проекту

Програма реалізована на мові Python 3 і складається з трьох основних модулів: main.py, analysis.py та viewer.py. Головний модуль main.py відповідає за створення графічного інтерфейсу користувача. Він містить клас CppHierarchyApp, який ініціалізує головне вікно програми з меню, панеллю інструментів та областю для відображення графа. Користувач може вибрати папку з проєктом або окремий файл через стандартні діалогові вікна. Після вибору програма запускає аналіз та візуалізацію. Інтерфейс включає кнопки для масштабування, меню для основних операцій та обробку подій миші для прокрутки.

Модуль analysis.py містить клас ProjectAnalyzer, який здійснює аналіз файлів проєкту. Метод scan_directory приймає шлях до папки або файлу, визначає кореневу папку проєкту та здійснює пошук усіх файлів з розширеннями .cpp, .h, .rc. Для кожного файлу метод _parse_includes зчитує його вміст та за допомогою регулярного виразу знаходить усі директиви #include, зберігаючи лише імена файлів (без шляхів). Отримані залежності зберігаються у словнику files_data.

Ключовим елементом модуля є метод _calculate_levels, який визначає рівень кожного файлу в ієрархії. Він використовує рекурсивну функцію get_level_ds, що реалізує DFS з мемоізацією. Для кожного файлу функція обчислює максимальний рівень серед його залежностей та додає 1. Для запобігання зацикленню при виявленні циклічної залежності функція повертає -1. Отримані рівні інвертуються для візуалізації (щоб файли з найбільшою кількістю залежностей були вгорі). Результати зберігаються у словнику levels, де ключами є рівні, а значеннями - списки файлів.

Модуль viewer.py містить клас HierarchyViewer для візуалізації графа. Він створює полотно (Canvas) з вертикальною та горизонтальною прокруткою. Метод _draw_scene_scaled відповідає за малювання всіх елементів графа з урахуванням поточного масштабу. Файли розташовуються за рівнями: кожен рівень центрується по горизонталі, а файли в межах рівня розподіляються рівномірно. Для кожного файлу малюється прямокутний блок з тінню, колір якого залежить від типу файлу (світло-блакитний для .h, світло-жовтий для .cpp, сірий для .rc). У верхній частині блока відображається ім'я файлу, у нижній - список включень (обмежений п'ятьма елементами).

Залежності малюються у вигляді стрілок між блоками. Для уникнення накладання стрілок точки приєднання розраховуються динамічно: на верхній межі дочірнього блока та нижній межі батьківського блока вибираються точки, що відповідають порядку файлів. Посередині кожної стрілки розміщується напис "#include" на білому прямокутнику. Програма підтримує масштабування за допомогою кнопок або коліщатка миші, прокрутку в обох напрямках та автоматичне перемалювання при зміні розміру вікна.

Висновки

В результаті виконання роботи було розроблено повнофункціональну програму для автоматизованої візуалізації #include-ієрархії файлів у проєктах мовою C++. Програма успішно вирішує поставлені завдання: аналізує директиву #include у файлах проєкту, будує граф залежностей, визначає рівні ієрархії та наочно відображає отриману структуру у вигляді діаграми. Реалізований алгоритм DFS з мемоізацією ефективно обробляє навіть великі проєкти та коректно виявляє циклічні залежності.

Основними перевагами розробленої програми є простота використання, інтуїтивний інтерфейс, крос-платформеність та відсутність необхідності додаткових встановлень. Користувач може швидко отримати уявлення про структуру проєкту, виявити потенційні проблеми з залежностями та прийняти обґрунтовані рішення щодо рефакторингу коду.

Модульна архітектура програми забезпечила чітке розділення відповідальності між компонентами, що спрощує супровід та можливість майбутніх розширень. Використання стандартних бібліотек Python робить програму легко переносимого між різними операційними системами.

Загалом, розроблена програма є практично корисним інструментом для розробників C++, який спрощує аналіз та оптимізацію структури проєктів, сприяє підвищенню якості коду та продуктивності розробників. Робота демонструє ефективне застосування алгоритмів теорії графів та принципів об'єктно-орієнтованого програмування для вирішення реальної практичної задачі.

