Розв'язання олімпіадних задач з програмування

Цей навчальний посібник буде корисний в першу чергу усім, хто цікавиться програмуванням, складанням алгоритмів та хоче поглибити свої знання і вміння в цій галузі. Всі розглядувані теми супроводжуються багатьма ілюстраціями, блок-схемами алгоритмів, та їх реалізацією мовою Паскаль.

Навчальний посібник може використовуватися в якості довідника, так як йому притаманна чітка структура й наявність додаткового матеріалу та безпосередніх посилань на джерела.

Рекомендовано школярам 7-11 класів, тим хто цікавиться програмуванням, зокрема, олімпіадним програмуванням, вчителям для підготовки уроків, шкільних олімпіад, а також, для поглиблення знань при написанні науково-дослідних робіт з інформатики в секціях МАН.

ЗМІСТ

ВСТУП

При написанні науково-дослідних робіт з інформатики на секціях МАН учні досліджують різні проблеми прикладної математики, які здебільшого потребують наявність ґрунтовної бази знань як з математики, так і з інформатики, а саме — програмування. Для вирішення нетривіальних задач з програмування, шкільного курсу — недостатньо. Тому, учням, членам МАН потрібно постійно поглиблювати свій рівень знань в цій галузі.

Програмування — один з предметів, в якому в повній мірі відчувається вплив практики на розуміння матеріалу. Тобто, для якісного засвоєння навчального матеріалу потрібно постійно тренуватися — розробляти алгоритми, писати програми, тестувати, аналізувати отримані результати, модифікувати існуючі алгоритми, займатися оптимізацією та інше.

Часто зустрічаються зовсім різні по своїй суті й по галузі застосування задачі, які з точки зору програмування передбачають роботу з однаковими структурами даних, або ж, взагалі, мають подібні алгоритми розв'язання. Постійна практика розв'язання задач з програмування натреновує учня таким чином, що він починає якісно класифікувати різні задачі, бачити спільні та відмінні елементи, одразу ж вносити можливі покращення в існуючі алгоритми розв'язання.

В шкільних закладах проводиться багато олімпіад з різних предметів. Та олімпіада з програмування — це олімпіада, в якій найкраще можна перевірити логіку, кмітливість, здатність до аналізу, синтезу, дедуктивне й індуктивне мислення конкурсантів. Саме ці якості є невід'ємними у молодих дослідників.

Олімпіада, як засіб навчання для молодих програмістів є гарною нагодою перевірити свій рівень знань, підготуватися, розширити свій кругозір, ознайомитися з новими технологіями, підходами в програмуванні, і надалі, використовувати отримані знання та навички при написанні науково-дослідницьких робіт в МАН.

На обласному рівні олімпіада з програмування проводиться в два тури. Зазвичай на кожному з двох турів учасникам пропонується розв'язати та запрограмувати три задачі за п'ять годин. Кожен учасник розв'язує задачі самостійно,

користуючись одним комп'ютером на якому є встановлене необхідне програмне забезпечення для написання, тестування та відлагодження програм (середовище розробки[1], компілятор[2], дебагер[3]).

Суворо забороняється спілкування з іншими учасниками, використання літератури, інтернету й інших матеріалів. Для розв'язання задачі необхідно написати програму мовою *Pascal* чи *C++*, і здати її представнику журі по закінченні туру. Після завершення змагань програма оцінюється шляхом автоматичного запуску її на наборі секретних тестів (зазвичай у кількості 10—20 тестів). Учасник отримує бали за кожен тест, з яким у відведений час (наприклад, 1—2 секунди) і використовуючи обмежений об'єм пам'яті (наприклад, не більше 64 МБ) впоралася написана ним програма.

Отримані за два тури змагань бали сумуються по кожному з учасників, підсумковий результат учасника цілком визначає успіх його виступу. Залежно від результату учень, що брав участь в олімпіаді, може бути нагороджений дипломом першого, другого чи третього ступеня (а може й залишитися ненагородженим). Учасники з усіх паралелей (8—11-ті класи) розв'язують однакові задачі, але присудження призових місць відбувається для кожної паралелі окремо. Кількість призерів у кожній з паралелей, згідно з положеннями олімпіади, не має перевищувати 30% від загальної кількості учасників з цієї паралелі. 10—20 учасників, чиї виступи виявилися найбільш вдалими, потрапляють на тренувально-відбіркові збори для підготовки й остаточного відбору для участі у Всеукраїнській олімпіаді з інформатики.

Та окрім олімпіад, які проводяться Міністерством освіти і науки України, на які не кожен має можливість потрапити, існує багато не менш цікавих змагань в

1 **Середовище розробки (IDE) —** від *Integrated Development Environment* (також можливі інтерпретації *Integrated Design Environment* — інтегроване середовищепроектування; чи *Integrated Debugging Environment* — інтегроване середовище відлагодження) — система програмних засобів, що використовується програмістом для розробки програмного забезпечення. Приклади IDE: *Borland Pascal 7.1, Borland Delphi 5* та інші.

2 **Компілятор** (англ. *Compiler* від англ. *to compile* збирати в ціле) — комп'ютерна програма (або набір програм), що перетворює (компілює) програмний код, написаний певноюмовою програмування (наприклад *Pascal, C, C++,* та ін.), на код необхідний для виконання програми комп'ютером.

3 **Дебагер (**частіше використовується термін *відлагоджувач* або *відладчик*, також часто користуються англ. *дебагер*,англ. *debugger* — це комп'ютерна програма, яка використовується для тестування і відлагодження інших програм. Як варіант, код для розгляду може бути запущено на емуляторі інструкцій, що дає більший контроль зупинки процесу при заданих умовах, але, як правило, тоді виконання коду відбувається набагато повільніше, ніж якщо це робиться напряму на процесорі. Інакше кажучи, дебагер — це програма, за допомогою якої можна взнати, чому написана вами програма веде себе не так, як було задумано[1].

мережі Інтернет:

- **онлайн інтернет-олімпіади, які відбуваються в режимі реального часу**
  - ***http://neerc.ifmo.ru/school/io/rules-team.html*** — Олімпіади з інформатики, Санкт-Петербург, Росія (рос.)
- **заочна інтернет-олімпіада**
  - ***http://olympiada.km.ua/info/ol_09_10/index.html*** — Відкрита заочна олімпіада з програмування 2009-2010 н.р. серед учнів навчальних закладів Хмельницької області
  - http://vippo.byethost12.com/index.php?pagename=olimpiada заочна олімпіада з інформатики, що організовується Волинським інститутом післядипломної освіти
- **олімпіадні сервери**
  - ***http://www.olymp.vinnica.ua*** — Центр підтримки та проведення олімпіад школярів з використанням можливостей INTERNET
  - ***http://acm.lviv.ua*** — **ACM-Контестер** Український портал ACM-спільноти
  - ***http://e-olimp.com*** — Інтернет-портал організаційно-методичного забезпечення дистанційних олімпіад з програмування для обдарованої молоді навчальних закладів України
  - ***http://acm.timus.ru*** — **Timus Online Judge**, архив задач с проверяющей системой (рос.)

  та багато інших.

При чому, для участі у цих змаганнях потрібно лиш вихід в інтернет і бажання!

Зазвичай на серверах задачі класифіковані по рівням складності, а отже, будь-хто, від початківця до професіоналу, зможе з легкістю знайти задачу, яка відповідала б як його інтересам, так і рівню-майстерності програмування.

Як відомо, *кількість переростає у якість*. Постійно тренуючись, розв'язуючи задачі з програмування, принаймні з тих серверів, що вище перелічені — можна спершу “набити руку” на олімпіадних задачах, і надалі, виконуючи наукове дослідження — поділити його на підзадачі. Складові частини дослідження зазвичай

мають своє відображення в олімпіадних задачах. А отже, якщо учень неоднократно розв'язував задачі певного класу — то він без проблем зорієнтується в методах розв'язання.

РОЗДІЛ 1. НЕОБХІДНИЙ МІНІМУМ ЗНАНЬ

1.1. Особливості ACM-олімпіад

На олімпіадних серверах встановлене спеціальне програмне забезпечення, яке в режимі реального часу здійснює перевірку відправленого розв'язку й одразу видає результат.

Олімпіадне програмування передбачає, здебільшого, розв'язання задач, принцип дії яких можна зобразити наступною схемою:

Схема роботи програми

Суть — розробити саме блок “Програма”, який би для будь-яких наперід означених вхідних даних давав би *коректні* вихідні дані (для програми, що нормально скомпілювалася й запустилася). Взагалі кажучи, роботу тестової системи можна зобразити такою схемою:

Схема роботи тестової системи

Результат тестування — або ТАК, або НІ. Тобто програма на заданому тесті може його або ПРОЙТИ або НЕ ПРОЙТИ.

Автоматизована тестувальна система

Всі тести виконуються в одному і тому ж порядку по зростанню складності. Зазвичай перші тести співпадають з прикладами умови задачі. Номер тесту, на якому сталася помилка вказується в результаті перевірки розв'язку — і є номер першого тесту, який не пройшов перевірку. Наприклад, якщо тестова система видала повідомлення *Time limit, 9*[4]*,* то це означає, що

1. Тести з 1 по 8 успішно пройшли перевірку.
2. На більш складному ніж попередні вісім тестів, ваша програма паревищила відведений їй час роботи.

В даному випадку, можна стверджувати, що алгоритм частково застосовний, проте, загалом не є **коректним**.

Кажуть, що алгоритм **коректний**, якщо для будь-яких вхідних даних результатом його роботи є коректні вихідні дані. Ми говоримо, що коректний алгоритм **вирішує** конкретну обчислювальну задачу. Якщо алгоритм некоректний, то для деяких вхідних даних він може взагалі не завершити свою роботу, або видати відповідь, відмінну від очікуваного.[2]

Типові помилки при здачі задач

Усі автоматичні тестові системи працюють за подібним принципом. Помилки, які зазвичай розпізнає тестова система (за [3]):

- **Помилка компіляції (Compilation Error, CE)**

  Це може означати що допущена синтаксична помилка в тексті програми і система не змогла скомпілювати ваш розв'язок. Або надісланий на перевірку розв’язок не відповідає стандарту компілятора, який використовується у системі. Також, можливий випадок, коли тестова система підтримує не один компілятор, а декілька, й при надсиланні розв'язку ви вказали тип компілятора відмінний від того, на який ви розраховували.

4 Можливі типи помилок розглядаються далі

- **Помилка часу виконання (Runtime Error, RE)**

  Це може означати, що під час виконання програми відбулася критична помилка, при якій подальше виконання неможливе. Типовими помилками можуть бути: ділення на 0 (нуль), доступ до неіснуючого елементу у масиві. Також ця помилка може виникати у випадку, якщо код завершення програми не дорівнює нулю[5].

- **Неправильна відповідь (Wrong Answer, WA)**

  Це може означати, що Ваша програма принаймні на одному з тестів дала неправильну відповідь, або формат виведення не відповідає умові. Тобто, це означає, що варто ретельно перечитувати умову задачі, а саме — формат виведення даних, так як зазвичай, тестова система сприйме різними наступні відповіді: *"*`1_2`*"* та *"*`1____2`*"*.

  Також, варто слідкувати за тим, в якому регістрі (та якою саме мовою повинен виводитися результат), тобто "`YES`" та "`Yes`" будуть різними відповідями.

- **Ліміт часу (Time limit, TL)**

  Це може означати, що Ваша програма не вклалася у відведений для її роботи час. Цей ліміт подається в умові задачі і рахується для кожного тесту окремо. Принципи підрахунку часу роботи програми в різних тестових систем можуть відрізнятися. На ***www.acm.lviv.ua*** час виконання програми – це загальний час від моменту запуску програми і аж до її повного завершення.

  Виникнення помилки такого типу може свідчити здебільшого про те, що у вашій програмі використано неоптимальний алгоритм, тобто, алгоритм роботи програми потребує оптимізації — редагування алгоритму з метою спрощення, що б передбачало "відкидання" зайвих обчислень, їх спрощення, виконання попередніх обрахунків перед основним циклом, і інше. Зазвичай, для того, щоб здійснювати оптимізацію написаного алгоритму, потрібно досконало знати й розуміти принцип роботи

5 Ті хто проорамує на мові *Pascal*, не стикаються з поняттям коду *завершення програми*. В програмах написаних на *C/C++* обов'язково кінцевим оператором функції `main()` повинен бути `return 0;`

алгоритму. Нерідко це потребує грунтовних математичних знань, чи знання й уміння застосовувати на практиці раніше вивчені алгоритми чи структури даних.

Нерідко, причиною виникнення помилки цього типу може бути очікування введення даних (наприклад, зайвий оператор `readln`, що був поставлений вкінці програми — з метою "затримки" екрану по завершенню роботи програми)

- **Ліміт пам'яті (Memory limit, ML)**

  Це означає, що ваша програма використовує більше пам'яті, ніж було для неї відведено. Обмеження пам'яті зазвичай вказуються в умові задачі (Наприклад, *64KB*). Для того, щоб уникнути помилки ліміту пам'яті варто робити принаймні орієнтовні підрахунки по використанню пам'яті — на етапі написання програми ви повинні знати які типи та структури даних ви використовуєте в своїй програмі. Приблизне підрахування обсягу пам'яті, який потрібен для розташування всіх даних програми дасть можливість відслідкувати цю помилку. Наприклад, змінна типу `byte` займає один байт пам'яті[6]. Таким чином, структура — масив зі 100 змінних типу `byte` займатиме в пам'яті `100` байт.

- **Ліміт виведення (Output limit, OL)**

  Результат роботи вашого розв'язку, крім того що є неправильним, є занадто великим по розміру. Відведений об'єм результату не подається, але він завжди більший очікуваного результату. Зазвичай це може означати про хибність алгоритму, зокрема, про його зациклення.

- **Заборонена функція (DF)**

  При написанні програми Ви використали функцію, яка є забороненою. Забороняється використовувати додаткові модулі, бібліотеки, експортувати зовнішні функції, використовувати ассемблерні вставки, користуватися файлами (якщо інше не зазначено в умові), викликати функції операційної системи.

6 Об'єми пам'яті під змінні різних типів наводяться далі.

Варто також мати на увазі, що якщо Ви отримали повідомлення про помилку *"WA"* або *"OL"*на першому тесті, то це означає здебільшого те, що виведення результату роботи алгоритму запрограмовано неправильно. І перш, ніж заново переписувати алгоритм — варто перевірити процедури виведення результатів і їх відповідність вимогам умови задачі та коректність заповнення всіх форм при надсиланні розв'язку.

1.2. Мови програмування

Скільки ти мов знаєш — стільки разів ти людина (народна мудрість) — *та це не стосується програмістів :)*

Тестові сервери підтримують переважно по кілька мов програмування. Це сповна логічно, тому що саме задача визначає ту мову програмування, розв'язок якою її легше і якісніше можна запрограмувати.

Приклади використання мов програмування на олімпіадних тестових серверах[7]:

acm.lviv.ua	**C++**: **Pascal**:	VC++ (Express edition) Delphi7
olymp.vinnica.ua	**C/C++, Pascal, Python**	
e-olimp.com	**C++**: **Pascal**: **Java**:	GNU C++, VC++ 6.0, VC++ 9.0; Delphi 7.0, Free Pascal; JDK 1.6.0
acm.timus.ru	**C++**: **C#:** **Pascal**: **Java**:	Intel C++ Compiler 7.0 (в режимі сумісності з Microsoft Visual Studio 7.0) Microsoft Visual C# 2008 Free Pascal 2.0.4. (в режимі сумісності з Borland Delphi) J2SE Development Kit (JDK) 6.0 Update 11

Потрібно чітко знати який компілятор використовує тестова система, щоб

7 Дані взято з довідкових матеріалів по користуванню відповідними вказаними серверами.

передбачити можливі помилки, так як компілятори мають свої особливості. Знання цих особливостей приходить в процесі постійного програмування цими засобами.

