

Ім'я користувача:
Volodymyr Donchenko

Дата перевірки:
28.12.2023 11:04:43 EET

Дата звіту:
28.12.2023 11:13:30 EET

ID перевірки:
1016039566

Тип перевірки:
Doc vs Internet

ID користувача:
100012947

Назва документа: Лазоренко Олександр Сергійович 1 – копія

Кількість сторінок: 51 Кількість слів: 8005 Кількість символів: 59937 Розмір файлу: 1.66 MB ID файлу: 1015733354

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

11.9%
Схожість

Найбільша схожість: 2.71% з Інтернет-джерелом (https://shron1.chtyvo.org.ua/Yavorskyi_Nazarii/Laboratnyi_praktyk...)

11.9% Джерела з Інтернету

184

Сторінка 53

Пошук збігів з Бібліотекою не проводився

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.3%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 9 слів та 0%)

0.3% Вилучення з Інтернету

30

Сторінка 54

Немає вилучених бібліотечних джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи

29

Підозріле форматування

8
сторінок

ВСТУП

В даний час широко поширені CASE- системи (Computer-Aided System/Software Engineering), які дозволяють автоматизувати процес розробки програмних продуктів. Однак така автоматизація, як правило, не є повною, так як внутрішні коди методів класів пропонується вписувати вручну.

Отже, актуальною є задача повної автоматизації процесу створення програмних продуктів, в рамках якої реалізується власна система автоматичної генерації програмних кодів по графічним схемам алгоритму (ГСА).

На даний момент існує безліч способів графічного зображення алгоритму. Серед найбільш відомих способів можна назвати наступні: блок-схема, UML-діаграми (діяльності, стану, послідовності), блок-схема і діаграма Нассі-Шнейдермана.

Аналіз існуючих графічних нотацій бізнес-процесів показує, що найбільш ефективними з точки зору відображення концептуальних і реалізаційних особливостей є модельний апарат UML. Даний факт в сумі з простотою вивчення робить UML найбільш популярним графічним мовою опису проектних рішень [1].

Особливістю UML-діаграм є те, що вони підтримують об'єктно-орієнтовану парадигму. Серед статичних моделей об'єктно-орієнтованих програм основною є діаграма класів. Генерація об'єктно-орієнтованого програмного коду по UML-діаграмі класів дає на виході так званий «скелет» програмного продукту, так як він визначає реалізацію конкретних методів, але не дозволяє отримати повний виконуваний код [2]. З цієї причини CASE-системи, які використовують UML, не дозволяють повністю автоматизувати процес створення програмного продукту. Разом з тим слід зазначити, що великий відсоток витрат у рамках проекту пов'язаний саме з кодуванням тіл методів.

В роботі [3] представлена система, яка генерує вихідний код для класу, який не містить методів, що значно звужує область використання такої системи. В роботі [4] наведено опис системи з широким колом функціональних можливостей, але є вузькоспеціалізованою - підсумкова згенерувала програма призначена для роботи тільки з графічними зображеннями або потоковим відео.

Система, представлена в роботі [5], підтримує тільки структурну модель програмування, залишаючи поза увагою об'єктно-орієнтовану, що також звужує коло вирішуваних нею завдань.

Алгоритм трансляції неструктурованої моделі процесу в структурну форму наведено в роботі [6].

Автори роботи [7] представляють систему генерації, яка підтримує BPMN і DSL (domainspecific language), що також не гарантує отримання виконаного коду, проте вимагає створення нової предметно-орієнтованої мови або ознайомлення з попередньо створеною предметно-орієнтованою мовою для певної задачі.

Генерація коду, присутня в роботі [8], однак вихідним описом для представленої в цій роботі системи є бізнес-модель, що є досить високим рівнем абстракції алгоритму, і, отже, не завжди підходить для вирішення завдання проєктування.

Об'єкт дослідження - генерація кодів по графічним схемам алгоритму.

Предмет дослідження - системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем.

Мета роботи - розробка системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем.

Досягнення зазначеної мети передбачає вирішення таких основних завдань:

- Провести огляд та аналіз існуючих систем візуального проєктування та автоматичної генерації програмного коду на основі блок-схем;
- Провести моделювання та аналіз програмного забезпечення модуля для середовища розв'язання задач з програмування;
- Розробити математичну модель для системи візуального проєктування та автоматичної генерації програмного коду за графічними схемами алгоритмів;
- Розробити та реалізувати систему візуального проєктування та автоматичної генерації програмного коду на основі блок-схем.

Методи дослідження - методи системного аналізу, аналітичної геометрії, теорії графів, теорії множин, теорії алгоритмів, математичної логіки, комбінаторики.

У першому розділі виконано рецензію та аналіз існуючих систем візуального проєктування та автоматичної генерації програмного коду на основі блок-схем. Розглянуті блок-схеми, області застосування і враховані стандарти. Додатково, розкрито основні алгоритмічні структури.

У другому розділі проведено детальний аналіз процесу розробки програмного забезпечення для системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем. Проведено моделювання та описана архітектура розробленого додатка. Описано математичну модель системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем.

Третій розділ присвячений етапу розробки системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем. Розроблений прототип інтерактивного побудови блок-схем дозволяє шляхом маніпулювання графічними об'єктами редагувати логіку програм, автоматично формувати програмні коди для різних мов програмування

(Pascal, C / C ++, Алгоритмічна мова). Програма написана на мові C ++ на основі бібліотеки Qt4.

РОЗДІЛ 1

АНАЛІЗ СИСТЕМ ВІЗУАЛЬНОГО ПРОЄКТУВАННЯ ТА АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ НА ОСНОВІ БЛОК-СХЕМ

1.1. Роль моделювання при вирішенні складних завдань

Формальний підхід до визначення об'єктів і систем матеріального світу вимагає введення в розгляд моделей (описів), що дозволяють досліджувати існуючі та розробляти нові системи. Моделі утворюють ієрархію, між рівнями якої встановлюється ланцюг відображень (рис. 1.1).



Рис. 1.1. Класифікація моделей за ступенем формалізації

1. Реальна система описується системою понять (концептів), між якими встановлюються причинно-наслідкові зв'язки.
2. Кожна концепція може бути зафіксована з різним ступенем деталізації, рівень визначається дослідником.
3. Інструментом створення формальних моделей на основі концептуальних є формальні мови (наприклад, U-мову, мову висловлювань), що дозволяють вести докази і міркування. Розробка формальної моделі, як правило, супроводжується графічними побудовами (граф станів, граф переходів системи).

4. Отримані формальні моделі можуть бути перетворені в математичні:

а) аналітичні моделі (показники ефективності виражаються в вигляді функцій щодо параметрів системи),

б) імітаційні моделі (моделювання за допомогою алгоритмів і програм, що імітують функціонування досліджуваної системи).

5. Математичні моделі не містять відомостей про те, як повинна перероблятися інформація, щоб отримати результат. Порядок переробки інформації необхідно визначити і представити у вигляді алгоритму розв'язання задачі (реалізації відповідної математичної моделі), тобто у вигляді алгоритмічної, а потім програмної моделі.

6. Основи алгоритмізації і програмування є фундаментальними і мають універсальний характер, що надає їм відповідність основним принципам математики, мови або логіки.

7. У процесі виконання завдання із застосуванням комп'ютера дослідник проходить ряд етапів (рис. 1.2).



Рис. 1.2. Етапи виконання завдання на комп'ютері

1.2. Аналітичний огляд існуючих систем візуального проектування та автоматичної генерації програмного коду на основі блок-схем

Для реалізації поставлених завдань необхідним є вивчення предметної області, а саме існуючих програм-редакторів блок-схем, їх переваг і недоліків.

Microsoft Office Visio 2007 - це програма для створення креслень і діаграм, яка сприяє візуалізації, аналізу та обміну складною інформацією для фахівців у галузі IT та бізнесу [1]. Вона дозволяє конвертувати важкі для розуміння тексти та таблиці в прості та зрозумілі діаграми Visio. Замість статичних малюнків користувачі можуть створювати динамічні діаграми Visio, тісно пов'язані з даними, що спрощує оновлення та підвищує продуктивність. Різноманітні діаграми Visio в рамках Office Visio 2007 допомагають зрозуміти інформацію про організаційні системи, ресурси та процеси на всьому підприємстві, полегшуючи процес прийняття рішень та обміну цією інформацією.

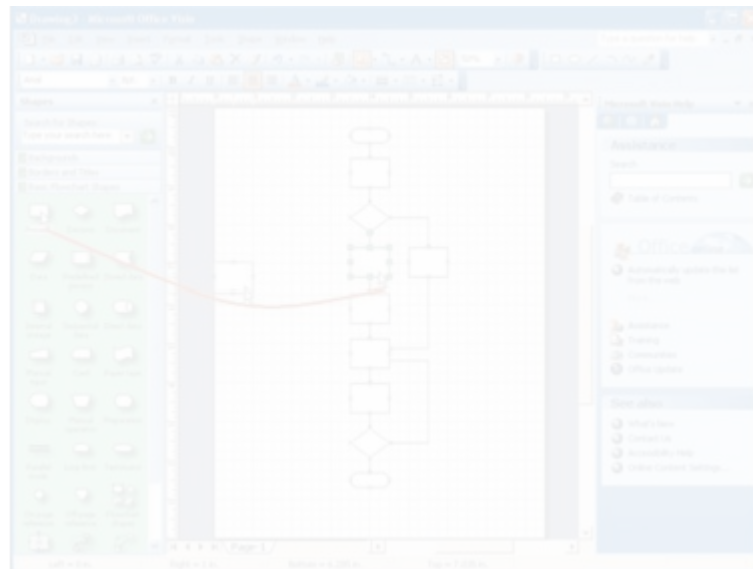


Рис. 1.3. Microsoft Office Visio 2007

Microsoft Office Visio 2007 має наступні переваги:

1. Візуалізація, дослідження і публікація систем, ресурсів, процесів і пов'язаних з ними даних [2]. Широкий вибір типів схем Office Visio 2007 забезпечує ефективну візуалізацію, дослідження і публікацію процесів, ресурсів, систем і пов'язаних з ними даних.