Тексти програм:

main.py:

```
import tkinter as tk
from tkinter import filedialog, messagebox
import os
from analysis import ProjectAnalyzer
from viewer import HierarchyViewer


class CppHierarchyApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Візуалізатор #include-ієрархії C++")
        self.root.geometry("1000x800")


        toolbar = tk.Frame(self.root, bd=1, relief=tk.RAISED)
        toolbar.pack(side=tk.TOP, fill=tk.X)


        btn_folder = tk.Button(toolbar, text="  Вибрати папку",
command=self.open_folder_dialog)
        btn_folder.pack(side=tk.LEFT, padx=5, pady=2)


        btn_file = tk.Button(toolbar, text="  Вибрати файл",
command=self.open_file_dialog)
        btn_file.pack(side=tk.LEFT, padx=5, pady=2)


        tk.Label(toolbar, text="  |  Масштаб: ").pack(side=tk.LEFT)


        tk.Button(toolbar, text=" + ",
command=self.on_zoom_in).pack(side=tk.LEFT, padx=2)
        tk.Button(toolbar, text=" - ",
command=self.on_zoom_out).pack(side=tk.LEFT, padx=2)
        tk.Button(toolbar, text=" 100% ",
command=self.on_zoom_reset).pack(side=tk.LEFT, padx=2)


        self.analyzer = ProjectAnalyzer()
        self.viewer = HierarchyViewer(self.root)


        menubar = tk.Menu(self.root)


        file_menu = tk.Menu(menubar, tearoff=0)
        file_menu.add_command(label="Відкрити папку проекту...",
command=self.open_folder_dialog)
```

```
        file_menu.add_command(label="Відкрити головний файл...",
command=self.open_file_dialog)
        file_menu.add_separator()
        file_menu.add_command(label="Вихід", command=self.root.quit)


        view_menu = tk.Menu(menubar, tearoff=0)
        view_menu.add_command(label="Збільшити (+)",
command=self.on_zoom_in)
        view_menu.add_command(label="Зменшити (-)",
command=self.on_zoom_out)
        view_menu.add_command(label="Скинути (100%)",
command=self.on_zoom_reset)


        menubar.add_cascade(label="Файл", menu=file_menu)
        menubar.add_cascade(label="Вигляд", menu=view_menu)
        self.root.config(menu=menubar)


        self.root.bind("<Configure>", self.on_resize)


    def open_folder_dialog(self):
        folder_selected = filedialog.askdirectory(title="Виберіть папку
проекту")
        if folder_selected:
            self.load_project(folder_selected)


    def open_file_dialog(self):
        file_selected = filedialog.askopenfilename(
            title="Виберіть головний файл проекту (.cpp, .h)",
            filetypes=[
                ("C++ Files", "*.cpp *.h *.c *.hpp *.rc"),
                ("All Files", "*.*")
            ]
        )
        if file_selected:
            self.load_project(file_selected)


    def load_project(self, path):
        if os.path.isfile(path):
            folder_path = os.path.dirname(path)
            display_name = f"File: {os.path.basename(path)}"
        else:
            folder_path = path
```

```
            display_name = "Folder Mode"

        self.root.title(f"Візуалізатор - {folder_path} ({display_name})")

        self.analyzer.scan_directory(path)

        levels, dependencies = self.analyzer.get_hierarchy_data()

        if not levels:
            messagebox.showinfo("Інфо", "Файлів проекту не знайдено або
вони не мають залежностей.")
            return

        self.viewer.set_data(levels, dependencies)

    def on_zoom_in(self):
        self.viewer.zoom_in()

    def on_zoom_out(self):
        self.viewer.zoom_out()

    def on_zoom_reset(self):
        self.viewer.reset_zoom()

    def on_resize(self, event):
        if event.widget == self.root:
            self.viewer.redraw()

if __name__ == "__main__":
    root = tk.Tk()
    app = CppHierarchyApp(root)
    root.mainloop()
```

analysis.py:

```
import os
import re

class ProjectAnalyzer:
    def __init__(self):
        self.files_data = {}
        self.levels = {}
        self.project_root = ""

    def scan_directory(self, path):

        self.files_data = {}

        if os.path.isfile(path):
            self.project_root = os.path.dirname(path)
        else:
            self.project_root = path

        if not self.project_root:
            return

        try:
            all_files = os.listdir(self.project_root)
            code_files = [f for f in all_files if f.endswith(('.cpp',
'.h', '.rc', '.hpp', '.c'))]
        except OSError as e:
            print(f"Error reading directory: {e}")
            return

        for filename in code_files:
            full_path = os.path.join(self.project_root, filename)
            includes = self._parse_includes(full_path)
            self.files_data[filename] = includes

        self._calculate_levels()

    def _parse_includes(self, file_path):
        includes = []
        regex = re.compile(r'^\s*#include\s+["<](.*?)[">]')
```

```
        try:
            with open(file_path, 'r', encoding='utf-8', errors='ignore')
as f:
                for line in f:
                    match = regex.match(line)
                    if match:
                        included_file = match.group(1)
                        includes.append(os.path.basename(included_file))
        except Exception as e:
            print(f"Error parsing {file_path}: {e}")

        return includes

    def _calculate_levels(self):
        memo = {}
        visiting = set()
        local_files = set(self.files_data.keys())

        def get_level_dfs(file_node):
            if file_node in memo:
                return memo[file_node]

            if file_node in visiting:
                return -1

            visiting.add(file_node)

            max_child_level = -1

            dependencies = self.files_data.get(file_node, [])

            for dep in dependencies:

                if dep in local_files:
                    child_lvl = get_level_dfs(dep)
                    if child_lvl > max_child_level:
                        max_child_level = child_lvl

            my_level = max_child_level + 1
```

```
            memo[file_node] = my_level
            visiting.remove(file_node)

            return my_level

        for file in local_files:
            get_level_dfs(file)

        if not memo:
            self.levels = {}
            return

        max_height = max(memo.values())

        self.levels = {}
        for file, height in memo.items():
            visual_level = max_height - height

            if visual_level not in self.levels:
                self.levels[visual_level] = []
            self.levels[visual_level].append(file)

    def get_hierarchy_data(self):
        return self.levels, self.files_data
```

