Управління освіти, науки та молоді
Волинської облдержадміністрації
Волинська обласна Мала академія наук

Мазурик В.В., Процик А.П.

ОРГАНІЗАЦІЯ НАУКОВО-ДОСЛІДНИЦЬКОЇ ДІЯЛЬНОСТІ СЛУХАЧІВ ВІДДІЛЕННЯ КОМП'ЮТЕРНИХ НАУК МАЛОЇ АКАДЕМІЇ НАУК УКРАЇНИ

Луцьк – 2016

Мазурик В.В. Організація науково-дослідницької діяльності слухачів відділення комп’ютерних наук Малої академії наук України / В.В. Мазурик, А.П. Процик. – Луцьк, 2016. – 79 с.

Рекомендовано до друку методичною радою КУ «Волинська обласна Мала академія наук (протокол № 1 від 10.05.2016 року).

Рецензенти: Михайлюк В.О., доктор фізико-математичних наук, професор, завідувач кафедри прикладної математики та інформатики Східноєвропейського національного університету імені Лесі Українки;

Щегельський Т.С., керівник гуртків відділення комп’ютерних наук КУ «Волинська обласна Мала академія наук».

Зміст

ВСТУП

Три шляхи ведуть до знань:
шлях роздумів - цей шлях найблагородніший,
шлях наслідування – це найлегший,
і шлях дослідження – найскладніший
Конфуцій

В умовах суспільних, економічних та політичних викликів сьогодення потреба у високопрофесійних, інтелектуальних, творчих фахівцях, які здатні здійснити розвиток науки, економіки та новітніх технологій, є надзвичайно високою. Мала академія наук України забезпечує початкову ланку формування наукової еліти України. Структура Малої академії наук України представлена 12 науковими відділеннями, які налічують більше шістдесяти сучасних напрямів досліджень вітчизняної та світової науки. У відділенні комп'ютерних наук навчаються одні з найпрогресивніших школярів України, які прагнуть володіти новітніми ІТ-технологіями та успішно реалізовувати їх через власні розроблені проекти та програми.

Навчально-виховна робота у відділенні комп'ютерних наук Волинської обласної МАН здійснюється відповідно до трьох навчально-організаційних рівнів – початкового (5-7 кл.), основного (8-9 кл.) та вищого(10-11 кл.).

Початковий рівень спрямований на ознайомлення слухачів з основами алгоритмізації та програмування, проектно-дослідницької діяльності, широке використання ними елементів дослідницької роботи, формування нескладних гіпотез науково-дослідницького характеру та перевірку їх при розв'язуванні практичних завдань з використанням інформаційно-комунікаційних технологій. Водночас важливим завданням початкового навчання у відділенні є виявлення задатків та схильностей, сфери потенційної обдарованості дитини, аналіз особистих предметних вподобань школярів, їх інтересів до певних видів навчально-дослідницької діяльності, закріплення та вдосконалення знань з математики, розвиток алгоритмічного, креативного та логічного мислення, творчих здібностей. Відповідно до навчальних програм поступово відбувається

ознайомлення слухачів з науковою термінологією та методами наукового пошуку у галузі комп'ютерних наук, оволодіння уміннями та навичками пошукової та дослідницької діяльності. Тим самим формуючи інтерес до наукової творчості, прагнення довести правильність власної позиції, потреби у саморозвитку та творчості.

Основний рівень навчання у відділенні комп'ютерних наук забезпечує продовження роботи з удосконалення бази спеціальних знань, умінь та навичок з програмування вихованців, розвиток їх інтересів до систематизованої навчально-дослідницької. Важливим є включення учнів в організовану практичну навчально-дослідницьку діяльність, виконання ними індивідуальних творчих завдань дослідницького характеру, що сприятиме підвищенню ефективності їх самостійної науково-дослідницької роботи. Вагомим у ствердженні юних науковців є презентація та захист перших власних програмних проектів на виставках-конкурсах та науково-практичних конференціях.

Вищий рівень спрямований на організацію систематизованої індивідуальної та колективної пошуково-дослідницької роботи слухачів наукових секцій під керівництвом науковців та педагогів-практиків. На цьому етапі продовжується робота з удосконалення бази знань слухачів з базової дисципліни та основ дослідницької діяльності. Реалізації науково-дослідницьких здібностей сприяє участь у інтелектуальних масових заходах (конкурсах-захистах науково-дослідницьких робіт, турнірах, наукових конференціях) обласного та всеукраїнського рівнів, у Міжнародних та всеукраїнських наукових та освітніх проектах.

Відповідно таке поетапне включення школяра в систему діяльності МАН, сприяє, по-перше, кращій адаптації учня до незвичних (не репродуктивних, а творчо-пошукових) форм навчальної діяльності, а по-друге, забезпечує повноцінну підготовку слухача до організації та проведення відповідальної науково-дослідницької роботи під керівництвом досвідчених вчителів та наукових працівників.

Запропоновані матеріали розроблені на допомогу керівникам гуртків та слухачам відділення комп'ютерних наук. У першому розділі методичних матеріалів наведено практичні рекомендації щодо перспективного напряму досліджень у відділенні комп'ютерних наук, вибору теми, дієвих методів, збору та обробки необхідного матеріалу, структури, змісту та оформлення наукової роботи, загальні вимоги до програмного продукту слухача МАН. У другому розділі для програмістів-початківців подані матеріали практикуму з основ програмування С++. До кожної з підтем наведені практичні рекомендації, роз'яснення та розв'язки задач.

Науково-дослідницька робота з інформатики слухача МАН – перший крок на шляху до серйозних наукових досліджень та відкриттів у цій перспективній галузі. Розвиваючи науково-дослідницькі здібності та власний хист до програмування ще з шкільної лави, діти відкривають для себе неймовірні перспективи власного розвитку та самовдосконалення, водночас забезпечуючи відродження авторитету України як високоінтелектуальної держави у світовій спільноті.

РОЗДІЛ 1. ОСНОВНІ ЕТАПИ РОБОТИ НАД НАУКОВИМ ДОСЛІДЖЕННЯМ З ІНФОРМАТИКИ

1.1. Перспективні напрями наукових досліджень

Тематика наукового дослідження має відповідати одній із шести секцій відділення комп'ютерних наук:

- комп'ютерні системи та мережі;
- безпека інформаційних та телекомунікаційних систем;
- інформаційні системи, бази даних та системи штучного інтелекту;
- технології програмування;
- Інтернет-технології та Web-дизайн;
- мультимедійні системи, навчальні та ігрові програми.

Коротко зазначимо особливості науково-дослідницьких робіт кожної секції окремо.

До першої секції відносяться роботи, що стосуються розробок у галузі комп'ютерних систем та мереж. До цієї секції можна віднести програми для роботи із комп'ютерною мережею: моніторинг роботи комп'ютерних мереж (обрахунок трафіку, перевірка завантаженості вузлів КМ, ведення статистики звернень до ресурсів КМ, тощо), передачі інформації по КМ, зокрема програми для передачі файлів, обміну повідомленнями, роботи по створенню програм-архіваторів. Також на сьогодні є актуальними роботи, що стосуються створення програм для організації паралельних та розподілених обчислень. Розподілені обчислення – спосіб вирішення трудомістких обчислювальних завдань з використанням двох і більше комп'ютерів, об'єднаних у мережу. Системи розподілених обчислень запрошують велику кількість користувачів для розв'язання об'ємних завдань, занадто складних для одного комп'ютера.

У секції безпеки інформаційних та телекомунікаційних систем найбільш актуальними є роботи, що стосуються розробки алгоритмів та на їх основі програм для шифрування даних, створення криптографічних систем на основі

власних та вже існуючих алгоритмів. До цієї секції також відносять роботи, пов’язані із створенням програм-антивірусів, програм для виявлення шкідливих або потенційно шкідливих програм, програм для фільтрування небажаної електронної пошти, програм для обмеження доступу до даних користувача (захист паролем, приховання даних та ін.), програм для розробки та візуалізації політик безпеки, програм для модулів вже існуючих програм для захисту від DoS, DDoS ((Distributed) Denial-of-service – напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними до користувачів) атак на сервери, програм для тестування та оцінки надійності захисту операційних систем, мережі та окремих програмних продуктів.

У секції інформаційних систем, баз даних та систем штучного інтелекту, слухач обирає тематику, що стосується розробки алгоритмів та на їх основі програм для надійного зберігання даних, швидкого пошуку за різноманітними критеріями, розробки великих та надвеликих баз даних для роботи із складними проектами (наприклад: для збереження даних, необхідних для пошукових серверів), створення оболонок та інструментів адміністрування для вже існуючих баз даних, розробка систем керування базами даних(щось на зразок MS Access), моделювання штучних нейронних мереж. До цієї секції відносять також програми для розпізнавання образів (зображень, звуку, відеорядів та ін.), розробки алгоритмів та програм для представлення знань в системах штучного інтелекту та семантичного аналізу текстів.

До секції технології програмування відносять роботи, пов’язані з написанням програм для налаштування, тестування та моніторингу роботи операційних систем, програми для розв′язку обчислювальних задач (наприклад: програма для визначення необхідної кількості деревини для виготовлення виробу за заданими кресленнями), розробки програм для автоматизації наукових досліджень (програми для побудови графіків функцій, програми для точного та наближеного розв’язування рівнянь, систем рівнянь, програми для проведення статистичних обрахунків (знаходження середніх значень, моди, дисперсії,

медіани, середньо квадратичного відхилення, тощо)), створення надбудов для вже існуючих програм з метою полегшення обміну даними між різними програмами (конвертори форматів файлів).

В секції Інтернет-технологій та WEB-дизайну перспективними є наступні напрями наукових досліджень:

- розробка та супровід WEB-сайтів школи, класу (як статичних на мові HTML так і динамічних із використанням PHP, ASP.NET, Flash, MS Siverlight, тощо);

- проектування систем керування контентом та модулів і плагінів для існуючих систем керування контентом (Joomla, e107, Wordpres та ін.);

- розробка фреймворків для розробки сайтів;

- створення сайтів соціальних мереж;

- розробка додатків для сайтів Facebook, Vkontakte;

- розробка сервісів на основі технології «хмарних обчислень» (технологія обробки даних, в якій програмне забезпечення надається користувачеві як Інтернет-сервіс. Користувач має доступ до власних даних, але не може управляти і не повинен піклуватися про інфраструктуру, операційну систему і власне програмне забезпечення, з яким він працює. «Хмарою» метафорично називають Інтернет, який приховує всі технічні деталі. Згідно документу IEEE, опублікованому в 2008 році, «Хмарні обчислення — це парадигма, в рамках якої інформація постійно зберігається на серверах в мережі Інтернет і тимчасово кешується на клієнтській стороні, наприклад, на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах тощо).

- розробка поштових серверів, Інтернет-магазинів.

- розробка алгоритмів роботи дуже завантажених сайтів, тобто таких, які можуть коректно опрацьовувати одночасне звернення кількох тисяч користувачів.

Серед програмних продуктів, що відносяться до секції мультимедійних систем, навчальних та ігрових програм, найбільш актуальними є:

1. Навчальні програми. Вони поділяються на такі типи: тестуючі, інформаційні, демонстраційні, моделюючі, експертні системи; можливі також комбіновані варіанти (наприклад, інформаційно-тестові). Головний критерій якості навчальної програми – це підтвердження того, що використання її в процесі навчання дає відносно хороші результати. А для цього необхідно дослідити потреби навчальних закладів, методик навчання і самого матеріалу, що пропонується вивчати. Робота повинна якнайповніше охоплювати весь матеріал, наявний з цієї проблеми.

2. Програми для візуалізації різноманітних процесів (хімічні реакції, фізичні досліди, робота окремих видів техніки та її вузлів).

3. Ігрові програми. Ці програми передусім повинні носити розвивальний характер і не викликати агресивності.

4. Мультимедійні системи (програми для перегляду та організації бібліотек малюнків та фотографій, програми для створення мультимедійних програвачів: музичні, відео, універсальні).

1.2. Основні етапи роботи над науковим дослідженням у відділенні комп'ютерних наук

1.2.1. Вибір теми наукового дослідження

Однією з умов успішного виконання науково-дослідницької роботи учнів є дослідження методами навчання з елементами наукових методів пізнання, а також відповідність між віковими та розумовими особливостями розвитку школярів. Занадто складні для розуміння юним дослідникам або зовсім недоступні методи розв'язання проблеми можуть спричинити втрату інтересу до виконання науково-дослідницької роботи.

Одним з перших кроків керівника гуртка є вивчення науково-пізнавальних інтересів учнів, що впливає як на вибір теми дослідження, так і на саму роботу. Відомо, що навіть найцікавіша тема, зумовлена потребами часу або нав'язана учневі педагогом, не сприятиме успішному виконанню роботи. Пріоритетним,

визначальним фактором у виборі теми є стійкий пізнавальний інтерес до неї дослідника та бажання внести щось нове в її розкриття.

Тематика наукового дослідження має бути тісно пов’язана з програмною розробкою. Тут можливі два варіанта:

1) автор майбутньої роботи вже має деякі власні напрацювання, відповідно до яких він хоче обрати тему;

2) автор хоче спочатку вибрати тему роботи, а потім вже розпочати розробку програми.

Дослідження має бути оригінальним, практичним, виконане самостійно, з урахуванням досягнень інших науковців у цій проблематиці. Досліднику бажано знати про найбільш розповсюджені програмні продукти, що стосуються цієї проблематики. Також можуть створюватися програми, в яких пропонується оригінальна, але менша за розміром і швидша у роботі реалізація існуючої програми (предметом дослідження у цьому випадку будуть способи її реалізації).

1.2.2. Збір і обробка необхідного матеріалу

Розпочати роботу доцільно з визначення переліку джерел і літератури, необхідного для самостійного вивчення і розкриття теми. Пошукові цієї літератури допоможуть систематичний та алфавітний каталоги, а також різноманітні бібліографічні покажчики. Літературу доцільно записувати на окремі картки чи в зошиті, зазначаючи всі дані про працю - прізвище та ініціали автора, назву монографії, статті чи збірника статей, тез, місце, рік видання, назву видавництва чи журналу, кількість сторінок, короткий зміст або цитати.

У першу чергу варто ознайомитися з відповідною літературою із систем програмування, які будуть використовуватись при розробці програмного продукту. Це може бути література довідкового характеру, а також література більш загального характеру, яка містить практичні рекомендації щодо розробки окремих систем або ж описує певні методи і принципи програмування.

Посилює достовірність одержаних результатів комбіноване використання джерел різних типів, але дуже важливо, щоб ці джерела точно відповідали поставленим завданням і співвідносились із темою наукової роботи.

Варто, однак, пам’ятати, що наукова робота тільки базується на відібраному матеріалі, а головне в дослідженні – це власні ідеї.

1.3. Структура та зміст наукової роботи

Науково-дослідницька робота з інформатики повинна мати таку структуру:

1. Титульна сторінка.
2. Тези.
3. Зміст.
4. Вступ.
5. Основна частина.
6. Висновки.
7. Список використаних джерел.
8. Додатки (за потреби).

Невід’ємною частиною науково-дослідницької роботи є власний програмний продукт, який має знаходитися на електронному носії.

Титульна сторінка.

Титульна сторінка має єдиний загальний стандарт (додаток 1).

Тези.

У тезах (текст обсягом 1 сторінка) дається стисла характеристика змісту науково-дослідницької роботи з визначенням основної мети, актуальності та завдань наукового дослідження. Також у них зазначаються висновки та отримані результати проведеної роботи.

У заголовку тез наводяться такі дані: назва роботи; прізвище, ім’я, по батькові автора; назва базового навчального закладу; клас; населений пункт; прізвище, ім’я, по батькові та посада наукового керівника (додаток 2).

Зміст.

Зміст є другою сторінкою, де визначено структуру наукової роботи з послідовною назвою всіх розділів, підрозділів, висновків, використаних джерел, назви додатків та номери сторінок, з яких вони починаються (додаток 3).

Вступ.

Вступ повинен мати обсяг 1-2 сторінок. Вступ має відповідні складові частини, що розташовуються у певній послідовності: актуальність дослідження, об'єкт та предмет дослідження, мета, завдання, наукова новизна, теоретичне та практичне значення.

Для аналізу всіх структурних елементів наукової роботи візьмемо для прикладу тему, назва якої може бути сформульована так: **«Програма для створення та симуляції нейронних мереж «Neuro Master»**.

Актуальність теми. Автор дослідження дає пояснення, чому на його думку, обрана тема стала об'єктом наукового аналізу, обґрунтовує доцільність її розробки. Висвітлення актуальності не повинно бути багатослівним. Досить кількома реченнями або абзацами висловити головне – сутність проблеми або наукової задачі.

Наприклад, для теми нашої роботи актуальність може бути такою:

«*Для наближення процесу обробки інформації до такого, який відбувається в мозку, була розроблена математична модель нервової системи – нейронна мережа. Водночас постала проблема реалізації цієї мережі. Двома основними способами є створення спеціалізованих мікросхем (апаратний підхід) та розробка програмного забезпечення. Перший спосіб дає досить високу швидкодію за внаслідок використання аналогових схем, але через свою високу ціну та складність у впровадженні не набув широкого розповсюдження. Другий досить просто реалізується за допомогою спеціалізованих програм-симуляторів нейронних мереж, але він не дає такої швидкодії, як використання аналогових схем. Незважаючи на це, програмно реалізовані нейронні мережі є основним інструментом для дослідження цього питання. Наведені вище міркування*

обґрунтовують ***актуальність*** *проблеми моделювання нейронних мереж та обумовлюють вибір теми наукового дослідження».*

Предмет та об'єкт дослідження. Предмет та об'єкт дослідження визначаються на основі аналізу стану вивчення тієї чи іншої наукової проблеми і відображають, яку саме частину проблеми буде розглянуто. Об'єкт дослідження – це процес або явище, що породжує проблемну ситуацію і обране для вивчення. Предмет дослідження – це частина об'єкту, що вивчається автором роботи, його якості.

Наприклад, у нашому випадку «***Об'єктом*** *дослідження є штучні нейронні мережі.* ***Предмет дослідження*** *– багатошарові перцептронні нейронні мережі з алгоритмом навчання «зворотнього поширення похибки»*.

