Тема: Підпрограми.

Мета: Навчитися працювати із підпрограмами, широко використовуючи мовні засоби; розглянути ситуації, які потребують при написанні програмного коду використання процедур та функцій, ознайомитися із влаштованими підпрограмами для роботи із рядковими типами даних. Приділити увагу роботі із графічними можливостями Delphi.

1. Поняття підпрограми. Оголошення підпрограми, її тіло та оператор її виклику.

Уявимо собі наступну ситуацію: в нашій програмі одна й та ж сама послідовність дій повторюється та прописана в різних її частинах. У чому недоліки такого дублювання? Перш за все, більш об'ємним стає власне програмний код. Але не це є головним. Справа в тому, що пряме копіювання фрагментів коду викликає ускладнення при подальшій модифікації коду програми – необхідно вносити зміни в різні частини програми. Щоб запобігти такому нераціональному способу роботи, доцільно використовувати підпрограми.

Підпрограма являє собою набір операторів та команд, оформлених спеціальним чином. Підпрограму можна викликати із основної програми, причому необмежену кількість разів. Виносячи деякий код у підпрограму, ми усуваємо його дублювання у програмі і, відповідно, зменшуємо загальний обсяг коду програми. Використання підпрограм надає додатку більш структурованої форми. В певній мірі підпрограми спрощують і читання коду іншим користувачам. Більш того, використання підпрограм дозволяє організувати поділ праці при роботі над великим проектом – кожен працює над своєю задачею, а всі результати швидко та просто підключаються до основної програми.

У мові Delphi підпрограми оформляються у вигляді процедур та функцій. Ім'я процедури та функції задається у відповідності із правилами створення імен ідентифікаторів (змінних). Воно може складатись лише із латинських літер, цифр та знаку підкреслювання, при цьому не може розпочинатися із цифри. Імена повинні бути унікальними – в одній програмі не може існувати декілька підпрограм із однаковими іменами.

Таким чином, підпрограмою називається поіменована логічно завершена група операторів, яку можна викликати по імені (тобто, виконати) довільну кількість разів із різних частин програми. За структурою підпрограма практично ідентична самій програмі – вона містить заголовок, блок опису, блок реалізації. В загальному випадку робота із підпрограмою здійснюється у два етапи. Спочатку потрібно описати процедуру (функцію), у протилежному випадку основна програма її просто не відшукає. Після того, як процедура (функція) описана, її можна викликати із основної програми. При цьому, звичайно, ніхто не заважає редагувати підпрограму та програму паралельно. Таким чином, ми можемо спочатку просто описати підпрограму та «навісити» на неї найпростішу дію типу виведення віконця із повідомленням, потім прописати її виклик у всіх потрібних місцях, протестувати працездатність програми, після чого продовжити написання підпрограми.

Відмінності у використанні підпрограм та функцій носять принциповий характер.

Функції використовуються у тому випадку, коли підпрограма повинна повернути (обрахувати та вернути) одне значення скалярного типу. При цьому функція може використовуватися у виразах у ролі операнда, на місці якого представляється результат роботи цієї функції.

Якщо нічого не потрібно повертати або ж необхідно повернути більше одного результату, то використовують процедури.

2. Поняття функції. Створення та використання власних функцій.

Вивчення підпрограм розглянемо на прикладі функції. В загальному виді опис функції має наступний вигляд:

Function <ім'я функції> (<список формальних параметрів>): <тип результату>;
Const …;
Type …;
Var …;
Begin
<оператори>
End;

Необхідно написати функцію, яка визначає суму двох дійсних чисел.

Наведемо два способи повернення обрахованих значень – обидві функції ідентичні за своїм змістом.