viewer.py:

```
import tkinter as tk
from tkinter import ttk


class HierarchyViewer(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.pack(fill=tk.BOTH, expand=True)


        self.canvas = tk.Canvas(self, bg="white")
        self.v_scroll = ttk.Scrollbar(self, orient="vertical",
command=self.canvas.yview)
        self.h_scroll = ttk.Scrollbar(self, orient="horizontal",
command=self.canvas.xview)


        self.canvas.configure(yscrollcommand=self.v_scroll.set,
xscrollcommand=self.h_scroll.set)


        self.v_scroll.pack(side=tk.RIGHT, fill=tk.Y)
        self.h_scroll.pack(side=tk.BOTTOM, fill=tk.X)
        self.canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)


        self.BASE_BOX_WIDTH = 220
        self.BASE_BOX_HEIGHT = 100
        self.BASE_H_GAP = 100
        self.BASE_V_GAP = 150


        self.scale_factor = 1.0


        self.current_levels = None
        self.current_deps = None


        self.node_info = {}


        self.canvas.bind_all("<MouseWheel>", self._on_mousewheel)
        self.canvas.bind_all("<Shift-MouseWheel>",
self._on_shift_mousewheel)
        self.canvas.bind_all("<Button-4>", self._on_linux_scroll_up)
        self.canvas.bind_all("<Button-5>", self._on_linux_scroll_down)


    def set_data(self, levels_data, dependencies):
```

```
        self.current_levels = levels_data
        self.current_deps = dependencies
        self.redraw()

    def zoom_in(self):
        self.scale_factor += 0.1
        self.redraw()

    def zoom_out(self):
        if self.scale_factor > 0.2:
            self.scale_factor -= 0.1
            self.redraw()

    def reset_zoom(self):
        self.scale_factor = 1.0
        self.redraw()

    def redraw(self):
        if self.current_levels and self.current_deps:
            self._draw_scene_scaled()

    def _draw_scene_scaled(self):
        self.canvas.delete("all")
        self.node_info = {}

        box_w = self.BASE_BOX_WIDTH * self.scale_factor
        box_h = self.BASE_BOX_HEIGHT * self.scale_factor
        h_gap = self.BASE_H_GAP * self.scale_factor
        v_gap = self.BASE_V_GAP * self.scale_factor

        font_title_size = int(9 * self.scale_factor)
        font_content_size = int(8 * self.scale_factor)
        font_arrow_size = int(7 * self.scale_factor)

        if font_title_size < 4: font_title_size = 4
        if font_content_size < 3: font_content_size = 3

        sorted_levels = sorted(self.current_levels.keys())
        max_width = 0
        current_y = 50 * self.scale_factor
```

