Сумський державний педагогічний університет імені А. С. Макаренка
Фізико-математичний факультет
Кафедра інформатики

УДК 378.016:51:004

Рубець Артем Іванович

ОСОБЛИВОСТІ ВИВЧЕННЯ ТЕМИ «МОВА ПРОГРАМУВАННЯ ТА СТРУКТУРА ДАНИХ» У ПРОФІЛЬНІЙ ШКОЛІ

Галузь знань: 01 Освіта
Спеціальність 014 Середня освіта (Інформатика)
Кваліфікаційна робота на здобуття освітнього рівня «Магістр»

Науковий керівник:
_____ Сергій ПЕТРЕНКО
кандидат педагогічних наук, доцент,
доцент кафедри інформатики

Виконавець:
_____ Артем РУБЕЦЬ

Суми – 2024

ЗМІСТ

ВСТУП

Вивчення теми «Мова програмування та структура даних» у профільній школі передбачено навчальними програмами сучасного курсу інформатики. Разом з тим вивчення цієї теми стикається з низкою викликів, серед яких різнорівнева підготовка учнів за попереднім рівнем навчання, складність теми через абстрактність понять, які в ній вивчаються, що вимагає від учителя застосування різних методів навчання, посилення мотивації учнів та відсутність сформованих міжпредметних зв'язків.

Методичні особливості вивчення теми подані в науково-методичній літературі. Зокрема, цією проблемою займалися В. Биков, К. Гілберт, Е.Дейкстра, М. Жалдак, В. Конюшко, Б. Маккарті, Д. Рометті, Г. Шілдт та ін.. Науковці наголошують на важливості розвитку критичного мислення учнів в умовах швидкого розвитку технологій та вважають програмування тим інструментом, який цьому сприяє. Водночас узагальнення науково-методичних джерел свідчить, що з розвитком ІТ потребують перегляду і оновлення як зміст теми «Мова програмування та структура даних», так і її навчально-методичне забезпечення, зокрема, уточнення основних понять, вибір мови програмування, типові задачі та типові учнівські помилки.

Об'єкт дослідження: процес навчання інформатики у профільній школі.

Предмет дослідження: особливості вивчення теми «Мова програмування та структура даних» у профільній школі.

Мета дослідження: виявити особливості вивчення теми «Мова програмування та структура даних» у профільній школі.

Поставлена мета дослідження обумовила вирішення низки **завдань**:

1) обґрунтувати важливість вивчення програмування молоддю та представити короткий огляд популярних мов програмування для вивчення у ЗЗСО;

2) провести аналіз найбільш поширених структур даних та подати особливості їхньої побудови.

3) проаналізувати навчальні програми з інформатики профільної школи за темою дослідження;

4) виявити типові задачі теми та провести аналіз типових помилок учнів при їх розв'язуванні.

Для досягнення мети використано

теоретичні методи дослідження: аналіз і систематизація наукових джерел для характеристики стану розробленості проблеми дослідження; контент-аналіз для характеристики мов програмування; структурно-логічний аналіз для характеристики структур даних; аналіз і узагальнення для виявлення формальних характеристик вивчення теми «Мова програмування та структура даних»;

емпіричні методи дослідження – спостереження для виявлення типових помилок учнів при вивченні теми «Мова програмування та структура даних».

Теоретична і практична значущість дослідження полягає у виявленні особливостей вивчення теми «Мова програмування та структура даних».

Апробація матеріалів дослідження здійснювалася на наукових заходах різних рівнів, серед яких: студентська звітна конференція фізико-математичного факультету (травень, 2024) [20; 21].

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків та списку використаних джерел.

У першому розділі «Вивчення мов програмування у ЗЗСО» обґрунтовано важливість вивчення програмування молоддю та представлено короткий огляд популярних мов програмування для вивчення у ЗЗСО.

У другому розділі «Вивчення структур даних у ЗЗСО» проведено аналіз найбільш поширених структур даних та представлено особливості їхньої побудови.

У третьому розділі «Методичні особливості вивчення теми «Мова програмування та структура даних» в умовах профільної школи» проаналізовано навчальні програми з інформатики профільної школи,

наведено типові задачі теми та проведено аналіз типових помилок учнів при їх розв’язуванні.

Загальний обсяг роботи 80 сторінок, з яких 75 сторінок основного тексту. Список використаних джерел включає 48 одиниць. Робота містить 20 рисунків та 4 таблиці.

Робота буде цікавою працюючим і майбутнім учителям інформатики, які цікавляться проблемами навчання інформатики загалом та програмування і структур даних, зокрема.

Розділ 1.
ВИВЧЕННЯ МОВ ПРОГРАМУВАННЯ У ЗЗСО

1.1. Роль програмування в навчанні інформатики в ЗЗСО

Програмування як розділ інформатичної галузі займає одне з ключових місць у сучасній освітній системі, особливо в умовах швидкого розвитку інформаційних технологій (ІТ). Сьогодні володіння мовами програмування стало не лише необхідною навичкою для фахівців в ІТ-сфері, але й важливим інструментом для розвитку критичного, алгоритмічного та аналітичного мислення у студентів. Програмування дозволяє учням не лише зрозуміти принципи роботи комп'ютерів і програмних систем, але й формує вміння розв'язувати комплексні проблеми шляхом розробки алгоритмів. Тому навчання програмувати стає важливою складовою освіти, зокрема у профільній школі.

