МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

ЧАСТИНА 1. БАЗОВІ КОНЦЕПЦІЇ ПРОГРАМУВАННЯ. ЛАБОРАТОРНИЙ ПРАКТИКУМ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»
спеціальності 122 Комп’ютерні науки

Укладачі: В. В. Романов, Т. І. Просянкіна-Жарова, О. Ю. Безносик

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент *Терентьєв О. М.,* доктор технічних наук, доцент, провідний науковий співробітник Інституту телекомунікацій і глобального інформаційного простору НАН України

Відповідальний редактор *Тимощук О. Л.*, кандидат технічних наук, доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 02.09.2022 р.)
за поданням Вченої ради Інституту Прикладного системного аналізу
(протокол № 6 від 29.06.2022 р.)

Навчальний посібник є складовою навчально-методичного забезпечення дисципліни «Алгоритмізація та програмування». У посібнику викладено теоретичні основи алгоритмізації та програмування, практичні поради щодо застосування у програмуванні алгоритмічних структур, проектування алгоритмів, в тому числі із використанням різних структур даних, приділено увагу процедурно-орієнтованій технології розробки програмних продуктів, наведено приклади виконання завдань та запропоновано перелік завдань для виконання під час лабораторних занять.

Посібник призначений для здобувачів освітнього ступеня бакалавр за освітньою програмою «Інтелектуальні сервіс-орієнтовані розподілені обчислювання» спеціальності 122 Комп'ютерні науки, буде також корисним для всіх, хто навчається програмуванню

Реєстр. № НП 22/23-034. Обсяг 7,1 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ВСТУП

Дисципліна "Алгоритмізація та програмування" спрямована на ознайомлення студентів із сучасними науковими концепціям, поняттями, методами алгоритмізації та технологій програмування, властивостями і особливостями алгоритмів, а також процесів створення працездатних програм мовою високого рівня. Завданням дисципліни є вивчення типових алгоритмічних конструкцій та засобів представлення алгоритму; отримання знання про синтаксис і семантику базових конструктивних елементів мови програмування: лексем, виразів та операторів; отримання знання про різновиди типів даних, як простих, так і складених (масивів, структур, об’єднань); усвідомлення особливості програмування функцій; усвідомлення парадигми імперативного програмування; вивчення особливостей програмування динамічних та файлових структур даних; отримання знання про основні прийоми структурного програмування; формування комплексного уявлення про етапи розробки програми, основні поняття та методи технологій програмування; оволодіння прийомами та технологією налагодження та тестування програм; отримання знання про основні вимоги до документування програмних продуктів

Мета: ознайомлення студентів із спеціальними знаннями, які знаходяться на межі математики та інформатики, з сучасними поглядами на алгоритмічні процеси; набути навичок розробки та аналізу алгоритмів; навчитися застосовувати математичний апарат по дослідженню алгоритмів та створенню комп'ютерних програм; ознайомити з сучасними мовами програмування для вирішення прикладних задач.

Предмет вивчення – теоретичні основи алгоритмізації та програмування, техніка застосування у програмуванні базових алгоритмічних структур і базових структур даних, практичні навички професійного володіння комп’ютером та процедурно-орієнтована технологія розробки програмних продуктів під час вирішення прикладних завдань в області комп’ютерних наук.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ ПО ВИКОНАННЮ ЛАБОРАТОРНИХ РОБІТ

Вимоги до підготовки та захисту лабораторних робіт

Підготовка до виконання лабораторних робіти повинна починатися з вивчення лекційного матеріалу до відповідної теми та загальних відомостей (теоретичного матеріалу), наведених в методичних вказівках до виконання лабораторної роботі. Поглибити теоретичні знання можна використовуючи рекомендовані інформаційні джерела, представлені у даному посібнику та у відповідному курсі, представленому у системі дистанційного навчання університету.

Завдання до виконання лабораторної роботи складається із загального та індивідуального завдання. Загальне завдання є однаковим для всіх студентів, а індивідуальне завдання виконується згідно варіанту. Номер варіанту індивідуального завдання співпадає з порядковим номером прізвища студента в списку групи. Якщо студент виконує лабораторну роботу не за своїм варіантом, така робота не зараховується.

Текст програми повинен бути підготовлений по початку виконання лабораторної роботи в обчислювальній лабораторії. Рекомендується ретельно і багато разів перевірити алгоритм та правильність написання коду програми, оскільки це істотно скорочує загальний час розробки і відлагоджування програми.

За результатами виконання кожної лабораторної роботи студент формує звіт. Зарахування лабораторної роботи здійснюється після захисту звіту. Оцінка за виконання лабораторної роботи формується згідно РСО дисципліни. Складові оцінки:

10% - постановка задачі;

10% - блок-схема;

40% - код програми;

20% - оформлення звіту, висновки;

20% - захист роботи.

Захист роботи може здійснюватись в очному та дистанційному режимах. Під час захисту звіту про виконання лабораторної роботи у дистанційному режимі, студент під'єднується до Інтернет-конференції згідно розкладу занять. Захист лабораторної роботи починається з ідентифікації студента. Для цього, студент демонструє документ, що засвідчує його особу та власне зображення.

На момент захисту звіт про виконання лабораторної роботи, оформлений згідно вимог, повинен бути розміщений у системі дистанційного навчання. Звіт оформлюється засобами MS Word з використанням інструментів форматування.

Порядок виконання лабораторної роботи в обчислювальній лабораторії

До роботи у обчислювальній лабораторії студенти допускаються тільки після проходження інструктажу з охорони праці. Інструктаж проводиться відповідно до вимог Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці від 26.01.2005 № 15 щодо організації проведення інструктажів з питань охорони праці та наказів ректора Університету щодо проведення вступного інструктажу з питань охорони праці для здобувачів вищої освіти, зарахованих на перший курс, а також первинного інструктажу, проведеного безпосередньо у обчислювальній лабораторії. Повторний інструктаж проводиться згідно графіку.

Студенти, допущені до виконання лабораторної роботи в обчислювальній лабораторії можуть працювати використовуючи персональні комп'ютери як розташовані в обчислювальній лабораторії, так і власні. Студенти повинні використовувати наявну в лабораторії обчислювальну техніку виключно за призначенням: виконувати лабораторні роботи. Під час лабораторного заняття

студент опрацьовує код програми та формує звіт про виконання лабораторної роботи. Про закінчення роботи на персональному комп'ютері і повідомляє про це викладача.

Правила оформлення звіту про виконання лабораторної роботи

За результатами виконання лабораторної роботи студент формує звіт та розміщує його у системі дистанційного навчання.

Під час оформлення звіту про виконання лабораторної роботи слід дотримуватись стандарту ДСТУ 3008:2015 «Звіти у сфері науки і техніки. Структура та правила оформлювання».

Відповідно до п. 7 цього Стандарту:

7.1. Звіт викладають на паперовому та/чи електронному носієві (паперовий та електронний документи відповідно).

7.1.4 Символи в рівняннях і формулах, написи та пояснювальні дані на рисунках, схемах, графіках, діаграмах і в таблицях створюють і вводять у текст з використанням відповідних редакторів комп’ютерної програми.

7.1.5 Звіт друкують шрифтом Times New Roman чорного кольору прямого накреслення через півтора-два міжрядкові інтервали кеглем 14.

Розмір шрифту для написання заголовків у рядках і колонках таблиць і пояснювальних даних на рисунках і в таблицях встановлює виконавець звіту.

7.1.9 У звіті не бажано вживати іншомовних слів і термінів за наявності рівнозначних слів і термінів мови, якою подано звіт.

7.1.11 Рекомендовано на сторінках звіту використовувати береги такої ширини: верхній і нижній — не менше ніж 20 мм, лівий — не менше ніж 25 мм, правий — не менше ніж 10 мм.

7.1.12 Під час оформлювання звіту треба дотримуватися рівномірної насиченості, контрастності й чіткості зображення. Усі лінії, літери, цифри та знаки мають бути чіткі й нерозпливчасті в усьому звіті.

7.1.19 Для розділів і підрозділів наявність заголовка обов’язкова. Пункти

й підпункти можуть мати заголовки.

7.1.20 Заголовки структурних елементів звіту та заголовки розділів треба друкувати з абзацного відступу великими літерами напівжирним шрифтом без крапки в кінці. Дозволено їх розміщувати посередині рядка.

7.1.21 Заголовки підрозділів, пунктів і підпунктів звіту потрібно друкувати з абзацного відступу з великої літери без крапки в кінці.

7.1.22 Абзацний відступ має бути однаковий упродовж усього тексту звіту й дорівнювати п’яти знакам.

7.1.23 Якщо заголовок складається з кількох речень, їх розділяють крапкою. Розривати слова знаком переносу в заголовках заборонено.

7.1.24 Відстань між заголовком, приміткою, прикладом і подальшим або попереднім текстом має бути не менше ніж два міжрядкових інтервали. Відстань між основами рядків заголовка, а також між двома заголовками приймають такою, як у тексті звіту.

7.1.25 Не дозволено розміщувати назву розділу, підрозділу, а також пункту й підпункту на останньому рядку сторінки.

7.3.1 Сторінки звіту нумерують наскрізно арабськими цифрами, охоплюючи додатки. Номер сторінки проставляють праворуч у верхньому куті сторінки без крапки в кінці.

7.3.3 Титульний аркуш входить до загальної нумерації сторінок звіту. Номер сторінки на титульному аркуші не проставляють.

7.5.2 Рисунок подають одразу після тексту, де вперше посилаються на нього, або якнайближче до нього на наступній сторінці, а за потреби — в додатках до звіту.

7.5.6 Рисунки нумерують наскрізно арабськими цифрами, крім рисунків у додатках.

Дозволено рисунки нумерувати в межах кожного розділу. У цьому разі номер рисунка складається з номера розділу та порядкового номера рисунка в цьому розділі, які відокремлюють крапкою, наприклад, «Рисунок 3.2» — другий рисунок третього розділу.

7.5.7 Рисунки кожного додатка нумерують окремо. Номер рисунка додатка складається з познаки додатка та порядкового номера рисунка в додатку, відокремлених крапкою. Наприклад, «Рисунок В.1 — _____», тобто перший рисунок додатка В. назва рисунка

7.5.8 Якщо в тексті звіту лише один рисунок, його нумерують відповідно до 7.5.6.

7.5.9 Назва рисунка має відображати його зміст, бути конкретною та стислою. Якщо з тексту звіту зрозуміло зміст рисунка, його назву можна не наводити. За потреби пояснювальні дані до рисунка подають безпосередньо після графічного матеріалу перед назвою рисунка.

Назву рисунка друкують з великої літери та розміщують під ним посередині рядка, наприклад, «Рисунок 2.1 — Схема устаткування».

7.5.10 Рисунок виконують на одній сторінці аркуша. Якщо він не вміщується на одній сторінці, його можна переносити на наступні сторінки. У такому разі назву рисунка зазначають лише на першій сторінці, пояснювальні дані — на тих сторінках, яких вони стосуються, і під ними друкують: «Рисунок_____, аркуш _____ ».

7.6.1 Цифрові дані звіту треба оформлювати як таблицю відповідно до форми, поданої на рисунку 1.

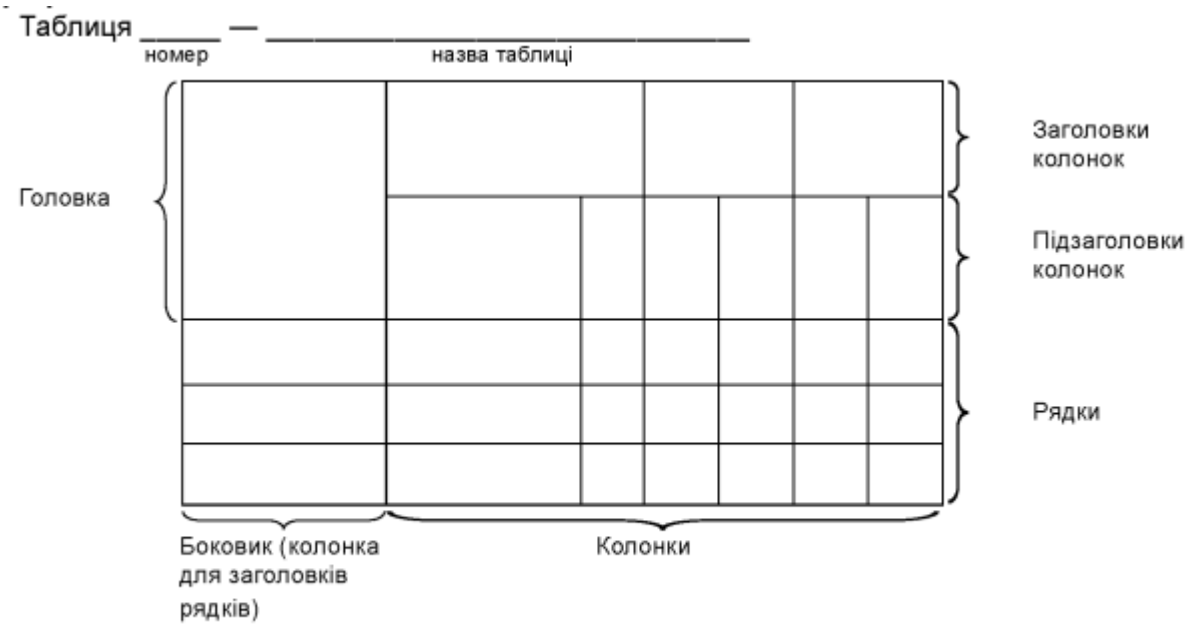


Рисунок 1

7.6.3 Таблицю подають безпосередньо після тексту, у якому її згадано вперше, або на наступній сторінці. На кожну таблицю має бути посилання в тексті звіту із зазначенням її номера.

7.6.4 Таблиці нумерують наскрізно арабськими цифрами, крім таблиць у додатках.

Дозволено таблиці нумерувати в межах розділу. У цьому разі номер таблиці складається з номера розділу та порядкового номера таблиці, відокремлених крапкою, наприклад, «Таблиця 2.1» — перша таблиця другого розділу.

7.6.10 Заголовки колонок таблиці починають з великої літери, а підзаголовки — з малої літери, якщо вони становлять одне речення із заголовком.

7.6.11 Підзаголовки, які мають самостійне значення, подають з великої літери. У кінці заголовків і підзаголовків таблиць крапки не ставлять. Переважна форма іменників у заголовках — однина.

7.10 Формули та рівняння

7.10.1 Формули та рівняння подають посередині сторінки симетрично тексту окремим рядком безпосередньо після тексту, у якому їх згадано.

Найвище та найнижче розташування запису формул та рівнянь має бути на відстані не менше ніж один рядок від попереднього й наступного тексту.

7.10.2 Нумерують лише ті формули та/чи рівняння, на які є посилання в тексті звіту чи додатка.

7.10.3 Формули та рівняння у звіті, крім формул і рівнянь у додатках, треба нумерувати наскрізно арабськими цифрами. Дозволено їх нумерувати в межах кожного розділу.

7.10.4 Номер формули чи рівняння друкують на їх рівні праворуч у крайньому положенні в круглих дужках, наприклад (3). У багаторядкових формулах або рівняннях їхній номер проставляють на рівні останнього рядка.

7.10.6 Пояснення познак, які входять до формули чи рівняння, треба подавати безпосередньо під формулою або рівнянням у тій послідовності, у

якій їх наведено у формулі або рівнянні.

Пояснення познак треба подавати без абзацного відступу з нового рядка, починаючи зі слова «де» без двокрапки. Познаки, яким встановлюють визначення чи пояснення, рекомендовано вирівнювати у вертикальному напрямку.

Приклад оформлення математичної формули:

Відомо, що

$$Z = \frac{M_1 - M_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}, \qquad (1)$$

де M_1, M_2 — математичне очікування;
σ_1, σ_2 — середні квадратичні відхили [23].

7.10.9 У формулах і рівняннях верхні та нижні індекси, а також показники степеня, в усьому тексті звіту мають бути однакового розміру, але меншими за букву чи символ, якого вони стосуються.

7.10.10 Переносити формули чи рівняння на наступний рядок дозволено лише на знаках виконуваних операцій, які пишуть у кінці попереднього рядка та на початку наступного. У разі перенесення формули чи рівняння на знакові операції множення застосовують знак «х ». Перенесення на знаку ділення «:» слід уникати.

7.10.11 Кілька наведених і не відокремлених текстом формул пишуть одну під одною і розділяють комами.

Приклад

$$f_1(x, y) = S_1, \qquad (29)$$

$$f_2(x, y) = S_2 \qquad (30)$$

Для малювання блок-схем використовують ГОСТ 19.701-90 (ИСО 5807-85) «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения». Співвідношення розмірів блоків: *b=1,5a*

Звіт про виконання лабораторної роботи повинен складатися з таких частин:

1. Аналіз умови задачі. Містить:
   - Підзадачі, які можна виділити з загальної задачі, представленої у індивідуальному завданні
   - Визначення необхідних вхідних змінних. Треба вказати скільки їх, якого типу, яке ім’я змінній присвоєне. Тип даних слід вибирати виходячи з умови задачі, ідентифікатор – таким чином, щоб відображало логічний сенс змінної.
   - Визначення вихідних змінних програми, а саме їх тип, ідентифікатори, тощо. Визначення переліку допоміжних змінних, якщо це потрібно.
   - Визначення, які змінні є локальними, які глобальними.
   - Визначення формул для обчислення значень, якщо це необхідно.
   - Вказання алгоритмів сортування, пошуку, роботи з масивами та даними інших типів
   - Опис користувацьких функцій із вказанням фактичних і формальних параметрів.
2. Блок-схема (або псевдокод) розв’язку задачі.
3. Код програми.
4. Скріншот результатів роботи програми.
5. Висновки.

ЛАБОРАТОРНА РОБОТА №1
ОПАНУВАННЯ НАВИЧОК РОЗРОБКИ ПРОГРАМ МОВОЮ С. ПРОГРАМУВАННЯ БАЗОВИХ АЛГОРИТМІЧНИХ КОНСТРУКЦІЙ

Мета роботи: ***оволодіти навичками*** розв’язання прикладної задачі на комп'ютері, зокрема, ***розробки алгоритму обчислювального процесу із використанням базових алгоритмічних конструкцій***

ТЕОРЕТИЧНІ ВІДОМОСТІ

Робота з розв’язання прикладної задачі на комп'ютері передбачає виконання:

1) постановки задачі;
2) математичну формалізацію;
3) побудову алгоритму;
4) складання програми на мові програмування (кодування);
5) налагодження і тестування програми;
6) проведення розрахунків і аналіз одержаних результатів.

При розробці алгоритму слід пам’ятати, що властивостями алгоритму є:

- визначеність (детермінованість) – усі інструкції алгоритму повинні мати однозначне трактування і бути зрозумілими виконавцю алгоритму, тобто при однакових вхідних даних повинні бути одержані однакові результати;
- дискретність – алгоритм повинен представляти процес вирішення задачі як послідовне виконання простих кроків (кожна передбачена алгоритмом дія виконується тільки після того, як буде виконана попередня дія);

- масовість – дозволяє знайти рішення для цілої групи задач, що відрізняються вхідними даними. При вирішенні унікального завдання цю властивість можна проігнорувати;
- результативність – робота алгоритму повинна закінчуватись за кінцеве число кроків, хоча деякі кроки можуть повторюватись багаторазово. Алгоритм повинен досягти поставленої мети і не може перериватися через непередбачену ситуацію;
- ефективність – усі кроки алгоритму повинні бути закінчені за певний час, а загальний час роботи - не перевищувати допустимий для вирішення цього завдання.

Для опису алгоритму можна використовувати різні способи, зокрема:

1) словесний – кожну дію алгоритму описують словами;

2) графічний – алгоритм представляють у вигляді рисунків;

3) табличний – кроки алгоритму записують у вигляді таблиці;

4) блок-схеми – опис алгоритму за допомогою геометричних фігур, кожна з яких відповідає за виконання певної дії алгоритму.

В рамках даного курсу пропонується використання блок-схем алгоритмів. Блок-схеми алгоритмів повинні відповідати ДСТУ-19.701-90 (ISO 5807-85).

Не зважаючи на те, що написання коду програми на мові С не регламентується, а стандарт мови допускає вільний запис, все ж слід дотримуватись певної структури програми. Структура програми має забезпечувати функціонування програмних блоків як єдиного цілого та бути зрозумілою іншим програмістам. Тому, код програми, повинен бути структурований таким чином, щоб всі компоненти програми: бібліотеки, класи, модулі, структури, тощо, були згруповані у відповідні програмні блоки. Код програми мовою С зазвичай складається з блоків:

1. директиви препроцесора
2. оголошення глобальних змінних, функцій, класів, структур

3. визначення функцій
4. головна функція (точка входу в програму) та її код
5. визначення методів класів, функцій.

Директиви препроцесора визначають дії з перетворення тексту початкової програми перед компіляцією. За їх допомогою також здійснюється об'ява іменованих констант, що використовуються у всій програмі, підключення бібліотек, інших файлів, тощо. Директиви – команди, які активують роботу компілятора. Препроцесор «готує» програму до компіляції. Директива препроцесора є першим рядком програми і починається з символу «#» та директиви include. Ця директива слугує для підєднання заголовних файлів системи програмування та (або) файлів користувача. Імя файлу, що підключається може вміщуватись або в трикутні дужки(якщо файл є частиною системи програмування та розташовується у чітко визначеній папці, як правило, вона називається INCLUDE) або в лапки (переважно це файли, розроблені користувачем, вони розташовуються у робочій папці, частіше за все в тій, де розміщуються файли програми).

Машинно-залежні оператори мови С реалізуються за допомогою функцій. Основні функції, які необхідні для роботи С, зведені в бібліотеки, які забезпечують, зокрема виконання операцій введення виведення (заголовний файл stdio.h), .використання математичних функцій (наприклад, заголовний файл math.h), тощо. Виклик необхідної для використання в конкретній програмі групи функцій здійснюється за директивою препроцесора,

Вказівки компілятору – це спеціальні інструкції для компілятора мови С (можуть відрізнятися для різних компіляторів). Компілятор перевіряє синтаксис програмного коду, його відповідність семантиці мови, здійснює перетворення тексту коду на машинну мову та створює об’єктний файл, який містить код програми на машинній мові.

Оголошення глобальних змінних, функцій, класів, структур здійснюється для того, щоб повідомити про те, які саме дані, класи та структури використовуються в програмі.

На мові С програма являє собою одну або декілька підпрограм, які називають функціями. Слід зазначити, що на відміну від інших мов програмування, в мові С не розрізняють підпрограми на процедури та функції, всі підпрограми називають функціями. **Функції** описують сукупність дій, які потрібно виконати в програмі. Для того, щоб програма на мові С була скомпільована і виконана, вона повинна містити, принаймні, одне визначення функції.

Одна з функцій має ім'я «main» та є головною. Вона виконується завжди першою. Якщо програма містить тільки одну функцію, то вона і є головною (і повинна мати ім'я **`main`**). Залежно від особливостей компілятора, функція **`main`** повертає ціле значення (зазвичай при нормальному завершенні програми у виразі в операторові **`return`** задається число **`0`**) або значення типу void. У функцію **`main`** не передаються ніякі аргументи.

Важливо пам’ятати, що в тексті програми можуть бути використані не всі перераховані блоки, але порядку їх розташування в коді необхідно дотримуватись. Це не лише сприяє прискоренню компіляції програми, а й полегшує перевірку коду програми та виправлення помилок.

Виконувані оператори розташовуються в середині функцій та визначають дії програми з реалізації алгоритму її роботи. Ознакою закінчення операторів оголошень і виконуваних операторів є символ "**`;`**".

Оголошення змінних, як правило, здійснюється на початку коду програми, але стандарт мови дозволяє оголошувати змінні й за їх безпосереднього використання. Слід пам’ятати, що всі змінні повинні бути описані (оголошені) до їх використання.

При оголошенні змінних та констант їм задають імена (ідентифікатори) і атрибути змінної, **функцій -** їх імена, типи значень, що повертають функції і атрибути формальних параметрів. Оголошення функцій (їх називають також **прототипами функцій**) можуть знаходитися тільки поза визначеннями функцій.

Тип змінних – множина значень, які може набувати змінна, формат представлення даних в пам'яті комп'ютера та сукупність операцій над ними, обирається серед стандартних типів або може створюватись користувачем.

Ідентифікатор (ім'я) – послідовність літер, знаків підкреслювання та цифр, що починається не з цифри (регістр в C/C++ розрізняється!), яка служить для іменування змінних та констант.

Змінні можуть бути оголошені як поза визначеннями функцій, так і всередині функцій або блоків. У першому випадку змінні називаються **глобальними змінними** або **змінними верхнього рівня**, оскільки область дії змінної в цьому випадку (її називають також **зоною видимості змінної)** обмежується частиною програми від точки оголошення змінної до кінця файлу. У другому випадку, змінні називаються локальними **змінними** або змінними **нижнього рівня.** Зона видимості локальних змінних – функція або блок, в якому вони визначені. У С всі локальні змінні повинні бути оголошені до їх використання в середині виконуваного оператора, функції або блоку.

Константа – це лексема, що являє собою зображення фіксованого числового, рядкового чи символьного значення. Для оголошення констант використовується специфікатор **const.**

Мова С статично типізована, це означає, що змінна, параметр функції, значення, що повертає функція мають тип даних, який заданий у момент оголошення і який пізніше змінити не можна.

Для проведення розрахунків, в програмі також можуть бути використані операції та вирази. Операція – це дія, яку виконують над одним,

двома або трьома операндами. Її називають, відповідно, унарна, бінарна або тернарна операції. Результатом виконання операції є числове значення. Над об'єктами в мові С можна виконувати різні операції, наприклад присвоювання, відношення тощо. **Вираз** – це інструкція мови, яку сприймає компілятор як керівництво до певних дій над даними, наприклад, c=a+2; return 0; printf("1\t2\n3\t\b4"); тощо.

У виразах можуть бути використані й математичні функції мови С, які описані в заголовному файлі **math.h** Математичні функції бібліотеки виконують деякі арифметичні дії, результат яких присвоюється імені функції. Математичні функції мають наступний формат виклику:

ім'я-функції **(***вираз***)**

де *вираз* – будь-який арифметичний вираз. Тип значення, що повертається даною функцією – **double**.

Математичні функції, як і змінні, можуть використовуватися в арифметичному виразі, причому обчислення функції здійснюється відповідно до їх пріоритету.

В різних середовищах програмування після виконання оператора **return** вікно виконання програми може автоматично закриватися. Щоб цього не відбувалося, зазвичай перед оператором **return** записують оператори **getchar(); або system("PAUSE");** який викликає функцію введення символу. В цьому випадку вікно виконання програми закривається при введенні будь-якого символу (або просто натиснення клавіші «**Enter**»).

Іноді програма вже очікує на введення символів. В цьому випадку необхідно двічі повторити виклик функції **getchar()**. Також замість оператора **getchar();** можливо використовувати оператор **system("PAUSE");**

Функції введення-виведення визначені в заголовному файлі **stdio.h**, який «відповідає» за використання бібліотеки введення-виведення, затвердженої ANSI як стандарт (тому дану бібліотеку називають **стандартною бібліотекою** мови С).

Мова С не має стандартних операторів для введення і виведення інформації. У будь-якій програмі, написаній на мові С, обов'язково застосовують заголовний (бібліотечний) файл з ім'ям <stdio.h> (standard input/output header). Він містить готові функції для організації стандартного введення та виведення.

Функції, оголошені в stdio.h умовно можна поділити на:

- функції для операцій введення з клавіатури і виведення на екран монітору
- функції для операцій з файлами.

Зокрема, функція форматованого виведення **printf()** стандартної бібліотеки мови С формує і виводить значення, що задаються аргументами, на дисплей у вигляді символьних рядкиів. Функція має змінну кількість аргументів і записується у вигляді:

printf("*рядок-формату"* [**,** *аргумент-1* [**,** *аргумент-2*...]]**);**

де *рядок-формату* – символьний рядок, що визначає вид (формат) даних, що виводяться. Цей рядок складається із звичайних символів, спеціальних символів і, якщо за рядком формату слідує один або декілька аргументів, специфікацій полів формату виводу (поодинці для кожного аргументу).

Проста специфікація формату містить тільки символ "**%**" і символ, що позначає специфікатор типу даних, що виводиться. Позначення основних специфікаторів типів даних представлені в табл. 1.1.

Таблиця 1.1 – Основні специфікатори типів даних

%c – символ %d – ціле десяткове число %i – ціле десяткове число %o – цілое восьмеричне число %x – целое шістнадцядкове число (a1) %X – ціле шістнадцяткове число (A1) %u – беззнакове десяткове	%f – дійсне число xx.xxx %F – дійсне число xx.xxx %e – дійсне число x.xx e+ xx %E – дійсне число x.xx E+ xx %G – %F або %E (що компактніше) %g – %f або %e (що компактніше) %s – рядок символів %p – вказівник %%– символ %

Якщо число специфікацій формату більше числа аргументів, зайві специфікації ігноруються. У зворотному випадку результат виводу не визначений.

Вхідний або вихідний послідовний набір інформації, який може бути наданий як файлом, так і пристроєм, називають потоком. Потік може бути текстовим та бінарним. Текстовий потік – це послідовність символів ASCII.

Бінарний потік – послідовність байтів, які можуть містити будь-яку інформацію (в тому числі і текст).

На початку роботи програми відкрито наступні потоки:

- stdin (клавіатура) – зарезервований для читання команд користувача або вхідних даних;
- stdout (екран) – зарезервований для виведення даних;
- stderr (екран) – зарезервований для виведення діагностичних і налагоджувальних повідомлень у текстовому вигляді.

Функція **`printf()`** повертає ціле число типу **`int`** – кількість виведених символів або негативне значення, якщо виникла помилка введення-виводу, проте, як правило, повертане значення цієї функції не використовується.