1 варіант:

```
Function sum (a, b: real): real;
Begin
 Sum:=a+b;
End;
```

2 варіант:

```
Function sum (a, b: real): real;
Begin
 Result:=a+b;
End;
```

Правила опису та використання функцій:

1. Заголовок функції розпочинається зі службового слова function, за яким вказується ім'я функції. У наведених прикладах – sum.
2. За іменем функції в круглих дужках перераховуються вхідні параметри та їх типи (в даних прикладах два параметри a, b типу real).
3. Якщо функція не має формальних параметрів, то круглі дужки в заголовку не ставляться.
4. Після дужок, в яких поміщені вхідні параметри, через двокрапку вказують тип значення, яке повертає функція. Функція призначена для обрахунку лише одного значення, яке передається у програму через ім'я функції. Причому тип значення, яке повертається, - це довільний скалярний тип даних (числовий, символьний, булевий, рядковий чи вказівник).
5. Всередині функції перед її виконуваною частиною можуть бути оголошені змінні, які називаються локальними. Ці змінні поза функцією не існують,

оскільки для них відводиться пам'ять при кожному виклику функції, а при виході із функції ці змінні знищуються.

6. Всі змінні, описані у програмі до оголошення функції, також доступні всередині функції. Ці змінні називаються глобальними. Для унеможливлення виникнення помилок доступні змінні всередині функції використовувати не рекомендується!
7. Тіло функції розміщується поміж begin … end; та являє собою набір команд.
8. Для повернення в програму результату функції ім'я функції у виконуваній частині повинно отримати своє значення. З цією метою можна також застосовувати внутрішню змінну Result. Перевага використання такої змінної полягає в тому, що вона може брати участь у виразах як операнд. Наприклад, для функції знаходження суми можна використати присвоєння result:=result+a, проте недопустимо використовувати присвоєння sum:=sum+a.

Виклик функції використовується всередині основної програми (або іншої функції). Наприклад, для записаної функції sum можливий наступний виклик:

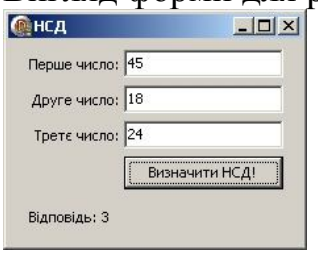
```
Var x, y, s: real;
Begin
 // введення змінних x та y
 S:=sum(x,y);
 // виведення змінної S
End;
```

Приклад. Знайти найбільший спільний дільник (НСД) трьох цілих чисел.

Розв'язання. Врахуємо, що НСД (a, b, c) = НСД (НСД (a, b), c)

Як алгоритм визначення найбільшого спільного дільника для двох чисел скористаємось алгоритмом Евкліда. Використовуються змінні типу cardinal (змінні цього типу являють собою цілі 4-байтові числа без знаку).

Вигляд форми для розв'язання цієї задачі:



Програмний код:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    LabeledEdit1: TLabeledEdit;
    LabeledEdit2: TLabeledEdit;
    LabeledEdit3: TLabeledEdit;
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  a,b,c: cardinal;

implementation
{$R *.dfm}

function NOD(x,y: cardinal): cardinal;
begin
  while x<>y do          //допоки два числа не стануть рівними
   if x>y then x:=x-y    //від більшого числа
         else y:=y-x;    //віднімаємо менше число
  result:=x;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 a:=strtoint (labelededit1.Text);
 b:=strtoint (labelededit2.Text);
 c:=strtoint (labelededit3.Text);
 label1.Caption := 'Відповідь: '+ inttostr (nod(nod(a,b),c));
end;

