

МЕТОДИЧНА РОЗРОБКА

**для проведення лекційних занять
з НАВЧАЛЬНОЇ ДИСЦИПЛІНИ**

«Основи програмування та алгоритмічні мови»
для студентів денної форми навчання за спеціальністю
121 «Інженерія програмного забезпечення»

Розроблено викладачем

_____/Сайко Т.С./

Розглянуто та схвалено

на засіданні ЦК ПІ

Протокол №__ від __. _____. 2023 р.

Голова ЦК ПІ

_____/Ланська С.С./

«__»_____ 2023 р.

Лекція № 1

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Етапи складання програм. Алгоритм. Форми запису алгоритмів.
Лінійний алгоритм.

Мета заняття Ознайомити студентів із предметом вивчення дисципліни, етапами вирішення задач, поняттям алгоритму, властивостями алгоритму, способами його описання і базовими алгоритмічними структурами

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	5 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	65 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

Вступ

Програмування – це процес складання програми за заданим алгоритмом, для вирішення математичної або логічної задачі на комп'ютері.

Програма – описання алгоритму вирішення задачі, представлене на мові обчислювальної машини.

При програмуванні ключовими питаннями є:

- 1 Складання алгоритму.
- 2 Вибір мови програмування з врахуванням характеру задачі, особливостей алгоритму та наявності відповідного програмного забезпечення.
- 3 Організація вхідних і вихідних даних відповідної структури (масиви, структури, файли тощо).
- 4 Написання програмного коду на обраній мові програмування.

Одними з підходів до програмування є:

- 1 Структурне програмування – це процедурно-орієнтоване програмування.
- 2 Об'єктно-орієнтоване програмування – вид програмування при якому всі об'єкти реального світу моделюються за допомогою об'єктів мови програмування та зв'язуються.

1 Етапи вирішення задач

До етапів вирішення задач відносяться:

- 1 *Постановка задачі* – це широке описання задачі на природній мові.
- 2 *Розробка математичної моделі та визначення методу вирішення задачі* – процес визначення змінних та вказання розрахункових формул. При побудові математичної моделі відкидаються вторинні фактори і задачі переводиться на математичну мову.
- 3 *Розробка алгоритму вирішення задачі.*

Алгоритм – система формальних правил, яка чітко, однозначно, і за кінцеву послідовність дій, визначає процес виконання задачі (незалежно від її характеру), що приводить до правильного результату.

Алгоритм – процедура вирішення задач в термінах:

- операції, які йому необхідно виконати;
- послідовність цих операцій.

Алгоритм ділить процес вирішення задачі на ряд малих кроків, які можуть бути представлені у вигляді лінійних, розгалужених та циклічних процесів.

- 4 *Написання програмного коду* – процес переведення алгоритму у форму зрозумілу для комп'ютера.
- 5 *Відлагодження і тестування програми* – складається з наступних етапів:
 - 1 Синтаксична перевірка програмного коду.
 - 2 Відлагодження окремих частин програми.
 - 3 Відлагодження і тестування програми в цілому.
- 6 *Проведення обчислень і аналіз результатів* – передбачає оцінку відповідності результатів роботи програми змісту задачі.
- 7 *Публікація результатів або передача замовнику результатів.*
- 8 *Супроводження програми* – усунення помилок у роботі програми, виявлених під час користування кінцевим користувачем, консультування замовника по роботі з програмою, навчання користувачів.

2 Властивості алгоритму. Способи описання алгоритму

З визначення алгоритму витікають наступні його властивості:

- 1 *Детермінованість* – чіткість, ясність, однозначність.
- 2 *Кінцевість* – отримання результату за кінцеву кількість кроків.
- 3 *Масовість* – алгоритм повинен бути складений так, щоб він задовольняв вирішення не однієї конкретної задачі, а цілого класу подібних задач.

Для запису алгоритму використовуються наступні 4 способи (форми):

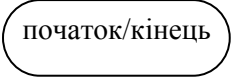
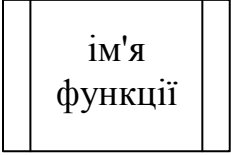
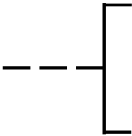
- 1 *Природний* (словесний, описовий) спосіб – кожна дія описується словами і дії нумеруються (див. задачу № 1).
- 2 *Графічний* (у вигляді блок-схем, структурних схем) спосіб – кожна дія (етап) позначається певною плоскою геометричною фігурою. Всі фігури (називаються блоками) мають логічну зв'язки, які позначаються лініями переходів.
- 3 *На спеціальній мові описання алгоритмів* – псевдокод.
- 4 *Програма на алгоритмічній мові*.

3 Базові алгоритмічні структури

У графічному способі описання алгоритму лінії переходів, які з'єднують блоки можуть бути тільки вертикальними або горизонтальними. Всі згини ліній можуть бути тільки під кутом 90° . Лінії переходів повинні мати стрілки на кінці в наступних випадках:

- 1 Якщо лінія має згин.
 - 2 Немає згину, але лінія вказує на напрямок справо-наліво, або знизу-вгору.
- Лінії переходів не мають перетинатись.

Для позначення певних дій (етапів) використовуються наступні геометричні фігури, які повинні мати співвідношення сторін: 1:1.5 (окрім блоків термінаторів та з'єднувачів. У блока-термінатора висота блока дорівнює половинній висоті інших блоків. Розмір блока-з'єднувача обирається пропорційно до розмірів інших блоків, але не менше розміру для вміщення всіх написів.)

 початок/кінець	– термінатор. Позначає початок/кінець програми, вхід/вихід з функції;
 операції	– процес. Позначає виконання операції або групи операцій;
 умови	– рішення (умова). Позначає перевірку умови, та вибір шляху виконання програми в залежності від умови (оператор if);
 опис виразів	– рішення (умова). Позначає перевірку умови, та вибір шляху виконання програми в залежності від умови (оператор switch);
 Вв./Вив. даних	– дані. Позначає процес введення/виведення даних (оператори cin, cout);
 опис виразів	– регулярний цикл (цикл з параметром, оператор for);
 ім'я функції	– попередньо визначений процес (відображує групу операцій, що визначені в іншому місці, наприклад у функції)
 № блок.	– з'єднувач. Використовується для обриву лінії та продовження її в іншому місці (на поточній сторінці), яке вказане номером наступного блоку;
 стор. № бл.	– з'єднувач. Використовується для обриву лінії та продовження її в іншому місці (на окремій сторінці), яке вказане номером сторінки та номером наступного блоку;
	– коментар (для додавання пояснювальних записів).

Яким би не був об'ємним або складним алгоритм рішення задачі, в ньому можливо виділити частини 3-х основних типів:

- 1 Лінійний алгоритм – блоки виконуються по 1 разу в порядку їх слідування і немає перевірки умови (див. задачу № 2).
- 2 Розгалужений алгоритм – блоки виконуються по 1 разу та є перевірка умов (див. задачу № 1, 3, 4).
- 3 Циклічний алгоритм – блоки можуть виконуватись по декілька раз (див. задачу № 4).

Задача № 1. За допомогою словесного та графічного способу вирішити наступну задачу – ввести з клавіатури три числа, знайти найбільше з них.

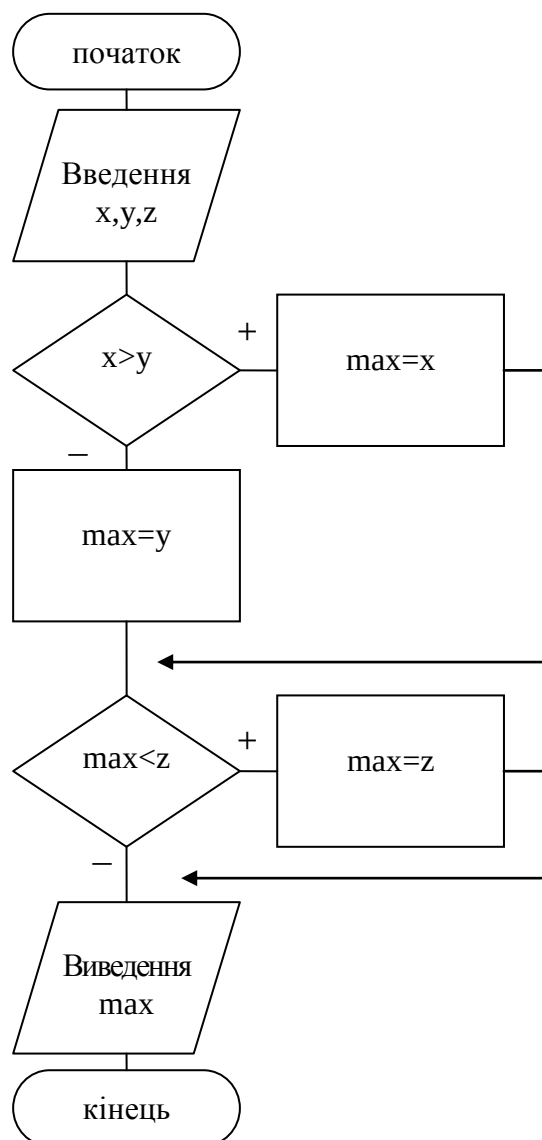
Вирішення.

Позначимо числа, які вводяться з клавіатури буквами x , y , z , а максимальне значення – $\max$. Опишемо різними способами алгоритм.

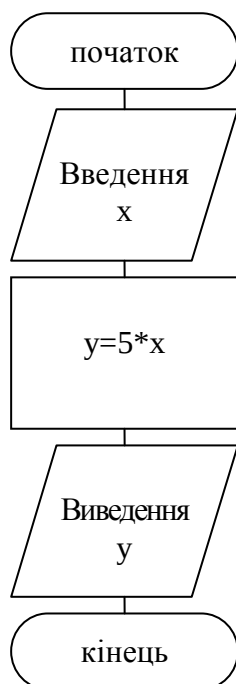
Словесний спосіб:

- 1 Початок.
- 2 Ввести з клавіатури три числа x , y та z .
- 3 Порівняти: якщо $x > y$, то $\max = x$, якщо ні – $\max = y$.
- 4 Порівняти: якщо $\max < z$, то $\max = z$.
- 5 Вивести результат – $\max$.
- 6 Кінець

Графічний спосіб:

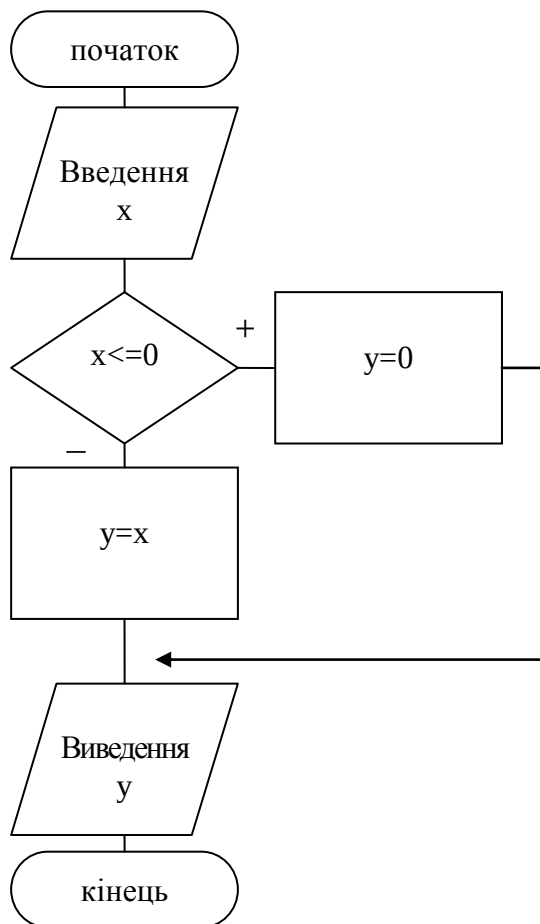


Задача № 2. Скласти блок-схему для вирішення наступної задачі – ввести з клавіатури X та обчислити значення функції $y=5x$.



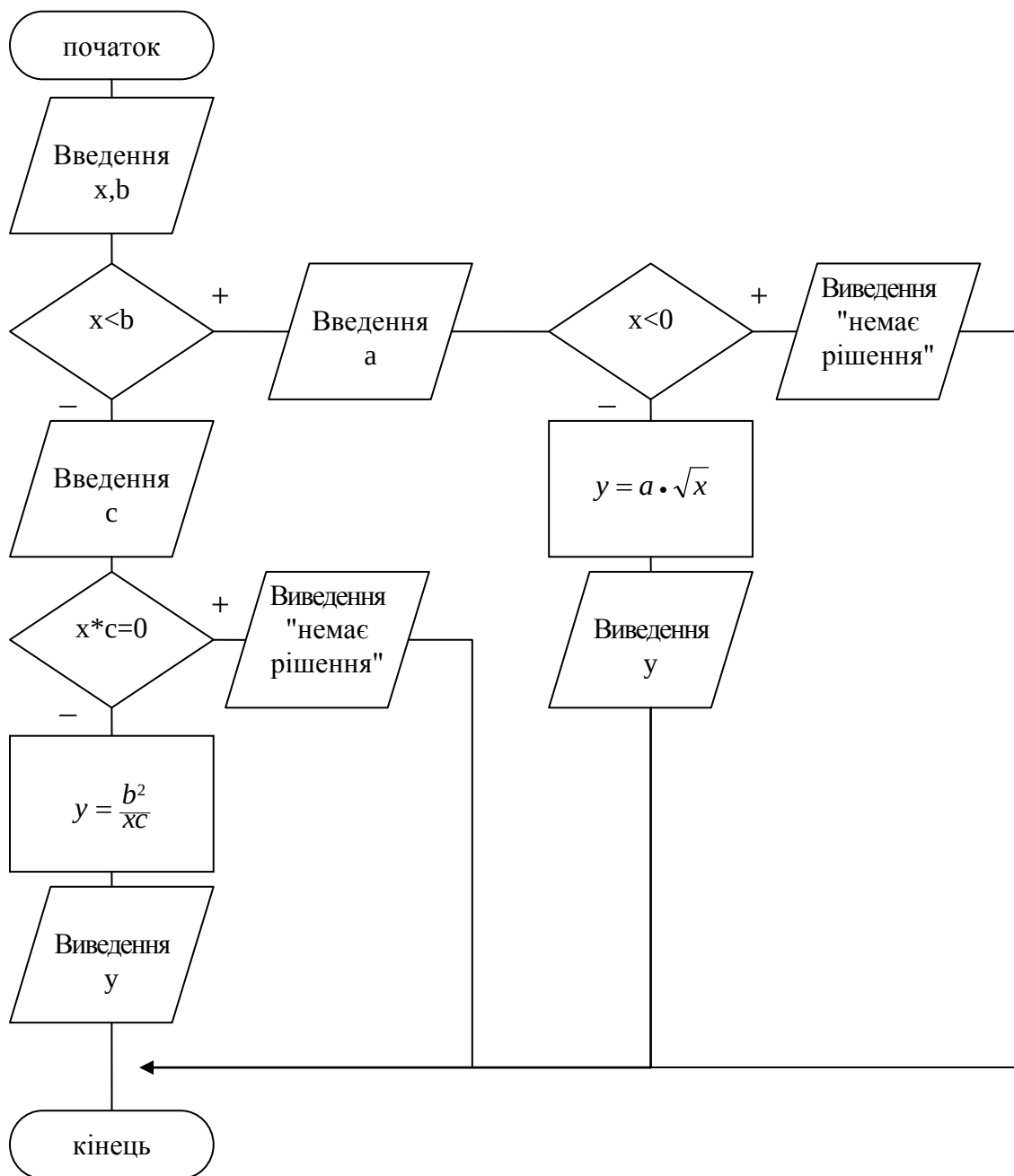
Задача № 3. Скласти блок-схему для вирішення наступної задачі – обчислити значення функції y , заданої умовою.

$$y = \begin{cases} 0, & \text{якщо } x \leq 0 \\ x, & \text{якщо } x > 0 \end{cases}$$

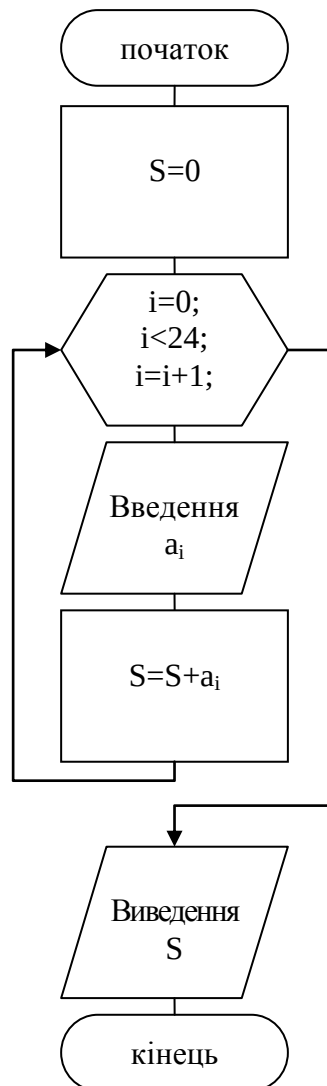


Задача № 4. Скласти блок-схему для вирішення наступної задачі – обчислити значення функції y , заданої умовою.

$$y = \begin{cases} a \cdot \sqrt{x}, & \text{якщо } x < b \\ \frac{b^2}{xc}, & \text{якщо } x \geq b \end{cases}$$



Задача № 5. Скласти блок-схему для вирішення наступної задачі – ввести з клавіатури 25 чисел та порахувати їх суму. $S = a_1 + a_2 + \dots + a_{25}$



Контрольні питання

- 1 Дайте визначення поняття алгоритм.
- 2 Опишіть властивості алгоритму.
- 3 Якими способами (формами) можливо задати алгоритм?
- 4 Яким чином позначаються дії у графічному способі описання алгоритму?
- 5 Під яким кутом можливо виконувати згин на лініях переходу у блок-схемах?
- 6 Яке співвідношення сторін має бути у блоків у блок-схемах?
- 7 Якою фігурою позначається блок початку/кінця програми (вхід/вихід з функції)?

- 8 Що являє собою лінійний алгоритм?
- 9 Якою фігурою позначається регулярний цикл for?
- 10 Якою фігурою позначається умова (оператор if)?
- 11 Що являє собою розгалужений алгоритм?
- 12 Якою фігурою позначається дія (група дій)?
- 13 Якою фігурою позначається умова (оператор switch)?
- 14 Якою фігурою позначається ввід/вивід даних?
- 15 Що являє собою циклічний алгоритм?

Домашнє завдання

Скласти блок-схему для вирішення наступної задачі – обчислити значення функції y , заданої умовою.

$$y = \begin{cases} a + b, & \text{якщо } x = 4 \\ a + \sqrt{b}, & \text{якщо } x = 2 \text{ або } x = 5 \\ 1, & \text{якщо } x > 5 \\ \frac{c}{\sqrt{d}}, & \text{якщо } \forall x \end{cases}$$

Лекція № 2

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Алгоритм розгалуження.

Мета заняття Ввести поняття алгоритму. Дати характеристику властивостям алгоритму. Показати способи опису алгоритмів

Час – 2 години

План проведення лекції

- 1 Поняття алгоритму.
- 2 Властивості алгоритму.
- 3 Способи опису алгоритмів

Навчальні матеріали лекції

1 Поняття алгоритму

Люди щоденно користуються різноманітними правилами, інструкціями, рецептами тощо, що складаються з певної послідовності команд (вказівок). Деякі з них настільки увійшли до нашого життя, що ми виконуємо їх майже не замислюючись, іноді кажуть, автоматично. Так для приготування яєчні потрібно виконати послідовність команд:

1. Поставити сковороду на плиту.
2. Покласти на сковороду шматочок вершкового масла.
3. Увімкнути конфорку.
4. Чекати, поки масло на сковороді розтане.
5. Розбити два яйця і вилити їх вміст на сковороду.
6. Посолити.
7. Чекати, поки загусне білок.
8. Вимкнути конфорку.

Розглянемо алгоритми розв'язування такої задачі.

Задача 1. Є посудина місткістю 8 л, яка заповнена рідиною, і дві порожні посудини місткістю 5 л і 3 л. Потрібно одержати в одній з посудин 1 літр рідини і повідомити в якій.

Розглянемо виконавця, який має таку систему команд:

1. Перелити рідину з однієї посудини в іншу.
2. Наповнити одну з посудин рідиною з іншої посудини.
3. Вивести повідомлення.

Для цього виконавця алгоритм розв'язування цієї задачі буде таким:

1. Наповнити 3-літрову посудину з 8-літрової.
2. Перелити з 3-літрової посудини в 5-літрову.
3. Наповнити 3-літрову посудину з 8-літрової.
4. Наповнити 5-літрову посудину з 3-літрової.
5. Вивести повідомлення: «1 л одержано в 3-літровій посудині».

Такі послідовності команд (вказівок) називають алгоритмами. **Алгоритм** – це скінченна послідовність команд (вказівок), що визначає, які дії та в якому порядку потрібно виконати виконавцю алгоритму, щоб досягти поставленої мети. Кожна команда алгоритму є спонукальним реченням, що вказує, яку дію має виконати виконавець алгоритму. Виконавцем алгоритму може бути людина, тварина, автоматичний пристрій, такий як робот, верстат з програмним керуванням, електронна обчислювальна машина тощо. Множину всіх команд, які може виконувати даний виконавець, називають **системою команд цього виконавця**. Наприклад, у систему команд виконавця, що виконуватиме другий з наведених вище алгоритмів, повинні входити такі команди:

Слово алгоритм походить від імені видатного вченого середньовічного Сходу Мухаммеда бен-Муси альХорезмі (783–850), який у своїх наукових працях з математики, астрономії та географії описав і використовував індійську позиційну систему числення, а також сформулював у загальному вигляді

ді правила виконання чотирьох основних арифметичних дій: додавання, віднімання, множення та ділення. Європейські вчені ознайомилися з його працями завдяки перекладам їх на латину. Під час перекладу ім'я автора було подано як *Algorithmus*. Звідси й пішло слово алгоритм. А розроблені ним правила виконання арифметичних дій вважають першими алгоритмами.

Однак пізніше під словом алгоритм стали розуміти правила знаходження найбільшого спільного дільника, які були викладені ще в працях великого давньогрецького математика Евкліда (III ст. до н.е.). За наших часів поняття алгоритму було узагальнено, і словом "алгоритм" стали позначати опис будь-якої послідовності дій. Поняття алгоритму є одним із фундаментальних у сучасній математиці й інформатиці.

2 Властивості алгоритму

Властивостями алгоритму є ***дискретність, визначеність, виконуваність, скінченність, результативність і масовість***.

Дискретність (лат. *discretus* – розділений, розривний) алгоритму означає, що його виконання зводиться до виконання окремих дій (кроків) у певній послідовності. Причому, кожна команда алгоритму повинна виконуватися за скінченний інтервал часу.

Визначеність (або детермінованість (лат. *determinans* – визначальний)) алгоритму означає, що для заданого набору значень початкових (вхідних) даних алгоритм однозначно визначає порядок дій виконавця та результат цих дій. Алгоритм не повинен містити команди, які можуть сприйматися виконавцем неоднозначно, наприклад «Узяти 2–3 ложки цукру», «Трохи підігріти молоко», «Вимкнути світло через кілька хвилин», «Поділити число x на одне з двох даних чисел a або b » тощо. Крім того, в алгоритмах недопустимі ситуації, коли після виконання чергової команди виконавцю незрозуміло, яку команду він повинен виконувати наступною.

Виконуваність алгоритму означає, що алгоритм, призначений для певного виконавця, може містити тільки команди, які входять до системи команд цього виконавця. Так, наприклад, алгоритм для виконавця «Учень першого класу» не може містити команду «Побудуй бісектрису даного кута», хоча така команда може бути в алгоритмі, який призначений для виконавця «Учень восьмого класу». Зазначимо, що *виконавець повинен лише вміти виконувати кожену команду зі своєї системи команд, і не важливо, розуміє він її чи ні. Говорять, що виконання алгоритмів виконавцем носить формальний характер: виконавець може не розуміти жодну з команд, може не знати мети виконання алгоритму, і все одно отримає результат.* Так, наприклад, верстат з програмним керуванням не розуміє жодної з команд, які він виконує, але завдяки своїй конструкції успішно виготовляє деталі.

Скінченність алгоритму означає, що його виконання закінчиться після скінченної (можливо, досить великої) кількості кроків і за скінченний час при будь-яких допустимих значеннях початкових даних. Наведені вище послідовності команд є скінченними, а наступна послідовність команд – нескінченна:

1. Взяти число 2.
2. Помножити взяте число на 10.
3. Додати до одержаного числа 5.
4. Якщо одержане число додатне, то виконати команду 3, якщо ні, то припинити виконання алгоритму.

Результативність алгоритму означає, що після закінчення його виконання обов'язково одержуються результати, які відповідають поставленій меті. Результативними вважаються також алгоритми, які визначають, що дану задачу не можна розв'язати або дана задача не має розв'язків при заданому наборі початкових даних.