Вивчення актуальних наукових розвідок виявило низку наукових результатів, які висвітлюють концептуальні засади навчання програмувати (В. Биков, К. Гілберт, Е. Дейкстра, М. Жалдак, В. Конюшко, Б. Маккарті, Д. Рометті, Г. Шілдт та ін.). Проблемі освітніх прогалин у співпраці школи та вишу під час вивчення мов програмування та шляхам їхнього подолання присвячено дослідження Л. Гришко, В. Клочка, Н. Морзе, О. Овчарук, В. Ряжської та інших. Так, М. Жалдак описує невідповідність між обсягом знань з програмування, що вимагаються в закладах освіти, та обмеженим часом на їх здобуття і пропонує розв'язання проблеми за рахунок залучення активних методів навчання та зміни послідовності вивчення мов програмування [28]; І. Минтій пропонує для навчання програмувати використати теорію винахідницьких рішень – технологію творчості як прототип методичної системи [29]. Активізація пізнавальної діяльності в процесі навчання програмувати розглядається Ю. Машбіцем, В. Паламарчуком, О. Співаковським, Т. Шамовою та іншими. Зокрема, О. Співаковський пропонує діяльнісний підхід та активні методи навчання,

які призводять до підвищення мотивації до вивчення програмування, активізації пізнавальної діяльності учнів, розвитку вміння вчитися самостійно, формування навичок роботи в команді та регулювання самооцінки учнів [30]; Т. Вдовичин, І. Дединський, Л. Лазурчак та інші акцентували увагу на методі проєктів як ефективному способі розв'язування проблеми формування алгоритмічного мислення в молоді. С. Іщеряков, навпаки, наголошує на ефективності та результативності задачних підходів до навчання програмувати, що ґрунтуються на принципі «одна задача – одне рішення» [31]; Ю.Руденко з колегами розкриває особливості активного і інтерактивного навчання програмувати [32]; Ю. Машбіц, М. Лапчик, С. Овчаров та Ю. Плаксій присвятили свої праці проблемі вдосконалення підготовки вчителів інформатики в контексті навчання школярів програмуванню.

Дослідженням особливостей навчання основ програмування в системі загальної середньої освіти займаються також науковці: О. Тихоненко, який розглядає методичні підходи до формування та розвитку алгоритмічного мислення учнів на уроках інформатики [33]; В. Базурін – вивчає середовища програмування як засіб навчання школярів основ програмування [34]; О. Шаран, В. Шаран та М. Кулініч розглядають особливості використання електронних матеріалів у процесі розвитку алгоритмічного мислення учнів [35]; І. Васильків та Р. Романчук [36], які вивчають особливості формування алгоритмічного мислення учнів початкової школи на уроках інформатики. У роботі [37] описуються проблеми навчання програмувати учнів старших класів та шляхи їх подолання

Програмування є одним з ключових напрямів вивчення інформатики, який виходить на передній план у сучасній освітній системі. Вивчення програмування стає не лише важливим для розвитку компетентностей, але й формує основи критичного та алгоритмічного мислення, що є важливими результатами навчання учнів. Вивчення програмування тісно пов'язане з сучасними освітніми тенденціями, такими як інтеграція технологій в освітній

процес, проєктно-орієнтоване навчання та міждисциплінарність, а також індивідуалізація навчання.

З розвитком цифрових технологій програмування стало одним з ключових елементів у навчанні інформатики. Воно перетворилося з дисципліни для вузького кола спеціалістів на навичку, що є основою сучасної цифрової грамотності (табл. 1.1).

Таблиця 1.1

Програмування як центральний елемент інформатики

Роль програмування в сучасній економіці	У глобалізованому світі програмування стало універсальною мовою, яка використовується у різних сферах – від науки та медицини до бізнесу й дизайну. Вміння програмувати дозволяє фахівцям не тільки ефективніше вирішувати завдання, але й створювати інноваційні рішення, які можуть суттєво змінити спосіб, у який функціонують різні галузі. Відповідно, навчальні програми все більше інтегрують програмування, щоб забезпечити учнів необхідними навичками для успішної кар'єри у майбутньому.
Програмування як інструмент розвитку критичного та алгоритмічного мислення	Вивчення програмування сприяє розвитку алгоритмічного мислення, яке полягає у здатності розділяти складні проблеми на окремі кроки та шукати ефективні шляхи для їхнього вирішення. Це навичка, яка є корисною не тільки в ІТ, але й у будь-якій сфері діяльності. Крім того, програмування вчить критично оцінювати інформацію, аналізувати її та знаходити оптимальні рішення, що робить його важливим інструментом загального інтелектуального розвитку.

Популяризація програмування серед школярів	Програмування перестало бути прерогативою тільки університетських курсів або спеціалізованих шкіл. Сучасні шкільні програми все частіше включають курси з основ програмування для учнів різного віку. Зокрема, такі мови програмування, як Scratch для початкової школи або Python для середньої та старшої школи, допомагають учням освоїти основи логіки, умовних операторів та циклів у доступній формі.
Використання програмування для міждисциплінарних проєктів	У багатьох країнах програмування вже використовується для виконання міждисциплінарних проєктів, які охоплюють не лише інформатику, але й природничі науки, мистецтво, економіку та інші сфери. Це допомагає учням усвідомити, що програмування не є вузькоспеціалізованим предметом, а універсальним інструментом для вирішення складних проблем у різних галузях.

Вивчення програмування надає сучасним школярам важливі переваги, серед яких розвиток критичного й алгоритмічного мислення, здатність вирішувати складні задачі, структуровано підходячи до їх розв'язання, а також вміння працювати з технологіями, які стають основою сучасної економіки. Програмування сприяє розвитку креативності та інноваційного мислення, дозволяючи учням створювати власні проєкти та вирішувати реальні проблеми. Окрім того, ця навичка відкриває можливості для майбутньої кар'єри в різних галузях, не лише в IT, і робить школярів більш конкурентоспроможними на ринку праці.

У праці [38] приділяється увага роздумам про важливість знань та вмінь з програмування. Зважаючи на це, можна виділити три основні відповіді на запитання «Чому важливо вивчати програмування?» (рис. 1.1):

Рис. 1.1. Важливість вивчення програмування

– Сучасний ринок праці постійно змінюється, і більшість нових професій потребуватиме знань у галузі інформаційних технологій. Програмування є базовою компетенцією для багатьох професій, таких як розробка програмного забезпечення, аналіз даних, кібербезпека, автоматизація виробничих процесів тощо.