Мета дослідження. Мета дослідження дає можливість окреслити коло питань, які не знайшли достатнього висвітлення у існуючих дослідженнях, та що саме буде розглядати автор у науково-дослідницькій роботі. Мета формулюється чітко та зрозуміло і тісно пов'язана з його об'єктом і предметом, а також з кінцевим результатом і шляхом його досягнення. Слід запобігати таким висловлюванням: «Дослідження…», «Вивчення…», оскільки це засіб досягнення мети, а не власне мета. Тому варто дотримуватись таких формулювань: «Мета роботи полягає в…», «Довести, що…», «Створити…», тощо. Наприклад: ***«Мета наукової роботи*** *– створити цілісний програмний продукт, який би поєднував в собі як основні засоби для конструювання, навчання та симуляції штучних нейронних мереж, так і зручний користувацький інтерфейс».*

Завдання дослідження. Виходячи з поставленої мети, визначаються дослідницькі завдання, яких повинно бути не менше ніж 3-4 (проаналізувати, розглянути, висвітлити, дослідити тощо). Вирішення кожної поставленої задачі – це етап дослідження. Завдання можуть передбачати висунення проблеми, виявлення нових фактів, встановлення нових зв'язків, нову постановку відомої проблеми, оригінальні висновки та рекомендації щодо впровадження отриманих експериментальних даних.

Наприклад:

*«Реалізація мети потребувала вирішення таких **завдань**:*

1. Дослідити теоретико-математичну основу штучних нейронних мереж.

2. Вибрати тип мережі та тип навчання для подальшої роботи з ними.

3. Вибрати середовище та засоби програмування для створення програми.

4. Дослідити питання зручного введення початкових даних та наочного представлення вихідних результатів.

5. Проаналізувати питання моделювання нейронних мереж для розв'язання прикладних задач».

Наукова новизна отриманих результатів повинна бути обґрунтована та аналогічно доведена із зазначенням відмінностей порівняно з тими результатами, які були відомі раніше. Наприклад: *«**Новизною роботи** є реалізація в комплексному підході програмної розробки із зручним користувацьким інтерфейсом. Крім того, для розширення класу задач, які можна реалізувати за допомогою програми «NeuroMaster», була розроблена система плагінів для підготовки вхідних даних. Таким чином, була усунена необхідність впроваджувати в програмі підтримку різноманітних типів вхідних даних: графічних, звукових, відео та числових. Для кожної конкретної задачі можна створити власний плагін та приєднати його до основної програми. Це дозволить максимально повно використати потенціал штучної нейронної мережі, адже вхідні дані, якими вона буде оперувати, будуть найбільш точно відповідати поставленій задачі».*

Вступну частину наукової роботи можна завершити повідомленням про те, на яких конкурсах, науково-практичних конференціях, інших заходах оприлюднені результати досліджень (апробація результатів), чи є публікації в учнівській пресі, різних регіональних виданнях.

Основна частина.

Основна частина роботи обов'язково повинна складатися з декількох розділів (принаймні - двох), які в свою чергу можуть мати підрозділи, пункти та

підпункти. Кожен розділ (підрозділ, пункт чи підпункт) повинен мати назву. Основна частина наукової роботи з інформатики складається з теоретичної (1-2 розділи) та практичної частин (2 або 3 розділ).

Перший розділ – теоретичний. В ньому потрібно висвітлити теоретичні основи роботи, розкрити зміст використаних термінів, стан вивчення питання за літературними джерелами.

За теоретичною частиною йде практична частина, у якій подають опис власного програмного продукту, в якому треба вказати: структуру розробки, її особливості, можливості, призначення, властивості інтерфейсу. Можна коротко описати роботу програми, подати певні інструкції та іншу довідкову інформацію. Обов'язково слід зазначити системні вимоги до програмного продукту та умови, у яких він буде коректно працювати.

Висновки. Висновки є логічним завершенням наукової роботи. У них виокремлюють основні результати дослідження, наголошуючи на тому, що автором зроблено самостійно; вказують, які перспективи має розроблений програмний продукт і яким чином його можна вдосконалити у подальшому. Наприклад: *«Провівши дослідження проблеми побудови штучних нейронних мереж та розглянувши велику кількість архітектур цих мереж, було створено цілісний програмний продукт, який має зручний користувацький інтерфейс та зберігає основні принципи побудови, навчання та симуляції нейронних мереж. Цей факт дозволяє стверджувати про виконання поставленої мети.*

Розроблена програма оперує багатошаровими нейронними мережами без зворотніх зв'язків. Алгоритмом навчання було вибрано алгоритм зворотного поширення похибки, як найбільш пристосований для впровадження на ЕОМ. Отримані результати роботи продемонстрували, що ще є багато питань для досліджень, такі як: ефективність структури мережі, збіжність мережі при різних вхідних даних та багато інших. Це дозволяє постійно вдосконалювати як сам програмний продукт, так і алгоритми, які в ньому використовуються».

Список використаних джерел відображає обсяг використаних джерел та ступінь вивченості досліджуваної теми. Він повинен містити бібліографічний опис джерел, що використані дослідником. Укладаючи його, необхідно дотримуватися вимог державного стандарту. Кожне джерело необхідно починати з нового рядка згідно алфавітного порядку авторів та назв праць, спочатку видання українською мовою, потім – іноземними. Бібліографічні записи у списку повинні мати порядкову нумерацію. Також джерела у списку можна розміщувати іншими способами: за порядком появи посилань у тексті або у хронологічному порядку. Спосіб оформлення списку використаних джерел визначає автор наукової роботи за погодженням з науковим керівником.

Додатки. У додатках містяться допоміжні або додаткові матеріали, необхідні для повноти сприйняття роботи, кращого розуміння отриманих результатів: вікна програми, структуру файлів, програмну реалізацію основних модулів тощо.

1.4. Оформлення наукової роботи

Правильне оформлення наукової роботи є важливим елементом її виконання і, безумовно, впливає на загальну оцінку роботи. Передусім звертається увага на змістовний аспект викладу матеріалу (логічність і послідовність, загальна грамотність тощо), а також на текст роботи, список використаних джерел і додатки.

Оформлення тексту.

Обсяг науково-дослідницької роботи складає 15-20 друкованих сторінок. Комп’ютерний набір: текстовий редактор Word, шрифт 14, Times New Roman, через 1.5 інтервали, з одного боку білого паперу формату А-4.

Поля: ліве, верхнє та нижнє – не менше 20 мм; праве – не менше 10 мм.

Абзацний відступ повинен бути скрізь однаковий і дорівнювати п’яти знакам.

Оформлення елементів тексту. Розділи і параграфи наукової роботи прийнято нумерувати арабськими цифрами без знаку №. Всі сторінки, враховуючи тези та додатки, нумеруються. Першою сторінкою вважається

титульна, на якій цифра 1 не ставиться, другою – зміст, яка також не нумерується. На наступній сторінці ставиться цифра 3 і далі згідно з порядком. Порядковий номер сторінки проставляється у правому верхньому куті сторінки без крапки в кінці.

Ілюстрації (фотографії, схеми, графіки, діаграми) позначають словом «Рис.» і нумерують послідовно в межах розділу, за винятком ілюстрацій, поданих у додатку. Номер ілюстрації повинен складатися з номера розділу і порядкового номера ілюстрації, між якими ставиться крапка. Наприклад: *Рис.1.2.(другий рисунок першого розділу).* Номер ілюстрації, її назву і пояснювальні підписи розміщують послідовно під ілюстрацією. Якщо в науковій роботі одна ілюстрація, її нумерують за загальними правилами.

При графічному зображенні яких-небудь процесів, дані, які охоплюють різні періоди часу на графіках повинні бути пропорційні величинам тривалості періодів. Цифрові значення величин відображаються на графіках з точністю до однієї десятої.

Таблиці нумерують послідовно (за винятком таблиць поданих у додатках) в межах розділу. В правому верхньому куті над відповідним заголовком таблиці розміщують напис «Таблиця» із зазначенням її номера. Номер таблиці повинен складатися з номера розділу і порядкового номера таблиці, між якими ставиться крапка, наприклад: *Таблиця 1.2. (друга таблиця першого розділу)*. Якщо в науковій роботі одна таблиця, її нумерують за загальними правилами. Кожна таблиця повинна мати назву, яку розміщують над таблицею симетрично до тексту. Назву і слово «Таблиця» починають з великої букви.

Дробові числа подаються у вигляді десяткових дробів. В комірках таблиці не можна залишати вільні місця. Якщо дані відсутні, то ставиться «тире».

Математичні формули і вирази, які є в роботі, необхідно пояснити. Якщо формула запозичена з літератури, то можна обмежитися посиланням на джерело і розкрити суть символів, що входять в неї. Оригінальні формули пояснюються в ході їх обґрунтування. Всі математичні вирази друкуються прописними буквами.

Формули в науковій роботі нумеруються в межах розділу арабськими цифрами. Номер формули складається з номера розділу і порядкового номера формули, розділених крапкою, наприклад: *1.4.(четверта формула першого розділу).*

При написанні роботи потрібно давати **посилання** на літературні джерела, з яких запозичені матеріали. Оформлення посилання в тексті: [1, 20] – означає, що цитата запозичена із 20 сторінки 1 джерела в бібліографії. Посилання на декілька видань оформлюється так: [5; 7; 10; 23] (вказуються порядкові номери у списку джерел).

При посиланні на ілюстрацію вказують «рис. 1.2», на таблицю – «у табл. 1.3». Якщо є декілька посилань на одну таблицю, то слово «дивись» скорочують – «див. табл.1.2.».

Звертання до цитат є обов'язковим елементом наукової роботи. Звертатись до них доцільно тільки у тих випадках, коли цитата дійсно містить потрібну аргументацію. Слід пам'ятати, що цитування - це не засіб для захисту авторитетною думкою власного тексту або аргументованого переконання опонентів. Цитата повинна бути лише засобом розвитку власних міркувань за допомогою раніше висунутих іншими авторами правильних чи неправильних (на думку автора даної роботи) ідей. Щоб вказати джерела цитат, запозичень конкретних положень, формул, статистичних даних та деяких інших відомостей, а також, щоб навести відомості про проаналізовані у тексті раніше опубліковані праці, застосовують бібліографічні посилання.

Відомості про джерела складаються відповідно до вимог, зазначених у стандартах: ДСТУ ГОСТ 7.1: 2006 «Система стандартів з інформації бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання», затверджений наказом Державного комітету України з питань технічного регулювання та споживчої політики від 10.11.2006 № 322; ДСТУ 4331:2004 «Правила описування архівних документів», затверджені наказом Держспоживстандарту України від 17.08.2004 № 181; ДСТУ 3582: 2013

«Інформація та документація. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила», затверджений наказом Мінекономрозвитку від 22.08.2013 № 1010; ДСТУ 3008:95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення», затверджений наказом Держстандарту України від 23.02.1995 № 58.

Додатки оформляють як продовження наукової роботи на останніх сторінках, розміщуючи їх у порядку появи посилань у тексті. Кожний додаток починається з нової сторінки, має заголовок, який друкують вгорі малими буквами з першої великої симетрично до тексту сторінки. Посередині рядка над заголовком або праворуч друкують слово «Додаток_» і арабську цифру або велику літеру, що позначає додаток.

1.5. Загальні вимоги до програмного продукту слухача МАН

Як було зазначено вище, особливість наукової роботи з інформатики полягає в тому, що в її основі має лежати або самостійно створена програмна розробка, або певним чином перероблений чи доопрацьований інший програмний продукт. Основними вимогами до програмного продукту слухача МАН є:

1. Програмний продукт має коректно працювати в операційній системі Windows, хоча можна використовувати альтернативні ОС (Unix, Linux, Free BSD тощо). Автор розробки може використовувати будь-яку доступну йому конфігурацію комп'ютера, задіяти всі його ресурси.

2. Програмна розробка повинна мати зручний користувацький інтерфейс. Загальноприйняті вимоги до інтерфейсу користувача:

Технологічність використання. Означає підтримку такого стилю діалогу, який би дозволяв працювати попередньо не ознайомленому з продуктом користувачу, і передбачає представлення в звичайному вигляді, а також широкі підказки про можливі дії користувача і реалізацію принципу «foolproof», тобто адекватну реакцію інформаційного продукту на довільні дії користувача. Іншими словами, інтерфейс повинен бути «дружнім» по відношенню до користувача.

Наприклад, реалізується принцип WYSIWYG «What You See Is What You Get» («Що бачите, те і маєте»).

Технологічність проектування. Визначається наявністю інструментальних засобів (інтерактивних або пакетних), які дозволяють проектувати інтерфейс користувача незалежно від програмної реалізації даного інформаційного продукту.

Прозорість інтерфейсу. Розглядається відносно операційної системи - користувач не повинен бачити повідомлення операційної системи і мати доступ на мові директив операційної системи. Іншими словами, інтерфейс користувача повинен підтримувати об'єктно-орієнтовану модель «світу користувача».

Наочність. Диктується бажанням створити користувачу той рівень комфорту, який відповідає його роботі за столом – для цього на екрані повинна відображатися траєкторія попередніх дій, поточний стан «світу користувача» і можливі варіанти дій.

Окремі вимоги визначаються для робіт у секції мультимедійні системи, навчальні та ігрові програми, а саме до навчальних програм. Серед яких:

1. Програма повинна мати методичні рекомендації щодо її використання для проведення викладачем різних типів занять та самостійної роботи користувача.

2. Навчальний матеріал має бути розподілений на розділи, модулі, що відповідають окремим темам навчальної програми. В межах модуля має бути забезпечена можливість розгляду основних теоретичних положень, застосування їх на практиці. Модулі мають бути замкненими, перехід до різних видів діяльності з певної теми має бути організований в межах модуля.

3. У структурі змісту кількість рівнів вкладеності має залежати від віку учнів, на яких розрахована програма. Для школярів молодшого шкільного віку бажано, щоб навігація була максимально спрощеною і структура змісту матеріалу містила не більше двох рівнів.

Варто врахувати, що через деякий час запропонована програмна розробка може виявитись дещо застарілою, тому розробник може створювати програми, які будуть мати демонстраційний або ж експериментальний характер. Але при цьому бажано зазначити шляхи доопрацювання програми, яким чином її можна удосконалити, які ще завдання вона зможе вирішити в майбутньому, які функції вона буде виконувати.

РОЗДІЛ 2. ПРАКТИКУМ З ОСНОВ ПРОГРАМУВАННЯ С++

2.1. Лінійні програми

Програма, що працює на комп'ютері, нерідко ототожнюється з самим комп'ютером, оскільки людина, що використовує програму, «вводить в комп'ютер» початкові дані з клавіатури або за допомогою мишки і "комп'ютер видає результат" на екран (так для прикладу працює програма «Калькулятор»). Насправді перетворення початкових даних, що вводяться з клавіатури, і результат, що виводиться на екран монітора, виконує процесор комп'ютера відповідно до послідовності команд деякої програми, яку попередньо написав і зберіг на комп’ютері програміст. Так, щоб комп'ютер виконав деяку роботу, необхідно розробити цю послідовність команд, або, як кажуть, написати програму. Вираз написати програму відображає тільки один з етапів створення комп'ютерної програми, коли розробник програми дійсно записує команди (інструкції) на папері або за допомогою текстового редактора.

Програмування — це процес створення (розробки) програми. Іншими словами, це процес написання тексту програми в деякому текстовому редакторі або на листку паперу. Текст програми пишеться на деякій мові програмування.

Програма – це опис обчислень.

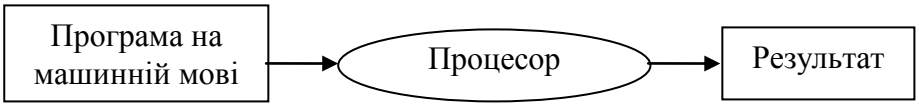
Обчислення – це дії, здійснення яких доручається деякому виконавцю, який повинен їх розуміти. Програма – це текст.

Мови програмування

Знакові системи, що використовуються для опису процесів обчислень, які виконуються на комп’ютері – мови програмування. Процес формування опису – програма. Розрізняють наступні мови програмування: машинні, асамблерні, мови високого рівня.

Машинні мови

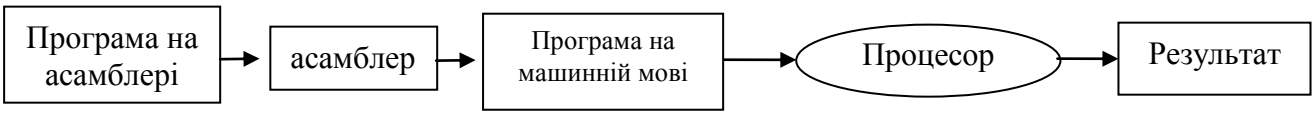
Знакова система, яка допускає безпосередньо виконувати процесором комп’ютера програм, що написані на цій мові. Одиниця програми в машинній мові – це машинна команда.



Команда складається з двох частин: операційна, адресна. В першій розміщається вказівка процесору, яку дію треба виконати, в другій – над чим треба виконати дану дію, тобто вказується значення. Вказівка значень визначається за допомогою адреси, тобто числа, що описує місце розташування значення в пам'яті комп'ютера. Для виконання переважної більшості операцій недостатньо однієї команди. Таму команди машинної мови об'єднуються в так звані речення.

Асамблерні мови

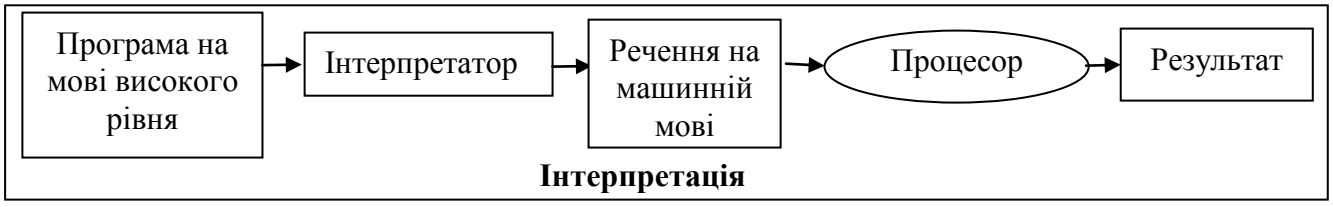
Це мова класу, вищого ніж машинна мова. Це також знакова система, ще дуже близька до машинної, але команди асамблерної мови не можуть безпосередньо виконуватися процесором, тому тексти цієї мови перекладають на машинну мову. Процес перетворення здійснюється за допомогою спеціальної програми асемблера і називають **асамблюванням.**



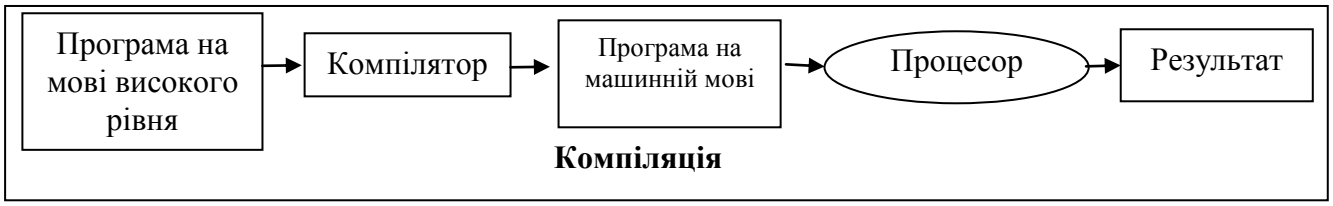
Мови високого рівня

Це знакова система, команди якої дуже близькі до природної мови. Для виконання процесором команд мови високого рівня повинні бути перетворені у машинну мову. Процес перетворення – трансляція і може відбуватися двома шляхами: інтерпретацією або компіляцією.

