Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи №1

**На тему:**

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

з дисципліни “Об’єктно-орієнтоване програмування”

**Лектор:**

доц. каф. ПЗ

Коротєєва Т.О.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

асис. каф. ПЗ

Дивак І.В.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

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

**Мета**: 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 відомості

1. **Одиниці Трансляції**

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++.

Перетворення тексту програми у фінальний файл (виконавчий чи бібліотеку) проходить у кілька етапів:

* компіляція окремих одиниць трансляції в об’єктні файли (.o/.obj);
* створення із об’єктних файлів статичних бібліотек (.a/.lib);
* компонування об’єктних фалів та статичних бібліотек у виконавчі файли чи динамічні бібліотеки (.exe/.so/.dll).

Нижче подано загальну схему процесу збірки C++ програми:



.c/.cpp/.cxx



.c/.cpp/.cxx



.o/.obj



.o/.obj



.c/.cpp/.cxx



.o/.obj



.c/.cpp/.cxx



.o/.obj



.c/.cpp/.cxx



.o/.obj



.a/.lib



.a/.lib



.exe/.so/.dll

Рис. 1. Схема збірки C++ програми. Сині стрілки – компіляція, оранжеві – створення статичних бібліотек, зелені – компонування.

1. **Типи оголошень та визначень:**

*Таблиця 1. Типи оголошень та визначень*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | **Оголошення** | **Визначення** |
| **Код** |  |  | int func(int x); | int func(int x) { ... } |
| **Дані** | Глобальні | Ініціалізовані | extern int x; | int x = 1; (в області файлу) |
| Неініціалізовані | extern int x; | int x; (в області файлу) |
| Локальні | Ініціалізовані | --- | int x = 1; (в області функції) |
| Неініціалізовані | --- | int x; (в області функції) |
| Динамічні |  | --- | int \* p = (int \*)malloc(sizeof(int)); |

1. **Вивід nm і значення:**

*Таблиця 2. Типи оголошень та визначень*

|  |
| --- |
| Symbols from sample.o:    Name Value Class Type Size Line Section    fn\_c |000000000000000f| T | FUNC |0000000000000059| |.text  func\_a | | U | NOTYPE | | |\*UND\*  func\_b |0000000000000000| t | FUNC |000000000000000f| |.text  g\_init\_x |0000000000000000| D | OBJECT |0000000000000004| |.data  g\_init\_y |0000000000000004| d | OBJECT |0000000000000004| |.data  g\_uninit\_x |0000000000000000| B | OBJECT |0000000000000004| |.bss  g\_uninit\_y |0000000000000004| b | OBJECT |0000000000000004| |.bss  g\_z | | U | NOTYPE | | |\*UND\* |

1. **поля Class може мати такі значення:**

* U – невизначені посилання. Це і є ті порожні значення, які генерує компілятор для оголошених, але не визначених змінних та функцій. В нашому прикладі їх два: func\_a та g\_z. На деяких системах nm також друкує колонку Section із значенням \*UND\* або UNDEF в такому випадку.
* t – код, локальний для цього файлу. Тобто функція була оголошена із ключовим словом static. Секція для цього класу - .text.
* T – код, до якого можна звертатись із інших файлів. Функція була оголошена без ключового слова static або із ключовим словом extern. Секція для цього класу - .text.
* d – ініціалізована глобальна змінна, локальна для цього файлу. Вона була оголошена із ключовим словом static. Секція для цього класу - .data.
* D - ініціалізована глобальна змінна, до якої можна звертатись із інших файлів. Вона була оголошена без ключового слова static або із ключовим словом extern. Секція для цього класу -

.data.

* b – неініціалізована глобальна змінна (або ініціалізована нулем), локальна для цього файлу. Вона була оголошена із ключовим словом static. Секція для цього класу - .bss.
* B або C – неініціалізована глобальна змінна (або ініціалізована нулем), до якої можна звертатись із інших файлів. Вона була оголошена без ключового слова static або із ключовим словом extern. Секція для цього класу - .bss або \*COM\*.