end.
```

3. Поняття процедури. Створення та виклик процедур.

За своєю структурою процедури нагадують програму: вони, як і програма, складаються із заголовка та тіла процедури. Заголовок процедури містить зарезервоване слово procedure, ім'я процедури та список формальних параметрів, які у випадку їх використання розміщені у круглих дужках. Ім'я процедури – це ідентифікатор, унікальний у межах програми. Формальні параметри – це дані, які Ви передаєте у процедуру для опрацювання, та дані, які процедура повертає. Якщо процедура не отримує даних ззовні та нічого не повертає, формальні пар матери (у тому числі й круглі дужки) не записуються. Тіло процедури являє собою локальний блок, за структурою аналогічний до програми:

Procedure <ім'я процедури> (<список формальних параметрів>);
Const …;
Type …;
Var …;
Begin
 <оператори>
End;

Описи констант, типів даних та змінних дійсні тільки в межах даної процедури. В тілі процедури можна використовувати довільні глобальні константи та змінні, а також викликати будь-які підпрограми (процедури та функції).

Із підпрограмами, які реалізовані у вигляді процедур, ми уже зустрічалися багато разів – довільний обробник будь-якої події є підпрограмою. Коли ми створюємо обробник натискування кнопки, з'являється заготовка коду, де є заголовок із іменем, наприклад, Button1Click, далі йде блок begin … end; - блок реалізації, а вище нього ми вписуємо свої змінні, константи та інші необхідні елементи.

Виклик процедури для виконання здійснюється за її іменем, за яким йде поміщений в круглі дужки список фактичних параметрів, тобто, даних, які передаються у процедуру:

<ім'я процедури> (<список фактичних параметрів>);

Якщо процедура не потребує даних, то список фактичних параметрів (у тому числі й круглі дужки) не вказується.

Поняття процедури є надзвичайно важливим, так як саме воно лежить в основі однієї із найбільш популярних технологій розв'язування задач на мові Delphi: задача розбивається на декілька логічно завершених відокремлених частин, кожна з яких оформлюється у вигляді окремої процедури. Будь-яка процедура може містити в собі інші процедури, кількість яких обмежується лише об'ємом пам'яті Вашого комп'ютера.

Виникає питання: у якому місці потрібно розміщувати підпрограму? По-перше, її необхідно писати у такому місці, де програма зможе відшукати її. Підпрограми можна описувати у розділі описів, у якому описуються змінні, але тільки вище їх. Наприклад, якщо наша підпрограма використовується лише в обробнику натискування кнопки та більше ніде, то її можна розмістити у розділі опису цього обробника. Тоді із інших обробників підпрограму не буде видно. Друге правило: довільна підпрограма може бути описана вище того місця, звідки вона викликається. Викликати підпрограму, яка розміщена нижче, не можна, так як компілятор у цей момент часу знати про підпрограму не буде. І третє правило: щоб підпрограма була видна із довільного місця поточного модуля (поточної форми), її потрібно описати у розділі **implementation** вище всіх обробників подій.

4. Підпрограми з параметрами. Параметри процедур та функцій.

Параметри призначені для передачі вхідних даних до підпрограм та прийому результатів роботи цих підпрограм.

Вхідні дані передаються у підпрограму за допомогою вхідних параметрів, а результати роботи підпрограми повертаються через вихідні параметри. Параметри також можуть бути вхідними та вихідними одночасно.

Вхідні параметри оголошуються за допомогою ключового слова **const**; їх значення не можуть бути змінені всередині підпрограми.

Розглянемо у якості прикладу функцію відшукання найменшого значення із двох цілих чисел:

Function min (const a, b: integer): integer;

```
Begin
 If a<b then Result:=a else Result:=b;
End;
```

Для оголошення вихідних параметрів служить ключове слово **out**:

```
Procedure SizeForm (out width, height: integer);
Begin
 Width:=Form1.ClientWidth;
 Height:=Form1.ClientHeight;
End;
```

Присвоєння значень вихідних параметрів всередині підпрограми призводить до присвоєння значень змінних, які передані в ролі аргументів:

```
Procedure TForm1.Button1Click (Sender: TObject);
Var w, h: integer;
Begin
 SizeForm (w, h);
 Caption:=inttostr(w)+’ ’+inttostr(h);
End;
```

Після виклику процедури SizeForm змінні w та h будуть містити значення, присвоєні формальним параметрам width та height відповідно.

Якщо параметр є одночасно й вхідним і вихідним, то він описується із ключовим словом **var**:

```
Procedure exchange (var a, b: integer);
Var c: integer;
Begin
 c:=a;
 a:=b;
 b:=c;
End;
```

Зміна значень **var**-параметрів всередині підпрограми призводить до зміни значень змінних, переданих у ролі аргументів:

```
Var x, y: integer;
Begin
 X:=7;
 Y:=11;
 …
 Exchange (x, y);
 // тепер x=11, y=7
 …
End;
```

При виклику процедури на місце **out**- та **var**-параметрів можна підставляти тільки змінні, але не константи чи вирази.

Якщо при описанні параметру не вказане жодне із ключових слів – const, out, var, - то параметр вважається вхідним, його можна змінювати, при цьому всі зміни не впливають на фактичний аргумент, оскільки вони виконуються із копією

аргументу, створеною на час роботи підпрограми. При виклику підпрограми в ролі такого параметру можна використовувати константи та вирази.

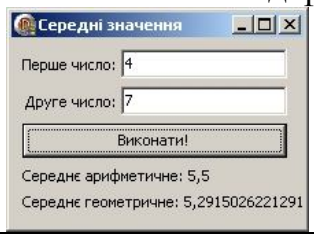
Різні способи передачі параметрів: із використанням const, out, var та без них можна суміщати в одній підпрограмі.

Способи передачі параметрів та їх значення

Ключове слово	Призначення	Спосіб передачі
<відсутнє>	Вхідний	Передається копія значення
Const	Вхідний	Передається копія значення або посилання на значення в залежності від типу даних
Out	Вихідний	Передається посилання на значення
Var	Вхідний і вихідний	Передається посилання на значення

Приклад. Створимо процедуру Average, яка приймає чотири параметри. Перші два параметри (x, y) є вхідними та призначені для передачі початкових даних. Інші два аргументи є вихідними та призначені для прийому в основній програмі результатів обрахунків середнього арифметичного (sa) та середнього геометричного (sg) значень x та y.

Зовнішній вигляд форми:




```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    LabeledEdit1: TLabeledEdit;
    LabeledEdit2: TLabeledEdit;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a, b, s1, s2: real;
implementation
{$R *.dfm}
procedure average (const x,y: real; out sa, sg: real);
```

```
begin
  sa:=(x+y)/2;
  sg:=sqrt(x*y);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 a:=strtofloat (labelededit1.text);
 b:=strtofloat (labelededit2.Text);
 average (a,b,s1,s2);
 label1.Caption:='Середнє арифметичне: '+floattostr(s1);
 label2.Caption:='Середнє геометричне: '+floattostr(s2);
end;
end.
```

Якщо передається значення, то підпрограма використовує копію аргументу. Такий спосіб називається передачею за значенням (by value).

Якщо передається посилання на значення, то підпрограма використовує безпосередньо аргумент, звертаючись до нього через передану адресу. Даний спосіб називається передачею за посиланням (by reference).

5. Прив’язка підпрограм до форми.

Якщо написані нами підпрограми займаються діями, не пов’язаними із візуальними компонентами форми (це, наприклад, якісь арифметичні обрахунки), то вони можуть просто розміщуватися у коді. Проте спроби звернутися із підпрограми до форми чи до її компоненту ні до чого не призведе – підпрограма «не знає» про нашу форму. Для того, щоб в процедурі або функції можна було працювати з компонентами форми, необхідно виконати певні дії – оформити їх у вигляді метода форми.

Розглянемо конкретний приклад. Припустимо, у нас на формі є три кнопки – Button1, Button2, Button3. Нам потрібно у різних частинах програми по черзі включати та виключати ці кнопки (використовуючи властивість Enabled). Щоб не копіювати один і той же код, краще написати підпрограму, яка буде робити все необхідне.

Запишемо нашу процедуру:

```
Procedure EnableButtons (Enabl: Boolean);
Begin
 Button1.Enabled:=Enabl;
 Button2.Enabled:=Enabl;
 Button3.Enabled:=Enabl;
End;
```

Проте даний код програми видасть помилку компіляції. Це пов’язане із тим, що розроблена процедура «не бачить» кнопки Button.

Для того, щоб «прив’язати» процедуру до форми, нам потрібно виконати наступні дії:

1. Необхідно додати заголовок методу в описі самої форми. Для цього потрібно скопіювати заголовок нашої процедури в секцію private або

public, в розділ type, де описана наша форма. Вибір секції – private або public, у кожному випадку потрібно визначати окремо. Все, що написано в private, доступно тільки в рамках даної форми. А все, що написане в public, може бути доступним й із інших форм та модулів.

2. В розділі implementation в заголовок процедури перед назвою потрібно додати ім'я класу форми. Ім'я класу форми в Delphi автоматично формується із літери «T» та імені самої форми. Наприклад, для форми Form1 ім'я класу буде TForm1. Між іменем класу форми та іменем підпрограми повинна стояти крапка. В результаті заголовок процедури буде мати наступний вигляд:
   Procedure TForm1.EnableButtons (Enabl: Boolean);

Якщо виклик створеної нами процедури здійснюється із самої Form1, то можна звертатися до процедури просто по імені. Наприклад, помістимо на форму прапорець типу TCheckBox та в обробнику його події OnClick напишемо:

```
Procedure TForm1.CheckBox1Click (Sender: TObject);
Begin
 EnableButtons (CheckBox1.Checked);
End;
```

6. Влаштовані процедури та функції: рядкові, перетворення типів даних, генератор псевдовипадкових чисел.

Всі процедури та функції в Delphi поділяються на дві групи: влаштовані та визначені програмістом. Влаштовані процедури та функції є частиною мови та можуть викликатися за іменем без попереднього опису.

Арифметичні функції:

Назва	Призначення	Тип результату
Abs (x)	Повертає абсолютне значення аргументу x	Integer, Real
Exp (x)	Повертає значення експоненти в степені x	Real
Ln (x)	Повертає натуральний логарифм аргументу x	Real
Pi	Повертає значення числа π	Extended
Sqr (x)	Повертає квадрат аргументу x	Integer, Extended
Sqrt (x)	Повертає корінь квадратний аргументу x	Extended

Приклади:

Вираз	Результат
Abs (-7)	7
Exp (1)	2.17828182845905
Ln (Exp (1))	1
Pi	3.14159265358979
Sqr (7)	49
Sqrt (64)	8

Тригонометричні функції:

Назва	Призначення	Тип результату
ArcTan (x)	Повертає кут, тангенс якого дорівнює x (в радіанах)	Extended
Cos (x)	Повертає косинус аргументу x (x задається в радіанах)	Extended

Sin (x)	Повертає синус аргументу x (x задається в радіанах)	Extended

Приклади:

Вираз	Результат
ArcTan (Sqrt (3))	1.04719755119660
Cos (Pi/3)	0.5
Sin (Pi/6)	0.5

Потрібно зазначити, що до складу середовища Delphi входить стандартний модуль Math, який містить множину додаткових підпрограм для тригонометричних, логарифмічних, статистичних та фінансових функцій.

Функції виділення цілої та дробової частини:

Назва	Призначення	Тип результату
Frac (x)	Повертає дробову частину аргументу x	Extended
Int (x)	Повертає цілу частину дійсного числа x	Extended
Round (x)	Округлює дійсне число x до цілого	Int64
Trunc (x)	Повертає цілу частину дійсного числа x. Результат належить цілому типу	Int64

Приклади:

Вираз	Результат
Frac (2.5)	0.5
Int (2.5)	2.0
Round (2.7)	3
Trunc (2.5)	2

Порядковими змінними називаються величини типів integer, char, boolean, а також похідні від цих типів. Для всіх них передбачені операції > та <, які визначають, яке значення є попереднім, а яке – наступним.

Підпрограми для роботи із порядковими величинами:

Назва	Призначення	Тип результату
Chr (x)	Повертає символ, порядковий номер якого дорівнює x	Char
Dec (x [,n])	Зменшує цілу змінну x на 1 (або на вказане число n)	Ordinal type, Pointer
Inc (x [,n])	Збільшує цілу змінну x на 1 (або на вказане число n)	Ordinal type, Pointer
Odd (x)	Повертає True, якщо аргумент x є непарним числом	Boolean
Ord (x)	Повертає порядковий номер аргументу x у своєму діапазоні значень	Ordinal type
Pred (x)	Повертає значення, яке йде попереду аргументу x у діапазоні вказаного типу	Ordinal type
Succ (x)	Повертає значення, яке йде після аргументу x у діапазоні вказаного типу	Ordinal type

Приклади:

Вираз	Результат
Chr (65)	‘A’

Odd (3)	True
Ord (‘A’)	65
Pred (‘B’)	‘A’
Succ (‘A’)	‘B’

Всі розглянуті підпрограми реалізовані у вигляді функцій, які повертають результат у залежності від значення аргументу. В стандарті мови є також процедури управління, призначені для зміни порядку виконання операторів управління.

Процедури передачі управління:

Назва	Призначення
Break	Перериває виконання циклу
Continue	Розпочинає нове повторення циклу
Exit	Перериває виконання поточного блоку
Halt	Зупиняє виконання програми та повертає управління операційній системі

Стандартні процедури та функції для роботи з рядками.

Оскільки опрацювання рядків виконується практично у кожній працюючій у візуальному середовищі програмі, стандартно підключений модуль System має набір процедур та функцій. Всі вони можуть бути застосовані як до коротких, так і до довгих рядків.

Процедури та функції для роботи з рядками:

Назва	Призначення
