УДК 510.5:336

**В.Ю. ДУБНИЦКИЙ[1], А.М. КОБЫЛИН[1], О.А. КОБЫЛИН[2]**

*[1]Харьковский институт банковского дела Университета банковского дела НБ Украины*
*[2]Харьковский национальный университет радиоэлектроники, Украина*

# ОБРАТНАЯ ПОЛЬСКАЯ ЗАПИСЬ АЛГОРИТМА КАК СРЕДСТВО ПОВЫШЕНИЯ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ БИЗНЕС-КРИТИЧЕСКИХ СИСТЕМ

*Приведен аналитический обзор экспериментальных исследований по использованию обратной польской записи в обычном калькуляторе и программируемом специализированном стековом калькуляторе для финансовых вычислений с использованием класса Stack на языке программирования C# в среде программирования Visual Studio .NET 2008. Отмечено, что количество операций в стековом интервальном калькуляторе, реализующем вычисления с использованием обратной польской записи, уменьшились по сравнению с программами, использующими традиционный способ записи формул.*

***Ключевые слова:*** *обратная польская запись, интервальные вычисления, стек, интервальный стековый калькулятор.*

## Введение

Одной наиболее важной из структур данных являются стек. Эта структуры встречаются в программировании буквально на каждом шагу, в самых разнообразных ситуациях. Особенно интересен стек, который имеет самые неожиданные применения. Следует отметить, что в первых советских электронных программируемых калькуляторах, таких, как Б3-19М, Б3-21, Б3-26, МК-61 была использована так называемая «обратная польская запись». Эта форма представления алгоритма была в последствии незаслуженно забыта. Об эффективности такого способа нотации формул можно судить по тому, что на этих калькуляторах были программно реализованы вычисления основных специальных функций математической физики [1].

В свое время при разработке серии ЭВМ IBM 360 в начале 70-х годов XX века фирма IBM совершила драматическую ошибку, не предусмотрев аппаратную реализацию стека. Эта серия содержала много других неудачных решений, но, к сожалению, была скопирована в Советском Союзе под названием ЕС ЭВМ (Единая Серия), а все собственные разработки были приостановлены. Это отбросило советскую промышленность на много лет назад в области разработки компьютеров.

**Анализ литературы./** RPN является сокращением от Reverse Polish Notation (обратная польская запись, обратная польская нотация или ОПН) Обратная польская запись была предложена в 1920 году Яном Лукасевичем как способ записи математического выражения без использования круглых и квадратных скобок. Компанией Hewlett-Packard Co. Метод Лукасевича был признан идеальным для решения стандартных алгебраических выражений на калькуляторах или компьютерах, поэтому метод обратной польской записи был использован для создания первого научного карманного калькулятора HP35 в 1972 году [2, 3].

**Постановка задачи.** Целью предлагаемой работы было создание специализированного интервального калькулятора, реализующего обратную польскую запись с использованием стека для выполнения финансовых расчетов. Для этого требовалось выполнение анализ проблемы использования структуры данных типа стек, разработка методов программной реализации обратной польской записи, проектирование и разработка специализированного программного стекового калькулятора для финансовых расчетов.

## Изложение результатов

Стек — самая популярная и, пожалуй, самая важная структура данных в программировании. Стек представляет собой запоминающее устройство, из которого элементы извлекаются в порядке, обратном их добавлению. Это как бы неправильная очередь, в которой первым обслуживают того, кто встал в нее последним. В программистской литературе общепринятыми являются аббревиатуры, обозначающие дисциплину работы стека.

Дисциплина работы стека обозначается LIFO, последним пришел — первым уйдешь (Last In First Out).

Стек можно представить в виде трубки с подпружиненным дном, расположенной вертикально. Верхний конец трубки открыт, в него можно добавлять, или, как говорят, заталкивать элементы. Общепринятые английские термины в этом плане очень красочны, операция добавления элемента в стек обозначается push, в переводе "затолкнуть, запихнуть". Новый добавляемый элемент проталкивает элементы, помещенные в стек ранее, на одну позицию вниз. При извлечении элементов из стека они как бы выталкиваются вверх, по-английски pop ("выстреливают").

Примером стека может служить стопка бумаг на столе, стопка тарелок и т.п. Отсюда произошло название стека, что по-английски означает стопка. Тарелки снимаются со стопки в порядке, обратном их добавлению. Доступна только верхняя тарелка, т.е. тарелка на вершине стека. Хорошим примером будет также служить железнодорожный тупик, в который можно составлять вагоны.

