Á³áë³îòåêà «Øê³ëüíîãî ñâ³òó»
Çàñíîâàíà ó 2003 ð.

Андрій Ставровський
Ірина Скляр

ПРОГРАМУЄМО ПРАВИЛЬНО

Посібник

У двох частинах
Частина 2

²íôîðìàòèêà. Á³áë³îòåêà

Êè¿â
«Øê³ëüíèé ñâ³ò»
2007

ББК 32.973-018я721
С76

Ставровський, Андрій.

С76 Програмуємо правильно: Посібник. : У 2 ч. / А. Ставровський, І. Скляр. — К. : Шк. світ, 2007. — Ч. 2. — 128 с. — (Б-ка «Шк. світу»).

ISBN 978-966-451-000-1.
ISBN 978-966-451-075-9. (Ч. 1)
ISBN 978-966-451-091-9. (Ч. 2)

Посібник містить другу частину початкового курсу програмування в рамках шкільної дисципліни «Інформатика». У першій чатсині представлено основи алгоритмізації та програмування мовою Паскаль, у другій частині — роботу з основними структурами даних (масиви, файли, записи, множини). Необхідний теоретичний матеріал супроводжується практичними прикладами, докладним розв'язанням типових задач, контрольними запитаннями та задачами для самостійної роботи.

Посібник призначено для учнів, які вивчають інформатику за програмою фізико-математичних шкіл, а також для вчителів інформатики, студентів педагогічних ВНЗ і всіх, хто бажає навчитися програмувати.

ББК 32.973-018я721

ISBN 978-966-451-000-1(б-ка «Шк. світу»)
ISBN 978-966-451-075-9 (Ч. 1)
ISBN 978-966-451-091-9 (Ч. 2)

ЗМІСТ

ВІД АВТОРІВ

Посібник містить теми шкільного курсу «Масиви» та «Рядки», а також теми, які в загальноосвітній школі не вивчаються, але є необхідними для підготовки до олімпіад з інформатики всіх рівнів: від районної до Всеукраїнської. Тут представлено обробку текстових файлів, різноманітні методи сортування, зокрема, швидкого, роботу з багатоцифровими числами (так звана «довга арифметика»), складність алгоритму та її оцінки, нестандартні задачі з використанням множин. Усі програми супроводжуються детальним розбором.

У цьому посібнику, як і в попередньому, не всі задачі є авторськими — деякі взято з Інтернету або інших посібників. Наведені тут задачі є досить цікавими, мають нескладне оригінальне розв'язання й дозволяють учням поступово готуватися до участі у різних конкурсах та олімпіадах Юних програмістів.

Посібник також містить довідкові матеріали, корисні для набуття практичних навичок роботи з системою програмування Turbo Pascal.

ðîçä³ë 1

Ì ÀÑÈÂÈ ÒÀ ÔÀÉËÈ

1. ПОНЯТТЯ МАСИВУ

У багатьох задачах у пам'яті програми треба зберігати великі набори однотипних даних. Для цього використовують масиви

Масив — це змінна, утворена послідовністю значень, які називаються ***елементами*** (***компонентами***), є однотипними й ідентифікуються номерами (***індексами***). Множина індексів (***діапазон***) фіксується в оголошенні масиву та при виконанні програми не змінюється. Кількість елементів індексної множини називають ***довжиною масиву***.

- Елементи масиву є ***рівнодоступними***, тобто можливість обробки елемента не залежить від його місця в послідовності.
- Тип масиву задають виразом, у якому вказують множину індексів ***I*** та тип елементів ***T***: **`array [I] of T`**. Слова **`array`** (масив) і **`of`** — ключові.
- Типом елементів може бути довільний тип, а типом індексів — перелічуваний тип, указаний іменем або діапазоном. Типи **`integer`**, **`word`** та **`longint`** як типи індексів у Turbo Pascal ***заборонено***, оскільки ця мова програмування накладає обмеження на загальний розмір масиву 64K.
- Якщо тип індексів оголошено як діапазон, то його задають ***двома константами***, можливо, іменованими. Записувати імена змінних у діапазоні заборонено.

Приклад.

1. Багаточлен з дійсними коефіцієнтами, який має степінь не більше 99, можна подати масивом типу **array [0..99] of real**. Індекси його дійсних елементів — від **0** до **99**.

2. Щоб підрахувати, скільки разів у деякому тексті зустрівся кожен символ, можна заповнити масив типу **array[char] of word**. Індексами його 256 цілих змінних є символи, значеннями — кількості цих символів.

3. Оголошення типу масиву на кшталт **array[1..n] of real** з іменем змінної **n** — ***типова помилка початківців***.

Вираз, що задає тип масиву, можна записати в оголошенні змінної, але, як правило, краще окремо оголосити ім'я типу масивів, а потім цим іменем задавати тип змінних.

Приклад. Можливі обидва такі оголошення.

1) **type aReal = array [0..99] of real;**
   var X : aReal;
2) **var X : array [0..99] of real;**

Перша буква **a** імені **aReal** вказує на те, що ім'я позначає саме тип масивів (***a***rray). Перший варіант оголошень краще, оскільки для типу оголошено ім'я, яке можна використовувати надалі замість виразу **array[0..99] of real**. В обох випадках масив **X** містить 100 дійсних змінних **X[0]**, **X[1]**, ..., **X[99]**. Елементи масиву позначають виразами вигляду **X[*індексний_вираз*]**, де ***індексний_вираз*** має значення від **0** до **99**. Їм, як і всім змінним, можна присвоювати значення або використовувати їх значення у виразах, тобто можливі оператори такого вигляду.

```
A[0]:=1; A[1]:=A[0]+2;
for k:=2 to 99 do readln(A[k]);
writeln(A[0]+A[1]+A[2]+A[3])
```

Обробку масивів описують, головним чином, через обробку їх елементів, але ***однотипні масиви можна присвоювати***.

Наприклад, при дії оголошень

```
var a,b:array[char] of integer; ch:char;
```

замість оператора циклу

```
for ch:=chr(0) to chr(255) do
  b[ch]:=a[ch]
```

можна написати таке лаконічне присвоювання: **b := a**

2. ПРИКЛАДИ УТВОРЕННЯ ТА ОБРОБКИ МАСИВІВ

У конкретних задачах значення елементів масиву утворюються в три основні способи:

- шляхом уведення з зовнішнього джерела даних;
- за допомогою генератора псевдовипадкових чисел;
- як результат обчислень.

Розглянемо приклад уведення значень за допомогою клавіатури.

Приклад. Треба отримати від клавіатури послідовність цілих чисел довжиною не більше 10, визначити найбільше з них і позиції у послідовності, на яких було це число.

На початку програми оголосимо тип масиву. Окремо оголосимо ім'я константи, яка задає кількість елементів масиву. За умовою, потрібен масив, у якому використовуються не більше 10 елементів.

```
const maxN = 10;
aInt = array[1..maxN] of integer;
```

Розглянемо найпростіший варіант уведення значень елементів числового масиву: спочатку задано кількість значень, а потім самі значення в цій кількості. Реалізуємо введення процедурою **readAInt**. Масив і кількість елементів, які отримують значення, мають використовуватися після виклику процедури, тому оголосимо їх як параметри-змінні.

```
procedure readAInt(var a:aInt; var n:byte);
  var k:byte; {поточний індекс}
begin
  repeat
   write('Кількість елементів(1..', maxN, ')>');
   readln(n);
  until (1 <= n) and (n <= maxN);
  for k:=1 to n do begin
    write(k, '-е ціле значення>');
    readln(a[k]);
  end;
end;
```

Після виклику цієї процедури значення одержують тільки перші ***n*** елементів масиву. Значення решти непередбачувані і, по суті, є «сміттям». Саме тому необхідний параметр ***n***, що представляє справжню кількість заданих значень.

Після того, як уведено числа, проходимо масивом і визначаємо найбільше значення. Потім ще раз проходимо масивом і виводимо номери елементів із цим найбільшим значенням. Зауважимо: якби було потрібно лише максимальне число, а не його позиції, можна було визначати його при введенні чисел, не використовуючи масив.

Отже, програма матиме такий вигляд:

```
program maxIntNumber;
  const maxN=10;
  aInt=array[1..maxN] of integer;
  var x:aInt; {масив цілих чисел}
  max:integer;{максимальне число}
  n,k:byte; {кількість та поточна позиція}
  procedure readAInt(var a:aInt; var n:byte);
    ... {див. вище в тексті}
  end;
begin
  readAInt(x,n);{x i n визначаються у виклику}
  max:=x[1];
  for k:=2 to n do
    if x[k] > max then max := x[k];
  writeln('Максимальне число: ', max);
  write('Його позиції: ');
  for k:=1 to n do
    if x[k]=max then write(k,' ');
  readln;
end.
```

Якщо за її виконання задати кількість чисел **8**, а потім увести числа **21**, **13**, **21**, **20**, **–1**, **0**, **21**, **5**, то відповідь має бути, що максимальним є число **21**, а його позиції **1**, **3**, **7**.

▣

У багатьох задачах (від моделювання природних або соціальних процесів до розкладання грального карт) використовують послідовності чисел, що належать певній множині, але більше ніяк

не пов'язані одне з одним. Такі числа називаються ***випадковими***. Проте часто замість випадкових чисел використовують числові послідовності, в яких наступні елементи певним чином обчислюються за попередніми, але виглядають як випадкові. Послідовності таких чисел називають ***псевдовипадковими***. Їх отримують за допомогою спеціальних підпрограм — ***генераторів псевдовипадкових чисел***. Багаторазові виклики такої підпрограми породжують послідовність псевдовипадкових чисел.

У системі Turbo Pascal функція-генератор псевдовипадкових чисел має ім'я **`random`**. Її виклик без аргументів повертає псевдовипадкове дійсне число з інтервалу [0; 1). Якщо у виклику вказано аргумент типу **`word`** зі значенням ***R***, повертається ціле число від 0 до ***R***–1.

Корисна також процедура **`randomize`** без параметрів, яка ініціалізує генератор випадковим числом, отриманим за допомогою системного годинника комп'ютера. Якщо процедуру **`randomize`** не викликати, то кожне виконання послідовності викликів функції **`random`** буде давати одну й ту саму послідовність чисел.

Розглянемо заповнення масиву за допомогою псевдовипадкових чисел та на основі обчислень.

Приклад. У школі вчаться учні, зріст яких від 110 до 220 см (ціле число). Треба отримати дані про їх зріст та вивести кількість учнів кожного можливого зросту.

Утворимо дані про зріст за допомогою функції **`random`**. Діапазон значень зросту — від 110 до 220, тому потрібне значення утворимо як 110 плюс випадкове число в межах від 0 до 110. Отже, багаторазові виклики функції матимуть вигляд **`random(111)`**.

Перший спосіб. В умові не сказано про можливу кількість учнів. Припустимо, що їх не більше двох тисяч і запам'ятаємо числа в масиві, індекси якого від 1 до 2000 відповідають учням. Потім за цим масивом визначимо, скільки учнів мають зріст 110, скільки — 111 тощо.

Отже, оголосимо тип масиву з даними про зріст учнів.

```
const maxN=2000;
aSize=array[1..maxN] of word;
```

Дані про зріст утворимо за допомогою такої процедури, на початку виконання якої від клавіатури отримується кількість учнів.

```
procedure genASize(var a:aSize; var n:word);
  var k:word; {поточний індекс}
begin
  repeat
    write('Кількість учнів(1..', maxN, ')>');
    readln(n);
  until (1<=n) and (n<=maxN);
  Randomize;
  for k:=1 to n do
    a[k]:=110+Random(111);
end;
```

Тоді програма з цією процедурою матиме такий вигляд:

```
program maxIntNumber;
  const maxN=2000;
  aSize=array[1..maxN] of word;
  var S:aSize; {масив даних про зріст учнів}
    n,cnt,k:word;
 {кількість та лічильник учнів, номер учня}
  t:byte;       {поточний зріст}
  procedure genASize(var a:aSize;
                     var n:word);
    ... {див. вище в тексті}
  end;
begin
  genASize(S,n);{S i n визначаються у виклику}
  for t:=110 to 220 do begin
    cnt:=0;
    for k:=1 to n do
      if S[k]=t then inc(cnt);
    if cnt>0 then {лише ненульові кількості}
      writeln('Зріст ',t,': ',cnt,' учнів');
  end;
  readln;
end.
```

Другий спосіб. Насправді масив, індекси якого відповідають учням, ***не потрібен***. Замість його заповнення використаємо масив, індекси якого — можливі значення зросту, а значення — кількості повторень цих значень серед даних. Оголосимо тип такого масиву лічильників.

```
type aCnt=array[110..220] of word;
```

У процедурі утворення даних збільшуються лічильники, індексовані значеннями зросту. Кількість учнів тепер обмежено тільки типом **word**.

```
procedure genACnt(var a:aCnt);
{genACnt - генерувати масив лічильників
  (скорочення від GENerate Array of CouNTers}
  var n:word; {кількість учнів}
      t:byte;    {значення зросту}
begin
  repeat
    write('Кількість учнів>');
    readln(n);
  until (1<=n);
  {ініціалізація лічильників нулями}
  for t:=110 to 220 do a[t]:=0;
  randomize;
  for k:=1 to n do
    inc(a[110+random(111)]);
end;
```

Виведемо дані за допомогою процедури **writeACnt**. Масив лічильників задамо як її параметр. Під час її виконання параметр має залишатися без змін, тому оголосимо його як параметр-константу.

Програма з цими двома процедурами має такий вигляд:

```
program maxIntNumber;
  type aCnt=array[110..220] of word;
  var C:aCnt;  {дані про зріст учнів}
  procedure genACnt(var a:aCnt);
    ... {див. вище в тексті}
  end;
  procedure writeACnt(const a:aCnt);
    var t:byte;  {поточний зріст}
```

```
  begin
    for t:=110 to 220 do
      if a[t]>0 then
      writeln('Зріст ',t,': ',a[t],' учнів');
  end;
Begin
  genACnt(C);
  writeACnt(C);
  readln;
End.
```

Приклад. Повернемося до прикладу статті 8 розділу 6 першої частини, де потрібно було прочитати натуральне число типу **longint** та вивести його p-ковий запис ($p \le 36$).

За добре відомим алгоритмом значення цифр числа утворюються, починаючи від молодшої, а вивести їх треба, починаючи зі старшої. Запам'ятаємо ці значення в масиві байтів. Найбільше число типу **longint** має 31 двійкову цифру, тому потрібен масив з 31 елемента. Заповнення масиву байтів реалізуємо процедурою **makeDigs**, друкування цифр — процедурою **writeDigs**. Зауважимо: якщо вхідне число 0, то його запис матиме 0 цифр, і це буде особливим випадком при друкуванні цифр.

```
program numberOutput;
  var n:longint; p:byte; {число та основа}
  const maxND=31; {максимальна кількість цифр}
  type aDigs:array[0..maxND] of byte;
  var x:aDigs;     {масив значень цифр}
      m:byte;      {кількість цифр}
function digit(v:byte):char;
begin  {повернення цифри за її значенням v}
  case v of
   0..9: digit:=chr(v+ord('0'));
   10..35: digit:=chr(v-10+ord('A'));
  end;
end;
procedure writeDigs(const x:aDigs; m:byte);
{m - кількість цифр у масиві x}
  var k:byte; {поточний індекс цифри}
```

```
begin
  if m=0 {особливий випадок: вхідне число 0}
    then write(0)
  else
   for k:=m-1 downto 0 do write(digit(x[k]));
  writeln;
end;
procedure makeDigs(n:longint; p:byte;
                   var x:aDigs; var m:byte);
{m - кількість цифр, які збережено в масиві x}
begin
  m:=0;
  while n>0 do begin
    x[m] := n mod p;
    inc(m);
    n := n div p;
  end;
end;
Begin
  readln(n,p);
  makeDigs(n,p,x,m);
  writeDigs(x,m);
End.
```
▣

Приклад. Повернемося до кролів Фібоначчі (статтю 2 розділу 5 першої частини). Припустимо, що пара кролів живе повних t місяців і ще декілька днів, а досягає зрілості й починає народжувати через b місяців після появи на світ ($b < t \le 10$). Першого січня є одна пара новонароджених кролів. Визначити кількість пар через задану кількість місяців m. Наприклад, за $t = 4$, $b = 2$, $m = 5$ відповідь буде 6 (якби кролі були «безсмертними», було б 8 пар, а так одна пара померла й одна не народилася).

Зрозуміло, що кількість пар кролів змінюється залежно від того, скільки є пар того чи іншого віку. Подамо пари кролів у масиві `R`, індекси елементів якого відповідають віку кролів (у місяцях від 0 до t), а значення виражають кількість пар кролів цього віку.

```
const maxAge=10; {максимальний вік кролів}
var R:array[0..maxAge] of longint;
```

Розглянемо, як за поточним значенням масиву визначається наступне. За місяць пари віку 0 (новонароджені) стануть парами віку 1, віку 1 — парами віку 2, ..., віку $t-1$ — парами віку t, а пари віку t вимруть. Отже, за кожного k від 0 до $t-1$ значення елемента `R[k]` має стати значенням `R[k+1]`. Кількість новонароджених пар буде сумою кількостей пар, вік яких від b до t.

Описаний перерахунок кількості пар кожного віку треба провести для кожного місяця, врахувавши у початковому значенні масиву, що «через 0 місяців» є одна новонароджена пара.

На основі наведених міркувань побудуємо таку програмку:

```
program rabbits;
  const maxAge=10; {межа віку кролів}
  var R:array[0..maxAge] of longint;
  var b,t:byte; k:byte;
      m,month:word;
      s:longint; {сумарна кількість пар}
begin
  writeln('Зрілість та тривалість
                              (до 10)>',b,t);
  writeln('Кількість місяців>',m);
  for k:=1 to t do R[k]:=0;
   R[0]:=1; {спочатку є одна новонароджена
пара}
  for month:=1 to m do begin
    for k:=t downto 1 do R[k]:=R[k-1];
                  {саме в такому порядку!!!}
    {підрахунок пар, які народяться}
    R[0]:=0;
    for k:=b to t do inc(R[0],R[k]);
  end;
  s:=0;
  for k:=0 to t do inc(s,R[k]);
  writeln('Загальна кількість пар: ', s);
end.
```

3. ДВОВИМІРНІ МАСИВИ. ПРИКЛАДИ ВИКОРИСТАННЯ

Прямокутну таблицю з $m \times n$ однотипних елементів можна подати як складену з ***m*** рядків по ***n*** елементів у кожному. Її можна розглядати як масив, елементами якого є масиви, або як ***двовимірний масив***. Якщо елементи двовимірної таблиці самі є масивами або таблиці утворюють послідовність, то виникає ***тривимірний*** масив тощо.

Оголошення двовимірних масивів, або ***матриць***, і зображення їх елементів у мові Паскаль опишемо за допомогою простого прикладу.

Приклад. Позицію в грі «хрестики-нулики на полі 3 на 3» подано квадратною таблицею із символів `'x'`, `'0'` та `' '` (пропуск). Пронумеруємо клітини поля, як у шахах, — буквами `'a'`, `'b'`, `'c'` по горизонталі та числами 1, 2, 3 по вертикалі. Тоді рядки таблиці — це масиви такого типу.

```
type Row=array['a'..'c'] of char;
```

Таблиця — це масив, складений трьома рядками.

```
type Table=array[1..3] of Row;
```

Масиви типу **Table** мають два виміри: ***номер рядка*** та ***номер стовпчика*** в ньому. Вимір **1..3**, що нумерує рядки, називається ***зовнішнім***, вимір **'a'..'c'**, що нумерує символи в рядках, — ***внутрішнім***.

Тип **Table** можна задати, не оголошуючи імені типу **Row**.

```
type Table=array [1..3] of
           array['a'..'c'] of char;
```

У мові Паскаль є інша форма запису вимірів: через кому в спільних дужках. Наприклад, тип **Table** можна оголосити так.

```
type Table=array[1..3,'a'..'c'] of char;
```

За будь-якої форми оголошення типу **Table** елементи масиву цього типу, наприклад, **A**, можна позначати як **A[i,j]** або **A[i][j]**, де $1 \le i \le 3$, $'a' \le j \le 'c'$. Вираз вигляду **A[i]**, де $1 \le i \le 3$, позначає рядок таблиці, тобто змінну типу **array['a'..'c'] of char**.

- Елементи багатовимірних масивів розташовуються в пам'яті послідовно, найшвидше в них змінюється внутрішній

індекс, найповільніше — зовнішній. Зокрема, двовимірні масиви розташовуються ***рядками***.

Наприклад, послідовні елементи масиву типу **array[1..2,'a'..'b'] of byte** мають набори індексів **[1,'a']**, **[1,'b']**, **[2,'a']**, **[2,'b']**, а послідовні символи в масиві типу **Table** — **[1,'a']**, **[1,'b']**, **[1,'c']**, **[2,'a']**, ..., **[3,'c']**.

Приклад. Треба прочитати два натуральних числа ***m*** і ***n*** не більше 20, утворити числову матрицю розмірами $m \times n$ (***m*** рядків, ***n*** стовпчиків), вивести її на екран, а потім відобразити її симетрично відносно вертикальної осі та вивести на екран.

Спочатку оголосимо тип числової матриці, вважаючи, що значення її елементів мають тип **byte**.

```
const maxSz=20; {максимальний розмір}
type Matrix=array[1..maxSz,1..maxSz] of byte;
```

Уведемо розміри матриці та утворимо значення її елементів за допомогою генератора випадкових чисел у такій процедурі.

```
procedure gMatr(var a:Matrix; var m,n:byte);
  var i,k:byte; {поточні індекси}
begin
 write('Рядків та стовпчиків  (1..',maxSz,')>');
  readln(m,n);
  Randomize;
  for i:=1 to m do
    for k:=1 to n do
      a[i,k]:=Random(256);
end;
```

Процедура виведення значень у матриці на екран також дуже проста. Значення виводяться рядками; числа типу **byte** мають не більше трьох десяткових цифр, тому ширина поля виведення 4 забезпечить пропуски між числами на екрані.

```
procedure wrMatr(const a:Matrix; m,n:byte);
  var i,k:byte; {поточні індекси}
begin
  for i:=1 to m do begin
    for k:=1 to n do
      write(a[i,k]:4); {виведення в рядок}
```

```
      writeln;  {перехід на новий рядок}
    end;
  end;
```

Розглянемо симетричне відображення матриці відносно вертикальної осі. Перший стовпчик має помінятися місцями з останнім, другий — з передостаннім тощо. Якщо кількість стовпчиків непарна, то «серединний» стовпчик залишається без змін, а якщо парна, то такого стовпчика немає. Отже, кожен стовпчик з номером **k** від **1** до **n div 2** треба поміняти місцями зі стовпчиком, номер якого — **n-k+1**. Цей самий процес можна розглядати інакше: у кожному рядку (від **1** до **m**) обмінюються значення елементів з номерами **k** від **1** до **n div 2** та відповідних їм елементів з номерами **n-k+1**.

Реалізуємо цей обмін у такій процедурі:

```
procedure vertSymm(var a:Matrix; m,n:byte);
  var i,k,t:byte;
begin
  for i:=1 to m do
    for k:=1 to n div 2 do begin
      t:=A[i,k]; A[i,k]:=A[i,n-k+1];
      A[i,n-k+1]:=t
    end
end;
```

Нарешті, напишемо програму розв'язання задачі, указавши в ній наведені процедури скорочено:

```
program matrixes;
  const maxSz=20; {максимальний розмір}
  type Matrix=array[1..maxSz,1..maxSz] of byte;
  procedure gMatr(var a:Matrix; var m,n:byte);
    ... {див. вище в тексті}
  end;
  procedure wrMatr(const a:Matrix; m,n:byte);
    ... {див. вище в тексті}
  end;
  procedure vertSymm(var a:Matrix; m,n:byte);
    ... {див. вище в тексті}
  end;
```

```
  var x:Matrix; m,n:byte;
Begin
  gMatr(x,m,n);
  writeln('Утворено таку матрицю');
  wrMatr(x,m,n);
  writeln('Відображення зліва направо');
  vertSymm(x,m,n);
  wrMatr(x,m,n);
  readln;
End.
```

Приклад. В умові попередньої задачі замість відображення матриці розглянемо її поворот на 180° (байдуже, за годинниковою стрілкою чи проти неї).

Наведемо лише процедуру повороту матриці A на 180°. Неважко переконатися, що при цьому за будь-яких i від 1 до m та k від 1 до n обмінюються значення $A[i, k]$ та $A[m - i + 1, n - k + 1]$. Значення *всіх* елементів верхньої половини матриці опиняються в нижній половині і навпаки. Отже, треба перебрати рядки матриці з номерами i від 1 до m div 2 та поміняти місцями значення $A[i, k]$ та $A[m - i + 1, n - k + 1]$ *при всіх* k від 1 до n. Проте, якщо m непарне, залишиться «серединний» рядок, в якого $i = m - i + 1$. У ньому треба обміняти місцями перше значення з останнім, друге — з передостаннім тощо, тобто при обміні $A[i, k]$ та $A[m - i + 1, n - k + 1]$ індекс k має прийняти значення від 1 до n div 2.

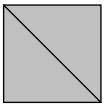
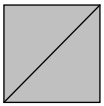
```
procedure turn180(var a:Matrix; m,n:byte);
  var i,k,t:byte;
begin
  for i:=1 to m div 2 do
    for k:=1 to n do begin
      t:=A[i,k]; A[i,k]:=A[m-i+1,n-k+1];
      A[m-i+1,n-k+1]:=t
    end;
  if odd(m) then begin
    i := m div 2 +1;{номер «серединного» рядка}
    for k:=1 to n div 2 do begin
      t:=A[i,k]; A[i,k]:=A[i,n-k+1];
```

```
          A[i,n-k+1]:=t
        end;
      end;
    end;
```

Перевірити цю програму треба як за парного, так і за непарного першого виміру матриці, оскільки дії в цих ситуаціях мають бути різними.

▣

Масиви, в яких кількості рядків і стовпчиків однакові, називаються ***квадратними*** й мають дві діагоналі (головну та бічну).

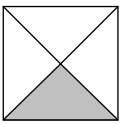
Головна діагональ  Бічна діагональ 

Елементи на головній діагоналі мають індекси (1, 1), (2, 2), (3, 3), ..., (***i***, ***i***), ..., (***n***, ***n***), тобто номери рядка й стовпчика рівні. Індекси елементів на бічній діагоналі (1, ***n***), (2, ***n*** – 1), (3, ***n*** – 2), ..., (***i***, ***n*** + 1 – ***i***), ..., (***n***, 1) зв'язано формулою ***i*** + ***j*** = ***n*** + 1, де ***n*** — кількість рядків (стовпчиків), ***i*** та ***j*** — номер рядка та стовпчика масиву відповідно.

Діагоналі визначають у масиві вісім «трикутників» (чотири великих, що є «половинами» масивів над кожною діагоналлю та під нею, та чотири малих між діагоналями). Будемо вважати, що елементи діагоналей належать відповідним трикутникам. Наприклад, при ***n*** = 4 «верхній лівий трикутник» при бічній діагоналі містить елементи з індексами (1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (4, 1), а «нижній трикутник» між діагоналями — елементи з індексами (3, 2), (3, 3), (4, 1), (4, 2), (4, 3), (4, 4). При ***n*** = 3 аналогічні «трикутники» містять елементи з індексами (1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (3, 1), та, відповідно, (2, 2), (3, 1), (3, 2), (3, 3).

Традиційними є задачі пошуку елементів з деякими властивостями в трикутниках, визначених діагоналями.

Приклад. Знайти у відміченому трикутнику максимальне від'ємне значення.



Розв'язання задачі розбивається на кілька підзадач: заповнення масиву, пошук указаного елемента та виведення його на екран. Масив заповнимо в процедурі **InpMatr** за допомогою генератора псевдовипадкових чисел і для контролю відразу виведемо на екран. Сам масив та кількість заповнених рядків повертаються з виклику процедури як параметри-змінні.

Пошук максимального від'ємного значення оформимо функцією. Звичайний алгоритм пошуку максимуму починається з того, що деякому ***еталону*** присвоюється початкове значення, наприклад, перше в масиві. Отже, в цій задачі треба врахувати область пошуку, визначити, в якому порядку йдуть її елементи й які індекси має перший з них. Але головна неприємність — необхідно шукати максимальне ***від'ємне*** значення, а перший елемент може не бути від'ємним. Більше того, в області пошуку взагалі може не бути від'ємних чисел.

