Зміст

Введення

Мова програмування С була розроблена в 1972 р. Деннісом Рітчі, який є одним з авторів операційної системи UNIX. Мова С зручна для програмування системних задач. Поява мікрокомп'ютерів закріпила позиції мови С. Було створено близько 30 його нових компіляторів, а після проведення Американським національним інститутом стандартів (American National Standards Institute) робіт по стандартизації в області програмування почали розроблятися компілятори, що відповідали опублікованому весною 1986 р. проекту стандарту.

Першим компілятором по стандарту ANSI являлась система Турбо С версії 1.0 фірми Borland International. Ця система, складається з компілятора мови С, пов'язаного з ним редактора, компоновщика і бібліотек яка забезпечувала інтегровану операційну оболонку, визначальними параметрами є висока швидкість компіляції, висока якість згенерованого коду та невеликий об'єм оперативної пам'ять. Мова С універсальна, але найбільш ефективне її використання в задачах системного програмування - розробки трансляторів, операційних систем, інструментальних засобів.

Основними перевагами мови С вважаться висока передача написаних на ній програм між комп'ютерами з різною архітектурою, між різними операційними середовищами.

Окремі особливості мови С:

- в мові С реалізовані окремі операції низького рівня (а саме операції над бітами). Окремі з таких операцій напряму відповідають машинним командам;

- мова С підтримує механізм показчиків на змінні і функції;

- базові типи даних мови С відображають тіж об'єкти, з якими потрібно

мати справу програмі на мові Асемблера,- байти, машинні слова, символи, строки;

- як в ніякій іншій мові програмування високого рівня в мові С накладаються лише незначні обмеження на перетворення типів даних;

- не дивлячись на ефективність і потужність конструкції мови С, він відносно малий по об'єму, але в системне оточення мови С входить бібліотека стандартних функцій, в який реалізовані вбудовані оператори введення - виведення, динамічного розподілу пам'ять, управління процесами.

ЗАГАЛЬНА ЧАСТИНА

1. Загальна частина. Мова програмування C++ Builder

1.1. Основні поняття C++ Builder

Середовище розробки C++ Builder являє собою SDI-додаток, головне вікно якого містить налагоджувану інструментальну панель і палітру компонентів. Крім цього, за умовчанням при запуску C++ Builder з'являються вікно інспектора об'єктів і форма нової програми. Під вікном форми знаходиться вікно редактора коду. Форми є основою додатків C++ Builder. Створення інтерфейсу додатка полягає в додаванні в вікно форми елементів об'єктів C++ Builder, званих компонентами. Компоненти C++ Builder розташовуються на палітрі компонентів, виконаної у вигляді багато сторінкового блокнота. Важлива особливість C++ Builder полягає в тому, що він дозволяє створювати власні компоненти і налаштовувати палітру компонентів, а також створювати різні версії палітри компонентів для різних проектів. Компоненти поділяються на видимі (візуальні) і невидимі (не-візуальні). Візуальні компоненти з'являються під час виконання точно так само, як і під час проектування. Прикладами є кнопки і редаговані поля. Не візуальні компоненти з'являються під час проектування як піктограми на формі. Вони ніколи не видно під час виконання, але володіють певною функціональністю (наприклад, установ підтримують доступ до даних, викликають стандартні діалоги Windows і ін.) Для додавання компонента у форму можна вибрати мишею потрібний компонент у палітрі і клацнути лівою клавішею миші в потрібному місці проектованої форми. Компонент з'явиться на формі, і далі його можна переміщати, змінювати розміри та інші характеристики. Кожен компонент C++ Builder має три різновиди характеристик: властивості, події і методи.

Якщо вибрати компонент з палітри і додати його до форми, інспектор об'єктів автоматично покаже властивості та події, які можуть бути використані з цим компонентом. У верхній частині інспектора об'єктів є список, що випадає, дозволяє вибирати потрібний об'єкт з наявних на формі.