## Использование стека в программировании

Стек применяется довольно часто, причем в самых разных ситуациях. Объединяет их следующая цель: нужно сохранить некоторую работу, которая еще не выполнена до конца, при необходимости переключения на другую задачу. Стек используется для временного сохранения состояния не выполненного до конца задания. После сохранения состояния компьютер переключается на другую задачу. По окончании ее выполнения состояние отложенного задания восстанавливается из стека, и компьютер продолжает прерванную работу.

Почему именно стек используется для сохранения состояния прерванного задания? Предположим, что компьютер выполняет задачу **A**. В процессе ее выполнения возникает необходимость выполнить задачу **B**. Состояние задачи **A** запоминается, и компьютер переходит к выполнению задачи **B**. Но ведь и при выполнении задачи **B** компьютер может переключиться на другую задачу **C**, и нужно будет сохранить состояние задачи **B**, прежде чем перейти к **C**. Позже, по окончании **C** будет вначале восстановлено состояние задачи **B**, затем, по окончании **B**, — состояние задачи **A**. Таким образом, восстановление происходит в порядке, обратном сохранению, что соответствует дисциплине работы стека.

Стек позволяет организовать рекурсию, т.е. обращение подпрограммы к самой себе либо непосредственно, либо через цепочку других вызовов. Пусть, например, подпрограмма **A** выполняет алгоритм, зависящий от входного параметра **X** и, возможно, от состояния глобальных данных. Для самых простых значений **X** алгоритм реализуется непосредственно. В случае более сложных значений **X** алгоритм реализуется как сведение к применению того же алгоритма для более простых значений **X.** При этом подпрограмма **A** обращается сама к себе, передавая в качестве параметра более простое значение **X.** При таком обращении предыдущее значение параметра **X**, а также все локальные переменные подпрограммы **A** сохраняются в стеке. Далее создается новый набор локальных переменных и переменная, содержащая новое (более простое) значение параметра **X.** Вызванная подпрограмма **A** работает с новым набором переменных, не разрушая предыдущего набора. По окончании вызова старый набор локальных переменных и старое состояние входного параметра **X** восстанавливаются из стека, и подпрограмма продолжает работу с того места, где она была прервана.

На самом деле даже не приходится специальным образом сохранять значения локальных переменных подпрограммы в стеке. Дело в том, что локальные переменные подпрограммы (т.е. ее внутренние, рабочие переменные, которые создаются в начале ее выполнения и уничтожаются в конце) размещаются в стеке, реализованном аппаратно на базе обычной оперативной памяти. В самом начале работы подпрограмма захватывает место в стеке под свои локальные переменные, этот участок памяти в аппаратном стеке называют обычно **блок локальных переменных** или по-английски **frame** ("кадр"). В момент окончания работы подпрограмма освобождает память, удаляя из стека блок своих локальных переменных.

Кроме локальных переменных, в аппаратном стеке сохраняются адреса возврата при вызовах подпрограмм. Пусть в некоторой точке программы **A** вызывается подпрограмма **B**. Перед вызовом подпрограммы **B** адрес инструкции, следующей за инструкцией вызова **B**, сохраняется в стеке. Это так называемый **адрес возврата** в программу **A**. По окончании работы подпрограмма **B** извлекает из стека адрес возврата в программу **A** и возвращает управление по этому адресу. Таким образом, компьютер продолжает выполнение программы **A**, начиная с инструкции, следующей за инструкцией вызова. В большинстве процессоров имеются специальные команды, поддерживающие вызов подпрограммы с предварительным помещением адреса возврата в стек и возврат из подпрограммы по адресу, извлекаемому из стека.

В стек помещаются также параметры подпрограммы или функции перед ее вызовом. Порядок их

помещения в стек зависит от соглашений, принятых в языках высокого уровня. Так, в языке Си или C++ на вершине стека лежит первый аргумент функции, под ним второй и так далее. В Паскале наоборот, на вершине стека лежит последний аргумент функции. (Поэтому, кстати, в Си возможны функции с переменным числом аргументов, такие, как `printf`, а в Паскале нет.). В языке C# для работы со стеком создан специальный класс Stack, свойства и методы которого представлены в табл. 1 и 2.