2. Візуалізація складної інформації шляхом виведення даних в схемах і виконання відповідних дій. Візуалізація даних в діаграмах дозволяє краще зрозуміти суть даних і виконувати необхідні дії в залежності від результатів аналізу. Використовуючи засіб Microsoft Office Visio 2007 «Малюнки, пов'язані з даними», можна зображувати дані в будь-якій діаграмі у вигляді тексту, елементів даних, значків і колірних позначень.

3. Швидке створення діаграм завдяки автоматичному підключенню до фігур Visio. У Office Visio 2007 є нова функція автоматичного з'єднання, яка може автоматично підключати, розміщувати і вирівнювати фігури на схемі, полегшуючи роботу користувача. Досить перетягнути фігуру на сторінку документа і розташувати її над однією з синіх стрілок, які знаходяться над фігурою, вже розміщеної на сторінці.

4. Подання комплексних даних з використанням нових шаблонів і фігур. Уявлення даних завдяки новим і вдосконаленим шаблонам, і фігурам. Наприклад, у випуску Office Visio Professional 2007 можна складати схеми процесів ІТ-служб за допомогою нового шаблону ITIL (Information Technology Infrastructure Library), створювати діаграми на основі економічної методології і візуалізувати ефективніші виробничі процеси за допомогою нового шаблону схеми потоку створення вартості.

5. Підвищення продуктивності завдяки інтеграції схем з даними з різних джерел. Джерела комплексних візуальних, текстових і числових даних. Діаграми, підключені до даних, забезпечують візуальний контекст для даних і дають повне уявлення про систему або процесі. Зв'язок схем з даними з різних джерел стала простіше завдяки функції зв'язування даних в Office

Visio Професійний 2007. Прив'язка даних до фігур діаграми виконується за допомогою нового майстра автоматичного зв'язування.

6. Дослідження даних для відстеження тенденцій, виявлення проблем та позначки винятків за допомогою зведених схем. Шаблон зведеної схеми дозволяє наочно представляти і аналізувати бізнес-дані в Office Visio Professional 2007 в ієрархічній формі у вигляді груп і підсумків даних. Є можливість деталізувати комплексні дані, відображати інформацію за допомогою засобу «Малюнки, пов'язані з даними», динамічно створювати різні представлення даних і краще розуміти складну інформацію. Зведені схеми можна вставляти в будь-яку діаграму Visio, щоб отримувати доступ до показників і звітів, які допоможуть відслідковувати поточний стан процесу або системи. Зведені схеми створюються шляхом підключення до різних джерел даних, таким як Microsoft Office SharePoint Server 2007, Microsoft Office Project 2007 і Microsoft Office Excel 2007. Можливість створення наочних звітів програм Office SharePoint Server 2007 і Office Project 2007 в формі зведеної схеми підвищує ефективність контролю ресурсів і проектів, керованих за допомогою цих програм.

7. Ефективне поширення інформації за допомогою професійно оформлених схем. Для створення діаграм Visio професійної якості досить вибрати колір або ефект (текст, заливку, тінь, лінії, формат з'єднувачів) для всієї діаграми з використанням нової функції «Тема». На вибір в Visio пропонується ряд вбудованих тем, але можна створювати і власні теми. У Office Visio 2007 використовуються ті ж вбудовані теми, що і в інших програмах системи Microsoft Office 2007.

Редактор блок-схем - це спеціалізована програма, яка надає необхідний набір інструментів для створення блок-схем. Це є важливою перевагою цієї програми порівняно з графічними редакторами. Наявність додаткових опцій у редакторі дозволяє оптимізувати процес розробки блок-схем і їх подальше перетворення в процедури і функції мови програмування.

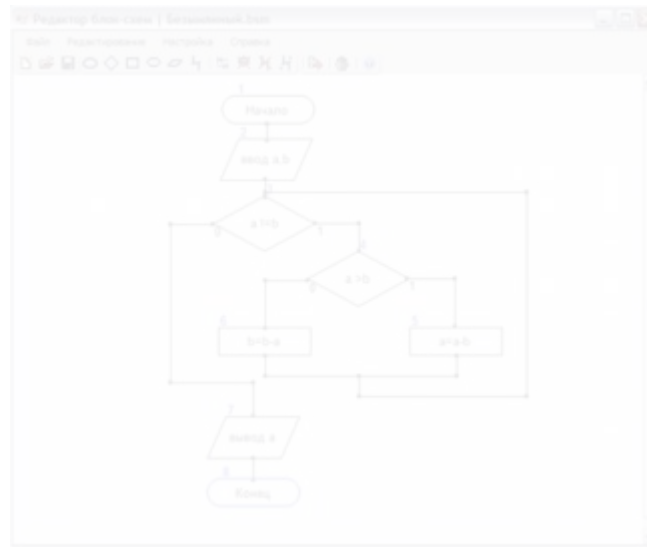


Рис. 1.4. Редактор блок-схем

Системні вимоги програми дуже скромні і вона запускається практично на будь-якому комп'ютері з будь-якою версією Windows.

Звернемо увагу на ключові можливості редактора, серед яких важливі наступні:

1. Використання шаблонів при створенні блок-схем.
2. Імпорт процедур і функцій із мов програмування. Редактор дозволяє імпортувати реалізації процедур і функцій, написаних на популярних мовах програмування. Ця функція допомагає краще розібратися в структурі алгоритму, реалізованого на конкретній мові програмування.
3. Експорт блок-схем у вигляді процедур і функцій для мов програмування.
4. Експорт блок-схем в різні графічні формати.

FCEditor. Заснована ідея цієї програми - зобразити блок-схему з блоків з довільним за величиною (мається на увазі текст) змістом. У більшості редакторів, якщо і є можливість автоматично змінювати розмір компонентів,

то всі стрілки і переходи все одно треба розставляти вручну. У FCEditor це робиться автоматично.

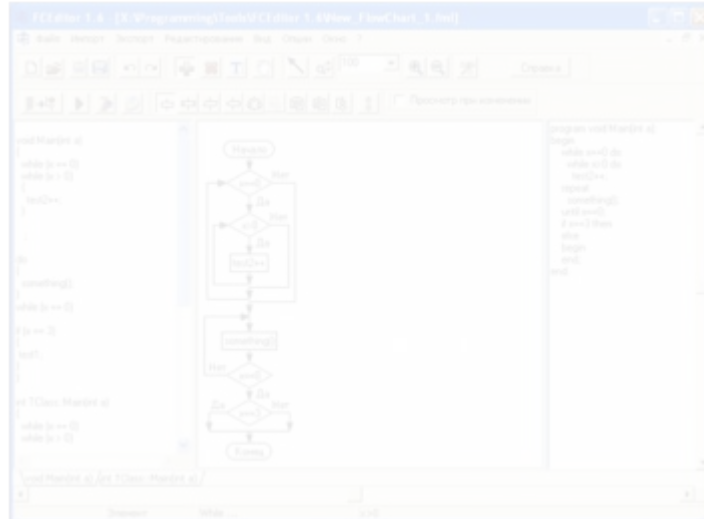


Рис. 1.5 FCEditor

Можливості FCEditor:

1. імпорт схеми з програмного коду;
2. автоматичне вирівнювання блоків і стрілок;
3. можливість зміни типу блоків;
4. копіювання і вставка блоків схеми;
5. окрема схема для кожної процедури;
6. можливість вставки розривів сторінок;
7. експорт схеми в графічний файл;
8. експорт схеми в код.

Таким чином, незважаючи на невелику вагу FCEditor є досить багатофункціональною програмою-редактором схем.

Розглянувши існуючі програми-редактори схем, ми можемо їх узагальнити, кожна з них:

1. Чи може використовувати шаблони для створення схем.
2. Може експортувати створену схему в різні формати.

3. Володіє автоматичної «підгонкою» блоків і стрілок.

4. Чи може створювати різні типи схем.

Однак у проаналізованих програм-редакторів схем є недоліки - вони не пристосовані для українських графічних стандартів. Жодна з розглянутих програм не має функцію автоматичного розміщення блоків і не дозволяє отримати код програми відповідної побудованої блок схемою.

Проектована нами програма повинна володіти позитивними властивостями, описаними вище, а також повинна мати можливість будувати схеми за українськими стандартами, наприклад ГОСТ 19.701. Так само в створювану програму буде включено інтерпретатор програмного коду, згенерованого програмою, з його допомогою програму можна буде використовувати для вирішення алгоритмічних задач або для навчання програмуванню. Алгоритм програми можна буде редагувати в візуальному режимі, а не в режимі набору програмного коду. Потім за допомогою програми цей код можна буде виконати і перевірити результат роботи алгоритму. Це істотно підвищить ефективність процесу навчання програмуванню на початковому етапі.

1.3. Графічний спосіб запису алгоритмів







Графічний спосіб відображення алгоритмів є більш компактним і зрозумілим у порівнянні зі словесним варіантом. У графічному представленні алгоритм подається у вигляді послідовності взаємопов'язаних функціональних блоків, кожен із яких відповідає виконанню конкретної дії чи групи дій.

Такий графічний опис часто називають схемою алгоритму або блок-схемою. У блок-схемі кожному типу дій (введення вихідних даних, обчислення значень виразів, перевірки умов, управління повторенням дій, завершенню обробки і т. д.) відповідає геометрична фігура, яка представлена у вигляді блочного символу. Ці блочні символи з'єднуються лініями

переходів, які вказують порядок виконання дій. У таблиці 1.2 наведено найбільш поширені символи, які використовуються в блок-схемах.