Приклади використання функції printf:

Приклад 1. **`printf("\nВведіть x:");`**

Буде виведено:

```
Введіть x:
```

Приклад 2.
```
printf("\nx=%d y=%f z=%e",x,y,z);
```

Якщо значення *x=5*, *y=0,8* і *z=-0,5*, буде виведено:

```
x=5 y=0.8 z=-0.5e+00
```

Приклад 3.
```
printf("\nЗавдань: %d\nЗ них правильно вирішено:
%d (%f%%)",
        total,right_answered,(float)
    right_answered*100/total);
```

Якщо значення *total=12*, *right_answered=4*, буде виведено:

```
Завдань: 12
З них правильно вирішено: 4 (33.333333%)
```

Приклад 4.
```
int outerror;
        outerror = printf("\nsum=%d", sum);
        if(outerror < 0)  /* Якщо помилка при виведенні sum */
         return 1;     /* вихід з програми */
```

Приклад 5.
```
printf("Hello, World!\n");
```

Буде виведено `Hello, World!`

Приклад 6.
```
printf("x=%f, y=%f\n",x,y);
```

Буде виведено x = <значення змінної> x y= <значення змінної> y

Приклад 7.
```
printf("%05d", 15);
```
// Надрукує *00015*

Команда форматованого введення scanf призначена для введення інформації користувачем за допомогою клавіатури. Ця функція, як і функція `printf()` належить до команд форматованого введення/виведення**,** що знаходиться в розділі `stdio.h` стандартної бібліотеки С. У загальному випадку, команда має вигляд:

```
scanf("рядок-формату" , &[аргумент-1],& [  аргумент-2], &[...]);
```

де: «*рядок формату*» − рядок специфікаторів формату у подвійних лапках. Застосовуються ті ж специфікатори, що й для **`printf,`** *аргумент* − це ім’я змінної, значення якої вводиться. знак & є елементом синтаксису і означає операцію отримання адреси змінної у пам’яті.

Приклад 8. scanf("%f", &a); //введення значення змінної а дійсного типу

Приклад 9. scanf("%i %f", &x, &y) // введення значень змінної х цілого типу та змінної у дійсного типу

При роботі програми дії виконуються послідовно. Проте, часто буває необхідно змінити порядок їх виконання. Для цього використовуються оператори, що передають управління. У мові С існують наступні види операторів передачі управління:

- умовний оператор;
- оператори циклу;
- оператор вибору;
- оператор безумовного переходу.

Базова алгоритмічна конструкція «розгалуження» може бути використана в повній та неповній формі (рис. 1.1).

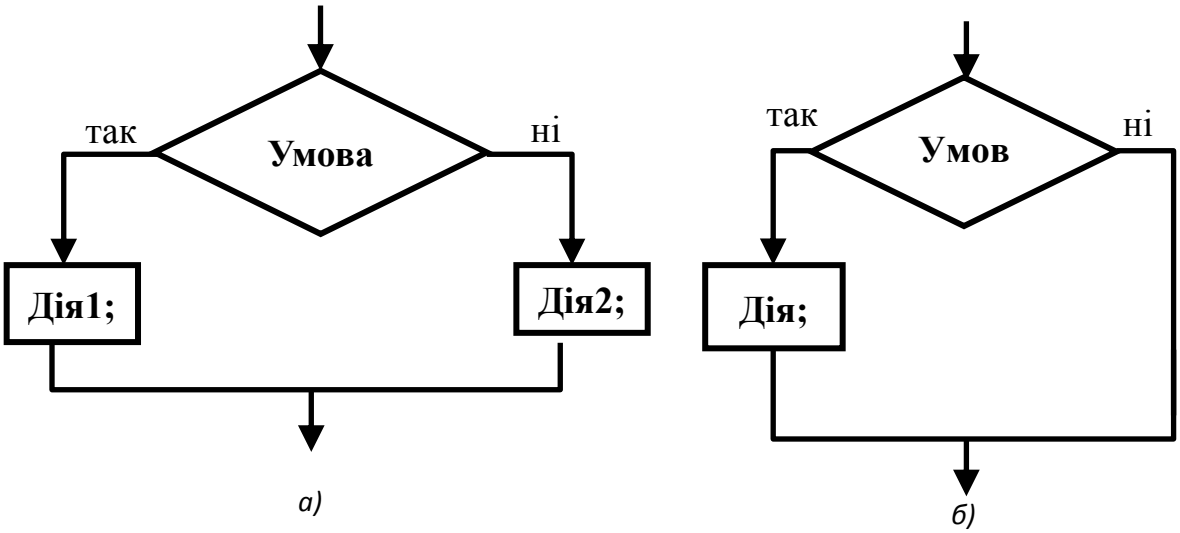


Рисунок 1.1 – Блок-схеми алгоритмічної конструкції «розгалуження» (а) повна форма, б) не повна форма)

Умовний оператор (або розгалуження) if виконує оператор або блок операторів залежно від значення вираження в умові. Може мати повну та неповну форми.

Оператор має наступний формат:

```
if (вираз_1)
        блок_оператора_1;
else if (вираз_2)
        блок_оператора_2;
...
else
        блок_операторів_n
```

де: *вираз_1 і вираз_2 - будь-які вирази, які можуть бути оцінені як "істина" або "хибність"; блок-операторов-1, блок-операторов-2 і блок-операторов-n – це один оператор або декілька операторів (блок операторів має бути поміщений у фігурні дужки).*

При реалізації розгалуження спочатку обчислюється вираз_1. Якщо значення виразу - ненульове ("істина"), то виконується блок-операторов-1 і відбувається перехід до виконання оператора, що йде за умовним оператором. Якщо вираз_1 дорівнює 0 ("хибність"), робиться перевірка виразу_2, і, якщо воно істинне, то виконуватиметься блок-операторов-2 і виконання умовного оператора закінчується і т. д. Блок-операторов-n виконується, коли усі попередні умови не виконуються. Оператори else if і else можуть бути опущені. В цьому випадку, якщо значення виразу_1 - "хибність", блок-операторов-1 не виконується і відразу виконується оператор, наступний в програмі за оператором if.

Приклади використання оператора if .. else :

Приклад 10.

```
if (value1 == 0)
```

```
            printf("Нульове значення\n");
    else
            value1=value1 - 1;
```

Приклад 11.

```
        if (value1 == 0)
                sign_value=0;
    else if (value1 > 0)
            sign_value=1;
    else
    {
            sign_value = - 1;
            printf ("Від'ємне значення");
    }
```

Оператор множинного вибору (оператор-перемикач)

Оператор-перемикач switch використовується для вибору одного з декількох шляхів виконання програми (рис. 1.2).

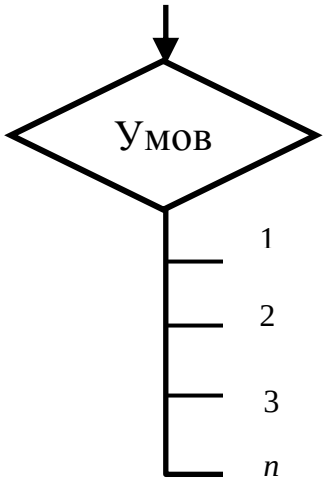


Рисунок 1.2 – Блок-схеми алгоритмічної конструкції «вибір» («перемикач»)

Оператор має наступний формат:

```
switch (вираз)
```

```
{
    case константний_вираз_1: блок_операторів
    case константний_вираз_2: блок_операторів
    ...
    ...
    case константний_вираз_n: блок_операторів
    default: оператори
}
```

«Вираз» та усі константні-вирази мають бути цілочисельними або мати тип char.

Виконання оператора починається з обчислення виразу. Потім послідовно перевіряються усі гілки **case**. Якщо значення виразу співпадає з одним з константних-виразів_і., виконуються усі оператори_і, що стоять за цим оператором case (якщо вихід із перемикача явно не вказаний за допомогою оператора **break**) і виконання оператора завершується. Інакше перевіряється наступний оператор case і так далі. У разі, якщо жоден з операторів case не задовольняє умови, виконується оператор default. Число операторів case і константних виразів в операторові switch може бути будь-яким. Оператор **default** може бути опущений.

Приклад 12. Використання оператора switch:

Необхідно визначити кількість літер «A» і «a», а також підрахувати загальну кількість літер в тексті:

```
char c;
...
switch (c)
{
    case 'A ':
        nl_A++;
```

```
    case 'a ':
            nl_a++;
    default:
            nl_total++;
}
```

Якщо значення **c** дорівнює 'A', то виконуються усі три оператори (збільшення значення лічильників nl_A, nl_a і nl_total на 1). Якщо значення **c** дорівнює 'a', то виконуються останні два оператори (збільшення значення лічильників nl_a і nl_total на 1). Якщо значення **c** не дорівнює ні 'A ', ні 'a ', то виконується тільки останній оператор (збільшення значення лічильника nl_total на 1). Як видно з аналізу програми, вона працює не зовсім правильно, оскільки, якщо значення **c** дорівнює 'A ', то значення лічильника символу 'a' - nl_a також збільшується на 1. В даному випадку треба використовувати оператор **break.**

Організація переривання виконання програми

Для дострокового завершення виконання операторів циклу for, while і do…while, а також оператора switch використовується оператор **break**. Цей оператор викликає негайний вихід з самого внутрішнього з циклів, що охоплюють його, або операторів switch (і оператори циклів і оператори switch в мові С можуть бути вкладеними один в одного).

Приклад 13. Використання оператора break

Виконання циклу припиняється, як тільки значення суми перевищить 50:

```
    int sum, i, a1;              /* Оголошення змінних */
    for (i=0, sum=0, a1=5; i < 10; i++) /* Визначення
циклу */
    {
        sum += (a1+i);           /* Підсумовування елементів
     */
```

```
    if (sum > 50) /* Якщо значення суми перевищує 50
 */
         break;            / * Вихід з циклу */
}
printf ("Сума =%d\n", sum);    /* Виведення суми */
```

Оператор break часто використовує як останній оператор в гілках case оператора вибору switch для того, що перервати перегляд наступних гілок вибору і вийти з оператора switch.

Приклад 14. Використання оператора switch з оператором break

Для визначення кількості літер «A» і «a», а також підрахунку загальної кількості літер в тексті можна застосувати оператор switch та організувавши переривання виконання програми за допомогою оператора break:

```
char c;
...
    nl_total++;
    switch (c)
{
    case 'A ':
         nl_A++;
             nl_total++;
             break;
 case 'a ':
         nl_a++;
             nl_total++;
             break;
     default:
         nl_total++;
}
```

Якщо значення **c** дорівнює 'A', виконується перша гілка оператора, потім вібдудеться вихід з оператора. Якщо значення **c** дорівнює 'a ', то буде виконана

друга гілка, а якщо значення **c** не дорівнює ні 'A ', ні 'a ', то виконується тільки гілка default.

Оператор продовження **continue** викликає перехід на наступну ітерацію, зокрема в операторах циклу. При цьому оператори тіла циклу, що йдуть за оператором continue, не виконуються. У операторах циклу while і do..while наступна ітерація починається з обчислення умовного виразу, а для оператора for - з обчислення виразу приросту, а потім умовного виразу.

Приклад 15. Використання оператора continue:

Розрахувати суму непарних членів арифметичної прогресії.

```
int sum, i, a1;              /* Оголошення змінних */
for (i=0, sum=0, a1=5; i < 10; i++) /* Визначення циклу */
{
    if ((a1 +i) % 2)  /* Якщо число - парне */
        continue;        /* Не виконувати додавання */
    sum += (a1+i);    /* Додавання елементів */
}
printf ("Сума =%d\n", sum);    /* Виведення суми */
```

Оператор безумовного переходу goto

Оператор переходу goto, формат, що має, :

```
goto мітка;
```

передає управління оператору в тій же функції, поміченому міткою, за якою йде символ "**:**". Мітка є звичайним ідентифікатором мови С.

Можна увійти до блоку, тіла циклу, умовного оператору і оператору-перемикача по мітці, проте не можна передати управління оператори case або default в тілі оператора-перемикача.

Приклад 16. Використання оператора goto:

Розрахувати суму непарних членів арифметичної прогресії.

```
int sum=0, i=0, a1=5;      /* Оголошення змінних */
sum_loop:                  /* Мітка циклу */
sum += (a1+i);             /* Додавання елементів */
i++;                       /* Збільшення індексу на 1 */
if (i < 10)               /* Якщо значення індексу менше 10 */
      goto sum_loop;     /* перехід на мітку sum_loop */
}
printf ("Сума =%d\n", sum);    /* Виведення суми */
```

Хоча використання оператора goto не вважається хорошим стилем в програмуванні, його по-можливості слід уникати, проте в деяких випадках цей оператор може стати в нагоді. Наприклад, якщо необхідно здійснити вихід з найглибшого вкладеного циклу за межі зовнішнього циклу.

До операторів переходу належить і оператор повернення **return**.

У С реалізовані операторів циклу, які дозволяють використовувати всі циклічні алгоритмічні конструкції:

оператор покрокового виконання циклу for;

оператор виконання циклу з перевіркою умови до початку виконання операторів тіла циклу (цикл while);

оператор виконання циклу з перевіркою умови по закінченню виконання операторів тіла циклу (цикл do … while).

Оператор покрокового виконання циклу (цикл з лічильником) for має формат:

for (початковий_вираз; умовний_вираз; вираз_прирощування)

<блок операторів тіла циклу>

На блок-схемі цикл з лічильником зображується як представлено на рис. 1.3.

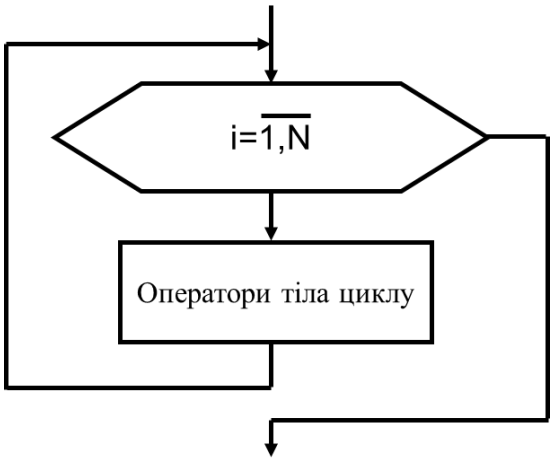


Рисунок 1.3 – Зображення циклу з лічильником за допомогою блок-схеми

Першим кроком при виконанні циклу є обчислення початкового_виразу (якщо воно є). Потім – блок операторов: один оператор або декілька операторів, поміщених у фігурні дужки, виконується до тих пір, поки умовний_вираз не матиме нульового значення ("хибність"). При виконанні тіла циклу може використовуватися вираз_прирощування (якщо воно є).

Будь-який з трьох виразів може бути відсутнім, але символи ";" опускати не можна. За відсутності початкового_виразу і виразу_прирощування, вважається, що їх просто немає в цьому циклі, за відсутності умовного_виразу, вважається, що його значення завжди не дорівнює 0 ("істина").

Приклад 17. Використання оператора for

Обчислити суму натуральних чисел від 5 до 14

```
    int sum = 0;        /* Початкове значення суми */
    int a = 5;          /* Значення першого члена прогресії
*/
    int i;              /* Індекс циклу */
    for (i=0; i < 10; i++)     /* Визначення циклу */
```

```
        sum += (a+i);          /* Підсумовування елементів
    */
    printf ("Сума =%d\n", sum);    /* Друк суми */
```

Операція кома зв'язує два вирази в один і гарантує, що вираз, що знаходиться ліворуч, обчислюватиметься першим. Ця операція має синтаксис:

вираз-1, вираз-2, ..., вираз-n

Значення усього виразу відповідає значенню виразу, що знаходиться справа, наприклад у виразі: `x = (y = 3, (z = ++y + 2) + 5);` спочатку змінній *y* присвоюється начення 3, потім значення *y* збільшується до 4, далі до 4 додається до 2, а отриманий результат (рівний 6) присвоюєтья змінній **z**, далі до **z** додається 5, і, нарешті, змінній **x** присвоюється значення 11**.**

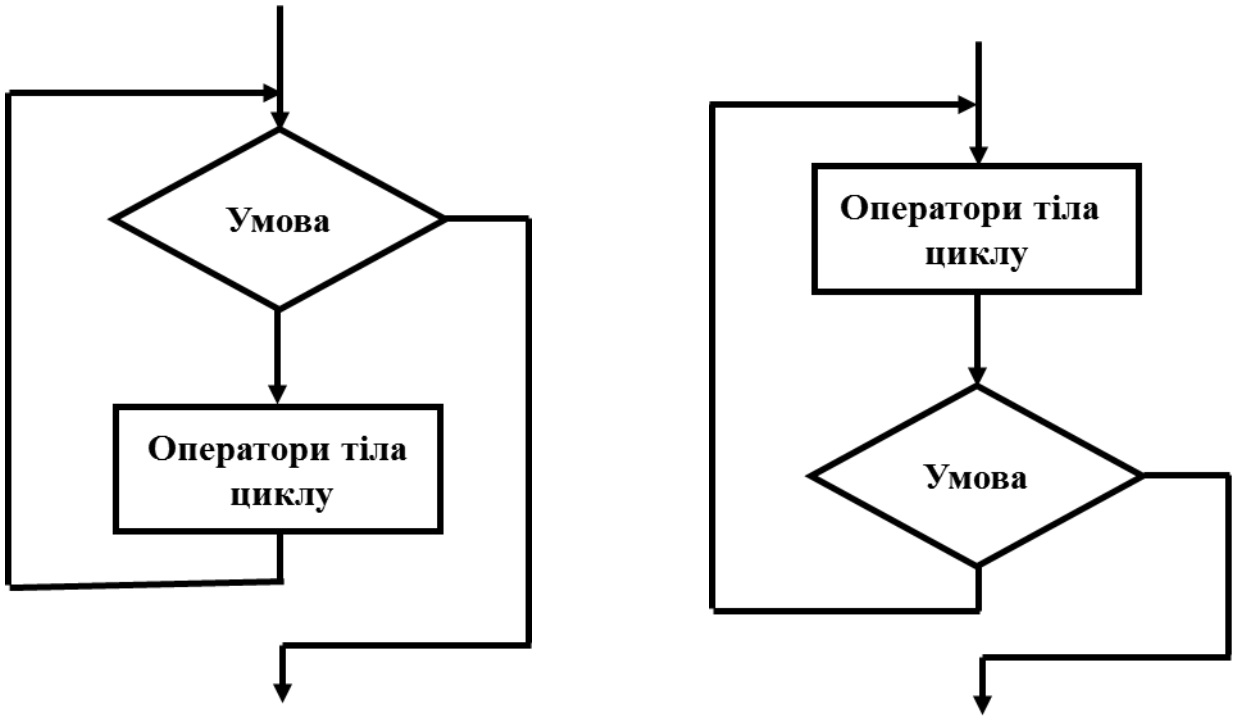
Зазвичай операція кома використовується для включення додаткової інформації в циклі for.

Приклад 18. Використання операції кома в операторі for :

Обчислити суму натуральних чисел від 5 до 14

```
    int sum, i, a;              /* Оголошення змінних */
    for(i=0, sum=0, a=5; i < 10; i++)/* Визначення циклу
*/
        sum += (a+i);     /* Підсумовування елементів */
    printf("Сума =%d\n", sum);     /* Друк суми */
```

Представлення циклічних алгоритмічних конструкцій циклу з попередньою перевіркою умови та циклу з перевіркою умови по закінченню виконання операторів тіла циклу за допомогою блок-схем наведене на рис. 1.4.



а) *б)*

Рисунок 1.4 – Блок-схеми циклів з попередньою перевіркою умови (а) та перевіркою умови після виконання операторів тіла циклу (б)

Оператор циклу з передумовою while має формат:

```
while (умовний_вираз)
    блок-операторів (тіло циклу)
```

Оператор циклу з пост-умовою do…while має формат:

```
do

 блок-операторів (тіло циклу)
while (умовний_вираз);
```

Блок-операторів - один оператор або декілька операторів (у фігурних дужках) виконуються доти, доки **умовний_вираз** не матиме значення 0 ("не істина"). Відмінність циклу while від циклу **do..while** в тому, що в циклі

while значення умовного_-виразу обчислюється до початку виконання блока-операторів, а в циклі **do..while** - потім, тобто цикл **do..while** завжди виконується хоч би один раз.

Приклад 19. Використання оператора while :

Обчислити суму натуральних чисел від 5 до 14

```
int sum = 0;        /* Початкове значення суми */
int a = 5;          /* Значення першого члена прогресії
*/
int i = 0;          /* Індекс циклу */
while (i < 10)      /* Визначення циклу */
{
     sum += (a+i);      /* Додавання елементів */
     i++;               /* Збільшення індексу на 1*/
}
printf ("Сума =%d\n", sum);    /* Друк суми */
```

Приклад 4. Відмінності у роботі операторів while і do - while:

```
int sum = 0, i = 1;
while (i < 1)
{
sum+=i;
    i ++;
 }
```

```
int sum = 0, i = 1;
do
{
sum+= i;
    i ++;
}
while (i < 1);
```

Цикл, організований із використанням конструкції **while** не виконуватиметься жодного разу і в результаті, значення **sum** та **i** не зміняться.

Цикл, організований із використанням конструкції **do**… **while** буде виконаний один раз і в результаті значення **sum** дорівнюватиме 1, а значення **i** дорівнюватиме 2.

Загалом, найскладнішими конструкціями, використовуваними в програмі

є саме циклічні. Адже вони забезпечують отримання результату шляхом багаторазового повторення деякої послідовності дій (операторів тіла циклу) і потребують особливої уваги щодо правильної організації завершення роботи циклу. Розрізняють цикли із заданим і невідомим числом повторень. До останніх належать ітераційні цикли. В ітераційних циклах вихід здійснюється не після того, як оператори тіла циклу виконались задану кількість разів, а якщо виконано загальну умову, пов'язану з перевіркою значення, яке монотонно змінюється в циклі.

Прикладом застосування ітераційного циклу є задача визначення суми (добутку) деякої послідовності із заданою точністю, тобто, оператори тіла циклу виконуватимуться до тих пір, поки не буде досягнуто виконання умови досягнення необхідної точності розрахунку на деякому кроці ітераційного процесу, що реалізується алгоритмом.

Приклад виконання завдання

Завдання: Дано значення аргументів x та y. Вивести на екран результат обчислень, виконаних за формулою (1.1):

$$F = \frac{(x^3 + 1)}{\ln|x|} - \sin^2\left(\frac{\pi}{4} + 0.5y\right) \qquad (1.1)$$

при *x=-0.951* і *y=8.149*.

Аналіз умови задачі

Згідно умови задачі необхідно розрахувати значення змінної F за заданою формулою. Число π оголошується як константа, тип даних double (дійсний подвійної точності). Для реалізації функції виведення необхідно приєднати бібліотеку stdio, для використання математичних функцій – синус, логарифм, піднесення до степеня - math.h.Змінні, використовувані для розв’язку задачі представлені в табл. 1.2.

Таблиця 1.2 – Перелік змінних, використовуваних у програмі

Ідентифікатор	Тип даних	Призначення
F	дійсне число (float)	результат розрахунку
x	дійсне число (float)	аргумент
y	дійсне число (float)	аргумент

Значення змінних задані в умові задачі, тому з клавіатури нічого вводитись не буде, всі значення змінних та констант будуть задані під час їх ініціалізації. На екран буде виведено результат розрахунку – значення змінної F та значення аргументів – x та y. Порядок виконання дій під час розрахунку значення F – послідовне виконання розрахунків.

Блок-схема алгоритму представлена на рис. 1.1:

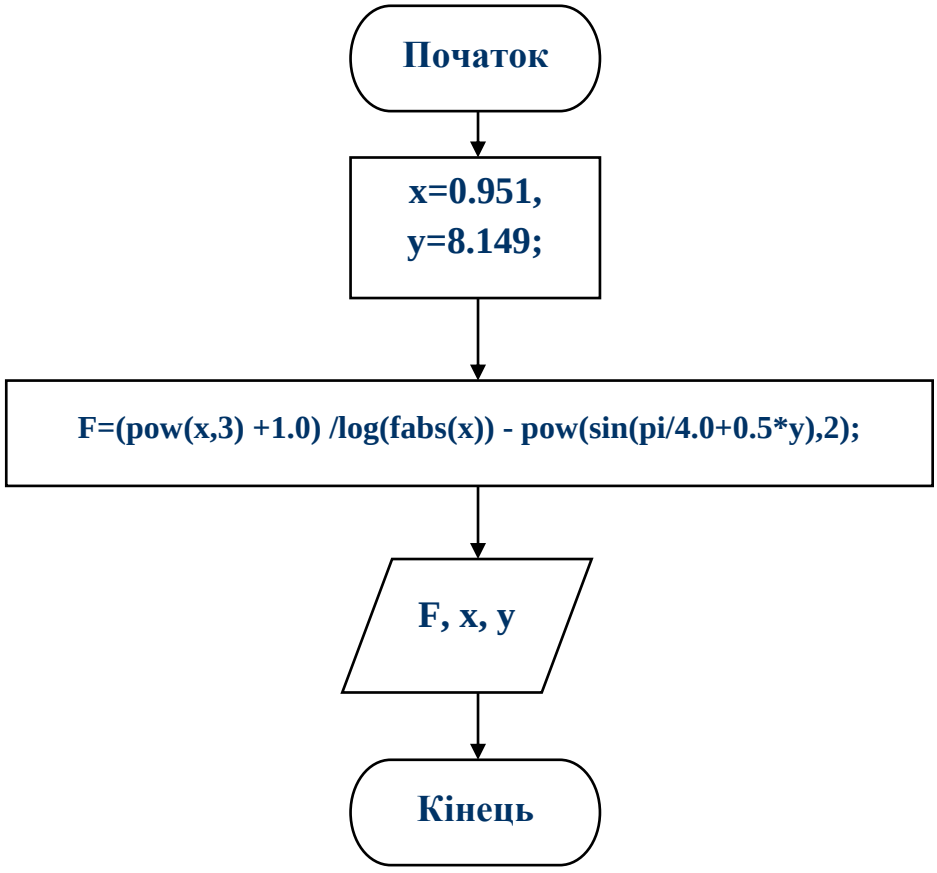


Рисунок 1.1 – Блок схема алгоритму розрахунку значення функції F

Код програми мовою С:

```
#include <stdio.h> /*Виклик головного модуля функцій введення-виводу */
#include <math.h> /* Виклик головного модуля математичних функцій */
void main() /* Опис головної функції */
{ /* Початок блоку */
 float F, x, y; /* Опис змінних */
  x=0.951, y=8.149; /* Ініціалізація змінних */
    const double pi=3.1415926; /* Описи та ініціалізація константи */
 F=(pow(x,3) +1.0) /log(fabs(x)) - pow(sin(pi/4.0+0.5*y),2); /* Обчислення значення функції */
 printf("x=%f y=%f Функція F=%f\n",x,y,F); /* Друк вхідних даних і результату */
} /* Кінець блоку */
```

Реузьтат роботи програми:

```
x=0.951000 y=8.149000 Функція F=-38.001472
```

Висновки. В результаті виконання лабораторної роботи розроблено програму розрахунку значення функції за заданими значеннями аргументів. Побудовано алгоритм роботи програми та реалізовано його мовою програмування С. У програмі використано математичні функції та засоби форматованого виведення мови С. Програма працює вірно, правильність розрахунків перевірено, розрахунки за даною формулою виконано й у MS Excel.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

Ознайомитись з теоретичними відомостями, представленими на лекції, розуміти поняття алгоритму, його властивості, знати базові алгоритмічні структури та форми представлення алгоритмів, вміти побудувати алгоритм та розробити програму на С; знати типи даних, опис констант, змінних, стандартних функцій; введення виведення даних різного типу, правила запису арифметичних виразів. Виконайте загальне та індивідуальні завдання (згідно варіанту).

ЗАГАЛЬНЕ ЗАВДАННЯ

В ході виконання лабораторної роботи:

- проаналізуйте умову задачі.
- оберіть алгоритмічні конструкції, що будуть використані у процесі розробки алгоритму.
- ознайомтесь з принципами відладки програми у обраному середовищі програмування та використання системи довідки.
- дайте відповіді на запитання для самоконтролю.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Оберіть індивідуальне завдання згідно з номером варіанту

- проаналізуйте умову задачі
- розробіть алгоритм таким чином, щоб результати обчислення *x* та *y* були виведені на екран окремо для кожного набору формул
- реалізуйте програму розв’язання задачі використовуючи, оператори умови, циклу, множинного вибору
- перевірте коректність роботи програми обравши необхідні контрольні приклади
- оформіть звіт.