Масовість алгоритму означає, що алгоритм може бути застосований до цілого класу однотипних задач, для яких спільними є умова та хід розв'язування та які відрізняються тільки початковими даними. Таким, на-

приклад, є алгоритм розв'язування квадратного рівняння, який дає змогу знайти дійсні корені квадратного рівняння з довільними дійсними коефіцієнтами або визначити, що при певних значеннях коефіцієнтів рівняння не має дійсних коренів. Масовим також є, наприклад, алгоритм побудови бісектриси довільного кута з використанням циркуля та лінійки. Однак, крім масових алгоритмів, складаються й застосовуються алгоритми, які не є масовими. Таким, наприклад, є алгоритм розв'язування конкретного квадратного рівняння (наприклад, $2x^2 + 5x + 20$) або алгоритм приготування конкретного салату (наприклад, грецького) на конкретну кількість осіб.

3 Способи опису алгоритмів

Словесний запис алгоритмів

Для зображення алгоритмів можна користуватися різними способами запису, які відрізняються ступенем наочності і точності. Деякі способи орієнтовані на виконавця - людину, інші - на виконання комп'ютером, треті є допоміжними (використовуються для полегшення міркувань). Розглянемо три способи подання алгоритмів: за допомогою звичайної мови спілкування, із використанням блок-схем і за допомогою алгоритмічної мови.

Словесний спосіб запису заснований на тій чи іншій природній мові спілкування (див. алгоритм Евкліда). Однак словесний запис алгоритму відрізняється від звичайних мовних конструкцій ретельнішим доббором слів і фраз, який не допускає повторень або двозначного тлумачення. Крім того, у запису алгоритму можуть використовуватися математичні символи і вирази.

Розглянемо словесний спосіб запису ще на одному простому прикладі. Маємо знайти модуль величини x (тобто значення $|x|$) і надати цього значення змінній y . Під час побудови алгоритму скористаємося визначенням модуля: $|x| = x$ при $x \geq 0$ і $|x| = -x$ при $x < 0$. Алгоритм можна записати у такий спосіб.

1 Початок.

- 2 Ввести числове значення величини x .
- 3 Якщо $x \geq 0$, то y надати значення x , інакше y надати значення $-x$.
- 4 Вивести значення y .
- 5 Кінець.

Словесний запис найчастіше застосовується на початковому етапі вивчення алгоритмів і призначається для використання алгоритму людиною. Однак ця форма запису алгоритму має два істотних недоліки. По-перше, вона недостатньо наочна і, по-друге, її важко безпосередньо перекласти мовою програми.

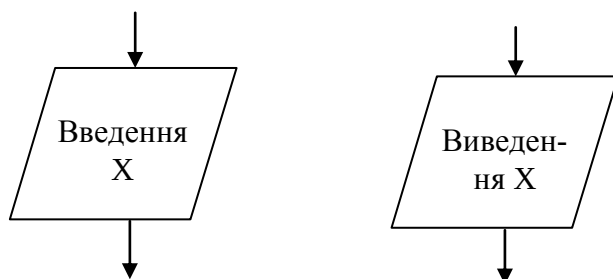
Блок-схеми алгоритмів

Наочною формою запису алгоритмів є блок-схеми, що складаються з геометричних фігур - блоків. Кожний блок відповідає певній дії. Наприклад, запис алгоритму починається і закінчується такими блоками:



Ці елементи називаються блоками початку і кінця алгоритму. Стрілки вказують напрямок виконання алгоритму. Блок **Початок** має одну вихідну стрілку, а блок **Кінець** - одну вхідну стрілку.

У алгоритмах часто використовуються команди введення і виведення значень. Цим командам відповідають блоки введення-виведення:



Тут лівий блок означає введення величини X , а правий блок - виведення X . За допомогою наведених вище блоків ви можете скласти найпростіший алгоритм введення-виведення величини X (див. рисунок 3.1).

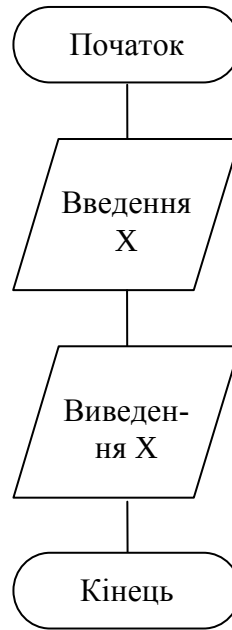
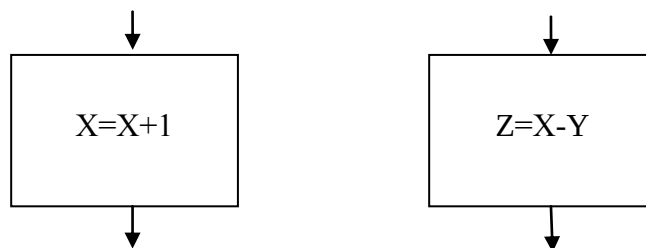


Рисунок 3.1. Блок-схема алгоритму введення виведення X.

Відповідно до цього алгоритму в програму вводиться значення величини X. Однак програма, що складається тільки з операції введення, навряд чи доцільна. Звичайно над введеною величиною виконуються певні дії, що позначаються прямокутними (операторними) блоками:



У середині прямокутників записані вирази, що виконуються над величинами. Лівий блок означає присвоювання змінній x значення суми $X+1$ (про операції присвоювання йтиметься далі). Правий блок відповідає за знаходження різниці $X-Y$ і надання значення різниці змінній Z . Операторні блоки можуть мати кілька входів і лише один вихід.

Зобразимо у вигляді блок-схеми найпростіший алгоритм обчислення квадрата якогось числа (див. рисунок 3.2):

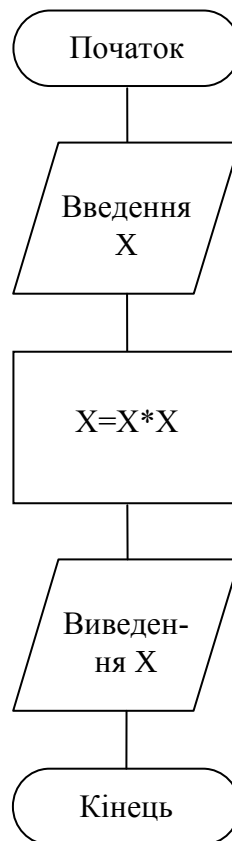
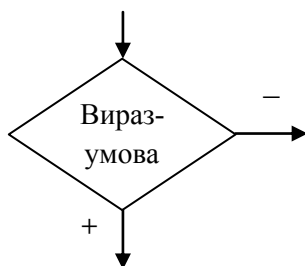


Рисунок 3.2. Блок-схема обчислення квадрата числа.

Відповідно до цього алгоритму вводиться величина X , потім обчислюється квадрат цієї величини (добуток $X * X$) і виводиться отримане значення.

Усі наведені вище блоки дозволяють організувати послідовне виконання інструкцій алгоритму. Однак на практиці часто виникають ситуації, коли залежно від виконання будь-якої умови маємо змінити послідовний хід обчислень. Прикладом такої умови є нерівність $X \geq 0$ в алгоритмі знаходження модуля числа x . У схему алгоритму логічна умова вводиться за допомогою умовного блока. Цей блок прийнято зображати у вигляді ромба з одним входом і двома виходами:



Якщо умова, зазначена на зображенні блока, виконується (умова має значення *Істина*), то відбувається перехід по стрілці +, якщо не виконується (значення *Хибність*) - по стрілці -. Завдяки умовному блоку обчислювальний процес ніби розгалужується, тобто умовний блок використовується для організації розгалуження.

Наведемо приклад алгоритму обчислення модуля числа (див. рис. 3.3):

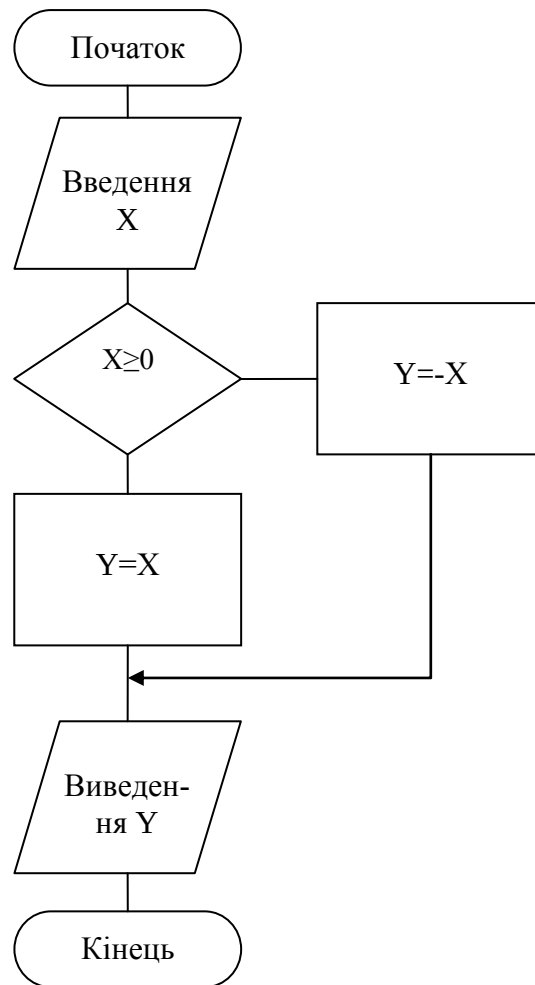


Рисунок 3.3. Блок-схема обчислення модуля числа.

Запис цього алгоритму обмежують блоки початку і кінця. За блоком початку алгоритму йде блок введення значень x , а за ним - умовний блок. В умовному блоці виконується перевірка умови $x \geq 0$ і в результаті перевірки здійснюється перехід по одній із гілок Так або Ні. На кожній із гілок розташований операторний блок надання значень змінній y . Після операції надання гілки алгоритму сходяться, і наступна інструкція алгоритму міститься в блоці виведення отриманого значення y .

Алгоритмічні мови

Блок-схеми є зручними для наочного подання алгоритмів, проте для складних алгоритмів вони стають громіздкими. Більш компактною формою подання алгоритму є його опис на алгоритмічній мові.

Алгоритмічною мовою називається штучна мова, призначена для подання алгоритмів. Алгоритмічні мови дозволяють подати алгоритм у вигляді тексту, який складається за певними правилами із застосуванням спеціальних слів, які називаються службовими. Кількість таких слів обмежена. Кожне службове слово має точно визначений смисл, призначення і спосіб застосування. Службові слова добираються так, щоб їх смисл в алгоритмічній мові перекликався із звичайним значенням слова. Алгоритм, поданий алгоритмічною мовою, легко сприймається, тому що він читається як звичайний текст.

Алгоритмічна мова, конструкції якої однозначно перетворюються на команди комп'ютеру, називається ***мовою програмування*** (від грец. *programma* — розпорядження, припис, оголошення). Текст алгоритму, записаний мовою програмування, називається ***програмою***.

Висновки

З алгоритмами ми зустрічаємося на кожному кроці. Вони вказують, яку послідовність дій необхідно виконати, щоб досягти потрібного результату. Кожний алгоритм розраховується на виконавця, який спроможний зрозуміти й виконати команди алгоритму. За алгоритмом процес розв'язання задачі постає як виконання скінченної послідовності окремих простих команд, кожна з яких точно визначає дії виконавця. Це розкриває шлях до створення технічних пристроїв, які сприймають і виконують алгоритми. Алгоритми подають у різних формах. Для наочного подання алгоритму використовуються блок-схеми. Подання алгоритму у вигляді тексту здійснюється за допомогою штучних мов. Навчальні алгоритмічні мови створюються для засвоєння основ

складання алгоритмів. Мови програмування орієнтовані на їх «розуміння» комп'ютером, який у такому разі виступає виконавцем алгоритму.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що таке алгоритм? Дайте визначення цього поняття.
- 2 Назвіть виконавців для таких алгоритмів:
 - а - спосіб розв'язання задачі, що записує на дошці вчитель;
 - б - інструкція про те, як завести автомобіль.
- 3 Назвіть відомі вам властивості алгоритмів.
- 4 Чи буде вважатися алгоритмом послідовність дій, що не приводить до будь-якого результату? Що таке результативність алгоритму?
- 5 Наведіть приклади властивості масовості алгоритму.
- 6 Назвіть відомі вам способи зображення алгоритмів.
- 7 Які переваги графічного зображення алгоритмів перед словесним записом?
- 8 Як властивість дискретності алгоритму пов'язана із зображенням алгоритму у вигляді блок-схеми?
- 9 Назвіть компоненти блок-схем алгоритмів.
- 10 Чи може умовний блок мати один вихід?
- 11 Що таке алгоритмічна мова?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Укажіть команди серед наведених речень:

- a) Закрити вікно.
- b) Не заважай читати.
- c) Котра година?
- d) Якщо йде дощ, візьми парасольку.
- e) $+ 2 \square 5$.
- f) Я живу в Києві.

2 Сформулюйте лінійні правила-алгоритми, які ви вивчали на уроках української мови, математики, інших предметів. Подайте ці алгоритми у вигляді блок-схем.

3 Виконайте алгоритм:

- 1. Накреслити відрізок АВ завдовжки 5 см.
- 2. Поставити вістря циркуля в точку А.
- 3. Побудувати коло, радіус якого дорівнює довжині відрізка АВ.
- 4. Поставити вістря циркуля в точку В.
- 5. Побудувати коло, радіус якого дорівнює довжині відрізка АВ.
- 6. Провести пряму через точки перетину побудованих кіл.

Яку б назву ви дали цьому алгоритму? Які геометричні задачі можна розв'язувати за цим алгоритмом? Складіть його блок-схему.

4 Складіть алгоритм приготування вашої улюбленої страви. Подайте його в словесній формі.

Лекція № 3

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Алгоритм вибору.

Мета заняття Ознайомити студентів із поняттям тернарної операції ?:, вивчити властивості та принципи використання оператора switch та оператора goto

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.:

Біном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Тернарна операція ?:

В тернарній операції ?:, на відміну від унарних і бінарних операцій, використовується три операнди і вона є єдиною тернарною операцією.

Вираз1 ? Вираз2 : Вираз3;

Першим обчислюється значення "виразу1". Якщо воно істинне, то обчислюється значення "виразу2", яке стає результатом. Якщо при обчисленні "виразу1" вийде 0, то як результат береться значення "виразу3".

З двох виразів "вираз2" та "вираз3" обчислюється тільки одно з них і ніколи обидва разом!

Наприклад,

`x < 0 ? -x : x ;` // обчислюється абсолютне значення x

2 Оператор switch

Оператор if дозволяє здійснити вибір тільки між двома варіантами.

Для того, щоб проводити вибір один з декількох варіантів використовується оператор switch.

Загальна форма запису:

```
switch(константний_вираз)
{
case константне_значення1:
    оператор1;
    [оператори;]
    break;
case константне_значення2:
```

```

                                оператор2;
                                [оператори;]
                                break;

. . .
case константне_значенняN:
                                операторN;
                                [оператори;]
                                break;

[default: оператор;]
}

```

Оператор виконується таким чином:

- 1) обчислюється вираз в дужках оператора switch;
- 2) обчислене значення порівнюється з мітками (константними значеннями) в опціях case;
- 3) порівняння проводиться до тих пір, поки не буде знайдена мітка, відповідна даному значенню, після цього виконується оператор відповідної гілки;
- 4) якщо відповідна мітка не знайдена, то виконується оператор в опції default.

Альтернатива default може бути відсутньою, тоді не буде проведено ніяких дій.

Оператор break здійснює вихід з оператора switch і перехід до наступного за ним оператора. За відсутності оператора break виконуватимуться всі оператори, починаючи з поміченого даною міткою і кінчаючи оператором в опції default.

Константний вираз (вираз, операнди якого константи) повинен бути цілого типу (включаючи char).

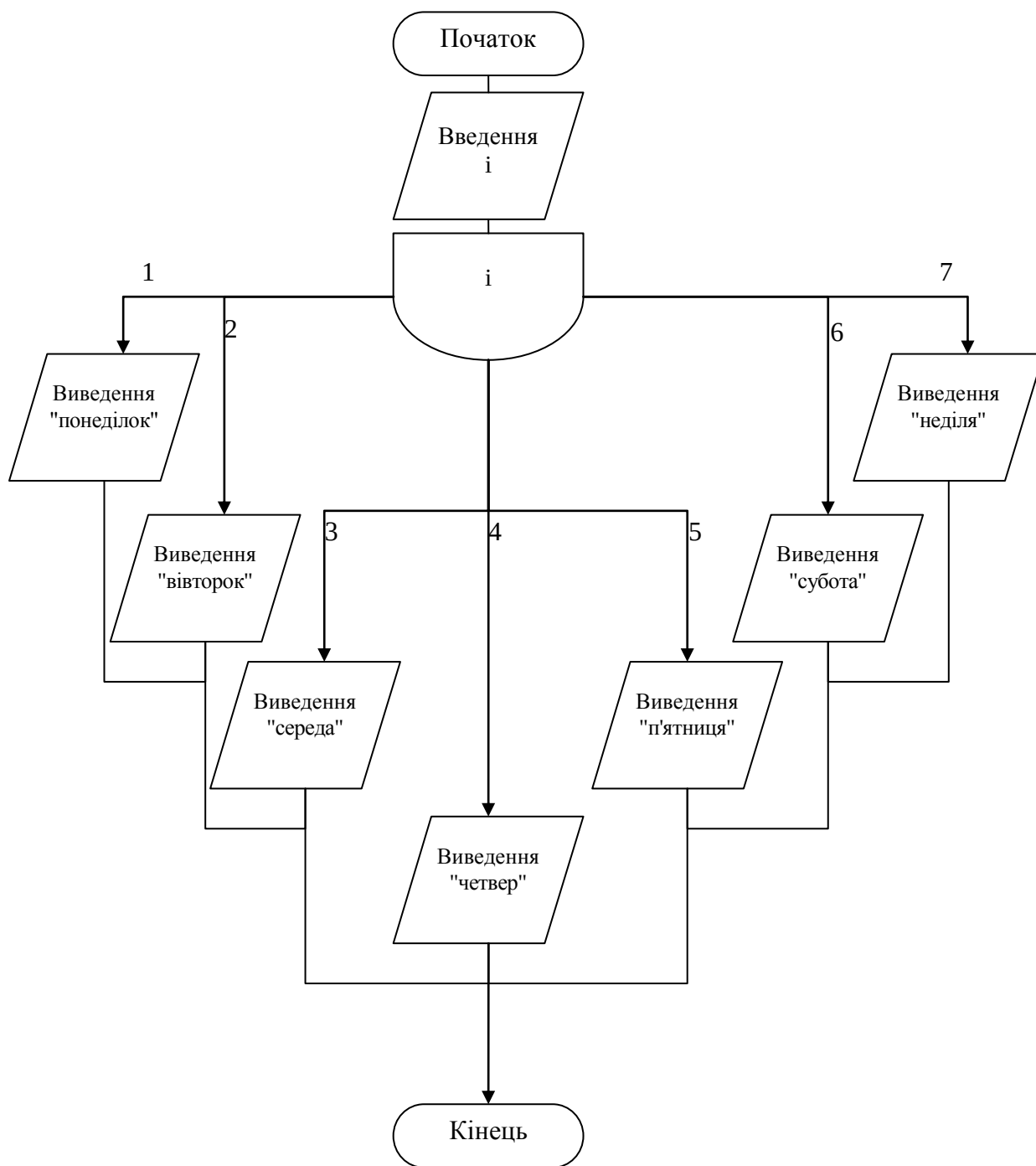
Наприклад. Розробити програму, що визначає день тижня по його введеному номеру. Програма повинна реагувати на невірний номер дня тижня.

```

#include <stdio.h>
#include <conio.h>
int main()
{

```

```
int i;
printf("\n Введіть номер дня тижня: ");
scanf("%i",&i);
switch(i)
{
    case 1:
        printf("\n Понеділок.");
        break;
    case 2:
        printf("\n Вівторок.");
        break;
    case 3:
        printf("\n Середа.");
        break;
    case 4:
        printf("\n Четвер.");
        break;
    case 5:
        printf("\n П'ятниця.");
        break;
    case 6:
        printf("\n Субота.");
        break;
    case 7:
        printf("\n Неділя.");
        break;
    default:
        {
            printf("\n Невірно введений");
            printf("номер дня тижня.");
        }
}
```



Результат роботи програми:

Введіть номер дня тижня: 2

Вівторок.

3 Оператор безумовного переходу goto

Загальна форма запису

```
goto мітка;
```

```
... .
```

```
мітка: <оператор>;
```

Виконання оператора `goto` викликає передачу управління в програмі операторові, поміченому міткою. У тілі тієї ж функції обов'язкова повинна бути присутня *мітка*.

Мітка – це звичайний ідентифікатор, областю видимості якого є функція. Для відділення мітки від оператора використовується двокрапка. Мітка з поміченим оператором може з'явитися в програмі як до оператора `goto`, так і після. Імена міток утворюються по тих же правилах, що і імена ідентифікаторів.

Наприклад,

```
if (size>12)
    goto a;
else
    goto b;
a: cost = cost*3;
flag=1;
b: s= const*flag;
```

Застосування `goto` порушує принципи структурного і модульного програмування, за якими всі блоки, з яких складається програма, повинні мати тільки один вхід і лише один вихід. Мова C/C++ реалізована таким чином, що можна програмувати без оператора `goto`.

Використання оператора `goto` виправдане, якщо необхідно виконати перехід з декількох вкладених циклів або перемикачів вниз по тексту програми або перейти в одне місце функції після виконання різних дій.

Не можна передавати управління всередину операторів `if`, `switch` і циклів. Не можна переходити всередину блоків, що містять ініціалізацію, на оператори, які стоять після ініціалізації.

Приклад помилкової роботи програми при неправильному використанні оператора безумовного переходу `goto`.

```
int k;
goto m;
. . .
{int a=3, b=4;
k=a+b;
m: int c=k+1;
```



```
. . .  
}
```

В даному прикладі при переході на мітку `m` не виконуватиметься ініціалізація змінних `a`, `b` і `k`.

Контрольні питання

- 1 Скільки операндів входить до складу тернарної операції?
- 2 Напишіть загальний вигляд тернарної операції ?:.
- 3 Опишіть призначення оператора `switch`.
- 4 Якого типу може бути константний вираз для порівняння з заданими варіантами у операторі `switch`?
- 5 Для чого призначене ключове слово `case` в операторі `switch`?
- 6 Для чого в операторі `switch` використовується оператор `break`?
- 7 Чи буде порушена логіка роботи програми в разі відсутності оператора `break` в гілках оператора `switch`?
- 8 Опишіть призначення гілки `default` в операторі `switch`.
- 9 Опишіть призначення оператора `goto`?
- 10 З яких символів можуть складатися імена міток?

Домашнє завдання

[1] с. 87-91, 97-98

Лекція № 4

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Циклічні алгоритми. Регулярний цикл. Знаходження суми, добутку, кількості.

Мета заняття Ознайомити студентів із різновидами циклічних структур, вивчити властивості та принципи використання операторів for, while та do-while, а також засвоїти правила використання управляючих конструкцій break та continue

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.:

Біном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

Розрізняють наступні види циклів:

- регулярні цикли;
- ітераційні цикли.

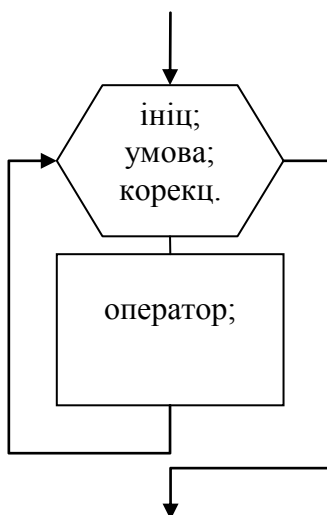
Група дій, що повторюються в циклі, називається його *тілом*. Одноразове виконання циклу називається його *кроком*.

1 Регулярний цикл for

Цикл for – цикл з фіксованим числом повторень, яке заздалегідь відоме.

Загальна форма запису:

```
for(ініціалізація; перевірка_умови; корекція)  
    <оператор>;
```



Для організації такого циклу повинні розглядатися три операції:

- ініціалізація лічильника;
- порівняння його величини з деяким граничним значенням;
- зміна значення лічильника при кожному проходженні тіла циклу.

Ці три операції записуються в дужках і розділяються крапкою з комою (;).

Перший вираз (*ініціалізація*) служить для ініціалізації лічильника. Ініціалізація здійснюється тільки один раз – коли цикл `for` починає виконуватися.

Другий вираз (*перевірка_умови*) служить для перевірки умови перед кожним можливим виконанням тіла циклу. Коли вираз стає брехнею (рівним нулю), цикл завершиться.

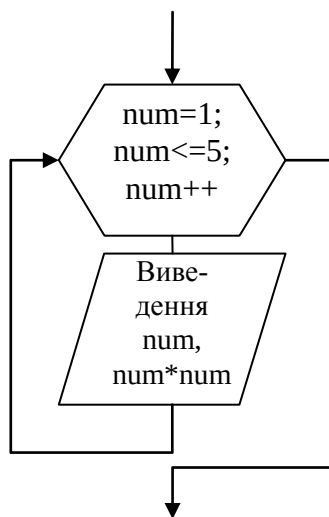
Третій вираз (*корекція*) обчислюється в кінці кожного виконання тіла циклу. Лічильник може як збільшуватися, так і зменшуватися.