Спираючись на те, що в більшості загальноосвітніх шкіл, в якості мови програмування, що вивчається на уроках інформатики обрано мову *Паскаль* — далі всі програмні реалізації структур, алгоритмів, програм будуть наводитися саме мовою Паскаль. В подальших прикладах не буде використовуватися тонких особливостей цієї мови, а отже читач без зайвих труднощів зможе переписати вихідні коди мовою Паскаль — на іншу мову програмування.

Усі вихідні коди програм, що зустрічаються в цих методичних рекомендаціях були протестовані компілятором Free Pascal 2.4.0[8] й без проблем компілюються та коректно виконуються.

1.3. Базові структури даних та їх програмування

Стандартні та користувацькі типи даних

Поняття **ЗМІННА**. З кожною змінною програми пов'язаний один і тільки один тип. Саме цей тип характеризує діапазон значень, яких може набувати ця змінна, а також операції, які з нею можна виконувати.

В мові Паскаль є чотири стандартні типи даних: *цілий*, *дійсний*, *логічний* і *символьний*.

Цілі типи[9][4]

Назва типу	Допустимий діапазон значень	Ємність пам'яті в байтах
`Byte`	0 .. 255	1
`Shortint`	-128 .. 127	1
`Smallint`	-32768 .. 32767	2
`Word`	0 .. 65535	2
`Integer`	так як `smallint` або `longint`[10]	відповідно 2 або 4

8 **Free Pascal** (FPC) — компілятор (та середовище) мови програмування Паскаль з відкритими вихідними кодами, сумісний з *Borland Pascal 7* і *Object Pascal*, але при цьому має ряд додаткових можливостей. Дізнатися більше й скачати останню версію *FPC* можна на сайті проекту: http://www.freepascal.org.

9 Як вже говорилося раніше, тут і надалі опираємося на використання саме *Free Pascal*.

10 В 16-бітних компіляторах (*TP 1.0-6.0, TP for Windows, BP 7.0, MS Quick Pascal* та інші, переважно старі версії

`Cardinal`	те ж саме що і `longword`	4
`Longint`	-2147483648 .. 2147483647	4
`Longword`	0 .. 4294967295	4
`Int64`	-9223372036854775808 .. 9223372036854775807	8
`QWord`	0 .. 18446744073709551615	8

Операції й функції, які можна виконувати над цілими числами:

//за умови що:

```
a:=10; b:=20; c:=30;
```

+	додавання `a:=b+c;`	`a=50`
-	віднімання `a:=b-c;`	`a=-10`
*	множення `a:=b*c;`	`a=600`
div	ділення без остачі	`a:=c div b;` `a=1`
mod	остача від ділення	`a:=c mod b;` `a=10`
abs(n)	модуль числа	`a:=abs(b-c);` `a=10`
sqr(n)	квадрат числа	`a:=sqr(a);` `a=100`
pred(n)	попереднє число	`a:=pred(b);` `a=19`
succ(n)	наступне число	`a:=succ(c);` `a=31`

Дійсні типи[5]

Назва типу	Допустимий діапазон значень	Мантиса	Ємність пам'яті в байтах
`Real`	Залежно від платформи	???	4 чи 8
`Single`	1.5E-45 .. 3.4E38	7-8	4
`Double`	5.0E-324 .. 1.7E308	15-16	8
`Extended`	1.9E-4932 .. 1.1E4932	19-20	10
`Comp`	-2E64+1 .. 2E63-1	19-20	8

Варто знати й розуміти, що кожна операція з дійсними числами має похибку. Під час виконання великих програм похибка накопичується, і результат може бути

компіляторів) тип *integer* еквівалентний типу *smallint*, в той час як в 32-бітних компіляторах мови Паскаль (*GNU Pascal, Free Pascal, Delphi 7.0,* та інші) тип *integer* еквівалентний типу *longint*.

спотворений. Щоб уникнути цього варто контролювати проміжні результати обчислень.

Також, варто звернути увагу на те, що процедура `write(R:p:t)` виконує виведення у такому вигляді:

Де `p` — ширина поля (кількість усіх позицій відведених під число), t — точність (кількість знаків після крапки). Якщо число не поміщається на визначеній ширині поля, то поле автоматично розширюється.

Під час виведення, `p` і `t` варто вибирати так, щоб не виводити зайвих цифр, які не важливі для результату, або цифр які визначені неточно.

Операції й функції, які можна виконувати над дійсними числами: +, -, *, abs, sqr як і над цілими числами. Варто зауважити, що для дійсних чисел недоречно виконувати ділення з остачею, а отже, **div** та **mod** ми ніяким чином не зможемо застосувати до дійсних чисел.

```
//за умови що:
a:=10; b:=20; c:=1.5;
```

/	ділення	`a:=b/a;`	`a:=13.333...`
trunc(x)	ціла частина числа	`a:=trunc(c);`	`a=1`
round(x)	заокруглене ціле	`a:=round(c);`	`a=2`
sin(x)	синус	`a:=sin(c);`	`a=0.99749...`
cos(x)	косинус	`a:=cos(c);`	`a=0.070737...`
arctan(x)	арктангенс	`a:=atan(c);`	`a=0.98279...`

Логічний тип

Логічний тип в Паскалі представлений типом `boolean`.

Логічні змінні можуть набувати одного з двох значень: `true` (істина) або `false` (хиба).

Над змінними логічного типу можна виконувати операції:

not	логічне заперечення
or	логічне “або” (диз'юнкція)
and	логічне “і” (кон'юнкція)
xor	Додавання за модулем 2

Таблиця істинності логічних операцій:

x	y	**not** x	x **and** y	x **or** y	x **xor** y
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	TRUE	FALSE	FALSE

Символьні типи

Символьний тип в Паскалі представлений типом `char`. Значення змінної символьного типу є символ. Символ — це один елемент з таблиці ASCII, який має свій порядковий номер і визначене значення, відображення.

У програмах символьні сталі, що мають значення, записують між одинарними лапками, наприклад `'z', '~'`.

До змінних типу `char` можна застосовувати наступні функції:

ord(c)	отримати порядковий номер символу c в таблиці ASCII
chr(n)	отримати символ з номером n в таблиці ASCII

Наприклад, результатом виконання коду

```
var c1, c2 : char;
begin
    c1:='A';
    c2:=chr(ord(c1)+1);
    writeln('c1=', c1, ' c2=', c2);
end.
```

буде:

```
c1=A c2=B
```

Стрічковий тип є по суті масивом символів. В паскалі стрічковий тип наз `string`, що дає можливість в змінній зберігати набір символів (текст) довжиною до 255 символів.

Функції які виконуються над змінними стрічкового типу[6]:

`length(s)`	`S : string;`	Отримати ціле число — довжину стрічки `s`.
`concat(s1, s2, ..., sn)`	`s1, s2, ..., sn : string;`	Результат функції — стрічка, яка дорівнює злиттю n стрічок: `s1`, `s2`, ..., `sn`. Якщо довжина перевищуватиме `255`, то результуюча стрічка буде обрізана.
`copy(s, index, count)`	`s : string; index, count : integer;`	Результат функції — стрічка — частина рядка `s`, довжиною count, починаючи від символу з номером `index`.
`delete(s, index, count)`	`s : string; index, count : integer;`	Вилучає з рядка `s` частину цього рядка довжиною `count`, починаючи від символу з номером index.
`insert(what, s, index)`	`what, s : string; index : integer;`	Вставляє рядок `what` в рядок `s` починаючи з позиції `index`.
`pos(what, s)`	`what, s : string;`	Результат функції — номер першої позиції, з якої в рядку `s` розміщений рядок `what`. Якщо немає жодного входження рядка `what` в `s`, то функція повертає `0`.

Масиви та матриці

Дуже часто, а процесі вирішення різних задач ми стикаємося з наступною проблемою: нам потрібно мати кілька занумерованих змінних одного типу для їх подальшого опрацювання. Це може бути, наприклад, задача сортування певного набору значень.

Для зручного зберігання кількох змінних одного типу в одній структурі, й для полегшення доступу до них використовують **масиви**.

Одновимірні масиви дають можливість описати наступну структуру даних (в якій, кожна змінна має свій порядковий номер):

Опис масиву мовою Паскаль (в розділі опису змінних):

```
var <назва змінної-масиву> : array[<нижня межа>..<верхня межа>] of
<базовий тип>;
```

де

- *<назва змінної-масиву>* - це ідентифікатор, який користувач присвоює описаному масиву, й надалі, користувач повинен звертатися до масиву саме через цей ідентифікатор.
- *<нижня межа>* - це порядковий номер, який матиме перший елемент масиву.
- *<верхня межа>* - це порядковий номер, який матиме останній елемент масиву.
- *<базовий тип>* - це тип даних, елементи якого міститимуться в масиві. При чому, кожен елемент масиву буде по суті змінною вказаного типу.

Опишемо масив цілих чисел, який складатиметься з 5 елементів:

```
var m : array[1..5] of integer;
```

Тепер, ми по суті, отримали 5 змінних типу `integer`. Для того, щоб звернутися до першої змінної, тобто, першого елементу масиву нам потрібно писати: `m[1],` й виконувати будь-які операції, які ми могли виконувати над цілими числами. Наприклад:

```
m[1]:=23;                  // m[1]=23
m[2]:=10 div 2;            // m[2]=5
m[3]:=m[1]+m[2];           // m[3]=28
m[4]:=m[3] mod 3;          // m[4]=1
m[5]:=(m[1]-m[2]) div 4;   // m[5]=4
writeln(m[1]-m[5]);        // буде виведено "19"
```

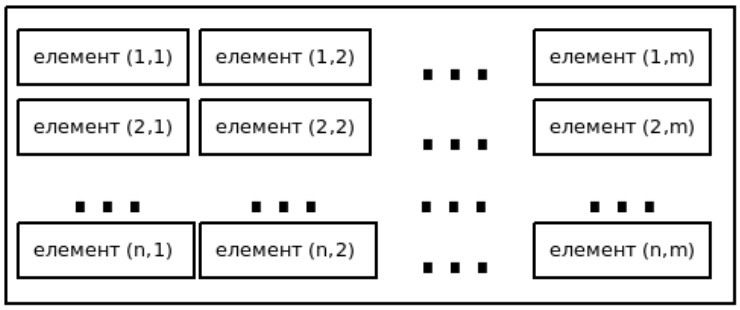
Бачимо, що елементи масиву справді поводять себе як прості змінні вказаного типу. Головна особливість полягає у тому, що ми до кожного елементу масиву

можемо звертатися за його індексом. А отже, ми можемо автоматизувати перебір значень масиву. Наприклад, для виведення значень масиву ми можемо написати код:

```
for i:=1 to 5 do
    writeln('елемент №',i,' містить значення ', m[i]);
```

Результатом роботи програми буде:

```
елемент №1 містить значення 23
елемент №2 містить значення 5
елемент №3 містить значення 28
елемент №4 містить значення 1
елемент №5 містить значення 4
```



Для вирішення деяких задач іноді нам потрібно опрацьовувати табличні дані одного типу. Наприклад, ведення статистичних даних, відображення й опрацювання будь-яких числових таблиць та інше.

Однотипні табличні дані, або ж, дані, збереження яких можна організувати в вигляді таблиці можна зберігати й опрацьовувати в мові Паскаль. Для цього використовують **двовимірні масиви.** Двовимірний масив — це є прямокутна таблиця, кожен елемент якої, на відміну від одновимірного масиву отримує подвійну індексацію, як на Декартовій площині — *(x, y)*:

Опис масиву мовою Паскаль відбувається майже такий самий, як і опис одновимірного масиву, лише з маленькими відмінностями:

```
var <назва змінної-масиву> :  array[<нижня межа1>..<верхня межа1>,
                    <нижня межа2>..<верхня межа2>] of <базовий тип>;
```

де відмінність від опису одновимірного масиву полягає у тому, що задаються дві пари меж:

<нижня межа1>..<верхня межа1>, <нижня межа2>..<верхня межа2>

Саме це передбачає наявність так званої подвійної індексації, як на Декартовій площині. Тобто, в нас буде, умовно кажучи, шафа з n комірками по горизонталі і m комірками по вертикалі. І для того, щоб відкрити одну з комірок-шухляд шафи потрібно задати дві координати — положення комікри-шухляди по вертикалі і по

горизонталі.

Для прикладу, опишемо масив цілих чисел, який складатиметься з 9 елементів: таблиця 3x3.

```
var m : array[1..3, 1..3] of integer;
```

Тепер, ми по суті, отримали 9 змінних типу `integer`. Кожна з яких має ім'я: `m[x, y]`, де x, y — відповідні координати. Для того, щоб звернутися до елементу, який є перший по горизонталі, і перший по вертикалі (на попередньому рисунку — елемент (1,1)) — потрібно написати `m[1, 1]`. Зрозуміло, що по аналогії з одновимірними масивами, над елементами двовимірних масивів ми можемо виконувати будь-які операції, які ми могли виконувати над цілими числами. Наприклад:

```
m[1, 1]:=23;                          // m[1, 1]=23
m[2, 1]:=10 div 2;                    // m[2, 1]=5
m[3, 2]:=m[1, 1]+m[2, 1];             // m[3, 2]=28
m[3, 3]:=m[3, 2] mod 3;               // m[3, 3]=1
m[3, 1]:=(m[1, 1]-m[2, 1]) div 4;     // m[5, 1]=4
writeln(m[1, 1]-m[3, 1]);        // буде виведено "19"
```

Ми можемо автоматизувати перебір значень масиву. На відміну від одновимірних масивів, де ми для цього використовували лише один цикл, тут ми повинні використати два вкладені цикли. "Верхній цикл" відповідатиме за перебір "перших координат", а "внутрішній цикл" перебиратиме "другі координати". Наприклад, для виведення значень масиву ми можемо написати код:

```
for i:=1 to 3 do
Begin
    for j:=1 to 3 do
        write('(', i, ', ', j, ') = ', m[i, j]:3,'   ');
    writeln;
End;
```

Результатом роботи програми буде:

```
19
(1, 1) =  23   (1, 2) =   0   (1, 3) =   0
(2, 1) =   5   (2, 2) =   0   (2, 3) =   0
(3, 1) =   4   (3, 2) =  28   (3, 3) =   1
```

Записи

Якщо нам потрібно оперувати наступною інформацією: наприклад, про декількох людей, то непогано було, якби ми мали тип даних “людина”, який би складався з полів “ім'я”, “прізвище”, “рік народження”, “телефон”.

Як уже говорилися, засобами мови Паскаль користувач може створювати і опрацьовувати власні типи даних. Зокрема, користувач має можливість створювати складені типи. Для цього є спеціальний тип даних `record` — запис. Запис повинен складатися з полів — як у нашому випадку:

- ім'я, прізвище, телефон — стрічки,
- рік народження — натуральне число.

Запис мовою Паскаль описують так:

```
type
<ім'я типу> = record
    <ім'я поля 1> : <тип поля 1>;
    <ім'я поля 2> : <тип поля 2>;
    ...
    <ім'я поля n> : <тип поля n>;
end;
```

Таким чином, описаний вище запис “людина” ми опишемо мовою Паскаль так:

```
type
human = record
    name, surname, tel : string;
    year : integer;
end;
```

Описавши такий тип запису, ми можемо в програмі описувати змінну типу `human`, і в програмі здійснювати такі операції:

```
var me : human;
begin
    me.name := 'Nazar';
    me.surname := 'Gerasymchuk';
    me.year := 1980 + (27 div 3);
    me.tel := '+38063333333';
    writeln('Mr. ', me.name, ' ', me.surname);
    writeln(me.year);
    writeln('tel.: ', me.tel);
end.
```

В результат виконання цієї програми на екран буде виведено:

```
Mr. Nazar Gerasymchuk
```

```
1989
tel.: +38063333333
```

Для полегшення доступу до полів змінної `me` ми могли використати так званий **оператор доступу (команду приєднання) with**, яка існує спеціально для уникання незручностей, які виникають під час багаторазового написання складених імен:

```
with <ім'я запису> do
begin
    <дії над полями>
end;
```

Де між `begin-end` записують імена полів вказаної змінної-запису вже без тієї першої складової. В нашому б випадку, та ж сама програма мала б такий вигляд:

```
var me : human;
begin
    with me do
    begin
        name := 'Nazar';
        surname := 'Gerasymchuk';
        year := 1980 + (27 div 3);
        tel := '+38063333333';
        writeln('Mr. ', name, ' ', surname);
        writeln(year);
        writeln('tel.: ', tel);
    end;
end.
```

Результат виконання програми буде точно такий самий, проте, запис програмного коду став менш громіздкий та більш зрозумілий.

