Розділ 3

Моделі та технології обробки фінансової інформації

УДК 519.86:347.464

Дубницький В.Ю.
Кобилін А.М.
Самородов Б. В.

ВИКОРИСТАННЯ ЗВОРОТНЬОГО ПОЛЬСКОГО ЗАПИСУ ДЛЯ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СПЕЦІАЛІЗОВАНИХ ІНТЕРВАЛЬНИХ ПРОГРАМНИХ КАЛЬКУЛЯТОРІВ

Анотація. Запропоновано спосіб оцінювання параметрів, що визначають надійність програмного забезпечення. Для цього виконане дослідження і розв'язання найпростіших алгебраїчних рівнянь, коефіцієнти яких належать множині інтервальних чисел. Розроблено програмне забезпечення, що дозволяє користувачеві проводити обчислення, вибираючи з відповідних списків назви змінних, співпадаючих з назвами показників, вводити вихідні дані, вибирати тип операції, виконувати розрахунки , у випадку потреби зберігати їх на аркуші електронної таблиці та подавати результати в графічному вигляді.

Ключові слова: інтервальні числа, надійність програмного забезпечення, зворотний польський запис, стековий інтервальний калькулятор, модель Холстеда, модель Мілса.

Вступ. Однією з важливих задач у розробці програмного забезпечення є його тестування на наявність помилок (так званих багів). Це обумовлено тим, що на якість програмного продукту може впливати досить багато факторів. Вони змінюються в залежності від призначення програмного продукту, складності поставленої задачі, об'єму програми та інших причин. Дуже часто їх характеристики мають високий рівень нестохастичної невизначеності. Висока надійність, яку вимагають від програмних систем критичного застосування вимагає проведення великої кількості випробувань для достовірного визначення їх безвідмовної роботи. В [1] наведена оцінка необхідної кількості випробувань яка має наступний вид:

$$n = \frac{\lg(1-\beta)}{\lg(1-p)}; \qquad (1)$$

де: n - необхідна кількість випробувань,

β – довірча ймовірність,

p – верхнє припустиме значення ймовірності відмови.

Результати відповідних обчислень наведені в таблиці 1.

Таблиця 1

Визначення кількості випробувань програмної системи при заданій довірчій імовірності й імовірності відмови

Імовірність відмови p	Рівень довірчої ймовірності β			
	0,95	0,975	0,99	0,995
$1\cdot10^{-2}$	299	367	458	528
$1\cdot10^{-3}$	2995	3687	4603	5296
$1\cdot10^{-4}$	29956	36887	46049	52981
$1\cdot10^{-5}$	999572	368886	460515	529829
$1\cdot10^{-6}$	2995731	3688878	4605158	5298315

Із цієї таблиці видно, що при випробуванні програмних систем у режимі реального часу та високих вимогах до їх безвідмовності кількість випробувань досягає сотень тисяч і навіть мільйонів, а тривалість випробувань може бути порівнянною з місяцями або, навіть роками.

У цьому випадку адекватним математичним апаратом для кількісного аналізу результатів тестування комп'ютерних програм може слугувати апарат інтервальних обчислень [2, 3].

Мета роботи. Метою роботи є розробка методу, що дозволяє формулювати вимоги до параметрів, які визначають надійність програмного забезпечення при їх нестохастичній невизначеності. Подібні ситуації мають місце на різних стадіях попереднього проектування складних програмних продуктів.

Аналіз літератури. В зв'язку з необхідністю розв'язання зворотної задачі – підбору параметрів програми, що гарантують її необхідну надійність, далі використуємо систему аксіом $АК_2$, наведену в роботі [3] .

Припустимо, що символ «o» означає одну з операцій +, -, *, / які мають традиційний зміст. Використовуючи алгебраїчну символіку запишемо, що

$$o \in \{+,-,\cdot,/\}. \tag{2}$$

Тоді для двох інтервальних чисел $[A_1],[A_2]$ справедлива умова:

$$[A_1]o[A_2]=(\min U,\max U) \tag{3}$$

де

$$U=((a_1,a_2)\,o\,(b_1,b_{b_2}))=(a_1\,o\,b_1),(a_1\,o\,b_2),(a_2\,o\,b_1),(a_2\,o\,b_2) \tag{4}$$

Порівняємо виконання операції (-) у системах аксіом AK_1, наведеної в [2] і AK_2, наведеної в [3] використовуючи геометричну ілюстрацію (рис. 1).

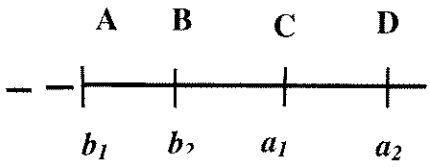

Рис. 1. Геометрична ілюстрація виконання операції (-)

Очевидно, що:

$DA=a_2-b_1$; $CA=a_1-b_1$; $DB=a_2-b_2$;

$CB=a_1-b_2$.