Будь-який вираз може бути відсутнім, але символи крапки з комою (;), що їх розділяють, повинні бути обов'язково.

Наприклад.

Вивести на екран монітора числа від 1 до 5 та їх квадрати.

```
int num;
for(num=1; num<=5; num++)
    printf("\n %d * %d = %d", num, num, num*num);
```



Параметри, що знаходяться в заголовку циклу, можна змінити при виконанні операцій в тілі циклу.

Наприклад.

```
int k=2;
for(n=3; k<=25; )
{
    ...
}
```

```

        k = k*n;
    }

```

У циклі `for` часто використовується операція «кома» (,) для розділення декількох виразів. Це дозволяє включити в специфікацію циклу декілька виразів, які будуть ініціалізуватися або коректуватися. Вирази, до яких застосовується операція «кома», обчислюватимуться зліва направо.

Наприклад.

```
for(i=0, j=1; i<n; i++, j=i);
```

У C/C++ допускаються вкладені цикли, тобто коли один цикл знаходиться усередині іншого.

Приклад

```

for( i=0; i<n; i++)
{
    for( j=0; j<n; j++)
    {
        ...
    }
    ...
}

```

Приклади використання регулярного циклу з параметром.

1) Збільшення лічильника;

```

for (n=0; n<10; n++)
    оператор;

```

2) Зменшення лічильника;

```

for (n=10; n>0; n--)
    оператор;

```

3) Довільна зміна кроку коректування;

```

for (n=2; n>60; n+=13)
    оператор;

```

4) Можливість перевіряти умову відмінну від умови, яка накладається на число ітерацій;

```

for (num=1; num*num*num<216; num++)
    оператор;

```

5) Корекція може здійснюватися не тільки за допомогою складання або віднімання.

```

for (d=100.0; d<150.0; d*=1.1)
{
    <тіло циклу>;
}

```

```

}
for (x=1; y<=75; y=5*(x++)+10)
{
<тіло циклу>;
}

```

2 Ітераційні цикли **while** та **do-while**

У ітераційних циклах заздалегідь невідома кількість разів виконання циклу.

1) Цикл **while** – ітераційний цикл з передумовою.

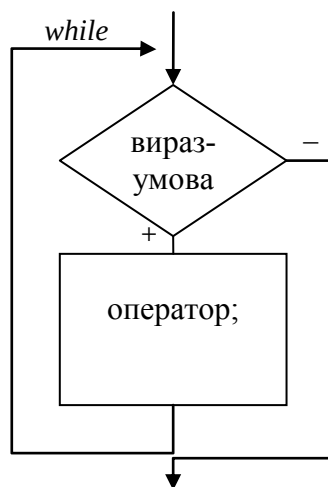
Загальна форма запису:

```

while (вираз-умова)
    оператор;

```

В якості <виразу-умови> найчастіше використовується відношення або логічний вираз. Якщо <вираз-умова> істинне (тобто не рівно нулю), то виконується оператор або група операторів, що входять в цикл **while**, та потім вираз перевіряється знову.



Послідовність дій, що складається з перевірки і виконання оператора, повторюється до тих пір, поки вираз не стане брехнею (тобто рівним нулю). Після цього відбувається вихід з циклу, і далі виконується оператор, що стоїть після оператора циклу. При побудові циклу **while**, в нього необхідно включити конструкції, що змінюють величину виразу, що перевіряється, так, щоб врешті-решт воно стало брехнею. Інакше виконання циклу ніколи не завершиться.

Цикл **while** – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу.

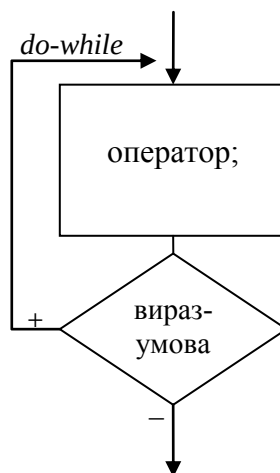
Наприклад.

```
k=5;
n=10;
while(k<n)
{
    printf("k=%d n=%d\n", до, n);
    k+=2;
    k=k+2;
    n++;
}
```

2) Do-while – ітераційний цикл з постумовою.

Загальна форма запису:

```
do
    оператор;
while (вираз-умова);
```



Цикл do-while — це цикл з постумовою, в якому істинність виразу-умови перевіряється після виконання всіх операторів, включених в тіло циклу. Тіло циклу виконується до тих пір, поки вираз не стане брехнею, тобто тіло циклу виконується хоч би один раз. Використовувати цикл краще всього в тих випадках, коли повинна бути виконана хоч би одна ітерація.

Наприклад.

```
do
{
    printf(" Введіть n>0");
    scanf("%d", &n);
}
while (n<0);
```

3 Оператори break і continue

У тілі будь-якого типу циклу можна використовувати оператора break, який дозволяє вийти з циклу, не завершуючи його. Оператор continue дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

1) break – оператор переривання циклу.

```
while (<вираз>)
{
    ...
    break;
    ...
}
```



Тобто оператор break доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

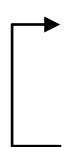
Наприклад.

Написати програму, яка підраховує суму чисел що вводяться з клавіатури до тих пір, поки не буде введено 100 чисел або число 0

```
for (s=0, i=1; i<100; i++)
{
    cin>>x;
    if (x==0) break; // якщо ввели 0, то підсумовування закінчується
    s+=x;
}
```

2) continue – оператор переходу до наступної ітерації циклу. Він використовується, коли тіло циклу містить галуження.

```
while (<вираз>)
{
    ...
    continue;
    ...
}
```



Наприклад.

Написати програму, яка підраховує кількість і суму додатних чисел

```
for ( k=0, s=0, x=1; x!=0; )
```



```

{
    cin>>x;
    if (x<=0) continue; //якщо ввели <=0, то перехід до наступної ітерації
    k++;s+=x;
}

```

При вкладених циклах дії операторів break і continue розповсюджується тільки на саму внутрішню структуру, в якій вони містяться.

Використання цих операторів можливе, але небажано, оскільки вони погіршують читаність програми, збільшують вірогідність помилок, утрудняють модифікацію.

Контрольні питання

- 1 Назвіть типи циклів.
- 2 Що називається тілом циклу?
- 3 Що називається кроком циклу?
- 4 Який оператор відповідає регулярному циклу?
- 5 Наведіть загальний вигляд оператора циклу з постумовою.
- 6 Який оператор відповідає циклу передумовою?
- 7 Наведіть загальний вигляд оператора регулярного циклу.
- 8 Який оператор відповідає циклу з постумовою?
- 9 Наведіть загальний вигляд оператора циклу з передумовою.
- 10 Які мінімальну кількість разів виконується цикл while?
- 11 Які мінімальну кількість разів виконується цикл do-while?
- 12 Охарактеризуйте призначення оператора break.
- 13 Охарактеризуйте призначення оператора continue?
- 14 Чи може в операторі регулярного циклу for бути відсутній будь-який з трьох виразів в заголовку?
- 15 Якщо мається декілька вкладених операторів циклів, то на який з них будуть мати вплив оператори break та continue?

Домашнє завдання

[1] с. 82-87, 91-97

Лекція № 5

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Масив. Вирішення завдань з використанням масивів.

Мета заняття Ознайомити студентів із поняттям масиву, описанням масиву у програмному коді, як представляються масиви у пам'яті комп'ютера; вивчити алгоритми знаходження екстремумів та сортування одномірних масивів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] /

Навчальні матеріали лекції

1 Одномірні масиви. Опис масивів, уявлення у пам'яті

Більшістю об'єктів мови C/C++, з якими ми мали справу, були змінні. Кожна змінна при оголошенні отримувала тип і ім'я, з яким зв'язувався певний елемент пам'яті. Проте розташування значень змінних по адресах пам'яті ніяк не упорядковувалося. При вирішенні багатьох завдань, особливо з великою кількістю однотипних даних, використання змінних з різними іменами, а значить не впорядкованих по адресах пам'яті, утрудняє або робить взагалі неможливим програмування. У подібних випадках в мові C/C++ використовують об'єкти, які називаються масивами.

Масив – це кінцева впорядкована послідовність однотипних даних. Кожному елементу масиву відводиться одна комірка пам'яті. Елементи одного масиву займають послідовно розташовані комірки пам'яті.

Всі елементи мають одне ім'я – ім'я масиву і відрізняються індексами – порядковими номерами в масиві. Кількість елементів в масиві називається його розміром.

Щоб відвести в пам'яті потрібну кількість комірок для розміщення масиву, треба заздалегідь знати його розмір. Резервування пам'яті для масиву виконується на етапі компіляції програми.

Загальна форма оголошення масиву

тип ім'я[розмір_масиву];

де *розмір_масиву* – значення цілого типу, що відповідає кількості елементів масиву, під які необхідно відвести пам'ять.

Елементи масиву завжди нумеруються з 0.

--	--	--	--	--

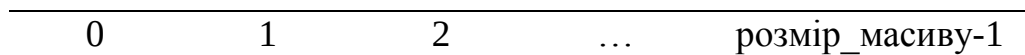


Рисунок 1.1 – Розташування елементів масиву у пам'яті

Наприклад.

```
int data[10]; //масив з 10 елементів цілого типу
```

Тут масив містить 10 елементів типу int: data[0], data[1], data[2],...,data[9].

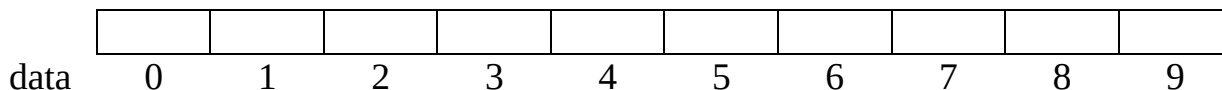


Рисунок 1.2 – Розташування елементів масиву data[10]у пам'яті

Щоб звернутися до елементу масиву, треба вказати ім'я масиву і номер елементу в масиві (індекс):

- a[0] – індекс задається як константа;
- a[8] – індекс задається як константа;
- a[i] – індекс задається як змінна;
- a[2*i] – індекс задається як вираз.

Ініціалізація масиву.

Існує два способи ініціалізації масиву:

- 1 Вказівка початкових значень при оголошенні масиву.
- 2 Організація циклу послідовного введення значень елементів масиву.

Загальна форма ініціалізації масиву при оголошенні

```
тип ім'я[n]={значення_0
значення_1
значення_2
...
значення_n-1};
```

Початкові значення елементів масиву включають у фігурні дужки. Якщо програміст не вказав в квадратних дужках розмір масиву, то компілятор сам задасть розмір по кількості приведених значень. Якщо початкові значення у фігурних дужках не задані, то всі елементи масиву будуть нульовими.

Наприклад.

```
int a[10]={1,2,3,4,5,6,7,8,9,10}; //ініціалізація 10 елементів
int a[10]={1,2,3,4,5}; //5 з 10 елементів задані, решта нулі
int a[]={1,2,3,4,5}; //створення масиву на основі 5-ти елементів
```

Організація циклу послідовного введення значень елементів масиву.

Цикл послідовного введення значень елементів масиву зручно організувати за допомогою оператора for.

Наприклад.

```
#include <stdio.h>
int main()
{
    int data[5];
    int j;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    printf("\n\n Введений масив:");
    for (j=0; j<=4; j++)
    {
        printf("\n Data[%i]=%i",j,data[j]);
    }
    return 0;
}
```

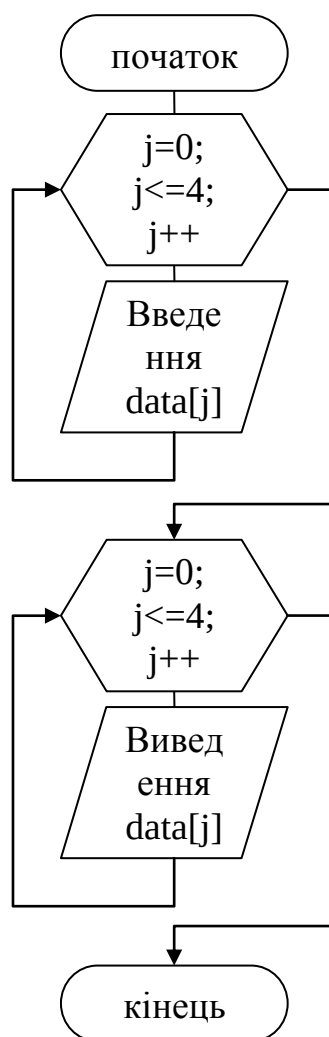


Рисунок 1.3 – Структурна схема лістингу програмного коду

Формування псевдодинамічних масивів.

При описі масиву в програмі треба обов'язково вказувати кількість елементів масиву для того, щоб компілятор виділив під цей масив потрібну кількість пам'яті. Це не завжди буває зручно, оскільки число елементів в масиві може змінюватися залежно від вирішуваного завдання. Динамічні масиви реалізуються за допомогою покажчиків.

Псевдодинамічні масиви реалізуються таким чином:

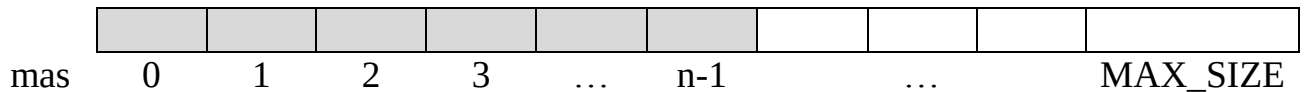
1) при визначенні масиву виділяється достатньо велика кількість пам'яті:

```
const int MAX_SIZE=100; //іменована константа
int mas[MAX_SIZE];
```

2) користувач вводить реальну кількість елементів масиву менше MAX_SIZE.

```
int n;
cout<<"\n Введіть розмір масиву меншу за "<<MAX_SIZE<<": ";
cin>>n;
```

3) подальша робота з масивом обмежується заданою користувачем розмірністю n .



2 Алгоритми знаходження екстремумів

Знаходження екстремумів полягає у знаходженні максимального та/або мінімального елемента (ів) у масиві.

Одним з класичних алгоритмів знаходження екстремумів є наступний: спочатку за максимальний (мінімальний) елемент береться будь-який елемент масиву, наприклад, нульовий елемент масиву, потім за допомогою циклу від 1-го до останнього елемента масиву необхідно порівнювати "максимум" ("мінімум") з елементами масиву і більший (менший) з них необхідно зберігати в "максимумі" ("мінімумі").

```
#include <stdio.h>
int main()
{
    int data[5];
    int j,max,min;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    max=data[0];
    min=data[0];
    for (j=0; j<=4; j++)
    {
        if (max<data[j]) max=data[j];
        if (min>data[j]) min=data[j];
    }
    printf("\n Max=%i \n Min=%i", max, min);
    return 0;
}
```

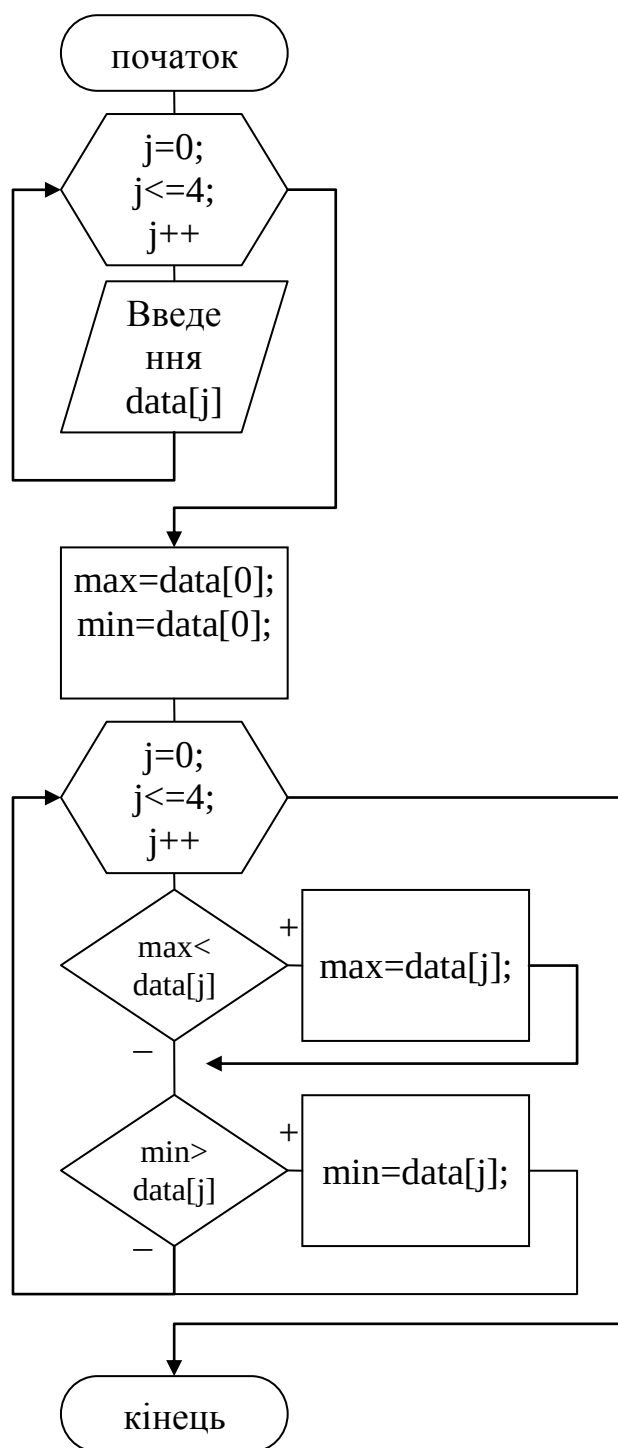


Рисунок 2.1 – Структурна схема лістингу програмного коду

3 Алгоритми сортування лінійних масивів

Сортування масивів полягає у розстановці елементів масиву у певному порядку – за зростанням або за спаданням. В залежності від способу порівняння і обміну елементів розрізняють різні типи алгоритмів сортування.

Основними алгоритмами сортування є:

1 Сортування бульбашкою (обміном).

Ідея методу: крок сортування полягає в проході від низу до верху по масиву. По дорозі порівнюються пари сусідніх елементів. Якщо елементи деякої пари знаходяться в неправильному порядку, то вони міняються місцями.

Наприклад.

```
#include <stdio.h>
int main()
{
    const int size = 5;
    int data[size];
    int i, j, tmp;
    for (j=0; j<=size-1; j++)
    {
        printf("\n Введіть елемент масиву data[%i]", j);
        scanf("%i", &data[j]);
    }
    for(i=0; i<size-1; i++) // i - номер проходу
    {
        for(j=0; j<size-1; j++) // внутрішній цикл проходу
        {
            if (data[j+1]< data[j])
            {
                tmp=data[j+1];
                data[j+1]=data[j];
                data[j]=tmp;
            }
        }
    }
    printf("\n\n Відсортований масив:");
    for (j=0; j<=size-1; j++)
    {
        printf(" %i ", data[j]);
    }
    return 0;
}
```

На практиці метод бульбашки, навіть з поліпшеннями, працює дуже поволі. А тому – майже не застосовується.

2 Сортування вибором.

Ідея методу полягає в тому, щоб створювати відсортовану послідовність шляхом приєднання до неї одного елементу за іншим в правильному порядку. Якщо вхідна послідовність майже впорядкована, то порівнянь буде стільки ж.

Наприклад.

```
#include <stdio.h>
int main()
{
    const int size = 5;
    int data[size];
    int i, j, tmp;
    for (j=0; j<=size-1; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    for(i=0; i<size; i++) // i - номер поточного кроку
    {
        int pos=i;
        tmp=data[i];
        for(j=i+1; j<size; j++) // цикл вибору найменшого елементу
        {
            if (tmp>data[j])
            {
                pos=j;
                tmp=data[j];
            }
        }
        data[pos]= data[i];
        data[i]=tmp; // міняємо місцями найменший з a[i]
    }
    printf("\n\n Відсортований масив:");
    for (j=0; j<=size-1; j++)
    {
        printf(" %i ",data[j]);
    }
    return 0;
}
```

3 Сортування вставками.

Сортування простими вставками в чомусь схоже на вищевикладені методи.

```
#include <stdio.h>
int main()
{
    const int size = 5;
    int data[size];
    int i, j, tmp;
    for (j=0; j<=size-1; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
}
```

```

for (i=1; i<size; i++) // цикл проходів, i - номер проходу
{
    tmp=data[i];
    for (j=i-1; j>=0; j--) // пошук місця елементу в готовій послідовності
        if (data[j]>tmp)
            data[j+1]=data[j]; // зміщуємо елемент направо, поки не дійшли
    data[j+1]=tmp; // місце знайдене, вставити елемент
}
printf("\n\n Відсортований масив:");
for (j=0; j<=size-1; j++)
{
    printf(" %i ",data[j]);
}
return 0;
}

```

Хорошим показником сортування є вельми природна поведінка: майже відсортований масив буде відсортований дуже швидко. Це, укупі із стійкістю алгоритму, робить метод хорошим вибором у відповідних ситуаціях.

Контрольні питання

- 1 Дайте визначення поняття "масив".
- 2 Що називається розміром масиву?
- 3 Як розташовані у пам'яті елементи масиву?
- 4 Що спільного мають всі елементи масиву?
- 5 Чим відрізняються всі елементи масиву?
- 6 Наведіть загальну форму оголошення масиву.
- 7 Що називається індексом елемента масиву?
- 8 З якого номера починається нумерація елементів в масиві?
- 9 Назвіть способи ініціалізації масиву.
- 10 Чим буде заповнений масив в разі ініціалізації масиву при оголошенні, якщо вказати значення не для всіх елементів масиву?
- 11 Опишіть класичний алгоритм знаходження екстремумів у масиві.
- 12 Назвіть основні алгоритми сортування масивів.
- 13 Яка ідея полягає у обмінному алгоритмі сортування (сортування бульбашкою)?

- 14 Чи можна змінювати розмір масиву у пам'яті під час роботи програми?
- 15 Що необхідно вказати, щоб звернутися до елемента масиву?

Домашнє завдання

[1] с. 102-106

Лекція № 6

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Матриця. Вирішення завдань з використанням матриць.

Мета заняття Ознайомити студентів із поняттям масиву, описанням масиву у програмному коді, як представляються масиви у пам'яті комп'ютера; вивчити алгоритми знаходження екстремумів та сортування одномірних масивів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] /

Навчальні матеріали лекції

1 Багатомірні масиви. Матриці

У мові C/C++ масиви бувають не тільки одновимірні, але і багатовимірні. Відмінність полягає в тому, що в одновимірному масиві положення елементу визначається одним індексом, а в багатовимірному – декількома.

Загальна форма оголошення багатовимірного масиву:

```
тип ім'я_масиву[індекс_1][індекс_2]...[індекс_n];
```

де тип – визначає тип даних елементів, з яких буде складатися масив,
ім'я_масиву – ідентифікатор масиву,
індекс_n – кількість елементів масиву у кожній його розмірності.

Елементи багатовимірного масиву розташовуються в послідовних елементах оперативної пам'яті за збільшенням адрес. Між елементами немає розривів. У пам'яті ЕОМ елементи розташовуються рядковим чином так, що найшвидше міняється останній індекс.

Приклад розташування в пам'яті ЕОМ тривимірного масиву `int d[2][2][2]`

```
d[0][0][0] d[0][0][1]  
d[0][1][0] d[0][1][1]  
d[1][0][0] d[1][0][1]  
d[1][1][0] d[1][1][1]
```

Простіший багатовимірний масив – двовірний. Двовірний масив являє собою список одновірних масивів. Для того щоб оголосити двовірний масив необхідно задати лише дві розмірності.

```
тип ім'я_масиву[індекс_1][індекс_2];
```

де тип – визначає тип даних елементів, з яких буде складатися масив,
ім'я_масиву – ідентифікатор масиву,

індекс_1 – кількість рядків матриці,
індекс_2 – кількість стовпців матриці.

Приклад розташування в пам'яті ЕОМ двовірного масиву `int d[3][4]` наведений у таблиці 1.1.

Таблиця 1.1 – Приклад розташування в пам'яті ЕОМ двовірного масиву

стовпці рядки	0	1	2	3
0	<code>d[0][0]</code>	<code>d[0][1]</code>	<code>d[0][2]</code>	<code>d[0][3]</code>
1	<code>d[1][0]</code>	<code>d[1][1]</code>	<code>d[1][2]</code>	<code>d[1][3]</code>
2	<code>d[2][0]</code>	<code>d[2][1]</code>	<code>d[2][2]</code>	<code>d[2][3]</code>

Наприклад, для оголошення двовірного масиву цілих чисел розмірністю 10 рядків та 20 стовпців, необхідно написати:

```
int twod [10][20];
```

В двовірному масиву позиція кожного елемента визначається двома індексами. Якщо уявити двовірний масив у вигляді таблиці даних, то один індекс відповідає рядку, а інший – стовпцю. З цього слідує, що, якщо доступ до елементів масиву уявити в порядку, в якому вони реально зберігаються у пам'яті, то правий індекс буде змінюватись швидше, ніж лівий.