Таблица 1

Свойство класса Stack

| Имя | Описание |
|---|---|
| Count | Возвращает число элементов в стеке |

Таблица 2

Методы Stack

| Имя | Описание |
|---|---|
| Pop | Возвращает элемент с вершины стека, одновременно удаляя его |
| Push | Добавляет элемент на вершину стека |
| Peek | Возвращает верхний элемент стека, не удаляя его |

### Стековый интервальный калькулятор и обратная польская запись формулы

В 1920 г. польский математик Ян Лукасевич предложил способ записи арифметических формул, не использующий скобок.[2, 3] В привычной нам записи знак операции записывается между аргументами, например, сумма чисел 2 и 3 записывается как `2 + 3`. Ян Лукашевич предложил две другие формы записи: префиксная форма, в которой знак операции записывается перед аргументами, и постфиксная форма, в которой знак операции записывается после аргументов. В префиксной форме сумма чисел 2 и 3 записывается как `+ 2 3`, в постфиксной – как `2 3 +`. В честь Яна Лукасевича эти формы записи называют прямой и обратной польской записью(ОПЗ).

В польской записи скобки не нужны. Например, выражение

```
(2+3)*(15-7)
```

записывается в прямой польской записи как

```
* + 2 3 - 15 7,
```

в обратной польской записи — как

```
2 3 + 15 7 - *.
```

В стековом интервальном калькуляторе с ОПЗ для вычисления выражения

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

необходимо записать

2 3 3 4 + 1 3 -

Если прямая польская запись не получила большого распространения, то обратная оказалась чрезвычайно полезной. Неформально преимущество обратной записи перед прямой польской записью или обычной записью можно объяснить тем, что гораздо удобнее выполнять некоторое действие, когда объекты, над которыми оно должно быть совершено, уже даны.

Обратная польская запись формулы позволяет вычислять выражение любой сложности, используя стек как запоминающее устройство для хранения промежуточных результатов. Обычные модели калькуляторов не позволяют вычислять сложные формулы без использования бумаги и ручки для записи промежуточных результатов. В некоторых моделях есть скобки с одним или двумя уровнями вложенности, но более сложные выражения вычислять невозможно. Также в обычных калькуляторах трудно понять, как результат и аргументы перемещаются в процессе ввода и вычисления между регистрами калькулятора. Калькулятор обычно имеет регистры `X`, `Y` и регистр памяти, промежуточные результаты каким-то образом перемещаются по регистрам, каким именно — запомнить невозможно.

В отличие от других калькуляторов, устройство стекового калькулятора вполне понятно и легко запоминается. Калькулятор имеет память в виде стека. При вводе числа оно просто добавляется в стек. При нажатии на клавишу операции, например, на клавишу +, аргументы операции сначала извлекаются из стека, затем с ними выполняется операция, наконец, результат операции помещается обратно в стек. Таким образом, при выполнении операции с двумя аргументами, например, сложения, в стеке должно быть не менее двух чисел. Аргументы удаляются из стека и на их место записывается результат, то есть при выполнении сложения глубина стека уменьшается на единицу. Вершина стека всегда содержит результат последней операции и высвечивается на дисплее калькулятора.

Для вычисления выражения надо сначала преобразовать его в обратную польскую запись (при некотором навыке это легко сделать в уме). В приведенном выше примере выражение `(2+3)*(15-7)` преобразуется к

```
2 3 + 15 7 - *.
```

Затем обратная польская запись просматривается последовательно слева направо. Если мы видим число, то просто вводим его в калькулятор, т.е. добавляем его в стек. Если мы видим знак операции, то нажимаем соответствующую клавишу калькулятора, выполняя таким образом операцию с числами на вершине стека.

Изобразим последовательные состояния стека калькулятора при вычислении по приведенной фор-

муле. Сканируем слева направо ее обратную польскую запись:

```
2 3 + 15 7 - *
```

Стек вначале пуст. Последовательно добавляем числа 2 и 3 в стек.

```
|    | вводим число 2 → | 2 | вво-
дим число 3 →  | 3 |
               | 2 |
```

Далее читаем символ + и нажимаем на клавишу “+” на калькулятора. Числа 2 и 3 извлекаются из стека, складываются, и результат помещается обратно в стек.

```
| 3 | выполняем сложение → | 5 |
| 2 |
```

Далее, в стек добавляются числа 15 и 7.

```
| 5 | вводим число 15 → | 15 |
вводим число 7
                              | 7  |
                              | 15 |
                              | 5  |
```