Властивості є атрибутами компонента, що визначають його зовнішній вигляд і поведінку. Багато властивостей компонента у колонці властивостей мають значення, що встановлюється за умовчанням (наприклад, висота кнопок). Властивості компонента відображаються на сторінці властивостей (Properties). Інспектор об'єктів відображає опубліковані (published) властивості компонентів. Крім published-властивостей, компоненти можуть і частіше за все мають загальні (public), опубліковані властивості, які доступні тільки під час виконання програми. Інспектор об'єктів використовується для установки властивостей під час проектування. Список властивостей розташовується на сторінці властивостей інспектора об'єктів. Можна визначити властивості під час проектування або написати код для видозміни властивостей компонента під час виконання програми.
При визначенні властивостей компонента під час проектування потрібно вибрати компонент на формі, відкрити сторінку властивостей в інспектора об'єктів, вибрати визначається властивість і змінити його за допомогою редактора властивостей.

Сторінка подій (Events) інспектори об'єктів показує список подій, які розпізнаються компонентом. Кожен компонент має свій власний набір обробників подій. У C++ Builder слід писати функції, звані обробники подій, і пов'язувати події з цими функціями. Створюючи обробник тієї чи іншої події, ви доручаєте програмі виконати написану функцію, якщо ця подія відбудеться.

Для того щоб додати обробник подій, потрібно вибрати на формі за допомогою миші компонент, якому необхідний обробник подій, потім відкрити сторінку подій інспектора об'єктів і двічі клацнути лівою клавішею

миші на колонці значень поруч з подією, щоб змусить C++ Builder згенерувати прототип обробника подій і показати його в редакторі коду. При цьому автоматично генерується текст порожній функції, і редактор відкривається в тому місці, де слід вводити код. Курсор позиціонується всередині операторних дужок {... }. Далі потрібно ввести код, який повинен виконуватися при настанні події. Оброблювач подій може мати параметри, які вказуються після імені функції в круглих дужках.

Метод є функцією, яка пов'язана з компонентом, і яка оголошується як частина об'єкту. Створюючи обробники подій, можна викликати методи, використовуючи наступну нотацію: ->, наприклад: Edit1-> Show (); Відзначимо, що при створенні форми пов'язані з нею модуль і заголовки-ний файл з розширенням *. h генеруються обов'язково, тоді як при створенні нового модуля він не зобов'язаний бути пов'язаний з формою (наприклад, якщо в ньому містяться процедури розрахунків). Імена форми і модуля можна змінити, причому бажано зробити це відразу після створення, поки на них не з'явилося багато посилань в інших формах і модулях.

Файли, що утворюють додаток - форми і модулі - зібрані в проект. Менеджер проектів показує списки файлів і модулів програми і дозволяє здійснювати навігацію між ними. Можна викликати менеджер проектів, вибравши пункт меню View / Project Manager. За замовчуванням знову створений проект отримує ім'я Project1.cpp. За замовчуванням проект спочатку містить файли для однієї форми і вихідного коду одного модуля. Однак більшість проектів міс-жать кілька форм і модулів. Щоб додати модуль або форму до проекту, потрібно клацнути правою кнопкою миші і вибрати пункт New Form з контекстного меню. Можна також додавати існуючі форми і модулі до проекту, використовуючи кнопку Add контекстного меню менеджера проектів і обираючи модуль або форму, яку потрібно додати. Форми і модулі можна видалити в будь-який момент протягом розробки проекту. Однак, через те, що форма пов'язані завжди з

модулем, не можна видалити одне без видалення іншого, за винятком випадку, коли модуль не має зв'язку з формою. Видалити модуль з проекту можна, використовуючи кнопку Remove менеджера проектів. Якщо вибрати кнопку Options у менеджері проектів, відкриється діалогова панель опцій проекту, в якій можна вибрати головну форму додатка, визначити, які форми будуть створюватися динамічно, які параметри компіляції модулів (в тому числі створених в Delphi, так як C + + Builder може включати їх у проекти) та компонування. Важливим елементом середовища розробки C + + Builder є контекстне меню, що з'являється при натисканні на праву клавішу миші і пропонуючи швидкий доступ до найбільш часто використовуваних команд. C + + Builder володіє вбудованою системою контекстно-залежної допомоги, доступної для будь-якого елемента інтерфейсу і є великим джерелом довідкової інформації про C + + Builder.