Масиви записів

Раніше описаний тип даних — запис може містити в собі інформацію саме про одну людину. Та зазвичай, задачі, які перед нами ставляться передбачають роботу не з однією людиною, а з цілим “масивом людей”. Прикладом цього може бути записна книжка, в якій зберігається інформація про конкретну людину.

Таким чином, ми підійшли до необхідності створення структури типу **масив записів:**

```
var <назва масиву записів> : array[<межі масиву>] of <тип запису>;
```

Тобто, якщо реалізувати, записну книжку на 10 записів, кожен елемент якої міститиме дані про людину у вигляді запису `human`, то ми матимемо наступний код:

```
var notebook : array[1..10] of human;
```

Далі ми можемо використовувати описану структуру в програмі наступним чином:

```
const n=10; now=2010;
type human = record
    name, surname, tel : string;
    year : integer;
end;
var notebook : array[1..10] of human;
    i : integer;
begin
    for i:=1 to n do
    with notebook[i] do
    Begin
        writeln('- - - Персона №', i, ' - - -');
        write('Введіть ім`я: ');    readln(name);
        write('Введіть прізвище: '); readln(surname);
        write('Введіть рік народження: ');    readln(year);
        write('Введіть телефон: '); readln(tel);
    End;
    writeln;
    writeln('Отримані дані: ');
    for i:=1 to n do
    with notebook[i] do
    Begin
        writeln('- - - Персона №', i, ' - - -');
        writeln('Ім`я: ', name);
        writeln('Прізвище: ', surname);
        writeln('Вік: ', now - year, ' років');
        writeln('Телефон: ', tel);
    End;
    writeln('Виведення закінчено.');
end.
```

Результатом роботи програми є діалог між комп'ютером та людиною, який може відбутися, наприклад(при зміненому значенні константи `const n=2`):

```
- - - Персона №1 - - -
Введіть ім`я: Nazar
Введіть прізвище: Gerasymchuk
Введіть рік народження: 1989
Введіть телефон: +380633333333
- - - Персона №2 - - -
Введіть ім`я: Taras
Введіть прізвище: Shchegelsky
Введіть рік народження: 1987
Введіть телефон: +380503333333
```

```
Отримані дані:
- - - Персона №1 - - -
Ім`я: Nazar
Прізвище: Gerasymchuk
Вік: 21 років
Телефон: +380633333333
- - - Персона №2 - - -
Ім`я: Taras
Прізвище: Shchegelsky
Вік: 23 років
Телефон: +380503333333
Виведення закінчено.
```

Процедури та функції

Часто виникає ситуація, коли в різних місцях програми доводиться виконувати один і той же алгоритм, що має самостійне значення. Це може бути, наприклад, відшукання найбільшого спільного дільника двох натуральних чисел, упорядкування елементів масиву за зростанням чи спаданням, розв'язування системи рівнянь тощо. Якщо цей алгоритм досить складний і є доволі великим фрагментом програми, то нераціонально виписувати його кожного разу в тому місці програми, де його треба використовувати. У більшості мов програмування, у тім числі в мові Паскаль, такий алгоритм можна виділити з основної програми і записати його тільки один раз, зобразивши у вигляді самостійного програмного об'єкта, який називають процедурою. Процедуру визначають за допомогою опису, що розміщений у розділі опису процедур і функцій. Вона має таку ж структуру, як і програма, тобто складається із заголовка і блоку — тіла процедури

В мові Паскаль існують 2 види підпрограм: **процедури** та **функції**. **Процедури** використовуються тоді, коли в підпрограмі необхідно одержати кілька результатів.

Процедура визначається у розділі опису процедур. Структура процедури:

1. Заголовок
2. Розділ описів:
    - мітки
    - константи
    - типи
    - змінні

- процедури і функції

3. Розділ операторів (тіло процедури)

Заголовок складається з ключового слова `procedure`:

```
procedure <ім'я процедури> (<формальні параметри з їх описами>);
<розділ описів>
begin
    <оператори>;
end;
```

Наприклад:

```
procedure my_procedure(b1, b2 : byte; var i3 : integer);
begin
    i3:=(b1-b2) div 2;
end;
```

де

`my_procedure` – ім'я процедури ,

`b1, b2, i3` – імена формальних параметрів

Виклик процедури здійснюється за допомогою оператора виклику процедури `my_procedure(x1, x2, i)`, де `my_procedure` – ім'я процедури, `x1, x2, i` – фактичні параметри, які відповідають формальним параметрам по кількості, типу і місцю розміщення.

В цьому прикладі формальні параметри `b1, b2` є параметрами-значеннями, через які в процедуру передається значення аргумента.

`i3` — є параметром-змінною, так як через цей параметр, в головну програму повертається результат виконання процедури.

Щоб відрізнити параметри-змінні від параметрів-значень у списку формальних параметрів перед параметром-змінною записується службове слово **`var`**. Отже, параметри-значення визначають початкові дані для процедур при кожному її виклику. Параметри-змінні визначають вихідний результат процедури, який передається в головну програму.

Процедура може бути як з параметрами, так і без них. Якщо процедура без параметрів, то в її заголовку є лише ім'я процедури і викликається вона лише по імені.

Локальні та глобальні змінні

При введенні програми розрізняють два типи змінних:

- змінні, які описуються перед описом процедури. Вони описуються в розділі опису змінних основної програми і називаються *глобальними змінними*. Вони зберігають своє значення як в головній програмі, так і в процедурі.
- змінні, що описуються в описі змінних процедури, називаються *локальними змінними*. Вони зберігають своє значення тільки на момент виконання процедури. Після її закінчення вони стають недоступними.

Основні правила роботи з процедурами

1. Число фактичних параметрів у виклику процедури має дорівнювати числу формальних параметрів в описі процедури. Якщо процедура не має параметрів, то оператор виклику процедури складається лише з імені процедури без круглих дужок.
2. Відповідність між формальними і фактичними параметрами встановлюється шляхом їх порівняння зліва направо: 1й=1й, 2й=2й...
3. Тип кожного формального параметра повинен відповідати типу фактичного параметра.
4. Слід чітко розрізняти які параметри є параметрами-значеннями, а які параметрами-змінними. Якщо формальний параметр є параметром-змінною (тобто перед ним стоїть `var`), то фактичний параметр не може бути виразом, а лише іменем змінної.
5. Необхідно розрізняти глобальні та локальні змінні. Глобальні – це змінні, які не є формальними параметрами і не описані в тілі процедури. В процедурі вони мають те ж значення, що й головній програмі. Локальні параметри – описані в тілі процедури.

В складних програмах не рекомендується використовувати глобальні змінні.

Функція — це та ж сама процедура, відмінність полягає лише в тому, що

функція обов'язково повинна повертати якесь значення.

Описується функція майже так само як і процедура:

```
function <ім'я функції> (<формальні параметри з їх описами>) :
                                <тип значення, яке повертатиме функція>;
<розділ описів>
begin
    <оператори>;
end;
```

Лише, в тілі функції, серед операторів обов'язково повинна бути операція присвоєння виду:

```
<ім'я функції> := <значення>;
```

Для прикладу, опишемо функцію, яка обчислюватиме середнє арифметичне двох чисел:

```
function seredne (x1, x2 : integer) : real;
begin
    seredne := (x1+x2) / 2;
end;
```

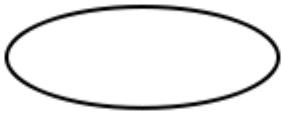
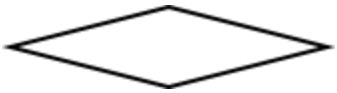
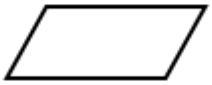
При чому, варто відмітити, що параметрами функції є цілі числа, а повертає функція дійсне значення.

1.4. Блок-схеми

Блок-схеми (графічні схеми) часто використовують для наочного зображення алгоритмів. Зазвичай, перед безпосереднім написанням програми варто побудувати (нарисувати) блок-схему алгоритму, за яким діятиме ця програма.

Блок-схема будується за допомогою структурних блоків (див. нижче), що позначають різні події, команди і т. д. Ці блоки з'єднуються лініями, які вказують на послідовність виконання команд.

Будемо використовувати такі основні елементи блок-схем:

Вигляд	Опис
	Елемент відображає вхід із зовнішнього середовища або вихід з нього (**найбільш часте застосування - початок і кінець програми**). Всередині фігури записується відповідна дія.
	Виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.
	Розгалуження — два варіанти (шляхи), з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.
	Цикл “для” (“`for`”). Всередині шестикутника вказуються умови циклу. Стрілка вниз від шестикутника означає початок тіла циклу, після команд якого відбувається повернення керування назад до циклу — стрілка, що йде до шестикутника зліва. Вихід з циклу — стрілка справа.
	Перетворення даних у форму, придатну для обробки. Зазвичай — отримання даних від користувача (зчитування інформації з консолі, з файлу).
	Виведення інформації (в консоль, в файл).

1.5. Економія часу

Іноді, в процесі розв'язання тієї чи іншої задачі, програмуючи і тестуючи її розв'язок доводиться неоднократно з клавіатури вводити вхідні дані для їх подальшого опрацювання програмою.

Цей етап можна дещо полегшити...

1. Можна, по можливості описати **сталий масив значень** — тобто такий масив, значення якого в ході виконання програми змінюватися не будуть. Це можна зробити наступним чином (в розділі опису констант написати):

```
const n=10;
a : array[1..n] of integer = (1, 9, 1, 5, 2, 3, 4, 2, 7, 8);
```

Тепер при написанні програми ми можемо використовувати дані масиву.

2. Іншим, більш зручним варіантом є використання **файлів**. При чому, для тестування здебільшого зручно діяти наступним чином: один раз записати вхідні дані в файл, а потім, програмою постійно зчитувати дані з файлу (не змінюючи файл з даними), а результат роботи програми — виводити на екран. Для досягення поставлених цілей потрібно виконати ряд дій:
   a) Налаштувати зв'язок між фізичним файлом і файловою змінною, тобто *приєднати файл* — команда `assign`:

      ```
      assign(input, '<шлях до файлу та ім'я файлу>');
      ```

      шлях до файлу можете не вказувати, якщо файл знаходиться в тому самому каталозі, в якому знаходиться програма
   b) Відкрити файл командою `reset:`

      ```
      reset(input);
      ```

   c) Здійснити читання з файлу:

      ```
      read(...)
      ```

   d) По закінченню читання з файлу — закрити його командою `close`:

      ```
      close(input);
      ```

Тобто, якщо перед нами стоїть проста задача — *отримати від користувача два цілих числа, й вивести їх суму*, то для тестування розв'язку створимо файл `input.txt` в каталозі з нашою програмою в якому напишемо наступний текст:

```
100  1
```

Програму розв'язку напишемо наступним чином:

```
var a, b : integer;
begin
    assign(input, 'input.txt');
    reset(input);
```

```
    readln(a, b);
    close(input);
    writeln(a + b);
end.
```

Для того, щоб випробувати написану програму на інших вхідних даних, очевидно, потрібно буде відкрити файл `input.txt` і змінити його вміст. При чому, програма залишається незмінною. Це дуже зручно у випадках, коли вхідні дані до програм дуже громіздкі й великі за обсягом.

При здачі задачі на автоматизовану перевірку потрібно не забувати коментувати команди по роботі з файлами (Звісно, якщо сервером не передбачено іншого)!

Тобто, якби нам потрібно було здати на перевірку написану вище програму, то здавали б її принаймні в такому вигляді:

```
var a, b : integer;
begin
    //assign(input, 'input.txt');
    //reset(input);
    readln(a, b);
    //close(input);
    writeln(a + b);
end.
```

РОЗДІЛ 2. РОЗБІР ЗАДАЧ.

2.1. Масиви

Задача 1. Слоненята.

Ця задача взята з розділу **“Задачі для початківців” Всеукраїнського Центру олімпіад школярів в інтернеті**, доступна в мережі за адресою:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=99

Сторінка для здавання на автоматичну перевірку задачі:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=823&task=slon

Задача Slon. Петрик П'яточкін вишикував у рядок слоненят та рахує їх по кожному кольору окремо. Всього буває 8 кольорів слоненят. У рядок вишикувались N (10<N<999) слоненят. Скільки слоненят кожного кольору стоїть перед Петриком? Бажано їх порахувати пройшовши всього один раз перед строєм.

Технічні умови. Програма зчитує з клавіатури ціле число N - кількість слоненят, потім, через пропуск — N чисел від 1 до 8, якими ми пронумерували кожен колір в тій послідовності, в якій вони потрапляли на очі Петрику від початку рядка. Програма виводить на екран в один рядок через пропуски пари цілих чисел, де перше число пари - колір, а друге - кількість слоненят такого кольору.

Приклад

Введення	**Виведення**
12 1 1 2 3 3 1 5 6 8 7 6 5	1 3 2 1 3 2 4 0 5 2 6 2 7 1 8 1

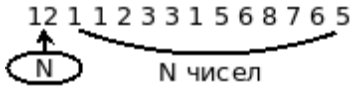
Аналіз задачі

Суть задачі — порахувати слоненят, при чому, слоненята можуть бути різних кольорів. Всього 8 можливих кольорів. Це означає, що у нас буде 8 груп слоненят, а отже, на підрахунок яких нам потрібно буде відвести 8 лічильників. Це може бути 8 цілочисельних змінних, при чому, виходячи з умови задачі, бачимо, що кожен з лічильників не може перевищувати 999, бо загальна кількість слоненят N лежить в межах (10<N<999). Замість 8 змінних, легше було б використати таку структуру, як

масив, що складатиметься з 8 елементів, кожен з яких може набувати цілочисельного значення.

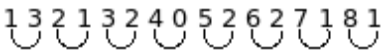
В умові перед нами ставиться задача, здійснити підрахунок пройшовши всього один раз перед строєм. Це означає, що почергово проходячи перед кожним слоненям ми повинні отримати колір цього слоненя й одразу збільшувати відповідний лічильник. Отже, за один цикл ми повинні виконати всі підрахунки.

Тепер розберемося з *вхідними та вихідними даними*.



Перше число, яке ми отримуємо на *вхід програми* це N - кількість слоненят, далі, ідуть записані через пробіл кольори кожного з N слоненят. Отже, програма повинна спершу прийняти число N, а потім N чисел, які означають кольори слоненят.