Оформимо пошук функцією **maxNeg**. Домовимося, що вона повертає 0, якщо від'ємних чисел в області пошуку немає, і будемо обробляти цей особливий випадок при виведенні результату.

Спочатку як еталонне значення візьмемо 0. Далі, якщо в області пошуку знайдено від'ємне значення й еталон рівний 0, то візьмемо це перше від'ємне значення як еталонне. Якщо ж знайдене від'ємне значення більше за еталонне, запам'ятаємо його як еталонне.

```
program maxNegative;
  const maxSz=10; {максимальний розмір}
  type Matrix=array[1..maxSz,1..maxSz] of real;
procedure inpMatr(var a:Matrix; var n:byte);
  var i, j : byte;
begin
  Randomize;
  n := 1+random(10);
  for i:=1 to n do begin
    for j:=1 to n do begin
      a[i,j] := random*100-random*50;
      write(a[i,j]:7:2);
    end;
   writeln;
  end;
end;
```

```
function maxNeg(a:Matrix; n:byte):real;
var i,j:byte; etalon:real;
begin
  etalon:=0;
  for i := n div 2 +1 to n do
    for j := n+1-i to i do
      if a[i,j]<0 then
        if (etalon=0)or(etalon<a[i,j])
        then etalon:=a[i,j];
  maxNeg:=etalon;
end;
  var Mas : Matrix; n : byte; x : real;
Begin
  inpMatr(Mas,n);
  x:=maxNeg(Mas,n);
  if x=0 then
    writeln('Від''ємних значень немає')
  else writeln('Max=',x:8:2);
  readln;
End.
```

4. ФАЙЛ, ФАЙЛОВА ЗМІННА, ФАЙЛОВІ ТИПИ

Під ***файлом*** прийнято розуміти деяку іменований набір даних на зовнішньому носії даних. У мовах програмування такий набір даних, як правило, називають ***фізичним файлом***, а під словом «файл» розуміють ***файлову змінну***, яка представляє файл у програмі Паскаль.

Файл (файлова змінна), як і масив, являє собою послідовність однотипних елементів. Проте, на відміну від масиву, у будь-який момент виконання програми з усіх елементів файла можна обробляти (читати або записувати) тільки один. Він називається ***доступним елементом***; інші в цей момент недоступні. Номер доступного елемента в послідовності елементів файла є значенням спеціальної неявної змінної — ***файлового вказівника*** (див. рис. 1 на с. 22). Номер доступного елемента (значення вказівника) змінюється при виконанні підпрограм обробки файлів

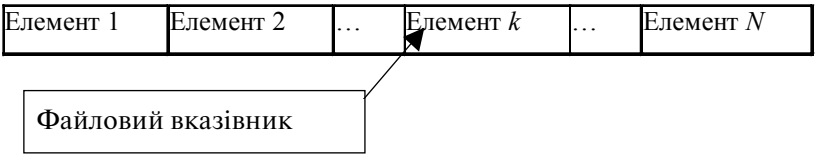


Рис. 1. Фізичний файл і файловий вказівник

Основними діями з файлом є ***введення*** (***читання***) — копіювання значення доступного елемента файла в «звичайну» змінну програми, і ***виведення*** (***запис***) — копіювання значення виразу в доступний елемент файла.

Обробку файлів у мові ***Turbo Pascal*** задають за допомогою стандартних підпрограм, з яких нам знайомі процедури **`read`**, **`readln`**, **`write`**, **`writeln`**. Ім'я файла записують як перший аргумент у викликах цих підпрограм.

У мові ***Turbo Pascal*** є три основних різновиди файлів: типізовані, текстові та безтипові. Головне, чим вони відрізняються, — це набори підпрограм їх обробки. Кожен різновид має специфічні підпрограми, означені тільки для нього. Спочатку представимо їх дуже стисло, а нижче розглянемо роботу тільки з текстовими файлами.

Типізований файл — це послідовність елементів деякого скалярного або структурного типу. Підкреслимо: ці елементи (компоненти) файла є ділянками пам'яті ***на зовнішньому носії даних***. Тип файла задають виразом вигляду **`file of тип`**, де **`тип`** є типом елементів файла.

Розглянемо приклади:

```
type fByte = file of byte; {тип файла байтів}
  aByte = array[1..9] of byte; {тип масиву}
  faByte = file of aByte; {тип файла масивів}
```

Текст — це послідовність символів, поділена на рядки. У мові ***Turbo Pascal*** для файлів-текстів означено тип з іменем **`text`**. Елементами цих файлів є символи, проте тип **`text`** відрізняється від типу **`file of char`**.

По-перше, для текстів означено процедури, незастосовні до типізованих файлів, наприклад, **`readln`** та **`writeln`**.

По-друге, в мові зафіксовано спеціальні символи, що позначають кінці рядків і кінець тексту та обробляються у специфіч-

ний спосіб. Кінець рядка позначається символом з номером 13 (**chr(13)** або **#13**), кінець тексту — символом **#26**.

Безтиповий файл розглядають як послідовність байтів; його тип має ім'я **file**. Наприклад, оголошення двох безтипових файлів може мати вигляд **var f,g:file**. Для них також означено спеціальні підпрограми.

- Одну й ту ж послідовність байтів можна розглядати та обробляти як послідовність або значень деякого типу, або символів із розбиттям на рядки, або байтів.
- Файли можуть бути елементами масивів, але не файлів.

Наприклад, оголошення типу масиву

```
type aText = array[1..2] of text;
```

є допустимим, але такі оголошення типів недопустимі.

```
type badTyp1=file of file of ...;
type badTyp2=file of text;
type badTyp3=file of array[1..2] of file of ...;
```

5. ЗВ'ЯЗУВАННЯ ТА ВІДКРИТТЯ ФАЙЛА

Нехай нижче ім'я **f** позначає файлову змінну (у реальних програмах краще давати файловим змінним змістовні імена, наприклад **workFile**, **inputFile** тощо).

Робота з файловою змінною починається зі ***зв'язування*** її з конкретним фізичним файлом. Для цього ім'я файлової змінної та ім'я фізичного файла в операційній системі (***зовнішнє*** ім'я) задають у виклику процедури **assign**, наприклад **assign(f,'myfile.dat')**. Зовнішнє ім'я записують у апострофах. Після виклику процедури **assign** ім'я файлової змінної позначає фізичний файл.

Файлові змінні з іменами **input** і **output** неявно (без указівки у програмі) оголошено як змінні типу **text** і зв'язано зі ***стандартними файлами введення та виведення*** — клавіатурою та екраном комп'ютера. Ці файли мають зовнішнє ім'я **'con'** (це скорочення від ***console*** — консоль).

- Імена **input** і **output** можна не записувати у викликах підпрограм обробки файлів. Саме такими були виклики **readln** і **writeln** у попередніх розділах.

Перш ніж обробляти зв'язаний файл, його необхідно ***відкрити***. Відкрити файл можна, щоб потім ***читати з нього*** або щоб ***записувати в нього***.

Для читання створеного раніше файла його відкривають за допомогою процедури **reset**. Після виклику **reset(f)** елементи файла можна читати, починаючи з першого.

Для запису файл відкривають за допомогою процедури **rewrite** («перезаписати»). Після виклику **rewrite(f)** послідовність елементів файла **f** стає порожньою. Цю процедуру викликають для створення нового фізичного файла або зміни старого зі знищенням попередніх даних.

Текстові файли можна відкривати, не знищуючи їхні попередні дані. Якщо файлову змінну **f** типу **text** зв'язано з уже існуючим фізичним файлом, то виклик **append(f)** забезпечує, що потім нові символи будуть дописуватися після наявних у файлі.

Після обробки файл треба ***закрити*** за допомогою процедури **close**: **close(f)**. Ця процедура не розриває зв'язку імені **f** із фізичним файлом, але введення та виведення за допомогою імені **f** неможливі до наступного відкриття або нового зв'язування з подальшим відкриттям.

- Якщо файл не закрити після записування в нього, то, можливо, не всі дані, записані у файлову змінну під час виконання програми, насправді потраплять у фізичний файл.
- Спроба закрити вже закриту або ще не відкриту файлову змінну призводить до аварійного завершення програми.

Отже, стандартний порядок дій із файловою змінною такий:

```
Зв'язування (assign)
Відкриття (reset, rewrite або append)
Обробка (read, write та деякі інші підпрограми)
Закриття (close)
```

6. ЗАПИС У ТЕКСТОВИЙ ФАЙЛ

Текстові файли доводиться створювати дуже часто. Наприклад, якщо програмі на вхід потрібні кілька числових констант, їх неважко набрати на клавіатурі. Проте коли таких констант десятки й більше або під час налаштування програми вони вводяться багаторазово, краще записати їх у текстовий файл і у програмі зчитувати дані з текстового файла.

Символи записуються в текст процедурами **write** і **writeln**. Їх перший аргумент — ім'я файлової змінної. При виконанні виклику **write(f,вираз)**, де вираз має скалярний або рядковий тип, обчислюється значення виразу, за ним утворюється константа, тобто послідовність символів, яка представляє значення, і виводиться в текст **f**.

Наприклад, при виконанні виклику

```
write(f,trunc(sqrt(9999)));
```

обчислюється значення 99 і у файл дописуються символи **'9'** і **'9'**.

У виклику можна записати кілька виразів через кому — **write(f,вираз1,вираз2,...);**. Такий виклик насправді виконується як послідовність викликів

```
write(f,вираз1); write(f,вираз2); ...
```

Процедура **writeln** відрізняється лише тим, що за виклику

```
writeln(f, список_виразів_скалярних_типів);
```

після останньої константи в текст додається позначення кінця рядка, точніше, символи **#13#10**. Список виразів може бути порожнім — тоді в текст записуються тільки **#13#10**.

Приклад. Розглянемо створення тексту з цілими константами, які отримуються від клавіатури, копіюються в текстовий файл та відокремлюються в ньому пропусками.

```
program CreateInts;
  var f:text;   {файл-текст}
      x:integer; {число від клавіатури}
begin
  writeln('Створення тексту.',
     'Кінець - <Ctrl+Z> замість константи');
  assign(f,'myfile.txt'); rewrite(f);
  write('Введіть цілу константу: ');
  while not eof do begin
    readln(x);
    write(f,' ',x); {перед константою пропуск}
    writeln('Введіть цілу константу: ');
  end;
  close(f)
end.
```

Для закінчення роботи замість введення нової константи треба натиснути **Ctrl + Z** і **Enter**.

Приклад. Розглянемо програму створення тексту, у першому рядку якого записано цілі числа ***m*** і ***n*** — кількості рядків і стовпчиків числової матриці (не більше 20). Наступні ***m*** рядків тексту містять по ***n*** цілих чисел (елементів матриці), відокремлених пропусками. Розміри матриці задаються на клавіатурі, а її елементи утворюються за допомогою генератора псевдовипадкових чисел.

```
program CreateMatr;
  const maxSz=20; {максимальний розмір}
  var f:text;     {файл-текст}
    i,k:byte;  {поточні індекси}
begin
  assign(f,'matrix.txt'); rewrite(f);
  write('Рядків та стовпчиків(1..',maxSz,')>');
  readln(m,n);
  writeln(f,m,' ',n);
  Randomize;
  for i:=1 to m do begin
    for k:=1 to n do
      write(f, Random(65535)-32768, ' ');
    writeln(f); {перехід на новий рядок}
  end;
  close(f)
end.
```

У викликах процедур **write** і **writeln** після виразу через двокрапку можна вказати ширину поля W для запису значення виразу, наприклад **write(f,sqr(x):4)**. Тут $W = 4$. Нехай для запису значення виразу насправді потрібно L символів, наприклад, в останньому виклику $L = 1$ при $x < 4$, $L = 2$ при $3 < x < 10$, $L = 3$ при $9 < x < 34$ тощо. Якщо $L < W$, то перед символами значення додається $W - L$ пропусків; якщо ж $L \le W$, виводяться всі символи. Отже, при $x = 3$ друкуються три пропуски й **9**, а при $x = 100$ — усі п'ять символів **10000**.

Після виразу дійсного типу можна також указати кількість N цифр дробової частини, які виводяться після десяткової крапки, наприклад **write(f, sqrt(x):7:3)**. Якщо N указано, то ви-

водиться константа з фіксованою крапкою та *N* цифрами після крапки, інакше — нормалізована з порядком. У цьому випадку при $x = 2$ виводяться два пропуски та `1.414`. Остання цифра є результатом округлення.

7. УВЕДЕННЯ ДАНИХ БАЗОВИХ ТИПІВ ІЗ ТЕКСТОВОГО ФАЙЛА

Для введення даних з текстових файлів використовують процедури **read** та **readln**. Спочатку розглянемо процедуру **read**. Її виклик має такий найпростіший вигляд.

```
read(f, ім'я_змінної_базового_типу);
```

Уведення символа. При виконанні виклику `read(f,X)` змінній `X` типу `char` присвоюється доступний символ тексту, яким би він не був, а доступним стає наступний за ним. Виняток — якщо доступний `#26`, то `X` отримує значення `#0` і символ `#26` залишається доступним.

Уведення числової константи. Ціла константа — це послідовність цифр, можливо, зі знаком `'+'` або `'-'` на початку, яка задає ціле число відповідного цілого типу; між знаком та першою цифрою в тексті не може бути пропусків. Дійсна константа — це послідовність цифр та інших символів зі структурою констант мови Паскаль, наприклад `1.1`, `2.`, `0.99`, `1e-3`, `-2.73E+02`.

Числові константи в текстах відокремлюються пропусками в довільній кількості. Символи табуляції та кінця рядка також будемо називати пропусками.

Виклик `read(f,X)`, де `X` — ім'я цілої або дійсної змінної, виконується так. З тексту від доступного символа читаються пропуски, а потім — символи константи до найближчого пропуску (можливо, до символу `#26` або до кінця файла). Доступним після читання константи буде перший пропуск після неї (відповідно, `#26` або кінець файла). Якщо символи утворюють константу потрібного типу, то за ними обчислюється значення й присвоюється змінній. При дійсній змінній `X` у тексті може знаходитися й ціла константа — за нею обчислюється дійсне значення.

Символи можуть не утворювати константу відповідного типу — тоді виникає помилкова ситуація і виконання програми аварійно

завершується. Наприклад, помилковими є послідовності символів **- 2** (пропуск між знаком і цифрою), **12345m**, **123-** (присутні нецифрові символи там, де їх не може бути) або **13.**, якщо читається значення цілої змінної.

Якщо доступний кінець файла або від поточної позиції в файлі до його кінця чи найближчого символа **#26** записано лише пропуски, то при спробі прочитати число відповідна змінна отримує значення 0.

В одному виклику процедури **read** можна указати кілька змінних.

```
read(f,
список_позначень змінних_числових_типів);
```

Цей виклик виконується так само, як і відповідна послідовність викликів.

```
read(f, ім'я_змінної_1);
read(f, ім'я_змінної_2);
...
```

Читання констант базових типів за процедурою **readln** аналогічно процедурі **read**. Відмінність полягає в тому, що після читання константи всі символи тексту, які залишилися до найближчого кінця рядка в тексті, пропускаються разом із ним. Доступним стає перший символ наступного рядка тексту (якщо до кінця рядка зустрінеться символ **#26**, доступним стане він). Якщо у виклику указано кілька імен змінних, то «хвіст» рядка пропускається тільки після останньої константи. Якщо список імен порожній, то виклик **readln(f)** пропускає поточний рядок тексту.

Приклад. Нехай у тексті **f** записано такі символи:

1		2		3	#13
	5	5	#26		

Нехай **x**, **y**, **z**, **t** — імена цілих змінних. Розглянемо приклади викликів процедур і значень, яких набудуть ці змінні.

Виклики	x	y	z	t
read(f,x,y); read(f,z,t)	1	2	3	55
readln(f,x,y); read(f,z,t)	1	2	55	0
readln(f,x); readln(f,y,z,t)	1	55	0	0

• Спосіб виконання процедур уведення дозволяє розглядати текст як послідовність даних, що задають значення різних типів, тобто як ***потік різнотипних даних***. Звичайно, дані мають відповідати типам змінних, які у процесі виконання програми отримують значення з цього потоку.

8. ПРИКЛАДИ ВВЕДЕННЯ ПОСЛІДОВНОСТЕЙ ДАНИХ

Типовою є ситуація, коли вхід програми утворено послідовністю однотипних значень у текстовому файлі. Послідовність уводиться в циклі, вигляд якого відповідає способу визначення кінця послідовності.

Розглянемо три таких способи:

- довжину послідовності задано на її початку;
- кінець послідовності задано спеціальним значенням;
- кінець даних визначено кінцем тексту.

У деяких задачах під час уведення можуть виникати умови, за яких процес уведення вхідних даних слід закінчити ще до того, як буде прочитано всі дані. Розглянемо також деякі особливості введення символів.

ДОВЖИНУ ПОСЛІДОВНОСТІ ЗАДАНО НА ЇЇ ПОЧАТКУ

Спочатку задається кількість значень, а потім самі значення у відповідній кількості. Потрібна кількість виконань циклу введення відома заздалегідь, тому в таких ситуаціях використовують **for**-оператор.

Приклад. Повернемося до прикладу в параграфі 3 й припустимо, що числова матриця не створюється за допомогою генератора псевдовипадкових чисел, а вводиться з текстового файла **matrix.txt**. У його першому рядку записано кількості рядків і стовпчиків матриці, а далі задано числові значення у відповідній кількості. Тоді у програмі (див. параграф 3) треба написати й викликати замість процедури **gMatr** таку процедуру.

```
procedure readMatr(var a:Matrix; var m,n:byte);
  var f:text; {файл-текст}
    i,k:byte; {поточні індекси}
```

```
begin
  assign(f, 'matrix.txt'); reset(f);
  readln(f,m,n);
  for i:=1 to m do
    for k:=1 to n do
      read(f,a[i,k]);
  close(f);
end;
```

КІНЕЦЬ ПОСЛІДОВНОСТІ ЗАДАНО СПЕЦІАЛЬНИМ ЗНАЧЕННЯМ

Якщо заздалегідь відомо спеціальне значення, яким у вхідній послідовності позначено її кінець, то введення зручно задати за допомогою **repeat**-оператора, оскільки треба ввести не менше одного значення перед тим, як буде виявлено ознаку кінця введення.

Приклад. У перукарні працює один перукар. Клієнти приходять, займають чергу (якщо вона є) і стрижуться в порядку черги. Для кожного клієнта відома тривалість його стрижки ***t***: клієнт залишає салон через ***t*** одиниць часу після початку стрижки. Моменти приходу клієнтів задано відносно початкового моменту часу. Треба вивести моменти виходу клієнтів у інший текстовий файл (по одному на рядок).

Уточнимо вигляд вхідних даних, які мають читатися з файла, зв'язаного зі змінною **f** типу **text**. Припустимо, що стрижка відбувається не миттєво, тому вхідними даними є пари цілих чисел **x1**, **t1**, **x2**, **t2**, ..., де числа **x** не спадають, а тривалості **t** додатні (**t** = 0 означає кінець вхідних даних). Пари чисел задано по одній на рядок; числа відокремлено пропуском.

Перше наближення до розв'язання є очевидним.

```
repeat
  readln(f,x,t);
  if t>t0 {значення t0=0 задає закінчення}
  then за x і t обчислити момент виходу у
until t=t0;
```

Припустимо, що клієнти стрижуться без пауз: стрижка нового клієнта, якщо він уже прийшов, починається відразу після виходу попереднього. За **x1** і **t1** можна обчислити момент виходу

y1: **y1 = x1+t1**. Проте кожен наступний клієнт починає стригтися тільки після того, як закінчив стригтися попередній, тому $y_i = \max\{y_{i-1} + t_i, x_i + t_i\}$. Щоб не розглядати окремо першого й наступних клієнтів, приймемо початковий момент за час виходу «нульового» клієнта.

Отже, програма набуває такого вигляду:

```
program Barber; {Barber – перукар}
  var x,t,t0,y:word;
      f,g:text; {вхідний та вихідний тексти}
begin
  t0:=0;
  assign(f, 'barber.in'); reset(f);
  assign(g, 'barber.out'); rewrite(g);
  y:=0;
  repeat
    readln(f, x, t);
    if t>t0 then begin
       if y < x
       then y := x+t
       else y := y+t;
       writeln(g, y);
    end
  until t = t0;
  close(f); close(g);
end.
```

Ще один спосіб задати кінець послідовності — повторити перше або останнє значення послідовності. Схема розв'язання залишається аналогічною, тільки «особливе значення» отримується після введення, а не присвоювання (**t0:=0** на початку тіла щойно наведеної програми).

КІНЕЦЬ ДАНИХ ВИЗНАЧЕНО КІНЦЕМ ТЕКСТУ

Прочитати послідовність значень, записану в текстовому файлі, можна за допомогою функції **eof** у циклі такого вигляду.

```
while not eof(f) do begin
  read(f,v); {v – ім'я змінної}
  обробка v
end;
```

З виклику **eof(f)** повертається значення **true**, якщо доступний кінець файла **f** або символ **#26**. Читання в тілі циклу відбувається після того, як із виклику **eof(f)** повернулося значення **false**, тобто ***кожному введенню передує успішна перевірка, чи не прочитано файл***.

- Якщо в наведеному вище циклі змінна **v** має числовий тип, то необхідно забезпечити, щоб між останньою числовою константою та кінцем вхідного файла не було порожніх символів. Якщо цього не зробити, буде прочитано зайве нульове значення, що не завжди бажано. Взагалі, якщо всередині тексту з числовими константами присутній символ **#26**, то наведений цикл оброблятиме лише частину тексту до символа **#26**.

Приклад. Текстовий файл містить послідовність цілих констант типу **integer**, відокремлених пропусками. Треба прочитати їх та надрукувати їх кількість і суму.

Будемо вводити константи й накопичувати їх суму, поки не прочитаємо весь текст.

```
program Summa;
  var f:text;          {вхідний текст}
      v:integer;       {поточне число}
      n,sum:longint;   {кількість і сума}
begin
  sum:=0; n:=0;
  assign(f, 'numbers.in'); reset(f);
  while not eof(f) do begin
    read(f,v);
    inc(n); inc(sum,v)
  end;
  writeln('Введено чисел:',n,'. Їх сума:',sum);
  close(f);
end.
```

Якщо між останньою константою та кінцем вхідного файла записано хоча б один пропуск, то буде враховано одне зайве число. Перевірте це.

Приклад. Відрізок [***a***; ***b***] прямої ***Ox*** задається парою чисел ***a***, ***b***, де $a \le b$. Перетином двох відрізків є або відрізок, або порожня

множина точок, наприклад, $[1;3]\cap[2;4]=[2;3]$, $[1;2]\cap[3;4]=\varnothing$, $[1;2]\cap[2;3]=[2;2]$. Треба ввести з текстового файла пари чисел, що задають відрізки, і знайти перетин заданих відрізків.

Припустимо, що кожен рядок вхідного тексту задає кінці відрізка. Числа в рядку відокремлено пропуском.

Для збереження поточного перетину означимо змінні **lb** і **hb** (скорочення від ***low bound*** і ***high bound*** — нижня й верхня межа). Ознакою того, що перетин став порожнім, буде умова **lb** > **hb**.

Спочатку відрізків немає, тому перетин порожній — виразимо це початковими значеннями **lb** = **1**, **hb** = **0**. Далі значеннями **lb** та **hb** мають стати кінці першого відрізка. Потім у циклі вводяться інші відрізки та обчислюється перетин.

Після того, як перетин уже прочитаних став порожнім, продовжувати читання відрізків немає сенсу — треба відразу видати відповідь і закінчити роботу. Тому до умови продовження циклу обробки відрізків додамо умову того, що перетин не став порожнім.

```
program segments;
  var f:text;
    a,b:real;    {поточний відрізок}
    lb,hb: real; {поточний перетин}
begin
  assign(f,'segments.txt'); reset(f);
  lb:=1; hb:=0;
  if not eof(f) then readln(f,lb,hb);
  while not eof(f) and (lb<=hb) do begin
    readln(f,a,b);
    if a>lb then lb:=a;
    if b<hb then hb:=b;
  end;
   {текстовий файл прочитано або перетин порожній}
  if lb>hb
  then writeln('перетин порожній')
  else writeln('[',lb,';',hb,']')
end.
```

ДЕЯКІ ОСОБЛИВОСТІ ВВЕДЕННЯ СИМВОЛІВ

Приклад. Розглянемо програму копіювання будь-якого файла як текстового в інший файл. Символи файла вводяться з тексту та виводяться по одному. Власне копіювання для ілюстрації оформимо у вигляді процедури, параметризованої файлами.

Параметри файлових типів оголошуються в заголовках підпрограм ***тільки як параметри-змінні***.

```
program FCopy;
  var f,g:text;
procedure copyText(var f,g:text);
  var c:char;
begin
  reset(f); rewrite(g);
  while not eof(f) do begin
    read(f,c); write(g,c);
  end;
  close(f); close(g);
end;
begin
assign(f,'inp.txt'); assign(g,'inpcopy.txt');
  copyText(f,g);
end.
```

Ця програма буде без проблем копіювати файл, якщо в ньому немає символа **#26**. Проте, як тільки у вхідному тексті доступним буде **#26**, з виклику **eof(f)** повернеться **true** і копіювання закінчиться незалежно від подальших символів у вхідному файлі.

Щоб копіювати фізичні файли з іншими іменами, перед викликом програми треба поміняти їх імена у викликах **assign**. Зокрема, щоб копіювати символи з клавіатури в текстовий файл, замість імені **'inp.txt'** треба указати **'con'**. Для виведення на екран треба указати **'con'** замість **'inpcopy.txt'**.

▣

Особливу ситуацію при посимвольному введенні з тексту може складати кінець рядка. Для визначення цієї ситуації використовують функцію **eoln**. Якщо доступним у тексті **f** є кінець рядка, з виклику **eoln(f)** повертається **true**, інакше повертається **false**.

Приклад. На вхід програми подається текст, кожен рядок якого містить послідовність дужок **(** і **)**, тобто ***дужковий вираз***; інших символів у рядку немає. Довжини рядків можна представити в типі **longint**. Дужковий вираз у рядку є ***правильним***, якщо це **()**, або якщо в ньому спочатку йде **(**, потім правильний вираз, потім **)**, або якщо він є послідовністю правильних виразів. Наприклад, вирази **(())**, **()()()** є правильними,**)(** — ні. Порожній вираз також вважається правильним. З'ясувати, чи є вирази в рядках правильними, і вивести в один рядок іншого тексту послідовність із символів **0** і **1** (**1**, якщо вираз у рядку правильний, інакше **0**).

Скористаємося лічильником **NOpen**: його значенням буде кількість відкритих і ще не закритих дужок у прочитаній частині рядка. На початку кожного рядка **NOpen** = 0. Рядок із дужковим виразом вводимо по одному символу. Кожна відкриваюча дужка «**(**» збільшує **NOpen** на 1, кожна закриваюча — «**)**»зменшує. Якщо з'являється «**(**», для якої не було попередньої «**)**», отримаємо **NOpen** < 0. У цій ситуації вираз неправильний — решту рядка можна пропустити, не аналізуючи далі. Якщо по закінченні рядка **NOpen<>0**, тобто кількість дужок незбалансовано, то вираз не є правильним.

Перед кожним введенням символу треба перевірити, чи не закінчився рядок. Для цього звернемося до функції **eoln**. Якщо з її виклику повертається **false**, тобто доступний символ тексту не позначає кінець рядка, то можна його вводити та обробляти. Інакше рядок закінчено й треба видати результат його обробки — **ord(NOpen=0)**.

Окрім виведення результату, у кінці рядка треба ще підготуватися до обробки наступного рядка, тобто пропустити кінець рядка та присвоїти **NOpen** значення 0. Отже, оформимо дії з обробки кінця рядка процедурою **endLine**.