Таблиця 1.2

Блокові символи

| Назва символу | Позначення і приклад заповнення | Пояснення |
|--------------------|---|---|
| Процес |  | Обчислювальна дія чи послідовність дій |
| Розгалуження |  | Перевірка умови |
| Модифікація |  | Початок циклу |
| Визначений процес |  | Обчислення по підпрограмі, стандартній підпрограмі |
| Введення-виведення |  | Введення-виведення у загальному виді |
| Пуск - зупинення |  | Початок, кінець алгоритму, вхід і вихід підпрограми |
| Документ |  | Виведення результатів на друк |

Блок "процес" служить для позначення дії або послідовності дій, що призводять до змін у значеннях, формі або розташуванні даних. Щоб поліпшити зрозумілість схеми, можна об'єднати кілька окремих блоків обробки в один. Подання окремих операцій може бути викладено вільно.

Блок "рішення" використовується для позначення переходів управління в залежності від умови. В кожному блоку "рішення" мають бути визначені питання, умова або порівняння, які визначають перехід.

Блок "модифікація" використовується для організації циклічних конструкцій. Термін "модифікація" вказує на видозміну або перетворення. Усередині блоку вказується параметр циклу, для якого вказуються початкове значення, гранична умова і крок зміни значення параметра для кожного повторення.

Блок "зумовлений процес" використовується для вказівки на звернення до допоміжних алгоритмів, які функціонують автономно як самостійні модулі, і для викликів бібліотечних підпрограм.

1.4. Блок-схеми, області застосування і стандарти

Блок-схема - це один із способів візуального представлення алгоритмів, орієнтований граф, який вказує порядок виконання команд алгоритму»[2]. Блок-схеми широко застосовуються у вивченні теорії алгоритмів, а також на практиці, там, де потрібно зручне наочне уявлення алгоритмів.

Для опису блок-схем існувало кілька стандартів, з яких найбільш свіжим є ГОСТ 19.701-90, затверджений 01 січня 1992 р. Нижче наведені уривки специфікації даного стандарту, присвячені так званим схемам програм.

"Справжній стандарт розповсюджується на умовні позначення (символи) в схемах алгоритмів, програм, даних і систем, встановлюючи правила виконання схем, що використовуються для представлення різних завдань обробки даних та методів їх вирішення.

Стандарт не впливає на формат записів і позначок, які розташовані всередині або поруч із символами і службовців для уточнення їхніх функцій.

Вимоги стандарту є обов'язковими.

ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Схеми алгоритмів, програм, даних і систем (далі - схеми) складаються з визначених символів, короткого пояснювального тексту та лінійних з'єднань.

1.2. Схеми можуть застосовуватися на різних рівнях деталізації, залежно від розміру та складності завдань обробки даних. Рівень деталізації повинен бути обраний так, щоб різні частини та їх взаємозв'язок були зрозумілі в цілому.

1.3. У цьому стандарті визначені символи, призначені для використання в документації щодо обробки даних, і надається керівництво з умовних позначень для їхнього використання в:

- 1) схемах даних;
- 2) схемах програм;
- 3) схемах роботи системи;
- 4) схемах взаємодії програм;
- 5) схемах ресурсів системи.

1.4. У стандарті використовуються такі поняття:

- 1) основний символ - символ, який застосовується у випадках, коли точний тип (вид) процесу або носія даних невідомий або не вимагає опису конкретного носія даних;
- 2) специфічний символ - символ, який використовується у випадках, коли відомий точний тип (вид) процесу або носія даних, або коли потрібно описати конкретний носій даних;
- 3) схема - графічне представлення визначення, аналізу або методу розв'язання задачі, в якому застосовуються символи для відображення операцій, даних, потоку, обладнання і т. д."

ОПИС СХЕМ

[...]

2.2. Схема програми

2.2.1. Схеми програм відображають послідовність операцій у програмі.

2.2.2. Схема програми складається з:

- 1) символів процесу, що вказують фактичні операції обробки даних (включаючи символи, що визначають шлях, якого слід дотримуватися з урахуванням логічних умов);
- 2) лінійних символів, що вказують потік управління;
- 3) спеціальних символів, що використовуються для полегшення написання і читання схеми.

[...]

Приклади виконання схем наведені в додатку.

ОПИС СИМВОЛІВ

3.1. Символи даних

3.1.1. Основні символи даних

3.1.1.1. Дані

Символ відображає дані, носій даних не визначено.

[...]

3.2. Символи процесу

3.2.1. Основні символи процесу

3.2.1.1. Процес

Символ відображає функцію обробки даних будь-якого виду (виконання певної операції або групи операцій, що приводить до зміни значення, форми або розміщення інформації або до визначення, за яким з декількох напрямків потоку слід рухатися).

3.2.2. Специфічні символи процесу

3.2.2.1. Зумовлений процес

Символ відображає зумовлений процес, який складається з однієї або декількох операцій або кроків програми, що визначені в іншому місці (в підпрограмі, модулі).

[...]

3.2.2.3. Підготовка

Символ відображає модифікацію команди або групи команд з метою впливу на деяку подальшу функцію (встановка перемикача, модифікація індексного регістра або ініціалізація програми).

3.2.2.4. Рішення

Символ відображає рішення або функцію перемикача типу, що має один вхід і ряд альтернативних виходів, один і тільки один з яких може бути активізований після обчислення умов, визначених всередині цього символу. Відповідні результати обчислення можуть бути записані по сусідству з лініями, що відображають ці шляхи.

3.2.2.5. Паралельні дії

Символ відображає синхронізацію двох або більше паралельних операцій.

[...]

3.2.2.6. Границя циклу

Символ, що складається з двох частин, відображає початок і кінець циклу. Обидві частини символу мають один і той же ідентифікатор. Умови для ініціалізації, збільшення, завершення і т. д. поміщаються всередині символу на початку або в кінці в залежності від розташування операції, перевіряє умову.

[...]

3.3. Символи ліній

3.3.1. Основний символ ліній

3.3.1.1. Лінія

Символ відображає потік даних або управління. При необхідності або для полегшення читання можуть бути додані стрілки-показчики.

3.3.2. Спеціфічні символи ліній

[...]

3.3.2.3. Пунктирна лінія

Символ показує альтернативний зв'язок між двома або більше символами. Крім того, цей символ використовується для виділення анованого фрагмента.

3.4. Спеціальні символи

3.4.1. З'єднувач

Символ відображає зв'язок між частинами схеми, використовуючи вихід та вхід з іншої частини цієї схеми. Використовується для обриву лінії та її продовження в іншому місці. Символи-з'єднувачі повинні мати унікальне позначення.

3.4.2. Термінатор

Символ вказує на вихід до зовнішнього середовища і вхід з нього (початок або кінець схеми програми, взаємодія зовнішніх даних та їхні джерела або пунктів призначення).

3.4.3. Коментар

Символ використовується для додавання описових коментарів або пояснювальних записів для пояснення чи вказівки. Пунктирні лінії у символі коментаря пов'язані з відповідним символом або можуть оточувати групу символів. Текст коментарів або приміток повинен бути розташований близько до меж фігури.

3.4.4. Пропуск

Символ (три крапки) використовується в схемах для позначення пропуску символу або групи символів, для яких не визначені тип чи кількість. Використовується переважно в схемах, що зображують загальні рішення з невідомою кількістю повторень.

[...]

4. ПРАВИЛА ЗАСТОСУВАННЯ СИМВОЛІВ І ВИКОНАННЯ СХЕМ

4.1. Правила використання символів

4.1.1. Кожен символ призначений для графічної ідентифікації функції, яку він відображає, незалежно від тексту всередині нього.

4.1.2. Розташування символів на схемі повинно бути рівномірним. Рекомендується дотримуватися адекватної довжини з'єднань і мінімізувати кількість довгих ліній.

4.1.3. Більшість символів розроблені так, щоб дозволяти включення тексту всередині них. Форми символів, визначені цим стандартом, повинні служити директивою для реально використовуваних символів і не повинні зазнавати змін в аспектах, таких як кути та інші параметри, що впливають на їх форму. Символи, якщо це можливо, повинні мати однаковий розмір. Їх можна зображувати в будь-якій орієнтації, проте, за можливість, горизонтальна орієнтація є бажаною. Дзеркальне відображення форми символу вказує на ту ж функцію, і не є кращим вибором.

4.1.4. Мінімальний обсяг тексту, необхідного для розуміння функції символу, слід поміщати всередину нього. Текст слід читати зліва направо та зверху вниз, незалежно від напрямку потоку.

[...]

4.1.5. У схемах може бути використаний ідентифікатор символу, пов'язаний з ним. Це ідентифікує символ для використання у довідкових цілях в інших частинах документації, таких як лістинг програми. Ідентифікатор символу повинен розташовуватися ліворуч вище символу.

[...]

4.1.6. У схемах може бути використаний опис символу - будь-яка інша інформація, яка може вказувати на специфічне застосування символу з перехресним посиланням або поліпшити розуміння функції як частини схеми. Опис символу має бути розташований справа вище символу.

[...]

4.1.8. У схемах може використовуватися детальне уявлення, позначене символом із смугою для процесу або даних. Цей символ із смугою вказує, що в тому ж комплекті документації гілку з більш детальним уявленням можна знайти в іншому місці. Символ із смугою представлений будь-яким

символом, на верхній частині якого проведена горизонтальна лінія. Між цією лінією та верхньою лінією символу розташований ідентифікатор, який вказує на детальне уявлення даного символу. Перший та останній символи цього уявлення повинні бути символами покажчика кінця. Перший символ покажчика кінця має містити посилання, яке також присутнє у символі із смугою.

[...]

4.2. Правила проведення з'єднань

4.2.1. Потоки даних або управління на схемах відображаються лініями. Стандартною вважається напрямок зліва направо і зверху вниз, а стрілки використовуються для підвищення ясності при необхідності або при з'єднаннях, де напрямок відрізняється від стандартного.

4.2.2. В уникайте перетинання ліній у схемах, оскільки перетинаючіся лінії не мають логічного зв'язку, і зміна напрямку в точках перетину не допускається.

[...]

4.2.3. Дві або більше входні лінії можуть об'єднуватися в одну вихідну лінію, і місце об'єднання повинне бути зміщеним.

[...]