Таблиця 1.2 – Варіанти індивідуальних завдань

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
1	$x=\frac{5}{\sqrt{1+ce^{-m}}};$	$y=\alpha(1-\frac{\cos(c)}{\sqrt{n^2-\sin^2(c)}})$	c=0.7; m=0.3×10^{-2}; α=5; n=1.2
	$\begin{matrix} x<0 \\ x>1.4 \\ 0\le x\le 1.4 \end{matrix}$, $x\in[-1,2]$, $\Delta x=0.2;$	$y=\begin{cases} x^2+1 \\ x-2.1 \\ \cos(ax) \end{cases}$	$a=0.8\pi,$
2	$x=\frac{1}{r}(1-e^{-r\frac{t}{k}});$	$y=z^2\arcsin(\frac{r}{\sqrt{100-r^2}})$	r=5; k=1.24×10^{-7}; t=0.1×10^{-6}; z=0.5×10^2
	$\begin{matrix} x\le 1 \\ x>1 \end{matrix}$, $x\in[0.4,2]$, $\Delta x=0.2;$	$y=\begin{cases} \sin^2(\sqrt{\lvert ax\rvert}) \\ \lg(x+1) \end{cases}$	$a=18.5,$
3	$x=e^{\sqrt{1-a\sin(b)}}$	$y=\frac{ak}{4\ln(\frac{\alpha-z}{z})}$	a=0.1; b=1.4; α=0.02; z=3×10^{-3}; k=4.5
	$\begin{matrix} x>1.6 \\ x\le 1.6 \end{matrix}$, $x\in[1,2]$, $\Delta x=0.2;$	$y=\begin{cases} e^{-ax}\sin(bx) \\ \sqrt{\frac{x^3}{x-a}} \end{cases}$	$a=0.3,$
4	$x=ae^{-bc}\cos(b)$	$y=0.315\sqrt{\frac{ac^3}{b}}$	a=3.4; b=1.1; c=9
	$\begin{matrix} x>0.4 \\ x\le 0.4 \end{matrix}$,	$y=\begin{cases} \frac{a\sin(bx)}{bx} \\ \frac{ax}{1+x^2}+\frac{1}{1+ax^2} \end{cases}$	$a=2.8,\ b=2\pi,$

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	$x \in [-1,1],\ \Delta x = 0.2;$		
5	$x = \delta e^{b^2} \sin(\alpha)$	$y = \sqrt{b\cos^2(a) + a}$	δ=0.8; b=1.5; a=3; α=0.394
	$x > 0.5$ $x \le 0.5$, $x \in [-3,3],$ $\Delta x = 0.5;$	$y = \begin{cases} (x+2)^2 (x-1)^3 + \sin(a\pi) \\ x^3 + 6x^2 - 3 \end{cases}$	$a = 0.1,$
6	$x = e^{-\lambda b} \cos(b)$	$y = b\cos(t\sqrt{c} + b)$	при λ=0.1; b=0.6; c=2.4×10^{-4}; t=15
	$1 \le x \le 2$ $x < 1$, $x \in [0.4,2],$ $x > 2$ $\Delta x = 0.2;$	$y = \begin{cases} ax^2 \ln(x) \\ 1 \\ e^{ax}\cos(bx) \end{cases}$	$a = 18.5,$
7	$x = a^2 e^{-(\frac{a}{b})^2}$	$y = arctg(\sqrt{(\frac{4k}{a^2 c})^{-1}})$	при a=0.1; b=88;; c=0.2×10^{-6}
	$x \le 0.5$, $x \in [-2,2],$ $x > 0.5$ $\Delta x = 0.5;$	$y = \begin{cases} \dfrac{x^2 - 5x + 6}{x^2 + 1} \\ x^{2/3} - (x^2 - 1)^{1/3} + e^a \end{cases}$	$a = 0.2,$
8	$x = \dfrac{a^{2c} + b^{-c}\cos(a+b)}{c+1}$	$y = \sqrt{c^2 + b} - b^2 \sin^3(\frac{c+a}{c})$	при a=0.3; b=0.9; c=0.61
	$x < 4$ $4 \le x \le 6$, $x > 6$ $x \in [0,12],\ \Delta x = 1;$	$y = \begin{cases} \dfrac{a}{x} + bx^2 \\ x \\ ax + bx^3 \end{cases}$	$a = 2.1,\ b = 1.8,$
9	$x = \dfrac{a + \dfrac{b}{c + \sqrt{a}}}{\lvert b - a \rvert + \sqrt{a}}$	$y = e^{z-1} + \arcsin(z)$	при a=38.9; b=-4.7; c=5; z=0.8

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	$\begin{cases} x > 0.5 \\ x \le 0.5 \end{cases}$, $x \in [-2,2]$, $\Delta x = 0.5;$	$y = \begin{cases} \ln(x^2+1)+(x^3+1) \\ a\sin(x)-\cos(ax) \end{cases}$	$a = -2,$
10	$x = \sqrt[4]{b+\sqrt[3]{a}-1}$	$y = \lvert a-b \rvert (\sin^2 z + tg(z))$	при a=15.123; b=9.563; z=0.717
	$\begin{cases} x \ge 1 \\ -1 < x < 1 \\ x \le 1 \end{cases}$, $x \in [-2,2]$, $\Delta x = 0.5;$	$y = \begin{cases} \ln(x) \\ \dfrac{\sqrt{x^2+a^3}}{a} \\ e^x \end{cases}$	$a = 1.8,$
11	$x = \dfrac{\sqrt{\lvert a-1 \rvert} - \sqrt[3]{b}}{1+c^2+a^3}$	$y = \dfrac{1+\cos(a-2)}{b^2+c+\sin a}$	a=2; b=2,13; c=0.37
	$\begin{cases} x < -1 \\ -1 \le x < 1 \\ x \ge 1 \end{cases}$, $x \in [-2,2]$, $\Delta x = 0.5;$	$y = \begin{cases} -x-1 \\ 1-x^2\cos(x\pi) \\ x - \dfrac{1}{\sqrt{a^3}} \end{cases}$	$a = 3.1,$
12	$x = \dfrac{\sqrt[3]{c+\lvert a-b \rvert^2+3}}{a^2+b^2}$	$y = e^{\lvert a-b \rvert}(tg^2(z)+1)$	при a=4.4; b=0.57; c=6; z=0.054
	$\begin{cases} x < 1.3 \\ x = 1.3 \\ x > 1.3 \end{cases}$, $x \in [0.8,2]$, $\Delta x = 0.1;$	$y = \begin{cases} x^2\pi - \dfrac{7}{x^2} \\ ax^3 + \dfrac{7}{\sqrt{x}} \\ \lg(x+7\sqrt{x}) \end{cases}$	$a = 1.5,$
13	$\dfrac{a^2c+e^{-c}\cos(bc)}{bc-e^{-c}\sin(bc)+1}$	$y = e^{2c}\ln(a+c) - b^{3c}\ln(b-c)$	при a=0.5; b=2.7; c=0.4

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	$\begin{matrix} x \le 0 \\ 0 < x \le 1 \\ x > 1 \end{matrix}$, $x \in [-1.4, 1.4]$, $\Delta x = 0.1;$	$y = \begin{cases} 0 \\ x^2 - \dfrac{x}{a} \\ x^3 - \sin(x^2 \pi) - 1 \end{cases}$	$a = 1.8,$
14	$x = (1 + tg^2 \dfrac{a}{2})^{\sqrt{\lvert b \rvert + c}}$	$y = 2ctg(\dfrac{a}{\lvert b \rvert})$	при a=4.5×10^{-4}; b=-2×10^{-5}; c=25
	$\begin{matrix} x > 0 \\ x < 0 \\ x = 0 \end{matrix}$, $x \in [-2, 3]$, $\Delta x = 0.5;$	$y = \begin{cases} \sqrt{\dfrac{x^3}{x + a}} - \ln(x) \\ 2a \ln(-x) \\ 0 \end{cases}$	$a = 0.7,$
15	$x = \ln(a + c^2) + \sin^2(\dfrac{c}{b})$	$y = e^{-kc} \dfrac{c + \sqrt{c + a}}{c - \sqrt{\lvert c - b \rvert}}$	при a=9.6; b=8.2; c=2; k=0.7
	$\begin{matrix} x < 1.2 \\ x = 1.2 \\ x \ge 1.2 \end{matrix}$, $x \in [1, 2]$, $\Delta x = 0.1;$	$y = \begin{cases} ax^2 + bx \\ \dfrac{a}{x} + \sqrt{x^2 + 1} \\ \dfrac{a + bx}{\sqrt{x^2 + 1}} \end{cases}$	$a = 2.8, b = -0.3,$
16	$x = \dfrac{b^{a+1}}{\sqrt[3]{\lvert b - c \rvert}} + \dfrac{a + \dfrac{b}{2}}{2\lvert a + b \rvert}$	$y = (a + 1)^{-\frac{1}{\sin a}}$	при a=1.256; b=13.5; c=4
	$\begin{matrix} x > 1 \\ x \le 1 \\ x = 0 \end{matrix}$, $x \in [-2, 2]$, $\Delta x = 0.5;$	$y = \begin{cases} ae^{-bx} \cos(x\pi) \\ (x - 5)^2 \sqrt[3]{(x + 1)^2} \\ 0 \end{cases}$	$a = 3.4, b = 2.1$

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
17	$x = ae^{-ba}\cos(c) + z$	$y = \frac{a^2 + 5a + 6}{a^2 + 1}$	при a=1.256; b=3.5; c=0.53; z=7
	$\begin{matrix} x < 1.4 \\ x = 1.4 \\ x > 1.4 \end{matrix}$, $x \in [0.8,2]$, $\Delta x = 0.2$;	$y = \begin{cases} x^2\pi - \frac{7}{x^2} \\ ax^3 + \frac{8}{\sqrt{x}}) \\ \ln(x + 9\sqrt{\lvert x + a \rvert} \end{cases}$	$a = 1.65$,
18	$x = btg^2(c) - \frac{1}{\sin^2(\frac{c}{a})}$	$y = ae^{-\sqrt{a}}\cos(\frac{bc}{a})$	при a=2.8; b=16.4; c=-5.4
	$\begin{matrix} x < 1.2 \\ x = 1.2 \\ x > 1.2 \end{matrix}$, $x \in [-2,2]$, $\Delta x = 0.4$;	$y = \begin{cases} e^{\frac{x^2}{2}} + \ln(a) \\ a\cos^3(\pi\frac{x}{2.7}) \\ x^2 + ax + 5 \end{cases}$	$a = 4$,
19	$x = 2^{b^4} + (3^a)^b$	$y = \frac{\lvert a - b \rvert(1 + \frac{\sin^2 c}{a + b})}{e^{\lvert a-b \rvert} + \frac{a}{2}}$	при a=2.953; b=0.254; c=0.5
	$\begin{matrix} x \le 1 \\ 1 < x < 2 \\ x \ge 2 \end{matrix}$, $x \in [-2,2]$, $\Delta x = 0.2$;	$y = \begin{cases} 1.7\cos^2(x) \\ (x-4)^2 + a \\ 5tg(x) \end{cases}$	$a = 5.1$,
20	$x = 3 - \sqrt[q]{a + \sqrt{\lvert b \rvert}}$	$y = \sqrt[4]{e^{a - \frac{1}{\sin c}}}$	при a=4.125; b= -1.234; c=0.487
	$\begin{matrix} x > a \\ x = a \\ x < 1 \end{matrix}$, $x \in [1,5]$,	$y = \begin{cases} x\sqrt[3]{x - a} \\ x\sin(ax) \\ e^{-ax}\cos(ax) \end{cases}$	$a = 2.5$,

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	$\Delta x = 0.5;$		
21	$x = \left\|\cos\|a\| + \cos(b)\right\|^{1-2\sin^2(b)}$	$y = \ln(b^{-\sqrt{\|a\|}})(a - \frac{b}{2})$	при a= -0.92; b= 0.58
	$ax < 1$ $ax = 1$, $x \in [0.1,1],$ $ax > 1$ $\Delta x = 0.1;$	$y = \begin{cases} ax - \lg(ax) \\ 1 \\ ax + \lg(ax) \end{cases}$	$a = 1.5,$
22	$x = \left\| a^{\frac{b}{a}} - \sqrt[3]{b-1} \right\|$	$y = \frac{(b-a)b - \frac{c}{b - \cos(a)}}{1 + (b-a)^2}$	при a=1.725; b=19; c= -2.153
	$x > 3.5$ $x \le 3.5$, $x \in [2,5],$ $\Delta x = 0.25;$	$y = \begin{cases} \sin(x)\lg(x) \\ \cos^2(ax) \end{cases}$	$a = 0.9,$
23	$x = (a-5)^2 \sqrt[3]{(b+1)^2} + \ln(a)$	$y = ae^{-bc}\sin(c)$	при a=3.457; b= 3.1; c=2
	$x < 0.5$ $x = 0.5$, $x \in [0.2,2],$ $x > 0.5$ $\Delta x = 0.2;$	$y = \begin{cases} \frac{\ln^3(x) + x^2}{\sqrt{x+a}} \\ \sqrt{x+a} + \frac{1}{x} \\ \cos x - a\sin^2 x \end{cases}$	$a = 2.5,$
24	$x = \sqrt{\frac{b^3}{b-a}} - \ln a$	$y = a\sin^3(c) + b\cos^3(c)$	при a=2.389; b= 3.1; c=17
	$x < 2.5$ $2.5 \le x < 6$, $x \in [0,7],$ $x \ge 6$ $\Delta x = 0.5;$	$y = \begin{cases} \frac{a+b}{e^x + \cos(x)} \\ \frac{a+b}{x+2} \\ e^x + \sin(x) \end{cases}$	$a = 2.8,\ b = -0.35,$
25	$x = e^{-bt}\sin(at+b) - \sqrt{\|bt+a\|}$	$y = b\sin(at^2\cos(2t)) - 1$	при a=-0.5; b= 1.7; t=0.44

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	$\begin{cases} x>1 \\ x\le 1 \end{cases}$, $x\in[0.8,2]$, $\Delta x=0.1;$	$y=\begin{cases} a\lg(x)+\sqrt[3]{\lvert x\rvert} \\ 2a\cos(x)+3x^2 \end{cases}$	$a=0.7,$
26	$x=a^3tg^2(a+b)^2+\frac{c}{\sqrt{a+b}}$	$y=\frac{ba^2-c}{e^{ac}-1}$	при a=0.816; b= 3.4; c=16.7
	$\begin{cases} \sin(\frac{x^2+1}{c})>0 \\ \sin(\frac{x^2+1}{c})<0 \end{cases}$, $x\in[1,10]$, $\Delta x=1;$	$y=\begin{cases} a\sin(\frac{x^2+1}{c}) \\ \cos(x+\frac{1}{c}) \end{cases}$	$a=0.3,\ c=10,$
27	$x=\sin^3(b^2+a^2)-\sqrt{\frac{b}{c}}$	$y=\frac{b^2}{a}+\cos(b+c)^3,$	при a=1.1; b= 0.2; c=4×10^{-3}
	$\begin{cases} x<0.1 \\ x=0.1 \\ x>0.1 \end{cases}$, $x\in[-1,1]$, $\Delta x=0.2;$	$y=\begin{cases} \sqrt{ax^2+b\sin(x)+1} \\ ax+b \\ \sqrt{ax^2+b\cos(x)+1} \end{cases}$	$a=2.1,\ b=0.4,$
28	$x=\frac{a+\left\lvert\frac{a}{bc}+\frac{b}{a+c}\right\rvert lna^2}{20a-\sqrt{b^2+c^2}}$	$y=a(arctg(c)+e^{(b+1)}$	a=3,15. b=4, c=3
	x>2,6 x=2,6 $x\in[-0,4;4]$ x<2,6 $\Delta x=0,2$	$y=\begin{cases} e^{x+0,2}+\sin(x^2-1,3) \\ \sqrt{x+5}\frac{(2+x)^2}{3x} \\ x^a+\sqrt[b]{x} \end{cases}$	a=1,5. b=5,
29	$x=\frac{a+b/(a^2+4)}{(1+b)e^{-a-2}+1/(c^2+4)}$	$y=\frac{1+cos(b-2)}{a^4+c+\sin(c)}$	a=2, b = 2.1285, c=0,38
	x>4 x=4 $x\in[2;6]$ x<4 $\Delta x=0,2$	$y=\begin{cases} \frac{a}{\sqrt{2x^2+\cos^2 x}} \\ \frac{(2+tg(x)^2}{ax} \\ x\frac{x^b+a}{b} \end{cases}$	a=1,25. b=4,
30	$x=\frac{2\cos(a-\frac{\pi}{6})}{0.5c+sin^2b}$	$y=1+\frac{c^2}{3+\frac{c^3}{b^{1/a}}}$	a=4, b=1,49. c=4.8

Номер варіанту	Формула для розрахунку x	Формула для розрахунку y	Значення констант
	x>1 x=1 $x \in [-1{,}5; 3{,}5]$ x<1 $\Delta x = 0{,}3$	$y = \begin{cases} \frac{a-x}{\sqrt{a+x}} \\ \sqrt{\lvert x-b \rvert} \\ (x^2+5x)(\frac{1}{x}+\sqrt{x+4}) \end{cases}$	a=3, b=1,2,
31	$x = \ln\left\lvert(b-\sqrt{\lvert a \rvert})\right\rvert (a-\frac{b}{c+a^2}$	$y = a - \frac{\sqrt{b}}{b^2} + \frac{c^2}{\sqrt[3]{c}}$	a=-3, b = -4,12. c=-6
	x>1,5 x=1,5 $x \in [0; 3]$ x<1,5 $\Delta x = 0{,}15$	$y = \begin{cases} \frac{a^b-x}{\sqrt{a-x}} \\ \frac{(a+x^b)^2}{bx+a} \\ 3tg(x+2)\sqrt{\lvert x \rvert} \end{cases}$	a=3. b=4,
32	$x = arctg(aln(b^3-5b+c)+7a$	$y = \lvert b-c \rvert log(a^2)+b^3$	a=3, b =3. c=12
	x>2 x=2 $x \in [0; 4]$ x<2 $\Delta x = 0{,}2$	$y = \begin{cases} x+2{,}5\sqrt[a]{3+x^2} \\ e^a + xe^{x-b} \\ x^2\sqrt{x^3+a} \end{cases}$	a=5. b=1,4,

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яка структуру має програма на мові C/C++?
2. Охарактеризуйте кроки перетворення програмного коду на мові С у виконуваний файл?
3. Що таке препроцесор? Як отримати одиницю трансляції?
4. За допомогою якої директиви препроцесора до програми долучають бібліотечні модулі (заголовочні файли)?
5. В яких бібліотеках містяться математичні функції?
6. Яке призначення мають фігурні та круглі дужки в С?
7. Назвіть послідовність створювання програми у середовищі розробки яке Ви обрали.
8. В який спосіб можна зберегти програму?

9. Що таке консольна програма?
10. Які функції введення-виведення даних у консольному режимі Вам відомі?
11. Які заголовні файли слід долучити для використання функцій введення-виведення?
12. Вкажіть призначення і можливий синтаксис функції main() .
13. Назвіть послідовність створювання консольного додатку середовищі розробки яке Ви обрали.
14. Поясніть зміст поняття оператор.
15. Що розуміється під типом даних?
16. Які типи даних використовуються у мові С?
17. Дайте означення виразу.
18. Наведіть приклади операцій з однаковим пріоритетом.
19. Вкажіть операції з найвищим та найнижчим пріоритетом.
20. Які засоби мови С призначені для вводу/виводу даних різних типів?
21. Чи може умовний оператор містити в собі:
    а) інші умовні оператори?
    б) оператор безумовного переходу?
22. В чому різниця між повною і короткою формою запису умовного оператора?
23. Для чого призначений оператор вибору SWITCH-CASE ?
24. Який тип даних допускається у виразі-селекторі оператора вибору SWITCH-CASE ?
25. Що необхідно зробити в програмі якщо по заданій умові вимагається провести більше однієї дії ?
26. Що таке інструкція безумовного переходу? Який принцип її використання? Які переваги і недоліки використання інструкцій безумовного переходу?

ЛАБОРАТОРНА РОБОТА №2
ВИКОНАННЯ ОБЧИСЛЕНЬ ЗА ДОПОМОГОЮ РЕКУРЕНТНИХ СПІВВІДНОШЕНЬ. ЗАДАНА ТОЧНІСТЬ. ФОРМАТОВАНЕ ВВЕДЕННЯ-ВИВЕДЕННЯ

Мета роботи: ***оволодіти навичками програмування обчислювальних процесів, організація введення-виведення із заданою точністю***

ТЕОРЕТИЧНІ ВІДОМОСТІ

За допомогою рекурентних співвідношень можна розв'язати чимало задач. Головне – знайти рекурентне співвідношення й початкові умови. Далі безпосередньо за ними не важко запрограмувати циклічні обчислення.

Якщо існує функція *f*, що $a_i = a(i) = f(i)$, то кажуть, що послідовність визначена за допомогою загального члена. Прикладами можуть слугувати арифметична та геометрична прогресії, в яких відомі формули загального члена.

Однак, зазвичай, визначити формулу загального члена послідовності дуже важко (якщо навіть у принципі це можливо). Найчастіше значення елемента послідовності визначають за допомогою інших елементів цієї ж послідовності (рекурентно). Найпростішим рекурентним співвідношенням (чи рекурентним рівнянням) називають деяку функцію, яка визначає значення поточного (чергового) елемента послідовності через одне або декілька значень попередніх елементів цієї ж послідовності, причому відомими є значення відповідної кількості початкових елементів.

Припустимо, що перші *k* елементів послідовності a_0, a_1, …, a_n, … задано явно (це початкові умови a_0, a_1, …, a_{k-1}) і є закон *F*, який дозволяє за номером елемента та/або деякими попередніми елементами знайти всі інші елементи:

$a_i = F(i, a_{i-1}, \ldots, a_0)$ за всіх $i \geq k$.

Рівність називають рекурентним співвідношенням. Кожне рекурентне співвідношення разом із початковими умовами визначає певну послідовність.

Однією з переваг методу є те, що незважаючи на наявність індексів у формулах рекурентних співвідношень, створювати масив для зберігання усіх членів послідовності, зазвичай, не потрібно.

Це стосується задач, в яких необхідно обчислити кінцеве значення послідовності або використовувати поточні значення тільки на одній ітерації циклу.

Під час обчислення сум зі скінченною чи нескінченною кількістю доданків перед початком циклу обов’язково необхідно присвоїти значення 0 (нуль) змінній, в якій накопичуватиметься сума цих доданків.

Аналогічно, під час обчислення добутків зі скінченною чи нескінченною кількістю множників перед початком циклу обов’язково необхідно присвоїти значення 1 (один) змінній, в якій накопичуватиметься добуток цих множників (наприклад, обчислення факторіалу).

Однак, не завжди для суми ряду існує аналітичний вираз. У подібних випадках для визначення суми застосовуються наближені обчислення, які виконуються за допомогою ітераційного процесу.

Ітераційний процес реалізується за допомогою циклу, а саме: при кожному проході циклу обчислюється окремий член ряду й додається до значення суми. У деяких випадках, щоб скоротити об'єм обчислень при знаходженні члена ряду, можна використати результат попереднього проходу циклу. Тоді наступний член ряду визначається за допомогою рекурентного співвідношення.

У наближених обчисленнях спадного ряду враховується скінченне число членів ряду. Їх кількість визначається на основі одного з наступних критеріїв:

1. При обчисленні суми спадного ряду враховується рівно стільки членів, скільки необхідно для досягнення заданої точності обчислення ε. Зазвичай, такий підхід застосовують, якщо відомо точне значення суми.

2. При обчисленні суми спадного ряду враховуються члени, що перевищують задане значення σ, наступні члени, які менші σ, відкидаються.

Основна проблема при обчисленні суми числового ряду полягає в обчисленні чергового члена послідовності. Можна виділити три підходи до розвязку цієї проблеми:

1) використання формули спільного члена ряду;

2) використання рекурентного співвідношення;

3) змішаний підхід, заснований на двох попередніх.

Рекурентне співвідношення використовується в тих випадках, коли наступний член ряду можна виразити через попередній (або попередні), а використання формули загального члену призведе до появи вкладених циклів, що робить програму неефективною.

Визначимо рекурентне співвідношення для ряду, суму елементів якого необхідно знайти:

$$S = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

$$a_0 = \frac{x^0}{0!} = 1; \quad a_1 = \frac{x^1}{1!} = x; \quad a_i = \frac{x^i}{i!}; \quad a_{i-1} = \frac{x^{i-1}}{(i-1)!}$$

$$f(i) = \frac{a_i}{a_{i-1}} = \frac{x^i \cdot (i-1)!}{x^{i-1} \cdot i!} = \frac{x}{i} \qquad a_i = \begin{cases} 1, & \text{если } i = 0 \\ a_{i-1} \cdot \frac{x}{i}, & \text{если } i > 0 \end{cases}$$

Приклад 1. Необхідно написати програму обчислення суми N перших членів ряду:

$$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

$$a_0 = 1, \; a_1 = -\frac{x^2}{2!}, \; a_2 = \frac{x^4}{4!}, \; a_i = (-1)^i \frac{x^{2i}}{(2i)!}, \; a_{i-1} = (-1)^{i-1} \frac{x^{2\cdot(i-1)}}{(2\cdot(i-1))!}$$

Оскільки кожний член ряду може бути отриманий з попереднього домноженням на визначений множник, а обчислення «в лоб» кожного члена ряду призведе до появи вкладених циклів, ефективніше буде використовувати рекурентне співвідношення. Виведемо:

$$\frac{a_i}{a_{i-1}} = \frac{x^{2i} \cdot (-1)^i \cdot (2\cdot(i-1))!}{x^{2\cdot(i-1)} \cdot (-1)^{i-1} \cdot (2i)!} = -\frac{x^2}{(2i-1)\cdot(2i)}$$

Одержане рекурентне співвідношення:

$$a_i = \begin{cases} 1, \ если \ i = 0 \\ a_{i-1} \cdot \left(-\dfrac{x^2}{(2i-1)\cdot(2i)} \right), \ если \ i > 0 \end{cases}$$

Фрагмент програми - розрахунок суми:

```
s=1;
     a=1;
     for (i=1;i<n;i++)
     {
      a=-a*x*x/(2*i-1)/2/i;
      s=s+a;
     }
```

Приклад 2. Необхідно обчислити значення змінної S з точністю 0,00001. S задана формулою:

$$S = \sum_{n=1}^{\infty} \frac{-x^n}{(2n-1)!}$$

Особливістю даної задачі є те, що її можна розв'язати як використовуючи рекурсію, так і уникнути її використання, застосувавши ітераційний варіант циклу. Як можна помітити, кожний член ряду може бути представлений у вигляді:

$$y_{n+1} = y_n * R,$$

$y_{n+1} = \frac{-x^{n+1}}{2(n+1)-1}, y_n = \frac{-x^n}{2n-1},$ де R – рекурентний множник, який можна обчислити з відношення двох членів ряду, розташованих поряд.

В даному випадку:

$$R = \left(\frac{-x^{n+1}}{2(n+1)-1}\right) \Big/ \left(\frac{-x^n}{2n-1}\right) = \frac{x(2n-1)}{2n-1}$$

Перший член ряду: $y_1 = -x$

Змінним, використовуваним для накопичення суми, підрахунку кількості ітерацій, необхідно надати початкові значення, тобто S=0, k=1.

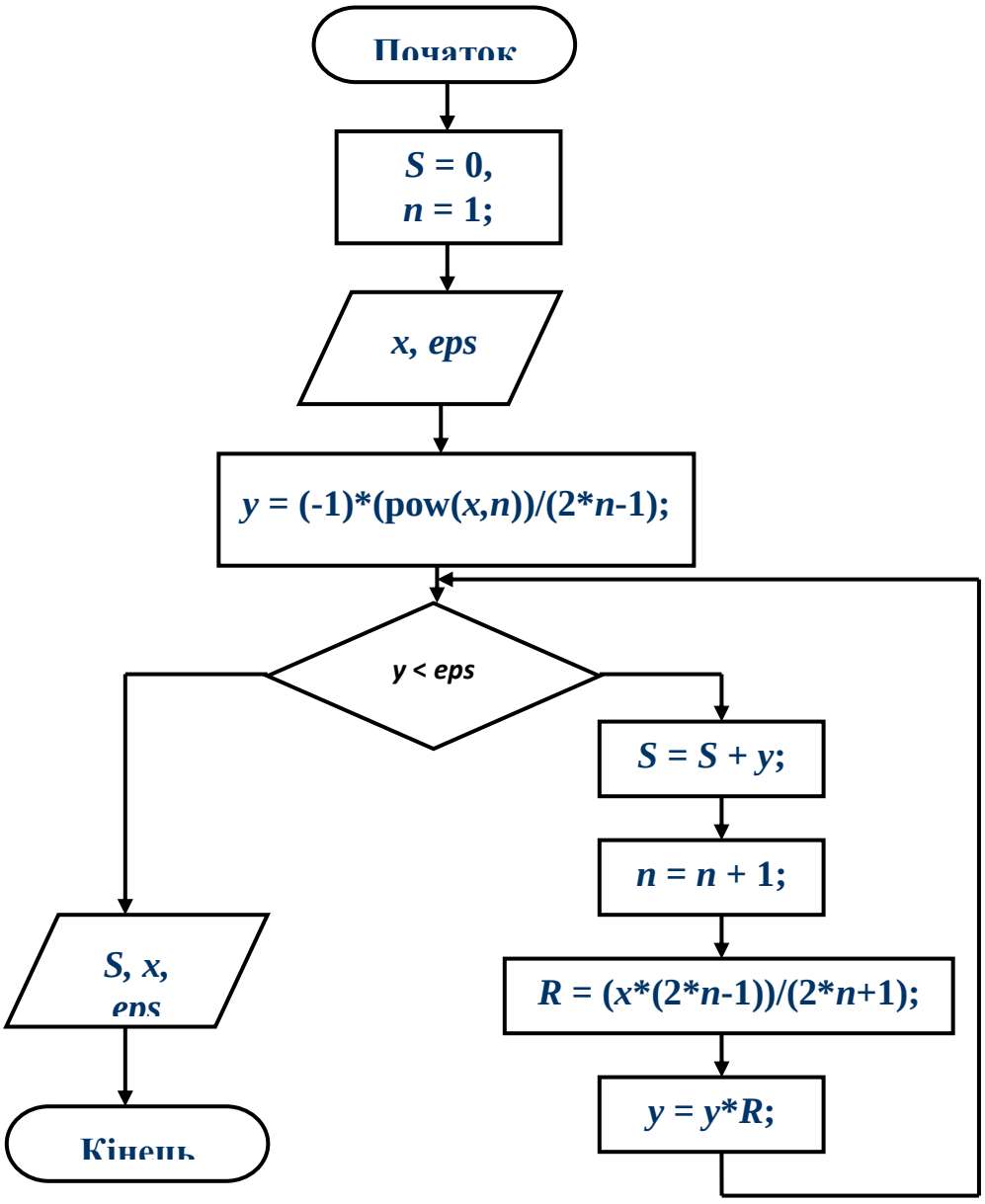


Рисунок 3.1 – Блок –схема алгоритму знаходження суми елементів послідовності із заданою точністю

Приклад 3. Обчислити суму ряду:

$$y = \sum_{k=1}^{\infty} \frac{(-1)^k x^{2k}}{2k(2k-1)!}$$

підсумовуючи члени ряду, значення яких за модулем є більшими за задану точність ε. Визначити кількість доданків. Значення x та ε вводити з клавіатури.

```
float x, y = 0, u, eps;
```

```
scanf (“%f”, &x);
scanf (“%f”, &eps);
int i, f, k = 1;
do
{
for(i = 1, f = 1; i <= 2 * k - 1; i++)
f *= i; u = pow(-1, k)*pow(x, 2 * k) / (2 * k * f);
printf (“%f”, “%f”, k, u);
y += u; k++;
}
while (fabs(u) >= eps) ;
printf( “%f”, y);
printf (“%f”, (k – 1);
```

Подамо суму ряду у $y=\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k}}{2k(2k-1)!}$ вигляді $y=\sum_{k=1}^{\infty}u_k$,

де $u_k=\frac{(-1)^k x^{2k}}{2k(2k-1)!}$.]

Оскільки рекурентний множник R – це співвідношення двох поряд розташованих членів ряду:

$$u_2=R\cdot u_1,\ u_3=R\cdot u_2,\ \ldots,\ u_k=R\cdot u_{k-1},\ u_{k+1}=R\cdot u_k,$$

$$R=\frac{u_k}{u_{k-1}}\ \text{або}\ R=\frac{u_{k+1}}{u_k}$$

можна обчислити рекурентний множник зі співвідношення з попереднім членом ряду або зі співвідношення з наступним членом ряду.