Для того щоб отримати доступ до елемента матриці необхідно вказати ім'я масиву та два індекси – індекс рядка та індекс стовпця. Наприклад, для доступу до елемента масиву `twod` з координатами 3,5, необхідно записати:

```
twod[3][5];
```

Необхідно пам'ятати, що місце зберігання всіх елементів масиву визначається під час компіляції. Окрім того, пам'ять, виділена для зберігання масиву, використовується в продовж всього терміну існування масиву. Для визначення кількості байт пам'яті, яку займає двовірний масив, необхідно використовувати наступну формулу:

число байт = кількість рядків x кількість стовпців x розмір типу в байтах

Відповідно, цілочисельний масив розмірністю 10×5 займає в пам'яті $10 \times 5 \times 2 = 100$ байт (якщо цілочисельний тип має розмір 2 байт).

При ініціалізації багатовимірного масиву список ініціалізаторів кожної розмірності (підгрупу ініціалізаторів) можна заключити у фігурні дужки. Наприклад, для ініціалізації масиву `mas[3][5]` можна використати вираз:

```
int mas[3][5]={
    {1, 5, -5, 0, 7},
    {0, -5, 100, 20, 19},
    {0, -2, -3, -4, 10}
};
```

При використанні підгруп ініціалізаторів члени підгруп, яких не вистачить, будуть ініціалізовані нульовими значеннями автоматично.

Застосування двовірних масивів. Двовимірні масиви, як правило, ототожнюються з матрицями, тому типовими завданнями, в яких застосовуються двовірні масиви, є завдання з матрицями: транспонування матриць, обчислення визначника, визначення середніх значень матриці і так далі.

2 Використання конструкцій мови C/C++ для обробки матриць

Для обробки матриць, зазвичай, використовують два регулярних цикли – зовнішній для переходів по рядкам, внутрішній для переходів по стовпцям.

Приклад коду для введення елементів матриці та виведення їх у загальному прийнятому вигляді наведений у лістингу 2.1.

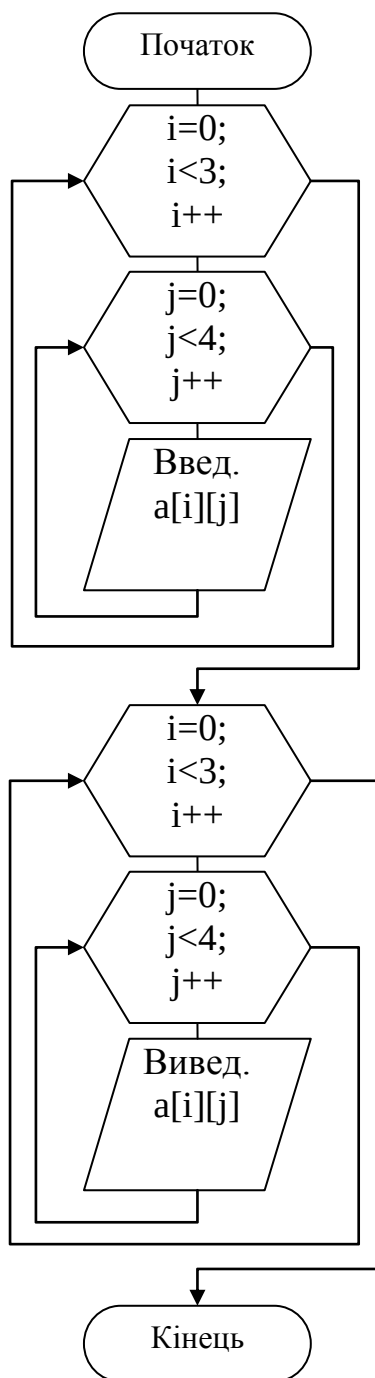
Лістинг 2.1 – Приклад коду введення елементів матриці та виведення їх на екран

```
#include<iostream.h>
#include <conio.h>
int main()
{
```



```
int a[3][4];
int i,j;
for(i=0;i<3;i++)
{
    for(j=0;j<4;j++)
    {
        cout<<"\n Введіть A["<<i+1<<","<<j+1<<"]: ";
        cin>>a[i][j];
    }
}
system("cls");
for(i=0;i<3;i++)
{
    for(j=0;j<4;j++)
    {
        cout.width(3); //встановлення ширини виводу 3 позиції
        cout<<a[i][j];
    }
    cout<<endl;
}

getch();
return 0;
}
```



Приклад.

Дано матрицю A[3][4]. Обчислити добуток від'ємних парних елементів у кожному з рядків.

Лістинг 2.1 – Приклад коду програми для обчислення добутку від'ємних парних елементів у кожному з рядків матриці

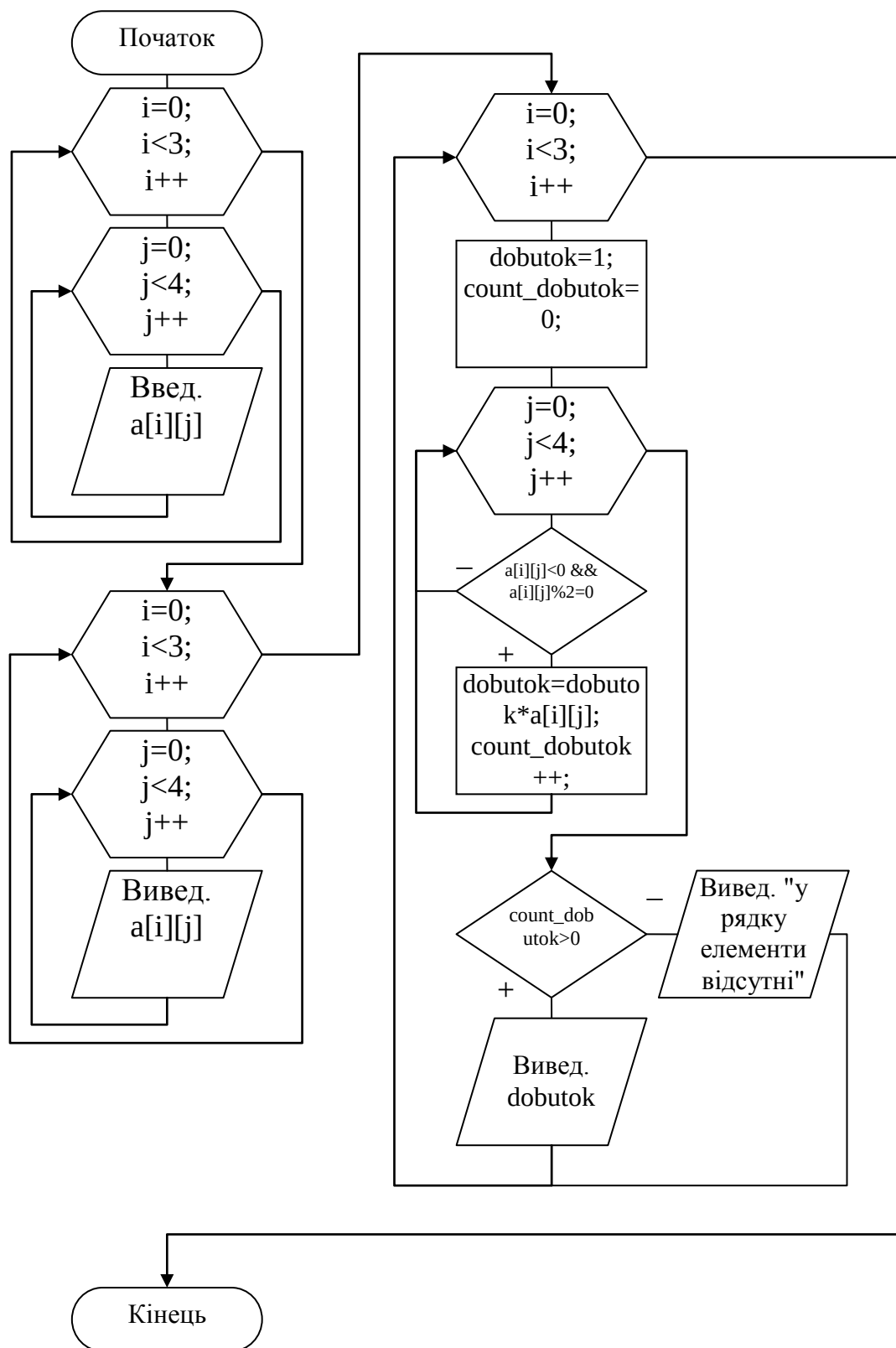
```
#include <iostream.h>
#include <conio.h>

int main()
{
    int a[3][4];
    int dobutok=1;
    int count_dobutok=0;
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<<endl<<"Введіть A["<<i+1<<","<<j+1<<"] : ";
            cin>>a[i][j];
        }
    }
    system("cls");
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout.width(3); //встановлення ширини виводу 3 позиції
            cout<<a[i][j];
        }
        cout<<endl;
    }
    for(i=0;i<3;i++)
    {
        dobutok=1;
        count_dobutok=0;
        for(j=0;j<4;j++)
        {
            if (a[i][j]<0 && a[i][j]%2==0)
            {
                dobutok=dobutok*a[i][j];
                count_dobutok++;
            }
        }
        if (count_dobutok>0)
            cout<<endl<<"В "<<i+1<<"-у рядку добуток від'ємних парних= "<<dobutok;
        else
            cout<<endl<<"В "<<i+1<<"-у рядку від'ємні парні відсутні";
    }
}
```

```

    getch();
    return 0;
}

```



Контрольні питання

- 1 Що називається багатовимірним масивом?
- 2 Наведіть загальну форму оголошення багатовимірного масиву.
- 3 З якого числа починається нумерація індексів багатовимірного масиву?
- 4 Як розташовується в пам'яті багатовимірний масив?
- 5 Наведіть загальну форму оголошення двовимірного масиву.
- 6 Наведіть приклад ініціалізації багатовимірного масиву.
- 7 Скільки і яку назву мають індекси, необхідні для звернення до елемента матриці?
- 8 Як визначити загальний розмір пам'яті, який займе матриця?
- 9 Які ініціалізатори використовує компілятор у випадку, коли у підгрупах ініціалізаторів не вистає членів підгруп?
- 10 Які типи циклів і в якому порядку зазвичай використовують для виконання будь-яких дій над матрицями?

Домашнє завдання

[1] с. 114-120

Лекція № 7

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Ітераційні цикли і їх особливості.

Мета заняття Ознайомити студентів із принципами використання операторів while та do-while

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заклучна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] /

Навчальні матеріали лекції

Розрізняють наступні види циклів:

- регулярні цикли;
- ітераційні цикли.

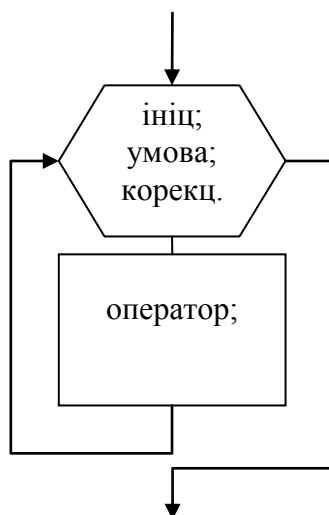
Група дій, що повторюються в циклі, називається його *тілом*. Одноразове виконання циклу називається його *кроком*.

1 Регулярний цикл for

Цикл for – цикл з фіксованим числом повторень, яке заздалегідь відоме.

Загальна форма запису:

```
for(ініціалізація; перевірка_умови; корекція)  
    <оператор>;
```



Для організації такого циклу повинні розглядатися три операції:

- ініціалізація лічильника;
- порівняння його величини з деяким граничним значенням;
- зміна значення лічильника при кожному проходженні тіла циклу.

Ці три операції записуються в дужках і розділяються крапкою з комою (;).

Перший вираз (*ініціалізація*) служить для ініціалізації лічильника. Ініціалізація здійснюється тільки один раз – коли цикл `for` починає виконуватися.

Другий вираз (*перевірка_умови*) служить для перевірки умови перед кожним можливим виконанням тіла циклу. Коли вираз стає брехнею (рівним нулю), цикл завершиться.

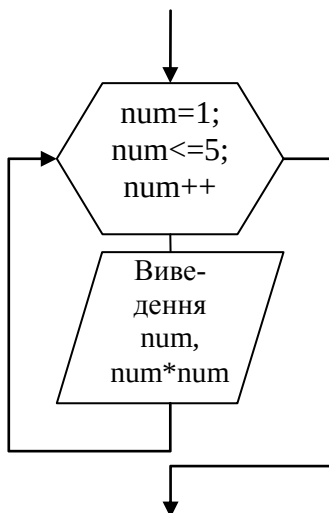
Третій вираз (*корекція*) обчислюється в кінці кожного виконання тіла циклу. Лічильник може як збільшуватися, так і зменшуватися.

Будь-який вираз може бути відсутнім, але символи крапки з комою (;), що їх розділяють, повинні бути обов'язково.

Наприклад.

Вивести на екран монітора числа від 1 до 5 та їх квадрати.

```
int num;
for(num=1; num<=5; num++)
    printf("\n %d * %d = %d", num, num, num*num);
```



Параметри, що знаходяться в заголовку циклу, можна змінити при виконанні операцій в тілі циклу.

Наприклад.

```
int k=2;
for(n=3; k<=25; )
{
    ...
    k = k*n;
```



```
}
```

У циклі `for` часто використовується операція «кома» (,) для розділення декількох виразів. Це дозволяє включити в специфікацію циклу декілька виразів, які будуть ініціалізуватися або коректуватися. Вирази, до яких застосовується операція «кома», обчислюватимуться зліва направо.

Наприклад.

```
for(i=0, j=1; i<n; i++, j=i);
```

У C/C++ допускаються вкладені цикли, тобто коли один цикл знаходиться усередині іншого.

Приклад

```
for( i=0; i<n; i++)
{
    for( j=0; j<n; j++)
    {
        ...
    }
    ...
}
```

Приклади використання регулярного циклу з параметром.

1) Збільшення лічильника;

```
for (n=0; n<10; n++)
    оператор;
```

2) Зменшення лічильника;

```
for (n=10; n>0; n--)
    оператор;
```

3) Довільна зміна кроку коректування;

```
for (n=2; n>60; n+=13)
    оператор;
```

4) Можливість перевіряти умову відмінну від умови, яка накладається на число ітерацій;

```
for (num=1; num*num*num<216; num++)
    оператор;
```

5) Корекція може здійснюватися не тільки за допомогою складання або віднімання.

```
for (d=100.0; d<150.0; d*=1.1)
{
    <тіло циклу>;
}
```

```

for (x=1; y<=75; y=5*(x++)+10)
{
<тіло циклу>;
}

```

2 Ітераційні цикли while та do-while

У ітераційних циклах заздалегідь невідома кількість разів виконання циклу.

1) Цикл while – ітераційний цикл з передумовою.

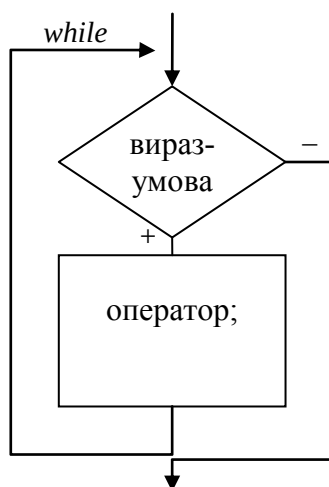
Загальна форма запису:

```

while (вираз-умова)
    оператор;

```

В якості <виразу-умови> найчастіше використовується відношення або логічний вираз. Якщо <вираз-умова> істинне (тобто не рівно нулю), то виконується оператор або група операторів, що входять в цикл while, та потім вираз перевіряється знову.



Послідовність дій, що складається з перевірки і виконання оператора, повторюється до тих пір, поки вираз не стане брехнею (тобто рівним нулю). Після цього відбувається вихід з циклу, і далі виконується оператор, що стоїть після оператора циклу. При побудові циклу while, в нього необхідно включити конструкції, що змінюють величину виразу, що перевіряється, так, щоб врешті-решт воно стало брехнею. Інакше виконання циклу ніколи не завершиться.

Цикл while – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу.

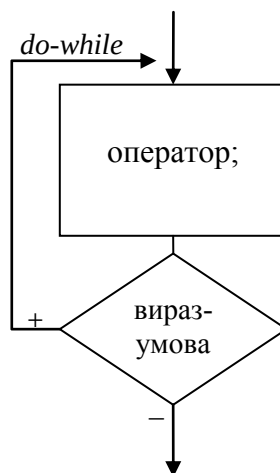
Наприклад.

```
k=5;
n=10;
while(k<n)
{
    printf("k=%d n=%d\n", до, n);
    k+=2;
    k=k+2;
    n++;
}
```

2) Do-while – ітераційний цикл з постумовою.

Загальна форма запису:

```
do
    оператор;
while (вираз-умова);
```



Цикл do-while — це цикл з постумовою, в якому істинність виразу-умови перевіряється після виконання всіх операторів, включених в тіло циклу. Тіло циклу виконується до тих пір, поки вираз не стане брехнею, тобто тіло циклу виконується хоч би один раз. Використовувати цикл краще всього в тих випадках, коли повинна бути виконана хоч би одна ітерація.

Наприклад.

```
do
{
    printf(" Введіть n>0");
    scanf("%d", &n);
}
while (n<0);
```

3 Оператори break і continue

У тілі будь-якого типу циклу можна використовувати оператора break, який дозволяє вийти з циклу, не завершуючи його. Оператор continue дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

1) break – оператор переривання циклу.

```
while (<вираз>)
{
    ...
    break;
    ...
}
```



Тобто оператор break доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

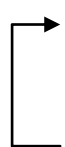
Наприклад.

Написати програму, яка підраховує суму чисел що вводяться з клавіатури до тих пір, поки не буде введено 100 чисел або число 0

```
for (s=0, i=1; i<100; i++)
{
    cin>>x;
    if (x==0) break; // якщо ввели 0, то підсумовування закінчується
    s+=x;
}
```

2) continue – оператор переходу до наступної ітерації циклу. Він використовується, коли тіло циклу містить галуження.

```
while (<вираз>)
{
    ...
    continue;
    ...
}
```



Наприклад.

Написати програму, яка підраховує кількість і суму додатних чисел

```
for ( k=0, s=0, x=1; x!=0; )
```

```

{
    cin>>x;
    if (x<=0) continue; //якщо ввели <=0, то перехід до наступної ітерації
    k++;s+=x;
}

```

При вкладених циклах дії операторів break і continue розповсюджується тільки на саму внутрішню структуру, в якій вони містяться.

Використання цих операторів можливе, але небажано, оскільки вони погіршують читаність програми, збільшують вірогідність помилок, утрудняють модифікацію.

Контрольні питання

- 1 Назвіть типи циклів.
- 2 Що називається тілом циклу?
- 3 Що називається кроком циклу?
- 4 Який оператор відповідає регулярному циклу?
- 5 Наведіть загальний вигляд оператора циклу з постумовою.
- 6 Який оператор відповідає циклу передумовою?
- 7 Наведіть загальний вигляд оператора регулярного циклу.
- 8 Який оператор відповідає циклу з постумовою?
- 9 Наведіть загальний вигляд оператора циклу з передумовою.
- 10 Які мінімальну кількість разів виконується цикл while?
- 11 Які мінімальну кількість разів виконується цикл do-while?
- 12 Охарактеризуйте призначення оператора break.
- 13 Охарактеризуйте призначення оператора continue?
- 14 Чи може в операторі регулярного циклу for бути відсутній будь-який з трьох виразів в заголовку?
- 15 Якщо мається декілька вкладених операторів циклів, то на який з них будуть мати вплив оператори break та continue?

Домашнє завдання

[1] с. 82-87, 91-97

Лекція № 8

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Абетка мови C++, ідентифікатори, структура програми. Етапи обробки програмб. Опис змінних. Символьний та логічний тип даних. Приведення типів. Коментарі, константи.

Мета заняття Ознайомити студентів із поняттям ідентифікатора та зарезервованих слів, правилами побудови та використання констант цілого, плаваючого, символьного та рядкового типів, управляючих кодів, засвоїти правила їх побудови та використання

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. C++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Склад мови C/C++

У тексті на будь-якій природній мові можна виділити чотири основні елементи: символи, слова, словосполучення та речення. Алгоритмічна мова також містить такі елементи, тільки слова називають лексемами (елементарними конструкціями), словосполучення – виразами, речення – операторами. Лексеми утворюються з символів, вирази з лексем і символів, оператори з символів виразів і лексем.

Таким чином, елементами алгоритмічної мови є:

1) *Алфавіт*, який для мови C/C++ включає:

- прописні і рядкові латинські букви і знак підкреслення;
- арабські цифри від 0 до 9;
- спеціальні знаки " { , | [] () + - / % * . \ : ; & ? < > = ! # ^ ;
- пробільні символи (пропуск, символ табуляції, символи переходу на новий рядок).

2) *Лексеми*. З символів формуються лексеми мови:

- ідентифікатори – імена об'єктів C/C++-програм. У ідентифікаторі можуть бути використані латинські букви, цифри і знак підкреслення, довжиною не більше 32 символів (проте значущі тільки перші 8 символів). Прописні і рядкові букви розрізняються, наприклад, PROG1, prog1 і Prog1 – три різні ідентифікатори. Першим символом повинні бути буква або знак підкреслення (але не цифра). Пропуски в ідентифікаторах не допускаються;

- ключові (зарезервовані) слова – це слова, які мають спеціальне значення для компілятора. Їх не можна використовувати як ідентифікатори;
- знаки операцій – це один або декілька символів, що визначають дію над операндами. Операції діляться на унарні, бінарні і тернарні по кількості операндів, що беруть участь в цій операції, відповідно, один операнд, 2 операнди, 3 операнди;
- константи – це незмінні величини. Існують цілі, дійсні, символьні і рядкові константи. Компілятор виділяє константу як лексему (елементарну конструкцію) і відносить її до одного з типів по її зовнішньому вигляду;
- роздільники – дужки, крапка, кома, пробільні символи.

2 Константи в C/C++

Константа – це лексема, що представляє собою фіксоване числове, рядкове або символьне значення, яке не підлягає зміні.

Константи діляться на 5 груп:

- цілі;
- дійсні (з плаваючою крапкою);
- символьні;
- рядкові;
- перерахування (enum).

Компілятор виділяє лексему і відносить її до тієї або іншої групи, а потім усередині групи до певного типу по її формі запису в тексті програми і по числовому значенню.

Цілі константи можуть бути:

- десятковими – константа визначається як послідовність десяткових цифр, що починається не з 0, якщо це число не 0 (наприклад, 8, 0, 192345);
- вісімкові – це константа, яка завжди починається з 0. За 0 слідує вісімкові цифри (наприклад, 016 – десяткове значення 14, 01 – десяткове значення 1);
- шістнадцятирічними – послідовність шістнадцятирічних цифр, яким передують символи 0x або 0X (наприклад, 0xA – десяткове значення 10, 0X00F – десяткове значення 15).

Залежно від значення цілої константи компілятор по-різному представить її в пам'яті комп'ютера (тобто компілятор припише константі відповідний тип даних).

Дійсні константи мають іншу форму внутрішнього уявлення в пам'яті комп'ютера. Компілятор розпізнає такі константи по їх вигляду.

Дійсні константи складаються з наступних частин:

- ціла частина – послідовність цифр;
- десяткова крапка;
- дробова частина – послідовність цифр;
- символу експоненти e або E;
- експоненти у вигляді цілої константи (можливо зі знаком);

Будь-яка (але не обидві відразу) з нижченаведених можуть бути опущена:

- ціла або дробова частина;
- десяткова крапка або символ e(E) і експонента у вигляді цілої константи.

Дійсні константи можуть мати дві форми уявлення:

- з фіксованою крапкою. Вид константи з фіксованою крапкою: [цифри].[цифри] (наприклад, 5.7, .0001, 41.).
- з плаваючою крапкою. Вид константи з плаваючою крапкою: [цифри][.][цифри]E[e[+|-]][цифри] (наприклад, 0.5e5, .11e-5, 5E3).

Символьні константи – це один символ або зворотній слеш і один символ, включені в апострофи (одинарні лапки), наприклад, 'z', 'a'. Послідовності, що починаються із знаку \, називаються керуючими послідовностями і використовуються:

- для представлення символів, що не мають графічного відображення;

Таблиця 2.1 – Значення керуючих послідовностей

№ з/п	Код	Значення
1.	\b	Повернення на одну позицію
2.	\f	Подача сторінки (для переходу до початку наступної сторінки)
3.	\n	Новий рядок
4.	\r	Повернення каретки
5.	\t	Горизонтальна табуляція
6.	\v	Вертикальна табуляція
7.	\a	Звуковий сигнал (дзвінок)
8.	\N	Восьмирічна константа (де N – сама восьмирічна константа)
9.	\xN	Шістнадцятирічна константа (де N – сама шістнадцятирічна константа)
10.	\0	Кінець рядка

- для представлення символів: \, ', ?, " (\, \', \?, \").

Всі символьні константи займають в пам'яті один байт. Значенням символьної константи є числове значення її внутрішнього коду.

