**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Державний університет телекомунікацій**

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Методичні вказівки з дисципліни «Інформатика» призначені для студентів всіх форм навчання і напряму підготовки 050903Телекомунікації, 050901 Радіотехніка, освітньо-кваліфікаційного рівня - бакалавр, спеціальності:

7.05090302 Телекомунікаційні системи та мережі

**Методичний посібник розроблений**

Старшим викладачем Придибайло О.Б.

Київ – 2015

**ВСТУП**Будь-яка людина щодня зустрічається з безліччю повсякденних і  
професійних завдань. Для вирішення багатьох з них існують  
певні правила (інструкції, розпорядження), що пояснюють, як вирішувати  
певне завдання. У процесі вирішення можна застосовувати готові правила  
або формулювати власні. Чим точніше і зрозуміліше описані правила  
вирішення завдань, тим швидше людина опанує ними і буде ефективніше їх  
застосовувати. Вирішення багатьох завдань людина передає технічним пристроям -ПК, автоматам, роботам і т. Д. Їх застосування пред'являє дуже суворі вимоги до точності опису правил і послідовності виконання  
дій. Тому розробляються спеціальні алгоритми для чіткого і  
суворого опису різних правил.

Алгоритмізація − це розділ інформатики, що вивчає методи і прийоми  
побудов алгоритму, а також їх властивості. Вона є основним, базовим  
компонентом комп'ютерної грамотності в сучасному комп'ютерному світі.  
Для досягнення позитивних результатів важливу роль відіграє вміння  
розробляти оптимальний алгоритм вирішення поставленого завдання, що  
вимагає від виконавця наявності певних навичок алгоритмізації та  
системного аналізу, а також знання математики, фізики, хімії, економіки та  
інших дисциплін. Основу діяльності фахівця практично будь-якій області становить вміння ставити завдання, розробляти алгоритми, отримувати рішення, проводити аналіз отриманих даних і робити висновки. Тому в своїй  
майбутньої професійної діяльності студенти повинні вміти грамотно  
застосовувати персональний комп'ютер (ПК) для вирішення наукових і  
виробничих завдань. У даному посібнику розглянуто основні структури алгоритмів і типові прийоми алгоритмізації. Наведені приклади і завдання,  
охоплюють основні типи обчислювальних процесів.

**ОСНОВНІ ЕТАПИ РОЗВ'ЯЗАННЯ ЗАВДАНЬ З ВИКОРИСТАННЯМ КОМП'ЮТЕРА**

Створення комп'ютерної програми - дуже складний, багатоетапний і  
трудомісткий процес. Він включає в себе послідовність дій від  
постановки завдання до одержання рішення.

1. Загальна формулювання завдання. На цьому етапі завдання формулюється в змістовних термінах, визначаються вхідні і вихідні дані завдання.

2. Математичне формулювання завдання. На цьому етапі визначаються  
математичні величини, які будуть описувати завдання, а також  
математичні зв'язки між ними, т. е. складається математична модель.  
Неправильна чи погана модель прирече на невдачу весь подальший проект,  
тому цей етап є вкрай важливим.

3. Вибір методу рішення. Виходячи як з суб'єктивних причин (знання  
тих чи інших математичних методів), так і об'єктивних (наявні  
ресурси), з великої кількості математичних методів вибирається той,  
який доцільно використовувати для вирішення поставленого завдання.

4. Складання алгоритму рішення. На цьому етапі повинна чітко  
простежуватися зв'язок з попереднім. В ході нього розробляється  
ефективний алгоритм, т. е. такий, реалізація якого потребуватиме  
найменшої кількості ресурсів комп'ютера.

5. Складання та налагодження програми. На цьому етапі застосовуються  
основні правила запису та перетворення команд, записаних на  
природній мові, на мову машинних кодів.

6. Тестування програми. Відбувається підтвердження або  
спростування правильності роботи алгоритму. Для цього, як правило,  
вирішуються завдання з такими вихідними даними, для яких відомо  
достовірне рішення, або застосовуються непрямі свідчення.

7. Розв'язання поставленого завдання і представлення результатів. на  
даному етапі здійснюється зручний і наочний виведення результатів.  
При вирішенні конкретних завдань деякі з цих етапів можуть  
виключатися самою постановкою завдання. Для кожного з етапів створення та використання програми існують певні прийоми забезпечення  
якості програми.

Велику роль у створенні продуктів високої якості відіграє глибина і ретельність опрацювання схеми алгоритму. На етапі розробки алгоритму рекомендується дотримуватися наступних правил його складання:

1. Алгоритм повинен бути максимально простий і зрозумілий.  
2. Алгоритм повинен складатися з дрібних кроків.  
3. Складне завдання повинна розбиватися на досить прості, легко   
сприймаються частини (блоки).

4. Логіка алгоритму повинна спиратися на мінімальне число досить  
простих базових керуючих структур.

У підсумку процес розробки алгоритму має бути спрямований на  
отримання чіткої структури алгоритмічних конструкцій.

**ПОНЯТТЯ АЛГОРИТМА**

Поняття алгоритму належить до основних понять сучасної  
обчислювальної математики та інформатики.  
Алгоритм - описана на деякій мові точна кінцева система  
правил, що визначає зміст і порядок дій над деякими  
об'єктами, суворе виконання яких дає рішення поставленого завдання.  
Або (більше коротко):

*Алгоритм -* це строго певна послідовність дій, необхідних для вирішення даного завдання.

Розробити алгоритм рішення означає розбити задачу на  
послідовно виконуються етапи. Можна сказати, що алгоритм описує  
процес перетворення вихідних даних в результати, т. к. для вирішення  
будь-якої задачі необхідно:

1) ввести вихідні дані;

2) перетворити вихідні дані в результати (вихідні дані);

3) вивести результати.

Алгоритм призначений завжди для певного *виконавця* - людини,  
робота, комп'ютера, мови програмування і т. д. Уміння виконувати певні команди є властивістю, що характеризує будь-якого виконавця. Сукупність усіх команд, які даний виконавець може виконувати, називається системою команд виконавця. Відповідно алгоритм описується в командах певного виконавця, який буде його реалізовувати. Об'єкти, над якими виконавець може вчиняти дії, утворюють середу виконавця. Розробка алгоритму розв'язання задачі - це розбиття задачі на послідовно виконуються етапи. Результати виконання попередніх етапів можуть використовуватися при виконанні наступних. Зміст кожного етапу і порядок виконання етапів повинні бути чітко вказані. Окремий етап алгоритму має бути простим і зрозумілим без пояснень, або являти собою іншу, більш просту задачу, алгоритм рішення якої відомий (розроблений заздалегідь). Тому опис алгоритму рішення завдання виконується у відповідності з наступними правилами:  
Визначаються вихідні дані завдання.