Для визначення R до формули $R=\frac{u_k}{u_{k-1}}$ слід підставити $u_k=\frac{(-1)^k x^{2k}}{2k(2k-1)!}$ та u_{k-1}. Для обчислення u_{k-1} підставимо до виразу для u_k $(k-1)$ замість k:

$$u_{k-1}=\frac{(-1)^{k-1}x^{2(k-1)}}{2(k-1)(2(k-1)-1)!}=\frac{(-1)^{k-1}x^{2(k-1)}}{2(k-1)(2k-3)!};$$

$$R=\frac{u_k}{u_{k-1}}=\frac{\frac{(-1)^k x^{2k}}{2k(2k-1)!}}{\frac{2(k-1)(2k-3)!}{(-1)^{k-1}x^{2(k-1)}}}=\frac{(-1)^k\cdot x^{2k}\cdot(2k-2)\cdot(2k-3)!}{(-1)^{k-1}\cdot x^{2k-2}\cdot 2k\cdot(2k-1)!}=$$

$$=\frac{(-1)^{k-k+1}\cdot x^{2k-(2k-2)}\cancel{(2k-2)}}{2k}\cdot\frac{\cancel{1\cdot 2\cdot\ldots\cdot(2k-3)}}{\cancel{1\cdot 2\cdot\ldots\cdot(2k-3)}\cdot\cancel{(2k-2)}\cdot(2k-1)}=-\frac{x^2}{2k(2k-1)}.$$

Окрім рекурентного множника R, треба обчислити перший член ряду за $k=1$:

$$u_1=\frac{(-1)^1 x^2}{2(2-1)!}=-\frac{x^2}{2}.$$

Наприклад:

$$R=\frac{u_{k+1}}{u_k}=\frac{\frac{(-1)^{k+1}x^{2(k+1)}}{2(k+1)(2k+1)!}}{\frac{(-1)^k x^{2k}}{2k(2k-1)!}}=\frac{(-1)^{k+1}\cdot x^{2k+2}\cdot 2k\cdot(2k-1)!}{(-1)^k\cdot x^{2k}\cdot 2(k+1)\cdot(2k+1)!}=$$

$$=\frac{(-1)^{k+1-k}\cdot x^{2k+2-2k}\cancel{k}}{(k+1)}\cdot\frac{\cancel{1\cdot 2\cdot\ldots\cdot(2k-1)}}{\cancel{1\cdot 2\cdot\ldots\cdot(2k-1)}\cdot\cancel{2k}\cdot(2k+1)}=-\frac{x^2}{2(k+1)(2k+1)}.$$

Окрім рекурентного множника R, треба обчислити перший член ряду за $k=1$:

$$u_1=\frac{(-1)^1 x^2}{2(2-1)!}=-\frac{x^2}{2}.$$

Програмна реалізація:

Перший варіант:

```
float x, y, u, r, eps; int i, f, k = 1;
scanf (“%f”, &x);
```

```
scanf (“%f”, &eps);
u = -x*x/2;
y = u;
printf(“%f”, k);
printf(“%f”,u);
do
{
k++;
r = -x*x/(2*k*(2*k-1));
u *= r ;
printf (“%f”, k);
printf(“%f”,u);
y += u;
}
while(fabs(u)>=eps);
printf (“%f”, k);
printf(“%f”,y);
```

Другий варіант:

```
float x, y, u, r, eps; int i, f, k = 1;
scanf (“%f”, &x);
scanf (“%f”, &eps);
u = -x*x/2;
y = u;
printf (“%f”, k);
printf(“%f”,u);
do
{
```

```
r = -x*x/(2*(k+1)*(2*k+1));
u *= r ;
printf ("%f", k);
printf("%f",u);
y += u;
k++;
}
while(fabs(u)>=eps);
printf ("%f", k);
prin
tf("%f",y);
```

Використовуючи представлений підхід можна реалізувати ефективні алгоритми розв'язку таких задач, як обчислення значень основних математичних функцій, послідовності чисел Фібоначчі, знаходження мінімального (максимального) елемента нескінченної послідовності.

Наприклад, можна виконати наближене обчислення математичних функцій, які представлені у вигляді суми елементів збіжного числового ряду, із забезпеченням заданої точності розрахунку. Тоді дана задача зводиться до обчислення суми *S* нескінченної збіжної послідовності $s = \sum_{i=0}^{\infty} a_i = a_0 + a_1 + a_2 + ... + a_n$, яка буде отримана після розкладання функції в ряд. Оскільки ряд збігається, то для будь-якого значення змінної ***a***, достатньою умовою забезпечення заданої точності розрахунку (***eps***) є досягнення черговим (n) членом ряду значення, коли умова a_n < **eps** буде істинною. Множник **q**, який відображає залежності сусідніх членів ряду, визначається з наступного рекурентного співвідношення: $a_i = a_{i-1} \times q$, де a_{i-1} і a_i - відповідно попередній і наступний члени ряду. Алгоритм наближеного обчислення значень математичних функцій із використанням циклу з пост-умовою представлений на рис. 3.2. Приклади

алгоритмів із застосуванням циклів представлені на рис. 3.3.

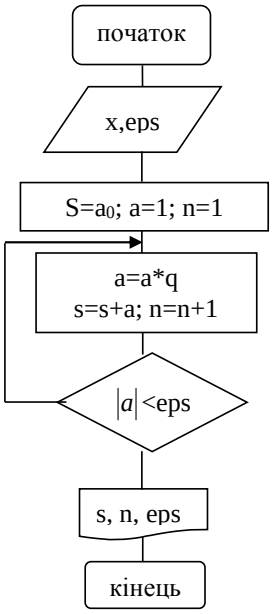
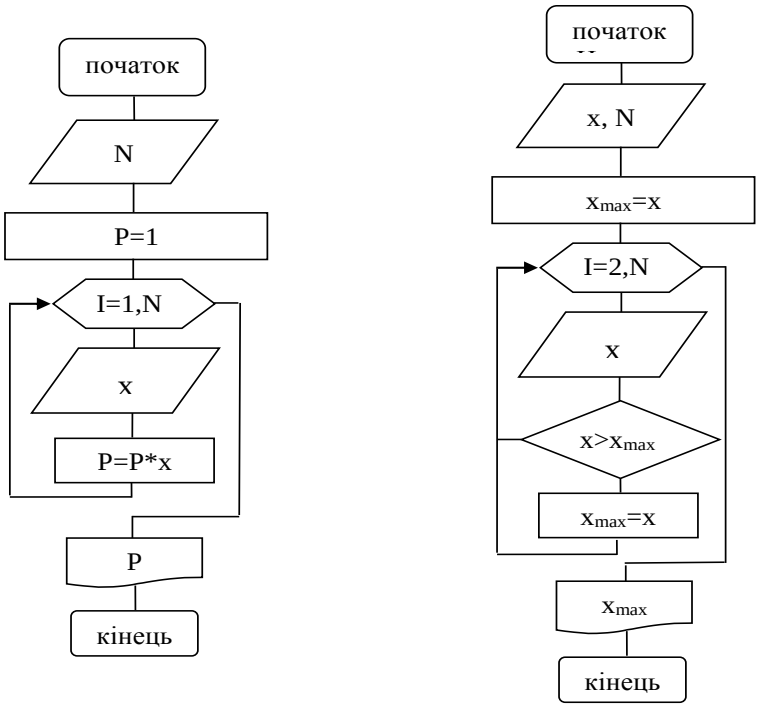


Рисунок 3.2 – Алгоритм наближеного обчислення математичної функції



а) *б*)

Рисунок 3.3 – Застосування циклу з лічильником у алгоритмі обчислення довільного кінцевого числа співмножників (*а*) та в алгоритмі знаходження максимального елемента зі значень, що вводяться (*б*)

Форматоване (форматне) виведення реалізується функцією:

printf (<форматний рядок>, <список аргументів>);

<форматний рядок>

<форматний рядок> – рядок символів, вміщених у лапки, що показує, як повинні бути надруковані аргументи. Наприклад:

```
printf ("Value PI equals %f\n", pi);
```

Форматний рядок може містити символи тексту для виведення, специфікації перетворення та управляючі символи. Кожному аргументу відповідає своя специфікація перетворення, наприклад

%d – десяткове ціле число;

%f – число із плаваючою крапкою типу `float` або `double`;

%e – число в експонентній формі типу float або double;;

%c – символ

%s – рядок.

Таким чином, специфікація перетворення починається із символу «%» і закінчується символом перетворення (*d, f, c* та ін.). Після символу «%» перед символом перед петворення може перебувати:

– (Знак мінус), що вказує на вирівнювання перетвореного аргументу по лівому краю поля виведення

Ширина поля виведення – задає мінімальне число позицій для перетворено-го аргументу. Якщо величина аргументу вимагає більшого числа позицій, то поле виведення автоматично розширюється. Якщо ж величина має менше символів, чим зазначена ширина поля, то ліворуч від її додаються пробіли.

Точність – число, яке стоїть через крапку після запису ширини поля виведення й показує, скільки позицій ширини поля виведення виділяється під дробову частину аргументу типу **float** або **double**. Для специфікацій **f** та **e** точність за замовчуванням дорівнює 6.

printf ("x=%20.8f\n", x);/*Виведеня чення аргументу x у поле шириною 20 з 8 знаками після десяткової крапки *****/

printf ("x=%-10d\ny=%-10d", x, y);

– виведення значень x та y рядки з вирівнюванням зліва.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися із теоретичним відомостями і методичними вказівками до виконання лабораторної роботи.

2. Скласти блок-схеми алгоритмів і програми:

а) для обчислення суми довільної кількості чисел, що вводяться з клавіатури;

б) для обчислення добутку;

$$P = \frac{x_1^2}{1-x_1} \times \frac{x_2^2}{1-x_2} \times \ldots \times \frac{x_n^2}{1-x_n};\ x_i \neq 1$$

в) для обчислення числа сполучень з m чисел по n.

$$P = C_m^n = \frac{m(m-1)\ldots(m-n+1)}{n!};$$

3. Скласти програму пошуку мінімального елемента з низки довільних чисел N, що вводяться з клавіатури.

4. Відлагодити складені програми і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Скласти алгоритм і програму обчислення значення функції, представленої у вигляді суми елементів нескінченного збіжного ряду з точністю **eps**1 = 0.01 і **eps**2 = 0.0001. Вивести значення суми елементів ряду **s,** кількість ітерацій **(n**), за яких було досягнуто заданої точності **(eps)**. Результати представити у вигляді звіту. Завдання обирається згідно номеру варіанту.

1. $S = e^{x} \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$ $|x| \le \infty$

2. $S = e^{-x} \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots$ $|x| \le \infty$

3. $S = \sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$ $|x| \le \infty$

4. $S = \cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$ $|x| \le \infty$

5. $S = sh(x) \approx x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} \ldots$ $|x| \le \infty$

6. $S = ch(x) \approx 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots$ $|x| \le \infty$

7. $S = arctg(x) \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$ $|x| \le 1$

8. $S = arth(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \cdots$ $|x| \le 1$

9. $S = \ln(1 + x) \approx x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots$ $|x| \le 1$

10. $S = \ln(1 - x) \approx -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \ldots$ $|x| \le 1$

11. $S = \frac{1}{1 + x} \approx 1 - x + x^2 - x^3 + \ldots$ $|x| < 1$

12. $S = \frac{1}{1 - x} \approx 1 + x + x^2 + x^3 + \ldots$ $|x| < 1$

13. $S = \ln(\frac{1 + x}{1 - x}) \approx 2(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} \ldots)$ $|x| < 1$

14. $S=\frac{1}{(1+x)^2}\approx 1-2x+3x^2-4x^3+\ldots \qquad |x|<1$

15. $S=\frac{1}{1+x^2}\approx 1-x^2+x^4-x^6\ldots \qquad |x|<1$

16. $S=\frac{1}{(1+x)^3}\approx 1-\frac{2\times 3}{2}x+\frac{3\times 4}{2}x^2-\frac{4\times 5}{2}x^3+\ldots \qquad |x|<1$

17. $S=e^{1/x}\approx 1+\frac{1}{x}+\frac{1}{2!x^2}+\frac{1}{3!x^3}+\ldots \qquad |x|<1$

18. $S=\frac{e^x}{x}\approx \frac{1}{x}+1+\frac{x}{2!}+\frac{x^2}{3!}+\ldots \qquad |x|<1$

19. $S=(1+x)^{-2/3}=1-\frac{2}{3}x+\frac{2}{3}\frac{5}{6}x^2-\frac{2}{3}\frac{5}{6}\frac{8}{9}x^3+\cdots |x|<1$

20. $S=(1-x)^{-2/3}=1+\frac{2}{3}x+\frac{2}{3}\frac{5}{6}x^2+\frac{2}{3}\frac{5}{6}\frac{8}{9}x^3+\cdots |x|<1$

21. $S=(1+x)^{-1/4}\approx 1-\frac{1}{4}x+\frac{1}{4}\frac{5}{8}x^2-\frac{1}{4}\frac{5}{8}\frac{9}{12}x^3+\ldots \qquad |x|<1$

22. $S=(1-x)^{-1/4}\approx 1+\frac{1}{4}x+\frac{1}{4}\frac{5}{8}x^2+\frac{1}{4}\frac{5}{8}\frac{9}{12}x^3+\ldots \qquad |x|<1$

23. $S=(1+x)^{-1/3}\approx 1-\frac{1}{3}x+\frac{1}{3}\frac{4}{6}x^2-\frac{1}{3}\frac{4}{6}\frac{7}{9}x^3\ldots \qquad |x|<1$

24. $S=(1-x)^{-1/3}\approx 1+\frac{1}{3}x+\frac{1}{3}\frac{4}{6}x^2+\frac{1}{3}\frac{4}{6}\frac{7}{9}x^3+\ldots \qquad |x|<1$

25. $S=(1+x)^{-1/2}\approx 1-\frac{1}{2}x+\frac{1}{2}\frac{3}{4}x^2-\frac{1}{2}\frac{3}{4}\frac{5}{6}x^3+\ldots \qquad |x|<1$

26. $S=(1-x)^{-1/2}\approx 1+\frac{1}{2}x+\frac{1}{2}\frac{3}{4}x^2+\frac{1}{2}\frac{3}{4}\frac{5}{6}x^3+\ldots \qquad |x|<1$

27. $S=(1+x)^{-3/2}\approx 1-\frac{3}{2}x+\frac{3}{2}\frac{5}{4}x^2-\frac{3}{2}\frac{5}{4}\frac{7}{6}x^3+\ldots \qquad |x|<1$

28. $S = (1-x)^{-3/2} \approx 1 + \frac{3}{2}x + \frac{3}{2}\frac{5}{4}x^2 + \frac{3}{2}\frac{5}{4}\frac{7}{6}x^3 + \ldots \qquad |x| < 1$

29. $S = (1+x)^{-5/2} \approx 1 - \frac{5}{2}x + \frac{5}{2}\frac{7}{4}x^2 - \frac{5}{2}\frac{7}{4}\frac{9}{6}x^3 + \ldots \qquad |x| < 1$

30. $S = (1-x)^{-5/2} \approx 1 + \frac{5}{2}x + \frac{5}{2}\frac{7}{4}x^2 + \frac{5}{2}\frac{7}{4}\frac{9}{6}x^3 + \ldots \qquad |x| < 1$

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Які обмеження накладаються при використанні змінної циклу?
2. Вказати основні правила організації вкладених циклів?
3. Чи можливий вихід з внутрішнього циклу до його повного завершення?
4. Що таке ітераційний циклічний процес? Його відмінності від циклу із заданим числом повторень?
5. Які умови збіжності методу ітерацій?
6. Чому при підрахунку значення поточного члена $\mathbf{a_n}$ використовується проста змінна, а не індексована?
7. Чи можна додатково змінювати всередині циклу значення параметрів циклу.
8. Чи можна увійти до циклу з середини тіла циклу?
9. Чи визначено значення параметра циклу при виході з циклу?
10. Чи можна вийти з циклу перш ніж вичерпається значень параметра циклу?
11. 10.Чи повинен співпадати тип параметра циклу з типом початкового і кінцевого значень циклу?
12. Перерахувати можливі способи організації циклу за допомогою а) умовного оператора; б) операторів циклу.
13. Чи можливо в якості змінної циклу використати змінну типу float?
14. В чому відмінність між циклами WHILE та DO-WHILE?
15. Чи буде виконуватися циклічна частина програми, якщо логічний вираз є помилковим із самого початку в операторі циклу WHILE?

ЛАБОРАТОРНА РОБОТА № 3
ОБРОБКА ЧИСЛОВИХ ПОСЛІДОВНОСТЕЙ. ВИКОРИСТАННЯ ЦИКЛІВ

Мета роботи: *оволодіти навичками програмування циклічних обчислювальних процесів, організація обробки числових послідовностей.*

ТЕОРЕТИЧНІ ВІДОМОСТІ

Послідовність даних не завжди треба зберігати в пам'яті комп'ютера. Послідовність можна обробляти по мірі надходження її елементів: при читанні файлу, при введенні певних даних з клавіатури та т.і.

В даній роботі розглядаються числові послідовності, елементи яких вводяться з клавіатури (або генеруються випадково) по одному. Ознакою завершення числової послідовності може бути введення певного значення, яке не є її елементом, наприклад, 0 або введення (генерація) певної кількості елементів.

Для генерації випадкових значень використовуються функції, наведені в Додатку А.

Приклад 1. Підрахувати кількість мінімальних елементів послідовності дійсних чисел.

```
#include <stdio.h>
int main( )
{
    double a, /* значення поточного елементу послідовності */
    min; /* мінімальний елемент послідовності */
```

```
    int n = 0; /* кількість мінімальних елементів */
    printf("a = ");
    scanf("%lf", &a);
    min = a;  /* початкове значення мінімального елемента */
    while (a != 0)
    {
        if (a == min)
            n++;
        else
            if (a < min)
            {
            /* якщо введений елемент виявився менше всіх попередніх */
            /* елементів послідовності, тоді перевизначаємо min і n */
                min = a;
                n = 1;
            }
            printf("a = ");
            scanf("%lf", &a);
    }
    printf("n = %d\n", n);
    return 0;
}
```

Приклад 2. В послідовності цілих чисел знайти максимальну кількість додатних елементів, що йдуть підряд.

В коді, що наведений у прикладі, в якості початкового значення змінної max (максимальна кількість додатних елементів, що йдуть підряд) береться значення 0, тому що максимум шукають серед невід’ємних елементів. Якщо ж

треба знайти мінімум або максимум серед всіх елементів певної скінченної множини, елементами якої є довільні числа, то ні в якому випадку неможна в якості початкового значення для min і max брати число 0. Для цього випадку є значна кількість підходів, наприклад, в якості початкового значення можна взяти довільний елемент множини, що розглядається.

```
#include <stdio.h>
int main( )
{
     int a, /* значення поточного елемента послідовності */
     n, /* кількість додатних елементів, що йдуть підряд */
     max; /* максимальна кількість додатних елементів, що йдуть підряд */
     max = 0; /* початкове значення максимуму */
     printf("a = ");
     scanf("%d", &a);
     while (a != 0)
     {
          n=0;
          /* рахуємо кількість додатних елементів, що йдуть підряд */
          while (a > 0)
          {
               n++;
               printf("a = ");
               scanf("%d", &a);
          }
          if (n > max)
               max = n;
          /* переглядаємо всі від'ємні елементи, що йдуть підряд */
          while (a < 0)
```

```
        {
            printf("a = ");
            scanf("%d", &a);
        }
    }
    printf("max = %d\n", max);
    return 0;
}
```

Приклад 3. Нехай є послідовність цілих чисел. Необхідно вивести довжини всіх серій додатних елементів, що йдуть підряд, та від'ємних елементів, що йдуть підряд.

/* варіант з вкладеними циклами */

```
#include <stdio.h>
int main( )
{
    int a, n;
    n = 0;
    printf("a = ");
    scanf("%d", &a);
    while (a != 0)
    {
        while (a > 0)
        {
            n++;
            printf("a = ");
            scanf("%d", &a);
        }
```

```
            if (n > 0)
            {
                printf("+: %d\n", n);
                n = 0;
            }
            while (a < 0)
            {
                n++;
                printf("a = ");
                scanf("%d", &a);
            }
            if (n > 0)
            {
                printf("-: %d\n", n);
                n = 0;
            }
        }
        return 0;
}
```

/* варіант без вкладених циклів */

```
#include <stdio.h>
int main( )
{
    int a, n_plus, n_minus;
    n_minus = n_plus = 0;
    printf("a = ");
    scanf("%d", &a);
    while (a != 0)
```

```
{
    if (a > 0)
    {
        n_plus++;
        if (n_minus > 0)
        {
            printf("-: %d\n", n_minus);
            n_minus = 0;
        }
    }
    else
    {
        n_minus++;
        if (n_plus > 0)
        {
            printf("+: %d\n", n_plus);
            n_plus = 0;
        }
    }
    printf("a = ");
    scanf("%d", &a);
}
```

/*після циклу потрібна ще одна перевірка*/

```
if (n_minus > 0)
    printf("-: %d\n", n_minus);
if (n_plus > 0)
    printf("+: %d\n", n_plus);
return 0;
```

```
}
```

Приклад 4. В послідовності цілих чисел знайти довжини ділянок суворої монотонності. Наприклад, в послідовності 1, 2, 3, 2, 2 є дві ділянки суворої монотонності, а саме 1, 2, 3 і 3, 2.

```
#include <stdio.h>
int main( )
{
    int a, /* значення попереднього елемента послідовності */
    b, /* значення поточного елемента послідовності */
    n; /* довжина ділянки суворої монотонності */
    printf("a = ");
    scanf("%d", &a);
    if (a != 0)
    {
        n = 1;
        printf("b = ");
        scanf("%d", &b);
        while (b != 0)
        {
            /* ділянка монотонного зростання */
            while (b != 0 && b > a)
            {
                n++;
                a = b;
                printf("b = ");
                scanf("%d", &b);
            }
```

```
            if (n > 1)
            {
                printf ("n = %d\n", n);
                n = 1;
            }
            /* ділянка монотонного спадання */
            while (b != 0 && b < a)
            {
                n++;
                a = b;
                printf("b = ");
                scanf("%d", &b);
            }
            if (n > 1)
            {
                printf ("n = %d\n", n);
                n = 1;
            }
            /* переглядаємо всі однакові елементи, що йдуть підряд */
            while (b != 0 && b == a)
            {
                printf("b = ");
                scanf("%d", &b);
            }
        }
    }
    return 0;
}
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися із теоретичним відомостями і методичними вказівками до виконання лабораторної роботи.

2. Скласти блок-схему алгоритму і програми.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Завдання мають бути виконані без застосування масивів. Елементи послідовностей мають генеруватися за допомогою функції генерації випадкових чисел.

1. Дані натуральні числа n, a_1,..., a_n. Визначити кількість членів a_k послідовності a_1,..., a_n, які є квадратами парних чисел.

2. Дані натуральні числа n, a_1,..., a_n. Визначити кількість членів a_k послідовності a_1,..., a_n, що задовольняють умові $2^k < k*a_k$.

3. Дані натуральні числа n, a_1,..., a_n. Визначити кількість членів a_k послідовності a_1,..., a_n, які мають парні порядкові номери і є непарними числами.

4. Дані натуральні числа n, a_1,..., a_n. Знайти ті члени a_k послідовності a_1,..., a_n, які при діленні на 7 дають залишок 1, 2 або 5.

5. Дано натуральне число n, цілі числа a_1,..., a_n. Знайти кількість і суму тих членів даної послідовності, які діляться на 5 і не діляться на 7.

6. Дані натуральні числа n, p, цілі числа a_1,... a_n. Отримати добуток членів послідовності a_1,..., a_n, кратних p.

7. Дано натуральне число n, дійсні числа a_1,..., a_n. Отримати суму членів, що належать відрізку [3, 7], а також число таких членів.

8. Дано натуральне число n, дійсні числа a_1,..., a_n. Отримати кількість від'ємних членів і членів, що належать відрізку [1, 2].

9. Дано натуральне число n, цілі числа a_1,... a_n. Обчислити кількість членів, більших 7.

10. Дано натуральне число n, цілі числа p, a_1,... a_n. Якщо в послідовності a_1,..., a_n є хоча б один член, рівний p, то отримати суму всіх членів, що йдуть за першим таким членом; інакше відповіддю повинно бути число p.

11. Дані натуральне число n, b_1,... b_n. Обчислити $f(b_0) + f(b_1) + ... + f(b_n)$, де

$f(b_i) = b_i^2$, якщо b_i кратне 3,

$f(b_i) = b_i$, якщо b_i при діленні на 3 дає залишок 1,

$f(b_i) = [b_i/3]$, в інших випадках.

12. Дані цілі числа a, n, x_1,..., x_n (n > 0). Визначити, яким за рахунком йде в послідовності x_1,..., x_n член, рівний a. Якщо такого члена немає, то відповіддю повинне бути число 0.

13. Дано натуральне число n, дійсні числа a_1,...,a_n. Отримати min (a_2, a_4, ...) + max (a_l, a_3, ...).

14. Дано натуральне число n, дійсне число x. Серед чисел $e^{\cos(x^{2k})}\sin(x^{3k})$, k = 1,..., n знайти найближче до будь-якого цілого.

15. Дано натуральне число n, дійсні числа a_1,...,a_n. Отримати всі натуральні *i* (2 <= *i* <= n-1) для яких $a_{i-1} < a_i < a_{i+1}$.

16. Нехай $x_l = 0.3$; $x_2 = -0.3$; $x_i = i + \sin(x_{i-2})$, *i* = 3, 4,..., n. Серед x_1,..., x_n знайти найближче до будь-якого цілого.

17. Нехай $a_i = \frac{i-1}{i+1} + \sin\frac{(i-1)^3}{i+1}, i = 1,2,...$ Дано натуральне n. Серед a_1,..., a_n знайти всі додатні числа, серед додатних вибрати найменше.

18. Дані натуральне число n, дійсні числа a_1,...,a_n. В послідовності a_1,..., a_n визначити кількість сусідств двох додатних чисел.

19. Розглядається послідовність a_1,..,a_n, де $a_k = \sin^2(3k+5) - \cos(k^2 - 15)$. Визначити, скільки членів послідовності з номерами 1, 2, 4, 8, 16,... мають

значення, менші ніж 0,25.

20. Дано натуральне число n, дійсні числа $a_1,...,a_n$. В послідовності $a_1,...,a_n$ визначити кількість сусідств двох чисел різного знаку.

21. Дано натуральне число n, дійсні числа $x_1,...,x_n$. Знайти $\max(|x_l|,..., |x_n|)$.

22. Розглядається послідовність $a_1,.., a_n$, де $a_1 = 0.01$, $a_k = \sin(k+a_{k-1})$, k = 2,..., n. Визначити, скільки членів послідовності з номерами 1, 2, 4, 8, 16,... мають значення, менші ніж 0.75.

23. Дано натуральне число n, дійсні числа $a_1,...,a_n$. В послідовності $a_1,...,a_n$ визначити кількість сусідств двох чисел одного знаку, причому модуль першого числа повинен бути більше модуля другого числа.

24. Дано натуральне число n. Отримати всі такі натуральні q, щоб n ділилося на q і не ділилося на q^3.

25. Дані натуральні числа m, n (m≠0, n≠0). Отримати всі їх натуральні загальні кратні, менші або рівні m * n.

26. Дані цілі числа m, n (m≠0, n≠0). Отримати всіх їхні загальні дільники (додатні та від'ємні).

27. Дані дійсні числа x, y (x>0, y>1). Отримати ціле число K (додатне, від'ємне або рівне нулю), що задовольняє умові $y^{k-1} <= x < y^k$.

28. Дано натуральне число n. Обчислити добуток перших n множників $\frac{2}{1}\times\frac{2}{3}\times\frac{4}{3}\times\frac{4}{5}\times\frac{6}{5}\times\frac{6}{7}\ldots.$

29. Дано натуральне число n, дійсні числа $a_1,...,a_n$. З'ясувати, чи утворюють зростаючу послідовність числа $a_l,...,a_n$.

30. Дані натуральне число n, цілі числа $a_1,...,a_n$. Знайти номер останнього непарного члена послідовності $a_1,..., a_n$. Якщо таких членів немає, вивести відповідне повідомлення.

ЛАБОРАТОРНА РОБОТА № 4

ОДНОВИМІРНІ МАСИВИ

Мета роботи: оволодіння навичками роботи із структурованими типами даних – масивами, особливостями введення, обробки і виведення.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Масиви є одними з поширених структур даних. Окремі змінні в масиві називаються елементами. Масив складається з елементів одного типу, що називається базовим, тому структура масиву є однорідною. Базовий тип даних може бути як простим, так і структурованим, тобто елементами масиву можуть бути числа, символи, рядки, структури, а крім того, й масиви. Кількість елементів масиву є фіксованою, тому обсяг пам’яті залишається незмінним. З кожним елементом масиву пов'язаний один або декілька індексів, за якими можна звернутися до значення елементу. Вони однозначно визначають місце елемента в масиві та прямий доступ до нього. Індекси масиву належать до певного порядкового типу, тому їх можна обчислювати. Індекс має бути цілою константою або арифметичним виразом цілого типу. Число індексів, що визначають елемент масиву, називається розмірністю масиву, а кількість елементів в масиві – розміром масиву. Залежно від кількості індексів розрізняють одномірні та багатовимірні масиви. Допустима кількість індексів масиву (його розмірність) і діапазони зміни їх значень встановлюються мовою програмування. Згідно стандарту мови С, розмірність масиву не може перевищувати 31, а нумерація індексів починається з нуля, тобто індекс змінюється від 0 до n-1, де n – кількість значень індексу.

Цієї інформації достатньо як для доступу до елементів масиву, так і для контролю над тим, щоб значення індексів не виходили за встановлені діапазони. Доступ до будь-якого і-го елемента в одновимірному масиві здійснюється за

адресою.

Масив зберігається у пам’яті комп’ютера таким чином, що всі елементи розміщуються в суміжних ділянках пам’яті підряд, починаючи з адреси, що відповідає початку масиву. Елементи одновимірного масиву розміщуються в послідовності: a_0, a_1, ..., a_{n-1}. Вся інформація, що необхідна для управління масивом, задається при його описі в програмі. Опис містить ім’я масиву, тип елементів, який однозначно визначає довжину елемента, діапазони зміни індексів або число значень індексів, якщо нижні границі індексів фіксовані. Таким чином, загальна кількість елементів масиву і розмір пам’яті для масиву повністю визначаються описом масиву. Транслятор виділяє необхідну пам’ять і будує керуючий блок масиву – дескриптор або інформаційний вектор, наприклад, такого типу (для одновимірного масиву): тип структури, адреса початку масиву A, тип елементу (довжина елемента масиву L), нижня границя індексу, верхня границя індексу.

Масив в мові С визначається таким чином:

тип описувач [конст-вираз-1] [конст-вираз-2] .. [констний-вираз-n];

де *тип* – один з базових типів, тип (enum), що перераховує, структура (struct) або об'єднання (union); *описувач* – це ідентифікатор масиву, який може бути простою змінною, або описом змінної типу, що перераховує, описом структури, об'єднання, функції або покажчика; *констний-вираз-1*, *константний-вираз-2*, *константний-вираз-n* – кількість елементів для і-го виміру масиву.

Кількість пар квадратних дужок дорівнює розмірності масиву.

Приклади оголошення масивів:

1. `int a[3]; /* Оголошений масив змінних цілого типу, що складається з трьох елементів */`
2. `char N[20]; /* Оголошений масив змінних символьного типу, що складається з двадцяти елементів */`
3. `float mass[30]; /* Оголошений масив змінних дійсного`