Завдання

## Частина 1. Компіляція. Компонування

1. Завантажити та встановити LLVM (див. Додаток 1)
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’

clang -c lab1\_main.cpp -o lab1\_main.o

1. Проглянути вміст об’єктного файлу ‘lab1\_main.o’ виконавши команду, подану нижче. Пояснити значення колонки Class для кожного символу. Рядок для символу ‘@feat.00’ можна пропустити.

llvm-nm -f sysv lab1\_main.o

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

clang -c lab1\_utils.cpp -o lab1\_utils.o

1. 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.

llvm-nm -f sysv lab1\_utils.o

1. Скомпонувати два об’єктних файли у виконавчий виконавши команду, подану нижче. Створиться файл ‘lab1.exe’

clang lab1\_main.o lab1\_utils.o -o lab1.exe

1. 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.

lab1.exe

echo %errorlevel%

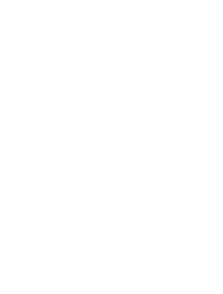
0

lab1.exe a b c

echo %errorlevel%

3

1. Побудувати діаграму структури програми, на якій відобразити об’єктні файли із символами та зв’язки між ними



lab1\_main.o

lab2\_utils.o



## 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\_rel.o’. Опція -O3 задає третій (найвищий) рівень оптимізації. Опція -flto вмикає оптимізацію під час компонування.

clang -c lab1\_main.cpp -O3 -flto -o lab1\_main\_rel.o

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

clang -c lab1\_utils.cpp -O3 -flto -o lab1\_utils\_rel.o

1. 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.

clang -fuse-ld=lld-link -flto lab1\_main\_rel.o lab1\_utils\_rel.o -o lab1\_rel.exe

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.

clang -c lab1\_main.cpp -O0 -g -o lab1\_main\_gdb.o

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

clang -c lab1\_utils.cpp -O0 -g -o lab1\_utils\_dbg.o

1. Скомпонувати два об’єктних файли у виконавчий, застосувавши команду, подану нижче. Створиться файл ‘lab1\_gdb.exe’.

clang -g lab1\_main\_dbg.o lab1\_utils\_dbg.o -o lab1\_dbg.exe

Складіть таблицю із розмірами об’єктних та виконавчих файлів для збірок ‘Release’ та ‘Debug’ як показано нижче:

*Таблиця 3. Розміри файлів для збірок ‘Release’ та ‘Debug’*

|  |  |  |
| --- | --- | --- |
|  | Розмір для збірки ‘Release’ (байт) | Розмір для збірки ‘Debug’ (байт) |
| lab1\_main.o | *візьміть розмір lab1\_main\_rel.o* | *візьміть розмір lab1\_main\_dbg.o* |
| lab1\_utils.o | *візьміть розмір lab1\_utils\_rel.o* | *візьміть розмір lab1\_utils\_dbg.o* |
| lab1.exe | *візьміть розмір lab1\_rel.exe* | *візьміть розмір lab1\_dbg.exe* |

Проаналізуйте та поясніть різницю у розмірах.

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’.

clang -c lab1\_lib.cpp -o lab1\_lib.o

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

clang -c lab1\_utils.cpp -o lab1\_utils.o

1. Об’єднайте об’єктні файли ‘lab1\_lib.o’ та ‘lab1\_utils.o’ у статичну бібліотеку виконавши команду, подану нижче. Створиться файл ‘lab1\_lib.lib’.

llvm-ar rcs lab1\_lib.lib lab1\_lib.o lab1\_utils.o

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

llvm-ar t lab1\_lib.lib

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

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

clang -DLAB1\_LIB\_BUILD\_AS\_SHARED\_LIB -c lab1\_lib.cpp -o lab1\_lib.o

1. Скомпонуйте об’єктні файли ‘lab1\_lib.o’ та ‘lab1\_utils.o’ у динамічну бібліотеку виконавши команду, подану нижче. Створиться файл ‘lab1\_lib.dll’.

clang -shared lab1\_lib.o lab1\_utils.o -o lab1\_lib.dll

Варіант №25

Завдання №1

1. Переходимо теки “oop\_lab1\_prog”.
2. Створюємо об’єктний файл “lab1\_main.o”.
3. Переглядаємо вміст файлу “lab1\_main.o”.
4. За таблицею можемо дізнатися, що “GetNumberOfArgumentsByArgc\_25” – оголошення функції, а “main” – визначення функції із зовнішнім зв’язуванням.