1.2. Об'єктно-орієнтоване програмування

Об’єктно-орієнтовне програмування – методологія, яка концентрується більше на зв'язках між об'єктами, ніж на деталях реалізації. Ефективність ООП проявляється тільки при утворенні і застосуванні груп зв'язаних між собою об'єктів. Такі групи називають ієрархіями класів. Розвиток цих ієрархій класів є основою в діяльності ООП. Об’єктно-орієнтований підхід заснований на систематичному використанні моделей для мовно-незалежної розробки програмної системи. Об’єктно-орієнтований підхід допомагає справитися з такими складними проблемами як зменшення складності програмного забезпечення, підвищення надійності програмного забезпечення, забезпечення можливості модифікації окремих компонентів програмного забезпечення без зміни інших його компонентів, забезпечення можливості повторного використання окремих компонентів програмного забезпечення. Однією з переваг об'єктно-орієнтованого програмування, яке упускається часто з виду, — це концепція захоплення ресурсів при

ініціалізації, що належить Б’ярну Страуструпу. Конструктори в C++ викликаються при створенні об'єкта, а деструктори — при його видаленні, оскільки він стає більш не потрібний. Об'єкти, що вимагають ресурсів, такі як файли або блоки пам'яті повинні успішно захоплювати потрібні ресурси ще до того, як їх можна буде вважати дійсно створеними.

У такий спосіб в об'єктно-орієнтованому програмуванні досягається одна з заповітний цілей – якщо об'єкт створений, то можна бути упевненим у тому, що він створений цілком а не залишається в якому-небудь нестійкому половинчатому стані.

C++ оснований на трьох основних концепціях, які називаються: інкапсуляція, поліморфізм і наслідування.

1.2.1. Інкапсуляція

Інкапсуляція – об'єднання в єдиному об'єкті даних і кодів, що оперують з цими даними. У термінології ООП дані називаються членами даних (data members) об'єкта, а коди - об'єктними методами або функціями-членами (methods, member functions).Інкапсуляція дозволяє в максимальному ступені ізолювати об'єкт від зовнішнього оточення. Вона суттєво підвищує надійність розроблювальних програм, тому що локалізовані в об'єкті функції обмінюються з програмою порівняно невеликими об'ємами даних, причому кількість і тип цих даних зазвичай ретельно контролюються. У результаті заміна або модифікація функцій і даних, інкапсульованих в об'єкт, як правило, не тягне за собою погано простежуються наслідків для програми в цілому (з метою підвищення захищеності програм в ООП майже не використовуються глобальні змінні).Іншим важливим наслідком інкапсуляції є легкість обміну об'єктами, перенесення їх з однієї програми в іншу. Простота і доступність принципу інкапсуляції ООП стимулює програмістів до розширення Бібліотеки Візуальних Компонент, що входить до складу C + + Builder.

1.2.2. Спадкування

Спадкування - механізм дозволяє описати новий клас на основі вже існуючого. Новий клас називають дочірнім класом, нащадком, підкласом або похідним класом по відношенню до батьківського або базового класу. І найважливіше, що дозволяється підставляти батьківський клас об'єктом дочірнього класу.

Систематичне застосування об’єктно-орієнтованого підходу дозволяє розробляти добре структуровані, надійні в експлуатації, досить програмні системи, що модифікуються просто.

1.2.3. Поліморфізм

Поліморфізм - механізм дозволяє змінити реалізацію методів, тобто поведінку об'єкта в дочірніх класах. Змінні методи називаються віртуальними. Зокрема, якщо клас містить оголошення методу, реалізація якого передбачається тільки в дочірніх класах, то він називається абстрактним класом. Все це дозволяє зручно обробити об'єкти дочірніх класів через батьківський клас.

1.3. Оператори

1.3.1. Цикл for

У всіх процедурних мовах програмування цикли for дуже схожі. Проте в C цей цикл особливо гнучкий і потужний. Загальна форма оператора for наступна: for (ініціалізація; умова; прирощення) оператор;Цикл for може мати велику кількість варіацій. У найбільш загальному вигляді принцип його роботи наступний. Ініціалізація - це привласнення початкового значення змінної, яка називається параметром циклу. Умова являє собою умовний вираз, що визначає, чи слід виконувати оператор циклу (часто його

називають тілом циклу) в черговий раз. Оператор прирощення здійснює зміну параметра циклу при кожній ітерації. Ці три оператори (вони називаються також секціями оператора for) обов'язково розділяються крапкою з комою. Цикл for виконується, якщо вираз умова набуває значення TRUE. Якщо воно хоча б один раз прийме значення FALSE, то програма виходить з циклу і виконується оператор, наступний за тілом циклу for.

