МІНІСТЕРСТВО АГРАРНОЇ ПОЛІТИКИ І ПРОДОВОЛЬСТВА УКРАЇНИ
Аграрний коледж управління і права
Полтавської державної аграрної академії

Основи алгоритмізації Програмування на мові QBasic

ПОЛТАВА – 2011

Укладач: к.пед.н. **Кононец Наталія Василівна**, викладач інформатики та комп'ютерних технологій Аграрного коледжу управління і права Полтавської державної аграрної академії, викладач-методист

Рецензент: Кайшева Л.І., кандидат фіз.-мат. наук, викладач АКУП ПДАА

Посібник містить короткі теоретичні відомості з теми «Основи алгоритмізації та програмування», завдання практичних робіт з теми «Програмування у середовищі QBASIC», методичні рекомендації щодо їх виконання та завдання для індивідуальної самостійної роботи при вивченні курсу "Інформатика" (цикл загальноосвітньої підготовки).

Розраховано на студентів денної форми навчання фінансового, юридичного відділень вищих навчальних закладів І-ІІ рівнів акредитації. Посібник стане в пригоді не тільки студентам при самостійній роботі, а й викладачам інформатики та комп'ютерної техніки для роботи із студентами при проведенні практичних занять.

Розглянуто та схвалено
цикловою комісією математики, інформатики
та комп'ютерних технологій Аграрного коледжу
управління і права Полтавської
державної аграрної академії
Протокол №__від "___"_____2011 р.
Голова циклової комісії _____Худолій І.І.

ЗМІСТ

Поняття алгоритму

Короткі теоретичні відомості

Поняття алгоритму в інформатиці є фундаментальним, тобто таким, котре не визначається через інші ще більш прості поняття (для порівняння у фізиці - поняття простору і часу, у математиці - крапка).

Слово «алгоритм» походить від «algorithmi» — латинської форми написання імені великого математика аль-Хорезмі, який сформулював правила виконання арифметичних дій. Тому спочатку під алгоритмом розуміли тільки правила виконання чотирьох арифметичних дій над багатоцифровими числами в десятковій системі числення. Зараз він є одним із фундаментальних понять інформатики. **Алгоритм** може являти собою деяку послідовність обчислень, а може — послідовність дій нематематичного характеру. Але, у кожному разі, перед його складанням повинні бути чітко визначені початкові умови й те, що має бути одержано.

Алгоритм – послідовність дій, спрямованих на розв'язання поставленої задачі.

Алгоритм — кінцева послідовність кроків у рішенні завдання, що приводить від вихідних даних до необхідного результату.

Алгоритмом називається зрозуміле і точне розпорядження виконавцю виконати послідовність дій, спрямованих на досягнення зазначеної мети чи на розв'язання поставленої задачі.

В останньому означенні використовується поняття "виконавець". Що це означає? Під виконавцем алгоритму ми розуміємо будь-яку істоту (живу чи неживу), яка спроможна виконати алгоритм. Все залежить від того, якої мети ми намагаємося досягнути. Наприклад: риття ями (виконавці - людина або екскаватор), покупка деяких товарів (один із членів родини), розв'язування математичної задачі (учень або комп'ютер) тощо.

Виконавець алгоритму

Кожний алгоритм створюється з розрахунку на конкретного виконавця, тому можна сказати, що **алгоритм** — це точні розпорядження (указівки, команди, операції, інструкції) виконавцеві здійснити послідовність дій, спрямованих на розв'язання поставленої задачі.

Алгоритм складається із команд — окремих указівок виконавцеві виконати деякі конкретні дії. Команди алгоритму виконуються одна за одною, і на кожному кроці відомо, яка команда повинна виконуватися. Почергове виконання команд за кінцеве число кроків приводить до розв'язання задачі. Для того щоб виконавець міг

розв’язати задачу за заданим алгоритмом, він повинен уміти виконувати кожну з дій, що вказується командами алгоритму.

Виконавцями алгоритмів можуть бути людина, тварини, автомати, тобто ті, хто розуміє та може виконати вказівки алгоритму.

Система команд виконавця — сукупність команд, які можуть бути виконані виконавцем; кожна команда алгоритму входить до системи команд виконавця.

В основі роботи автоматичних пристроїв лежить положення, що найпростіші операції, на які розпадається процес розв’язання задачі, може виконати машина, яка спеціально створена для виконання окремих команд алгоритму і виконує їх у послідовності, вказаній в алгоритмі.

Розробляти алгоритми може тільки людина. Виконують алгоритми люди й усілякі пристрої — комп'ютери, роботи, верстати, супутники, складна побутова техніка й навіть деякі дитячі іграшки.

Будь-який виконавець (і комп'ютер зокрема) може виконувати тільки обмежений набір операцій (екскаватор копає яму, вчитель вчить, комп'ютер виконує арифметичні дії). Тому алгоритми повинні мати наступні **властивості**.

Властивості алгоритму

Виконуючи алгоритм, виконавець може не вникати в зміст того, що він робить, і разом із тим отримати потрібний результат, тобто виконавець діє формально. Тому для правильної побудови алгоритму необхідно знати систему команд виконавця, бути впевненим, що виконання алгоритму завершиться за кінцеве число кроків. Тому кажуть про деякі *загальні властивості алгоритмів*.

1. ***Зрозумілість***. Щоб виконавець міг досягти поставленої перед ним мети, використовуючи даний алгоритм, він повинен уміти виконувати кожну його вказівку, тобто розуміти кожну з команд, що входять до алгоритму. *Зрозумілість* - це властивість алгоритму, що полягає в тім, що кожен алгоритм повинен бути написаний у командах, зрозумілих даному виконавцю.
2. **Дискретність.** Алгоритм розв’язання задачі повинен складатися з послідовності окремих кроків — відокремлених одна від одної команд (указівок), кожна з яких виконується за кінцевий час. Тільки закінчивши виконання однієї команди, виконавець переходить до виконання іншої.
3. **Визначеність** (однозначність). Кожна команда алгоритму однозначно визначає дії виконавця і не припускає подвійного тлумачення. Суворо визначеним є й порядок виконання команд.
4. **Формальність.** Будь-який виконавець, який володіє заданою системою команд, може виконати заданий алгоритм, не вникаючи в суть задачі.
5. **Скінченність алгоритму.** Послідовність команд, які потрібно виконати, має бути скінченною.
6. **Результативність.** Виконання алгоритму не може закінчуватися невизначеною ситуацією або зовсім не закінчуватися. Будь-який алгоритм передбачає, що його виконання при допустимих початкових даних за кінцеве число кроків приведе до очікуваного результату.

7. **Масовість.** Алгоритм має передбачати можливість зміни початкових (вхідних) даних у деяких допустимих межах і можливість використання його для розв’язання задач одного класу (універсальність алгоритму). Отож, під масовістю алгоритму мається на увазі можливість його застосування для вирішення великої кількості однотипних завдань.

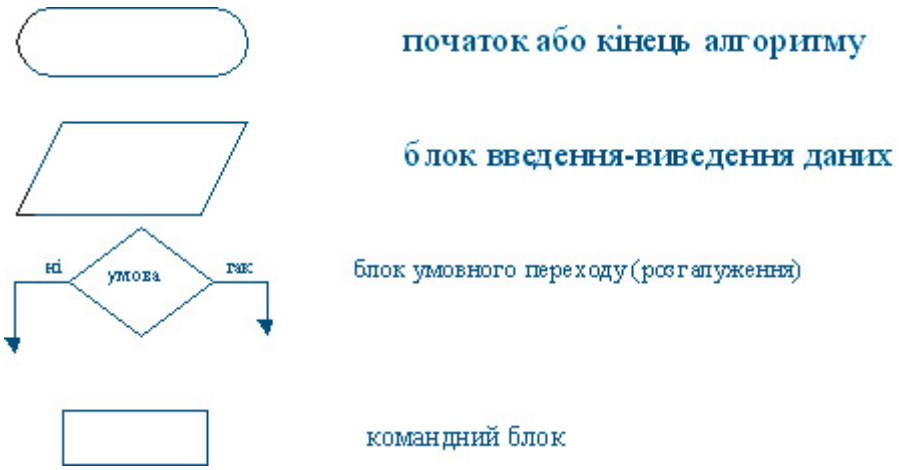
Саме через ці властивості часто дається визначення поняття **алгоритму** як скінченної однозначно визначеної послідовності операцій, формальне виконання якої приводить до розв’язання певної задачі за кінцеве число кроків.

Тепер залишається з'ясувати, яким чином можна подати алгоритм виконавцю. Існує кілька методів запису алгоритмів, вибір яких залежить від виконавця та того, хто його задає.

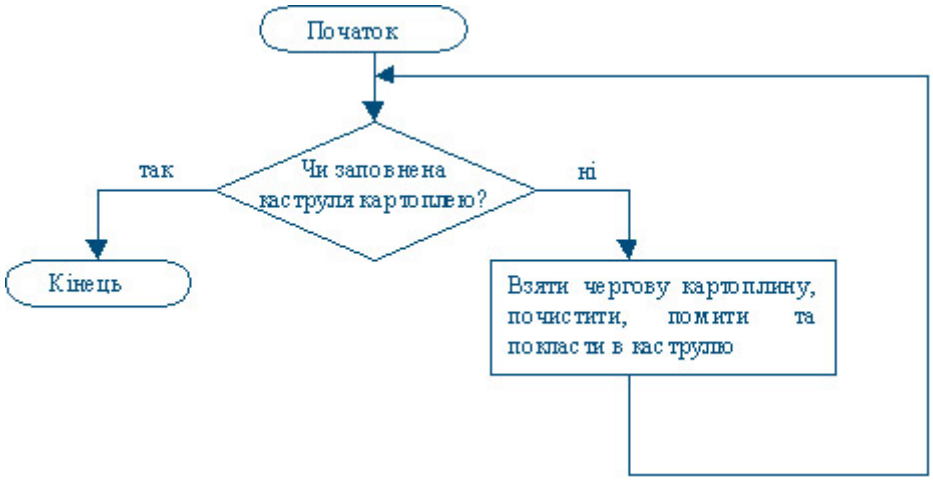
1. Перший спосіб - це ***словесний опис*** алгоритму. Це, по суті, звичайна мова, але з ретельним відбором слів і фраз, що не допускають зайвих слів, двозначностей і повторень. Доповнюється мова звичайними математичними позначеннями й деякими спеціальними відношеннями. Алгоритм описується у вигляді послідовності кроків. На кожному кроці визначається склад виконуваних дій і напрямок подальших обчислень. При цьому, якщо на поточному кроці не вказується який крок повинен виконуватися наступним, то здійснюється перехід до наступного кроку.

2. Другий спосіб - це подача алгоритму у вигляді ***таблиць, формул, схем, малюнків*** тощо. Наприклад, всіх вас вчили в дитячому садочку правилам поведінки на дорозі. І найкраще діти, вочевидь, сприймають алгоритм, що поданий у вигляді схематичних малюнків. Дивлячись на них, дитина, а потім і доросла людина, відпрацьовує ту лінію поведінки, що їй пропонується. Аналогічно можна навести приклади алгоритмів, що записані у вигляді умовних позначок на купленому товарі, щодо його користування (заварювання чаю, прання білизни тощо). У математиці наявність формул дозволяє розв'язати задачу, навіть "не використовуючи слів".

3. Третій спосіб - запис алгоритмів за допомогою ***блок-схеми***. Цей метод був запропонований в інформатиці для наочності представлення алгоритму за допомогою набору спеціальних блоків. Основні з цих блоків наступні:



Використовуючи дані блоки, можна подати, наприклад, алгоритм чищення картоплі в такому вигляді:



Примітка: У цьому алгоритмі, як правило, можна знайти неточності (наприклад, не відомо, що значить "взяти" і де "взяти", що значить "почистити" і таке інше). Тому степінь деталізації алгоритму залежить від виконавця. Ми, наприклад, передбачаємо, що наш виконавець в своїй системі команд має (тобто їх розуміє) команди "взяти чергову картоплю", "почистити картоплю", "помити" тощо.

4. Четвертий спосіб - ***навчальні алгоритмічні мови*** (псевдокоди). Ці мови мають жорстко визначений синтаксис і вже максимально наближені до машинної мови (мови програмування). Але створені вони з навчальною метою, тому мають зрозумілий для людей вигляд. Таких псевдокодів зараз існує велика кількість, починаючи з графічних середовищ "Алгоритміка", "Роботландія", "Лого-світи", "Черепашка" тощо і закінчуючи текстовими "національними" реалізаціями алгоритмічних мов, подібних до Бейсіка, Паскаля. Ці псевдокоди мають програмну реалізацію і дуже широко застосовуються на етапі навчання основам програмування.

5. П'ятий спосіб максимально наближений до комп'ютера - це ***мови програмування***. Справа в тому, що найчастіше у практиці виконавцем створеного людиною алгоритму являється машина (комп’ютер) і тому він повинен бути написаний мовою, зрозумілою для комп'ютера, тобто мовою програмування.

