Зміст

Введення

В наш час бурхливого розвитку інформаційних технологій ми постійно маємо справу з різного типу програмним забезпеченням. Інколи користувачу доводиться самому створювати чи змінювати програми.

Велика кількість персональних комп'ютерів знаходиться під управлінням операційної системи Windows. Через це створюються мови програмування спеціально інтегровані під неї. Раніше створенням програмного забезпечення займалася лише професіонали, що присвячували вивченню певних мов програмування роки. Єдиним середовищем розробки ПЗ для Windows був Borland C ++ for Windows, попередником якої була C – мова, за допомогою якої була створена Unix.

Бурхливий розвиток обчислювальної техніки з'явилася потреба в ефективних засобах розробки програмного забезпечення, що призвало до появи систем програмування орієнтованих на так звану швидку розробку, серед яких можна відмітити Microsoft Visual Basic і Borland Delphi. В їх основі лежить технологія візуального проектування і подійного програмування, суть яких лежить в тому, що середовище розробки бере на себе велику частину роботи, а програмісту залишається лише написання функцій обробки подій і створення вікон.

Успіх Delphi привів до того, що фірма Borland вирішила випустити середовище швидкої розробки, що підтримувало об'єктно-орієнтоване програмування – Borland C ++ Builder.

1.Загальний розділ

1.1. Основні поняття С+ + і С + + Builder

В 1972 Денісом Річі та Брайеном Керніганом була розроблена мова програмування С. Вона склала в собі можливості мов високого та низького рівнів, а також потужний потенціал для подальшого розвитку.

В 1979 році Барном Страуструпом, працівником американського науково-дослідного центру AT&T Bell Laboratories (Нью-Джерсі) була розроблена С + +.

З часом С + + удосконалювався і змінювався. Був сформований комітет для створення єдиного стандарту С + +. Туди увійшли ISO та Американський національний інститут стандартів (ANSI). В 1994 на світ з'явилася бібліотека стандартних шаблонів (Standard Template Library, STL). Вона набагато спрощувала процес програмування і була внесена до складу С + +.

С + + є мовою обєктно-орієнтованого програмування, що значною мірою зпрощує та пришвидшує процес програмування в ньому, що дозволяє займатися створенням певних додатків програмістам-початківцям. Це призвело до того, що С + + на даний момент є одим з найпоширеніших середовищ програмування в світі. Н С + + написані більшість програмнго прикладного забезпечення для Windows, а на С було створено деякі операційні системи (Unix, Linux) і т.д.

Зараз компанія Borland випустила вже шосту версію язика програмування С + + Builder. Існує декілька варіантів цього пакету:

1. Standard
2. Professional
3. Client/Server Suite

Системні потреби у C + + Builder не надто великі, а працювати вона може з усіма операційними системами Windows, крім 95-ї.

На інсталяційному диску, окрім звичайно самого C + +, можуть міститися сервер бази даних InterBase, утіліти BDE Administrator, Visi broker, інформаційна допомога, та інше.

Сам інтерфейс C + + Builder побудований дуже дружньо до користувача. При роботі в C + + перед нашими очима знаходяться :

1. Головне вікно, де знаходяться строка меню, панелі інструментів, палітра компонентів та інше.
2. Вікно стартової форми. З неї будується майбутній вигляд вікна програми, яку ми створюємо.
3. Вікно редактора властивостей об'єкта Object inspector. В ньому встановлюються властивості об'єктів (кнопок, полів і т.д.). Кожний об'єкт має свої, індивідуальні властивості.
4. Вікно перегляду списку об'єктів - Object TreeView, в якому зазначені усі об'єкти, що створені нами та знаходяться на формі.
5. Вікно редактора коду – в ньому пишеться код програми, що створюється.

Інтерфейс C + + є схожим з іншими прикладними програмами, що встановлюються в Windows, тому розібратися у ньому не складно.

Самий принцип роботи у C + + Builder, звичайно, дуже схожий на той, що використовується у Borland Delphi, за вийняток лише того що використовується в ньому C, а не Pascal, хоча C + + Builder дозволяє одночасно програмувати і на C + + і Pascal.

1.2. Основи Обєктно-орієнтованого програмування

ООП зародилося на основі таких мов програмування як: Pascal, ADA, Smalltalk, C + +. До введення об'єктно-орієнтованного програмування технологія створення комп'ютерних програм базувалася на процедурному

програмуванні, в якому основою програми була функції і процедури. Створена таким чином програма мала чіткий алгоритм роботи, а саме послідовність дій, направлених на досягнення поставленої цілі.

Об’єктно-орієнтоване програмування представляє собою такий спосіб програмування, що нагадує людське мислення. В ООП головною відправною точкою при проектуванні програми є об’єкт.

Об’єктно-орієнтоване програмування базується на трьох основних принципах:

1. Наслідування.
2. Інкапсуляція.
3. Поліморфізм.

Програма що побудована відповідно з цими принципами являє собою сукупність об’єктів і способів їх взаємодії. Обміни інформацією між об’єктами в ООП відбуваються за допомогою повідомлень.