1. Процес рішення задачі розбивається на етапи, зрозумілі і  
однозначні для виконавця.

2. Вказується порядок, в якому виконуються етапи, а також ознака завершення процесу.

3. Визначається, що є результатом розв'язання задачі.

**ВЛАСТИВОСТІ АЛГОРИТМА**

Значення слова «алгоритм» є синонімом таких понять, як «набір  
інструкцій »,« послідовність дій »,« метод ». Однак, на відміну від  
них, алгоритм характеризується певними властивостями. властивості  
алгоритму - це набір властивостей, що відрізняють алгоритм від будь-яких приписів та забезпечують його автоматичне виконання. алгоритм має  
наступним набором основних властивостей: дискретністю, масовістю,  
формальністю, результативністю, визначеністю.

*Дискретність* (розривність) - це властивість алгоритму, що характеризує  
його структуру: кожен алгоритм складається з окремих закінчених дій,  
т. е. «ділиться на кроки».

*Масовість* - придатність алгоритму до всіх завдань розглянутого  
типу, при будь-яких вихідних даних.

*Визначеність* (детермінованість, точність) - властивість алгоритму,  
вказує на те, що кожен крок алгоритму має бути строго визначений  
і не допускати різних тлумачень; також строго повинен бути визначений  
порядок виконання окремих кроків.

*Результативність* - властивість, яке у тому, що будь-який алгоритм  
повинен завершуватися за кінцеве (нехай навіть дуже велике) число кроків.  
 *Формальність* - це властивість вказує на те, що будь-який виконавець,  
здатний сприймати і виконувати інструкції алгоритму, діє  
формально, т. е. відволікається від змісту поставленого завдання і лише строго виконує інструкції. Іншими словами, механічно виконуючи всі  
зазначені в алгоритмі етапи в необхідному порядку, виконавець може завжди  
правильно вирішити завдання.

Процес розробки алгоритму називається **алгоритмізацією** і вимагає  
чіткого і повного розуміння завдання. сутність алгоритмізації  
обчислювального процесу проявляється в наступних діях, що відображають  
його властивості:

- Виділення закінчених частин обчислювального процесу;  
- Формальної записи кожного з них; - Призначення певного порядку виконання виділених частин;

- Перевірки правильності обраного алгоритму з реалізації заданого  
методу обчислень.

**СПОСОБИ ОПИСУ АЛГОРИТМІВ**

На будь-якій стадії існування алгоритми представляють за допомогою  
конкретних образотворчих засобів, склад і правила вживання яких  
утворюють конкретні способи або форми запису. До теперішнього часу  
склалися п'ять найбільш уживаних способів запису:  
словесний опис;

формульно-словесний опис;

псевдокод;  
графічний спосіб (блок-схема);

програма (спосіб опису за допомогою мов програмування).

**Словесний опис**

Словесний опис являє алгоритм - інструкцію про виконання  
дій в певній послідовності за допомогою слів і пропозицій  
природної мови. Форма викладу довільна і встановлюється  
розробником.  
Цей спосіб опису не має широкого розповсюдження, т. К. Строго не  
формалізуємо (під «формальним» розуміється те, що опис абсолютно  
повне і враховує всі можливі ситуації, які можуть виникнути в ході  
рішення), допускає неоднозначність тлумачення при описі деяких  
дій, страждає багатослівністю.

Приклад 1.

Алгоритм знаходження найбільшого спільного дільника (НСД) двох  
натуральних чисел.

1) задати два числа;

2) якщо числа рівні, то взяти будь-яке з них в якості відповіді і  
зупинитися, в іншому випадку продовжити виконання алгоритму;

3) визначити більше з чисел;

4) замінити більше з чисел різницею більшого і меншого з чисел;

5) повторити алгоритм з п. 2.

Приклад 2.

Алгоритм виконання домашнього завдання з математики.  
1) відкрити щоденник;

2) подивитися, що задано;

3) взяти підручник і зошит з предмета;

4) прочитати параграф і виконати письмові вправи;

5) порівняти отримані результати з відповідями в кінці підручника;

6) якщо відповіді збігаються, закрити зошит і підручник, якщо ні - повторити  
алгоритм з п. 4.

**Формульно-словесний спосіб**

Формульно-словесний спосіб запису дій містить формальні  
символи і вирази (формули) у поєднанні зі словесними поясненнями. Тобто алгоритм записується у вигляді тексту з формулами по пунктах,  
визначальним послідовність дій. Цей спосіб опису наочний,  
лаконічний, але не є суворо формальним.

Приклад 3.

Алгоритм обчислення наступного виразу: у = 2а - (х + 6).

1) ввести значення а і х;

2) знайти суму (х + 6);

3) знайти твір (2 \* а);

4) обчислити y як різниця y = 2a - (x + 6);

5) вивести у як результат обчислення виразу.

Приклад 4.

Алгоритм розв'язання задачі з геометрії.

1) Дано: висота трикутника AH = 2 см, основу трикутника BC = 5см.  
2) Знайти: площа S ΔABC.

3) Рішення: Площа трикутника знаходиться за формулою S *=1/2\*АН\*ВС*

4) Підставимо дані завдання: S *=1/2*\*2\*5  
5) Результат: S = 5см.

**Псевдокод**

Псевдокод являє собою опис структури алгоритму на  
природному, частково-формалізованому мовою, що дозволяє виявити  
основні етапи рішення задачі перед точної його записом на мові  
програмування.  
У псевдокоді використовуються деякі формальні конструкції і  
загальноприйнята математична символіка. Даний спосіб тісно пов'язаний зі  
структурним підходом до програмування. псевдокод займає  
проміжне положення між природним мовою та мовою програмування. Його застосовують переважно для того, щоб докладніше пояснити роботу програми, що полегшує перевірку правильності програми.

Приклад 5.

Алгоритм складання двох чисел.

**Псевдокод:**1) Введення двох чисел a і b.

2) Обчислення суми S = a + b.

3) Висновок S.

4) Кінець.

Приклад 6.

Алгоритм заповнення залікової відомості групи з 20 студентів (i - номер

студента).  
 **Псевдокод**

**початок циклу**

**для**

i від 1 до 20 з кроком 1

**повторювати:** 1.1. ввести прізвище студента

1.2. поставити оцінку.