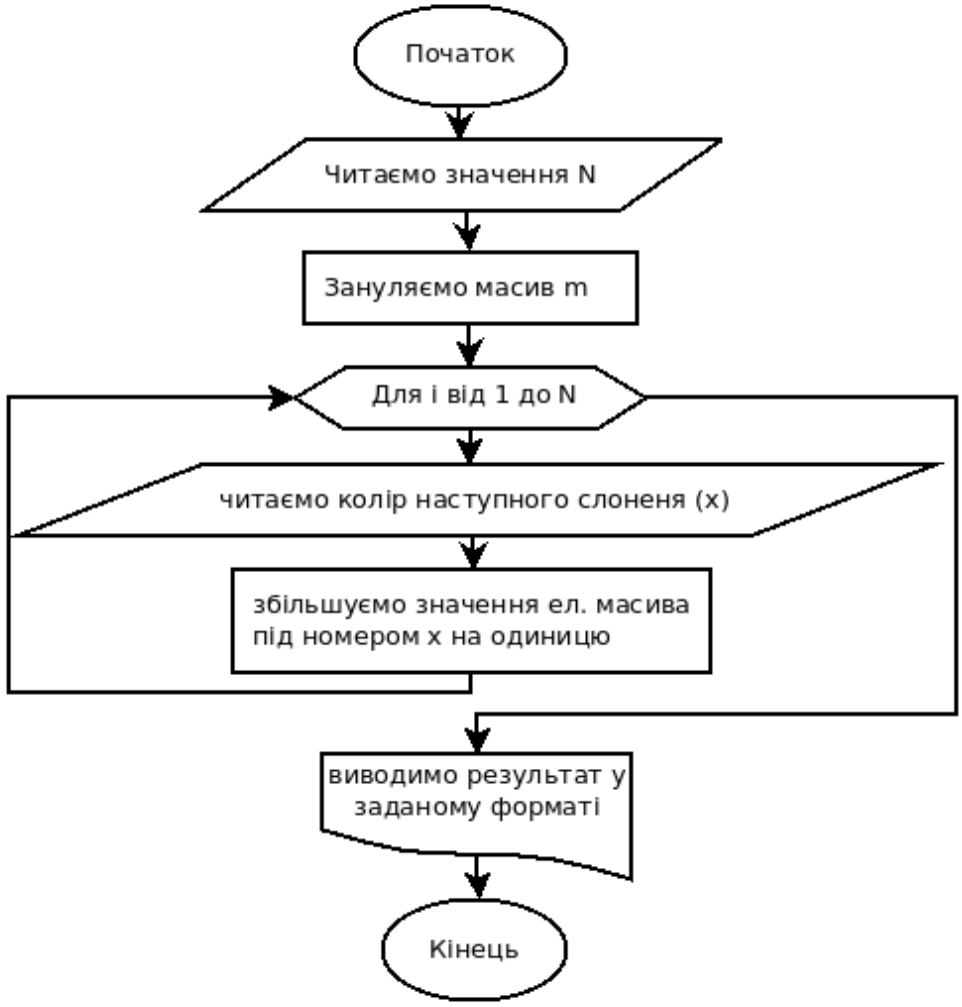
В якості *вихідних* даних, програма повинна в один рядок вивести пари чисел, де перше число символізує номер кольору, а друге число — кількість слоненят цього



кольору — і так для всіх восьми допустимих кольорів.

Складемо блок-схему алгоритму розв'язання задачі[11]:

11 Надалі в блок-схемах в циклах ДЛЯ (for) у шестикутнику замість тексту “для і від 1 до N” будемо користуватися записов *i=1, N*, що означає точно те саме, тобто, те, що змінна *i* почергово з кроком 1 починаючи з першого значення 1 і закінчуючи останнім *N* змінюється.



Програмування розв'язку задачі

```
program Slon;
var m : array[1..8] of integer;
    i, n, x : integer;
begin
    read(n);
    for i:=1 to 8 do m[i]:=0;
    for i:=1 to n do
    Begin
         read(x);
         inc(m[x]);
    End;
    for i:=1 to 8 do write(i, ' ', m[i], ' ');
end.
```

Де m — масив, в якому в і-му елементі зберігається інформація про кількість слоненят, які мають і-ий колір, тобто, якщо в 3-му елементі масива після завершення

роботи програми отримається значення 2, то це означатиме, що в заданому переліку слоненят зустрілося 2 слоненя, які мали 3-ій колір.

Шматок коду `inc(m[x])` є еквівалентний наступному: `m[x]:=m[x]+1;`

Процедура `inc` збільшує на одиницю передану їй змінну. Змінна повинна бути перелічувана (або символом). Також, можливий варіант використання цієї функції у такому вигляді: `inc(a, 2)` — що означатиме "збільшити значення змінної a на 2".

Існує протилежна до `inc` функція — `dec`, яка по використанню повністю схожа до inc, проте виконує відніманя одиниці від переданої їй змінної.

Задача 2. Зсунь елементи

Ця задача (922) з "E-Oplimp", доступна в мережі за адресою:
http://www.e-olimp.com.ua/ua/problems/922

Задано одновимірний масив `A` цілих чисел довжини `h`. Зсунути елементи масиву циклічно праворуч на 1 крок.

Технічні умови

Вхідні дані. У першому рядку задано натуральне число `h` - кількість елементів масиву (`h≤100`). У другому рядку задано самі елементи масиву, значення кожного з яких за модулем не перевищує `100`.

Вихідні дані. В єдиному рядку вивести через проміжок `h` чисел: нові значення елементів масиву.

Приклад

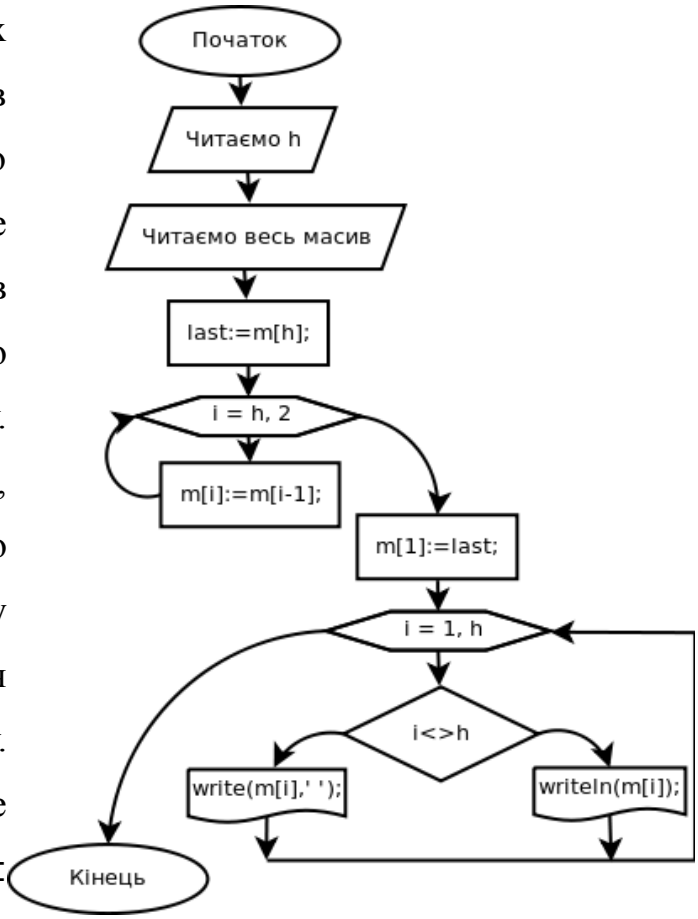
Введення	**Виведення**
4 1 2 3 4	4 1 2 3

Аналіз задачі

Суть задачі є проста і зрозуміла. Виходимо з того, що є масив, в якому потрібно виконати зсув (ця процедура нерідко може зустрічатися як складова частина складних алгоритмів опрацювання масивів). Зсув вправо на 1 крок означає, що кожен елемент в масиві змінює своє положення на одну комірку вправо - "підсовується вправо"; лише для останнього (самого правого в масиві) елемента — нікуди рухатися, отже він "міняється місцями" з першим.

Програмування розв'язку задачі

Бачимо, що спочатку у вхідних даних подається кількість елементів масиву, а потім ідуть самі елементи цього масиву. Отже, зчитати масив не складе труднощів. Починаємо *зсув* з того, що в тимчасову змінну `last` збережемо значення останнього елемента масиву. Далі, циклічно виконуємо безпосередньо, сам зсув — з елемента під номером `h` до елемента під номером `2` — в зворотньому порядку — виконуємо переприсвоєння значення `(i-1)`-го елемента `i`-тому. Після цього, присвоюємо раніше збережене значення в змінній `last` першому елементу масива.



Під час виведення результуючого масиву, ми не повинні виводити ніяких символів, тобто, якщо ми напишемо код виведення елементів масиву наступним чином:

```
for i:=1 to h do
    write(m[i],' ');
```

то в результаті, на екран буде виведено зайвий символ — пробіл, після останнього елемента масиву. Таким чином, нам потрібно проконтролювати цю ситуацію — як тільки ми будемо виводити останній елемент — потрібно після нього нічого не виводити — лише символ закінчення рядка, який вставляється при використанні процедури `writeln`. Отже, перепишемо код виведення масиву, матимемо:

```
for i:=1 to h do
    if i<>h then write(m[i],' ')
    else writeln(m[i]);
```

Розв'язок матиме вигляд:

```
program zsuv;
var h, i, last : integer;
    m : array[1..100] of integer;
begin
    readln(h);
    for i:=1 to h do read(m[i]);
    last:=m[h];
    for i:=h downto 2 do
        m[i]:=m[i-1];
    m[1]:=last;
    for i:=1 to h do
        if i<>h then write(m[i],' ')
        else writeln(m[i]);
end.
```

Задача 3. Вінні

Ця задача взята з розділу **"Задачі для початківців" Всеукраїнського Центру олімпіад школярів в інтернеті**, доступна в мережі за адресою:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=100

Сторінка для здавання на автоматичну перевірку задачі:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=823&task=vinni

Задача Vinni. Вінні Пух любить складати віршики говорячи речення задом наперед. Якось йому попалось довге складне речення і він забув свій віршик, пробуючи його виговорити. Складіть програму, яка б допомагала ведмедику легко складати такі віршики. Зауваження: віршик може складатись як із 1 слова, так і з декількох, розділених пропусками.

Технічні умови. Програма зчитує з клавіатури стрічку-віршик. В кінці віршика ніколи не ставиться крапка. Довжина віршика менша за 255 символів.. Програма виводить на екран стрічку, яку отримано внаслідок повороту.

Приклад.

Введення	**Виведення**
роза	азор
Все медведи любят мед	дем тябюл идевдем есВ

Аналіз задачі

Згідно з умовою задачі, нам потрібно зчитати рядок символів, а потім — вивести на екран цей самий рядок символів, лише в оберненому порядку.

Для розв'язання задачі будемо використовувати масив символів. Так як нам сказано, що стрічка повинна бути меншою за 255 символів, то відповідно і розмірність масиву повинна бути 255:

```
var m : array[1..255] of char;
```

Здійснювати читання стрічки будемо допоки не досягнемо кінця стрічки, тобто допоки виконується умова **NOT** EOLN. При читанні посимвольно, ми повинні мати змінну-вказівник(і), яка вказуватиме, в яку позицію в масиві потрібно вставити щойно зчитаний символ.

По завершенню читання (досягнення кінця стрічки) — нам потрібно вивести у зворотньому порядку всі елементи масиву. Так як і вказує на позицію в масиві, куди потрібно вставити щойно зчитаний символ, то відповідно, останнім символом буде (і-1)-ий. Отже, зменшуємо і на одиницю, й починаємо низхідний цикл по і, допоки і>0. В цьому циклі ми повинні виконувати наступні операції: виводити на екран елемент масиву під номером і, а після цього — зменшувати значення і на 1.

Програмування розв'язку задачі

```
var m : array[1..255] of char;
    i : byte;
begin
    i:=1;
    while (NOT EOLN) do
    Begin
         read(m[i]);
         inc(i);
    End;
    dec(i);
    while i>0 do
    Begin
         write(m[i]);
         dec(i);
    End;
end.
```

Задачі на самостійне опрацювання

1. ***http://www.e-olimp.com.ua/ua/problems/494***

Голосні

До голосних літер в латинському алфавіті відносяться літери A, E, I, O, U і Y. Інші літери вважаються приголосними. Напишіть програму, яка підраховує кількість голосних літер в тексті.

Технічні умови. Вхідні дані. У вхідному файлі міститься один рядок тексту, який складається лише із великих латинських літер та пробілів. Довжина рядка не перевищує 100 символів.

Вихідні дані. У вихідний файл вивести одне ціле число – кількість голосних у вхідному тексті.

2. ***http://www.e-olimp.com.ua/ua/problems/914***

Модуль максимального

Задано одновимірний масив A дійсних чисел, пронумерованих від 1 до h. Визначити значення модуля максимального елемента масиву.

Технічні умови. Вхідні дані. У першому рядку задано натуральне число h - кількість елементів у масиві ($h \le 100$). У наступному рядку через пропуск задано h дійсних чисел - самі елементи масиву, значення яких не перевищують по модулю 100.

Вихідні дані. Єдине число - відповідь до задачі, виведене з точністю 2 знаки після десяткової крапки.

3. ***http://www.e-olimp.com.ua/ua/problems/908***

Ті, що діляться на 6

Для N цілих чисел визначити суму й кількість додатних чисел, які діляться на 6 без остачі.

Технічні умови. Вхідні дані. У першому рядку задано кількість чисел N ($0 < N \le 100$), у наступному рядку через пропуск задано самі числа, значення яких по модулю не перевищують 10000.

Вихідні дані. У єдиному рядку виведіть спочатку кількість вказаних чисел і через пропуск їх суму.

4. ***http://www.e-olimp.com.ua/ua/problems/907***

Перший не більший за 2,5

Задано одновимірний масив A дійсних чисел, пронумерованих від 1 до h. Визначити перший елемент масиву, який не перевищує 2.5.

Технічні умови. Вхідні дані. У першому рядку задано кількість елементів масиву h ($0 < h \le 100$), у наступному рядку задано h дісних чисел, відокремлених пропуском.

Вихідні дані. Вивести у одному рядку спочатку індекс знайденого першого вказаного елемента масиву і через пропуск його значення з точністю 2 знаки після десяткової крапки. У випадку відсутності вказаного елементу в масиві вивести "Not Found" (без лапок).

5. ***http://www.e-olimp.com.ua/ua/problems/904***

Збільшити на 2

Задано одновимірний масив A цілих чисел. Збільшити на 2 кожний невід'ємний елемент масиву.

Технічні умови. Вхідні дані. У першому рядку задано натуральне число h - кількість елементів масиву (h≤100). У другому рядку через проміжок задано самі елементи масиву, значення кожного з яких за модулем не перевищує 100.

Вихідні дані. В єдиному рядку вивести через проміжок h чисел: нові значення елементів масиву, у тому ж порядку, в якому їх було задано.

6. ***http://www.e-olimp.com.ua/ua/problems/928***

Сума найбільшого та найменшого

Задано одновимірний масив A цілих чисел. Визначити суму найменшого та найбільшого елементів масиву.

Технічні умови. Вхідні дані. У першому рядку задано натуральне число h - кількість елементів масиву (h≤ 100). У другому рядку через проміжок задано самі елементи масиву, значення кожного з яких за модулем не перевищує 100.

Вихідні дані. В єдиному рядку вивести одне число - відповідь до задачі.

7. ***http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=188***

Sweets.

Маленький хлопчик потрапив до казкової країни і побачив там дорогу, вздовж якої розкладено мішки з цукерками. На кожному мішку написана кількість цукерок. Хлопчик може взяти в кожну руку два мішки, що лежать поруч. Яку найбільшу кількість цукерок він може взяти?

Технічні умови. Ви вводите з клавіатури кількість мішків N (4<=N<=10000), а потім N чисел через пропуск - кількість цукерок у кожному мішку (всі числа непід'ємні і не перевищують 1000000). Ви виводите на екран єдине шукане число.

2.2. Найбільший спільний дільник та найменше спільне кратне

Необхідні знання з теми

Найбільшим спільним дільником (надалі, використовуватимемо скорочення **НСД**) дільник двох чисел є найбільше число, що ділить обидва дані числа без залишку[12].

Наприклад, для чисел 25 та 15 НСД буде рівний 5.

Для того, щоб обчислити НСД часто користуються методом розкладу числа на

12 Розгядатимемо НСД саме додатніх чисел.

прості множники. Наприклад, знайдемо $НСД(792,\ 360)$. Для цього нам потрібно числа 792 та 360 подати у вигляді: $\begin{matrix} 360 = 2^3 \cdot 3^2 \cdot 5^1 \\ 792 = 2^3 \cdot 3^2 \cdot 5^0 \cdot 11^1 \end{matrix}$.

Тепер, НСД отримуємо як добуток усіх спільних для обох чисел простих множників: $2^3 \cdot 3^2 = 72$.