1.3.2. Цикл do…while

Структура do…while використовується для організації циклічного виконання оператора або сукупності операторів, званих тілом циклу, до тих пір, поки не виявиться порушеною деяка умова.

Структура працює таким чином. Виконується оператор тіла циклу. Потім обчислюється умова – вираз, який повинен повертати результат булева типу. Якщо вираз повертає true (не нульове значення), то повторюється виконання тіла циклу і після цього знову обчислюється вираз. Таке циклічне повторення циклу продовжується до тих пір, поки вираз, що перевіряється, не поверне false (нуль). Після цього цикл завершується і управління передається оператору наступному за структурою do…while.

Оскільки перевірка виразу здійснюється після виконання циклу, то цикл буде явно виконаний хоча б один раз, навіть якщо вираз відразу помилковий. З другого боку, програміст повинен бути упевнений що вираз рано чи пізно поверне false. Якщо цього не відбудеться то програма «зациклиться», тобто цикл виконуватиметься нескінченно. Іноді такі нескінченні цикли використовуються. Але в цьому випадку в середині тіла циклу повинна бути передбачене його переривання в якийсь момент наприклад, оператором break або іншими способами.

Звичайно оператора do доцільно використовувати для організації пошуку серед безлічі об’єктів такого, який володіє якоюсь певною властивістю. При чому на перед повинно бути відомо що безліч об’єктів не

порожньо тобто хоча б один об’єкт в ньому є. До того ж повинен бути критерій що дозволяє перевірити чи не є поточний об’єкт останнім. Тоді тіло циклу включає операторів переходу до нового об’єкту і якоїсь його обробки, а умова while включає перевірку, чи є об’єкт не останнім і чи відсутні в нього шукані властивості. Якщо об’єкт останній або шукані властивості знайдені, виконання циклу уривається. Якщо ж об’єкт не останній і шукані властивості у нього не знайдені, здійснюється перехід до наступного об’єкту.

Якщо безліч об’єктів, що перевіряються може бути порожнім, слід використовувати іншого оператора циклу – while. Якщо число повторень циклів на перед відомі краще застосувати оператора for. Перевірка наявності у рядку заданого фрагмента перевіряється функцією strstr. Закінчення файлу перевіряється функцією feof.

1.3.3. Цикл while

Оператор while використовується для організації циклічного виконання тіла циклу, поки виконується деяка умова.

Структура працює таким чином. Спочатку обчислюється умова, яка повинна повертати результат булева типу. Якщо вираз повертає true, то виконується оператор циклу, після чого знову обчислюється вираз, що визначає умову. Таке циклічне повторення виконання оператора і перевірки умови продовжується до тих пір, поки умова поверне false. Після цього цикл завершується і управління передається оператору, наступному за структурою while.

Оскільки перевірка виразу здійснюється перед виконання оператора циклу, то якщо умова відразу помилкова оператор не буде виконаний жодного разу.

Вираз рано чи пізно поверне false. Якщо цього не відбудеться то програма «зациклиться», тобто цикл виконуватиметься нескінченно. Іноді такі нескінченні цикли використовуються. Але в цьому випадку у середині

тіла циклу повинне бути передбачене його переривання в якийсь момент, наприклад оператором break, що перериває цикл або іншими способами.

Часто оператор while використовується для організації пошуку серед безлічі об’єктів такого який володіє якоюсь певною властивістю. При чому не виключається що безліч об’єктів може бути порожньою, тобто що не містить жодного об’єкту. До того ж повинен бути критерій, що дозволяє перевірити чи не є поточний об’єкт останнім. Тоді тіло циклу включає операторів переходу до нового об’єкту і якоїсь його обробки, а умова while включає перевірку чи є об’єкт не останнім і чи не володіє він шуканою властивістю. Якщо одна із цих умов порушується (об’єкт останній або має шукану властивість), виконання циклу уривається.