При побудові алгоритму часто виникає необхідність пояснити виконавцю деякі складні дії, якщо їх виконання не входить в систему команд виконавця. Наприклад, перший раз даючи дитині завдання пришити ґудзик до плаття, їй треба пояснити, як необхідно підбирати нитки для шиття, як вдягати нитку в голку, як тримати голку та ґудзик при роботі, яка різниця між пришиванням ґудзика до тоненької сорочки та товстої куртки (в другому випадку ґудзик робиться на "ніжці"). В подальшому такі пояснення будуть вже зайві, бо алгоритм "пришивання ґудзика" стає вже командою в системі команд виконавця "дитина".

Взагалі кажучи, кожна дія людини (якщо вона її може виконати) може вважатися командою її "системи команд", хоча колись, на етапі навчання, учитель

або хтось інший ретельно пояснював, яку треба виконати послідовність дій, щоб досягти поставленої мети.

Узагальнюючи сказане, можна сказати, що кінець кінцем кожну задачу можна вважати окремою командою виконавцю, якщо його навчено виконувати поставлене завдання. Якщо ж виконавець не знає, як розв'язувати запропоновану задачу, виникає потреба розкласти її на такі *підзадачі*, що являються "посильними" для виконання, тобто входять до системи команд виконавця. Продовжуючи цей процес, остаточно отримують алгоритм, що складається з простих команд, зрозумілих виконавцю, або остаточно переконуються, що дана задача непосильна для вибраного виконавця, тому що в його системі команд не існує необхідних для цього команд. Наприклад, як би ми не деталізували алгоритм побудови багатоповерхової будівлі для дитини, задача кінець кінцем являється для неї непосильною.

Запропонований підхід до конструювання алгоритмів називається ***методом покрокової деталізації зверху вниз***. Вочевидь, що при такому підході кожна операція остаточно буде подана у вигляді лише одного з трьох типів базових структур алгоритмів - *лінійної* (в літературі часто ця структура називається *слідування*), *розгалуження* або *повторення* (циклу). Степінь деталізації алгоритму при цьому сильно залежить від того, на якого виконавця його орієнтовано.

Алгоритми, що складаються для розв'язування окремих підзадач основної задачі, називаються ***допоміжними***. Вони створюються при поділі складної задачі на прості або при необхідності багаторазового використання одного ж того набору дій в одному або різних алгоритмах.

Допоміжний алгоритм повинен мати тільки один вхід та один вихід, причому того, хто користується ним, зовсім не цікавить, як реалізований цей алгоритм. Головне, щоб всі команди, що входять до складу допоміжного алгоритму входили до системи команд обраного виконавця. Зверніть увагу ще на те, що в реальному житті допоміжні алгоритми можуть виконувати, навіть, зовсім інші виконавці. Наприклад, якщо батьки вдома вирішили зробити ремонт, це не значить, що вони самостійно повинні зробити собі шпалери та клей. Алгоритми виробництва матеріалів існують і їх хтось виконує, а ми тільки користуємось результатами їх роботи.

Таким чином, можна вважати допоміжний алгоритм своєрідним "чорним ящиком", на вхід якого подаються деякі вхідні дані, а на виході ми отримуємо очікуваний результат. Головне чітко домовитись про правила оформлення вхідних даних та вигляд результату. Невиконання домовленостей може привести до збою у виконанні допоміжного алгоритму або до отримання неочікуваного результату.

Описаний метод послідовної деталізації лежить в основі технології структурного програмування і широко застосовується при використанні таких мов програмування, як Бейсік, Паскаль, С, С++ та інших.

При описуванні програми для комп'ютера мовами високого рівня допоміжні алгоритми реалізовуються у вигляді підпрограм. Правила опису, звернення до них та повернення в точку виклику визначаються конкретною мовою програмування.

Для зручності часто використовувані підпрограми можна об'єднувати в бібліотечні модулі і при необхідності підключати їх в свої програми.

Базові структури алгоритму

Базові структури алгоритму — це структури, за допомогою яких створюється алгоритм для розв'язання певної задачі. Існують три основні (базові) алгоритмічні структури, або три основні типи алгоритмів: *лінійний, розгалужений* та *циклічний.*

1. Лінійний алгоритм (послідовне виконання, структура слідування) — це алгоритм, який забезпечує отримання результату шляхом одноразового виконання послідовності дій, незалежно від вхідних даних і проміжних результатів. Дії в таких алгоритмах виконуються послідовно, одна за однією, тобто лінійно.

Алгоритм ***РАНОК***

1.Встати о 6.30 годині.
2.Виконати гімн. вправи.
3.Умитися.
4.Поснідати.
5.Вийти з дому о 7.30 годині.

2. Розгалужений алгоритм (умова, структура вибору) — у класичному варіанті ця структура розглядається як вибір дій у разі виконання або невиконання заданої умови. Галуження бувають повними і неповними.

Повне галуження — це галуження, в якому певні дії визначені й у разі виконання, і в разі невиконання умови. Неповне галуження — це розгалуження, в якому дії визначені тільки у разі виконання (або у разі невиконання) умови.

Якщо *логічний вираз*, **то** *команда 1*, **інакше** *команда 2*.
Серія команд – це декілька команд.

Алгоритм ***ВЕЧІР***

1.Повернутися з коледжу додому після занять.
2.Пообідати.
3.Якщо погода хороша, то попрацювати в саду, інакше піти в бібліотеку, взяти книжку, повернутися додому.
4.Зробити домашнє завдання.
5.Повечеряти.
6.Якщо є цікава телепередача, то подивитися телевізор, інакше почитати книжку.
7.Лягти спати.

3. Циклічний алгоритм (цикл, структура повторення) — це алгоритм, у якому передбачено повторення деякої серії команд. За допомогою цієї структури описуються однотипні дії, що повторюються декілька разів. Такі алгоритми забезпечують виконання довгої послідовності дій, записаних порівняно короткою послідовністю команд. Саме використання циклів дозволяє у повній мірі реалізувати швидкодію комп'ютерів.

Циклом називають процес повторення дій. Циклічні алгоритми забезпечують повторне виконання деяких команд скінчену кількість разів.

Доки *логічний вираз*, **виконати** *команди*

Алгоритм ***КОЛЛЕДЖ***

1.Іти на першу пару.
2.Доки не закінчилися заняття іти на наступну пару.
3.Іти додому.

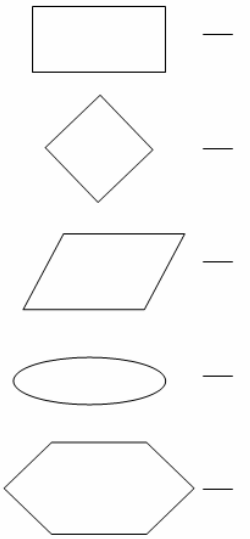
Основна особливість базових алгоритмічних структур — це їх повнота, тобто цих структур достатньо для створення найскладнішого алгоритму.

Способи запису алгоритму

Процес алгоритмізації — це визначення елементарних дій та порядку їх виконання для розв'язання поставленого завдання. Існують різні способи запису алгоритмів (словесний, формульно-словесний, метод блок-схем, програмний та ін.), які застосовуються для представлення алгоритму у вигляді, що однозначно розуміється і розробником, і виконавцем алгоритму.

Для опису алгоритмів людина часто користується природною мовою, але для запису багатьох алгоритмів природна мова виявилась незручною, тому виникла необхідність у створенні штучних мов, наприклад мови математичних формул, хімічних процесів тощо. Існує спеціальна навчальна алгоритмічна мова, яка була створена для запису алгоритмів на папері; вона використовує слова природної мови, але має більш жорстку структуру. Найбільше поширення для запису логічної структури алгоритмів отримали графічні (структурні) схеми, які спрощують складання та аналіз алгоритму, полегшують перехід від запису алгоритму до написання програми.

Блок-схема алгоритму — це графічне зображення алгоритму у вигляді спеціальних блоків з необхідними словесними поясненнями. Кожний етап алгоритму представляється у вигляді геометричної фігури (блоку), що має певну форму в залежності від характеру операції. Блоки на схемі з'єднуються стрілками (лініями зв'язку), які визначають послідовність виконання операцій та утворюють логічну структуру алгоритму.



— прямокутник - арифметичний блок, в який записується математична формула, наприклад, a=b+c; D=b^2-4ac і т.д.
В таких блоках знак "=" – це знак присвоєння, іноді його записують так ":=".

— ромб – логічний блок, в який записується умова.
Наприклад, a>b, D≥0 і т.д.

— паралелограм - в який записуються дані, які вводяться і виводяться з алгоритму.

— еліпс - записують початок і кінець алгоритму.

— шестикутник - запис умови циклу "**для**"

Важливою особливістю базових структур алгоритмів є те, що вони мають один вхід і один вихід, що дозволяє при відносній незалежності конструювати

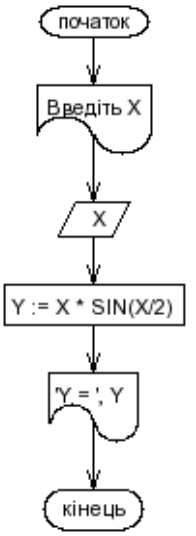
окремі блоки алгоритмів, а потім окремо розроблені структури з'єднувати між собою (вихід однієї базової структури сполучається із входом іншої). Весь алгоритм представляє лінійну послідовність базових структур.

Розглянемо приклади блок-схем.

Приклад 1. Створити блок-схему для обчислення функції X*SIN(X/2) по введеному значенню аргументу X

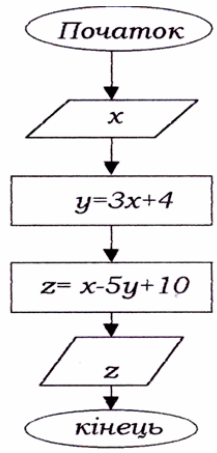
В цій задачі треба послідовно виконати ряд кроків, однакових для всіх вхідних даних, отже це буде **лінійний алгоритм**.

Блок-схема лінійного алгоритму



Приклад 2. Знайдіть значення виразу z = x-5y+10, де y = 3x+4

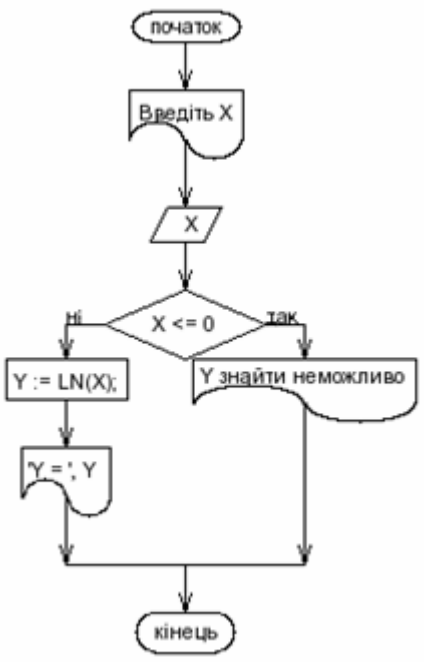
Блок-схема лінійного алгоритму має вигляд:



Приклад 3. **С**творити блок-схему для обчислення функції LN(X), яка перевіряє допустимість значення аргументу X

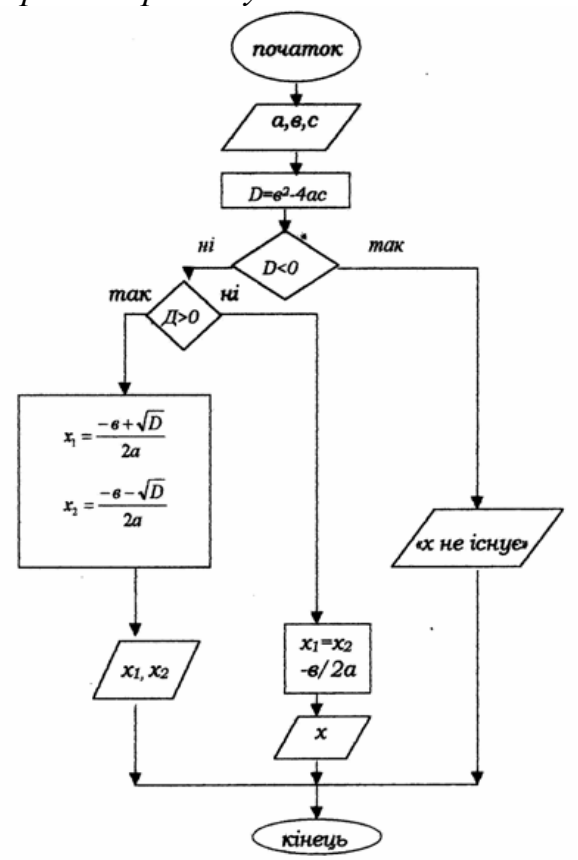
У цій задачі треба виконати ряд кроків, однакових для всіх вхідних даних, а деякі кроки будуть різними в залежності від значення аргументу, отже це буде **алгоритм з розгалуженням**.

