МАТЕРІАЛИ РЕГІОНАЛЬНОЇ КОНФЕРЕНЦІЇ СТУДЕНТІВ І МОЛОДИХ ВЧЕНИХ

(Квітень 1998 року)

С.В. Кур'ята

БІБЛІОТЕКА КЛАСІВ БАЗОВИХ МАТЕМАТИЧНИХ СТРУКТУР ДАНИХ ДЛЯ ПРОГРАМНОГО КОМПЛЕКСУ "DSR OPEN LAB 1.0"

(Науковий керівник к.т.н., доц. Колодницький М.М.)

У даній роботі розглядається метод побудови базових структур даних програмного комплексу "DSR Open Lab 1.0" на основі стандартної бібліотеки шаблонів STL. Також наводиться загальний опис математичних властивостей основних математичних об'єктів, які використовувались у цій розробці.

У наш час, в зв'язку з швидким розвитком комп'ютерних платформ, програмного забезпечення та постійним підвищенням вимог до них, серед головних задач, що виникають у процесі розробки проекту, особливо гостро постає задача підвищення тривалості життєвого циклу комплексу. Необхідною умовою вирішення цієї задачі є правильно спроектована базова структура даних. Розглянемо проектування цієї структури на прикладі програмного комплексу "DSR Open Lab 1.0", що розроблюється на кафедрі ПЗОТ ЖІТІ [2, 4].

Комплекс "DSR Open Lab 1.0" – це пакет програм моделювання динамічних систем. Він, головним чином, призначений для використання у навчальному процесі, в той час як існуюче прикладне програмне забезпечення класу CAD/CAM/CAE/PDM-систем є в основному орієнтованим на застосування у виробничій сфері.

За своїм функціональним призначенням пакет охоплює наступну множину математичних структур: абстрактні множини, числові системи, алгебраїчні системи, векторний простір, системи нелінійних алгебраїчних рівнянь, звичайні диференційні рівняння, оптимізаційні моделі. І цей список далеко не повний. Тому грунтовно спроектована базова структура даних є одним із головних чинників, що впливає на якість роботи, життєвий цикл комплексу, можливість його перенесення на різноманітні комп'ютерні платформи, надійність математичних обчислень, можливість розвитку пакета з появою нових вимог до його функціональності, а також на збільшення продуктивності праці спеціалістів при розробці пакета.

Отже, як бачимо, все вищевикладене накладає суворі вимоги до розробки структури даних.

Бібліотека класів базових математичних структур даних для програмного комплексу "DSR Open Lab 1.0" побудована на основі бібліотеки Standard Template Library (STL). STL підтримується компіляторами таких інструментальних засобів, як Borland C++ 5.0, Borland C++ Builder, Microsoft Visual C++ версії 4.0 та вище. Для розробки пакета "DSR Open Lab 1.0" використовується мова програмування Microsoft Visual C++ версії 5.0. STL є складовою частиною великої бібліотеки Standard C++ Library, яка була розроблена в результаті стандартизації мови програмування C++ такими організаціями, як International Standards Organization (ISO) та American National Standards Institute (ANSI).

Головними частинами ANSI/ISO Standard C++ Library є: велика кількість структур даних та алгоритмів їх обробки, потоки введення/виведення, шаблонізований клас string, шаблонізований клас подання комплексних чисел, засоби управління пам'яттю, засоби управління виняткових ситуаціями процесора.

Як бачимо, STL – це основна частина Standard C++ Library, яка складається з колекції визначень класів для стандартних структур даних та колекції загальних алгоритмів, які використовуються для маніпулювання такими структурами.

Отже, фундаментом нашої бібліотеки класів є потужні стандартні структури даних, що значно спрощує розробку та перекладає велику частину функціональності на вже розроблені та відлагоджені алгоритми.

За базові математичні структури даних ми вирішили обрати наступні: класи чисел, класи виняткових ситуацій процесора, класи векторів, матриць, списку та дека. Дамо опис цим структурам даних.