Об’єкт – поняття, абстракція чи любий інший предмет, що має сенс в контексті даної проблем.

Клас – особлива структура, що може мати у своєму складі поля, методи і свойства. Клас виступає ву ролі об’єктного типу даних, а об’єкт являється конкретним екземпляром об’єднаного класу. Об’єднання об’єктів у класи визначається семантикою.

ООП є найкращим інструментом побудови ієрархічних дерев, структур, даних. Процес, за допомогою якого один тип даних наслідує характеристики іншого називається наслідуванням. Якщо якась характеристика об’єкта колись прийнята, то всі об’єкти, що знаходяться нижче нього, наслідують цю характеристику.

Об’єкт – це складна конструкція, поведінка і характеристика складових частин якої знаходяться під взаємодією. Для забезпечення надійності функції програми приміряється правило інкапсуляції. Виключення прямого доступу

до полів об’єкту за допомогою безпосередніх операцій зчитування і оновлення їх внутрішнього складу.

Найбільш раціональним способом отримання доступу до полів даних є використання процедур або функцій, що описані у середині об’єкта. Функція, яку можна примінити до об’єктів даного класу називається операцією. Одна й та сама операція може примінитися до об’єктів різних класів. Така операція називається поліморфною.

Метод – це процедура чи функція, що включена в об’єкт таким чином, що що екземпляр даного типу становиться для неї з середини. При визначенні метода він додатково ідентифікується іменем типу.

Поля і метод являються складовим частинами нової структури – об’єкту. Тобто кожній операції відповідає певний метод, таким чином реалізується ця операція для об’єктів даного класу. Операція це специфікація методом, а метод – реалізація операцій.

Кожна операція має свій неявний аргумент (об’єкт) до якого вона приміняється. У середині об’єкта метод визначається заголовком процедури чи функції, що діє як метод.

Сукупність даних і методів Її читання називається свойством. Свойства можна встановлювати під час проектування програми.

Середовищем взаємодії об’єктів є повідомлення, що генеруються в результаті настанні різних подій.

1.3.Синтаксис мови

Основні синтаксичні правила запису програм мовою С + + сходяться до наступного:

1. Прописні і малі літери вважаються різними символами. Тому, наприклад, ідентифікатори DATABASE, DataBase, Database і database відносяться до зовсім різних перемінних, чи константам об'єктів. При запису ідентифікаторів можуть використовуватися латинські букви, цифри, символ

підкреслення "__". Ідентифікатор не може починатися з цифри і не може містити прбілів. Довжина ідентифікатора не обмежена, але заради зручності читання програми треба прагнути використовувати короткі ідентифікатори. Пробіли (пробіли, знаки табуляції, символ нового рядка, коментар) можуть розміщатися в будь-якім місці тексту, але не усередині идентифікатора.

2. Коментарі в тексті полягають у дужки виду / * текст комментаря*/. Такі коментарі можуть вводитися в будь-якім місці тексту, зокрема, у середині операторів, і займати будь-як кількість рядків. Вкладені комментарі звичайно не допускаються. Вважається, що коментар закінчився, якщо в тексті зустрілися перші символи "*/". Утім, у C + + Builder можна забезпечити використання вкладених коментарів. Для цього треба включити опцію Nested Comments на сторінці Advanced Compiler вікна опцій проекту. Однак у стандарті C вкладені коментарі не допускаються, так що їх використання робить код не переносимим на інші платформи. Саме тому дана опція вимкнена. Ще один спосіб уведення комментаря – розміщення його після двох символів слэш "//". Цей коментар повинен займати кінець рядка, у якій він уведений, і не може переходити рядок, що йде далі. Будь-який текст у рядку, поміщений після символів "//", вважається за коментар.

3. Кожне речення мови кінчається символом крапка з комою ";". Деякі виключення з цього правила будуть обговорені особливо.

4. У рядку може розміщатися кілька операторів. Треба писати програму так, щоб її було легко читати. Треба виділяти об'єднані змістом оператори в групи, широко використовуючи для цього відступи і коментарі. Фігурні дужки {} виділяють складений оператор. Всі оператори, що знаходяться між ними, сприймаються синтаксично як один оператор.

5. Усі використовувані типи, константи, перемінні, функції повинні бути оголошені чи описані до їхнього першого використання. Оголошення можуть зустрічатися в будь-якім місці тексту.

1.4. Директиви препроцесора

Обробка програми препроцесором відбувається перед її компіляцією. На цьому етапі попередньої обробки ви можете виконати наступні дії:

1. Включити в компилируемый файл інші файли.
2. Визначити символічні константи і макроси.
3. Задати режим умовної компіляції програмного коду й умовного виконання директив препроцесора.

Усі директиви препроцесора починаються із символу "#" і до початку директиви в рядку можуть знаходитися тільки символи пробілу. Будь-який рядок, начинающая-ся із символу "#", сприймається як директива препроцесора. Наприкінці директив препроцесора не ставиться крапка з комою.

1.5. Оголошення змінних

Змінна є ідентифікатором, що позначає деяку область у пам'яті, у якій зберігається значення перемінної. Це значення може змінюватися під час виконання додатка.

