Якщо в програмі не передбачено виходу з циклу, то кількість разів виконання циклу необмежена (зациклювання програми). Переривання роботи програми виконується натисканням клавіш <Ctrl> + <Break>.

■ Структурний підхід до побудови алгоритмів

Структурний підхід до побудови алгоритмів — це методика розробки алгоритмів, що забезпечує легкість розуміння алгоритму, простоту для перевірки правильності алгоритму, зручність модифікації.

Структурне програмування — складення програм з урахуванням структурного підходу.

■ Реалізація структурного програмування в середовищі Turbo Pascal

Для реалізації структурного програмування в середовищі Turbo Pascal використовуються такі методи:

- визначення в розділі описів усіх імен і типів даних, що використовуються в програмі;
- модульне програмування;
- використання підпрограм;

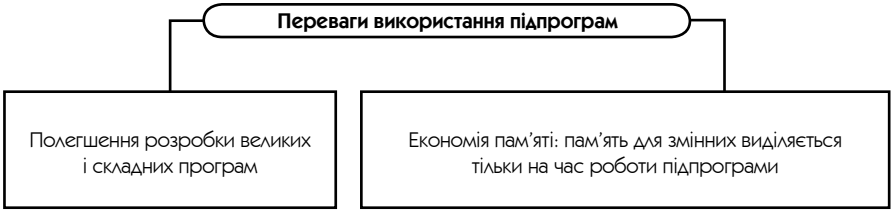
Модульне програмування — побудова програми за допомогою модулів — окремо виконаних і відкомпільованих програмних одиниць, що зберігаються на диску.

У Turbo Pascal є набір стандартних модулів: System, Overlay, Graph, DOS, Crt і Printer, що здійснюють підтримку програм (стандартні процедури і функції, робота з диском, принтером, графікою і т. д.). Для використання модулів у програмі їх необхідно описати в розділі опису модулів у довільному порядку.

Програма, що використовує команди модулів Graph і Crt для малювання різнокольорових кіл випадкового радіуса і положення

```
program Random_Fill_Circle;                                    {Заголовок}
uses Graph, Crt;                          {Використовувати команди модулів Graph і Crt}
var                                                          {Розділ опису змінних}
    GraphDriver, GraphMode: Integer;    {Змінні для типу і режиму відеоконтролера}
    R, X, Y, C: Integer;
                    {Змінні для зберігання значень радіуса, положення, кольору}
begin                                                  {Початок розділу операторів}
    GraphDriver := 9; GraphMode:= 2;
                                {Відеоконтролер VGA; режим 640*480, 16 кольорів}
    InitGraph(GraphDriver, GraphMode,'c:\Pas\Bin');
{Переключення в графічний режим, визначення реального шляху до модуля Graph}
    repeat                                                       {Початок циклу}
        SetFillStyle (Random(12), Random(succ(GetMaxColor)));
                                                  {Встановлення стилю заповнення}
        X := Random(GetMaxX);
            {Присвоїти X випадкове значення від 0 до максимально припустимого}
        Y := Random(GetMaxY);
            {Присвоїти Y випадкове значення від 0 до максимально припустимого}
        C := Random(succ(GetMaxColor)); SetColor (C);
                                        {Встановити поточний номер кольору}
        Circle (X,Y, Random(GetMaxY div 2));
                                        {Намалювати коло поточним кольором}
        FloodFill (X,Y,C);                                  {Заштрихувати коло
                                                  з центра поточним кольором}
        Delay (1000);                {Затримка часу (для швидкого комп'ютера)}
    until KeyPressed;          {Доки не натиснуто будь-яку клавішу, повернення
                                                          до початку циклу}
    CloseGraph;                                  {Переключення в текстовий режим}
end.                                    {Кінець розділу операторів і всієї програми}
```

Використання підпрограм — побудова програм за допомогою самостійних частин програми, що реалізують певний алгоритм. Підпрограми зручно використовувати, коли в програмі потрібно виконувати одні й ті ж дії, але з різними даними.