Див. Рис. 2:

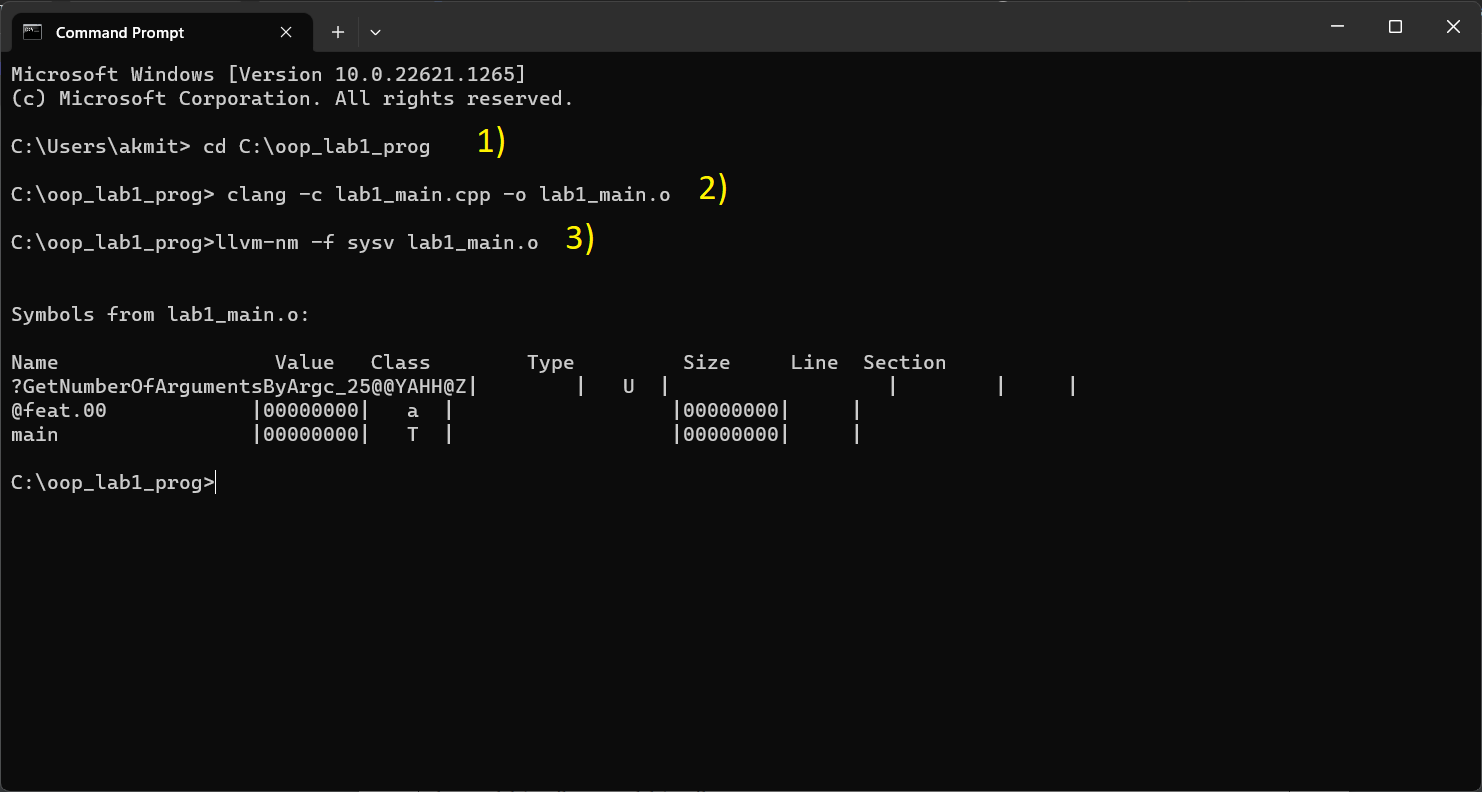


Рис. 2 створюємо об’єктний файл та переглядаємо його вміст

1. визначення функції із зовнішнім зв’язуванням “GetNumberOfArgumentsByArgc\_25”.
2. визначення функції із зовнішнім зв’язуванням “GetTriangleAreaByHeron\_25”.
3. визначення функції із внутрішнім зв’язуванням “IsTriangleValid\_25”.
4. Неініціалізована глобальна змінна із зовнішнім зв’язуванням “g\_iNumbersOfCalls\_25”.
5. Оголошення функції “sqrt”.