Оголошення змінної має вид:

тип список_ідентифікаторів_перемінних;

Список ідентифікаторів може складатися з ідентифікаторів пзмінних, розділених комами. Наприклад:

int x1, x2;

Одночасно з оголошенням деякі чи всі змінні можуть бути ініціалізовані. Наприклад:

int x1 = 1, x2 = 2;

Для ініціалізації можна використовувати не тільки константи, але і певні вирази, що містять оголошені раніше константи і змінні. Наприклад:

int x1 = 1, x2 = 2 * xl;

Оголошення змінних може бути окремим оператором чи робитися усередині таких операторів, як, наприклад, оператор циклу:

for (int i = 0; i < 10; i++).

1.6. Класи пам’яті

Кожна змінна характеризується класом пам’яті, який визначає її час життя – період, на протязі якого ця змінна існує в пам’яті. Одні змінні існують недовго, інші – неодноразово створюються і зникають, треті – існують на протязі всього часу виконання програми.

В C++ Builder є 4 специфікації класу пам’яті: auto, register,ezxtern і static. Специфікація класу пам’яті ідентифікатора визначає його клас пам’яті, область дії.

Область дії ідентифікатора називається область програми, в якій на дану зміну можно сослатися.

1.7. Оголошення й опис функцій

Функції являють собою програмні блоки, що можуть визиватися з різних частин програми. При виклику в них передаються деякі зміни, константи, вирази, що є аргументами, що у самих процедурах і функціях сприймаються як формальні параметри. При цьому функції повертають значення визначеного типу, що заміщає у виразі ім'я викликаної функції.

Наприклад, оператор

I = 5 * F(X) ;

викликає функцію F з аргументом X, множить повернуте нею значення на 5 і привласнює результат перемінної I.

Допускається також виклик функції, що не використовує значення, що повертається нею. Наприклад:

F(X);

У цьому випадку значення, що повертається функцією, ігнорується. Функція описується в такий спосіб:

тип_ що повертається_значення ім'я_функції(список параметрів)

{

оператори тіла функції

}

Перший рядок цього опису, що містить тип значення, що повертається, ім'я функції і список параметрів, називається заголовком функції. Тип значення, що обертається, може бути будь-яким, крім масиву і функції. Можуть бути також функції, що не повертають ніякого значення. У заголовку таких функцій тип значення, що повертається, з'являється void.

Якщо тип значення, що повертається, не зазначений, він вважається рівним int.

Список параметрів, що укладається в дужки, у найпростішому випадку представляє собою поділений комами список виду

тип_параметра ідентифікатор__параметра

Наприклад, заголовок:

double FSum(double X1, double X2, int A)

повідомляє функцію з ім'ям FSum, із трьома параметрами XI, X2 і A, з яких перші два мають тип double, а останній – int. Тип повертаємого результата – double. Імена параметрів X1, X2 і A – локальні, тобто вони мають значення тільки усередині даної функції і ніяк не зв'язані з іменами аргументів, переданих при виклику функції. Значення цих параметрів на початку виконання функції дорівнюють значенням аргументів на момент виклику функції.

Нижче приведений заголовок функції, що не повертає ніякого значення:

void SPrint (AnsiString S)

Вона приймає один параметр типу рядка і, наприклад, відображає його в якому-небудь вікні додатка.

Якщо функція не приймає ніяких параметрів, то чи дужки залишаються порожніми, чи в них записується ключове слово void. Наприклад:

```
void Fl (void )
```

чи

```
 void Fl()
```

Як правило, крім опису функції в тексті програми включається також прототип функції - її попереднє оголошення. Прототип представляє собою той же заголовок функції, але з крапкою і коми " ; " наприкінці . Крім того, у прототипі можна не вказувати імена параметрів. Якщо все-таки вказувати імена, то їхньою областю дії є тільки цей прототип функції. Можна використовувати ті ж ідентифікатори в будь-якім місці програми в будь-якій якості. Таким чином, вказівка імен параметрів у прототипі звичайно переслідує тільки одна мета – документування програми. Приклади прототипів приведених вище заголовків функцій:

```
double FSum(double  X1,double  X2, int A) ;
void SPrint(AnsiString S) ;
void Fl (void);
```

чи

```
double FSum(double, double, int);
void SPrint(AnsiString);
void Fl ();
```

Введення в програму прототипів функцій переслідує кілька цілей. По-перше, це дозволяє використовувати в даному модулі функцію, описану в якому-небудь іншому модулі. Тоді з прототипу компілятор одержує інформацію, скільки параметрів, якого типу й у якій послідовності одержує дана функція. По-друге, якщо на початку модуля ви визначили прототипи функцій, то послідовність розміщення в модулі опису функцій не важлива. При відсутності прототипів будь-яка використовувана функція повинна бути описана до її першого виклику в тексті. І, нарешті, прототипи, розміщені в

одному місці (звичайно на початку модуля), роблять програму більш наочною і самодокументованою. Особливо у випадку, якщо вводити прототипи хоча б короткими коментарями.