Особливістю в кінці тексту є те, що останній рядок може як мати, так і не мати позначення кінця. Якщо кінець рядка не позначено, то, щоб видати результати обробки цього рядка, треба знати, чи було введено символи між останнім кінцем рядка та кінцем тексту. Для цього оголосимо змінну **NInp** — лічильник символів, прочитаних у поточному рядку. Спочатку та після закінчення кожного рядка йому присвоюється 0. Якщо в кінці

тексту **NInp>0**, то останній рядок прочитано, але результат його обробки не виведено, тому треба викликати **endLine**.

```
program Balance;
  var f,g:text;     {файли входу та виходу}
      c  : char;    {поточний символ}
      NOpen:longint; {лічильник дужок}
      NInp :longint; {лічильник символів}
procedure endLine; {обробка кінця рядка}
begin
  readln(f);
  write(g,ord(NOpen=0));
  NOpen:=0; NInp:=0;
end;
Begin
  assign(f, 'balance.txt'); reset(f);
  assign(g, 'balance.sol'); rewrite(g);
  NInp:=0; NOpen:=0;
  while not eof(f) do
     if eoln(f)
     then endLine {кінець рядка}
     else
     begin
       read(f,c); inc(NInp);
       if c='('
       then inc(NOpen)
       else dec(NOpen);
       if NOpen<0 {баланс неможливий}
       then endLine
     end;
  {файл прочитано; в останньому рядку
    може не бути позначення кінця рядка}
  if NInp>0 then endLine;
  close(f); close(g);
End.
```

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке масив? Як він описується?

2. Яким може бути тип елементів масиву, а яким — індексів масиву?

3. Як здійснюється доступ до окремих елементів масиву?

4. Як здійснюється обробка масивів?

5. Які обмеження накладаються на розмірність масиву в Turbo Pascal?

6. Як зберігається багатовимірний масив у пам'яті комп'ютера?

7. Які масиви називають квадратними і які вони мають особливості?

8. Що таке файл і що таке файлова змінна?

9. Що таке файловий вказівник і для чого він використовується?

10. Які різновиди файлів є в Turbo Pascal і чим вони відрізняються?

11. Файлами якого типу є клавіатура та екран?

12. Що таке текст? Чим відрізняється обробка файлів цього типу від інших?

13. Яким є стандартний порядок дій з файловою змінною?

14. Що відбувається при зв'язуванні файлової змінної з файлом?

15. Як можна відкрити текстовий файл?

16. Які наслідки можливі, якщо не закрити файл, відкритий для запису?

17. Як програмується зчитування з файла з невідомою наперед кількістю елементів?

ЗАДАЧІ

1. Задано одновимірний числовий масив.

Без використання допоміжного масиву:

a) значення елементів масиву циклічно зсунути на одну позицію ліворуч;

b) значення елементів масиву циклічно зсунути на одну позицію праворуч;

c) значення елементів масиву циклічно зсунути на k позицій ліворуч;

d) значення елементів масиву циклічно зсунути на k позицій праворуч;

e) реалізувати його дзеркальне перетворення;

f) перебудувати масив так, щоб спочатку підряд у тому ж порядку було розташовано всі ненульові значення елементів масиву, а потім усі нульові;

g) за даним значенням x перебудувати масив так, щоб спочатку було розташовано всі значення елементів, що менші x, потім усі рівні x, а потім — решту значень. Порядок значень усередині групи можна змінювати.

2. За лінійним масивом цілих чисел знайти k, при якому значення виразу $|A[1] + A[2] + ... + A[k] - A[k + 1] - A[k + 2] - ... - A[n])|$ (модуль різниці сум елементів правої та лівої частини, на які k розбиває масив) є мінімальним.

3. У лінійному масиві цілих чисел знайти найдовшу довжину «пилки» — послідовності чисел, що чергуються за зростанням та спаданням, наприклад **2 5 3 7 4 6 5 9**.

4. У лінійному масиві цілих чисел поміняти місцями два фрагменти масиву від позиції **k** до позиції **m** та від позиції **p** до позиції **s**. Наприклад, якщо є масив

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

і $k = 5$, $m = 8$, $p = 13$, $s = 19$, то результатом обміну буде масив

1 2 3 4 13 14 15 16 17 18 19 9 10 11 12 5 6 7 8 20

5. Дано прямокутну матрицю цілих чисел розмірами M на N. В її кожному рядку вилучити елементи, значенням яких є номер їх рядка. Всі наступні елементи зсунути ліворуч, а останні елементи рядка заповнити нулями.

6. Дано двовимірний масив цілих чисел розмірами M на N. Для кожного рядка знайти найменший спільний дільник елементів цього рядка.

7. У прямокутному масиві цілих чисел знайти число, що зустрічається найчастіше. Якщо таких чисел декілька, визначити найменше з них.

8. У прямокутну матрицю цілих чисел розмірами M на N вставити після її k-го рядка перший з рядків, що містять максимальний елемент цього масиву. Врахувати, що рядків побільшає.

9. На прямокутному клітчастому полі розмірами N на M намальовано прямокутники, що містять по кілька цілих клітин, які не накладаються один на одного і не мають спільних меж. Незамальовані клітини поля подано значенням 0, кольори прямокутників — цілими числами від 1 до 255. Підрахувати кількість прямокутників та знайти площу найбільшого з них.

10. Перевірити, чи становить квадратний масив магічний квадрат (його заповнено цілими числами від 1 до n^2 так, що суми чисел у кожному рядку, кожному стовпчику та в обох діагоналях однакові).

11. Елемент матриці називається сідловою точкою, якщо він одночасно є найменшим у своєму рядку та найбільшим у своєму стовпчику. За матрицею цілих чисел розмірами M на N з'ясува-

ти, чи містить вона сідлові точки, і, якщо так, обчислити індекси рядка й стовпчика однієї з них.

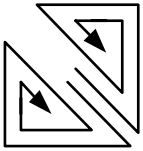
12. За квадратною числовою матрицею вивести послідовність чисел обходом матриці:

a) «змійкою», починаючи по горизонталі з лівого верхнього кута;

b) «спіраллю» проти годинникової стрілки з лівого верхнього кута;

c) «спіраллю» за годинниковою стрілкою з лівого верхнього кута.

13. Утворити два одновимірні масиви шляхом копіювання в них елементів заданої цілочислової квадратної матриці. В один масив переписати всі елементи вище головної діагоналі матриці, в інший — нижче її. Порядок елементів указано на рисунку.



14. Перетворити задану квадратну матрицю дійсних чисел так, щоб верхній над її бічною діагоналлю трикутник містив її максимальне значення, нижній — мінімальне.

15. Створити текстовий файл із таблицею степенів числа 2 від 1 до 30.

16. Підрахувати кількість рядків у текстовому файлі.

17. Дописати до одного текстового файла другий текстовий файл.

18. Прочитати текстовий файл і вивести кількості повторень кожного з символів, які в ньому зустрічаються.

19. Дано два текстові файли. Визначити, чи збігаються посимвольно відповідні рядки першого та другого файлів. За збігу рядків вивести в рядок третього файла відповідь «OK», інакше вивести символ з рядка першого тексту, яким починається відмінність, та всі подальші символи до кінця рядка. Якщо довжини файлів різні, вивести про це окреме повідомлення.

20. Текстовий файл має рядки довжиною не більше 80. Прочитати та вивести його зміст на екрані «сторінками»: після виведення кількох рядків тексту треба запитати, чи продовжувати виведення, і, залежно від відповіді користувача, продовжувати або завершити роботу.

21. Прочитати натуральне число типу **longint** та вивести його p-ковий запис ($p \le 36$) у вигляді многочлена зі степенями числа p; піднесення до степеня задати знаком **^**. Якщо цифра 0, відповідний степінь числа p виводити не треба, а якщо 1, то вивести його без цифри як множник, наприклад:

1407 = 10^3+4*10^2+7*10^0.

22. З текстового файла зчитується послідовність ненульових чисел $a_1, a_2, a_3, ..., a_n$, яка закінчується нулем (він у послідовність не входить). Кількість чисел нічим не обмежено.

Написати програму, яка:

a) визначає найбільший елемент послідовності;

b) обчислює середнє арифметичне введених чисел;

c) обчислює середнє геометричне введених чисел;

d) визначає кількість додатних та від'ємних чисел;

e) перевіряє послідовність на неспадання (незростання);

f) обчислює кількість суміжних чисел різного знака;

g) підраховує суму $a_1 \cdot a_2 + a_2 \cdot a_3 + ... + a_{n-1} \cdot a_n + a_n \cdot a_1$.

23. У текстовому файлі записано цілі числа; їх кількість нічим не обмежено. Відомо, що тільки одне з них повторюється непарну кількість разів. Знайти це число, зчитавши текстовий файл один раз.

24. Написати програму створення текстового файла з рядками довжиною до одного мільйона символів. Текст має бути вхідним для програми (див. останні приклад розділу).

25. Є два непорожні текстових файли, у кожному рядку яких записано додатне ціле число, причому послідовність чисел неспадні. Записати в третій текстовий файл неспадну послідовність чисел, яка є результатом злиття двох заданих. Числа вивести по 10 на рядок (останній рядок може бути неповним). Наприклад, при заданих послідовностях (2, 2, 4, 6), (1, 3, 6, 7) утворюється (1, 2, 2, 3, 4, 6, 6, 7).

26. Множину цілих чисел подано текстом, у кожному рядку якого записано цілі числа, упорядковані за зростанням. За двома такими файлами створити третій файл, який також є упорядкованим і представляє їх: а) об'єднання; б) перетин; в) різницю.

27. У текстовому файлі записано послідовність цілих чисел; їх кількість нічим не обмежено. Відомо, що одне з них зустрічається у послідовності частіше ніж усі інші, разом узяті. Знайти це число.

ðîçä³ë 2

ÑÎÐÒÓÂÀÍÍß ËªÍªÉÍÈÕ ÌÀÑÈÂ²Â

1. ПОНЯТТЯ СОРТУВАННЯ

Загальне значення слова «***сортування***» — *це розподіл елементів на групи за деякою ознакою*, наприклад, розподіл яблук по сортах за їх якістю, листів за поштовими індексами, банок із фарбами за кольором тощо. Проте у програмуванні під сортуванням розуміють ***упорядкування елементів*** (за деякою характеристикою), наприклад, вишикування учнів за зростом на уроці фізкультури, розташування слів у словнику в алфавітному порядку, упорядкування точок площини за зростанням координати *x* тощо.

Як бачимо, об'єктами впорядкування можуть бути найрізноманітніші елементи. Головне, щоб їх можна було ***порівнювати***, тобто існував би закон, який дозволяє довільні два елементи правильно розташувати один відносно іншого («*A* має стояти після *B* або перед ним»).

Величину, за якою сортуються елементи, називають ***ключем сортування***. Це може бути число, послідовність символів або складніша величина, наприклад, дата, яка містить рік, місяць та день.

Як відомо, дані в комп'ютері подано в числовому вигляді, тому програмісти під сортуванням традиційно розуміють упорядкування за зростанням або спаданням числових величин.

Сортування з'явилося у практичній діяльності задовго до появи комп'ютерів і є необхідним у багатьох практичних задачах.

Мета сортування — прискорити подальший пошук та обробку даних.

Приклад 1. Якщо нам невідомо, де в словнику розташовано слово, що починається тією чи іншою буквою, ми розкриваємо словник приблизно на середині. Якщо слово має бути в словнику далі, шукаємо його тільки в другій половині словника. Тепер ми розкриваємо словник на середині другої половини. Аналогічно, якщо слово має бути розташовано в першій половині, шукаємо тільки в ній. Щоразу, заглядаючи в словник, ми поділяємо «простір пошуку» навпіл, зменшуючи його приблизно вдвічі. Описаний пошук називається ***бінарним***. Він дозволяє нам знайти слово в словнику за кілька секунд завдяки саме тому, що словник упорядковано за алфавітом. Якби цього не було, довелося б перебирати всі слова підряд. А якщо їх сто тисяч, то скільки тривав би цей процес?

Отже, упорядкування даних має дуже велике значення. Недарма за недовгу історію програмування було створено десятки алгоритмів сортування та їх численні реалізації.

Традиційно розрізняють ***внутрішнє*** та ***зовнішнє*** сортування. Перше впорядковує масиви в оперативній (внутрішній) пам'яті комп'ютера, друге оперує з даними на зовнішніх носіях (файлами), об'єм яких не дозволяє записати їх цілком в оперативну пам'ять.

Нижче подано тільки методи внутрішнього сортування. Їх можна класифікувати за кількома ознаками. Головною з них є ***швидкість виконання***; важливим є також об'єм додаткової пам'яті.

Усі алгоритми сортування поділяють на ***базові*** та ***швидкі***. Базові працюють порівняно повільно, проте їх легко записати та налагодити. Швидкі, як правило, є громіздкими та складними для розуміння, але сортують набагато швидше, особливо великі масиви даних. Щоб уточнити, що означає «швидкий алгоритм», нам знадобиться поняття складності алгоритму, описане в наступному параграфі.

Перш ніж розглядати поняття складності та алгоритми сортування, зафіксуємо деякі деталі цих алгоритмів. Вважатимемо, що треба відсортувати за неспаданням лінійний масив цілих чисел з індексами від **1** до **N_max**, тобто діють такі оголошення.

```
const N_max = 1000;
type aInt = array [1..N_Max] of integer;
```

У процедури сортування буде передаватися масив (**var A:aInt**) та кількість елементів у його початку, що мають сортуватися (**n:word**).

У алгоритмах сортування дуже часто обмінюються значення двох елементів. Будемо вважати, що цей обмін реалізує така допоміжна процедура.

```
procedure swap(var a, b : integer);
  var t : integer;
begin
  t := a: a := b; b := t;
end;
```

2. ПОНЯТТЯ СКЛАДНОСТІ АЛГОРИТМУ

Нагадаємо: більшість задач у програмуванні є масовими, тобто вимога «обчислити...» або «визначити...» стосується деякої множини конкретних вхідних даних. Ці конкретні дані визначають ***екземпляри задачі***. Наприклад, конкретні трійки числових коефіцієнтів визначають екземпляри задачі розв'язання квадратного рівняння. Або конкретний числовий масив визначає екземпляр задачі упорядкування значень у масиві за неспаданням.

Екземпляри багатьох задач можна охарактеризувати значенням деякого числового параметра, який називають ***розміром екземпляра задачі***. Цим розміром прийнято вважати ***кількість біт*** у вхідних даних, якими подано екземпляр. Проте у багатьох задачах під розміром екземпляра розуміють деяку величину, прямо пропорційну вказаній кількості біт. Найчастіше це ***кількість скалярних значень***, що визначають екземпляр задачі, наприклад, довжина масиву або кількість чисел, які треба ввести.

- Отже, найчастіше (хоча й не завжди) ***екземпляр задачі має розмір n***, якщо задається даними, складеними з *n* скалярних значень.

Головна мета наших міркувань — познайомитися з тим, як розмір екземпляра задачі впливає на тривалість виконання того чи іншого алгоритму. Для цього нам знадобиться поняття елементарної дії.

Узагальнимо операції (присвоювання, порівняння, додавання, множення тощо) над скалярними значеннями (числами, байтами тощо) терміном ***елементарна дія***. Вважатимемо, що час виконання будь-якої елементарної дії не залежить від її операндів і самої дії. Тоді час виконання програми буде прямо пропорційним кількості елементарних дій. Якщо не звертати уваги на коефіцієнт цієї пропорційності, то ***час виконання можна вимірювати кількістю елементарних дій***.

• Головну роль у понятті складності алгоритму відіграє не кількість елементарних дій, а характер її зростання при збільшенні розміру екземплярів задачі.

Пояснимо це твердження. Нехай A — алгоритм розв'язання деякої масової задачі. При розв'язанні екземпляра задачі за цим алгоритмом виконується конкретна кількість елементарних дій. Кожному можливому значенню розміру $n = 1, 2, 3, \ldots$ відповідає найбільша кількість елементарних дій по всіх екземплярах цього розміру. Ця відповідність є функцією, яка виражає ***залежність кількості дій від розміру***.

• Функція $F_A(n)$, означена як найбільша кількість елементарних дій при розв'язанні екземплярів задачі розміру n за алгоритмом A, називається ***часовою складністю*** (***складністю***) алгоритму A.

Аналітичне вираження функції $F_A(n)$ для реальних алгоритмів, як правило, неможливе й не потрібне. Практичне значення має так званий ***порядок зростання*** $F_A(n)$ відносно n. Його виражають за допомогою іншої функції, яка має простий аналітичний вираз і є ***оцінкою*** для $F_A(n)$.

Функція $G(n)$ називається ***оцінкою згори*** для функції $F(n)$, якщо існують додатне число C_2 та натуральне m, для яких $F(n) \le C_2 G(n)$ при $n > m$. Цей зв'язок між функціями позначають як «O»: $F(n) = O(G(n))$.

Функція $G(n)$ називається ***оцінкою знизу*** для функції $F(n)$, якщо існують додатне число C_1 та натуральне m, для яких $C_1 G(n) \le F(n)$ при $n > m$. Цей зв'язок між функціями позначають як «Ω»: $F(n) = \Omega\big(G(n)\big)$.

Функція $G(n)$ називається ***оцінкою*** для функції $F(n)$, або $F(n)$ є ***функцією порядку*** $G(n)$, якщо існують додатні скінченні числа C_1, C_2 та натуральне m, для яких $C_1 G(n) \le F(n) \le C_2 G(n)$ при $n > m$. Для позначення такого зв'язку між функціями вживають знак «Θ»: $F(n) = \Theta\big(G(n)\big)$.

Інколи користуються такими означеннями:

Функція $F(n)$ називається функцією порядку $G(n)$ при великих n, якщо $\lim\limits_{n\to\infty} \dfrac{F(n)}{G(n)} = C$, де $0 < C < \infty$. Функція $F(n)$ називається

функцією порядку менше $G(n)$ за великих n, якщо $\lim\limits_{n\to\infty}\dfrac{F(n)}{G(n)}=0$.

Цей зв'язок між функціями позначають як «o»: $F(n) = o(G(n))$.

Для оцінювання складності реальних алгоритмів використовують логарифмічну, степеневу та експоненціальну функції, а також їх суми, добутки та підстановки. Усі вони монотонно зростають і просто виражаються. Наприклад, $n(n-1)=\Theta\left(n^2\right)$, оскільки $0,5n^2 < n(n-1) < n^2$ при $n > 2$. Аналогічно неважко переконатися, що $n^3+100n^2=\Theta\left(n^3\right)=o\left(n^{3,1}\right)=o\left(2^n\right)$, $100\log_2 n+10000=\Theta\left(\log_2 n\right)=\Theta\left(\lg n\right)=o(n)$. Очевидно також, що будь-яка додатна константа C має оцінки $O(1)$ та $\Theta\left(1\right)$.

Приклад. Розглянемо задачу обчислення суми $n + 1$ доданків $s=1/0!+1/1!+1/2!+...+1/n!$. Екземпляр задачі визначається скалярним значенням n, яке й приймемо за розмір екземпляра задачі. Для обчислення суми треба додати $n + 1$ число, тому складність додавань має оцінку $\Theta(n)$. Якщо кожен доданок $a_k = 1/k!$ обчислювати безпосередньо за цією формулою, то складність буде $\Theta(k)$. Тому сумарна по $k = 0, 1, 2, ..., n$ складність буде

$$\Theta(0)+\Theta(1)+\Theta(2)+...+\Theta(n)=\Theta(1+2+...+n)=\Theta(n(n-1))=\Theta(n^2).$$

Проте черговий доданок a_k можна обчислити за допомогою попереднього a_{k-1}: $a_k = a_{k-1}/k$. Для цього незалежно від номера k потрібно поділити та присвоїти, тобто виконати $\Theta(1)$ дій. Складність цього алгоритму має оцінку $\Theta(n)$, що є $o(n^2)$, причому за n порядку 1000 другий алгоритм працює приблизно ***в 1000 разів швидше!***

3. ОБМІННЕ СОРТУВАННЯ

Розглянемо алгоритм сортування, який вважається найпростішим. Нехай **A[1]**, **A[2]**, ..., **A[n]** — елементи масиву, які треба відсортувати за неспаданням. Порівняємо **A[1]** з **A[2]**: якщо **A[1]>A[2]**, поміняємо їх значення місцями. Потім порівняємо **A[2]** з **A[3]** і, якщо треба, поміняємо місцями їх значення. Тоді **A[3]** буде мати найбільше значення серед **A[1]**, **A[2]**, **A[3]**. Продовжимо ці порівняння та обміни до кінця — **A[n]** одержить найбільше значення. Наприклад, послідовність значень <3, 4, 2, 1> перетвориться на <3, 2, 1, 4>. Якщо значення елементів уподібнити розмірам бульбашок, то порівняння й обміни будуть схожі на те, як найбільша бульбашка спливає нагору, відтісняючи інші. Тому цей метод називається ***бульбашковим сортуванням***.

В аналогічний спосіб перемістимо друге за величиною значення в елемент **A[n-1]**, перетворивши, наприклад, <3, 2, 1, 4> у <2, 1, 3, 4>. Потім третє за величиною значення перемістимо в **A[n-2]** тощо. На останньому кроці порівняємо тільки **A[1]** з **A[2]** і, якщо треба, поміняємо місцями їх значення.

Уточнимо бульбашкове сортування процедурою **bubbleSort** (*bubble* — бульбашка).

```
procedure bubbleSort(var A:aInt; n:word);
  var i, k : word;
begin
  for k:=n downto 2 do
    for i:=1 to k-1 do
      if A[i]>A[i+1] then
        swap(A[i], A[i+1])
end;
```

Очевидно, що найбільша кількість елементарних дій прямо пропорційна загальній кількості порівнянь, яких у найгіршому випадку $(n-1) + (n-2) + ... + 1 = n(n-1)/2$. Звідси складність сортування n-елементного масиву описаним способом має оцінку $\Theta(n^2)$.

• У задачах до цього розділу запропоновано реалізувати кілька ідей, які для багатьох конкретних масивів відчутно змен-

шують кількість дій. Проте ці ідеї не дають жодного ефекту, якщо у початковому масиві значення «відсортовано навпаки», тобто розташовано за спаданням, наприклад, <4, 3, 2, 1>.

4. СОРТУВАННЯ ВИБОРОМ

Переглянемо елементи масиву від першого до останнього, знайдемо елемент із найбільшим значенням і поміняємо місцями це значення зі значенням **A[n]**. Потім виберемо найменше значення серед **A[1]**, ..., **A[n-1]** і поміняємо його з **A[n-1]** тощо. Оскільки після знаходження максимального елемента нам необхідно міняти його місцями з останнім елементом поточної частини масиву, доцільніше шукати не максимальне значення, а його індекс.

```
procedure selectSort(var A:aInt; n:word);
  var iMax:word;
  {індекс максимального значення}
      i, k : word;
begin
  for k:=n downto 2 do
  begin
    iMax:=1;
    for i:=2 to k do
      if A[i]>A[iMax] then iMax:=i;
    if iMax<>k
      then swap(A[k], A[iMax])
  end
end;
```

Очевидно, що складність цього алгоритму також має оцінку $\Theta\left(n^2\right)$. У порівнянні з бульбашковим сортуванням він потребує меншої кількості обмінів — не більше $n - 1$.

- Наведений алгоритм має один непомітний недолік, який може бути важливим у деяких реальних задачах. Однакові значення після сортування змінюють свій взаємний порядок, тому цей алгоритм називають ***нестійким***. На відміну від нього, бульбашкове сортування упорядковує однакові значення, зберігаючи їхнє взаємне розташування, тому є ***стійким***.

5. СОРТУВАННЯ ВСТАВКАМИ

Алгоритми сортування вставками відрізняються від наведених вище алгоритмів тим, що шукаються не «елементи для місць», а «місця для елементів». З сімейства цих алгоритмів розглянемо ***алгоритм прямих вставок***.

У масиві виділяємо дві частини: відсортовану та невідсортовану. Спочатку відсортована частина містить тільки перший елемент (сам по собі впорядкований). Далі щоразу беремо перше значення з невідсортованої частини та «перетягуємо» його до відсортованої так, щоб вона не втратила впорядкованості. Для цього пересуваємо поточне значення ліворуч доти, доки воно не займе своє правильне місце у відсортованій частині масиву, тобто коли значення ліворуч і праворуч поточного стоятимуть «правильно» відносно нього. Отже, елемент «вставляється» у відсортовану частину масиву (звідки й назва методу).

```
procedure insertSort(var A:aInt; n:word);
  var i, k : word;
begin
  for k:=2 to n do begin
    i:=k;
    while (A[i]<A[i-1])and(i>1) do begin
       swap(A[i], A[i-1]); dec(i);
    end;
  end;
end;
```

Оскільки зсув елемента на одну позицію ліворуч «коштує» три команди присвоювання, рекомендуємо запам'ятати еталонний елемент (той, для якого шукається місце у відсортованій частині) у додатковій змінній. Тоді зсув можна виконати одним присвоюванням, а запам'ятований елемент потім одним присвоюванням поставити на звільнене місце.

```
procedure insertSort(var A:aInt; n:word);
  var i, k : word; etalon : integer;
begin
  for k:=2 to n do begin
    etalon:=A[k]; i:=k;
    while (i>1) and (A[i-1]>etalon) do begin
       A[i]:=A[i-1]; dec(i);
```

```
      end;
      A[i]:=etalon;
    end;
end;
```

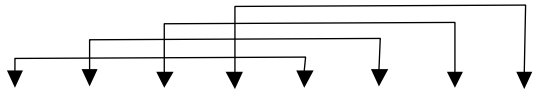
Складність цього методу також має оцінку $\Theta(n^2)$. У ньому, як і в сортуванні «бульбашкою», виконується багато обмінів сусідніх елементів.

У 1959 році Д.Шелл запропонував удосконалення алгоритму прямих вставок. Його ідея — порівнювати елементи, що знаходяться на певній відстані один від одного і покроково зменшувати цю відстань.

Розглянемо приклад. Нехай масив має такий вигляд:

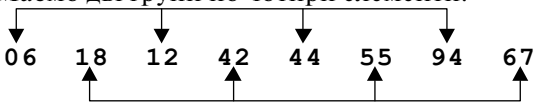
```
44  55  12  42  94  18  06  67
```

Спочатку окремо згрупуємо й упорядкуємо за допомогою прямих вставок елементи, які знаходяться на відстані 4 (четверне впорядкування). У цьому прикладі вісім елементів, тому групи містять по два елементи.



```
44  18  06  42  94  55  12  67
```

Тепер згрупуємо елементи на відстані 2 (подвійне впорядкування). Маємо дві групи по чотири елементи.



```
06  18  12  42  44  55  94  67
```

Нарешті, виконаємо звичайне (одинарне) впорядкування, порівнюючи сусідні елементи.

```
06  12  18  42  44  55  67  94
```

З описаного зрозуміло, що початкова відстань $h = n/2$. Після кожного проходу вона зменшується вдвічі.

```
procedure ShellSort(var A:aInt; n:word);
  var i, j, h : word;
    temp : integer;
begin
  h := n div 2;
  while h>0 do begin
    for i := h to n-h+1 do
```

```
    begin
       j:=i; temp:=A[i];
       while (j>=h)and(temp<A[j-h]) do begin
         A[j]:=A[j-h];
         dec(j,h);
       end;
       A[j] := temp;
     end;
     h := h div 2;
   end;
end;
```

Отже, ідея методу Шелла: змінити масив так, щоб кожна група елементів на відстані h була впорядкованою. Це дозволяє за деякого ряду відстаней порівняння h, остання з яких дорівнює одиниці, отримати упорядкований масив. Але яку послідовність кроків слід обрати?

Було винайдено багато рядів відстаней h, які добре зарекомендували себе, але найкращої серед них немає. Доведено лише, що кращі результати дають відстані, які не є дільниками одна одної. В основному використовують спадні геометричні прогресії, оскільки в цій ситуації кількість кроків близька до $\log n$. Деякі з рядів відстаней наведено у задачах цієї глави.

6. ШВИДКІ АЛГОРИТМИ СОРТУВАННЯ

По суті, єдиною перевагою наведених вище алгоритмів є простота й швидкість програмування та наладки. Проте майже всі вони мають оцінку складності $\Theta(n^2)$ і за великих n працюють надто повільно.

Нижче подано алгоритми, які мають оцінку складності $\Theta(n \log n)$ і за великих n працюють значно швидше.[1] Проте їх важче зрозуміти й налагодити, деяким з них потрібна додаткова пам'ять великого розміру.