Тоді відрізком найбільшої довжини буде відрізок DA, найменшої - відрізок CB. У такий спосіб показаний збіг цих двох систем аксіом тому, що умови (2, 3, 4) збігаються з умовою (1).

Відповідно роботі [3] назвемо інтервали $[A]=(a_1,a_2)$ й $\overline{[A]}=(a_2,a_1)$ спряженими.

У цій же роботі показано, що розв'язання алгебраїчних рівнянь, коефіцієнти яких є інтервальними числми, можна отримати в наступному виді.

Рівняння I типу:

$$[A]+[X]=[B], \tag{5}$$

тоді

$$[X]=[B]-[A]. \tag{6}$$

Рівняння II типу:

$$[A]\cdot[X]=[B], \quad 0\notin[A], \tag{7}$$

тоді

$$[X]=\frac{[B]}{[A]}. \tag{8}$$

Рівняння III типу:

$$[A][X]+[B]=[C], \tag{9}$$

тоді

$$[X]=\frac{[C]-\overline{[B]}}{\overline{[A]}}, \tag{10}$$

за умови, що $0\notin[A]$. Наведемо чисельні приклади. Розглянемо розв'язання рівняння типу I.

[X] + [5; 7] =[14; 18]

[X]=[14; 18] - [7; 5]

U = (7; 11; 9; 13)

min U =7, max U = 13

[Y] = [7,13].

Розглянемо розв'язання рівняння типу II.

[X] [5; 7] = [14; 18].

$$[X]=\frac{[14;18]}{[7;5]};$$

$$U=\left(2;\frac{14}{5};\frac{18}{7};\frac{18}{5}\right), \quad \min U=2; \quad naxU=\frac{18}{5}$$

$$[Y]=\left[2;\frac{18}{5}\right].$$

Розглянемо розв'язання рівняння типу III.

Нехай [A] = [2; 5]; [B] = [4; 9]; [C] = [15; 23].

Тоді:

[2; 5] * [X] + [4; 9] = [15; 23]

Відповідно до умови (28) :

$$[X]=\frac{[15;23]-[9;4]}{[5;2]}=\left[\frac{6}{5};\frac{19}{2}\right].$$

Отже, при розв'язанні прямих задач застосування аксіом $АК_1$ і $АК_2$ приводить до однакових результатів, при розв'язанні зворотні задачі варто використовувати систему аксіом $АК_2$.

Виклад результатів. Далі розглянемо основні моделі оцінки надійності програмного забезпечення, наведені в роботі [4], але для їхнього аналізу використаємо апарат інтервальних обчислень.

Розглянемо модель Холстеда

$$N_{ошибок} = \frac{V}{E_{критическое}} \tag{11}$$

де:

V – об'єм програми;

$E_{критичне}$ – емпірична постійна $E = 2*10^3 \ldots 5*10^3$;

$N_{помилок}$ – кількість помилок у програмі.

В інтервальному виді ця модель має вигляд:

$$[N_{помилок}] = [A]\ [V], \tag{12}$$

де

$$[N_{ошибок}] = [2 \cdot 10^{-4}; 5 \cdot 10^{-4}]\ [V], \tag{13}$$

тобто отримане рівняння виду I.

На рис. 2 Web-форма рішення зворотної задачі оцінки надійності програмного забезпечення з використанням інтервального калькулятора моделі Холстеда

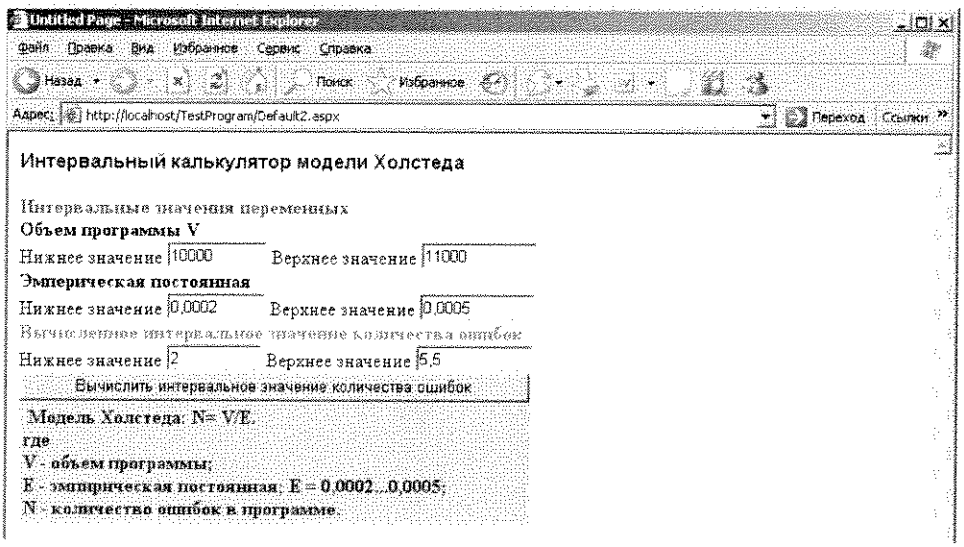

Рис. 2. Web-форма рішення зворотної задачі оцінки надійності програмного забезпечення з використанням інтервального калькулятора моделі Холстеда