Якщо передбачається, що якісь з описаних у модулі функцій можуть використовуватися в інших модулях, прототипи цих функцій варто включати в заголовний файл. Тоді в модулях, що використовує дані функції, достатньо буде написати директиву #include, що включає даний заголовний файл, і не треба буде повторювати прототипи функцій.

Звичайно функції приймають зазначене в прототипі число параметрів вказаних типів. Однак можуть бути функції, що приймають різне число параметрів (наприклад, бібліотечна функція printf) чи параметри невизначених заздалегідь типів. У цьому випадку в прототипі замість невідомого числа параметрів чи замість параметрів невідомого типу ставиться три крапки "...". Три кпапки можуть міститися тільки наприкінці списку параметрів після відомого числа параметрів типу, чи цілком заміняти список параметрів. Наприклад:

int prf(char *format, . ..);

Функція з подібним прототипом приймає один параметр format типу char * (наприклад, рядок форматування) і довільне число параметрів довільного типу. Функція з прототипом

oid Fp (...);

може приймати довільне число параметрів довільного типу.

Якщо в прототипі зустрічаються три крапки, то типи відповідних параметрів і їхня кількість компілятором не перевіряються.

Оголошенню функції можуть передувати специфікатори класу памяті extern чи static. Специфікатор extern передбачається за замовчуванням, так що записувати його не має сенсу. До функцій, оголошених як extern, можно одержати доступ з інших модулів програми. Якщо ж оголосити функцію зі специфікатором static, наприклад

static void F(void);

то доступ до неї з інших модулів неможливий. Це треба використовувати у великих проектах щоб уникнути непорозумінь при випадкових збігах імен функцій у різних модулях.

1.8. Оператор do...while

Структура do...while використовується для організації циклічного виконання оператора чи сукупності операторів, називаних тілом циклу, доти, поки не виявиться порушеним деяка умова. Синтаксис управляючої структури do...while:

do оператор while (умова);

Структура працює в такий спосіб. Виконується оператор тіла циклу. Потім обчислюється умова – вираз, що повинен повертати результат булевого типу. Якщо вираження повертає true (не нульове значення), то повторюється виконання тіла циклу і після цього знову обчислюється выраз. Таке циклічне повторення циклу продовжується доти , поки вираз, що перевіряється, не поверне false (нуль). Після цього цикл завершується і керування передається оператору, що випливає за структурою do...while.

Оскільки перевірка виразу здійснюється після виконання тіла циклу, то цикл буде свідомо виконаний хоча б один раз, навіть якщо вираз відразу ложний. З іншого боку, програміст повинний бути упевнений, що вираз рано чи пізно поверне false. Якщо цього не відбудеться, то програма "зациклиться", тобто цикл буде виконуватися нескінченно. Іноді такі нескінченні цикли використовуються. Але в цьому випадку усередині тіла циклу повинне бути передбачене його переривання в якийсь момент, наприклад, оператором break, чи іншими способами.

Звичайно оператор do доцільно використовувати для організації пошуку серед безлічі об'єктів такого, котрий володіє якоюсь визначеною властивістю. Причому заздалегідь повинно бути відомо, що множина об'єктів не порожня,

тобто хоча б один об'єкт у ній мається. До того ж повинний бути критерій, що дозволяє перевірити, чи не є поточний об'єкт останнім. Тоді тіло циклу включає оператори 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 чи шукані властивості знайдені, виконання циклу переривається. Якщо ж об'єкт не останній і шукані властивості в нього не знайдені, здійснюється перехід до наступного об'єкту.

Якщо множина об'єктів, що перевіряються, може бути порожньою, треба використовувати інший оператор циклу – while. Якщо число повторень циклів заздалегідь відомо, краще застосовувати оператор for .

Нижче приведений приклад, у якому у файлі File1.txt шукається рядок, що містить фрагмент тексту (послідовність символів з урахуванням регістра), зазначений користувачем у вікні редагування Edit1. Перевірка наявності в рядку заданого фрагмента перевіряється функцією strstr. Закінчення файлу перевіряється функцією feof.

```
FILE *F;
 char S[256] = "" ;
AnsiString SKey = Edit1->Text;
if ((F = fopen ("File1/txt", "r")) = = NULL)
{
ShowMessage("фрагмент не знайдений");
return;
}
Do
fgets (S,256,F);
while(!feof(F) &&(strstr(S,SKey.c_str()) = = NULL));
fclose(F);
if (strstr(S,SKey.c_str()) = = NULL)
```

Цикл буде виконуватися, доти , поки не досягнуть кінець файлу і поки функція strstr повертає NULL (фрагмент не знайдений). Якщо хоча б одна з цих умов порушується (досягнуть кінець файлу чи знайдений фрагмент), виконання циклу припиняється.

2. Спеціальний розділ

2.1 Обчислення елементарних функцій за допомогою степенних рядів

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

Степенні ряди є важливим апаратом для табулювання елементарних функцій, тому що їхнє застосування дозволяє звести задачу обчислень значень функції з заданою точністю до задачі обчислення багаточлена.