– Навчання програмуванню розвиває когнітивні навички, які є важливими в будь-якій сфері діяльності. Це вміння вирішувати проблеми, аналізувати інформацію, планувати свої дії та приймати рішення на основі логічних висновків.

– Вивчаючи програмування, учні отримують можливість створювати власні програми, вебсайти, ігри та додатки. Це відкриває перед ними нові можливості для самовираження та реалізації своїх ідей, що є важливим для мотивації до навчання.

Програмування тісно пов’язане з іншими темами інформатики, оскільки воно є основою для розуміння та реалізації багатьох аспектів цієї дисципліни. Одним із ключових зв’язків є алгоритми, які визначають кроки для вирішення

певної задачі. Програмування реалізує алгоритми на практиці, перетворюючи їх у програми, що працюють. Для ефективного зберігання та обробки даних програми використовують структури даних, такі як масиви, списки, дерева та графи, які дозволяють організовувати дані й виконувати на них операції швидко та ефективно. Бази даних, ще один важливий аспект інформатики, тісно інтегровані з програмуванням, оскільки саме через програми відбувається доступ до даних, їхній запис, редагування й пошук.

Мережі та програмування також взаємодіють через розробку протоколів та додатків, які забезпечують передачу даних між комп'ютерами, зокрема через вебдодатки, мережеві програми або API. У контексті кібербезпеки, програмування відіграє значну роль у створенні захищених програм і систем, що включає написання захищеного коду, виявлення вразливостей та розробку механізмів захисту. Отже, програмування виступає універсальним інструментом, який інтегрує в собі всі ці аспекти інформатики, забезпечуючи їхню практичну реалізацію та взаємодію.

1.2. Огляд сучасних мов програмування, які викладаються в школах

Сучасні мови програмування відіграють важливу роль у шкільному навчанні інформатики, оскільки вони не лише навчають учнів базовим принципам програмування, але й розвивають критичне та алгоритмічне мислення. Основні відмінності між різними мовами програмування полягають у їхніх синтаксичних правилах, рівні складності, галузі застосування та принципах роботи. Наприклад, Python відомий своїм простим і зрозумілим синтаксисом, що робить його популярним для початківців, оскільки він дозволяє швидко писати програми та концентруватися на логіці [11]. Натомість C++ або Java мають складніший синтаксис і вимагають детальнішого управління ресурсами комп'ютера, таких як пам'ять, що дозволяє створювати більш оптимізовані програми, але потребує більше часу для навчання. JavaScript відрізняється тим, що широко використовується для

створення веб-додатків і розширює можливості фронтенд-розробки, тоді як C# використовується в розробці ігор та додатків під платформу .NET.

Вибір мови програмування для вивчення в ЗЗСО залежить від кількох факторів (рис. 1.2).

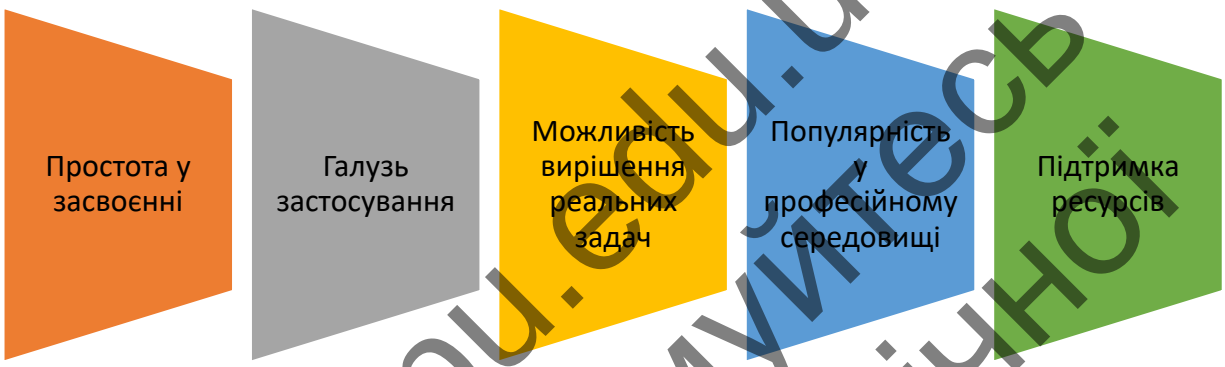

Рис. 1.2. Фактори вибору мов програмування

– Мови, такі як Python, часто обирають для навчання через їхній легкий для розуміння синтаксис і можливість швидко побачити результати своєї роботи.

– Якщо мета навчання полягає у веб-розробці, обирають мови, як-от HTML, CSS та JavaScript. Для вивчення основ програмування підходять універсальні мови на зразок Python або Java.

– Мови повинні дозволяти учням працювати з реальними проблемами. Python, наприклад, добре підходить для роботи з даними та штучним інтелектом, а Scratch використовується для візуалізації процесів і навчання дітей програмуванню через інтерактивні проєкти.

– Мови, що є популярними на ринку праці (як Python, JavaScript, C++), зазвичай обирають для того, щоб підготувати учнів до подальшої професійної діяльності в галузі програмування.

– Вибір мови також залежить від наявності навчальних матеріалів, підручників, інтерактивних платформ та інших освітніх ресурсів, що спрощують процес навчання.

Отже, вибір мови програмування в школі залежить від її простоти, галузі застосування, можливостей для вирішення практичних задач та відповідності сучасним вимогам ринку праці.

У школах України вивчаються кілька сучасних мов програмування, серед яких: Python, Scratch, C++, Java, JavaScript, Ruby та Pascal.

Python – одна з найпопулярніших мов для навчання програмуванню у школах. Розробка мови Python відбулася ще в кінці 1980-х років [5] співробітником голландського інституту CWI Гвідо ван Россумом. Її простий синтаксис та універсальність роблять Python ідеальною для початківців. Вона використовується для вивчення основ програмування, алгоритмів та роботи з даними, а також у навчальних проєктах зі штучного інтелекту та автоматизації [11].