Рядкова константа – це послідовність символів, включена в лапки. У середині рядків також можуть використовуватися керуючі символи. Наприклад,

Лістинг 2.1 – Приклад рядкових констант

```
"\n Новий рядок".
```

```
"\n \"Алгоритмічні мови програмування високого рівня\" \"".
```

Лапки не входять до рядка, а лише обмежують його. Технічно рядкова константа являє собою масив символів і за цією ознакою може бути віднесена до складу складних об'єктів мови C/C++. Проте, рядкову константу зручніше розглядати разом з іншими константами.

В кінці кожної рядкової константи компілятор вміщує символ '\0', щоб програма могла визначити кінець рядка. Таке уявлення означає, що розмір

рядкової константи не обмежений яким-небудь розміром, але для визначення довжини рядкової константи її необхідно повністю переглянути.

Оскільки рядкова константа складається з символів, то її розмір дорівнює кількості явних символів у ній та плюс один керуючий символ ('\0') для завершення рядкової константи. Варто чітко розуміти, що символ константа і рядок з одного символу не одне й те саме: 'x' не є "x". Перше – символ, використаний для числового представлення букви 'x', а друге – рядкова константа, яка включає символ 'x' та '\0'. Якщо в програмі рядкові константи записані одна за одною через роздільники, то при виконанні програми вони будуть "склеєні".

Перерахування – константи, що вводяться за допомогою ключового слова `enum`. Це звичайні цілі константи, яким приписані унікальні і зручні для використання позначення. Наприклад,

Лістинг 2.2 – Приклад констант `enum`

```
enum {one=1,two=2,three=3, four=4};
enum {zero,one,two,three}
```

Якщо у визначенні перерахування опустити знаки `=` і числові значення, то значення приписуватимуться за замовчуванням. При цьому крайній лівий ідентифікатор отримає значення 0, а кожен подальший збільшуватиметься на 1.

Лістинг 2.3 – Приклад констант `enum` з різними фіксованими значеннями

```
enum {ten=10, three=3, four, five, six};
enum {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday};
```

3 Коментарі

Коментар – спеціальна частина програмного коду, яка не впливає на виконання програмного коду і не розглядається компілятором при трансляції програми.

Коментар дозволяється скрізь, де припустимі пробіли, переклади на новий рядок і переклад сторінки, коментар використовуються як роздільник та пояснювальний текст до програмного коду.

Для підвищення читабельності програми рекомендується використовувати символ табуляції.

Коментар бувають двох типів:

- однорядковий;
`// це однорядковий коментар`
- багаторядковий.
`/* це
багаторядковий
коментар
*/`

Коментарі не вкладаються друг у друга.

Контрольні питання

- 1 Що включає в себе алфавіт мови C/C++?
- 2 Які базові лексеми включає в себе мова C/C++?
- 3 Що таке ідентифікатор і з чого він може складатися?
- 4 Чи вірне оголошення ідентифікатора `ab`?
- 5 Дайте визначення поняття константа?
- 6 На які основні групи можливо розділити константи в мові C/C++?
- 7 Приведіть приклад цілої десяткової константи.
- 8 Чи вірне оголошення ідентифікатора `_7`?
- 9 З яких частин складається дійсна константа?
- 10 Приведіть приклад дійсної константи з фіксованою крапкою.
- 11 Приведіть приклад дійсної константи з плаваючою крапкою.
- 12 Що являє собою символьна константа?
- 13 Який об'єм займає в пам'яті символьна константа (у байтах)?
- 14 Як називаються послідовності символів, які починаються із знаку `\`?
- 15 Чи вірне оголошення ідентифікатора `7a`?

- 16 Яким символом закінчується рядкова константа у мові C/C++?
- 17 Чи однакове значення мають константи 'x' і "x" для мови C/C++?
- 18 Чи однакові ідентифікатори XB1, Xb1 та xb1 для мови C/C++?
- 19 Наведіть приклад однорядкового коментарю для мови C/C++.
- 20 Наведіть приклад багаторядкового коментарю для мови C/C++.

Домашнє завдання

[1] с. 63-66

Лекція № 9

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Операції, їх знаки, вирази. Тернарна операція. Оператори cout, cin, маніпулятори. Розбір найпростішої програми.

Мета заняття Ознайомити студентів із різними типами операцій, які можуть застосовуватись у виразах, пріоритетом операцій та процесами перетворення типів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.:

Біном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Знаки операцій в C/C++

Знаки операцій забезпечують формування виразів. Вирази складаються з операндів, знаків операцій і роздільників (дужок). Кожен операнд є, у свою чергу, виразом або окремим випадком виразу – константою або змінною.

За кількістю операндів, які приймають участь в операції вирази діляться:

- унарні – операція з одним операндом;
- бінарні – операції з двома операндами;
- тернарні – операції з трьома операндами.

По типу виконуваної операції вирази діляться на:

- арифметичні;
- операції зсуву;
- порозрядні логічні;
- логічні;
- операції відношення;
- операції привласнення.

Унарні операції наведені у таблиці 1.1.

Таблиця 1.1 – Знаки унарних операцій

Знак операції	Призначення
&	Отримання адреси операнду
*	Звертання за адресою (розіменування)
-	Унарний мінус, змінює знак арифметичного операнду
!	Логічне заперечення (НЕ). В якості логічних значень використовується 0 – брехня і не 0 – істина, запереченням 0 буде 1, запереченням будь-якого ненульового числа буде 0
++	Збільшення на одиницю: – префіксна операція – збільшує на одиницю операнд до його

Знак операції	Призначення
	використання; – постфіксна операція – збільшує на одиницю операнд після його використання. <code>int m=1, n=2;</code> <code>int a=(m++)+n; //a=3, m=1, n=2</code> –постфіксна <code>int a=(++m)+n; //a=4, m=2, n=2</code> –префіксна
--	Зменшення на одиницю: – префіксна операція – зменшує на одиницю операнд до його використання; – постфіксна операція – зменшує на одиницю операнд після його використання.
sizeof	Обчислення розміру (в байтах) для об'єкту того типу, який має операнд. Має 2 форми: – sizeof вираз; – sizeof (тип). Приклади: – sizeof (float) //4 – sizeof (1.0) /*8, оскільки дійсні константи за замовчуванням мають тип double*/

Бінарні операції.

Арифметичні операції наведені у таблицях 1.2 та 1.3.

Таблиця 1.2 – Знаки бінарних арифметичних адитивних операцій

Знак операції	Призначення
+	Бінарний плюс (складання арифметичних операндів)
-	Бінарний мінус (знаходження різниці арифметичних операндів)

Таблиця 1.3 – Знаки бінарних арифметичних мультиплікативних операцій

Знак операції	Призначення
*	Знаходження добутку операндів арифметичного типу
/	Ділення операндів арифметичного типу (якщо операнди цілі, виконується цілочислене ділення з відкиданням дробової частини). Наприклад, <code>10.4 / 2 = 5.2</code> //операція над дійсним числом. <code>5 / 2 = 2</code> //операція над цілим числом, дробова частина (0,5) відкинута. <code>2 / 5 = 0</code> //операція над цілим числом, дробова частина (0,4) відкинута.
%	Отримання залишку від ділення цілочислених операндів. <i>Формат x%y</i> Результат дорівнює залишку від ділення x на y, і відповідно 0, якщо x ділиться на y без залишку. Наприклад, <code>8%3 = 2</code> <code>4%2 = 0</code> <code>5 % 2 = 1</code> <code>3%8 = 3</code>

Знак операції	Призначення
	<i>Окремий випадок: $x\%2$ – перевірка на парність/непарність числа x. Якщо залишок дорівнює 0, число парне, якщо не 0 – непарне.</i>

В C/C++ мається є 6 операцій для роботи з окремими бітами у внутрішньому двійковому уявленні числа. Їх можна здійснювати тільки над цілими операндами (char, int).

До таких операцій відносяться:

- операції зсуву (таблиця 1.4). Формат виразу з операцією зсуву:

операнд_лівий операція_зсуву операнд_правий

Таблиця 1.4 – Знаки операцій зсуву

Знак операції	Призначення
<<	Зсув вліво бітового уявлення значення лівого цілочисленого операнду на кількість розрядів, яке дорівнює кількості розрядів правого операнду. Зсув вліво є найбільш швидким способом множення числа на 2, 4, 8 і т.д. Приклад, n=13>>2 0000 0000 0000 1101 //13 15>>2 0000 0000 0000 0011 //2 n= 0000 0000 0000 0011 //3
>>	Зсув вправо бітового уявлення значення лівого цілочисленого операнду на кількість розрядів, яке дорівнює кількості розрядів правого операнду. Розряди, які звільняються обнулюються, якщо операнд беззнакового типу і заповнюються знаковим розрядом, якщо – знакового. Зсув вліво є найбільш швидким способом ділення числа на 2, 4, 8 і т.д. Приклад, n=5<<3 0000 0000 0000 0101 //5 5<<3 0000 0000 0010 1000 //40 n= 0000 0000 0010 1000 //40

- порозрядні логічні операції (таблиця 1.5).

Таблиця 1.5 – Знаки порозрядних логічних операцій

Знак операції	Призначення
&	Порозрядна кон'юнкція (И) бітових уявлень значень цілих операндів (біт=1, якщо відповідні біти обох операндів =1). Часто використовується для обнулення деякої групи розрядів. Приклад, n=15&3

	$n = 0000\ 0000\ 0000\ 1111 // 15$ $\& 0000\ 0000\ 0000\ 0011 // 3$ $n = 0000\ 0000\ 0000\ 0011$
	Порозрядна диз'юнкція (ИЛИ) бітових уявлень значень цілих операндів (біт=1, якщо відповідний біт хоча б одного із операндів =1). Часто використовується для встановлення 1 деякої групи розрядів. Приклад, $n = 3 16$ $n = 0000\ 0000\ 0000\ 0011 // 3$ $0000\ 0000\ 0001\ 0000 // 16$ $n = 0000\ 0000\ 0001\ 0011 // 19$
^	Порозрядна сума за модулем 2 (виключне ИЛИ) бітових уявлень значень цілих операндів (біт=1, якщо відповідний біт ТІЛЬКИ ОДНОГО з операндів =1). Приклад, $n = 5 \wedge 6$ $n = 0000\ 0000\ 0000\ 0101 // 5$ $\wedge 0000\ 0000\ 0000\ 0110 // 6$ $n = 0000\ 0000\ 0001\ 0011 // 3$
~	Порозрядне інвертування внутрішнього двійкового коду цілого операнду (побітове інвертування). Приклад, $n = \sim 65\ 535$ $n = 1111\ 1111\ 1111\ 1110 // 65\ 534$ $\sim$ $n = 0000\ 0000\ 0000\ 0001 // 1$

Бінарні операції порівняння наведені у таблиці 1.6 та 1.7: результатом є true (не 0) або false (0). Приклад застосування логічних операцій наведений у таблиці 1.8.

Формат виразу з логічною операцією:

операнд_лівий логічна_операція операнд_правий

Таблиця 1.6 – Знаки операцій відношення

Знак операції	Призначення
<	Дорівнює true, якщо лівий операнд менше правого операнду
>	Дорівнює true, якщо лівий операнд більше правого операнду
<=	Дорівнює true, якщо лівий операнд менше або дорівнює правому операнду
>=	Дорівнює true, якщо лівий операнд більше або дорівнює правому операнду
==	Дорівнює true, якщо лівий операнд дорівнює правому операнду
!=	Дорівнює true, якщо лівий операнд не дорівнює правому операнду

Таблиця 1.7 – Знаки логічних операцій

Знак операції	Призначення
&&	Кон'юнкція (И) цілих операндів або відношень, цілий результат брехня (0) або істина (не 0). Дає результат 1, якщо обидва операнди дорівнюють 1.
	Диз'юнкція (ИЛИ) цілих операндів або відношень, цілий результат брехня (0) або істина (не 0). Дає результат 1, якщо хоча б один із операндів дорівнює 1.

Таблиця 1.8 – Приклад застосування логічних операцій

p	q	p && q	p q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Операції привласнення.

Формат операції простого привласнення:

операнд1=операнд2

Особливості операції привласнення:

- окрім пересилення значення вона має також і значення самого результату привласнення, тому можливо в записі одного речення мови використовувати декілька операцій привласнення. Наприклад,

a=b=c=d=4; //a=4, b=4, c=4, d=4;

- допускаються комбіновані операції привласнення, такі як:

+=, -=, *=, /=, %=, &=, ^=, |=, ~=, >>=, <<=

Лівоприпустиме значення (L-значення) – вираз, який адресує деяку ділянку пам'яті, тобто в нього можна занести значення. Ця назва пішла від операції привласнення, оскільки саме ліва частина операції привласнення визначає, в яку область пам'яті буде занесений результат операції. Змінна – це окремий випадок ліводопустимого виразу.

Тернарні операції.

Умовна операція – на відміну від унарних і бінарних операцій в ній використовується три операнди і є єдиною тернарною операцією.

Вираз1 ? Вираз2 : Вираз3;

Першим обчислюється значення "виразу1". Якщо воно істинне, то обчислюється значення "виразу2", яке стає результатом. Якщо при обчисленні "виразу1" вийде 0, то як результат береться значення "виразу3".

З двох виразів "вираз2" та "вираз3" обчислюється тільки одно з них і ніколи обидва разом!

Наприклад,

`x < 0 ? -x : x ; // обчислюється абсолютне значення x`

2 Вирази

Під виразом в мові C/C++ розуміється послідовність операндів, знаків операцій і символів роздільників. Кожним виразом є правило обчислення нового значення. Якщо вираз формує ціле або дійсне число, то воно називається арифметичним. Пара арифметичних виразів, об'єднана операцією порівняння, називається відношенням. Якщо відношення має ненульове значення, то воно – істинне, інакше – брехня.

Компілятор дотримується чіткого порядку інтерпретації виразів, який називається правилами передування. Цей порядок (наведений у таблиці 2.1) може бути змінений за допомогою круглих дужок.

Таблиця 2.1 – Пріоритет операцій у виразах

Ранг	Операції	Порядок виконання
1	<code>() [] -> .</code>	→
2	<code>! ~ - ++ -- & * (унарні) (тип) sizeof(тип)</code>	←
3	<code>* / % (мультиплікативні бінарні)</code>	→
4	<code>+ - (адитивні бінарні)</code>	→
5	<code><< >> (порозрядного зсуву)</code>	→
6	<code>< > <= >= (відношення)</code>	→
7	<code>== != (відношення)</code>	→
8	<code>& (порозрядна кон'юнкція "И")</code>	→
9	<code>^ (порозрядна сума за модулем 2 (виключне "ИЛИ"))</code>	→
10	<code> (порозрядна диз'юнкція "ИЛИ")</code>	→
11	<code>&& (кон'юнкція "И")</code>	→
12	<code> (диз'юнкція "ИЛИ")</code>	→
13	<code>?: (тернарна умовна операція)</code>	←
14	<code>= *= /= %= -= &= ^= = <=> >=> (операції привласнення)</code>	←
15	<code>,</code> (операція кома)	→

3 Приведення типів

Якщо у виразі змішані різні типи літералів і змінних, компілятор приводить (перетворює) їх до одного типу.

По-перше, всі `char` та `short int`-значення автоматичну перетворюються (з розширенням "типорозміру") до типу `int`. Цей процес називається *цілочисленим розширенням*.

По-друге, всі операнди перетворюються (також з розширенням "типорозміру") до типу найбільшого операнду. Цей процес називається розширенням типу, причому він виконується поопераційно. Наприклад, якщо один операнд має тип `int`, а інший – `long int`, то тип `int` розширюється в тип `long int`. Або, якщо хоча б один з операндів має тип `double`, будь-який інший операнд приводиться до типу `double`. Це значить, що такі перетворення, як з типу `char` в тип `double`, цілком припустимі. Після перетворення обидва операнди будуть мати один і той самий тип, а результат операції – тип, який співпадає з типом операндів.

Розглянемо, наприклад, перетворення типів, схематично наведене на рисунку 3.1. Спочатку символ `ch` проходить процесу "розширення" типу і перетворюється в значення типу `int`. Потім результат операції `ch/i` приводиться до типу `double`, оскільки результат добутку `f*d` має тип `double`. Результат всього виразу отримає тип `double`, оскільки до моменту його обчислення обидва операнди будуть мати тип `double`.

```
char ch;
int i;
float f;
double d;
```

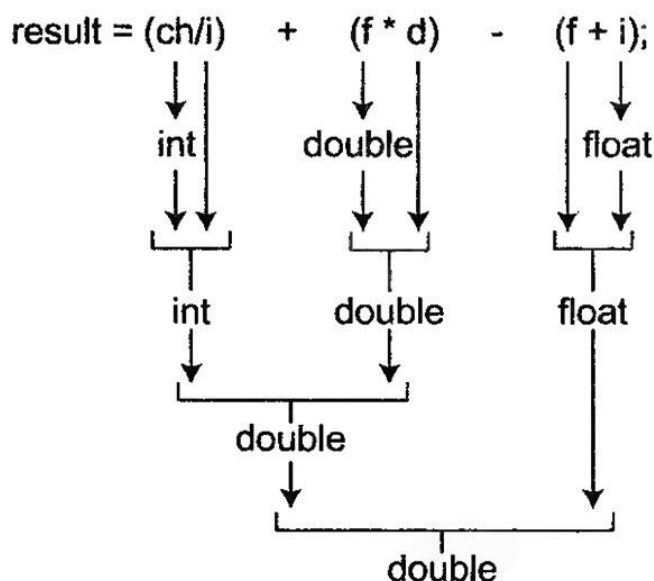


Рисунок 3.1 – Приклад приведення типів у виразах

В C/C++ є можливість встановити для виразу заданий тип. Для цього використовується операція приведення типів.

Існує дві форми операцій приведення типів – канонічна і функціональна:

- 1 (ім'я_типу) операнд.
- 2 ім'я_типу (операнд).

Тут "ім'я типу" означає тип, до якого необхідно привести вираз. Наприклад, якщо ви хочете, щоб вираз `x/2` мав тип `int`, то необхідно код, приведений у лістингу 1.1.

Лістинг 3.1 – Приклад операцій приведення типів

```
(int)x/2 //канонічна форма
int(x/2) //функціональна форма
```

Приведення типів розглядається як унарний оператор, і тому має такий саме пріоритет, як і інші унарні оператори.

Контрольні питання

- 1 Дайте визначення поняттю "вираз"?
- 2 З чого складається вираз?

- 3 Що таке операнд?
- 4 Яка операція називається операцією відношення?
- 5 В якому випадку відношення вважається брехнею, а в якому – істиною?
- 6 Перерахувати знаки операцій, які відносяться до операцій відношення?
- 7 Які операції називаються унарними?
- 8 Приведіть приклади унарних операцій.
- 9 Які операції називаються бінарними?
- 10 Приведіть приклади бінарних операцій.
- 11 Які операції називаються тернарними?
- 12 Приведіть приклади тернарних операцій.
- 13 Які різниця між постфіксною і префіксною операціями інкремента (декремента)?
- 14 Перерахувати операції привласнення, які існують в C/C++?
- 15 Перерахувати знаки операцій, які відносяться до арифметичних операцій?
- 16 Що таке ліводопустиме значення?
- 17 Що називається приведенням типів в C/C++?
- 18 Вкажіть дві форми приведення типів в C/C++.
- 19 Що називається операцією розширення типу?
- 20 Якщо хоча б один з операндів виразу має тип double, до якого типу будуть приведення всі інші операнди?

Домашнє завдання

[1] с. 66-76

Лекція № 10

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Типи розгалужень, оператор if. Операції відношення. Логічні операції. Оператор switch

Мета заняття Ознайомити студентів із різними видами управляючих структур, поняттям порожнього оператора, блоку та складеного оператора, вивчити конструкції if та if-else

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.

З Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Базові конструкції структурного програмування

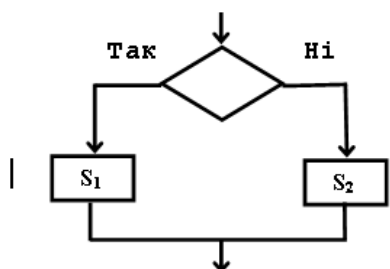
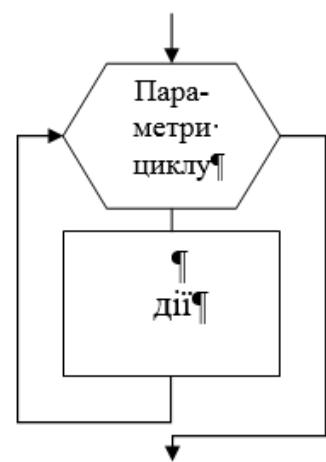
У теорії програмування доведено, що програму для вирішення завдання будь-якої складності можна скласти тільки з трьох структур: лінійної, розгалуженої і циклічної. Ці структури називаються базовими конструкціями структурного програмування.

Лінійною називається конструкція, що є послідовним з'єднанням двох або більшої кількості операторів.

Розгалуження – задає виконання одного з двох операторів, залежно від виконання умови.

Цикл – задає багатократне виконання оператора.

Таблиця 1.1 – Блок-схеми різних типів алгоритмів програмування

Лінійні алгоритм	Розгалужений алгоритм	Циклічний алгоритм
		

Метою використання базових конструкцій є отримання програми простої структури. Таку програму легко читати, відлагоджувати і при необхідності вносити до неї зміни. Структурне програмування також називають програмуванням без goto, оскільки часте використання операторів переходу утрудняє розуміння логіки роботи програми. Але іноді зустрічаються ситуації, в яких застосування операторів переходу, навпаки, спрощує структуру програми.

Оператори управління роботою програми називають управляючими конструкціями. До них відносять:

- складені оператори;
- оператори вибору;
- оператори циклів;
- оператори переходу.

2 Оператор «вираз»

Будь-який вираз, що закінчується крапкою з комою, розглядається як оператор, виконання якого полягає в обчисленні цього виразу. Окремим випадком виразу є порожній оператор ";".

Наприклад:

```
i++;  
a+=2;  
x=a+b;
```

3 Складені оператори

До складених операторів відносять власне складені оператори і блоки. В обох випадках це послідовність операторів, ув'язнена у фігурні дужки { }. Блок відрізняється від складеного оператора наявністю визначень в тілі блоку.

Наприклад:

```
//це складений оператор
{
n++;
summa+=n;
}
//це блок
{
int n=0;
n++;
summa+=n;
}
```

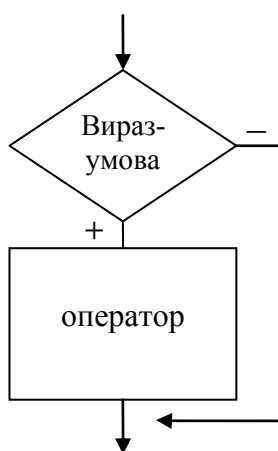
4 Оператор вибору

Одним з операторів вибору є умовний оператор.

Умовний оператор if має повну і скорочену форму.

Скорочена форма:

```
if (вираз-умова)
оператор;
```

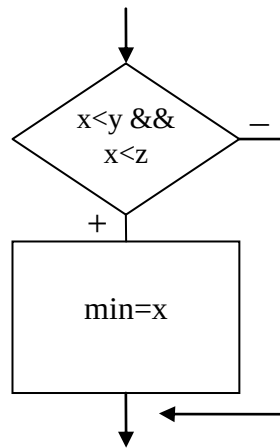


Як вираз-умова можуть використовуватися арифметичний вираз, відношення і логічний вираз. Якщо значення виразу-умови відмінне від нуля (тобто істинно), то виконується "*оператор*".

Якщо "*оператор*" повинен складатися з декількох операторів, то необхідно використовувати складений оператор (блок).

Наприклад:

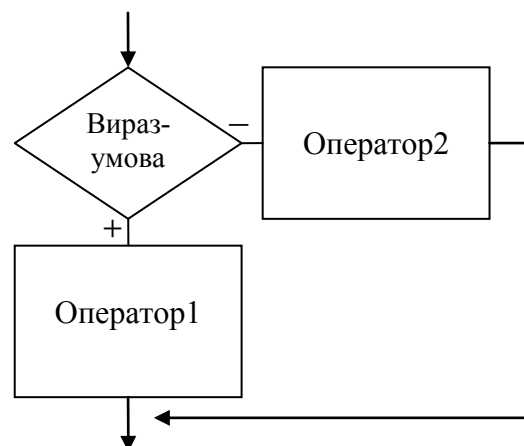
```
if (x<y && x<z)
    min=x;
```



Повна форма

```

if (вираз-умова)
    оператор1;
else
    оператор2;
  
```

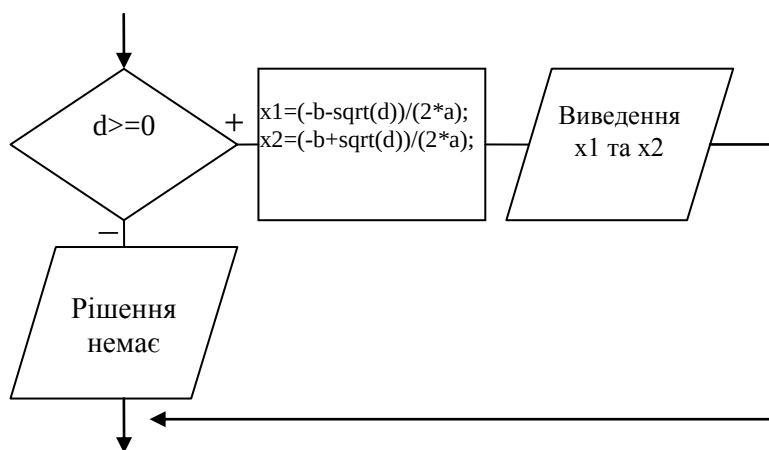


Якщо значення виразу-умови відмінне від нуля, то виконується *оператор1*, при нульовому значенні виразу-умови виконується *оператор2*.