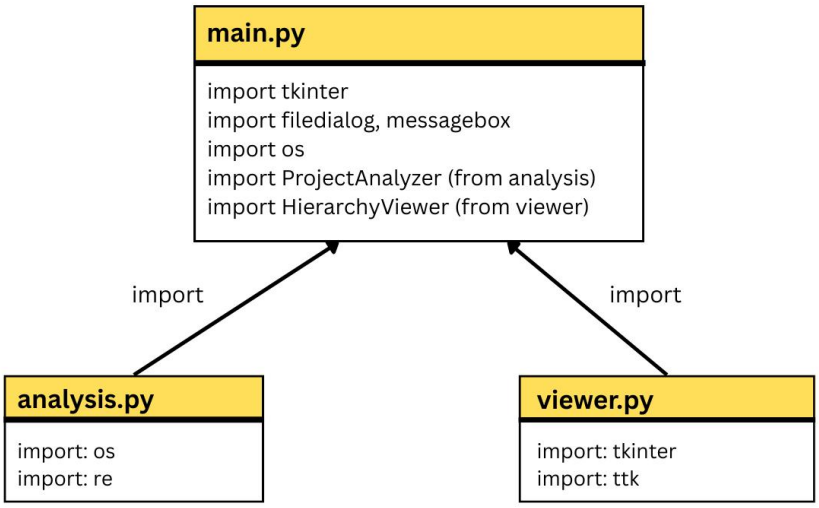
```
        for level in sorted_levels:
            files = self.current_levels[level]
            total_level_width = len(files) * (box_w + h_gap) - h_gap

            canvas_w = self.winfo_width()
            start_x = max(50, (canvas_w - total_level_width) // 2)
            if start_x < 50: start_x = 50

            current_x = start_x

            for filename in files:
                self.node_info[filename] = {
                    'x': current_x,
                    'y': current_y,
                    'w': box_w,
                    'h': box_h
                }
                current_x += box_w + h_gap
                if current_x > max_width: max_width = current_x

            current_y += box_h + v_gap

        shadow_offset = 4 * self.scale_factor
        for filename, coords in self.node_info.items():
            x, y, w, h = coords['x'], coords['y'], coords['w'],
coords['h']
            self.canvas.create_rectangle(
                x + shadow_offset, y + shadow_offset,
                x + w + shadow_offset, y + h + shadow_offset,
                fill="#C0C0C0", outline=""
            )

        parents_of = {f: [] for f in self.node_info}
        children_of = {f: [] for f in self.node_info}

        for parent, includes in self.current_deps.items():
            if parent not in self.node_info: continue
            for child in includes:
                if child in self.node_info:
```

```
                    parents_of[child].append(parent)
                    children_of[parent].append(child)

        for f in self.node_info:
            parents_of[f].sort(key=lambda p: self.node_info[p]['x'])
            children_of[f].sort(key=lambda c: self.node_info[c]['x'])

        for child, parents in parents_of.items():
            if not parents: continue
            child_info = self.node_info[child]
            step_out = child_info['w'] / (len(parents) + 1)

            for i, parent in enumerate(parents):
                parent_info = self.node_info[parent]
                try:
                    child_index = children_of[parent].index(child)
                    total_children = len(children_of[parent])
                except ValueError: continue

                step_in = parent_info['w'] / (total_children + 1)

                start_x = child_info['x'] + step_out * (i + 1)
                start_y = child_info['y']
                end_x = parent_info['x'] + step_in * (child_index + 1)
                end_y = parent_info['y'] + parent_info['h']

                self.canvas.create_line(
                    start_x, start_y, end_x, end_y,
                    arrow=tk.LAST, fill="#444444", width=1.5 *
self.scale_factor
                )

                ratio = 0.40 if (i % 2 == 0) else 0.60
                text_x = start_x + (end_x - start_x) * ratio
                text_y = start_y + (end_y - start_y) * ratio

                text_w = 45 * self.scale_factor
                text_h = 14 * self.scale_factor

                self.canvas.create_rectangle(
```