Класи чисел

Числа відіграють одну з найважливіших ролей у будь-якому алгоритмі. Це обумовлено тим, що вони є основою для побудови кожної структури даних, і саме при роботі з числами виникає найбільша кількість помилок. Тому нами були реалізовані такі класи чисел: клас натурального числа, клас цілого числа, клас дійсного числа. Зрозуміло, що ці класи мають багато спільного, тому на початку розглянемо їх спільні властивості. По-перше, ці класи реалізують всі стандартні функції (операції +, – і т. д.) для чисел, а також надають ряд додаткових. Серед додаткових функцій наступні: min() – повертає найменше число з допустимого діапазону, max() – повертає найбільше число з допустимого діапазону. По-друге, при виникненні переповнення класами цілого та натурального чисел генерується виняткова ситуація CDSRIntOverflow (див. опис класів виняткових ситуацій). По-третє, класи цілого та натурального чисел мають дві програмні реалізації. Це пов'язано з тим, що функція перевірки на переповнення вимагає зайвих витрат процесорного часу, що зменшує продуктивність системи, але не у всіх задачах вимагається реалізація описаних вище властивостей. Перша реалізація (вона є реалізацією за замовчанням) не виконує жодних перевірок. Щоб використати другу програмну реалізацію класів, необхідно визначити макрос CHECK_INT_OVERFLOW, тобто в текст програми вставити рядок #define CHECK_INT_OVERFLOW.

Клас натурального числа

В багатьох математичних задачах і алгоритмах використовуються натуральні числа. Інструментальні засоби не надають попередньо визначених типів даних, що відображають властивості натуральних чисел, тому стає очевидною необхідність у реалізації власного класу натурального числа.

Клас натурального числа ми назвали CDSRNatural. Цей клас реалізує наступні функції для роботи з натуральними числами: перевірку на переповнення; перевірку на вихід з діапазону (наприклад, при спробі присвоювання числа менше одиниці); операції введення/виведення в потік; арифметичні операції; логічні операції.

Клас цілого числа

В ході розв'язку деяких задач можуть з'явитися числа великого порядку, наприклад, обчислення числа 100!. При цьому інтервали значень цілих змінних, що надають більшість з існуючих мов програмування (наприклад, ANSI C і C++), не задовольняють поставленим задачам і не контролюють вихід значення за допустимий діапазон, тобто не контролюють переповнення. Тому виникла необхідність в реалізації власного класу цілого числа, що виконує перевірку на переповнення при операціях з цілими числами, а також ряд інших перевірок. Клас цілого числа ми назвали CDSRInteger.

Клас дійсного числа

При реалізації практичних задач, особливо різноманітних чисельних методів, велика кількість помилок виникає під час роботи з дійсними типами даних. До помилок, що найбільш часто зустрічаються, відносяться: переповнення (до нього, наприклад, може призвести перемноження великих чисел, піднесення експоненти до великого (позитивного або негативного) степеня, ділення на нуль); втрата значущих цифр. У зв'язку з цим нами був спроектований клас дійсного числа. Він був названий CDSRReal. Клас CDSRReal надає повний набір стандартних функцій, а також ряд додаткових.

Виняткові ситуації

Нижче будуть описані виняткові ситуації, які можуть бути згенеровані при роботі з числами-векторами, матрицями, списками, деками. Весь список виняткових ситуацій наведено в табл. 1.

Таблиця 1

Список виняткових ситуацій

№	Назва виняткових ситуацій
1	CDSRIntOverflow
2	CDSRNaturalRangeOut
3	CDSRRangeOut
4	CDSRCompareException

Виняткова ситуація CDSRIntOverflow

Ситуація CDSRIntOverflow може бути згенерована класами цілого або натурального чисел. Ця ситуація генерується при математичних операціях, які призвели до переповнення ти-

пу. Цей клас має два даних члени класу: value1 – значення першого числа, при операції з яким була згенерована виняткова ситуація; value2 – значення другого числа.

Виняткова ситуація CDSRNaturalRangeOut

Ситуація CDSRNaturalRangeOut може бути згенерована класом натурального числа. Ця ситуація генерується при спробі присвоїти натуральному числу значення, яке не належить до множини значень натурального числа. Цей клас має одну змінну – член класу: value – значення числа, яке намагалися присвоїти натуральному числу перед тим, як була згенерована виняткова ситуація.

Виняткова ситуація CDSRRangeOut

Ситуація CDSRRangeOut може бути згенерована кожним з класів вектора, матриці, списку або дека. Ця ситуація генерується при спробі виходу за межі масиву. Цей клас має два даних-членів класу: index – номер елемента, до якого було звернення перед тим, як була згенерована виняткова ситуація; size – розмір масиву.

Виняткова ситуація CDSRCompareException

Ситуація CDSRCompareException може бути згенерована кожним з класів вектора, матриці, списку або дека. Ця ситуація генерується при спробі виконати операцію з масивами, що мають різні розміри. Цей клас має два даних-членів класу: size1 – розмір першого масиву; size2 – розмір другого масиву.

