Міністерство освіти і науки, молоді та спорту України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**ЗВІТ**

**Про виконання лабораторної роботи № 1**

«Ознайомлення із основами створення програм з використанням мови С++. Опції компіляції. Типизбірок. Виконавчі файли, статичні та динамічні бібліотеки»

**з дисципліни «Об’єктно-орієнтоване програмування»**

**Лектор:**

доцент кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-14

Губик А. С.

**Прийняв:**

доцент кафедри ПЗ

Дяконюк Л. М.

«\_\_\_» \_\_\_\_\_\_ 2023 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема роботи:** Ознайомлення із основами створення програм з використанням мови С++. Опції компіляції. Типи збірок. Виконавчі файли, статичні та динамічні бібліотеки

**Мета роботи:** Засвоїти принципи створення програм написаних мовою С++, навчитись перетворювати текст програми на виконавчий файл за допомогою командного рядка. Навчитись задавати опції компіляції, створювати різні типи збірок - виконавчі файли, статичні та динамічні бібліотеки

**Теоретичні відомост**і

*Структура програми*

Програма чи бібліотека складається із однієї чи багатьох одиниць трансляції (translation unit).Зазвичай, одиницею трансляції є файл із розширенням .c для коду C та із розширенням .cpp чи .cxx для коду C++.

Кожна одиниця трансляції складається із функцій та змінних. Вони можуть бути як внутрішні (для використання лише в межах однієї одиниці трансляції), так і зовнішні (для зв’язків між одиницями та доступу ззовні).

Для зручності використання функцій та змінних між різними одиницями трансляції існують файли-заголовки (header files) із розширенням .h чи .hpp. Їхній вміст підставляється препроцесором у місце директиви #include.

Файл-заголовок може відповідати якійсь одиниці трансляції, або існувати сам по собі та зазвичай містить оголошення типів, змінних та функцій, які визначені в якійсь одиниці трансляції.

Нижче подано приклад структури C++ програми, що складається із двох файлів-заголовків та трьох одиниць трансляції.

*Структура C файлу*

Для початку варто розуміти відмінність між оголошенням (declaration) та визначенням (definition).Визначення пов’язує ім’я із реалізацією цього імені – даними або кодом:

• визначення змінної вказує компілятору виділити певне місце під цю змінну та, можливо,

заповнити його певним значенням;

• визначення функції вказує компілятору згенерувати код цієї функції.

Оголошення повідомляє компілятору, що визначення із таким ім’ям та типом існує деінде у програмі, ймовірно, в іншому C файлі. Для змінних, визначення поділяють на два види:

• глобальні змінні, які існують протягом усього життя програми і які доступні з багатьох різних

функцій;

• локальні змінні, які існують лише протягом виконання певної функції та доступні лише з цієї функції.

Під «доступні» мається на увазі, що можна використовувати ім’я із визначення для звертання до цих змінних.

Також є кілька особливих випадків:

• статичні (static) локальні змінні є насправді глобальними змінними, бо вони існують протягом усього часу виконання програми, незважаючи на те, що вони доступні лише із однієї функції;

Для функцій, як і для глобальних змінних, ключове слово «static» обмежує доступ до неї по імені лише до функцій із того ж C файлу.

Для глобальних та локальних змінних, ми також розрізняємо, ініціалізована змінна чи ні, тобто чи місце, пов’язане із конкретним ім’ям, попередньо заповнене певним значенням.

Нарешті, дані можуть зберігатись у пам’яті, що виділена динамічно за допомогою malloc чи new. Вона не пов’язана із певним ім’ям, доступ до неї відбувається за допомогою вказівників – іменованих змінних, які містять адресу неіменованої адреси у пам’яті. Ці частини пам’яті звільняються за допомогою free або delete.

**Індивідуальне завдання (Варіант 3)**

*Частина 1. Компіляція. Компонування*

1. Завантажити та встановити LLVM

2. У файловій системі створити директорію ‘oop\_lab1\_prog’ та файли ‘lab1\_main.cpp’,

‘lab1\_utils.h’, ‘lab1\_utils.cpp’. Текст файлів подано у Додатку 2. В усіх файлах необхідно

замінити NN у суфіксі ‘\_NN’ на номер варіанту, наприклад \_01 для першого варіанту, \_02 для другого і так далі.

