Основи програмування

Заняття №2

1. Що повинна робити програма?

Оскільки програма являє собою опис способу вирішення певної задачі, записаний у мові з жорстко заданими , формалізованими правилами, то вона виконуватиме лише ті команди які ми їй сказали. У програмуванні слова не мають двозначностей як у людській мові, яку щоб розуміти , не раз доводиться застосовувати логіку та інтуїцію. З мовами програмування все набагато простіше , одне слово – одна команда, немає двозначностей.

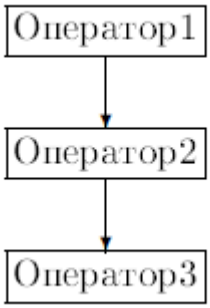
2. Що таке Алгоритм?

Спосіб вирішення задачі, розкладений у логічну послідовність елементарних дій та зв'язків між ними, називається ***алгоритмом***. Зрозумівши алгоритм, його вже не важко записати у будь-якій відомій мові програмування. Отже текст програми є лише формою втілення деякого алгоритму, а алгоритм є змістом, або сенсом тексту програми.

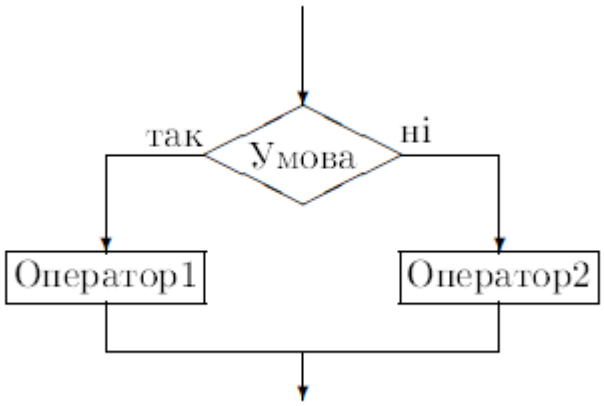
Отже, **Алгоритм** – це сукупність елементарних операцій та правил, які визначать, в якому порядку ці операції виконуються .

3. Поняття і приклади алгоритму?

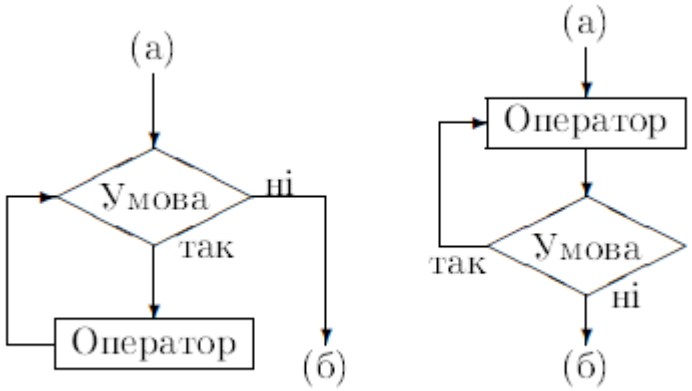
1.Найпростіші позначення в блоках схеми



2.Послідовне виконання операторів



3.Умовна конструкція (розгалуження)



4.Цикли з передумовою та постумовою

Зручним та наочним способом опису алгоритму є графічне зображення у вигляді блок- схеми. Кожен оператор зображується прямокутником; напис у прямокутнику вказує сенс оператору(1).

В найпростішому випадку оператори з'єднані між собою стрілками таким чином, що з кожного оператору виходить рівно одна стрілка, яка йде на вхід наступного оператору. Це означає, що оператори утворюють ланцюжок і виконуються послідовно, один за одним. Так, у прикладі, показаному на рис. 2, спочатку виконується оператор 1, потім оператор 2, а за ним — оператор 3.

Зрозуміло, що одного лише послідовного виконання недостатньо для вирішення всіх задач програмування. Справді, в послідовних програмах вся сукупність дій жорстко задана наперед, отже програма не може гнучко змінювати свою поведінку залежно від поточної ситуації. Таку гнучкість, або здатність приймати рішення «на ходу», абсолютно необхідну для всіх реальних задач, забезпечує ***умовний* блок**, який ще називають *розгалуженням* (рис. 3). Коли на умовний блок передається виконання (за стрілкою, що на рисунку входить зверху), спочатку в поточному стані пам'яті перевіряється умова. Якщо умова істинна то виконується оператор 1, в протилежному випадку виконується оператор 2. Далі, незалежно від того, яким з двох шляхів пішло