Див. Рис. 3:



Рис. 3 Створюємо об’єктний файл “lab1\_utils.o” і переглядаємо його вміст:

1. Компонуємо об’єктні файли у .exe
2. Передаємо 0 параметрів – програма повертає 0
3. Передаємо 3 параметри – програма повертає 3

Див. Рис. 4:

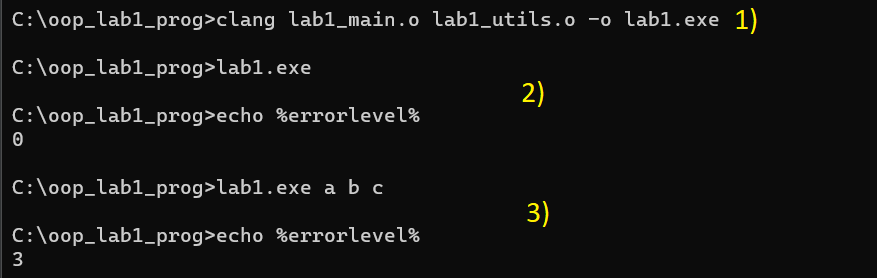


Рис. 4 Перевіряємо роботу виконавчого файлу “lab.exe”

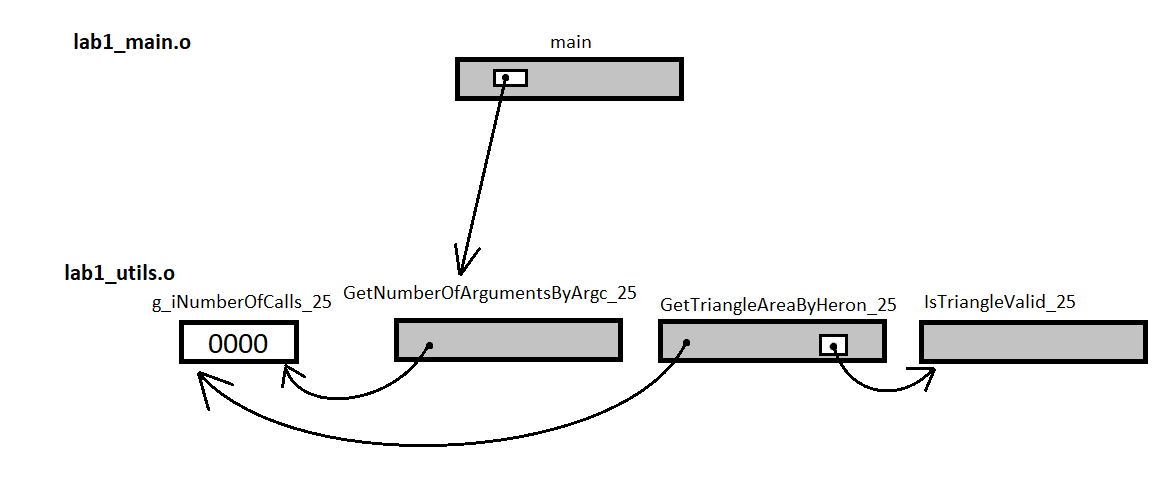


Рис. 5. Діаграма структури програми

Завдвння№2

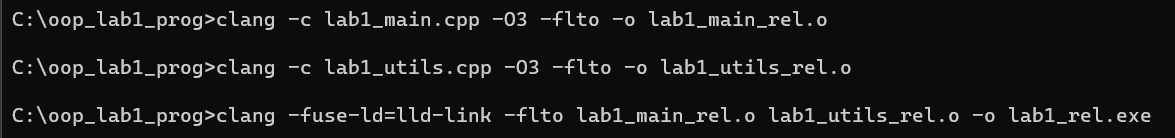


Рис. 6 Створення збірки для випуску

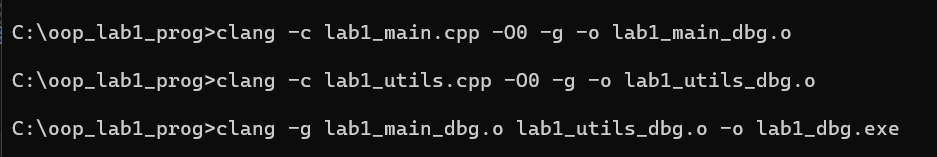


Рис. 7 Створення збірки для відлагодження

*Таблиця 4. Розміри файлів для збірок ‘Release’ та ‘Debug’*

|  |  |  |
| --- | --- | --- |
|  | Розмір для збірки ‘Release’ (байт) | Розмір для збірки ‘Debug’ (байт) |
| lab1\_main.o | *2,500* | *3,320* |
| lab1\_utils.o | *3,584* | *5,649* |
| lab1.exe | *95,744* | *566,272* |

Розмір об’єктних файлів відрізняється через різний рівень оптимізації, що використовувався у процесі компіляції.