4.2.4. Лінії на схемах повинні підходити до символу або зліва, або зверху, а виходити або справа, або знизу, і вони повинні бути направлені до центру символу.

4.2.5. Лінії у схемах можна розривати, якщо це необхідно для уникнення зайвих перетинів або довгих ліній, а також у випадку, якщо схема складається з декількох сторінок. З'єднувач на початку розриву називається зовнішнім з'єднувачем, а з'єднувач в кінці розриву - внутрішнім з'єднувачем.

4.2.6. Посилання на сторінки можуть бути подані разом із символом коментаря для їх з'єднувачів.

[...]

4.3. Спеціальні умовні позначення

4.3.1. Кілька виходів

4.3.1.1. Декілька виходів з символу слід відображати:

- 1) декількома лініями від даного символу до інших символів;
- 2) однією лінією від даного символу, яка потім розгалужується в відповідне число ліній.

[...]

4.3.1.2. Кожен вихід з символу повинен мати відповідні значення умов, щоб вказати логічний шлях, який він представляє, і щоб ці умови та відповідні посилання були ідентифіковані.

[...]

4.3.2. Періодичне уявлення

[...]


4.3.2.3. Лінії можуть входити або виходити з будь-якої точки перекриття символів, проте вони повинні дотримуватися вимог пункту 4.2.4. Порядок більше одного символу не змінюється за допомогою точки, в якій лінія входить або виходить.

1.5. Базові алгоритмічні структури

Алгоритми можна уявляти як конструкції, які складаються з окремих базових (основних) елементів. Логічна структура будь-якого алгоритму може бути виражена комбінацією трьох основних структур: послідовності, умовного розгалуження та циклу. При вивченні принципів конструювання алгоритмів цієї підходить доцільно розпочати з основних елементів. Для їх опису можна скористатися мовою схем алгоритмів та стандартною алгоритмічною мовою.

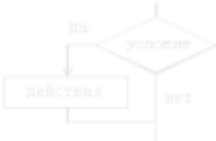

Кожна з базових структур має характерну особливість у вигляді одного входу та одного виходу. Розглянемо першу з них:

1. **Структура "Послідовність" (слідування):** ця структура утворюється послідовністю дій, які виконуються одна за одною.

| Шкільна алгоритмічна мова | Мова блок-схем |
|----------------------------------|--|
| дія 1 дія 2 дія n |  |

2. **Основна структура "Розгалуження":** Ця структура забезпечує можливість вибору одного з альтернативних напрямків виконання алгоритму, залежно від результату перевірки певної умови (чи вона виконана, чи ні). Кожен з варіантів роботи алгоритму призводить до одного загального виходу, що гарантує продовження роботи алгоритму незалежно від вибору конкретного напрямку. Структура розгалуження може приймати чотири основні форми:

- "якщо – то";
- "якщо – щось - інакше";
- "вибір";
- "вибір - інакше".

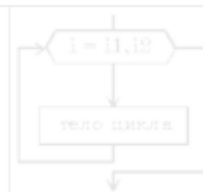
| Шкільна алгоритмічна мова | Мова блок-схем |
|---|--|
| 1. якщо - то | |
| якщо умова то дії усе |  |
| 2. якщо – щось - інакше | |
| якщо умова то дії 1 інакше дії 2 усе |  |

3. вибір**вибір****при умова 1: дії 1****при умова 2: дії 2****.....****при умова N: дії N****Усе****4. вибір - інакше****вибір****при умова 1: дії 1****при умова 2: дії 2****.....****при умова N: дії N****інакше дії N + 1****Усе**

3. **Основна структура "Цикл":** Ця структура дозволяє повторно виконувати певний набір дій, що називається тілом циклу. Основні види циклів наведено в таблиці:

| Шкільна алгоритмічна мова | Мова блок-схем |
|--|----------------|
| Цикл "ПОКИ" визначає, що тіло циклу має виконуватися доти, доки виконується умова, вказана після ключового слова "поки". | |
| нц поки умова тіло циклу (послідовність дій) кц | |
| Цикл "ДЛЯ" вказує на виконання тіла циклу для кожного значення певної змінної (параметра циклу) в заданому діапазоні. | |

НЦ для i від i1 до i2
тіло циклу
(послідовність дій)
кц



1.6. Висновки до розділу 1

У проаналізованих системах візуального проєктування та автоматичної генерації програмного коду на основі блок-схем виявлені недоліки, оскільки вони не відповідають українським графічним стандартам. Жодна з розглянутих програм не має функцію автоматичного розміщення блоків і не дозволяє отримати код програми відповідної побудованої блок схемою.

Проєктована нами програма повинна володіти позитивними властивостями, описаними вище, а також повинна мати можливість будувати схеми за стандартами. Так само в створювану програму буде включено інтерпретатор програмного коду, згенерованого програмою, з його допомогою програму можна буде використовувати для вирішення алгоритмічних задач або для навчання програмуванню. Алгоритм програми можна буде редагувати в візуальному режимі, а не в режимі набору програмного коду. Потім за допомогою програми цей код можна буде виконати і перевірити результат роботи алгоритму. Це істотно підвищить ефективність процесу навчання програмуванню на початковому етапі.

РОЗДІЛ 2

МОДЕЛЮВАННЯ СИСТЕМИ ВІЗУАЛЬНОГО ПРОЄКТУВАННЯ ТА АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ НА ОСНОВІ БЛОК-СХЕМ

Моделювання процесів системи є дуже важливим етапом, тому що саме тут описуються вимоги до системи з боку користувача, основні функції, алгоритм роботи системи, описується взаємодія користувача з системою і об'єктів системи між собою.

Процес бізнес-моделювання [10] може бути реалізований за допомогою різних методик, які відрізняються переважно своїм підходом до того, що представляє собою модельована організація. Згідно з різними уявленнями про організацію, методики розподіляються на об'єктні та функціональні (структурні).

Об'єктні методики розглядають модельовану організацію як набір взаємодіючих об'єктів - виробничих одиниць. Об'єкт визначається як відчутна реальність - предмет або явище, що має чітко визначену поведінку. Метою застосування даної методики є виділення об'єктів, які становлять організацію, та розподіл відповідальностей за виконувані дії між ними.

Функціональні методики, де найвідомішою є методика IDEF, розглядають організацію як набір функцій, що перетворюють вхідний потік інформації в вихідний потік. Процес перетворення інформації вимагає використання певних ресурсів. Основна відмінність від об'єктної методики полягає в чіткому відділенні функцій (методів обробки даних) від самих даних.

З точки зору бізнес-моделювання, кожен з представлених підходів має свої переваги. Об'єктний підхід дозволяє побудувати більш стійку до змін систему і краще відповідає існуючим структурам організації. Функціональне моделювання добре проявляє себе в тих випадках, коли організаційна структура перебуває в процесі змін або взагалі слабо визначена. Підхід,

орієнтований на виконанні функції, інтуїтивно краще зрозумілий виконавцям при отриманні від них інформації про їх поточну роботу.

2.1. Розробка функціональної моделі системи

Методологія функціонального моделювання IDEF0 є простим інструментом, який надає розробникам корпоративних інформаційних систем змогу вивчити сферу діяльності замовника та вирішувати завдання, спрямовані на підвищення ефективності цієї діяльності [3].

Застосування функціонального моделювання не лише допомагає вирішувати технічні проблеми, пов'язані з інформаційними технологіями, але також вирішує проблеми, що відносяться до сфери діяльності замовника. Це перетворює проект інформаційної системи з "пачки паперів", за яку замовник не хоче платити, в послугу, яка може принести замовникові додатковий ефект, порівняно з подальшою автоматизацією.

Використання діаграм декомпозиції дозволяє моделювати ситуації, де будь-яка дія, операція або функція може бути поділена (декомпозована) на менш складні елементи. Зображення функцій у вигляді блоків дозволяє докладно розглядати їх структуру і склад, подібно до того, як можна заглянути всередину блоку.

В ході проектування структури інтерфейсу системи з використанням стандарту IDEF0 ми отримали наступні функціональні блоки:

1. Блок A0: Створити програму за допомогою візуального редагування її блок-схеми (рисунок 2.1).
2. Блок A1: Підготувати умови для створення схеми (рисунок 2.2).
3. Блок A2: Розробити бета-версію схеми (рисунок 2.2).
4. Блок A3: Редагувати блок-схему (рисунок 2.2).
5. Блок A11: Погодити з замовників умови створення блок-схеми (рисунок 2.3).
6. Блок A12: Описати вимоги до реалізації завдання (рисунок 2.3).
7. Блок A13: Формалізувати опис завдання (рисунок 2.3).

8. Блок A21: Визначити список змінних, процедур і функцій, які будуть використовуватися в програмі (рисунок 2.4).

9. Блок A22: Побудувати структуру схеми (рисунок 2.4).

10. Блок A23: Підписати блоки в схемі (рисунок 2.4).



Рис. 2.1. Контекстна діаграма «Створити програму за допомогою візуального редагування її блок-схеми»



Рис. 2.2. Декомпозиція контекстної діаграми «Створити програму за допомогою візуального редагування її блок-схеми»



Рис. 2.3. Діаграма декомпозиції блоку A1 «Підготувати умови для створення блок-схеми»

У даній системі діаграми IDEF0 допомогли виявити основні функції системи і нормативні документи, з якими потрібно узгоджуватися при проектуванні програми. Було виявлено що схема певного типу може складатися тільки з деякою сукупності блоків. Для цього блоки були розділені на класи.

В ході проектування схеми IDEF0 були виявлені основні функції майбутньої системи, які був згодом реалізовані в програмі.

Діаграма потоків даних дозволяє визначити основні вимоги до даних ще на етапі функціонального моделювання [4]. На цій діаграмі відображаються завдання, які входять до описуваного бізнес-процесу, а також показуються входи та виходи кожного з завдань. Ці входи та виходи представлені як інформаційні або матеріальні потоки, і, при цьому, виходи одного завдання можуть слугувати входами для інших завдань. Використання елементів для опису джерел, приймачів і сховищ даних у діаграмах DFD

сприяє більш ефективному та наочному висвітленню процесу документообігу.