Інтерпретація – процес, в якому команди мови високого рівня перетворюються у речення машинної мови і виконуються процесором в міру їх утворення. Тому машинна програма в пам'яті цілком не запам'ятовується. Програма, яка здійснює перетворення називається інтерпретатором.



Компіляція – процес, в якому на мові високого рівня цілком перетворюється у програму машинної мови, а вже потім програма на машинній мові виконується процесором. Тому компільована машина програма на відміну від інтерпретованої запам'ятовується цілком. Програму, що виконує процес компіляції називають компілятором.



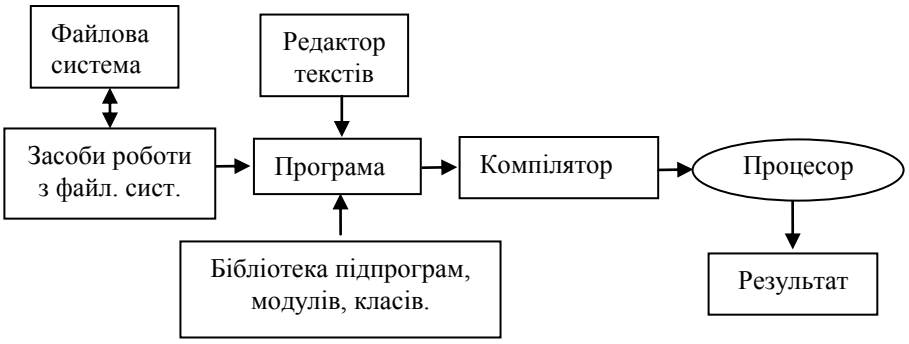
Компіляція

Першою мовою високого рівня, що стала концептуальною основою більшості сучасних мов була мова ALGO 58/60.

На сьогодні в комп'ютерному світі існує безліч мов програмування високого рівня. Найпопулярніші сьогодні – це Pascal, C, Ада, JAVA. Яка з мов краща? Відповідь на це питання не так проста. Однак можна зазначити, що C++ не «навчальна», не «іграшкова» мова, вона використовується для розробки складних «професійних» програм, у тому числі працюючих в середовищі Windows.

Середовище програмування

Середовище програмування - це системи програм, що забезпечують початкове кодування та відлагодження програми. Крім засобів трансляції програм (компілятора для більшості мов), вони містять у собі засоби для роботи з файловою системою та операційною системою (ОС). Такі середовища програмування називають інтегрованими. Інтегроване середовище програмування містить:



- редактор для створення програм;
- засоби трансляції програми у машинну

мову (компілятор або інтерпретатор) ;

- відладчик, що забезпечує різні режими компіляції (він допомагає програмісту відлагоджувати програму, знаходити та виправляти помилки);
- розвинуту бібліотеку підпрограм, модулів, класів, мегамодулів (готових програм, які може використати програміст у своїй програмі).

Директиви препроцесора

Перш ніж приступити до компіляції програми, компілятор C++ запускає спеціальну програму, яка називається препроцесором. Препроцесор шукає в програмі рядки, що починаються з символа #, наприклад #include або #define. Якщо препроцесор, наприклад, зустрічає директиву #include, він включає вказаний в ній файл у ваш початковий файл, нібито ви самі друкували вміст файлу, що включається, у вашому початковому коді.

Препроцесор – це програма, яка опрацьовує директиви. Директиви препроцесора – це команди компілятора відповідної мови програмування, які виконуються на початку компіляції програми.

Директиви мови C++ починаються із символа #.

Директива #include означає, що до програми необхідно приєднати

#include <назва файлу.розширення> або файлу.розширення”	**#include** ”шлях до файлу\назва
#include <math.h>	**#include** ”d:\stud\MyBib.h”

програмний код із зазначеного після неї файлу (header-файлу, модуля, бібліотеки). У першому випадку бібліотека math.h є стандартною (усі стандартні бібліотеки розміщені у папці INCLUDE середовища C++). Бібліотека MyBib.h не є стандартною і знаходиться за вказаним шляхом.

Перша програма

Розглянемо програму, у результаті виконання якої на екран буде виведено повідомлення: Привіт! Я C++! Ключове слово void можна не писати (замість Main(void) записати Main()). Коментар *// Моя перша програма* не впливає на роботу програми. Його можна записати по-іншому*: /* Моя перша програма*/* - це більш універсальний запис коментаря. Директива #include <iostream.h>

під’єднує бібліотечний файл iostream.h. Саме в цьому файлі описана команда - конструкція Cout <<. **Cout <<** забезпечує виведення на екран монітора повідомлення ”Привіт! Я С++!”. Функція Main(void) (або те ж саме Main()) може мати тип. Наприклад, Int Main() – означає, що функція повертатиме в точку виклику результат типу Int, тобто цілого типу. В нашому випадку в точку виклику функція повертає ціле значення 0 (return 0).

```
// Моя перша програма
#include <iostream.h>
Main(void)
{
  Cout <<”Привіт! Я
С++!”;
  Return 0;
```

```
// Моя перша програма
#include <iostream.h>
Int Main()
{
  Cout <<”Привіт! Я С++!”;
  Return 0;
}
```

Інший варіант цієї програми:

```
// Моя перша програма
#include <iostream.h>
Void Main()
{
  Cout <<”Привіт! Я
С++!”;
}
```

Така функція називається функцією типу Void. Вона не повертає в програму жодних значень. Тому команду Return писати не треба.

І ще один варіант нашої першої програми:

```
// Моя перша програма
#include <iostream.h>
#include <conio.h>
Void Main()
{
  Clrscr();
  Cout <<”Привіт! Я
С++!”;
  Getch();
}
```

#include <conio.h> - під’єднання бібліотеки, де знаходяться функції Clrscr() та Getch().

Clrscr() – очищення екрана.

Getch() – затримка на екрані результату виконання програми до тих пір, поки не буде натиснута довільна клавіша на клавіатурі.

2.1.1. Керуючі послідовності

Керуючі послідовності – це комбінація спеціальних символів, які використовуються для введення та виведення деяких даних. Керуюча послідовність складається із символу слеш «\» і спеціально означеного символу. Вони призначені для форматованого виведення на екран, наприклад, для переходу на новий рядок, подання звукового сигналу, а також для виведення на

екран деяких спеціальних символів: апострофа, лапок тощо. Основні керуючі послідовності наведені у таблиці.

Символи керуючих послідовностей	**Призначення**
\a , \7	Подати звуковий сигнал
\n	Перейти на новий рядок.
\t	Переведення курсора до наступної позиції табуляції
\\	Вивести символ \
\’	Вивести символ ‘
\”	Вивести символ ”
\?	Вивести символ ?

Керуючі послідовності разом з повідомленнями записуються у лапках після конструкції cout <<.

Розглянемо програму:

```
#include <iostream.h>
#include<conio.h>
Void main()
{
 Clrscr();
 Cout<<”Привіт! \n”;
 Cout<<”Я C++”;
 Getch();
}
```

або

```
#include <iostream.h>
#include<conio.h>
Void main()
{
 Clrscr();
 Cout<<”Привіт! \nЯ C++”;
 Getch();
}
```

Результат роботи програми:
Привіт!Я C++

Зауваження 1. Якщо використати послідовність, невизначену у мові C++ (наприклад, \u), то компілятор пропустить символ послідовності (похилу риску) і виведе на екран лише символ, записаний після риски (в нашому випадку літеру u).

Зауваження 2. Замість керуючої послідовності \n можна використати команду cout<<endl – кінець рядка. Попередню програму з використанням команди endl можна переписати у таких виглядах:

```
#include <iostream.h>
#include<conio.h>
Void main()
{
 Clrscr();
 Cout<<”Привіт! \n”;
 Cout<<endl;
 Cout<<”Я C++”;
 Getch();
}
```

```
#include <iostream.h>
#include<conio.h>
Void main()
{
 Clrscr();
 Cout<<”Привіт!
“<<endl;
 Cout<<”Я C++”;
 Getch();
}
```

```
#include <iostream.h>
#include<conio.h>
Void main()
{
 Clrscr();
 Cout<<”Привіт! “<<endl
<<”Я C++”;
 Getch();
}
```

Програма виведення на екран Вашої домашньої адреси

1. З використанням керуючої послідовності \n

```
#include<iostream.h>
#include<conio.h>
Void main()
{
  Clrscr();
  Cout<<”Моя домашня адреса. \n”;
  Cout<<”*******************\n”;
  Cout<<”        Волинська обл” \n;
  Cout<<”        Маневицький р-н\n”;
  Cout<<”        смт. Колки\n ”;
  Cout<<”        вул. Центральна, 5\n”;
  Getch();
}
```

1. З використанням команди cout<<endl

```
#include<iostream.h>
#include<conio.h>
Void main()
{
  Clrscr();
  Cout<<”Моя домашня адреса.
”<<endl;

Cout<<”*******************”<<endl;
  Cout<<”        Волинська обл” <<endl;
  Cout<<”        Маневицький р-н”
<<endl;
  Cout<<”        смт. Колки ” <<endl;
  Cout<<”        вул. Центральна, 5”
<<endl;
  Getch();
}
```

2.2. Найпростіші програмні об’єкти. Сталі та змінні

Розглянуті вище програми були простими. Однак у міру того, як ваші програми починають вирішувати більш складніші задачі, вони повинні зберігати інформацію під час виконання. Наприклад, програмі, що друкує файл, потрібно знати ім'я файлу і, можливо, число копій, які ви хочете надрукувати. В процесі виконання програми зберігають таку інформацію в пам'яті комп'ютера. Щоб використовувати певні елементи пам'яті, програми використовують **літерали, константи та змінні – це найпростіші програмні об’єкти.** Простіше кажучи, елементи пам'яті, в яких можна зберігати конкретне значення.

Загальні відомості про програмні об’єкти

Будь-який, навіть найпростіший, об’єкт має: позначення (назву, ім’я), значення, тип. Для об’єкта визначені ті чи інші операції, які можна застосувати до даного об’єкта.

Для кожного об’єкта виділяється ділянка в пам’яті комп’ютера. Тобто, перш ніж використовувати ті чи інші об’єкти в програмі, їх спочатку необхідно

створити або, інакше кажучи, описати. Пам'ять під об'єкт виділяється згідно з їх описом.

Сталі (константи)

Стала – це об'єкт для якого не можна застосувати операцію зміни його значення. Значення сталої протягом виконання усієї програми не змінюється.

```
Const <назва сталої 1> = <значення сталої 1>  або  Const <тип> <назва сталої 2> = <значення сталої 2>
```

Опис сталої

```
Приклад
Const vik = 20, rist = 156;
Const float g = 2.78
```

Якщо тип сталої не вказується, то вона вважається цілою (типу int)- для сталої 1. Стала 2 називається типізованою. Для неї вказується тип. Сталі vik та rist – не типізовані, які мають значення 20 та 156 відповідно. Стала g – типізована. Вказує на те, що це дійсна стала. Її значення 2.78. Зверніть увагу, що в одному блоці const можна описати кілька констант, перерахувавши їх через кому (vik = 20, rist = 156).

Розглянемо 2 варіанти програми, яка виводить дані про учня.

```
#include<iostream.h>
#include<conio.h>
const vik = 15, rist = 156;
void main()
{
 clrscr();
 cout<<"Мій вік =
"<<vik<<endl;
 cout<<"Мій зріст =
"<<rist<<endl;
 getch();
}
```

або

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 const vik = 15, rist = 156;
 cout<<"Мій вік =
"<<vik<<endl;
 cout<<"Мій зріст =
"<<rist<<endl;
 getch();
}
```

```
Результат роботи програми
Мій вік = 15
Мій зріст = 156
```

Зауваження 1. Результат роботи програми той же що і у випадку, коли сталі описані перед головною функцією, так і у випадку, коли сталі описані в тілі функції безпосередньо перед використанням.

Зауваження 2. Правила хорошого стилю програмування вимагають описувати усі об'єкти в межах тієї функції, де вони використовуються. В нашому випадку опис const vik = 15, rist = 156 правильніше робити в тілі головної функції (другий варіант програми).

Змінні

Змінні – це об’єкти, які можуть набувати а також змінювати свої значення під час виконання програми. Їх оголошують так:

```
<тип змінних 1> <список змінних 1>;
……..
<тип змінних N> <список змінних N >;
```

У списку змінні записуються через кому.

```
Int a,c;
float b, d = 2.5;
char w, q = ‘a’;
```

Наприклад, змінні оголошуються так: a та c – змінні цілого типу (типу Int); b, d – змінні дійсного типу (типу float). Змінній d надається деяке значення 2.5 відразу під час оголошення (або, іншими словами, змінна d ініціалізується під час оголошення значенням 2.5); w, q – змінні символьного типу (char). Змінна q ініціалізується значенням ‘a’ під час оголошення.

Приклад Програма знаходження суми та добутку двох чисел.

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 const a = 3, b = 2;
 int sum, prod;
 sum = a+b;
 prod = a*b;
 cout<<"3 + 2 = "<<sum<<endl;
 cout<<"3 * 2 = "<<prod<<endl;
 getch();
}
```

Результат роботи програми:

```
3 + 2 = 5
3 * 2 = 6
```

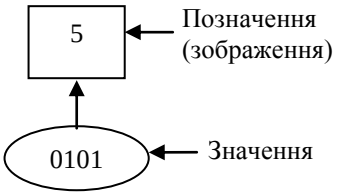
2.2.1. Найпростіші програмні об’єкти: літерал, константа, змінна

Загальні поняття.

Будь-який програмний об’єкт складається з двох частин: з позначення (зображення) та значення (вмісту). Позначення програмних об’єктів розташовані в тексті програми. Це зовнішні частини програмних об’єктів. Значення розташоване в пам'яті і входить до складу внутрішньої конструкції програмних об’єктів. Коли об’єкт створюється, тоді по позначенні будується значення

об’єкта. Коли об’єкт використовується, тоді по позначенні здійснюється доступ до значення об’єкта.

Літерал – найпростіший програмний об’єкт. Значення та позначення літералу збігаються.



Приклади літералів:

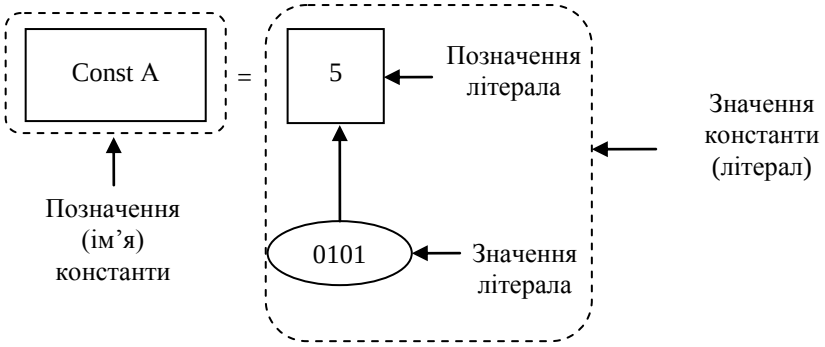
5 – його значення 0101 (тобто «п’ять» у двійковому представленні);

‘a’ – його значення 0001100001 (код в таблиці ASCII. 0001100001 в десятковому представленні 97). Використовують літерали в програмі без попереднього їх опису. Значення літералів відповідає його позначенню. Зверніть увагу, що символьний літерал береться в апострофи.

```
#include<iostream.h>
main()
{
   cout<<’a’;
}
```
на екрані буде символ a.

```
#include<iostream.h>
main()
{
   cout<<5;
}
```
на екрані буде число 5.

Константа – програмний об’єкт, який також складається з двох частин: позначення (ім’я) та значення. Значенням константи в свою чергу є літерал.



Використання констант в програмі доцільно по двох причинах:

- По-перше, коли літерал з однаковим значенням використовуються більше одного разу (це може бути наприклад число «пі»).

- По-друге, використовувані для констант позначення, що відповідають логічному змісту програми покращують розуміння програми.

Змінна – програмний об’єкт, значення якого не можна визначити по позначенню як у літералу, чи по опису, як у константи.

Конструкція змінна, як літерал та константа складається з позначення (імені) та значення (вмісту). Ім'я змінної в свою чергу складається з двох частин: позначення імені (ідентифікатор) та посилання . Посилання зберігає інформацію про те, де розташоване значення-вміст змінної (наприклад, 5). Крім адреси посилання містить і кількість пам’яті, що займає значення змінної. Змінна, як і константа має бути попередньо описана. Як бачимо зі схеми на відміну від літерала та константи для змінної зв’язок її позначення зі значенням не прямий. Позначення напряму зв’язане з посиланням, тобто адресою деякої комірки пам’яті, а що в цій комірці – позначення «не знає». Така конструкція дає можливість змінювати значення змінної. Адже пам’ять відведену під значення змінної можна витирати та заповнювати іншим значенням. При цьому ім’я змінної не змінюється.

2.2.2. Надання значень змінним. Введення значень з клавіатури. Найпростіші арифметичні операції

Ми вже знаємо, що у С++ немає вбудованих команд введення-виведення і для того, щоб вивести значення або деяке повідомлення необхідно попередньо підключити бібліотеку iostrem.h.

Пристрої введення-виведення.

Операції введення-виведення залежать від того:

- з яких пристроїв буде здійснюватися введення (клавіатура, файл, тощо);

- на які пристрої здійснюється виведення (монітор, файл, тощо).

Очевидно дії введення-виведення тісно пов’язані з функціонуванням цих пристроїв, зовнішніх по відношенню до ЕОМ. Устрій і робота цих пристроїв залежить від їх типу. Тому неможливо розробити універсальні команди для забезпечення введення-виведення на конкретні пристрої.

Поняття потоку

При організації введення-виведення використовують деякий проміжний пристрій – потік (stream). Якщо розглядати виведення, то спочатку дані виводяться в потік, а тоді вже цей потік даних перенаправляється на монітор. При передачі даних потоку набір даних розглядається як неперервна послідовність знаків та байтів.

За замовчуванням стандартними пристроями для потоків виведення даних є монітор, а для потоку введення – клавіатура. Стандартний потік для виведення даних – cout. Він використовує команду введення (>>).

Стандартний потік для введення даних – cin. Він використовує команду виведення (<<). Потоки cout та cin описані в модулі iostrem.h.

Команда введення даних.