Розглянемо модель Міллса

$$N_{ош} = \frac{N_{отм} - N_{внес}}{N_{внесв}} \tag{14}$$

де:

$N_{ош}$ – кількість власних дефектів у програмі;

$N_{отм}$ – кількість виявлених власних дефектів у програмі;

$N_{внес}$ – кількість внесених дефектів;

$N_{внесв}$ – кількість виявлених внесених дефектів.

Для розв'язання зворотної задачі можливі три варіанти: розв'язання відносно $N_{отм}$, відносно $N_{внес}$, відносно $N_{внесв}$. Однак, у кожному разі задача зводиться до рівняння виду II. Його варіанти наведені в табл 2.

Таблиця 2.

Умови задачі по визначенню параметрів моделі Міллса

[A]	*[X]*	*[B]*
$[N_{отм}]/[N_{внесв}]$	$[N_{внес}]$	$[N_{ош}]$
$[N_{внес}]/[N_{внесв}]$	$[N_{отм}]$	$[N_{ош}]$
$[N_{отм}]/[N_{внес}]$	$[X]=[1;1][N_{внес}]$ $[N_{внес}]=[X]^{-1}$	$[N_{ош}]$

На рис. 3 показана Web-форма розв'язання прямої і зворотніьої задачі оцінки надійності програмного забезпечення з використанням інтервального калькулятора моделі Мілса

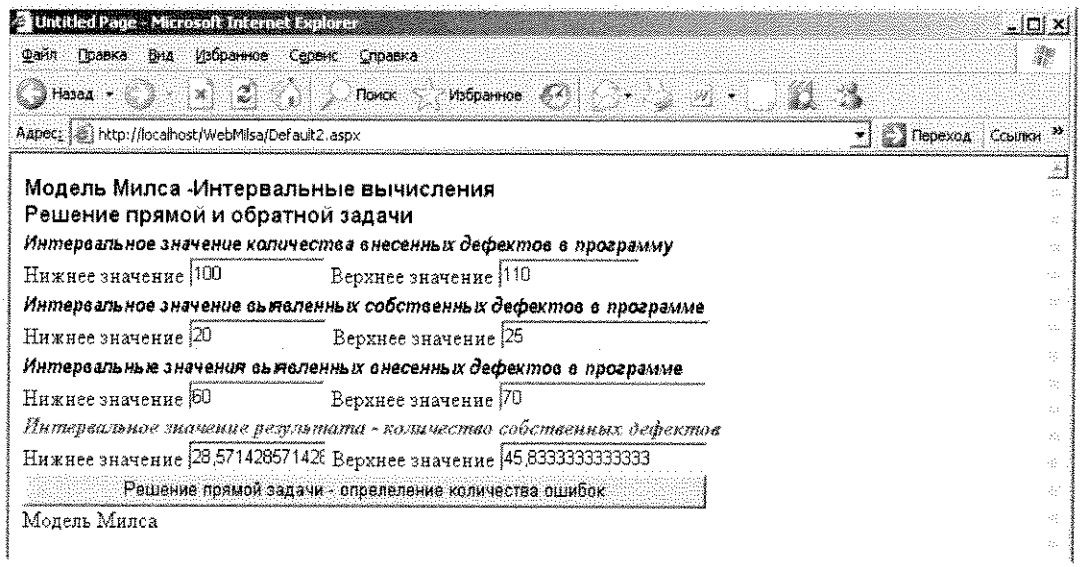

Рис. 3. Web-форма розв'язання прямої і зворотньої задачі оцінки надійності програмного забезпечення з використанням інтервального калькулятора моделі Мілса

Розглянемо модель фірми IBM

$$N = \alpha \cdot MUM + \gamma \cdot UM \ , \tag{15}$$

де:

α, γ – емпіричні постійні, $\alpha=23$; $\gamma=2$.

MUM – кількість модулів, що виправляються багаторазово (більше десяти разів);

UM - кількість модулів, що виправляються, (не більше десяти разів).

У свою чергу

$$UM = 0{,}9\ NM + 0{,}15\ CUM; \tag{16}$$

$$MUM = 0{,}15UM + 0{,}06CUM; \tag{17}$$

де *NM* – кількість модулів, *CUM* – кількість старих модулів, що виправляються.

