**Міністерство освіти і науки України**

**Львівський національний університет імені Івана Франка**

**Факультет прикладної математики та інформатики**

**ЗВІТ**

Виконали:

Студенти групи ПМп-22

Томочко Віталій

Лесів Олег

Львів 2017

***Постановка завдання 1***

Написати бібліотеку, що дає змогу працювати з такими cтруктурами даних :

* однозв'язними списками
* двозв'язними списками
* стеками
* чергами

передбачити створення, додавання/вилучення елемента в кінець, на початок,

на конкретну позицію, дописування одного до іншого, обертання порядку елементів, сортування,

пошук. Для деяких типів, наприклад, стека, не всі ці операції можливі, тому необхідно обдумати,

що має зміст реалізовувати, а що - ні.

***Шляхи реалізації***

Для реалізації програми, необхідно знати, що являють собою такі структури даних як списки , стеки і черги.

Списки зазвичай реалізуються або у вигляді зв'язаних списків (або однобічно, або двобічно) або у вигляді масивів, як правило, змінної довжини або динамічних масивів.

Для реалізації однобічного та двобічного списків ми створили окремі класи ,які знаходяться у файлах List.h , DoubleList.h ,де були описані поля та методи а також структуру де зберігалась інформація (вказівник на наступний/попередній елементи, дані елементів).

Черга-це динамічна структура даних, що працює за принципом «перший прийшов — перший пішов».Черга може бути реалізована за допомогою масива , в якому зберігаються дані та двох додаткових змінних *head* та *tail*, в яких зберігаються індекси відповідно "голови" та "хвоста" черги.

Стек — це різновид лінійного списку, структура даних, яка працює за принципом (дисципліною) «останній прийшов — перший пішов»

Стек може бути реалізований за допомогою масиву та додаткової змінної: Для зберігання елементів стеку резервується масив *S[1..n]* певного розміру та додаткова змінна *top[S]*, яка буде зберігати індекс верхівки стеку.

***Перелік операції***

В програмі описані такі операції із стеками:

## Основні операції з чергою:

## ***enqueue***— "поставити в чергу". Операція додавання елемента в "хвіст" черги. При цьому довжина черги збільшується на одиницю.

## ***dequeue*** — "отримання з черги". Операція, яка повертає елемент з голови та видаляє його з черги, таким чином встановлюючи голову на наступний за видаленим елемент та зменшуючи довжину на одиницю.

## Основні операції з однозв'язними та двозв'язними списками:

конструктор для створення порожнього списку;

**pushBack** операція для додавання об'єкта у список (в кінець);

**insert** операція для додавання об'єкта в список на ***і*** позицію;

**pop** операція для видалення зі списку останнього елемента;

**size** операція ,яка вертає розмір списку;

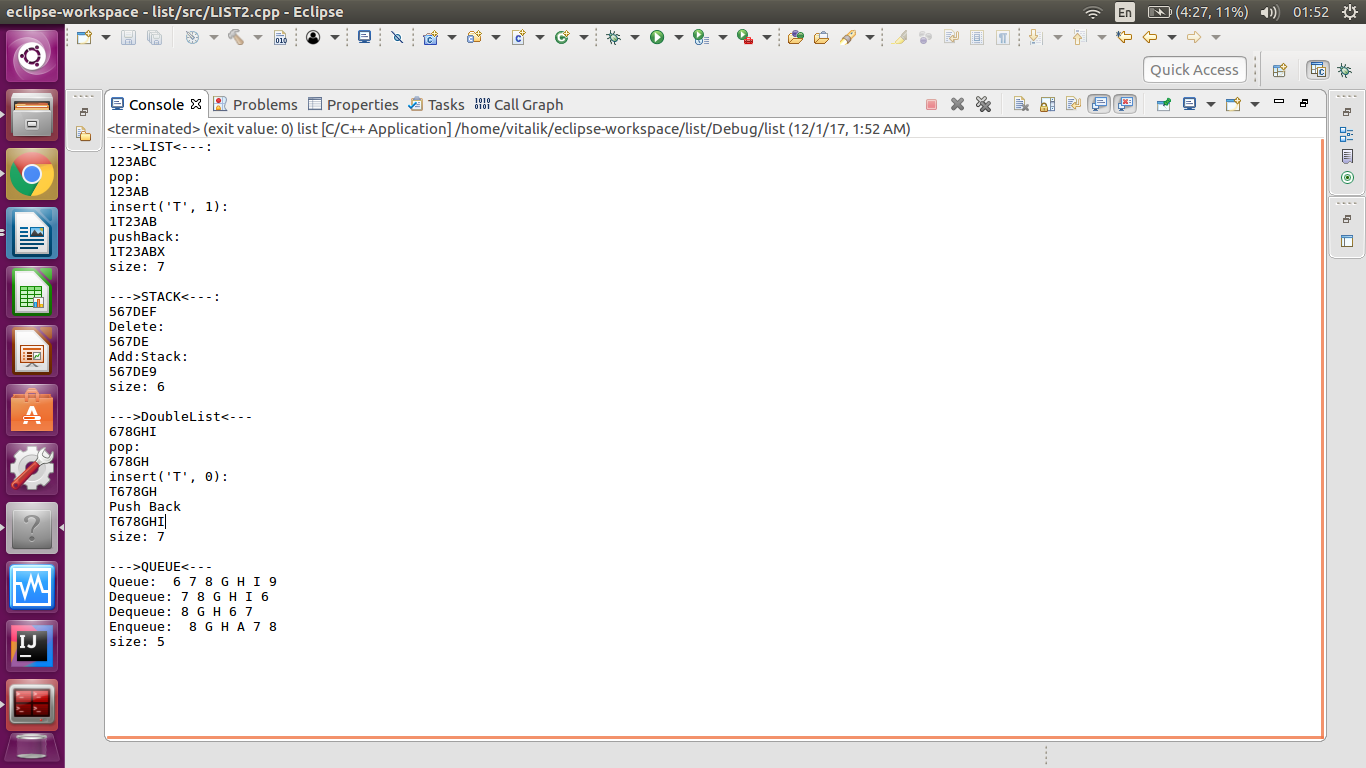
**print** операція для друку списку;