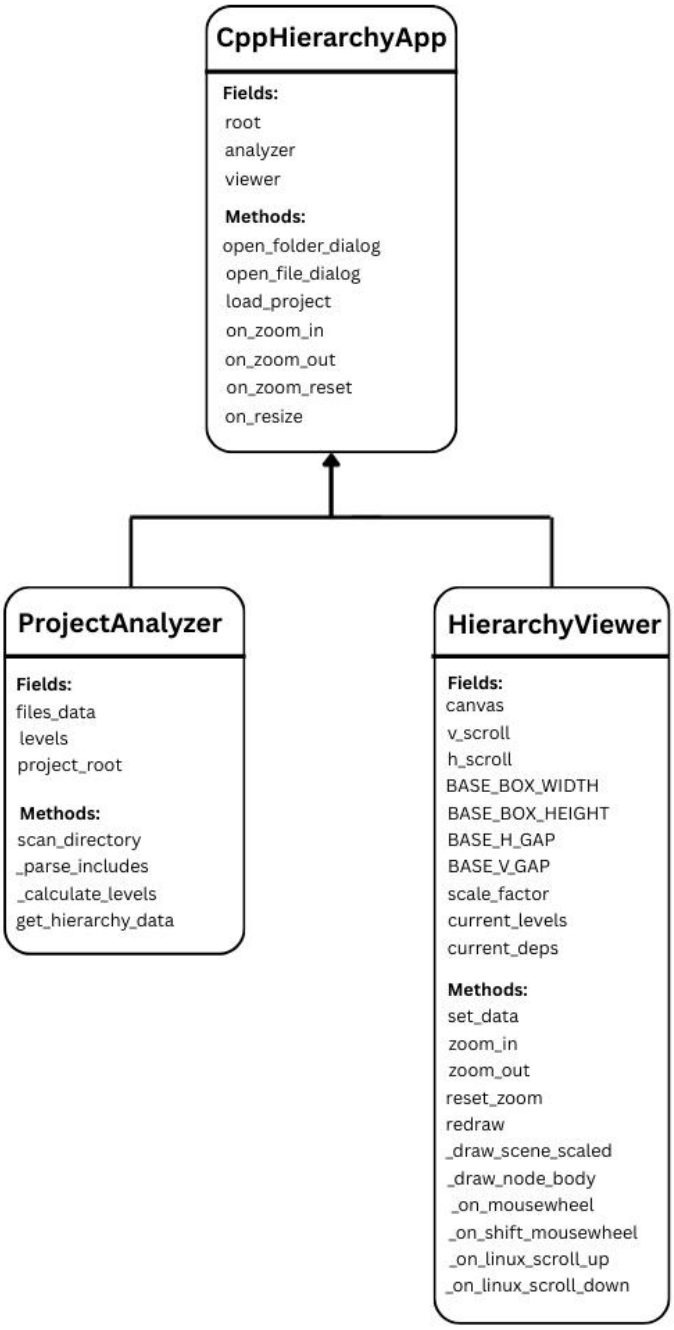
```
                    text_x - text_w/2, text_y - text_h/2,
                    text_x + text_w/2, text_y + text_h/2,
                    fill="white", outline=""
                )
                self.canvas.create_text(
                    text_x, text_y, text="#include",
                    fill="#0000AA", font=("Arial", font_arrow_size,
"italic")
                )

        for filename, coords in self.node_info.items():
            self._draw_node_body(coords, filename,
self.current_deps.get(filename, []), font_title_size, font_content_size)

        self.canvas.config(scrollregion=self.canvas.bbox("all"))

    def _draw_node_body(self, coords, title, includes, f_title,
f_content):
        x, y, w, h = coords['x'], coords['y'], coords['w'], coords['h']

        bg_color = "#E0FFFF" if title.endswith('.h') else "#FFFFE0"
        if title.endswith('.rc'): bg_color = "#E0E0E0"

        self.canvas.create_rectangle(x, y, x+w, y+h, fill=bg_color,
outline="black")

        header_h = 25 * self.scale_factor
        self.canvas.create_line(x, y + header_h, x + w, y + header_h,
fill="black")
        self.canvas.create_text(x + w/2, y + header_h/2, text=title,
font=("Arial", f_title, "bold"))

        content_text = ""
        if includes:
            for inc in includes[:5]:
                content_text += f"#include \"{inc}\"\n"
            if len(includes) > 5:
                content_text += "..."
        else:
            content_text = "(base)"

        self.canvas.create_text(x + 5*self.scale_factor, y + header_h +
5*self.scale_factor,
```

```
                                    text=content_text, anchor="nw",
font=("Consolas", f_content), fill="#555555")

    def _on_mousewheel(self, event):
        self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")

    def _on_shift_mousewheel(self, event):
        self.canvas.xview_scroll(int(-1*(event.delta/120)), "units")

    def _on_linux_scroll_up(self, event):
        self.canvas.yview_scroll(-1, "units")

    def _on_linux_scroll_down(self, event):
        self.canvas.yview_scroll(1, "units")
```