Програми, написані цією мовою, вимагають встановлення на комп'ютер користувача відповідної версії інтерпретатора, тому існує досить велика кількість середовищ розробки програм для Python. Завдяки своїй лаконічності Python дозволяє опанувати синтаксис мови, а відсутність «успадкування» у вигляді аксіом, що формувалися протягом багатьох років, сприяє освоєнню об’єктно-орієнтованого програмування.

Мова програмування Python працює практично на всіх відомих платформах, від персональних комп'ютерів до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, HaikuOS, AS/400 і навіть OS/390, Windows Mobile, Symbian і Android [46]. Мова Python вирізняється легкістю інтеграції з наявними компонентами, що дає змогу впроваджувати Python у застосунки. Python відома своєю легкістю у навчанні, різноманітністю бібліотек та широким застосуванням у різних галузях – від веброзробки до машинного навчання.

Серед переваг Python [5]:

– швидке виконання програм;

– різні стилі програмування на Python: об'єктно-орієнтований, процедурний і функціональний;

– у стандартній бібліотеці Python є інструменти для роботи з електронною поштою, інтернет-протоколами, ftp, http, базами даних тощо.

– скрипти, написані Python, можуть виконуватися на більшості сучасних операційних систем. Така кросплатформна сумісність дає змогу використовувати Python у найрізноманітніших галузях;

– Python підходить для розв'язання великої кількості завдань (офісні додатки, веб-додатки тощо).

Легкість у вивченні, універсальність та високий попит на ринку праці роблять Python вдалим вибором для школярів. Зауважимо, що деякі підручники з інформатики рекомендують вивчати саме цю мову програмування (підручник з інформатики для учнів 10-11-х класів ЗЗСО за редакцією В.Д. Руденко, Н.В. Речич, В.О. Потієнко [22]).

Scratch – візуальна мова програмування, яка дозволяє учням створювати програми за допомогою блоків. Scratch використовується, здебільшого, у молодших класах для розвитку логічного мислення та введення в основи програмування через створення ігор, анімацій і простих додатків. У Scratch використовуються об'єкти-спрайти, якими керують самі учні за допомогою алгоритмів.

У Scratch є кілька типів блоків, які можна змусити працювати: рух, зовнішній вигляд, звук, перо (за допомогою Turtle Graphics), елементи керування, датчики, операції та змінні. Кожен блок має свої налаштування та параметри програмування для певних дій. Найпоширеніше застосування Scratch – навчання у вигляді створення мультфільмів та ігор. Крім програмування, Scratch можна використовувати для створення ілюстративного матеріалу на уроках історії, біології, фізики тощо. У Scratch є функція звукового редактора, яка відкриває можливість роботи з різними типами даних.

Проєкт зі створення Scratch ініційований у 2003 році за фінансової підтримки підприємств Science Foundation, Intel Foundation, Microsoft, MacArthur Foundation, LEGO Foundation, Code-to-Learn Foundation, Google, Dell, Fastly, Inversoft і MIT Media Lab research consortia [10]. Scratch був розроблений в 2007 році в лабораторії Lifelong Kindergarten Массачусетського технологічного інституту під керівництвом професора Мітчела Резника (Mitchel Resnick). У 2014 році було випущено версію Scratch для дітей молодшого віку під назвою ScratchJr. Це мобільний застосунок для Android та iOS, що дає змогу дітям маніпулювати спрайтами: у блоках не використовують текст, а набір доступних дітям дій обмежений (просте переміщення спрайта, робота зі звуком і маніпуляції із зображеннями).

Scratch послужив основою для кількох інших мов візуального програмування. Найвідоміша з них – Snap! і її головна особливість – можливість створювати власні блоки та використовувати об'єкти першого класу. Програмування Arduino також засноване на Scratch, що полегшує цей процес.

До переваг цього середовища можна віднести наочність створення та запуску програм, а також низький «поріг входження», тобто створювати програми відносно легко як для учнів з гуманітарними схильностями, так і для тих, хто має добре розвинені алгоритмічні навички. До недоліків варто віднести відсутність можливості створювати додатки, які працюють незалежно від середовища та на відносно низькому рівні (пряма взаємодія з файлами, пам'яттю, графічною підсистемою операційної системи тощо).

Мова ***C++*** класична мова програмування, що залишається актуальною для навчання складніших аспектів програмування та алгоритмів. C++ широко використовується в олімпіадних задачах з інформатики, оскільки дає змогу працювати з ефективними алгоритмами та керувати ресурсами комп'ютера. Програми, написані цією мовою, вимагають попередньої компіляції перед виконанням. C++ підтримує процедурне програмування, об'єктно-орієнтоване програмування та програмування загального призначення [3]. Мова має

досить велику стандартну бібліотеку, яка містить загальні контейнери, алгоритми введення/виведення, регулярні вирази та підтримку багатозадачності. Мова програмування C++ поєднує властивості як високорівневих, так і низькорівневих мов [2]. Має складніший синтаксис, порівняно з більш високорівневими мовами, і вимагає доброго розуміння концепцій управління пам'яттю, що може бути важким для новачків.

Мова програмування C++ була розроблена Б'ярне Строуструпом у 1983 році і до сьогодні є однією з найпопулярніших мов для розробки програмного забезпечення. Галузі застосування мови включають розробку операційних систем, різних додатків, драйверів пристроїв, вбудованих систем, високопродуктивних серверів і розважального програмного забезпечення.

Нині існує досить багато реалізацій мови програмування C++, як безкоштовних, так і комерційних, і для різних платформ. Як приклад можна навести GCC, Visual C++, Intel C++ Compiler і Embarcadero (Borland) C++ Builder [2]. Синтаксис мови програмування C++ успадкований від мови програмування C. Одним із принципів її розробки було збереження сумісності з попередником. Однак C++ не є розширеною версією C. Хоча велика кількість програм може бути однаково опрацьована компіляторами як C, так і C++. Доступ до функціональності стандартної бібліотеки C++ забезпечується включенням у програму відповідних стандартних заголовних файлів (за допомогою директиви #include) У стандарті C++ визначено загалом 79 таких файлів. Стандартні бібліотечні засоби оголошуються в просторі імен std [27].