Варто відмітити наступні властивості НСД:

- Від перестановки чисел НСД не змінюється: $НСД(a,\ b) = НСД(b,\ a)$.
- НСД лежить в межах: $1 \le НСД(a,\ b) \le min(a,\ b)$.
- Для того, щоб знайти НСД трьох чисел a, b, c необхідно знайти спершу $НСД(a,\ b)$, а потім використати знайдений результат й знайти шукане НСД трьох чисел як $НСД(НСД(a,\ b),\ c)$. Це ж правило можна поширити для довільної кількості вхідних даних.

Для обчислення НСД існує дуже зручний метод, що потребує значно менше обчислень. Це **<u>алгоритм Евкліда</u>**.

Алгоритм Евкліда заснований на тому, що НСД двох чисел не змінюється, якщо від більшого числа відняти менше. Наприклад, `21` є НСД чисел `252` та `105`: $\begin{matrix} 252 = 21 \cdot 12 \\ 105 = 21 \cdot 5 \end{matrix}$. Оскільки $252 - 105 = 147$, $НСД(147,\ 105) = 21$.

Оскільки більше з двох чисел постійно зменшується, повторне виконання цього кроку дає все менші числа, поки одне з них не дорівнюватиме нулю. Коли одне з чисел дорівнюватиме нулю, інше, відмінне від нуля і є НСД. Так виглядатиме процес пошуку $НСД(20,\ 12)$:

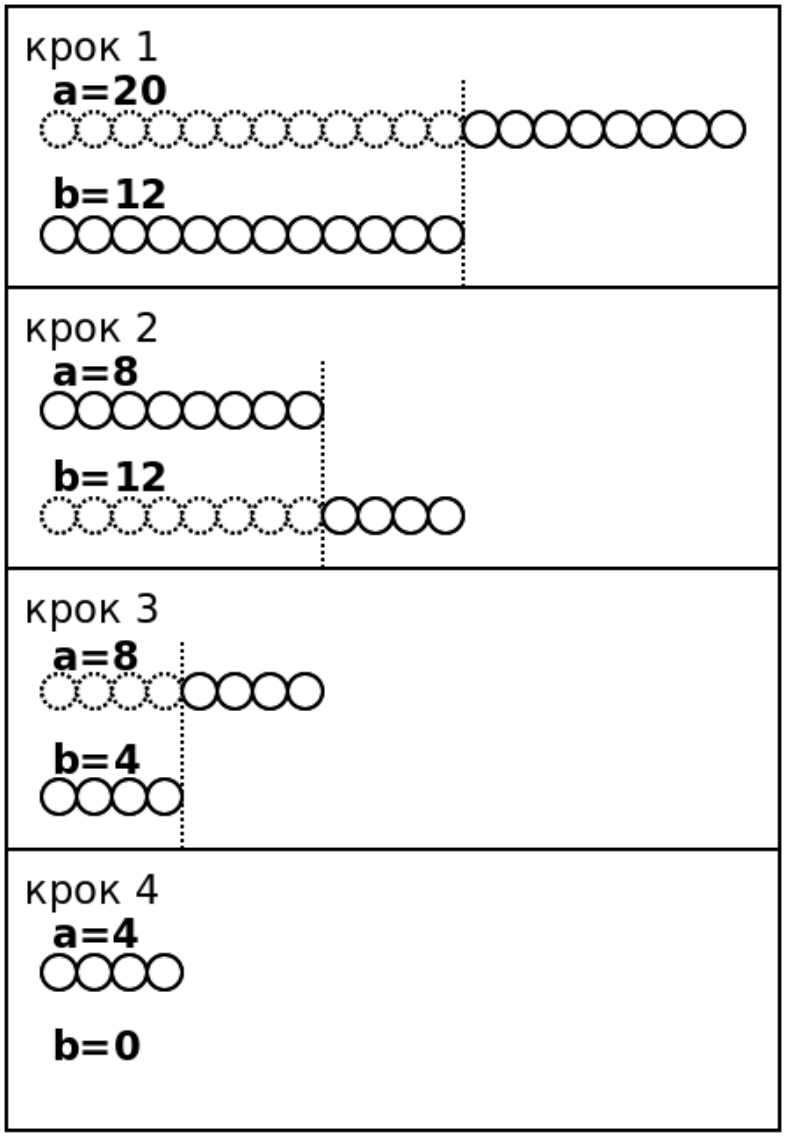


Схема виконання алгоритму пошуку НСД двох чисел

На кроці 4 ми дійшли до того, що $b=0$, а отже, $НСД(20,\ 12) = a = 4$.

Для того, щоб запрограмувати процес знаходження НСД за допомогою алгоритму Евкліда, зручно ввести **рекурсивне означення НСД**, яке ґрунтується на покращенні методу за рахунок обчислення остач від ділення на кожному кроці. Тобто, для обчислення $НСД(40,\ 4)$ ми б не виконували дії:

$$\begin{array}{cc} 40 & 4 \\ 40-4=36 & 4 \\ 36-4=32 & 4 \\ 32-4=28 & 4 \\ \ldots & \ldots \end{array}$$

А виконали б:

$$\begin{array}{rr} 40 & 4 \\ 40\,mod\,4=0 & 4 \end{array}$$

І одразу отримали результат: $НСД(40,\ 4)\ =\ 4$.

Отже, **рекурсивне означення НСД[7]**:

$$НСД(a,b)=\begin{cases} a,\ якщо\ b=0, \\ b,\ якщо\ a=0, \\ НСД(a\,mod\,b,b),\ якщо\ a\geq b, \\ НСД(a,b\,mod\,a),\ якщо\ a<b. \end{cases}$$

Хоча, взагалі кажучи, користуючись властивістю НСД (від перестановки чисел НСД не змінюється) НСД можна означити наступним чином:

$$НСД(a,b)=\begin{cases} b,\ якщо\ a=0, \\ НСД(b\,mod\,a,a),\ якщо\ a\neq 0. \end{cases}$$

що значно спрощує програмну реалізацію функції НСД.

```
function NSD(a, b : integer) : integer;
begin
    if a=0 then NSD:=b
    else NSD:=NSD(b mod a, a);
end;
```

Найменше спільне кратне (надалі - **НСК**) двох чисел $НСК(a,\ b)$ є найменшим натуральним числом, яке ділиться без залишку на обидва числа a та b .

Наприклад, для чисел 25 та 15 НСК буде рівне 75.

Для того, щоб обчислити НСК, як і для обчислення НСД користуються методом розкладу числа на прості множники. Наприклад, знайдемо $НСК(792,\ 360)$. Для цього нам потрібно числа 792 та 360 подати у вигляді (як ми це робили вище):

$$\begin{array}{l} 360\ =\ 2^3\cdot 3^2\cdot 5^1 \\ 792\ =\ 2^3\cdot 3^2\cdot 5^0\cdot 11^1 \end{array}.$$

Тепер, НСК отримуємо як добуток простих множників обох чисел в найбільшій степені: $2^3\cdot 3^2\cdot 5^1\cdot 11^1\ =\ 3960$.

Відмітимо, що:

- Для НСК також характерним є те, що від перестановки чисел НСК не змінюється: $НСК(a,\ b)\ =\ НСК(b,\ a)$.
- Для того, щоб знайти НСК трьох чисел a,b,c необхідно знайти

спершу $НСК(a,\ b)$, а потім використати знайдений результат й знайти шукане НСК як $НСК(НСК(a,\ b),\ c)$. Це ж правило можна поширити для довільної кількості вхідних даних.

- НСК легко знаходиться з наступної формули:

$$НСД(a,b)\cdot НСК(a,b)=a\cdot b \quad \Rightarrow \quad НСК(a,b)=\frac{a\cdot b}{НСД(a,b)}$$

Так як для прикладу НСД та НСК двох чисел ми використовували ті ж самі числа, перевіримо отриманий результат.

Бачимо, що формула “працює”, а отже ми можемо сміливо нею користуватися.

$$НСД(360,\ 792)=72,$$
$$НСК(360,\ 792)=3960,$$
$$72\ \cdot\ 3960\ =\ 360\ \cdot\ 792.$$

Маючи описану функцію `NSD`, знаходження НСК можемо описати:

```
function NSK(a, b : integer) : integer;
begin
    NSK:=a*b div NSD(a,b);
end;
```

Задача 1. НСД

Ця задача взята з “E-Olimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/137

Знайти НСД (найбільший спільний дільник) `N` чисел.

Технічні умови

Вхід: У першому рядку кількість чисел `N` (`1<N<101`).

У другому - через пропуск `N` натуральних чисел, що не перевищують `30000`.

Вихід: НСД цих чисел.

Приклад.

Введення	**Виведення**
2 15 25	5

Аналіз задачі

Для розв'язання задачі потрібно використати правило:

- Для того, щоб знайти НСД трьох чисел a,b,c необхідно знайти спершу $НСД(a,\ b)$, а потім використати знайдений результат й знайти шукане НСД трьох чисел як $НСД(НСД(a,\ b),\ c)$. Це ж правило можна поширити для довільної кількості вхідних даних.

Зрозуміло, що для розв'язання поставленої задачі потрібно мати описану процедуру знаходження НСД двох чисел. І надалі, наприклад, для чотирьох чисел a, b, c, d їх НСД шукатимемо наступним чином:

$$НСД(НСД(НСД(a,\ b),\ c),\ d)$$

Відповідно, за цим же принципом шукатимемо НСД більшої кількості чисел.

Отже, спочатку, нам потрібно зчитати 2 числа й знайти їх НСД. Це значення зберегти в тимчасову змінну (z), і надалі, поступово читаючи нові числа — переприсвоювати значення z, надаючи їй значення, що обчислюється, як НСД нового числа і поточного значення z. Для цього достатньо наявності циклу від 3 до n, при чому, якщо n=2 (тобто на вхід подається лише 2 числа), то тіло циклу не виконається жодного разу, а одразу виведуться результати.

Програмування розв'язку задачі

```
var n, x, z, y, i : integer;

function NSD(a, b : integer) : integer;
begin
   if a=0 then NSD:=b
   else NSD:=NSD(b mod a, a);
end;

begin
   readln(n);
   read(x,y);
   z:=nsd(x,y);

   for i:=3 to n do
   Begin
        read(x);
        z:=nsd(z,x);
   End;
   writeln(z);
end.
```

Задача 2. НСК

Ця задача взята з “E-Olimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/135

Знайти найменше спільне кратне (НСК) `N` натуральних чисел.

Технічні умови

Вхід: У першому рядку кількість чисел `N (1 < N < 21)`.

У другому - через пропуск `N` натуральних чисел, що не перевищують `100`.

Вихід: НСК цих чисел.

Приклад

Введення	**Виведення**
2 2 3	6

Аналіз задачі

Розв'язання задачі, будемо здійснювати по аналогії з попередньою задачею, тобто, використовуючи узагальнену формулу для знаходження НСК кількох чисел. При чому, для знаходження НСК будемо користуватися формулою

$$НСК(a,b)=\frac{a\cdot b}{НСД(a,b)}$$

З попередньої задачі, функція знаходження значення НСД в нас є. Єдина проблема, з якою можемо зустрітися при розв'язанні задачі — це великі числа — значення НСК. Отже, використаємо тип даних `int64`.

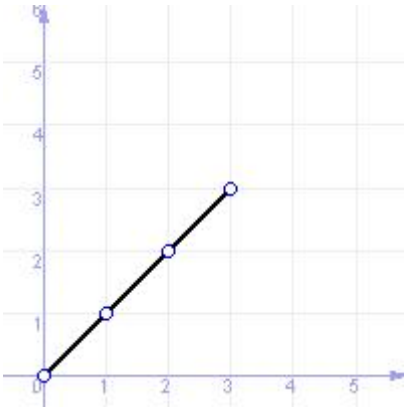
Програмування розв'язку задачі

```
var n, i, a, b : integer;
    rez : int64;

function NSD(a, b : int64) : int64;
begin
    if a=0 then NSD:=b
    else NSD:=NSD(b mod a, a);
end;

function NSK(a, b : int64) : int64;
```

```
begin
   NSK:=a*b div NSD(a,b);
end;

Begin
   readln(n);
   read(a,b);
   rez:=nsk(a,b);

   for i:=1 to n-2 do
   Begin
        read(a);
        rez:=NSK(rez,a);
   End;
   writeln(rez);
End.
```

Задачі на самостійне опрацювання

1. ***http://www.e-olimp.com.ua/ua/problems/136***



Відрізок

Задано відрізок, кінці якого мають цілочисельні координати. Підрахувати кількість точок відрізку, які мають цілочисельні координати.

Технічні умови. Вхідні дані. 4 числа — координати X1, Y1, X2, Y2 кінців відрізку. Всі вхідні дані не перевищують за модулем $2 \cdot 10^9$.

Вихідні дані. Єдине число - шукана кількість точок.

2. ***http://www.e-olimp.com.ua/ua/problems/571***

Найбільший спільний дільник

Задано N натуральних чисел. Напишіть програму, яка обчислює найбільший спільний дільник цих чисел.

Технічні умови. Вхідні дані. У першому рядку вхідного файлу знаходиться натуральне число N (N<=1000) - кількість чисел. Далі йде N натуральних чисел, кожне з яких не перевищує $2 \cdot 10^9$.

Вихідні дані. У вихідний файл виведіть єдине число - найбільший спільний дільник заданих чисел.

3. ***http://www.e-olimp.com.ua/ua/problems/752***

Цілі точки

Многокутник (не обов'язково опуклий) на площині задано координатами своїх вершин. Потрібно порахувати кількість точок з цілочисельними координатами, які лежать всередині нього (але не на його границі).

Технічні умови. Вхідні дані. У першому рядку міститься N ($3 \le N \le 1000$) – число вершин многокутника. У наступних N рядках йдуть координати (Xi, Yi) вершин многокутника у порядку обходу за годинниковою стрілкою. Xi і Yi - цілі числа, які по модулю не перевищують 1000000.

Вихідні дані. У вихідний файл вивести одне число – шукану кількість точок.

2.3. Сортування

Необхідні знання з теми

Знання й розуміння методів сортування зазвичай є необхідним при розв'язуванні олімпіадних задач.

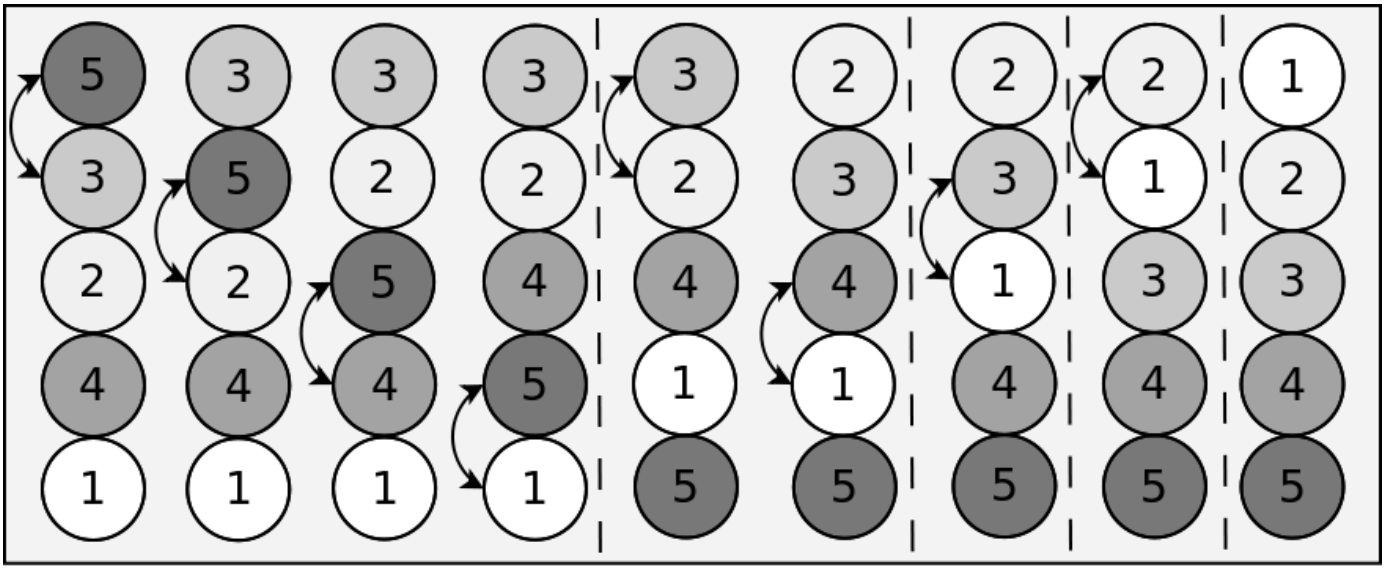
На сьогодні, існують багато різних типів сортування, з-поміж них найвідомішими є: “сортування бульбашкою”, “сортування вставками”, “сортування вибором”, “швидке сортування”, “метод Шелла” та інші.