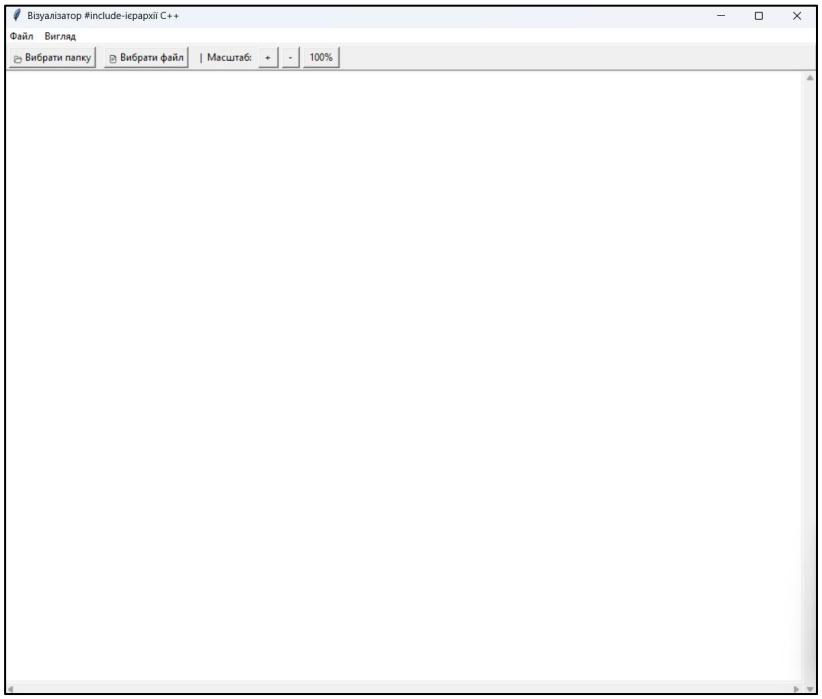
Діаграма #include-ієрархії модулів

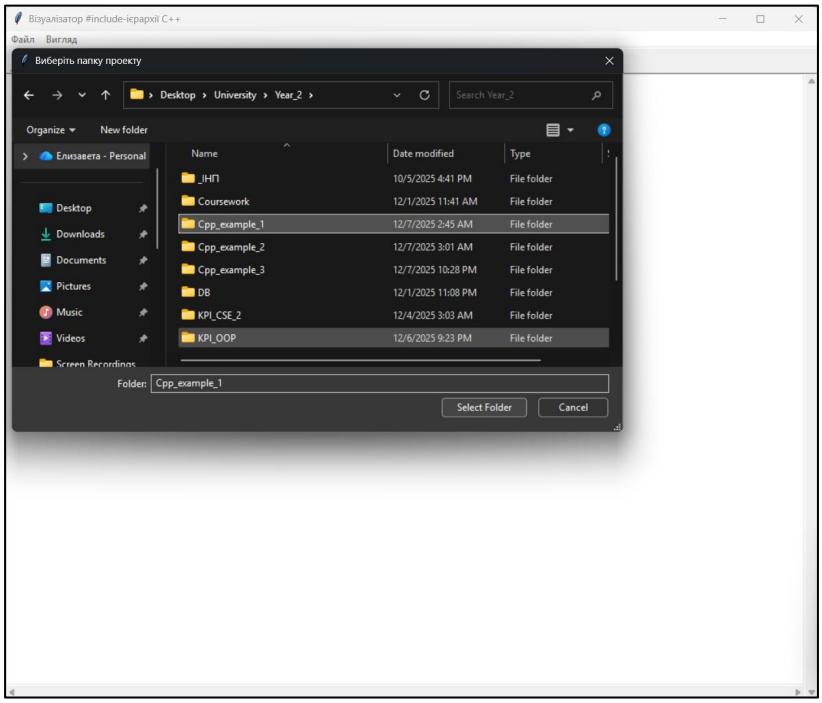