```
типу, що складається з 30 елементів */
```

4. `double v[1500]; /* Оголошений масив змінних дійсного типу подвійної точності, що складається з 1500 елементів */`
5. `int a[5]={9, 33, -23, 8, 1}; /* a[0]=9, a[1]=33, a[2]=-23, a[3]=8, a[4]=1 */`
6. `float b[10] = {1.5, -3.8, 10}; /* b[0]=1.5, b[1]=-3.8, b[2]=10, b[3]=b[4]=…=b[9]=0 */`
7. `char s[5]="Фото"; /* s[0]= 'Ф', s[1]='о', s[2]='т', s[3]= 'о', s[4]= '\0' */`
8. `char code[]="abc"; /* code[0]='a', code[1]= 'b', code[2]= 'c', code[3]='\0' Оголошено масив символів з чотирьох елементів. Четвертим елементом є символ '\0', який завершує усі рядки символів. Якщо рядок є коротше за оголошений розмір масиву символів, то решта елементів масиву ініціалізується нулем (символом '\0') */`

Масиви в програмах можна оголошувати також зі створенням типу користувача:

```
typedef <тип_даних> <ім'я_типу> [<розмір_масиву>];
<ім'я_типу> <ім'я_масиву>;
```

ПРИКЛАД 1. Створити тип з ім'ям `mass` як масив з 10-ти додатних цілих чисел і оголосити два масиви – a та b – створеного типу:

```
typedef unsigned short mass[10];
mass a, b;
```

Для оголошення масивів можна використовувати типізовані константмасиви, які дозволяють водночас і оголосити масив, і задати його значення як константи:

```
const <тип_даних> <ім'я_масиву> [<розмір_масиву>] = {<значення_елементів_масиву>};
```

ПРИКЛАД 2. Створити константи-масиви:

```
const int arr[5]={9, 3, -7, 0, 123}; /*масив з 5-ти цілих чисел */
const float f[5]={1.5, 6, 8, 7.4, -1.125}; /*масив з 5-ти дійсних чисел */
const C[5]={15, -6, 546}; /*масив з 5-ти цілих чисел типу int, де C[0] = 15, C[1] = -6, C[2] = 546, C[3] = 0 і C[4] = 0 */
```

Як видно з прикладів, елементи масиву можуть ініціалізуватися. Для цього потрібно після опису масиву задати символ "=" і вказати у фігурних дужках список значень елементів масиву. Елементи списку відділяються один від одного комами. Список не повинен містити порожніх елементів, проте можуть бути задані не усі елементи масиву, наприклад

```
int a[10] = {1,2,3};
```

В цьому випадку перші три елементи набудуть значень відповідно до 1, 2 і 3, а іншим елементам масиву компілятор С присвоїть значення 0.

Компілятор С видає повідомлення про помилку, якщо кількість значень елементів в списку ініціалізації перевищує оголошену розмірність масиву.

Константний вираз в квадратних дужках може бути опущений, якщо в оголошенні масиву його елементи перерахувалися і ініціалізувалися, наприклад, оголошення

```
int a[] = {1,2,3,4,5};
```

задає масив з п'яти елементів і привласнює йому початкові значення.

Розмір масиву можна також опустити, або коли масив оголошується як формальний параметр функції, або це оголошення є посиланням на оголошення масиву у іншому місці програми.

Оскільки компілятор С виділяє для масивів місце в пам'яті відповідно до кількості елементів масиву і типу елемента масиву. Обсяг пам'яті, виділеної під масив, можна визначити за допомогою операції sizeof. Кількість елементів в масиві можна визначити за допомогою наступного виразу:

```
sizeof имя-массива / sizeof (тип-массива)
```

наприклад:

```
int nmatrix;
nmatrix = sizeof matrix / sizeof (float);
```

Індексація елементів масиву в С починається з нуля і, таким чином, останній елемент масиву має індекс, на 1 менший, ніж кількість елементів масиву.

ПРИКЛАД 3. Поелементне введення масиву

```
int i,a[n];
for(i=0; i<n; i++)
{
   printf("a[%d] =", i);
   scanf("%d", &a[i]);
}
```

ПРИКЛАД 4. Заповнення масиву псевдовипадковими числами

```
#include <stdio.h>
#include <stdlib.h>
#define N 10
int main( )
{
   int a[N], /* масив а розміру N */
   i; /* лічильник */
   for (i = 0; i < N; i++)
   {
```

```
        a[i] = rand()%100;
        printf("%d\n", a[i]);
    }
    return 0;
}
```

Пам'ять комп'ютера є (у спрощеному виді) масивом послідовно пронумерованих і адресованих осередків, що містять машинні команди і дані, з якими можна працювати окремо або зв'язними ділянками.

Один байт може зберігати значення типу char, два байти – тип short, чотири байти – типи int, long або float, вісім байт – тип double і десять байт – тип long double.

Кожна змінна в програмі – це об’єкт, що має ім’я та значення. За іменем можна звернутись до змінної та отримати її значення. З точки зору машинної реалізації ім’я змінної відповідає адресі тієї ділянки пам’яті, яка виділена для неї, а значення змінної – вмісту цієї ділянки пам’яті.

Співвідношення між іменем та адресою умовно представлено на рис. 1.

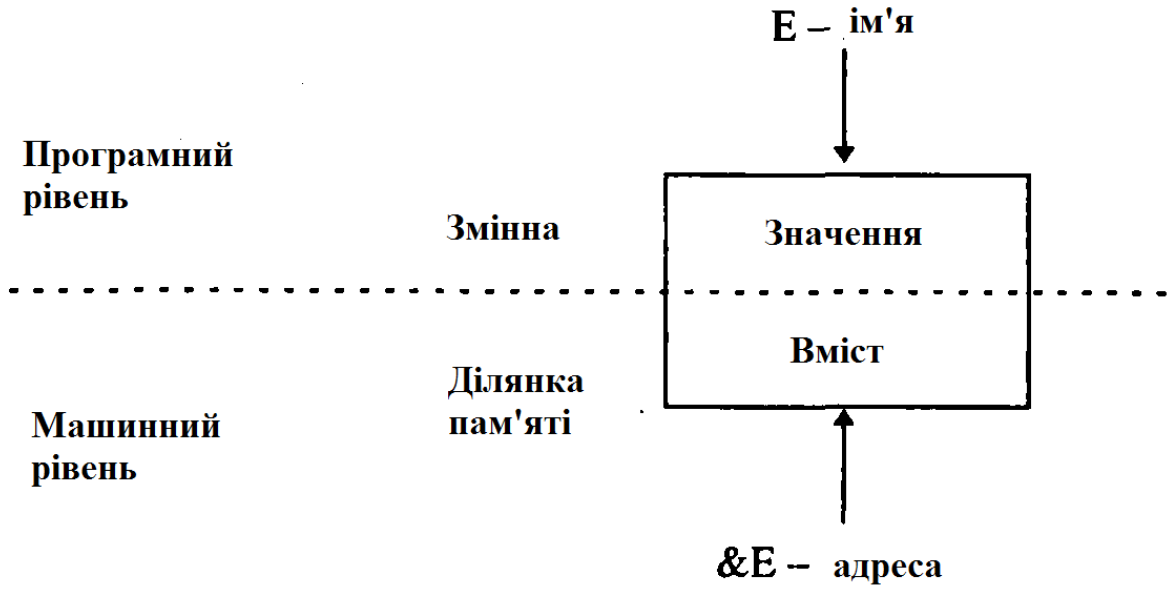


Рис. 1 – Співвідношення між іменем та адресою

Для отримання адреси в явному вигляді застосовують операцію &. Операція & може бути застосована тільки до об’єктів, які мають ім’я та розміщені в пам’яті. На рис. 2 проілюстровано зв'язок між іменами, адресами та значеннями змінних. Нехай передбачається, що в програмі використана така послідовність визначень (з ініціалізацією): char ch = ‘G’; int date = 1937; float summa = 2.015E -6;.

У С існує спеціальний тип змінних, що називається вказівником, призначений для зберігання адреси об'єкту деякого типу. Оголошення вказівника має наступний формат:

*тип *описувач;*

де *тип* – це тип об'єкту, на який може вказувати ця змінна; *описувач* – це ідентифікатор вказівника, який може бути простою змінною, масивом, функцією або вказівником. Описувач може також містити перед ідентифікатором кваліфікатори const або volatile.

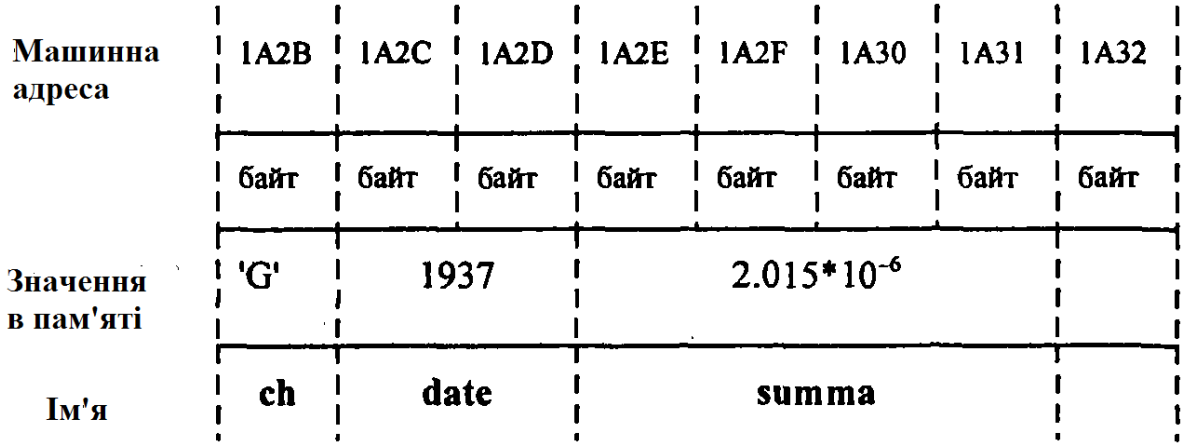


Рис. 2 – Різні типи даних в пам’яті ЕОМ

Розмір пам'яті для вказівника і формат представлення адреси залежать від використовуваної комп'ютерної платформи і операційного середовища.

Подібно іншим змінним змінна типу вказівник має власне ім’я, власну адресу в пам’яті та значення. Адреса вказівника може бути отримана за

допомогою операції &. Вираз &ім’я_вказівника визначає, де в пам’яті розміщений вказівник. Співвідношення між іменем, адресою та вказівником представлено на рис. 3.

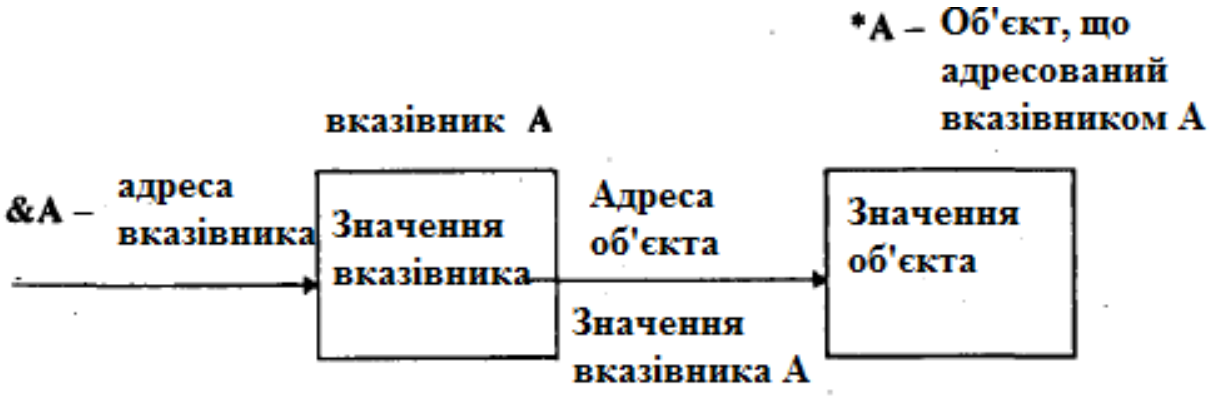


Рис. 3 – Ім’я, адреса та значення вказівника

Для ініціалізації вказівників можна використовувати константу з ім'ям NULL, визначену у файлі stdio.h стандартної бібліотеки. Ця константа визначає нульове значення вказівника, при якому він не посилається на жодний програмний об'єкт.

Приклади оголошення вказівників:

1. `int i, *pi; /* Оголошена ціла змінна і та вказівник pi на змінну цілого типу */`

2. `char *(*pa1); /* pa1 оголошено як вказівник на вказівник на змінну символьного типу */`

Змінна типу масив бере участь у виразах як константа-вказівник на значення заданого іменем-типа типу.

Як вже говорилося, символьні рядки представляються в С за допомогою масивів типу char. Якщо описати змінну **pmessage** як

```
char *pmessage;
```

то присвоєння

```
pmessage = "рядок символів";
```

помістить в pmessage вказівник на символьний рядок. Це не копіювання рядка; тут беруть участь тільки вказівники. У мові С не передбачені будь-які операції для обробки усього рядка символів як єдиного об’єкту.

Існує важлива відмінність між наступними визначеннями:

```
char amessage[] = "рядок символів"; /* масив */
char *pmessage  = "рядок символів"; /* вказівник */
```

amessage – це масив, який має такий об'єм, що в ньому якраз поміщається задана послідовність символів і '\0'. Окремі символи усередині масиву можуть змінюватися, але **amessage** завжди посилається на одне і те ж місце в пам'яті. Вказівник **pmessage** ініціалізований посиланням на символьний рядок. Його значення можна змінити, і тоді він посилатиметься на що-небудь інше.

Для вказівників допустимі наступні операції:

- надання значення вказівника іншому вказівнику того ж типу;
- присвоєння вказівнику значення NULL;
- порівняння значення вказівника зі значенням NULL;
- операція адресації;
- операція розкриття посилання (непряма адресація);
- складання і віднімання вказівника і цілого виразу;
- складання, віднімання і порівняння двох вказівників, що посилаються на елементи одного і того ж масиву.

Оператор присвоєння присвоює значення одного вказівника іншому вказівнику, або присвоює вказівнику значення NULL, наприклад:

```
int *pa, *pb = NULL;
pa = pb;
```

Будь-який вказівник можна також порівняти на рівність або нерівність з NULL.

Оператор адресації "&" видає адресу змінної або елементу масиву, наприклад:

```
int x[] = {1,2,3}, *pxi;
pxi = &x[1];
```

Оператор розкриття посилання або непрямої адресації "*" видає об'єкт, на який посилається цей вказівник, наприклад:

```
int z =0, w, *pz;
pz = &z;
w=*pz;
```

Оператори "&" і "*" мають той же пріоритет, що і оператори "++" і "--", тобто вищий пріоритет, ніж арифметичні оператори.

Приклади використання операторів присвоєння, порівняння, адресації і розкриття посилання для вказівників:

```
int x = 1, y = 2; /* Оголошення і ініціалізація змінних x і y типу int */
int *ip1, *ip2 = NULL; /* Оголошення вказівників ip1 і ip2 на змінні типу int і ініціалізація ip2 */
ip1 = &x; /* Вказівник ip1 вказує на x */
y = *ip1; /* y стало рівним 1 */
*ip1 += 10; /* Значення x стало рівним 11 */
if (ip2==NULL)
    ip2=ip1; /* Вказівнику ip2 присвоєно значення вказівника ip1 */
++*ip1; /* Значення x стало рівним 12 */
printf ("Значення для ip2=%i", *ip2); /* Буде виведено 12 */
```

У мові С існує сильний взаємозв'язок між вказівниками і масивами. Будь-яку операцію, що можна виконати за допомогою індексів масиву, можна зробити і за допомогою вказівників, причому варіант з вказівниками зазвичай виявляється швидшим. Опис:

```
int a[10], *pa;
```

визначає масив розміром 10, тобто набір із 10 послідовних об'єктів, що називаються a[0], a[1], ..., a[9] та pa – вказівник на цілий тип. Запис **a[i]** відповідає елементу масива через **i** позицій від початку. Присвоювання:

```
pa = &a[0];
```

призводить до того, що **pa** вказує на нульовий елемент масиву **a**; це означає, що **pa** містить адресу елементу **a[0]**. Якщо **pa** вказує на деякий певний елемент масиву **a**, то за визначенням **pa+1** вказує на наступний елемент, і взагалі **pa-i** вказує на елемент, що стоїть на відстані **i** позицій до елементу, на який вказує **pa**, а **pa+i** – на елемент, що стоїть на **i** позицій далі. Таким чином, якщо **pa** вказує на **a[0]**, то ***(pa+1)** посилається на вміст **a[i]**, **pa+i** – адреса **a[i]**, а ***(pa+1)** – вміст **a[i]**. Оскільки вказівник визначається для конкретного типу змінної, ці зауваження справедливі незалежно від типу змінних в масиві **a**.

Таким чином, для вказівників визначені операції додавання цілої величини та віднімання цілої величини.

При використанні вказівників масивів необхідно звернути увагу на одну відмінність між ім'ям масиву і вказівником, яку необхідно мати на увазі. Вказівник є змінною, так що операції **pa=a** та **pa++** мають сенс. Але ім'я масиву є вказівником-константою, тому його значення змінити неможливо: конструкції типу **a=pa** або **a++,** а також **pa=&a** будуть неправильними.

Якщо вказівники вказують на елементи одного і того ж масиву, то для них можна використовувати операції відношення, а також додавання та віднімання. Нехай **p** і **q** – вказівники відповідно на i-й та j-й елементи масиву (i<j). Тоді **p** < **q** істинно, а **q - p + 1** – число елементів від і до j включно.

Окрім звичайної пам'яті (стека), в якій автоматично розміщуються змінні за їхнього оголошення, існує ще й динамічна пам'ять (heap − купа), в якій змінні можуть розміщуватися динамічно. Це означає, що пам'ять виділяється під час

виконання програми, й лише тоді, коли у програмі зустрінеться спеціальна інструкція. Основна потреба у динамічному виділенні пам'яті виникає, коли розмір або кількість даних заздалегідь є невідомі, а визначаються в процесі виконання програми.

Варіанти виділення динамічної пам'яті в мові С існують різні.

Для виділення динамічної пам'яті використовується функція **malloc**:

```
void *malloc(size_t size);
```

(від англ. "memory allocation" – виділення пам'яті), яка робить запит до ядра операційної системи щодо виділення ділянки пам'яті заданої кількості байтів. Єдиний аргумент цієї функції size − кількість байтів, яку треба виділити. Функція повертає вказівник на початок виділеної пам'яті. Якщо для розміщення заданої кількості байтів пам'яті недостатньо, функція **malloc** повертає NULL. Вміст ділянки лишається незмінним, тобто там може залишитися "бруд". Якщо аргумент size дорівнює 0, функція повертає NULL. Наприклад, команда:

```
int *a = (int*) malloc(sizeof(int));
```

виділяє пам'ять під зберігання цілого числа й адресу початку цієї ділянки пам'яті записує у вказівник **a**.

Виділення пам'яті під 10 дійсних чисел за допомогою цієї функції:

```
float *a = (float*) malloc(sizeof(float)*10);
```

Функція

```
void *calloc(size_t num, size_t size);
```

виділяє блок пам'яті розміром num x size (під num елементів по size байтів кожен) і повертає вказівник на виділений блок. Кожен елемент виділеного блока ініціалізується нульовим значенням (на відміну від функції **malloc**). Функція **calloc** повертає NULL, якщо не вистачає пам'яті для виділення нового блока, або якщо значення num чи size дорівнюють 0.

Виділення пам'яті під 10 дійсних чисел за допомогою функції **calloc**:

```
float *a = (float*) calloc(10,sizeof(float));
```

Функція

```
void *realloc(*bl, size);
```

змінює розмір виділеного раніше блока пам’яті з адресою *bl на новий обсяг, який становитиме size байтів. Якщо змінення відбулося успішно, функція **realloc** повертає вказівник на виділену ділянку пам’яті, а інакше повертається NULL. Якщо *bl є NULL, функція **realloc** виконує такі самі дії, що й **malloc**. Якщо size є 0, виділений за адресою *bl блок пам’яті звільнюється, і функція повертає NULL. Для ілюстрації роботи функції **realloc** наведемо приклад змінення розміру раніш виділеної пам’яті під одновимірний масив з 10-ти дійсних елементів до 20-ти елементів:

```
a = (float*) realloc(a, 20);
```

Пам’ять, виділену за допомогою функцій **malloc** і **calloc**, звільнюють за допомогою функції **free**:

```
void free(*bl);
```

де *bl – адреса початку виділеної раніше пам’яті, наприклад:

```
free(a);
```

Використання згаданих функцій разом з вказівниками надає можливість керувати розподілом динамічної пам’яті.

Якщо не звільняти динамічно виділену пам’ять, коли вона стає більше не потрібною, у системі може виникнути нестача вільної пам’яті. Іноді це називають «витоком пам’яті».

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися з загальними відомостями даної лабораторної роботи, вивчивши:

- способи опису і обробки одновимірних масивів (статичних та динамічних);

- способи ініціалізації, копіювання, введення і виведення масивів.

2. Скласти блок-схему алгоритму і програми.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Елементи масивів мають генеруватися за допомогою функції генерації випадкових чисел.

1. Надрукувати ті елементи масиву A(50), індекси яких є степенями двійки (1, 2, 4, 8, 16, ...).

2. Знайти максимальний і мінімальний елементи масиву X і поміняти їх місцями.

3. Визначити суму елементів масиву A(20), які кратні числу три.

4. Надрукувати ті елементи масиву A(50), індекси яких є квадратами натуральних чисел (1, 4, 9, 16, 25, ...).

5. Скласти програму циклічного зсуву елементів масиву A(10) на 5 позицій вліво.

6. Заданий цілочисельний вектор A(10). Побудувати вектор B(10), прийнявши першими його компонентами всі від'ємні компоненти вектора A (із збереженням порядку проходження), а в якості останніх – всі додатні компоненти вектора A. Роздрукувати A і B.

7. Надрукувати ті елементи масиву A(50), індекси яких є числами Фібоначчі (1, 2, 3, 5, 8, 13, ...).

8. Скласти програму циклічного зсуву елементів масиву B[10] на 6 позицій вправо.

9. Елемент вектора називається локальним мінімумом, якщо він строго менше двох своїх сусідів. Підрахувати кількість локальних мінімумів вектора X(25). Роздрукувати значення локальних мінімумів та їхніх сусідів, а також

порядкові номери цих елементів у векторі X.

10. Дана послідовність з 20 різних цілих чисел. Знайти суму чисел цієї послідовності, розташованих між максимальним і мінімальним числами (в суму включити і обидва ці числа).

11. Дані дійсні числа $a_1, a_2, ..., a_{30}$ – кількість опадів, що випали в Києві протягом 30 років. Треба обчислити середню кількість опадів і відхилення від середнього значення для кожного року.

12. Дані дійсні числа $a_1, ..., a_{20}$. Отримати числа $b_1, ..., b_{20}$, де b_i – середнє арифметичне всіх членів послідовності $a_1, ..., a_{20}$, окрім a_i ($i = 1, 2, \ldots, 20$).

13. Дані дійсні числа $a_1, ..., a_n$; $b_l, ..., b_n$. Обчислити $(a_1 + b_n)*(a_2 + b_{n-1})*...*(a_n + b_1)$.

14. Дані дійсні числа $a_l, a_2, ..., a_{2n}$. Отримати $a_1, a_{n+1}, a_2, a_{n+2},...,a_n, a_{2n}$.

15. Дані дійсні числа $a_1, ..., a_n$. Якщо в результаті заміни від`ємних членів послідовності $a_1, ..., a_n$ їхніми квадратами члени будуть утворювати неспадну послідовність, то отримати суму членів початкової послідовності; інакше отримати суму зміненої послідовності.

16. Дані дійсні числа $a_1, a_2, ..., a_{2n}$. Отримати $a_1, a_{2n}, a_2, a_{2n-1}, a_3,...,a_n, a_{n+1}$.

17. Дані цілі числа $a_1, ..., a_{20}$. Отримати нову послідовність, вилучивши з початкової всі члени із значенням $\max(a_1, ..., a_{20})$.

18. Дані дійсні числа $a_1, a_2, ..., a_{2n}$. Отримати $a_1+a_{2n}, a_2+a_{2n-1}, ..., a_n+a_{n+1}$.

19. Дані цілі числа $a_1, ..., a_{20}$. Отримати нову послідовність, вилучивши з початкової всі члени із значенням $\min(a_1, ..., a_{20})$.

20. Дані дійсні числа $a_1, a_2, ..., a_{2n}$. Отримати $a_{n+1}, a_{n+2},...,a_{2n}, a_n, a_{n-1}, ..., a_1$.

21. Дані цілі числа $a_1, ..., a_n$. Всі члени послідовності з парними номерами, попередні першому за порядком члену із значенням $\max(a_1, ..., a_n)$, домножити на $\max(a_1, ..., a_n)$.

22. Дані цілі числа $a_1, ..., a_n$, кожне з яких відмінне від нуля. Якщо в послідовності від'ємні і додатні члени чергуються (+, -, +, -, ... або -, +, -, +, ...),

то відповіддю повинна служити сама початкова послідовність. Інакше отримати всі від'ємні члени послідовності, зберігши порядок їхнього проходження.

23. Дано натуральне число m, дійсні числа $a_1, ..., a_{20}$, які попарно різні. В послідовності $a_1, ..., a_{20}$ поміняти місцями найбільший член і член з номером m (m<=20).

24. Дані дійсні числа $a_1, ..., a_{20}$. Отримати max($a_1+a_{20}, a_2+a_{19}, ..., a_{10}+a_{11}$).

25. Дані дійсні числа $a_1, ..., a_{20}$. Перетворити цю послідовність за правилом: більше з a_i і a_{10+i} (i = 1, ..., 10) прийняти як нове значення a_i, а менше – як нове значення a_{10+i}.

26. Дані цілі числа $a_1, ..., a_{20}$. Найменший член послідовності $a_1, ..., a_{20}$ замінити цілою частиною середнього арифметичного всіх членів, інші члени залишити без змін. Якщо в послідовності декілька членів із значенням min(a_1, ..., a_{20}), то замінити останній за порядком.

27. Дані цілі числа $a_1, ..., a_{20}$. Отримати нову послідовність з 20 цілих чисел, замінюючи a_i нулями, якщо $|a_i|$ = max($a_l, ..., a_{20}$), інакше замінюючи a_i одиницями.

28. Дані цілі числа $a_1, ..., a_{10}$; $b_1, ..., b_{10}$. Перетворити послідовність $b_1, ..., b_{10}$ за правилом: якщо a_i <= 0, то b_i збільшити в 10 раз, інакше b_i замінити нулем.

29. Дані дійсні числа $a_1, ..., a_{10}$. Потрібно домножити всі члени послідовності $a_1, ..., a_{10}$ на квадрат її найменшого члена, якщо a_1 > = 0, і на квадрат її найбільшого члена, якщо a_1 < 0.

30. Дані дійсні числа $a_1, ..., a_{10}$ (всі числа попарно різні). Поміняти в цій послідовності місцями найбільший і останній члени.

КОНТРОЛЬНІ ПИТАННЯ

1. Чи може бути будь-яким тип елементів масиву?
2. Чи можуть елементи деякого масиву бути числами 16, 16.17, 24.7,24?
3. Чи може масив містити один елемент? Жодного елемента?

4. Індекс масиву може бути типу **int** або **float**?
5. В чому полягає особливість використання прийомів програмування при обробці масивів ?
6. Чи можна переписати масив X в масив Y за допомогою оператора присвоєння Y=X?
7. Вказати особливості введення і виведення масивів.
8. Що являє собою пам’ять комп’ютера?
9. Що таке адрес комірки?
10. Що таке вказівник?
11. Наведіть синтаксис оголошення вказівника?
12. Яке тлумачення має значення вказівника NULL?
13. В який спосіб можна отримати адресу змінної?
14. Яке призначення операції &?
15. В який спосіб можна дізнатися значення, на яке посилається вказівник?
16. Поясніть відмінності поміж змінними a та b, оголошення яких має вигляд: а) int a; int b; б) int *a; float *b;
17. У чому полягає відмінність роботи функцій `malloc()` та `calloc()`?
18. Яка функція застосовується для звільнення пам’яті, виділеної функцією `calloc`?
19. Яким є призначення функції `realloc()`?

ЛАБОРАТОРНА РОБОТА № 5

ДВОВИМІРНІ МАСИВИ

Мета роботи: закріплення навичок роботи із структурованими типами даних – масивами; оволодіння навичками реалізації найпростіших алгоритмів роботи з масивами:

- підсумовування елементів масиву, діагональних елементів матриці, елементів рядків матриці, двох масивів;
- транспонування матриці;
- множення матриці на вектор, матриці на матрицю;
- видалення(включення) k-го рядка з(в) матриці(ю);
- перестановка рядків матриці;
- пошук мінімального (максимального) елемента в масиві;
- перетворення матриці в одновимірний масив і навпаки;
- сортування (впорядкування) масиву.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Багатовимірні масиви у мові С розглядаються як сукупність масивів меншої розмірності. Наприклад, двовимірний масив – сукупність одновимірних масивів (його рядків), тривимірний масив – це сукупність матриць, матриці – сукупності рядків, а рядок – сукупність елементів одновимірного масиву.

Елементи розташовуються в оперативній пам’яті таким чином, що швидше змінюються праві індекси, тобто елементи одновимірного масиву розташовуються підряд, двовимірного – по рядках, тривимірного – по матрицях, а матриці – по рядках.

Багатовимірний масив оголошується у програмі в такий спосіб:

```
<тип> <ім’я> [<розмір1>] [<розмір2>] … [<розмірN>];
```

Кількість елементів масиву дорівнює добуткові кількості елементів за кожним індексом. У прикладі

```
int a[3][4];
```

оголошено двовимірний масив з 3-х рядків та 4-х стовпчиків (12-ти елементів) цілого типу:

```
a[0][0], a[0][1], a[0][2], a[0][3],
a[1][0], a[1][1], a[1][2], a[1][3],
a[2][0], a[2][1], a[2][2], a[2][3];
```

Під масив надається пам’ять, потрібна для розташування усіх його елементів. Елементи масиву один за одним, з першого до останнього, запам’ятовуються у послідовно зростаючих адресах пам’яті так само, як і елементи одновимірного масиву. Наприклад, масив int a[3][4] зберігається у пам’яті у такий спосіб (рис. 1).

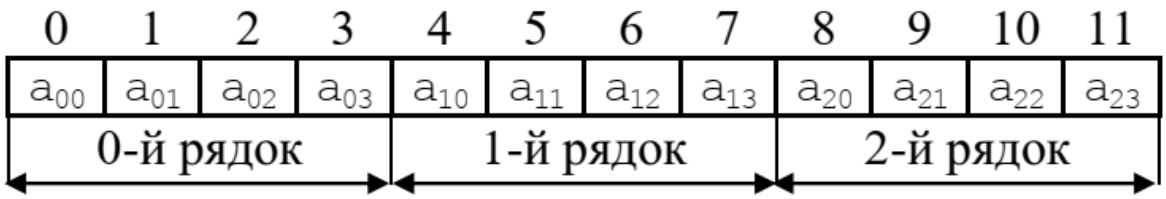


Рисунок 1 – Приклад зберігання двовимірного масиву у пам’яті комп’ютера

Приклади оголошення масивів:

```
float Mas1 [5][5]; // Матриця 5x5=25 елементів дійсного типу
char Mas2 [10][3]; // Двовимірний масив з 10x3=30 елементів символьного типу
double Mas3 [4][5][4]; // Тривимірний масив з 4x5x4=80 елементів дійсного типу
```

Приклади оголошення масивів з ініціалізацією:

```
int w[3][3]={ { 2, 3, 4 },{ 3, 4, 8 },{ 1, 0, 9 } }; /*
```

оголошено й ініціалізовано масив цілих чисел w[3][3]. Елементам масиву присвоєно значення зі списку: w[0][0]=2, w[0][1]=3, w[0][2]=4, w[1][0]=3 тощо. Списки, виокремлені у фігурні дужки, відповідають рядкам масиву./*

```
float C[4][3]={1.1, 2, 3, 3.4, 0.5, 6.8, 9.7, 0.9};
float C[4][3]={{1.1, 2, 3},{3.4, 0.5, 6.8},{ 9.7, 0.9}};
float C[4][3]={{1.1, 2},{3, 3.4, 0.5},{6.8},{9.7, 0.9}};
```

Записи другого й третього прикладів еквівалентні, і в них елементи останнього рядка не ініціалізовані, тобто невизначені. У таких випадках у числових масивах неініціалізовані елементи набувають значення 0 (рис. 2).

Розташування елементів у прикладах 2 та 3

1.1	2	3
3.4	0.5	6.8
9.7	0.9	0
0	0	0

Розташування елементів у прикладі 4

1.1	2	0
3	3.4	0.5
6.8	0	0
9.7	0.9	0

Рисунок 2 – Приклади розташування елементів масиву

Розмір масиву можна опустити, коли масив оголошується як формальний параметр функції, або це оголошення є посиланням на оголошення масиву у іншому місці програми. Проте для багатовимірного розміру може бути опущена тільки перша розмірність, наприклад:

```
int matrix [][5] = { {0, 1, 2, 3, 4},{5, 6, 7, 8, 9},{10, 11, 12, 13, 14} };
```

При звертанні до елемента масиву вказується ім’я масиву та індекс або індекси елемента. Кожний індекс береться у квадратні дужки, наприклад код:

```
float x;
```

```
x = matrix[0][2];
matrix[2][0] = - 1;
```

присвоює змінній x значення 2, а одинадцятий елемент масиву matrix набуде значення -1.

Масиви в програмах можна оголошувати також зі створенням типу користувача:

```
typedef <тип_даних> <i'мя_типу>[<розмір1>][<розмір2>] …
[<розмір n>];
<i'мя_типу> <i'мя_масиву> ;
```

Наприклад:

```
typedef unsigned short matr[10][7];
matr M1, M2;
```

Для оголошення масивів можна використовувати типізовані константмасиви, які дозволяють водночас і оголосити масив, і задати його значення як константи:

```
const <тип_даних> <ім'я_масиву>[<розмір1>][<розмір2>] …
[<розмір n>] = {<значення_елементів_масиву>};
```

але змінювати значення елементів константмасивів у програмі неприпустимо.

Наприклад:

```
const int arr[2][5] = { {9,3,0,12,-5} , {-7,23,2,4,0} };
```

В C++ можна використовувати перетин масиву (section). Це поняття подібно до поняття мінору матриці в математиці. Перетин формується внаслідок вилучення однієї чи кількох пар квадратних дужок. Пари квадратних дужок можна відкидати лише справа наліво й лише послідовно. Перетини масивів використовуються при організації обчислювального процесу в функціях мовою C++, розроблюваних користувачем.

Нехай задано

```
int x[5][3];
```

Якщо при звертанні до певної функції написати x[0], то передаватиметься нульовий рядок масиву x.