3. Відкрити командний рядок та перейти у директорію ‘oop\_lab1\_prog’. Скомпілювати файл ‘lab1\_main.cpp’ виконавши команду, подану нижче. Створиться об’єктний файл ‘lab1\_main.o’

5. Скомпілювати файл ‘lab1\_utils.cpp’ виконавши команду, подану нижче. Створиться об’єктний

файл ‘lab1\_utils.o’

6. Проглянути вміст об’єктного файлу ‘lab1\_utils.o’ виконавши команду, подану нижче.

Пояснити значення колонки Class для кожного символу. Рядок для символу ‘@feat.00’ та

символів, що починаються символом підкреслення, можна пропустити.

7. Скомпонувати два об’єктних файли у виконавчий виконавши команду, подану нижче.

Створиться файл ‘lab1.exe’

8. Виконайте отриманий lab1.exe передаючи йому різну кількість параметрів і переконайтесь,

що програма працює коректно. Для цього перевірте код виконання програми за допомогою

інструкції echo %errorlevel%. Змінна %errorlevel% містить код завершення програми, що

виконувалась останньою.

9. Побудувати діаграму структури програми, на якій відобразити об’єктні файли із символами та

зв’язки між ними

*Частина 2. Опції компіляції, типи збірок*

Створіть два типи збірки – для випуску (Release) та відлагодження (Debug).

Для створення збірки для випуску виконайте такі кроки:

1. Відкрити командний рядок та перейти у директорію ‘oop\_lab1\_prog’. Скомпілювати файл

‘lab1\_main.cpp’ виконавши команду, подану нижче. Створиться об’єктний файл

lab1\_main.o

lab2\_utils.o

‘lab1\_main\_rel.o’. Опція -O3 задає третій (найвищий) рівень оптимізації. Опція -flto вмикає

оптимізацію під час компонування.

2. Скомпілювати файл ‘lab1\_utils.cpp’ виконавши команду, подану нижче. Створиться об’єктний

файл ‘lab1\_utils\_rel.o’.

3. Скомпонувати два об’єктних файли у виконавчий файл, застосувавши команду, подану

нижче. Створиться файл ‘lab1\_rel.exe’.

Прапорець -fuse-ld=lld-link потрібен для того, щоб використовувати lld-link компонувальник,

який вміє працювати із опцією -flto.

Для створення збірки для відлагодження виконайте такі кроки:

1. Відкрити командний рядок та перейти у директорію ‘oop\_lab1\_prog’. Скомпілювати файл

‘lab1\_main.cpp’ виконавши команду, подану нижче.

Створиться об’єктний файл ‘lab1\_main\_dbg.o’. Опція -O0 задає нульовий рівень оптимізації

(тобто оптимізація вимкнена). Опція -g вмикає генерацію потрібної для відлагодження

інформації.

2. Скомпілювати файл ‘lab1\_utils.cpp’, виконавши команду, подану нижче. Створиться

об’єктний файл ‘lab1\_utils\_dbg.o’.

3. Скомпонувати два об’єктних файли у виконавчий, застосувавши команду, подану нижче.

Створиться файл ‘lab1\_gdb.exe’.

*Частина 3. Статичні та динамічні бібліотеки.*

1. У файловій системі поруч із директорією ‘oop\_lab1\_prog’ створити директорію ‘oop\_lab1\_lib’

та файли ‘lab1\_lib.h’, ‘lab1\_lib.cpp’, ‘lab1\_utils.cpp’. Текст файлів подано у Додатку 2.

2. Відкрити командний рядок та перейти у директорію ‘oop\_lab1\_lib’. Скомпілювати файл

‘lab1\_lib.cpp’ виконавши команду, подану нижче. Створиться об’єктний файл ‘lab1\_lib.o’.

3. Скомпілювати файл ‘lab1\_utils.cpp’ виконавши команду, подану нижче. Створиться об’єктний

файл ‘lab1\_utils.o’.