У всіх випадках виконання починається з блоку ПОЧАТОК і закінчується у блоці КІНЕЦЬ. *Блок-схема алгоритму з розгалуженням:*



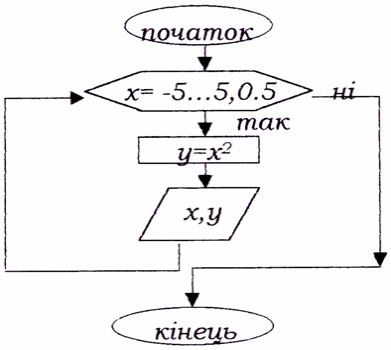
Приклад 4. Знайти корені квадратного рівняння $ax^2+bx+c=0$, де a, b, c > 0.

Блок схема представляє алгоритм з розгалуженням:



Приклад 5. Знайти значення функції $y=x^2$ для x від -5 до 5 з кроком 0,5

Блок-схема циклічного алгоритму:



Поняття мови програмування

Процес роботи комп'ютера полягає у виконанні програм, тобто деякого набору команд, що надходять у визначеному порядку. Машинний код команди складається з нулів та одиниць та указує, яку саме дію треба виконати центральному процесору. Отже, щоб задати комп'ютеру послідовність дій, яку він має виконати, треба задати послідовність двійкових кодів відповідних команд. Писати такі програми дуже складна справа. Раніше для цього програміст повинен був пам'ятати не тільки всі комбінації нулів та одиниць двійкового коду кожної команди, але й двійкові коди адрес даних, що використовувалися під час виконання програми. Щоб полегшити роботу програмістів, було розроблено багато мов програмування, які в більш наочному для людини вигляді подавали послідовність дій комп'ютера.

Алгоритмічні мови, призначені для побудови описів алгоритмів, що виконуються електронними обчислювальними машинами, називаються ***мовами програмування***.

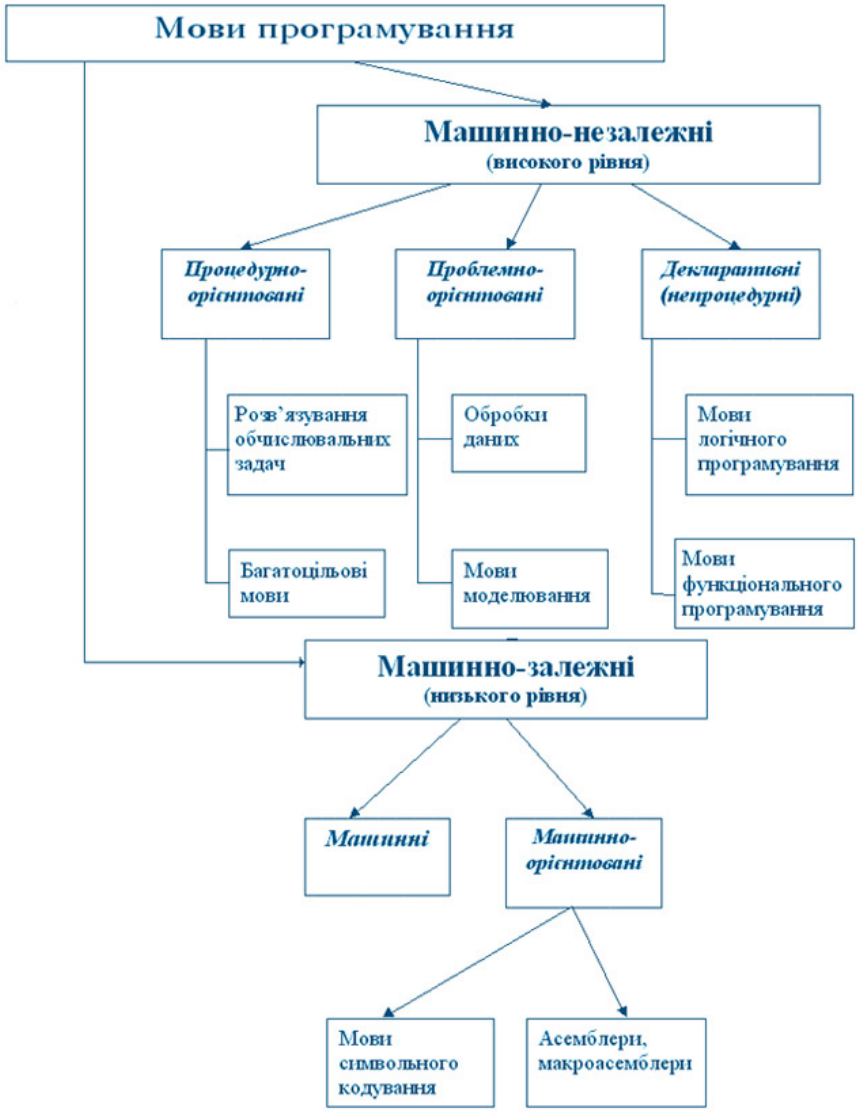
Описи алгоритмів мовою програмування називають ***програмами***.

Набагато легше написати програму мовою, що наближена до людської, а перекладання цієї програми на машинні коди доручити комп'ютеру. Так з'явилися мови, що призначені спеціально для написання програм - мови програмування.

Існує багато різних мов програмування (дивись малюнок). Взагалі, для розв'язування більшості задач можна використовувати будь-яку з них. Тільки досвідчені програмісти знають, яку мову програмування краще використовувати для розв'язування складних спеціалізованих задач, щоб урахувати особливості тієї чи іншої з них. Всі існуючи мови програмування можна поділити на дві групи:

- мови низького рівня;
- мови високого рівня.

До мов низького рівня належать мови ***асемблера*** (від англ. to assemble - складати, компонувати). У мові асемблера використовуються символьні позначення команд, які легко зрозуміти і запам'ятати. Замість послідовностей двійкових кодів команд записуються їх символьні позначення, а замість двійкових адрес даних, які використовуються під час виконання програми, - символьні імена цих даних. Іноді мову асемблера називають ***мнемокодом або автокодом***.



Більшість програмістів при складанні програм користуються деякою мовою високого рівня. Для описування алгоритмів такою мовою використовується певний набір символів - алфавіт мови. З цих символів складаються так звані ***службові слова*** мови, кожне з яких має певне призначення. Службові слова зв'язуються одне з

одним в речення за певними синтаксичними правилами мови і визначають деяку послідовність дій, які мусить виконати комп'ютер.

Використання мов високого рівня надає можливість описувати програми для комп'ютера, використовуючи загальноприйняті позначення операцій і функцій. Та програми, що написані на мовах програмування високого рівня (алгоритмічних мовах програмування), комп'ютер "не розуміє". Для того, щоб він міг виконати програму, її потрібно перекласти на машинну мову. Для такого перекладу використовують спеціальні програми, що мають назву - *транслятори*.

> ***Транслятор*** - це програма, що призначена для перекладу тексту програми з однієї мови програмування на іншу.

Процес перекладання називається ***трансляцією***. Розрізняють два типи трансляторів: *компілятори* та *інтерпретатори*.

> ***Компілятор*** - це програма, призначена для перекладу в машинні коди програми, що написана мовою високого рівня. Процес такого перекладання називається ***компіляцією***.

Кінцевим результатом роботи компілятора є програма в машинних кодах, яка потім виконується ЕОМ. Скомпільований варіант програми можна зберігати на дискові. Для повторного виконання програми компілятор вже не потрібен. Досить завантажити з диска в пам'ять комп'ютера скомпільований перед цим варіант і виконати його.

Існує інший спосіб поєднання процесів трансляції та виконання програм. Він називається ***інтерпретацією***.

> ***Інтерпретатор*** - це програма, що призначена для трансляції та виконання вихідної програми по командах (на відміну від транслятора, який цей процес виконує в цілому). Такий процес називається ***інтерпретацією***.

У процесі трансляції відбувається перевірка програми на відповідність до правил її написання. Якщо в програмі знайдені помилки, транслятор виводить повідомлення про них на екран монітора. Інтерпретатор повідомляє про знайдені помилки після трансляції кожної команди програми, а компілятор - після завершення компіляції всієї програми. Знайти та виправити в цьому випадку помилки значно складніше, ніж при інтерпретації. Через це програми-інтерпретатори розраховані, в основному, на мови, що призначені для навчання програмуванню, і використовуються програмістами-початківцями.

Як правило, програми компілятори та інтерпретатори називаються так само, як і мови, для перекладу з яких вони призначені. Слова **Паскаль, Бейсік, Сі** можна сприймати і як назви мов, і як назви відповідних програм-трансляторів.

Основні поняття мови програмування QBASIC

BASIC-программисты - люди меченые,
путём насилия или подкупа нас можно
заставить работать на другом языке,
но думать-то мы всё равно будем на BASIC...
С.Г. Зиновьев

Написання тексту програми — це запис алгоритму на алгоритмічній мові програмування.

Основні поняття мови програмування:

1. Алфавіт — символи, дозволені для запису команд і даних. Як правило, це літери латинського алфавіту, арабські цифри, розділові знаки, спеціальні символи.
2. Команди — перелік і правила запису дій, які передбачені мовою програмування.
3. Дані — вхідна інформація і результати обробки. Дані, які в процесі обробки змінюють своє значення, називаються змінними. Постійні дані називаються константами.
4. Ідентифікатори — символьне позначення (ім'я, адреса) даних, функцій, програм тощо.
5. Коментарі — пояснення до команд і програм.
6. Синтаксис — правила запису команд, опису і позначення даних, використання розділових знаків, коментарів тощо.
7. Семантика — (спрощено) опис дій для виконання написаних без синтаксичних помилок команд і визначення даних.

Мова QBASIC (Beginner's All-purpose Symbolic Instruction Code) розроблена Джоном Кемені й Томасом Куртцем у Дартмутському коледжі, США, у середині 1960 р.

QBASIC займає особливе місце серед всіх мов високого рівня. Із самого початку вона розроблялася як універсальна мова для початківців.

Переваги QBASIC (з погляду масового користувача):

- ✓ простота синтаксису;
- ✓ простота організації даних і керуючих структур;
- ✓ велика кількість *вбудованих* команд і функцій, що дозволяють легко виконувати такі операції, як керування текстовим і графічним екраном, обробка символьних рядків і т.п.)
- ✓ Одна з основних переваг мови Бейсік - *можливість розробки програм у діалоговому режимі*.

Ви, наприклад, можете почати налагодження програми, набравши всього один рядок. Переконавшись у тім, що цей рядок "працює" так, як ви й припускали, можна продовжити набір програми; якщо ні, то після аналізу отриманого результату можна швидко виправити виявлені помилки. Крім того, безпосередній режим роботи інтерпретатора надає можливість проведення експериментів по виявленню особливостей застосування того або іншого оператора або вбудованої функції, що також сприяє швидкому освоєнню мови.

- ✓ Особливою перевагою QBASIC варто вважати можливість роботи в режимі *інтерпретації*, що різко спрощує процес налагодження програм: виконання майже кожної команди можна перевірити відразу після написання (Shift +F5).