## Основні операції з стеком:

**push** («заштовхнути елемент»): елемент додається в стек та розміщується в його верхівці. Розмір стеку збільшується на одиницю.

**pop** («виштовхнути елемент»): отримує елемент з верхівки стеку. При цьому він видаляється зі стеку і його місце в верхівці стеку займає наступний за ним.

***Презентація результатів***

******

***Список учасників та їх внески***

Томочко Віталій реалізував роботу із чергами та однозв’язними списками.

Лесів Олег реалізував роботу із стеками та двозв’язними списками.

***Постановка завдання 2***

Написати два будь-яких алгоритми сортування масиву;

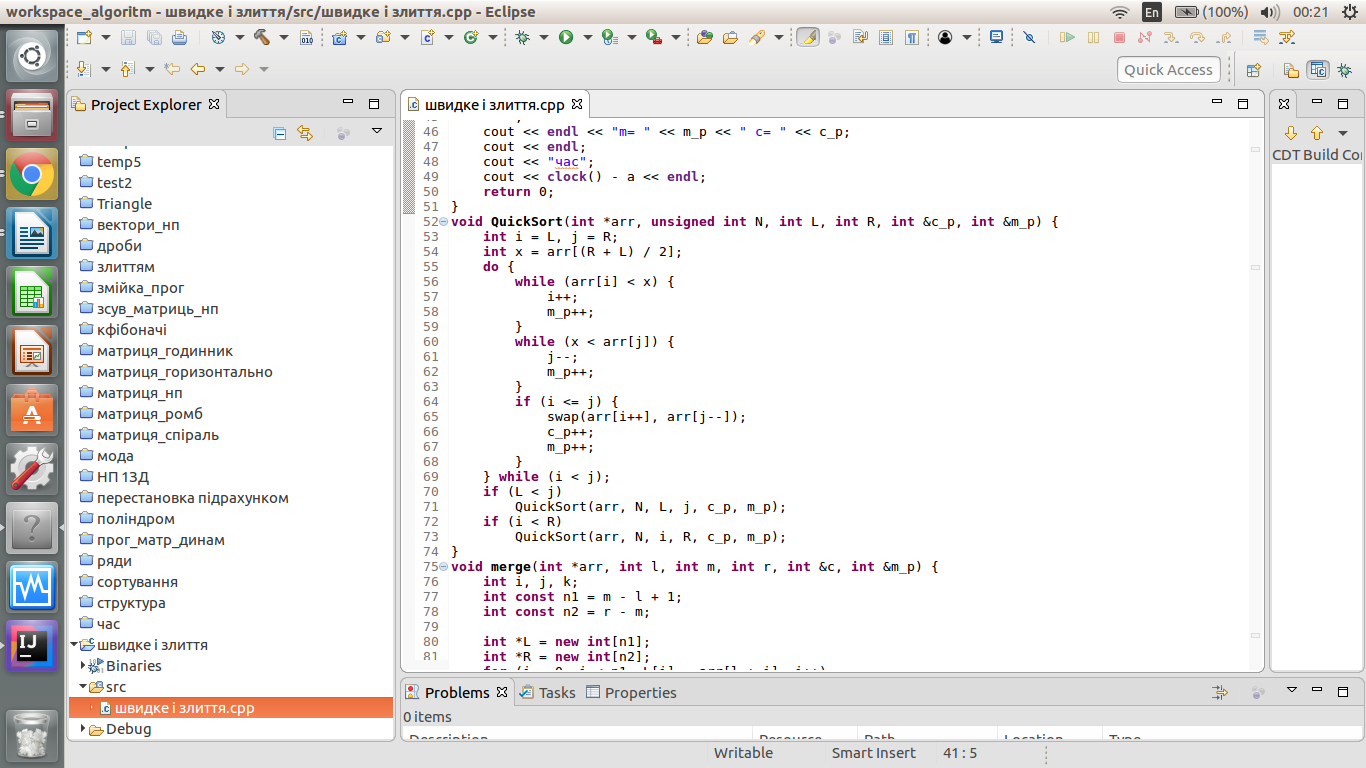
Ми реалізували такі алгоритми сортування:

* швидке сортування
* сортування злиттям

***Шляхи реалізації***

*Реалізація швидкого сортування:*

Розглянемо масив Array, який необхідно відсортувати. Для даного алгоритму, нам знадобляться наші звичайні ітератори iter і jter, N – кількість елементів масиву, L і R – ліва і права межі сортування відповідно, . Реалізація даного алгоритму на C++ виглядає наступним чином:



Отже при першому виклику функції QuickSort, ітератори iter і jter ініціалізовуються у значення лівої і правої меж відповідно, тобто рівні індексам початку і кінця масиву. Далі ми обераємо середній з усіх елементів масиву, який ми і будемо порівнювати з усіма іншими елементами. При входженні у зовнішній цикл, ми одночасно входимо і в перший внутрішній цикл, який зупиняється при найпершому більшому елементі від обраного серединного (лінійний пошук з інваріантою). Після виходу з даного циклу, ітератор iter, буде містити в собі індекс найпершого більшого елементу від серединного починаючи зліва. Далі ми входимо у другий внутрішній цикл (також лінійний пошук з інваріантою), який шукає більший елемент від серединного починаючи з правої межі, і після закінчення виконання циклу в ітераторі jter буде міститись індекс даного елементу. Далі ми натрапляємо на оператор вибору, який перевіряє чи значення ітератора iter менше від значення ітератора jter. Основна ціль виконання перевірки даної умови полягає у тому, щоб перевірити, чи вони не розминулися. Якщо вони все таки розминулись – не виконується обмін значеннями і ітерації зовнішнього циклу зупиняються. Після чого функція може сама себе викликати, якщо сортування ще не закінчено. Сортування закінчується, якщо ліва межа (тобто L) більша за значення ітератора jter і права межа менша за ітератор iter.

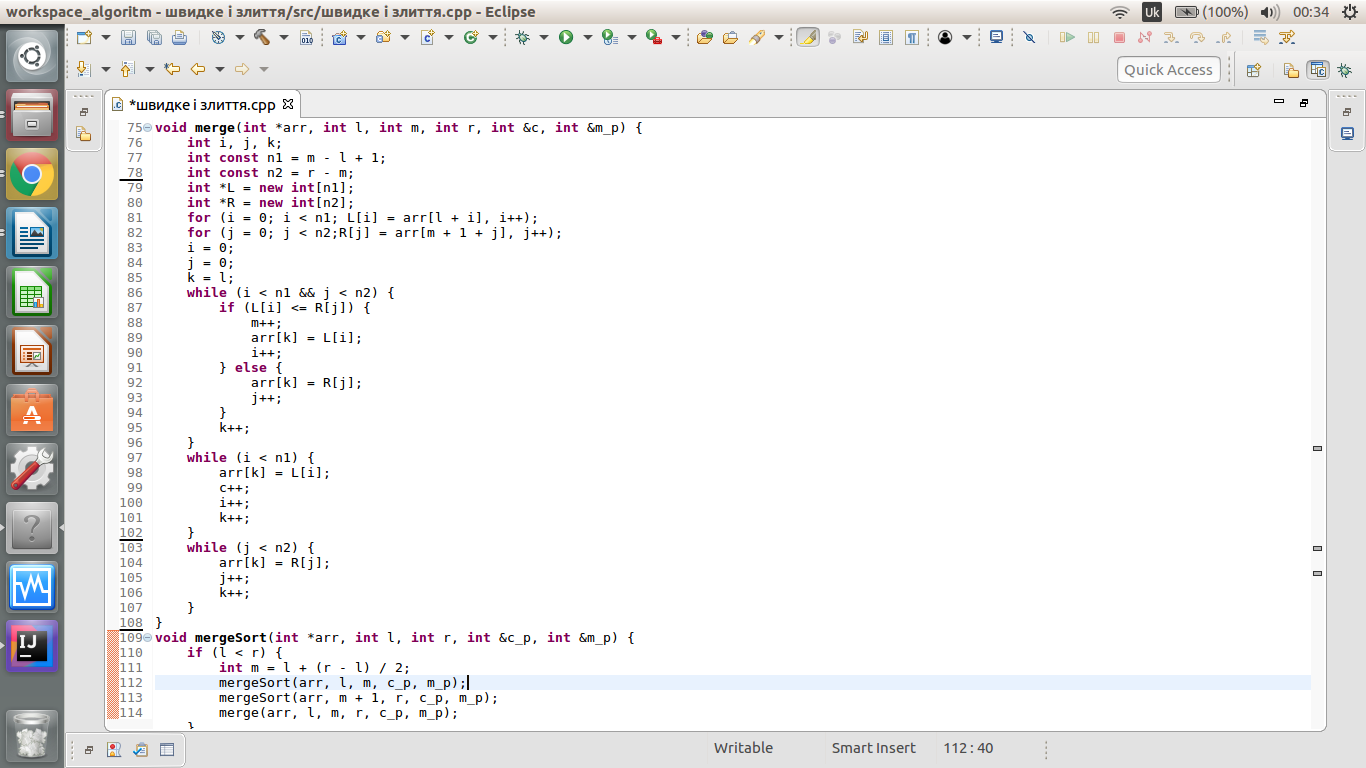
Тобто алгоритм просто ділить масив на половину, і перекидає усі менші елементи вліво від центрального, а усі більші – вправо. І це відбувається доти, доки межі не зіллються.