Мова програмування C++ – дуже потужний інструмент для розроблення компільованого програмного забезпечення різної складності [27]. У мову вбудовано багато середовищ, призначених для розробки програмного забезпечення для конкретних платформ від мікроконтролерів до потужних серверів. Вивчення C++ в школах допомагає учням зрозуміти основи низькорівневого програмування, ефективне управління ресурсами та важливі концепції, такі як покажчики і динамічне виділення пам'яті.

Мова програмування ***Java*** – одна з найпопулярніших мов програмування у світі, яка використовується у школах для вивчення основ об’єктно-орієнтованого програмування. Java застосовується як у веброзробці, так для створення мобільних додатків, що робить її корисною для учнів, які планують працювати в IT. Мова Java розроблена компанією Sun Microsystems (пізніше придбаною Oracle) у 1991 році, але офіційна дата – 23 травня 1995 року. Спочатку мова називалася Oak і була розроблена для програмування побутової електроніки, але пізніше її перейменували на Java, і її почали використовувати для написання аплетів, додатків і серверного програмного забезпечення [1].

Мову Java була створена в рамках проєкту зі створення складного програмного можна використовувати на різних процесорах під різними операційними системами, без необхідності компіляції окремо для кожної архітектури [26].

Загалом, Java – це потужна, безпечна та масштабована мова програмування, яка підходить для широкого спектру задач, від мобільних додатків до великих серверних систем. Її універсальність, велика кількість інструментів і підтримка роблять її чудовим вибором як для навчання, так і для професійної кар’єри. Java забезпечує міцну базу для розуміння об'єктно-орієнтованого програмування та широко використовується в індустрії.

JavaScript – це високорівнева, інтерпретована мова програмування, яка є однією з основних технологій веброзробки, поряд з HTML та CSS. Це головна мова для фронтенд-розробки, яка дозволяє створювати динамічний контент та взаємодію на вебсторінках. JavaScript був створений Бренданом Айком у 1995 році. Спочатку мова називалася Mocha, потім LiveScript, і, нарешті, отримала назву JavaScript.

JavaScript особливо популярна серед розробників через свою універсальність. Її основні особливості включають:

– Подієво-орієнтована модель – JavaScript може реагувати на події (наприклад, кліки миші, натискання клавіш).

– Асинхронність – завдяки функціям зворотного виклику, промісам та async/await, JavaScript може обробляти асинхронні операції ефективніше.

– Динамічна типізація – дані типів змінних визначаються під час виконання програми.

– Вбудована підтримка DOM – JavaScript дозволяє взаємодіяти з DOM (Document Object Model), що дозволяє динамічно змінювати зміст та структуру вебсторінок.

Основна складність мови JavaScript полягає в тому, що хоча синтаксис є досить простим, концепції асинхронного програмування та робота з DOM можуть бути важкими для початківців. Крім того, різноманітні фреймворки та бібліотеки (наприклад, React, Angular, Vue) потребують додаткового часу на вивчення.

JavaScript має легкий для опанування синтаксис і дозволяє швидко побачити результати своїх дій, що робить його ідеальним для новачків, тому і для вивчення в школах. Володіння JavaScript відкриває багато можливостей у галузі веброзробки та інших сферах IT. Завдяки роботі з подіями та асинхронністю, учні розвивають здатність до структурованого мислення. Знання JavaScript дозволяє учням створювати власні інтерактивні вебсайти та проєкти, що стимулює креативність та підвищує мотивацію до навчання.

Ruby хоч не є найпопулярнішою для вивчення у закладах освіти мовою програмування, проте деякі навчальні програми використовують її через простоту та елегантність, особливо для навчання основ веброзробки і роботи з базами даних. Мова є інтерпретованою, повністю об'єктно-орієнтована мова програмування з сильною динамічною типізацією, що поєднує Perl-подібний синтаксис з об'єктно-орієнтованим підходом. Деякі риси запозичені з мов програмування Python, Lisp, Dylan і CLU. Кросплатформені реалізації інтерпретатора мови Ruby поширюються на умовах відкритого програмного забезпечення. У ньому можуть розібратися навіть люди, які не розуміють програмування.

Ruby – це ретельно збалансована мова. Розробник Юкіхіро Мацумото (також відомий як «Matz») об'єднав деякі зі своїх улюблених мов (Perl, Smalltalk, Eiffel, Ada і Lisp), щоб створити нову мову, що врівноважує парадигму функціонального програмування з принципами імперативного програмування. Розробник Ruby часто повторював, що «намагається зробити Ruby природньою, але не простою мовою, яка відображає життя» [16].

Ruby вважається однією з найкращих мов програмування для вивчення в якості першої мови програмування. Швидкий цикл розробки (edit-run-edit), використання інтерпретатора, початкова об'єктна орієнтація та нетипізовані змінні, які не потрібно оголошувати, – все це дає змогу учням зосередитися на загальних принципах програмування. Не менш важлива багатоплатформеність Ruby і той факт, що це вільно розповсюджуване програмне забезпечення. Ще один вагомий аргумент на користь Ruby – його практичне застосування в найрізноманітніших галузях [13].

Ruby вважається чудовим інструментом для навчання програмуванню, оскільки вона поєднує простоту та потужність, роблячи процес навчання захоплюючим та продуктивним.

Pascal – високорівнева мова програмування, створена з метою заохочення до розробки добре структурованого та організованого коду. Вона є відмінним інструментом для навчання основ програмування. Це одна з найбільш відомих мов програмування [6], які використовуються для навчання програмувати. Pascal є основою для багатьох інших мов, включаючи Ada, Modulus-2 та Delphi.