Розглянемо такі методи швидкого сортування:

- сортування «злиттям»;
- швидке сортування;
- пірамідальне сортування.

[1] Існують навіть алгоритми, які за деяких обмежень на множину можливих значень у масиві мають оцінку складності $\Theta(n)$, але їх вивчення виходить за межі цієї книжки.

7. СОРТУВАННЯ «ЗЛИТТЯМ»

В основі алгоритмів сортування «злиттям» лежить об'єднання двох упорядкованих послідовностей в одну. Це схоже на перебудову двох колон учнів, вишикуваних за зростом, в одну. Учні, перші у своїх колонах, порівнюються; вищий стає у нову колону, інший залишається першим у своїй. Після цього в колоні, з якої пішов учень, наступний за зростом учень стає першим. Знову порівнюються перші, і так вони діють, доки в одній з колон нікого не залишиться — тоді решта іншої колони перейде у хвіст нової без зміни порядку.

Нехай у масиві **A** з елемента **A[L]** починається упорядкована ділянка довжиною **m-L+1**, а з елемента **A[m+1]** — ділянка довжиною **R-m**. Під упорядкованою ділянкою (відрізком або серією) розуміємо ділянку, яка не є частиною іншої упорядкованої ділянки. Наприклад, довжина таких упорядкованих ділянок дорівнює **m-L+1** = 3 і **R-m** = 3.

1	3	13	2	5	19
L		m	m+1		R

Вони об'єднуються в таку ділянку довжиною **R-L+1** у допоміжному масиві **B**.

1	2	3	5	13	19
L		m	m+1		R

Розглянемо процедуру злиття в масив **Z** пари суміжних ділянок масиву **X**, в якому ліва містить індекси від **L** до **m**, а права — від **m+1** до **R**.

```
procedure merge(var X,Z:aInt; L,m,R:word);
  var k:word; {індекс у цільовому масиві}
    i,j:word; {індекси у половинах}
begin
  i:=L; j:=m+1;
  for k:=L to R do
  {заповнення елементів Z[L], ..., Z[R]}
    if i>m then
    begin Z[k]:=X[j]; inc(j) end
    else if j>r then
         begin Z[k]:=X[i]; inc(i) end
```

```
            else if X[i]<X[j] then
              begin Z[k]:=X[i]; inc(i) end
              else begin Z[k]:=X[j]; inc(j) end
  end;
```

Очевидно, тіло циклу виконується $R - L + 1$ разів, і щоразу виконується $\Theta(1)$ елементарних дій. Отже, складність виконання виклику процедури **merge** є $\Theta(R-L+1)$.

Алгоритм сортування злиттям полягає в повторенні таких кроків злиття пар. У масиві відбувається пошук пари суміжних упорядкованих ділянок, які об'єднуються в одну ділянку допоміжного масиву, наприклад, за допомогою процедури **merge**. Потім відбувається пошук та об'єднується наступна пара тощо. Можливо, наприкінці залишиться ділянка, яка не має пари, — її буде скопійовано без змін. На наступному кроці відбувається подібне злиття пар ділянок допоміжного масиву в основний. Кроки повторюються, поки якийсь із масивів не перетвориться на одну упорядковану ділянку. Якщо це допоміжний масив, він копіюється в основний.

Продемонструємо виконання алгоритму на масиві A = <11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1> довжиною n = 11.

Упорядковані послідовності виділено дужками < >, пари ділянок, які об'єднуються, відокремлені «;», B — ім'я допоміжного масиву.

A = <<11><10>; <9><8>; <7><6>; <5><4>; <3><2>; <1>>
B = <<10, 11><8, 9>; <6, 7><4, 5>; <2, 3><1>>
A = <<8, 9, 10, 11><4, 5, 6, 7>;<1, 2, 3>>
B = <<4, 5, 6, 7, 8, 9, 10, 11><1, 2, 3>>
A = <1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11>

Як бачимо, масив відсортовано за чотири кроки злиття.

Після першого кроку злиття довжина упорядкованих ділянок не менше 2 (за винятком, можливо, «хвоста» довжиною 1), після другого — не менше 4 (крім, можливо, «хвоста» меншої довжини) тощо. Отже, після i-го кроку довжина упорядкованих ділянок, крім, можливо, «хвоста», не менше 2^i. Нехай n — кількість елементів масиву. На останньому, k-му, кроці об'єднуються дві ділянки; перша з них має довжину не менше 2^{k-1}, причому $2^{k-1} < n$. Отже, кількість кроків $k < \log n + 1$. За рахунок можливого додаткового копіювання кількість кроків треба збільшити на 1,

але оцінка $\Theta(\log n)$ збережеться. На кожному кроці загальна кількість елементарних дій є $\Theta(n)$, тому складність алгоритму — $\Theta(n \cdot \log n)$.

Реалізуємо алгоритм сортування злиттям за допомогою процедури **sortByMrg**. Крок злиття ділянок одного масиву в інший оформимо функцією **mergeStep**. Вона повертає ознаку того, що на кроці злиття було знайдено хоча б одну пару упорядкованих ділянок. Якщо пару не знайдено, то масив, початковий у виклику функції, відсортовано. На непарних кроках злиття функція **mergeStep** виконує злиття ділянок основного масиву **A** у допоміжний масив **B**, на парних — навпаки. Якщо масив, початковий у виклику **mergeStep**, виявився відсортованим на парному кроці злиття, значить, це масив **B**, і його треба скопіювати в основний масив **A**. А якщо на непарному кроці, то це масив **A**, і більше нічого робити не треба.

Пару суміжних упорядкованих ділянок n-елементного масиву (перший із них починається індексом **left**) шукає функція **findPair**. Вона повертає ознаку того, що пару знайдено. Праві межі ділянок зберігаються в її параметрах **mid** і **right**. Якщо після її виклику справджується умова **(left=1)and(right=n)**, то пару не знайдено, тобто масив відсортовано.

Для злиття використовуємо процедуру **merge** (див. вище). Допоміжна процедура копіювання ділянки масиву в інший масив **copyAr** є очевидною.

```
procedure copyAr(var X,Y:aInt;
                 left,right:word));
  var i:word;
begin
  for i:= left to right do Y[i]:= X[i]
end;
function findPair(var X:aInt; n,left:word;
         var mid, right : word): boolean;
{функція повертає ознаку того, що пару
 суміжних
  упорядкованих ділянок знайдено}
begin
  findPair := false;
  if left <= n then
```

```
    begin
      mid := left;
      while (mid<n)and(X[mid]<=X[mid+1]) do
         inc(mid);
      {(mid=n) or (X[mid]>X[mid+1])}
      if mid=n    {права межа - кінець масиву}
      then right:=mid
      else
      begin {пошук правої межі}
      findPair := true;
      right := mid+1;
      while(right<n)and(X[right]<=X [right+1])
             do inc(right);
      {(right=n) or (X[right]>X[right+1]}
      end;
    end;
  end;
  function mergeStep(var X,Y:aInt; n:word):
boolean;
  {функція повертає ознаку того, що злиття
    масиву X у масив Y відбулося}
    var left,mid,right:word;
  begin
    mergeStep:=true;
    left:=1;
    while findPair(X,n,left,mid,right) do
      begin merge(X,Y,left,mid,right);
      left:=right+1
    end;
    {останній виклик findPair повернув false}
    if (left=1)and(right=n)
    then mergeStep:=false {масив X відсортовано}
    else if left<=n
         then copyAr(X,Y,left,n);
         {копіювання «хвоста»}
  end;
  procedure sortByMrg(var A:aInt; n:word);
    var B:aInt;    {допоміжний масив}
```

```
    step:word;   {номер кроку}
    notSorted:boolean;{ознака невідсортованості}
begin
  step:=0;
  notSorted:=true;
  while notSorted do begin
     inc(step);
     if odd(step)
     then notSorted:=mergeStep(A,B,n)
     else notSorted:=mergeStep(B,A,n)
  end;
  {після злиття один з масивів відсортовано}
  if not odd(step) then copyAr(B,A,1,n)
end;
```

- Алгоритм сортування злиттям сортує масиви з великою довжиною значно швидше, ніж базові методи. Проте його головний недолік — йому потрібен ***додатковий масив розміром n***.
- Алгоритм сортування злиттям зберігає взаємне розташування однакових значень, тобто ***є стійким***.
- В алгоритмах, заснованих на злитті, елементи послідовності обробляються у порядку їх розташування, тобто, по суті, без прямого доступу. Тому саме ці алгоритми використовуються для зовнішнього сортування.

8. ШВИДКЕ СОРТУВАННЯ

Перший алгоритм швидкого сортування 1960 року розробив Ч.А.Р. Хоар (C.A.R.Hoare). Цей алгоритм є одним із найпопулярніших, оскільки порівняно нескладний у реалізації, добре працює на різноманітних видах вхідних даних і не вимагає величезних обсягів додаткової пам'яті (окрім відносно невеликого стека).

Ідея швидкого сортування така. У деякий спосіб вибирається ***еталонне значення*** **e**. Значення елементів масиву *A* обмінюються так, що масив розбивається на дві ділянки — ліву та праву. Елементи в лівій ділянці мають значення не більше *e*, а у правій — не менше. Після цього достатньо окремо відсортувати ці дві ділянки.

Існує простий, але досить ефективний спосіб вибору *e* в ділянці масиву **A[L]**, ..., **A[R]**: **e = A[(L+R) div 2]**. Для розбиття використовуються два індекси — «лівий курсор» **i** та «правий курсор» **j**. Спочатку **i**=**L**, **j**=**R**; далі вони рухаються назустріч один одному в такий спосіб. Значення менше *e* («хороші») в лівій частині пропускаються, а на «поганому» рух зупиняється. Аналогічно після цього у правій частині пропускаються значення, що більші *e*. Якщо **i** ще не став більше **j**, то це означає, що обидва вони вказують на «погані» значення. Ці значення обмінюються місцями, а курсори зсуваються назустріч один одному. Рух курсорів продовжується, поки **i** не стане більше **j**. Тоді всі елементи від **A[L]** до **A[j]** мають значення не більше *e*, а всі від **A[i]** до **A[R]** — не менше. Ці дві ділянки, якщо їх довжина більше 1, поділяються й сортуються рекурсивно. Якщо ж ділянка має довжину 1, то її вже відсортовано.

Оформимо сортування частини масиву **A[L]**, ..., **A[R]** такою процедурою:

```
procedure quickSort(var A:aInt; L,R:word);
  var e:integer; {еталонне значення}
    i,j:word; {лівий та правий курсори}
begin
  i:=L; j:=R;
  e:=A[(i+j) div 2];
  while i<=j do
  begin
    while A[i]<e do inc(i);
    while A[j]>e do dec(j);
    if i<=j then begin
      swap(A[i],A[j]);
      inc(i); dec(j);
    end;
  end;
  {i > j};
  if L<j {сортувати ліву ділянку}
  then quickSort(A,L,j);
  if i<R {сортувати праву ділянку}
  then quickSort(A,i,R);
end;
```

Оцінимо складність наведеної реалізації швидкого сортування. Нехай масив A має n елементів. У найгіршому випадку в кожному виклику процедури **quickSort** еталонним значенням є найменше в ділянці від **A[L]** до **A[R]**, і розбиття ділянки масиву довжиною m дає ділянки довжиною 1 і $m-1$. Оскільки $m = n$, $n-1$, ..., 2, глибина рекурсії досягає $\Theta(n)$, і при кожному значенні m розбиття ділянки довжиною m має складність $\Theta(m)$. Тоді сумарна складність має оцінку $\Theta(n)+\Theta(n-1)+...+\Theta(2)=\Theta(n^2)$.

Проте описаний найгірший випадок у реальних даних практично ніколи не трапляється. Для переважної більшості даних розбиття ділянки масиву дає дві ділянки з приблизно рівними довжинами. Тому розміри масивів, які сортуються, при переході на наступний рівень рекурсії зменшуються приблизно вдвічі. Отже, ***в середньому*** кількість рівнів рекурсії оцінюється як $\Theta(\log n)$. Очевидно, що на кожному рівні рекурсії сумарна складність є $O(n)$, тому загальна складність ***в середньому*** має оцінку $O(n \log n)$.

- Численні практичні дослідження свідчать, що наведений алгоритм в середньому працює швидше, ніж інші алгоритми сортування, які мають складність найгіршого випадку $O(n \log n)$.
- Швидке сортування не зберігає взаємного порядку однакових значень, тобто є ***нестійким***.

Для вибору еталонного значення існує багато способів, а не тільки вибір $A[(L + R)/2]$. Якщо еталонним є одне зі значень у сортованій частині масиву, адже це дозволяє не турбуватися про запобігання виходу за межі сортованої частину й не перевіряти додаткових умов.

Вибір «золотої середини» (значення, яке має бути посередині, або ***медіана***) вимагає чималих додаткових зусиль і може взагалі погіршити оцінку складності. Проте досить ефективним є вибір середнього з трьох значень — першого (з індексом L), останнього (R) та серединного ($(L + R)/2$). Для цього перед вибором еталону їх можна обміняти місцями у такий спосіб.

```
if A[L] > A[(L+R) div 2]
then swap(A[L],A[(L+R) div 2]);
if A[(L+R) div 2] > A[R]
then swap (A[(L+R) div 2],A[R]);
if A[L] > A[(L+R) div 2]
then swap(A[L],A[(L+R) div 2]);
e := A[(L+R) div 2];
```

Ще два способи вибору еталонного значення наведено у задачах до цієї глави.

Ітеративна версія алгоритму. Запишемо ітеративну процедуру, в якій явно реалізуємо обробку стека, необхідну для виконання рекурсивної процедури.

Змоделюємо програмний стек за допомогою масиву **Stack**, в якому будемо зберігати ліві та праві межі сортованих підмасивів. Спочатку присвоїмо елементам масиву **Stack** значення 0 за допомогою стандартної підпрограми **fillchar**.

```
procedure QuickSort(var A:aInt; n:word);
  var i, j, L, R, top : word;
    e : integer;
    Stack : array[1..N_max,1..2] of word;
Begin
  FillChar(Stack,SizeOf(Stack),0);
  top:=1; {індекс верхівки стека}
  Stack[top,1] := 1; Stack[top,2] := N;
  while top>0 do
  begin
    L:=Stack[top,1]; R:=Stack[top,2];
    dec(top);
    e:=A[(L+R) div 2];
    i:=L; j:=R;
    repeat
      while (A[i]<e) do inc(i);
      while (A[j]>e) do dec(j);
      if i<=j then begin
        Swap(A[i],A[j]);
        inc(i); dec(j);
      end;
    until i>=j;
    if L<j then begin
       inc(top);
       Stack[top,1]:=L; Stack[top,2]:=j;
    end;
    if i<R then begin
       inc(top);
       Stack[top,1]:=i; Stack[top,2]:=R;
```

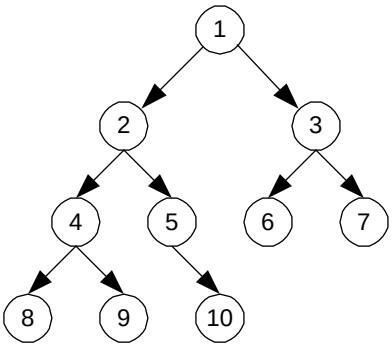
```
      end;
    End;
End;
```

9. ПІРАМІДАЛЬНЕ СОРТУВАННЯ

Описане тут сортування в середньому повільніше, ніж швидке, хоча й має складність у найгіршому випадку $\Theta(n \log n)$. Проте цей алгоритм застосовується для сортування файлів і розв'язання деяких інших задач.

Пірамідальне сортування (або ***сортування за допомогою купи***) використовує ***бінарне дерево*** (***піраміду*** або ***купу***). Розглянемо це поняття.

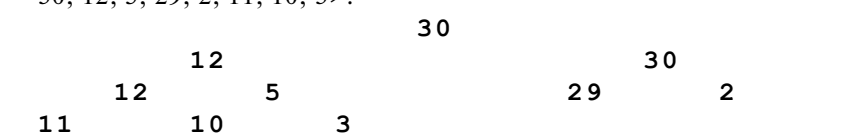
Розташуємо елементи масиву з індексами 1..*n* рядками, подвоюючи кількість елементів у рядках: у першому рядку — перший елемент (з індексом 1), у другому — з індексами 2 і 3, у третьому — з індексами 4—7, далі — 8—15 тощо. Останній рядок може залишитися неповним. Наприклад, при $n = 10$ буде утворено піраміду індексів, як на рисунку.



Розглянемо піраміду як ***бінарне дерево*** з коренем нагорі та листям унизу. Дерево утворено ***вузлами***, що відповідають індексам, і зв'язками між ними — ***дугами***. ***Коренем*** дерева є вузол 1. Вузли 2 і 3 назвемо ***синами*** вузла 1 (їх ***батька***), вузли 4 і 5 — синами 2 тощо. Вузли, які не мають синів, називаються ***листками*** (на рисунку це вузли 6—10. Отже, якщо вузол *i* має синів, то ними є вузли з індексами $2i$ та $2i + 1$, а його батьком є вузол з індексом i div 2.

Далі будемо розглядати вузли дерева, які містять значення елементів масиву з відповідними індексами. Для сортування важливим є упорядкування значень у масиві, за якого при кожному можливому *i* справджується нерівність $A[i \text{ div } 2] \geq A[i]$, тобто «син не більше батька». Ця властивість не стосується лише кореня дерева, оскільки він не має батька. Дерево з цією властивістю нази-

вають ***правильним***, або ***правильною купою***. Розглянемо приклад правильного дерева, яке відповідає послідовності значень <30, 12, 30, 12, 5, 29, 2, 11, 10, 3>.



Сортування за допомогою купи використовує те, що корінь правильної купи містить її максимальне значення. Спочатку в масиві утворимо правильну купу, далі поміняємо місцями значення першого елемента масиву (кореня дерева) та останнього. Найбільше значення займе «своє» останнє місце в масиві, і далі про нього можна забути. Серед решти елементів масиву відновимо основну властивість купи та обміняємо місцями значення першого елемента та тепер уже передостаннього, і так діятимемо далі, доки дерево не скоротиться до одного елемента.

```
procedure heapSort(var A:aInt; n:word);
  var j:word; {индекс останнього}
begin
  build(A,n); {початкова побудова купи}
  for j:=n downto 2 do begin
    swap(A[1],A[j]);
    rebuild(A,1,j-1) {відновлення правильності}
  end
end;
```

Спочатку уточнимо процедуру **rebuild** відновлення правильності купи. Якщо значення в кореневому вузлі змінилося, основна властивість у ньому може не виконуватися, тому за потреби більший з синів обмінюється з батьком, після чого основна властивість перевіряється й відновлюється для сина. Дії з відновлення правильності купи в частині масиву $A[f], ..., A[d]$ неважко уточнити рекурсивною процедурою.

```
procedure rebuild(var A:aInt; f,d:word);
var maxSon:word; {індекс максимального сина}
begin
  maxSon:=f;
  if (2*f<=d)and(A[f]<A[2*f])
  then maxSon := 2*f;
  if (2*f+1<=d)and(A[maxSon]<A[2*f+1])
```

```
    then maxSon:=2*f+1;
    if maxSon<>f then begin
       swap(A[f],A[maxSon]);
       rebuild(A,maxSon,d);
    end;
  end;
```

Наведена процедура є прикладом так званої «хвостової рекурсії», коли після рекурсивного виклику немає жодних дій. У подібних ситуаціях рекурсію можна замінити циклом, умова завершення якого відповідає умові «дна рекурсії». Очевидно також, що перебудову дерева можна закінчити, якщо значення максимального сина та батька місцями не обмінювалися. У цій ситуації, щоб припинити виконання циклу, присвоїмо індексу «батька» **f** значення за межами частини масиву, що перебудовується.

Отже, наведемо нерекурсивний варіант процедури **rebuild**.

```
procedure rebuild(var A:aInt; f,d:word);
var maxSon:word; {індекс максимального сина}
begin
  while (2*f<=d)do
  begin
    maxSon:=f;
    if (A[f]<A[2*f])
    then maxSon:=2*f;
    if (2*f+1<=d)and(A[maxSon]< A[2*f+1])
    then maxSon:=2*f+1;
    if maxSon<>f then begin
       swap(A[f],A[maxSon]);
      f:=maxSon {далі перебудова піраміди сина}
    end
    else f:=d+1;
  end;
  {2*f > d або A[f] >= A[maxSon]}
end;
```

Нарешті уточнимо початкову побудову купи за процедурою **build**. Помітимо, що в масиві з n елементами максимальним індексом вузла-батька є n div 2. Отже, послідовно утворимо правильні купи у піддеревах, коренями яких є вузли n div 2, n div 2 – 1, ..., 1. Це дозволить стверджувати, що побудований масив являє собою правильну купу.

```
procedure build(var A:aInt; n:word);
  var i:word;
begin
  for i:=n div 2 downto 1 do
    rebuild(A,i,n); {перебудова частини масиву}
end;
```

Оцінимо складність алгоритму. Очевидно, що вона прямо пропорційна загальній кількості викликів **rebuild**. При виконанні **build** процедура **rebuild** викликається n div 2 рази, а при виконанні циклу **for** процедури **treeSort** — ще $n - 2$ рази, тобто загальна кількість викликів процедури **rebuild** з інших процедур є $\Theta(n)$.

Оцінимо складність виконання одного виклику процедури **rebuild**. Помітимо, що в циклі значення змінної **f** не менш ніж подвоюється, а цикл не буде продовжуватися, якщо це значення стане більше d, яке не більше n. Отже, таких подвоювань не може бути більше ніж $\lfloor \log_2 n \rfloor$. Кількість дій у тілі циклу є константою, тому загальна складність має оцінку $O(n \log n)$.

- ***Висотою вузла*** дерева називають кількість ребер у найдовшому шляху з цієї вершини вниз до листків; висоту кореня називають ***висотою дерева***. Дерево, яке утворено як піраміда з n вузлів, має висоту $\lfloor \log_2 n \rfloor$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Для чого потрібно сортувати послідовності даних?

2. Дайте словесний опис алгоритмів сортування.

3. Назвіть головний недолік базових алгоритмів сортування.

4. Назвіть переваги й недоліки алгоритмів сортування, наведених у цьому розділі.

5. Дайте оцінки обсягів додаткової пам'яті, необхідної для сортування масиву з n елементами за різними алгоритмами сортування.

ЗАДАЧІ

1. Якщо в алгоритмі бульбашкового сортування на проході масиву не відбулося жодного обміну, то масив уже відсортовано, і подальші проходи не потрібні. Модифікуйте алгоритм, щоб сортування закінчувалося, якщо на деякому проході не було обмінів.

2. В алгоритмі бульбашкового сортування на проході масиву після останнього обміну значень решта масиву є відсортованою. Реалізуйте покращення алгоритму: запам'ятати індекс останнього елемента, що підлягав перестановці на проході, та на наступному проході використати цей індекс як праву межу невідсортованої частини масиву.

3. Алгоритм бульбашкового сортування вдосконалюється у так званому методі «шейкера», який дає відчутний виграш, коли багато значень у вхідному масиві стоять далеко від своїх правильних позицій. Наприклад, при сортуванні за зростанням набору **100 60 25 5 10 5 2 3** очевидно, що великі числа (**100**, **60**) займуть свої місця за два проходи («спливуть» наче бульбашки). Проте маленькі (**2**, **3**) будуть просуватися на свої місця повільно, оскільки прохід масивом здійснюється зліва направо. Щоб прискорити просування маленьких чисел до початку масиву, реалізуйте на кожному кроці алгоритму два проходи — зліва направо та справа наліво.

4. Сортування вибором допускає модифікацію, аналогічну методу «шейкера» (див. задачу 3): на проході шукати як максимальне, так і мінімальне значення, а потім «розкидати» їх на свої місця в кінцях масиву.

5. Алгоритм сортування ***простими вставками*** дозволяє створити відсортований масив із значень, що поступово надходять, наприклад, від клавіатури. Перше значення присвоюється першому елементу масиву. Друге значення порівнюється з першим і, якщо воно менше, то «витісняє» перше на друге місце. Інакше нове значення йде на друге місце. Потім третє порівнюється з другим та записується або на третє місце, або витісняє значення з другого місця на третє та порівнюється з тим, що на першому місці. Наприклад, за читання послідовності значень 3, 1, 2 створюються послідовності значень у масиві <3>, <1, 3>, <1, 2, 3>.

Узагалі, після читання $k - 1$ елемента є відсортована частина масиву $A[1]A[2]...A[k - 1]$. Нове значення v порівнюємо зі значенням $A[k - 1]$. Якщо $A[k - 1] > v$, то $A[k - 1]$ зсуваємо на k-е місце. Після цього порівнюємо v з $A[k - 1]$: якщо $A[k - 2] > v$, то $A[k - 2]$ зсуваємо на $(k - 1)$-е місце тощо. Коли за чергового порівняння $A[i] \le v$, то v записується на $(i + 1)$-е місце. Якщо всі

значення в масиві більше v, то вони зсуваються, а v записується на перше місце. Реалізувати наведений алгоритм.

6. Для модифікації методу Шелла Д. Кнут запропонував таку послідовність кроків:

1 4 13 40 121 364 1093 3280 9841...

Починаючи з 1, наступне число утворюється множенням попереднього на 3 та додаванням 1. Першим кроком є найбільше з наведених чисел, менше за n, другим — попереднє з цього ряду тощо до кроку 1. Ці кроки забезпечують сортування, яке має оцінку складності $O(N \cdot \sqrt{N})$.

Якщо використати ряд кроків

1 8 23 77 281 1073 4193 16577...,

утворений за законом $4^i + 3 \cdot 2^{i-1} + 1$ при $i > 0$, то оцінка складності в середньому буде $O\left(N \cdot \sqrt[3]{N}\right)$.

Ще кращою в середньому є послідовність

1 5 19 41 109 209 505 929 2161 3905...,

отримана чергуванням послідовностей $4^{i+1} - 3 \cdot 2^{i+1} + 1$ та $9 \cdot (4^i - 2^i) + 1$ при $i \geq 1$.

Реалізуйте метод Шелла з указаними рядами кроків.

7. У методі швидкого сортування значення, рівні еталонному, можна зібрати так, щоб ліворуч від них були лише значення, менші за еталонне, а праворуч — більші. Визначивши межі «меншої» та «більшої» груп, можна далі сортувати лише їх, не обробляючи більше еталонні значення, які вже зайняли свої місця. За великої кількості повторень еталонного значення таке звуження меж сортування дає відчутний виграш.

8. Оригінальний метод реалізації цієї ідеї запропонували в 1993 році Бентлі (*Bentley*) та Мак-Ілрой (*McIlroy*). Еталонні значення накопичуються в кінцях підмасиву (сортованої частини масиву). Коли курсори перетнуться, стануть відомими права межа «меншої» групи та ліва межа «більшої». Тоді еталонні значення з початку підмасиву шляхом обмінів переміщаються в кінець «меншої» групи, а з кінця підмасиву — у початок «більшої». Ці пересування незначно уповільнюють розбиття, але коли значення елементів масиву часто повторюються, сортування в цілому прискорюється.

9. Швидке сортування можна прискорити, якщо короткі частини масиву сортувати за допомогою нерекурсивних викликів, а іншо-

го способу, більш ефективного саме для коротких підмасивів. Одним з таких методів є метод вставки. Модифікуйте алгоритм швидкого сортування так, щоб підмасиви, довжина яких менше деякої ***порогової*** довжини, сортувалися за методом вставки. Завдяки експериментам відомо, що оптимальна порогова довжина підмасивів лежить у інтервалі від 5 до 20, залежно від кількості елементів у масиві. Варіант модифікації полягає в тому, що підмасиви з довжиною, меншою за порогову, просто ігноруються, а після завершення цього неповного сортування весь масив даних впорядковується найкращим у цій ситуації методом — прямими вставками.

10. Прочитати n цілих чисел у масив та виконати такі дії:

a) одержати всі числа, які входять до масиву по одному разу;

b) з'ясувати, чи є в масиві хоча б два однакових значення;

c) одержати всі числа, які входять до масиву більше ніж по одному разу;

d) підрахувати кількість різних значень у масиві;

e) одержати масив, який містить усі числа з початкового масиву, але тільки по одному разу;

f) з'ясувати, яке число зустрічається в масиві найбільшу кількість разів.