**кінець циклу (кц)**

виведення відомості

**Графічний запис**

Графічний запис, або **блок-схема**, - опис структури алгоритму з  
допомогою геометричних фігур з лініями зв'язками, що показують порядок  
виконання окремих інструкцій. Опис алгоритмів за допомогою схем -  
один з найбільш наочних і компактних способів. Цей спосіб має ряд  
переваг перед іншими:

- наочність відображення базових конструкцій алгоритму;

- концентрація уваги на структурі алгоритму;

- використання принципу блочності при колективному вирішенні складної  
задачі;  
- перетворення алгоритму методом укрупнення (відомості до єдиного  
блоку) або деталізації (розбиття на ряд блоків);

- швидка перевірка розробленого алгоритму.

У блок-схемі кожному типу дій (введення вихідних даних,  
обчисленню значень виразів, перевірці умов, управлінню повторенням  
дій, закінченню обробки і т. п.) відповідає геометрична фігура,  
звана блоком. Блокові символи з'єднуються лініями переходів  
(стрілками), визначальними черговість виконання дій.  
Умовні графічні зображення, використовувані при побудові схем,  
називаються символами. Система символів і правила побудови алгоритмів  
визначені відповідними стандартами: блок-схема вибудовується в  
одному напрямку: або зверху вниз, або зліва направо, в порядку  
виконання дій.

Для наочності операції різного виду зображуються у схемі  
різними геометричними фігурами (таблиця).

**Програма**

Програма - це алгоритм, записаний у вигляді послідовності  
команд, зрозумілих ЕОМ (машинних команд). При запису алгоритмів у вигляді програм для ЕОМ використовуються мови програмування - системи  
кодування приписів та правила їх використання. Такі мови є  
штучними мовами зі строго визначеними синтаксисом і  
пунктуацією. Вони не допускають вільного тлумачення для своїх  
конструкцій, як це характерно для природної мови. існує велика  
кількість мов програмування, призначених для вирішення  
прикладних задач.

Для запису алгоритмів у вигляді програм характерна висока ступінь  
формалізації. Перед складанням програм найчастіше використовуються  
словесно-формульний або графічний способи.

Приклад 7.

Алгоритм складання двох чисел. Програма, записана мовою Turbo Pascal:  
**Program** Suma\_2;

**Var** s,a,b;

**Begin**

Read(a,b);

S:=a+b;

Write(s);

**End**.

**ОСНОВНІ АЛГОРИТМІЧНІ СТРУКТУРИ (ВИДИ АЛГОРИТМІВ)**

Алгоритми можна представляти як деякі структури, що складаються з окремих базових (тобто основних) елементів. Ці елементарні кроки  
об'єднуються в алгоритмічні конструкції. Залежно від особливостей своєї побудови алгоритми можна розділити на наступні групи:

1) лінійні (послідовні);

2) розгалужені;

3) циклічні;

4) рекурсивні.

Різноманітність алгоритмів визначаться тим, що будь-який алгоритм складається з фрагментів, кожен з яких представляє собою алгоритм одного з  
зазначених видів. Тому важливо знати структуру кожного з алгоритмів і  
принципи їх складання. Для вирішення будь-якої задачі можуть бути побудовані кілька алгоритмів, призводять до отримання результату її вирішення. З усіх можливих алгоритмів слід вибирати найкращий за різними критеріями: за точністю рішення задачі, тимчасових витратах, кількості етапів в алгоритмі, їх простоті і т. д.

**Лінійна алгоритмічна структура**

**Лінійним** називається алгоритм, у якому всі етапи рішення задачі  
виконуються рівно один раз і строго послідовно. Т. е. Лінійний  
(послідовний) алгоритм виконується в природному порядку його  
написання і не містить розгалужень і повторень.  
Прикладами лінійних алгоритмів є: алгоритм відмикання дверей -  
дістати ключ, вставити ключ у замкову щілину, відкрити замок; алгоритм  
заварювання чаю - дістати чайник, насипати в нього чай, залити окропом,  
настояти 5-10 хв.

Лінійний алгоритм застосовується при обчисленні арифметичного  
вирази, якщо в ньому використовуються тільки найпростіші алгебраїчні  
дії.

**Оператори реалізації лінійних обчислювальних процесів.**

**1. Складений оператор.**

*Складений оператор* - це послідовність довільних операторів програми, укладена в операторні дужки - зарезервовані слова **begin** . . . **end**. Складені оператори - важливий інструмент Турбо Паскаля, що дає можливість писати програми за сучасною технологією структурного програмування (без операторів переходу **GOTO**).

Мова Турбо Паскаль не накладає ніяких обмежень на характер операторів, що входять у складений оператор. Серед них можуть бути й інші складені оператори - Турбо Паскаль припускає довільну глибину їх вкладеності:

**begin**

**…….**

**begin**

**……..**

**begin**

………

………

**end**;

……..

**end;**

**………..**

**end**;

Фактично, увесь розділ операторів, обрамлений словами **begin** . . . **end**, являє собою один складений оператор. Оскільки зарезервоване слово **end** є закриваючою операторною дужкою, воно одночасно вказує й кінець попереднього оператора, тому ставити перед ним символ **";"** необов'язково, і далі у всіх прикладах ми не будемо цього робити. Наявність крапки з коми перед **end** у попередніх прикладах означало, що між останнім оператором і операторною дужкою **end** розташовується порожній оператор. Порожній оператор не містить ніяких дій, просто в програму додається зайва крапка з комою. В основному порожній оператор використовується для передачі управління в кінець складеного оператора.

**2.** **Введення даних (процедури READ та READLN)**

Як тільки під час виконання програми зустрічається процедура READ, робота програми призупиняється, і машина чекає введення даних. Коли дані введені, процес виконання програми продовжується.

Процедура введення має вигляд:

**READ (a1,a2,…,a*n*);** та **READLN (a1,a2,…,a*n*);**

де а1,а2,… - змінні, розділені комами, яким послідовно присвоюються значення, що вводяться.

Значення, що вводяться, вказуються через пробіл, ознакою закінчення введення є

натиснення клавіші <Enter>.

В **READLN** спочатку вводяться значення а1, а2,…, а потім переводиться курсор на новий рядок.

Допускається використання процедури READLN без параметрів. Вона використовується найчастіше **для зупинки** роботи програми. При цьому програма продовжить роботу при натисненні клавіші <Enter>.

**3. Оператор присвоєння.**

Оператор присвоєння в Паскалі наказує запам’ятати конкретне значення у вигляді змінної.

Форма запису оператора:

**<ідентифікатор>:=<вираз>;**

Типи значень ідентифікатора та виразу повинні **строго** співпадати, за виключенням того випадку, коли змінній типу *real* присвоюється значення виду типу *integer*.

**4. Виведення даних (процедури WRITE та WRITELN).**

Процедура виведення має вигляд

**WRITELN(a1,a2,…,a*n*);**

де а1,а2, … - список параметрів, які можуть бути або константами, або рядками символів, які містяться в апострофах.