Наприклад:

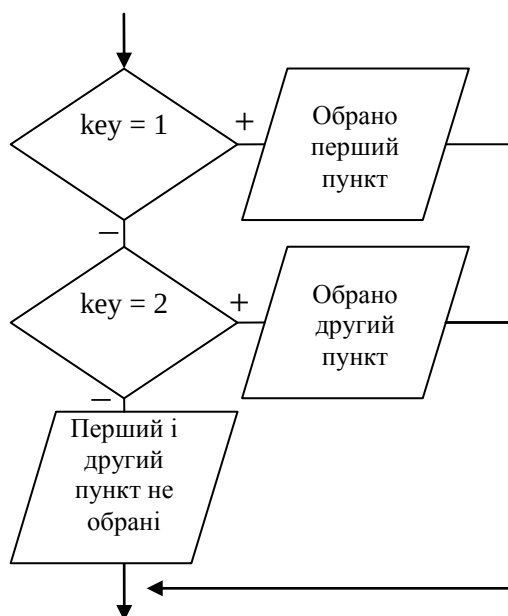
```

if (d>=0)
{
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    cout<< "\nx1="<<x1<<"x2="<<x2;
}
else
    cout<<"\nРешения немає";
  
```



```

if (key == 1)
    printf("\n Обрано перший пункт");
else
    if (key == 2)
        printf("\n Обрано другий пункт");
    else
        printf("\n Перший і другий пункти не обрані");
  
```



Контрольні питання

- 1 Назвіть базові конструкції програмування.
- 2 Що називають управляючими конструкціями?
- 3 Що відноситься до управляючих конструкцій?
- 4 Що називається оператором?

- 5 Виконайте загальну блок-схему лінійного алгоритму.
- 6 Що таке складений оператор?
- 7 Виконайте загальну блок-схема розгалуженого алгоритму.
- 8 Що таке блок?
- 9 Виконайте загальну блок-схему циклічного алгоритму.
- 10 Наведіть повну форму коду умовного оператора if.
- 11 Виконайте загальний вигляд блок-схеми повного умовного оператора.
- 12 Наведіть скорочену форму коду умовного оператора if.
- 13 Виконайте загальний вигляд блок-схеми скороченого умовного оператора.
- 14 Що може включати "вираз-умова" в операторі if?
- 15 Що необхідно задіювати, якщо необхідне виконання не одного оператора, а деякого набору операторів в операторі if?

Домашнє завдання

[1] с. 78-81

Лекція № 11

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Типи циклів. Оператори for, continue, break.

Мета заняття Ознайомити студентів із різновидами циклічних структур, вивчити властивості та принципи використання оператора for, а також засвоїти правила використання управляючих конструкцій break та continue

Час – 2 години.

План проведення лекції

- 1 Регулярний цикл for
- 2 Ітераційні цикли while та do-while
- 3 Оператори break і continue

Навчальні матеріали лекції

Розрізняють наступні види циклів:

- регулярні цикли;
- ітераційні цикли.

Група дій, що повторюються в циклі, називається його *тілом*. Одноразове виконання циклу називається його *кроком*.

1 Регулярний цикл for

Цикл for – цикл з фіксованим числом повторень, яке заздалегідь відоме.

Загальна форма запису:

```
for(ініціалізація; перевірка_умови; корекція)  
<оператор>;
```

Зображення регулярного циклу наведено на рисунку 1.1.

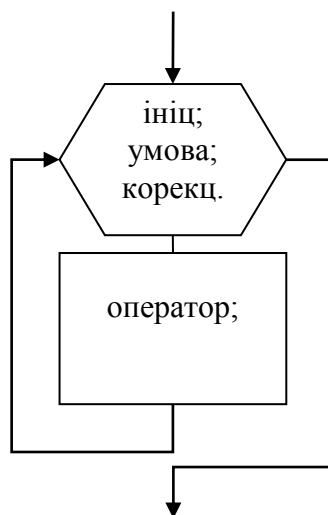


Рисунок 1.1. Регулярний цикл в блок-схемі.

Для організації такого циклу повинні розглядатися три операції:

- ініціалізація лічильника;
- порівняння його величини з деяким граничним значенням;
- зміна значення лічильника при кожному проходженні тіла циклу.

Ці три операції записуються в дужках і розділяються крапкою з комою (;).

Перший вираз (*ініціалізація*) служить для ініціалізації лічильника. Ініціалізація здійснюється тільки один раз – коли цикл `for` починає виконуватися.

Другий вираз (*перевірка умови*) служить для перевірки умови перед кожним можливим виконанням тіла циклу. Коли вираз стає брехнею (рівним нулю), цикл завершиться.

Третій вираз (*корекція*) обчислюється в кінці кожного виконання тіла циклу. Лічильник може як збільшуватися, так і зменшуватися.

Будь-який вираз може бути відсутнім, але символи крапки з комою (;), що їх розділяють, повинні бути обов'язково.

Наприклад.

Вивести на екран монітора числа від 1 до 5 та їх квадрати.

```
int num;
for(num=1; num<=5; num++)
printf("\n %d * %d = %d", num, num, num*num);
```

Блок-схема алгоритму наведеного фрагменту надана на рисунку 2.1.

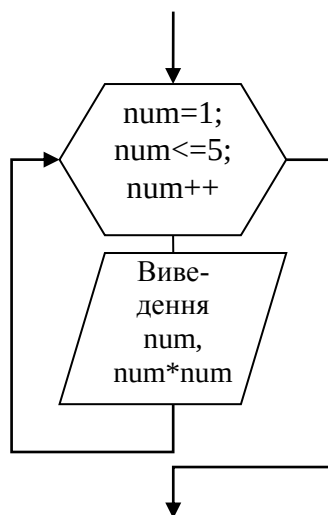


Рисунок 1.2. Блок- схема алгоритму.

Параметри, що знаходяться в заголовку циклу, можна змінити при виконанні операцій в тілі циклу.

Наприклад.

```

int k=2;
for(n=3; k<=25; )
{
  ...
  k = k*n;
}
  
```

У циклі `for` часто використовується операція «кома» (,) для розділення декількох виразів. Це дозволяє включити в специфікацію циклу декілька виразів, які будуть ініціалізуватися або коректуватися. Вирази, до яких застосовується операція «кома», обчислюватимуться зліва направо.

Наприклад.

```

for(i=0, j=1; i<n; i++, j=i);
  
```

У C/C++ допускаються вкладені цикли, тобто коли один цикл знаходиться усередині іншого.

Приклад

```

for( i=0; i<n; i++)
{
  for( j=0; j<n; j++)
  {
    ...
  }
  ...
}
  
```

Приклади використання регулярного циклу з параметром.

1) Збільшення лічильника;

```
for (n=0; n<10; n++)  
оператор;
```

2) Зменшення лічильника;

```
for (n=10; n>0; n--)  
оператор;
```

3) Довільна зміна кроку коректування;

```
for (n=2; n>60; n+=13)  
оператор;
```

4) Можливість перевіряти умову відмінну від умови, яка накладається на число ітерацій;

```
for (num=1; num*num*num<216; num++)  
оператор;
```

5) Корекція може здійснюватися не тільки за допомогою складання або віднімання.

```
for (d=100.0; d<150.0; d*=1.1)  
{  
<тіло циклу>;  
}  
for (x=1; y<=75; y=5*(x++)+10)  
{  
<тіло циклу>;  
}
```

2 Ітераційні цикли while та do-while

У ітераційних циклах заздалегідь невідома кількість разів виконання циклу.

Цикл while – ітераційний цикл з передумовою.

Загальна форма запису:

```
while (вираз-умова)  
оператор;
```

В якості <виразу-умови> найчастіше використовується відношення або логічний вираз. Якщо <вираз-умова> істинне (тобто не рівно нулю), то виконується оператор або група операторів, що входять в цикл while, та потім вираз перевіряється знову. Зображення на блок-схемі циклу з передумовою наведено на рисунку 2.1.

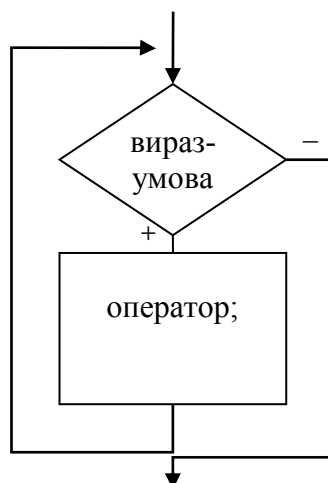


Рисунок 2.1. Зображення на блок-схемі циклу з передумовою

Послідовність дій, що складається з перевірки і виконання оператора, повторюється до тих пір, поки вираз не стане брехнею (тобто рівним нулю). Після цього відбувається вихід з циклу, і далі виконується оператор, що стоїть після оператора циклу. При побудові циклу `while`, в нього необхідно включити конструкції, що змінюють величину виразу, що перевіряється, так, щоб врешті-решт воно стало брехнею. Інакше виконання циклу ніколи не завершиться.

Цикл `while` – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу.

Наприклад.

```

k=5;
n=10;
while (k<n)
{
printf("k=%d n=%d\n", до, n);
k+=2;
k=k+2;
n++;
}
  
```

Do-while – ітераційний цикл з постумовою.

Загальна форма запису:

```

do
оператор;
while (вираз-умова);
  
```

Зображення на блок-схемі циклу з постумовою наведено на рисунку 2.2.

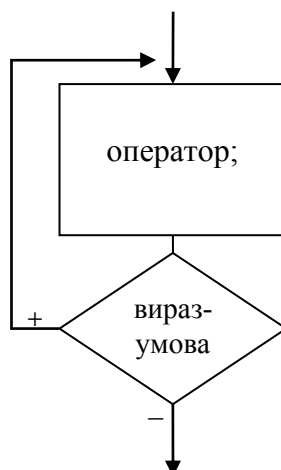


Рисунок 2.2. Зображення на блок-схемі циклу з постумовою

Цикл `do-while` — це цикл з постумовою, в якому істинність виразу-умови перевіряється після виконання всіх операторів, включених в тіло циклу. Тіло циклу виконується до тих пір, поки вираз не стане брехнею, тобто тіло циклу виконується хоч би один раз. Використовувати цикл краще всього в тих випадках, коли повинна бути виконана хоч би одна ітерація.

Наприклад.

```
do
{
printf(" Введіть n>0");
scanf("%d",&n);
}
while (n<0);
```

3 Оператори `break` і `continue`

У тілі будь-якого типу циклу можна використовувати оператора `break`, який дозволяє вийти з циклу, не завершуючи його. Оператор `continue` дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

1) `break` – оператор переривання циклу.

```
while (<вираз>)
{
...
break;
...
}
```

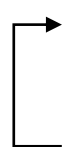
Тобто оператор `break` доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

Наприклад.

Написати програму, яка підраховує суму чисел що вводяться з клавіатури до тих пір, поки не буде введено 100 чисел або число 0

```
for(s=0,i=1; i<100;i++)
{
    cin>>x;
    if(x==0) break; // якщо ввели 0, то підсумовування закінчується
    s+=x;
}
```

2) `continue` – оператор переходу до наступної ітерації циклу. Він використовується, коли тіло циклу містить галуження.



```
while(<вираз>)
{
    ...
    continue;
    ...
}
```

Наприклад.

Написати програму, яка підраховує кількість і суму додатних чисел

```
for( k=0,s=0,x=1;x!=0;)
{
    cin>>x;
    if (x<=0) continue; //якщо ввели <=0, то перехід до наступної ітерації
    k++;s+=x;
}
```

При вкладених циклах дії операторів `break` і `continue` розповсюджується тільки на саму внутрішню структуру, в якій вони містяться.

Використання цих операторів можливе, але небажано, оскільки вони погіршують читаність програми, збільшують вірогідність помилок, утрудняють модифікацію.

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Назвіть типи циклів.
- 2 Що називається тілом циклу?
- 3 Що називається кроком циклу?
- 4 Який оператор відповідає регулярному циклу?
- 5 Наведіть загальний вигляд оператора циклу з постумовою.
- 6 Який оператор відповідає циклу передумовою?
- 7 Наведіть загальний вигляд оператора регулярного циклу.
- 8 Який оператор відповідає циклу з постумовою?
- 9 Наведіть загальний вигляд оператора циклу з передумовою.
- 10 Які мінімальну кількість разів виконується цикл while?
- 11 Які мінімальну кількість разів виконується цикл do-while?
- 12 Охарактеризуйте призначення оператора break.
- 13 Охарактеризуйте призначення оператора continue?
- 14 Чи може в операторі регулярного циклу for бути відсутній будь-який з трьох виразів в заголовку?
- 15 Якщо мається декілька вкладених операторів циклів, то на який з них будуть мати вплив оператори break та continue?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Використовуючи оператор циклу з передумовою while або оператор циклу з постумовою do...while скласти програму рішення задачі:
 - 1.1 Підраховувати суму та добуток непарних чисел серед тих цілих чисел, що вводяться користувачем, поки не буде введено число 0.
 - 1.2 Підраховувати середнє арифметичне непарних чисел серед тих цілих чисел, що вводяться користувачем, поки не буде введено число з діапазону [-4;0].

2 Використовуючи оператор циклу for скласти програму рішення задачі

2.1 Обчислити середнє арифметичне непарних чисел із 13-ти цілих чисел, що вводяться користувачем.

2.1 Порахувати добуток парних чисел з діапазону [-3;16] серед 9-ти цілих чисел, що вводяться користувачем.

Література

1 Шилдт, Герберт. С++: Руководство для начинающих, 2-е издание.: Пер. з англ.: [Текст] / Г. Шилдт. – М.: Издавничий дім "Вильямс", 2005. – 672 с.: іл.

2 Динман, М.И. С++. Освой на примерах. [Текст] / М.И.Динман – СПб.: БХВ-Петербург, 2006 – 384с.: іл.

3 Стефан Р, Дэвис. С++ для "чайников", 4-е издание.: Пер. с англ.: [Текст] / Стефан Р. Дэвис — М. : Издавничий дім "Вильямс", 2003 — 336 с. : іл.

4 Ишкова, Э.А. С++. Начала программирования. Изд. 3-е, перераб. и доп. [Текст] / Э.А. Ишкова. – М.: «Бином – Пресс», 2004 – 368 с.: іл.

5 Шилдт, Герберт. Полный справочник по С++, 4-е издание.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Издавничий дім "Вильямс", 2006. – 800 с.: іл.

Лекція № 12

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Оператори ітераційних циклів. Складання програм з використанням while або do-while

Мета заняття Ознайомити студентів із різновидами циклічних структур, вивчити властивості та принципи використання операторів while та do-while

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.

З Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

Розрізняють наступні види циклів:

- регулярні цикли;
- ітераційні цикли.

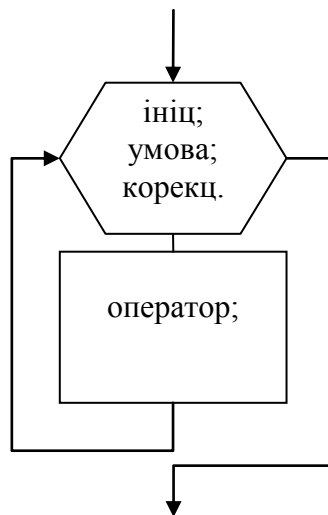
Група дій, що повторюються в циклі, називається його *тілом*. Одноразове виконання циклу називається його *кроком*.

1 Регулярний цикл for

Цикл for – цикл з фіксованим числом повторень, яке заздалегідь відоме.

Загальна форма запису:

```
for(ініціалізація; перевірка_умови; корекція)  
    <оператор>;
```



Для організації такого циклу повинні розглядатися три операції:

- ініціалізація лічильника;
- порівняння його величини з деяким граничним значенням;
- зміна значення лічильника при кожному проходженні тіла циклу.

Ці три операції записуються в дужках і розділяються крапкою з комою (;).

Перший вираз (*ініціалізація*) служить для ініціалізації лічильника. Ініціалізація здійснюється тільки один раз – коли цикл `for` починає виконуватися.

Другий вираз (*перевірка_умови*) служить для перевірки умови перед кожним можливим виконанням тіла циклу. Коли вираз стає брехнею (рівним нулю), цикл завершиться.

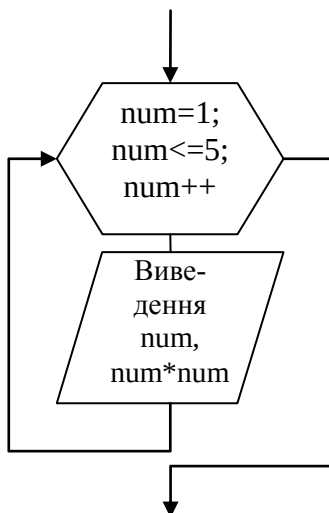
Третій вираз (*корекція*) обчислюється в кінці кожного виконання тіла циклу. Лічильник може як збільшуватися, так і зменшуватися.

Будь-який вираз може бути відсутнім, але символи крапки з комою (;), що їх розділяють, повинні бути обов'язково.

Наприклад.

Вивести на екран монітора числа від 1 до 5 та їх квадрати.

```
int num;
for(num=1; num<=5; num++)
    printf("\n %d * %d = %d", num, num, num*num);
```



Параметри, що знаходяться в заголовку циклу, можна змінити при виконанні операцій в тілі циклу.

Наприклад.

```
int k=2;
for(n=3; k<=25; )
{
    ...
    k = k*n;
```

```
}
```

У циклі `for` часто використовується операція «кома» (,) для розділення декількох виразів. Це дозволяє включити в специфікацію циклу декілька виразів, які будуть ініціалізуватися або коректуватися. Вирази, до яких застосовується операція «кома», обчислюватимуться зліва направо.

Наприклад.

```
for(i=0, j=1; i<n; i++, j=i);
```

У C/C++ допускаються вкладені цикли, тобто коли один цикл знаходиться усередині іншого.

Приклад

```
for( i=0; i<n; i++)
{
    for( j=0; j<n; j++)
    {
        ...
    }
    ...
}
```

Приклади використання регулярного циклу з параметром.

1) Збільшення лічильника;

```
for (n=0; n<10; n++)
    оператор;
```

2) Зменшення лічильника;

```
for (n=10; n>0; n--)
    оператор;
```

3) Довільна зміна кроку коректування;

```
for (n=2; n>60; n+=13)
    оператор;
```

4) Можливість перевіряти умову відмінну від умови, яка накладається на число ітерацій;

```
for (num=1; num*num*num<216; num++)
    оператор;
```

5) Корекція може здійснюватися не тільки за допомогою складання або віднімання.

```
for (d=100.0; d<150.0; d*=1.1)
{
    <тіло циклу>;
}
```

```

for (x=1; y<=75; y=5*(x++)+10)
{
<тіло циклу>;
}

```

2 Ітераційні цикли while та do-while

У ітераційних циклах заздалегідь невідома кількість разів виконання циклу.

1) Цикл while – ітераційний цикл з передумовою.

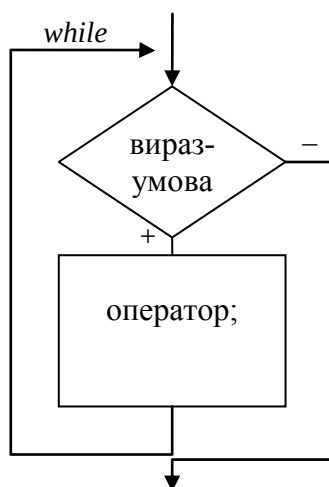
Загальна форма запису:

```

while (вираз-умова)
    оператор;

```

В якості <виразу-умови> найчастіше використовується відношення або логічний вираз. Якщо <вираз-умова> істинне (тобто не рівно нулю), то виконується оператор або група операторів, що входять в цикл while, та потім вираз перевіряється знову.



Послідовність дій, що складається з перевірки і виконання оператора, повторюється до тих пір, поки вираз не стане брехнею (тобто рівним нулю). Після цього відбувається вихід з циклу, і далі виконується оператор, що стоїть після оператора циклу. При побудові циклу while, в нього необхідно включити конструкції, що змінюють величину виразу, що перевіряється, так, щоб врешті-решт воно стало брехнею. Інакше виконання циклу ніколи не завершиться.

Цикл while – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу.

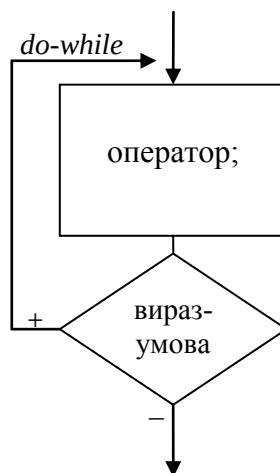
Наприклад.

```
k=5;
n=10;
while(k<n)
{
    printf("k=%d n=%d\n", до, n);
    k+=2;
    k=k+2;
    n++;
}
```

2) Do-while – ітераційний цикл з постумовою.

Загальна форма запису:

```
do
    оператор;
while (вираз-умова);
```



Цикл do-while — це цикл з постумовою, в якому істинність виразу-умови перевіряється після виконання всіх операторів, включених в тіло циклу. Тіло циклу виконується до тих пір, поки вираз не стане брехнею, тобто тіло циклу виконується хоч би один раз. Використовувати цикл краще всього в тих випадках, коли повинна бути виконана хоч би одна ітерація.

Наприклад.

```
do
{
    printf(" Введіть n>0");
    scanf("%d", &n);
}
while (n<0);
```

3 Оператори break і continue

У тілі будь-якого типу циклу можна використовувати оператора break, який дозволяє вийти з циклу, не завершуючи його. Оператор continue дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

1) break – оператор переривання циклу.

```
while (<вираз>)
{
    ...
    break;
    ...
}
```



Тобто оператор break доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу.

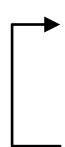
Наприклад.

Написати програму, яка підраховує суму чисел що вводяться з клавіатури до тих пір, поки не буде введено 100 чисел або число 0

```
for (s=0, i=1; i<100; i++)
{
    cin>>x;
    if (x==0) break; // якщо ввели 0, то підсумовування закінчується
    s+=x;
}
```

2) continue – оператор переходу до наступної ітерації циклу. Він використовується, коли тіло циклу містить галуження.

```
while (<вираз>)
{
    ...
    continue;
    ...
}
```



Наприклад.

Написати програму, яка підраховує кількість і суму додатних чисел

```
for ( k=0, s=0, x=1; x!=0; )
```

```

{
    cin>>x;
    if (x<=0) continue; //якщо ввели <=0, то перехід до наступної ітерації
    k++;s+=x;
}

```

При вкладених циклах дії операторів break і continue розповсюджується тільки на саму внутрішню структуру, в якій вони містяться.

Використання цих операторів можливе, але небажано, оскільки вони погіршують читаність програми, збільшують вірогідність помилок, утрудняють модифікацію.

Контрольні питання

- 1 Назвіть типи циклів.
- 2 Що називається тілом циклу?
- 3 Що називається кроком циклу?
- 4 Який оператор відповідає регулярному циклу?
- 5 Наведіть загальний вигляд оператора циклу з постумовою.
- 6 Який оператор відповідає циклу передумовою?
- 7 Наведіть загальний вигляд оператора регулярного циклу.
- 8 Який оператор відповідає циклу з постумовою?
- 9 Наведіть загальний вигляд оператора циклу з передумовою.
- 10 Які мінімальну кількість разів виконується цикл while?
- 11 Які мінімальну кількість разів виконується цикл do-while?
- 12 Охарактеризуйте призначення оператора break.
- 13 Охарактеризуйте призначення оператора continue?
- 14 Чи може в операторі регулярного циклу for бути відсутній будь-який з трьох виразів в заголовку?
- 15 Якщо мається декілька вкладених операторів циклів, то на який з них будуть мати вплив оператори break та continue?

Домашнє завдання

[1] с. 82-87, 91-97

Лекція № 13-14

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Масиви. Опис масивів, уявлення у пам'яті. Ініціалізація масивів. Складання програми для обробки лінійних масивів.

Мета заняття Пояснити студентам поняття масиву даних, представлення його в пам'яті ЕОМ, опис масиву, способи ініціалізації масиву. Розглянути алгоритми обробки лінійних масивів.

Час – 2 години

План проведення лекції

- 1 Одномірні масиви. Опис масивів, уявлення в пам'яті.
- 2 Ініціалізація масиву.
- 3 Алгоритми обробки лінійних масивів

Навчальні матеріали лекції

- 1 Одномірні масиви. Опис масивів, уявлення в пам'яті

Більшістю об'єктів мови C/C++, з якими ми мали справу, були змінні. Кожна змінна при оголошенні отримувала тип і ім'я, з яким зв'язувався певний елемент пам'яті. Проте розташування значень змінних по адресах пам'яті ніяк не упорядковувалося. При вирішенні багатьох завдань, особливо з великою кількістю однотипних даних, використання змінних з різними іменами, а значить не впорядкованих по адресах пам'яті, утрудняє або робить взагалі неможливим програмування. У подібних випадках в мові C/C++ використовують об'єкти, які називаються масивами.