Concat (s1, s2, …, sn): string	Повертає рядок, який отриманий в результаті склеювання рядків s1, s2, …, sn. За принципом роботи функція Concat аналогічна операції склеювання рядків (+).
Copy (s: string, index, Count: integer): string	Копіює із рядка s підрядок довжиною count символів розпочинаючи із позиції index.
Delete (var s: string, index, count: integer)	Видаляє із рядка s count символів розпочинаючи із позиції index.
Insert (source: string, var s: string, index: integer)	Вставляє рядок Source в рядок s розпочинаючи із позиції index.
Length (s: string): integer	Визначає реальну довжину рядка S в символах.
SetLength (var s: string, NewLength: integer)	Встановлює для рядка S нову довжину NewLength.
Pos (Substr, s: string): integer	Відшукує перше входження підрядка Substr в рядок s. Повертає номер тієї позиції, де знаходиться перший символ підрядка Substr. Якщо в рядку s не відшукано підрядка Substr, то результат дорівнює 0.
AnsiLowerCase (const s: string): string	Перетворює заголовні літери рядка s в рядкові літери із урахування мови, встановленої в середовищі Windows.
AnsiUpperCase (const s: string): string	Перетворює рядкові літери рядка s в заголовні літери із урахування мови, встановленої в середовищі Windows.

Приклади:

Вираз	Значення S

S:=concat (‘Object ’,’Pascal’);	‘Object Pascal’
S:=copy(‘програмування’,3,3);	огр
S:=’програмування’; Delete (s, 2, 3);	‘прамування’
S:=’програмування в’; Insert (‘Delphi’,s,15);	‘програмування Delphi в’
Pos (‘ра’, ’програмування’);	5
Length (‘програмування в’);	18

Перетворення типів даних.

Використання візуальних компонентів передбачає необхідність перетворення великої кількості типів даних. Наведемо список процедур та функцій в середовищі Delphi, за допомогою яких виконуються ці перетворення.

Процедури та функції перетворення типів:

Назва	Призначення
IntToStr (value: integer): string	Перетворює ціле число value в рядок
StrToInt (const s: string): integer	Перетворює рядок в ціле число
FloatToStr (value: extended): string	Перетворює дійсне число value в рядок
StrToFloat (const s: string): extended	Перетворює рядок у дійсне число
Str (x [:width [:decimals]], var s: string)	Перетворює числове значення величини x в рядок s. Необов’язкові параметри width (ширина поля результуючого рядка) та decimals (кількість символів у дробовій частині запису дійсного числа) є цілочисельними виразами.
Val (s: string, var v; var code: integer)	Перетворює рядок s у величину цілого чи дійсного типу та поміщує результат у змінну v. Якщо під час операції помилки не виникло, значення змінної code дорівнює 0; якщо ж помилку відшукано (рядок містить неприпустимі символи), code містить номер позиції першого помилкового символу, а значення v не визначене.
StrToBool (const s: string): boolean	Перетворює рядок у булеве значення.
BoolToStr (b: boolean; UseBoolStrs: boolean=False): string	Перетворює булеве значення у рядок.
DateTimeToStr (const datetime: TDateTime): string	Перетворює значення дати та часу у рядок.
StrToDateTime (const s: string): TDateTime	Перетворює рядок у значення дати та часу.

Розглянемо приклад використання функцій, які призначені для роботи з рядками.

Завдання 1. На формі ввести вагу у вигляді «числове значення», пробіл, «кг», наприклад: «89 кг». Натискуванням на кнопку «Подвоїти вагу!» вивести значення подвоєної ваги, наприклад: «178 кг».

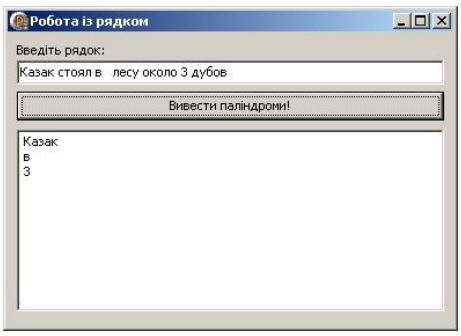
Програмний код для розв'язування цієї задачі:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    LabeledEdit1: TLabeledEdit;
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var s, st: string; ves: real;
begin
 s:=labelededit1.Text;
 st:=copy (s,1,pos(' ',s)-1);
 delete (s,1,pos(' ',s)-1);
 ves:=strtofloat(st);
 ves:=2*ves;
 s:=floattostr(ves)+s;
 label1.Caption := 'Результат: '+s;
end;
end.
```