Сортування бульбашкою

Сортування бульбашкою — один із найпростіших методів сортування. Цей метод сортування отримав свою назву через те, що якщо спостерігати за процесом сортування, то можна помітити, що “найлегші” елементи ніби поступово спливають вверх, як бульбашки.

Як видно, **ідея методу** полягає у тому, щоб у декілька кроків пробігати з самого низу масиву до верху, й переглядати пари сусідніх елементів. У випадку, коли в парі елементів перший елемент більший за другий — міняємо місцями ці елементи. Повторювати перегляд масиву будемо до тих пір, поки масив не буде повністю відсортований. [13]

13 На ілюстрації сортування пунктирними лініями відділяються етапи входження в цикл `while` — див. далі



При написанні алгоритму сортування, варто врахувати, що ми можемо значно зменшити кількість “непотрібних” переглядів масиву у випадку, наприклад, коли масив є повністю відсортований, або відсортований на певному етапі сортування. Для того, щоб відслідкувати цей момент потрібно стежити за тим, чи на попередньому кроці сортування ми переставляли сусідні елементи місцями. В тому випадку, якщо перестановок не відбулося — це означатиме що їх уже й не потрібно робити, а отже — масив повністю відсортований.

Опишемо масив:

```
const n=10;
var a : array[1..n] of integer;
```

Запрограмуємо алгоритм сортування у вигляді процедури згідно блок-схеми:

```
procedure sort_array;
{сортування методом бульбашки}
var treba : boolean;
    i, j, t : integer;
begin
    treba:=true;
    while treba do
    Begin
```

```
        treba:=false;
        for i:=1 to n-1 do
            if a[i]>a[i+1] then
            Begin
                t:=a[i];
                a[i]:=a[i+1];
                a[i+1]:=t;
                treba:=true;
            End;
    End;
End;
```

Для зручності, опишемо додаткові процедури для роботи з масивом: процедуру створення масиву з довільними значеннями елементів та процедуру виведення на екран значення усіх елементів масиву — `make_array` та `print_array` відповідно:

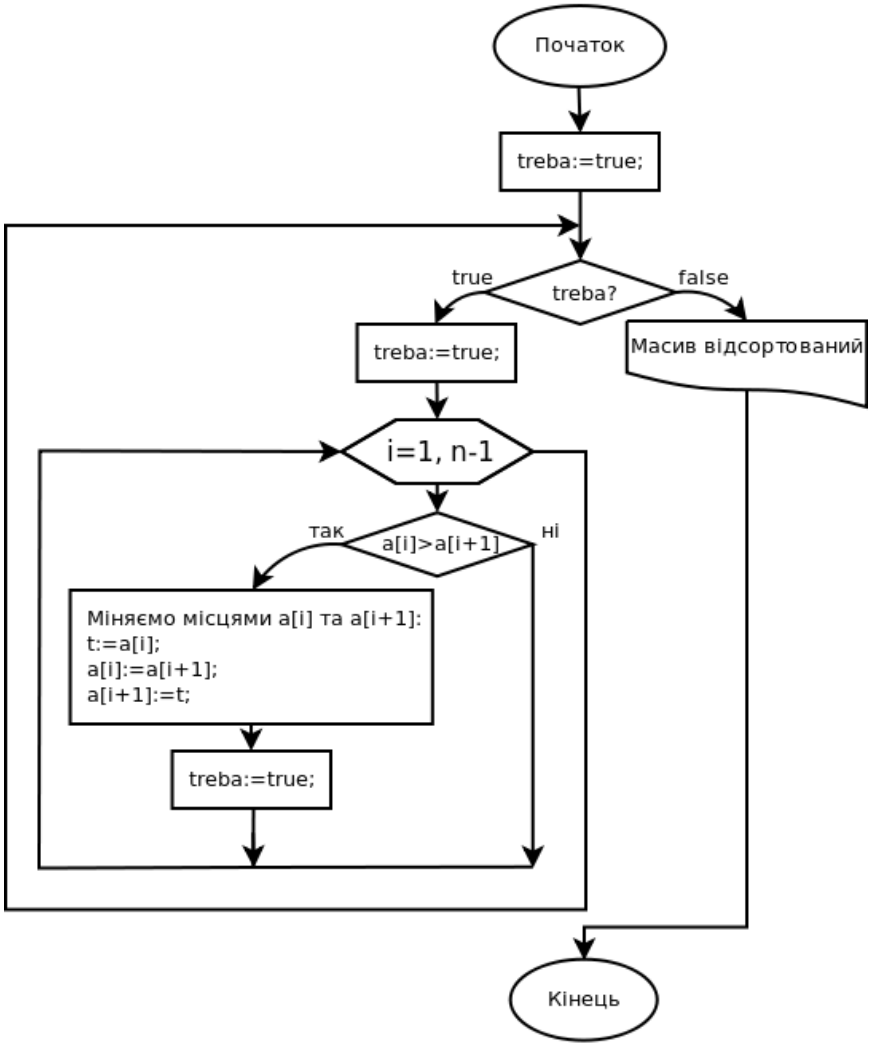
```
procedure make_array;
{генерація масиву}
var i : integer;
begin
    randomize;
    for i:=1 to n do a[i]:=random(1000);
end;

procedure print_array;
{виведення масиву}
var i : integer;
begin
    for i:=1 to n do write(a[i]:5);
    writeln;
end;
```

Де процедура `randomize` встановлює генератор випадкових чисел на наступне число. Функція `random(x : integer)` повертає випадкове значення в межах $[0; x]$. Якщо цю функцію використовувати без аргументу, то значення, яке поверне функція буде *випадкове дійсне число* в межах $[0; 1)$. Для отримання щоразу випадкових чисел потрібно у програмі перед використанням функції `random` виконати процедуру `randomize`.

Для тестування, тіло основної програми опишемо наступним чином:

```
Begin
    make_array;
    writeln('Створений масив:');
    print_array;
```

```
    sort_array;
    writeln('Відсортований масив:');
```




```
    print_array;
End.
```

Виконавши написану програму, будемо мати щось подібне до:

```
Створений масив:
  878  749  625  816  746  815   36  780  538  307
Відсортований масив:
   36  307  538  625  746  749  780  815  816  878
```

Варто розуміти, що результат виконання програми щоразу буде різним, так як ми використовуємо елементи роботи з випадковими числами.

Спробуйте поекспериментувати з різними наборами даних — щоб остаточно зрозуміти й засвоїти суть цього методу сортування.

Сортування вибором

Сортування методом вибору є одним з простих у реалізації методів сортування.

Також, як і метод сортування бульбашкою, цей метод не потребує додаткової пам'яті. В деяких ситуаціях дає кращі результати ніж сортування бульбашкою. Примітним є те, що в житті ми здебільшого й користуємося цим методом сортування коли нам потрібно відсортувати набір речей. Наприклад, **розкладаючи гральні карти однієї масті по старшинству**.

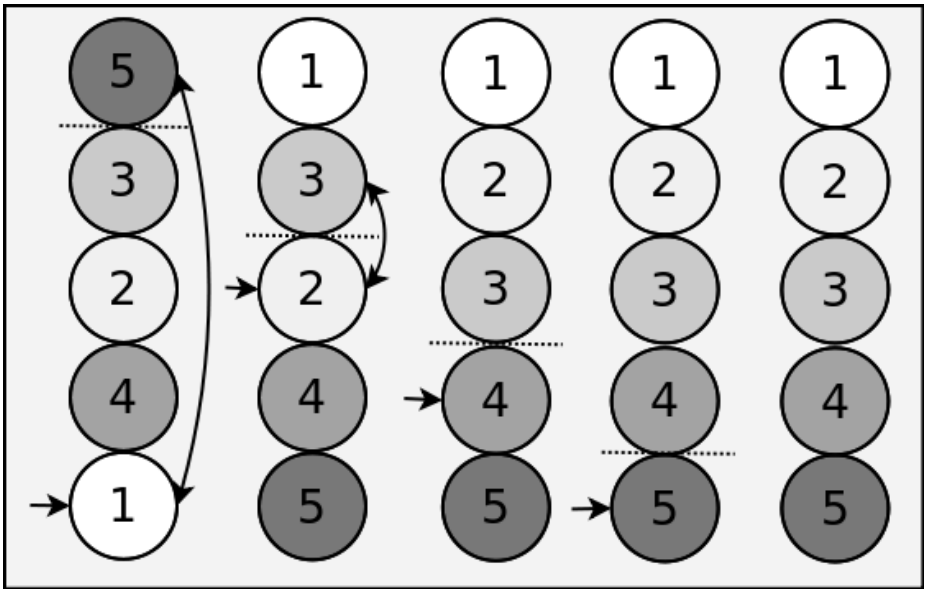
Яким чином ми це робимо?

Спершу ми знаходимо найменшу карту (6) й кладемо її з лівого краю, окремо від решти карт. Далі, шукаємо найменшу карту з решти — 7. Знайшовши її — кладемо поруч біля 6. Таким чином у нас корти, ніби, розділилися на 2 групи: в першій групі карти відсортовані, а в другій — ні. Попередні кроки повторюємо, допоки на якомусь кроці не отримаємо лише одну групу карт — відсортованих.

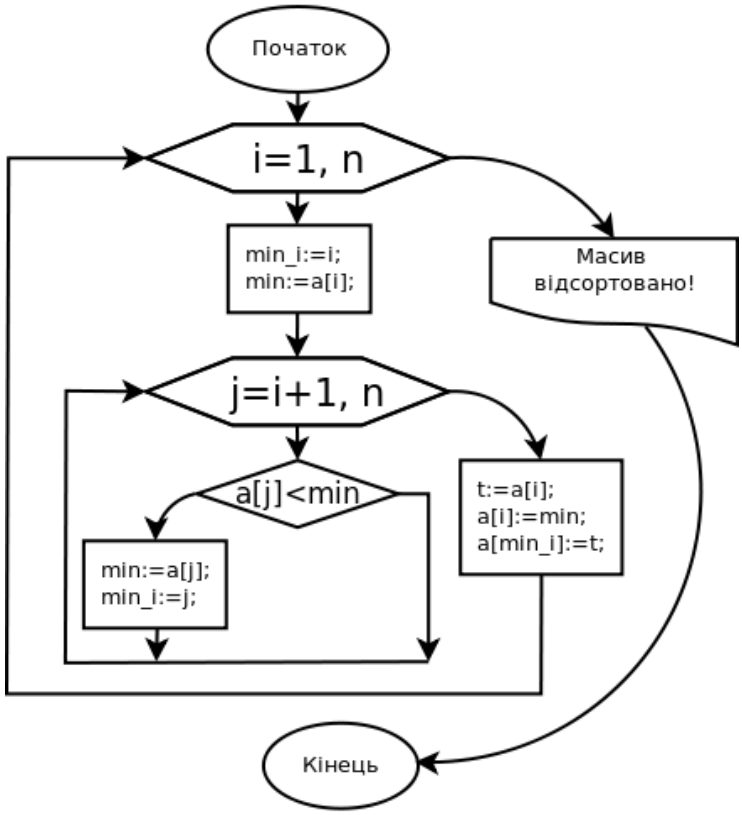
Опишемо словесно алгоритм сортування вставками:

1. Знаходимо в масиві найменше значення
2. Міняємо знайдене найменше значення місцями із першим значеннями у масиві
3. Повторюємо кроки 1-2, доки список не завершиться (починаючи з другої позиції).

Тобто, для набору (5, 3, 2, 4, 1) процес сортування відбуватиметься наступним чином:



Блок схема алгоритму матиме вигляд:



Реалізовуючи алгоритм сортування вибором, будемо використовувати попередні позначення й описані вище процедури заповнення та виведення масиву.

```
procedure sort_array;
{сортування методом вибору мінімального елемента}
var i, j, min, min_i, t : integer;
begin
    for i:=1 to n do
    Begin
        min_i:=i;
        min:=a[i];
        for j:=i+1 to n do
            if (a[j]<min) then
            Begin
                min:=a[j];
                min_i:=j;
            End;
        t:=a[i];
        a[i]:=min;
        a[min_i]:=t;
    End;
end;
```

В змінних `min` та `min_i` зберігаємо значення найменшого елементу та індекс цього ж найменшого елементу відповідно.

Задача 1. Медіана чисел

Ця задача (1141) взята з “E-Olimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/1141

Медіаною набору різних чисел називається таке число *m*, що кількість чисел, більших за *m*, рівна кількості чисел, менших за *m*. Наприклад, медіаною множини *{1, 4, 2, 5, 7}* буде *4*, так як існує два числа, більших *4* (числа *5* і *7*), і два числа, менших *4* (*1* і *2*). Множина *{1, 5, 8, 3}* не має медіани, так як жоден з її елементів не задовольняє наведеним вище умовам.

Технічні умови

Вхідні дані. У єдиному рядку вхідного файлу задано `n (n ≤ 1000)` різних натуральних чисел, кожне з яких не більше `1000`.

Вихідні дані. Виведіть значення медіани вхідних чисел. Якщо медіани не існує, вивести `-1`.

Приклад.

Введення	**Виведення**
`1 4 2 5 7`	`4`

Аналіз задачі

Алгоритм розв'язку задачі відбуватиметься в такі етапи:

- читання масиву елементів,
- сортування елементів по зростанню,
- знаходження медіани.

Виходячи з умови можна сказати, що який би не був набір парної кількості елементів, ми в будь-якому разі не зможемо знайти медіани. Отже, одразу після завершення читання даних ми повинні відслідкувати цей випадок і одразу виводити результат.

Якщо ж кількість чисел непарна, тоді необхідно буде здійснювати сортування. Для наочності, використаємо сортування “бульбашкою”.

Останнім етапом є знаходження медіани — вочевидь, медіана у відсортованому масиві, з непарною кількістю елементів міститиметься саме на позиції (n **div** 2)+1.

Програмування розв'язку задачі

Програмування розв'язку зрозуміле з вихідних кодів.

```
program mediana;
var m : array[1..1000] of integer;
  n, i, temp : integer;
  treba : boolean;
begin
  n:=0;
  while (not eoln) do
  Begin
      inc(n);
      read(m[n]);
  End;
  if (n mod 2 = 0) then writeln(-1)
  else
  Begin
      treba:=true;
      while treba do
      Begin
          treba:=false;
          for i:=1 to n-1 do
              if m[i]>m[i+1] then
              begin
                  temp:=m[i];
                  m[i]:=m[i+1];
                  m[i+1]:=temp;
                  treba:=true;
              end;
      End;
      writeln(m[(n div 2)+1]);
  End;

end.
```

Задача 3. Перестановка

Ця задача (1141) взята з “E-Olimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/354

Дано послідовність, що складається з N натуральних чисел. Написати програму, що визначає, чи є ця послідовність перестановкою перших N натуральних чисел.

Технічні умови

Вхідні дані. У єдиному рядку задано спочатку число N, а потім — N натуральних чисел через пропуск. N - не більше 10000, а кожне з чисел менше 2000000.

Вихідні дані. Вивести 0, якщо послідовність виявиться перестановкою, а якщо ні - мінімальне число, що не входить в цю послідовність.

Приклад.