Підставив (16) і (17) в (15) в інтервальному виді одержимо рівняння

$$N=[3{,}45;3{,}45]*[MUM]+[1{,}8;1{,}8]*[NM]+[1{,}68;1{,}68]*[CUM] \tag{18}$$

Нехай

$[3{,}45;\ 3{,}45] = [F]$; $[1{,}8;\ 1{,}8] = [G]$;

$[1{,}68;\ 1{,}68] = [Q]$

Тоді

$$[N]=[F]*[NUM]+[G]*[NM]+[Q]*[CUM] \tag{19}$$

При розв'язанні цього рівняння щодо одного з параметрів одержимо рівняння типу III.

Варіанти постановки задачі наведені в табл.3.

Таблиця 3.

Умови задачі по визначенню параметрів моделі фірми IBM

[A]	*[X]*	*[B]*	*[C]*
[A]	*[M]*	*[B]*[NM]+[C]*[CUM]*	*[N]*
[B]	*[NM]*	*[A]*[M]+[C]*[CUM]*	*[N]*
[C]	*[CUM]*	*[A]*[M]+[B]*[NM]*	*[N]*

Програмування калькулятора в режимі зворотного польського запису є істотним методом підвищення надійності програм, скорочення її об'єму, зокрема, відповідно до метрики Холстеда, кількість помилок у програмі після остаточної її розробки визначається по формулі (11). Обгрунтування цього наведено в роботі [5]. Для скорочення об'єму програм пропонується використання так званого зворотного польського запису (ЗПЗ). Порівняльний аналіз об'єму програм, розроблених у звичайній нотації і ЗПЗ, показав, що для формул, до складу яких входить більш 5 операндів, економія становить приблизно 17% об'єму програми.

Для реалізації ЗПЗ використовували стек, що є однією найбільш важливою зі структур даних. Ці структури зустрічаються в програмуванні буквально на кожному кроці, у найрізноманітніших ситуаціях. Особливо цікавий стек, що має самі несподівані застосування. У свій час при розробці серії ЕОМ IBM 360 на початку 70-х років XX століття фірма IBM зробила драматичну помилку, не передбачивши апаратну реалізацію стека. Ця серія містила багато інших невдалих рішень, але, на жаль, була скопійована в Радянському Союзі за назвою ЄС ЕОМ (Єдина Серія), а всі власні розробки були припинені. Це відкинуло радянську промисловість на багато років назад в області розробки комп'ютерів.

Стек - найпопулярніша та, мабуть, найважливіша структура даних у програмуванні. Стек являє собою запам'ятовувальний пристрій, з якого елементи виймають у порядку, зворотному їхньому додаванню. Це як би неправильна черга, у якій першим обслуговують того, хто встав у неї останнім. У програмістській літературі загальноприйнятими є абревіатури, що позначають дисципліну роботи стека. Дисципліна роботи стека позначається LIFO, останнім прийшов - першим підеш (Last In First Out).

Стек можна представити у вигляді трубки з підпружненим дном, розташованою вертикально. Верхній кінець трубки відкритий, у нього можна додавати, або, як говорять, заштовхувати елементи. Загальноприйняті англійські терміни в цьому плані дуже барвисті, операція додавання елемента в стек позначається push, у перекладі "заштовхнути, запхнути". Новий додаваємий елемент, що, проштовхує елементи, поміщені в стек раніше, на одну позицію вниз. При добуванні елементів зі стека вони як би виштовхуються нагору, по-англійському pop ("вистрілюють").

Прикладом стека може служити стопка паперів на столі, стопка тарілок і т.п. Звідси відбулася назва стека, що по-англійському означає стопка. Тарілки знімають із стопки в порядку, зворотному їхньому додаванню. Доступна тільки верхня тарілка, тобто тарілка на вершині стека. Гарним прикладом буде також служити залізничний тупік, у який можна становити вагони.

Стек застосовується досить часто, причому в самих різних ситуаціях. Поєднує їх наступна мета: потрібно зберегти деяку роботу, що ще не виконана до кінця, при необхідності перемикання на іншу задачу. Стек використовують для тимчасового збере-

ження стану не виконаного до кінця завдання. Після збереження стану комп'ютер перемикається на іншу задачу. По закінченні її виконання стан відкладеного завдання відновлюється із стека і комп'ютер продовжує перервану роботу.

Чому саме стік використовується для збереження стану перерваного завдання? Припустимо, що комп'ютер виконує задачу **A**. У процесі її виконання виникає необхідність виконати задачу **B**. Стан задачі A запам'ятовується, і комп'ютер переходить до виконання задачі B. Але й при виконанні задачі **B** комп'ютер може перемкнутися на іншу задачу **C** і потрібно буде зберегти стан задачі **B** перш, ніж перейти до **C**. Пізніше, по закінченню C, буде спочатку відновлений стан задачі **B**, потім, по закінченню **B** - стан задачі **A**. Таким чином, відновлення відбувається в порядку, зворотному збереженню, що відповідає дисципліні роботи стека.