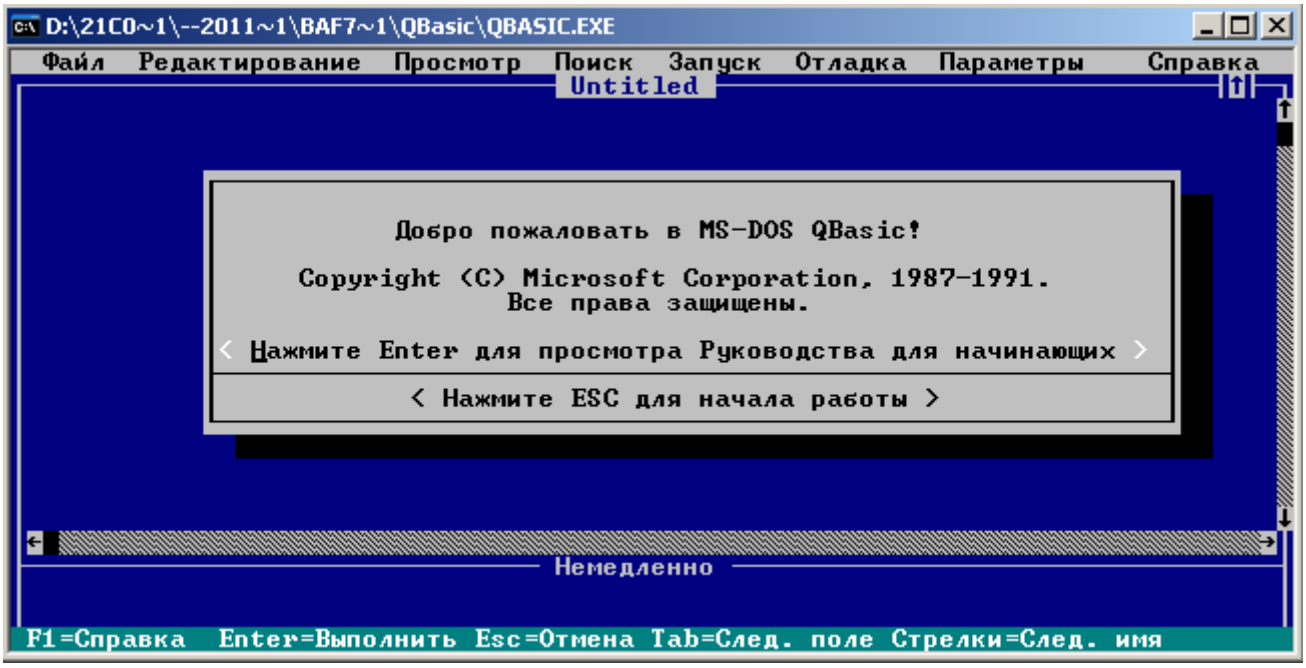


Рис. Вікно запуску середовища QBASIC

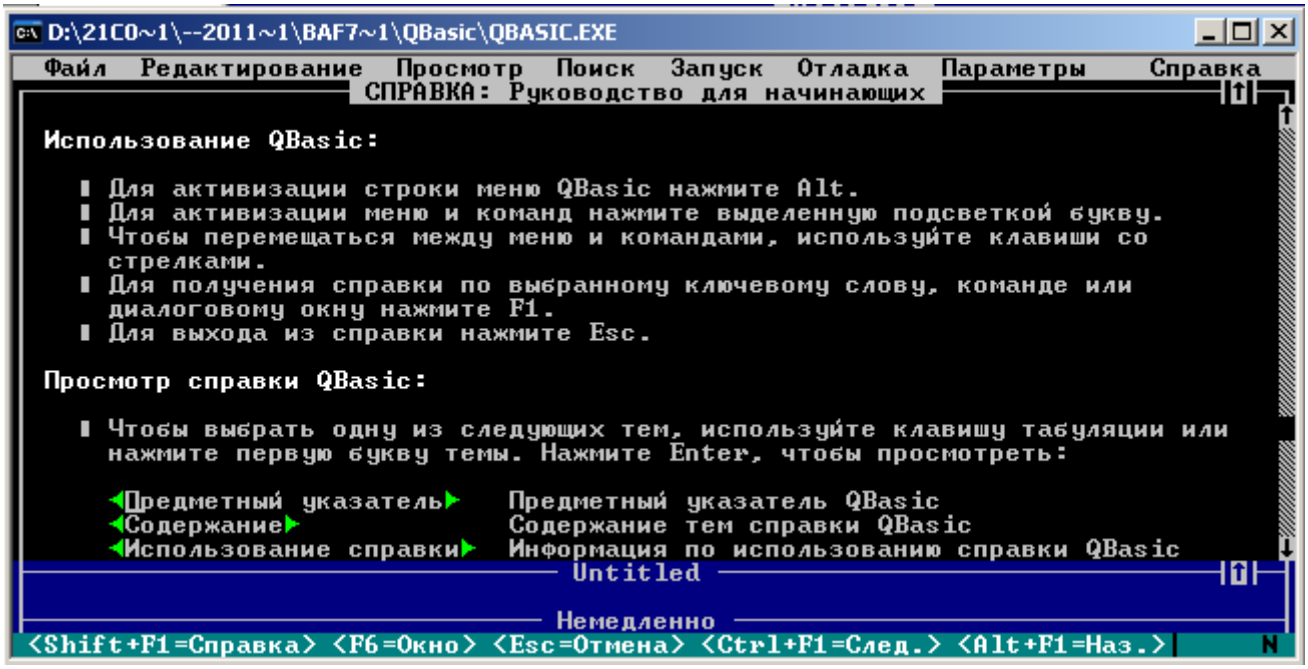


Рис. Середовище QBASIC: довідка-керівництво для користувача-початківця

✍ Клавіша **Esc** – для виходу із довідки

Формат програмного рядка QBASIC

Програма, яка написана на мові *Бейсік*, являє собою сукупність пронумерованих програмних рядків. Рядок програми може містити декілька операторів *Бейсіка*, які відокремлюються один від одного символом "**:**". Загальна довжина програмного рядка не повинна перевищувати *255* знаків.

Приклади рядків Бейсік-програми:

```
10 A=12
20 S=A+B
met1: I=1
```

```
10 LET x = (7 + 8) / 3
20 PRINT x
30 END
```

Програма на QBasic складається з команд, які являють собою сукупність різних символів - цифр, букв і спеціальних знаків, перерахованих нижче. В одному рядку програми може міститися одна або кілька команд (в останньому випадку команди відділяються друг від друга двокрапкою, наприклад: A = 1: B = 12: K = 5).

Алфавіт мови QBasic

Алфавіт QBasic включає:

- 26 латинських прописних *букв*:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z ;
– 10 арабських *цифр*:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
(нуль у програмах на QBasic звичайно перекреслюється похилою рисою, щоб не можна було сплутати його з буквою ПРО);

- *знаки арифметичних операцій:*

- віднімання або присвоєння знака мінус,
+ додавання,
* множення,
/ ділення,
^ піднесення до степеня.

- спеціальні знаки (символи) QBasic

( ) : ; . , ? > < = \ % & ! # $)

Константи QBasic

У *Бейсік-програмі* можуть використовуватися *константи* — *числа* або *ланцюжки знаків*, написані безпосередньо в тексті програми.

Константи поділяються на *числові* та *символьні* (*рядкові*).

Символьна константа — це послідовність знаків, довжиною не більше ніж *255*, яка береться в лапки (").

Числові константи поділяються на *цілі* та *дійсні*.

Цілі константи можуть бути представлені в *десятковій*, *восьмирічній* та *шістнадцятирічній* формах; дійсні константи — в *десятковій* та *експоненціальній* формах.

Цілі константи в десятковій формі лежать в межах від *–32768* до *+32767*.

Ціла константа в восьмирічній формі — це послідовність восьмирічних цифр (*0,1,...,7*), перед якою стоїть префікс **&O** або просто **&**.

Ціла константа в шістнадцятирічній формі — це послідовність шістнадцятирічних цифр (*0,1,...,9,A,B,C,D,E,F*), якій передує префікс **&H**.

Дійсні константи можуть бути представлені в двох форматах: із звичайною та подвійною точністю.

Змінні QBasic

Для позначення вихідних даних і результатів обчислень використовуються змінні.

Змінні – величини, значення яких може змінюватися в ході виконання програми. Для позначення змінних у програмі на QBasic використовують *імена* змінних (*ідентифікатори*), які можуть містити до 40 знаків (без пробілів) – букв і цифр, наприклад:

C, A3, ARMIR, PROV12, AMM6SV, DV24M і т.д.

Змінні, як і константи, поділяються на *числові* та *символьні* (*рядкові*). Числовій змінній можна присвоїти значення будь-якого числа, символьній — значення рядка знаків довжиною від *0* до *255* символів.

Імена змінних можуть бути будь-якої довжини, але *Бейсік* відрізняє імена за першими 40 символами. В іменах змінних можуть використовуватися літери, цифри та крапка. Імена змінних визначають їх *тип* (числові або символьні), а також *точність* числової змінної.

Ім'я символьної змінної повинно закінчуватися знаком **$**.

В імені числової змінної останній символ визначає:

%	*ціла змінна*
!	*змінна звичайної точності*
#	*змінна подвійної точності*

Масиви QBasic

Поряд з константами та змінними в програмах можуть використовуватися *масиви*. Перед використанням масиву в програмі оголошується *ім'я*, *тип* та *кількість елементів* масиву. Для цього використовується оператор розмірності масиву **DIM**.

Арифметичні операції

Для виконання арифметичних операцій в *Бейсік-програмі* використовуються наступні оператори:

+	*додавання*
–	*віднімання, зміна знаку*
*	*множення*
/	*ділення з плаваючою крапкою*
\	*цілочисельне ділення*
^	*піднесення до степеню*
MOD	*залишок від ділення*

Дії в арифметичних виразах виконують ***зліва направо*** залежно від їхнього пріоритету. Для того, щоб змінити природний порядок дій використовують круглі дужки. *Вирази в круглих дужках виконуються в першу чергу.*

Правила обчислення арифметичних виразів:

а) виконуються всі операції усередині дужок, починаючи із самих внутрішніх;

б) порядок обчислення усередині дужок визначається пріоритетом операції (значення функції; піднесення до степеня; множення - ділення; додавання - віднімання);

в) при наявності декількох операцій одного пріоритету обчислення виконуються послідовно зліва направо.

Операції відношення

В *операціях відношення* порівнюються два значення, які можуть бути обидва числовими або обидва символьними. В результаті отримуємо одне із значень: "*так*" (*–1*) або "*ні*" (*0*), які можуть використовуватися для управління ходом виконання програми (оператор **IF**).

Використовуються наступні операції відношення:

=	*дорівнює*
<> або ><	*не дорівнює*
<	*менше*
>	*більше*
<= або =<	*не більше*
>= або =>	*не менше*

Логічні операції

В логічних операціях виконуються побітові дії над операндами, попередньо перетворені в цілі (якщо це необхідно). Значення операндів завжди знаходяться в межах від *–32678* до *+32767*. Кожний операнд розглядається як послідовність шістнадцяти біт, над якими виконуються дії з допомогою наступних операторів:

NOT	*заперечення* (*логічне доповнення*)
AND	*і* (*кон'юнкція*)
OR	*або* (*диз'юнкція*)
XOR	*виключаюче або*
EQV	*еквівалентність*
IMP	*імплікація*

Символьні вирази

Поняття "*символьний вираз*" означає окремі символьні константи та змінні, а також їх комбінації з використанням *оператора зчеплення*. Для оператора зчеплення використовується символ +. В результаті операції зчеплення виконується об'єднання (зчеплення) символьних значень двох змінних або змінної та символьної константи.

У виразах можуть використовуватися ***вбудовані функції***:

ABS (x) - модуль x $|x|$

SQR (x) - корінь квадратний з x ($\sqrt{x}$).

INT (x) - ціла частина x

SIN (x) - синус x (аргументом служить радіанна міра кута)

COS (x) - косинус x

TAN (x) - тангенс x

ATN (x) - арктангенс x

LOG (x) - натуральний логарифм x

EXP (x) - експонента x (e^x)
SGN (x) - визначення знака числа x
RND (x) – генератор випадкових чисел

Аргументом функції може бути довільне арифметичний вираз, чисельне значення або змінна.

Наприклад. Записати по правилам Бейсіка математичні вирази:

1) $\frac{X^2+2X-5.12}{X^2+12.51}$ (X^2+2*X-5.12)/(X^2+12.51)

2) $\frac{COS(X) - SIN(X)}{|COS(X) + SIN(X)|}$ (COS(X)-SIN(X))/(ABS(COS(X)+SIN(X)))

ПРИКЛАДИ ОПИСУ АЛГЕБРАЇЧНИХ ВИРАЗІВ

№	Математичний запис	Опис мовою Basic
1	$\frac{\sin x + 5{,}3}{2{,}3^2 - \sqrt{\cos(x+y)}}$	$(SIN(X) + 5{,}3)/(2{,}3^2 - SQR(COS(X+Y)))$
2	$\frac{tg^2(e^{x+y})^3}{\lvert\ln^5(3x+4)\rvert}$	$(TAN(EXP(X+Y)\text{^}3))\text{^}2/ABS(LOG(3*X+4))\text{^}5$
3	$ax^{3{,}4} + bx + c$	$A*X^{3{,}4} + B*X + C$
4	$\frac{(a+b)^3 + 4{,}10^{-15}}{arctg(a^b - \pi)}$	$((A+B)^3 + 4{,}1E-15)/ATN(A^B - 3{,}1415)$
5	$\frac{b^2 - 4ac}{2a}$	$(B^2 - 4*A*C)/(2*A)$
6	$\sqrt[4]{c^3 + \sqrt[6]{a^5}}$	$C\text{^}(3/4) + A\text{^}(5/6)$

ОПИС ПОЗНАЧЕНЬ ФУНКЦІЙ

Функція	Мовою Basic	Примітки
$\sin x$	SIN(X)	X – арифметичний вираз (в радіанах)
$\cos x$	$COS(X)$	X - арифметичний вираз (в радіанах)
$tg\ x$	$TAN(X)$	$X \neq \pi/2 \pm \pi k$ (в радіанах)
$arctg\ x$	$ATN(X)$	X- арифметичний вираз
e^x	$EXP(X)$	X - арифметичний вираз
$\ln x$	$LOG(X)$	$X > 0; X$ - арифметичний вираз
$\lvert x\rvert$	$ABS(X)$	X - арифметичний вираз
$\sqrt{x}$	$SQR(X)$	$X \geq 0; X$ - арифметичний вираз
$sig\ \ n = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$	$SGN(X)$	X - арифметичний вираз
Генератор випадкових чисел з рівномірним розподілом ймовірностей на (0,1)	$RND(X)$	аргумент X - фіктивний
Ціла частина – найбільше ціле число, яке не перевищує X	$INT(X)$	X - арифметичний вираз
Відкидання дробової частини числа X, отримання цілої частини	$FIX(X)$	X арифметичний вираз (функція не округлює чисел)

Програмування лінійних алгоритмів

Програма на QBASIC складається з послідовності операторів.

Оператор є основним елементом мови й описує дії, які необхідно виконати для реалізації алгоритму рішення завдання. Він містить **службове слово** (ім'я оператора) і параметри.

Оператори записуються в рядки. Нумерація рядків в QBASIC не обов'язкова. На одному рядку при необхідності може бути кілька операторів, тоді вона розділяється двокрапкою " : "

Програма лінійного алгоритму представляє послідовність операторів, кожний з яких виконується один раз у порядку його проходження.