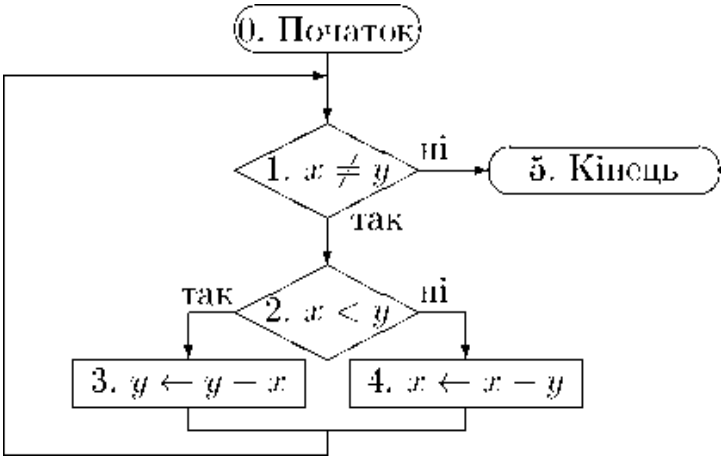
виконання умовного блоку, управління передається на наступний блок, на який вказує стрілка, що на даному рисунку виходить вниз.

Хоча умовна конструкція вже значно збагачує програмування, набагато розширюючи коло можливих програм, цих програмних структур ще зовсім не достатньо. Залишається ще один фундаментальний різновид програмних структур: повторювати деяку дію доти, поки залишається істинною деяка умова. Таку конструкцію **називають *циклом*** (рис. 4). Отже, циклічна конструкція забезпечує багаторазове виконання одного й того самого оператору (його називають *тілом циклу),* поки не буде досягнуто певної мети. Важливо, що кількість повторів заздалегідь невідома, вона визначається по ходу виконання алгоритму: тіло треба повторити не 5, 10 чи 100 разів, а стільки, скільки виявиться потрібним, щоб стала хибною умова.

Увага! На перший погляд може здатися, що ця схема нічим принципово не відрізняється від умовної структури: як в умовній конструкції, так і в циклі перевіряється умова, і залежно від неї виконання алгоритму продовжується одним з двох можливих шляхів. Але дуже важлива відмінність полягає в тому, що в умовній конструкції обидві гілки просувають виконання алгоритму ближче до завершення, тоді як в циклічній конструкцій одна з гілок направлена в бік завершення, а друга — у зворотній бік, повертаючи хід алгоритму до попередньої точки.

Розрізняють два основні різновиди циклу. Цикл з *передумовою* (на рисунку ліворуч) полягає в тому, що спочатку перевіряється умова, і, якщо вона істинна, виконується тіло циклу, після чого ця процедура повторюється знов. Неважко помітити, що такий цикл в загальному випадку забезпечує виконання тіла 0 чи більше разів: справді, якщо в момент входу в цикл за стрілкою (а) умова вже виявиться хибною, то управління одразу передається за стрілкою (б), обминаючи тіло. Цикл з *постумовою* (праворуч), натомість, спочатку виконує тіло, а потім вирішує, чи треба продовжувати далі. Такий цикл забезпечує виконання тіла 1 чи більше разів.

Приклад алгоритму:



Блок-схема алгоритму Евкліда (для знаходження найбільшого спільного кратного двох чисел)

Розглянемо в деталях процес виконання алгоритму, заданого блок-схемою . Алгоритм являє собою циклічну конструкцію з умовою в блоці 1, тілом якої є умовний оператор, у якого умова міститься в блоці 2, а гілками є оператори 3 та 4.

Нехай початковий стан пам'яті обчислювальної машини складається з двох змінних, значення яких відповідно x = 15 y = 9. Після запуску алгоритму управління передається па оператор розгалуження під номером 1. Перевірка умови x = y при підстановці значень змінних з поточного стану пам'яті дає 15 = 9 — істинне твердження. Оператор розгалуження лише перевіряє умову, не присвоюючи нових значень, тому стан пам'яті залишається незмінним: x =15 y = 9. За стрілкою, поміченою словом «так», управління передається па оператор під номером 2, теж оператор розгалуження. Підстановка значень змінних дає співвідношення (15 < 9), ця умова хибна, тому за стрілкою з позначкою «ні» управління передається на оператор 4, стан пам'яті поки що залишається незмінним.