Базові контейнерні структури даних

Перед тим, як розглядати кожну структуру даних окремо, щоб уникнути повторень, розглянемо їх спільні властивості. По-перше, вектор, матриця, список та дек відслідковують наступні виняткові ситуації процесора:

- вихід за межі контейнера, при цьому генерується виняткова ситуація CDSRRangeOut (див. опис виняткових ситуацій);
- виконання операцій з контейнерами різних розмірів, при цьому генерується виняткова ситуація CDSRCompareException (див. опис виняткових ситуацій).

По-друге, кожна структура даних має "дружний оператор" (friend operator) порівняння двох її об'єктів. По-третє, якщо у векторі, матриці, списку або дека будуть зберігатися об'єкти типу користувача, то для них необхідно визначити наступні функції: конструктор за замовчанням, конструктор копіювання, оператор порівняння "==", оператор порівняння "<", оператор порівняння ">".

Вектор

З математичної точки зору вектор являє собою одновимірний масив елементів, для якого визначена множина операцій над ним. Базові операції з цієї множини наведені в табл. 2.

Таблиця 2

Множина операцій над вектором, матрицею, списком та деком

	Властивість	Вектор	Матриця	Список	Дек
1	Отримання доступу до елемента за індексом	√	√	√	
2	Визначення кількості елементів, що зберігаються у контейнері	√	√	√	√
3	Вилучення елемента	√		√	√
4	Приєднання нового елемента	√		√	√
5	Знаходження елемента з заданим значенням	√		√	√
6	Додавання значень двох контейнерів	√	√		
7	Знаходження різниці значень двох контейнерів	√	√		
8	Перемноження значень двох контейнерів	√	√		
9	Перемноження на число	√	√		

У бібліотеці STL вектор – це контейнерний клас, що концептуально реалізує звичайний масив мови Сі. Як і масив, вектор – це структура даних, що має індексований доступ. Значення індексу може змінюватись від 0 до значення, що на одиницю менше за кількість елементів, що зберігаються у векторі. Однак, властивості вектора відрізняються від властивостей масиву у наступному: вектор містить більше інформації про себе, ніж звичайний масив, наприклад, вектор має значення кількості елементів, що він зберігає, та скільки він може потенціально зберігати і т. д; розмір вектора може динамічно змінюватись, нові елементи можуть бути вставлені як в кінець вектора, так і в середину.

Отже, виходячи з цього всього, для реалізації класу вектора нами було спроектовано три класи:

- CDSRBaseVector – базовий клас;
- CDSRArray – класс системного вектора (масиву);
- CDSRMVector – клас математичного вектора.

Ієрархія цих класів наведена на рис. 1. Як видно з цього рисунка, всі перераховані вище класи виведені з класу бібліотеки Standard Template Library (STL) vector.

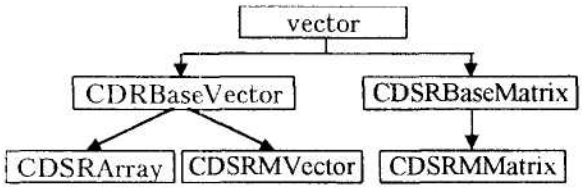

Рис.1. Ієрархія класів вектора та матриці

Для всіх класів реалізована перевірка на вихід за межі вектора та недопустимість операцій з векторами різних розмірів.

Клас CDSRBaseVector

Клас CDSRBaseVector є шаблонним класом, нащадком класу vector бібліотеки STL. Він надає можливість зберігати дані будь-якого типу. Набір базових операцій для роботи з вектором, що реалізує клас CDSRBaseVector, наведений у таблиці 3.

Таблиця 3

Набір базових операцій, що реалізує клас CDSRBaseVector

Функція	Опис
CDSRBaseVector(CDSRBaseVector& vec)	Конструктор копіювання
CDSRBaseVector(T* array, unsigned long size)	Конструктор, де array – масив об'єктів типу T, size – розмір цього масиву
CDSRBaseVector(unsigned long size = 0)	Конструктор, де start – початковий розмір вектора
T& operator [] (unsigned long index)	Оператор доступу до елемента масиву за індексом, де index – номер (індекс) елемента
unsigned long size()	Функція, що повертає кількість елементів, які зберігаються в масиві
unsigned long find(T& value, unsigned long startIndex=0)	Функція проводить пошук елемента value в масиві, починаючи з елемента з номером startIndex. Якщо елемент був знайдений, то повертає його номер у масиві. У протилежному випадку – повертає максимальне значення діапазону типу unsigned long

Клас CDSRArray

Клас CDSRArray є шаблонним класом, нащадком класу CDSRBaseVector. Він являє собою клас масиву (системного вектора). Клас надає можливість зберігати дані будь-якого типу. Клас CDSRArray реалізує набір базових операцій для роботи з масивом. Оскільки він є нащадком класу CDSRBaseVector, то він має весь набір функцій, який надає користувачу клас-предок. Список додаткових функцій, що реалізує цей клас, наведений у таблиці 4.

