**Інтелектулаьний шаблон**

**Заяц Р. В**

Отже це завдання потребувало трішки часу і вмінь для того щоб виконати його правильно. Перша ідея реалізація виникла під час прочитання завдання. На одній з лекцій нам було показано функцію **typeid** з бібліотеки **RTTI.** Після написання невеликого шаблону який підтримує лише вектор, і подальшої перевірки роботи, я прийшов до висновку, що ця ідея марна, адже я не зміг придумати як дізнатись тип даних який зберігає контейнер, а це важливо, оскільки typeid.name() видає щось накшталт “class std::set<int,struct std::less<int>,class std::allocator<int> >”

Другою ідеєю стало винесення атрибутів контейнерів в окремі шаблонні класи і перевірка їх в основній шаблонній функції. Спочатку я оголосив пустий шаблонний клас

template<typename T>

class Unique\_Cont

{

};

Це було зроблено з метою отримати змогу створювати майбутні класи в які я зможу передавати контейнери. Потім я перезавантажив клас **Unique\_Cont** з спеціалізацією для вектора

template<typename T>

class Unique\_Cont<vector<T>>

{

public:

inline static const string view = "\*\*\* ВЕКТОР \*\*\*";

inline static const string start = "[";

inline static const string end = "]";

inline static const string empty = "порожній";

};

Оголосив новий шаблон, щоб додатково передавати в **Unique\_Cont<тут контейнер<T(тип даних, що зберігає контейнер)>>.** Додав в публічну частину поля які зберігають рядок, який характеризує атрибути контейнера **гасло і обрамлення.** Поля зробив константними, адже вони незмінні static використав для того, щоб після першого проходження данні для цього вектора зберігались для можливості повторного використання готових полів, а не повторного їх створення. А також зробив поля inline, бо у мене в коді спеціалізація для декількох контейнерів і поля повторюють свої назви(Проект не компілюється без **inline),** а inline запобігає цьому.

Написав саму функцію друку.

template<typename Container>

void printCont(const Container& C, ostream& out)

{

Unique\_Cont<Container> temp;

out << temp.view << '\n';

if(!C.empty())

for (auto it = C.begin(); it != C.end(); ++it)

{

out << temp.start << \*it << temp.end << ' ';

}

else

{

out << temp.start << temp.empty << temp.end;

}

}

Шаблон приймає тип контейнера з бібліотеки STL який підтримує наші вимоги(**Контейнер вміє надавати ітератори на свій вміст за допомогою методів begin(), end()).** Функція отримує адресу на конкретний контейнер, який буде друкувати і адресу на потік в який друкувати. Створюється екземпляр шаблонного спеціаліованого класу, для того, щоб використовувати його поля в друці, і якщо контейнер не порожній то ми ітератором цього контейнера проходимось по ньому і друкуємо кожен елемент в обрамленні(temp.start, temp.end).

Перевіривши роботу, я впевнився що все працює і правильно виконує завдання згідно поставлених умов. Додав більше спеціалізацій для різних контейнерів.

Ось їх вигляд:

template<typename T>

class Unique\_Cont<list<T>>

{

public:

inline static const string view = "\*\*\* Двозв'язний Список \*\*\*";

inline static const string start = "<";

inline static const string end = ">";

inline static const string empty = "порожній";

};

template<typename T>

class Unique\_Cont<set<T>>

{

public:

inline static const string view = "\*\*\* МНОЖИНА \*\*\*";

inline static const string start = "{";

inline static const string end = "}";

inline static const string empty = "порожній";

};

Для остаточної перевірки додав гугл тести (Вкладено файлом main.cpp), там явно видно що Підхід 1 не є робочим, а Підхід 2 працює без помилок.

P.s Після написання тестів помітив що поле Empty можна винести в загальний доступ або в перший шаблон класу, адже воно одинакове для всіх контейнерів. Також в методі друку не обовязково створювати екзепляр достатньо скористатись полями через простір імен Unique\_Cont::view і т.д, адже поля статичні і до них можна звертатись напряму