Надавати значення змінним можна двома способами:

- за допомогою команди присвоєння (наприклад x = 3.1)
- або команди введення даних з клавіатури. Команда введення з клавіатури має такий загальний вигляд:

```
Cin>> <змінна>;
```

Якщо необхідно ввести значення відразу кількох змінним, то можна або використати декілька потоків введення, або записати усі змінні в одному потоці cin, застосувавши для цього декілька команд >>.

Наприклад. Знайти суму та добуток двох цілих чисел, що вводяться з клавіатури.

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 int a,b;
 int sum, prod;
 cin>>a;
 cin>>b;
 sum = a+b;
 prod = a*b;
 cout<<a<<" + "<<b<<" =
"<<sum<<endl;
 cout<<a<<" * "<<b<<" =
"<<prod<<endl;
 getch();
}
```

або

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 int a,b;
 int sum, prod;
 cin>>a>>b;
 sum = a+b;
 prod = a*b;
 cout<<a<<" + "<<b<<" =
"<<sum<<endl;
 cout<<a<<" * "<<b<<" =
"<<prod<<endl;
 getch();
}
```

Результат роботи програми:
5 7
5 + 7 = 12
5 * 7 = 35

Програми необхідно складати так, щоб ними могли користуватися не лише укладачі, але й інші особи. Тобто програми мають бути масовими та зрозумілими. Перед командою введення даних варто записувати команду виведення на екран текстового повідомлення-підказки про те, що саме слід ввести. Враховуючи сказане попередню програму змінимо наступним чином:

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 int a,b;
 int sum, prod;
 cout<<"a = ";
 cin>>a;
 cout<<"b = ";
 cin>>b;
 sum = a+b;
 prod = a*b;
 cout<<a<<" + "<<b<<" = "<<sum<<endl;
 cout<<a<<" * "<<b<<" = "<<prod<<endl;
 getch();
}
```

Результат роботи програми
a = 2
b = 3
2 + 3 = 5
2 * 3 = 6

Крім операцій додавання (+) та множення (*) допустимі і дві інші арифметичні операції:

- додавання (+);
- множення (*);
- віднімання (-);
- ділення (/).

2.3. Типи даних. Вказівка присвоювання

Як вже вказувалось дані що беруть участь у розв'язуванні задачі мають певний тип. Тип визначає:

- допустимі значення;
- операції, які можна виконувати над значеннями цього типу;
- обсяг пам’яті, яка резервується для нього.

Типи числових даних поділяють на цілі, дійсні та символьні.

Цілі типи

Змінні цілого типу описуються так: int <ім’я змінної>.

Допустимі значення: усі цілі з діапазона -32768 … +32767.

Обсяг пам’яті: 2 байти.

```
void main()
{
  int x;     // x отримує випадкове ціле значення
  x = 66;  // x отримує значення 66
}
```

```
void main()
{
  int x = 66;     // x отримує значення 66
}
```

Дійсні типи

Змінні дійсного типу описуються так: float <ім’я змінної>

Допустимі значення: усі цілі та дробові з діапазона $\pm 3.4 \cdot 10^{-38} \ldots \pm 3.4 \cdot 10^{38}$

Обсяг пам’яті: 4 байти

```
void main()
{
 float x;   // x отримує випадкове дійсне значення
 x = 2.5;  // x отримує значення 2.5
}
```

```
void main()
{
 float x = 2.5;  // x отримує значення 2.5
}
```

Символьний тип

Змінні символьного типу описуються так: char <ім’я змінної>

Допустимі значення: усі 255 символів кодової таблиці комп’ютера ASCII

Обсяг пам’яті: 1 байт.

Увага Змінним символьного типу можна надавати значень двома способами:

Безпосередньо (якщо x типу char, то після виконання команди x = ‘A’ x отримає значення A). Використовуючи код ASCII (після виконання команди x = 65 x отримає значення A, бо код символу A – 65).

```
void main()
{
  char x; // x отримує випадкове символьне значення
  x = 'A'; // x отримує значення – символ A
}
```

```
void main()
{
  char x = 'A'; // x отримує значення – символ A
  x = 65; // x отримує значення – символ B (її код в ASCII 65)
  }
```

Вказівка присвоєння. Правила узгодження типів.

Команда присвоєння призначена для надання змінним значень.

<Назва змінної>=<вираз> або <Назва змінної1>=<Назва змінної2>=…=<Назва

Команда присвоєння має такий загальний вигляд:

Дія команди. Обчислюється вираз і його значення надається змінній або кільком змінним. Вираз може містити числа, сталі, змінні, назви функцій, з’єднувані символами операцій.

Наприклад, a = 5, a = 8 – 3; c = d = a + 4; e = d/5 + c.

Змінна або вираз не обов’язково повинні бути одного типу. Крім того, у виразі можуть бути дані різних числових типів (змішані вирази). Якщо тип змінної не збігаються з типом виразу, то у C++ відбувається узгодження типів, яке буває двох типів: **явне** та **неявне**.

```
//Приклад неявне перетворення типів
void main()
{
  int a = 2; float c = 3.8;
  int b;
  b = a * c;     //2*3.8 = 7.6, b = 7
}
```

```
//Приклад  явне перетворення типів
void main()
{
  int a = 2; float c = 3.8;
  int b;
  b = a * int(c);  // 2 * 3 = 6, b = 6
}
```

Пріоритет арифметичних операцій

Пріоритет	Операції	Зміст операції	Приклад	Коментар
Найвищий	+ –	Присвоєння знака.	2 * –5 = – 10	Спочатку 5 змінює знак , а тоді множиться на 2
Середній	* / %	Множення, ділення, остача від ділення.		
Найнижчий	+ –	Додавання, віднімання.	4 + 2* –5 = –6	Додавання виконується останньою

Для зміни пріоритету, як і в математиці використовуються круглі дужки.

Наприклад

2 * –5 = –6	7 % 3 = 1	12 / (4 – 2) = 6
2 * (–5 + 4) = –2	12 / 4 – 2 = 1	7 % 3 * –5 = –5

Операції інкременту (++) та декременту (– –)

Операція інкременту збільшує значення змінної на 1, а декременту – зменшує на 1.

```
void main()
{
 int a;
 a = 5; // a = 5
 a++;  // a = 6
}
```

```
void main()
{
 int a;
 a = 5; // a = 5
 ++a;  // a = 6
}
```

```
void main()
{
 int a;
 a = 5;        // a = 5
 a = a + 1;  // a = 6
}
```

В усіх випадках після виконання команди a++ (++a чи a = a + 1) значення змінної a збільшується на 1. Якщо команду a++ (++a чи a = a +1) замінити командою a– – (– –a чи a = a – 1), то значення змінної a зменшиться на 1. Як бачимо, операція інкременту аналогічна команді a = a + 1, а декременту – команді a = a – 1.

На перший погляд різниці між тим, де записано оператор інкременту (декременту), перед чи після змінної, немає. Але це не так для тих випадків, коли дана операції присутня у виразах.

Операції інкременту і декременту існують у двох формах:

- префіксній – якщо символи ++ (– –) записані перед змінною;
- постфіксній – якщо символи ++ (– –) записані після змінної.

```
void main()
{
 int a,b ;        //a = 836   b = -28724
 a = 5;           // a = 5     b = -28724
 b = 3*++a;   // a = 6      b = 18
}
```

```
void main()
{
 int a,b ;        //a = 836   b = -28724
 a = 5;           // a = 5     b = -28724
 b = 3*a++;   // a = 6      b = 15
}
```

Якщо змінна, над якою виконується операція інкременту чи декременту присутня у деякому виразі, то:

- для префіксної форми спочатку виконується операція інкременту (декременту), а потім обчислюється значення виразу;

- для постфіксної форми спочатку обчислюється значення виразу, а потім виконується операція інкременту (декременту).

Або іншими словами:

- у префіксній формі операція інкременту (декременту) має найвищий пріоритет за усі інші операції, присутні у виразі;

- у постфіксній формі операція інкременту (декременту) має найнижчий пріоритет за усі інші операції, присутні у виразі;

Операція присвоєння, суміщена з арифметичною операцією

Використовується для зміни значення деякої змінної:

- збільшити на +=;
- зменшити в /=;
- зменшити на –=;
- знайти остачу від ділення %=.
- збільшити в *=;

Наприклад, ці команди ідентичні:

1) a = a + 10 та a +=10 2) b = 4 * b та b *=4

3) c = c % 5 та c %= 5

Зауваження! Якщо справа від операції присвоєння суміщеної з арифметичною операцією стоїть деякий вираз, то спочатку обчислюється значення цього виразу, а тоді виконується суміщена операція присвоєння. Для зміни пріоритету можна використати дужки. Наприклад:

```
void main()
{
 int a,b;
 a = 5;
 b = a *= 2 + 1; // a = 15   b = 15
}
```

```
void main()
{
 int a,b;
 a = 5;
 b = (a *= 2) + 1; // a = 10   b = 11
}
```

2.3.1. Математичні функції

Усі стандартні математичні функції у C++ описані в бібіліотеці <math.h>.

Назва ф-ції	Матем запис	Назва ф-ції	Матем запис	Назва ф-ції	Матем запис
abs(x)	|x|	exp(x)	e^x	ceil(x)	заокруглює число x до більшого цілого. Наприклад, ceil(5.7) = 6
cos(x)	cos x	pow10(x)	10^x		
sin(x)	sin x	log10(x)	lg x	floor(x)	Відкидає дробову частину числа x. Наприклад, floor(5.7) = 5
tan(x)	tg x	fabs(x)	|x|		

log(x)	ln x	acos(x)	arccos x	fmod(x,y)	Обчислює остачу від ділення числа x на число y (аналогічна операції %). Наприклад, fmod(7,3) = 1
pow(x,y)	x^y	asin(x)	arcsin x		
aqrt(x)	$\sqrt{x}$	atan(x)	arctg x		

Приклад. Програма обчислення значення функції $y = \sqrt[5]{x^2 + 7.2} - |x - 5| + \sin\frac{\pi x}{3}$ для x = 2

```
#include<iostream.h>
#include<math.h>
#include<conio.h>
void main()
{
 clrscr();
 const float pi = 3.1415926;
 float x = 2,y;
 y = pow(x*x + 7.2, 1.0/5) - fabs(x - 5) + sin(pi *
x/3);
 cout<<"y = "<<y<<endl;
 getch();
```

При піднесені до степені 1/5 ми записали 1.0/5. Справа в тому, що компілятор C++ при обробці цілих чисел результат може транслювати як ціле число. Тобто 1/5 = 0 Якщо арифметичні операції виконуються над дійсними числами, то результат буде дійсним. Тобто 1.0/5 = 0.2.

2.4. Адреси даних. Вказівники.

Операція визначення адреси &

Змінна – це ділянка пам’яті. Вона має адресу. Щоб визначити адресу змінної використовується оператор визначення адреси &. Наприклад &a – визначення адреси змінної a. Адреси комірок пам’яті записується парою чисел (адреса сегмента та зміщення), представлених у 16-вій системі числення. Наприклад, 8fc0: 0f4e – це два числа 36800 : 3918.У шістнадцятковій системі числення для представлення чисел крім цифр 0..9 використовуються літери латинського алфавіту a, b, c, d, e, f.

Приклад
```
void main()
{
 int a = 5;  // a : 5,  &a : 9005:0FFE
a = 10;      // a : 10,  &a : 9005:0FFE
}
```

Змінюючи значення змінної a (спочатку 5 а потім 10), її адреса не міняється (&a в обох випадках рівне 9005:0FFE).

Вказівники.

Як ви вже знаєте, програми на C++ зберігають змінні в пам'яті. Необхідно відразу усвідомити, що покажчик (вказівник) є ділянкою пам’яті, в якій зберігається адреса іншої ділянка пам'яті. Говорять, що вказівник вказує (або

посилається) на певну ділянку пам’яті. Операції з покажчиками широко використовуються в С++.

```
void main()
{
 int a;
 a = 5;
 int *b;    //a: 5  &a: 9004:0FFE   b: 000B:8FCF
 b = &a;  //a: 5  &a: 9004:0FFE   b: 9004:0FFE
}
```

Вказівники – це змінні, значенням яких є не саме дане, а деяка адреса іншої комірки пам’яті, де може зберігатися (чи зберігається) деяке дане. Змінна-вказівник описується як звичайна змінна з тією відмінністю, що перед іменем вказівника вказується *. Змінній-вказівнику можна надавати лише значень адрес (наприклад адреса інших змінних: b = &a . Тут b – змінна-вказівник, a – звичайна (статична) змінна). Після виконання оператор b = &a змінна b отримала значення адреси змінної a, тобто 9004:0FFE.

Зміна значень статичних змінних через вказівну змінну

```
void main()
{
 int a;
 a = 5;
 int *b;      //a: 5     &a: 9004:0FFE   b : 000B:8FCF
 b = &a;     //a: 5     &a: 9004:0FFE   b: 9004:0FFE
 *b = 10;    //a: 10   &a: 9004:0FFE   b: 9004:0FFE
}
```

Якщо змінна b вказівна, то змінити значення комірки пам'яті, на яку вказує ця змінна можна за допомогою вказівки присвоєння: *b =<деяке значення>, де b – вказівник, *b - вміст (значення) змінної на яку вказує вказівник.

З наведених прикладів можна зробити висновок, що надати значення деякій статичній змінній можна двома способами:

- використовуючи її ім’я (a = 5);
- через вказівник, значенням якого є адреса цієї змінної (b = &a; *b = 10).

Переадресація вказівників

Відразу після створення вказівника b (int *b), він отримує значення випадкової адреси (b = 000B:8FCF). Варто пам’ятати, що комірка пам’яті з адресою 000B:8FCF – випадкова, тобто в цій комірці може зберігатися значення деякої іншої змінної (неможливо сказати якої, бо адреса її вибиралася випадково). Тому, якщо відразу після створення вказівника b (int *b) виконати оператор *b =

10, то таким чином можна змінити значення деякої змінної (визначити якої неможливо). Тому після створення вказівника необхідно:

- або переадресувати її на потрібну статичну змінну (як у наведених прикладах b = &a – переадресація вказівника b на статичну змінну a);

- або створити так звану динамічну змінну, адрес якої буде міститися у вказівнику).

Динамічна пам'ять (купа). Команди new та delete

Динамічні змінні, як і статичні можуть містити значення відповідного типу. Над ними можна виконувати такі ж операції, як і над статичними змінними того ж типу. Різниця між динамічними змінними та статичними:

1. Динамічна змінна немає імені, а статична моє.

2. До статичної змінної можна звертатися як через її ім'я, так і через деякий вказівник, адресований на цю змінну;

3. До динамічної змінної можна звертатися лише через деякий вказівник, адресований на неї.

3. Динамічні змінні знаходяться в так званій динамічній пам'яті (в «купі» або в heap). А статична – в статичній.

4. Статична змінна створюється один раз (при описі) і існує до кінці роботи програми, займаючи часто дорогоцінну пам'ять. Динамічну змінну можна створювати і видаляти (звільняти пам'ять) в довільній точці програми.

Щоб створити динамічну змінну (зарезервувати під нею пам'ять в купі) необхідно застосувати команду **new** до деякого вказівника. Щоб знищити динамічну змінну (вивільнити пам'ять) необхідно застосувати команду **delete** до вказівника, що направлений на цю змінну. Сам вказівник після цього необхідно занулити (присвоїти значення 0 (NULL)) **Наприклад**