Програма повинна мати складність $O(n \log n)$.

11. За n точками на дійсній прямій визначити всі пари точок, відстань між якими найменша. Програма повинна мати складність $O(n \log n)$.

12. Задано послідовність із n цілих чисел. Визначити номер її k-го за величиною елемента (можливо, різних значень у послідовності менше ніж k).

13. Відрізки на прямій задано парами кінців $[a_i, b_i]$, $i=1, ..., n$. Перевірити, чи є відрізком об'єднання відрізків і, якщо це так, обчислити його кінці. Програма повинна мати складність $O(n \log n)$.

14. Написати процедуру обчислення N ($N \le 1000$) найменших значень послідовності дійсних чисел, яка вводиться з файла й має необмежену довжину, за умови:

a) числа враховуються по одному разу незалежно від кількості їх повторень;

b) число враховується стільки разів, скільки зустрічається у файлі (але не більше ніж N);

с) кожне число враховується один раз, але ведеться додатковий облік повторень.

15. Дано послідовність *n* натуральних чисел. Знайти мінімальне натуральне число, яке не дорівнює жодній з сум чисел цієї послідовності (числа включаються до суми без повторень). Наприклад, за послідовності 1, 3 цим числом є 2.

16. Дано дві послідовності натуральних чисел $a[1], a[2], ..., a[n]$ та $b[1], b[2], ..., b[n]$. Визначити перестановку $i_1, i_2, ..., i_n$ чисел 1, 2, ..., *n*, при якій сума добутків

$$a[1]b[i_1] + a[2]b[i_2] + ... + a[n]b[i_n]$$

мінімальна.

17. Задано *n* натуральних чисел, які задають точки на прямій. Задано довжину відрізка. Визначити найменшу можливу координату початку відрізка, за якої відрізок накриває щонайбільше точок. Наприклад, за точок 14, 24, 6, 18, 12, 20 та довжини відрізка 10 найменшою координатою початку є 10.

ðîçä³ë 3

ÐßÄÊÈ ÒÀ Ì Í ÎÆÈ Í È

1. РЯДКИ

Рядок у загальному значенні цього слова — це скінченна ***послідовність символів***. У мові *Turbo Pascal* значення-рядки записуються за допомогою апострофів, наприклад, `'ABC'` або `'12345'`. Порожній рядок, тобто послідовність символів довжиною 0, позначається `''`.

Для подання та обробки рядків мова *Turbo Pascal* надає спеціальний тип **`string[`***n***`]`**, де *n* — ціла константа (можливо, іменована) не більше **`255`**. Значення типу **`string[`***n***`]`** — це послідовності символів довжиною від **`0`** до до *n*.

Змінна типу **`string[`***n***`]`** являє собою ***масив символів*** з індексами від **`0`** до *n*. Нехай **`s`** — змінна типу **`string[`***n***`]`**. Елемент **`s[0]`** відіграє особливу роль. Значенням *L* = **`ord(s[0])`** може бути число від 0 до *n* — ***довжина рядка***, що є значенням **`s`**. Сам рядок-значення представлено елементами з індексами від 1 до *L*. Наприклад, якщо змінна **`s`** має тип **`string[5]`**, то такі послідовності байтів

#3	'A'	'B'	'C'	?	?
#5	'1'	'2'	'3'	'4'	'5'
#0	?	?	?	?	?

представляють її значення відповідно `'ABC'`, `'12345'` та `''`.

- Елементи масиву-рядка з індексами від *L* + 1 до *n* ***невизначені***; спроба їх використання може призвести до неочікуваних

наслідків. Їх значення варто розглядати як «сміття», що позначено в наведених байтах знаком **?**.

- Довжина рядка зберігається в одному байті й не перевищує 255. Замість **string[255]** можна писати **string**.

Найпростішими виразами рядкового типу є ім'я змінної-рядка, рядкова константа (літерал) та вираз типу **char**. Для рядків означена двомісна операція ***конкатенації*** (***дописування***) **+**. Результат утворюється шляхом дописування правого операнда до лівого: значенням **'12'+'3'** є **'123'**.

- Ціла довжина рядка повертається з виклику функції **length** із аргументом-рядком: **length('123')** = 3.

Рядковий вираз можна ***присвоїти*** рядковій змінній. Символи присвоюються елементам змінної, починаючи з першого. Довжина значення виразу стає довжиною значення змінної. Якщо ця довжина більша за максимально можливу довжину *n* змінної, то елементам змінної присвоюються *n* перших символів.

Приклади. Нехай змінна **s** має тип **string[3]**. Після присвоювання **s:='12345'** її послідовні елементи мають значення **#3**, **'1'**, **'2'**, **'3'**; після присвоювання **s:='12'** вони мають значення **#2**, **'1'**, **'2'**, а елемент **s[3]** невизначений. Нехай за **s='12'** виконуються оператори **s:=s+'7';s:='9'+s**. Тоді **s** послідовно одержить значення **'127'** та **'912'**. Після **s:=''** маємо: **s[0]=#0**, а всі інші елементи невизначені.

Рядки, як і інші масиви, допускають використання та обробку їх окремих елементів, але ***тільки в межах значення***, поданого масивом. Наприклад, за **s** типу **string[3]** і **s='12'** можна присвоїти **s[1]** і **s[2]** символьні значення або використати їх, але **s[3]** є невизначеним. Можна також використовувати та змінювати **s[0]**. Наприклад, якщо за **s=''** виконати **s[0]:=chr(2)**, то довжиною значення **s** стане 2, тобто **s[1]** і **s[2]** будуть визначеними.

- Явне використання окремих елементів рядків, особливо елемента з індексом 0, ***не рекомендується***.

Для рядків означено операції порівняння **=**, **<>**, **<**, **<=**, **>**, **>=**. Рядки ***рівні***, якщо мають однакову довжину і в її межах відповідні символи збігаються; інакше рядки не рівні.

Рядки порівнюються відповідно до їх ***лексикографічного поряд-***

ку, тобто як слова в словнику, тільки символами алфавіту є **#0**, **#1**, ..., **#255**, упорядковані за номерами. Точніше, для не порожніх рядків *s1* < *s2*, якщо існує *i*, $1 \le i \le \min\{\text{length}(s1), \text{length}(s2)\}$, при якому $s1[i] < s2[i]$, а всі відповідні елементи з індексами менше *i* у рядках рівні. Крім того, *s1* < *s2*, якщо *s1* є початком *s2*. Наприклад, **'AAA'<'AZ'**, **'12'<'9'**, **'ZZZ'<'Za'**, **'12'<'123'**, а порожній рядок **''** є найменшим.

2. УВЕДЕННЯ ТА ВИВЕДЕННЯ РЯДКІВ

- Ім'я рядкової змінної можна вказати у виклику **read** або **readln**, а рядковий вираз — **write** або **writeln**.

Уведення. Якщо змінна **s** має рядковий тип, а **f** є файловою змінною типу **text**, то при виконанні **read(f,s)** символи до найближчого кінця рядка або до кінця файла чи його ознаки **#26** читаються та присвоюються елементам рядка **s**; доступним буде кінець рядка **#13** (або кінець тексту **#26**). Сам символ **#13** або **#26** у рядок-змінну не записується. Якщо символів тексту до кінця рядка або тексту більше, ніж уміщається в **s**, то **s** заповнюється до кінця, і доступним стає перший символ після прочитаних у **s**. Якщо перед читанням рядка **s** за допомогою процедури **read** доступний кінець рядка, то він ***залишається доступним***, а значенням **s** стає ***порожній рядок***.

- Застосовувати **read** при введенні рядків ***небезпечно***, особливо в циклі. Якщо після введення рядка доступним символом стане **#13**, то подальші спроби читання рядків залишатимуть його доступним, і виконання програми може «***зациклитися***».

Приклад. Нехай змінні **s1**, **s2**, **s3** мають тип **string[3]**, а текст **f** містить таку послідовність символів.

'1'	'2'	'3'	'4'	'5'	#13	#10	'a'	#26	'c'

Тоді після **read(f,s1,s2,s3)** змінні мають такі значення: **s1='123'**, **s2='45'**, **s3=''**; доступним залишається символ **#13**. Подальші спроби вводити з цього тексту в рядкові змінні будуть надавати їм значення **''** й залишати доступним символ **#13**. ▣

Уведення рядків з клавіатури є аналогічним. Щоб виконання виклику **read** було закінчено, треба натиснути клавішу **<Enter>**.

Наприклад, якщо в умовах наведеного прикладу на клавіатурі набрати символи **12345** і натиснути клавішу **<Enter>**, то змінні отримають ті ж самі значення.

- Уведення рядків з клавіатури за допомогою процедури **read** ***не рекомендується***, оскільки має додаткові «підводні камені» (їх розгляд виходить за межі цієї книжки).

Виконання процедури **readln** аналогічно **read**, але після заповнення рядкової змінної або досягнення кінця рядка в тексті частина тексту разом із найближчим кінцем рядка пропускається й доступним стає перший символ наступного рядка. В умовах попереднього прикладу після виконання **readln(f,s1,s2,s3)** змінні матимуть такі самі значення, але доступним буде символ **'a'**. Якби виконувалися виклики **readln(f,s1); readln(f,s2); readln(f,s3)**, то **s1** мало б значення **'123'**, **s2** — **'a'**, **s3** — **''**, а доступним став би символ **#26**.

- Уведення рядків з текстових файлів та з клавіатури за процедурою **readln** ***надійніше***, ніж за процедурою **read**.

Виведення. Виведення рядкових виразів не має особливостей — символи значення виразу виводяться в текстовий файл або на екран. Наприклад, якщо рядкова змінна **s** має значення **'12'**, то при виконанні виклику **write(s+'*'+s)** виводяться символи **12*12**.

Приклад. Розглянемо програму, яка отримує рядки тексту від клавіатури та виводить у текстовий файл.

```
program CreateText;
  var f:text; {файл-текст}
    s:string; {рядок для введення з клавіатури}
begin
  writeln('Створення тексту.');
  assign(f,'myfile.txt'); rewrite(f);
  write('Рядок і <Enter>: ');
  while not eof do begin
    readln(s);
    writeln(f,s);
    writeln('Рядок і <Enter>: ');
  end;
```

```
    close(f)
  end.
```

Для закінчення роботи замість введення нового рядка треба натиснути **<Ctrl+Z>** і **<Enter>**.

- Внаслідок особливостей системи *Turbo Pascal* довжина рядків, що набираються на клавіатурі, не більше 127.

Приклад. Розглянемо «копіювання» тексту за таких умов: рядки тексту мають довжину не більше 255, а порожні рядки або ті, що містять лише пропуски, не копіюються.

Власне «копіювання» опишемо процедурою; вхідний і цільовий файли-тексти задамо як її параметри. За умовою, рядки тексту «уміщуються» в змінних типу **string**, тому для введення тексту використаємо стандартну процедуру **readln** та змінну-рядок. Обробляючи рядок, визначимо, чи є в ньому хоча б один символ, відмінний від пропуску. Якщо є, рядок копіюється в цільовий текст.

```
program copyPress;
  var f,g : text;
    fName : string;
procedure copyNonEmpty(var f,g:text);
  var s:string;  {поточний рядок тексту}
      k:byte;    {лічильник позицій}
      empty:boolean; {ознака порожності}
begin
  reset(f); rewrite(g);
  while not eof(f) do
  begin
    readln(f,s);
    {обробка рядка s}
    k:=1; empty:=true;
    while (k<=length(s)) and empty do
      if s[k] <> ' '
      then begin
          writeln(g,s);
          empty:=false
      end
      else inc(k);
```

```
      {рядок оброблено}
    end;
    close(f); close(g);
  end;
  begin
    write('Вхідний текст: '); readln(fName);
    assign(f,fName);
    write('Цільовий текст: '); readln(fName);
    assign(g,fName);
    copyNonEmpty(f,g);
  end.
```

Для перевірки цієї програми достатньо вхідного тексту, в якому лише три рядки — порожній, складений кількома пропусками та рядок, що містить, крім пропусків, хоча б один інший символ. У цільовий текст має скопіюватися тільки останній рядок.

3. КІЛЬКА СТАНДАРТНИХ ПІДПРОГРАМ ОБРОБКИ РЯДКІВ

• Рядки є єдиним нескалярним типом, значення якого ***можуть повертатися функціями***.

У системі Турбо Паскаль означено чимало корисних підпрограм обробки рядків. Розглянемо шість із них, позначивши іменем `string` будь-який можливий рядковий тип.

Функція `copy` з параметрами `s:string; start,count:integer` повертає підрядок рядка `s`, що починається з символа `s[start]` і має довжину `count`. Якщо `start+count-1>length(s)`, то повертається «хвіст» рядка від `s[start]` до кінця. Якщо `start≤0`, то повертається підрядок, що починається з `s[1]`. Наприклад, `copy('abcd',2,2)='bc'`, `copy('abcd',2,4)='bcd'`, `copy('abcd',-1,2)='ab'`.

Функція `pos` із параметрами `subs,s:string` повертає найменший номер того елемента в рядку `s`, починаючи з якого `subs` входить у `s` як підрядок (якщо не входить, повертається 0). Наприклад, `pos('bc','abcabc')=2`, `pos('aa','abca')=0`.

Процедура `delete` з параметрами `var s:string; start,count:integer` знищує `count` символів, починаючи

з позиції **start** у рядку **s**. За умов **start** = 0 або **count** = 0 або **start>length(s)** рядок не змінюється. За умови **start+count>length(s)** із **s** вилучається підрядок до кінця рядка. Наприклад, якщо **s='abcdef'**, то після виклику **delete(s,3,3)** значенням **s** буде **'abf'**, а після **delete(s,3,5)** — **'ab'**.

Процедура **insert** з параметрами **subs:string; var s:string; start:byte** вставляє **subs** як підрядок в **s**, починаючи з позиції **start**. Наприклад, якщо **s='abf'**, то після виклику **insert('cde', s, 3)** значенням **s** буде **'abcdef'**. Якщо довжина значення **s** після вставки перевищує максимально можливу, «хвіст» обрізається. Наприклад, якщо **s:string[5]** має значення **'abc'**, то після **insert('ffff',s,1)** її значенням буде **'ffffa'**.

Процедура **str** з параметрами **v; var s:string** перетворює значення першого аргументу (вираз цілого або дійсного типу) на рядок **s** так само, як при виконанні **writeln**. Зокрема, дійсні значення можна вивести в нормалізованому вигляді. Для цього після виразу через «**:**» можна вказати ширину поля виведення, а для дійсного виразу — ще кількість дробових цифр. Якщо зображення значення виходить за межі рядка, в ньому залишаються тільки старші цифри. Наприклад, рядок **s:string[3]** після виклику **str(1234,s)** одержить значення **'123'**.

Процедура **val** з параметрами **s:string, var v; var ErrCode:integer** перетворює зображення числа в рядку **s** на відповідний числовий тип і присвоює його змінній **v**. Якщо перетворення можливе, то значенням **ErrCode** буде **0**, інакше — позиція із символом у рядку, починаючи з якого перетворення неможливе. Тип аргументу для параметра **v** має відповідати змісту рядка **s**, а зміст рядка повинен задавати число, яке представляється в типі аргументу. Наприклад, за **s='1.3'** або **s='1E2'** другий аргумент має бути дійсним, а не цілим. Аналогічно при його типі, наприклад, **integer** у рядку не може бути значень, що представляють числа більше **32767** або менше **-32768**. Пропуски перед константою в рядку ігноруються.

- Процедуру **val** часто використовують для перевірки, чи є деяка послідовність символів числовою константою потрібного типу.

4. ПРИКЛАДИ ВИКОРИСТАННЯ ПІДПРОГРАМ ОБРОБКИ РЯДКІВ

Приклад. Прочитати з клавіатури два рядки та визначити всі позиції у першому рядку, починаючи з яких другий входить у нього як підрядок. Наприклад, у рядку **'abababa'** підрядок **'aba'** починається у позиціях 1, 3 та 5.

Нехай **s** і **p** — перший та другий рядки. Позицію першого входження поверне виклик **pos(p,s)**. Щоб отримати наступне входження, треба вилучити з рядка **s** початок з першим символом знайденого входження, і вже до скороченого рядка **s** застосувати **pos(p,s)**. При цьому треба врахувати кількість символів, вилучених з **s**. Запам'ятаємо цю кількість в змінній **deleted**. Повторювати пошук і вилучення будемо, поки **pos(p,s)>0**.

```
program allSubs;
  var s,p:string; {перший та другий рядки}
    deleted, {кількість вилучених символів}
    start:byte; {позиція початку підрядка}
begin
  readln(s); readln(p);
  deleted:=0;
  start:=pos(p,s);
  while start>0 do begin
    inc(deleted,start);
    s:=copy(s,start+1,length(s));
    write(deleted,' ');
    start:=pos(p,s);
  end;
end.
```

Приклад. Прочитати з клавіатури три рядки та замінити всі входження другого рядка у перший третім рядком. Врахувати, що входження заміняються від початку рядка й після заміни пошук продовжується з символу, що йде першим за вставленим рядком. Наприклад, у рядку **'ababac'** після заміни **'aba'** рядком **'*'** залишається **'*bac'**. Якби входження замінялися з кінця рядка, то залишилося б **'ab*c'**.

Нехай **s**, **p**, **r** — задані рядки. Як і у попередньому прикладі, знайдемо входження **p** в **s** за допомогою виклику **pos(p,s)**.

Нехай **start** — позиція, з якої починається входження, **len** — довжина рядка **p**. Починаючи з позиції **start** в **s**, знищимо **len** символів та вставимо **r** у рядок **s**, починаючи з позиції **start**. Продовжимо, якщо після цього **pos(p,s)>0**.

```
program replaceSubs;
  var s, p, r : string; {рядки}
    start, {позиція початку входження}
    len : byte; {довжина третього рядка}
begin
  readln(s); readln(p); readln(r);
  len := length(p);
  start := pos(p,s);
  while start>0 do begin
    delete(s,start,len);{вилучимо другий рядок}
    insert(r,s,start);  {та вставимо третій}
    start := pos(p,s);
  end;
  writeln(s);
end.
```

Приклад. Прочитати з клавіатури рядок, у якому записано дійсні константи, розділені пропусками в довільній кількості. Деякі з констант можуть бути помилковими. Вивести в двох окремих рядках правильні константи в нормалізованому вигляді та помилкові константи, відокремлені одним пропуском. Наприклад, за вхідного рядка

```
11.E2       mmm       1E2E2            0
```

виводяться два таких рядки.

```
Right: 1.1000000000E+03 0.0000000000E+00
Wrong: mmm 1E2E2
```

Щоб виділити константу з рядка, вилучимо пропуски перед нею. Вилучення пропусків на початку рядка оформимо процедурою з заголовком **delBlanks(var s:string)**.

Після виклику **delBlanks** символи рядка до першого пропуска або до кінця рядка утворюють константу або рядок порожній. Якщо рядок порожній, константи немає. Інакше визначимо позицію першого пропуска за допомогою функції **pos**. Якщо це 0, символи константи займають весь рядок, інакше його початок. Скопіюємо їх та вилучимо з рядка. Описані дії офор-

мимо функцією, що повертає виділену константу (можливо, порожню), з таким заголовком.

```
function getConst(var s:string):string;
```

Для того щоб перевірити, чи є дійсна константа правильною, застосуємо процедуру **val**. Вона присвоює своєму другому аргументу значення типу **real** і третьому аргументу 0, якщо константа правильна. За помилкової ж константи третій аргумент отримує ненульове значення, що й є ознакою помилки.

За умовою, правильні та помилкові константи треба вивести окремо, тому до закінчення обробки входу будемо дописувати константи до двох рядків **right** і **wrong** («правильні» та «помилкові»). Нормалізовану константу, відповідну числу типу **real**, отримаємо за допомогою процедури **str**.

```
program getcons;
procedure delBlanks(var s:string);
var i, len : byte; {поточний індекс та довжина}
begin
  i:=1;
  len:=length(s);
  while (i<=len) and (s[i]=' ') do inc(i);
  delete(s,1,i-1)
end;
function getConst(var s:string):string;
  var p:byte; {поточна позиція}
begin
  delBlanks(s);
  if s=''
  then getConst := ''
  else begin
     p:=pos(' ',s); {позиція пропуска}
     if p=0 then begin {пропуску немає}
       getConst:=s;  {скопіювати константу}
       s:=''         {й вилучити її}
     end
     else begin {за константою є пропуск}
       getConst:=copy(s,1,p-1);
       delete(s,1,p-1)
     end;
```

```
  end;
end;
procedure select(s:string; var r,w:string);
  var sNum:string ; {рядок для константи}
      num:real;
      err:integer;  {ознака помилки}
begin
  r:=''; w:='';
  while (s<>'') do
  begin
    sNum:=getConst(s);
    if sNum<>'' then
    begin
      val(sNum,num,err);
      if err<>0
      then w:=w+' '+sNum
      else begin
         str(num,sNum);
         r:=r+' '+sNum
      end;
    end;
  end;
end;
var s, right, wrong : string;
Begin
  readln(s);
  select(s,right,wrong);
  writeln('Right:', right);
  writeln('Wrong:', wrong);
  readln;
End.
```

5. МНОЖИНИ ТА ЇХ ПРЕДСТАВЛЕННЯ

Поняття «множина» є одним із первинних у математиці; воно не має точного означення, а лише розуміється як деякий набір або сукупність елементів. У математиці та інших науках множини утворюються з елементів, які мають деяке ***спільне походження***.

Наприклад, можна розглядати множини учнів класу, школи або країни, різні множини чисел, але множини, в яких були б одночасно й учні, й числа, насправді ніколи не розглядаються.

Будь-який елемент може ***належати*** або ***не належати*** множині. Елементи в множині не повторюються, тобто всі ***елементи множини є попарно різними***. Множини вважаються ***рівними***, якщо утворені одними й тими самими елементами.

Мова Паскаль надає зручні засоби для представлення та обробки множин, які складено значеннями ***перелічуваних типів*** — символів, цілих чисел (щоправда, не всіх) та типів, оголошених програмістом.

Спочатку розглянемо, як множини позначаються у програмі. Константу-множину задають явним переліком її елементів у дужках **[]**. Наприклад, множина чисел {1, 2, 3, 9} має вигляд **[1,2,3,9]**, порожня множина ∅ — **[]**. Якщо множина містить кілька значень, що йдуть у перелічуваному типі поспіль, можна записати їх як діапазон, наприклад, {1, 2, 3, 9} можна задати як **[1..3,9]**, а множину символів {**'a'**, **'i'**, **'j'**, **'k'**, **'l'**, **'m'**, **'n'**} — як **['a','i'..'n']**.

Якщо **T** — перелічуваний тип, то вираз **set of T** визначає ***тип множини*** (***множинний тип***). Його значеннями є всі можливі множини значень типу **T**. Наприклад, значеннями типу **set of Boolean** є множини булевих значень: **[]**, **[false]**, **[true]**, **[false,true]**, а значеннями типу **set of 'a'..'z'** — усі можливі множини, утворені з 26 малих латинських букв (цих множин 2^{26}). Тип **T** називається ***базовим*** для типу **set of T**.

Коли створювалася мова Паскаль, множини спочатку призначалися для подання лише множин символів. Цей тип має 256 елементів, і це обмеження було поширено на всі типи, що можуть бути базовими для множин.

- Базовим типом для множини може бути лише ***перелічуваний тип***, кількість елементів у якому ***не більше 256***.

У машинній програмі множину представляє бітовий масив (***бітове поле***), у якому кожен біт відповідає елементу базового типу й своїм значенням 0 вказує, що цей елемент не належить множині, а значенням 1 — що належить. Наприклад, дані типу **set of [0..9]** — це поля довжиною 10 біт, відповідних числам від 0 до 9; множину **[1..3,9]** представляє таке бітове поле.

Число	0	1	2	3	4	5	6	7	8	9
Наявність	0	1	1	1	0	0	0	0	0	1

- Номери бітів відповідають номерам елементів базового типу (починаючи з 0), тому елементи множини автоматично впорядковано за зростанням.
- Насправді множини займають цілу кількість байтів, тому для наведеного прикладу кількість біт збільшується до 16.

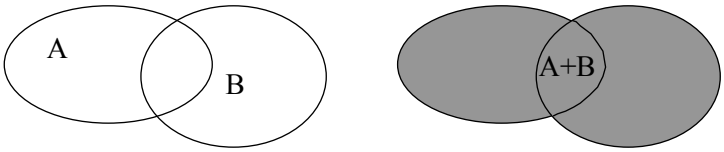
Змінні множинних типів оголошуються у звичайний спосіб. Тип у заголовку підпрограми задається іменем, а не виразом, тому множинні типи варто оголошувати окремо десь на початку програми. Розглянемо приклади.

```
type set_number = set of byte;
     set_char = set of char;
var Numbers : set_number; Symbols : set_char;
```

Змінній множинного типу можна присвоювати значення, однотипні зі змінною. Такі значення задаються за допомогою операцій з множинами.

6. ОПЕРАЦІЇ З МНОЖИНАМИ

Об'єднанням двох множин називається множина, якій належать елементи ***хоча б однієї*** з цих множин. У мові Паскаль операція об'єднання позначається знаком **+**.

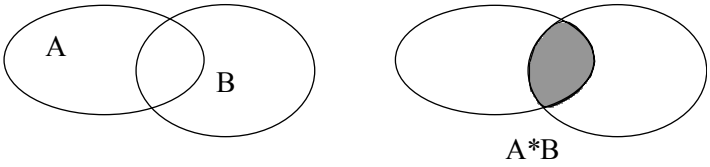


Наприклад:

```
['A','S']+['P','Q'] = ['A','P','Q','S'];
[1..10]+[5..15,20..30] = [1..15,20..30].
```

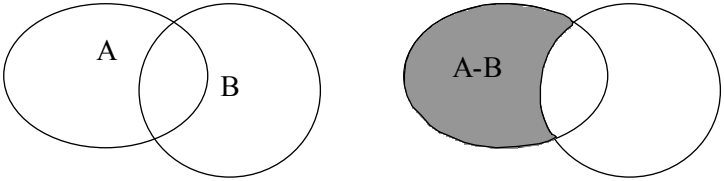
За допомогою операції об'єднання до множин можна додавати нові елементи. Наприклад, якщо змінна **A** типу **set of byte** має значення **[2,3,5,7]**, то після виконання присвоювання **A:=A+[5,9]** множина **A** матиме значення **[2,3,5,7,9]**.

Перетином двох множин називається множина, складена елементами, які належать ***одночасно обом*** цим множинам, тобто є спільними. У мові Паскаль ця операція позначається знаком *****.



Наприклад, **['C','H']*['R','W']** = **[]** (спільних елементів немає); **[1..5,10..15]*[3..12]** = **[3..5,10..12]**.

Різниця двох множин — це множина, складена елементами, які належать першій з цих множин і не належать другій. Цю операцію позначають знаком **-**.



Наприклад, **['A','B']-['R','W']** = **['A','B']** (всі елементи першої множини не належать другій); **[1..5]-[4..9]** = **[1..3]**.

Операція визначення, чи належить елемент множині, позначається зарезервованим словом **in**. Її результатом є **true**, якщо елемент належить множині, інакше **false**. Наприклад, вираз **13 in [10..20]** має значення **true**, оскільки **13** належить діапазону цілих чисел **10..20**; вираз **'я' in ['a'..'z']** має значення **false**, оскільки українська літера «я» не належить множині латинських літер.

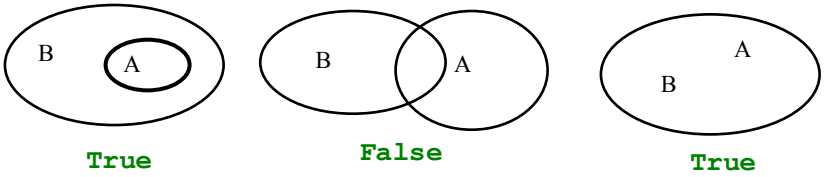
У мові Паскаль є операції порівняння множин, які мають булів тип. Операція **=** визначає, чи є дві множини рівними, тобто складеними з тих самих елементів. Наприклад, значенням виразу **[1..3]=[1,2,3]** є **true**, а виразу **[1..3]=[1,3]** — **false**.