4. Об’єднайте об’єктні файли ‘lab1\_lib.o’ та ‘lab1\_utils.o’ у статичну бібліотеку виконавши

команду, подану нижче. Створиться файл ‘lab1\_lib.lib’.

5. Прогляньте вміст статичної бібліотеки та переконайтесь, що в ній містяться файли ‘lab1\_lib.o’

та ‘lab1\_utils.o’

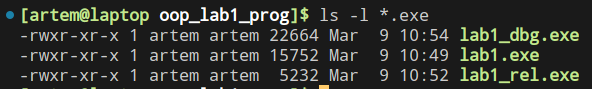
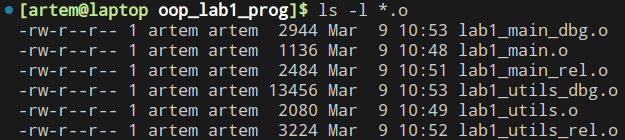
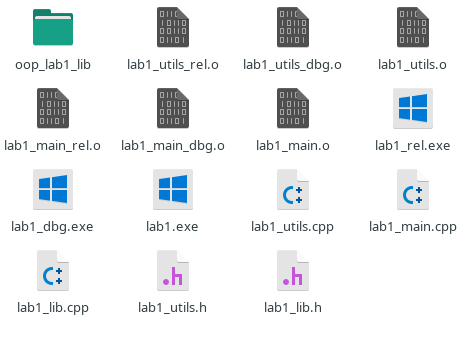
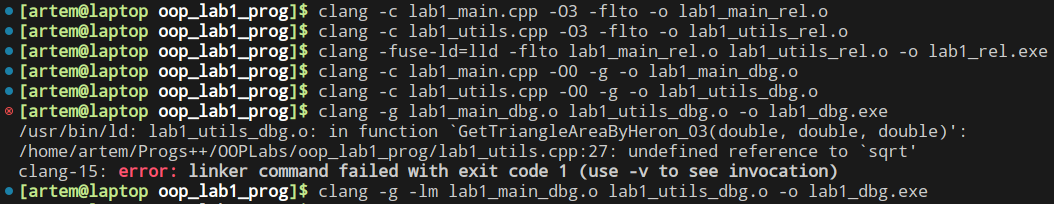
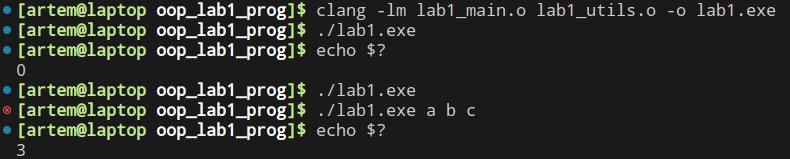
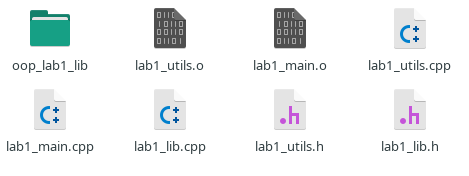
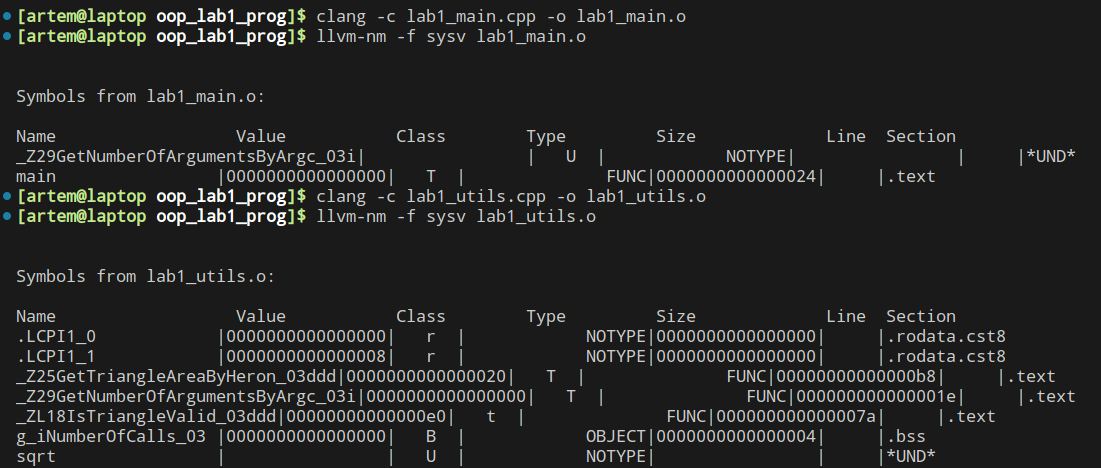
6. Перекомпілюйте файл ‘lab1\_lib.cpp’, вказавши прапорець -