Таблиця 4

Додаткові функції, що реалізує клас CDSRArray

Функція	Опис
void add(T value)	Функція додає значення змінної value в кінець вектора. При цьому розмір вектора збільшується на 1
void add(unsigned long count, T value)	Функція додає число (count) копій змінної value в кінець вектора. При цьому розмір вектора збільшується на число (count) одиниць
void erase(unsigned long index)	Функція вилучає елемент з номером index з масиву. При цьому розмір вектора зменшується на 1
void insert(unsigned long index, T value)	Функція вставляє значення змінної value в комірчину масиву, яка має номер index. При цьому розмір вектора збільшується на 1

Якщо порівняти набір функцій класу масиву та набір функцій, який надає клас списку, що буде описаний нижче, то можна побачити, що вони майже не відрізняються один від одно-

го. При цьому може скластися хибне уявлення про те, що один з цих класів зайвий. Але це не так. Ці два класи суттєво відрізняються своєю внутрішньою побудовою. Якщо масив зберігає дані у великому єдиному блоці пам'яті, то у списку кожен об'єкт зберігається в окремій комірчині пам'яті. Це є перевагою списку. Але швидкість доступу до будь-якого значення, що зберігається у векторі, значно вище за швидкість цього доступу у списку (це є перевагою вектора). Таким чином, і клас масиву, і клас списку має право на існування, а застосування кожної з цих структур визначається конкретною задачею.

Клас CDSRMVector

Клас CDSRMVector є шаблонним класом, нащадком класу CDSRBaseVector. Він є класом математичного вектора. Клас надає можливість зберігати дані будь-якого типу. Клас CDSRMVector реалізує набір базових операцій для роботи з математичним вектором. Оскільки є нащадком класу CDSRBaseVector, то він має весь набір функцій, який надає користувачу клас-предок. До списку додаткових функцій, що реалізує цей клас, входять: додавання двох векторів, віднімання двох векторів, перемноження вектора на число.

Якщо у векторі будуть зберігатися об'єкти типу користувача, то для них необхідно визначити функції, що були зазначені вище та додатково наступні: оператор додавання "+", оператор віднімання "–", сператор множення "*".

Матриця

У зв'язку з тим, що майже в усіх математичних задачах використовуються матриці, то виникла необхідність у реалізації класу матриці, який би використовувався в усіх алгоритмах, вносячи в них одноманітність.

Матрицею є двовимірний масив зі своєю множиною базових операцій. Множина цих базових операцій наведена в табл. 2.

У бібліотеці STL не існує реалізованого класу матриці. Тому ми реалізували його на основі шаблонного класу vector бібліотеки STL.

Для реалізації поставленої задачі нами було спроектовано два класи матриці:

- CDSRBaseMatrix – базовий клас;
- CDSRMMatrix – клас математичної матриці.

Ієрархія цих класів наведена на рис. 1. Як видно з цього рисунка, всі перераховані вище класи виведені з класу бібліотеки Standard Template Library (STL) vector.

Для всіх класів реалізована перевірка на вихід за межі матриці та недопустимість операцій з матрицями різних розмірів.

Клас CDSRBaseMatrix

CDSRBaseMatrix є шаблонним класом, нащадком класу vector бібліотеки STL. Він надає можливість зберігати дані будь-якого типу. Клас CDSRBaseMatrix реалізує набір базових операцій для роботи з матрицею. Список функцій, що реалізує цей клас, наведений у таблиці 5.