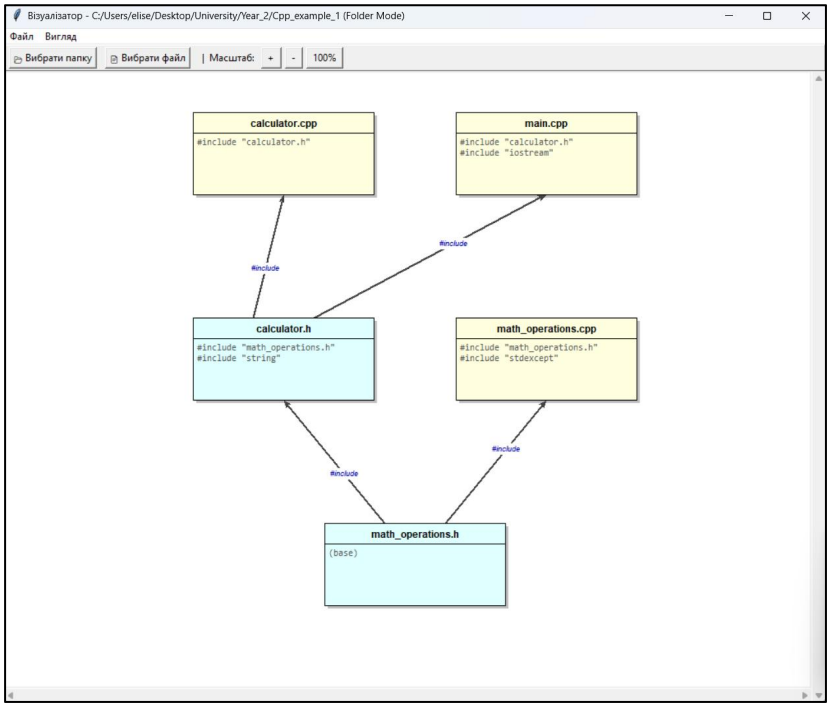
Діаграма класів

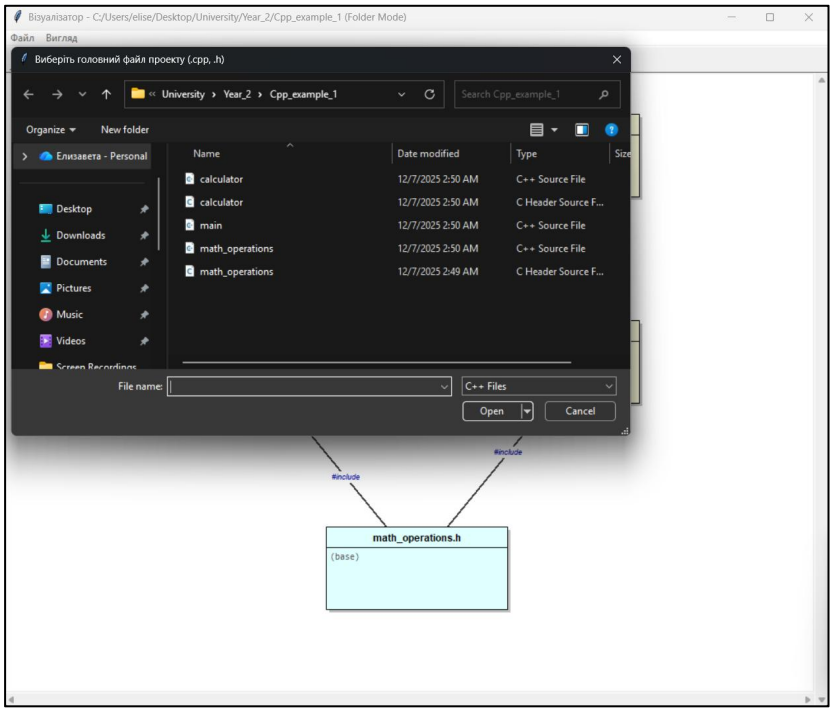


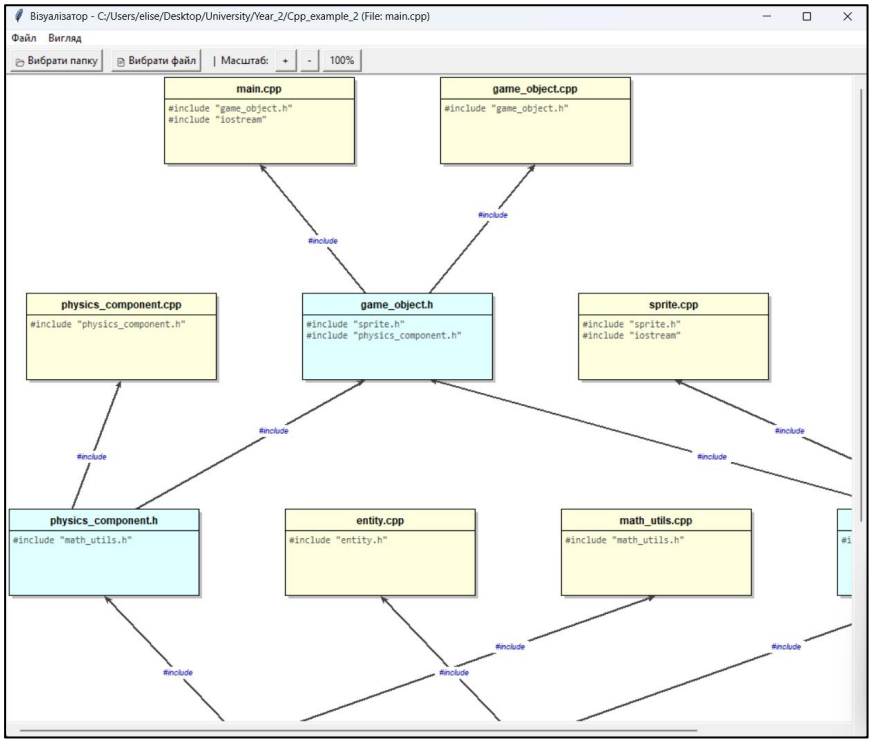
Скріншоти:

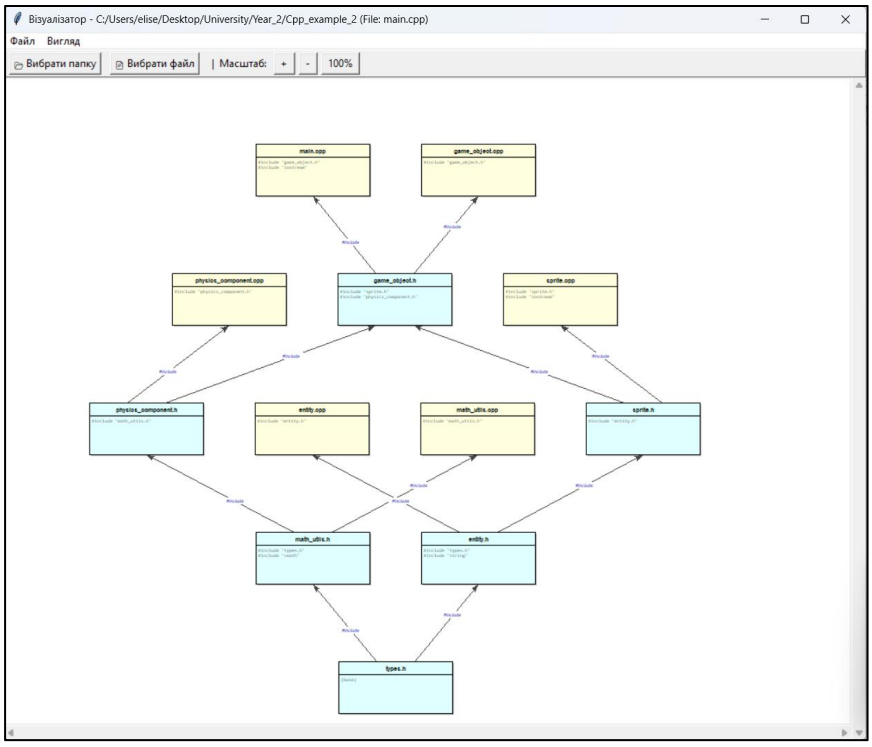

Візуалізатор #include-ієрархії C++
Файл
Вигляд
Вибрати папку
Вибрати файл
Масштаб:
+
-
100%


Візуалізатор #include-ієрархії C++
Файл
Вигляд
Виберіть папку проекту
Desktop
University
Year_2
Search Year_2
Organize
New folder
Елизавета - Personal
Desktop
Downloads
Documents
Pictures
Music
Videos
Screen Recordings
Name
Date modified
Type
_ІНП
10/5/2025 4:41 PM
File folder
Coursework
12/1/2025 11:41 AM
File folder
Cpp_example_1
12/7/2025 2:45 AM
File folder
Cpp_example_2
12/7/2025 3:01 AM
File folder
Cpp_example_3
12/7/2025 10:28 PM
File folder
DB
12/1/2025 11:08 PM
File folder
KPI_CSE_2
12/4/2025 3:03 AM
File folder
KPI_OOP
12/6/2025 9:23 PM
File folder
Folder:
Cpp_example_1
Select Folder
Cancel


Візуалізатор - C:/Users/elise/Desktop/University/Year_2/Cpp_example_1 (Folder Mode)
Файл
Вигляд
Вибрати папку
Вибрати файл
Масштаб:
+
-
100%
calculator.cpp
#include "calculator.h"
main.cpp
#include "calculator.h"
#include "iostream"
#include
#include
calculator.h
#include "math_operations.h"
#include "string"
math_operations.cpp
#include "math_operations.h"
#include "stdexcept"
#include
#include
math_operations.h
(base)


Візуалізатор - C:/Users/elise/Desktop/University/Year_2/Cpp_example_1 (Folder Mode)
Файл
Вигляд
Виберіть головний файл проекту (.cpp, .h)
University
Year_2
Cpp_example_1
Search Cpp_example_1
Organize
New folder
Елизавета - Personal
Desktop
Downloads
Documents
Pictures
Music
Videos
Name
Date modified
Type
calculator
12/7/2025 2:50 AM
C++ Source File
calculator
12/7/2025 2:50 AM
C Header Source F...
main
12/7/2025 2:50 AM
C++ Source File
math_operations
12/7/2025 2:50 AM
C++ Source File
math_operations
12/7/2025 2:49 AM
C Header Source F...
File name:
C++ Files
Open
Cancel
#include
#include
math_operations.h
(base)


Візуалізатор - C:/Users/elise/Desktop/University/Year_2/Cpp_example_2 (File: main.cpp)
Файл
Вигляд
Вибрати папку
Вибрати файл
Масштаб:
+
-
100%
main.cpp
#include "game_object.h"
#include "iostream"
game_object.cpp
#include "game_object.h"
#include
#include
physics_component.cpp
#include "physics_component.h"
game_object.h
#include "sprite.h"
#include "physics_component.h"
sprite.cpp
#include "sprite.h"
#include "iostream"
#include
#include
#include
#include
physics_component.h
#include "math_utils.h"
entity.cpp
#include "entity.h"
math_utils.cpp
#include "math_utils.h"
#include
#include
#include
#include


Візуалізатор - C:/Users/elise/Desktop/University/Year_2/Cpp_example_2 (File: main.cpp)
Файл
Вигляд
Вибрати папку
Вибрати файл
Масштаб:
+
-
100%
main.cpp
game_object.cpp
physics_component.cpp
game_object.h
sprite.cpp
physics_component.h
entity.cpp
math_utils.cpp
sprite.h
math_utils.h
entity.h
types.h