Елементарні функції можна розкласти в степенний 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 прийнято a = 0, завдяки чому виходить звичайний степенний ряд по ступенях x. Обриваючи ряд у потрібному місці, одержимо багаточлен, що наближає дану функцію.

Найбільш уживаними є статечні ряди для функцій $e^x, \cos x, \sin x$, що сходяться при будь-якім значенні x:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots,$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots, \qquad (2.2.)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{n+1}}{(2n+1)} + \ldots,$$

$$\ln \frac{1+x}{1-x} = 2(x + \frac{x^3}{3} + \frac{x^5}{5} + \ldots \frac{x^{2n+1}}{2n+1} + \ldots).$$

При обчисленні елементарних функцій за допомогою степенних рядів зручно користуватися рекурентними співвідношеннями, що дозволяють обчислювати черговий член ряду не безпосередньо, а через вже обчислені попередні члени. Рекурентним співвідношенням називають рівність, що зв'язує між собою два чи кілька сусідніх членів чи послідовності ряду. За допомогою такої рівності можна визначити наступний член ряду через попередні.

Для приведених вище рядів рекурентні співвідношення можуть бути легко виведені безпосередньо. Найпростіше узяти відношення двох сусідніх членів. Розглянемо, наприклад, статечної ряд для функції , загальний член котрого має вид $a_n = x^n / n!$. За допомогою відношення наступного члена до попереднього одержимо

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

Таким чином, для двох сусідніх членів ряду одержуємо рекурентне співвідношення

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

Яке зручно для послідовного обчислення членів ряду.

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

Якщо члени ряду убувають досить швидко і притім із самого початку, то вигідно використовувати знакозмінний ряд, погрішність якого легко оцінюється. Сума знакозмінного ряду менше його першого члена (по

абсолютної величині). Тому при заміні суми такого ряду його частковою сумою допускається погрішність, що не перевищує по мудулю першого з відкинутих членів. Якщо ж члени ряду убувають, або убувають хоча і дуже швидко, але не спочатку, а перші члени ряду достатньо великі, фактична погрішність буде помітно більша через погрішності вирахування перших великих членів. Наприклад, члени рядів для обчислення cosx і sinx убувають досить швидко і ряди сходяться при будь-якому x. Проте при великих x ($x \geq 10$) перші члени цих рядів досить швидко зростають, і визначення cosx і sinx за допомогою цих рядів доволі важке. Виникає така втрата точності при вирахуванні великих перших членів, яку не можна возмістити обчисленням великого числа доданків.

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

Розглянуті статечні ряди – зручний засіб для програмування елементарних функцій. Кожний з рядів легко програмується як звичайний цикл, якому 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 деякого об'єкта при заданій величині вхідного впливу згідно рівняння:

$$y = f(x). \qquad (2.5.)$$

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

Використовуємо оператори циклів do...while. Програма повинна дозволяти вводити вхідні дані з клавіатури, вихідні дані (результати обчислень) відображати на екрані монітора і записувати у файл з програмою. Процедуру обчислення робити з наперед заданою оператором точністю, але

не більш 30 ітерацій. Додатково забезпечити можливість відображення номера останньої ітерації, величину останнього члена ряду, і отриману точність обчислень.

Факторіали розрахуємо таким чином:

$$\begin{aligned} &1! = 1; \\ &2! = 1 \times 2 = 2; \\ &3! = 1 \times 2 \times 3 = 6; \\ &4! = 1 \times 2 \times 3 \times 4 = 24. \end{aligned} \qquad (2.7.)$$

Потрібно розрахувати похідну для першої частини рівняння:

$$\begin{aligned} &y = 2x; \\ &y^{I} = 2e^{2x}; \\ &y^{II} = 4e^{2x}; \\ &y^{III} = 8e^{2x}; \\ &y^{VI} = 16e^{2x}; \end{aligned} \qquad (2.8.)$$

Створимо програму обчислення ряду:

$$f(x) = e^{2\times 0} + \frac{2e^{2\times 0}}{1!} \times (x-0) + \frac{4e^{2\times 0}}{2!} \times (x-0)^2 + \frac{8e^{2\times 0}}{3!} \times (x-0)^3 + \frac{16e^{2\times 0}}{4!} \times (x-0)^4 = \qquad (2.9.)$$
$$= e + \frac{2}{1!} \times x + \frac{4}{2!} \times x^2 + \frac{8}{1!} \times x^3 + \frac{16}{4!} + x^4;$$

Таким чином, для першої итерації при x = 1 будемо мати:

$$1: y = e^{2\times 1} = e^2 = 1 + \frac{2}{1!} = 3. \qquad (2.10.)$$

Для другої, третьої и четвертої итерації відповідно:

$$\begin{aligned} &2: y^{I} = e^{2\times 1} = e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} = 3 + 2 = 5; \\ &3: y^{II} = e^{2\times 1} = e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} + \frac{8}{3!} = 3 + 2 + 1{,}3 = 6{,}3; \\ &4: y^{III} = e^{2\times 1} = e^2 = 1 + \frac{2}{1!} + \frac{4}{2!} + \frac{8}{3!} + \frac{16}{4!} = 3 + 2 + 1{,}3 + 0{,}7 = 7; \end{aligned} \qquad (2.11.)$$

