ЛАБОРАТОРНА РОБОТА №6

Назва роботи: реактивне програмування.

Мета роботи: ознайомитися з парадигмою реактивного програмування.

1. Парадигма реактивного програмування

*(детальніша інформація про **реактивне програмування** в матеріалах лекційних занять)*

Реактивне програмування (РП) — це парадигма програмування, побудована на потоках даних і розповсюдженні змін. Це означає, що у мовах програмування має бути можливість легко виразити статичні чи динамічні потоки даних, а реалізована модель виконання буде автоматично розсилати зміни через потік даних.

Наприклад, в імперативному програмуванні вираз, a=b+c означає, що a отримує результат виконання b+c безпосередньо під час обчислення виразу, і потім, якщо значення b і c міняться, це не впливатиме на вже обчислене значення a. Водночас, в реактивному програмуванні значення a буде автоматично оновлено за новими значеннями, що є протилежністю до функційного програмування. Сучасна програма електронної таблиці в деякій мірі є прикладом реактивного програмування. Комірки електронної таблиці можуть мати значення, або формули, приміром «=B1+C1», що обчислюються за значеннями інших комірок. Коли б не змінилося значення іншої комірки, значення формули оновлюється автоматично. Інший приклад — це мова опису апаратних засобів, приміром VHDL або Verilog. У цьому випадку, реактивне програмування дозволяє моделювати зміни по мірі їх розповсюдження по електричному ланцюгу.

Реактивне програмування спочатку пропонувалося як засіб простого створення інтерфейсів користувача, анімацій у системах реального часу, але стало загальною парадигмою програмування. Наприклад, у архітектурі модель-вид-контролер, реактивне програмування дозволяє змінам у моделі автоматично відображатися у виді, і навпаки.

Реактивні мови програмування можуть різнитись від дуже явних, де потоки даних встановлюються через використання стрілок, до неявних, де потоки даних є похідними від мовних конструкцій, як у імперативному або функційному програмуванні. Наприклад, в функціональному реактивному програмуванні (FRP) виклик функції може неявно спричинити побудову вузла в графі потоку даних. Реактивні бібліотеки програмування для динамічних мов(таких як «Cells» у Ліспі або «Trellis» для Python'у), можуть будувати граф залежностей через аналіз значень, зчитуваних під час виконання певної функції.

Іноді термін «реактивне програмування» відноситься до архітектурного рівня розробки програм, де окремі вузли в графі потоку даних є звичайними програмами, які взаємодіють одна з одною.

Статика реактивного програмування. Реактивне програмування може бути чисто статичним, де потоки даних встановлюються статично, або бути динамічним, де потоки даних можуть змінюватися під час виконання програми.

Використання умовних переходів в графі потоку даних може змусити статичний граф потоку даних виглядати як динамічний. Проте, чисто динамічне реактивне програмування може використовувати імперативне програмування для реконструкції графу потоку даних.

Реактивне програмування вищого порядку. Реактивне програмування можна назвати програмуванням вищого порядку, якщо воно підтримує ідею про те, що потоки даних можуть бути використані для побудови інших потоків даних. Тобто, результуюче значення з потоку даних є інший потік даних.

Диференціація потоку даних. В ідеальному випадку всі зміни даних поширюються миттєво, але це не може бути забезпечено на практиці. Замість цього може бути потрібно надати різні пріоритети обчислення різним частинам графу потоку даних. Це можна назвати диференційованим реактивним програмуванням. Наприклад, в текстовому процесорі маркування орфографічних помилок не обов'язково має бути повністю синхронним з вставкою символів. Тут диференційоване реактивне програмування потенційно може бути використане для перевірки орфографії з нижчим пріоритетом, що дозволяє перевірці бути відкладеною, зберігаючи при цьому миттєвість інших потоків даних.

2. Приклад програми.

У *лістингу 2.1* показано обчислення виразу r = C2*a + C1*b + C0 для парадигми реактивного програмування з використанням бібліотеки ***RxCpp***. Для компіляції потрібно помістити папку з кодом бібліотеки у ту ж папку, що містить файл з кодом з *лістингу 2.1*.

Також було запаковано файли бібліотеки разом з наведеним прикладом з лістингу 2.1 у один файл та розміщено за посиланням: https://github.com/KozakNazar/srcacmlab6/blob/master/acmlab6__with_rxcpp_src.cpp . В цьому випадку приклад коду не вимагає жодних інших файлів бібліотеки ***RxCpp***, оскільки ця бібліотека повністю міститься в цьому єдиному файлі. Сам код прикладу розміщено на початку цього файлу.

! Для використання наведеного прикладу програми в єдиному файлі https://github.com/KozakNazar/srcacmlab6/blob/master/acmlab6__with_rxcpp_src.cpp **, з запакованою в ньому бібліотекою *RxCpp,* файл має мати назву:**

"acmlab6__with_rxcpp_src.cpp"

*Це пов'язано з тим, що програма використовує оголошення бібліотеки **RxCpp** до їх декларування, а тому має включати файл, в якому містить себе:* `#include "acmlab6__with_rxcpp_src.cpp"` .

Лістинг 2.1.

```
#include "stdafx.h" //

#define WORK_SPACE int a = 0, b = 0
#define EXPRESSION ( C2*a + C1*b + C0 )
#define RUN_FOR_UPDATE EXPRESSION
#define C2 4
#define C1 4
#define C0 4

#include "rxcpp/rx.hpp"
namespace rx = rxcpp;
namespace rxsub = rxcpp::subjects;
namespace rxu = rxcpp::util;
#include <cctype>
#include <clocale>

WORK_SPACE; // r = C2*a + C1*b + C0;

void runForUpdateA(int value){
	a = value;
}
void runForUpdateB(int value){
	b = value;
}
void runForUpdateR(){
```

```
	std::cout << "(C2=" << C2 << " C1=" << C1 << " C0=" << C0 << " a=" << a << " b=" << 
b << ")" << std::endl;
	std::cout << "r = C2*a + C1*b + C0 = " << RUN_FOR_UPDATE << std::endl;
}
int main()
{
	std::cout << "(use key a, b, r; use <Ctrl+C> for exit)" << std::endl << "Run('r'<= 
merge('a', 'b')):" << std::endl;
	auto keys = rx::observable<>::create<int>(
		[](rx::subscriber<int> dest){
		for (;;) {
			int key = std::cin.get(), value;
			if (std::tolower(key) == 'a') {
				std::cout << "=";
				std::cin >> value;
				runForUpdateA(value);
			}
			else if (std::tolower(key) == 'b') {
				std::cout << "=";
				std::cin >> value;
				runForUpdateB(value);
			}
			dest.on_next(key);
		}
	}).
		publish();

	auto a = keys.
		filter([](int key){return std::tolower(key) == 'a'; });

	auto b = keys.
		filter([](int key){return std::tolower(key) == 'b'; });

	auto r = keys.
		filter([](int key){return std::tolower(key) == 'r'; });

	r.merge(a, b).
		subscribe([](int key){
		if (std::tolower(key) == 'r'){
			runForUpdateR();
		}
		else{
			std::cout << "value updated" << std::endl;
		}
	});

	keys.connect(); // run the loop in create

	return 0;
}
```

4. Завдань

Детально ознайомитися з наведеним прикладом програми*(при захисті потрібно вміти пояснити будь-яку стрічку цієї програми)*. Значення констант обрати згідно варіанту.

n	Константи		
	C2	C1	C0
1	111	112	107
2	145	146	141
3	165	166	161

4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421
17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

5. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.