У Turbo Pascal є два різновиди підпрограм: процедури і функції. Головна відмінність між ними — це те, що функція повертає єдине значення і може бути використана у виразі, а процедура може бути викликана на виконання. Для використання процедур і функцій у програмі їх необхідно описати в розділі описів головної програми.

Оголошення процедур:

procedure <ім'я процедури> (<параметри>);

Оголошення функцій:

function <ім'я функції> (<параметри>): <тип результату>;

Структура опису процедур і функцій така ж, як і структура звичайної програми.

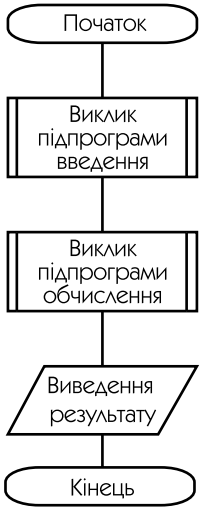
Дві відмінності між описами програм, процедур і функцій:

- процедури і функції мають заголовок **procedure** або **function** відповідно, а не **program**;
- процедури і функції закінчуються символом «;» (крапкою з комою), а не символом «.» (крапкою).
- в описі процедур і функцій не використовується **uses.**

Формальні параметри — це параметри, що описуються в заголовку процедури.

Фактичні параметри підставляються замість формальних параметрів при виклику підпрограми. Кількість і тип фактичних параметрів повинні чітко відповідати кількості й типам формальних параметрів. Зміст фактичних параметрів, що використовуються, залежить від того, в якому порядку вони перелічені під час виклику підпрограми.

Програма, яка обчислює значення функції $\frac{ax+bx^2+cx^3}{14x-x}$, використовуючи функції і процедури




```
program Big_Function;                                   {Заголовок}
var                                                     {Розділ опису змінних}
    A, B,C, X, Result: Real;
procedure Input_Data(var D, E, F, W:Real);              {Опис}
begin                                                   {процедури введення даних}
    WriteIn ( Введіть A, B,C:');
    ReadIn (D, E, F);
    WriteIn ('Введіть X:');
    ReadIn (W);
end;
function Process(var G, H, I, Z: Real) : Real;         {Опис}
begin                                   {функції, яка обчислює значення формули}
    Process := (G*Z+H* sqr(Z)+I*sqrt(Z)*Z)/(14*Z–Z);
end;
begin                                                   {Початок розділу операторів}
    Input_Data(A, B, C, X);             {Виклик процедури введення}
    Result := Process (A, B, C, X);     {Присвоювання змінній
                                        Result значення функції обчислення}
    WriteIn ('Значення функції дорівнює ',Result); Readln;
end.                                    {Кінець розділу операторів і всієї програми}
```

Після компіляції і запуску програми першим виконується виклик процедури Input_Data(A, B, C, X) і її формальні параметри D, E, F, W замінюються на фактичні параметри A, B, C, X. Ключове слово var в описі Input_Data говорить про те, що фактичні параметри повинні бути змінними і що значення змінних можуть бути змінені і повернуті програмі, яка викликала процедуру. При завершенні роботи Input_Data управління вертається в головну програму на оператор, наступний за викликом Input_Data.
Наступний оператор — виклик функції Process. Після виконання функції в тілі головної програми значення, що вертається Process, присвоюється Result і виводиться на монітор. Ще один приклад програми з процедурами наведений у кінці даного розділу.

Локальні дані — це дані, описані в розділі описів підпрограми. Локальні дані можуть використовуватися і мінятися тільки операторами даної підпрограми.

Глобальні дані — це дані, описані в розділі описів головної програми. Глобальні дані доступні з будь-якого місця програми.

Для уникнення помилок не рекомендується використовувати в програмах і підпрограмах однакові імена змінних, хоч це й припустимо.

■ Структуровані типи даних

Структуровані (складені) типи даних — типи даних, які складаються з простих (базових): цілих, речовинних, символьних, логічних. Складені типи даних зручно використовувати, коли обробляється багато змінних одного типу або об'єднуються декілька даних різного типу в одну групу, або обробляється деякий набір різних елементів одного типу.