Мова Pascal була створена Ніклаусом Віртом між 1968 і 1969 роками після участі в роботі Комітету з розробки мовного стандарту Algol-68. Мова названа на честь французького математика, фізика, літератора і філософа Блеза Паскаля, який створив першу в світі механічну машину, здатну складати два числа. Перша публікація Вірта про цю мову датується 1970 роком. Представляючи Pascal, автор пояснює, як створити невелику, але ефективну

мову, яка впливає на стиль програмування та використовує структуроване програмування та структуровані дані [25].

Мова характеризується суворою типізацією і наявністю засобів структурного програмування. Pascal є піонером серед подібних мов. Крім суворої типізації, синтаксис розроблено таким чином, щоб звести до мінімуму двозначність, а сам синтаксис інтуїтивно зрозумілий [24].

Найвідомішою реалізацією Pascal, яка забезпечила широке поширення та розвиток мови, був Turbo Pascal від Borland, який перетворився на Object Pascal для DOS та Windows, і навіть Delphi, де були представлені більш важливі мовні розширення [8]. Більшість програмістів, які пишуть сьогодні програми мовою Pascal, використовують PascalABC. Згідно з аналізом науково-методичних джерел [8; 9], базовою платформою для навчання змістовної лінії основ алгоритмізації є процедурні мови програмування, зокрема Pascal, який залишається вдалим вибором для початківців.

Мова програмування ***Object Pascal (Delphi)*** бере свій початок від класичної мови Pascal. Це об'єктно-орієнтоване розширення мови Pascal, розроблене компанією Apple. Object Pascal має кілька діалектів, одним із яких є Delphi. Кожен діалект Object Pascal додає нові елементи і реалізує ці елементи трохи по-іншому.

Delphi є зареєстрованою торговою маркою в багатьох країнах, тому навіть після того, як компанія Borland (власник мови) змінила назву, у більшості країн-послідовників вона, як і раніше, називається Object Pascal. За словами представників Borland, основною причиною зміни назви стало те, що Object Pascal вважався застарілим. Деякі програмісти стверджували, що перейменування було антиконкурентним заходом, оскільки Object Pascal не може бути зареєстрований як торгова марка.

Object Pascal – це суворо типізована мова високого рівня, що підтримує структуроване та об'єктно-орієнтоване проєктування [18]. Переваги мови полягають у тому, що код легко читається, швидко компілюється і дає змогу використовувати кілька файлів модулів для модульного програмування.

Отже, у ЗЗСО для вивчення можна пропонуватися такі мови програмування Python, Scratch, C++, Java, JavaScript, Ruby, Pascal та Object Pascal. Аналіз мови програмування, які можуть бути рекомендовані для вивчення у шкільному курсі інформатики дає підстави охарактеризувати їх за певними параметрами (табл. 1.2).

Таблиця 1.2

Порівняльна характеристика мов програмування

Мова програму-вання	Тип мови	Тип виконання	Останнє велике оновлення	Розробник	Розширен-ня файлів	Система типів	Основні середовища розробки	Сфера застосування
Python	Інтерпретована, об'єктно-орієнтована	Інтерпретатор	Python 3.11 (жовтень 2022)	Python Software Foundation	.py	Динамічна	PyCharm, Visual Studio Code, Jupyter Notebook	Веброзробка, наукові обчислення, машинне навчання, автоматизація
Scratch	Візуальна, блокова	Інтерпретована	Регулярні оновлення	MIT Media Lab	.sb3	-	Веб-інтерфейс Scratch	Навчання програмування для дітей, створення простих ігор та анімацій
C++	Компільована, об'єктно-орієнтована	Компілятор	C++20 (2020)	ISO	.cpp, .h	Статична	Visual Studio, Code::Blocks, CLion	Системне програмування, розробка ігор, наукові обчислення
Java	Компільована, об'єктно-орієнтована	Віртуальна машина Java (JVM)	Java 19 (середина 2023)	Oracle Corporation	.java	Статична	Eclipse, IntelliJ IDEA, NetBeans	Веброзробка, розробка мобільних додатків, корпоративні системи
JavaScript	Інтерпретована, об'єктно-орієнтована	Браузер або Node.js	Регулярні оновлення	Ecma International	.js	Динамічна	Visual Studio Code, WebStorm	Фронтенд і бекенд веброзробка, розробка ігор
Ruby	Інтерпретована, об'єктно-орієнтована	Інтерпретатор	Ruby 3.2 (грудень 2022)	Matsumoto Yukihiro (Matz)	.rb	Динамічна	RubyMine, Visual Studio Code, Sublime Text	Веброзробка, автоматизація, DevOps
Pascal	Компільована, процедурна	Компілятор	Різні діалекти з різними датами випуску	Niklaus Wirth	.pas	Статична	Lazarus, Free Pascal	Навчання програмування, наукові обчислення
Object Pascal	Компільована, об'єктно-орієнтована	Компілятор	Різні діалекти з різними датами випуску	Anders Hejlsberg (Delphi)	.pas	Статична	Delphi, Lazarus	Розробка додатків для Windows, кросплатформна розробка

З табл. 1.? видно, що, незважаючи на те, в якому році було створено перші версії мов програмування, більшість із них затребувані й сьогодні.

Крім того, мови програмування оцінюються за статистикою ресурсів TIOBE, наведемо рейтинг мов [6] станом на жовтень 2024 року (рис. 1.3)

Oct 2024	Oct 2023	Change	Programming Language	Ratings	Change
1	1		Python	21.90%	+7.08%
2	3	↑	C++	11.60%	+0.93%
3	4	↑	Java	10.51%	+1.59%
4	2	↓	C	8.38%	-3.70%
5	5		C#	5.62%	-2.09%
6	6		JavaScript	3.54%	+0.64%
7	7		Visual Basic	2.35%	+0.22%
8	11	↑	Go	2.02%	+0.65%
9	16	⇈	Fortran	1.80%	+0.78%
10	13	↑	Delphi/Object Pascal	1.68%	+0.38%
11	9	↓	SQL	1.64%	-0.15%
12	14	↑	MATLAB	1.48%	+0.22%
13	20	⇈	Rust	1.45%	+0.53%
14	12	↓	Scratch	1.41%	+0.05%
15	8	⇊	PHP	1.21%	-0.69%
16	10	⇊	Assembly language	1.13%	-0.51%
17	17		R	1.09%	+0.12%
18	19	↑	Ruby	0.99%	+0.07%
19	24	⇈	COBOL	0.99%	+0.23%
20	15	⇊	Swift	0.98%	-0.09%