Потрібно розрахувати похідну для другої частини рівняння:

$$\begin{aligned} &y = 5e^{x}; \\ &y^{I} = 5e^{x}; \\ &y^{II} = 5e^{x}; \end{aligned} \qquad (2.12.)$$

$$y^{III} = 5e^x;$$
$$y^{VI} = 5e^x.$$

Створимо програму обчислення ряду:

$$f(x) = 5e^{1\times0} + \frac{5e^{1\times0}}{1!}\times(x-0) + \frac{5e^{1\times0}}{2!}\times(x-0)^2 + \frac{5e^{1\times0}}{3!}\times(x-0)^3 + \frac{5e^{1\times0}}{4!}\times(x-0)^4 = \quad (2.13.)$$
$$= 5 + \frac{5}{1!}\times x + \frac{5}{2!}\times x^2 + \frac{5}{1!}\times x^3 + \frac{5}{4!} + x^4;$$

Відповідне програмне забезпечення має такий вигляд:

```
#include <vcl.h>
 #include <stdio.h>
#include <stdio.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;
float y;
float d;
float y0;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{
Button2->Enabled=false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit1Change(TObject *Sender)
{
 Button2->Enabled = false;
Label1->Visible = false;
Label2->Visible = false;
Label3->Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit2Change(TObject *Sender)
{
  Button2->Enabled = false;
Label1->Visible = false;
Label2->Visible = false;
Label3->Visible = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  //Если кнопка нажата то:
 switch (Button1->Tag)
        {
        case 0:
        Button2->Enabled = true; // стає видима кнопка Button2
```

```
        Button2->Caption = "Зберегти";// на Button2 зявляється напис "Зберегти"
        Button2->Tag = 0;  // Значению Tag присваивается значение 0
        break;
        }
    // При натисненні кнопки:

    // Вводимо змінну всередину процедури Button1Click
    float r; float rm;
    float b; float w;
    float da; float dm;
    float t; float tm;
    float e; float em;
     float dd;  float xm;
     float ym;
    // Задаємо початкові значення змінних:
    r = 1; rm=1;
    b = 30;// Задаємо максимальне значенне ітерацій
    da = 0; dm=0;
    x = 1; xm=1;
    // Ввод концевої точност ітерацій с обєкта Edit1,
    // наприклад 1E-5 (відповідає 0.00001);
    d = StrToFloat(Edit1->Text);
    // Вводимо показник ступеня з обєкта Edit2
    // наприклад 2E-1 (відповідає 0.2)
    y = StrToFloat(Edit2->Text); ym=StrToFloat(Edit2->Text);
    da = 1; dm=1; // задаємо шаг ітерації
    t = 1; tm=1;
    y0 = y;
```

```
e = 0; em=0;
// Огранізуємо цикл кількості ітерації розрахунку в ступені a2,
//  розрахунок факторіала
do {
        da = da + 1; // привласнення у наступному значенні циклу da більше на 1
        r = da -1;// кількість ітерацій (у циклі do...while перевірка відповідності
// умов проводиться в кінці циклу вже після розрахунків нових значень)
        y =2*(y*y0);  // разрахунок в ступені a2
        t *= r;   // разрахунок факторіала
        e += y/(y0*t);// разрахунок сумми членів ряда без першого дорівнює 1
        x = (e + 1); // обчислення значення e в заданому ступені
        dd = y/(y0*t);// обчислення значення останнього члена ряда
// Вивід результатів розрахунків на форму
Label6->Caption = "Результат:" + FloatToStrF(x,ffGeneral,7,2);
// Кінець циклу, перевірка виконання умов: допустимого числа ітерпцій
// і точності розрахунку
    } while (da <= b ^ dd <= d);
do {
        dm = dm + 1; // привласнення у наступному значенні циклу da більше на 1
        rm = dm -1;// кількість ітерацій (у циклі do...while перевірка відповідності
// умов проводиться в кінці циклу вже після розрахунків нових значень)
        ym =(ym*y0);  // разрахунок в ступені a2
        tm *= rm;   // разрахунок факторіала
```

```
        em += ym/(y0*tm);// разрахунок сумми членів ряда без першого дорівнює 1
        xm = 5*(em + 1); // обчислення значення е в заданому ступені
        dd = y/(y0*t);// обчислення значення останнього члена ряда
    // Вивід результатів розрахунків на форму
    Label9->Caption = "Результат:" + FloatToStrF(xm,ffGeneral,7,2);
    // Кінець циклу, перевірка виконання умов: допустимого числа ітерпцій
    // і точності розрахунку
      } while (dm <= b ^ dd <= d);
    w=(x-xm)/4*y0;
    Label10->Caption = "Результат :" + FloatToStrF(w,ffGeneral,7,2);
    }
    //---------------------------------------------------------------------------
    void __fastcall TForm1::Button2Click(TObject *Sender)
    {
     // Процедура запису і читання даних в і із файлу Test.txt
       float d1, y1, x1;  // Вводимо додаткові змінні
       char s[10], s1[10]; // Обявленяе рядка длінною 10 Символів
    strcpy(s,"Raschet"); // Додання до рядка s слова  "Raschet"
    int handle; // запис у файл
     // Якщо кнопка Button2 нажата 1 раз
     switch (Button2->Tag)
          {
          case 0:
          Button2->Tag = 1; // значенню Tag привласнюється значення 1
    if((handle = open("Test.txt", O_WRONLY | O_CREAT | O_BINARY)) == -1)
     {
      // Повідомлення якщо файл не вдається створити:
```

```
 ShowMessage("Файл не вдається створити");
 return;
 }
 write(handle, &d, sizeof(int));  //запис d
 write(handle, &y0, sizeof(int)); //запис y0
 write(handle, &x, sizeof(int));  //запис x
 write(handle, s, strlen(s)+1);   // запис рядка s
 close(handle);
 Button2->Caption = "Читати"; // вивід на кнопці Button2 напису "Читати"
 break;

 case 1: // Если Button2 натиснути вдруге
 // процедура читання з файлу
 if ((handle = open("Test.txt", O_RDONLY | O_BINARY)) == -1)
{
// Повідомлення при помилці читання файлу
 ShowMessage("Файл не відкрити відкрити");
 return;
}
 read(handle, &d1, sizeof(int));  // Читання d
 read(handle, &y1, sizeof(int));  // Читання y
 read(handle, &x1, sizeof(int));  // Читання x
 read(handle, s1, strlen(s)+1);   // читання рядка
// активація відображень на формі элементів 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 ступінь:" +
FloatToStrF(y1,ffGeneral,7,2);
        Label3->Caption = "Записане число результат:" +
FloatToStrF(x,ffGeneral,7,2);
        close(handle);
         break;
        }
       }
```