*Реалізація сортування злиттям:*

Цей алгоритм як і швидке сортування керується принципом „розділяй та владарюй“, тобто він ділить множину елементів, які потрібно відсортувати, на підмножини які сортуються цим же алгоритмом.Як і швидке сортування, цей алгоритм базується на рекурсії.

Саму роботу алгоритму можна розбити на два кроки:

* 1. Поділити масив на дві рівні частини і застосувати сортування злиттям до кожної з них.
  2. Злити два відсортовані масиви з попереднього кроку у один таким чином, щоб він був відсортований.



Тепер давайте детальніше розглянемо кожен крок детальніше:

* 1. Масив ділиться елементарним чином, а саме діленням націло пополам. Тобто якщо у нас парна кількість елементів (2n), то ми отримаємо два одинакові масиви (перший: 2n div 2 = n, а другий: 2n-n=n (загальна довжина мінус довжина першого)), якщо ж непарна (2n+1), то один з масивів буде на один елемент довший за інший (перший: (2n+1) div 2 = n, другий: (2n+1) - n = n+1). Потім застосовуємо функцію сортування на кожному з двох масивів.
  2. В результаті попереднього кроку ми отримуємо два відсортовані масиви, які потрібно злити у один відсортований масив.

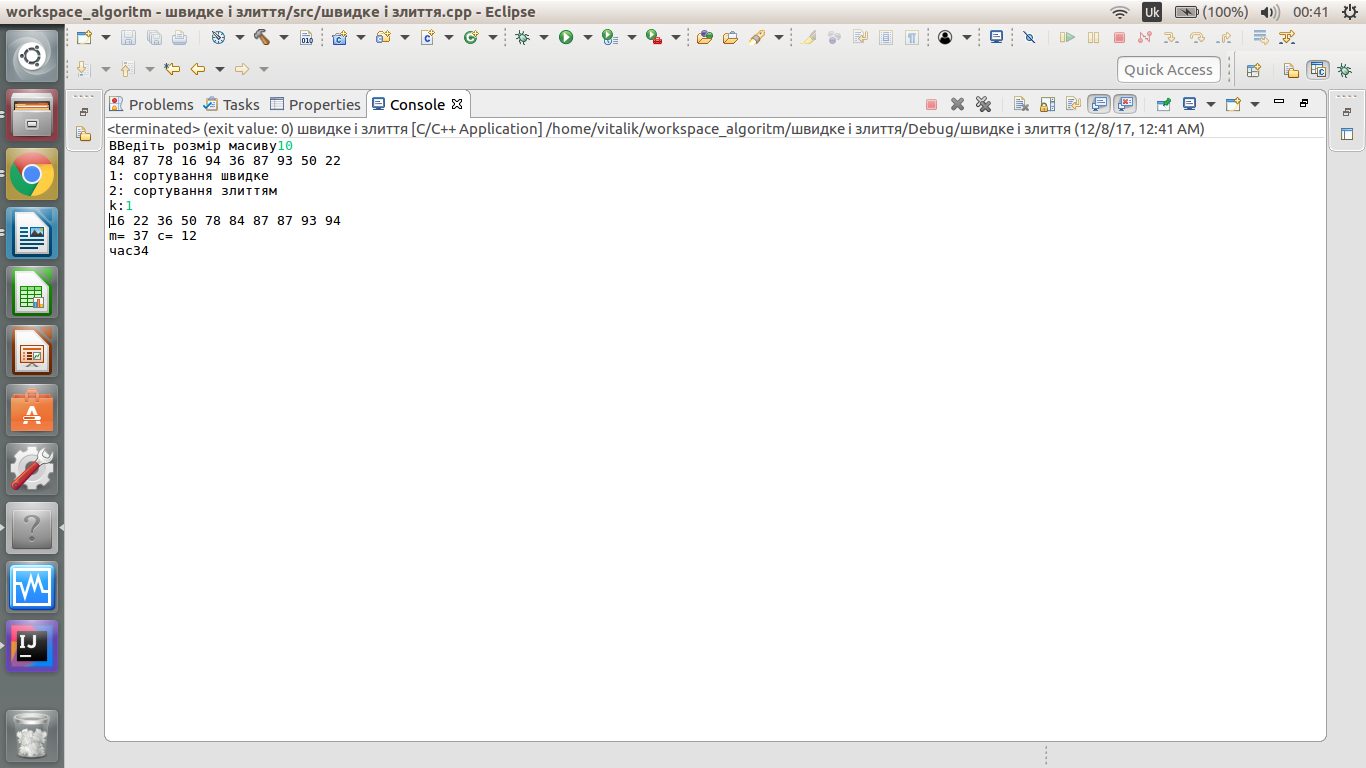
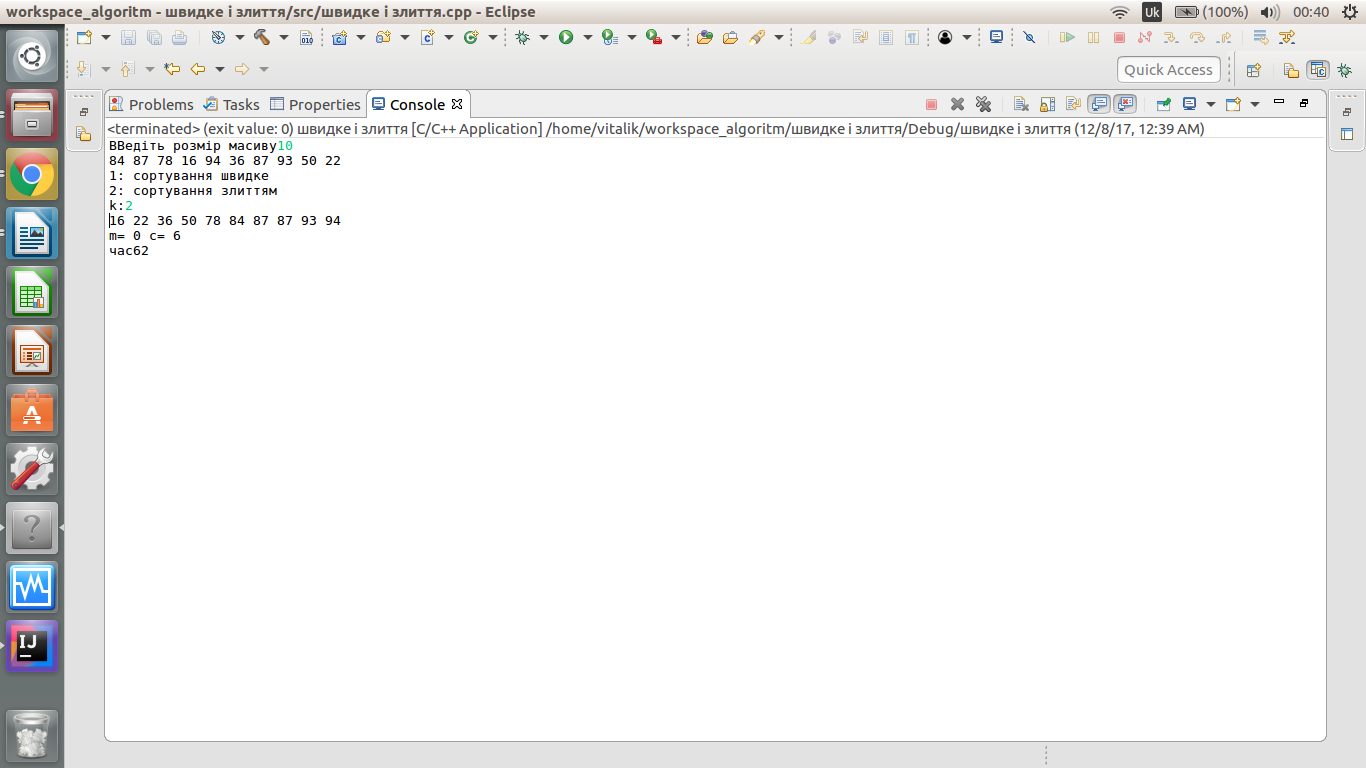
***Перелік операції***

В програмі описані такі операції :

## 1-сорутвання масиву за допомогою алгоритму швидкого сортування

## 2-сортування масиву за допомогою алгоритму сортування злиттям

***Презентація результатів***

******

***Список учасників та їх внески***

Томочко Віталій реалізував швидке сортування.