Рис. 1.3. Рейтинг популярності мов програмування за даними індексу TIOBE станом на жовтень 2024

Лідером рейтингу є мова Python (21,9%), другу позицію займає мова програмування C++ (11,6%), третє місце за мовою Java (10,51%). Далі JavaScript на шостій позиції (3,54%), Object Pascal (1,68%) утримує 10 місце, а Scratch (1,41%) і Ruby (0,99%) займають 14 і 18 місця відповідно. Станом на жовтень 2024 р. за останній рік мови C++, Java, Ruby та Object Pascal піднялися у рейтингу, а Scratch знизив свої позиції. Мови Python та JavaScript займають ті ж місця, що і в минулому році.

На рис. 1.4 відображено графік популярності деяких мов програмування за рейтинговою статистикою ресурсу TIOBE [6].

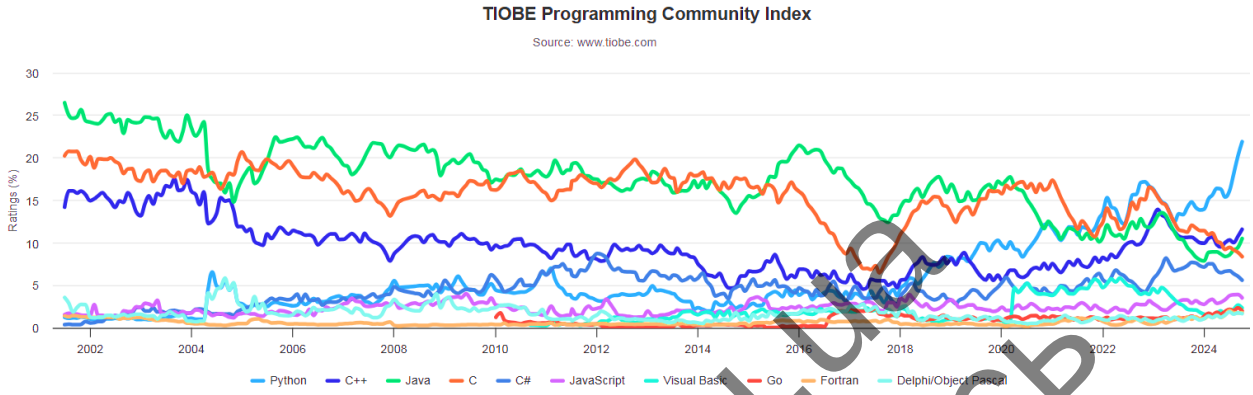

Рис. 1.4. Популярність найрентинговіших мов програмування (2002-2024 роки)

Бачимо, що майже до 2020 року постійним лідером серед мов програмування була мова Java. А до 2021 року ще й мова С. Але варто виділити швидке зростання рейтингу мови Python, яка, починаючи з 2018 року, вийшла на провідні позиції. Станом на січень 2020 року Python мав рейтинг 9,7%, а вже станом на сьогодні – 21,9%. Тому констатуємо популярність означених нами мов програмування –Python, C++ та Java, а для вивчення в школі – Scratch, Object Pascal, C++, Python, Java, JavaScript та Ruby.

Отже, вивчення розглянутих нами мов програмування, дає школярам можливість освоїти базові концепції програмування та розвивати алгоритмічне мислення. Кожна з цих мов має свої особливості та переваги, що робить їх корисними для різних навчальних цілей. Наприклад, Python забезпечує легкий вхід у світ програмування завдяки простому синтаксису, тоді як C++ і Java пропонують учням більш глибоке розуміння алгоритмів і структур даних. Інтеграція мов програмування в шкільний курс інформатики сприяє формуванню у молоді практичних навичок, необхідних для подальшої роботи у сфері інформаційних технологій та підвищує їх конкурентоспроможність на сучасному ринку праці.

Висновки до розділу 1

У першому розділі «Вивчення мов програмування у ЗЗСО» обґрунтовано важливість вивчення програмування молоддю та представлено короткий огляд популярних мов програмування для вивчення у ЗЗСО.

Показано, що програмування відіграє центральну роль у навчанні інформатики в ЗЗСО, оскільки воно слугує інструментом для розвитку критичного мислення, логіки та творчих здібностей учнів. Через програмування школярі вчаться вирішувати складні задачі, розуміти базові принципи роботи алгоритмів та структур даних, що є основою інформаційних технологій. Вивчення програмування сприяє також міждисциплінарній освіті, поєднуючи в собі елементи математики, логіки та навіть творчих підходів. У сучасній школі програмування є засобом формування цифрової грамотності, підготовки до сучасного ринку праці та сприяння розвитку креативного мислення, що робить його ключовою складовою освітнього процесу.

У контексті загальної середньої освіти вибір мов програмування є ключовим фактором, який впливає на формування алгоритмічного мислення. Обґрунтовано, що мови програмування Python, JavaScript, C++, Java та інші використовуються в освітньому процесі завдяки їхній простоті, гнучкості та поширеності у професійних середовищах. Подано короткий огляд мов Python, C++, Java, JavaScript та інших. Наведено їх порівняльні характеристики та рейтинг популярності.

РОЗДІЛ 2.
ВИВЧЕННЯ СТРУКТУР ДАНИХ У ЗЗСО

2.1. Структури даних для розв'язування алгоритмічних задач

Програмування є інструментом розв’язання алгоритмічних задач, які лежать в основі більшості обчислювальних процесів. Алгоритмічні задачі – задачі, які вимагають використання формалізованих методів для досягнення певного результату, і їх успішне розв’язання вимагає не лише чіткого розуміння логіки процесу, але й ефективної організації даних.