Висновки

В данному курсовому проекті проведена розробка програми визначення регулюючого впливу, що розраховується комп’ютерною системою керування. Програма складена за допомогою циклу do…while, містить об’єм 256 Кбайт.

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 доработку. Та є відкритим програмним забезпеченням.

Перелік посилань

1. Шилдт Г. – Самоучитель С++: Пер. с англ. – 3-е изд. – СПб.: БХВ-Петербург, 2003. – 688с.
2. Архангельский А.Я. – Програмування в С++Builder 6. – М.: ЗАО «Издательство БИНОМ», 2004 г. – 1152 с.6 ил.
3. Культин Н.Б. – Самоучитель С++ Builder. – СПб.: БХВ-Петербург, 2004. – 320 с.: ил.
4. Дьяконов В.П. – Справочник по расчетам на микрокалькуляторах. – 3-е изд., доп. И перераб. – М.: Наука. Гл ред. Физ-мат.лит., 1989. – 464 с.
5. Вирт Н. – Алгоритми і структури данних / Пер. с англ. М.: Мир, 1989.- 360 с., ил.
6. Гринзоу Лу. Філософія програмування для Windows 95/NT/ Пер. с англ. – СПб.: Символ-Плюс, 1997. – 640 с., ил.
7. Культин Н.Б. - С/С++ в примерах и задачах. – СПб.: БХВ-Петербург, 2001.- 288 с., ил.
8. Страуструп Б. – Мова програмування С++. – М.: Радио и связь 1991.