Лесів Олег реалізував сортування злиттям.

***Постановка завдання 3***

Спроектувати та реалізувати програму для менеджера авіакомпанії.з можливістю внесення в систему літака ,аеропрортів,рейсів,тощо.

***Шляхи реалізації***

Для реалізації цієї програми ми створили окремі класи ,які знаходяться у файлах Plane.h,Place.h,Airport.h,Passanger.h,Flight.h,де були описані поля та методи а також структуру де зберігалась інформація (інформація про літак ,інформація про аеропорт,тощо).

***Перелік операції***

В програмі описані такі операції із стеками:

create\_passenger <name> <surname>-створити пасажира

create\_airport <name> <city> створити аеропорт

create\_flight <place\_of\_departure> <place\_of\_landing> <departure\_time> <Arrival\_time> <airplane\_number> <number\_of\_places> створити рейс

create\_plane <airplane\_number> <number\_of\_places> створити літак

info\_plane <airplane\_number> інформація про літак

info\_flight <place of departure> інформація про рейс

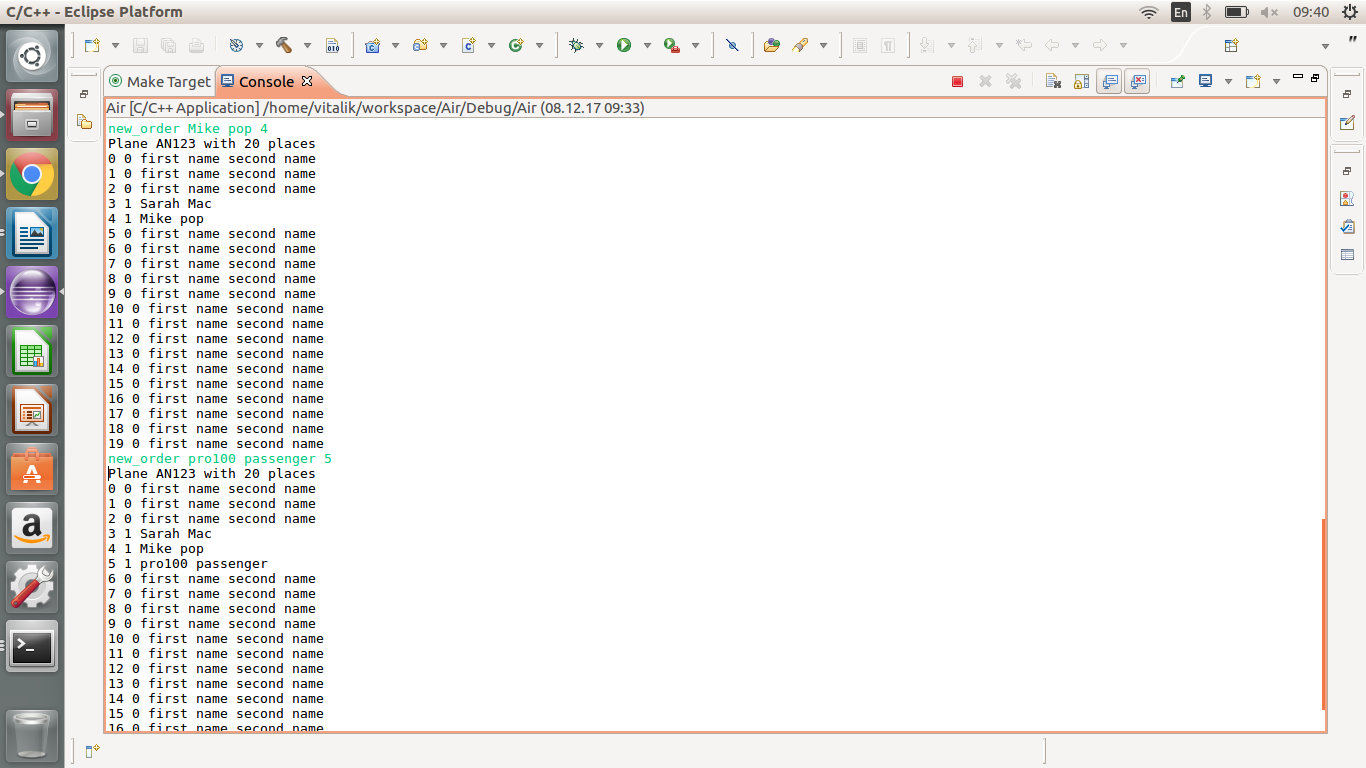
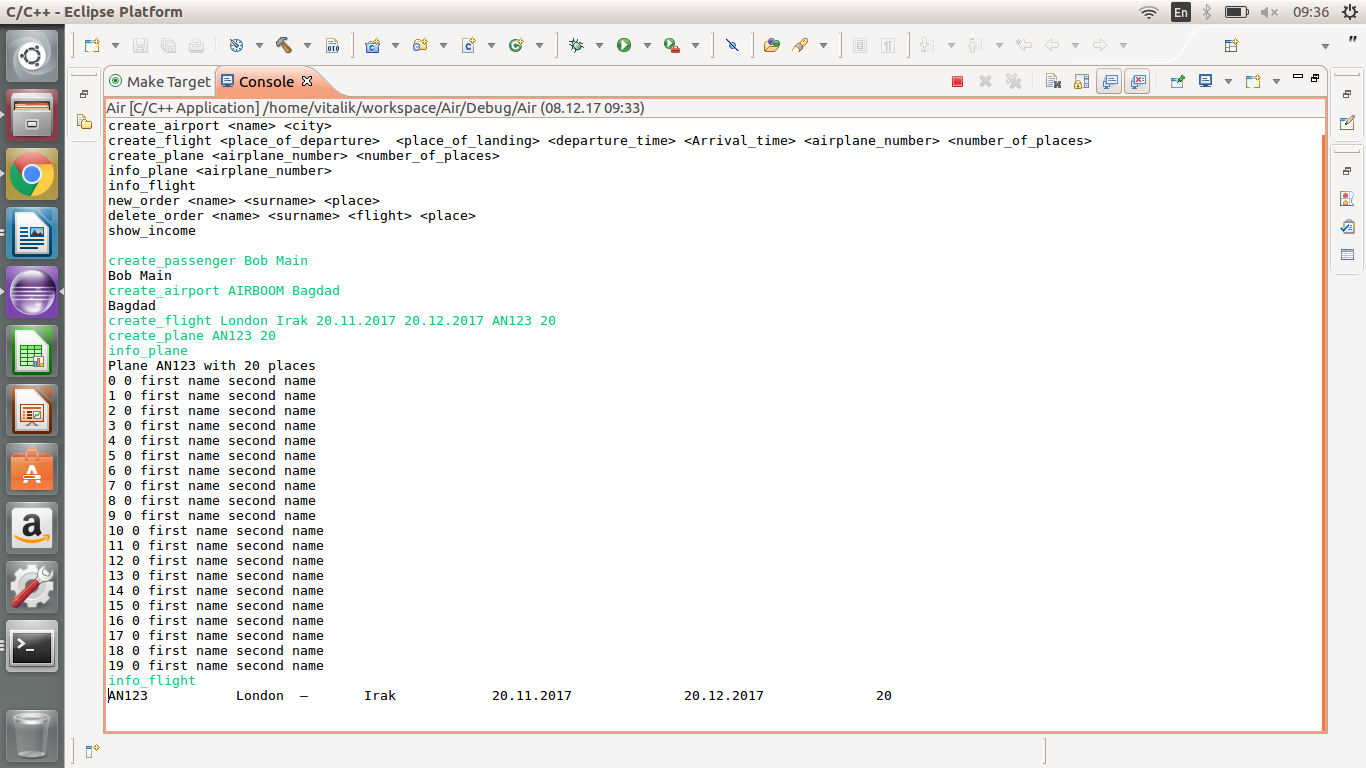
new\_order <name> <surname> <flight> <place> бронювання місць