Оператор присвоювання

Служить для присвоювання змінним тих або інших значень відповідно до алгоритму завдання. В QBASIC оператор присвоювання - " = ".

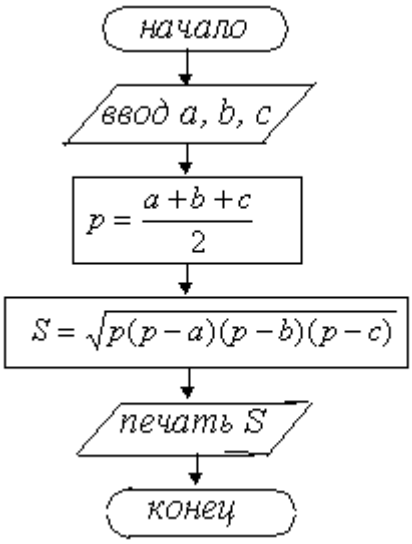
Приклад:
A=35; F=3.4; L=F-1; X=X+1.

Оператори зупинки й кінця програми

STOP і **END** - служать для припинення виконання програми.
STOP - логічне завершення програми; може стояти в будь-якому місці програми і їх може бути кілька.
END - фізичне завершення; коштує в самому кінці програми.
Оператори STOP і END у програмі на QBASIC можуть бути відсутні.
Приклад. Обчислити

$$S = \sqrt{p(p-a)(p-b)(p-c)} \text{ , де } p = \frac{a+b+c}{2}$$

Блок-схема



Приклад програми

```
a = 3
b = 4
c = 5
p = (a+b+c)/2
S = SQR(p*(p-a)*(p-b)*(p-c))
PRINT "S="; S
```

Оператор введення INPUT

Для введення даних в діалоговому режимі використовується оператор INPUT.

INPUT <СПИСОК УВЕДЕННЯ>

При його виконанні на екрані друкується "?" і машина очікує введення даних.

Приклад:

1) INPUT A, B, C
 ? 2, 3, 4
2) INPUT "Уведіть значення A, B, C" A, B, C
 Уведіть значення A, B, C ? 2, 3, 4

Оператори введення READ I DATA

Коли в програмі багато вихідних даних, які не підлягають частій зміні, або програма вже налагоджена для введення вихідних даних, використовують оператор **READ**. Числові значення вводимих змінних містяться в списку оператора **DATA**. Головне, щоб загальна кількість змінних у всіх операторах READ не перевищувало сумарне число значень в операторах DATA.

Оператор DATA містить дані, які при виконанні операторів READ будуть вводитися в змінні, перераховані в списку введення операторів READ. Дані, які підлягають введенню, можуть розташовуватися в одному операторі DATA або в декількох операторах DATA, але в тім порядку, у якому ці дані повинні використовуватися оператором (або операторами) READ. Оператори DATA можуть розташовуватися в будь-якому місці програми. У рядку, що містить оператор DATA, не допускається використання інших операторів.

Приклад:

```
. . . . . .
DATA 1, 2, 3
. . . . . .
READ A, B, C
. . . . . .
```

Числові значення зі списку DATA можуть використовуватися повторно й привласнюватися іншим змінним, якщо в програмі помістити оператор **RESTORE**.

Приклад:

```
DATA 5, 3
READ A, B
RESTORE
READ C, D
```

Оператор друку PRINT

Призначений для виведення на екран (або принтер) числових значень і тексту.

Оператор PRINT може містити числа, змінні, вирази й тексти, поміщені в лапки. Оператор PRINT без списку викликає друк порожнього рядка.

Приклад:

A = 3

```
B = -10
PRINT 1; A, 2; B
PRINT "Сума A+B = "; A+B
```

";" - забезпечує вивід розділених значень через один пробіл (знак "+" не друкується);
"," - позначає кінець зони з 14 позицій і наступне виведене значення буде надруковано на початку наступної зони.

Приклад:

```
PRINT B; A; A+B, A*B
 -10_ _ 3_ -7_ _ _ _ _-30
```

Кожний оператор PRINT здійснює друк з нового рядка. Однак, якщо наприкінці списку виведення є "," або ";", то наступний оператор буде продовжувати вивід на тім же рядку у форматі, що відповідає роздільнику.

Оператор REM або '

Використовується для пояснення до тексту програми.

Приклад:

```
REM Програма лінійного алгоритму
' Програма лінійного алгоритму
REM kvadrat
```

Програмування алгоритмів з розгалуженням

Оператор безумовного переходу

Безумовний перехід організується оператором

< **GOTO N**>

N – мітка (номер рядка програми), на яку здійснюється перехід.

Приклад:

```
 . . . . . .
GOTO 5
 . . . . . .
5 A= A+B
```

Умовний оператор

Умовний оператор призначений для зміни порядку виконання операторів після перевірки деякої умови

Перший формат умовного оператора:

IF *умова* **THEN**
блок_операторів_1
ELSE
блок_операторів_2

де:

умова - довільний вираз, значенням якого є true (не нуль) або false (нуль);

блок_операторів_1, блок_операторів_2 – один чи декілька операторів, записаних на одному чи більше рядках.
Дія такого умовного оператора полягає у наступному: якщо умова істинна, то виконується перший блок операторів, після чого здійснюється перехід до оператора, який іде наступним за умовним оператором. Якщо умова хибна, то виконується другий блок операторів, після чого здійснюється перехід до оператора, який іде наступним за умовним оператором.

Другий формат умовного оператора:
IF *умова* **THEN** *оператори*
При записі умови можуть використовуватися службові слова, що показують деяке логічні співвідношення між умовами (AND - і; OR - або)

Приклади:

```
1) INPUT A, B
 IF A<=B THEN X=A ELSE X=B
 PRINT X

2) INPUT A, B
 IF A>=B THEN X=A: GOTO 10
 X=B
 10 PRINT X

3) INPUT A, B, C
 IF A=B AND B=C THEN Y=1 ELSE Y=2
PRINT Y
```

Програмування циклічних алгоритмів

Будь-який алгоритм циклічної структури в загальному випадку містить наступні дії: задання початкових значень змінних; дії, виконувані безпосередньо в циклі, які називаються **тілом циклу**; зміна значень змінних циклу за деяким законом; перевірка умови продовження або закінчення циклу.

Оператор циклу:

FOR V=A1 **TO** A2 **STEP** A3
оператори тіла циклу
NEXT V

FOR - для; TO - до; STEP - крок; NEXT - наступний; V - ім'я керованої змінної або параметра циклу; A1, A2, A3 - вирази, що визначають відповідно початкове, кінцеве значення й крок параметра циклу (значення кроку за замовчуванням =1).

Така структура пропонує виконати всі оператори, розташовані між операторами FOR і NEXT, для всіх значень параметра циклу V, що змінюються від початкового A1 до кінцевого A2 із кроком A3.
Конструкція STEP A3 може бути опущена, якщо A3=1.

Приклад:

```
X=1
FOR I=1 TO 10
X=X+1
PRINT X
NEXT I
```

Якщо цикл містить у собі один або кілька циклів, то утримуючий усередині себе інші цикли називається **зовнішнім**, а цикл, що міститься в іншому циклі - **вкладеним.**
При програмуванні вкладених циклів необхідно виконати правило: внутрішній оператор циклу й приналежна йому область дії повинні повністю міститися усередині області зовнішнього циклу, у такий спосіб зовнішній цикл завжди починається раніше, а закінчується пізніше, ніж внутрішній.

Приклад:

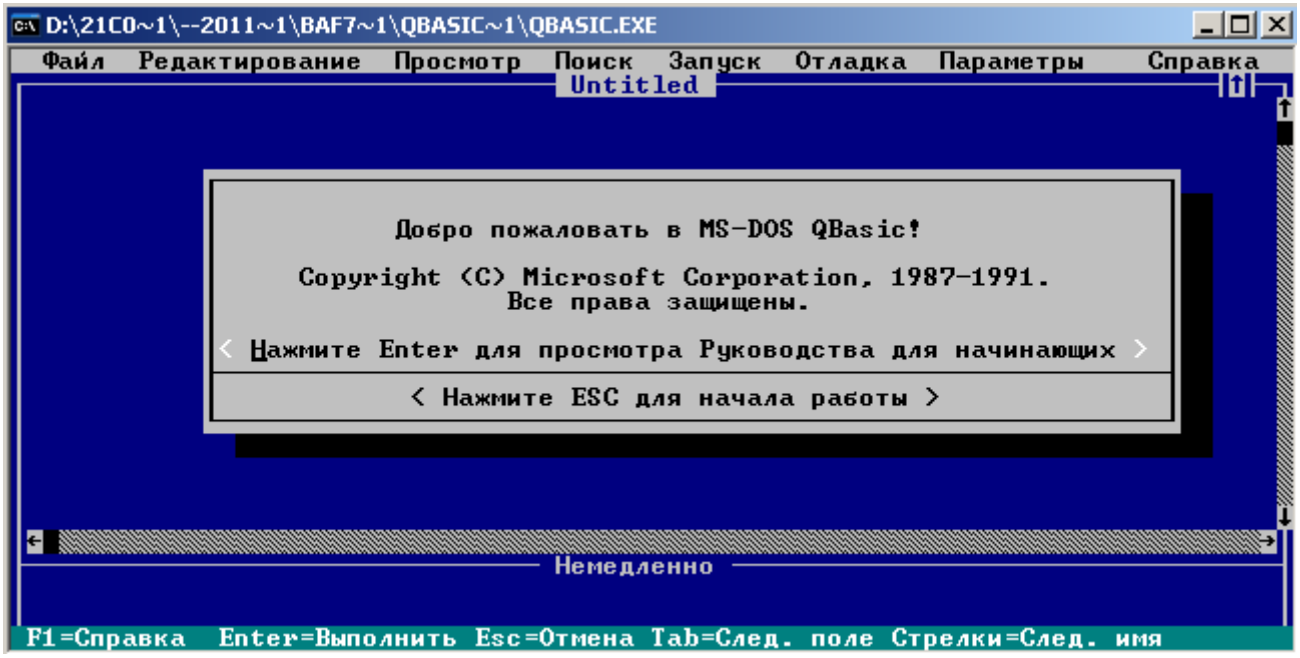
```
 X=1
FOR I=0 TO 1 STEP 0.1
 FOR J=1 TO 3
 X=X+I*J
 PRINT X
 NEXT J
NEXT I
```

Почитай!

http://www.gaps.tstu.ru/win-1251/lab/qb/win-1251/basic_met.html

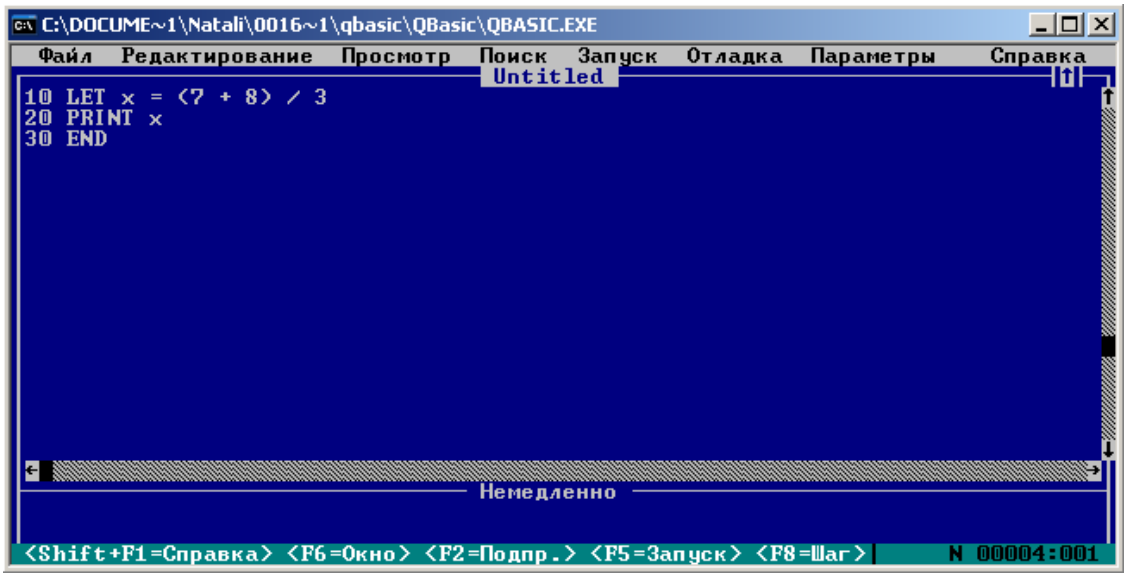
РОБОТА В СЕРЕДОВИЩІ QBASIC

Для запуску системи програмування Qbasic слід завантажити файл



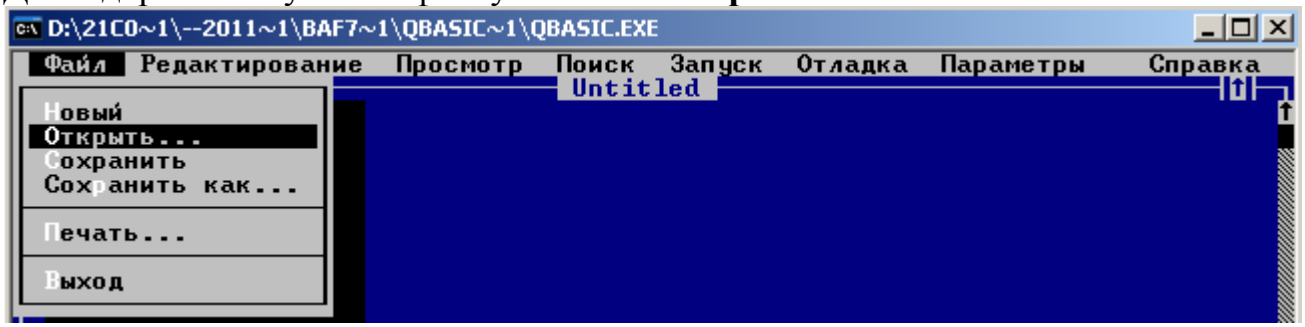
Для переходу до роботи у середовищі – натиснути **ESC:**