```
int y[2][3][4];
```

При звертанні до масиву y можна записати, наприклад, y[1][2] – і буде передаватися вектор з чотирьох елементів, а звертання y[1] надасть двовимірний масив розміром 3x4. Не можна писати y[2][4], маючи на увазі, що передаватиметься вектор, тому що це не відповідає обмеженню, накладеному на використання перетинів масиву.

ПРИКЛАД 1. Знаходження максимального елемента двовимірного масиву та його індексу (рис. 3).

```
float a[20][20];
int i, j, n, m, imax, jmax;
float max;
// введення матриці...
max=a[0][0];
imax=0; jmax=0;
for(i=0;i<n;i++)
    for(j=0;j<m;j++)
        if (a[i][j]<max)
        {
            max=a[i][j];
            imax=i;
            jmax=j;
        }
printf("\nmax=%f\n",max);
printf("\nimax=%d\n",imax);
printf("\njmax=%d\n",jmax);
```

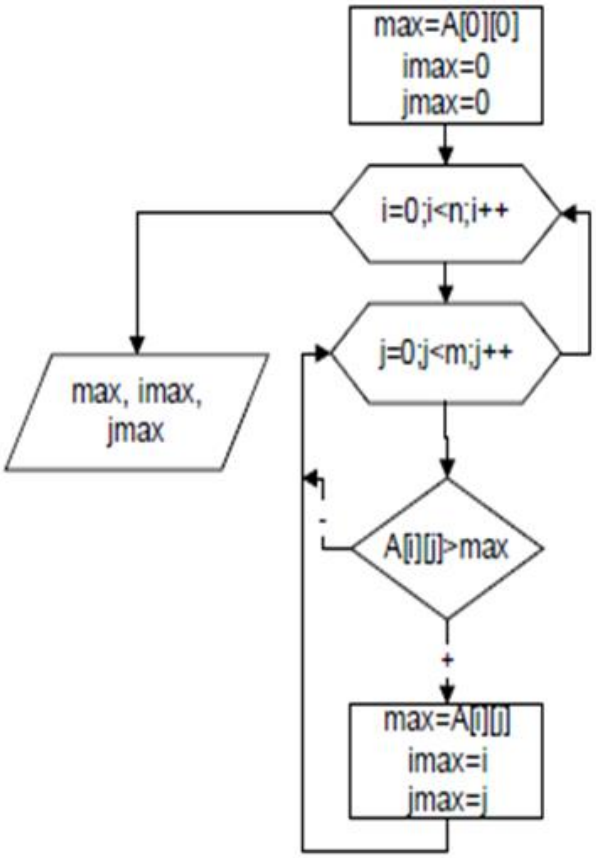


Рисунок 3 – Фрагмент блок-схеми алгоритму знаходження максимального елемента двовимірного масиву та його індексу

Важливо пам’ятати про властивості елементів матриць (рис. 4):

- якщо значення номеру рядка елемента співпадає з номером стовпця ($i = j$), то елемент розташовано на головній діагоналі матриці;
- якщо значення номеру рядка перевищує номер стовпця ($i > j$), то елемент знаходиться нижче головної діагоналі;
- якщо номер стовпця більше номеру рядка ($i < j$), то елемент знаходиться вище головної діагоналі.
- елемент лежить на побічній діагоналі *квадратної матриці*, якщо його індекси задовольняють рівності $i + j + 1 = n$;
- нерівність $i + j + 1 < n$ характерна для елемента, що знаходиться вище побічної діагоналі *квадратної матриці*;
- відповідно, елементу *квадратної матриці*, що лежить нижче побічної діагоналі відповідає вираз $i + j + 1 > n$.

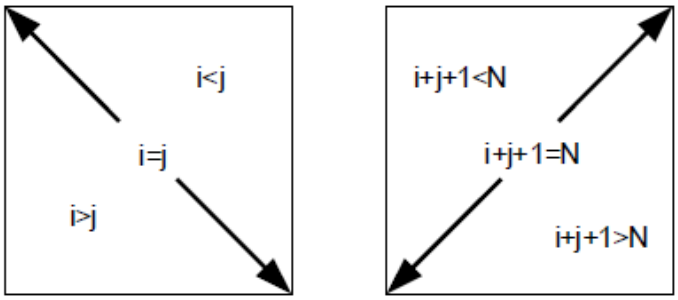


Рисунок 4 – Співвідношення індексів елементів матриці

ПРИКЛАД 2. Знаходження суми елементів матриці, що знаходяться вище головної діагоналі (рис. 5, 6).

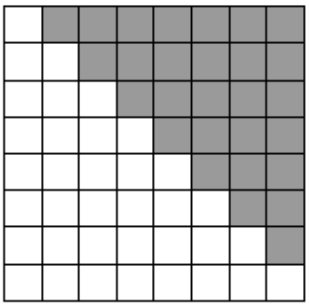


Рисунок 5 – Зображення елементів матриці, що знаходяться вище головної діагоналі

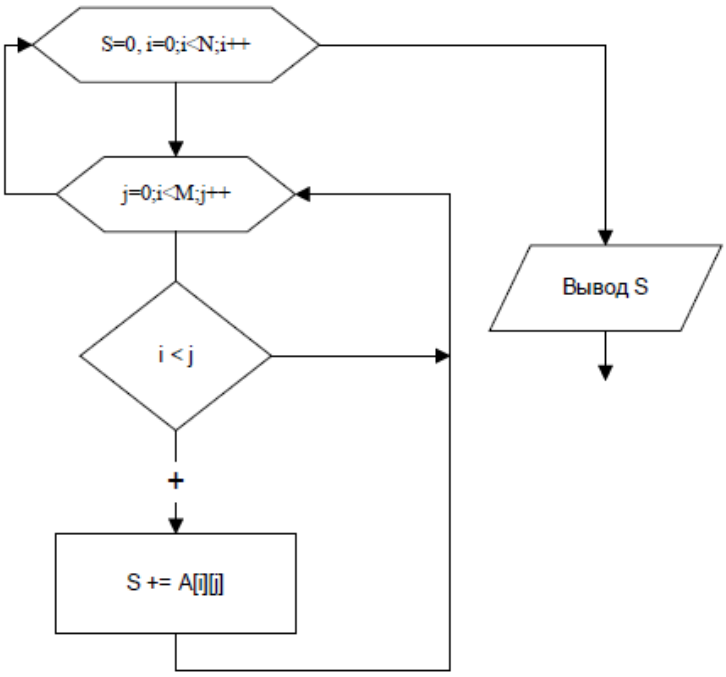


Рисунок 6 – Знаходження суми елементів матриці, що знаходяться вище головної діагоналі

```
int main()
{
    int S,i,j,N,M,a[20][20];
    cout<<"N=";cin>>N;
    cout<<"M=";cin>>M;
    cout<<"Input Matrix A"<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            cin>>a[i][j];
    for(S=i=0;i<N;i++)
        for(j=0;j<M;j++)
            if (j>i) S+=a[i][j];
     cout<<"S="<<S<<endl;
}
```

Інший варіант алгоритму представлений на рис. 7.

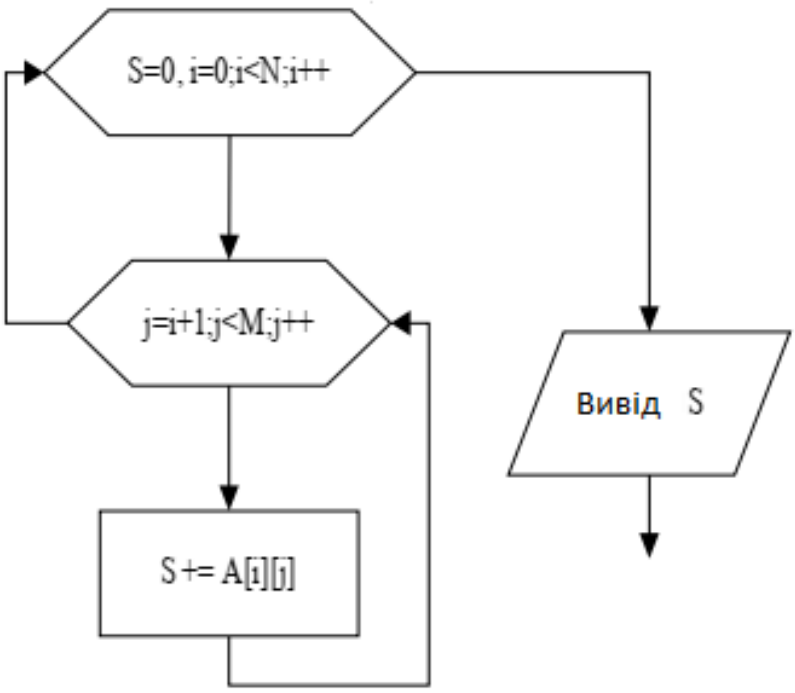


Рис. 7 – Фрагмент блок-схеми знаходження суми елементів матриці

ПРИКЛАД 3. Поміняти місцями елементи головної та побічної діагоналі квадратної матриці (рис. 8).

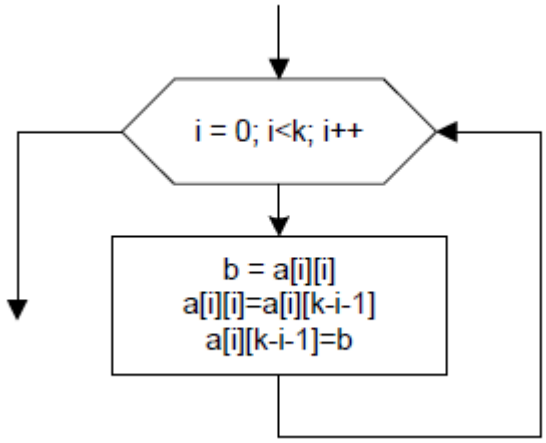
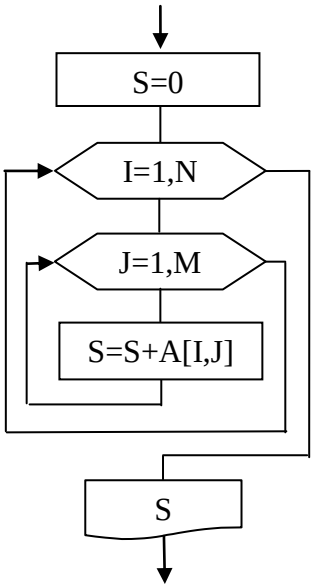


Рисунок 8 – Фрагмент блок-схеми обміну елементів головної і побічної діагоналей матриці

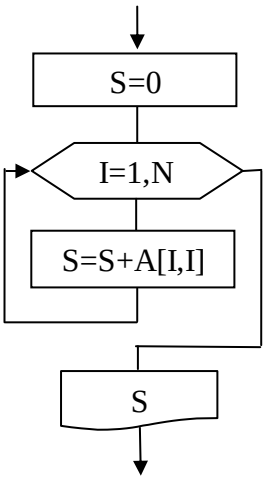
```
int main()
{
    int i,j,k;
    double b,a[20][20];
    cout<<"k=";
    cin>>k;
    cout<<"Input Matrix A"<<endl;
    for(i=0;i<k;i++)
        for(j=0;j<k;j++)
            cin>>a[i][j];
    for(i=0;i<k;i++)
    {
        b=a[i][i];
        a[i][i]=a[i][k-1-i];
        a[i][k-1-i]=b;
    }
    cout<<"Output Matrix A"<<endl;
    for(i=0;i<k;cout<<endl,i++)
        for(j=0;j<k;j++)
            cout<<a[i][j]<<"\t";
}
```

Блок-схеми алгоритмів деяких інших задач обробки масивів представлені на рис. 9. На рисунку використовуються такі позначення: N – кількість рядків матриці; M – кількість стовпців; I – параметр зовнішнього циклу; J – параметр внутрішнього циклу; A, B, C – матриці; D – одновимірний масив (вектор). Суть алгоритмів сортування масиву представлена у вигляді текстового запису.

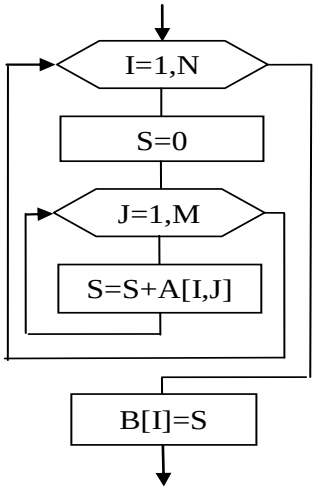
Ще одним із поширених видів алгоритмів обробки масивів є сортування. Розрізняють три основні алгоритми сортування елементів масиву: сортування вибором, вставками, обміном (метод бульбашок).



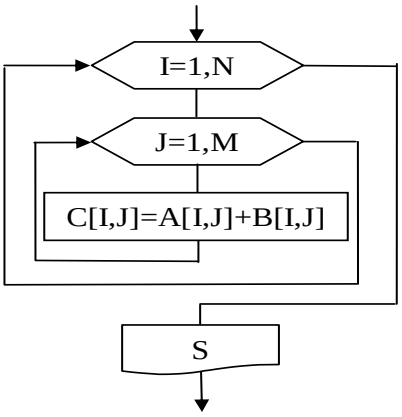
а) Схема алгоритму підсумовування елементів масиву



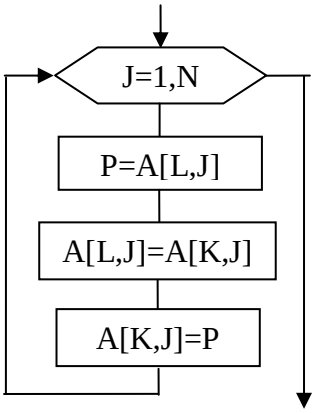
б) Схема алгоритму підсумовування діагональних елементів матриці



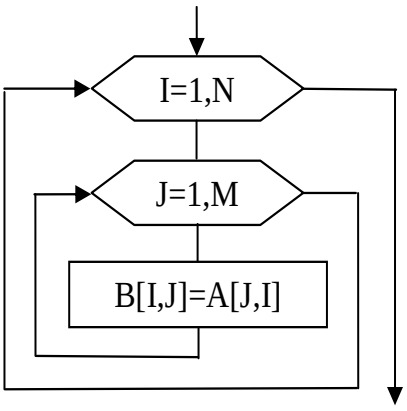
в) Схема алгоритму підсумовування елементів рядків матриці



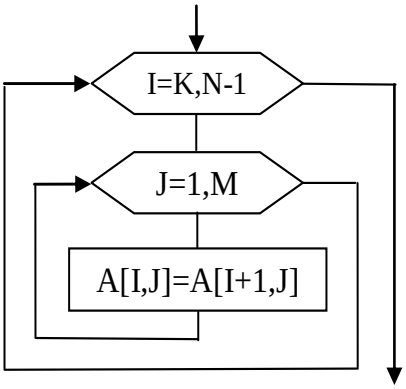
г) Схема алгоритму підсумовування двох матриць



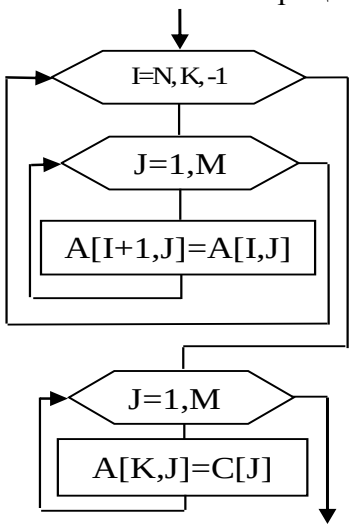
д) Схема алгоритму перестановки L-го та K-го рядків матриці A



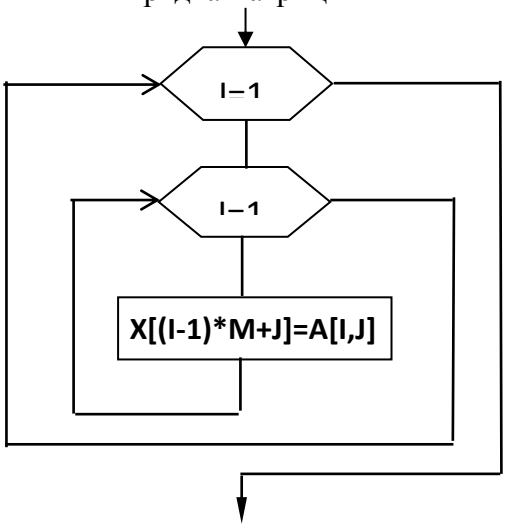
е) Схема алгоритму транспонування матриці A



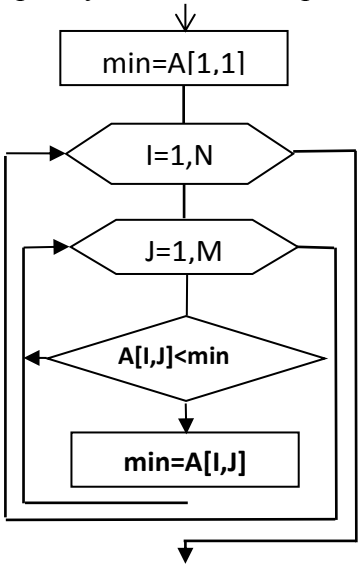
є) Схема алгоритму видалення k-ого рядка матриці A



ж) Схема алгоритму вставки k-ого рядка матриці A



з) Схема алгоритму перетворення матриці A в одновимірний масив X



и) Схема алгоритму пошуку мінімального елементу матриці A

Рисунок 9 – Блок-схеми деяких алгоритмів

ПРИКЛАД 4. Сортування вибором.

Знаходиться максимальний (мінімальний) елемент і переноситься в кінець (початок) масиву; потім ця операція застосовується до всіх елементів, окрім останнього і т.д.

```
const int n=20;
int b[n];
int imin, i, j, a;
/* . */
for (i=0;i<n-1;i++)
{
    imin=i;
    for (j=i+1;j<n;j++)
        if (b[j]<b[imin]) imin=j;
    a=b[i];
    b[i]=b[imin];
    b[imin]=a;
}
```

ПРИКЛАД 5. Сортування вставками.

Якщо перші K елементів масиву вже впорядковані, наприклад, по зростанню, то береться (K+1)-ий елемент і розміщується серед перших K елементів так, щоб впорядкованими виявилися вже K+1 перших елементів; цей алгоритм застосовується при K від 1 до n-1.

```
const int n=20;
int b[n];
int i,j,c;
/* . */
for (i=1;i<n;i++)
{
```

```
    c=a[i];
    for (j=i-1;j>=0 && a[j]>c;j--)
        a[j+1]=a[j];
    a[j+1]=c;
}
```

ПРИКЛАД 6. Сортування обміном.

Послідовно порівнюються пари сусідніх елементів X_k і $X_{к+1}$ (К = 1, 2, 3, .., n-1) і якщо $X_k > X_{к+1}$ (для зростання) або $X_k < X_{к+1}$ (для спадання), то вони переставляються; тим самим найбільший (найменший) елемент виявиться на своєму місці в кінці масиву; потім цей метод застосовується до всіх елементів, окрім останнього, і т.д.

```
for (i=0; i<N-1; i++)
    for (j=i+1; j<N; j++)
        if (a[i]>a[j])
        {
            x=a[j];
            a[j]=a[i];
            a[i]=x;
        }
```

ПРИКЛАД 7. Упорядкувати рядки з непарними індексами по спаданню, з парними – за зростанням, метод впорядкування – сортування «бульбашкою» (рис. 10).

```
int i,j,k,N,M;
double b,a[20][20];
cout<<"M=";cin>>M;
cout<<"N=";cin>>N;
cout<<"Input Matrix A"<<endl;
for(i=0;i<M;i++)
```

```
        for(j=0;j<N;j++)
            cin>>a[i][j];
for(i=0;i<M;i++)/*Перевірка номеру рядка на парність*/
        if(i%2==0)
        {
            //Упорядкування парного рядка за зростанням
            for(k=1;k<N;k++)
                for(j=0;j<N-k;j++)
                    if(a[i][j]>a[i][j+1])
                    {
                        b=a[i][j];
                        a[i][j]=a[i][j+1];
                        a[i][j+1]=b;
                    }
        }
        else
        {
            //Упорядкування непарного рядка за спаданням
            for(k=1;k<N;k++)
                for(j=0;j<N-k;j++)
                    if(a[i][j]<a[i][j+1])
                    {
                        b=a[i][j];
                        a[i][j]=a[i][j+1];
                        a[i][j+1]=b;
                    }
        }
    cout<<"Output Matrix A"<<endl;
```

```
for(i=0;i<M;cout<<endl,i++)
    for(j=0;j<N;j++)
        cout<<a[i][j]<<"\t";
```

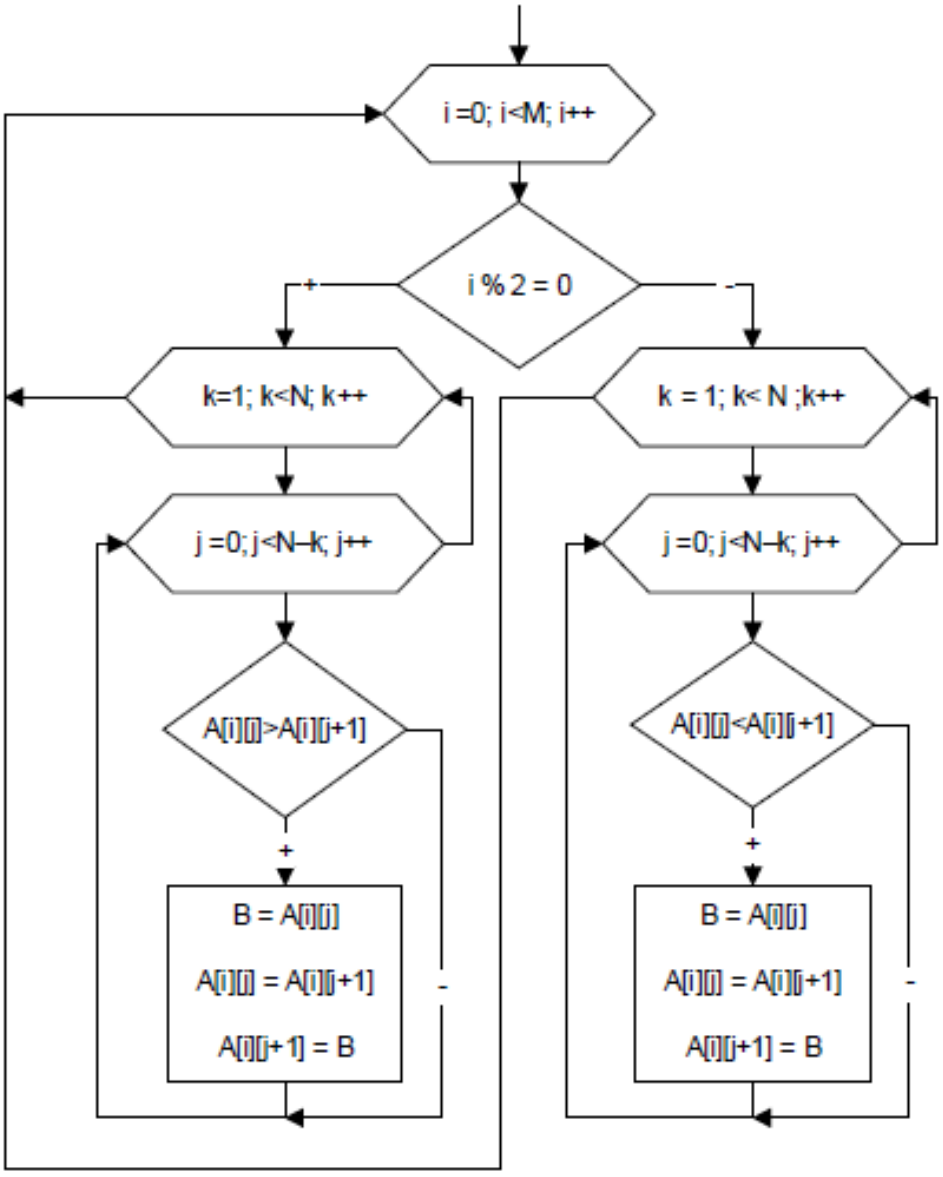


Рисунок 10 – Фрагмент блок-схеми алгоритму упорядкування рядків

Окрім звичайної пам’яті (стека), в якій автоматично розміщуються змінні за їхнього оголошення, існує ще й динамічна пам’ять (heap − купа), в якій змінні можуть розміщуватися динамічно. Це означає, що пам’ять виділяється під час виконання програми, й лише тоді, коли у програмі зустрінеться спеціальна інструкція. Основна потреба у динамічному виділенні пам’яті виникає, коли

розмір або кількість даних заздалегідь є невідомі, а визначаються в процесі виконання програми.

ПРИКЛАД 8. Ввести розміри матриці та виділити для неї місце в пам'яті під час роботи програми.

Проблема: розміри матриці заздалегідь не відомі.

Шляхи вирішення проблеми:

1) виділяти окремий блок пам'яті для кожного рядка;
2) виділити пам'ять одразу на всю матрицю.

Якщо виділяти блок пам'яті на кожний рядок, то (рис. 11):

- матриця = масив рядків;
- адреса матриці = адреса масиву, де зберігаються адреси рядків;
- адреса рядка = вказівник;
- адреса матриці = адреса масиву вказівників.

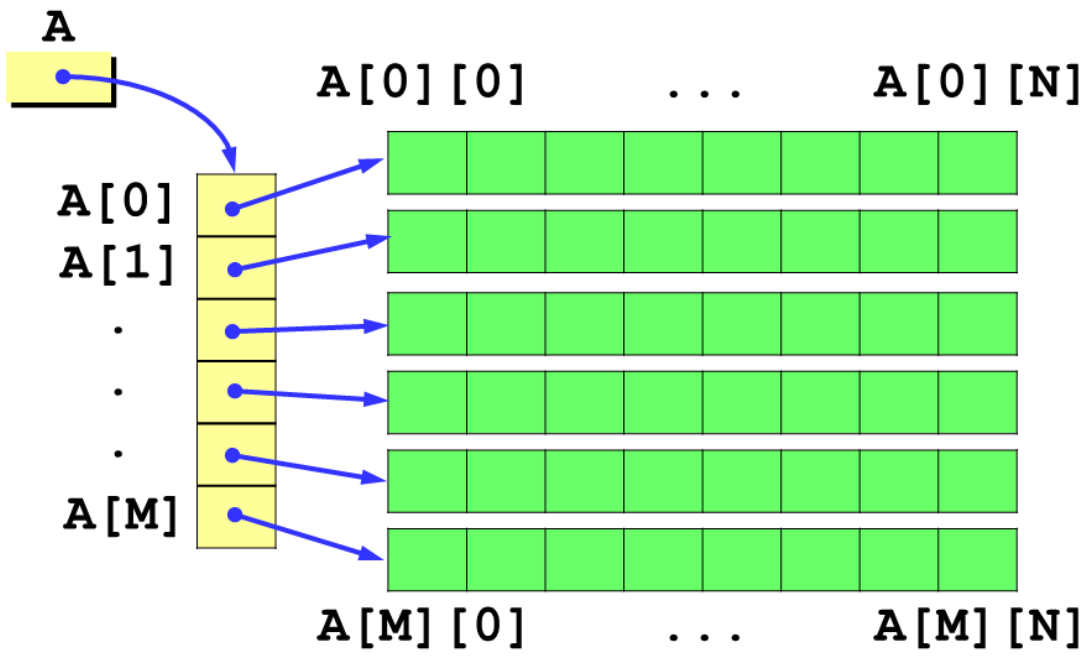


Рисунок 11 – Виділення пам'яті під двовимірний масив

Оголошення динамічної матриці (на C++):

```
int **A;
/* або через оголошення нового типу даних pInt =
```

```
вказівник на int */
    typedef int *pInt;
    pInt *A;
```

Якщо виділяти один блок на матрицю (рис. 12):

```
A = new pInt[M];
A[0] = new int [M*N];
```

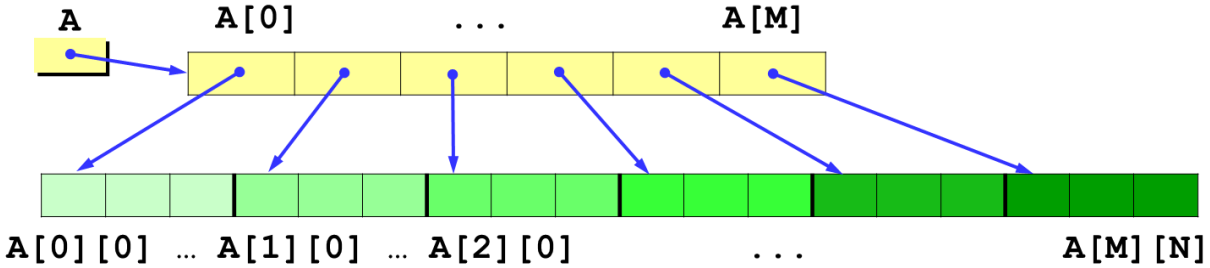


Рисунок 12 – Виділення пам’яті під двовимірний масив одним блоком

Розстановка вказівників:

```
for (i = 1; i < N; i ++ ) A[i] = A[i-1] + N;
```

Вивільнення пам’яті:

```
delete A[0];
delete A;
```

ПРИКЛАД 9. Приклад створення динамічного масиву в мові С.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int M; // Число рядків
    int N; // Число стовпців
    int i,j; // Індекси для циклу for
    int **a; // Двовимірний масив
    printf("\nEnter M: "); scanf("%d",&M);
    printf("\nEnter N: "); scanf("%d",&N);
```

```
    a=(int**) malloc(sizeof(int*)*M);
    for (i=0; i<M; i++)
        *(a+i)=(int*)malloc(sizeof(int)*N);
    for (i=0; i<M; i++)
        for (j=0; j<N; j++) a[i][j]=i*j;
    for (i=0; i<M; i++)
    {
        for (j=0; j<N; j++) printf("%6d",a[i][j]);
        printf("\n");
    }
}
```

Результат виконання цієї програми представлений на рис. 13.