delete\_order <name> <surname> <flight> <place> скасувати замолвення

show\_income вивести дохід

help список доступних команд

***Презентація результатів***



***Список учасників та їх внески***

Томочко Віталій реалізував Place.h Place.cpp,Passenger.h,passenger.cpp,Flight.h.Flight.cpp.

Лесів Олег реалізував Airport.h,Airport.cpp,Plane.h.Plane.cpp.

***Постановка завдання 4***

Створити програму для створення та опрацювання плейлистів.

***Шляхи реалізації***

Для виконання завдання будо створено клас Playlist та його реалізація на файлах playlist.cpp та playlist.h. Окрім того опрацювання плейлистів реалізовано методом перевантажених операторів +, +=, -, -=. А також методів опрацювання консольним вікном cd, ls.

***Перелік операції***

В програмі описані такі операції:

**cd:** змінити директорію

**ls:** вивести список файлів у даній директорії

**оperator+=(string \_file):** додає до плейлиста якийсь один файл що задається повною директорією.

**operator +(string \_directory):** додає до плейлиста всі файли у вибраній директорії

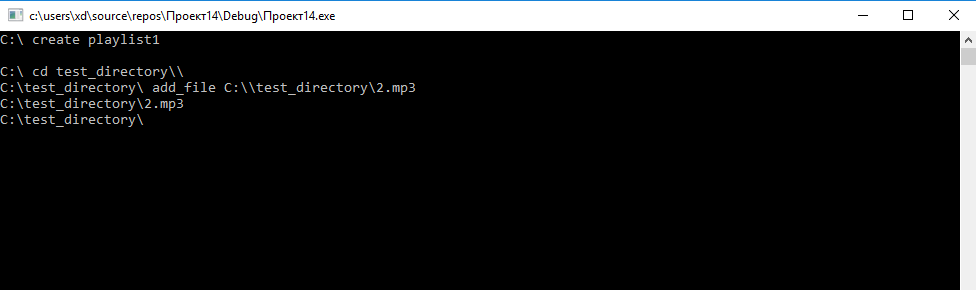
**operator-=(string \_file):** віднімає від плейлиста один файл що.