Виконання оператору 4 полягає в тому, що обчислюється значення виразу при підстановці значень змінних: x — y = 15 — 9 = 6. Оператор присвоювання змінює стан пам'яті: значенням змінної x відтепер стає число 6, а значення змінної ***y*** залишається таким, як було раніше, тобто 9. За стрілкою проходимо знову до умовного оператору 1, Тіло циклу вже виконалось один раз, і тепер машина перевіряє, чи потрібно продовжувати виконання циклу.

Виконання оператору 1 в новому стані пам'яті встановлює, що умова x ≠y істина (6 ≠ 9), це призводить до продовження циклу. Відбувається перехід до оператору 2. Він, в свою чергу, передає управління на оператор 3, оскільки істинною виявляється умова x < y (6 < 9). Результатом виконання оператору 3 стає новий стан памяті, в якому x=6(значення не змінилося), y=3 (нове значення). Управління пердається на оператор 1

Оператор 1 передає управління на оператор 2 (виконання циклу продовжується), оскільки твердження x ≠ y істинне. Оператор 2 з'ясовує, що x < y хибне та передає управління на оператор 4. Той в свою чергу присвоює змінній *x* нове значення яке обчислюється з виразу x -y при підстановці наявних в поточному стані значень змінних, це значення дорівнює 3. Управління знов передається на перевірку умови циклу, блок 1.

Цього разу умова x = y виявляється хибною, і управління передається на оператор 5, який є завершальним. Отже, алгоритм завершує свою роботу, його результатом є прикінцевий стан пам'яті: x = 3 y = 3.

Весь процес виконання алгоритму зручно зобразити у вигляді таблиці. Перша колонка таблиці — номер кроку процесу виконання, друга — номер блоку, який на цьому кроці виконується. В наступних колонках наведено значення змінних, які складають стан пам'яті *після* виконання даного кроку. Якщо на даному кроці значення деякої змінної змінилося, домовимося нове значення позначати жирним шрифтом. В останню колонку будемо записувати істинність чи хибність умов в розгалуженнях та

циклах, а також номер блоку, на який здійснюється перехід. Таким чином, описаному вище процесу застосування алгоритму до заданого початкового стану пам'яті відповідає табл. 1.

Щодо даного прикладу залишається сказати, що алгоритм, який тут розглядався, є алгоритмом Евкліда для обчислення найбільшого спільного дільника двох чисел.

Табл. 1. Протокол процесу виконання алгоритму Евкліда

Крок	Блок	x	У	Примітка
0	0	15	9	Початок. Перехід до 1
1	1	15	9	*x* = y істинне, перехід до 2
2	2	15	9	*x* < *y* хибне, перехід до 4
3	4	6	9	перехід до 1
4	1	6	9	x = y істинне, перехід до 2
5	2	6	9	x < y істинне, перехід до 3
6	3	6	3	перехід до 1
7	1	6	3	x = y істинне, перехід до 2
8	2	6	3	x < y хибне, перехід до 4
9	4	3	3	перехід до 1
10	1	3	3	x = y хибне, перехід до 5
11	5	3	3	Алгоритм завершено

4. Для чого нам потрібні алгоритми?

Оскільки лишень правильна послідовність дій дозволить нам досягнути бажаного результату, то можна сміливо сказати що алгоритми є незамінними в програмуванні.

Тепер розберемося з тим, що таке алгоритми в реальному житті. При приготуванні різних страв господиня керується рецептом. Наприклад, при приготуванні макаронів потрібно виконати наступний алгоритм дій:

- Довести воду до кипіння.
- Посолити її.

- Засипати макарони.
- На малому вогні, помішуючи їх, довести до кипіння.
- Після того як вони зварилися, злити воду.
- Додати вершкове масло.
- Розмішати до однорідного стану.
- Страва готова.

Якщо подивитися на це очима програміста, то це звичайний лінійний алгоритм, написаний у текстовому вигляді. Так що не так вже й рідко в нашому житті *хитромудре* це поняття зустрічається.

5. Змінні, загальні поняття?

Програма здійснює свої обчислення над деякими величинами, або даними. Наприклад, програма знаходження середнього арифметичного з двох чисел обробляє величини a, b (вихідні дані) та *m* (результат).