**Приклад.** Якщо в програмі є така послідовність операторів:

a:=15;

writeln(‘Значення змінної А=’,a);

то у випадку:

* якщо змінна а - *integer* , то на екрані буде: *Значення змінної А=15;*
* якщо змінна а – *real,* то на екрані з'явиться *: Значення змінної А=1.5000…0Е+0001*

За замовчуванням процедура writeln виводиться на екран 15 десяткових розрядів для мантиси и 4 розряди для порядку.

*Процес роботи.*

**WRITELN.** Всі елементи списку параметрів перетворюється в символьний вигляд і об’єднуються в один рядок символів. Рядок починає виводитись з того місця на екрані, на якому в даний момент знаходиться вказівник курсору, і продовжується від нього вправо. Якщо символьний рядок не вміщується повністю на сторінці екрану(>80 позицій), остача символів переноситься на наступний рядок. *Після завершення виведення символів курсор переміщується на початок наступного рядка.* Дана процедура без параметрів використовується для виведення порожніх рядків.

**WRITE.** В цій процедурі курсор залишається на тому ж рядку екрану зразу за останнім виведеним символом.

Для виведення цілих та дійсних чисел можна після змінної через двокрапку вказувати формат, таким самим чином як і у системі АЛГОРИТМ. Для дійсних чисел формат складається з двох величин . Перша – загальне поле значення, що виводиться, друга – поле дробової частини.

**Розгалужена алгоритмічна структура**

На практиці рідко вдається уявити схему алгоритму розв'язання задачі в  
вигляді лінійної структури. Завдання може містити умову, залежно від  
якого обчислювальний процес йде з тієї чи іншої гілки, т. е. з'являється  
**розгалуження** алгоритму.

**Розгалуженою** (ветвящейся) називається конструкція, яка  
забезпечує вибір між двома альтернативами залежно від значення  
вхідних даних.

Або **розгалужений алгоритм** - це алгоритм, в якому в залежності від умови виконується або одна, або інша послідовність дій.

Приклади алгоритмів, що розгалужуються: якщо пішов дощ, то треба відкрити зонт; якщо квиток в театр коштує не більше ста гривень, то купити квиток і зайняти своє місце в залі, інакше (якщо вартість квитка більше 100 гривень) повернутися додому. У загальному випадку кількість гілок у такому алгоритмі не обов'язково дорівнює двом. При кожному конкретному наборі вхідних даних розгалужується алгоритм зводиться до лінійного.

Структура РОЗГАЛУЖЕННЯ існує в двох основних варіантах:

* повне;
* неповне;

**Оператори реалізації розгалужених обчислювальних процесів.**

Умовний оператор дозволяє перевірити умову і, в залежності від результату перевірки, виконати то чи іншу дію.

Структура умовного оператора має наступний вигляд:

**if** <умова> then <оператор1> else <оператор2>**;**

де **if** (якщо), then (тоді), else (інакше) – зарезервовані слова;

<умова> - довільний вираз логічного типа;

<оператор1>, <оператор2> - довільні оператори мови Пас­каль.

Перед *else* не ставиться ";"!

Алгоритм роботи оператора.

1. Обчислюється <умова>.

2. Якщо результат умови - true, то виконується <оператор1>, а  
<оператор2> пропускається; якщо результат - false, то виконується  
<оператор2>, а <оператор1> пропускається.

Якщо необхідно після then чи else виконати декілька операторів, вони беруться в операторні скобки Begin ... End (складений оператор).

Частина оператора else **<оператор2>** може бути опущена. Тоді, якщо <умова> дорівнює true, виконується <оператор **1** >, що знаходиться після then, впротивному випадку цей оператор пропускається. При цьому необхідно ставить ";" після **<**оператор1**>.**

Оскільки <оператор **1** > і **<оператор2>** можуть бути довільними операторами, в тому числі і складеними, і в той самий час не кожний з «вкладених» умовних операторів може мати частину else **<оператор2>**, то виникає неоднозначність трактовки умов, яка розв’язується наступним чином: будь-яка частина else, яка зустрічається відповідає найближчій до неї «згори» частини then умовного оператора.

**Циклічна алгоритмічна структура.**

Часто при розв'язанні завдань доводиться перевіряти одна умова для  
декількох значень або повторювати деякі дії кілька разів.  
Алгоритми, окремі дії яких багаторазово повторюються,  
називаються алгоритмами циклічної структури. Вона дозволяє істотно  
скоротити обсяг алгоритму, представити його компактно за рахунок організації повторень великого числа однакових обчислень над різними даними для отримання необхідного результату.

**Циклічною** називається структура, в якій якась, що йде підряд група дій (кроків), може виконуватися кілька разів, залежно від вхідних даних або умови задачі.

Сукупність повторюваних дій алгоритму називають **циклом**, тобто **циклічні алгоритми** включають в себе цикли.

Група повторюваних дій на кожному кроці циклу називається **тілом циклу.**

Приклади циклічних алгоритмів: фарбування паркану, прийом документів у  
абітурієнтів у приймальній комісії.

Циклічний алгоритм включає в себе:

1. Підготовку циклу - дії, пов'язані із завданням вихідних даних,  
використовуваних в циклі;

2. Тіло циклу - повторювані дії для обчислення шуканих величин,  
а також підготовка значень, необхідних для повторного виконання  
дій в тілі циклу;

3. Умови продовження циклу - дії, що визначають необхідність  
подальшого виконання тіла циклу.

Існує кілька видів циклічних конструкцій, за допомогою  
яких можна організувати цикли.

**Оператори циклу (повторень).**

У Turbo Pascal існує три види операторів циклу, які дозволяють запрограмувати дії, що повторюються. Вживання циклів в програмі дозволяє зменшити довжину програми і скоротити час її налагодження. Оператори для запису циклів є складеними, оскільки в їх склад входять інші оператори.

Вхід в цикл можливий тільки через його початок. Змінні оператора циклу повинні бути визначені до входу в циклічну частину. Обчислення, що повторюються, записуються тільки один раз. Необхідно передбачити вихід з циклу: або по природному його закінченню, або по оператору переходу (що украй небажано). Якщо цього не передбачити, то циклічні обчислення будуть повторюватися нескінченно. В цьому випадку говорять, що сталося зациклювання програми.

***Виконання програми в середовищі Turbo Pascal 7.0 завжди можна перервати шляхом натиснення комбінації клавіш* <Ctrl>+<Break>.**

**1. Оператор FOR.(О**ператор циклу з параметром**).**

Зазвичай застосовується, якщо заздалегідь відома кількість повторень.