Стек дозволяє організувати рекурсію, тобто звернення підпрограми до самої себе або безпосередньо, або через ланцюжок інших викликів. Припустимо, наприклад, що підпрограма **A** виконує алгоритм, що залежить від вхідного параметра **X** і, можливо, від стану глобальних даних. Для найпростіших значень X алгоритм реалізується безпосередньо. У випадку більш складних значень **X** алгоритм реалізується як відомість до застосування того ж алгоритму для більш простих значень **X**. При цьому підпрограма **A** звертається сама до себе, передаючи як параметр більш просте значення **X**. При такому зверненні попереднє значення параметра **X**, а також всі локальні змінні підпрограми A зберігаються в стеці. Далі створюється новий набір локальних змінних і змінна, утримуюча нове (більш просте) значення параметра **X**. Викликана підпрограма **A** працює з новим набором змінних, не руйнуючи попереднього набору. По закінченню виклику старий набір локальних змінних і старий стан вхідного параметра **X** відновлюються із стека і підпрограма продовжує роботу з того місця, де вона була перервана.

Насправді навіть не доводиться спеціальним образом зберігати значення локальних змінних підпрограми в стеці. Справа в тому, що локальні змінні підпрограми (тобто її внутрішні, робочі змінні, які створюються на початку її виконання та знищуються наприкінці) розміщають в стеці, реалізованому апаратно на базі звичайної оперативної пам'яті. На самому початку роботи підпрограма захоплює місце в стеці під свої локальні змінні, цю ділянку пам'яті в апаратному стеці називають блоком локальних змінних або frame ("кадр"). У момент закінчення роботи підпрограма звільняє пам'ять, видаляючи із стека блок своїх локальних змінних.

Крім локальних змінних, в апаратному стеці зберігаються адреси повернення при викликах підпрограм. Нехай у деякій точці програми **A** викликається підпрограма **B**. Перед викликом підпрограми **B** адреса інструкції, що викливає за інструкцією виклику **B**, зберігається в стеці. Це так звана адреса повернення в програму **A**. Після закінчення роботи підпрограма **B** виймає із стека адресу повернення в програму **A** і повертає керування по цій адресі. Таким чином, комп'ютер продовжує виконання програми **A**, починаючи з інструкції, що випливає за інструкцією виклику. У більшості процесорів є спеціальні команди, що підтримують виклик підпрограми з попереднім розміщенням адресу повернення в стек і повернення з підпрограми за адресою, що виймається із стека. Звичайно команда виклику називається call, команда повернення - return.

У стекці розміщують також параметри підпрограми або функції перед її викликом. Порядок їхнього розміщення в стеці залежить від угод, прийнятих у мовах високого рівня. Так, у мові **C** або **C++** на вершині стека знаходиться перший аргумент функції, під ним другий і так далі. У Паскалі все навпаки, на вершині стека перебуває останній аргумент функції. (Тому, до речі, у **C** можливі функції зі змінним числом аргументів, такі, як printf, а в Паскалі немає.). У мові **C#** для роботи із стеком створений спеціальний клас Stack, властивості й методи якого представлені в таблицях 4 і 5.

Таблиця 4

Властивість класу Stack

ім'я	Опис
Count	Повертає число елементів у стеці

Таблиця 5

Методи Stack

ім'я	Опис
Pop	Повертає елемент із вершини стека, одночасно видаляючи його
Push	Додає елемент на вершину стека
Peek	Повертає верхній елемент стека, не видаляючи його

У роботі розглядається створення спеціалізованого інтервального калькулятора, що реалізує зворотний польський запис із використанням стека для виконання фінансових розрахунків.

Стековий інтервальний калькулятор і зворотний польський запис формул. В 1920 р. польський математик Ян Лукашевич запропонував спосіб запису арифметичних формул, що не використовує дужок. У звичній для нас формі запису знак операції записують між аргументами, наприклад, сума чисел 2 і 3 записується як 2 + 3. Ян Лукашевич запропонував дві інші форми запису: префіксна форма, у якій знак операції записують перед аргументами, і постфіксна форма, у якій знак операції записують після аргументів. У префіксній формі сума чисел 2 і 3 записується як + 2 3, у постфіксній - як 2 3 +. На честь Яна Лукашевича ці форми запису називають прямою й зворотним польським записом.

У польському записі дужки не потрібні. Наприклад, вираз

(2+3)*(15-7)

записують в прямому польському записі як

* + 2 3 - 15 7,

у зворотному польському записі - як

2 3 + 15 7 - *.