Завдання 2. Ввести на форму довільний рядок символів (слова можна відділяти довільною кількістю пробілів). Вивести всі слова рядка, які є паліндромами.

Паліндромами можуть бути не лише числа (типу 313, 10101 тощо). Слова або фрази, які мають однаковий зміст при читанні їх як зліва направо, так і у зворотному напрямку, також паліндроми. Приклад – «корок».

Інтерфейс задачі:



Програмний код:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    LabeledEdit1: TLabeledEdit;
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

function udal_probel (st: string): string;
begin
  st:=st+' ';               //додати в кінець рядка пробіл
  while pos ('  ',st)>0 do    //якщо є подвійні пробіли
   delete (st, pos ('  ',st),1);  //вилучити один із них
  if st[1]=' ' then          //якщо перший символ - пробіл
   delete (st, 1, 1);        // вилучити його
  result:=st;
end;

function palindrom (st: string): boolean;
var i: integer;
begin
  result:=true;                    //слово є паліндромом
  for i:=1 to trunc(length(st)/2) do    //перевіряємо пари символів
    if ansiuppercase(st[i])<>ansiuppercase(st[length(st)+1-i]) then
     result:=false;                //якщо символи неоднакові
end;

procedure TForm1.Button1Click(Sender: TObject);
var s, slovo: string;  i: integer;
```

```
begin
 s:=labelededit1.Text;                //прочитати рядок
 s:=udal_probel(s);                  //вилучити зайві пробіли
 memo1.Lines.Clear;                   //очистити поле memo
 while s<>'' do
 begin
   slovo:=copy(s,1,pos(' ',s)-1);       //скопіювати слово
   delete (s,1,pos(' ',s));            //вилучити слово разом із пробілом
   if palindrom(slovo) then             //якщо слово - паліндром,
    memo1.Lines.Add(slovo);             //вивести його в memo
 end;
end;
end.
```

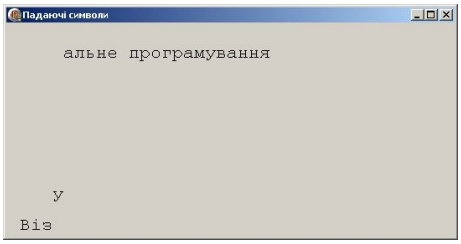
Запропонований алгоритм виділення слова у рядку є досить поширеним. Тому його можна використовувати щоразу, коли потрібно отримати доступ до окремих слів у вхідному рядку.

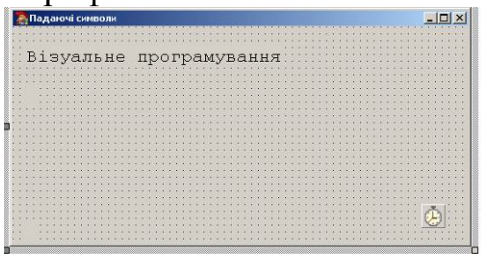
Розглянемо ще один підхід до роботи із текстовою інформацією.

Завдання 3. Створити програму, яка послідовно «осипає» донизу рядок, який знаходиться у верхній частині форми.

Опадання літер здійснюється за допомогою компоненту TTimer, робота якого завершується із завершенням рядка.

На малюнку наведено приклад падіння літер. Щоб рядок візуально не скорочувався, доцільно встановити шрифт Courier New.





Програмний код:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, ExtCtrls,
StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
  f: boolean=true;
  k: integer=1;
  t,l: integer;
implementation
{$R *.dfm}
procedure TForm1.Timer1Timer(Sender: TObject);
var c: char;                                          //падаючий символ
begin
 if f then
 begin
 c:=label1.caption[k];                                //скопіювати 1 символ та замінити
 label1.Caption:=copy(label1.caption,1,k-1)+' '+copy(label1.Caption,k+1,255);     //його пробілом
 label2.Caption:=c;
 if label1.caption='' then                            //якщо нічого не залишилось -
  timer1.Enabled := false;                            //завершити
 f:=false;
 k:=k+1;
 label2.Top:=t;                                       //відновити позиції
 label2.Left:=l;
 end
 else
 begin
   label2.Top:=label2.Top+5;                          //зсунути вниз
   if label2.Top>=label3.top then                     //якщо досягнуто низу
   begin
     label3.Caption:=label3.Caption+label2.Caption;   //перекинути символ
     l:=l+label2.Width;
     f:=true;
   end;
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  doublebuffered:=true;                               //ввімкнути подвійну буферизацію
  t:=label2.Top;                                      //запам'ятати позицію другого Label
  l:=label2.Left;
end;
end.
```