В програмуванні, як і в математиці, прийнято такі величини називати *змінними*. Кожна змінна в програмі повинна мати своє *ім'я* — в даному прикладі іменами змінних є «a», «b» та *«m»*. Змінна в кожен момент часу має деяке *значення,* але значення кожної змінної в процесі роботи програми може змінюватися. Наприклад, програма обрахунку середнього арифметичного для того і призначена , щоб значенням змінної m наприкінці її роботи стало число $\frac{a+b}{2}$ (на початку роботи програми це значення було ще невідомим).

Сукупність значень всіх змінних в деякий момент виконання алгоритму (програми) разом утворює поточний *стан пам'яті* обчислювальної машини. Так, в попередньому прикладі можна розглянути стан пам'яті, поклавши значення $a = 2$, $b = 6$, або інший стан: $a = 28$, b = 32.

Якщо відомі значення змінних, то алгоритм може обчислювати значення різноманітних *виразів,* що складаються з імен змінних, числових констант та арифметичних операцій. Наприклад, якщо дано значення змінних $a = 28$, $b = 32$, то результатом обчислення виразу $a + b$ стане значення 60, а значенням виразу $\frac{a+b}{2}$ — число 30.

В процесі роботи алгоритму змінній може *присвоюватися* нове значення, отримане в результаті обчислення певного виразу. Присвоювання будемо позначати як *«x <- E»*, де *x* — ім'я змінної, E — вираз. Виконати присвоювання означає обчислити в даному стані (тобто при даних значеннях змінних) значення виразу Ета надалі вважати його значенням змінної x. Результатом присвоєння є новий стан памяті , в якому змінна x має нове значення, а інші змінні мають ті ж значення, що і в попередньому стані. Наприклад, якщо в стані a = 28, Ь = 32 виконати присвоювання $a <- a + b$, отримаємо повий стан a = 60, b = 32.

В той момент, коли змінній присвоюється значення, її старе значення безповоротно втрачається. Щоб підкреслити цю основну властивість, дану операцію називають ще ***деструктивним***, або руйнівним присвоюванням.

Присвоювання є частковим випадком операторів. Під ***операторами*** розуміють дії, які може виконувати машина, та з яких складаються програми.

Завдання :

If-else

1. Створити програму, що перевіряє і повідомляє на екран, чи є ціле число записане в змінну n, парним або непарним.
2. Створити програму, що виводить на екран найближче до 10 з двох чисел, записаних в змінні m і n. Наприклад, серед чисел 8,5 і 11,45 найближче до десяти 11,45.
3. У три змінні a, b і c записані три дійсних числа. Створити програму, яка буде знаходити і виводити на екран речові коріння квадратного рівняння $ax^2 + bx + c = 0$, або повідомляти, що коріння немає.

Loops

1. Створіть програму, що виводить на екран всі чотиризначні числа послідовності 1000 1003 1006 1009 1012 1015.
2. Створіть програму, що виводить на екран перші 55 елементів послідовності 1 3 5 7 9 11 13 15 17 ....
3. Створіть програму, що виводить на екран всі невід'ємні елементи послідовності 90 85 80 75 70 65 60 ....
4. Створіть програму, що виводить на екран перші 20 елементів послідовності 2 4 8 16 32 64 128 ....
5. Виведіть на екран всі члени послідовності 2an-1-1, де a1 = 2, які менше 10000.
6. Виведіть на екран всі двозначні члени послідовності 2an-1 + 200, де a1 = -166.
7. Створіть програму, яка обчислює факторіал натурального числа n, яке користувач введе з клавіатури.
8. Виведіть на екран всі позитивні подільники натурального числа, введеного користувачем з клавіатури.
9. Перевірте, чи є введене користувачем з клавіатури натуральне число - простим. Постарайтеся не виконувати зайвих дій (наприклад, після того, як ви знайшли хоча б один нетривіальний дільник вже ясно, що число складене і перевірку продовжувати не потрібно). Також врахуйте, що найменший дільник натурального числа n, якщо він взагалі є, обов'язково розташовується в відрізку $[2; \sqrt{n}]$.
10. Створіть програму, що виводить на екран 12 перших елементів послідовності 2an-2-2, де a1 = 3 і a2 = 2.

11. Виведіть на екран перші 11 членів послідовності Фібоначчі. Нагадуємо, що перший і другий члени послідовності рівні одиницям, а кожен наступний - сумою двох попередніх.

12. Для введеного користувачем з клавіатури натурального числа порахуйте суму всіх його цифр (заздалегідь не відомо скільки цифр буде в числі).