```
void main()
{
 int *b;      //Створюємо вказівник b                                                    b = main          *b = 1224
*b = 2;       // Така вказівка небезпечна, так як пам'ять під *b не виділено          b = main          *b = 2
 b = new;  //Створюємо динамічну змінну в купі. Вказівник b містить її адресу     b = 9131:0004    *b = 27966
*b = 5;      // Змінюємо значення динамічної змінної                                  b = 9131:0004    *b = 5
 delete b;  //Знищуємо динамічну змінну, на яку вказує b (вивільняємо динамічну пам'ять)
 b = 0;       //Занулюємо вказівник b                                                  b = NULL          *b =4200
}
```

Зверніть увагу, що при створенні вказівника b (int *b) він вказує на деяку випадкову змінну (зі значенням 1224). Після занулення його, він теж вказує на деяку іншу випадкову змінну (зі значенням 4200).

Нетипізовані вказівні змінні.

Розглядаючи динамічні змінні та змінні-вказівники, ми описували вказівник на деяку динамічну змінну наприклад цілого типу так: int *a, де a – вказівник на деяку динамічну змінну цілого типу. Значенням змінної-вказівника є адреса динамічної змінної, цілого типу. Але, оскільки адреси як змінних цілого типу, так і змінних інших типів мають один і той же формат (складаються з пари чисел, представлених у 16-вій системі), то очевидно одна і та ж вказівна змінна може містити як адресу динамічної змінної цілого типу, так і адресу динамічної змінної довільного іншого типу. Такі змінні-вказівники називаються **нетипізованими**.
При описі нетипізованої вказівної змінної замість типу вказується службове слово void. Наприклад, запис **void *a** означає, що вказівник a може містити адресу деякої динамічної змінної довільного типу.

2.5. Етапи розв’язання задач на комп’ютері

Перш ніж одержати очікуваний результат роботи програми на комп'ютері, необхідно виконати досить багато клопіткої підготовчої роботи.

1. Етап постановки задачі.

Розв'язування будь-якої задачі починається з її постановки, викладеної мовою чітко визначених математичних понять. На першому кроці необхідно добре уявити, в чому саме полягає дана задача, які необхідні початкові дані, яку інформацію вважати результатами розв'язання.

2. Етап побудови математичної моделі.

Не завжди умова сформульованої задачі містить в собі готову математичну формулу, яку можна застосувати для розв’язання задачі, не завжди розв'язок задачі вдається одержати в явному вигляді, що зв'язує вхідні дані та результат. Для цього створюється інформаційна математична модель об'єкта, що вивчається. Наприклад, розв'язуючи задачу про рух тіла під дією прикладених до нього сил,

ми перш за все записуємо рівняння його руху на основі законів механіки. Проте, крім сили тяжіння, на тіло діє і сила опору повітря. Постає питання достовірності математичної моделі і реальної картини досліджуваного об'єкта. Іноді буває неможливо врахувати всі реальні фактори, що впливають на нього. Тому дуже важливим є вміння виділити серед усіх факторів головні і другорядні, щоб останніми можна було знехтувати. При цьому може скластися ситуація, коли наперед невідомо, якими саме факторами можна знехтувати, і тому може бути кілька математичних моделей, які описують один і той самий об'єкт з різним ступенем достовірності.

3. Етап побудови алгоритму.

Наступним етапом є розробка алгоритму обробки інформації на основі побудованої математичної моделі. **Алгоритм** — точна послідовність команд, що визначає процес переходу від початкових даних до результату, записана зрозумілою людині мовою. Зазначимо, що програма – це також точна послідовність команд, що визначає процес переходу від початкових даних до результату, але записана зрозумілою комп’ютеру мовою.

Послідовність команд вважається алгоритмом, якщо вона володіє трьома наступними властивостями:

- визначеністю, тобто точністю, що не залишає місце для свавілля;
- універсальністю (масовістю), тобто можливістю використання алгоритму для різних значень початкових даних;
- результативністю, тобто спрямованістю на отримання результату.

Приклад — алгоритм ділення звичайних дробів.

Початкові дані: перший дріб (ділене), другий дріб (дільник).

Шуканий результат: дріб.

Алгоритм:

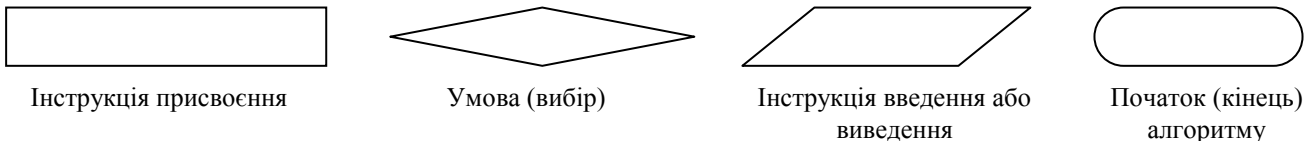
а) чисельник першого дробу помножити на знаменник другого;

б) знаменник першого дробу помножити на чисельник другого;

в) записати дріб, чисельник якого є результат виконання пункту а), знаменник —

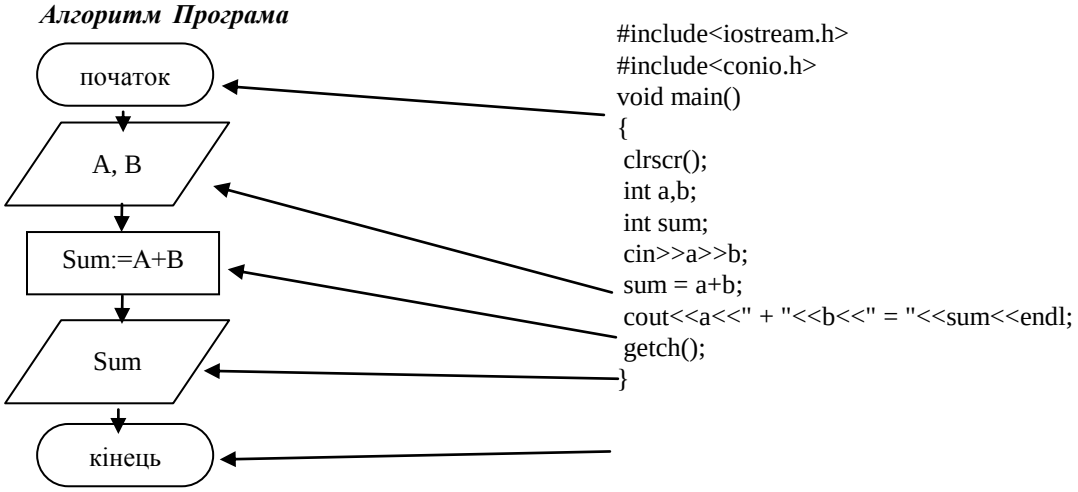
результат виконання пункту б).

Алгоритм рішення задачі може бути уявлений у вигляді словесного опису або графічно — у вигляді блок-схеми. Основні елементи блок-схеми:



Інструкція присвоєння | Умова (вибір) | Інструкція введення або виведення | Початок (кінець) алгоритму

Розглянемо програму та блок-схему алгоритму знаходження суми двох чисел:

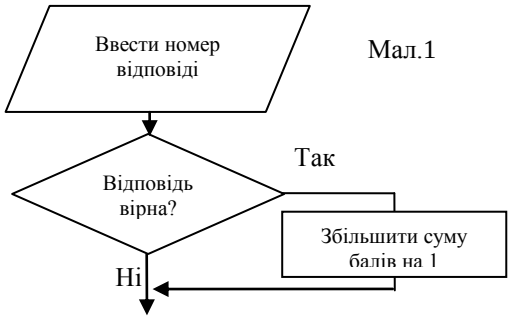


Метод покрокової розробки алгоритму. Під час створення складних алгоритмів застосовується метод покрокової розробки. Сутність цього методу полягає в тому, що алгоритм розробляється «зверху донизу». На кожному етапі приймається невелика кількість рішень, що призводить до поступової деталізації, уточнення як виконуваної, так і інформаційної структури алгоритму. Такий підхід дозволяє розбити алгоритм на окремі частини, кожна з яких розв'язує свою самостійну підзадачу. Це дає можливість сконцентрувати зусилля на розв'язуванні кожної підзадачі, що реалізується у вигляді окремої процедури.

4. Етап складання програми.

Цей етап потребує лише знання вибраної мови програмування. Суть його полягає в тому, щоб на основі розроблених алгоритмів і структур даних створити програму для комп'ютера.

2.6. Розгалуження



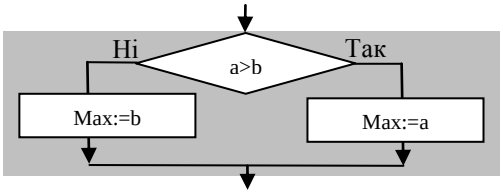
Мал.1

Алгоритми розв'язання більшості задач не є послідовними, коли усі команди програми чітко слідують одна за одною. Дії (обчислення), які необхідно виконати, можуть залежати від певної умови, наприклад, вхідних даних, або результатів, отриманих під час виконання програми. Наприклад, в програмі перевірки знань оцінка за вибрану з декількох варіантів відповідь, що додається до загальної суми балів, залежить від того, чи є відповідь правильною. Фрагмент блок-схеми алгоритму рішення цієї задачі подано на мал. 1.

Розглянемо найпростішу програму – знаходження більшого з двох чисел:

Приклад 1

```
#include
<iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int a,b, max;;
 cout <<"a = "; cin>>a;
 cout <<"b = "; cin>>b;
 if (a>b) max = a;
 else max = b;
 cout<<"Максимальне =
"<<max;
 getch();
}
```



Результат роботи програми:
a = 3
b = 6
Максимальне = 6

Загальний вигляд повного оператора розгалуження:

if (<умова>) **P1 else P2;**

де P1 та P2 — деякі команди.

Робота оператора розгалуження не викликає ніяких труднощів. Цей оператор в залежності від істинності або хибності умови (if a>b) вибирає той чи інший шлях наступного виконання алгоритму — виконання команди P1 (істинність умови a>b – max = a) або команди P2 (хибність умови a>b – else max = b). Після цього робота алгоритму продовжується далі за вказаними вказівками.

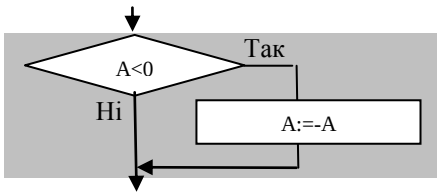
Ми розглянули роботу повного оператора розгалуження. Існує ще і скорочена

форма оператора розгалуження.

Приклад 2: Розглянемо програму знаходження модуля дійсного числа

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int a;
 cout <<"a = ";
cin>>a;
 if (a<0) a = -a;

cout<<"Абсулютн
а величина введеного числа = "<<a;
 getch();
}
```



Результат роботи програми
a = -2
Абсулютна величина введеного числа = 2

Загальний вигляд скороченого оператора розгалуження: **if** (<умова>) P;

Відмінність між двома формами розгалуженого оператора:

- в повній — незалежно від істинності чи хибності умови якісь дії обов'язково будуть виконані, а вже потім продовжено виконання алгоритму далі,
- у скороченій — у випадку, коли умова набуде істинного значення**,** будуть виконані якісь дії, а потім продовжено виконання алгоритму, а у випадку хибності умови**,** виконання алгоритму зразу ж буде продовжено далі.

Складена команда.

Під час написання програми може виникнути потреба трактувати декілька команд як одну. Така команда називається складеною. Складена команда – це конструкція такого вигляду:

```
{
  <Команда 1>;
  <Команда 2>;
   …….;

  <Команда N>;
}
```

Перед закриваючою фігурною дужкою «;» ставити обов'язково. Після закриваючої дужки «;» ставити не обов’язково.

Зауваження Запис ;; називається порожньою командою. Якщо у складеній команді поставити символ «;» після закриваючої дужки, то компілятор це розглядатиме як порожню команду, що не впливає на результат виконання

програми.

Приклад 3. Розглянемо ще один варіант алгоритму пошуку найбільшого з двох заданих чисел A та B.

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int a,b;
 cout<<"a = "; cin>>a;
 cout<<"b = "; cin>>b;
 int max;
 if (a>b)
   {
      cout<<"Перше число більше"<<endl;
      max = a;
   }
 else
   {
      cout<<"Друге число більше"<<endl;
      max = b;
    }
 cout<<"Воно = "<<max<<endl;
 getch();
}
```

Результат роботи програми

```
a = 23
b = -87
Перше число більше
Воно = 23
```

2.7. Логічні вирази та логічні оператори

Умову ще називають логічним виразом.

Логічний вираз, як і арифметичний, може набувати значень. Але, на відміну від арифметичного виразу, лише два значення:

- істина (true);
- та хиба (false).

Яким би не був логічний вираз, він завжди набуває одне з цих двох значень.

Часто **true** інтерпретують як 1, **false** – як 0.

Найпростіший логічний вираз складається з двох значень (або змінних) між якими стоїть оператор порівняння.

Оператор порівняння	Приклад (на мові С++)	Зауваження	Математ. еквівалент	Приклад (в математиці)
==	5 == 5	Знаком = позначається оператор присвоєння	=	5 = 5
!=	5 != 7	На клавіатурі немає символу ≠	≠	5 ≠ 7
<	5 < 7	–	<	5 < 7
>	9 > 7	–	>	9 > 7
<=	7 <= 7	На клавіатурі немає символу ≤	≤	7 ≤ 7
>=	7 >= 7	На клавіатурі немає символу ≥	≥	7 ≥ 7

Складені логічні вирази.

Складеними логічними виразами називають один або декілька простих логічних виразів на які діють так звані логічні оператори.

Логічні оператори «НЕ», «І», «АБО»

Ми будемо розглядати три логічні оператори:

1. ! (логічний оператор «Не» або логічне заперечення);
2. && (логічний оператор «І» або логічне множення);
3. || (логічний оператор «АБО» або логічне додавання).

Приклад складеного логічного виразу:

(5 > 3) || (5 == 3)

Він складається з двох простих логічних виразів 5 > 3 та 5 == 3 між якими стоїть логічний оператор «АБО»

Дія логічних операторів

A	B	A && B	A || B	! A
False (0)	False (0)	False (0)	False (0)	True (1)
False (0)	True (1)	False (0)	True (1)	True (1)
True (1)	False (0)	False (0)	True (1)	False (0)
rue (1)	True (1)	True (1)	True (1)	False (0)

Тут А та В – прості логічні вирази.

Логічна операція **&&** дає результат true тоді і тільки тоді, коли обидва прості логічні вирази мають значення true.

Логічна операція || дає результат true тоді і тільки тоді, коли хоча б один простий логічний вираз має значення true.

Логічна операція **!** завжди дає результат, протилежний значенню простого логічного виразу

2.8. Цикли

При розв'язанні багатьох задач деяку послідовність дій доводиться виконувати кілька разів. Наприклад, програма контролю знань виводить питання, приймає відповідь, додає оцінку за відповідь до суми балів і повторює цю дію ще раз, і ще

до тих пір, поки не будуть вичерпані всі питання. Інший приклад. Щоб знайти прізвище людини в списку, треба перевірити перше прізвище списку, потім друге, третє і т.д. до тих пір, поки не буде знайдена потрібна або не буде досягнутий кінець списку. Такі дії, що повторюються, називаються циклами і реалізуються в програмі з використанням вказівок циклів.

В мові С++ циклічні обчислення реалізуються за допомогою інструкцій **FOR, WHILE** і **DO-WHILE**

Команда **FOR** має вигляд: **for** (<вираз1>; <логічний вираз2>; <вираз3>) <команда1>;

Вираз1 призначений для підготовки циклу і виконується один раз. Переважно у виразі1 задаються початкові значення змінних циклу (підготовлюють цикл). У логічному виразі2 записують умову виходу із циклу. У виразі3 – команди зміни параметрів циклу. Якщо за допомогою одного із виразів необхідно виконати декілька дій, то використовують команду «кома». Вирази 1 і 3 або один із них у команді for можуть бути відсутні. Але символ «;» в цьому випадку опускати неможна. <Команду1> ще називають тілом циклу.

Приклад1. Розглянемо декілька варіантів обчислення суми цілих чисел з проміжку від 1 до 15.

```
void main()
{
 int n = 1, sum = 0;
 for (;n<=15;n++) sum+=n;
}
```

```
void main()
{
 for (int n = 1,sum = 0;n<=15;n++) sum+=n;
}
```

```
void main()
{
 for (int n = 1,sum =
0;n<=15;sum+=n++);
}
```

В результаті виконання усіх варіантів програми n = 16 sum = 120.

Дія команди.

1. Обчислюється значення виразів 1 та 2.
2. Якщо значення виразу 2 істинне виконується тіло циклу, а потім знаходиться значення виразу 3.
3. Потім повторно визначається значення виразу 2 і пункт 2 повторюється до тих пір, поки значення виразу 2 істинне. Як тільки значення виразу 2 стане хибне, відбувається вихід з тіла циклу.

Приклад2. Кілька варіантів знаходження кількісті та добутку лише парних чисел з діапазону 4..11

```
void main()       //варіант1
{
 for (int n=4,prod=1,k=0;n<=11;prod*=n,n+=2,k++);
}
```

```
void main()     //варіант2
{
 for (int n=4,prod=1,k=0;n<=11;n+=2) (prod*=n,k++);
}
```

```
void main()       //варіант3
{
 for (int n=4,prod=1,k=0;n<=11;n+=2)
   {
    prod*=n;    k++;
   }
}
```

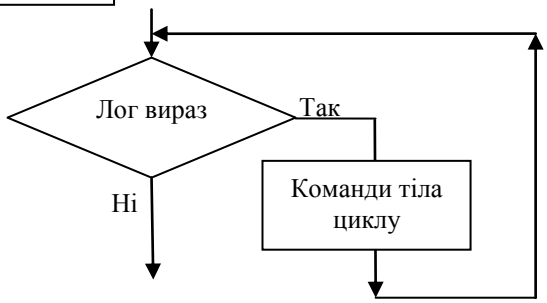
У першому варіанті у виразі1 та у виразі3 використовується кома як команда. Тіло циклу взагалі відсутнє.У другому варіанті тіло складається з двох команд, яка розглядається як одна (використовується кома як команда). У третьому варіанті тілом циклу є складена команда.

Цикл WHILE. Наближені обчислення.

Цикл WHILE має вигляд: While (<логічний вираз>)<команда1>;

команда1 – тіло циклу

Дія циклу While:



1) Спочатку обчислюється значення логічного виразу
2) Якщо його значення істинне, то виконується тіло циклу і повторно обчислюється значення логічного виразу

   Якщо значення логічного виразу хибне, то здійснюється вихід з циклу.
3) Процес виконується доти, доки логічний вираз істинний.
4) **Приклад 1. Знайти суму цифр числа 1234**

```
void main()//варіант 1
{
 int a =1234,s=0;
 while (a>0)
   {
    s+=a%10;
    a/=10;
   }
}
```

Деяке ціле a = 1234.

Операція a%=10 дасть результат 4. Після виконання команди a/=10 змінна a отримає ціле значення 123. Повторне виконання операції

```
void main() //варіант 2
{
 int a =1234,s=0;
 while (a>0,s+=a%10,a/=10);
}
```

a%=10 дасть результат 3, а команди a/=10 – результат a = 12. Очевидно описаний процес слід повторювати до тих пір поки a>0. Результати роботи програми в обох варіантах ідентичні: s = 10, a = 0.

В другому варіанті тіло циклу відсутнє. Зате в якості логічного виразу виступає три вирази розділені комою (кома як команда інтерпретує ці вирази як єдиний).

Приклад2. Визначення обсягу вільного місця в динамічній пам'яті оперативної пам'яті в даний момент часу.

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();  int n=0;
 while (new char) n++;
 cout<<"Вільноі пам'яті - "<<n<<endl;
 getch();
}
```

Результат роботи програми:
Вільноі пам'яті - 3780

Логічний вираз (new char) має істине значення якщо вдається виділити під динамічну змінну типу char місце в динамічній пам'яті (в купі). Як тільки вільного місця в динамічній пам'яті немає вираз (new char) отримує хибне значення. Тілом циклу є звичайний лічильник, оскільки змінна типу char займає один байт.

Типовими прикладами використання циклу WHILE є обчислення із заданою точністю.

Приклад 3. Нехай x деяке число, яке необхідно ввести з клавіатури. З клавіатури вводиться також точність e (дуже мале число). Обчислити суму елементів:

$sum = 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots$ з точністю e.

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr();
 float sum=0,e;
 long int fact=1; int n = 1;
 cout<<"Похибка e = "; cin>>e;
 while (1.0/fact>=e)
   {
     sum+=1.0/fact;
     fact*=++n;
   }
 cout<<"Summa = "<<sum<<endl;
 cout<<"Просумовано елементів - "<<n<<endl;
 getch();
}
```

N! = 1*2*3*…*N.

Наприклад 5! = 1*2*3*4*5 =120, а 1/5!<0.01

Тобто 5-й елемент 1/5! уточнює значення суми величини, яка менша 0.01

```
Похибка e = 0.00001
Summa = 1.718279
Просумовано елементів – 9
```

Для введеної похибки e = 0.00001 знаходиться значення Summa з точністю до 4-го знака після коми Summa = 1.7182

```
Похибка e = 0.00000000001
Summa = 1.718282
Просумовано елементів - 17
```

Зауваження! В умові циклу замість 1/fact>e ми записали 1.0/fact>e, так як компілятор C++ значення виразу 1/fact буде трактувати як ціле (тобто рівне нулю для всіх fact більших 1).

Приклад 4. Знайти значення числа π із заданою точністю e.

Обчислення значення π засновано на тому, що сума ряду 1 – 1/3 + 1/5 – 1/7 + 1/9 + ... наближається до значення $\pi/4$ при достатньо великій кількості членів ряду.

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr(); float e;
 cout<<"Введіть точність e = ";
cin>>e;
 float pi = 0;  int n = 1,k = 1;
 while (1.0/n>e/4)
     {
       pi+=k*1.0/n;n+=2;k*=-1;
     }
```

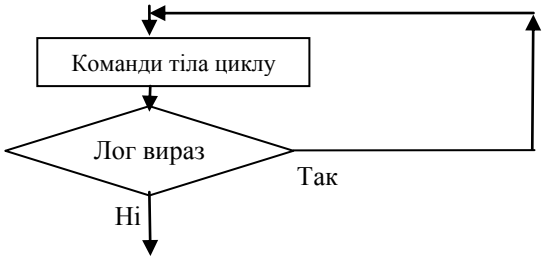
Результат роботи програми:
Введіть точність e = 0.0001
3.141538

Команда циклу з післяумовою DO-WHILE

Команда **DO-WHILE**, як і команда **WHILE**, використовується в програмі, якщо треба провести деякі обчислення (цикл), що повторюються, і як для WHILE число повторів наперед не відоме і визначається самим ходом обчислення.