Введення	**Виведення**
3 1 4 2	3

Аналіз задачі[14]

Цю задачу можна розв'язати кількома методами. Звісно, можна перевіряти входження кожного з чисел від 1 до n у вхідну послідовність; можна використати знання про суму арифметичної прогресії. Ми ж вдамося до іншого методу:

- зчитати масив чисел
- посортувати числа в порядку зростання
- 0-му елементу масиву присвоїти значення “0”
- від 0-го і до останнього елемента масиву перевіряти значення різниці сусідніх елементів масиву:
  - як тільки знайдемо два елемента, різниця між якими відмінна від одиниці — закінчуємо цикл перегляду елементів масиву й виводимо знайдений результат.

Програмування розв'язку задачі

```
var n, i, rez : longint;
    a : array[0..10000] of longint;

procedure sort_array;
```

14 Розв'язання задачі за [8].

```
{сортування методом вибору мінімального елемента}
var i, j, min, min_i, t : longint;
begin
    for i:=1 to n-1 do
    Begin
        min_i:=i;
        min:=a[i];
        for j:=i+1 to n do
            if (a[j]<min) then
            Begin
                min:=a[j];
                min_i:=j;
            End;
        t:=a[i];
        a[i]:=min;
        a[min_i]:=t;
    End;
end;
begin
   read(n);
   for i:=1 to n do read(a[i]);
   sort_array;
   a[0]:=0;   rez:=0;   i:=0;
   while (i<=n-1) and (rez=0) do
   Begin
       if a[i+1]-a[i]<>1 then rez:=a[i]+1;
       inc(i);
   End;
   writeln(rez);
end.
```

Задачі на самостійне опрацювання

1. http://acm.lviv.ua/fusion/viewpage.php?page_id=9&id=1017&

1017 — Ланцюг

Є N шматків ланцюга, кожен i -й з яких містить L_i ланок. Можна розгинати довільні ланки та потім згинати їх знову, з'єднуючи окремі шматки.

Завдання. Напишіть програму, що за кількістю шматків ланцюга N та кількістю ланок у шматках L_i визначає мінімальну кількість ланок, яку потрібно розігнути та зігнути знову, щоб з'єднати усі шматки в один ланцюг. Ланцюг не може мати розгалужень, тобто кожна його ланка повинна бути з'єднана з двома іншими ланками (крім двох ланок з країв ланцюга, що повинні бути з'єднані лише з однією ланкою).

Вхідні дані. В першому рядку знаходиться ціле число N ($2 \leqslant N \leqslant 10000$). В другому рядку знаходяться N цілих чисел L_i ($1 \leqslant L_i \leqslant 1000000000$), розділених пропуском.

Вихідні дані. В єдиному рядку повинно знаходитися ціле число — найменша кількість ланок, яку потрібно розігнути та зігнути знову, щоб отримати один ланцюг з усіх шматків.

2. Спробуйте розв'язати задачу **Перестановка** оптимальнішим методом, відмінним від описаного.

2.6. Довга арифметика

Задача 1. Везунчик.

Ця задача взята з **Завдань 4-го туру олімпіади NetOI-2003 Всеукраїнського Центру олімпіад школярів в інтернеті**, доступна в мережі за адресою:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=199

Сторінка для здавання на автоматичну перевірку задачі:

http://www.olymp.vinnica.ua/index_ua.php?lng=ua&cid=823&task=lucky

Задача Lucky. Герой олімпіад, десятикласник Василько Пупкін, прямуючи на 4-й тур NetOI-2002 купив в трамваї квиток з номером `N` та зберіг його. Через рік, в цьому ж трамваї, по дорозі на 4-й тур NetOI-2003, увінчаний минулими перемогами Василь купив квиток, сума цифр якого виявилась такою ж, як і в торішнього. Бажаючи перевірити своє щастя, Василь дізнався в кондуктора, що номери квитків - послідовний ряд натуральних чисел, квитки продаються строго по порядку номерів, і така сума цифр не зустрічалась в жодного з проданих між турами квитків. З яким номером купив квиток Василь вдруге? Номер квитка містить не більше `100` цифр і не починається з `0`.

Технічні умови. Ви вводите з клавіатури число `N` та виводите на екран єдине шукане число.

Приклад

Введення	**Виведення**
54	63

Аналіз задачі

Проаналізувавши задачу, бачимо, що перед нами стоїть задача за заданим

попереднім номером квитка знайти наступний по порядку квиток, сума цифр номера якого буде дорівнювати сумі цифр даного квитка.

Найлегше — розв'язувати цю задачу методом перебору, або ж, по суті, моделюванням процесу. Ми можемо, маючи початковий номер квитка N "продавати" почергово по одному усі квитки, при чому, при кожній "продажі" перевіряти суму цифр проданого квитка. В цьому випадку нас цікавитиме самий перший квиток, який матиме таку саму суму цифр.

Звернемо увагу на обмеження вхідних даних, а саме на те, що номер квитка може містити до 100 цифр. Число, що містить 100 цифр у своєму десятковому записі — не "поміститься" ні в які доступні нам типи даних. Отже, варто одразу цю задачу розв'язувати як задачу довгої арифметики. При чому, згідно з обраним методом розв'язання, бачимо, що нам потрібно буде запрограмувати лише одну процедуру, а саме процедуру додавання одиниці до "довгого числа".

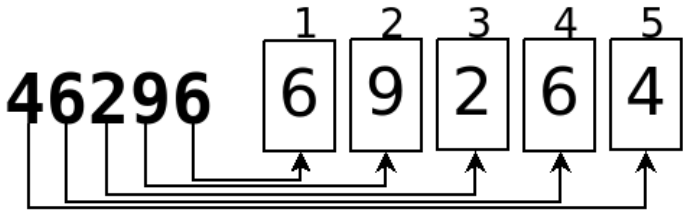
Також нам потрібно буде для вирішення задачі реалізувати наступні процедури:

- читання довгого числа
- виведення довгого числа
- додавання до довгого числа одиниці
- знаходження суми цифр довгого числа.

Для того, щоб код програми був зрозумілішим, спробуємо написати його, вдаючись до процедурного підходу, й проектуючи "знизу-вверх", тобто спершу опишемо всю систему загалом, а потім, поступово "опускаючись вниз" поописуємо усі складові елементи — необхідні процедури та функції.

Програмування розв'язку задачі

Для роботи з довгим числом будемо використовувати масив зі ста елементів, кожен з яких набуватиме значення від 0 до 9. При чому, число в масиві будемо записувати у зворотньому порядку.



Тобто, в першому елементі масиву будуть міститися одиниці, в другому — десятки, в третьому — сотні, й так далі. Для цього опишемо наступний тип даних:

```
type chuslo = 0..9;
var m : array[1..100] of chuslo;
    n : integer;
```

Змінна цілочисельного типу `n` — необхідна для збереження “довжини числа”, тобто кількості цифр в числі. Взагалі кажучи, для розв'язання задачі нам достатньо лише наявності одного такого довгого числа, а отже, немає змісту придумувати якісь значно складніші структури даних.

Процедура читання числа матиме вигляд:

```
procedure init;
var c : char;
    x, kod, p, t : byte;
begin
    p:=1;
    while (not eoln) do
    Begin
         read(c);
         val(c,x,kod);
         m[p]:=x;
         inc(p);
    End;
    n:=p-1;
    for p:=1 to (n div 2) do
    Begin
         t:=m[p];
         m[p]:=m[n-p+1];
         m[n-p+1]:=t;
    End;
end;
```

Де зчитування відбувається посимвольно, в циклі `while`, до кінця стрічки (умова `not eoln`). Для цього ми використовуємо процедуру `val`.

Процедура `val` використовується для перетворення стрічки в якій містяться числа в цілий чи дійсний тип. Ця процедура потребує наявності трьох аргументів:

```
val(number_string, number, error_pos);
```
Де

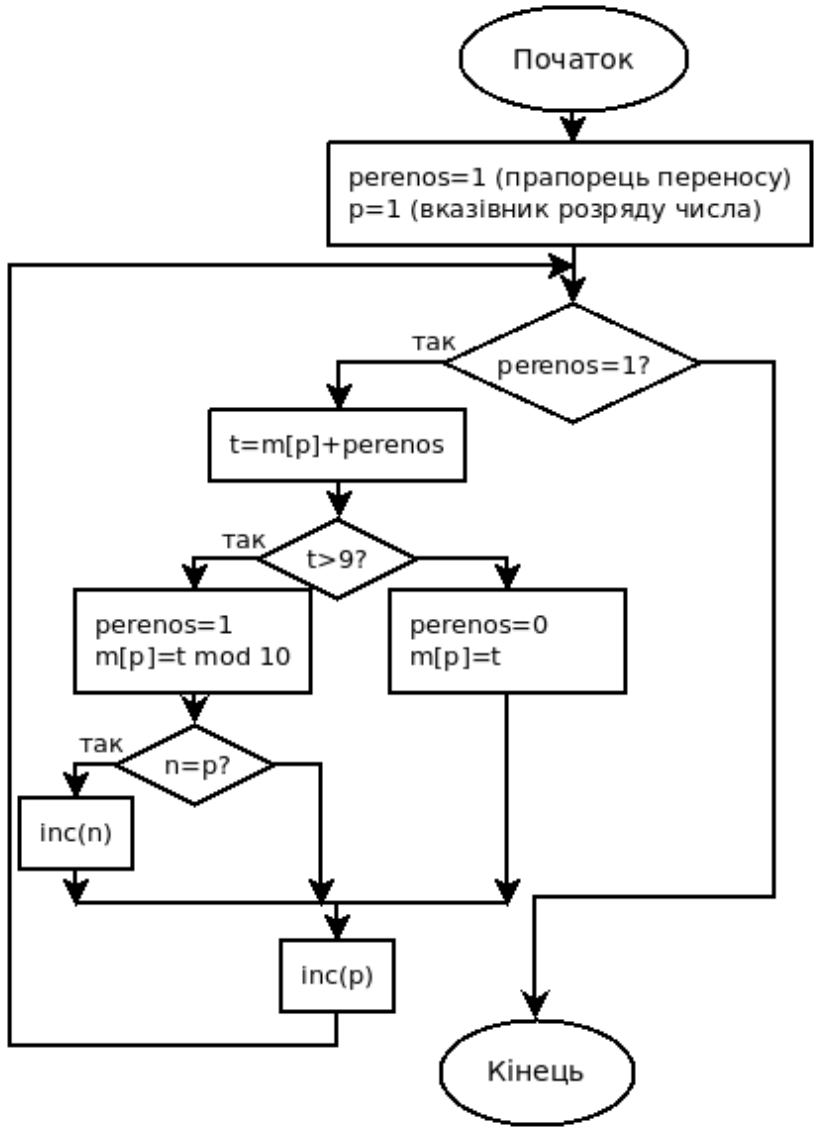
- `number_string` — це стрічка, ***з якої*** потрібно отримати числове значення
- `number` — дійсна, або цілочисельна змінна, ***в яку*** запишеться отримане значення
- `error_pos` — перший символ стрічки `number_string`, на якому сталася помилка перетворення стрічки в число. Якщо помилки не сталося, і число успіщно було конвертовано, то `error_pos` встановлюється `0`.

Після того, як ми прочитали все число, ми це число переписуємо у зворотньому порядку, в той же самий масив, в якому це число й містилося. Варто звернути увагу на межі циклу, а саме, від `1` до `(n div 2)`, так якщо здійснювати цикл до `n`, то результату ніякого не отримаємо — відбудеться `n` перестановок елементів масиву, при чому, після середини масиву, щойно переставлені елементи "ставатимуть" на свої місця. По виконанню процедури, в змінній `n` в нас міститиметься кількість елементів масиву `m`, тобто кількість цифр в десятковому записі довгого числа.

Процедура виведення довгого числа зрозуміла з вихідних кодів*:*

```
procedure answer;
var p : byte;
begin
    for p:=n downto 1 do write(m[p]);
    writeln;
end;
```

Процедура *додавання до довгого числа одиниці* відбуватиметься по наступній



схемі: Додавання відбувається абсолютно ідентичне до додавання в стовпчик — завжди контролюємо, чи немає у нас перенесення одиниці у наступний розряд. А виходячи з того, що ми додаємо просто одиницю, то починаємо з того, що перенос в наступний розряд у нас уже є. Тобто, до розряду одиниць ми додаємо одиницю. Процедуру додавання можна завершити як тільки в нас зникне перенос в наступний розряд.

Перевірка `n=p` нам потрібна для контролювання ситуації, коли, наприклад, до `999` (де `n=3`) додавати `1`, ми матимемо `1000`, тобто перенесення в розряд тисяч одиниці, а отже, й кількість цифр в записі числа збільшиться на одиницю (`n=4`).

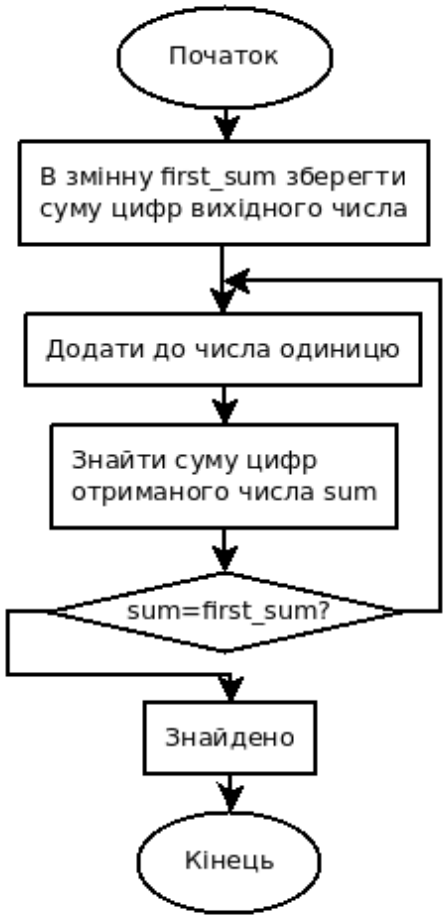
```
procedure plus_one;
var p, t, perenos : byte;
```

```
begin
    p:=1;
    perenos:=1;
    while (perenos=1) do
    Begin
        t:=m[p]+perenos;
        if t>9 then
        Begin
            perenos:=1;
            m[p]:=t mod 10;
            if n=p then inc(n);
        End
        else
        begin
            perenos:=0;
            m[p]:=t;
        end;
        inc(p);
    End;
end;
```

Процедура знаходження суми цифр довгого числа є еквівалентна задачі знаходження суми елементів масиву:

```
function sum : integer;
var p : byte;
    rez : integer;
begin
    rez:=0;
    for p:=1 to n do inc(rez, m[p]);
    sum:=rez;
end;
```

Маючи усі необхідні процедури й функції, основну процедуру програми можна зобразити наступною схемою: Що не складає труднощів запрограмувати:




```
procedure find;
var first_sum : integer;
begin
    first_sum:=sum;
    repeat
        plus_one;
    until first_sum = sum;
```

```
end;
```

Таким чином, розв'язок задачі міститиме усі описані вище процедури й функції, а основна програма матиме компактний вигляд:

```
begin
    init;
    find;
    answer;
end.
```

Легко зрозуміти, що на великих даних, складена програма буде надто довго працювати, так як потребуватиметься *велике число* додавань одиниці до великого числа.

При здачі задачі, маємо:

```
Прошло тестов: 20 из 33.
Набрано баллов: 60 из 100.
```

Бачимо, що усі не пройдені тести помічені позначкою “Time Out”, а не “Wrong answer”. Це з одного боку гарні новини, а з іншого — погані. Гарні тому що на жодному тесті (окрім, покищо тих, на яких “Time Out”) ми не отримали повідомлення про неправильну відповідь, а отже, можна сподіватися на те, що описаний алгоритм є коректний; і погані новини — це те, що можна зробити висновок про те, що далеко не не всіх тестах скалдена програма може дати результат за вказаний обсяг часу.

Отже, потрібно переглянути підхід до розв'язання задачі.