Таблиця 5

Список функцій, що реалізує клас CDSRBaseMatrix

Функція	Опис
CDSRBaseMatrix(CDSRBaseMatrix& matr);	Конструктор копіювання
CDSRBaseMatrix(T** matr, unsigned long m, unsigned long m);	Конструктор, де matr – матриця об'єктів типу T; n, m – розміри цієї матриці
CDSRBaseMatrix(unsigned long m, unsigned long m);	Конструктор, де n, m – початкові розміри матриці
T& operator () (unsigned long i, unsigned long j);	Оператор доступу до елемента матриці за індексами
unsigned long n_row();	Функція, яка повертає кількість рядків у матриці
unsigned long n_column();	Функція, яка повертає кількість стовпчиків у матриці
CDSRBaseVector* diag();	Функція, яка повертає діагональ матриці
CDSRBaseVector* row(unsigned long Row);	Функція, яка повертає рядок матриці, де Row – номер рядка
CDSRBaseVector* column(unsigned long Column);	Функція, яка повертає стовпчик матриці, де Column – номер стовпчика
CDSRBaseMatrix operator = (CDSRBaseMatrix& matr);	Оператор привласнення

Клас CDSRMMatrix

Клас CDSRMMatrix є шаблонним класом, нащадком класу CDSRBaseMatrix. Він є класом математичної матриці. Клас CDSRMMatrix надає можливість зберігати дані будь-якого типу. Клас CDSRMMatrix реалізує набір базових операцій для роботи з математичною матрицею. Оскільки цей клас є нащадком класу CDSRBaseMatrix, то він має весь набір функцій, який надає користувачу клас-предок. Список додаткових функцій, що реалізує цей клас, наведений у табл. 6.

Якщо у векторі будуть зберігатися об'єкти типу користувача, то для них необхідно визначити функції, що зазначалися в описі базового класу, та додатково наступні: оператор додавання "+", оператор віднімання "–", оператор множення "*".

Таблиця 6

Список додаткових функцій, що реалізує клас CDSRMMatrix

Функція	Опис
CDSRMMatrix<T> operator + (CDSRMMatrix<T>&, CDSRMMatrix<T>&);	Додає дві матриці
CDSRMMatrix<T> operator – (CDSRMMatrix<T>&, CDSRMMatrix<T>&);	Віднімає дві матриці
CDSRMMatrix<T> operator * (CDSRMMatrix<T>&, CDSRMMatrix<T>&);	Перемножує дві матриці
CDSRMVector<T> operator * (CDSRMMatrix<T>&, CDSRMVector<T>&);	Перемножує матрицю на вектор
CDSRMVector<T> operator * (CDSRMVector<T>&, CDSRMMatrix<T>&);	Перемножує вектор на число
CDSRMVector<T> operator * (CDSRMVector<T>&, T&);	Перемножує число на матрицю

Список (List)

Багато алгоритмів для своєї роботи потребують реалізації такої структури даних, як список. У зв'язку з цим виникла необхідність проектування власного класу списку, який ми назвали CDSRList.

Лінійний список (ЛС) – це множина, структурні властивості якої обмежуються лише лінійним (одновимірним) відносним положенням її елементів [3]. Елементи ЛС часто ще називають вузлами або записами, тобто вони можуть бути об'єктами необов'язково скалярного типу. Кількість вузлів у послідовності може змінюватися. Список операцій, що можуть виконуватись над лінійним списком, наведено в таблиці 2.

Клас CDSRList є шаблонним класом, нащадком класу list бібліотеки STL. Він надає можливість зберігати дані будь-якого типу. Клас CDSRList реалізує набір базових операцій для роботи зі списком. Список функцій, що реалізує цей клас, наведений у табл. 7.

Таблиця 7

Список функцій, що реалізує клас CDSRList

Функція	Опис
CDSRList(CDSRList& _list);	Конструктор копіювання
CDSRList(T* array, unsigned long size);	Конструктор, де array – масив об'єктів типу T; size – розмір цього масиву
CDSRList(unsigned long size = 0);	Конструктор, де start – початковий розмір списку
T& operator [] (unsigned long index);	Оператор доступу до елемента списку за індексом, де index – номер (індекс) елемента списку
unsigned long size();	Функція, що повертає кількість елементів, які зберігаються у списку
unsigned long find(T& value, unsigned long startIndex = 0);	Функція проводить пошук елемента value у списку, починаючи з елемента з номером startIndex. Якщо елемент був знайдений, то повертає його номер у списку. У протилежному випадку – повертає максимальне значення діапазону типу unsigned long
void add(T value);	Функція додає значення змінної value в кінець списку. При цьому розмір списку збільшується на 1
void add(unsigned long count, T value);	Функція додає число (count) копій змінної value в кінець списку. При цьому розмір списку збільшується на число (count) одиниць
void erase(unsigned long index);	Функція вилучає елемент з номером index зі списку. При цьому розмір списку зменшується на 1
void insert(unsigned long index, T value);	Функція вставляє значення змінної value в комірчину списку, яка має номер index. При цьому розмір списку збільшується на 1

Дек (Deque)

Деякі алгоритми для своєї роботи потребують реалізації такої структури даних, як дек. У зв'язку з цим виникла необхідність проектування власного класу дека, який ми назвали CDSRDeque.