У даній системі за допомогою DFD діаграми були виявлені зв'язки між деякими класами. Було виявлено, що схема доступна для редагування може перевірятися на будь-якому етапі її проектування. Таким чином, процедура перевірки коректності схем була вбудована в клас вікна відповідального за редагування схеми. Було виявлено, що схеми потрібно зберігати для подальшого редагування або в файл .jpg. Таким чином, ми з'ясували, що вся схема повинна бути збережена в формат, відриваючи який можна відновити схему повністю. На основі отриманої вимоги було прийнято рішення використовувати серіалізацію об'єктів, тобто збереження всіх об'єктів схеми і їх станів на жорсткий диск. Для цього був обраний клас XmlSerializer, який дозволяє зберігати стан об'єктів в форматі XML і потім відновлювати об'єкти з цього формату. Для збереження в .jpg була використана стандартна процедура збереження зображення в файл, діаграма представлена на рисунку 2.4.



Рис. 2.4. Рівень A0 DFD-діаграми «Спроектувати схему програми».

Існує так само альтернативний спосіб виявлення необхідної функціональності системи. Виявити функціональність системи можна за допомогою діаграми прецедентів.

2.2. Логічне представлення моделі поведінки програмної розробки

Поведінка програмної розробки визначається безліччю об'єктів, що обмінюються повідомленнями.

Діаграми прецедентів, один із п'яти видів діаграм в UML для моделювання динамічних аспектів системи [4], відіграють ключову роль у визначенні поведінки системи, підсистеми або класу. Кожна така діаграма відображає різноманітні прецеденти, акторів і їх взаємозв'язки.

Використання діаграм прецедентів спрямоване на моделювання системи з точки зору прецедентів або варіантів використання. Зазвичай це охоплює моделювання контексту системи, підсистеми або класу, або визначення вимог до їх поведінки.

Діаграми прецедентів є важливим інструментом для візуалізації, специфікації та документування поведінки елементів. Вони сприяють кращому розумінню систем, підсистем або класів, надаючи зовнішній огляд того, як ці елементи можуть використовуватися в конкретному контексті. Крім того, ці діаграми є важливим інструментом для тестування реалізованих систем під час прямого проектування та для розуміння їх внутрішньої структури під час зворотного проектування.

Діаграма прецедентів представлена на рисунку 2.5.



30

Рис. 2.5. Діаграма прецедентів.

В даній системі за допомогою діаграми прецедентів були виявлені основні функції, які повинна виконувати система. Було отримано список функцій, які необхідно реалізувати в системі.

Якщо ми порівнюємо результат застосування діаграми прецедентів з результатом застосування діаграм IDEF0 і DFD, то ми побачимо, що діаграми IDEF0 і DFD допомогли більш точно сформулювати вимоги, що пред'являються до функціональності системи. В результаті декомпозиції окремих процесів основні елементи інтерфейсу програми були визначені ще до написання програмного коду.

2.3. Логічне уявлення статичної моделі структури програмної розробки

Комплексний проєкт програмної системи є компіляцією моделей логічних і фізичних концепцій, які повинні бути взаємно узгоджені. В мові UML для статичного відображення системних моделей використовуються діаграми класів, які визначають структуру класів об'єктів та їх взаємозв'язки.

На діаграмі класів представлено докладний опис класів використовуються в програмі.

Спочатку розглянемо класи блоків. На рисунку 2.6 приведена діаграма класів блоків.

Головний блок (ChartMainBlock) реалізує метод збереження схеми. У програмі схема зберігається в XML формат. Для цього був обраний клас XmlSerializer з простору імен System.Xml.Serialization.

Простір імен System.Xml.Serialization містить класи, використовувані для серіалізації об'єктів в документи або потоки формату XML.

Центральним класом в просторі імен є клас XmlSerializer, який дозволяє зберігати стан об'єктів в форматі XML і потім відновлювати об'єкти з цього формату. XmlSerializer серіалізуються і десеріалізують об'єкти в документи XML і з них. XmlSerializer дозволяє контролювати спосіб

кодування об'єктів в XML. Приклад XML файлу згенерованого програмою можна подивитися в додатку Б.

Кожен блок успадковується від абстрактного класу Block. Текст блоку зберігається в поле Text, для циклів додатково використовується поле TextAtTheEnd (визначає текст блоку, який закриває цикл), тип блоку визначається класом блоку. Кожен блок визначає власний метод для відтворення.

Блок певного типу реалізує тільки власний метод відтворення, все інше він успадковує від абстрактного класу Block.



Рис. 2.6. Діаграма класів

Далі розглянемо клас, який представляє всю блок-схему. Діаграма класу блок-схеми наведена на рисунку 2.7.

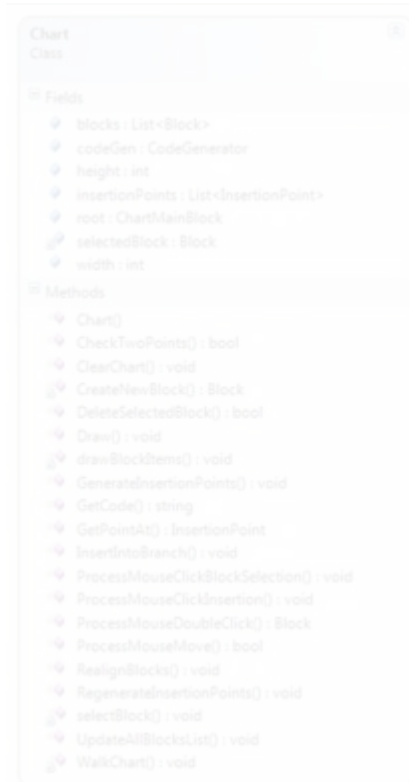


Рис. 2.7. Діаграма класу блок-схеми

Для зберігання всієї схеми в програмі є клас `Chart`. Він зберігає кореневий елемент схеми (блок початку схеми), список точок вставки, об'єкт для генерації коду програми зі схеми, а так само службові дані. Клас схеми реалізує методи вставки та видалення блоків, очищення всієї схеми і автоматичного вирівнювання блоків і точок вставки.

Клас блок схеми реалізує методи для автоматичного вирівнювання блоків і точок вставки (`RealignBlocks`, `RegenerateInsertionPoints`, `GenerateInsertionPoints`), методи для вставки і видалення блоків (`InsertIntoBranch`, `CreateNewBlock`, `DeleteSelectedBlock`, `ClearChart`), методи

для обробки подій миші (ProcessMouseClicked, ProcessMouseClickedBlockSelection, ProcessMouseMove, ProcessMouseDoubleClick), так само цей клас має метод для отримання коду програми (GetCode).

Далі розглянемо клас точки вставки InsertionPoint. Він наведений на рисунку 2.8.

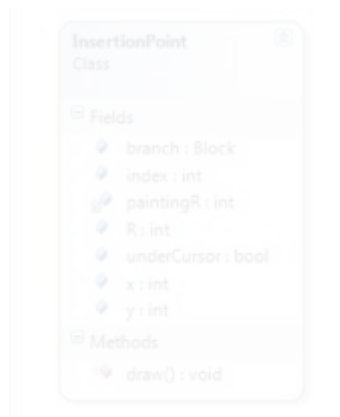


Рис. 2.8. Діаграма класу точки вставки

Далі розглянемо клас генератора коду програми. Він наведений на рисунку 2.9.

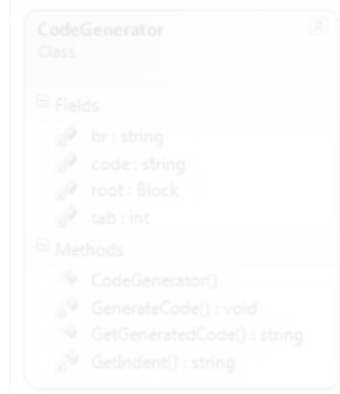


Рис. 2.9. Діаграма класу генератора коду

2.4. Математичний опис програми

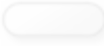
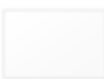

В ході реалізації виділених вище функцій необхідно описати алгоритми і моделі даних на формалізованому або природною мовою.


2.4.1. Опис моделі даних

Основною структурною одиницею, що розробляється є блок. Таблиця 2.1 містить використовувані в програмі блоки.

Таблиця 2.1

Основні блоки, використовувані в програмі

| Найменування | Позначення | Функція |
|---|---|---|
| Блок початок-кінець (пуск-зупинка) |  | Об'єкт відображає вхід або вихід зовнішнього середовища, що часто використовується для позначення початку або завершення програми. У середині символу фігурує відповідна дія. |
| Блок обчислень, висновок даних, зумовлений процес |  | Виконання однієї або кількох операцій, обробка різних типів даних (зміна їх значень, форми подання чи розташування). У межах символу додають записи, що відображають самі операції. Також в цей блок можна включати операції виведення. Цей блок вказує на виконання процесу, який складається з однієї або кількох операцій, який визначений в іншому місці програми (у підпрограмі, модулі). У межах символу вказується назва процесу і передані йому дані, такі як виклик процедури чи функції. Цей блок завжди має один вхід і один вихід. |
| Логічний блок (блок умови) |  | Відтворює вибір або дію перемикача, який має один вхід і два можливих виходи, проте лише один з них обирається в результаті обчислення умов, визначених всередині цього елемента. Вхід у елемент позначається лінією, що входить в його верхню вершину, тоді як кожен вихід відображається лінією, що виходить з бічних вершин. Кількість входів – два, кількість виходів – один. |

| | | |
|---------------|---|---|
| Границя циклу |  | Символ складається із двох сегментів: відповідно, початку і кінця циклу. Операції, які виконуються всередині циклу, розташовані між цими сегментами. Умови циклу та збільшення записуються всередині символу початку або кінця циклу, в залежності від типу структури циклу. Кожен такий блок має один вхід і один вихід. |
|---------------|---|---|

2.4.2. Математичний опис використовуваних моделей даних

Списки використовуються для подання кортежів.