У стековому інтервальному калькуляторі з ОПН для обчислення виразу

[2;3]+[3;4]-[1;3]

необхідно записати

2 3 3 4 + 1 3 -

Якщо прямий польський запис не отримав великого поширення, то зворотній виявився надзвичайно корисним. Неформально перевагу зворотного запису перед прямим польським записом або звичайним записом можна пояснити тим, що набагато зручніше виконувати деяку дію, коли об'єкти, над якими вона повинне бути зроблена уже надані.

Зворотний польський запис формули дозволяє обчислювати вираз будь-якої складності, використовуючи стек як пам'ять для збереження проміжних результатів. Звичайні моделі калькуляторів не дозволяють обчислювати складні формули без використання паперу і ручки для запису проміжних результатів. У деяких моделях є дужки з одним або двома рівнями вкладеності, але більш складні вираження обчислювати неможливо. Також у звичайних калькуляторах важко зрозуміти, як результат і аргументи переміщаються в процесі введення й обчислення між регістрами калькулятора. Калькулятор звичайно має регістри **X**, **Y** і регістр пам'яті, проміжні результати якимсь способом переміщаються по регістрах, яким саме - запам'ятати неможливо.

На відміну від інших калькуляторів, пристрій стекового калькулятора цілком зрозуміло та легко зберігається у пам'яті. Калькулятор має пам'ять у вигляді стека. При уведенні числа воно просто додається в стек. При натисканні на клавішу операції, наприклад, на клавішу +, аргументи операції спочатку виймаютьають із стека, потім з ними виконується операція, нарешті, результат операції повертається в стек. Таким чином, при виконанні операції із двома аргументами, наприклад, додавання, у стеці повинне бути не менш двох чисел. Аргументи видаляються зі стека і на їх місце записується результат, тобто при виконанні додавання глибина стека зменшується на одиницю. Вершина стека завжди містить результат останньої операції і відображається на дисплеї калькулятора.

Реалізація стекового інтервального калькулятора на C#. Розглянемо проект, що реалізує стековий інтервальний калькулятор на C#. Така програма досить корисна, оскільки дозволяє проводити обчислення, не прибігаючи до запису проміжних результатів на папері.

Програма складається із трьох файлів: "Form1.cs", "Form2.cs" і "Program.cs". Перші два файли реалізують стек для виконання обчислень, ця реалізація вже розглядалася раніше. Файл "Form1.cs" реалізує інтервальний стековий калькулятор на базі ОПН. Файл Form2.cs реалізує фінансові обчислення структур даних типу стек на ОПН. Файл Program.cs - основний файл проекту.

Нижче наведений зміст двох фрагментів файлу "Form1.cs". Функція main, описана в цьому файлі, організує діалог з користувачем у режимі команда-відповідь. Користувач може ввести число з клавіатури, це число просто додається в стек. При введенні одного із чотирьох знаків арифметичних операцій +, -, *, / програма виймає із стека два числа, виконує зазначену арифметичну дію над ними і вміщає результат в стек. Значення результату відображається також на дисплеї. Крім арифметичних операцій, користувач може ввести назву однієї зі стандартних функцій: a^x, x^a, exp(x), ln, log (десятковий логарифм), sqrt(x). При цьому програма виймає із стека аргумент функції, обчислює значення функції та вміщує його в стек. При бажанні список стандартних функцій і можливих операцій можна розширити. Кожна команда стекового калькулятора реалізується за допомогою окремої функції. Наприклад, додавання реалізується за допомогою функції AddIntOPN():

```
private void AddIntOPN_Click(object sender, EventArgs e)
{
   if (EntryInProgress)
      InitializeRegisterFromDisplay();
     if (RegStack1.Count >= 2)
  {
   double op4 = (double)RegStack1.Pop();
   double op3 = (double)RegStack1.Pop();
   double op2 = (double)RegStack1.Pop();
   double op1 = (double)RegStack1.Pop();
   double a = op1 + op3;
   double b = op2 + op4;
   string display1 = a.ToString(FormatString);
   string display2 = b.ToString(FormatString);
   textBox3.Text = display1;
   textBox4.Text = display2;
   RegStack1.Push(a);
   RegStack1.Push(b);
   Display1.Text = " ";
   Display1.Text = " ";
  }
 }
```

На початку функції перевіряються на те, що глибина стека не менше двох. У противному випадку, видається повідомлення про помилку і функція завершується. Далі із стека виймають операнди **y** і **x** операції віднімання. Елементи виймають із стека в порядку, зворотному їхньому розміщенню в стек, тому **y** витягає раніше, ніж **x**. Потім обчислюють різницю y - x, її значення розміщують знов у стек і відображають на дисплеї, для друку вершини стека викликається функція display.