Читаем символ – и нажимаем на клавишу “-“ на калькулятора. Со стека при этом снимаются два верхних числа 7 ; 15 и выполняется операция вычитания. Причем уменьшаемым является то число, которое было введено раньше, а вычитаемым — число, введенное позже. Иначе говоря, при выполнении некоммутативных операций, таких как вычитание или деление, правым аргументом является число на вершине стека, левым — число, находящееся под вершиной стека.

```
| 7 | выполняем вычитание → | 8 |
|15 |
| 5 |
```

Наконец, читаем символ * и нажимаем на клавишу “*” на калькулятора. Калькулятор выполняет умножение, со стека снимаются два числа, перемножаются, результат помещается обратно в стек.

```
| 8 | выполняем умножение → | 40 |
| 5 |
```

Число 40 является результатом вычисления выражения. Оно находится на вершине стека и высвечивается на дисплее стекового калькулятора.

### Реализация стекового калькулятора на C#

Рассмотрим проект, реализующий стековый калькулятор на C#. Такая программа весьма полезна, поскольку позволяет проводить вычисления, не прибегая к записи промежуточных результатов на бумаге.

Программа состоит из трех файлов: "Form1.cs", "Form2.cs" и "Program.cs". Первые два файла реализуют стек вещественных чисел, эта реализация уже рассматривалась ранее. Файл "Form1.cs" реализует интервальный стековый калькулятор на базе ОПН. Файл Form2.cs реализует финансовые вычисления структур данных типа стек на ОПН. Файл Program.cs – основной файл проекта.

Ниже приведено содержимое двух фрагментов файла "Form1.cs". Функция `main`, описанная в этом файле, организует диалог с пользователем в режиме команда-ответ. Пользователь может ввести число с клавиатуры, это число просто добавляется в стек. При вводе одного из четырех знаков арифметических операций +, –, *, / программа извлекает из стека два числа, выполняет указанное арифметическое действие над ними и помещает результат обратно в стек. Значение результата отображается также на дисплее.

Кроме арифметических операций, пользователь может ввести название одной из стандартных функций: $a^x$, $x^a$, exp(x), ln, log (натуральный логарифм), sqrt(x). При этом программа извлекает из стека аргумент функции, вычисляет значение функции и помещает его обратно в стек. При желании список стандартных функций и возможных операций можно расширить.

Наконец, можно выполнять еще несколько команд (табл. 3).

Таблица 3

Примеры расширения команд

| | |
|---|---|
| **pop** | удалить вершину стека; |
| **clear** | очистить стек; |
| **=** | напечатать вершину стека; |
| **show** | напечатать содержимое стека; |
| **help** | напечатать подсказку; |
| **quit** | завершить работу программы. |

Каждая команда стекового калькулятора реализуется с помощью отдельной функции. Например, сложение реализуется с помощью функции 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 = " ";}}
```

В начале функции проверяется, что глубина стека не меньше двух. В противном случае, выдается сообщение об ошибке, и функция завершается. Далее из стека извлекаются операнды `y` и `x` операции сложения. Элементы извлекаются из стека в порядке, обратном их помещению в стек, поэтому `y` извлекается раньше, чем `x`. Затем вычисляется сумма `y + x`, ее значение помещается обратно в стек и печатается на дисплее, для печати вершины стека вызывается функция `display`.

Приведем начальный фрагмент программы, описывающий объявление экземпляров класса Stack и переменных.