Кортеж - це кінцева послідовність, можливо з повтореннями, елементів деякої множини E . Елементами кортежу можуть бути числа, символи деякого алфавіту, точки площині і т.д. У більш складних випадках елементами кортежу, в свою чергу, можуть бути також кортежі. Елементи, які не є кортежами, називаються атомами. Кількість елементів в кортежі називається його довжиною. Зручно розглядати кортежі, що не містять жодного елементу. Такі кортежі називаються порожніми. Довжина порожнього кортежу вважається рівною 0.

Елемент кортежу характеризується своїм номером у послідовності (кортежних номером) і змістом, тобто елементом безлічі E . Якщо довжина кортежу дорівнює n , $n > 0$, то кортеж S зручно розглядати як відображення s безлічі $N = \{1, 2, \dots, n\}$ в безліч E . Таким чином, $s(i)$ - це i -й елемент кортежу S .

Термін "список" використовується як узагальнююча назва різних структур даних, використовуваних для подання кортежів в пам'яті комп'ютера. При поданні кортежу в пам'яті з'являється ще одна характеристика елементу кортежу - його позиція в пам'яті. У деяких випадках номер елемента в кортежі і його позиція в пам'яті пов'язані один з одним арифметичними співвідношеннями таким чином, що по номеру легко обчислюється позиція i , навпаки, по позиції обчислюється номер. В інших випадках зв'язок між номерами і позиціями задається "таблично" або здійснюється за допомогою алгоритмічних процедур. Безліч позицій

позначимо через P . Іноді зручно вважати, що в безлічі P є спеціальний елемент nil , який вказує на неіснуючу область пам'яті. Таким чином, при розгляді того чи іншого списку ми маємо справу з трьома множинами E , N , P і з відображеннями на цих множинах.

Типовими при роботі зі списками є такі операції:

- знаходження позиції елемента в пам'яті по його номеру в кортежі;
- знаходження позиції елемента, наступного в кортежі за елементом з заданої позиції;
- знаходження позиції елемента, що передує в кортежі елементу із заданої позиції;
- видалення елемента, що знаходиться в заданій позиції;
- вставка в кортеж нового елемента перед елементом, розташованим в заданій позиції;
- визначення довжини кортежу.

2.4.3. Опис структур даних

Для більш глибокого розуміння завдання слід описати використовувані структури даних математично.

Нехай $L = \langle q, Listofblocks, pointer(K) \rangle$ - поточний блок,

де q - булева змінна, яка характеризує тип блоку: $q = 0$ блок є звичайним блоком, $q = 1$ -блок є блоком-гілкою.

$$\begin{cases} Listofblocks = \{\}, \text{ якщо } q = 1 \\ Listofblocks = \{ L_1, L_2, \dots, L_n \}, \text{ якщо } q = 0 \end{cases}$$

де L_1, L_2, \dots, L_n - список блоків, що належить даній гілці; N -кількість блоків, що належить даній гілці; $Pointer(K)$ - покажчик на гілку K , всередині якої знаходиться блок.

Вся схема представляється набором гілок. Головний блок зберігає в собі одну головну гілку. Блок-умова в списку вкладених блоків зберігає дві гілки, вони позначають лівий і правий виходи з блоку-умови. У середині блоку-

циклу зберігається одна гілка, яка, зберігає всі блоки, які знаходяться всередині циклу.

Точки вставки не зберігаються за допомогою блоками, вони представляються окремими об'єктами.

Нехай $T \langle L, \text{Pointer}(K) \rangle$ - точка вставки. Де L - це блок, після якого розташовується точка; $\text{Pointer}(K)$ Показчик на гілку, всередині якої знаходиться блок відповідний точці вставки.

2.5. Висновки до розділу 2

Проведено аналіз процесу розробки програмного забезпечення системи візуального проектування та автоматичної генерації програмного коду на основі блок-схем. Проведено моделювання та описана архітектура розробленого додатка. Описано математичну модель системи візуального проектування.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ СИСТЕМИ ВІЗУАЛЬНОГО ПРОЄКТУВАННЯ ТА
АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ НА ОСНОВІ
БЛОК-СХЕМ3.1. Обґрунтування вибору середовища розробки системи для
вирішення завдань

Qt Creator є недавно розробленим кросплатформним інтегрованим середовищем, призначеним для розробки Qt-додатків на мовах програмування, таких як C, C++, і QML. Він включає зручний відладчик та різноманітні візуальні інструменти розробки інтерфейсу, використовуючи QtWidgets і QML. Розробники цього програмного рішення ставили перед собою завдання значно спростити розробку додатків за допомогою фреймворка Qt для різних платформ.

Qt Creator - це повністю інтегроване середовище розробки (IDE), яке надає інструменти для проєктування і розробки складних додатків для безлічі настільних і мобільних платформ.



Проекти

Однією з ключових переваг Qt Creator є його здатність дозволяти розробникам працювати над проектом на різних платформах за допомогою спільних інструментів для розробки та налагодження. Проекти важливі, оскільки Qt Creator використовує їхню інформацію для компіляції та запуску додатків, подібно до вимог компілятора. Створення проекту дозволяє групувати файли, додавати власні кроки збірки, включати форми та файли ресурсів, а також встановлювати параметри для запуску додатків.

Qt Creator підтримує як створення нового проекту, так і імпорт існуючого, генеруючи необхідні файли відповідно до типу обраного проекту. Наприклад, обираючи створення додатка з графічним інтерфейсом користувача (GUI), Qt Creator автоматично генерує порожній .файл, який можна налаштувати за допомогою вбудованого Qt Designer.

Редактори

Qt Creator постачається з редактором коду і Qt Designer для проектування і створення графічних інтерфейсів користувача (GUI) з використанням віджетів Qt.

Редактор коду

Як інтегроване середовище розробки (IDE), Qt Creator володіє функціональністю, що виходить за рамки звичайного текстового редактора. Він розуміє мови програмування C++ і QML як код, дозволяючи писати добре форматований код, автоматично доповнювати його, відображати повідомлення про помилки і попередження, надавати контекстно-залежну довідку та забезпечувати ряд інших функцій для зручності розробки.

Дизайнер інтерфейсу

Qt Designer дозволяє проектувати та налаштовувати віджети та діалоги графічного інтерфейсу користувача (GUI) з використанням віджетів Qt. Ці створені за допомогою Qt Designer елементи можуть легко інтегруватися в програмний код за допомогою механізму сигналів і слотів Qt, що полегшує

визначення поведінки графічних елементів. Властивості, встановлені в Qt Designer, можуть бути динамічно змінені в коді, а також можна використовувати власні віджети та модулі для подальшого вдосконалення функціональності.

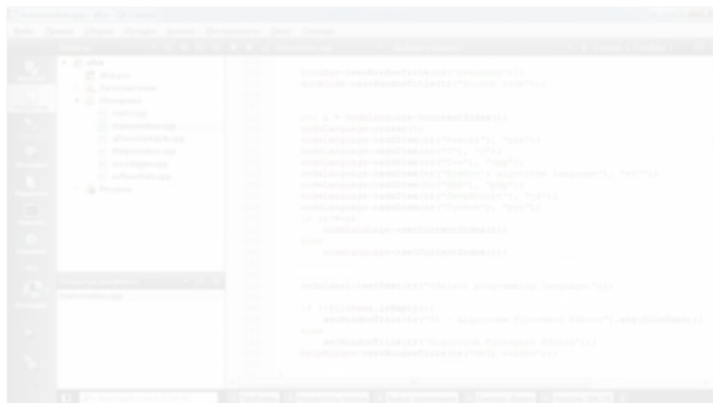


Рис. 3.1. Інтерфейс середовища розробки

Мови

Ви маєте можливість використовувати редактор для написання коду на мові Qt C++ або на мові декларативного програмування QML.

QML

Для створення дуже гнучкого інтерфейсу користувача з широким набором елементів QML ви можете використовувати мову QML. QML сприяє колаборації між розробниками та дизайнерами, допомагаючи створювати гнучкі та користувацькі інтерфейси для портативних пристроїв, таких як мобільні телефони, медіаплеєри, неттопи і нетбуки.

QML представляє собою розширення JavaScript, що надає механізм декларативної збірки дерева об'єктів з елементів QML. Це розширення поліпшує інтеграцію між JavaScript і існуючою системою Qt, яка базується на QObject, додаючи підтримку автоматичного зв'язування властивостей і забезпечуючи мережеву прозорість на рівні мови.

Цілі

Qt Creator надає підтримку для збірки і запуску додатків на Qt на різних платформах, таких як настільні комп'ютери (Windows, Linux і Mac OS) і мобільні пристрої (Symbian, Maemo і MeeGo). Налаштування збірки дозволяють швидко перемикатися між різними цілями збірки. Коли ви збираєте додаток для мобільного пристрою, підключеного до вашого комп'ютера, Qt Creator генерує пакет установки, встановлює його на пристрої і запускає його.

Ви можете відправити пакет установки в Ovi Store. Пакети для пристроїв Symbian повинні бути підписані.

Інструменти

Qt Creator вбудований із набором корисних інструментів, таких як системи контролю версій та емулятор Qt.

Системи управління версіями

Рекомендованим способом для розробки проекту є використання системи управління версіями. Для доступу до ваших сховищ Qt Creator використовує консольні клієнти систем управління версіями. Підтримуються наступні системи управління версіями:

- Git
- Subversion
- Perforce
- CVS
- Mercurial

Доступні функції в Qt Creator можуть варіюватися залежно від системи контролю версій. Основні можливості доступні для всіх підтримуваних систем, включаючи порівняння файлів з останньою версією у сховищі, відображення відмінностей, перегляд історії версій і деталей змін, анотацію файлів, а також можливість застосовувати та скасовувати зміни.

Емулятор Qt

Для перевірки додатків на Qt призначених для мобільних пристроїв в схожому оточенні, ви можете використовувати емулятор Qt. Ви можете змінити інформацію пристрою про його налаштуваннях і оточенні.