Приведемо початковий фрагмент програми, що описує оголошення екземплярів класу Stack і змінних.

```
namespace Калькулятор_Інтервальний_ОПН
{
  public partial class IntSQRT : Form
  {
   public IntSQRT()
    {
     InitializeComponent();
      }
       Stack RegStack1 = new Stack();
       Stack RegStack2 = new Stack();
       string FormatString = "f2";
       const int MaxChars = 21;
       private bool FixPending = false;
       private bool DecimalInString = false;
       private bool EntryInProgress = false;
       private void
       InitializeRegisterFromDisplay()
        {
double x = (Display1.Text.Length == 0 || Display1.Text == ",") ? 0.0 :
Convert.ToDouble(Display1.Text);
RegStack1.Push(x);
double x1 = (Display2.Text.Length == 0 || Display2.Text == ",") ? 0.0 :
Convert.ToDouble(Display2.Text);
RegStack1.Push(x1);
  }
```

У таблиці 6 наведені порівняльні результати для виражень різного ступеня складності на звичайному калькуляторі, калькуляторі з ОПН і интервальному калькуляторі з ОПН.

При виконанні розрахунків для нарахування простих і складних відсотків, кількість операцій скоротилося на дві. При виконанні розрахунків на інтервальному стековому калькуляторі кількість операцій скоротилося на три. У такий спосіб з підвищенням складності розрахунків ефект від використання ОПН збільшується.

Таблиця 6.

Формули нарахування простих і складних відсотків

Звичайний калькулятор			
Дії на звичайному калькуляторі		Дії на калькуляторі з ОПН	
$S=P(1+rt)$	$S=P(1+r)^t$	$S=P(1+rt)$	$S=P(1+r)^t$
0,1	1	0,1	0,1
*	+	Введення	Введення
5	0,1	5	1
=	=	*	+
+	x^y	1	5
1	5	+	a^x

Інтервальний калькулятор	
Дії на інтервальному калькуляторі з ОПН	
$[S_н;S_к]=[P_н;P_к]\cdot(1+[r_н;r_к])\cdot[t_н;t_к]$	$[S_н;S_к]=[P_н;P_к]\cdot$ $\cdot(1+[r_н;r_к])^{[t_н;t_к]}$
[0,1; 0,1] Уведення	[1;1] Уведення
[5,0; 5,1] Уведення	[0,1;0,1] Уведення
Множення інтервальне	Додавання інтервальне
[1;1] Уведення	[5;5,1] Уведення
Додавання інтервальне	Піднесення в ступінь інтервальне a^x
[1000;1100] Уведення	[1000;1100] Уведення
Множення інтервальне [1500;1717,1]	Множення інтервальне [1610,51;1873,01]

S- накопичена сума, P - сума внеску, r - процентна ставка, t - строк внеску.
P=1000, r=0.1, t=5.

S - накопичена сума ренти, r - розмір члена ренти,

i - річна процентна ставка, n - термін ренти (у роках)

Особливо ефективно використання ОПН для виконання в дистанційному режимі розрахунків, пов'язаних з визначенням дійсної вартості бізнесу. Це пов'язано з необхідністю виконання великої кількості розрахунків. Скорочення довжини програми за рахунок ОПН дає можливість підвищити надійність програми та скоротити час приймання рішення. Нижче наведено Web-сторінку з розрахунками вартісного еквіваленту зникнення специфічних ризиків [6]:

$$RP = \frac{CF_1 \times \Delta pr}{(r-g)(r-g+\Delta pr)},$$

де:

- CF_1 – сальдо грошових потоків, очікуваних в найближчому році;
- r – ставка дисконтування грошових потоків з урахуванням ризиків, прийнята для сектора, в якому функціонує оцінюваємий бізнес;
- g – очікуваний постійний темп росту грошових потоків в безстроковій перспективі;
- Δpr- різниця між преміями за ризики інвестування у новий створюваний бізнес і вже створений і стабільно функціонуючий.
-

$$\Delta pr = \frac{(p_n - p_m)(1+r_f)}{1-p_n-p_m},$$

де:

- r_f– безризикова ставка;
- p_n - ймовірність банкрутства нового бізнесу;
- p_m - ймовірність банкрутства стабільно функціонуючого бізнесу.

Приймемо, що r-g =u, тоді

$$RP = \frac{CF_1 \times \Delta pr}{u(u+\Delta pr)}.$$

Саме цей вираз реалізований в приведеному калькуляторі. (рис.4).