Структура оператора:

**for** <парам. циклу>:= <поч.знач.> **to** < кін. знач.> **do** <оператор>**;**

де for (для), to (до),do (виконати) - зарезервовані слова;

<парам.циклу> - змінна будь-якого порядкового типу,

<поч.знач.> - початкове значення - вираз того ж типа ,

< кін. знач> - кінцеве значення - вираз того ж типа,

<оператор> - довільний оператор.

**Алгоритм роботи оператора**.

1. Обчислюється вираз <поч. знач.>.

2. Проводиться присвоєння <пар. циклу> := <поч. знач.>.

3. Після чого циклічно повторюється:

- перевірка умови <пар. циклу> <= <кін. знач.>; якщо умова не виконується, оператор for завершує роботу;

- виконання оператора <оператор>;

- нарощування на одиницю <пар. циклу>.

**Program** Sumlnt;

**Var** i, п, s : integer;

**Begin**

Write('N='); Readln(n);*{Ввод N}*

s:=0; *{Начальное значение суммы}*

**for** i:=1 **to** n **do** s:=S+i;

Writeln('CyMMa =', s);

**End.**

**2. Оператор з передумовою WHILE.**

Структура оператора:

**while** <умова> **do** <оператор>;

де <умова> - вираз логічного типу,

<оператор> - довільний оператор.

*Алгоритм роботи*. Якщо **<умова>** має значення **true,** то виконується **<оператор>,** після чого повторюється перевірка умови. Якщо **<умова>** має значення **false**, оператор **while** припиняє свою роботу.

**Program**  SumInt;

**Var** i, n, s : integer;

**Begin**

Writeln(‘N=’);

Readln(n);  *{Ввод N}*

s:=0;  *{Начальное значение суммы}*

i:=l; *{Начальное значение параметра цик­ла}*

**while** i <= n **do**

**begin**

s:=s+l; i:=i+l;

**end;**

Writeln('CyMMa =',s);

**End.**

**3. Оператор с післяумовою REPEAT…UNTIL.**

Структура оператора:

**repeat** <тіло циклу> **until** <умова>;

Тут <тіло циклу> - довільна послідовність операторів Turbo Pascal,

<умова> - вираз логічного типу.

Алгоритм роботи: <тіло циклу> повторюється хоч би один раз. Далі перевіряється <умова> і якщо його значення false, то повторюються оператори тіла циклу, якщо - true, то оператор repeat… until припиняє свою роботу.

*На відміну від оператора* for *, в операторах циклу*  while *і*  repeat… unti*l програміст повинен сам піклуватися про зміну параметра циклу.*

**Рекурсивна алгоритмічна структура**

**Рекурсивним** називається алгоритм, який в процесі виконання на  
якомусь кроці прямо або побічно звертається сам до себе. Як правило, в  
основі такого алгоритму лежить рекурсивне визначення якогось поняття.  
Приклад рекурсивного визначення - визначення факторіала числа n:  
n! = 1 \* 2 \* 3 \* 4 \* ... \* n .

**Program** faktorial;

**Var** n, i I integer; f : real;

**Begin**

Write('N='); Readln(n);*{Ввод N}*

f:=1; *{Начальное значение факториала}*

i:=l; *{Начальное значение параметра цикла}*

**repeat**

f:=f\*i; i;=i+l;

**until** i > N;

Writeln(‘факторіал=’,f);

**End.**

Мітки та оператори переходу.

Оператор переходу GOTO. Наявність цього оператора в мові Turbo Pascal є зайвим, оскільки можна написати програму будь-якої складності, не використовуючи цього оператора. Однак, іноді застосування оператора переходу спрощує написання програми, але в цьому випадку програма може вийти заплутаною и складною для розуміння. Автори книг і посібників настійно рекомендують відмовитися від використання цього оператора у ваших програмах.

Тим не менш, в деяких випадках використання операторів переходу може спростити програму.

Структура оператора:

GOTO <метка>;

*Алгоритм работы*: оператор передає керування позначеному операторові (тобто тому, який стоїть після мітки).

Ім'я мітки може бути довільним, але воно підлягає виконанню тим самим правилам, що й ім'я змінних. В якості змінних також можна використовувати цілі числа без знаку. Для зручності роботи міткам слід давати осмислені імена : *metka1, metka2* і т.д*.*

Рядок, який містить мітку, повинен мати вигляд:

*metka: оператор;*

Мітка розташовується безпосередньо перед, що позначається оператором і відокремлюється від нього двокрапкою. Оператор можна позначати кількома мітками, які в цьому випадку відокремлюються друг від друга двокрапкою. Перед тем як з'явитися в програмі, мітка повинна бути описана. Опис міток складається із зарезервованого слова LABEL (мітка), за яким іде список міток:

**label**

loop, lb1, lb2;

**begin**

**………**

**goto** lbl;

**………..**

loop:………

……………

lb:lb2:……

…………..

**goto** lb2;

При використанні міток необхідно керуватися наступними правилами:

• мітка, на яку посилається оператор GOTO, повинна бути описана в розділі описів і вона обов'язково повинна зустрітися де-небудь у тілі програми;

• мітки, описані в процедурі (функції), локалізуються в ній, тому передача керування ззовні процедури (функції) на мітку усередині неї неможлива.

Оператор вибору.

Оператор вибору дозволяє вибрати одне з декількох продов­жень програми. Параметром, по якому здійснюється вибір, є ключ вибору (<селектор>) - вираз будь-якого порядкового типу.

Структура оператора:

саsе <селектор> of

<Константа выбора1> : <оператор1>;

<константа выбора2> : <оператор2>;

……………………………………………..

<константа выбора*n*» : <оператор*n*>;

[else <оператор*n+1*>] end;

где *саsе* (в случае), *of* (из), *else* (иначе), *end*(кінець) - зарезервовані слова,

<селектор> - см. выше;

< константа вибору> - вираз обов'язково того ж типу, що й <селектор>;

<оператор> - довільний оператор.

Алгоритм роботи оператора.

1. Обчислюється вираз <селектор>.

2. У послідовності констант вибору шукається така, яка дорівнює обчисленому вираженню <селектор> і, якщо така є, виконується <оператор>, що стоїть за нею, після чого оператор вибору припиняє роботу.

3. Якщо в списку вибору не буде знайдена константа, відповідна обчисленому значенню селектора, то виконується <оператор>, що стоїть після else.