В загальному вигляді команда циклу з післяумовою DO-WHILE має вигляд:

```
Do
  {
    <Команда 1>
  }
  while
(<логічний
вираз>);
```



Приклад 1: Знайти суму чисел, що вводяться з клавіатури.

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();  float a,sum;
 do
   {
    cout<<"Число (для завершення 0) = "; cin>>a;
    sum+=a;
   }
 while (a!=0);
 cout<<"Сума введених чисел = "<<sum<<endl;
 getch();
}
```

Результат роботи програми:
Число (для завершення 0) = 2
Число (для завершення 0) = 3
Число (для завершення 0) = 4
Число (для завершення 0) = 1
Число (для завершення 0) = 0
Сума введених чисел = 10

Команда виконується таким чином:

1. Виконуються команди наступні за словом **Do**.
2. Обчислюється значення логічного виразу . Якщо його значення істинне (значення рівне TRUE), то повторно виконується команда циклу. Якщо ж значення рівне FALSE, то виконання циклу припиняється. Таким чином, команди, що знаходяться між Do і While виконуються, до тих пір, поки логічний вираз має істинне значення.

Приклад 2. Розкласти на прості множники число введене з клавіатури.

Варіатн 1

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int n,i=2;
 cout<<"Ціле число = ";
 cin>>n;
 cout<<n<<" = 1";
 do  {for (;n%i==0;n/=i) cout<<" * "<<i;
       i++; }
 while (i<=n);
 cout<<endl;
 getch();
}
```

Варіант 2

```
#include <iostream.h>
#include <conio.h>
void main()
{
 clrscr();
 int n,i=2;
 cout<<"Ціле число = ";
 cin>>n;
 cout<<n<<" = 1";
 do  {while (n%i==0) {cout<<" * "<<i;
                      n/=i;}
       i++;
     }
 while (i<=n);
 cout<<endl;
 getch();
}
```

Результат роботи програми:
Ціле число = 30030
30030 = 1 * 2 * 3 * 5 * 7 * 11 * 13

Або інший варіант

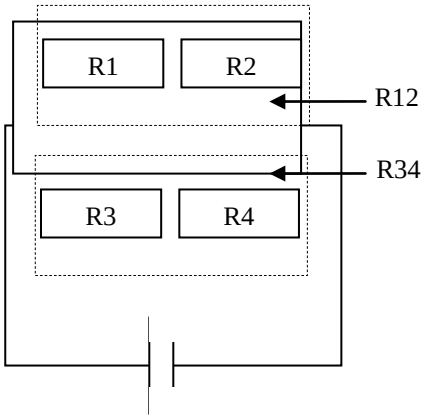
Результат роботи програми:
Ціле число = 4760
4760 = 1 * 2 * 2 * 2 * 5 * 7 * 17

2.9. Підпрограми

Інкапсуляція (в загальному випадку) – процес створення оболонки навколо речовини. Оболонка називається капсулою, речовина – інкапсульованою.

Інкапсуляція в програмуванні – це процес, в результаті виконання якого частина програми, що розв'язує під задачу агрегується в незалежну структуру, утворюючи капсулу. Капсули можуть використовуватися в програмі, як єдині закінчені частини програми.

Приклад. Нехай ділянка кола, що складається з чотирьох резисторів (див мал.) споживає різницю потенціалів U. Визначити силу струму I, яка протікає через дане коло.



Якби нам був відомий загальний опір R усієї ділянки, то струм в колі ми знайшли б за законом Ома :

$$I = \frac{U}{R}$$

Якщо ми уважно подивимося на наше коло, то помітимо, що воно складається з двох паралельно з'єднаних ділянок R12 та R34. А з курсу фізики відомо, що:

- опір двох послідовно з'єднаних провідників визначається за формулою:

$$R12 = R1 + R2; \qquad R34 = R3 + R4$$

- опір двох паралельно з'єднаних провідників визначається за формулою:

$$R = \frac{R12 \cdot R34}{R12 + R34}$$

Для розв'язання задачі, можна попередньо створити дві незалежні структури:

- структуру, що обчислює опір послідовно з'єднаних провідників
- та структуру, що обчислює опір паралельно з'єднаних провідників.

Тоді, використовуючи першу структуру, можна знайти опори R12 та R34, а використовуючи другу – опір R.

Побудова капсули здійснюється шляхом оформлення частини програми як незалежного елемента програми і створення оболонки (інтерфейсу), що забезпечує правильне використання капсули. Через інтерфейс відбувається взаємодія під програмної капсули з головною програмою. Так капсула, що знаходить опір двох послідовно з'єднаних провідників повинна мати оболонку (інтерфейс), що відповідає наступним вимогам:

1.З головної програми капсулі можна було б передати значення двох опорів (наприклад R1 та R2).

2.Капсула повинна передавати головній програмі значення загального опору (а саме опору R12).

Об’єкти, що забезпечують передачу значень з головної програми всередину капсули, і навпаки, передачу результату роботи капсули в головну програму називаються параметрами.

Програміст створивши підпрограмну капсулу має можливість багаторазового використання капсули. Так одну підпрограмну капсулу призначену, наприклад, для знаходження опору двох послідовно з’єднаних провідників можна використати двічі: для знаходження опору R12 та опору R34.

Підпрограмні капсули будемо називати **підпрограмами.**

Підпрограма

Спочатку підпрограми розглядалися як частина програми, що повторюється, а отже як спосіб скорочення запису програми в цілому. Підпрограми описуються окремо, незалежно від головної програми і використовуються в програмі за допомогою викликів з головної програми.

Процес використання підпрограм розділяється на 2 етапи:

-опис підпрограми (підпрограма утворює незалежну капсулу);

-виклик підпрограми з головної програми (передача управління від головної програми до під програмної капсули).

На сьогодні до підпрограм пред’являються дещо інші вимоги: підпрограма це не стільки спосіб скорочення запису програми, скільки засіб розкладання програми на логічно зв’язані закінчені і замкнуті компоненти, що визначають структуру програми в цілому, полегшують її розуміння. Однак необхідність попереднього опису та виклику підпрограм збереглися.

На основі підпрограм будується програмні конструкції, що називаються функціями та процедурами.

Механізми процедури та функції є обов’язковими для мов високого рівня.

Для мови С++ суттєвої відмінності між процедурами та функціями немає. Процедура виступає як різновид функції.

Опис підпрограми розглянемо на прикладі опису підпрограмної капсули (функції) знаходження опору провідників:

1. Послідовно з’єднаних (successive – в перекладі з англ. послідовне)

2. Паралельно з’єднаних (parallel – в перекладі з англ. паралельне)

Послідовне з’єднання

```
float successive(float R1,float R2) //заголовок функції
  {                                 //тіло функції
   Float R;
   R = R1 + R2;
   Return R;
  }
```

Паралельне з’єднання

```
float parallel (float R1,float R2) //заголовок функції
  {                                //тіло функції
   Float R;
   R := (R1*R2)/(R1 + R2);
   Return R;
  }
```

Структура заголовку функції: <тип функції><назва функції> (<список формальних параметрів>)

Список **формальних параметрів** забезпечують взаємодію процедури з головною програмою, а саме:

- передачу значень з головної програми в процедуру. В наведеному прикладі це параметри R1 та R2. Вони дійсного типу. Це означає, що функція може отримати від головної програми два дійсні значення для їх обробки. Значення, які отримує підпрограма від головної програми називають **фактичними параметрами.**

Передача значень від функції головній програмі.

В наведеному прикладі передача відбувається через дійсну R.

Return R означає передати головній програмі обчислене в підпрограмній капсулі значення R. **Текст програми в цілому:**

Перший варіант цієї програми:

```
#include<iostream.h>
#include<conio.h>
//---------------------------------------------
float succesive(float r1,float r2)
{                                 //Опис функції succesive
 float r;  r = r1 + r2;
 return r;
}
//---------------------------------------------
float parallel(float r1,float r2)
{                                 //Опис функції parallel
 float r;  r = r1*r2/(r1+r2);
 return r;
}
void main()                       //Головна програма
{
 clrscr();
 float R1,R2,R3,R4;
 cout<<"R1 = ";cin>>R1;
 cout<<"R2 = ";cin>>R2;
 cout<<"R3 = ";cin>>R3;
 cout<<"R4 = ";cin>>R4;
 float R;
 R = parallel(succesive(R1,R2),succesive(R3,R4));
 cout<<"R = "<<R<<endl; getch();
}
```

```
#include<iostream.h> // Другий варіант цієї програми:
#include<conio.h>
float succesive(float,float); //Оголошення функції succesive
float parallel(float,float);    // Оголошення функції parallel
void main()                     //Головна програма
{
 clrscr();
 float R1,R2,R3,R4;
 cout<<"R1 = ";cin>>R1;
 cout<<"R2 = ";cin>>R2;
 cout<<"R3 = ";cin>>R3;
 cout<<"R4 = ";cin>>R4;
 float R;
 R = parallel(succesive(R1,R2),succesive(R3,R4));
 cout<<"R = "<<R<<endl;  getch();
}
//---------------------------------------------
float succesive(float r1,float r2)      //Опис функції succesive
{
 float r;  r = r1 + r2;
 return r;
}
//---------------------------------------------
```

В мові С++ опис функцій може здійснюватися як перед головною програмою (перший варіант), так і після головної програми (другий варіант). Але якщо ми описуємо деяку функцію після головної програми, необхідно все ж таки перед головною програмою оголосити цю функцію. При оголошенні функції назви параметрів не вказуються, вказуються лише їх типи.

2.9.1. Локальні та глобальні об’єкти підпрограми

Приклад1. Програма знаходження більшого з трьох чисел

```
#include<iostream.h>
#include<conio.h>
float m;       //Глобальна змінна
void max(float,float,float);
void main()
{
 clrscr();  float x,y,z;
 cout<<"x = ";cin>>x;
 cout<<"y = ";cin>>y;
 cout<<"z = ";cin>>z;
 max(x,y,z); //Змінюємо значення глобальної m
 cout<<"Максимальне - "<<m<<endl;
 getch();
}
//----------------------
void max(float a,float b,float c)
{
 if (a>b&&a>c) m = a;
    else  if (b>c) m = b;  else m = c;
}
```

Змінні, які описані поза всіма функціями, тобто на початку програми називаються глобальними (змінна m). До глобальних змінних можна звернутися з будь-якої функції та блока. Глобальні об’єкти можуть використовуватися для повернення результату підпрограми. Тобто в підпрограмі можна не використовувати команди return: передати значення головній програмі можна відразу в тілі підпрограми, надавши значення глобальній змінній. Але такий підхід не є професійним, так як в цьому випадку підпрограму стає неможливо багаторазово використовувати для зміни значень різним глобальним об’єктам. Функція max(float, float, float) має тип void. Це означає, що дана функції ніякого значення головній програмі не повертає. Аналогом функції типу void є процедури в інших мовах програмування (наприклад Паскаль). Саме в підпрограмах типу void команда return не використовується.

Локальні змінні. Одна з цілей, що досягається в застосуванні інкапсуляції – це обмежений доступ до конструкції, розташованої в середині під програмної капсули. Це можливість використовувати програмні об’єкти тільки всередині капсули. Такі об’єкти називаються локальними. Вони описуються в тілі підпрограми. Локальні об’єкти – це об’єкти які створюються в момент виклику

підпрограми, і існують до тих пір, поки виконується тіло підпрограми. Коли по завершенню роботи підпрограми керування передається головній програмі пам'ять виділена під локальні об'єкти вивільняється, тобто усі локальні об'єкти знищуються. Тому у різних функціях однієї і тієї програми можна використовувати змінні з однаковими іменами. Локальні об'єкти – це об'єкти описані всередині під програмної капсули.

Приклад 2. Програма знаходження більшого з трьох чисел

```
#include<iostream.h> #include<conio.h>
float Max(float,float,float);
void main()
{
 clrscr(); float x,y,z,m;
 cout<<"x = ";cin>>x;
 cout<<"y = ";cin>>y;
 cout<<"z = ";cin>>z;
 m = Max(x,y,z);
 cout<<"Максимальне - "<<m<<endl;
 getch();
}
float Max(float a,float b,float c)
{
 float m;                          //Локальна
 if (a>b&&a>c) m = a; else if (b>c) m = b; m = c;
return m;
}
```

Зауважимо, що змінна m оголошена відразу у двох функціях main() та Max().

Коли в головній функції виконується команда m = Max(x,y,z), то згідно із правилом дії оператора присвоєння спочатку викликається функція Max(x,y,z). Значення змінних x,y,z передаються параметрам a,b,c підпрограмної капсули Max(), а також створюється локальна змінна m (вона описана всередині підпрограми Max()), значення якої обчислюється в результаті роботи операторів цієї капсули. Команда return m передає це значення головній функції як значення функції Max(x,y,z), коли її робота завершується. При цьому локальна змінні m, оголошена в функції Max() знищується і існує лише єдина m, яка оголошена в головній функції main(). Цій змінній і надається значення функції Max(x,y,z).

Принципові розходження між параметрами і локальними об'єктами

Параметр сполучає підпрограму з її оточенням. Параметри використовуються тільки для передачі інформації між оточенням та підпрограмою. В описі підпрограми присутні формальні параметри, які створюються в момент виклику підпрограми. При виклику з головної програми у назві підпрограми перераховуються фактичні параметри, які повинні мати значення. Ці значення передаються формальним параметрам

Локальні ж об’єкти є тимчасовими ресурсами, необхідними для виконання під програмної капсули і є цілком схованими в середині підпрограми. Локальні об’єкти – це здебільшого проміжні змінні, необхідні для розв’язання під задачі, що ставиться перед підпрограмною капсулою. По завершенню роботи підпрограми локальні об’єкти знищуються.

У зв’язку з наявністю глобальних та локальних об’єктів, оболонка капсули є блоком і діє як мембрана, пропускаючи в себе глобальні об’єкти і не випускаючи локальних. Цей ефект в програмуванні носить назву мембранного ефекту. Але правило хорошого стилю програмування забороняє використання глобальних об’єктів в підпрограмі (для передачі значень в підпрограму використовуються параметри, а для повернення значень з підпрограми в головну програму – використовують команду return. Це робить підпрограму універсальною).

Посилання як формальні параметри підпрограм

Приклад1 Демонстрація дії статичних параметрів.

```
void swap(int,int);                  //-------------------------
void main()                          void swap(int a,int b)
{                                    {
 int nx=5,ny=6; // nx=5,ny=6         int t;
 swap(nx,ny);    // nx=5,ny=6         t = a; a = b; b = t;
}                                    }
```

Функція swap(int a,int b) має тип void, тобто ніякого значення в головну програму не повертає. Що ж відбувається з її параметрами? В момент виклику swap(nx,ny) з головної функції, створюються деякі a та b – цілком незалежні комірки пам’яті, яким передаються значення nx, ny. В процесі роботи swap() значення a та b міняються місцями і це не впливає на nx та ny.

Приклад2 Використання глобальних змінних в описі підпрограм (побічний ефект).

```
int nx,ny;                           //--------------------------
void swap();                         void swap()
void main()                          {
{                                     int t;
 nx = 5; ny = 6; // nx=5,ny=6        t = nx; nx = ny; ny = t;
 swap();          // nx=6,ny=5        }
}
```

Змінні nx,ny описані як глобальні. Функція void swap() – без параметрів, але в тілі цієї підпрограми використовуються глобальні змінні nx та ny, значення яких міняються місцями.

Якщо спостерігати за роботою лише головної програми, у якій викликається функція swap() без параметрів, то зміна значень глобальних nx та ny виглядає не логічно та непередбачливо. Це явище в програмуванні дістало назву побічного ефекту.

Приклад 3. Використання посилань.

```
void swap(int &,int &);          //------------
void main()                      void swap(int &a,int &b)
{                                {
 int nx=5,ny=6; // nx=5,ny=6      int t;
 swap(nx,ny);   // nx=6,ny=5     t = a; a = b; b = t;
}                                }
```

або

```
void swap(int*,int*);            //--------------------------
void main()                      void swap(int* a,int* b)
{                                {
 int nx=5,ny=6;    // nx=5,ny=6    int t;
 swap(&nx,&ny); // nx=6,ny=5     t = *a; *a = *b; *b = t;
}                                }
```

В першому випадку функція swap в якості параметрів отримує локальні змінні nx,ny. В тілі цієї функції відбувається обмін місцями значень, що знаходяться за адресами змінних nx та ny (&nx та &ny).

В другому випадку в тілі функції swap також відбувається обмін місцями значень, що знаходяться за адресами &nx та &ny. Але в цьому випадку функції swap передаються не самі змінні nx,ny, а їхні адреси (&nx та &ny).

Задача З клавіатури вводиться три цілі числа. Впорядкувати їх за зростанням

```
#include<iostream.h>
#include<conio.h>
void sorting(int &,int &,int &);          //-----------------------------------
void main()                               void sorting(int &a,int &b,int &c)
{                                         {
 clrscr();                                  int t;
 int nx,ny,nz;                              if (a > b) {t = a; a = b; b = t;}
 cout<<"Три довільні цілі числа: ";         if (b > c) {t = b; b = c; c = t;}
 cin>>nx>>ny>>nz;                           if (a > b) {t = a; a = b; b = t;}
 sorting(nx,ny,nz);                       }
 cout<<"Ці числа в порядку зростання: "   Результат роботи програми
 <<nx<<" "<<ny<<" "<<nz<<endl;            Три довільні цілі числа: 2 1 3
 getch();                                 Ці числа в порядку зростання: 1 2 3
}
```

Задачу розв'язано за так званим алгоритмом «бульбашки». Розглянемо дію алгоритму на прикладі.

Нехай nx = 3 ny = 2 nz = 1. В результаті виконання перших двох команд розгалуження отримаємо nx = 2 ny = 1 nz = 3. «Найлегші» елементи (2 та 1) піднялись до початку на один крок, а «найтяжчий» елемент (3) «потонув» в кінець.