Метод *“влоб”* не проходить, отже варто придумати щось хитріше. (див. Додаток)

Задача 1. Велика точність

Ця задача (11) з “E-Oplimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/11

Дано раціональний дріб $\frac{m}{n}$. Запишіть його у вигляді десяткового дробу з точністю до `k` знаків після крапки.

Технічні умови

Вхідні дані. В одному рядку записано `3` числа `m, n, k (0 < m, n ≤ 100, 0 ≤ k ≤ 1000)`.

Вихідні дані. Вивести k точних значущих цифр після десяткової крапки шуканого числа.

Приклад

Введення	Виведення
1 2 3	0.500

Аналіз задачі

Ця задача була віднесена до розділу “Довга арифметика”, так як для вирішення цієї задачі застосовуються дуже схожі принципи й підходи, як і до операцій над довгими числами.

З умови зрозуміло, що нам потрібно виконати операцію ділення двох чисел, при чому, з вказаною точністю. Одразу варто відзначити, що про заокруглення не йдеться мова.

Щодо вхідних даних, то нам дається чисельник та знаменник m та n раціонального дробу $\frac{m}{n}$ та необхідна точність k.

Ділення будемо виконувати за схемою, яку вчили в школі. Цікавим і на перший погляд несподіваним моментом є те, що ми будемо виконувати ділення, користуючись лише цілими числами.

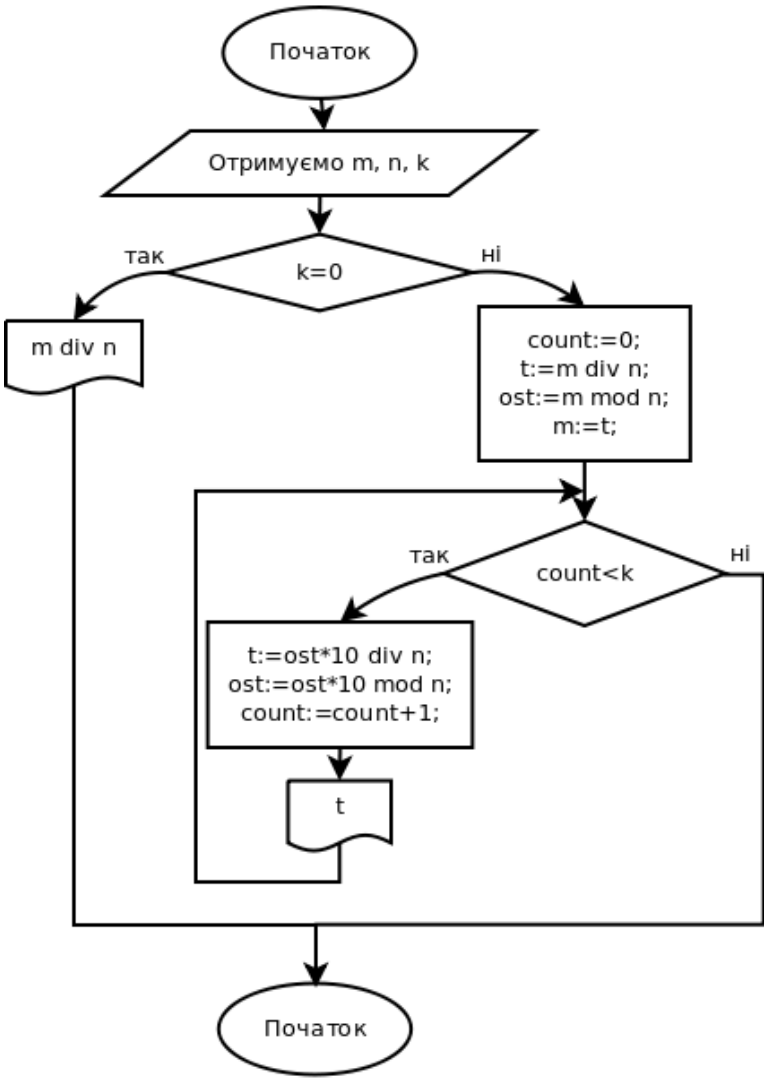
На першому етапі, потрібно одразу опрацювати випадок коли k=0, тобто, знаків після крапки не передбачається. Отже варто вивести всього лише значення цілочисельного ділення m на n. Інакше, потрібно виконувати ділення.

Суть ділення

1. Підготовчий етап:
    - покладаємо кількість обчислених знаків (count) рівною нулю
    - допоміжній змінній (t) присвоюємо значення результату цілочисельного ділення m на n
    - змінній остачі (ost) покладаємо значення результату цілочисельного ділення m на n
    - знаменник залишається той самий — n, а чисельник змінюється на t — тобто, виконуємо присвоєння змінній m значення t.

2. Крок ділення (який виконується допоки не буде досягнено заданої точності)

- як при діленні в стовпчик, ми домножували “знесене число” на 10, так і тут — домножуємо `ost` на 10 і на кожному кроці переприсвоюємо значення змінних `t` і `ost` (знаходимо частку від ділення та остачу):
  - `t:=ost*10 div n;`
  - `ost:=ost*10 mod n;`
- для контролю досягнення бажаної точності — збільшуємо на одиницю змінну, що відповідає за кількість знайдених знаків після крапки (`count`).



Програмування розв'язку задачі

```
Program p11_precision;
var m, n : byte;
   k, count, t, ost : integer;
begin
   read(m, n, k);
   if k=0 then writeln(m div n)
   else
   Begin
        count:=0;
        t:=m div n;
        ost:=m mod n;
        write(t,'.');
        m:=t;
        while count<k do
        Begin
             t:=ost*10 div n;
             ost:=ost*10 mod n;
             write(t);
             inc(count);
        End;
        writeln;
   End;
End.
```

Задача 3. Додавання

Ця задача (265) з “E-Oplimp”, доступна в мережі за адресою:

http://www.e-olimp.com.ua/ua/problems/265

Знайти суму двох цілих невід'ємних чисел *A* та *B*.

Технічні умови

Вхідні дані. У вхідному файлі задано два цілих невід'ємних числа A та B (A, B <= 10^{10000}), кожне у своєму рядку.

Вихідні дані. У вихідний файл вивести одне число, яке дорівнює сумі A та B.

Приклад

Введення	Виведення
3 5	8

Аналіз задачі

З умови задачі стає зрозуміло, що жоден з цілочисельних типів даних в Паскалі не передбачає операції над числами, кількість знаків яких досягає 10 тисяч. Зрозуміло, що нам доведеться самим вручну описувати роботу з такими числами.

Для розв'язання потрібно виконати наступні кроки:

- читання числа A
- читання числа B
- виконання додавання A + B = C
- виведення C

Для реалізації поставлених цілей використаємо структуру даних, схожу як в задачі Lucky:

```
type     digit = array[1..10001] of shortint;
var a, b, c : digit;
    na, nb, nc : longint ;
```

Де a і b — массиви-числа, які ми будемо додавати, а na и nb — відповідно їх довжини. c — результуюче число-масив, nc — його довжина.

Програмування розв'язку задачі

Програмування розв'язку задачі зрозуміле з вихідного коду:

```
procedure long_read(var a: digit; var n: longint);
{процедура читання довгого числа}
var c : char;
    code : integer;
    x : byte;
begin
{заповнюємо масив нулями}
    fillchar(a, sizeof(a), 0);
{довжина покищо рівна 0}
    n:=0;
{допоки не кінець стрічки}
    while not eoln do
    begin
    {зчитуємо по одному символу}
```

```
            read(c);
            inc(n);
            val(c, x, code); {і додаємо символ}
            if code=0 then a[n]:=x;
        end;
end;

procedure invert(var a: digit; n: int);
{процедура "обретання" довгого числа}
var i : integer;
    x : shortint;
begin
{йдемо до середини}
    for i:=1 to n div 2 do
    begin
    {попарно міняємо символи місцями}
        x:=a[i];
        a[i]:=a[n-i+1];
        a[n-i+1]:=x;
    end;
end;
```

Для зберігання одиниці, що переходить в наступний (старший) розряд заведемо окрему змінну зміщення (zm).

```
procedure add(a, b : digit; var c : digit;
                    na, nb : longint; var nc: longint);
{процедура додавання}
var i : integer;
    x, zm : byte;
    h : longint;
begin
{Довжина відповіді не може бути менше максимальної з довжин}
    if na>nb then nc:=na
    else nc:=nb;
{Спочатку, зміщення рівне 0}
    zm:=0;
{Розвертаємо масиви a і b}
    invert(a, na);
    invert(b, nb);
{Основний цикл}
    for i:=1 to nc do
    begin
    {Сумуємо цифри на поточних позиціях + зміщення}
        x:=a[i]+b[i]+zm;
    {зміщення враховано - зануляємо}
```

```
        zm:=0;
    {якщо x – двозначне число}
        if x>=10 then
        begin
        {робимо зміщення}
            zm:=1; {все одно, що sm:=x div 10;}
        {робимо число однозначним}
            x:=x-10; {все одно, що x:= x mod 10;}
        end;
    {записуємо відповідь в поточну позицію}
        c[i]:=x;
    end;
    {випадок коли na=nb і nc=na+1}
    if zm<>0 then
    begin
        inc(nc);
        c[nc]:=1;
    end;
{розвертаємо відповідь}
    invert(c, nc);
end;
```

Тіло програми матиме вигляд:

```
BEGIN
    long_read(a, na);
    long_read(b, nb);
    add(a, b, c, na, nb, nc);
    for i:=nc downto 1 do write(c[i]);
    writeln;
END.
```

Задачі на самостійне опрацювання

1. ***http://www.e-olimp.com.ua/ua/problems/266***

   Порівняння

   Порівняйте два числа A та B.

   Технічні умови. Вхідні дані. У вхідному файлі задано два цілих невід`ємних числа A та B (A, B <=1010000) кожне у своєму рядку.

   Вихідні дані. У вихідний файл виведіть "<", якщо A < B, "=", якщо A = B та ">", якщо A> B.

2. ***http://www.e-olimp.com.ua/ua/problems/272***

   Добуток

   Знайти добуток чисел A та B.

Технічні умови. Вхідні дані. У вхідному файлі задано два цілих невід'ємних числа A та B (A, B <=1010000), кожне у своєму рядку.

Вихідні дані. У вихідний файл вивести одне число, яке дорівнює добутку A та B.

3. ***http://acm.lviv.ua/fusion/viewpage.php?page_id=9&id=1015&rid=4cacefc7eb5bf***

1015 - Числа Фібоначчі

Числа Фібоначчі визначаються рекурентною формулою:

$$f_0 = 0;\ f_1 = 1;\ f_n = f_{n-1} + f_{n-2};$$

Початок послідовності має вигляд $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots.$

Вхідні дані. В єдиному рядку знаходиться число N ($0 \le N \le 10000$).

Вихідні дані. Виведіть N-те число Фібоначчі.

Додатки

1. ASCII-таблиця

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
10	LF	SUB	*	:	J	Z	j	z
11	VT	ESC	+	;	K	[	k	{
12	FF	FC	,	<	L	\	l	/
13	CR	GS	-	=	M	]	m	}
14	SO	RS	.	>	N	^	n	~
15	SI	US	/	?	O	_	o	DEL

Взято з ***http://en.wikipedia.org/wiki/ascii***

2. Розв'язок задачі Lucky

Вдамося до аналітичного методу. Просто проаналізуємо що і як є. Ось наприклад, для випадку вхідних даних — 54 — що нам потрібно змінити? Змінами в цьому числі лише кількість одиниць (а саме 4), ми ніяким чином не досягнемо шуканого результату. Отже, потрібно змінювати кількість десятків. Зменшувати кількість десятків — ми не можемо (власне, це стосується усіх розрядів), отже варто збільшити розряд десятків на мінімальну кількість, тобто 1 (матимемо 5+1=6). А далі, для досягнення необхідної суми цифр — зменшимо розряд одиниць на 1 (матимемо 4-1=3). Отже, отримали число 63.

Для випадку числа уже 540 — попередні викладки не матимуть місця, так як в

розряді одиниць від нуля ми нічого не віднімемо.

Використаємо наступну структуру даних:масив

```
sumHash : array[1..101] of integer
```

в якому будемо зберігати суми цифр числа, при чому:

- в першому елементі — сума з 1-ої по 1-шу цифр числа
- в другому елементі — сума з 1-ої по 2-гу цифр числа
- ...
- в n-ому елементі — сума цифр з 1-ої по n-ту цифр числа.

```
program Lucky;
type mystring = String[101];

var sumHash : array[1..101] of integer;

function reverse(strIn : mystring) : mystring;
{"перевертаємо" стрічку}
var tmpstr : mystring;
    i : integer;
begin
    tmpstr:='';
    for i:=length(strIn) downto 1 do
        tmpstr:=tmpstr+strIn[i];
    reverse := tmpstr;
end;

procedure calcHash(var strIn : mystring);
{обчислення сум}
var tmpSum : integer;
    i : integer;
begin
    tmpSum:=0;
    for i:=1 to Length(strIn) do
    begin
        tmpSum:=tmpSum + (ord(strIn[i])-ord('0'));
        sumHash[i]:=tmpSum;
    end;
end;

function getSum(strIn : mystring) : integer;
{обчислення суми цифр}
var tmpSum:integer;
    i: Integer;
```

```
begin
  tmpSum:=0;
  for i:=1 to Length(strIn) do
     tmpSum:=ltmpSum + (ord(lstrIn[i])-ord('0'));
  getSum :=tmpSum;
end;

var strIn, strOut, strWork : mystring;
    i, j, sum : Integer;
begin
    readln(strIn);
{"обертаємо" стрічку strIn}
    strWork:=reverse(strIn);
{шукаємо хеш-суму}
    calcHash(strWork);
{перескакуємо усі нулі}
    j:=1;
    while (j<=length(strWork)) and (strWork[j]='0') do inc(j);
    inc(j);
{перескакуємо усі дев'ятки}
    while (j<=Length(strWork)) and (strWork[j]='9') do inc(j);
{заповнюємо нулями всі позиції менші за j-1}
    for i:=1 to j-1 do
        strWork[i]:='0';
{якщо виходимо за межі числа, то додаємо старший розряд}
    if (j > Length(strWork)) then
        strWork:=strWork+'0';
{збільшуємо на 1-цю розряд на який вказує j}
    strWork[j]:=chr(ord(strWork[j])+1);
{сума всіх попередніх чисел зменшилась на 1}
    sum:=sumHash[j-1]-1;
{формуємо найменше нове число виходячи з суми яка утворилась}
    j:=1;
    while (sum>9) do
    begin
        strWork[j]:='9';
        inc(j);
        sum:=sum-9;
    end;
    strWork[j]:=chr(ord('0')+sum);
{отримали результат в оберненому вигляді, отже "перевертаємо"}
    strOut:=reverse(strWork);
    writeln(strOut);
end.
```

Список використаної літератури

1. Митчел М., Оулдем Дж., Самьюэл А. Программирование для Linux. Профессиональный подход.: М.: Издательский дом "Вильямс", 2003.-288 с.

2. Корман Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-е издание.: Издательский дом "Вильямс", 2005. - 1296 с.

3. Допомога, режим доступу: ***http://acm.lviv.ua/fusion/readarticle.php?article_id=24*** [станом на 01.10.2010]

4. Michaël Van Canneyt .Free Pascal : Reference guide., режим доступу: ***http://www.freepascal.org/docs-html/ref/refsu5.html#x27-260003.1.1*** [станом на 01.10.2010]

5. Michaël Van Canneyt .Free Pascal :Reference guide. 3.1.2 Real types, режим доступу: ***http://www.freepascal.org/docs-html/ref/refsu6.html#x28-310003.1.2***, [станом на 01.10.2010]

6. Семотюк В. Програмування у середовищі Турбо Паскаль.: Львів: БАК, 2000. - 248с.

7. Збірник олімпіадних задач з програмування (умови, вказівки та розв'язки).: Київ, 2004. - 85 с.

8. І.В.Гісь Розв'язання задач з програмування (для самопідготовки).: Луцьк, 2000. - 74 с.