Дек (deque, double ended queue) – це двостороння черга, тобто лінійний список, в якому всі приєднання елементів, вилучення та, взагалі, будь-який доступ здійснюється на обох його кінцях і тільки на них [3].

У бібліотеці STL дек – це контейнерний клас, властивості якого є об'єднанням можливостей вектора та списку, а саме: як і вектор, дек – це структура даних, що має індексований доступ, тобто значення елемента може бути отримано за його індексом; як і у списку, елементи ефективно додаються у початок або кінець; новий елемент може бути вставлений у середину дека.

На основі цього нами був спроектований клас CDSRDeque.

Клас CDSRDeque є шаблонним класом, нащадком класу deque бібліотеки STL. Він надає можливість зберігати дані будь-якого типу. Клас CDSRDeque реалізує набір базових операцій для роботи з деком. Список функцій, що реалізує цей клас, наведений у табл. 8.

Таблиця 8

Список функцій, що реалізує клас CDSRDeque

Функція	Опис
CDSRDeque(CDSRDeque& _deque);	Конструктор копіювання
CDSRDeque(T* array, unsigned long size);	Конструктор, де array – масив об'єктів типу T; size – розмір цього масиву
CDSRDeque(unsigned long size = 0);	Конструктор, де start – початковий розмір дека
T& operator [] (unsigned long index);	Оператор доступу до елемента дека за індексом, де index – номер (індекс) елемента дека
unsigned long size();	Функція, що повертає кількість елементів, які зберігаються в дека
unsigned long find(T& value, unsigned long startIndex=0);	Функція проводить пошук елемента value в дека, починаючи з елемента з номером startIndex. Якщо елемент був знайдений, то повертає його номер у дека. У протилежному випадку – повертає максимальне значення діапазону типу unsigned long
Void add(T value);	Функція додає значення змінної value в кінець дека. При цьому розмір дека збільшується на 1
Void add(unsigned long count, T value);	Функція додає число (count) копій змінної value в кінець дека. При цьому розмір дека збільшується на число (count) одиниць
Void erase(unsigned long index);	Функція вилучає елемент з номером index із дека. При цьому розмір дека зменшується на 1
Void insert(unsigned long index, T value);	Функція вставляє значення змінної value в комірчину дека, яка має номер index. При цьому розмір дека збільшується на 1

Отже, як бачимо, розроблена бібліотека класів базових математичних структур даних для програмного комплексу "DSR Open Lab 1.0" може легко переноситися на будь-яку платформу та модернізуватися, у разі необхідності, за короткий проміжок часу; вона вносить однотипність та структуровність у всі алгоритми комплексу. Це значно спрощує та стандартизує проектування та програмування будь-яких задач.

ЛІТЕРАТУРА

1. *Кнут Д.* Искусство программирования для ЭВМ. Т.1. Основные алгоритмы. – М.: Мир, 1976. – 736 с.
2. *Колодницкий Н.М.* Пакет программ "Dynamical systems research (DSR)" // Праці Житомирського філіалу КПІ. Серія А. Техніка. – Вип. 1. – Житомир: ЖФ КПІ, 1993. – С. 81–94.
3. *Колодницький М.М.* Технічне та програмне забезпечення комп'ютерних інформаційних технологій. – Житомир: ЖІТІ, 1995. – 231 с.
4. *Колодницький М.М., Рожик О.А., Левицький В.Г., Шкаленко С.П., Гладишев А.В.* Програмний комплекс для моделювання динамічних систем "DSR LAB. 1.0" // Сучасні технології в аерокосмічному комплексі. Матеріали III Міжнародної наук.–практ. конференції, 9–11 вересня 1997 року. – Житомир: ЖІТІ, 1997.– С. 97–99.

КУР'ЯТА Сергій Валерійович – студент V курсу факультету інформаційно-комп'ютерних технологій Житомирського інженерно-технологічного інституту.

Наукові інтереси:
– мови програмування високого рівня;
– комп'ютерна техніка.