Додаток А

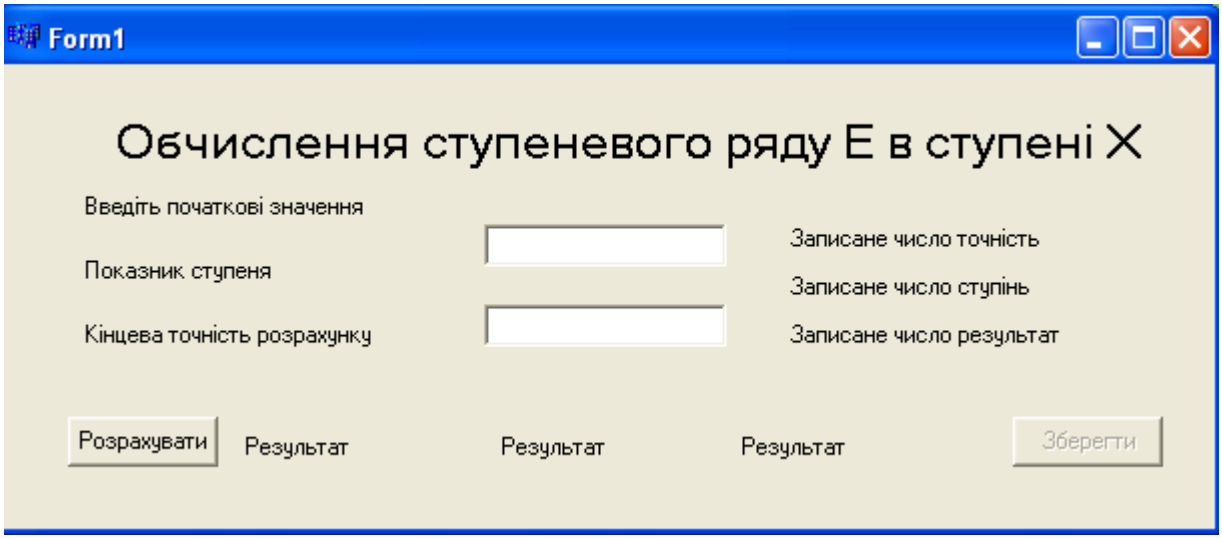

Рисунок Д.А.1.-Вид вікна програми до початку розрахунків

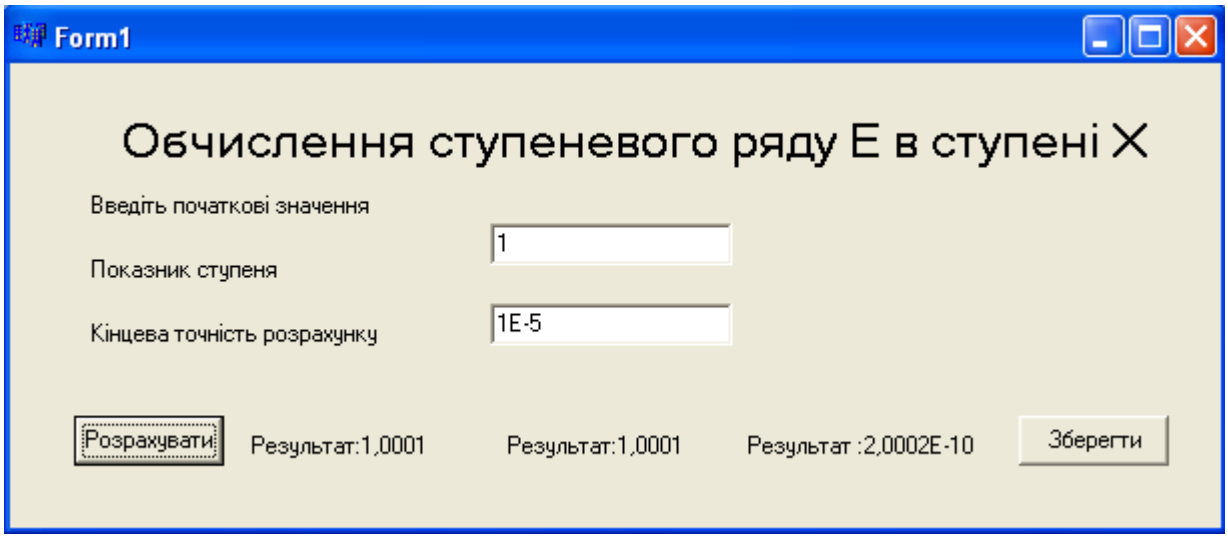

Рисунок Д.А.2.-Вид вікна програми після розрахунків

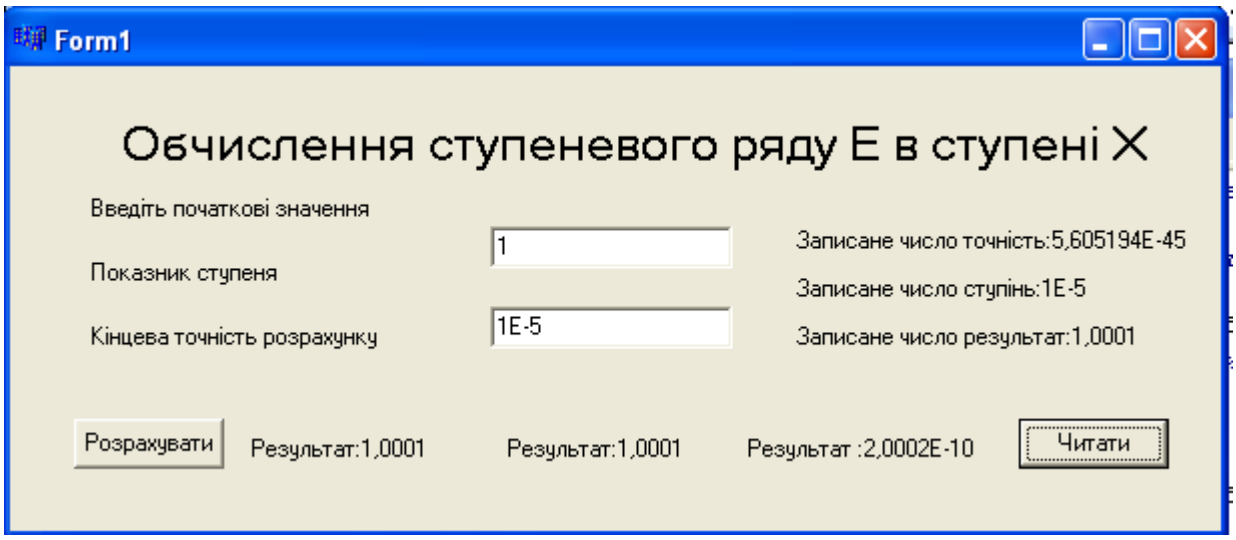

Рисунок Д.А.3.-Вид вікна після читання з файлу результатів розрахунків