Перевага використання такого типу даних у тому, що за допомогою них легше виконувати задачі упорядкування (сортування) даних, що дуже часто зустрічаються в повсякденному житті.
До структурованих типів даних належать масиви, рядки, записи, множини, файли, об'єкти.

■ Масиви (табличні величини)

Масив (таблиця) — це упорядкований набір фіксованої кількості однотипних елементів. Усі елементи масиву мають порядковий номер (індекс). Завдяки цій нумерації можна виділити будь-який елемент масиву і виконувати з ним операції, як із простим значенням базового типу. Елементи масиву ще називають індексованими змінними, на відзнаку від простих змінних.
Для використання масивів у програмі їх необхідно описати в розділі описів у довільному порядку.

Загальний вигляд опису масиву:

<ідентифікатор >: **array** [<діапазон>] **of** <базовий тип даних >

наприклад, опис

var A, B, C: array [1..50] of real

оголошує три масиви A, B, C дійсних чисел, кожний з яких містить по 50 елементів.

Опрацювання елементів масиву зручно виконувати за допомогою циклів.

Приклад. Програма сортування з неспадання масиву цілих чисел з п'яти елементів методом вибору.

Спочатку шляхом порівняння двох сусідніх елементів відшукується максимальний елемент і переставляється в кінець масиву, після чого цей метод застосовується для всіх елементів масиву, окрім останнього (він уже стоїть на своєму місці) і т. д.

```
program Regulate_Array;                                          {Заголовок}
const N=5;                                          {Визначення кількості елементів масиву}
var I, K, M, R: Integer;
    X: array [1..N] of Integer;                                  {Оголошення масиву}
begin                                                            {Початок розділу операторів}
    Writeln('Введіть елементи масиву');                          {Виведення повідомлення}
    for I := 1 to N do                                           {Цикл за кількістю елементів}
        begin                                                    {Початок складеного оператора}
            Write('X[',I,']=');Readln(X[I]);                     {Введення елементів масиву}
        end;                                                     {Кінець складеного оператора}
    for K := N downto 1 do                                       {Цикл від останнього елемента до 1-го}
        begin                                                    {Початок складеного оператора}
            M :=1;                                               {Пошук M — номера MAX}
            for I := 2 to K do if X[I]>X[M] then M := I;         {елемента масиву}
                R := X[K]; X[K] := X[M]; X[M] := R;              {Перестановка MAX}
            for I := 1 to N do Writeln('X [',I,']=',X[I],'');    {на останнє місце}
                Readln;          {і виведення елементів частково упорядкованого}
                                                    {масиву при натискуванні <Enter>}
                                                    {для унаочнення процесу упорядкування}
        end;                                                     {Кінець складеного оператора}
end.                                                {Кінець розділу операторів і всієї програми}
```

■ Рядкові величини

Рядок — це ланцюжок символів, обмежений апострофами (одинарними лапками).

Рядкові величини зручно використати для обробки текстів. Рядок дуже схожий на масив символьного типу, однак, на відзнаку від нього, рядок має змінну довжину (кількість символів), обмежену максимальним значенням рядкового типу String — 255 символів. До будь-якого елемента рядка можна звернутися за його індексом.

Для використання рядків у програмі їх необхідно описати в розділі описів.

Загальний вигляд опису рядкових даних:

var <ідентифікатор > **: String;**

Засоби обробки рядків

Функція	Опис
Concat (s_1, s_2, s_n): String	З'єднує послідовно рядки s_1, s_2, s_n
Copy (S: String; Index: Integer; Count: Integer)	Виділяє з рядка S підрядок довжиною в Count символів, починаючи з позиції Index
Length (S: String): Integer	Повертає довжину рядка
Pos (Substr: String; S: String): Byte	Повертає позицію, з якої підрядок Substr перший раз зустрічається в рядку S

Процедура	Опис
Delete (S: String; Index: Integer; Count: Integer)	Вилучає з рядка S Count символів, починаючи з позиції Index
Insert (Sourse: String; S: String; Index: Integer)	Вставляє фрагмент Sourse в рядок S, починаючи з позиції Index