СПЕЦІАЛЬНА ЧАСТИНА

2. Спеціальна частина. Демонстрація програмного забезпечення

2.1. Обчислення елементарних функцій за допомогою степеневих рядів

Рішення багатьох математичних задач вимагає обчислення значень елементарних функцій для тих або інших значень аргументу. Для обчислення значення елементарних функцій за допомогою ЕОМ існують стандартні програми, реалізовуючи алгоритми для обчислення з потрібною точністю значення цих функцій.

Статичні ряди є важливим апаратом для табуляції елементарних функцій, оскільки їх застосування дозволяє звести задачу обчислень значень функції із заданою точністю до задачі обчислення многочлена.

Елементарні функції можна розкласти в статечній ряд за допомогою загальної формули Тейлора:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots \qquad (2.1.)$$

Звичайно в ній вважають a=0, завдяки чому виходить звичайний статечній ряд по ступенях x. Обриваючи ряд в потрібному місці, отримаємо многочлен, що наближає дану функцію.

Самими споживаними є статечні ряди для функції *ex,* cos *x,* sin *x,* які сходяться при будь-якому значенні x:

$$\begin{aligned}
e^x &= 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots, \\
\cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots, \\
\sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{n+1}}{2n+1!} + \ldots, \\
\ln \frac{1+x}{1-x} &= 2(x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + \frac{x^{n+1}}{2n+1} + \ldots,)
\end{aligned} \qquad (2.2.)$$

При обчисленні елементарних функцій за допомогою статечних рядів

зручно користуватися рекурентними співвідношеннями, які дозволяють обчислювати черговий член ряду не посередньо, а через вже обчислені попередні члени. Рекурентним співвідношенням називають рівність, що зв'язує між собою два або декілька сусідніх членів послідовності або ряду. За допомогою такої рівності можна визначити подальший член ряду через попередні.

Для приведених вище рядів рекурентні співвідношення можуть бути легко виведений безпосередньо. Простіше всього узяти відношення двох сусідніх членів. Розглянемо, наприклад, статечної ряд для функції *ex,* загальний член якого має вид an= xn/n!. За допомогою відношення подальшого члена до попереднього отримаємо.

$$\frac{a_{n+1}}{a_n} = \frac{x^{n+1} * n!}{(n+1)! * x^n} = \frac{x}{n+1}. \qquad (2.3.)$$

Таким чином, для двох сусідніх членів ряду одержуємо рекурентне співвідношення яке зручно для послідовного обчислення членів ряду.

$$a_n = a_{n-1} * (x/n), \qquad (2.4.)$$

При обчислення з рядами замінюємо суму ряду його частковою сумою, тобто обмежуємося кінцевим числом членів. Виникає практичне питання: скільки членів ряду потрібно зберегти, щоб погрішність, що виходить, не перевищувала заданій.

Якщо члени ряду убувають достатньо швидко і притому із самого початку, то вигідно використовувати знакозмінний ряд, погрішність якого легко оцінюється. Сума знакозмінного ряду менше його першого члена (по абсолютній величині). Тому при заміні суми такого ряду його частковою сумою допускається погрішність, що не перевершує по модулю першого з відкинутих членів. Якщо ж члени ряду убувають поволі або убувають хоча і дуже швидко, але не спочатку, а перші члени ряду достатньо великі, фактична погрішність буде помітна більшій через погрішності віднімання перших великих членів. Наприклад, члени рядів для обчислення cos x і sin x

убувають достатньо швидко і ряди сходяться при будь-кому x. Проте при великих x (x>10) перші члени цих рядів досить швидко зростають, і обчислення cos x і sin x за допомогою цих рядів скрутно. Виникає така втрата точності при відніманні великих перших членів, яку не можна відшкодувати обчисленням великого числа доданків.

Для рядів з позитивними членами оцінка погрішності складніша і ніяких загальних методів оцінки, придатних для всіх рядів, запропонувати не можна.

Розглянуті статечні ряди зручний засіб для програмування елементарних функцій. Кожний з лав легко програмується як звичайний цикл, який можна писати і як арифметичний, і як ітераційний.