**Частина else може бути відсутня.**

**На відміну від умовного оператора *if* в операторі *case* перед варіантом *else* ставиться крапка з комою.**

Program digit;

Var і:integer;

Begin

Readln(i); {введення числа}

case і of

0,2,4,6,8 :Writeln('Четная цифра');

1,3,5,7,9:Writeln(‘Нечетная цифра');

else Writeln('Число больше 9’);

end; {оператора case}

End.

Програма вводить число з клавіатури і виводить повідомлення: "Парна цифра" (для цифр 0,2,4,6,8), "Непарна цифра" (для цифр 1,3,5,7,9), "Число більше 9".

**ТИПОВІ ПРИЙОМИ АЛГОРИТМІЗАЦІЇ**

При розв'язанні більшості завдань використовується певний набір  
типових прийомів алгоритмізації. Серед них розглянемо ті, які найбільш  
часто застосовуються при вирішенні практичних завдань.  
 *Зауваження:* При розв'язанні завдань, пов'язаних з обчисленнями і  
перетвореннями математичних виразів, їх слід проаналізувати:  
для всіх чи значень змінних їх можна обчислити. В алгоритмі  
необхідно передбачити попередню перевірку змінних на  
значення, для яких вираз не може бути визначене (підкореневий  
вираз кореня парного степеня не повинен набувати від'ємного   
значення, в дробу знаменник не може дорівнювати 0, в логарифмі основа  
має бути додатною і не рівною 1 і т. д.)

**Комбіновані алгоритми**

На практиці часто зустрічаються задачі, алгоритм розв'язання яких розпадається на частини, фрагменти і кожен фрагмент являє собою  
алгоритм одного з трьох описаних типів. Алгоритм, який містить  
кілька структур одночасно, називається комбінованим.  
В теорії алгоритмів доведено, що будь-який, як завгодно складний  
алгоритм може бути складений з трьох основних алгоритмічних структур:  
лінійної, розгалуження і циклу. Алгоритми розв'язання складних завдань можуть включати всі перераховані структури або комбінації деяких з них.

**Вкладені цикли**

Окремим випадком комбінованих алгоритмів є алгоритми, в  
яких необхідно використовувати кілька циклів, що змінюються  
одночасно. Наприклад, усередині одного циклу можуть знаходитися один або  
кілька інших циклів. Такі комбінації називаються вкладеними  
циклами.  
Охоплює цикл називається зовнішнім, а вкладені в нього цикли -  
внутрішніми (вкладеними), при цьому область дії внутрішнього циклу  
повинна повністю знаходитися в області зовнішнього циклу тобто цикли  
не повинні перетинатися .

Правила організації як зовнішнього, так і внутрішнього циклів такі ж,  
як і правила організації простого циклу. Параметри зовнішнього і внутрішнього повинні мати різний ім'я і змінюватися не одночасно, т. е. при одному значенні параметра зовнішнього циклу параметр внутрішнього циклу приймає по черзі всі свої значення. Перед циклом необхідно задавати початкові значення параметрів, а всередині - обчислювати поточні.

ДЕЯКІ ЗАДАЧІ МОВОЮ ПРОГРАМУВАННЯ TURBO PASCAL.

**Задача1.** Знайти суму ряду за формулою

programsuma;

usescrt;

varn,i:integer;s:real;

BEGIN

clrscr;

write('Vvedite n=');readln(n);

s:=0;

for i:=1 to n do

s:=s+1/(sqr(i)+1);

write('s=',s:5:3);

readkey;

END.

Для знаходження **добутку** та **кількості** необхідно змінити формулу та початкове значення.

В Паскалі початкове значення можна не вводити, тому що за замовчуванням при запуску програми воно завжди є нулем, але для строгості та відповідності до логіки міркувань ми це робимо.

**Доповнення.**  Заповнення масиву з використанням генератора випадкових

Чисел

........

RANDOMIZE;

For i:=1 tondo

x[i]:=Random(25)-7;

**Задача2.** Знайти добуток елементів масиву, що знаходяться в проміжку

[-1; 4].

programuutu;

usescrt;

const j=20

var x:array[1..j] ofinteger; i,n,a:integer;

BEGIN

clrscr;

writeln('vediterazmermasivan');

read(n);

for i:=1 to n do

begin

read(x[i]);

end;

clrscr;

for i:=1 to n do

begin

write('x[',i,']=',x[i],' ');

end;

writeln(' ');

s:=0;

for i:=1 to n do

if (x[i])>-1 and (x[i]<4)then S:=S+x[i];

write('s=',s:5:3);

readkey

END.

**Задача3.** Поміняти місцями перший та останній елементи масиву. Введення елементів масиву в ручному режимі або за допомогою генератора випадкових чисел.

programuutu;

usescrt;

label m1;

const j=20;

var x:array[1..j] ofinteger; i,n,a,p:integer;

BEGIN

clrscr; randomize;

writeln('vediterazmermasivan');

read(n);

m1: writeln('vedite p=1 abo p=2'); read(p);

case p of

1: beginwriteln('vediteelementymasiva'); for i:=1 to n doread(x[i]); end;

2: for i:=1 to n do x[i]:=random(25)-7;

elsegoto m1;

end;

clrscr;

for i:=1 to n do

begin

write('x[',i,']=',x[i],' ');

end;

writeln(' ');

a:=x[1];x[1]:=x[n];x[n]:=a;

for i:=1 to n do

begin

write('x[',i,']=',x[i],' ');

end;

readkey

END.

**Задача4.** Відсортувати масив за спаданням його значень.

programsort\_masiv;

usescrt;

var x:array[1..50] ofinteger; i,n,a,j:integer;

BEGIN

clrscr;

writeln('vediterazmermasivan');

read(n);

writeln('vediteelementymasiva X');

for i:=1 to n doread(x[i]);

clrscr;

writeln('Vihidnymasiv');

for i:=1 to n do

begin

write('x[',i,']=',x[i],' ');

end;

writeln(' ');

for j:=1 to n do

for i:=1 to n-1 do

if x[i]<x[i+1] thenbegin

a:=x[i];x[i]:=x[i+1];x[i+1]:=a;

end;

writeln('Rezultat');

for i:=1 to n dowrite('x[',i,']=',x[i],' ');

readkey

END.

**Задача5.** Введення матриці.

Programmatrica;

usescrt;

const p=50;r=50;

var m,n,i,j:integer;A:array[1..p,1..r] ofinteger;

BEGIN

clrscr;

write('m=');readln(m);

write('n=');readln(n);

writeln('Vvestielementy');

for i:=1 to m do

for j:=1 to n do

begin

write('A',i,j,'=',' ');

read(A[i,j]);

end;

clrscr;

for i:=1 to m do

begin

for j:=1 to n do

write(' A[',i,j,']=',a[i,j]:2);

writeln;

end;

readkey

end.