DLAB1\_LIB\_BUILD\_AS\_SHARED\_LIB. Для цього виконайте команду, подану нижче.

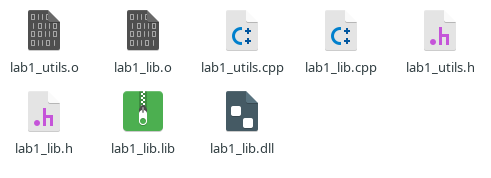
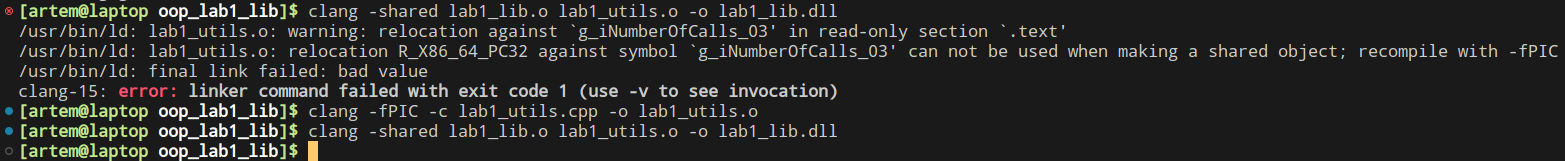
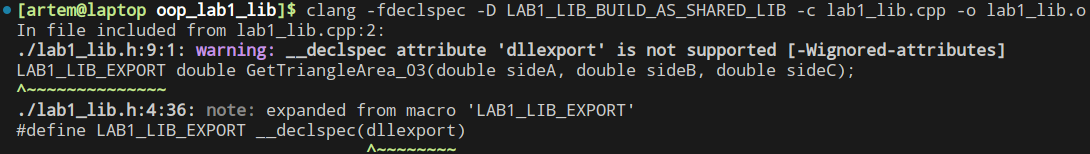
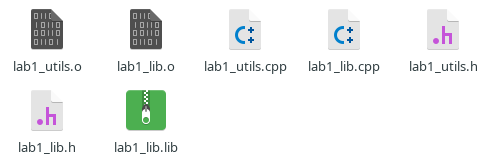
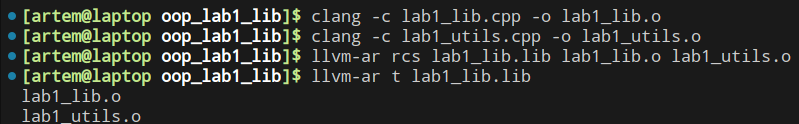
7. Скомпонуйте об’єктні файли ‘lab1\_lib.o’ та ‘lab1\_utils.o’ у динамічну бібліотеку виконавши

команду, подану нижче. Створиться файл ‘lab1\_lib.dll’.

**Завдання**



|  |  |  |
| --- | --- | --- |
|  | Розмір для збірки ‘Release’ (байт) | Розмір для збірки ‘Debug’ (байт) |
| lab1\_main.o | **2484** | **2944** |
| lab1\_utils.o | **3224** | **13456** |
| lab1.exe | **5232** | **22664** |



**Висновок**

Компіляція – це процес перетворення вихідного коду в об’єктний. Компілятор дає можливість проводити компіляцію з різними параметрами, наприклад з додатковою інформацією для дебагінгу або для збірки в динамічну бібліотеку.