2.2. Виконання програмного забезпечення

Складаємо програму мовою C++ Builder, обчислення величини регулюючого впливу, вироблюваного комп'ютерною системою керування для якогось об'єкта, при заданій величині вхідного впливу згідно рівняння:

$$y = f(x) \tag{2.5.}$$

при:
$$f(x) = \frac{e^{2x} - 5e^{x}}{x^{4}} \tag{2.6.}$$

Для цього використовуємо оператор циклу do…while. Ми вводимо вхідні дані із клавіатури, вихідні дані (результати обчислень), відображає на екрані монітора й записує у файл, що містить програму. Процедуру обчислення робимо з наперед заданої оператором точністю, але не більше 30 ітерацій. Додатково забезпечимо можливість відображення номера останньої ітерації, величину останнього члена ряду, і отримаємо точність обчислень.

Складемо програму обчислення функції:

$$f(x) = \frac{e^{2x} - 5e^{x}}{x^{4}}$$

Спочатку обчислюємо e^{2x}:

$$e^{2x} = f(x) = \frac{2e^{2*0}}{1!}(x-0) + \frac{4e^{2*0}}{2!}(x-0)^2 + \frac{8e^{2*0}}{3!}(x-0)^3 + \frac{16e^{2*0}}{4!}(x-0)^4;$$

Розраховуємо факторіали:

$1! = 1$

$2! = 1*2 = 2$

$3! = 1*2*3 = 6$

$4! = 1*2*3*4 = 24$

Потім виконуємо наступні ітерації для функції e^{2x}:

$$1: e^{2x} \approx e^2 = 1 + \frac{2}{1!} = 1 + \frac{2}{1} = 3$$

$$2: e^{2x} \approx e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} = 1 + \frac{2}{1} + \frac{4}{2} = 3 + 2 = 5$$

$$3: e^{2x} \approx e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} + \frac{8}{3!} = 1 + \frac{2}{1} + \frac{4}{2} + \frac{8}{6} = 5 + 1.33 = 6.33$$

$$4: e^{2x} \approx e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} + \frac{8}{3!} + \frac{16}{4!} = 2 + \frac{2}{1} + \frac{4}{2} + \frac{8}{6} + \frac{16}{24} = 6.33 + 0.66 = 6.99$$

Знаходимо функцію для $5e^x$:

$$5e^x = f(x) = 5 + \frac{5e^0}{1!}(x-0) + \frac{5e^0}{2!}(x-0)^2 + \frac{5e^0}{3!}(x-0)^3 + \frac{5e^0}{4!}(x-0)^4;$$

Потім виконуємо наступні операції для функції $5e^x$:

$$1: 5e^x \approx 5 + \frac{1}{1!} = 5 + \frac{1}{1} = 6$$

$$2: 5e^x \approx 5 + \frac{1}{1!} + \frac{1}{2!} = 5 + \frac{1}{1} + \frac{1}{2} = 6.5$$

$$3: 5e^x \approx 5 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} = 5 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} = 6.67$$

$$4: 5e^x \approx 5 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} = 5 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} = 1.711$$

Відповідне програмне забезпечення має вигляд:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>
#pragma hdrstop
#include "Unit1.h"
#include "math.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
float x;  // Вводимо змінну x(результат обчислень)
float y;  // Вводимо змінну y (показник степеня)
float d;  // Вводимо змінну d (кінцева точність розрахунку)
float y0; // Вводимо змінну y0

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
Button2->Enabled=false;//робимо невидимою кнопку Button2
}
//---------------------------------------------------------------------------
///Прописуємо події виникаючі при натисканні кнопку Button1
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//Якщо кнопка натиснута то:
switch (Button1->Tag)
      {
      case 0:
    Button2->Enabled=true;//стає видимою кнопка Button2
    Button2->Caption="Зберегти";//на Button2 з'являється напис "Зберегти"
    Button2->Tag=0;//Значеню Tag привласнюється значення 0
      break;
      }
//При натисненні кнопки:
//Вводимо змінні усередині процедури Button1Click
float r;  float rw;
float b;  float c;
float da; float dw;
float t;  float tw;
float e;  float ew;  float yw;
float dd; float xw;
//Задаємо початкові значення змінних:
r=1; rw=1;
b=30; //задаємо максимальне число ітерацій
da=0; dw=0;
x=1;  xw=1;
//Введення кінцевої точності ітерацій з об'єкту Edit1
//Наприклад 1E-5 (відповідає 0,00001)
d=StrToFloat(Edit1->Text);
```