```
    namespace Калькулятор_Интервальный_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);}
```

Для практической реализации изложенного подхода были приняты численные алгоритмы, используемые в задачах финансовой математики и описанные ранее авторами в работе [4].

В табл. 4 – 7 приведены сравнительные результаты для выражений различной степени сложности на обычном калькуляторе и калькуляторе с ОПН.

Так, при выполнении простых действий, количество операций сократилось на 1. При выполнении расчетов для начисления простых и сложных процентов, количество операций сократилось на две. При выполнении расчетов для расчета годовой ренты, количество операций сократилось на три. Таким образом с повышением сложности расчетов эффект от использования ОПН увеличивается.

Таблица 4

Выполнение арифметических действий

| Действия на обычном калькуляторе | | Действия на калькуляторе с ОПН | |
|---|---|---|---|
| Вычислить 2+2 | Вычислить ((2+2)*6)/3 | Вычислить 2+2 | Вычислить ((2+2)*6)/3 |
| 2 | 2 | 2 | 2 |
| + | + | Ввод | Ввод |
| 2 | 2 | 2 | 2 |
| = | = | + | + |
| | * | | 6 |
| | 6 | | * |
| | / | | 3 |
| | 3 | | / |
| | = | | |

Таблица 5

Формулы начисления простых и сложных процентов

| Действия на обычном калькуляторе | | Действия на калькуляторе с 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 |
| * | + | Ввод | Ввод |
| 5 | 0,1 | 5 | 1 |
| = | = | * | + |
| + | x^y | 1 | 5 |
| 1 | 5 | + | a^x |
| = | = | 1000 | 1000 |
| * | * | * | * (1610,51) |
| 1000 | 1000 | | |
| = | = (1610,51) | | |

В табл. 5: S – накопленная сумма, P – сумма вклада, r – процентная ставка, t – срок вклада.

P=1000, r=0.1, t=5.

Годовая рента равна

$$\mathbf{S} = \mathbf{R} \cdot \frac{(1+\mathbf{i})^{\mathbf{n}} - 1}{\mathbf{i}} = \mathbf{R} \cdot S_{n,i},$$

где ***S*** – накопленная сума ренты,

**R** – размер члена ренты,

**i** – годовая процентная ставка ,

**n** – срок ренты (в годах)

$S_{n;i}$ – коэффициент накопления ренты.

В табл. 7 приведены сравнительные результаты выполнении расчетов для начисления простых и сложных процентов с использованием интервального калькулятора, реализующего вычисления с обратной польской записью.

Таблица. 6

Вычисление накопленной суммы ренты

| Действия на калькуляторе с ОПН | |
|---|---|
| $S = R\frac{(1+i)^n - 1}{i}$ | $S = R\frac{(1+i)^n - 1}{i}$ |
| 1 | 1 |
| + | Ввод |
| 0.1 | 0,1 |
| = | + |
| x^y | 5 |
| 5 | a^x |
| = | 1 |
| - | - |
| 1 | 0.1 |
| = | / |
| / | 1000 |
| 0.1 | * |
| * | |
| 1000 | |
| = | |

Таблица 7

Формулы для простых и сложных процентов интервального калькулятора с ОПН.

| $S=P(1+rt)$ | $S = P(1+r)^t$ |
|---|---|
| [0,1; 0,1] Ввод | [1;1] Ввод |
| [5,0; 5,1] Ввод | [0,1;0,1] Ввод |
| Умножение интервальное | Сложение интервальное |
| [1;1] Ввод | [5;5,1] Ввод |
| Сложение интервальное | Интервальное a^x |
| [1000;1100] Ввод | [1000;1100] Ввод |
| Умножение интервальное [1500;1717,1] | Умножение интервальное [1610,51;1873,01] |

Как видим, количество операций в интервальном калькуляторе, реализующего вычисления с использованием обратной польской записи нею, не увеличилось, а даже сократилось на одну для тех же вычислений.

На рис. 1, 2 представлен внешний вид интервального калькулятора с результатами вычислений по вышеприведенным формулам.

Сравнение количества операторов, используемых при программировании основных задач финансовой математики (будущая стоимость ренты, приведенная стоимость ренты, приведенная стоимость инвестиций, оценка эффективности операций с ценными бумагами, оценка эффективности валютных операций) показало, что применение ОПН позволяет сократить длину программы на 17…20 процентов.

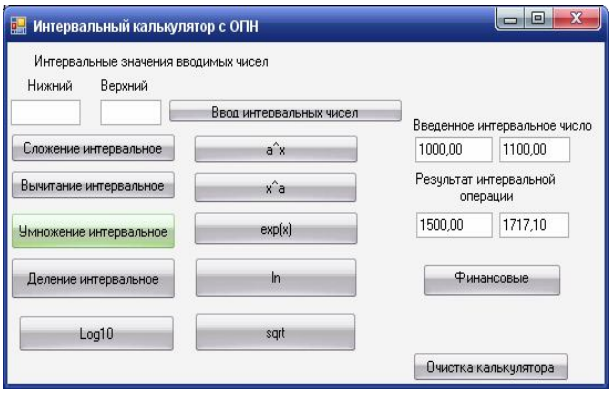


Рис. 1. Результаты расчета с простыми процентами


Рис. 2. Результаты расчета со сложными процентами

В соответствии с моделью Холстеда [5 – 7] количество ошибок в программе после окончания ее разработки линейно зависит от длины программы. Таким образом, использование ОПН позволяет без дополнительных затрат повысить надежность программного обеспечения примерно на 17…20 процентов.