< Нажмите ESC для начала работы >

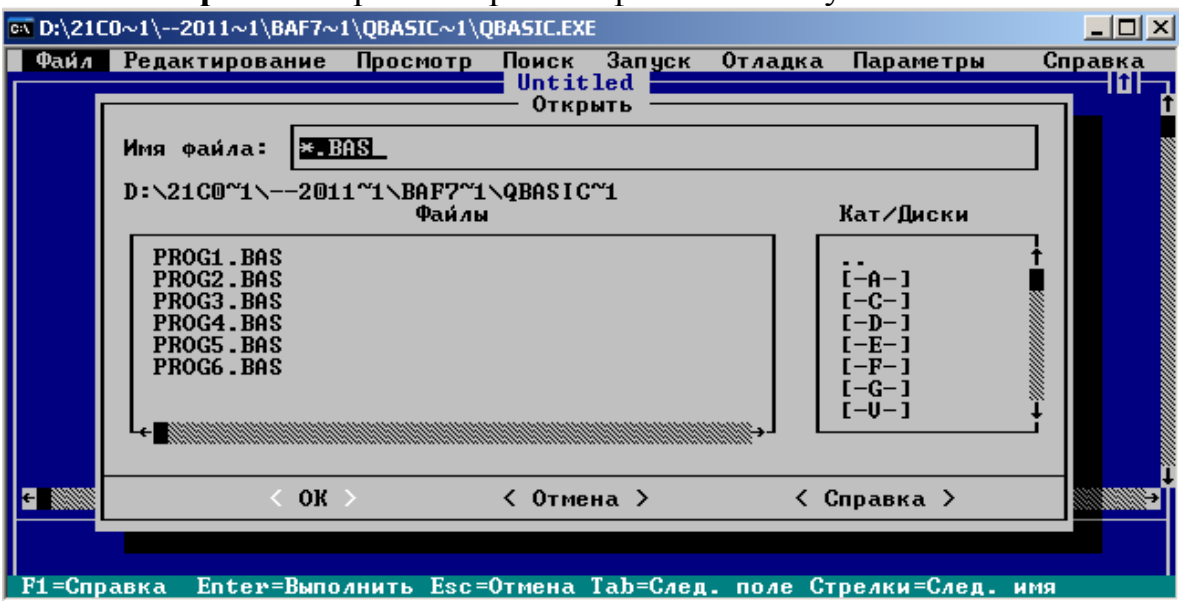


*Приклад простої програми на **Qbasic***

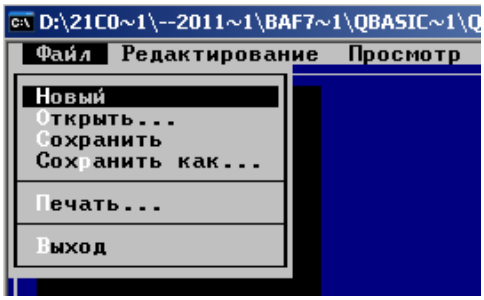
Для відкриття існуючого файлу: **Файл – Открыть**



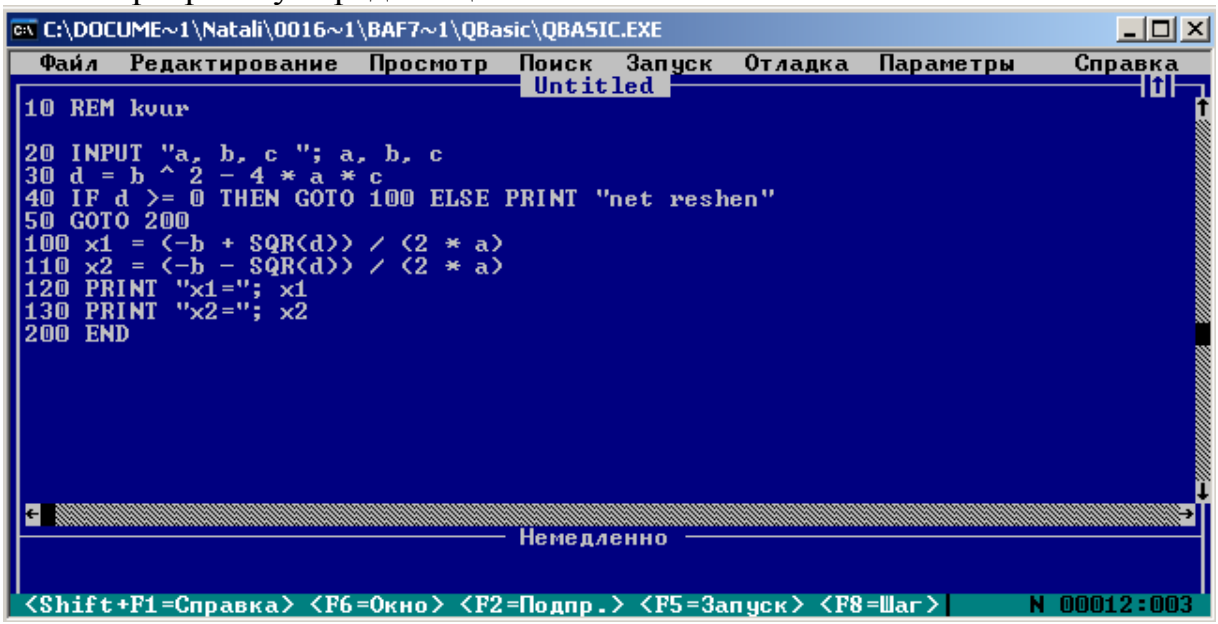
Із вікна **Открыть** вибрати потрібний файл і натиснути ОК**:**



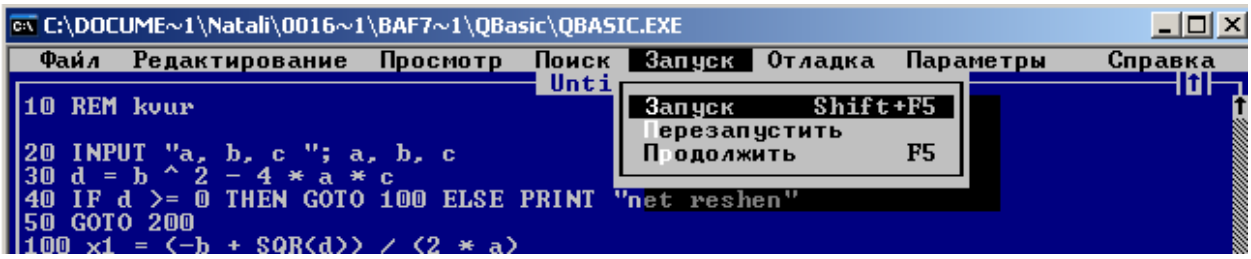
Для створення нового файла: **Файл – Новый**



Написання програми у середовищі:

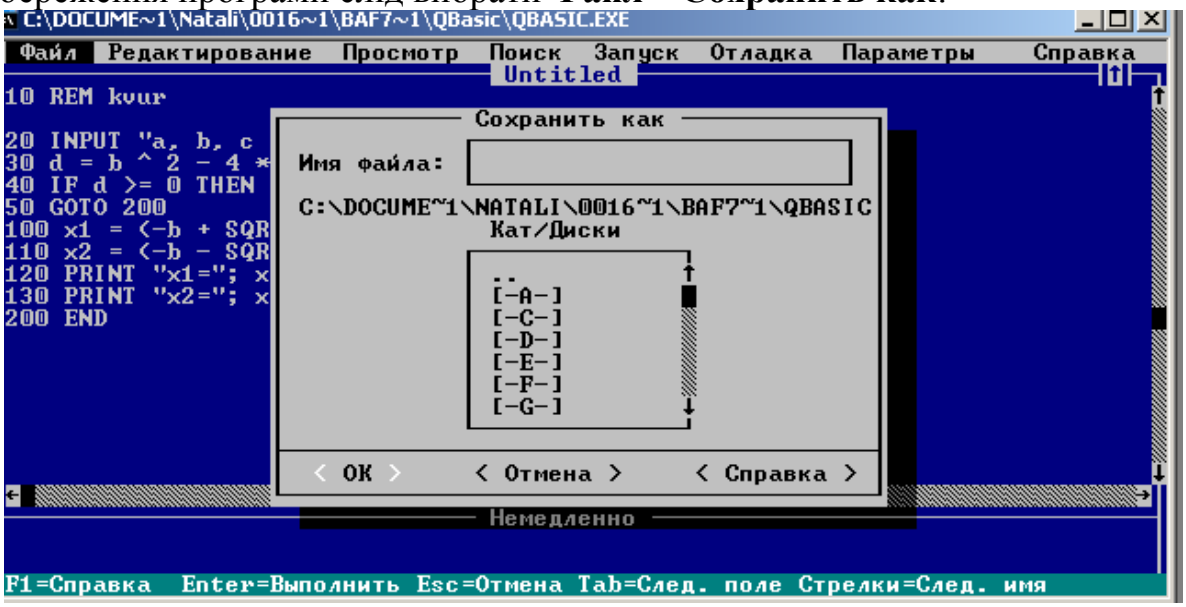


Для запуску і перевірки роботи програми: **Запуск – Запуск**

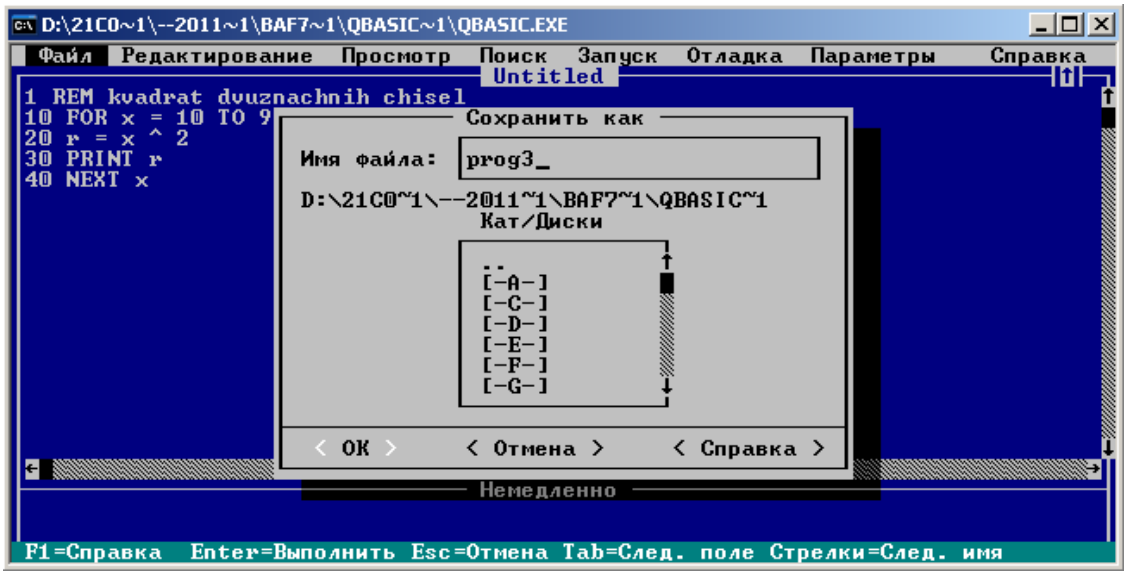


На рисунку зображено результат роботи програми «Розв’язування квадратного рівняння»:

Для збереження програми слід вибрати **Файл – Сохранить как**:



У вікні **Сохранить как** написати ім’я файлу (латинськими літерами) і натиснути **ОК**.