```
//Введення показника степення з об'єкту Edit2
//наприклад 2E-1 (відповідає 0,2)
y=StrToFloat(Edit2->Text);
yw=StrToFloat(Edit2->Text);
da=1; dw=1;//задаємо крок ітерацій
t=1;  tw=1;
y0=y;
e=0;  ew=0;
//організовуєм цикл розрахунку кількості ітерацій, підрахунку у в степені a2
//розрахунок факторіалу
do {
  da=da+1;  //привласнення в наступному циклі значенню da більше на 1
  r=da-1;  //кількість ітерацій (у циклі do...while перевірка дотримання
  //умов проводиться в кінці циклу вже після розрахунків нових значень)
  y=2*(y*y0);//розрахунок значень у в ступені a2
  t*=r;//розрахунок факторіалу
  e +=y/(y0*t);// розрахунок суми членів ряду без першого рівного 1
  x =e+1;//обчислення значення e в заданому ступені
  dd=y/(y0*t);//обчислення значення останнього члена ряду
  //Висновок результатів розрахунку на форму
  Label6->Caption="Результат:"+FloatToStrF(x,ffGeneral,7,2);
  //Завершення циклу, перевірка виконання умов: допустимого числа операцій
  //і точність розрахунку
  }
  while(da<=b^dd<=d);
  do {
  dw=dw+1;  //привласнення в наступному циклі значенню da більше на 1
  rw=dw-1;  //кількість ітерацій (у циклі do...while перевірка дотримання
```

```
  //умов проводиться в кінці циклу вже після розрахунків нових значень)
  yw=(yw*y0);//розрахунок значень у в ступені a2
  tw *=rw;//розрахунок факторіалу
  ew +=yw/(y0*tw);// розрахунок суми членів ряду без першого рівного 1
  xw = 5*(ew+1);//обчислення значення e в заданому ступені
  dd= yw/(y0*tw);//обчислення значення останнього члена ряду
  //Висновок результатів розрахунку на форму
  Label9->Caption="Результат:"+FloatToStrF(xw,ffGeneral,7,2);
  //Завершення циклу, перевірка виконання умов: допустимого числа операцій
  //і точність розрахунку
  }
  while(dw<=b^dd<=d);
  c = (x-xw)/y0*y0*y0*y0;
 Label10->Caption="Результат:"+FloatToStrF(c,ffGeneral,7,2);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
//Процедура запису і читання даних в і з файлу Test.txt
float d1,y1,x1;//Вводимо додаткові змінні
char s[10],s1[10];//оголошення рядка завдовжки 10 символів
strcpy(s,"Raschet");//Додавання до рядка s слова "Raschet"
int handle;//запис у файл
//Якщо кнопка Button2 натиснута 1 разів
switch (Button2->Tag)
      {
      case0:
      Button2->Tag=1;//значенню Tag привласнюється значення 1
```

```
        if ((handle=open("Test.txt",O_WRONLY|O_CREAT|O_BINARY))==-1)
        {
        //Повідомлення при помилці створення файлу:
        ShowMessage("Файл невдається створити");
        return;
        }
       write(handle,&d,sizeof(int));//запись d
       write(handle,&y0,sizeof(int));//запись y0
       write(handle,&x,sizeof(int));//запись x
       write(handle,s,strlen(s)+1);//запись рядка s
       close(handle);
       Button2->Caption="Читати";//висновок на кнопці Button2 напису "Читати"
       break;
       case 1: //Якщо Button2 натиснути повторно
       //процедура читання з файлу
       if((handle=open("Test.txt",O_RDONLY|O_BINARY))==-1)
       //повідомлення при помилці читання файлу
       ShowMessage("Файл не вдається відкрити");
       return;
       }
read(handle,&d1,sizeof(int));//читання d
read(handle,&y1,sizeof(int));//читання y1
read(handle,&x1,sizeof(int));//читання x1
read(handle,s1,strlen(s)+1);//читання рядка
//активація відображень на формі елементів Label1,2,3
Label1->Visible=true;
Label2->Visible=true;
Label3->Visible=true;
//вивід прочитаної інформації на форму
```