Операція <> визначає, чи є дві множини нерівними. Наприклад, значенням виразу **[1..3]<>[1,2,3]** є **false**, а виразу **[1..3]<>[1,3]** — **true**.

Ще дві операції пов'язані з поняттям включення множин. Кажуть, що перша множина ***включається*** в другу, або є ***підмножиною*** другої, якщо кожен її елемент належить другій множині. По-

мітимо: множини, які включаються одна в одну, є рівними.

Операція зі знаком **<=** визначає, чи включається множина, вказана ліворуч від знака, у множину, вказану праворуч. Наприклад, вирази **[1,2]<=[1..3]** та **[1,2]<=[1..2]** мають значення **true**, а вираз **[2..3]<=[1..2]** — **false**. Значення виразу **A<=B**, де **A** та **B** — множини, залежно від включення множин вказано на рисунку.



Операція зі знаком **>=** визначає, чи включається множина, вказана праворуч від знака, у множину, вказану ліворуч. Наприклад, вирази **[1..3]>=[1..2]** та **[1,2]>=[1..2]** мають значення **true**, а вираз **[2..3]>=[1..2]** — **false**.

- Операції об'єднання, перетину, різниці та порівняння застосовують лише до ***однотипних множин***.

7. ПРИКЛАДИ ВИКОРИСТАННЯ МНОЖИН

Наведемо приклади задач, пов'язаних з обробкою текстів та рядків, у розв'язанні яких природно й зручно користуватися множинами символів.

Приклад. Прочитати від клавіатури два слова, складені малими латинськими літерами, та визначити, чи можна утворити друге з них, використовуючи лише букви першого слова. Кількість повторень букв у словах не має значення. Наприклад, з букв слова **bao** можна утворити слова **boa** та **baobab**, а слово **bay** — ні.

Утворимо множини літер, з яких складено слова. Тоді залишиться тільки визначити, чи включається друга множина у першу. Будемо вважати, що слова можна прочитати в змінні типу **string**. Утворення множини літер за рядком оформимо процедурою **LatSet**. Щоб додати елемент до множини, подамо його як одноелементну множину та застосуємо об'єднання множин.

```
program TwoStrings;
  type Latines = set of 'a'..'z';
procedure LatSet(var S:string; var L:Latines);
  var i : byte;
begin
  L:=[]; {спочатку множина порожня}
  for i:=1 to length(S) do
    L:=L+[A[i]] {додавання літери}
end;
var S1, S2 : string;
      L1, L2 : Latines;
begin
  writeln('Уведіть два рядки');
  readln(S1); readln(S2);
  LatSet(S1,L1); LatSet(S2,L2);
  writeln('З букв слова ', S1);
  writeln('побудувати слово ', S2);
  if L2<=L1
  then writeln(' можна')
  else writeln(' не можна');
end.
```

• Функція не може повертати значення структурного типу «множина», тому множина повертається за допомогою параметра-змінної.

Приклад. Прочитати текст і визначити, які з латинських букв зустрічаються в ньому, а які — ні. Розглядати малі й великі літери окремо.

Прочитаємо текст посимвольно й утворимо множину латинських літер, які в ньому зустрічаються. Потім двічі переберемо всі символи та першого разу виведемо ті літери, що належать множині, а другого — що не належать.

Множини літер подамо в типі **CharS = set of char**. Множину латинських літер, що зустрічаються в тексті, утворимо шляхом додавання кожної прочитаної літери до множини, а множину решти літер — як різницю між множинами всіх латинських літер та літер першої множини.

Утворення множини латинських літер тексту оформимо про-

цедурою **textLets**. Її додатковим параметром буде множина літер, які взагалі можуть додаватися до утворюваної множини.

Перебір символів і друкування тих із них, що належать деякій множині, оформимо процедурою **writeSet**. Її параметром буде множина символів.

```
program LatSets;
type CharS = set of char;
procedure textLets(var S:CharS;
            const GlobSet:CharS);
var f : text; c : char;
begin
  assign(f,'latsets.pas');
  reset(f);
  S := [];
  while not eof(f) do begin
    read(f,c);
    if c in GlobSet then S:=S+[c];
  end;
  close(f);
end;
procedure writeSet(const S:CharS);
var c:char;
begin
  for c := #0 to #255 do
    if c in S then write(c);
  writeln;
end;
const Lats:CharS=['A'..'Z','a'..'z'];
var Present,Absend:CharS;
begin
  textLets(Present, Lats);
  Absend := Lats-Present;
  writeSet(Present);
  writeSet(Absend);
end.
```

При виконанні наведеної програми, якщо входом є її власний текст у файлі **'latsets.pas'**, буде визначено такі множини присутніх та відсутніх латинських літер.

```
ACGLPSZabcdefghilmnoprstuvwxyz
BDEFHIJKMNOQRTUVWXYjkq
```

Приклад. Знайти всі розв'язки ребуса МУХА + МУХА = СЛОН.

Найпростіший спосіб розв'язання — перебрати всі можливі чотирицифрові числа та перевірити їх на відповідність вказаному ребусу. У цій задачі можливі числа належать діапазону **1023..4987** — мінімальне та максимальне чотирицифрові числа з попарно різними цифрами, які при множенні на 2 дають чотирицифровий результат.

Для кожного числа та його подвоєння створимо множини цифр; відповідність чисел умові означає, що ці множини мають по чотири цифри, а їх перетин порожній.

Для реалізації наведеного алгоритму використаємо тип «множина цифр».

```
type setDigit = set of 0..9;
```

Створення множини цифр чотирицифрового числа опишемо процедурою **createSet**, у якій цифри виділяються як остачі від цілочислового ділення на 10 та додаються до множини, що є параметром-змінною.

Кількість елементів у множині цифр обчислимо за допомогою функції **nDigits**, яка підраховує, скільки з цифр 0, 1, ..., 9 належать множині цифр.

З використанням наведених підпрограм розв'язання стає очевидним.

```
program rebus;
  type setDigit = set of 0..9;
procedure createSet(a:longint;
          var M:setDigit);
begin {побудова множини цифр числа a}
  M:=[];
  while a>0 do begin
    M:=M+[a mod 10];
    a:=a div 10;
  end;
end;
function nDigits(M:setDigit):byte;
  var p,i:byte;
```

```
begin {підрахунок елементів у множині цифр}
  p:=0;
  for i:=0 to 9 do
    if i in M then inc(p);
  nDigits:=p;
end;
var M1,M2:setDigit; {множини цифр МУХА,СЛОН}
  i:word; {перевіряється як МУХА}
Begin
  For i:=1023 to 4987 do
  begin
    createSet(i,M1);
    if nDigit(M1)=4
    then begin
          createSet(2*i,M2);
          if (nDigit(M2)=4)and(M1*M2=[])
          then writeln(i,'+',i,'=',2*i);
         end;
  end;
End.
```

Приклад. Знайти всі прості числа на проміжку [2..*n*], де *n* — деяке натуральне число.

Одне з розв'язань — перебрати всі числа від 2 до *n* та визначити, чи є вони простими, для чого перевірити, чи має число дільник у діапазоні від 2 до квадратного кореня з числа. Якщо число має дільник, його відкидають, інакше воно є простим і його друкують.

Проте існує інший метод визначення простих чисел, який має назву ***«решето Ератосфена»***. Випишемо всі числа заданого діапазону, а потім почнемо викреслювати ті з них, що мають менші за них дільники, відмінні від одиниці. Покажемо це на числах першого десятку.

2 3 4 5 6 7 8 9 10

Перше число 2 є простим. Викреслимо всі числа, кратні 2.

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~

Серед чисел, що залишилися (2 не враховуємо), перше число 3 просте. Викреслимо серед наступних усі, кратні йому.

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~

Останнім кроком для цього набору чисел викреслимо числа, кратні 5, оскільки це перше зліва невикреслене число. Втім, число 10 уже викреслено, тому список чисел не зміниться. Він містить числа 2, 3, 5 та 7.

За більшого діапазону числа треба викреслювати далі, поки не дійдемо до кінця діапазону. Насправді роботу можна скоротити. Помітимо, що будь-яке складене число *m* діапазону має простий дільник, не більший за $\sqrt{m}$. Звідси з діапазону [2..*n*] в якості простих чисел, які можуть приводити до викреслювання кратних їм чисел, достатньо брати числа, не більші за $\sqrt{n}$. Окрім того, якщо знайдено нове просте число *i*, то всі числа вигляду *ki*, де $k < i$, уже були викреслені на попередніх кроках. Звідси найменше кратне, яке слід викреслювати, дорівнює i^2.

Для реалізації алгоритму скористаємося множинами. Спочатку розглянемо дуже обмежений діапазон чисел **2..255**, а потім розглянемо спосіб його розширення.

Утворимо множину чисел **Primes**, у яку спочатку включимо всі числа від 2 до 255. Потім, знайшовши перше ліворуч невикреслене число, вилучимо з множини числа, кратні йому. Наступним простим буде невикреслене число, яке є першим у множині праворуч від знайденого. Процес повторюється, доки не буде перевірено всі числа, присутні в множині.

```
program PrimeNumbers;
var i, {кандидат у прості числа}
  multiple:word; {число, кратне простому}
  Primes:set of byte;
begin
  Primes:=[2..255];
  i:=1;
  while i<=sqrt(255) do
  begin
    inc(i);
    while not(i in Primes) do inc(i);
    multiple:=i*i;
    while (multiple<=255) do begin
      Primes:=Primes-[multiple];
      multiple:=multiple+i;
```

```
      end;
    end;
    {виведення невикреслених чисел}
    for i:=2 to 255 do
      if i in Primes
      then write(i:5);
  end.
```

Зверніть увагу, що змінні **i** та **multiple** оголошено з типом **word**, щоб запобігти зацикленню програми при отриманні значення **multiple** більше 255. Якби **multiple** мала тип **byte**, результати були б непередбачуваними.

Розширимо діапазон чисел за допомогою ***масиву множин***, індексованого, починаючи з **0**, кожен елемент якого відповідає за свій діапазон чисел (0..255, 256..511, 512..767 тощо). За цього розподілу чисел по діапазонах число *m* подано множиною (елементом масиву) з номером *m* div 256 та її елементом *m* mod 256. Якщо оголосити масив з індексами 0..2000, то він уміститься в 64 K та являтиме собою понад півмільйона чисел (точніше, 511999 чисел, починаючи з 0).

```
program PrimeNumbers;
var n, {межа діапазону}
  i, multiple, {кандидат у прості та кратне}
  mDiv,mMod:longint; {множина й число в ній}
  Primes : array[0..2000] of set of byte;
begin
  write('ціле не більше 511999>'); readln(n);
  for i:=0 to 2000 do Primes[i]:=[2..255];
  i:=1;
  while i<=sqrt(n) do
  begin
    inc(i);
    while not((i mod 256) in Primes[i div 256])
      do inc(i);
    multiple:=i*i;
    while (multiple<=n) do
    begin
      mDiv := multiple div 256;
      mMod := multiple mod 256;
```

```
        Primes[mDiv]:= Primes[mDiv]-[mMod];
        multiple:=multiple+i;
      end;
    end;
    {виведення невикреслених чисел}
    for i:=2 to n do
      if (i mod 256) in Primes[i div 256]
      then write(i:5);
end.
```

Якщо на вхід цієї програми дати максимально можливе значення 511999, то отримаємо доволі довгий перелік простих чисел, останнім з яких буде 511997.

• Діапазон чисел, серед яких шукаються прості, можна ***збільшити вдвічі***, якщо врахувати, що парні числа, окрім 2, не є простими. Тоді кожен елемент множини відповідає за непарне число, починаючи з 3. Доступ до множин (елементів масиву) та їх елементів задається аналогічно — число $m \geq 3$ подано множиною з номером (m div 2 – 1) div 256 та її елементом (m div 2 – 1) mod 256. Звичайно, перед виведенням невикреслених чисел треба окремо вивести 2.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Чому довжину рядків у мові Turbo Pascal обмежено числом 255?

2. Чи можна оголосити тип «множина чисел від 1 до 1000» за допомогою конструктора `set`*?*

3. Скількі байтів займають дані типу: а) `set of 0..15`*; б)* `set of 'a'..'z'`*; в)* `set of byte`*; г)* `set of char`*?*

ЗАДАЧІ

1. Що друкується в результаті виконання такої програми?

```
program StrConc;
  var a:integer; c:char; s:string;
begin
  s:='';
  for a:=0 to 2 do begin
    c:=chr(ord('0')+a);
    s:=s+c+s; writeln(s);
  end;
end.
```

2.У тексті, зв'язаному з файловою змінною **f:text**, містяться такі символи:

'1'	'2'	'3'	'4'	'5'	#13	#10	'6'	'7'	#13	#26	'*'

Вказати значення, яких набудуть змінні **s1**, **s2**, **s3**, **s4** типу **string[3]** після виконання таких операторів:

a) **readln(f,s1,s2);readln(f,s3);readln(f,s4)**;

b) **readln(f,s1);readln(f,s2,s3);readln(f,s4)**;

c) **read(f,s1);read(f,s2);readln(f,s3,s4)**.

3. Перевірити, чи є рядок ***паліндромом***, тобто симетричною послідовністю символів (що читається однаково зліва та справа).

4. Рядок містить латинські літери та пропуски. В алфавітному порядку надрукувати всі літери, які:

a) зустрічаються в рядку, та кількості їх входжень;

b) зустрічаються в рядку тільки один раз;

c) мають найбільшу кількість входжень.

5. Словом вважається непорожня послідовність символів, які не є пропусками. Слова в рядку розділено пропусками в довільній кількості:

a) підрахувати кількість слів у рядку;

b) знайти всі слова, що стоять на парних місцях;

c) знайти найдовше слово в рядку та його довжину;

d) знайти всі слова заданої довжини;

e) замінити кожну послідовність пропусків одним пропуском (стиснути пропуски);

f) вилучити з рядка всі однолітерні слова;

g) вилучити з рядка всі слова, які містять не більше двох літер;

h) вилучити з рядка всі слова непарної довжини;

i) дзеркально відобразити кожне слово;

j) підрахувати кількість слів-паліндромів;

k) рівномірно доповнити рядок пропусками до фіксованої довжини. Звернути увагу на знаки пунктуації (після крапки, коми тощо повинен бути хоча б один пропуск).

6. Рядок містить запис числа, яке виражає кількість копійок. Побудувати в іншому рядку відповідний словесний запис у гривнях та копійках.

7. Дано рядок символів. Підрахувати в ньому кількість знаків арифметичних дій: **'+'**, **'-'**, **'*'**, **'/'** та **'^'** (піднесення до степеня).

8. Дано рядок, що містить текст та арифметичні вирази типу $a \oplus b$, де a та b — довільні числа, а $\oplus$ — знак додавання, віднімання, множення або ділення. Виписати всі арифметичні вирази та обчислити їх.

9. Дано рядок символів. Знайти (тобто вивести на екран) у ньому всі слова, що починаються та закінчуються голосною (приголосною) літерою.

10. Дано рядок символів. Знайти у ньому всі цілі числа (числа можуть містити до 200 цифр, яким може передувати знак).

11. Дано два текстових файли. У першому з них кожен рядок містить слово-абревіатуру й через тире його розшифровку (слів не більше 50). У другому файлі є текст. У третій файл вивести цей текст, причому після того, як у тексті перший раз зустрічається якась розшифровка, записати у дужках відповідну абревіатуру, а всі інші такі ж розшифровки замінити їх абревіатурами. Врахувати, що розшифровку в другому файлі може бути розбито на кілька рядків.

12. Дано два текстових файли. Перший з них містить довільний текст, другий — не більше 30 слів, відокремлених комами. У другому файлі слова утворюють пари: кожне парне (за порядком запису) слово є синонімом записаного перед ним непарного. Замінити у першому файлі слова синонімами та записати результат у новий файл.

13. Дано рядок, що містить ім'я файла, яке може мати або не мати повний шлях доступу до файла. Визначити, чи коректне воно з точки зору операційних систем DOS або WINDOWS.

14. У заданому рядку знайти найдовшу послідовність пропусків та вивести її довжину словесним повідомленням у граматично правильній формі, наприклад, «12 пропусків», або «2 пропуски», або «21 пропуск».

15. В заданому рядку знайти слово, що містить найбільшу кількість різних букв. Якщо їх кілька, вивести всі.

16. Дано рядок символів. Вивести в алфавітному порядку всі латинські літери, що повторюються: a) не менше двох разів; b) не більше двох разів; c) в точності два рази.

17. Отримати всі розв'язки для таких ребусів:

a) ТРИ + САМ = ЛОБ; b) ІСК + ІСК = КСІ;

c) АВ + ВС + СА = АВС; d) ТОЧКА + КРУГ = КОНУС;

e) СОН + СОН = НІЧ;

f) ONE + TWO + TWO + TWO + TWO = NINE;

g) ЛІНІЯ + ЛІНІЯ = ФІГУРА;

h) TEN + TEN + FORTY = SIXTY.

18. У двох рядках тексту записано хімічне рівняння: у першому рядку — хімічні елементи й дії, у другому — коефіцієнти при атомах. Приклади:

1) $2H_2+O_2=2H_2O$ 2) $CaO+H_2O=Ca(OH)_2$

Кожному пропуску першого рядка відповідає *не* пропуск другого, і навпаки. Перевірити, чи правильно розставлено коефіцієнти при доданках у рівнянні. Додати до цього ж тексту рядок з відповіддю **ТАК** або **НІ**.

19. У кожному рядку текстового файла записано правильний арифметичний вираз, який містить круглі дужки, цілі числа та знаки арифметичних дій **+** та **–**. Обчислити значення кожного виразу.

ðîçä³ë 4

ÀÐÈÔÌÅÒÈÊÀ ÁÀÃÀÒÎÖÈÔÐÎÂÈÕ ×ÈÑÅË

При розв'язуванні деяких задач виникають числа, які не можна подати у стандартних типах мови Паскаль, адже дані цих типів мають невеликий діапазон значень. Наприклад, уже 13! не можна подати в типі **longint**, а дійсні типи, хоча й дозволяють подати набагато більші числа, але далеко не всі підряд. Отже, щоб розширити діапазон чисел або підвищити точність їх запису, необхідні власні, нестандартні числові типи.

Нижче розглянемо представлення цілих чисел зі збільшеною кількістю цифр на основі масивів та реалізуємо операції з цими числами: введення та виведення, порівняння, додавання та віднімання, множення, цілочислове ділення (обчислення частки та остачі від ділення, аналогічне операціям **div** та **mod**), десяткове ділення (обчислення результату як періодичного десяткового дробу).

1. ПРЕДСТАВЛЕННЯ БАГАТОЦИФРОВИХ ЧИСЕЛ ЗІ ЗНАКОМ

Будемо обробляти десяткові числа, в яких може бути до 1000 знаків. Нижче називатимемо ці числа ***довгими***. Подати довге число можна рядком символів (цифр), якщо кількість цифр не перевищує 255, або масивом цілих чисел від 0 до 9 (нижче вони називаються цифрами). Числа в рядковому записі легко вводити й виводити, але виконання арифметичних операцій з ними відчутно

ускладнюється. З масивом чисел, навпаки, легко виконувати арифметичні дії, але введення-виведення ускладнено. Арифметичні дії «важать більше», тому зберігатимемо числа за допомогою масивів цілих чисел, які представляють десяткові цифри.

Отже, десяткові цифри числа подамо в масиві типу **array[1..max] of shortint**, де **max** — максимальна кількість цифр у числі. Звичним є запис, коли молодшу цифру числа розташовують праворуч, тому молодшій цифрі числа відповідатиме елемент масиву з індексом **max**. Якщо число має менше ніж **max** цифр, скажімо, **i** цифр, то їх зберігають елементи масиву з індексами від **max-i+1** до **max**, а елементи з індексами від **1** до **max-i** представляють незначущі нулі.

Незначущі нулі не беруть участі у виконанні арифметичних операцій і не виводяться. Щоб не займатися щоразу визначенням, де в масиві починаються значущі цифри, зручно зберігати кількість цифр числа в окремій змінній. У нашому представленні для цього достатньо типу **word**.

Нарешті, число може бути додатним або від'ємним, тобто треба ще зберігати його знак «**+**» або «**–**» в окремій змінній типу **char**.

Отже, число мають представляти три різнотипні частини — ***знак***, ***довжина***, ***масив цифр***. Числові величини у програмах моделюються змінними, тому нам потрібен тип даних, значення якого мали б ***кілька різнотипних частин***.

У мові Паскаль типи, значення яких мають кілька різнотипних частин, оголошують як типи ***записів***, або ***структур***. Ці типи описують виразом, в якому вказано імена частин (вони називаються ***полями***) та їх типів.

```
record
  ім'я1 : тип1;
  ім'я2 : тип2;
  ...
  ім'яN : типN
end;
```

Тип для представлення довгих чисел назвемо **Long**, поле знака числа назвемо **sign**, поле довжини числа — **len**, поле цифр — **number**. Отже, оголосимо такий тип даних.

```
const max = 1000;
type
```

```
Long = record
  sign   : char;
  len    : word;
  number: array[1..max+1] of shortint;
End;
```

За цим описом, у типі **Long** можна подати цілі числа від **-99…9** до **+99…9** (кількість дев'яток числі дорівнює **max**). Додатковий (**max+1**)-й елемент масиву є допоміжним; він буде використовуватися для прискорення деяких операцій із числами.

Поле змінної типу запис позначається його іменем, яке записують після імені змінної та крапки. Наприклад, якщо змінна **a** має тип **Long**, то **a.sign** позначає змінну типу **char**, що зберігає знак числа, **a.len** — змінну типу **word**, що задає довжину числа, **a.number** — масив цифр, **a.number[i]** — елемент цього масиву з індексом **i**.

- Усі поля запису повинні мати ***різні імена***.
- У мові Паскаль запис як єдине ціле можна ***присвоювати***. Інших операцій із записами як цілісними даними немає, тому операції з ними необхідно програмувати самостійно.
- Дані типу запис не можна повертати з функції, але параметри у підпрограмах можуть бути довільними структурами.
- У мові *Turbo Pascal* тип параметра в заголовку підпрограми можна задавати тільки іменем, тому типи записів треба оголошувати вище у програмі.

2. УВЕДЕННЯ ТА ВИВЕДЕННЯ

Уведення. Реалізуємо порозрядне уведення довгого числа з клавіатури, за яким визначаються поля запису типу **Long**.

Спочатку перевіримо, чи не є перший уведений символ знаком «**+**» або «**-**» (запис додатного числа може не мати знаку). Якщо першим символом є знак, він заноситься в поле **sign**; цифри починаються з другого символа. Інакше число є додатним (у поле **sign** записується «**+**») і вже перший символ є цифрою числа. Масив цифр заповнюється, починаючи з першого елемента, причому цифри (значення від 0 до 9) утворюються за символами за допомогою функції **ord**. Уведення триває до появи символа кінця рядка **#13**, який виявляє функція **eoln**.

Після введення число зсувається й «притискається» до правого кінця масиву, тобто молодша цифра (кількість одиниць) стає значенням елемента з індексом **max**. Кількість позицій зсуву визначається кількістю цифр числа **X.len**, яка підраховується при їх уведенні. Після зсуву незначущі розряди з індексами від **1** до **max-X.len** заповнюються нулями.

```
procedure input(var X:Long);
var i:word; {змінна циклу}
    S:char; {символ, що уводиться}
Begin
  read(S);
  if (S='-') or (S='+')
  then begin {перший символ є знаком числа}
    X.sign:=S; X.len:=0;
  end
  else begin {перший символ є цифрою}
    X.sign:='+'; X.len:=1;
    X.number[1]:=ord(S)-ord('0');
  end;
   while not eoln do begin {уведення цифр}
     read(s); inc(X.len);
     X.number[X.len]:=ord(S)-ord('0');
   end;
   for i:=X.len downto 1 do {зсув цифр}
     X.number[max-X.len+i]:=X.num[i];
  for i:=1 to max-X.len do {незначущі розряди}
     X.number[i]:=0;
End;
```

Виведення. Виведення довгого числа на екран ще простіше: спочатку, якщо число від'ємне, виводиться знак, а потім усі значущі цифри числа (їх кількість є значенням поля **len**).

```
procedure output(X:Long);
  var i:word;
Begin
  if X.sign='-' then write('-');
  for i:=max-X.len+1 to max do
    write(X.number[i]);
End;
```

3. ПОРІВНЯННЯ

Реалізуючи порівняння довгих чисел, зручно скористатися допоміжною функцією, яка порівнює модулі двох довгих чисел (без урахування їх знаків). Отже, спочатку напишемо цю функцію.

Нагадаємо: значущі цифри числа притиснуто до правого краю масиву, тому порівняти їх дуже просто, якщо почати ліворуч. Будемо рухатися по масивах і пропускати рівні цифри (спочатку, можливо, це незначущі нулі). Зупинитися потрібно на перших зліва нерівних цифрах. Проте, якщо числа рівні, то пропускання рівних цифр виведе за межі масивів. Щоб уникнути цього, можна під час руху перевіряти, що поточний індекс **i** в масивах не більше **max**.

```
while (i<=max)and(A.number[i]=B.number[i])
    do inc(i); {рух зліва направо}
```

Замість перевірки умови **i<=max**, яка дещо уповільнює роботу, скористаємося методом так званого ***бар'єрного елемента***. У напрямку проходу масивами забезпечимо наявність елементів, які обов'язково припиняють виконання циклу. За руху масивами зліва направо ці елементи повинні бути в масивах праворуч, тобто у позиції з номером **max+1**, і мати будь-які, але ***обов'язково нерівні значення***, наприклад, **1** і **2**.

Якщо рух масивами зупинився на бар'єрних елементах, то це означає, що числа рівні. Інакше порівняння нерівних цифр визначає, яке з чисел більше. Отже, результатів може бути три: **-1** — перше число ***менше***, **0** — числа ***рівні***, **1** — перше число ***більше***.

```
function comp_abs(A,B:Long):shortint;
  var i:word;
Begin {бар'єрні елементи}
  A.number[max+1]:=1; B.number[max+1]:=2;
  i:=1;
  while (A.number[i]=B.number[i])
    do inc(i);
  {A.number[i]<>B.number[i]}
  if i>max then comp_abs:=0
  else if A.number[i]<B.number[i]
       then comp_abs:=-1
       else comp_abs:=1;
End;
```

Порівняння чисел з урахуванням знаків. Якщо знаки чисел різні, то меншим є від'ємне. З двох додатних меншим є число, модуль якого менший, а з двох від'ємних — модуль якого більший. Функція порівняння також повертає `-1`, `0` або `1`, якщо, відповідно, перше число менше, числа рівні або перше більше.

```
function compare(A,B:Long):shortint;
Begin
  if A.sign=B.sign
  then if A.sign='-'
       then compare:=-comp_abs(A,B)
       else compare:=comp_abs(A,B)
  else if (A.sign='-')
       then compare:=-1
       else compare:=1;
End;
```

4. ДОДАВАННЯ ТА ВІДНІМАННЯ

Знову спочатку напишемо допоміжні функції додавання та віднімання модулів довгих чисел (без урахування їх знаків). Вони реалізують алгоритми «у стовпчик», відомі з початкових класів, за якими розряди обробляються справа наліво; при додаванні в сумі двох розрядів враховується перенесення із попереднього розряду, а при відніманні — позика.

- У додаванні та відніманні беруть участь лише значущі розряди; на перший погляд, в реалізації варто це врахувати й не обробляти незначущі нулі, але це дещо ускладнює підпрограми. Окрім того, бажано присвоювати `0` незначущим розрядам, щоб гарантувати, що в можливих подальших операціях число представлено правильно. Отже, будемо обробляти значущі й незначущі цифри ***однаково***. Тоді розряди, які в результаті операції стають незначущими, автоматично отримують значення `0`.

Нам знадобиться також функція обчислення ***довжини числа***, тобто кількості його значущих цифр. Щоб обчислити довжину, пропустимо в масиві цифр всі нулі ліворуч, застосувавши згадану вище техніку «бар'єрного елемента». Не забувайте: число нуль має довжину `1`! .

```
function longLen(a:Long):word;
  var i:word;
begin
   i:=1;
   a.number[max+1]:=1; {бар'єрний елемент}
   while a.number[i]=0 do inc(i);
   if i<=max then longLen:=max-i+1
   else longLen:=1;
end;
```

Додавання модулів. При додаванні можливе ненульове перенесення **p** зі старшого розряду. Це свідчить про те, що суму чисел не можна подати в типі **Long**. У цій ситуації в якості найпростішої реакції виведемо повідомлення **'Addition overflow'** на екран, але продовжимо обчислення, ніби нічого не трапилося. Реальні програми у таких ситуаціях, як правило, хоча й не завжди, аварійно завершуються.