Практична робота

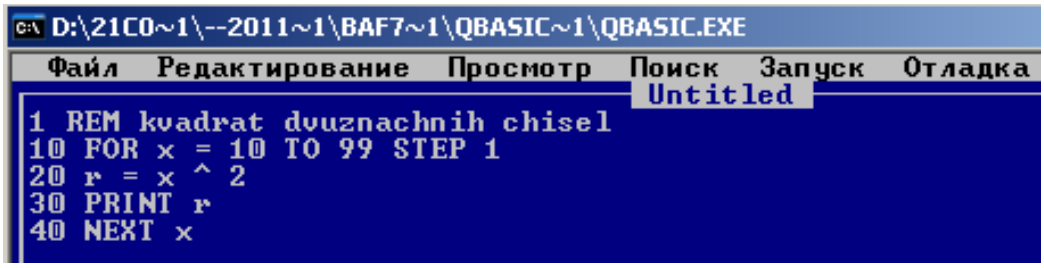
Програма 1. Скласти програму для розв'язування квадратних рівнянь. Зберегти під ім'ям **PROG1**.

```
10 REM KVUR
20 INPUT "A, B, C "; A, B, C
30 D = B ^ 2 - 4 * A * C
40 IF D >= 0 THEN GOTO 100 ELSE PRINT "NET RESHEN"
50 GOTO 200
100 X1 = (-B + SQR(D)) / (2 * A)
110 X2 = (-B - SQR(D)) / (2 * A)
120 PRINT "X1="; X1
130 PRINT "X2="; X2
200 END
```

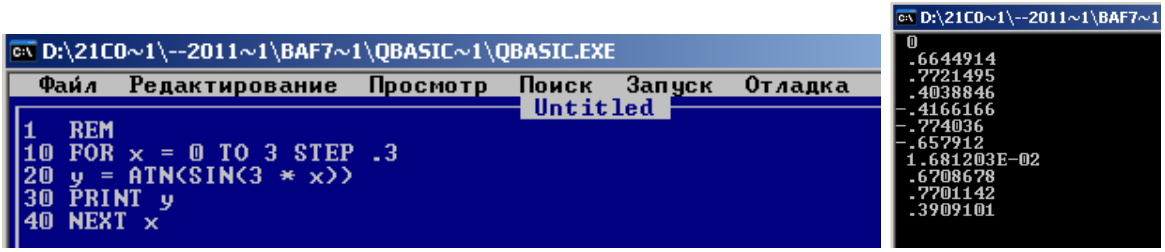
Таблиця для проведення обчислювального експерименту.

№ експерименту	Рівняння	Змінні	Результат
1	$4x^2+7x-2=0$	4; 7; -2	0,25; -2
2	$16x^2-40x+25=0$	16; -40; 25	1,25
3	$2008x^2-x+1001=0$	2008; -1; 1001	Немає коренів

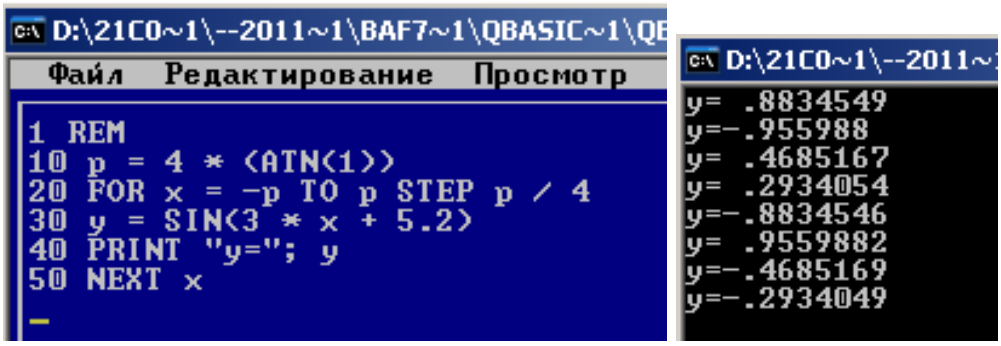
Програма 2. Скласти програму для обчислення квадратів двозначних чисел. Зберегти під ім'ям **PROG2**.



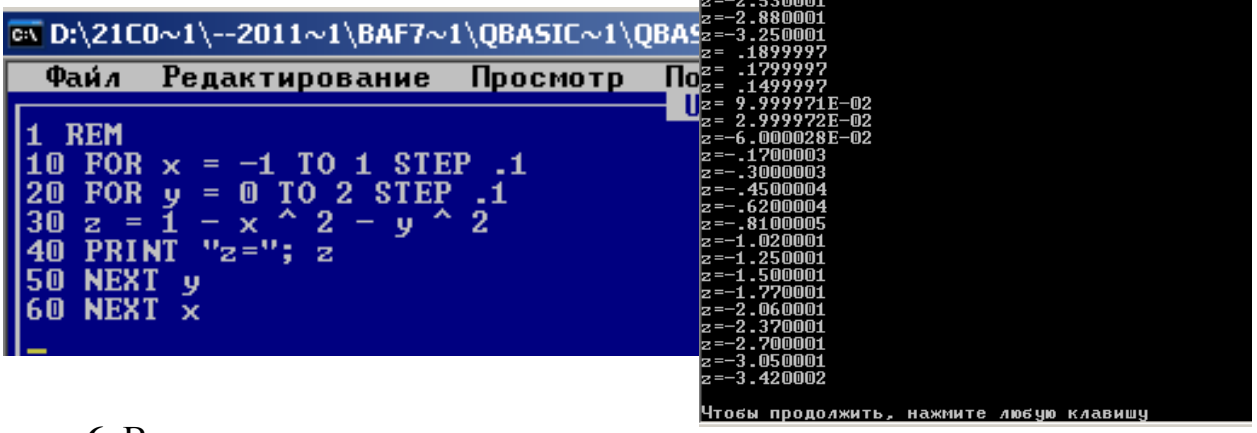
Програма 3. Обчислити значення функції $y = arctg(\sin 3x)$ при $x \in [0;3]$, з кроком 0,3. Зберегти під ім'ям **PROG3**.



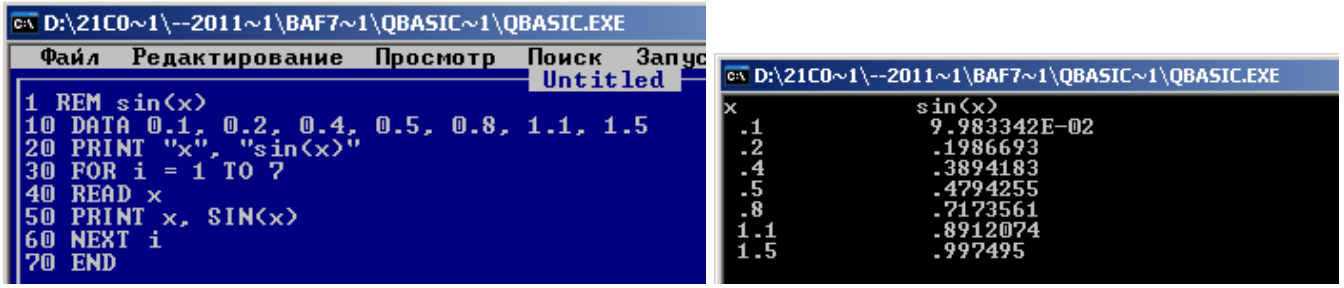
Програма 4. Обчислити значення функції $y = \sin(3x + 5{,}2)$ на проміжку $[-\pi\,;\pi]$ з кроком $\frac{\pi}{4}$. π = 4*ATN(1). Зберегти під ім'ям **PROG4**.



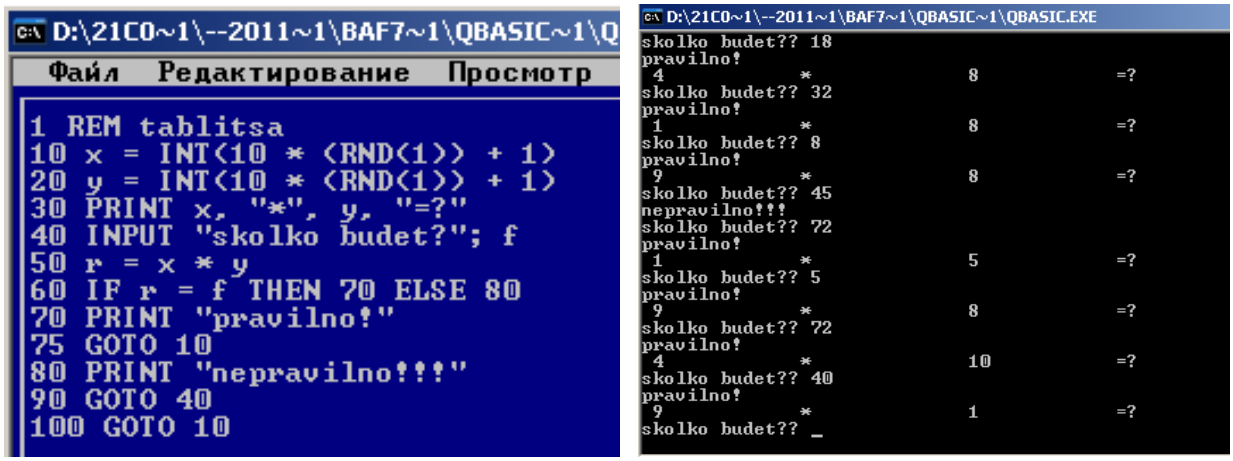
Програма 5. Обчислити значення функції $z = 1 - x^2 - y^2$ при $x \in [-1;1]$, $y \in [0;2]$ з кроком 0,1. Зберегти під ім'ям **PROG5**.



Програма 6. Вивести на екран дисплея таблицю значень функції sin(x) для x=0,1; 0,2; 0,4; 0,5; 0,8; 1,1; 1,5. Зберегти під ім'ям **PROG6**.

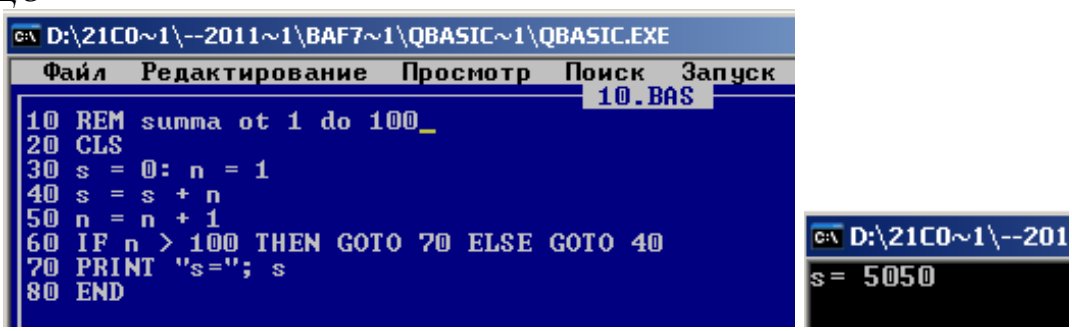


Програма 7. Скласти програму для перевірки таблиці множення. Зберегти під ім'ям **PROG7**.

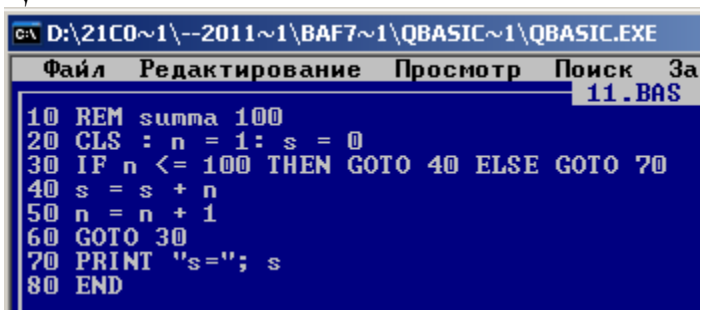


Програма 8. Обчислити суму цілих чисел від 1 до 100. Зберегти під ім'ям **PROG81, PROG82, PROG83**.

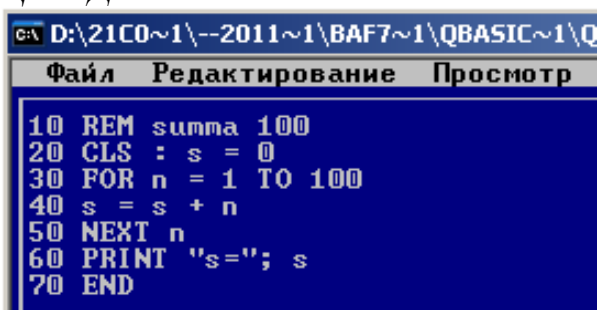
Цикл ДО



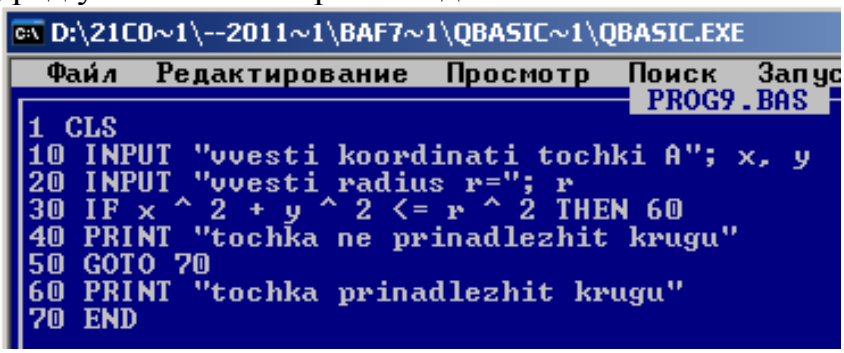
Цикл ПОКИ

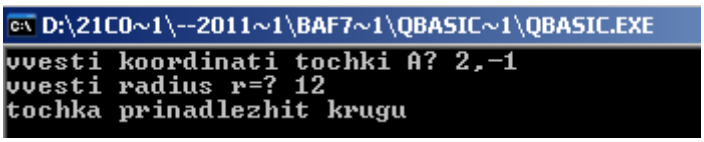


Цикл ДЛЯ



Програма 9. Написати програму, яка визначає, чи належить *точка A(x,y)* кругу, радіус якого *R*. Зберегти під ім'ям **PROG9.**





Завдання для самостійної роботи

Завдання 1. Скласти програму для обчислення квадратів двозначних чисел від 10 до 30.