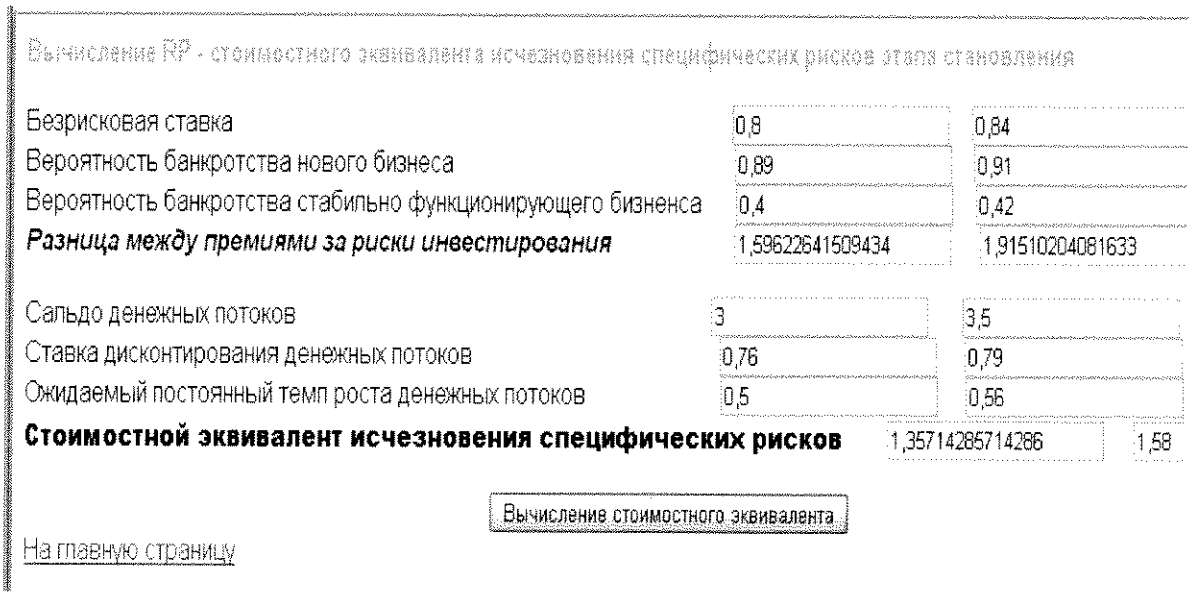

Рис.4. Вікно Web-сторінки з результатами розрахунків.

Напрямки подальших досліджень. У зв'язку зі збільшенням об'єму фінансових обчислень і з метою підвищення відповідальності за їхні результати до перспективних напрямків подальших робіт варто віднести організацію схеми дистанційного доступу до обчислювальних ресурсів й створення спеціалізованих мереж, призначених для аналізу фінансових даних.

Висновки. 1.Калькулятор дає можливість будь-який арифметичний вираз, елементи якого належать до поля інтервальних чисел обчислити без виконання користувачем проміжних дій.

2. Показано ефективність застосування інтервальних обчислень для рішення задач пов'язаних з тестування програм в умовах, у яких застосування традиційних методів прогнозування неможливо або ускладнено відсутністю відомостей про статистичні властивості змінних.

3. Наведено відомості про спеціалізований програмний стековий калькулятор, що реалізує правила інтервальної арифметики для оцінювання ефективності тестування комп'ютерних програм.

4. Показано, що застосування зворотного польського запису дозволяє скоротити об'єм програми, отже, підвищити надійність.

Література

1. Вентцель Е.С.Теория вероятностей : учебник./Е.С. Вентцель. М : Гос. изд-во физ-мат. лит-ры,1962. - 560 с.

2. Алефельд Г., Херцбергер Ю. Введение в интервальные вычисления. / Г. Алефельд,Ю. Херцбергер – М. : Мир, 1987. – 259с.

3. Алтунин А.Е. Семухин М.В., Модели и алгоритмы принятия решений в нечетких условиях./А.Е. Алтунин, М. В. Семухин.- Тюмень : Изд ТГУ, 2000.- 352с.

4. Харченко В.С., Скляр В.В., Тарасюк О.М. Методы моделирования и оценки качества и надежности программного обеспечения /В.С. Харченко, В.В. Скляр, О.М. Тарасюк. – Учеб. пособие. – Харьков : Нац. аэрокосм. ун-т "Харьк. авиац. ин-т", 2004. – 159 с.

5. Пайчун Б.П., Юсупов Р.М Оценка надежности программного обеспечения. ./ Б.П. Пайчун, Р.М. Юсупов. – Спб. : Наука, 1994. – 84с.

6. Козырь Ю.В. Стоимость компаний.Оценка и управленческое решение./Ю.В.Козырь .-М. : Изд. Альфа-пресс, 2009 с.369.

Summary. A method is proposed for evaluation of parameters that determine software reliability. To this end the simplest algebraic equations were studied and solved with their coefficients belonging to the set of interval numbers. The developed software permits calculation by selecting the names of variables coinciding with the names of indices from corresponding lists, introduction of initial data, selection of operation type, performance of calculations, storage of results on the sheet of electronic table and their graphic presentation, if necessary.

Keywords: interval numbers, software reliability, reversed Polish notation, stack interval calculator, Halstead model, Mills model.

Стаття надійшла до редакції 13.03.2010