Емулятор Qt встановлюється як частина Nokia Qt SDK. Після установки ви можете вибрати його в якості мети збірки в Qt Creator.

Відладчики

Qt Creator може допомогти вам з налагодженням ваших додатків. Він надає інтерфейси до GNU Symbolic Debugger (gdb) і Microsoft Console Debugger (CDB) для налагодження звичайних додатків на C++ і внутрішні відладчики для Java Script. Це включає можливість підключити мобільні пристрої до свого комп'ютера і налагоджувати запущені на них програми.

Qt Creator відображає сиру інформацію, надану відладчиками, явним і лаконічним чином з метою спростити процес налагодження наскільки можливо без обмеження можливостей відладчиків.

3.2. Алгоритм програми

Загальні принципи створення програм через блок-схеми представлені на рис. 3.2. Пропонована технологія не зачіпає весь процес конструювання програмного забезпечення, розглядається та частина діяльності, розробка якої не може обійтися без участі програміста: проектування та кодування.



Рис. 3.2. Технологія розробки програм

Запропонована технологія була використана для реалізації системи синтезу блок-схем окремих модулів програми, яка автоматично генерує текст програми на обраній мові програмування.

Керуючі структури. Щодо керуючих структур, відповідно до методології структурного програмування будь-яка програма може бути розглянута як структура, побудована з трьох основних типів конструкцій:

Послідовність - виконання операцій одноразово в порядку, визначеному в тексті програми.

Розгалуження - виконання однієї з двох або більше операцій, залежно від заданої умови.

Цикл - багаторазове виконання операції до тих пір, поки задовольняється певна умова.

Конструювання блок-схем модулів програми досягається шляхом вибору комбінацій конструкцій, відповідних обмеженому безлічі керуючих структур.

Затвердження. Для побудови будь-якої форми управління, яка може знадобитися при побудові блок-схеми програми, досить три базові конструкції: слідування, розгалуження і цикл.

Спочатку схема складається з двох блоків - початку і кінця. При натисканні кнопки певного блоку, програма перемикається в режим вставки. В режимі вставки всі доступні точки вставки T підсвічуються червоним кольором, а точки вставки, яка знаходиться безпосередньо під курсором синім. При кліку по точці вставки на місці цієї точки з'являється блок.

У програмі позиції блоків і точок вставки розраховуються кожен раз заново при зміні схеми. Точки вставки так само генеруються кожен раз заново при зміні схеми. Для кожної гілки за допомогою рекурсивної процедури, яка обходить всі вкладені блоки, розраховується її загальний розмір, тобто розмір з урахуванням усіх вкладених блоків. Потім блоки розміщуються на схемі, вирівнювання йде зверху вниз, зліва направо.

При видаленні блоку визначається гілка, якій належить цей блок, і зі списку елементів L_1, L_2, \dots, L_n цієї гілки видаляється відповідний блок L_i . Координати блоків схеми розраховуються заново.

Редактор надає можливість конвертувати блок-схему в вихідний текст програми для різних мов програмування, таких як Pascal, C/C++, або Алгоритмічна мова. Редактор блок-схем дозволяє експортувати зображення схеми в різні графічні формати: BMP, JPEG, PNG, TIFF, ICO, PPM, XBM, XPM, SVG. Програма написана на мові C++ на основі бібліотеки Qt 4.

Можливості

- генерація вихідного коду на основі блок-схеми в різні мови програмування;
- автоматичне розміщення блоків на схемі;
- експорт схеми в популярні растрові формати;
- експорт схем в векторний формат SVG;
- можливість роботи з буфером обміну;

- масштабування блок-схеми;
- підтримка декількох мов перекладів;
- конвертація блок-схеми в початковий код на кількох мовах програмування.

3.3. Опис середовища для розв'язання задач з програмування

Графічний інтерфейс середовища для розв'язання задач з програмування представлений на рис. 3.3.



Рис. 3.3. Графічний інтерфейс програми

На головній формі програми розташовані такі компоненти, що визначають функціональність програми:

1. головне меню;
2. панель інструментів і операцій;
3. вікно з вихідним кодом програми, відповідної побудованої схемою.

Головне меню програми має три розділи «Файл», «Правка» і «Довідка» програми.

Розділ «Файл» представлено на рисунку 3.4.

Рис. 3.4. Розділ «Файл» головного меню програми

На панелі інструментів програми розташовані блоки алгоритмічних структур рис.3.5.

Рис. 3.5. Панель інструментів програми

Редактор надає можливість конвертувати блок-схему в вихідний текст програми для різних мов програмування, включаючи Pascal, C/C++, Алгоритмічну мову, PHP, JavaScript і Python.

Для вставки нового блоку потрібно:

1. на панелі інструментів натиснути кнопку потрібного блоку;
2. на схемі з'являться точки для вставки нового блоку (рис. 3.6).

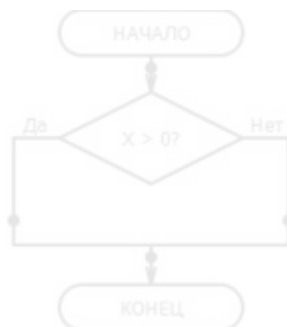


Рис. 3.6. Режим вставки нового блока

Клікнути по потрібній точці вставки. Вставиться новий блок і програма сама вирівняє всі блоки схеми (рисунок 3.7):



Рис. 3.7. Результат вставки нового блоку

По ходу редагування блок схеми, в правій частині вікна буде оновлюватися код схеми.



Рис. 3.8. Код схеми

Головним достоїнством створеної нами програми є простота створення схем, не потрібно відкривати безліч вікон і спливаючих меню, для того, щоб створити схему. Схема створена, тепер ви можете експортувати створену схему в популярні растрові формати або в векторний формат SVG.

3.4. Висновки до розділу 3

Розроблено систему візуального проектування та автоматичної генерації програмного коду на основі блок-схем. Візуальний редактор дозволяє експортувати блок-схему у вихідний текст програми для різних мов програмування, таких як Pascal, C/C++, Алгоритмічна мова. Також редактор блок-схем надає можливість експортувати зображення схеми в різні графічні формати, такі як BMP, JPEG, PNG, TIFF, ICO, PPM, XBM, XPM, SVG. Програма реалізована на мові C++ з використанням бібліотеки Qt 4.

ВИСНОВКИ

У магістерській роботі досліджено та розроблено систему візуального проєктування та автоматичної генерації програмного коду на основі блок-схем.

Розроблений прототип інтерактивної побудови блок-схем дозволяє шляхом маніпулювання графічними об'єктами редагувати логіку програм, автоматично формувати програмні коди в текстовому вигляді. Пропонується розглядати вихідний текст як внутрішнє представлення програми, а блок-схему як зовнішнє уявлення.

Для досягнення визначеної мети були вирішені такі завдання:

- Оцінено та проаналізовано існуючі системи візуального проєктування та автоматичної генерації програмного коду на основі блок-схем. У проаналізованих програм-редакторів схем є недоліки - вони не пристосовані для українських графічних стандартів. Жодна з розглянутих програм не має функцію автоматичного розміщення блоків і не дозволяє отримати код програми відповідної побудованої блок-схемою.

- Проведено аналіз процесу розробки програмного забезпечення системи трансляції програмного коду між графічним та текстовим представленням. Проведено моделювання та описана архітектура розробленого додатка. Надано математичний опис моделі системи перекладу програмного коду між графічним та текстовим представленням.

- Розроблено та реалізовано систему візуального проєктування та автоматичної генерації програмного коду на основі блок-схем. Розроблений прототип інтерактивної системи для створення блок-схем дозволяє користувачеві редагувати логіку програми за допомогою маніпулювання графічними об'єктами. Також автоматично створюються програмні коди для різних мов програмування, таких як Pascal, C / C ++, Алгоритмічна мова,

PHP, JavaScript, Python. Програма написана на мові C ++ на основі бібліотеки Qt 4.

Запропоновану систему можна використовувати як для навчальних цілей, так і в професійній сфері програмної інженерії. Для подальшого дослідження, корисною функцією для розроблюваної системи представляється інтерактивний зв'язок між графічною схемою і згенерованим кодом. Вона дозволяє визначити фрагмент коду, який відповідає зазначеним блокам графічної схеми. Така функція дозволить істотно прискорити пошук необхідного місця в програмному коді. Також корисною буде функція застосування відповідних змін до графічної схемою при їх внесення в згенерований код. Слід зазначити, що процес створення графічної схеми природним чином (малюючи на папері) набагато зручніше в порівнянні з маніпулюванням зумовленими прототипами блоків і подальшим їх налаштуванням за допомогою додаткових вікон властивостей. Тому, система розпізнавання образів описаних діаграм з подальшою трансляцією їх в стандартні примітиви (в термінах базової системи) є ефективним доповненням з точки зору користувача.