Генератор псевдовипадкових чисел.

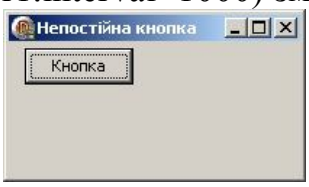
Всі сучасні ігри та додатки для створення різних подій використовують механізм випадкових чисел. Це може бути обумовлення вхідних даних, щоб не вводити їх у ручному режимі, або випадкова генерація лабіринту; вибір ігрового персонажу або хаотичного руху частинок чи мікроорганізмів. Прикладів таких подій багато, і всі їх об'єднує одне – те, що дані події носять випадковий характер. Для реалізації подібних подій використовується генератор випадкових чисел, хоча насправді тут мова йтиме не зовсім про випадкові числа – біль точніше, ці числа є псевдовипадковими.

Генератор псевдовипадкових чисел – це алгоритм, що генерує послідовність випадкових чисел, елементи якої майже не залежні один від іншого та підпорядковуються заданому розподілу (як правило, рівномірному).

Процедури та функції для роботи із псевдовипадковими числами:

Назва	Призначення
Random	Повертає випадкове дійсне число із проміжку від 0 до 1
Random (n)	Повертає випадкове дійсне число із проміжку від 0 до n
Randomize	Заново ініціалізує влаштований генератор випадкових чисел новим значенням, отриманим від таймера

Розглянемо роботу генератора псевдовипадкових чисел на прикладі наступної програми: на формі розміщена кнопка та таймер. Ця кнопка щосекунди (timer1.interval=1000) змінює своє положення (але не виходить за межі форми).



Програмний код:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate (Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
    button1.Left:=random(clientwidth-button1.Width);
    button1.Top:=random(clientheight-button1.Height);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  randomize;
end;
end.
```

Кнопка дійсно «скаче» з місця на місце. Для того, щоб кожного разу, коли запускається дана програма, кнопка переміщувалася на інші точки, потрібно в обробнику події OnCreate форми вписати процедуру Randomize.

7. Теоретичні питання для самоконтролю.

1. Чим обумовлена необхідність використання підпрограм?
2. У чому полягає різниця між застосуванням процедур та функцій?
3. Які типи даних може повертати функція?
4. Опишіть, як виглядає заголовок функції.
5. Яким чином можна записати результат всередині тіла функції?
6. Назвіть причини використання процедур.
7. В яких випадках при передачі параметрів використовується ключове слово var?
8. В яких випадках при передачі параметрів використовується ключове слово const?
9. В яких випадках при передачі параметрів використовується ключове слово out?
10. Що являє собою передача параметрів по посиланню та передача параметрів по значенню?
11. Перелічіть арифметичні функції, які використовуються при роботі із числовими змінними. Наведіть приклади.
12. Перелічіть тригонометричні функції. Наведіть приклади.
13. Перелічіть функції виділення цілої та дробової частин. Наведіть приклади.
14. Які функції Delphi призначені для обробки рядкових даних?
15. Перелічіть процедури, які використовуються при роботі із рядками.
16. Перелічіть функції перетворення типів даних.
17. Для чого використовуються псевдовипадкові числа?