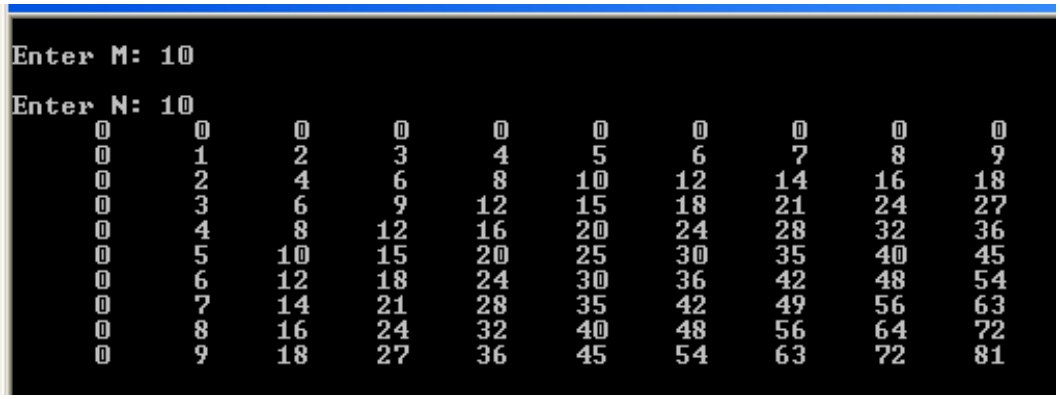


Рисунок 13 – Результат виконання програми

Нехай *n*-вимірний масив *A*, у якого кількість елементів по *i*-му виміру дорівнює m_i, а індексація по всім вимірам починається з нуля, представлений в динамічній пам'яті вектором B. Тоді елементу A $[i_1, i_2, \ldots, i_n]$ вихідного масиву буде відповідати елемент вектору $B[k]$ з індексом, що дорівнює:

$$k = ((\ldots(i_1*m_2+i_2)*m_3+i_3)*m_4 + \ldots + i_{n-1})*m_n + i_n$$

де $A\ [i_1, i_2, \ldots, i_n]$ – *n*-вимірний масив, *k* – індекс елемента вектору $B[k]$.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися з найпростішими алгоритмами обробки масивів, представленими в теоретичних відомостях.

2. Скласти блок-схему алгоритму і програми відповідно до індивідуального завдання.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

При виконані завдань використовувати динамічне виділення пам’яті та передбачити як псевдовипадкову генерацію елементів масивів, так й їх ручне введення.

1. Заданий масив A(N,M) і вектор B(M). Елементи першого стовпця масиву A впорядковані по спаданню. Включити масив B як новий рядок в масив A із збереженням впорядкованості за елементами першого стовпця.

2. Матриця розміщена в одновимірному масиві по рядках. Видалити K-ий рядок матриці з одновимірного масиву. Результат представити у вигляді матриці, роздрукованої по рядках.

3. Матриця розміщена в одновимірному масиві по рядках. Видалити K-ий стовпець матриці з одновимірного масиву. Результат представити у вигляді матриці, роздрукованої по рядках.

4. Матриця розміщена в одновимірному масиві по рядках. Поміняти місцями K-ий і L-ий рядки. Результат представити у вигляді матриці.

5. Матриця розміщена в одновимірному масиві по рядках. Поміняти місцями K-ий і L-ий стовпці. Результат представити у вигляді матриці.

6. Задана матриця A(N,N). Сформувати два одновимірні масиви. В один переслати по рядках верхній трикутник матриці, включаючи елементи головної

діагоналі, в інший – нижній трикутник. Роздрукувати верхній і нижній трикутники по рядках як матриці.

7. Квадратна матриця задана у вигляді одновимірного масиву по рядках. Надрукувати верхній трикутник матриці (включаючи елементи головної діагоналі) по рядках як матрицю.

8. Матриця, симетрична відносно головної діагоналі, задана верхнім трикутником у вигляді одновимірного масиву по рядках. Відновити початкову квадратну матрицю і надрукувати по рядках.

9. Задана квадратна матриця. Переставити рядок з максимальним елементом на головній діагоналі з рядком із заданим номером.

10. Задана квадратна матриця. Виключити з неї рядок і стовпець, на перетині яких розташований максимальний елемент головної діагоналі.

11. Задані матриця A(N,N) і число K (1<=K<=N). Рядок з максимальним по модулю елементом в K-ому стовпці переставити з K-й рядком.

12. Задані матриця A(N,N) і число K (1<=K<=M). Стовпець з максимальним (по модулю) елементом в K-ому рядку переставити з K-м стовбцем.

13. Задана матриця A(N,N). Знайти максимальний по модулю елемент матриці. Переставити рядки і стовпці матриці так, щоб максимальний по модулю елемент був розташований на перетині K-го рядка і K-го стовпця.

14. Елемент матриці називається локальним мінімумом, якщо він строго менше всіх своїх сусідів. Підрахувати кількість локальних мінімумів заданої матриці A(5,5). Роздрукувати їхні значення і положення.

15. Знайти максимальний елемент серед всіх елементів тих рядків заданої матриці, які впорядковані за збільшенням.

16. Елемент матриці називається локальним максимум, якщо він строго більше всіх своїх сусідів. Підрахувати кількість локальних максимумів заданої матриці A(5,5). Роздрукувати їхні значення і положення.

17. Знайти мінімальний елемент серед всіх елементів тих рядків заданої

матриці, які впорядковані за зменшенням.

18. Дана дійсна матриця A (N,M). Знайти середнє арифметичне кожного із стовпців, які мають парні номери.

19. Дана дійсна матриця A (N,M). Визначити числа $b_1,...,b_n$, які дорівнюють різницям найбільших і найменших значень елементів рядків відповідно.

20. Дана дійсна матриця A (N,M). Знайти суму найбільших значень елементів її рядків.

21. В даній дійсній матриці A (N,M) поміняти місцями рядок, що містить елемент з найбільшим значенням, з рядком, що містить елемент з найменшим значенням. Передбачається, що ці елементи єдині.

22. Дана дійсна матриця A (N,M), всі елементи якої різні. В кожному рядку вибирається елемент з найменшим значенням, потім серед цих чисел вибирається найбільше. Указати індекси елемента із знайденим значенням.

23. Дана квадратна матриця цілих A (N,N). Знайти найменше із значень елементів стовпця, сума модулів елементів якого є найбільшою. Якщо таких стовпців декілька, то взяти перший з них.

24. Дана дійсна квадратна матриця A (N,N). В рядках з негативним елементом на головній діагоналі знайти найбільший зі всіх елементів.

25. Дана дійсна квадратна матриця A (N,N). Розглянути ті елементи, які розташовані в рядках, що починаються з від’ємного елемента. Знайти суми тих з них, які розташовані відповідно нижче та вище головної діагоналі.

26. Дана дійсна квадратна матриця A (N,N). Обчислити суму тих з її елементів, розташованих на головній діагоналі і вище, які більше всіх елементів, які розташовані нижче головної діагоналі. Якщо на головній діагоналі і вище неї немає елементів з вказаною властивістю, то відповіддю повинне слугувати повідомлення про це.

27. Дана цілочисельна матриця A(N,M). Визначити кількість "особливих" елементів масиву A, вважаючи елемент "особливим", якщо він більше суми

інших елементів свого стовпця.

28. Дана цілочисельна матриця A(N,M). Визначити кількість "особливих" елементів масиву A, вважаючи елемент "особливим", якщо в його рядку зліва від нього знаходяться елементи, менші за нього , а справа – більші.

29. Матриця має сідлову точку A_{ij}; якщо елемент A_{ij} є мінімальним в i-ому рядку і максимальним в j-ому стовпці. Знайти індекси сідлових точок заданої матриці.

30. По заданій квадратній матриці A(N,N) побудувати вектор завдовжки 2*N-1, елементи якого – максимуми елементів діагоналей, паралельних головній діагоналі.

КОНТРОЛЬНІ ПИТАННЯ

1. Складіть фрагмент програми видалення К-го елемента з масиву A(10).
2. Складіть фрагмент програми, який включає числовий елемент в К-ту позицію масиву X(10).
3. Які правила множення матриці на вектор?
4. Складіть фрагмент програми включення числового елемента в масив A(10), впорядкований за збільшенням із збереженням впорядкованості.
5. Які умови виконання операції множення матриць?
6. Складіть фрагмент програми видалення К-го стовпця матриці X(4,5).
7. Пояснити суть алгоритмів сортування масивів.
8. Складіть фрагмент програми включення в К-у позицію стовпців матриці A(4,4) вектору B(4).

ЛАБОРАТОРНА РОБОТА №6

ФУНКЦІЇ.

Мета роботи: вивчити способи оголошення та використання функцій під час створення програм, область дії змінних, способи передачі параметрів у функції.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Програма, написана мовою C/C++, являє собою набір функцій, серед яких є користувацькі функції, що реалізують окремі підзадачі, та головна функція програми. Використання функцій значно покращує код, оскільки забезпечує його структурованість, дає можливість повторного використання коду тощо.

Функція – це автономна частина програми з ім'ям. Її оголошення має вигляд:

```
<тип> <ім'я> (<список параметрів>) { тіло функції }
```

<тип> визначає тип значення, яке функція повертає. Функція, яка не повертає значення, має тип **void.**

Значення, яке повертається функцією, є результатом роботи функції. Функція може повертати лише одне значення, але будь-якого типу даних. Якщо функція повертає значення, то в тілі функції обов'язково присутня директива **return <вираз>**. В якості виразу може бути задане ім'я однієї змінної, тип якої відповідає типу значення, що повертається;

<ім'я> – ідентифікатор, який визначає користувач;

<список параметрів> – це список елементів, що відокремлюються один від одного комою, кожний елемент списку складається з імені змінної та її типу. У загальному випадку список параметрів має вигляд: (<тип1> <ім'я1>, <тип2> <ім'я2>, …, <типk> <ім'яk>).

До списку параметрів включають змінні, які функція використовує у своїй

роботі. Ці параметри називаються формальними параметрами функції. Але функція може бути й без параметрів.

Тіло функції за своєю структурою повністю відповідає структурі головної програми main().

ПРИКЛАД 1. Передача параметрів у функцію.

```
float ymn (float x, float y)
{
    float z;
    z = y*x;
    return z;
}
```

або

```
float ymn (float x, float y)
{
    return x*y;
}
```

Якщо код функції розташований нижче коду основної функції (main()) програми, то функція має бути оголошена перед нею. Для цього заголовок функції має передувати заголовку функції main() програми. Код заголовка функції, який записується перед основною програмою, називається **прототипом функції**. Код прототипу функції закінчується символом «;».

```
float ymn (float a, float b); // Прототип функції
void main()// Головна функція програми
{
    int x=2, y=5;
    cout << x << "*" << y << "= " << ymn(x,y) << "\n";
    // виклик функції ymn
}
```

```
float ymn (float z, float w)
{
    return z*w; // Функція обчислює добуток двох чисел
}
```

Імена параметрів у прототипі можуть не співпадати з іменами, які використовуються у коді функції.

Усі змінні, що використовуються у програмі, можна розділити на локальні та глобальні.

Локальні змінні – це змінні, які визначаються всередині функції або блоку програми, використовуються лише всередині функції.

Блок програми – це набір директив (у тому числі опис змінних), які укладені у фігурні дужки. Локальні змінні створюються при вході в блок та ліквідуються при виході. Змінні, які оголошені в одному блоці, не мають жодного відношення до змінних іншого блоку навіть у разі збігу імен.

ПРИКЛАД 2. Використання глобальних та локальних змінних.

```
void main()
{
    // 1-й блок
    {
        int x, y, z;
        ...
    }
    // 2-й блок
    {
        int x, y, z;
        ...
    }
}
```

Змінні x, y, z першого блоку не пов'язані зі змінними x, y, z другого блоку. Це, в принципі, різні області пам'яті. Параметри, які перераховуються в заголовку функції, є локальними змінними.

Глобальні змінні – це ті змінні, які оголошуються в головній функції програми (перед заголовком функції або блоку програми). Глобальні змінні можна використовувати в будь-якій директиві, незалежно від того, у якій функції чи якому блоці ця директива використовується.

Локальні змінні мають пріоритет перед глобальними змінними. Це означає, що якщо імена глобальної та локальної змінних збіглися, то ці змінні між собою не пов'язані і визначають різні області пам'яті.

Для виклику функції (як уже зазначалося вище) достатньо там, де це необхідно, записати директиву, до якої входить ім'я функції зі списком параметрів. Наприклад, директива

```
r = 7 * ymn (3.5,2);
```

у змінну **r** посилає значення множення числа **7** на результат, який обчислюється функцією **ymn()** для чисел **3.5** та **2.**

Кожна функція, зокрема і головна функція main(), є блок, куди входять код програми та дані. Взагалі, цей блок закритий та недоступний для інших функцій, якщо лише у самому блоці немає викликів цих функцій. Іншими словами, на дані, визначені в одній функції, не можуть впливати інші функції, оскільки в різних функціях різні сфери дії. Але в будь-якій мові є інструменти (правила, механізми), які дозволяють, за необхідності, різним функціям взаємно впливати на свої області дій.

Локальні змінні створюються при зверненні до функції, і при виході з цієї функції вони знищуються, тобто інформація локальних змінних не зберігається між викликами функції. Винятки становлять глобальні змінні та змінні, які оголошуються як **static**. При використанні глобальних параметрів треба мати на увазі наступне. У разі збігу імен глобальних та локальних змінних пріоритет

мають локальні. Область дії функції у цьому разі не поширюється на ті глобальні змінні, імена яких збігаються з іменами локальних змінних.

У мові C/C++ є три способи передачі параметрів у функцію.

Перший спосіб – це **передача параметра за значенням**. Якщо у деяку функцію треба передавати параметр через значення, то формат заголовка функції має бути наступним:

```
<тип> <ім'я> (<тип> <ім'я>)
```

Наприклад, нехай функція має назву **void f1(int x)**. Передача параметра x в тіло цієї функції відбуватиметься за значенням, а саме: інформація з пам'яті програми, що викликає, пересилається (переписується, копіюється) в пам'ять функції, що викликається. Нехай A – адреса області пам'яті, яка під контролем програми, з якої функція викликається; B – адреса області пам'яті під контролем функції, що викликається. Число x з A пересилається в B. При такому способі передачі треба мати на увазі, що функція, що викликається, не може змінити інформацію в області пам'яті A.

Другий спосіб передачі параметрів полягає в тому, що у функцію передається не аргумент, а значення адреси аргументу. Загальний формат заголовку функції в цьому випадку:

```
<тип> <ім'я> (<тип> *<ім'я>)
```

Наприклад, нехай функція має заголовок **void f1(int *x)**. Локальним параметром функції у цьому випадку є вказівник, в який із програми, яка викликає, пересилається адреса змінної. Цей спосіб передачі параметра в тіло функції називається **передачею параметра за вказівником**. Він дозволяє з функції змінювати інформацію в пам'яті програми, що викликає. Для звернення всередині функції до відповідної ділянки пам’яті треба виконати операцію розіменування вказівника.

Третій спосіб передачі параметрів – передача параметра за посиланням. Загальний формат заголовка функції в цьому випадку має бути таким:

```
<тип> <ім'я> (<тип> &<ім'я>)
```

Наприклад, якщо функція має заголовок **void f1(int &x)**, то в цьому випадку функція працює безпосередньо зі змінними програми, яка викликає, і в цьому випадку застосовувати операцію розіменування не потрібно.

ПРИКЛАД 3. Передача параметрів у функцію різними способами.

```
#include <iostream>
void f1 (int a, int *b, int &c);
main ()
{
   int a = 1, b = 2, c = 3;
   cout <<a << ' ' << b << ' ' << c << '\n';
   f1 ( a, &b, c);
   cout << a << ' ' <<b << ' ' << c << '\n';
   return 0;
}
void f1 (int a, int *b, int &c)
{
   a++; (*b)++; c++;
}
```

Результат:
1 2 3
1 3 4

При передачі масивів, у функцію передається вказівник на масив, тобто адреса першого елемента масиву. Функція працює з масивом у програмі, що його викликає, і тому може змінювати або перетворювати інформацію, яка безпосередньо міститься у даному масиві. Формат передачі адреси масиву на функцію має різний синтаксис.

Наприклад, коди

```
void f_1(int a[]);
```

```
void f_1(int *a);
void f_1(int a[10]);
```

декларують передачу масиву **a** у функцію f_1. У всіх трьох випадках код виклику функції однаковий і має наступний синтаксис: **f_1(a)**, де **a** – масив, визначений у програмі, яка викликає.

При передачі двовимірного масиву у функцію треба переслати до неї адресу першого елемента масиву. Заголовок функції, до якої передається двовимірний масив, може мати наступний синтаксис:

```
void f_1(int x[][m]);
```

де **m** – раніше оголошена константа. За такої форми передачі завдання розміру **m** (числа елементів у рядку) є обов'язковим.

Якщо функція використовується для кількох двовимірних масивів різної розмірності, то двомірний масив можна інтерпретувати як одновимірний. Синтаксис заголовка функції буде наступним:

```
void f_1 (int *x, int n, int m)
```

де **x** – вказівник на перший елемент матриці; **m, n** – розміри матриці. У тілі функції доступ до елементів матриці за такого звернення можна здійснити лише через адреси елементів.

ПРИКЛАД 4. Функція введення і виведення матриць різної розмірності.

```
const int n1 = 3, m1 = 2;
const int n2 = 3, m2 = 3;
void vvod(int *x, int n, int m);
void vivod(int *x, int n, int m);
void main()
{
   int a[n1][m1], b[n2][m2];
   vvod(a[0], n1, m1);
   // Виклик функції введення елементів матриці
```

```
    vivod(a[0], n1, m1);
    // Виклик функції виведення елементів матриці
    vvod(b[0],n2,m2);
    vivod(b[0],n2,m2);
}
// Функція введення елементів матриці
void vvod(int *c, int n, int m)
{
    int i, j;
    printf("Ввести елементи матриці: \n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
        {
            printf("x(%d,%d)=", i, j);
            scanf("%d", c + i * m + j);
            // i * m + j – порядковий номер елемента
            // матриці з індексом i, j
            // c + i * m + j – адреса цього елемента
        }
}
// Функція виведення елементів матриці на екран
void vivod(int *c, int n, int m)
{
    int i, j;
    printf("\nВведена матриця:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
```

```
            printf("%d ", *(c + i * m + j));
        printf ("\n");
    }
}
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичними відомостями, звернувши особливу увагу на способи передачі параметрів у функцію.

2. Скласти блок-схему алгоритму і програми відповідно до індивідуального завдання.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

При виконані завдань реалізовувати окремий функціонал в різних функціях. Наприклад, якщо мова йде про роботу з масивом, то створення, заповнення, виведення на екран та сама обробка масиву відповідно до завдання мають бути реалізовані як окремі функції. Крім того, для повернення результатів роботи функції використовувати механізм вказівників, а не оператор **return**. Скажімо, при реалізації функції, що розв’язує квадратне рівняння, доцільним є повертати знайдені корені рівняння через механізм вказівників, а оператор **return** використати для повернення статусу завершення функції, який залежить від кількості знайдених коренів, для подальшої обробки цього статусу у функції, що викликала.

1. Дано дійсні числа a_0, ..., a_6. Отримати для x = 1, 2, 3, 4 значення p(x + 1) - p(x), де $p(y) = a_6y^6 + a_5y^5 + ... + a_0$.

2. Дано дійсні числа s, t, a_0, ... a_{12}. Отримати p(1) - p(t) + p^2(s-t) - p^3(t), де p (x) = $a_{12}x^{12} + a_{11}x^{11}$ + ... + a_0.

3. Дано дійсні числа a_1, ..., a_n, b_1, ..., b_m. У послідовності a_1, ..., a_n і послідовності b_1, ..., b_m всі члени, які йдуть за членом з найбільшим значенням (за першим по порядку, якщо їх декілька), замінити на 0.5.

4. Дано цілі числа a_1, ..., a_n, b_1, ..., b_m, K, M. Якщо в послідовності a_1, ..., a_n немає жодного члена зі значенням K, то перший по порядку член цієї послідовності, не менший всіх інших членів, замінити на значення K. За таким же правилом перетворити b_1, ..., b_m, замінивши значення на число M.

5. Обчислити z = (S_1 + S_2)/(K_1 + K_2), де S_1 і K_1 – сума і кількість позитивних елементів масиву X (N); S_2 і K_2 – сума і кількість позитивних елементів масиву Y(M).

6. Обчислити суми додатних елементів кожного рядка матриці A (N,M), B (K,L).

7. Обчислити Z = (X_1 + Y_1)/(X_2 - Y_2), де X_1 і X_2 – корені рівняння ax^2 + bx - c = 0; Y_1 і Y_2 – корені рівняння ky^2 + my - n = 0.

8. Дано масиви X(N) і Y(M).З найти найбільші елементи та їхні порядкові номери.

9. Переписати додатні елементи масивів X (N) і Y (M) в масив Z. Спочатку додатні елементи масиву X, потім масиву Y. Для запису елементів в масив Z використовувати функцію.

10. Дано матриці A (N,M) і B (K,L). Знайти найменші елементи і визначити номера рядків і стовпців, в яких вони розташовані.

11. Вивести на екран елементи цілочисельних матриць A(N,M) і M(K,L), кратні трьом.

12. Обчислити кількість від’ємних елементів кожного стовпця для матриць A (N,M), B (K,L).

13. Обчислити суми елементів верхньої трикутної матриці для матриць

A (N,N), B (M,M).

14. Знайти середні значення і стандартні відхилення для елементів масивів X (N), Y (M).

15. Обчислити суми і кількості елементів, що знаходяться в інтервалі від a до b для матриць X(N,M) і Y(K,L).

16. Перетворити масиви X(N) і Y(M), розташувавши в них поспіль тільки додатні елементи. Замість інших елементів записати нулі.

17. Обчислити $Z = (e^{S1} + e^{S2}) / (K_1 * K_2)$, де S_1 і K_1 – сума і кількість додатних елементів масиву X(N); S2 і K2 – сума і кількість від'ємних елементів масиву Y(M).

18. Обчислити корені квадратних рівнянь $ax^2 + bx + c = 0$; $ky^2 - my - n = 0$, використовуючи функції.

19. Обчислити $Z = (X_{max} - Y_{min}) / 2$, де X_{max} – максимальний елемент масиву X(N); Y_{min} – мінімальний елемент масиву Y(M). X_{max} і Y_{min} обчислювати в одній функції.

20. Обчислити суми додатних елементів масивів X(N), Y(M), Z(K), використовуючи функцію.

21. Обчислити, використовуючи функцію, $C = (A_{max} + B_{min}) / 3$, де A_{max} і B_{min} – максимальний і мінімальний елементи відповідно масивів A(N) і B(M).

22. Обчислити суми додатних елементів кожного рядка для матриць A(N,M) і B(K,L).

23. Обчислити суми елементів головних діагоналей матриць A(N,N), B(M,M), використовуючи функцію.

24. Обчислити суми елементів нижніх трикутників для матриць A(N,M) та B(K,L), використовуючи функцію.

25. Використовуючи функцію, визначити позицію першого входження заданого символу у вихідний рядок. Якщо рядок не містить шуканого символу, то результатом виконання функції повинно бути -1.

26. Дано відрізки a, b, c, d. Для кожної трійки відрізків, із яких можна побудувати трикутник, знайти площу даного трикутника. Для визначення можливості побудови трикутника та обчислення площі розробити функції.

27. Знайти мінімальний елемент серед максимальних елементів в рядках двовимірного масиву.

28. Знайти максимальний елемент серед мінімальних елементів в рядках двовимірного масиву.

29. Знайти мінімальний елемент серед максимальних елементів в стовпцях двовимірного масиву.

30. Знайти максимальний елемент серед мінімальних елементів в стовпцях двовимірного масиву.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке функція? Як вона оголошується?
2. Який результат може повертати функція?
3. Поясніть значення прототипу функції.
4. Яким чином можна забезпечити отримання за допомогою функції декількох результатів?
5. Які змінні називаються глобальними?
6. У чому відмінність глобальних та локальних змінних?
7. Яким чином функції передаються аргументи?
8. Що таке передача аргументу за значенням та за посиланням? Чим ці способи відрізняються і як реалізуються?
9. Яким чином у функцію передаються вказівники?
10. Яким чином у функцію передаються масиви?
11. Які аргументи можуть бути у головного методу програми – функції main()?
12. Що таке аргументи за замовчуванням і як вони визначаються?

13. Що таке рекурсія і в яких випадках вона використовується?

14. Що таке перевантаження функції? У яких випадках застосовується перевантаження функцій?

15. У чому переваги і недоліки використання вбудованих функцій?

ЛАБОРАТОРНА РОБОТА № 7

РЕКУРСІЯ. РЕКУРСИВНІ ФУНКЦІЇ

Мета роботи: вивчити рекурсивні алгоритми і основні схеми рішення задач рекурсивними способами, навчитися розробляти рекурсивні тріади і використовувати рекурсивні алгоритми при вирішенні задач на мові С/С++.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Рекурсія — це спосіб організації допоміжного алгоритму (функції), у якому ця функція під час виконання її операторів звертається сама до себе. Загалом, рекурсивним називається будь-який об'єкт, який частково визначається через себе.

У рекурсивному визначенні має бути обмеження, гранична умова, при виході на яку подальша ініціація рекурсивних звернень припиняється.

Звернення до рекурсивної функції не відрізняється від виклику будь-якої іншої. При цьому, за кожного нового рекурсивного звернення, в пам'яті створюється нова копія функції з усіма локальними змінними. Такі копії породжуватимуться до виходу на граничну умову. Очевидно, у разі відсутності граничної умови, необмежене зростання кількості таких копій призведе до аварійного завершення програми внаслідок переповнення стека.

Породження нових копій рекурсивної функції до виходу на граничну умову називається рекурсивним спуском. Максимальна кількість копій рекурсивної підпрограми, яка одночасно може бути завантажена в пам'яті комп'ютера, називається глибиною рекурсії. Завершення роботи рекурсивних функцій, аж до першої, що ініціювала рекурсивні виклики, називається рекурсивним підйомом.

Виконання дій у рекурсивній функції може бути організовано одним із варіантів (табл. 1, F() – рекурсивна функція).

Як очевидно з табл. 1, дії можуть виконуватися або на одному з етапів

рекурсивного звернення, або на обох одночасно. Спосіб організації дій визначається логікою алгоритму, що розробляється.

Таблиця 1 – Варіанти організації дій у рекурсивній функції

рекурсивний підйом (рекурсивне повернення)	рекурсивний спуск	рекурсивний спуск і рекурсивний підйом

{
F();
операторі;
}
	{
	операторі;
	F();
	}
		{
		операторі;
		F();
		операторі;
		}

ПРИКЛАД 1. Обчислення факторіалу числа: $n!$ = 1 * 2 * 3 * ... * n або (рекурентна формула): $n!=\begin{cases}1, & \text{якщо } n \le 1\\ (n-1)*n, & \text{якщо } n > 1\end{cases}$.

Рекурсивний варіант (граничною умовою є n<=1):

```
double Factorial(int n)
{
    double F;
    if (n<=1) F = 1;
    else F = Factorial(n-1)*n;
    return F;
}
```

ПРИКЛАД 2. Визначимо функцію K(n), яка повертає кількість цифр у натуральному числі n: $K(n)=\begin{cases}1, & \text{якщо } n < 10\\ (K(n/10)+1, & \text{якщо } n \ge 10\end{cases}$.