**operator -(string \_directory):** віднімає від плейлиста всі файли у вибраній директорії.

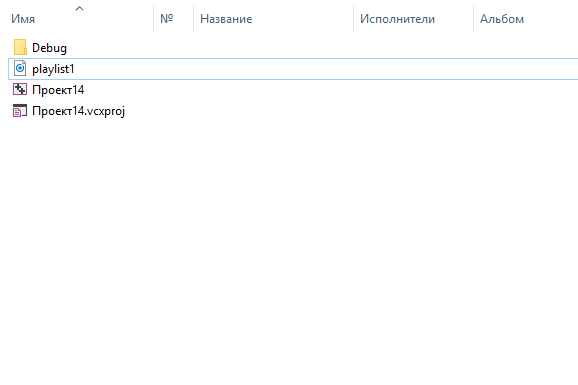
**outp(string \_name):** записує плейлист у файл розширення m3u.

***Презентація результатів:***

Попробуємо створити плейлист та добавити до нього файл.



В директорії проекту був створений наш файл.



***Список учасників та їх внески***

Томочко Віталій реалізував +=,+.

Лесів Олег реалізував -=,-.

***Постановка завдання 5***

Створити програму для закодування та декодування повідомлення в зображенні формату BMP.

***Шляхи реалізації***

Програму було реалізовано в межах одного меню з вибором операцій кодування чи декодування.

***Перелік операції***

В програмі описані такі операції:

**Encode:** з використанням одної з трьох змінних структури rgb\_l ми загодовуємо повідомлення

**Decode:** на жаль з ряду причин нам не вдалось реалізувати дану операцію.

***Презентація результатів:***

***Список учасників та їх внески***

Томочко Віталій реалізував structbmp.h , decode

Лесів Олег реалізував encode , створив меню та .

**Постановка завдання 6:**

Створити програмний продукт для роботи з двовимірними многокутниками. Проектування включає опис класів, які необхідні для імплементації необхідної функціональності (очевидно, що, окрім класу "многокутник", як мінімум, необхідно буде ввести клас "точка").

Функціональність:

* Додавання многокутників: результат - новий многокутник ;
* Віднімання многокутників: результат - новий многокутник ;
* Клонування многокутника: результат - новий об'єкт многокутника, що має абсолютно такі ж параметри '
* Масштабування многокутників: результат - "розтягнутий" або "стиснений" многокутник відносно центру мас;
* Масштабування многокутників з клонуванням: результат - "розтягнутий" або "стиснений" многокутник відносно центру мас ;
* Перенесення многокутника: результат - зміщений на певні величини по X та Y многокутник ;
* Перенесення з клонуванням многокутника: результат - зміщений на певні величини по X та Y многокутник .

**Хід виконання:**

Першочергово було створено два класи Point, Poly та стркутура Node. З часом, згідно до вимоги задачі про створення бібліотеки, все було закинуто в один хедер для створення namespace.

**Перелік команд:**

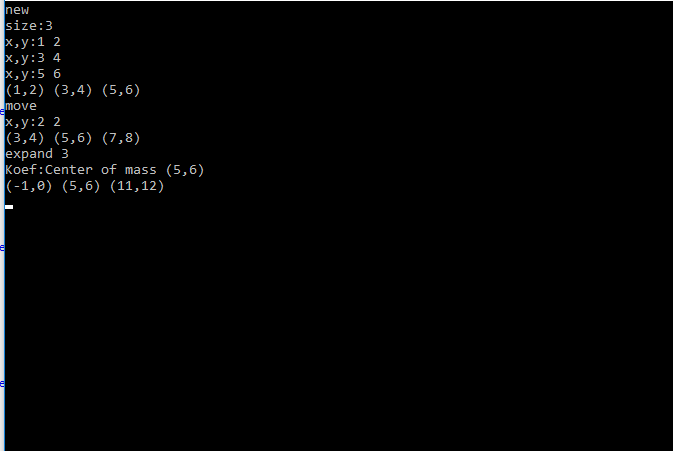
**"** new" - створення многокутника;

"move" - зміщення по X і Y;

"expand" - розширення;

"draw" - відображення на координатній площині.

**Презентація результатів:**



**Список учасників, та їх внески для реалізації:**

Лесів Олег - класи Point, меню,+,+=.

Томочко Віталій – клас Polygon \*,\*=.