Приклад. Задані рядок і два слова. Після кожного першого заданого слова, що зустрічається в даному рядку, вставити інше задане слово.

```
program Insert_Words;

uses Crt;
var
    S,                                                  {Допоміжний рядок}
    Word1,                          {Слово, після якого вставляється задане слово}
    Word2,                                              {Слово, яке вставляється}
    Str: String;            {Даний рядок}
    Len_Word1: Byte;                {Довжина слова, після якого вставляється слово}
begin
    ClrScr;                                             {Очищення екрана}
    Write ('Введіть речення ');
    Readln (Str);
    Write ('Введіть слово, після якого потрібно вставити слово ');
    Readln (Word1);
    Write ('Введіть слово, яке потрібно вставити в речення ');
    Readln (Word2);
    S := ' ';
```

```
        Len_Word1 := Length (Word1);
        while Pos (Word1, Str) > 0 do
            begin
                S := S+Copy(Str, 1, Pos (Word1,Str)+Len_Word1–1)+ ' ' + Word2;
                Delete (Str, 1, Pos (Word1, Str) + 1);
            end;
        Str := S + Str;
        Writeln ('Перетворений рядок: ', Str);
        Readln;
end.
```

Про наявність слова, після якого необхідно вставити інше слово, можна судити за значенням функції Pos (Word1, Str). Однак необхідно пам'ятати, що вона завжди вказує на перше входження підрядка в рядок. Тому якщо підрядок в рядку присутній, рядок поділяється на два: допоміжний рядок S буде містити частину вихідного рядка від першого його символу до останнього символу підрядка включно (до нього відразу ж додається друге задане слово S: = S + Copy (Str, 1, Pos (Word1, Str) + Len_Word1 – 1) + '' + Word2; вихідний рядок Str скорочується Delete (Str, 1, Pos (Word1, Str) + 1)) і містить «хвіст» від першого символу, який іде за першим зустрінутим словом, до останнього символу рядка. У подальшому функцією Pos (Word1, Str) шукається наступне входження першого слова в уже скороченому рядку. А після виходу з циклу «хвіст», що залишився в змінній Str і не містить першого слова, об'єднується з допоміжною змінною S в підсумковий рядок Str.

Перший байт у рядку має індекс 0 і містить поточну довжину рядка. Перший значущий символ рядка займає другий байт і має індекс 1.

Приклад з використанням основних типів даних і керуючих структур

```
  program Birthday;
  {Ця програма вводить дату у форматі ДД ММ РРРР і виводить на екран }
  {відповідний цій даті день тижня.}
var
      IsCorrectDate: Boolean;                    {Ознака правильної дати}
      d,m,y: integer;                  {Дата, що вводиться,— день, місяць і рік}

  Procedure InputDate(var d,m,y : integer; var correctly : Boolean);
                        {Процедура вводить у змінні d, m і y чергову дату і перевіряє її}
  {————————————————————}
```

```
begin
    Write('Введіть дату в форматі ДД ММ РРРР: ');
    ReadLn(d,m,y);        {Якщо дата правильна,}
                                        {встановлює correctly = true,
                                            інакше correctly = false}
    correctly := (d>=1) and (d<=31) and (m>=1) and (m<=12) and (y>=1582) and
(y<=4903)
end;
{————————————————}
Procedure WriteDay(d,m,y : integer);
const
    Days_of_week: array [0..6] of string [11] = ('неділя','понеділок',
    'вівторок', 'середа','четвер','п'ятниця','субота');
var
    c, w : integer;
begin
    if m < 3 then
        begin                                   {Місяць січень чи лютий}
            m := m + 10;
            y := y – 1
        end
    else
    m := m – 2;                                 {Інші місяці}
    c := y div 100;        {Обчислюємо сторіччя}
    y := y mod 100;                     {Знаходимо рік у сторіччі}
    w := abs(trunc(2.6*m–0.2)+d+y div 4+y+c div 4–2*c) mod 7;
    WriteLn(Days_of_week[w])
end;
{————————————————}
begin
    repeat
        InputDate(d,m,y,IsCorrectDate);
        if IsCorrectDate then
            WriteDay(d,m,y)
    until not IsCorrectDate
end.
```