Масив – це кінцева впорядкована послідовність однотипних даних. Кожному елементу масиву відводиться одна комірка пам'яті. Елементи одного масиву займають послідовно розташовані комірки пам'яті.

Всі елементи мають одне ім'я – ім'я масиву і відрізняються індексами – порядковими номерами в масиві. В C++ масиви можуть бути одномірними або багатомірними.

Одномірний (лінійний) масив – це список однотипних зв'язаних змінних. Доступ до окремого елементу лінійного масиву здійснюється за допомогою лише одного індексу. Кількість елементів в масиві називається його розміром.

Щоб відвести в пам'яті потрібну кількість комірок для розміщення масиву, треба заздалегідь знати його розмір. Резервування пам'яті для масиву виконується на етапі компіляції програми.

Загальна форма оголошення масиву

```
тип ім'я[розмір_масиву];
```

де тип – це тип елементу масиву, ім'я – ідентифікатор імені масиву, розмір_масиву – значення цілого типу, що відповідає кількості елементів масиву, під які необхідно відвести пам'ять.

Елементи масиву завжди нумеруються з 0.

Принцип розташування елементів масиву в пам'яті ЕОМ наведено на рисунку 1.1

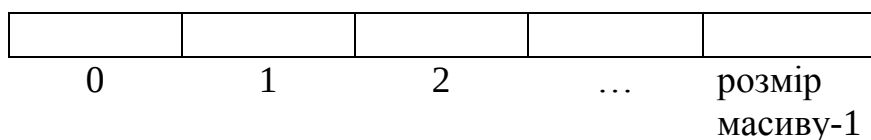


Рисунок 1.1 – Розташування елементів масиву у пам'яті

Наприклад.

```
int data[10]; //масив з 10 елементів цілого типу
```

Тут масив містить 10 елементів типу int:

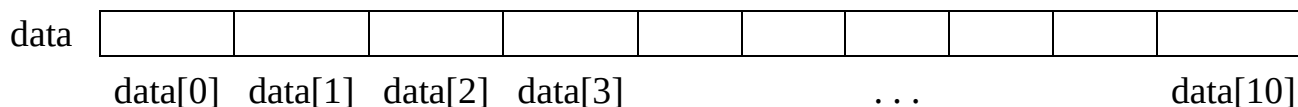


Рисунок 1.2 – Розташування елементів масиву data[10] у пам'яті

Щоб звернутися до елементу масиву, треба вказати ім'я масиву і номер елементу в масиві (індекс):

- `a[0]` – індекс задається як константа;
- `a[8]` – індекс задається як константа;
- `a[i]` – індекс задається як змінна;
- `a[2*i]` – індекс задається як вираз.

Така дія інакше називається адресацією або індексацією.

2 Ініціалізація масиву.

Існує два способи ініціалізації масиву:

- 1 Визначення початкових значень при оголошенні масиву.
- 2 Організація циклу послідовного введення значень елементів масиву.

Загальна форма ініціалізації масиву при оголошенні

тип ім'я[n]={значення_0, значення_1, значення_2, ..., значення_n-1};

Початкові значення елементів масиву включають у фігурні дужки. Якщо програміст не вказав в квадратних дужках розмір масиву, то компілятор сам задасть розмір по кількості приведених значень. Якщо початкові значення у фігурних дужках не задані, то всі елементи масиву будуть нульовими.

Наприклад.

```
int a[10]={1,2,3,4,5,6,7,8,9,10}; //ініціалізація 10 елементів
int a[10]={1,2,3,4,5}; //5 з 10 елементів задані, решта нулі
int a[]={1,2,3,4,5}; //створення масиву на основі 5-ти елементів
```

Організація циклу послідовного введення значень елементів масиву.

Обробку масивів треба виконувати поелементно в циклі. Цикл послідовного введення значень елементів масиву зручно організувати за допомогою оператора `for`.

Наприклад. Розробити програму введення з клавіатури масиву data[] із 5 цілих чисел та виведення на екран отриманого масиву.

Блок – схема алгоритму рішення задачі наведена на рисунку 2.1

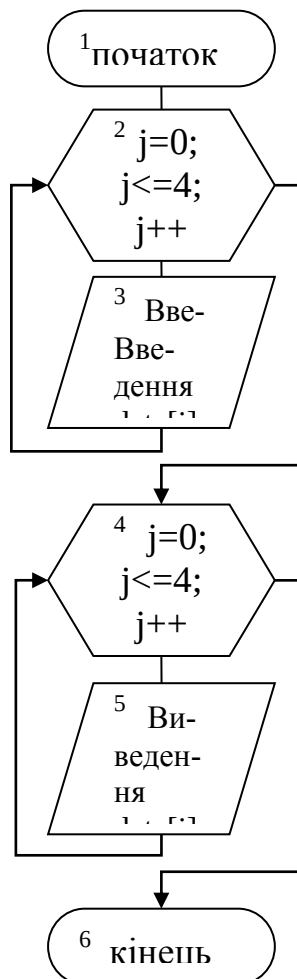


Рисунок 2.1 – Блок – схема введення-виведення одномірного масиву

Текст програми введення – виведення одномірного масиву із 5-ти цілих чисел наведено в лістингу 2.1

Лістинг 2.1

```

#include <stdio.h>
int main()
{
    int data[5];
    int j;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    printf("\n\n Введений масив:");
    for (j=0; j<=4; j++)
    {

```

```
    printf("\n Data[%i]=%i",j,data[j]);  
    }  
    return 0;  
}
```

3. Алгоритми обробки лінійних масивів

Приклад 1.

Для даного лінійного b масиву із 10 дійсних чисел знайти середньоарифметичне значення від'ємних елементів.

Програма буде містити три логічні блоки:

- блок введення значень елементів масиву;
- блок обчислення середньоарифметичного значення від'ємних елементів масиву;
- блок виведення результату. Враховувати, що від'ємні елементи в масиві можуть бути відсутні.

Блок – схема алгоритму рішення задачі приведена на рисунку 3.1.

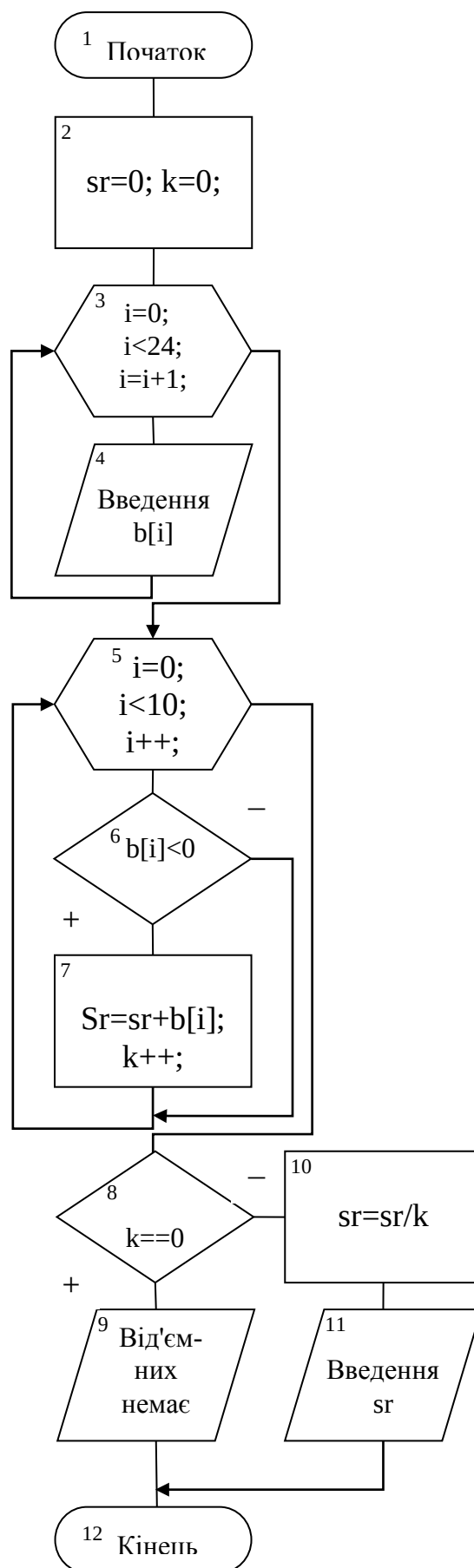


Рисунок 3.1. Блок – схема алгоритму пошуку середнього арифметичного значення від'ємних елементів лінійного масиву.

Програму рішення поставленої задачі наведено в лістингу 3.1

Лістинг 3.1

```
#include <iostream.h>
#include <conio.h>
int main (void)
{
    float b[10];//опис масиву
    float sr=0;
    int i;
    int k=0; //кількість від'ємних елементів у введеному масиві
    clrscr;
    // блок введення елементів масиву
    for (i=0;i<10;i++)
    { cout<<endl<<"Введите элемент массива b["<<i+1<<"]: ";
      cin>>b[i];
    }
    //блок обчислення середньоарифметичного значення
    for (i=0; i<10; i++)
    { if (b[i]<0) //перевіряємо, чи від'ємний елемент масиву
      { sr=sr+b[i];
        k++; //збільшуємо кількість від'ємних
      }
    }
    // блок виведення результату
    if (k==0)
    { cout<<endl<<"В массиве нет отрицательных элементов";
    }
    else
    { sr=sr/k;
      cout<<endl<<"Среднеарифметическое значение отрицательных
элементов= "<<sr;
    }
    cout<<endl<<"Для завершения работы нажмите Enter ";
    getch();
    return 0;
}
```

Приклад 2.

Дано лінійний масив цілих чисел $b[]$, що містить 12 елементів. Знайти парні елементи вхідного масиву, розташовані на непарних місцях та помістити їх в масив $rez[]$.

Програму рішення поставленої задачі наведено в лістингу 3.2

Лістинг 3.2

```
#include <iostream.h>
#include <conio.h>
int main (void)
{
    int b[12]; //опис вхідного масиву
```

```

int rez[12];    //опис вихідного масиву
int i;
int k=0;        //кількість шуканих елементів
    clrscr;

for (i=0;i<12;i++)
{ cout<<endl<<"Введіть елемент масива b["<<i+1<<"]: ";
  cin>>b[i];
}
//i отримує тільки парні значення, що відповідають непарним місцям
for (i=0; i<12; i=i+2)
//якщо елемент масиву b, розташований на непарному місці, парний
if (b[i]%2==0)    { rez[k]=b[i]; //записуємо його в масив rez
    k++;          // збільшуємо k
}

if (k==0)
    cout<<endl<<"Нет искомым элементов";
else
{ cout<<endl<<"Результирующий массив";
  for (i=0; i<k; i++)
    cout<<endl<<rez[i];
}

cout<<endl<<"Для завершения работы нажмите Enter ";
getch();
return 0;

}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дайте визначення поняття "масив".
- 2 Що називається розміром масиву?
- 3 Як розташовані у пам'яті елементи масиву?
- 4 Що спільного мають всі елементи масиву?
- 5 Чим відрізняються всі елементи масиву?
- 6 Наведіть загальну форму оголошення масиву.

- 7 Що називається індексом елемента масиву?
- 8 З якого номера починається нумерація елементів в масиві?
- 9 Назвіть способи ініціалізації масиву.
- 10 Чи можна змінювати розмір масиву у пам'яті під час роботи програми?
- 11 Що необхідно вказати, щоб звернутися до елементу масиву?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Написати програму, що вводить із клавіатури одномірний масив з 5 цілих чисел, після чого виводить кількість ненульових елементів.

2 Дано лінійний масив із 10 дійсних чисел. Написати програму, що перевіряє, чи перебуває уведене із клавіатури число в масиві.

3 Написати програму, що обчислює середню (за тиждень) температуру повітря. Вихідні дані повинні вводитися під час роботи програми. Рекомендований вид екрану наведено нижче (дані, введені користувачем, виділені напівжирним шрифтом).

Введіть температуру воздуха за неделю.

Понедельник **12**

Вторник **10**

Среда **16**

Четверг **18**

Пятница **17**

Суббота **16**

Воскресенье **14**

Средняя температура за неделю: **14.71 град.**

4 Написати програму, що перевіряє, чи представляють елементи уведеного із клавіатури масиву зростаючу послідовність.

5 Написати програму, що обчислює, скільки разів уведене із клавіатури число зустрічається в масиві.

6 Написати програму, що перевіряє, є чи в уведеному із клавіатури масиві елементи з однаковим значенням.

Лекція № 15

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Алгоритм та програма знаходження екстремумів. Алгоритм та програма сортування "бульбашковим" методом.

Мета заняття Пояснити студентам ідеї алгоритмів пошуку екстремумів та ідеї алгоритмів сортування елементів одномірного масиву: метод вибору, сортування обмінами, сортування вставками. Розглянути практичні приклади застосування наведених алгоритмів. Пояснити поняття та організацію псевдо динамічного масиву.

Час – 2 години

План проведення лекції

- 1 Алгоритми пошуку екстремумів.
- 2 Формування псевдодинамічних масивів.
- 3 Алгоритми сортування лінійних масивів
 - 3.1 Сортування масиву методом вибору.
 - 3.2 Сортування обмінами («бульбашковий» метод сортування).
 - 3.3 Сортування простими вставками.

Навчальні матеріали лекції

- 1 Алгоритми пошуку екстремумів

Знаходження екстремумів полягає у знаходженні максимального та/або мінімального елемента в масиві.

Одним з класичних алгоритмів знаходження екстремумів є наступний. Спочатку за максимальний (мінімальний) елемент береться будь-який (напри-

клад, нульовий) елемент масиву та заноситься в змінну «поточного» максимуму (мінімуму). Потім, за допомогою циклу від першого до останнього елементу масиву необхідно порівнювати змінну «поточного» максимуму (мінімуму) з елементами масиву i , якщо черговий елемент масиву більше (менше) «поточного» максимуму (мінімуму), необхідно зберігати цей елемент в відповідній змінній. Після проходження всього масиву в змінній «поточного» максимуму (мінімуму) буде міститись шуканий елемент.

Нехай

```
int data[5];    // вхідний масив з 5-ти цілих чисел
int max;        // змінна «поточного» максимуму
int min;        // змінна «поточного» мінімуму
```

Блок – схему алгоритму пошуку екстремумів масиву `data[]` наведено на рисунку 1.1

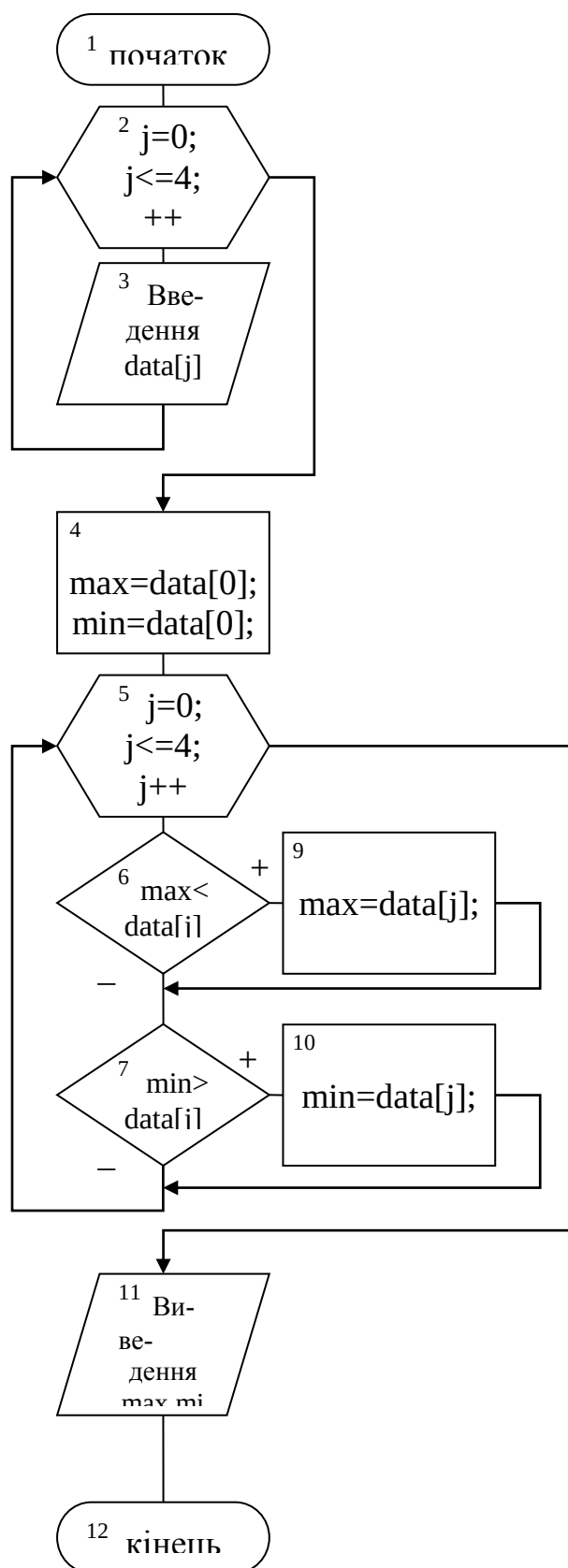


Рисунок 1.1 – Блок-схема алгоритму пошуку екстремумів.

Текст програми пошуку екстремумів лінійного масиву наведено у лістингу 1.1

Лістинг 1.1

```
#include <stdio.h>
int main()
{
    int data[5];
    int j,max,min;
    // Цикл введення масиву
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    // «Поточні» максимум та мінімум отримують стартові значення
    max=data[0];
    min=data[0];
    /* Цикл порівняння «поточних» екстремумів з кожним елементом
    i, в разі необхідності заміни їх значень */
    for (j=0; j<=4; j++)
    {
        if (max<data[j]) max=data[j];
        if (min>data[j]) min=data[j];
    }
    printf("\n Max=%i \n Min=%i", max, min);
    return 0;
}
```

Якщо масив містить декілька максимальних та/або мінімальних елементів, то приведена програма знайде перші з них за розташуванням в масиві.

Ускладнимо попередню задачу та знайдемо мінімальний непарний елемент лінійного масиву цілих чисел data[]. Для цього спочатку, в окремому циклі, знайдемо перший непарний елемент масиву і занесемо його в змінну «поточного» мінімуму. До речі, цей цикл допоможе нам з'ясувати, чи є у вхідному масиві непарні елементи.

Текст програми пошуку мінімуму серед непарних елементів лінійного масиву наведено у лістингу 1.2

Лістинг 1.2

```

#include <stdio.h>
int main()
{
    int data[5];
    int j,max,min;
    for (j=0; j<=4; j++)
    {
        printf("\n Введіть елемент масиву data[%i]",j);
        scanf("%i", &data[j]);
    }
    // пошук першого непарного елемента в масиві
    j=0;
    while (j<5 && data[j]%2==0) j++; //пропускаємо всі парні на
                                     //початку масиву
    if (j==5) //якщо кількість парних рівна розміру масиву
    {
        printf("\n В масиві відсутні непарні елементи");
        return 0; //вихід на кінець
    }
    // пошук мінімуму, починаючи з першого непарного
    min=data[j];
    for (int i=j+1; i<=4; i++)
    {
        //якщо черговий елемент непарний і менше «поточного» мінімуму
        if (data[i]%2!=0 && min>data[i])
            min=data[i]; //змінюємо значення «поточного» мінімуму
    }
    printf("\n  Min=%i", min);
    return 0;
}

```

2 Формування псевдодинамічних масивів.

При описі масиву в програмі треба обов'язково вказувати кількість елементів масиву для того, щоб компілятор виділив під цей масив потрібну кількість пам'яті. Це не завжди буває зручно, оскільки число елементів в масиві може змінюватися залежно від вирішуваного завдання. В цьому випадку в програмі формують псевдодинамічні масиви. Псевдодинамічні масиви реалізуються таким чином:

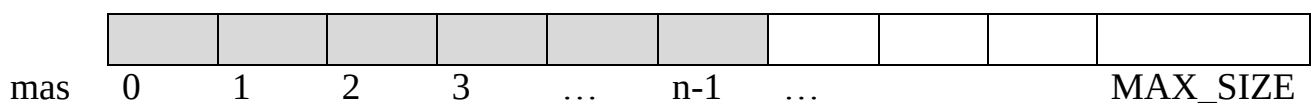
- 1) при визначенні масиву виділяється достатньо велика кількість пам'яті:

```
const int MAX_SIZE=100; //іменована константа
int mas[MAX_SIZE];
```

2) користувач вводить реальну кількість елементів масиву, але меншу за MAX_SIZE.

```
int n;
cout<<"\n Введіть розмір масиву, меншу за "<<MAX_SIZE<<": ";
cin>>n;
```

3) подальша робота з масивом обмежується заданою користувачем розмірністю n.



Як бачимо, під масив mas[] виділяється блок пам'яті фіксованого розміру, що вміщує MAX_SIZE цілих чисел. А кожного разу масив обробляється до елемента з індексом n-1. Тому цей масив і називається псевдодинамічним.

3 Алгоритми сортування лінійних масивів

Сортування масивів полягає у розстановці елементів масиву у певному порядку – за зростанням або за спаданням. В залежності від способу порівняння і обміну елементів розрізняють різні типи алгоритмів сортування.

Основними алгоритмами сортування є:

- сортування масиву методом вибору;
- сортування обмінами («бульбашковий» метод сортування);
- сортування простими вставками.

3.1 Сортування масиву методом вибору.

Одномірний масив із n цілих чисел відсортуємо за зростанням значень елементів методом вибору. Алгоритм полягає в тому, що вибирається най-

менший елемент і міняється місцями з першим елементом масиву, потім розглядаються елементи масиву, починаючи із другого, і найменший з них міняється місцями із другим елементом, і так далі $n-1$ раз (при останньому проході циклу при необхідності міняються місцями передостанній й останній елементи масиву).

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуємо його за зростанням елементів.

Блок – схема алгоритму сортування за зростанням одномірного масиву методом вибору представлена на рисунку 3.1

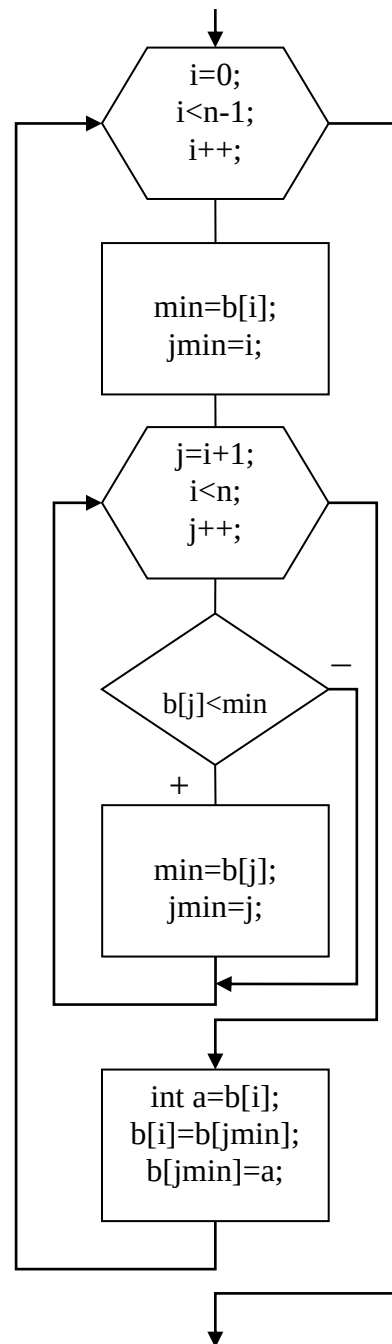


Рисунок 3.1. Блок-схема алгоритму сортування вибором.

Текст програми сортування за зростанням одномірного масиву методом вибору представлено в лістингу 3.1

Лістинг 1.1

```

#include <iostream.h>
#include <conio.h>
int main()
{
    //Організація псевдо динамічного масиву

```