```
int K(int n)
{
    int Kol;
    if (n<10)Kol = 1;
    else Kol = K(n/10)+1;
    return Kol;
```

```
}
```

ПРИКЛАД 3. Функція C(*m, n*), де 0 <= *m* <= *n*, обчислення біномного коефіцієнта C_n^m за формулою $C_n^0 = C_n^n = 1;\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ при $0 < m < n$, рекурсивний варіант:

```
int C(int m, int n)
{
   int f;
   if (m == 0 || m == n) f = 1;
   else f = C(m, n-1) + C(m-1, n-1);
   return f;
}
```

ПРИКЛАД 4. Обчислити суму елементів одновимірного масиву. При розв'язанні задачі використовуємо таке міркування: сума дорівнює нулю, якщо кількість елементів дорівнює нулю, і сумі всіх попередніх елементів плюс останній, якщо кількість елементів не дорівнює нулю.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int summa(int N, int a[100]);
int i, n, a[100];
void main()
{
   printf("\nКількість елементів масиву?");
   scanf("%d", &n);
   srand(time(NULL));
   for (i=0; i<n; i++)
   {
      a[i]= -10+rand(21);
```

```
        printf("%d ", a[i]);
    }
    printf("Сума: %d", summa(n-1, a));
}
int summa(int N, int a[100])
{
    if (N == 0) return a[0];
    else return a[N] + summa(N-1, a);
}
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичними відомостями.

2. Скласти блок-схему алгоритму і програми відповідно до індивідуального завдання.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДІДУАЛЬНІ ЗАВДАННЯ

1. Написати програму розрахунку функції Акермана для усіх невід’ємних цілих аргументів m і n

$$A(m,n) = \begin{cases} n+1, \text{ при } m = 0 \\ A(m-1,1), \text{ при } m > 0, n = 0; \\ A(m-1, A(m,n-1)), \text{ при } m > 0, n > 0. \end{cases}$$

2. Маємо рядок тексту. Надрукувати цей текст в зворотному напрямку, використовуючи рекурсію.

3. Реалізувати рекурсивну функцію, яка перевіряє, чи є рядок симетричним.

4. Числа Фібоначчі f_n обчислюються за формулами $f_0 = f_1 = 1$; $f_n = f_{n-1} + f_{n-2}$ при $n = 2,3,...$ Реалізувати рекурсивну функцію, яка за заданим номером n обчислюватиме значення f_n.

5. Реалізувати рекурсивну функцію для обчислення найбільшого спільного дільника двох натуральних чисел A та B, використовуючи алгоритм Евкліда і співвідношення: НСД(a,0) = НСД(0,*a*) = *a*; НСД(*a,b*) = НСД(*a*-*b*,*b*), якщо a>=b; НСД(*a,b*) = НСД(*a,b*-*a*), якщо *b*>*a*.

6. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу визначити, чи є задане натуральне число паліндромом, тобто читається однаково зліва направо і справа наліво.

7. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу знайти суму елементів масиву A.

8. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу знайти добуток елементів масиву A.

9. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу знайти максимальний елемент масиву A.

10. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу знайти мінімальний елемент масиву A.

11. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З допомогою методу знайти кількість додатних елементів масиву A.

12. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. З

допомогою методу знайти кількість від’ємних елементів масиву A.

13. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. Перевірити за допомогою методу, чи всі елементи масиву A є додатними.

14. Метод «розділяй і володарюй» використовує два рекурсивні виклики, кожен з яких працює приблизно з половиною вхідних даних. Перевірити за допомогою методу, чи всі елементи масиву A є від’ємними.

15. Розробити рекурсивну функцію, яка обчислює суму елементів масиву, що стоять на непарних позиціях.

16. Розробити рекурсивну функцію, яка обчислює добуток елементів масиву, що стоять на парних позиціях.

17. Розробити рекурсивну функцію, яка обчислює суму елементів масиву, що стоять на парних позиціях.

18. Розробити рекурсивну функцію, яка обчислює добуток елементів масиву, що стоять на непарних позиціях.

19. Розробити рекурсивну функцію, яка визначає, чи всі елементи масиву A є різними.

20. Розробити рекурсивну функцію, яка визначає, чи є масив впорядкованим за зростанням.

21. Розробити рекурсивну функцію, яка визначає, чи є масив впорядкованим за спаданням.

22. Розробити рекурсивний варіант алгоритму сортування «бульбашкою».

23. Розробити рекурсивну функцію, яка обчислює кількість входжень елемента X у масив A цілих чисел.

24. Розробити рекурсивний варіант алгоритму сортування вибором.

25. Реалізувати рекурсивну функцію для обчислення найменшого спільного кратного двох натуральних чисел A та B, використовуючи

алгоритм Евкліда для обчислення найбільшого спільного дільника.

26. Скласти рекурсивну функцію підрахунку кількості цифр в цілому числі N.

27. Обчислити суму цифр цілого числа N, використовуючи рекурсію.

28. Підрахувати кількість символів у рядку S зі значенням C, використовуючи рекурсію.

29. Скласти рекурсивну функцію, яка обчислює кількість великих латинських літер у рядку S.

30. Скласти рекурсивну функцію, яка обчислює кількість маленьких латинських літер у рядку S.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яке визначення називається рекурсивним? Наведіть приклади рекурсивних визначень.
2. Що таке гранична умова і яке його призначення в рекурсивній підпрограмі?
3. Що таке рекурсивний спуск та рекурсивний підйом в програмі з рекурсією?
4. Описати схему виконання дій як на рекурсивному спуску, так і на рекурсивному поверненні.
5. Що таке глибина рекурсії? Чому дорівнює глибина рекурсії у наведених вище прикладах?
6. На якому етапі виконання рекурсивної підпрограми можуть виконуватись її оператори?
7. Що таке рекурсивна тріада при розробці рекурсивного алгоритму?
8. К яким алгоритмам за витратами ресурсів відносяться рекурсивні алгоритми?
9. Переваги і недоліки рекурсивних алгоритмів?

ЛАБОРАТОРНА РОБОТА № 8

СТРУКТУРИ

Мета роботи: вивчити поняття, оформлення та визначення структур, доступ до елементів структур, навчитися вирішувати задачі з використанням структур мовою C/C++.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Оголошення структури

Структура – це послідовність елементів, які в загальному випадку можуть бути різного або одного типу. На логічному рівні відповідну структуру даних можна записати в такий спосіб:

```
struct [тег структури] { T1 S1; T2 S2; ......; Tn Sn; }
[список змінних структури]
```

Квадратні дужки відображають необов'язковість присутності зазначених полів; тег структури не є ім'ям змінної – це всього лише мітка для її формату; S1, ..., Sn – ідентифікатори полів; T1, ..., Tn – типи даних. Якщо Ti в свою чергу також є структурою, то отримуємо ієрархічну структуру.

Масив дозволяє працювати з наборами даних, всі елементи яких мають один і той же тип. Однак у багатьох задачах доводиться мати справу з наборами даних, що складаються з елементів різного типу. Прикладом такого набору даних є дата, яка містить номер дня у місяці (ціле число), найменування місяця (рядок), рік (ціле число) і день тижня (рядок). Іншим прикладом є опис книг, в якій вказується найменування книги (рядок), автор (рядок) і ціна (число з плаваючою точкою). Для роботи з такими наборами даних в C введені структури.

Структура – це одна або кілька змінних (можливо різних типів), які для зручності роботи з ними згруповані під одним ім'ям. Структури допомагають в організації складних даних, особливо в великих програмах, оскільки дозволяють групу пов'язаних між собою змінних розглядати не як безліч окремих елементів, а як єдине ціле.

Оголошення структури в С задається в одному з наступних двох форматів:

```
struct ім'я-типу
{
    список-оголошень-елементів-структури;
}
список-імен-змінних-структури;
```

або

```
struct ім'я-типу список-імен-змінних-структури;
```

де **ім'я-типу** в першому форматі – необов'язковий ідентифікатор (тег структури), який іменує структурний тип, що задається списком перерахування, зазначеному в операторі, а в другому форматі – посилання на структуру з відповідним **ім'ям-типу**, оголошену в іншому місці програми; **список-оголошень-елементів-структури** – послідовність з одного і більше оголошень елементів структури або бітових полів. Елементи структури можуть мати базовий тип, або бути масивом, вказівником, об'єднанням або, в свою чергу, структурою. Оголошення елементів структури як бітових полів буде описано пізніше; **список-імен-змінних-структури** – це ідентифікатори змінних структурного типу, або вказівник на структуру даного типу, або масив структур даного типу, або функція, яка повертає структуру даного типу.

Ідентифікатори елементів структури повинні відрізнятися між собою, а ідентифікатори елементів різних структур можуть збігатися.

Оголошення дати як структури може мати наступний вигляд:

```
struct date
{
    int unsigned day;  /* День місяця */
    char *month;       /* Найменування місяця */
    int unsigned year; /* Рік */
    char *week_day;    /* День тижня */
```

```
} current_date;        /* Структурна змінна */
```

Тут оголошено тип структури **date**, що описує компонування дати, і конкретна структура для поточної дати типу **date** з ім'ям **current_date**.

У програмі потім можна оголошувати і інші структури типу **date**, використовуючи другий формат оголошення:

```
struct date birth_date;
```

Оголошення інформації як структури о 4-х студентах:

```
struct st_info
{
   char st_surname[20];   /* Прізвище студента */
   char grupa[10];        /* Група*/
   float rating;          /* Рейтінг студента */
} st1,st2,st3,st4;/* Список структурних змінних */
```

Таким чином, маємо два варіанти оголошення структури – з окремим оголошенням структурного шаблона

```
struct date
{
   int day;
   int month;
}
struct date d1, d2, d3;
```

та одночасним оголошенням структурного шаблону і виділенням пам'яті під змінні структурного типу.

```
struct date
{
   int day;
   int month;
} d1, d2, d3;
```

При оголошенні структури часто використовується ключове слово **typedef**, котре створює псевдоніми типів даних.

Слово **typedef** зв’язує об’яву типу даних з ім’ям, яке називається

псевдонімом.

typedef об’ява псевдонім;

Зазвичай псевдонім починається з великої літери, але це лише погодження.

Допускається відсутність ідентифікатора структури (тега структури):

```
typedef struct date
{
   int day;
   int month;
} Date;
```

```
typedef struct
{
   int day;
   int month;
} Date;
```

В мові С оголошення **struct date today** і **Date today** рівнозначні, тобто був створений один і той самий структурний об’єкт **today**.

Ініціалізація і доступ до даних структури

Елементи структури можуть бути проініціалізовані, якщо після структурної змінної заданий символ "=" і вказано в фігурних дужках список початкових значень елементів структури в порядку їх слідування в оголошенні. Елементи списку відокремлюються один від одного комами.

```
struct date birth_date = {7, "лютого", 1978, "середа"};
```

Елемент структури не може бути структурою того ж типу, в якій він міститься, проте він може бути оголошений як вказівник на тип структури, в яку він входить. Це дозволяє створювати пов'язані списки структур, наприклад:

```
struct sample /* Об’ява структур x і y типу sample, */
{ /* які містять наступні елементи: */
   char h;       /* символьну змінну h */
   float *pf;    /* покажчик pf на змінну типу float */
   struct sample *next;  /*  покажчик  next  на  саму
структуру */
} x = {'1', NULL, NULL}, y;
```

При оголошенні структури ім'я типу відіграє таку ж роль, як ключові

слова **int** чи **float** для оголошення базових змінних. Тому можна оголошувати вказівники на структури, наприклад:

```
struct date your_birth_date, * dateptr;
```

Покажчик **dateptr** тепер можна використовувати для доступу до будь-якій структурі типу **date**.

Доступ до окремого елементу структури здійснюється за допомогою конструкції виду:

```
ім'я-структури.елемент-структури
```

Наприклад:

```
struct date today;
today.day = 17;
today.month = 5;
```

Можна створювати вкладені структури, тобто структури, елементами яких можуть бути інші структури, наприклад:

```
struct staff_name
{
   char first[30];   /* Ім'я */
   char last[30];    /* Прізвище */
}
struct student
{
   struct staff_name name;/* Ім'я і прізвище студента
*/
   unsigned int group_number;/* Номер групи студента */
}
```

Або:

```
typedef struct date
{
   int day;
   int month;
   int year;
} Date;
typedef struct time
{
   int hour;
   int minute;
```

```
} Time;
```

Побудуємо вкладену структуру:

```
typedef struct date_time
{
    Date today;
    Time now;
} DateTime;
```

Проініціалізуємо:

```
DateTime dt;
dt.today.year = 2021;
dt.now.hour = 10;
dt.now.minute = 25;
```

Ще один приклад:

```
typedef struct {
    char name[10];
    char surname[20];
    int year;
} ST;
ST st1 = {“Ivan”,“Petrenko”,2001};
```

Спроба проініціалізувати кожний елемент структури окремо усередині її тіла дасть помилку (char name [10] = “Ivan”), оскільки елементи структури не є змінними (а присвоювати значення можна тільки змінним). Тут єдина змінна **st1**.

В програмі можна проініціалізувати елементи цієї змінної наступним чином:

```
strcpy(st1.name, “Ivan”);
strcpy(st1.surname, “Petrenko”);
st1.year = 2001;
```

або

```
gets(st1.name);
```

```
gets(st1.surname);
scanf(“%d”,st1.year);
```

Виведення даних:

```
printf(“Name:%s \n”,st1.name);
printf(“Year:%d ”,st1.year);
```

Присвоювання значення одної структурної змінної іншій можливо тільки, якщо ці змінні мають один і той же тип:

```
Date d1, d2, d3;
```

тоді можна записати **d1 = d2; d2 = d3;**, але порівнювати їх не можна (**if(d1 == d2)**).

Єдиний доступний спосіб достовірно порівнювати структурні змінні – це їх поелементне порівняння:

```
if ((d1.day == d2.day) && (d1.month == d2.month))…
```

Ім’ям структури можна користуватися відразу ж після його появи в програмі, наприклад:

```
struct A
{
   int j;
   char titl [10];
   char x;
} Ab, Ac = {128,"Світ",'Q'};
```

Тут допустимі оператори присвоювання виду **Ab = Ac;**. В цьому випадку елементи структурного об'єкта **Ab** матимуть значення, що збігаються зі значеннями відповідних елементів об'єкта **Ac**.

Якщо були зроблені два такі оголошення:

```
typedef struct
{
   int day;
   int month;
} Date1;
typedef struct
{
   int day;
   int month;
}              Date2;
```

```
Date1 d1; Date2 d2; d1.day = 17; d1.month = 11;
```

то присвоєння **d2 = d1;** призведе до помилки, оскільки тут **d1** і **d2** відносяться до різних типів даних.

Єдино можливі операції над структурами – копіювання, присвоювання (для структур одного типу), взяття адреси та здійснення доступу до її членів. Можна також передавати структури до функцій в якості аргументів і повертати їх від функцій в якості результату.

Масив структур – це масив, кожен елемент якого є структурою. У пам'яті елементи масиву структур розміщуються послідовно. Масиви структур широко використовуються для структурної організації даних в прикладних програмах і системному програмному забезпеченні. З елементів структурного типу можна організувати масиви так само як з елементів стандартного типу. Для оголошення масиву структур слід спочатку визначити структуру, а потім оголосити масив змінних даного типу. Як і масиви змінних, масиви структур індексуються з нуля.

Структури одного типу можна об'єднувати в масиви, наприклад, для структури опису книги:

```
struct book
{
   char title[30];        /* Найменування книги */
   char author[20];       /* Автор книги */
   float price;           /* Ціна книги */
};
```

можна оголосити масив опису книг:

```
struct book library_book [100];
```

Доступ до імені другої книги в масиві буде в цьому випадку виглядати наступним чином: **library_book[1].title**.

Або:

```
typedef struct stud
{
    char name[15];
    char surname[20];
    int year;
} ST;
ST stud[25]; char a;
stud[5].surname = “Petrenko”;
a = stud[5].surname[0]; // a = ‘P’
```

Якщо структура передається як аргумент функції, то, на відміну від масивів, передача відбувається за значенням. Щоб здійснити передачу структури за посиланням, необхідно передати її адресу.

Крім того, що функціям можна передавати структури як аргументи, функції можуть також повертати структури, наприклад:

```
#include <stdio.h>
struct DOT
{
    double x;
    double y;
}
struct DOT Init(double x, double y)
{
    struct DOT A;
    A.x = x;
    A.y = y;
    return A;
}
int main( )
```

```
{
    struct DOT B = Init(2.4, 1.5);
    return 0;
}
```

Якщо дві структурні змінні мають один тип, то одну змінну можна присвоїти інший, наприклад:

```
struct DOT A = {1.2, -2.5}, B; B = A;
```

При цьому значеннями елементів однієї структури будуть значення відповідних елементів іншої структури, навіть якщо серед елементів структури присутні масиви. Також можна створювати масиви структури. Наприклад, можна задати послідовність з 10 елементів точок площини:

```
struct DOT a[10];
```

Тоді звернення до координат точок відбуватиметься так:

```
a[0].x, a[0].y,…..,a[9].x, a[9].y
```

Передача структур як аргументів функцій:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct Marks {
    char name[80];
    int phys;
    int progrm;
    int maths;
};
void set_one(Marks &str) {
    printf ("Student name: ");
    gets(str.name);
    str.phys = 3+rand()%3;
```

```
    str.progrm = 3+rand()%3;
    str.maths = 3+rand()%3;
}
void set_all (Marks *str, int m) {
    for(int i=0;i<m;i++) set_one(str[i]);
}
void get(Marks *str, int m) {
    bool state;
    char s[80];
    do {
        printf("What is the student name?");
        gets(s);
        if(!strcmp(s,"exit")) return;
        state=true;
        for(int i=0;i<m;i++) {
            if(!strcmp(str[i].name,s)) {
                state=false;
                printf("Physiscs:%d\n ",str[i].phys);
                printf("Programing: %d\n",str[i].progrm);
                printf("Mathematics:%d\n ",str[i].maths);
                break;
            }
        }
        if(state) printf("No such student \n");
    } while(true);
}
int main() {
    const int n = 3;
```

```
    Marks students[n];
    set_all(students, n);
    get(students, n);
    return 0;
}
```

Вказівник на структуру оголошується так само, як і вказівник на дані простих типів: використовується операція '*' і вказується тип даних. Тип даних структури вказується завданням ключового слова struct і імені шаблону цієї структури:

```
struct student * p_st1, * p_st2;
```

Вказівник на структуру забезпечує доступ до її елементів двома способами:

(*Вказівник_на_структуру).ім'я_елемента

або **Вказівник_на_структуру -> ім'я_елемента**

У першому випадку круглі дужки необхідні, щоб врахувати пріоритет операцій.

У другому випадку використовується операція "стрілка" (->), яка називається операцією непрямого вибору елемента структурного об'єкта, що адресується вказівником.

Наприклад:

(*pin).mas еквівалентно **pin -> mas**

(*pin).cord[0] еквівалентно **pin -> cord[0]**

```
dateptr = &your_birth_date;
yourday = (*dateptr).day;
```

Тут розкриття посилання має бути в дужках, оскільки пріоритет операції "." вище, ніж пріоритет операції "*".

```
myday = dateptr -> day;
```

Вказівники на структури можуть вводитися і для безіменних (що не мають імен) структурних типів.

```
struct
{
    char * name;
    int age;
} * Person; // вказівник на структуру
```

Якщо структура оголошена за допомогою **typedef**, то при визначенні вказівників назва цього типу може використовуватися без службового слова **struct**.

```
typedef struct {
    int a, b;
} ST;
ST *p1,*p2;
```

При визначенні вказівника на структуру він може бути ініціалізований. Коректно в якості ініціалізованого значення можна застосовувати адресу структурного об'єкта того ж типу, що і тип вказівника, який визначається.

Наприклад:

```
struct particle
{
    double mass;
    float coord[3];
} dot[3], point, *pin;
//ініціалізація вказівників
struct particle *p_d = &dot[1], *p_c = &point;
```

При визначенні елемента структури заборонено вказувати як елемент самого себе (через структурний об'єкт).

```
struct STUD
{// некоректне оголошення поля структури
```

```
    STUD t;
} a, b;
```

Однак елемент структури може бути покажчиком на обумовлену структуру.

Наприклад:

```
struct STUD
{
    STUD *pt;
} a, b;
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ

ЗАГАЛЬНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичними відомостями.

2. Скласти блок-схему алгоритму і програми відповідно до індивідуального завдання.

3. Відлагодити складену програму і вивести результати розрахунку на екран, показати їх викладачу.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Згадані в завданнях масиви структур можна створювати по різному, наприклад, вводити з клавіатури (краще) або жорстко прописувати безпосередньо в коді програми (гірше). Також можна створювати необхідні масиви структур за рахунок випадкової вибірки слів з наперед заданих в програмі масивів назв, прізвищ та т.і. Але в такому випадку, можливо (в залежності від завдання), знадобиться додатковий контроль за тим, щоб у масиві були тільки унікальні записи.

1. Ввести масив структур, який містить прізвища і номери телефонів

співробітників. Впорядкувати масив за прізвищами і роздрукувати вхідний і упорядкований масив.

2. Ввести масив структур, який містить прізвища і екзаменаційні оцінки. Записи впорядкувати за прізвищами. Програма обчислює середній бал для кожного прізвища і створює новий масив, що містить прізвища і середні бали.

3. Ввести масив структур, який містить прізвища співробітників і адреси. Записи впорядкувати за адресами. Необхідно видалити з масиву запис, який містить введене користувачем прізвище.

4. Ввести масив структур, який містить прізвища авторів книг і назви книг. Створити новий масив структур, який додатково містить інформацію про рік видання. Рік видання береться з цілочисленого масиву, який впорядкований відповідно до першого масиву.

5. Ввести масив структур, який містить прізвища і відповідні їм адреси. Масив впорядкувати за прізвищами. Замінити запис, який містить зазначене прізвище, новий.

6. Ввести масив структур, який містить прізвища співробітників, їх адреси та номери телефонів. Скласти програму, яка переносить ці дані в масив структур, який не містить відомості про номери телефонів.

7. Задано два масиви структур, кожен з яких містить прізвища і адреси співробітників. Скласти програму, яка переносить записи з двох масивів в третій, причому з двох однакових записів переноситься тільки один.

8. Задано масив структур, який містить прізвища і адреси. Побудувати новий масив структур, який містить прізвища із заданою початковою літерою і відповідні їм адреси.

9. Задано два масиви структур, кожен з яких містить прізвища і адреси студентів. Скласти програму, яка друкує однакові записи обох масивів в алфавітному порядку.

10. Задано два масиви структур, кожен з яких містить відомості про прізвища та адреси. Необхідно записи, які є в першому масиві і яких немає в другому масиві, переписати в третій масив.

11. Задано масив структур, який містить прізвища студентів та їхню екзаменаційну оцінку. Скласти програму, яка знаходить і друкує прізвища людей, які отримали задану оцінку.

12. Задано два масиви структур. Перший містить прізвища та адреси, другий – відповідні номери телефонів. Скласти програму, яка об'єднує записи обох масивів в третій масив.

13. Заданий масив структур, який містить прізвища і відповідні їм номери телефонів. Масив впорядкувати за прізвищами. Програма повинна вставляти новий запис до відповідного місця масиву в залежності від прізвища, розсуваючи інші елементи масиву.

14. Заданий масив структур, який містить прізвища і назви груп. Масив впорядкувати за прізвищами. Програма повинна видаляти запис, який містить задане прізвище, зсуваючи елементи вперед (заповнюючи ними порожнє місце).

15. Ввести масив структур, який містить прізвища та номери телефонів. Програма замінює запис, який містить зазначене прізвище, на новий, введений користувачем. Впорядкувати отриманий масив за прізвищами.

16. Задано два масиви структур, які містять прізвища та адреси. Масиви впорядкувати за прізвищами. Скласти програму, яка переносить записи з двох заданих масивів в третій, причому з двох однакових записів залишається тільки один.

17. Заданий масив структур, який містить прізвища і назви груп. Масив впорядкувати за прізвищами. Переписати в третій масив записи, які містять введену користувачем назву групи.

18. Задано два масиви структур, які містять прізвища та оцінки. Масиви впорядкувати за прізвищами. Програма повинна переносити записи першого

масиву, які відсутні в другому масиві, в третій масив.

19. Заданий масив структур, який містить прізвища та екзаменаційні оцінки. Масив впорядкувати за прізвищами. Скласти програму, яка обчислює середній бал всього масиву.

20. Заданий масив структур, який містить прізвища студентів і суми їхніх стипендій. Необхідно видалити з цього масиву всі записи, які мають величину стипендій нижче середнього рівня.

21. Заданий масив структур, який містить прізвища студентів і роки народження. Необхідно побудувати новий масив, в якому будуть тільки записи з роком народження від 2001.

22. Заданий масив структур, який містить назви факультетів і назви спеціальностей. Побудувати новий масив, доповнений записами, введеними користувачем, і упорядкований за факультетами.

23. Заданий масив структур, який містить спеціальності, курси та назви груп студентів. Вибрати з цього масиву записи, які відносяться до спеціальності «Комп’ютерні науки», і роздрукувати їх, а також окремо залишок масиву.

24. Заданий масив структур, який містить спеціальності, курси, групи студентів і середні бали груп. Побудувати новий масив, який містить відомості по групах студентів і середніх балах, якщо він не менше 4.5. Роздрукувати вхідний і вихідний масиви.

25. Задано два масиви структур. Один містить прізвища та імена, інший – прізвища та адреси. Необхідно побудувати новий масив, впорядкований за прізвищами, який містить прізвища, імена та адреси. Якщо є записи, які не узгоджуються, то повідомити про це.

26. Заданий масив структур, який містить прізвища студентів та їхні екзаменаційні оцінки. Визначити відсоток студентів, які мають середній бал більше 4.5. Роздрукувати вхідний масив та прізвища студентів з відповідним середнім балом.

27. Заданий масив структур, який містить прізвища авторів, назви книг, роки видання. Знайти в даному списку книги, в назві яких зустрічається деяке ключове слово. Ключове слово ввести з клавіатури.

28. Написати програму формування відомості про успішність студентів. Кожен запис цієї відомості повинен містити номер групи, прізвище студента, середній бал за останню сесію. Необхідно роздрукувати списки студентів по групах. У кожній групі прізвища студентів розмістити в порядку зменшення середнього балу.

29. Заданий масив структур, що містить ім'я, прізвище, по батькові студента, назву навчального закладу, номер групи. Визначити прізвища студентів-учнів заданої групи та заданого навчального закладу.

30. Заданий масив структур, який містить прізвища студентів та їхні екзаменаційні оцінки. Знайти студентів, які мають хоча б одну оцінку <=2, та видалити їх зі списку.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке структура, як визначається?
2. У чому важлива відмінність типів масив та структура?
3. Як розміщуються у пам'яті елементи структури?
4. Чому розмір структури не завжди збігається з сумарним розміром її полів?
5. Яким чином можна звернутися до даних структури?
6. У чому відмінність прямого та непрямого доступу до полів структури?
7. Чи завжди можна виконати безпосередньо операцію присвоєння значень об'єктів структури з однаковим набором полів?
8. При якому оголошенні структурних об'єктів можна виконати безпосередньо операцію присвоєння значень об'єктів структури?

9. Для моделювання яких даних доцільно використати масив структур? Як утворюються масиви структур?

10. Як створюються вказівники на екземпляри структури і як через вказівник здійснюється звернення до полів структури?

11. Представником якого типу даних є структура мови C/C++?

12. Які існують три основні форми оголошення структур?

13. За допомогою яких операцій можна здійснити доступ до елементів структур (полів даних)?

14. За допомогою якої операції обчислюється розмір пам'яті, яку займає структура?

15. Для вирішення яких завдань використовуються масиви структур?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Трофименко О.Г. С++. Алгоритмізація та програмування : підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. Одеса : Фенікс, 2019.

2. Васильєв, О. Програмування на С++ в прикладах і задачах : навчальний посібник / О.Васильєв. Київ :Видавництво Ліра-К, 2020.

3. Тверитникова О. Є. Базові алгоритми та основи програмування: теорія і практика : навчальний посібник для студентів спеціальностей "Автоматизація та комп’ютерно-інтегровані технології", "Метрологія та вимірювальна техніка" усіх форм навчання /О.Є. Тверитникова, В.А. Крилова, О.Г. Васильченков ; Міністерство освіти і науки України, Національний технічний університет "Харківський політехнічний інститут". Харків :НТУ "ХПІ" .А.М. Панов, 2020.

4. Ковалюк Т. В. Алгоритмізація та програмування: підручник для студентів вищих навчальних закладів, що навчаються за напрямами "Комп’ютерні науки", "Комп’ютерна інженерія", "Програмна інженерія" / Т.В. Ковалюк ; Міністерство освіти і науки, молоді та спорту України. Львів :Видавництво "Магнолія 2006". 2019.

5. Козак Л. І. Основи програмування: навчальний посібник / Л.І. Козак, І.В. Костюк, С.П. Стасевич. – Львів : Новий Світ. 2000. 2019.

6. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. К.: Видавничо-поліграфічний центр "Київський університет". 2012. 175 с.

ДОДАТОК А

Функції роботи з випадковими числами в С++

Прототип	Опис	Заголовний файл
`int random (int num);`	Повертає випадкове ціле додатне число в діапазоні від 0 до (num-1). Наприклад, команда `int x= random(100);` надасть змінній випадкове число в межах від 0 до 99	`stdlib.h`
`int rand(void);`	Повертає випадкове ціле число у діапазоні від 0 до символьної константи `RAND_MAX`. Ця константа визначена у файлі `<stdlib.h>` значенням 32767. Початкове значення генерованого числа задається звертанням до функції `srand()`. Для того щоб генерувати різні послідовності випадкових чисел, початкова точка має обиратися випадково. Для цього можна використовувати функцію `randomize()`. Наприклад, команда `int x=rand() % 100;` надасть змінній випадкове число в межах від 0 до 99	`stdlib.h`
`void randomize(void);`	Ініціалізує генератор випадкових чисел, використовуючи поточний час, який повідомляє комп'ютер	`stdlib.h, time.h`

`void srand (unsigned seed);`	Встановлює початкове число `seed` для генерованої за допомогою функції `rand()` послідовності випадкових чисел. Якщо `seed=1`, генератор випадкових чисел ініціалізується значенням за замовчуванням. Наприклад, команди
	`int i; time_t t;`
	`srand((unsigned) time(&t));`
	`for(i=0; i<10; i++)`
	`printf("%d\n", rand() % 100);`
	генеруватимуть послідовність з 10-ти випадкових чисел
		`stdlib.h`