Розмір .exe файлів відрізняється по-перше через різну вагу об’єктних файлів, проте найвагомішу частку займають підключені інструменти для відлагодження.

Завдання №3

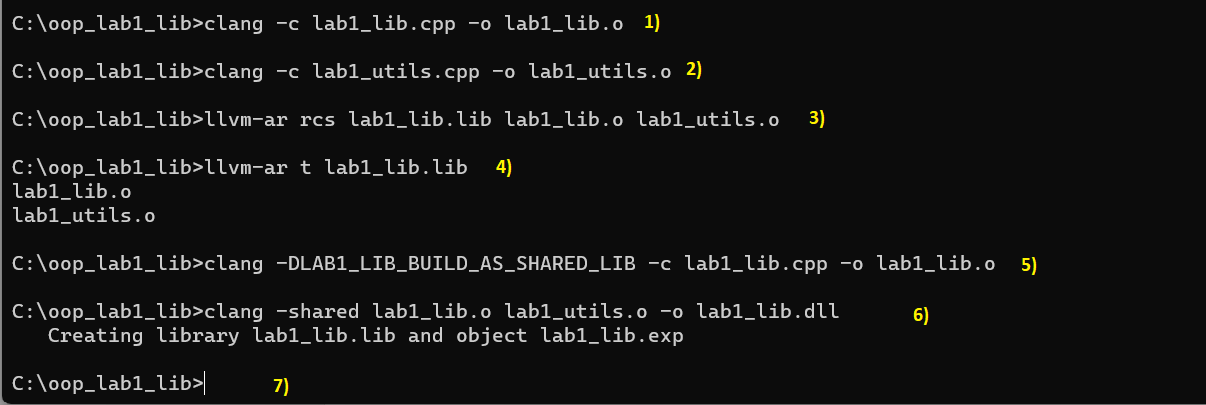


Рис. 8 Створення статичної та динамічної бібліотек

1. Компілюємо “lab1\_lib.cpp”
2. Компілюємо “lab1\_utils.cpp”
3. Об’єднуємо утворені об’єктні файли у статичну бібліотеку “lab1\_lib.lib”
4. Переглядаємо вміст бібліотеки “lab1\_lib.lib”
5. Перекомпільовуємо “lab1\_lib.cpp” в об’єктний файл, придатний до компонування у динамічну бібліотеку.
6. Створюємо динамічну бібліотеку “lab\_lib.dll” на основі файлів “lab1\_lib.o” та “lab1\_utils.o”.

Висновок

Під час виконання лабораторної роботи, я засвоїв принципи створення програм написаних мовою С++, навчився перетворювати текст програми на виконавчий файл за допомогою командного рядка. Навчився задавати опції компіляції, створювати різні типи збірок - виконавчі файли, статичні та динамічні бібліотеки.