**Процедури, функції та модулі користувача.**

**Процедура.**

*Підпрограма-процедура* призначена для виконання якоїсь закінченої послідовності дій . Будь процедура починається з заголовка. На відміну від основної програми заголовок у процедурі обов'язковий. Він складається з зарезервованого слова *procedure* , за яким слід ідентифікатор імені процедури , а далі в круглих дужках - список формальних параметрів:

**Procedure** <ім'я >(<список формальних параметрів>);

<ім'я> - правильний ідентифікатор.

За заголовком можуть йти такі ж розділи , що і в основній програмі . Розділ описів процедури, як і в основній програмі, складається з розділів міток, констант, типів, змінних і, в свою чергу, процедур і функцій. Розділ операторів береться в операторні дужки *begin end,* причому після *end,* на відміну від основної програми , ставиться символ **«;».**

**Функції програміста (користувача).**

*Підпрограма - функція* призначена для обчислення якого параметра , у цієї

підпрограми дві основні відмінності від процедури .

 - Перша відмінність функції в її заголовку . Він складається зі слова function , за яким слід ім'я функції , далі в круглих дужках - список формальних параметрів ( про формальні параметри буде далі в лекції) , потім через двокрапку записується тип функції – тип що повертається параметра. Функція може повертати параметри наступних типів: будь-якого порядкового , будь-якого дійсного , стандартного типу string , будь-якого покажчика , в тому числі і типу Char .

 - Друга відмінність полягає в тому , що в тілі функції хоча б раз імені функції повинно бути присвоєно значення .

***Інструкція звертання до функції***

**Function<ім'я >(<список параметрів>):<тип>;**

де

Function зарезервоване слово мови Паскаль,

<ім'я > - правильний ідентифікатор;

**<**список параметрів**> -** список формальних параметрів, тобто список змінних, які

використовуються для передання в функцію інформації,

необхідної для обчислення значень функції;

**<**тип**> -** тип результату, що повертається функцією (тип значення

функції);

За заголовком йде список стандартних директив і тіло функції.

***Відмінні особливості функції****.* 1. Має тільки один результат виконання, але може

мати кілька вхідних параметрів.

2. Результат позначається іменем функції і

передається в основну програму.

3. В розділі операторів обов’язково повинен бути оператор присвоєння, в якому імені функції присвоюється результат роботи функції.

Викликається функція за її іменем із зазначенням фактичних параметрів . При цьому виклик функції можна робити безпосередньо всередині виразу.

Якщо ім'я функції всередині її опису використовується в правій частині оператора присвоювання, то це означає, що функція викликає себе рекурсивно.

Для виклику функції з основної програми або іншої підпрограми слід у виразі, де

необхідно використовувати значення функції, вказати ім'я функції зі списком фактичних параметрів, які повинні співпадати за кількістю і типами з формальними параметрами функції.

Приклади програм.

Приклад. Написати програму обчислення за формулою  сочетания http://www.matburo.ru/tv/tvformul/image006.gif

Обчислення факторіалу оформити у вигляді функції.

**Program** razfact;

**Uses** crt;

**Var** k, n : integer; f:real;

**Function** fact (m : integer): real;

**var** i ,p: integer;

**begin**

p:=1;

**for** i :=1 **to** m **do** p := p\*i;

fact := p

**end**;

**Begin**

clrscr;

Write(' введите k='); Readln (k); Write(' введите n='); Readln (n);

f := fact(n)/fact(n-k)\*fact(k);

Writeln('peзультат F=', f);

**Repeat Until** Keypressed

**End**.

В цьому прикладі *fact* – ім'я функції. Результат обчислення факторіалу позначається її іменем. Тип результату, що повертається функцією – цілий. Формальним параметром є змінна цілого типу *m* . Викликається функція своїм іменем *fact* із зазначенням фактичних параметрів *k* і *n*.

Задача. Обчислити факторіал числа з процедурою користувача та

функцією користувача.

Program Factor\_Proc\_Func;

uses crt;

var n,L:byte;f:integer;

procedure Factor\_1(p:byte;var k:integer);

var i:byte;

begin

k:=1;

for i:=1 to p do k:=k\*i

end;

function Factor\_2(m:byte;d:integer):integer;

var i:byte;

begin

d:=1;

for i:=1 to m do d:=d\*i;

Factor\_2:=d;

end;

BEGIN

clrscr;

write('n=');

read(n);

clrscr;

Factor\_1(n,f);

Factor\_2(n,L);

write('znache factoriala iz functii');

writeln('n=',n,' n!=',Factor\_2(n,L));

Repeat Until keypressed

END.

**Завдання для самостійного розв'язання**

**Завдання на лінійний алгоритм**

1. Скласти алгоритм обчислення значення функції y = 7x + 5 при будь-якому  
значенні x.

2. Скласти алгоритм обчислення периметра квадрата, якщо відома його сторона.  
3. Скласти алгоритм обчислення довжини кола, якщо відомий її радіус.  
4. Скласти алгоритм обчислення площі кола, якщо відомий її діаметр.  
5. Скласти алгоритм обчислення гіпотенузи прямокутного трикутника, якщо відомі його катети.

6. Скласти алгоритм обчислення периметра прямокутного трикутника, якщо відомі його катети.

7. Скласти алгоритм обчислення периметра прямокутника і його  
діагоналі, якщо відомі його боку.

8. Скласти алгоритм обчислення площі кільця, якщо відомі радіуса зовнішньої і внутрішньої окружності.

9. Скласти алгоритм обчислення площі поверхні і об'єму куба, якщо  
відоме його ребро.

10. Скласти алгоритм обчислення периметра рівнобедрений трапеції,  
якщо відомі її заснування і висота.

11. Скласти алгоритм визначення щільності тіла, якщо відомі його обсяг  
і маса.

12. Скласти алгоритм обчислення суми, різниці, добутку і  
приватного двох чисел.

13. Скласти алгоритм обчислення середнього арифметичного і середнього  
геометричного двох позитивних чисел.

14. Скласти алгоритм обчислення щільності населення в державі, якщо  
відомі його площу і кількість жителів.

15. Скласти алгоритм обчислення площі поверхні і об'єму  
прямокутного паралелепіпеда, якщо відомі його ребра.

**Завдання на розгалужений алгоритм**

1. Скласти алгоритм вирішення задачі для визначення максимального і мінімального значення з двох різних дійсних чисел.

2. Скласти алгоритм вирішення задачі: чи впишеться коло в квадрат, якщо  
відомі сторона квадрата і радіус кола.

3. Скласти алгоритм вирішення задачі для обчислення значення функції

*у* =

4. Скласти алгоритм вирішення задачі для визначення чи впишеться квадрат  
в коло, якщо відомі сторона квадрата і радіус кола.