## Выводы

На примере специализированного программного калькулятора для решения задач финансовой математики показано, что применение обратной польской записи для изображения формул позволяет повысить надежность вычислительного процесса без существенных дополнительных затрат.

## Литература

*1. Цимринг Ш.Е. Специальные функции и определенные интегралы. Алгоритмы. Программы для микрокалькуляторов: Справочник. / Ш.Е. Цимринг. – М.: Радио и связь, 1988. – 272 с.*

*2. Лебедев В.Н. Введение в системы программирования / В.Н. Лебедев. – М., 1975. –338 с.*

*3. Ахо А.В. Компиляторы: принципы, технологии и инструменты / А.В. Ахо, Р. Сети, Д. Д. Ульма. – М.: Вильямс, 2003. – 510 с.*

*4. Дубницкий В.Ю. Система дистанционного оценивания интервальной надежности программного обеспечения, предназначенного для выполнения финансовых расчетов./ В.Ю. Дубницкий, А.М. Кобылин // Радіоелектронні і комп'ютерні системи – 2008. – № 6 (33). – С. 186-192.*

*5. Пальчун Б.П. Оценка надежности программного обеспечения./ Б.П. Пальчун, Р.М Юсупов. –СПб.: Наука, 1994. – 84 с.*

*6. ДСТУ 2850-94 (ISO/IEC 9126-91 E) Программные средства ЭВМ. Показатели и методы оценки качества: Введ. 01.01.1994. – К.: Госстандарт Украины, 1994. – 33 с.*

*7. ДСТУ 3524-97. Проектна оцінка надійності складних систем з урахуванням технічного і програмного забезпечення та оперативного персоналу. Чинний від 01.01.1998. – К.: Держстандарт України, 1997.– 20 с.*

*Поступила в редакцию 16.01.2010*

**Рецензент:** д-р техн. наук, проф., зав. кафедрой информатики Е.П. Путятин, Харьковский национальный университет радиоэлектроники, Харьков, Украина.

## ЗВОРОТНИЙ ПОЛЬСЬКИЙ ЗАПИС АЛГОРИТМУ ЯК ЗАСІБ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ БІЗНЕС – КРИТИЧНИХ СИСТЕМ

***В.Ю. Дубницький, А.М.Кобилін, О.А. Кобилін***

Наведено аналітичний огляд експериментальних досліджень по використанню зворотного польського запису у звичайному калькуляторі і спеціалізованому стековому калькуляторі для фінансових обчислень з використанням класу Stack на мові програмування C# в середовищі програмування Visual Studio .NET 2008. Відмічено, що кількість операцій в програмованому стековому інтервальному калькуляторі, що реалізовує обчислення з використанням зворотного польського запису зменшилося в порівнянні з програмами, що використовують традиційний спосіб запису формул.

**Ключові слова**: зворотний польський запис, інтервальні обчислення, стік, інтервальний стековий калькулятор.

## REVERSE POLISH ALGORYTHM RECORD AS A MEAN FOR INCREASING RELIABILITY OF BUSINESS CRITICAL SYSTEMS SOFTWARE

***V.Yu. Dubnitsky, A.M. Kobylin, O.A. Kobylin***

Analytical review of experimental research of Reverse Polish Record application in an ordinary calculator and programmed special stack calculator for financial calculations using Stack class in C# in Visual Studio .NET 2008 environment. It is noted that the number of operations in stack interval calculator that calculates with the help of Reverse Polish Record decreased as compared to the programs that use the traditional way of recording formulas.

**Key words**: Reverse Polish Record, interval calculations, Stack, interval stack calculator.

**Дубницкий Валерий Юрьевич –** канд. техн. наук, ст. науч. сотрудник, доцент кафедры высшей математики Харьковского института банковского дела Университета банковского дела НБУ, Украина, e-mail:valeriy_dubn@mail.ru.

**Кобылин Анатолий Михайлович –** канд. техн. наук, доцент, доцент кафедры информационных технологий Харьковского института банковского дела Университета банковского дела, Харьков, Украина, e-mail: kobilin@khibs.edu.ua.

**Кобылин Олег Анатольевич –** канд. техн. наук, доцент кафедры информатики Харьковского национального университета радиоэлектроники, Харьков, Украина, e-mail: kblin@mail.ru.