```

int b[100];          //виділення пам'яті під масив цілих
                     //максимальної довжини
int n;              // фактична кількість елементів масиву
int min, jmin;      //мінімальний елемент та його номер
int i, j;
cout<<endl<<"Введіть кількість елементів в масиві";
    cin >> n;
// Введення масиву із клавіатури
for (i = 0; i<n; i++)
{
    cout<<endl<<"Введіть елемент масива b["<<i+1<<"] ";
    cin >> b[i];
}

for (i=0; i<n-1; i++) // n-1 раз шукаємо найменший елемент
{
    // приймаємо за найменший перший з розглянутих елементів
    min = b[i]; jmin=i; // та запам'ятовуємо його номер
    // пошук мінімального елемента з неупорядкованих:
    for (j = i + 1; j<n; j++)
        if (b[j] < min)
            // якщо знайшли менший елемент, запам'ятовуємо його та його номер
            { min=b[j]; jmin = j;}
    // обмін елементів з номерами i та jmin
    int a = b[i];
    b[i] = b[jmin];
    b[jmin] = a;
}
// Виведення впорядкованого масиву
cout<<endl<<"Упорядоченный массив";
for (i = 0; i<n; i++)
    cout<< endl<<b[i];
cout<<endl<<"Для завершения программы нажмите Enter";
getch();
return 0;
}

```

3.2 Сортування обмінами («бульбашковий» метод сортування)

Одномірний масив із n цілих чисел відсортуємо за зростанням значень елементів. Алгоритм полягає в тому, що при перегляді вхідного масиву попарно порівнюються сусідні елементи. Якщо порядок їхнього проходження не відповідає заданому критерію впорядкованості, то елементи міняються місцями. В

результаті одного такого перегляду при сортуванні за збільшенням елементів елемент з найбільшим значенням переміститься на останнє місце в масиві. При наступному проході на своє місце переміститься другий за величиною елемент і т.д. Для постановки на свої місця n елементів слід зробити $n-1$ проходів. При кожному черговому проході кількість елементів, що порівнюються, треба зменшувати на одиницю.

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуємо його за зростанням елементів.

Блок – схема сортування за зростанням одномірного масиву обмінами наведена на рисунку 3.2

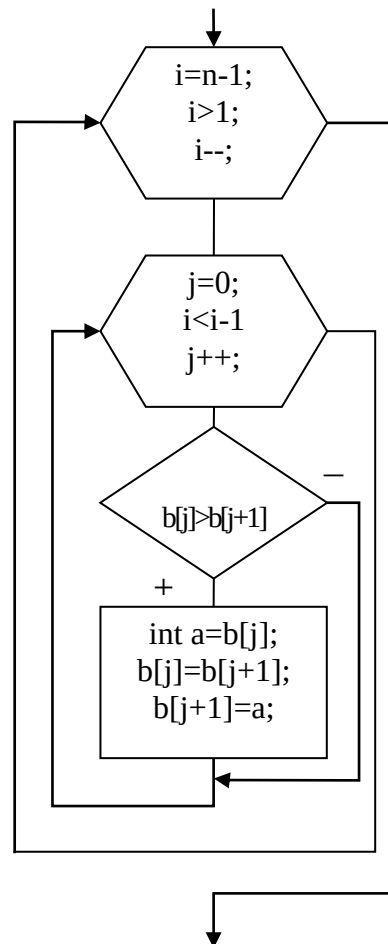


Рисунок 3.2 Блок – схема алгоритму сортування обмінами.

Текст програми сортування за зростанням одномірного масиву обмінами представлено в лістингу 3.2

Лістинг 3.2

```
#include <iostream.h>
#include <conio.h>
int main()
{
    //Організація псевдо динамічного масиву
    //виділення пам'яті під масив цілих максимальної довжини
    int b[100];
    int n;                // фактична кількість елементів масиву
    int min, jmin;        //мінімальний елемент та його номер
    int i, j;
    cout<<endl<<"Введіть кількість елементів в масиві";
    cin >> n;
    // Введення масиву із клавіатури
    for (i = 0; i<n; i++)
    {
        cout<<endl<<"Введіть елемент масива b["<<i+1<<"] ";
        cin >> b[i];
    }
    //Цикл проходів по масиву
    for (i=n-1; i>1; i--)
    { // цикл попарного порівняння сусідніх елементів
        for (j =0; j<=i-1; j++)
            if (b[j]>b[j+1])
            { // обмін елементів з номерами j та j+1
                int a = b[j];
                b[j] = b[j+1];
                b[j+1] = a;
            }
        }
    }
    // Виведення впорядкованого масиву
    cout<<endl<<"упорядоченный массив";
    for (i = 0; i<n; i++)
        cout<< endl<<b[i];
    cout<<endl<<"Для завершения программы нажмите Enter";
    getch();
    return 0;
}
```

3.3 Сортування простими вставками

Одномірний масив $b[]$ із n цілих чисел відсортуємо за зростанням значень елементів методом вибору. Ідея алгоритму: для кожного елемента масиву $b[i]$

вважаємо, що попередні елементи вже відсортовані. Елемент $b[i]$ потрібно вставити на відповідне місце в вже відсортовану послідовність $b[0] \dots b[i-1]$. Це місце визначається послідовним порівнянням $b[i]$ з вже впорядкованими елементами i , якщо необхідно - елементи міняються місцями.

Нехай $b(n)$ – масив із n цілих чисел (n вводиться користувачем). Організуємо псевдо динамічний масив та відсортуємо його за зростанням елементів.

Блок – схема сортування за зростанням одномірного масиву вставками наведена на рисунку 3.3

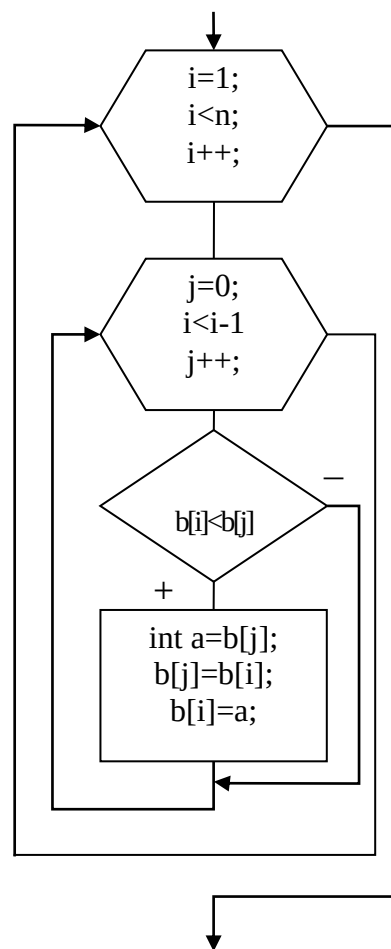


Рисунок 3.3 Блок – схема алгоритму сортування вставками.

Текст програми сортування за зростанням одномірного масиву вставками представлено в лістингу 3.3

Лістинг 3.3

```

#include <iostream.h>
#include <conio.h>
int main()
{
    //Організація псевдо динамічного масиву
    //виділення пам'яті під масив цілих максимальної довжини
    int b[100];
    int n; // фактична кількість елементів масиву
    int i, j;
    cout<<endl<<"Введите количество элементов в массиве";
    cin >> n;
    // Введення масиву із клавіатури
    for (i = 0; i<n; i++)
    {
        cout<<endl<<"Введите элемент массива b["<<i+1<<"] ";
        cin >> b[i];
    }
    // цикл порівняння b[i]-го елемента з усіма попередніми
    for (i=1; i<n; i++)
    {
        for (j =0; j<=i-1; j++)
            if (b[i]<b[j])
            {
                // обмін місцями елементу b[j]-го з b[i]-м
                int a = b[j];
                b[j] = b[i];
                b[i] = a;
            }
    }
    // Виведення впорядкованого масиву
    cout<<endl<<"Упорядоченный массив";
    for (i = 0; i<n; i++)
        cout<< endl<<b[i];
    cout<<endl<<"Для завершения программы нажмите Enter";
    getch();
    return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Опишіть алгоритм пошуку максимального (мінімального) елемента масиву.
- 2 Що треба змінити в наведеній програмі пошуку екстремумів (див. Лістинг 1.1), щоб отримати останні за розташуванням в масиві екстремальні елементи (тобто останній мінімум та останній максимум)?
- 3 В якому випадку максимальний елемент масиву дорівнюватиме мінімальному?
- 4 Як сформувати псевдодинамічний масив?
- 5 Який фактичний розмір псевдо динамічного масиву?
- 6 Чому масив називається псевдодинамічним?
- 7 Яка ідея полягає в основі алгоритму сортування обмінами?
- 8 Опишіть алгоритм сортування методом вибору?
- 9 Опишіть алгоритм сортування вставками?
- 10 Скільки проходів по масиву відбувається при сортуванні обмінами вибором? вставками?
- 11 На вхід подається відсортований масив. Як вдосконалити метод обмінами так, щоб сортування припинялось вже після першого проходу по масиву?

Завдання для самоперевірки засвоєного матеріалу на лекції

Розробити програми мовою C/C++ згідно постановки задачі:

- 1 Для заданого одномірного масиву дійсних чисел знайти останній мінімальний елемент та вивести його номер.
- 2 В заданому одномірному масиві дійсних чисел поміняти місцями мінімальний та максимальний елементи.

- 3 Для заданого одномірного масиву дійсних чисел знайти мінімальний серед від'ємних елементів масиву та вивести його номер.
- 4 В веденому масиві цілих чисел знайти мінімум парних елементів масиву та максимум непарних елементів. Вивести знайдені екстремуми або повідомити, що відповідні елементи в масиві відсутні.
- 5 В заданому одномірному масиві дійсних чисел відсортувати за зростанням тільки непарні елементи використовуючи метод вставками.
- 6 Написати програму, що методом прямого вибору сортує за спаданням введеній із клавіатури одномірний масив.
- 7 Написати програму, що методом обміну («бульбашковим» методом) сортує за спаданням введеній із клавіатури одномірний масив.
- 8 Введені з клавіатури два масиви дійсних чисел впорядкувати за зростанням. Об'єднати два впорядковані масиви в один, теж впорядкований масив.

Лекція № 16-17

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Матриця. Приклади вирішення задач з використанням матриці.

Мета заняття Пояснити студентам поняття багатомірного масиву даних, представлення його в пам'яті ЕОМ, опис масиву, способи ініціалізації масиву. Розглянути алгоритми обробки матриць.

Час – 2 години

План проведення лекції

- 1 Багатомірні масиви. Матриці. Опис, стартова ініціалізація.
- 2 Використання конструкцій мови C/C++ для обробки матриць.

Навчальні матеріали лекції

- 1 Багатомірні масиви. Матриці. Опис, стартова ініціалізація

У мові C/C++ масиви бувають не тільки одновимірні, але і багатовимірні. Відмінність полягає в тому, що в одновимірному масиві положення елемента визначається одним індексом, а в багатовимірному – декількома. Теоретично на число розмірностей масиву ніяких обмежень не накладається. При визначенні багатомірних масивів задається тип компонентів масиву, ім'я масиву, а потім, в окремих квадратних дужках, число елементів у першому "вимірі", другому, третьому й т.д.

Загальна форма оголошення багатовимірною масиву:

тип ім'я_масиву[індекс_1][індекс_2]...[індекс_n];

де тип – визначає тип даних елементів, з яких буде складатися масив,

ім'я_масиву – ідентифікатор масиву,

індекс_n – кількість елементів масиву у кожній його розмірності.

Елементи багатовимірного масиву розташовуються в послідовних елементах оперативної пам'яті за збільшенням адрес. Між елементами немає розривів. У пам'яті ЕОМ елементи розташовуються рядковим чином так, що найшвидше міняється останній індекс.

Приклад розташування в пам'яті ЕОМ тривимірного масиву `int d[2][2][2]`

```
d[0][0][0] d[0][0][1]
d[0][1][0] d[0][1][1]
d[1][0][0] d[1][0][1]
d[1][1][0] d[1][1][1]
```

Простіший багатовимірний масив – двовірний. Двовірний масив являє собою список одномірних масивів. Для того щоб оголосити двовірний масив необхідно задати лише дві розмірності.

```
тип ім'я_масиву [індекс_1] [індекс_2];
```

де тип – визначає тип даних елементів, з яких буде складатися масив,
 ім'я_масиву – ідентифікатор масиву,
 індекс_1 – кількість рядків матриці,
 індекс_2 – кількість стовпців матриці.

Наприклад, для оголошення двовірного масиву цілих чисел розмірністю 10 рядків та 20 стовпців, необхідно написати:

```
int twod [10][20];
```

В двовірному масиву позиція кожного елемента визначається двома індексами. Якщо уявити двовірний масив у вигляді таблиці даних, то перший індекс відповідає рядку, а другий – стовпцю. Якщо доступ до елементів масиву уявити в порядку, в якому вони реально зберігаються у пам'яті, то другий індекс буде змінюватись швидше, ніж перший.

Приклад розташування в пам'яті ЕОМ двовірного масиву `int d[3][4]` наведений у таблиці 1.1.

Таблиця 1.1 – Приклад розташування в пам'яті ЕОМ двовірного масиву

стовпці рядки	0	1	2	3
0	d[0][0]	d[0][1]	d[0][2]	d[0][3]
1	d[1][0]	d[1][1]	d[1][2]	d[1][3]
2	d[2][0]	d[2][1]	d[2][2]	d[2][3]

Для того щоб отримати доступ до елементу матриці необхідно вказати ім'я масиву та два індекси – індекс рядка та індекс стовпця. Наприклад, для доступу до елементу масиву `twod` з координатами 3,5, необхідно записати:

```
twod[3][5];
```

Необхідно пам'ятати, що місце зберігання всіх елементів масиву визначається під час компіляції. Окрім того, пам'ять, виділена для зберігання масиву, використовується в продовж всього терміну існування масиву. Для визначення кількості байт пам'яті, яку займає двовірний масив, необхідно використовувати наступну формулу:

*число байт = кількість рядків * кількість стовпців * розмір типу в байтах*

Відповідно, двовірний цілочисельний масив, описаний як

```
int mas [10][5];
```

займає в пам'яті $10 \times 5 \times 4 = 200$ байт.

При стартовій ініціалізації багатовимірний масив список ініціалізаторів кожної розмірності (підгрупу ініціалізаторів) можна заключити у фігурні дужки. Наприклад, для ініціалізації масиву `mas[3][5]` можна використати вираз:

```
int mas[3][5]={
    {1, 5, -5, 0, 7},
    {0, -5, 100, 20, 19},
    {0, -2, -3, -4, 10}
};
```

Початкових значень, як й в одномірних масивах, може бути менше, ніж число елементів масиву. Іншим елементам буде привласнюватися нульове значення. Наприклад:

```
int m[2][3]={ { 10, 20 }, { 30.40 } };
```

Це еквівалентно наступному явному визначенню:

```
int m[2][3]={ 10, 20, 0, 30, 40, 0 };
```

Подібно одномірним масивам, число початкових значень не повинне перевищувати числа елементів у рядку:

```
int m[2][3]={ {10,20,30,40},{50,60}}; // помилка!
```

Метод визначення розміру масиву шляхом специфікації початкових значень може використатися й для багатомірних масивів, але тільки частково. Дозволяється опускати число рядків, але потрібно задавати число стовпців. Компілятор буде підраховувати число початкових значень і визначати число рядків.

```
int b[ ][ 3 ]={{10,20,30},{ 0,50,60}};
```

Незалежно від того, указується число рядків, чи ні, не можна опускати число стовпців, це помилка:

```
int m[2][ ]={{10,20,30},{40,50,60}} // помилка!
```

Теоретично компілятор може підраховувати групи рядків і розуміти структуру матриці, однак він цього не робить. Можливо, краще задавати розмірність масивів явно.

2 Використання конструкцій мови C/C++ для обробки матриць

При переборі елементів багатомірного масиву варто використати вкладені цикли. У вкладених циклах багатомірних масивів внутрішній цикл завершує всі свої операції спочатку для однієї ітерації зовнішнього циклу, потім для наступної ітерації зовнішнього циклу й т.д.

Для обробки матриць, зазвичай, використовують два регулярних цикли – зовнішній для переходів по рядкам, внутрішній для переходів по стовпцям.

Розглянемо приклад коду для введення елементів матриці цілих чисел `a[][]`, що складається з 3-х рядків та 4-х стовпців, та виведення її у загально прийнятому вигляді (див. лістинг 2.1). для введення та виведення матриці використовуються регулярні цикли `for`:

```
for (i=0; i<3; i++)
for (j=0; j<4; j++)
...

```

Внутрішній цикл змінює індекс `j` від 0 до 3 для кожного значення зовнішнього циклу по індексу `i` (який міняється від 0 до 2).

Лістинг 2.1 – Приклад коду введення елементів матриці та виведення їх на екран

```
#include<iostream.h>
#include <conio.h>
int main()
{
    int a[3][4];
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<<"\n Введіть A["<<i+1<<","<<j+1<<"]: ";
            cin>>a[i][j];
        }
    }
    system("cls"); // очищення екрану
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout.width(3); /*встановлення ширини
                               виводу в 3 позиції*/
            cout<<a[i][j];
        }
        cout<<endl;
    }
    getch();
    return 0;
}
```

Блок – схему алгоритму рішення даної задачі наведено на рисунку 2.1

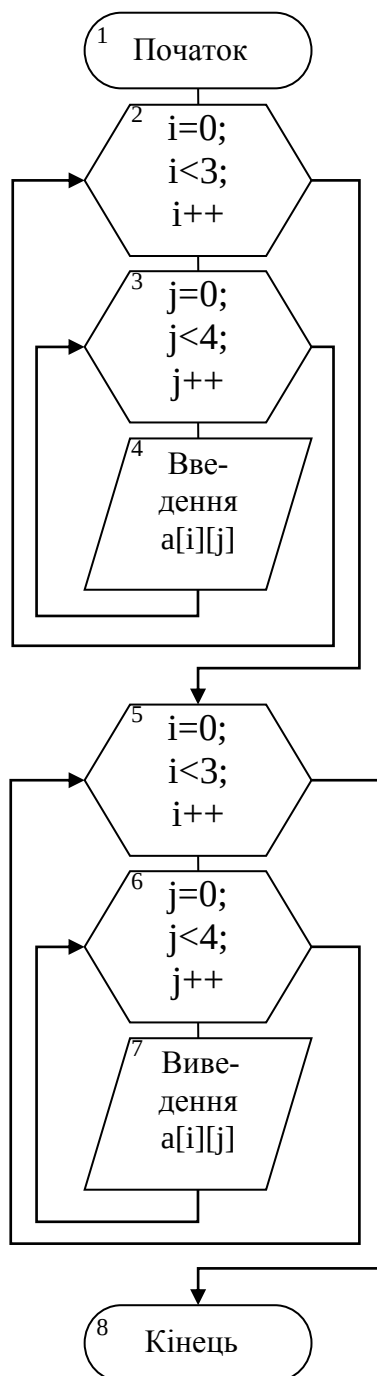


Рисунок 2.1. Блок – схема алгоритму введення – виведення матриці.

Приклад.

1 Постановка задачі.

Дано матрицю $A[3][4]$. Обчислити добуток від'ємних парних елементів у кожному з рядків.

2 Блок – схема алгоритму обчислення добутку від'ємних парних елементів у кожному з рядків матриці (див. рисунок 2.2).

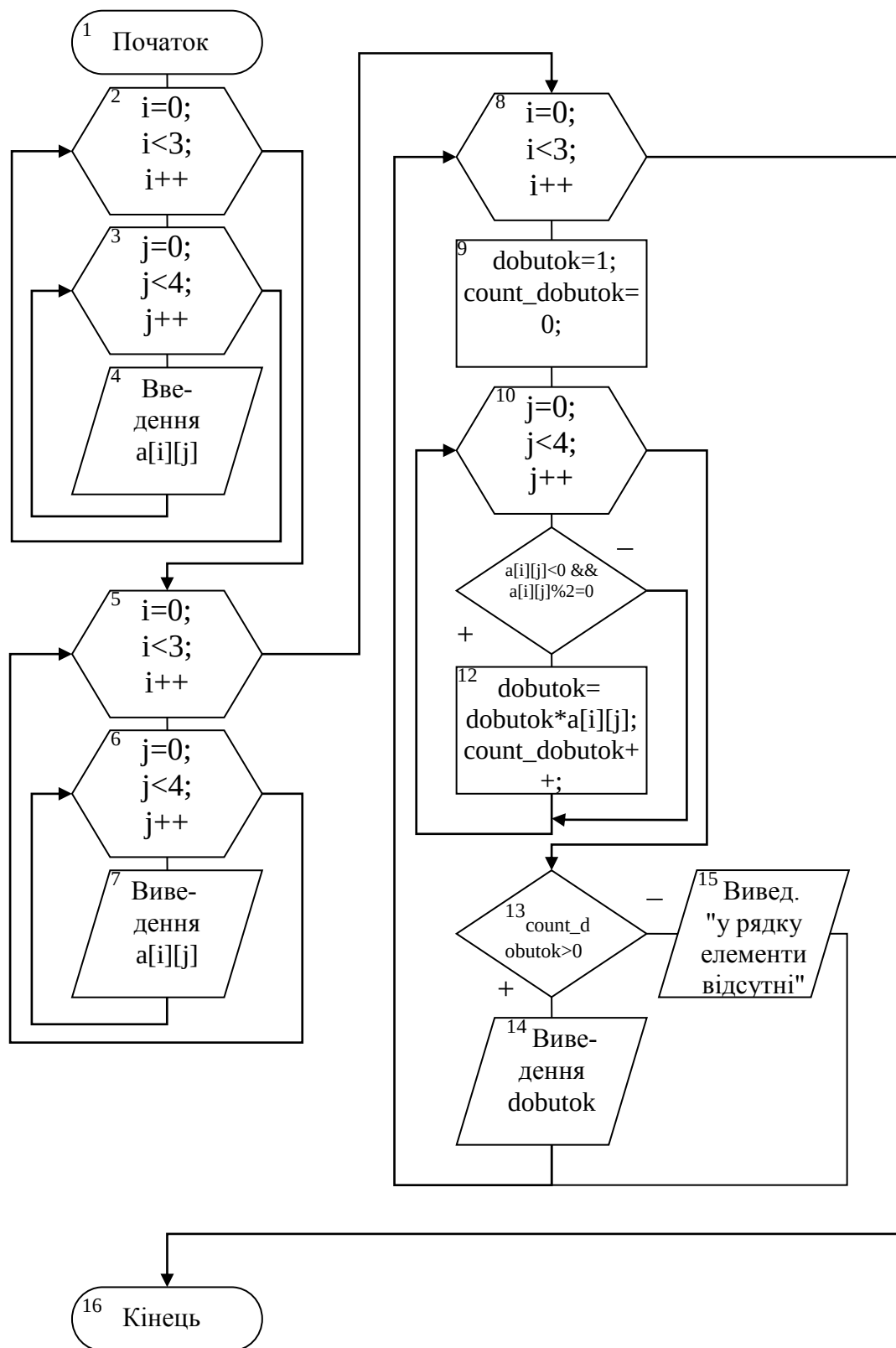


Рисунок 2.2. Блок – схема алгоритму обчислення добутку від'ємних парних елементів у кожному з рядків матриці

Лістинг 2.2 – Приклад коду програми для обчислення добутку від'ємних парних елементів у кожному з рядків матриці

```
#include <iostream.h>
#include <conio.h>
int main()
{
    int a[3][4];
    int dobutok=1;
    int count_dobutok=0;
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<<endl<<"Введіть A["<<i+1<<","<<j+1<<"]: ";
            cin>>a[i][j];
        }
    }
    system("cls");
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout.width(3);
            cout<<a[i][j];
        }
        cout<<endl;
    }
    for(i=0;i<3;i++)
    {
        dobutok=1;
        count_dobutok=0;
        for(j=0;j<4;j++)
        {
            if (a[i][j]<0 && a[i][j]%2==0)
            {
                dobutok=dobutok*a[i][j];
                count_dobutok++;
            }
        }
        if (count_dobutok>0)
            cout<<endl<<"В "<<i+1<<"-у рядку добуток від'ємних парних= "
                <<dobutok;
        else
```

```

        cout<<endl<<"В " <<i+1<<"-у рядку від'ємні парні відсутні";
    }
    getch();
    return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для актуалізації опорних знань та перевірки попереднього матеріалу.

- 1 Наведіть загальну форму оголошення масиву.
- 2 Що називається індексом елемента масиву?
- 3 З якого номера починається нумерація елементів в масиві?
- 4 Назвіть способи ініціалізації масиву.
- 5 Чи можна змінювати розмір масиву у пам'яті під час роботи програми?
- 6 Що необхідно вказати, щоб звернутися до елемента масиву?

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що називається багатовимірним масивом?
- 2 Наведіть загальну форму оголошення багатовимірного масиву.
- 3 З якого числа починається нумерація індексів багатовимірного масиву?
- 4 Як розташовується в пам'яті багатовимірний масив?
- 5 Наведіть загальну форму оголошення двовимірного масиву.
- 6 Наведіть приклад ініціалізації багатовимірного масиву.
- 7 Скільки і яку назву мають індекси, необхідні для звернення до елемента матриці?
- 8 Як визначити загальний розмір пам'яті, який займе матриця?
- 9 Які ініціалізатори використовує компілятор у випадку, коли у підгрупах ініціалізаторів не вистає членів підгруп?

10 Які типи циклів і в якому порядку зазвичай використовують для виконання будь-яких дій над матрицями?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Дана матриця $B(3,4)$. Знайти і роздрукувати мінімальний елемент і середнє арифметичне кожного стовпчика.

2 Дана матриця $A(4, 4)$. Знайти добуток та суму ненульових елементів спочатку головної діагоналі, а потім – останнього стовпчика.

3 Дана матриця $T(4,4)$. Знайти мінімальний елемент серед максимальних елементів стовпців.

4 Дані дві матриці: $A(5, 5)$ та $B(5, 5)$. Підрахувати добуток елементів головної діагоналі кожної матриці. Роздрукувати ту матрицю, у якій добуток більший.