5. Скласти алгоритм вирішення задачі для визначення більшого з двох  
дійсних чисел.

6. Скласти алгоритм вирішення задачі для обчислення значення функції

7. Скласти алгоритм вирішення задачі для визначення меншого з двох  
дійсних чисел.

8. Скласти алгоритм вирішення задачі для визначення більшої відстані  
з двох: одне зазначено в кілометрах, а інше в футах (1 фут = 0,45 м).  
9. Скласти алгоритм вирішення задачі для обчислення значення функції  
f(x) =

10. Скласти алгоритм обчислення частки двох чисел.  
11. Скласти алгоритм вирішення задачі для визначення більшої швидкості:  
одне значення вказано кілометрах в годину, а інше в метрах в секунду  
(1 м / с = 3,6 км / ч).

12. Скласти алгоритм вирішення задачі для визначення більшої площі,  
якщо відомі радіус кола і сторона квадрата.

13. Скласти алгоритм вирішення задачі для обчислення значення функції

14. Скласти алгоритм вирішення задачі для визначення більшого з двох  
значень: одне зазначено в міліметрах, а інше в дюймах (1 дюйм = 25,4  
мм).  
15. Скласти алгоритм вирішення задачі для визначення більшої щільності  
матеріалів двох тіл, якщо відомі їх обсяги і маси.

**Завдання на цикл з параметром**

1. Скласти алгоритм виведення таблиці переведення відстані в дюймах в  
сантиметри для значень 10, 11, .... 22 дюйми (1 дюйм = 25,4 мм).  
2. Скласти алгоритм виведення таблиці переведення перекладу 1, 2, ... 20 доларів США в гривні за поточним курсом (значення курсу вводиться довільно).  
3. Скласти алгоритм виведення значень e1, e2, ..., e20.

4. Скласти алгоритм виведення наступних чисел: 1.1, 2.1, ..., 21.1.

5. Скласти алгоритм виведення наступних чисел: 2.1, 2.2, ..., 2,9.

6. Скласти алгоритм виведення двадцяти перших парних чисел.

7. Скласти алгоритм виведення п'ятнадцяти перших непарних чисел.

8. Скласти алгоритм виведення значень наступних чисел: 2, 4, ..., 20.

9. Скласти алгоритм виведення вартості 2, 3, ..., 10 кг цукерок (ціна 1 кг  
цукерок вводиться довільно).

10. Скласти алгоритм табулирования функції y = x на відрізку [a; b].  
Значення a, b можуть змінюватися.

11. Скласти алгоритм табулирования функції y = ex - 1 на відрізку [a; b]. Значення a, b можуть змінюватися.

12. Скласти алгоритм табулирования функції 2x y = 1 на відрізку [a; b].  
Значення a, b можуть змінюватися.

13. Скласти алгоритм табулирования функції y = (x -1) 2 на відрізку [a; b].  
Значення a, b можуть змінюватися.

14. Скласти алгоритм обчислення значення виразу у для значень x,  
рівних 1, 2 ..., 20: y = 2t 2 + 2t + 2, t = 1 + x.

15. Скласти алгоритм обчислення значення виразу z для значень x,  
рівних 2, 4, ..., 20: z = 8f 3 - f, f = 2x.

**Завдання на цикл з умовою**

1. Скласти алгоритм вибору чисел, менше заданого числа P в  
послідовності 2, 3, 4 і т. д.

2. Дано два цілих числа A і B (A <B). Скласти алгоритм виведення всіх  
цілих чисел, розташованих між даними числами (не включаючи самі ці числа), в порядку їх зростання.

3. Дано два цілих числа A і B (A <B). Скласти алгоритм виведення всіх  
цілих чисел, розташованих між даними числами (не включаючи самі  
ці числа), в порядку їх убування.

4. Дано ціле число N (> 1). Скласти алгоритм виведення найменшого цілого  
K, при якому виконується нерівність 3K> N і самого значення 3K.  
5. Дано ціле число N (> 1). Скласти алгоритм виведення найбільшого цілого  
K, при якому виконується нерівність 3K <N.

6. Дано натуральне число N. Скласти алгоритм отримання всіх  
натуральних чисел, менше N.

7. Дано дійсне число а. Скласти алгоритм виведення всіх натуральних  
чисел n, при яких виконується умова a  
n ... 1 3 1 2 1+ 1 + + +>.  
8. Дано число n. Скласти алгоритм пошуку першого натурального числа,  
квадрат якого більше n.

9. Скласти алгоритм виведення мінімального числа більшу 200, яке  
остачі ділиться на 17.

10. Скласти алгоритм пошуку максимального з натуральних чисел, що не  
перевищують 600, яке без остачі ділиться на 28.

11. Почавши тренування, лижник в перший день пробіг 10 км. кожен  
Наступного дня він збільшував пробіг на 10% від пробігу попереднього  
дня. Скласти алгоритм визначення, в який день він пробіжить більше  
20 км.

12. Скласти алгоритм обчислення суми ряду 2n  
.... 2n 1 432y = 1 + + + - ззаданою точністю ε = 0,0001.  
13. Скласти алгоритм обчислення суми ряду ...8  
1412  
S = 1 1 + - + із заданою точністю ε = 0,0001.  
14. Скласти алгоритм виведення всіх натуральних чисел, кратних  
одинадцяти, менше 100.

**ВИСНОВОК**

Швидкий розвиток інформаційних технологій постійно змушує  
вносити зміни в зміст курсу «Інформатика». деякі теми  
дисципліни змінюються явно, багато інших по своїй суті залишаються без  
змін. Алгоритмізація, як розділ інформатики, який вивчає  
процеси створення алгоритмів, традиційно належить до теоретичної  
інформатики внаслідок свого фундаментального характеру.  
Розвиток нових інформаційних технологій, і зокрема технологій  
програмування, робить можливим у межах розділу «Основи  
алгоритмізації »розвивати вміння та навички, необхідні користувачеві при  
роботі з сучасним програмним забезпеченням, т. е. з'являється  
можливість зробити цей розділ сполучною ланкою між теоретичною і  
практичної інформатикою.

Проектування будь-якої задачі, незалежно від рівня її складності та  
сфери застосування, ставить розробника перед необхідністю вибору  
способу її вирішення і складання алгоритму.

Вивчення основ алгоритмізації має бути орієнтованим саме на  
розвиток алгоритмічного мислення, а не на вивчення певної мови  
програмування. Метою вивчення даної теми є навчити студентів  
аналізувати і ставити завдання, розробляти алгоритми їх вирішення.