- При роботі з довгими числами бажано передбачати можливу довжину чисел і резервувати такий розмір представлення чисел, щоб воно вміщувало результати будь-яких можливих операцій з числами, навіть множення.

```
procedure plus_abs(A,B:Long; var Res:Long);
  var i:word; p:byte; s:shortint;
Begin
  p:=0;
  for i:=max downto 1 do
  begin
     s := A.number[i]+B.number[i]+p;
     Res.number[i] := s mod 10;
     p := s div 10;
  end;
  if p>0 then writeln('Addition overflow');
  {визначення довжини результату}
  Res.Len:=LongLen(Res);
End;
```

Віднімання модулів. Виконується також порозрядно, причому на деякому кроці може виникнути ситуація віднімання від меншого розряду більшого і тоді розряд результату буде від'ємним. В цьому випадку необхідно взяти ***позику*** — одиницю старшого

розряду, яка у поточному розряді буде десяткою, а потім врахувати її на наступному кроці. Позику подамо змінною **z**.

```
procedure minus_abs(A,B:Long; var Res:Long);
  var i:word;
      z:byte;
Begin
  z:=0;
  for i:=max downto 1 do
  begin
    Res.number[i]:=A.number[i]-B.number[i]-z;
    if Res.number[i]<0
    then begin
           inc(Res.number[i],10);
           z:=1;
         end
    else z:=0;
  end;
  {визначення довжини результату}
  Res.Len:=LongLen(Res);
End;
```

Віднімання модулів даватиме хибний результат, якщо перший модуль менше другого. Отже, перед викликом наведеної процедури треба забезпечити, щоб ***перший модуль був не менше за другий***.

Додавання довгих чисел з урахуванням знаків. Реалізуємо очевидні, відомі з курсу математики, міркування.

Якщо знаки чисел однакові, необхідно обчислювати суму модулів і надавати результату знак доданків.

Якщо знаки різні, то від більшого з модулів чисел необхідно відняти менший і надати результату знак доданка, більшого за модулем. Якщо модулі однакові, то знаком результата буде **'+'**.

```
procedure plus(A,B:Long; var Res:Long);
 var s:shortint;{результат порівняння модулів}
Begin
  if A.sign=B.sign then begin
     plus_abs(A,B,Res);
     Res.sign:=A.sign;
  end
```

```
    else begin
       s:=comp_abs(A,B);
       if s>=0 then begin
          minus_abs(A,B,Res);
          if s>0 then Res.sign:=A.sign
          else Res.sign:='+';
       end
       else begin
          minus_abs(B,A,Res);
          Res.sign:=B.sign;
       end
    end
  End;
```

Віднімання довгих чисел з урахуванням знаків. Віднімання — це додавання від'ємника з протилежним знаком, тобто $A-B = A+(-B)$.

```
  procedure minus(A,B:Long; var Res:Long);
  Begin
    if B.sign='-' then B.sign:='+'
    else B.sign:='-';
    plus(A,B,Res);
  End;
```

5. МНОЖЕННЯ

Щоб помножити два числа, треба помножити перше на кожен розряд другого, а потім додати результати порозрядного множення, зсуваючи кожен з наступних результатів ліворуч на один розряд. Щоб реалізувати цей алгоритм, можна написати процедуру, яка множить перше число на цифру другого, зсуває результат на відповідну кількість розрядів (залежно від розряду цифри) та додає отриманий результат до попередньо накопиченої суми. Проте для множення на одну цифру не будемо писати окрему процедуру, а реалізуємо всі необхідні дії разом, враховуючи зсуви за рахунок індексування.

Помітимо: добуток цифр у останніх розрядах **max** множників дає цифру в останньому розряді **max** результату та, можливо, перенос у розряд **max-1**. Узагалі, за алгоритмом множення «в сто-

впчик» та представленням чисел неважко переконатися, що добуток цифр у розрядах **i** та **j** множників додається до значення в розряді **i+j-max** результату та, можливо, створює перенесення у розряд **i+j-max-1**.

Отже, спочатку масив цифр результату **rez** заповнимо нулями. Потім у зовнішньому циклі пройдемо по розрядах **j** другого множника **b** (від молодших до старших), у внутрішньому — по розрядах **i** першого множника **a**. Щоб не множити на незначущі нулі, врахуємо реальну довжину чисел. Суму добутку **i**-ї та **j**-ї цифр, можливого переносу **p** від попереднього розряду та значення (**i+j-max**)-го розряду результату збережемо в змінній **multi**. Тоді **multi mod 10** є новим значенням у (**i+j-max**)-му розряді результату, а **multi div 10** — переносом у наступний розряд. Перенос у розряд **max** є нульовим.

Проте при множенні можлива ситуація, коли добуток чисел типу **Long** має стільки цифр, що його не можна подати в цьому типі. Появу «зайвих» цифр відстежимо за допомогою булевої змінної **over**. Якщо вона отримала значення **true**, результат множення обчислено з помилкою, тому видається повідомлення **Multipl. overflow**.

Нарешті, знак добутку визначається дуже просто: за однакових знаків множників результат додатний, інакше від'ємний.

```
procedure multiple(a,b:Long; var Res:long);
var i, j : word;
    p, multi : byte;
    over : boolean; {ознака переповнення}
begin
  for i:=1 to max do Res.number[i]:=0;
  over:=false;
  for j:=max downto max-b.len do
  begin {множення на j-у цифру другого множника}
    p:=0;
    for i:=max downto max-a.len do
      if i-max+j<1 then over:=true
      else begin
        multi:=Res.number[i-max+j] +
               a.number[i]*b.number[j] +p;
        Res.number[i-max+j]:=multi mod 10;
```

```
          p:=multi div 10;
          if (i-max+j=1) and (p>0)
          then over:=true;
        end;
    end;
    {визначення довжини та знака результату}
    Res.len:=LongLen(Res);
    if a.sign=b.sign then Res.sign:='+'
    else Res.sign:='-';
    if over then writeln('Multipl. overflow');
end;
```

6. ЦІЛОЧИСЛОВЕ ДІЛЕННЯ

Ділення цілих чисел можна виконувати по-різному, а саме, як:

- ***цілочислове ділення*** з обчисленням ***частки*** та ***остачі***, аналогічне операціям **div** та **mod** мови Паскаль;
- ***скорочення*** дробу;
- ***десяткове ділення*** з обчисленням періоду десяткового дробу.

Розглянемо обчислення цілої частки та остачі від ділення «в стовпчик». Спочатку частку й остачу покладемо рівними 0. Далі визначимо, чи не дорівнює дільник 0. Якщо це так, видамо повідомлення **Division by zero**. Інакше в діленому візьмемо таку кількість цифр, яка є в дільнику, й запишемо їх у додаткове довге число — ***остачу*** від ділення. Якщо цифр у діленому менше ніж у дільнику, то частка дорівнює 0, а остачею повинно бути ділене.

Далі, якщо довжина діленого не менше довжини дільника, повторимо такі дії (кількість повторень — це різниця довжин діленого та дільника плюс 1):

1) поки остача не менше дільника, віднімаємо від неї дільник; кількість віднімань дає нову цифру частки;

2) допишемо отриману цифру до частки як молодшу (для цього значення частки зсунемо ліворуч, звільнивши місце для нової цифри);

3) допишемо до остачі наступну цифру діленого як молодшу (на останньому кроці цього робити не треба, оскільки цифри діленого вже вичерпано).

У поданому алгоритмі є дії, які доцільно реалізувати в допоміжних підпрограмах: зсув цифр числа вліво, віднімання, порівняння з 0.

Зсув цифр довгого числа **a** ліворуч на **k** розрядів та присвоювання 0 молодшим розрядам опишемо такою процедурою.

```
procedure shift(var A:long; k:word);
  var i:word;
begin
   for i:=1 to max-k do
     A.number[i]:=A.number[i+k];
   for i:=max-k+1 to max do A.number[i]:=0;
end;
```

Віднімання вже реалізовано процедурою **minus_abs** (див. параграф 4).

Порівняння числа з нулем дає відповідь «так» або «ні», тому реалізуємо його функцією, що повертає булеве значення. Для припинення циклу пошуку ненульової цифри використовуємо метод бар'єрного елемента і тоді рівність довгого числа нулю буде в тому випадку, коли після закінчення роботи циклу відбудеться вихід за межі масиву, тобто **i = max + 1**.

```
function comp_0(X:long):boolean;
  var i:word;
begin
  i:=1;
  X.number[max+1]:=1; {бар'єрний елемент}
  while X.number[i]=0 do inc(i);
  comp_0:=(i=max+1);
end;
```

Нам знадобиться також функція обчислення ***довжини числа*** (див. вище). Після віднімання діленого від остачі довжина остачі невідома, тому її треба визначити. Окрім того, коли утворено початкову остачу, вона може бути як більше діленого, так і менше. Залежно від цього перша цифра, яка записується в частку, є або не є значущою. Отже, кількість цифр остачі теж невідома, тому її теж треба визначити.

Нарешті наведемо процедуру **div_mod** ділення з остачею. Ділене та дільник подані параметрами-значеннями **A** і **B**, частка та остача — параметрами-змінними **Quot** і **Rest** (від англійських

quotient — частка, *rest* — остача). Додамо лише, що частка є додатною, якщо знаки діленого та дільника рівні, інакше від'ємною, а знак остачі збігається зі знаком діленого (саме так означено для операцій **div** і **mod** мови Turbo Pascal).

```
procedure div_mod(A,B:long;
          var Quot,Rest:long);
  var i, p, nd : word; s:byte;
begin
  {Частку та остачу покладемо рівними 0}
  for i:=1 to max do begin
    Quot.number[i]:=0; Rest.number[i]:=0;
  end;
  if comp_0(B) then
     writeln('Division by zero')
  else begin {Обчислення частки та остачі}
    if A.len<B.len then nd:=A.len
    else nd:=B.len;
    {Візьмемо з діленого nd початкових цифр}
    for i:=1 to nd do
       Rest.number[max-nd+i]:=
         A.number[max-A.len+i];
    {Якщо ділене коротше дільника, то частка}
    {дорівнює 0, а остачею повинно бути ділене}
    if A.len>=B.len then
      for p:=max-A.len+B.len to max do
      begin {Обчислюємо цифру частки}
        s:=0;
        Rest.len:=longLen(Rest);
        while comp_abs(Rest,B)>=0 do begin
          minus_abs(Rest,B,Rest);
          inc(s);
        end;
      {Дописуємо цифру до частки}
       shift(Quot,1);
       Quot.number[max]:=s;
       if p<max then begin
         {«Зносимо» наступну цифру діленого}
         shift(Rest,1);
```

```
          Rest.number[max]:=A.number[p+1];
        end;
      end;
      Quot.len:=longLen(Quot);
      Rest.len:=longLen(Rest);
    end;
    if A.sign=B.sign then Quot.sign:='+'
    else Quot.sign:='-';
    Rest.sign:=A.sign;
  end;
```

7. СКОРОЧЕННЯ ДРОБУ

Щоб скоротити дріб, треба поділити чисельник і знаменник на їх найбільший спільний дільник (НСД). Отже, спочатку розглянемо процедуру обчислення НСД двох довгих чисел.

За модифікованим алгоритмом Евкліда обчислення НСД треба, поки обидва числа не рівні 0, ділити з остачею більше число на менше й заміняти ділене цією остачею. Останній ненульовий дільник і буде шуканим НСД. У наступній процедурі ім'я GCD є скороченням англійських слів *greatest common divisor* — найбільший спільний дільник.

```
procedure _GCD(A,B:Long; var GCD:Long);
  var Quot:Long;
begin
  while not comp_0(A) and not comp_0(B) do
     if comp_abs(A,B)>0
     then div_mod(A,B,Quot,A)
     else div_mod(B,A,Quot,B);
  if comp_0(A) then GCD:=B else GCD:=A;
end;
```

Знайшовши НСД чисельника та знаменника, скоротимо їх на цей НСД і отримаємо результат у вигляді пари довгих чисел, які представляють нескоротний дріб. У процедурі скорочення (**reduce**) початкові чисельник і знаменник представлено параметрами-значеннями **A** та **B**, скорочені — параметрами-змінними **Num** і **Den** (ці імена є скороченнями від *numerator* — чисельник, *denumerator* — знаменник). Знаменник скороченого дробу

вважаємо додатним, а знак чисельника залежить від того, чи рівні знаки початкових чисел.

```
procedure reduce(A,B:Long; var Num,Den:Long);
  var GCD,Rest:Long;
      {змінні для НСД та остачі}
begin
  if comp_0(B) then
     writeln('Zero Denumerator')
  else begin
    _GCD(A,B,GCD);
    div_mod(A,GCD,Num,Rest);
    div_mod(B,GCD,Den,Rest);
  end;
  Den.sign:='+';
  if A.sign=B.sign then Num.sign:='+'
  else Num.sign:='-';
end;
```

8. ДЕСЯТКОВЕ ДІЛЕННЯ

Результатом десяткового ділення **A** на **B** є десятковий періодичний дріб. Цілу частину дробу знайдемо як частку від цілочислового ділення **A** на **B**. Остача **Rest** від цього ділення буде чисельником у дробовій частині. Якщо **Rest** дорівнює 0, дробова частина відсутня. Інакше відразу скоротимо **Rest** і **B** на їх НСД.

Правильний десятковий дріб має частину до періоду й період у дужках **()**. Знайдемо частину дробу до періоду на основі такого факту: кількість розрядів до початку періоду є максимальною з кількостей множників 2 та 5 у дільнику. Кількості множників знайдемо за допомогою функції **nMult**, що повертає степінь, з яким число **numb** (від 2 до 9) входить як множник у довге число **A**.

```
function nMult(A:Long; numb:byte):word;
  var i,k:word; S,Rest:Long;
begin
  k:=0;
  for i:=1 to max do S.number[i]:=0;
  S.number[max]:=numb; S.sign:='+'; S.len:=1;
  repeat
```

```
      div_mod(a,s,a,Rest);
      if comp_0(Rest) then inc(k);
    until not comp_0(Rest);
    nMult:=k;
  end;
```

Цифри дробової частини обчислюються шляхом ділення правильного дробу *A/B* у стовпчик. Обчислюється добуток 10*A* та ділиться націло на *B*. Частка є першою цифрою, а остача визначає чисельник правильного дробу зі знаменником *B*. Подальші множення цих чисельників на 10 та ділення на *B* дають наступні цифри. Один крок цього алгоритму (множення чисельника на 10, ділення на *B* та обчислення нового чисельника) реалізуємо функцією, яка повертає цілу частку від ділення (число від 0 до 9). Ім'я *entire* означає «ціле».

```
  function entire(var A,B:Long):byte;
    var n : byte;
  begin
    n:=0;
    shift(A,1); A.len:=longLen(A);
    while comp_abs(A,B)>=0 do begin
      minus_abs(A,B,A); inc(n);
    end;
    entire:=n;
  end;
```

Отже, виведемо дробову частину до періоду за допомогою **k** викликів функції **entire(Rest,B)**, де **k** — довжина частини дробу до періоду, обчислена раніше.

Остання остача, отримана при виведенні дробової частини до періоду, дасть першу цифру періоду. Запам'ятаємо її та продовжимо виводити цифри періоду, поки ще раз не отримаємо цю саму остачу — це й буде кінцем періоду. Період візьмемо в круглі дужки. Окремий випадок — якщо остача рівна 0, то період виводити не треба.

Проте довжина періоду може бути дуже великою, тому ***запам'ятовувати цифри дробової частини не будемо***. Обмежимо кількість цифр періоду, що виводяться, числом 2000, щоб вони уміщалися на екрані. Якщо після виведення 2000 цифр кінець періоду не досягнуто, виведемо **'...'** й на цьому закінчимо.

```
procedure div_dec(A,B:Long);
  var GCD_Rest_B, S, Quot, Rest : Long;
    i, k, k2, k5 : word;
    lPeriod : integer;
begin
  k:=0;
  div_mod(A,B,Quot,Rest);
  {Виведення цілої частини}
  output(Quot);
  {Якщо є дробова частина, вона виводиться}
  if not comp_0(Rest) then
  begin
    write(',');
    _GCD(Rest,B,GCD_Rest_B);
    div_mod(Rest,GCD_Rest_B,Rest,S);
    div_mod(B,GCD_Rest_B,B,S);
    k2:=nMult(B,2); k5:=nMult(B,5);
    if k2<k5 then k:=k5 else k:=k2;
    if k<>0 then {Виведення цифр до періоду}
      for i:=1 to k do write(entire(Rest,B));
    {Виведення періоду або його початку}
    if not comp_0(Rest) then
    begin
      write('(');
      lPeriod := 0;
      A:=Rest;
      repeat
        write(entire(Rest,B));
        inc(lPeriod)
    until(compare(A,Rest)=0) or
         (lPeriod=2000);
      if compare(A,Rest)<>0
         then write('...');
      write(')');
    end;
  end;
end;
```

Якщо викликати цю процедуру з аргументами, які представляють числа **145** і **12**, то відповіддю буде **12,08(3)**. За чисел **-12** і **4** відповіддю буде **-3**, за чисел **1** і **511997** період почнеться цифрами **00000195**, але його кінець не буде досягнуто, тому виведення закінчиться символами **540...)**.

9. МОДУЛЬ ДЛЯ РОБОТИ З БАГАТОЦИФРОВИМИ ЧИСЛАМИ

Підпрограми обробки довгих чисел можуть знадобитися у різних програмах, і кожна така програма має містити чимало з наведених вище підпрограм. Проте цього ***дублювання коду*** можна позбутися. Система *Turbo Pascal* дозволяє зібрати всі засоби (оголошення та підпрограми), необхідні, наприклад, для обробки довгих чисел, в окрему програмну одиницю — ***модуль***, або ***бібліотеку***. Якщо є цей «збірник оголошень та підпрограм», то у програмі треба лише вказати на його використання.

Модуль у мові Турбо Паскаль має такий загальний вигляд.

```
unit ім'я-модуля;
  Інтерфейсний розділ;
  Розділ реалізації;
  Розділ ініціалізації
end.
```

Слово **unit** («одиниця») є службовим. Інтерфейсний розділ починається службовим словом **Interface** і містить оголошення імен, які можуть використовуватися за межами модуля. Замість підпрограм тут записуються тільки їх заголовки. Слово інтерфейс стосовно програми можна розуміти як «зовнішній вигляд її виконання», а модуля — як «те, що в ньому видно зовні».

Розділ реалізації починається службовим словом **Implementation** («втілення», «реалізація») і містить усі підпрограми, вказані в інтерфейсному розділі, та, можливо, деякі інші. Інтерфейсні підпрограми тут можуть мати як повні заголовки, так і скорочені — за словом **function** або **procedure** записується тільки ім'я та крапка з комою.

Розділ ініціалізації задає дії, які виконуються один раз на початку виконання програми, що використовує оголошення модуля. Він має такий вигляд: **begin** ***послідовність операторів;***

В операторах, як правило, присвоюються початкові значення змінним, оголошеним у модулі. Цей розділ не є обов'язковим.

Розглянемо загальний вигляд модуля для роботи з довгими числами (розділу ініціалізації в ньому немає).

```
unit longNums;
Interface
  const max = 1000; {максимальна довжина чисел}
  type Long = ...;  {тип чисел}
  {заголовки інтерфейсних підпрограм}
  procedure input(var X:Long);
  procedure output(X:Long);
  function comp_abs(A,B:Long):shortint;
  function compare(A,B:Long):shortint;
  procedure plus_abs(A,B:Long; var Res:Long);
  procedure minus_abs(A,B:Long;
            var Res:Long);
  procedure plus(A,B:Long; var Res:Long);
  procedure minus(A,B:Long; var Res:Long);
  procedure multiple(a,b:Long; var Res:long);
  procedure div_mod(A,B:long;
            var Quot,Rest:long);
  function comp_0(X:long):boolean;
  procedure reduce(A,B:Long;
            var Num,Den:Long);
  procedure _GCD(A,B:Long; var GCD:Long);
  procedure div_dec(A,B:Long);
Implementation
  {допоміжні підпрограми}
  procedure shift(var a:long; k:byte); ... end;
  function longLen(a:long):byte; ... end;
  function entire(var A,B:Long):byte; ... end;
  {інтерфейсні підпрограми}
  procedure input(var X:Long); ... end;
  function compare; {скорочений заголовок}
  ...
  end;
  ... {решта підпрограм}
end.
```

Використання імен модуля у програмі або в іншому модулі називається ***використанням модуля***. Його вказують відразу після заголовка програми або слова **interface** у модулі за допомогою такого оголошення.

uses *ім'я-модуля;*

Якщо програма або модуль використовує декілька інших модулів, то їх імена записують через кому.

uses *ім'я1, ім'я2, ... ;*

Для прикладу розглянемо програму, яка, використовуючи засоби модуля **longNums**, вводить два довгих числа, що задають чисельник і знаменник дробу, та виводить дріб у десятковому представленні.

```
program Dec_Fraction;
  uses longNums;
  var A, B : Long; {чисельник і знаменник}
begin
  input(A); input(B);
  div_dec(A,B);
end.
```

Як і Паскаль-програми, Паскаль-модулі записуються у файли з розширенням **.pas**. Модуль транслюється у файл з розширенням **.tpu** (**tpu** — скорочення від Turbo Pascal Unit). Компілювати програми й інші модулі, що використовують модуль, можна тільки після його трансляції.

Наприклад, якщо модуль **longNums** записано у файл з іменем **longNums.pas**, то при трансляції буде створено файл **longNums.tpu**. Потім можна транслювати наведену вище програму **Dec_Fraction**. Тоді засоби з модуля **longNums.tpu**, що використовуються у цій програмі, буде додано до її машинного варіанту.

У системах програмування, як правило, стандартні підпрограми та ряд оголошень організуються в декілька модулів. Серед них, як правило, є «головний», який використовується практично всіма програмами. Він містить процедури введення та виведення, математичні функції та деякі інші підпрограми. Під час трансляції програми цей модуль підключається до неї неявно, тобто без посилання **uses**. Використання всіх інших модулів, як написаних програмістом, так і системних, треба задавати явно.

У системі Турбо Паскаль є вісім стандартних модулів (бібліотек). П'ять із них (**SYSTEM**, **DOS**, **CRT**, **PRINTER**, **OVERLAY**) зібрано в бібліотечний файл **TURBO.TPL**; модуль **SYSTEM** підключається автоматично. Є й інші модулі в окремих **TPU**-файлах, проте в цій книжці вони не вивчаються.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке запис як тип даних? Яким є формат його опису?

2. Чому виникає необхідність використання «довгих» чисел?

3. Назвіть основні засоби представлення «довгих» чисел.

4. Складіть словесний опис порозрядного додавання, віднімання, множення, ділення та порівняння цілих чисел.

5. Що таке модуль (бібліотека)? Як його оформити мовою Turbo Pascal?

6. Що описується в розділі інтерфейсу модуля?

7. Що повинен містити розділ реалізації модуля?

8. У розділі реалізації модуля можна вказувати неповні заголовки деяких підпрограм. Яких саме й чому?

9. Навіщо в модулі потрібен розділ ініціалізації?

ЗАДАЧІ

1. Реалізувати модуль **longNums**. Написати програми, які, використовуючи модуль, вводять два довгих числа, застосовують до них одну з арифметичних операцій та виводять результат.

2. Модифікувати процедуру **shift** зсуву числа (див. параграф 6), щоб поле **len** числа змінювалося належним чином. Врахувати, що зсув може призвести до недопустимого збільшення довжини.

3.На вхід програми від клавіатури в окремих рядках подаються довгі цілі числа та знаки операцій **+**, **-**, *****, **/** і **%** між числами (**/** і **%** позначають div і mod). Результат застосування операції виводиться на екран і стає першим операндом наступної операції, якщо її буде задано. Ознакою закінчення є натискання <Ctrl+Z> замість наступного знака операції. Наприклад, після уведення **1 + 2** (в окремих рядках) виводиться **3**, а після подальшого введення *** -4** виводиться **-12**.

4. Написати процедуру піднесення довгого числа до степеня з показником типу **byte**.

5. Написати процедуру, яка обчислює значення факторіала як довге число. Визначити найбільше число, факторіал якого можна представити у типі довгих чисел, який використано.

6. Написати процедуру, яка обчислює число Фібоначчі за його номером (як довге число). Визначити найбільший номер числа Фібоначчі, яке можна представити в типі **Long**.

7. Обчислити наближення до числа *e*. Скористатися формулою $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots$, накопичуючи доданки у вигляді дробу з довгими цілими чисельником і знаменником.

8. Обчислити наближення до числа π. Скористатися формулою $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$, накопичуючи доданки у вигляді дробу з довгими цілими чисельником і знаменником.

9. Кожна вершина трикутника має цілу додатну вагу. Середина кожної сторони трикутника стає вершиною другого трикутника, а її вага дорівнює сумі ваг вершин, прилеглих до сторони. Аналогічно утворюється третій трикутник, четвертий тощо. Вагою трикутника є сума ваг його вершин. Обчислити вагу *n*-го трикутника. На вхід подається номер *n* ($n \le 10000$) та три ваги вершин першого трикутника — цілі числа типу **longint**. Наприклад, за номером **3** й вагами **3 2 4** обчислюється **36**.

10. Задано числа *A* і *B* типу **Long**. Знайти запис *A* у системі числення з основою *B*. Цифри запису видати в такому вигляді: або це десяткова цифра, або записаний у квадратних дужках десятковий запис числа, що має не менше двох цифр. Наприклад, за чисел **123** і **21** вихід має вигляд **5[18]**.

11. Увести довге ціле число та вивести його розклад на прості множники. Між множниками вивести знак *****. Якщо множник у розкладі має степінь більше 1, вивести показник степеня після множника та знака **^**. Наприклад, за входу **12** виходом має бути **2^2*3**.

12. Обчислити кількість діагоналей опуклого *n*-кутника; *n* є числом типу **Long**, у якому не більше 10 цифр.

13. За числом *n*, у якому не більше 10 цифр, обчислити кількість пар діагоналей опуклого *n*-кутника, що перетинаються.

Наприклад, у чотирикутнику є одна пара діагоналей, у п'ятикутнику їх п'ять, у шестикутнику — 15.

14. Шляхові робочі мають плитки для тротуарів 1×1 та 1×2. Скількома різними способами вони можуть замостити доріжку розміром $2\times n$? Плитки 1×2 зроблено так, що вони лягають уздовж доріжки тільки широкою стороною. Врахувати, що $n\le 10^3$.

15. Чорнобильські орли мають довільну кількість голів. Коефіцієнт інтелекту (IQ) орла рівний кількості його голів. Орли об'єднуються в групи. IQ групи дорівнює добутку IQ орлів групи. Наприклад, у групи з трьох орлів, у яких 5, 3 та 2 голови, IQ дорівнює 30. За сумарною кількістю n голів орлів у групі визначити її максимально можливий IQ ($n\le 3000$). Наприклад, 5 голів дають IQ 6, 6 голів — 9.

ДОДАТОК

ОКРЕМІ МОЖЛИВОСТІ СЕРЕДОВИЩА TURBO PASCAL

ФАЙЛИ СЕРЕДОВИЩА TURBO PASCAL

Систему програмування *Turbo Pascal* (далі називатимемо її системою) встановлюють, як правило, у каталозі, ім'я якого **TP**, **BP**, **PASCAL** або схоже на них. Залежно від конкретної установки у цьому каталозі чи його підкаталозі **\BIN** ви знайдете файл **TURBO.EXE** (або **BP.EXE**). Ця програма є діалоговою системою програмування *Turbo Pascal* (відповідно, *Borland Pascal*, яка реалізує ту саму мову програмування) і містить текстовий редактор, транслятор, компонувальник, налагоджувач та завантажувач. Для роботи з нею необхідні файли **TURBO.TPL** та **TURBO.HLP** — головна бібліотека стандартних підпрограм та довідкова інформація про мову й систему програмування. Потрібний також файл конфігурації системи **TURBO.TP**, в якому зберігаються параметри її настроювання (пов'язані з використанням оперативної пам'яті, кольорами на екрані тощо).