```
Label1->Caption="Записане число точність:"+FloatToStrF(d1,ffGeneral,7,2);
Label2->Caption="Записане число степінь:"+FloatToStrF(d1,ffGeneral,7,2);
Label3->Caption="Записане число результат:"+FloatToStrF(d1,ffGeneral,7,2);
close(handle);
}
//Процедура обробки події клік мишки у вікні Edit1 або Edit2
//Кнопка Button2 і вивід результатів повинні стати недоступні для Edit2:
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit2Change(TObject *Sender)
{
Button2->Enabled=false;
Label1->Visible=false;
Label2->Visible=false;
Label3->Visible=false;
}
//---------------------------------------------------------------------------
//Для Edit1
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
Button2->Enabled=false;
Label1->Visible=false;
Label2->Visible=false;
Label3->Visible=false;
}
//---------------------------------------------------------------------------
```

Висновки

В даному курсовому проекті проведена розробка програми визначення регулюючого впливу, що розраховується комп’ютерною системою керування. Програма складена за допомогою циклу do…while, містить об’єм 1,46 Мбайт.

Розроблена програма дозволяє розраховувати керуючий вплив заданими рівняннями. Програма дозволяє вводити вхідні данні з клавіатури, вихідні данні відображаються на екрані монітору та записуються у файл, що містить програму.

Розроблене програмне забезпечення у разі необхідності дозволяє проводити подальше вдосконалення та доробку. Та є відкритим програмним забезпеченням.

Додаток А

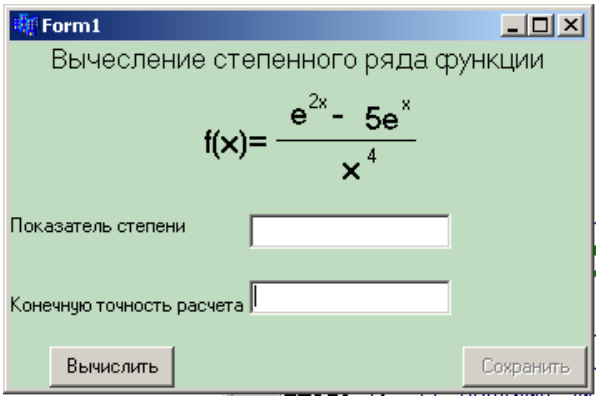

Рисунок 1. Вид вікна програми до початку обчислень

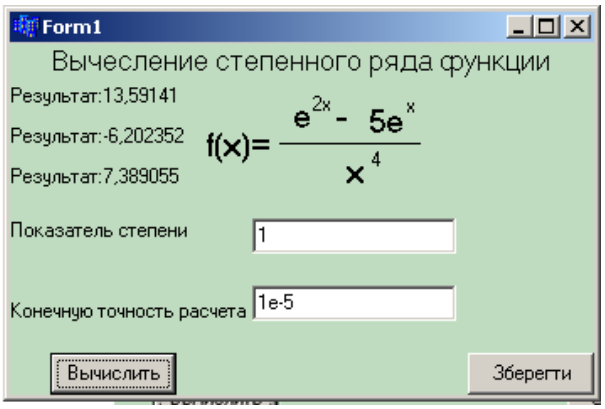

Рисунок 2. Вид вікна програми після обчислення

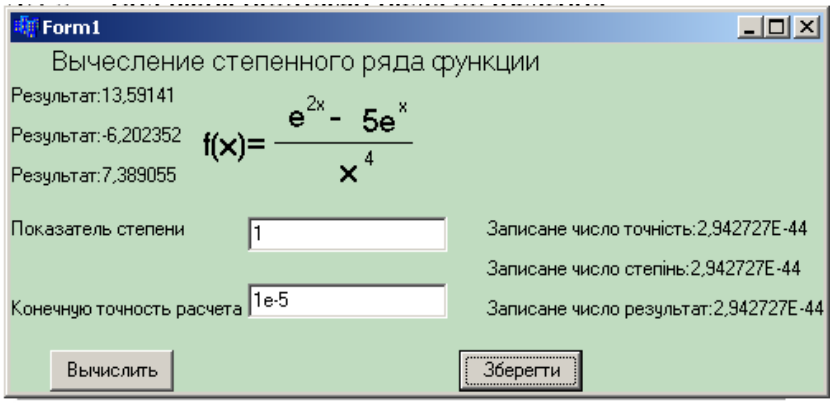

Рисунок 3. Вид вікна програми після читання з файлу результатів обчислень