Завдання 2. Скласти програму для обчислення кубів чисел першої дюжини.

Завдання 3. Скласти програму для обчислення площі круга при відомому значенні радіуса R. $S = \pi R^2$

Завдання 4. Обчислити значення функції $z = 1 - x^2 - y^2$ при $x \in [-1;0]$, $y \in [0;1,4]$ з кроком 0,2.

Завдання 5. Скласти програму для обчислення площі трапеції.

$S = \frac{a+b}{2}h$, *a i b* — основи трапеції, *h* — висота трапеції.

Завдання 6. Скласти програму для обчислення площі поверхні циліндра та його об'єму. Вважати, що π =3,1416.

$S = \frac{\pi d^2}{2} + \pi dh$, $V = \frac{\pi d^2}{4}h$

Завдання 7. Обчислити значення функції $y = \sin(2x - 1)$ на проміжку [- π ; π] з кроком $\frac{\pi}{4}$

Завдання 8. Обчислити значення функції $y = \cos 2x - 1$ на проміжку [- π ; π] з кроком $\frac{\pi}{4}$

Завдання 9. Обчислити значення функції $y = 3\cos x + 1$ на проміжку [- π ; π] з кроком $\frac{\pi}{2}$

Завдання 10. Скласти програму для обчислення площі трикутника за формулою Герона.

$S = \sqrt{p(p-a)(p-b)(p-c)}$, де a, b, c – сторони трикутника, p – півпериметр.

Завдання 11. Скласти програму для обчислення значень функції

$$y = \begin{cases} \sqrt{x}, & x > 0 \\ x^2, & x \le 0 \end{cases}$$

Завдання 12. Скласти програму для обчислення значень функції при $x \in [-1;8]$ з кроком 1:

$y = \frac{x^2 - 2x + 5}{2\sqrt{x}}$.

Індивідуальні завдання

ЛІНІЙНІ АЛГОРИТМИ

Скласти алгоритм, та програму на мові програмування для обчислення виразу, використовуючи вам відомі оператори, службові слова та стандартні математичні функції.

1. $y = \sqrt{x^2 + \frac{2}{a}}$	при x = 7,2; a = 25,46.
2. y = |x| + $\sin^2 a^2$	при x = $-2\cdot10^3$; a = 2,7.
3. $y = (2x+1)\sqrt{x^3}$	при x = 7,25.
4. $y = \frac{2x^2+3x+4}{x^2+3x+4}$	при x = -5.
5. $y = ax^2 + \frac{b}{x^3}$	при a = 25; x = 10,25; b = 4.
6. $y = \frac{x^3}{(x+2)(x-1)}$	при x = $0{,}255\cdot10^{-2.}$
7. $y = \sqrt{x}\cdot \lg x$	при x = 1000.
8. $y = \sqrt{\cos x + \sin^2 x}$	при x = 2,65.
9. $y = \sqrt{2\sin^2 x - 5\sin x + 3}$	при x = 0,25.
10.y=7+8sinx -$2\sin^2$x	при x = 33,3.
11. $y = \sin^2 x + \sin x\sqrt{\cos x}$	при x = 2,2.
12. $y = \sin x\sqrt{\frac{1-\cos x}{2}}$	при x = 1,1.
13.y=5tgx + 3ctgx	при x = 3.
14.y=(m-2)cos2x+(m+3)sin2x	при x = 1,12; m = 4.
15. $y = \frac{10}{2+\cos^2 x}$	при x = 0,77.
16. $y = \frac{1}{3}x^3 - \frac{1}{4}x^2$	при x = 7,17.
17.y = $4x^3$ - $6x^2$	при x = 1000.
18.y = $(4 - 3x)^2$	при x = 25,67.
19. $y = \frac{x^2-3}{x^2-4x}$	при x = 15,15.
20. $y = \frac{2x-1}{6x-x^2-9}$	при x = -3.
21. $y = x\sqrt[3]{x-2}$	при x = 7.
22. $y = \frac{3x}{2+x^2}$	при x = 10,101.
23.y = lg cosx	при x = 2,212.
24.y = lg(2 - sinx)	при x = 1,111.
25.y = (2-sinx)(sinx+4)	при x = 0,317.
26.y = cosx($\sin^2$x+3) + sin 2x	при x = 0,55.
27.y = tgxcosx + tg2x + cos 2x	при x = 1,231.

АЛГОРИТМИ З РОЗГАЛУЖЕННЯМ

Індивідуальні завдання

Намалювати блок-схему, написати алгоритм, програму на мові програмування та виконати її в середовищі програмування. (Вхідні дані, якщо вони не задані, підібрати самостійно)**.**

1. Ввести два числа. Більше зменшити на 55.
2. Написати алгоритм, з якого за номером уроку в понеділок визначиться його назва.
3. Ввести два числа. Менше піднести до куба.
4. Написати алгоритм, з якого за назвою столиці визначиться назва держави.
5. Ввести два числа. Менше поділити на 5, а більше піднести до квадрата.
6. Написати алгоритм, з якого за номером автобуса визначиться його маршрут.
7. Ввести два числа. Більше помножити на 7,5, а менше збільшити на 2.
8. Написати алгоритм, з якого за номером дня в тижні можна визначити його назву.
9. Написати алгоритм, з якого за номером зони в приміському поїзді визначиться вартість квитка.
10. Ввести два числа. Більше замінити їхньою сумою, а менше поділити на 2.
11. Ввести два числа. Менше замінити їх добутком, а більше — півсумою.
12. Написати алгоритм, в якому за номером місяця визначиться номер кварталу.
13. Ввести два числа. Більше замінити сумою квадратів цих чисел, а енше — різницею їх кубів.
14. Написати алгоритм, в якому за номером місяця визначиться кількість днів у ньому.
15. Ввести два числа. Менше з них, замінити різницею квадратів цих чисел, а більше— півсумою цих чисел.
16. Написати алгоритм, в якому за номером місяця визначиться пора року.
17. Ввести два числа. Більше замінити піврізницею, а менше — півсумою кубів цих чисел.
18. Написати алгоритм, в якому за назвою магнітофона визначиться його вартість.
19. Ввести два числа. Менше замінити квадратом різниці цих чисел, а більше — кубом їх суми.
20. Написати алгоритм, з якого визначиться вартість квитка до Києва, залежно від виду транспорту.
21. Задані дійсні додатні числа x, y, z. Написати алгоритм, за яким можна вияснити, чи існує трикутник з довжинами сторін x, y, z.
23. Ввести три числа, суму першого і третього розділити на 3, а друге число збільшити у 2,5 раза.
24. Написати алгоритм, з якого за назвою місяця можна визначити його порядковий номер.
25. Ввести два числа, більше зменшити в 7 раз, а менше піднести до п’ятого степеня.

Індивідуальні завдання

Намалювати блок-схему, написати алгоритм, програму на мові програмування та виконати її в середовищі програмування. Використовувати один з вищеописаних циклів. Вхідні дані, якщо вони не задані, підібрати самостійно.

1. Обчислити добуток перших *10* натуральних чисел.
2. Обчислити суму перших *10* натуральних чисел.
3. Обчислити добуток квадратів перших *10* натуральних чисел.
4. Обчислити суму кубів перших *10* натуральних чисел.
5. Надрукувати таблицю значень функції $Y=X^3$, при зміні *X* від *2* до *12* (крок 2).
6. Надрукувати таблицю співвідношення між масою в фунтах і масою в кілограмах для значень від *1* до *10* фунтів з кроком *1* фунт (*1 фунт — 400 грам*).
7. Обчислити значення функції $Y=2X^2+7X^4$, якщо аргумент змінюється від *0* до *50* з кроком *3*.
8. Обчислити значення функції $K = 2\sqrt{x} + 7\sqrt[3]{x}$, якщо аргумент змінюється від *1* до *20*.
9. Обчислити значення функції $Z = \sqrt{2x} + \sqrt{7} - x^2 + 4$ для $x=1$ до *15* з кроком *2*.
10. Надрукувати таблицю переводу температури з градусів по шкалі Цельсія (*C*) в градуси по шкалі Фаренгейта (*F*) для значень температури від *15* до *30*°C з кроком 1°C. (Перевід здійснюється за формулою *F=1,8C+32)*.
11. Густина повітря змінюється з висотою за законом $\rho=\rho_o e^{-hz}$ ($\rho_o=1{,}29$кг/м^3,$z=1{,}25\cdot10^{-4}$м$^{-1}$). Надрукувати таблицю залежності густини від висоти для значень від *0* до *1000* метрів через кожні *100* метрів.
12. Обчислити значення функції $Y=1/x^2$, *x* змінюється від *1* до *100* з кроком *5*.
13. Обчислити значення функції $Y=1/x^2+1/x^3$, *x* змінюється від *1* до *20*.
14. Обчислити значення функції $Y=2x^2$, *x* змінюється від *-10* до *10*.
15. Обчислити значення функції $Y=\sin x+\cos^2 x^3$, *x* змінюється від *0* до *15*.
16. Скласти таблицю множення для числа *12*.
17. Обчислити значення функції $S = \frac{mn + 7n}{m - n}$, *n* змінюється від *-10* до *15*, а *m=25*.
18. Скласти таблицю множення для числа *11*.
19. Обчислити значення функції $Y=\ln x +\cos x/\sin^2 x$, *x* змінюється від *10* до *25*.
20. Обчислити суму квадратів перших *7* натуральних чисел.
21. Обчислити значення функції $Y = \frac{2x^2 + 7}{x^2 - 1}$, *x* змінюється від *10* до *100*.
22 . Обчислити значення функції $Y = \frac{k}{k + 7k^2}$, *k* змінюється від *1000* до *1050* з кроком *3*.
23. Обчислити значення функції $n = \frac{5{,}2x^3 + \sqrt{x}}{2\sqrt{x}}$, *x* змінюється від *10* до *50*.

Література

1. Абрамов С. А., ЗимаЕ.В. Начала программирования на языке Паскаль. — М.: 1987.
2. Ахметов К. Курс молодого бойца. — М.: 1995.
3. ГлинськийЯ.М. Основи інформатики та обчислювальної техніки. Частина 1: Експериментальний підручник для 10 класу. — Львів.: 1996.
4. ГлинськийЯ.М. Основи інформатики та обчислювальної техніки. Частина 2: Експериментальний підручник для 10 класу. — Львів.: 1997.
5. ГлинськийЯ.М. Основи інформатики та обчислювальної техніки. Частина 3: Експериментальний підручник для 10 класу. — Львів.: 1996.
6. ГлинськийЯ.М. Основи інформатики та обчислювальної техніки. Частина 4: Експериментальний підручник для 10 класу. — Львів.: 1996.
7. Задачи и упражнения по программированию./ Под. ред. Савельева А. Я. — Ч 1. — М., 1989.
8. Задачи и упражнения по программированию./ Под. ред. Савельева А. Я. — Ч 2. — М., 1989.
9. Задачи и упражнения по программированию., под. ред. Савельева А. Я. — Ч 3. — М., 1989.
10. Задачи и упражнения по программированию., под. ред. Савельева А. Я. — Ч 4. — М., 1989.
11. Задачи и упражнения по программированию., под. ред. Савельева А. Я. — Ч 5. — М., 1989.
12. Іванишин М. Воробей О. Дидактичний матеріал з основ інформатики та обчислювальної техніки. Частина 1. — Івано-Франківськ, 1997.
13. Іванишин М. Воробей О. Дидактичний матеріал з основ інформатики та обчислювальної техніки. Частина 2. — Івано-Франківськ, 1997.
14. Руденко В. Д., Макарчук О. М., Патланжоглу Н. О. Практичний курс інформатики. — К.: 1997.
15. Светозарова Г. И. и др. Практикум по программированию на языке Бейсик. М.: — 1988.
16. Терминологический словарь по автоматике, информатике и вычислительной технике. — М.: 1989.
17. Чернов .И. Программирование на алгоритмических языках Бейсик, Фортран, Паскаль. — М.: 1991.
18. Юрчишин В. М. Інформатика, програмування і ЕОМ. — К.: 1991.
19. http://www.programmer.zp.ua/blsh/index_u.php
20. http://ostriv.in.ua/index.php?option=com_content&task=category&sectionid=34&id=420&Itemid=780&ft=1
21. http://www.gaps.tstu.ru/win-1251/lab/qb/win-1251/basic_met.html

ДЛЯ НОТАТОК