ОСНОВИ РОБОТИ В СЕРЕДОВИЩІ

Система Турбо Паскаль забезпечує роботу з кількома прямокутними областями екрана — вікнами, меню та полями.

Вікна містять тексти програмних одиниць, результати їх компіляції, інтерпретації та виконання, а також довідкові повідомлення й поля для уточнення команд. У будь-якому стані системи одне з вікон є ***активним***; його виділено ***подвійною рамкою***, решта — неактивні (в одинарній рамці). Після запуску системи активним є вікно редактора, відмічене іменем **NONAME00.PAS** (щоправда, можливе й інше).

Для роботи з вікнами використовують клавіатуру або «мишу». Якщо в системі вікрито кілька вікон, їх можна активізувати по черзі за допомогою <F6>. Ви можете також перемістити курсор миші на відкриту частину неактивного вікна й натиснути ліву кнопку. Вікно створюють за допомогою <F3>, знищують — <Alt+F3>.

Вікно може показувати не весь текст, з яким працює користувач, а лише його частину. Сам текст зберігається системою в окремій частині оперативної пам'яті — ***буфері***.

Меню містять позначення інших меню (підменю) або команд, які система може виконати. Ці позначення називаються ***кодовими словами*** або ***опціями***. Символи «...» після опції команди означають, що команду буде виконано не відразу, а після подальших дій користувача. Праворуч від деяких опцій вказано функціональні клавіші чи сполучення клавіш, наприклад, <F3> або <Alt+X>, — за їх допомогою задають команду, не використовуючи систему меню.

Потрібну опцію в меню виділяють за допомогою клавіш-стрілок, а вибирають, натискаючи <Enter> або ліву кнопку миші. Якщо праворуч від опції вказано «...», то при її виборі розгорнеться діалогове вікно з кількома полями, а якщо трикутний значок — додаткове меню.

Поля в діалогових вікнах дозволяють задавати параметри команди, наприклад, імена файлів або каталогів. При відкритті діалогового вікна одне з полів є активним; його виділено кольором. Для переходів між полями використовують клавішу <Tab>, а всередині поля — клавішу переміщення курсору. Діалогове вікно закривають за допомогою клавіші <Esc> (із утратою всіх установлених полів) або <Enter> (з виконанням команди або установок, заданих у полях).

ПЕРШИЙ СЕАНС РОБОТИ

Запустимо програму **TURBO.EXE** (**BP.EXE**) й побачимо її «головний екран». Верхній рядок — це ***головне меню*** режимів роботи з системою (пункти меню задано словами **File**, **Edit**, **Search** тощо). Нижче розташовано велике порожнє поле в рамці — ***вікно текстового редактора***. Нагорі записано **NONAME00.PAS** — це ім'я файлу, з яким ми будемо працювати (***поточного файла***). Цей файл буде записуватися у поточному каталозі, яким спочатку є каталог із системою. Знизу розміщено ще один рядок («***підказка***»), в якому перелічено функціональні клавіші та їх функції (<F1> — Help, <F2> — **Save**, тобто «врятувати», тощо).

Початок роботи з програмою. Відразу слід змінити поточний каталог, щоб ніяких файлів не додавати до каталогу із системою. Натиснемо функціональну клавішу <F10> — у головному меню буде підсвічено один із пунктів. За допомогою клавіш-стрілок (праворуч на клавіатурі) або миші виберемо пункт **File** і натиснемо <Enter>. З'явиться вертикальне меню роботи з файлами (пункти **New**, **Save**, **Open**

тощо). За допомогою клавіш-стрілок виберемо пункт **Change Dir** (змінити каталог) і натиснемо <Enter>. У вікні, що з'явиться, вкажемо шлях до потрібного каталогу і натиснемо <Enter>. Поточний каталог установлено.

Тепер задамо ім'я поточного файла. Знову виберемо пункт головного меню **File** (файл) і в ньому пункт **Save As** (записати як ...). У вікні, що з'явиться, вкажемо ім'я файла з розширенням **.pas** або без розширення (тоді система встановить його як **.pas**) і натиснемо <Enter>. Побачимо головне вікно редактора, де замість **NONAME00.PAS** ми задали ім'я.

Якщо при роботі з меню ви зробили якусь помилку, натискання на **Esc** (escape — утекти) поверне до вікна редактора.

Набирання програми. Наберемо у вікні редактора текст програми.

```
program First;
begin
  writeln('Hello!');
end.
```

Курсор у вікні редактора своєю формою вказує на ***режим*** роботи — ***вставку*** чи ***заміну***. При вставці курсор є підкресленням, при заміні — прямокутником. Режими переключаються клавішами <Ins> або <Insert> праворуч. При натисканні <Enter> (у режимі вставки) курсор буде переведено до нового рядка, а до тексту додано невидимий символ, що задає кінець рядка. Для виправлення помилок використовують клавіші <BackSpace> та <Delete>; <Ctrl+Y> задає видалення рядка, в якому перебуває курсор.

Запис програми на диск. Набравши програму, запишемо її за допомогою клавіші <F2> у поточний pas-файл поточного каталогу. Якби програма була більшою, цей запис слід було б повторювати, поступово додаючи текст у файлі.

Компіляція та виконання програми. Для компіляції натисніть <F9>, для компіляції та виконання — <Ctrl+F9>. Якщо помилок при набиранні не було, то у першому випадку з'явиться повідомлення про успішну компіляцію, а в другому щось «блимне й зникне». Натиснемо комбінацію клавіш <Alt+F5> й побачимо чорне ***вікно виведення програми*** з рядком **Hello!**. Далі будь-яка клавіша поверне до вікна редактора. Можна змусити програму чекати, поки ми не натиснемо <Enter>, додавши перед рядком зі словом **end** рядок із словом **readln**.

Помилки у програмі. Набираючи програму, можна помилитися. Тоді за спроби компіляції програми на її початку з'явиться напис, що починається словом **Error**, а курсор перебуватиме десь неподалік від місця помилки (у дійсності він може бути як вище, так і нижче за текстом). Натисніть будь-яку клавішу, і це поверне вікну редактора звичний вигляд.

Закінчення роботи з програмою. Закрити вікно редактора — <Alt+F3>. Закінчити роботу із системою — <Alt+X>.

Поновлення роботи. Якщо треба починати роботу з новою програмою, можна відкрити нове вікно за допомогою <F3> (одночасно можна відкрити до десяти вікон). Для роботи з існуючою програмою в головному меню оберемо пункт **File**, а в ньому — пункт **Open** (відкрити). У вікні, що відкриється, виберіть потрібний файл з програмою.

Вікно виведення. Щоб постійно бачити вікно виведення, в головному меню оберіть пункт **Debug** (наладка), а у вертикальному меню — пункт **Output** (виведення). Перехід між цими вікнами (і не лише цими) задається клавішею <F6>. При цьому активне вікно виділяється подвійною рамкою.

Зміна розмірів вікон. Розміри вікон можна змінити та розташувати на екрані так, щоб їх було видно одночасно. Натиснемо <Ctrl+F5> та перейдемо до режиму зміни параметрів активного вікна. Його рамка дещо змінюється, клавіші-стрілки дозволяють зсувати вікно по екрану, ті ж стрілки при натиснутій <Shift> — змінювати форму вікна, <Enter> — закінчити зміни та вийти з цього режиму, а <Esc> — відмовитися від змін. У режимі зміни ці клавіші вказуються в нижньому рядку-підказці.

Кожне вікно можна збільшити, щоб воно займало весь екран. Для цього натисніть <F5>. Та ж клавіша поверне вікно до попереднього розміру.

Вікно налаштування. Наберемо програму з прикладу про температуру (див. параграф 11 розділу 3 першої частини посібника). Перейдемо до вікна редактора з текстом програми та натиснемо <Ctrl+F7> — з'явиться рамка з написом **Add Watch** (додати елемент перегляду), і у віконці рамки наберемо ім'я **tCels**, яке є у програмі. Ці дії повторимо з іменем **tKelv**. За допомогою F6 переключимося на нове вікно з написом **Watches** і побачимо там (після виконання нашої програми) слова **tCels** і **tKelv**, а поруч з ними уведене число та суму його з 273.

Нарешті, змусимо систему показати покрокове виконання програми. Натиснемо <Ctrl+F2> (програма буде виконуватися спочатку), а потім кілька разів натиснемо <F7> або <F8> (яку саме, поки що байдуже) й подивимося на зміни, що відбуваються у вікнах редактора, виведення та наладки. Зокрема, у вікні редактора виділяється кольором рядок програми, який виконується на відповідному кроці, а у вікні наладки поруч із словами **tCels** і **tKelv** спочатку з'являються нулі, а потім уведене число та його сума з 273.

- Покрокове виконання програми — це, по суті, її ***інтерпретація***.

ГОЛОВНЕ МЕНЮ ТА ДЕЯКІ ЙОГО ПІДМЕНЮ

Вхід у головне меню — <F10>, вихід — <Esc>. Розглянемо деякі з можливостей меню та підменю, які можуть виявитися корисними на початку використання середовища. Підменю: **File**, **Edit**, **Search**, **Run**, **Compile**, **Debug**, **Tools**, **Options**, **Window**, **Help**. Вони позначають меню, які містять команди та підменю.

МЕНЮ FILE (РОБОТА З ФАЙЛАМИ)

New — створити нове вікно редактора.

Open... (праворуч <F3>) — відкрити вікно редактора для роботи з існуючим файлом. Каталог і файл у ньому вказують у діалоговому вікні.

Save (<F2>) — зберегти буфер активного вікна редактора у файлі, з яким зв'язано вікно.

Save as... — зберегти буфер активного вікна редактора у файлі, який буде вказано далі.

Save All — зберегти буфери всіх активних вікон у файлах, пов'язаних із вікнами.

Change Dir... — установити поточний каталог.

Print — надрукувати буфер активного вікна.

Exit (<Alt+X>) — закінчити роботу із середовищем Турбо Паскаль.

МЕНЮ EDIT (РОБОТА З РЕДАКТОРОМ)

Undo (<Alt+BkSp>) — скасувати останню команду редагування.

Redo — скасувати останню команду **Undo**.

Cut (<Shift+Del>) — перемістити виділений текст з вікна редактора до буфера обміну.

Copy (<Ctrl+Ins>) — скопіювати виділений текст в буфер обміну.

Paste (<Shift+Ins>) — уставити вміст буфера обміну у вікно редактора.

Clear (<Ctrl+Del>) — вилучити виділений текст.

Show clipboard — відкрити вікно буфера обміну й зробити його активним.

ÃÂÌ¸ Search (ÔÓ□ÛÍ ≥ Á‡ÏÏ≥Ì‡ ÚÂÍÒÚÛ)

Find... — знайти в тексті рядок, який задають далі в діалоговому вікні.

Replace... — знайти й замінити рядок.

Search again — повторити останню операцію пошуку або заміни.

Go to line number... — перемістити курсор у вказаний рядок.

МЕНЮ RUN (ВИКОНАННЯ ТА ІНТЕРПРЕТАЦІЯ)

Run (<Ctrl+F9>) — виконати програму в активному вікні.

Step over (<F8>) — виконати оператори в наступному рядку тексту (підпрограми виконуються як єдине ціле, тобто за один крок інтерпретації).

Trace into (<F7>) — виконати оператори в наступному рядку тексту, включаючи оператори в підпрограмах.

Go to cursor (<F4>) — виконати частину програми до рядка з курсором.

Program reset (<Ctrl+F2>) — завершити інтерпретацію та звільнити пам'ять, яку займає програма.

МЕНЮ COMPILE (КОМПІЛЯЦІЯ ТА КОМПОНУВАННЯ ПРОГРАМ)

Compile (<Alt+F9>) — компілювати програму в активному вікні.

Destination (**Memory** або **Disk**) — указати, куди записується відкомпільована програма чи модуль. Якщо праворуч вказано слово **Memory** (пам'ять), код завантажується й виконується, а якщо **Disk** — записується в exe-файл. Вибір цієї опції змінює напрямок коду.

Information... — вивести інформацію про програму (розмір коду, даних, програмного стека, купи тощо).

МЕНЮ DEBUG (НАЛАДКА ПРОГРАМ)

Call Stack (<Ctrl+F3>) — вивести список викликаних підпрограм (програму вказано внизу, активну підпрограму — згори).

Watch — вивести вікно стану пам'яті зі значеннями змінних і виразів, указаних за допомогою опції **Add watch** (див. нижче).

Output — зробити активним вікно виведення програми.

User screen (<Alt+F5>) — зробити активним вікно виведення програми, розгорнувши його на весь екран.

Evaluate/Modify... (<Ctrl+F4>) — обчислити значення виразу або задати значення змінної. Цю опцію використовують, якщо програма перебуває в режимі наладки. Діалогове вікно містить три поля (ліворуч) і чотири кнопки (праворуч). У полі **Expression** можна ввести будь-який вираз, зокрема, ім'я змінної. Після натискання кнопки **Evaluate** або <Enter> у полі **Result** з'явиться значення виразу при поточному стані пам'яті програми; якщо хоча б одне ім'я виразу не означено, ви побачите повідомлення **Unknown identifier** (невідомий ідентифікатор).

Якщо в першому полі введено ім'я скалярної змінної, її значення можна змінити, увівши його в поле **New value** і натиснувши <Enter>.

При виклику цієї опції середовище аналізує текст, який указано курсором. Якщо це ім'я змінної або константи, воно з'являється у вікні **Expression** автоматично.

Add watch... (<Ctrl+F7>) — додати змінні або вирази, значення яких показуються у вікні стану пам'яті (опція **Watch**). Ці значення називаються контрольними.

МЕНЮ OPTIONS (НАСТРОЮВАННЯ ІНТЕГРОВАНОГО СЕРЕДОВИЩА)

Compiler... — установити режими роботи компілятора, які використовуються за узгодженням. Діалогове вікно має 17 полів — вони вказують режими, які вмикаються або вимикаються за допомогою клавіші пропуску або одиночного щиголя мишею. Докладну інформацію про режими можна одержати за допомогою довідкової системи середовища (при виділеному режимі натиснути <F1>).

Memory sizes... — задати розмір програмного стека (не більше 65535 байт), мінімальний та максимальний розмір купи (не більше 655360 байт).

Environment — установити режими роботи інтегрованого середовища, редактора й миші. Для цього використовують п'ять діалогових вікон.

У вікні **Preferences** установлюють, зокрема, кількість рядків текстового екрана (25, 43 або 50) та режими автоматичного зберігання інформації перед запуском програми. Корисна опція зберігання файлів **Editor files**, особливо при виконанні програми, яка може «зависнути».

У вікні **Editor** установлюють режими роботи редактора системи (перейменування попередньої версії вхідного файла в BAK-файл, вставка або заміна символів при введенні, виділення операторів відступами, використання символів і клавіші табуляції тощо).

За допомогою вікна **Mouse** указують спосіб використання правої кнопки миші, а вікна **Startup** — особливості роботи з графічною інформацією, системною бібліотекою **SYSTEM.TPU** та оперативною пам'яттю редактора. Вікно **Colors** дозволяє встановити колірну палітру окремих елементів середовища (полів, меню й вікон).

Open... — указати ім'я файла конфігурації, в якому зберігається настроювання середовища.

Save — зберегти поточне настроювання середовища у файлі конфігурації (за узгодженням це файл **Turbo.tp** у каталозі з системою Turbo Pascal — файлом **Turbo.exe**).

Save as... — указати каталог і файл, в якому буде збережено поточне настроювання.

МЕНЮ WINDOW (КЕРУВАННЯ ВІКНАМИ)

Tile — розташувати вікна так, щоб вони не накладалися й мали приблизно однакові розміри.

Cascade — розташувати вікна так, щоб було видно рамки кожного з них (це зручно при використанні миші).

Close all — закрити всі вікна, відкриті в середовищі.

Refresh display — поновити екран (вилучити сліди виведення програми).

Size/Move (<Ctrl+F5>) — змінити розташування й розмір активного вікна.

Zoom (<F5>) — розгорнути активне вікно на весь екран. При повторному натисканні розміри вікна відновлюються.

Next (<F6>) — активізувати наступне вікно.

Previous (<Shift+F6>) — активізувати попереднє вікно.

Close (<Alt+F3>) — закрити активне вікно.

List... (<Alt+0>) — вивести список усіх відкритих вікон середовища.

МЕНЮ HELP (ДІАЛОГОВА ДОВІДКОВА СИСТЕМА)

У повідомленнях довідкової служби всі посилання на терміни, для яких існує окрема довідкова інформація, виділено кольором. Для одержання цієї інформації до посилання можна підвести курсор і натиснути <Enter> або клацнути на ньому лівою кнопкою миші.

Contents — відобразити зміст довідкової системи.

Index (<Shift+F1>) — вивести повний алфавітний список усіх посилань довідкової системи.

Topic search (<Ctrl+F1>) — видати довідку з терміну, відміченого курсором (ключове слово, ім'я системної підпрограми тощо).

Previous topic (<Alt+F1>) — вивести попереднє довідкове повідомлення.

Compiler directives — видати довідку про директиви компілятора.

Reserved words — видати довідку про зарезервовані слова.

Standard units — видати довідку про стандартні модулі.

Turbo Pascal language — видати довідку про основні елементи мови (з подальшим їх уточненням).

Error messages — видати довідку щодо повідомлень про помилки (компіляції та часу виконання).

About... — вивести інформацію про авторські права та версію середовища Турбо Паскаль.

ДЕЯКІ СЛУЖБОВІ СЛОВА МОВИ TURBO PASCAL

Зарезервовані слова як імена не оголошуються. Стандартні директиви та стандартні ідентифікатори оголошувати можна, але не рекомендується.

Зарезервовані слова

and	exports	mod	shr
asm	file	nil	string
array	for	not	then
begin	function	object	to
case	goto	of	type
const	if	or	unit
constructor	implementation	packed	until
destructor	in	procedure	uses
div	inherited	program	var
do	inline	record	while
downto	interface	repeat	with
else	label	set	xor
end	library	shl	

Деякі стандартні ідентифікатори

Модулі системи Турбо Паскаль: **Crt**, **Dos**, **Graph**, **Overlay**, **Printer**, **System**, **Graph3**, **Turbo3**.

Константи: **false**, **true**, **maxInt**, **maxLongInt**.

Типи: **boolean**, **byte**, **char**, **comp**, **double**, **extended**, **integer**, **longint**, **real**, **shortint**, **single**, **text**, **word**.

Файли: **input**, **output**.

Деякі функції: **abs**, **addr**, **arctan**, **chr**, **copy**, **cos**, **eof**, **eoln**, **exp**, **int**, **ln**, **length**, **odd**, **ord**, **pi**, **sqr**, **pos**, **pred**, **random**, **round**, **sin**, **sqrt**, **succ**, **trunc**.

Деякі процедури: **assign**, **break**, **close**, **continue**, **dec**, **delete**, **dispose**, **erase**, **exit**, **freemem**, **getmem**, **halt**, **inc**, **insert**, **new**, **randomize**, **read**, **readln**, **release**, **reset**, **rewrite**, **seek**, **str**, **val**, **write**, **writeln**.

ДИРЕКТИВИ КОМПІЛЯТОРА TURBO PASCAL

Список директив записують у дужках **{}** через кому, наприклад, **{$A+,$P-,$R+}**. Існує три види директив: перемикачі, умовні та параметричні.

Директиви-перемикачі впливають на режими, які вказано в діалоговому вікні **OPTIONS/COMPILER**. Їх задають однією буквою та знаком **+** або **–** (увімкнення відповідного режиму компіляції або його вимкнення). Деякі директиви-перемикачі є ***локальними***; їх можна записати в будь-якому місці тексту програми (модуля), і вони діють

до його кінця або до появи директив, протилежних їм. ***Глобальні*** директиви записуються на початку тексту й діють на весь текст.

Деякі директиви-перемикачі наведено в таблиці. Ліворуч записано стан директив за узгодженням. Знаком ***** відмічено локальні директиви. У дужках вказано дію протилежної директиви (зі знаком **+**, якщо за узгодженням діє **–**, і навпаки).

Директива ***Дія директиви***

{$A+} Вирівнювати дані на межу слова (байта).

{$B–}* Застосовувати «ледачі» обчислення булевих виразів (обчислювати їх повністю)

{$D+} Використовувати (не використовувати) вбудований налагоджувач.

{$E+} Увімкнути (вимкнути) режим програмної емуляції співпроцесора 80x87 (якщо співпроцесор відсутній).

{$F–} Використовувати ближню (дальню) модель виклику підпрограм.

{$G–}* При генерації коду не використовувати (використовувати) повний набір інструкцій процесора 80286.

{$I+}* Увімкнути (вимкнути) автоматичний контроль операцій введення-виведення.

{$L+} Включати (не включати) локальні символи в інформацію для налагоджувача.

{$N–} Операції з нефіксованою крапкою реалізувати програмно (використовувати числовий співпроцессор, завдяки якому будуть доступні додаткові дійсні типи).

{$Q–} Заборонити (дозволити) перевірку цілочисельних операцій на переповнення.

{$R–}* Заборонити (дозволити) контроль меж діапазонів.

{$S+}* Генерувати (не генерувати) спеціальний код на початку кожної підпрограми, який перевіряє можливе переповнення програмного стека.

{$T–} Результат операції **@** обробляти як нетипізований (типізований) вказівник.

{$V+}* Увімкнути (вимкнути) контроль довжини рядкових аргументів у викликах підпрограм.

{$X+} Використовувати (не використовувати) розширений синтаксис, який дозволяє записувати виклики функцій як оператори.

В ***умовних директивах*** використовуються ***умовні символи*** — імена, на які не може бути посилань у програмі. У директиві **{$DEFINE**

***умовний символ*}** задають установку умовного символу (він буде істинним), а в **{$UNDEF *умовний символ*}** — скасування установки. У директиві вигляду **{$IFDEF *умовний символ*}** перевіряється, чи встановлено символ. Якщо встановлено, то весь фрагмент програми між цією директивою та найближчою директивою **{$ELSE}** або **{$ENDIF}** буде компілюватися, а якщо не встановлено — він не компілюється. Якщо нижче записано директиву **{$ELSE}**, то при невстановленому символі фрагмент програми між нею та **{$ENDIF}** буде компілюватися, а при встановленому — не буде.

Директива **{$IFNDEF *умовний символ*}**, навпаки, задає компіляцію наступного за нею фрагмента, якщо символ не встановлено. Якщо нижче записано **{$ELSE}**, то при встановленому символі буде компілюватися фрагмент програми між нею та **{$ENDIF}**.

Умовні директиви можна використовувати разом із операторами наладки, наприклад, у такий спосіб.

```
{$IFDEF Debug}
  оператори наладки
{$ENDIF}
```

Якщо за допомогою директиви **{$DEFINE}** встановити умовний символ **Debug**, то оператори наладки буде відкомпільовано й виконано. По закінченні наладки установку **Debug** можна зняти й ще раз відкомпілювати програму — операторів наладки в машинній програмі не буде.

Розглянемо також дві ***параметричні директиви***. Глобальна директива вигляду **{$M *стек,низ,верх*}** задає розмір програмного стека в байтах, а також нижню та верхню межу адрес динамічної пам'яті, наприклад, **{$M 16384,0,655360}**. Між буквою **M** та першою константою записують хоча б один пропуск.

Директива вигляду **{$I *ім'я файла*}** задає включення файла. Файл має містити фрагмент тексту Паскаль-програми. Якщо в директиві не задано розширення файла, то шукається файл із цим іменем і розширенням .pas. За цією директивою компілятор «перемикається» на компіляцію тексту у файлі, а по його закінченні продовжує компіляцію тексту після цієї директиви.

КОДУВАННЯ СИМВОЛІВ

У першій таблиці представлено відповідність між номерами від 0 до 127 та символами, зафіксовану в стандарті ASCII (*American Standard*

Code for Information Interchange — Американський стандартний код для обміну інформацією). Символи з номерами від 0 до 31 мають спеціальне призначення, яке в цій книжці не описано. Замість їх зображення вказано мнемонічне позначення. Символ з номером 32 — пропуск.

Основна таблиця ASCII

0 NUL	16 DLE	32	48 0	64 @	80 P	96 `	112 p
1 SOX	17 DC1	33 !	49 1	65 A	81 Q	97 a	113 q
2 STX	18 DC2	34 "	50 2	66 B	82 R	98 b	114 r
3 ETX	19 DC3	35 #	51 3	67 C	83 S	99 c	115 s
4 EQT	20 DC4	36 $	52 4	68 D	84 T	100 d	116 t
5 ENQ	21 NAC	37 %	53 5	69 E	85 U	101 e	117 u
6 BEL	22 SYN	38 &	54 6	70 F	86 V	102 f	118 v
7 ACK	23 ETB	39 ë	55 7	71 G	87 W	103 g	119 w
8 BS	24 CAN	40 (	56 8	72 H	88 X	104 h	120 x
9 TAB	25 EM	41)	57 9	73 I	89 Y	105 i	121 y
10 LF	26 SUB	42 *	58 :	74 J	90 Z	106 j	122 z
11 VT	27 ESC	43 +	59 ;	75 K	91 [	107 k	123 {
12 FF	28 FS	44 ,	60 <	76 L	92 \	108 l	124 |
13 CR	29 GS	45 -	61 =	77 M	93]	109 m	125 }
14 SO	30 RS	46 .	62 >	78 N	94 ^	110 n	126 ~
15 SI	31 US	47 /	63 ?	79 O	95 _	111 o	127 □

Для номерів 128–255 існують різні відповідності (кодові таблиці). Нижче наведено одну з них.

Варіант кодової таблиці

128 А	144 Р	160 а	176 ░	192 └	208 ╨	224 р	240 Ё
129 Б	145 С	161 б	177 ▒	193 ┴	209 ╤	225 с	241 ё
130 В	146 Т	162 в	178 ▓	194 ┬	210 ╥	226 т	242 Є
131 Г	147 У	163 г	179 │	195 ├	211 ╙	227 у	243 є
132 Д	148 Ф	164 д	180 ┤	196 ─	212 ╘	228 ф	244 І
133 Е	149 Х	165 е	181 ╡	197 ┼	213 ╒	229 х	245 і
134 Ж	150 Ц	166 ж	182 ╢	198 ╞	214 ╓	230 ц	246 Ї
135 З	151 Ч	167 з	183 ╖	199 ╟	215 ╫	231 ч	247 ї
136 И	152 Ш	168 и	184 ╕	200 ╚	216 ╪	232 ш	248 °
137 Й	153 Щ	169 й	185 ╣	201 ╔	217 ┘	233 щ	249 •
138 К	154 Ъ	170 к	186 ║	202 ╩	218 ┌	234 ъ	250 ·
139 Л	155 Ы	171 л	187 ╗	203 ╦	219 █	235 ы	251 √
140 М	156 Ь	172 м	188 ╝	204 ╠	220 ▄	236 ь	252 №
141 Н	157 Э	173 н	189 ╜	205 ═	221 ▌	237 э	253 ¤
142 О	158 Ю	174 о	190 ╛	206 ╬	222 ▐	238 ю	254 ■
143 П	159 Я	175 п	191 ┐	207 ╧	223 ▀	239 я	255

Науково-виробниче видання

Бібліотека «Шкільного світу»

***Ставровський** Андрій, **Скляр** Ірина*

Програмуємо правильно
Посібник
Частина 2

Художній редактор *О. Голик*
Літературний редактор *Ю. Желєзна*
Коректор *І. Бірюкович*
Комп'ютерна верстка *К. Яскевич*
Набір *Т. Корольова*

Підписано до друку 02.07.07. Формат 60x84/16.
Папір офсетний № 1. Гарнітура Таймс. Друк офсетний.
Умовн. друк. арк. 7,44. Обл.-вид. арк. 7,2. Тираж пр.
Зам. №

ТОВ Видавництво «Шкільний світ»
01014, м.Київ, вул. Тимірязєвська, 2
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
серія ДК № 775 від 21.01.2002 р.

Видрукувано з готових діапозитивів в ОП «Житомирська облдрукарня»
10014, Житомир, вул. Мала Бердичівська, 17
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
серія ЖТ № 1 від 06.04.2001 р.