Після виконання третьої команди розгалуження впорядковуються перші два елементи.

Зауваження! Алгоритм «бульбашки» можна використовувати для впорядкування довільної кількості елементів. Детальніше з використання цього алгоритму для впорядкування великої кількості елементів ми ознайомимося при вивченні масивів.

Класи пам'яті

Для того, щоб безпосередньо вказати комп'ютеру як і де у його пам'яті мають зберігатися значення змінних чи функцій, як можна отримати доступ до цих даних, як визначити область видимості цих даних, використовують специфікатори класу пам'яті. Є п'ять специфікаторів:

- Auto
- Static
- Volatile
- Regicter
- Extern

Дія цих специфікаторів.

Auto – застосовується для локальних змінних по замовчуванню.

Область видимості – обмежена блоком, в якому вони оголошені.

Register – вказує компілятору, що значення слід зберігати в регістрах процесора (не в оперативній пам'яті). Це зменшує час доступу до змінної, що прискорює виконання програми.

Область видимості – обмежена блоком, в якому вони оголошені.

Static – застосовується як для локальних, так і для глобальних змінних.

```
int sum_next(int);                    //----------------
void main()                           int sum_next(int n)
{                                     {
 int n = 10,S;  //S = довільне ціле      static int S=0;//Обчисюється 1 раз
 for (int i=1;i<=n;i++)                  S+=n;          //під час компіл
       S = sum_next(i);  //S = 55        return S;
}                                     }
```

Область видимості – значення локальної статичної змінної зберігається після виходу з блока чи функції, де ця змінна оголошена. Під час повторного виклику функції змінна зберігає своє попереднє значення. Якщо змінна явно не ініцілізована, то за замовчуванням їй надається значення 0.

Ця програма обчислює суму перших 10 цілих додатних чисел.

Extern – використовується для передачі для передачі значень глобальних змінних з одного файлу в інший (часто великі програми складаються з кількох файлів).

Область дії – всі файли, з яких складається програма.

Volatile – застосовується до глобальних змінних, значення яких можуть надходити від переферійних пристроїв (наприклад від системного таймера)

2.9.2. Перевантаження та шаблони

Перевантаження функцій.

```
int inc(int i)                  void main()
{                               {
  i++;  return i;                char c='a';
}                                int i = 10;
char inc(char i);               c = inc(c); //c='b'
{                               i = inc(i);  //i=11
  i=char(int(++i));             }
  return i;
}
```

У C++ допускається використовувати одне і те саме ім'я функцій для різних наборів аргументів. Це називається **перевантаженням функцій**, або **поліморфізмом**. Перевантаження використовується, коли необхідно викликати функцію з аргументами різних типів, або коли функція залежить від різної кількості аргументів.

Задача. Програма впорядкування трьох введених з клавіатури величин або символьного, або цілого типу.

Головна функція

```
#include<iostream.h>
#include<conio.h>
void sorting(int &,int &,int &);
void sorting(char &,char &,char &);
void main()
{
 clrscr();
 int k;
 cout<<"Введ. 1 для  чисел, 0 - символів: ";
 cin>>k;
 if (k==1)
 {
   int nx,ny,nz;
   cout<<"Три довільні цілі числа: ";
   cin>>nx>>ny>>nz;
   sorting(nx,ny,nz);
   cout<<"Ці числа в порядку зростання: "
   <<nx<<" "<<ny<<" "<<nz<<endl;
 }
 if (k==0)
 {
   char nx,ny,nz;
   cout<<"Три довільні символи: ";
   cin>>nx>>ny>>nz;
   sorting(nx,ny,nz);
   cout<<"Ці символи в порядку зростання: "
   <<nx<<" "<<ny<<" "<<nz<<endl;
 }
 getch();
}
```

Перевантажувані функції

```
//---------------------------------------
void sorting(int &a,int &b,int &c)
{
  int t;
  if (a > b) {t = a; a = b; b = t;}
  if (b > c) {t = b; b = c; c = t;}
  if (a > b) {t = a; a = b; b = t;}
}
```

```
//---------------------------------------
void sorting(char &a,char &b,char &c)
{
  char t;
  if (a > b) {t = a; a = b; b = t;}
  if (b > c) {t = b; b = c; c = t;}
  if (a > b) {t = a; a = b; b = t;}
}
```

Шаблони функцій

```
#include<iostream.h>
#include<conio.h>
template <class My_type>                    //Оголошення
void sorting(My_type &,My_type &,My_type &); //шаблону
void main()
{
 clrscr();
 int k;
 cout<<"Введ. 1 для впорядкування чисел, 0 - символів: ";
 cin>>k;
 if (k==1)
 {
  int nx,ny,nz;
  cout<<"Три довільні цілі числа: ";
  cin>>nx>>ny>>nz;
  sorting(nx,ny,nz);          //створення екземпляра шаблону
  cout<<"Ці числа в порядку зростання: "
  <<nx<<" "<<ny<<" "<<nz<<endl;
 }
 if (k==0)
 {
  char nx,ny,nz;
  cout<<"Три довільні символи: ";
  cin>>nx>>ny>>nz;
  sorting(nx,ny,nz);          //створення екземпляра шаблону
  cout<<"Ці символи в порядку зростання: "
  <<nx<<" "<<ny<<" "<<nz<<endl;
 }
 getch();
}
```

```
//Опис шаблону
//-------------------------------------------------------------
template <class My_type>
void sorting(My_type &a,My_type &b,My_type &c)
{
 My_type t;
 if (a > b) {t = a; a = b; b = t;}
 if (b > c) {t = b; b = c; c = t;}
 if (a > b) {t = a; a = b; b = t;}
}
```

Шаблон функції – це опис функції, яка залежить від даних довільного типу. Під час виклику такої функції компілятор автоматично проаналізує тип фактичних аргументів, згенерує для них програмний код. Це називається неявним створенням **екземпляра шаблона**. Запис:

template <class My_type>

void sorting(My_type &,My_type &,My_type &) - оголошення шаблону.

My_type – назва деякого узагальнюючого типу. У списку формальних параметрів можуть бути присутні лише параметри узагальнюючих типів. Якщо шаблон описаний перед головною програмою, то він, як і звичайна функція, оголошення не потребує.

Підхід у програмуванні, що ґрунтується на використанні шаблонів функцій називається **узагальнюючим програмуванням**.

2.9.3. Рекурсія

```
#include<iostream.h>
#include<conio.h>
long int fact(int);
void main()
{
 long int y; int n; clrscr();
 cout<<"n = "; cin>>n;
 y = fact(n);
 cout<<n<<"! = "<<y<<endl;
 getch();
}
long int fact(int n)
{
 long int f;
 if (n == 1) f = 1;
 else f = n*fact(n-1);
 return f;
}
```

1

```
#include<iostream.h>
#include<conio.h>
void flop(int);
void flip(int);
void main()
{
 clrscr();flop(5);getch();
}
void flop(int n)
{
 cout<<"flop ";
 if (n>0) flip(n-1);
}
void flip(int n)
{
 cout<<"flip ";
 if (n>0) flop(n-1);
}
```

2

Рекурсія – це алгоритмічна конструкція, де підпрограма викликає сама себе (пряма рекурсія), або коли одна підпрограма викликає другу підпрограму, яка в свою чергу викликає першу. Приклад 1 – пряма рекурсія, 2 – взаємна.

2.10. Масиви

Масив — це структура даних, яку можна розглядати як набір змінних однакового типу, що мають спільне ім'я. Масиви зручно використовувати для зберігання однотипної інформації, наприклад, елементів таблиць, коефіцієнтів громіздких лінійних рівнянь та систем рівнянь, прізвищ працівників деякої великої фірми, назв днів тижня, тощо. Масив зберігається в послідовно розташованих комірках оперативної пам’яті, які мають спільну назву. Кожна окрема така комірка – це **елемент масиву**. Для ідентифікації елементів масиву, кожен елемент має свій індекс, за яким його можна знайти в масиві. Кількість індексів, а отже і кількість елементів визначає **розмірність масиву**. Розрізняють одномірні, двомірні та багатомірні масиви.

Оголошення масиву. Операції над елементами.

Загальний вигляд конструкції опису одномірного масиву: <тип> <ім’я масиву> [<розмір>]

Наприклад int a[10] – масив на ім’я ***a*** складається з десяти цілих елементів: a[0], a[1], a[2],…a[9].

Нумерація елементів масиву в С++ завжди починається з нуля.

Ім’я масиву є вказівником на його перший елемент. Тобто *a – це те саме, що a[0], *(a+1) – це те саме що a[1],…

Отже, звернутися до елементів масиву можна двома способами:

- за допомогою індексів масиву (a[2] – третій елемент масиву)
- або використовуючи вказівники (*(a+2) – також третій елемент масиву)

Приклад Утворити масив 0, 10, 20, 30, 40

```
void main()
{
 int a[5];
 for (int i=0;i<5;i++) *(a+i) = i*10;
}
```

або

```
void main()
{
 int a[5];
 for (int i=0;i<5;i++) a[i] = i*10;
}
```

```
a[0]: 0
a[1]:
10
a[2]:
20
```

Масив можна ініціалізувати повністю або частково відразу під час оголошення:

```
void main()                    1
{
 int a[5] = {0, 10};
 for (int i=2;i<5;i++) a[i] = i*10;
}
```

```
void main()                    2
{
 int a[] = {0,10,20,30,40};
}
```

```
a[0]: 0
a[1]: 10
a[2]: 20
a[3]: 30
a[4]: 40
```

Випадок 1 – ініціалізуємо при оголошені лише два перших елемента. Значення іншим трьом елементам надаємо в циклі for.

Випадок 2 – при оголошені ініціалізуємо всі елементи (в цьому випадку в [..] вказувати кількість елементів необов'язкова.

Задача. Утворити масив з перших десяти цілих чисел і обчислити суму всіх його значень.

Нижче – три варіанти розв'язку цієї задачі.

```
void main()
{
 int a[5];
 for (int i=0,s=0;i<5;)
 {
  *(a+i)=i;
  s+=*(a+i);
  ++i;
 }
}
```

```
void main()
{
 int a[5];
 for (int i=0,s=0;i<5;i++)
 {
  a[i]=i;
  s = s + a[i];
 }
}
```

```
void main()
{
 int a[5];
 for (int i=0,s=0;i<5;*(a+i)=i,s+=*(a+i),++i);
}
```

Результат роботи кожної з цих програм
a[0] = 0 a[1] = 1 a[2] = 2 a[3] = 3 a[4] = 4
S = 10

Введення-виведення масиву

Під час компіляції програмного коду для статично оголошених масиві надається пам'ять. Для ефективного використання пам'яті призначене динамічне оголошення масивів, а саме:

```
<тип вказівника> *<назва>  = <тип змінної> [<кількість>];
```

Звичайний статичний масив – це сукупність послідовно розташованих комірок пам’яті. Динамічний масив – це неперервна ділянка пам’яті розміром **sizeof**(тип змінної)*<кількість>. Щоб вивільнити пам’ять з-під динамічного масиву користуються командою

```
delete [ ]<назва вказівника на масива>
```

Введення-виведення статичного масиву

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr(); int a[5];
 for (int i=0;i<5;i++)
 { cout<<"a["<<i<<"] = "; cin>>*(a+i); }
 cout<<"Введений масив:\n";
 for (i=0;i<5;i++)
   cout<<"a["<<i<<"] =
"<<*(a+i)<<endl;
 getch();
}
```

Результат роботи програми

```
a[0] = 4
a[1] = 7
a[2] = 9
a[3] = 3
a[4] = -8
Введений масив:
a[0] = 4
a[1] = 7
a[2] = 9
a[3] = 3
a[4] = -8
```

Вводити та виводити масив можна не лише в циклі з параметром, а і в циклах while, do-while. В програмі ми вводимо дані в статичний масив. Кількість елементів такого масиву зазначається при описі. Тобто визначається на етапі компіляції.

Введення-виведення динамічного масиву

```
#include<iostream.h>
#include<conio.h>
void main()
{
 clrscr(); int n; cout<<"n = "; cin>>n;
 int *a = new int [n];
 for (int i=0;i<n;i++)
   {cout<<"a["<<i<<"] = ";cin>>a[i];}
 cout<<"Введений масив:\n";
 for (i=0;i<n;i++)   cout<<"a["<<i<<"] = "<<a[i]<<endl;
 delete [ ]a; getch();
}
```

```
n = 5
a[0] = 2
a[1] = 3
a[2] = 1
a[3] = 6
a[4] = 8
Введений масив:
a[0] = 2
a[1] = 3
a[2] = 1
a[3] = 6
a[4] = 8
```

Для динамічного масиву ми його розмір N можемо задавати з клавіатури.

2.10.1. Пошук у масиві

Задача. Знайти в цілочисельному масиві індекс введеного з клавіатури елемента

```
#include <iostream.h>
#include<conio.h>
void main()
{
 clrscr(); int n;
 cout<<"Введіть масив цілих
чисел:\n"
      <<"Кількість елементів = ";
 cin>>n;
 int a[100];
 for (int i=0;i<n;i++)
   {cout<<"a["<<i<<"] = ";
     cin>>*(a+i);  }
 int x;
 cout<<"Введіть шуканий елемент =
";
 cin>>x;
 for (i=0;(*(a+i)!=x)&&(i<n);i++);
 if (i==n) cout<<"Немає такого
елем\n";
  else  cout<<”Індекс шуканого елем =
“
             <<i<<endl;
 getch();
}
```

Результат роботи програми
Введіть масив цілих чисел:
Кількість елементів = 5
a[0] = 1
a[1] = 3
a[2] = 5
a[3] = 2
a[4] = 8
Введіть шуканий елемент = 5
Індекс шуканого елем = 2

Частину програми, що здійснює пошук елемента в масиві виділено сірим кольором.

Пошук мінімального елемента

Пошук найменшого елемента здійснюватимемо за таким алгоритмом. Від початку до кінця масиву відкриватимемо послідовно по одному елементу і на кожному кроці визначатимемо найменший елемент. На першому кроці найменшим буде саме цей перший елемент a[1]. На другому кроці порівняємо новий відкритий елемент a[2] з тим мінімумом, який ми вже маємо. Якщо новий елемент менший того мінімального, що визначений на попередньому кроці, то запам'ятаємо його, інакше залишимо старий результат. Таким чином, дійшовши до кінця масиву визначимо мінімальний елемент всього масиву.

```
#include <iostream.h>
#include<conio.h>
void main()
{
 clrscr(); int n;
 cout<<"Введіть масив цілих
чисел:\n"
    <<"Кількість елементів = ";
 cin>>n;
 int a[100];
 for (int i=0;i<n;i++)
  {cout<<"a["<<i<<"] = ";
       cin>>a[i]; }
 int min=*a,j=0;
 for (i=0;i<n;i++)
    if (*(a+i)<min)
      {min = *(a+i);j = i;}
 cout<<"Найменший елемент ="<<min<<endl;
 cout<<"Його індекс = "<<j<<endl;
 getch();
}
```

Результат роботи програми
Введіть масив цілих чисел:
Кількість елементів = 6
a[0] = 2
a[1] = -12
a[2] = 4
a[3] = -39
a[4] = 34
a[5] = 2
Найменший елемент =-39
Його індекс = 3

Запропонована програма не тільки знаходить найменший елемент, але ще й визначає, на якому місці в масиві він знаходиться.

Наведений приклад алгоритму визначає перший мінімальний елемент в масиві, хоча в ньому може бути і кілька таких елементів. Наприклад, якщо розглянути масив 1, 2, 2, 3, 1 – тут два мінімальні елементи, які рівні 1. Весь секрет полягає в умові a[i]<min. Адже новий елемент масиву буде запам'ятовуватися як найменший лише тоді, коли він строго менший за попередній.

Якщо в алгоритмі поміняти умову a[i]<min на умову a[i]<=min, то визначатиметься останній найменший елемент, оскільки новий елемент масиву буде запам'ятовуватися ще й тоді, коли він дорівнює попередньому.

Щоб розглянутий алгоритм виконував пошук найбільшого елемента, достатньо умову a[i]<min поміняти на a[i]>max.

Програма пошуку в масивах довільного типу та довільного розміру

```
#include<iostream.h>
#include<conio.h>
template <class typ>
void input(typ *mass,typ *elem,int n)
{
 for (int i=0;i<n;i++)
  { cout<<"a["<<i<<"] = ";
    cin>>mass[i]; }
 cout<<"Введ елемент, що шукається: ";
 cin>>*elem;
}
template <class M_t>
int search(M_t *mass,M_t elem,int n)
{
 int i=0;
 while (i<n && *(mass+i)!=elem) ++i;
 if (i==n) i=-1; return i;
}
//-----------------------------------------------
void main()
{
 clrscr();int k,n,i; int x1; float x2; char x3;
 cout<<"1-цілий масив, 2-дійсний, 3-символьний: ";
 cin>>k;
 cout<<"Кількість елементів: ";cin>>n;
 int *mas1 = new int[n]; float *mas2 = new float[n];
 char *mas3 = new char[n];
 if (k==1) {input(mas1,&x1,n);i = search(mas1,x1,n);}
 if (k==2) {input(mas2,&x2,n);i = search(mas2,x2,n);}
 if (k==3) {input(mas3,&x3,n);i = search(mas3,x3,n);}
 if (i==-1) cout<<"Не має такого елемента в масиві!\n";
   else cout<<"Шуканий елемент має індекс: "<<i<<endl;
 delete[ ] mas1;delete[ ] mas2;delete[ ] mas3;
 getch();
}
```

Приклад роботи програми для масивів різних типів:

1 – цілочисельний, 2 – дійсний, 3 – символьний.

input(*mass, *elem, N) – шаблон функції введення массиву **mass,** який має розмір **N** та шуканого елемента **elem**.

Search має тип int –значення, яке вона повертає в головну програму рівне індексу шуканого елемента (якщо такий елемент присутній) або -1 (якщо шуканого елемента немає).

2.10.2. Впорядкування масиву

```
1-цілий масив, 2-дійсний, 3-символьний: 1
Кількість елементів: 5
a[0] = 8
a[1] = -9
a[2] = 2
a[3] = 67
a[4] = 27
Введ елемент, що шукається: 67
Шуканий елемент має індекс: 3
```

```
1-цілий масив, 2-дійсний, 3-символьний: 2
Кількість елементів: 4
a[0] = 23.65
a[1] = -987.109
a[2] = .0987
a[3] = .01
Введ елемент, що шукається: .01
Шуканий елемент має індекс: 3
```

```
1-цілий масив, 2-дійсний, 3символьний: 3
Кількість елементів: 13
a[0] = a
a[1] = b
a[2] = g
a[3] = s
a[4] = l
a[5] = t
a[6] = w
a[7] = i
a[8] = m
a[9] = f
a[10] = q
a[11] = y
a[12] = x
Введ елемент, що шукається: w
Шуканий елемент має індекс: 6
```

Нехай дано деякий масив значень *a[1], a[2],..., a[n]*. Необхідно його елементи переставити місцями так, щоб вхідний масив перетворився в такий, для якого виконувалося б співвідношення *a[1] < a[2] < ... < a[n]*.

Метод вибору мінімальних елементів

Розглянемо такий приклад. Нехай ми на великій кількості карток напишемо деякі числа. Кожна окрема картка – це окремий елемент деякого масиву. Як навести порядок у картках, на яких написані ці числа? Спочатку необхідно знайти мінімальний елемент у всьому заданому масиві і поміняти його з елементом, який поки що стоїть на першому місці, тому що саме на цьому місці повинен стояти шуканий мінімальний елемент. Тепер вже перший елемент знаходиться на своєму місці і можемо його не розглядати далі, тобто будемо повторно починати пошук в масиві, але з другого елемента. З новим масивом виконаємо ті самі дії: знайдемо в ньому мінімальний елемент і поміняємо його місцями з першим відкритим елементом (насправді він є другим елементом у початковому масиві). Таким чином будемо продовжувати доти, доки на останньому кроці не залишаться два останні елементи. Якщо необхідно, поміняємо і їх місцями.