Схожість

Джерела з Інтернету

184

| | | |
|----|---|-----------------|
| 1 | https://shron1.chtyvo.org.ua/Yavorskyi_Nazarii/Laboratornyi_praktykum_z_dystsypliny_Alhorytmizatsiia_ta_prohramuvannia.pdf | 2.71% |
| 2 | http://ni.biz.ua/5/5_5/5_55969_opisanie-simvolov.html | 2.42% |
| 3 | http://ni.biz.ua/11/11_7/11_77566_pravila-primeneniya-simvolov-i-vipolneniya-shem.html | 6 джерел 2.39% |
| 4 | http://ni.biz.ua/11/11_8/11_8648_blok-shemi-algoritmov.html | 2 джерела 2.32% |
| 5 | http://ir.nmu.org.ua/jspui/bitstream/123456789/2719/1/%D0%9D%D0%A2%D0%91453510.pdf | 2.24% |
| 6 | http://um.co.ua/1/1-3/1-36342.html | 3 джерела 2.24% |
| 7 | http://dspace.kntu.kr.ua/jspui/bitstream/123456789/7615/1/%d0%a2%d0%9f%d0%9f%d0%a1-%d0%ba%d1%83%d1%80.docx | 12 джерел 2.1% |
| 8 | http://dspace.kntu.kr.ua/jspui/bitstream/123456789/11444/1/BD_%d0%9cet_rek_kurs_rob_2021.pdf | 12 джерел 2.1% |
| 9 | https://ukrdoc.com.ua/text/40172/index-1.html?page=3 | 2 джерела 1.96% |
| 10 | http://ni.biz.ua/7/7_13/7_134965_opredelenie-kolichestva-informatsii-edinitsi-izmereniya-informatsii.html | 1.92% |
| 11 | https://uadoc.zavantag.com/text/38092/index-6.html | 1.74% |
| 12 | https://topuch.ru/kontrolna-robota-z-disciplini-algoritmizaciya-ta-tehnologiyi/index.html | 1.7% |
| 13 | http://uadoc.zavantag.com/text/33472/index-10.html | 3 джерела 1.64% |
| 14 | http://reposit.nupp.edu.ua/bitstream/PoltNTU/10230/1/401-%d0%a2%d0%9a%20%d0%9c%d1%96%d0%bb%d1%8f%d1%80.docx | 1.64% |
| 15 | http://eir.zp.edu.ua/bitstream/123456789/1757/1/Korniienko_Guidelines_for_the.pdf | 3 джерела 1.52% |
| 16 | https://studfile.net/preview/9720248/page:3 | 3 джерела 1.5% |
| 17 | http://dspace.nuft.edu.ua/jspui/bitstream/123456789/8170/1/Vudu%20ta%20formu%20predstavleniya.pdf | 3 джерела 1.49% |
| 18 | http://nm2.univd.edu.ua/download/138503 | 6 джерел 1.45% |
| 19 | http://ni.biz.ua/3/3_7/3_75883_strukturi-algoritmov.html | 4 джерела 1.4% |
| 20 | https://jak.koshachek.com/articles/grafichne-predstavleniya-algoritmu.html | 7 джерел 1.39% |

| | | | |
|----|---|-----------|-------|
| 21 | http://studcon.org/rozrobka-algorytmu-funkcionuvannya-gnuchkoyi-avtomatyzovanoyi-dilnyci-gad?page=1 | 7 джерел | 1.37% |
| 22 | https://evnuir.vnu.edu.ua/bitstream/123456789/21329/1/%d0%9e%d0%90%d1%82%d0%9f_%d0%92%d0%a1%d0%95 | 2 джерела | 1.3% |
| 23 | http://um.co.ua/4/4-16/4-168838.html | 3 джерела | 1.29% |
| 24 | https://studfile.net/preview/9720248 | | 1.15% |
| 25 | http://eir.zntu.edu.ua/bitstream/123456789/1728/1/Korniienko_Methodical_pointing.pdf | 2 джерела | 1.06% |
| 26 | https://uadoc.zavantag.com/text/35027/index-6.html | 3 джерела | 1.02% |
| 27 | http://um.co.ua/6/6-13/6-139007.html | | 1.02% |
| 28 | http://ni.biz.ua/2/2_4/2_4852_graficheskoe-izobrazhenie-tehnologicheskogo-protssesa.html | 2 джерела | 1% |
| 29 | https://ela.kpi.ua/handle/123456789/48125 | 11 джерел | 0.97% |
| 30 | https://present5.com/algoritm-algoritm-zazdalegid-zadane-zrozumile-i-tochne | | 0.86% |
| 31 | http://www.institut.cn.ua/uploads/files/m_7/Vstup-do-fahu-Metodichn-vkazvki-do-lr.doc | | 0.82% |
| 32 | https://studfile.net/preview/7291614/page:9 | | 0.8% |
| 33 | http://ni.biz.ua/9/9_12/9_129192_blok-shema-resheniya-zadachi.html | | 0.79% |
| 34 | https://ecopy.posibnyky.vntu.edu.ua/txt/2018/Kaplun_ost-b_lukichov_mv_kr_tex-progr_p013.pdf | 8 джерел | 0.79% |
| 35 | http://um.co.ua/6/6-13/6-139004.html | 3 джерела | 0.77% |
| 36 | http://uadoc.zavantag.com/text/1719/index-9.html | | 0.77% |
| 37 | http://ir.nmu.org.ua/bitstream/handle/123456789/149254/CD862.pdf?isAllowed=y&sequence=1 | 4 джерела | 0.74% |
| 38 | https://studfile.net/preview/9720248/page:2 | | 0.72% |
| 39 | https://infopedia.su/16xdb7a.html | 2 джерела | 0.71% |
| 40 | http://document.kdu.edu.ua/metod/2021_3383.pdf | | 0.71% |
| 41 | https://ukrdoc.com.ua/text/14266/index-1.html?page=7 | 2 джерела | 0.67% |
| 42 | https://ela.kpi.ua/handle/123456789/33651 | 2 джерела | 0.64% |

| | | | |
|----|---|-----------|-------|
| 43 | http://citroen.thebest-on.com/zahist-u-gjek | 2 джерела | 0.64% |
| 44 | https://studopedia.info/1-29506.html | 5 джерел | 0.62% |
| 45 | https://lagao.at.ua/Lectur/Tema_8.pdf | | 0.61% |
| 46 | http://dspace.puet.edu.ua/handle/123456789/12852 | 3 джерела | 0.61% |
| 47 | https://studfile.net/preview/5775181/page:7 | 2 джерела | 0.56% |
| 48 | http://um.co.ua/5/5-1/5-120303.html | 5 джерел | 0.56% |
| 49 | http://ni.biz.ua/9/9_2/9_23284_soderzhanie-otcheta-po-rabote.html | | 0.56% |
| 50 | https://nadc.org.ua/docs/2021/DP/metod_posibnyk_dp2021_KN.pdf | 4 джерела | 0.54% |
| 51 | https://ela.kpi.ua/bitstream/123456789/57383/1/Magisterska_dysertatsiia_organizatsiia_vykonannia_zahystu_vymohy | 2 джерела | 0.52% |
| 52 | http://uk.wikipedia.org/wiki/Схема_(інформатика) | 2 джерела | 0.49% |
| 53 | https://studfile.net/preview/16420706 | 2 джерела | 0.42% |
| 54 | https://dut.edu.ua/repozitorii/ipz/2022/%d0%9f%d0%94%2042/%d0%a4%d1%80%d0%b0%d0%bd%d1%86%d0%b5%d0%b2%... | | 0.42% |
| 55 | https://studopedia.net/3_74290_osnovni-algoritmi-funktsionuvannya.html | | 0.39% |
| 56 | http://ni.biz.ua/5/5_10/5_109419_opisanie-shem.html | | 0.39% |
| 57 | http://ni.biz.ua/11/11_7/11_77565_opisanie-shem-espd.html | | 0.36% |
| 58 | http://ni.biz.ua/5/5_10/5_109420_opisanie-simvolov.html | | 0.3% |
| 59 | http://um.co.ua/9/9-2/9-27814.html | | 0.29% |
| 60 | http://ir.nmu.org.ua/bitstream/handle/123456789/1968/%d0%9d%d0%a2%d0%91452176.pdf?isAllowed=y&sequence=1 | 2 джерела | 0.27% |
| 61 | https://studfile.net/preview/5994725/page:3 | 4 джерела | 0.26% |
| 62 | https://ua-referat.com/uploaded/rishennya-shodo-podaleshoyi-rozrobki-rozriznyayute-taki-tipi-p/index4.html | | 0.26% |
| 63 | https://ukrdoc.com.ua/text/14266/index-1.html?page=6 | | 0.24% |
| 64 | https://studfile.net/preview/5686313/page:68 | 3 джерела | 0.22% |

| | | |
|----|---|-------|
| 65 | http://ni.biz.ua/2/2_7/2_74767_lektsiya---algoritmi-i-algoritmizatsiya.html | 0.16% |
| 66 | https://studopedia.net/1_40614_vkazivki-do-vikonannya-referativ-rozrahunkovo-grafichnih-robot-kurovih-robot-proektiv.html | 0.14% |
| 67 | https://ela.kpi.ua/handle/123456789/34869 | 0.11% |
| 68 | http://lib.htei.org.ua/sites/default/files/83/2017/metodichni_vkazivki_do_napisannya_vipusknih_kvalifikaciyних_robot_dlya_zdob... | 0.11% |
| 69 | http://ni.biz.ua/9/9_7/9_74108_primer-.html | 0.11% |

Вилучення

Вилучення

30

| | |
|---|----------------|
| https://ua-referat.com/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B... | 0.1% |
| http://antibotan.com/file.html?work_id=523323 | 0.1% |
| https://prog.bobrodobro.ru/42002 | 14 джерел 0.1% |
| https://studopedia.ru/18_65196_ponyattya-obiektu-i-klasu.html | 0.1% |
| http://citm.ho.ua/Dist/Txt/course186.pdf | 2 джерела 0.1% |
| https://knteu.kiev.ua/file/MTc=/adb099cfb596b8fe5a685807aa58dd2e.pdf | 2 джерела 0.1% |
| https://studfile.net/preview/7786740/page:7 | 2 джерела 0.1% |
| https://studfile.net/preview/5219410 | 0.1% |
| http://dspace.cuspu.edu.ua/jspui/bitstream/123456789/2673/1/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B8%20%D1%9... | 0.1% |
| http://eprints.cdu.edu.ua/1481/1/pro.pdf | 0.1% |
| http://kafelec.nau.edu.ua/Materialu/Diplomne_proektuvannya_Bakalavr_2013_traven.pdf | 0.1% |
| https://uchika.in.ua/metodichni-vkazivki-dlya-studentiv-specialenosti-170101-bezpek.html?page=10 | 0.1% |
| https://studfile.net/preview/8940063/page:8 | 0.1% |
| http://dspace.mdu.edu.ua/jspui/bitstream/123456789/292/3/%d0%a3%d1%81%d1%82%d0%b5%d0%bd%d0%ba%d0%be%2c%20... | 0.1% |