```
#include <iostream.h>
#include <conio.h>
void input(int *a, int n)
{
 for (int i=0;i<n;i++)
  { cout<<"a["<<i<<"] = ";
   cin>>*(a+i); }
}
//-----------------------
void output(int *a, int n)
{
 for (int i=0;i<n;i++)
   cout<<a[i]<<" ";
 cout<<endl;
}
//-----------------------
void sorting(int *a,int n)
{
 int j,k,min;
 for (int i=0;i<n-1;i++)
    {
     min =*(a+i);k=i;
     for (j=i+1;j<n;j++)
         if (*(a+j)<min){min=*(a+j);k=j;}
     j=*(a+i);*(a+i)=min;*(a+k)=j;
    }
}
//-----------------------
void main()
{
 clrscr();
 cout<<"Введіть масив:\n";
 cout<<"Кількість елем. = ";
 int n; cin>>n;
 int a[100]; input(a,n);
 sorting(a,n);
 output(a,n);getch();
}
```

Метод прямого обміну («бульбашки»)

```
void sorting(int *a,int n)
{
   int j,k;
   for (int i=0; i<n–1; i++)
     for (j=0; j<n–1; j++)
       if (*(a+j+1)<*(a+j))
         { k=*(a+j);  *(a+j)=*(a+j+1); *(a+j+1)=k; }
}
```

В основі алгоритму лежить метод обміну сусідніх елементів масиву. Кожен елемент масиву, починаючи з першого,

порівнюється з наступним і якщо він більший наступного, то ці елементи міняються місцями. Таким чином, елементи з меншим значенням переміщаються до початку масиву (спливають), а елементи з більшим значенням – до кінця масиву (тонуть). Тому цей метод називають методом «бульбашки». Цей процес повторюється на 1 менше разів, чим елементів у масиві.

2.11. Структури даних

На практиці приходиться мати справу з даними, які складаються з інших даних. Наприклад, інформація про учня містить: прізвище, ім'я, по-батькові, дату народження, адресу, та іншу інформацію. Така інформація складається з даних різних типів. Ці дані називають **полями**. Сукупність полів з різнотипних даних являє собою структуру.

Опис структури має вигляд:

```
Struct <назва типу структури>
    {
    <тип поля1> <назва поля1>;
    <тип поля2> <назва поля2>;
    ……..
    <тип поляN> <назва поляN>;
    }
```

Наприклад

```
Struct person
    {
    char fam[15];          // прізвище (масив з 15 символів)
    char name[15];         // ім'я (масив з 15 символів)
    inr day, month, year;  // цілі: день, місяць та рік народження
    char address[50];      // адреса проживання (масив 50 символів)
    }
```

Після опису структури можна оголосити змінну.

Наприклад Person student;

Змінна student містить в собі різнотипні дані. Звернутися до того чи іншого даного змінної типу структури можна так:

<назва змінної>.<назва поля>

Наприклад student.fam = "Степанюк"

Структури використовуються для створення баз даних. Базою даних є масив, кожен елемент якого має структурний тип, де **student** – масив з п'яти елементів, кожен з яких містить описані в типі **person** поля.

Наприклад, створити масив типу Person можна так: **person *student = new person[5]** або **person student[5]**

Задача. Створити та вивести на екран базу даних з 5 студентів. Полями кожного студента є: прізвище; ім'я; день, місяць та рік народження, адреса (дані вводяться з клавіатури)

Звернутися до поля (наприклад name) деякого k-го елемента массиву **student** можна так: student[k].name

```
#include<iostream.h>
#include<conio.h>
struct person
  {
   char fam[15],name[15];
   int day,month,year;
   char address[50];
  };
//----------------------------------------
void input(person *stud, int n)
{
 for (int i=0;i<n;i++)
  {
   cout<<"\t3апис "<<i+1<<":\n";
   cout<<"Прізвище -
";cin>>stud[i].fam;
      cout<<"Ім'я - "; cin>>stud[i].name;
   cout<<"День народж. - "; cin>>stud[i].day;
   cout<<"Місяць - "; cin>>stud[i].month;
   cout<<"Рік - "; cin>>stud[i].year;
   cout<<"Адреса - "; cin>>stud[i].address;
  }
}
//----------------------------------------
void output(person *stud, int n)
{
 for (int i=0;i<n;i++)
  {
   cout<<"\t3апис "<<i+1<<":\n";
   cout<<stud[i].fam<<" "
       <<stud[i].name<<"\t"
       <<stud[i].day<<"."
       <<stud[i].month<<"."
       <<stud[i].year<<"\t"
       <<stud[i].address<<endl;
  }
}
//----------------------------------------
void main()
{
 clrscr();
 person *student = new person[5];
 input(student,5);
 output(student,5);
 delete [ ]student;
 getch();
}
```

В програмі ми оголосили динамічний масив з п’яти елементів типу person, де тип person є структурою. Можна, звичайно використовувати і статичний масив (замість **person *student = new person[5]** записати **person student[5]**). Результат роботи програми при цьому не зміниться.

2.11.1. Вказівники на структури. Списки: стек, черга

В програмуванні дуже широко використовуються вказівники на структури.

<назва структури> *<назва вказівника>

Наприклад **person *student:** student – вказівник на структуру person.

Доступ до полів вказівника на структуру здійснюється дещо інакше, ніж до полів відповідної змінної, а саме: <назва вказівника>-><назва поля>**.** Крім того, одним із полів структури може бути вказівник. Більш того цей вказівник може вказувати на таку ж структуру.

Наприклад, розглянемо наступний опис:

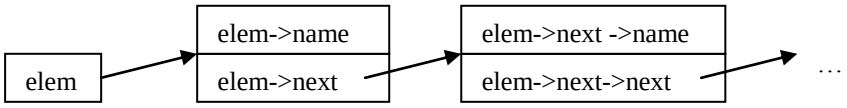
Struct person

```
{
char name[15]; // ім’я (масив з 15 символів)
person *next;    // вказівник next на структуру person
}
```

Тут поле next є вказівником на змінну типу person, яка в свою чергу також містить поле next.

Якщо деяку змінну - вказівник elem оголосити так: **person *elem;**

то графічно таку змінну можна зобразити так:



Тут вказівник elem вказує на структуру, одне із полів якої (next) є вказівником на таку ж структуру. Такий набір даних називається **списком**. Елемент списку складається як мінімум з двох частин: поля даних та поля-вказівника на наступний елемент. Варто зауважити, що при описі змінної-вказівника elem, як і для довільного іншого вказівника, виділяється лише стільки пам'яті, скільки необхідно для збереження адреси першого елемента списку. Перший елемент списку є динамічною структурою, що містить поле даних та поле-вказівник на іншу таку ж динамічну структуру. Оскільки кожен елемент списку є динамічною структурою, то для його створення (виділення пам'яті) використовують команду new. Списки бувають трьох типів:

- стек; - черга; - та дек.

Ми розглянемо лише стек та чергу.

Стек

```
#include <iostream.h>
#include <conio.h>
struct person
  {
  int dat;
  person *next;
  };
void Add_stack(person *&elem,person *&stack)
 {
  if (stack!=0) elem->next = stack;
  stack = elem;
 }
void print_stack(person *&stack)
{
 while (stack->next!=0)
   {
    cout<<stack->dat<<" ";
    stack = stack->next;
   }
 getch();
}
void main()
{
 clrscr();
 int data=10;
 person *elem;
 person *stack;
 do
   {
   if (data!=0)
          { elem = new(person);
            elem->dat = data;
            Add_stack(elem,stack); }
   data--;
   }
 while (data!=0);
 print_stack(stack);
}
```

Результат роботи програми: 1 2 3 4 5 6 7 8 9 10

Спочатку в список добавляється елемент 10, потім 9, …. Тобто кожен наступний елемент добавляється на початок списку. Самий останній елемент буде на першому місці, тобто вийде зі списку самим першим. Стек – це список, що працює за принципом «перший зайшов – останній вийшов».

Процедуру void Add_stack(person *&elem,person *&stack) добавляє елемент elem на початок списку stack (elem->next = stack). Після цього необхідно направити вказівник stack на елемент elem, який є початком списку (stack = elem).

Черга

```
#include <iostream.h>
#include <conio.h>
struct person
  {
  int dat;
  person *next;
  };
void Add_cherga(person *&elem,person *&cherga,person
*&head)
 {
  if (head == 0)
    { head = elem;
      cherga = elem;}
  else {cherga->next=elem;
        cherga=elem; }
 }
void print_cherga(person *&cherga)
{
 while (cherga!=0)
   { cout<<cherga->dat<<" ";
     cherga = cherga->next;   }
 getch();
}

void main()
{
 clrscr();
 int data=10;
 person *elem;
 person *cherga;
 person *head;
 head = 0;
 do
   {
   if (data!=0)
           { elem = new(person);
             elem->dat = data;
             Add_cherga(elem,cherga,head); }
   data--;
   }
 while (data!=0);
 print_cherga(head);
}
```

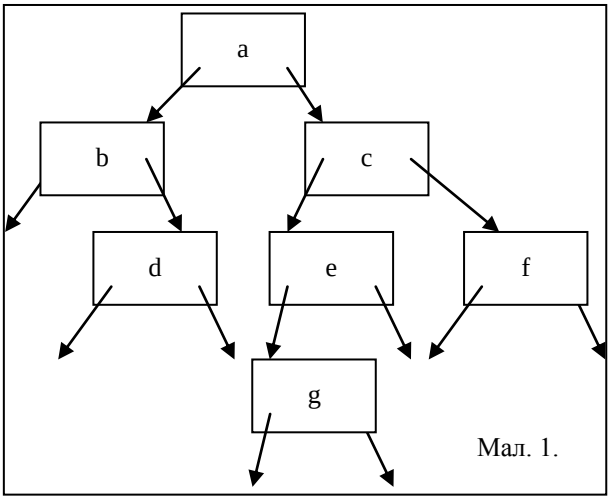
Результат роботи програми: 10 9 8 7 6 5 4 3 2 1

Спочатку в список добавляється елемент 10, потім за ним 9, …. Тобто кожен наступний елемент добавляється в кінець списку. Самий останній елемент буде на останньому місці, тобто вийде зі списку останнім. Черга – це список, що працює за принципом «перший зайшов – першим вийшов»

Процедуру Add_cherga(person *&elem,person *&cherga,person *&) добавляє елемент elem в кінець списку cherga (cherga->next=elem). Після цього необхідно направити вказівник cherga на елемент elem, який є в кінці списку (cherga = elem). Вказівник head вказує на перший елемент черги (голова черги).

2.11.2. Дерева. Бінарне дерево

Деревом називається динамічна структура, у якій кожен вузол містить не один, а декілька вказівників на декілька інших вузлів.



Мал. 1.

Кореневий вузол (корінь дерева) – це самий верхній вузол (вузол a – кореневий) . Якщо з деякого вузла (наприклад, вузла a) вказівники вказують на інші вузли (в нашому випадку на вузли b та c), то такий вузол називають предком. Вузли ж на які вказують вказівники від предка називаються потомками. Лівий потомок вузла a – вузол b Правий потомок – вузол c. Вузли b та c мають спільного предка – вузол a . Якщо вузол не має потомків, то такий вузол називають листком дерева. На мал. 1 вершини d, g та f – є листками.

Задача. Сформуємо дерево керуючись таким принципом: значення вершини кожного лівого потомка менше або рівне за значення вершини предка, а значення вершини правого потомка – більше значення вершини предка.

Опис структури дерево:

```
struct tree
{
  int top;      //вершина дерева
  tree *left;  //ліва гілка (потомок)
  tree *right;//права гілка
};
```

Виведення дерева на екран:

```
void write_tree(tree *t)
{
  cout<<t->top<<" ";
  if (t->left != NULL) write_tree(t->left);
  if (t->right != NULL) write_tree(t->right);
 //cout<<t->top<<" ";
```

Знищення дерева:

```
void del_tree(tree *t)
{
 if (t->left != NULL) del_tree(t->left);
 if (t->right != NULL) del_tree(t->right);
 delete t;
}
```

Добавлення елемента data в дерево tree

```
tree *add_tree(int data,tree *t)
{
 tree *new_elem = new tree;
 new_elem->top = data;
 new_elem->left = NULL;
 new_elem->right =NULL;
 tree *buf; buf = t;
 while (t!=NULL)
   if (data<=t->top)
     {
      if (t->left==NULL)
        { t->left = new_elem;
          t = t->left;  }
       t = t->left;
     }
     else
     {
       if (t->right==NULL)
         { t->right = new_elem;
           t = t->right;  }
         t = t->right;
     }
  return buf;
}
```

Головна функція:

```
void main()
{
 tree *tr = new tree;
 int data;
 cout<<"Елемент = "; cin>>data;
 tr->top = data;
 tr->left = NULL;
 tr->right = NULL;
 do
 {
   cout<<"Елемент = "; cin>>data;
   if (data!=0) tr = add_tree(data,tr);
 }
 while (data!=0);
 write_tree(tr);
 del_tree(tr);
 cout<<endl;
}
```

Додаток 1

Зразок оформлення титульного аркуша

Управління освіти, науки та молоді Волинської облдержадміністрації

Волинське територіальне відділення МАН України

Відділення: комп'ютерних наук

Секція: інформаційні системи, бази даних та системи штучного інтелекту

«Програма для створення та симуляції нейронних мереж «Neuro Master».

Роботу виконав:
Дацюк Денис Олегович,
учень 11 класу
Луцької гімназії № 21
імені Михайла Кравчука

Науковий керівник:
Щегельський Тарас Сергійович,
керівник гуртків інформатики
ВО МАН

Луцьк – 2015

Додаток 2

Зразок оформлення тез

Програма для створення та симуляції нейронних мереж «Neuro Master»

Дацюк Денис Олегович, Волинська обласна МАН

Луцька гімназія №21 імені Михайла Кравчука, 11 клас

Щегельський Тарас Сергійович, керівник гуртків інформатики ВО МАН

В наш час активно досліджуються анатомічні особливості будови мозку як тварин так і людей. Найбільший інтерес викликають такі можливості мозку як запам'ятовування, побудова асоціативних зв'язків і розпізнавання образів. При цьому всі ці операції виконуються з великою швидкістю, що унеможливлює їх відтворення за допомогою сучасних комп'ютерних систем.

Програмно реалізовані нейронні мережі є основним інструментом для дослідження штучних нейронних мереж. Саме цей факт обґрунтовує актуальність проблеми моделювання нейронних мереж та обумовлює вибір теми наукового дослідження.

Мета наукової роботи – створення цілісного програмного продукту, який би поєднував в собі як основні засоби для конструювання, навчання та симуляції штучних нейронних мереж так і зручний користувацький інтерфейс.

Новизною роботи є реалізація в комплексному підході програмної розробки із зручним користувацьким інтерфейсом. Крім того для розширення класу задач, які можна реалізувати за допомогою програми «NeuroMaster» була розроблена система плагінів для підготовки вхідних даних. Таким чином, була усунена необхідність впроваджувати в програмі підтримку різноманітних типів вхідних даних: графічних, звукових, відео та числових.

Розроблена програма оперує багатошаровими нейронними мережами без зворотних зв'язків. Алгоритмом навчання було вибрано алгоритм зворотного поширення похибки, як найбільш пристосований для впровадження на ЕОМ.

Додаток 3

Зразок оформлення змісту

Список використаних джерел

1.Боэм Б.У. Инженерное программирование для проектирования программного обеспечения / Б.У. Боэм. – М. : Радио и связь, 1985. – 512с.

2.Войтович І. Науково-дослідницька робота з інформатики у середніх та позашкільних навчальних закладах : навч.-метод. посіб. / І. Войтович, В. Сергієнко ; [відп. за вип. О. Лісовий]. – К. : ТОВ «Праймдрук», 2012. – 57 с.

3.Глинський Я. М. C++ Builder : навч. посіб. / Я. М. Глинський, В.Є. Анохін, В. А. Ряжська. – 4-те вид. – Л. : СПД Глинський, 2008. – 190 с.

4.Крушельницька О.В. Методологія та організація наукових досліджень : Навч. посіб. – К. : Кондор, 2003. – 192 с.

5.Михалюк Т.В. Організація науково-дослідницької роботи у відділі обчислювальної техніки та програмування МАН / Т.В. Михалюк. – Луцьк, 2007. – 36 с.

6. Мазурик В.В. Наукова робота з інформатики / В.В. Мазурик, Т.С. Щегельський. – Луцьк, 2010. – 26 с.

7.Малімон Л.Я. Творчо обдарована особистість: теорія дослідження та практика виховання ВО МАН / Л.Я. Малімон, Л.Б. Мазурик. – Луцьк, 2008.

8.Навчально-методичні та нормативно-правові засади діяльності наукового товариства у навчальному закладі нового типу: збірник інструктивно-методичних матеріалів / Відп. за випуск: Г.А.Толстіхіна; [уклад. Л.Я.Малімон, Л.Б.Мазурик]. – Луцьк, 2008. – 63с.

9.C++. Основи програмування. Теорія та практика : підручник / [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.]; за ред. О.Г.Трофименко. – Одеса : Фенікс, 2010. – 544 с.

10.Паппас К. Visual C++ : для пользователя : пер. с англ. / К. Паппас. – К. : Изд. группа BHV, 2000. – 335 с.

11.Шейко В.М. Організація та методика науково-дослідницької діяльності : Підруч. / В.М. Шейко, Н.М. Кушнаренко. - 3-є вид., стер. – К. : Знання-Прес, 2003. – 296 с.