При розв'язанні алгоритмічних задач структура даних є одним із ключових елементів, який визначає ефективність та продуктивність програмних рішень. Структури даних – структури, які передбачають певну організацію та зберігання даних у пам'яті комп'ютера для ефективного доступу й оброблення. У процесі розробки алгоритмів вибір відповідної структури даних має важливе значення, оскільки він впливає на час виконання операцій та використання ресурсів. Не всі алгоритми можуть бути однаково ефективними при використанні будь-якої структури даних: наприклад, пошук або вставка елементів у масиві відбувається значно повільніше, ніж у хеш-таблиці, однак масив може бути оптимальним для лінійного сканування.

Крім того, структури даних дозволяють програмістам організовувати інформацію таким чином, щоб вона відповідала вимогам конкретної задачі. Так, якщо треба вирішити задачу з великою кількістю даних, де необхідно часто виконувати операції додавання або видалення елементів, то неправильний вибір структури даних може призвести до неефективності, коли програма працюватиме занадто повільно або займатиме багато пам'яті. Наприклад, алгоритми пошуку найкоротшого шляху в графах (як BFS або Dijkstra) вимагають використання спеціальних структур даних, таких як черги або купи, для досягнення оптимальних результатів. Це підкреслює, що правильний підбір структури даних є невід'ємною частиною проєктування ефективних алгоритмів.

Структура даних у програмуванні – це спосіб організації та зберігання інформації, щоб її можна було легко використовувати. Можна уявити собі структуру даних як контейнер, в якому зберігаються різні предмети (дані), і від того, як цей контейнер організований, залежить, наскільки швидко і зручно можна знайти, додати або змінити ці предмети. Наприклад, простий список покупок є структурою даних, де можна швидко знайти потрібний товар або додати новий.

Структури даних є фундаментальними концепціями в програмуванні, які дозволяють ефективно організувати та обробляти дані. Розглянемо кілька прикладів структур даних (рис. 2.1).

Рис. 2.1. Приклади структур даних

Масиви – найпростіша структура даних, яка складається з елементів, що зберігаються у послідовному порядку. Масиви використовуються для зберігання колекцій даних одного типу. Наприклад, масиви часто використовують для зберігання списків учнів або чисел.

Списки (Linked Lists) – динамічна структура даних, де кожен елемент містить дані та посилання на наступний елемент. Списки можуть бути однобічними або двобічними. Вони використовуються для реалізації таких структур, як черги, стеки та інші.

Стеки (Stacks) – структура працює за принципом LIFO (Last In, First Out), тобто останній доданий елемент видаляється першим. Стеки використовуються в алгоритмах рекурсії, для обробки виразів у компіляторах, та для зберігання історії перегляду сторінок у веббраузерах.

Черги (Queues) – працюють за принципом FIFO (First In, First Out), тобто перший доданий елемент видаляється першим. Черги використовуються для

управління завданнями у процесорах, у системах обробки подій та при організації черг завдань в операційних системах.

Хеш-таблиці (Hash Tables) – структура даних, яка дозволяє швидко знаходити, додавати та видаляти елементи за допомогою хеш-функцій. Використовується у базах даних для індексації, у реалізації асоціативних масивів (словників) та у кешування.

Дерева (Trees) – ієрархічна структура даних, де кожен елемент називається вузлом, а посилання між ними називаються гілками. Дерева використовуються для представлення ієрархій (наприклад, файлових систем), у пошукових алгоритмах та у базах даних.

Графи (Graphs) – структура складається з вузлів (вершин) та ребер (зв'язків між вершинами). Графи використовуються для моделювання складних структур, таких як соціальні мережі, маршрутизація в інтернеті та в задачах оптимізації.

Кожна з цих структур даних має свої переваги та недоліки, і вибір конкретної структури залежить від конкретних вимог задачі. У табл. 2.1 розглянуто кожну структуру даних відповідно до її особливості, переваг та недоліків.

Таблиця 2.1

Огляд структур даних

Структура даних	**Особливість**	**Переваги**	**Недоліки**
Масиви	Фіксований розмір, швидкий доступ до елементів за індексом	Простота реалізації, швидкий доступ до будь-якого елемента	Фіксований розмір, неефективне додавання/видалення елементів, якщо масив заповнений
Списки	Динамічна структура, де	Легке додавання/	Повільний доступ до елементів (потрібно

Структура даних	**Особливість**	**Переваги**	**Недоліки**
	кожен елемент посилається на наступний	видалення елементів, особливо в середині списку	проходити по списку від початку до необхідного елемента)
Стеки	Працюють за принципом LIFO	Прості у реалізації, ефективні для задач, пов'язаних з обробкою виразів та рекурсією	Обмежений доступ (тільки до верхнього елемента)
Черги	Працюють за принципом FIFO	Ефективні для управління чергами задач, обробки подій	Обмежений доступ (тільки до голови черги)
Хеш-таблиці	Використовують хеш-функції для швидкого доступу до елементів	Швидке знаходження, додавання та видалення елементів	Можливість колізій, потреба у добрій хеш-функції
Дерева	Ієрархічна структура	Ефективні для реалізації пошукових алгоритмів, представлення	Складні у реалізації, потребують більше пам'яті

Структура даних	Особливість	Переваги	Недоліки
		ієрархічних даних	
Графи	Складаються з вузлів і ребер	Можуть моделювати складні відносини та структури	Складність реалізації та обробки, потребують значних ресурсів

Отже, вивчення структур даних є важливою частиною програмування, оскільки допомагає зрозуміти, як створювати більш ефективні програми. Від того, яку структуру даних обрати для зберігання інформації, залежатиме, як швидко програма виконає завдання. Структури даних вчать мислити логічно, будувати правильні алгоритми, і краще розуміти, як працюють комп'ютерні програми на базовому рівні. Тому розуміння структур даних робить навчання програмуванню більш глибоким та цілісним.

2.2. Побудова структур даних (масиви, списки, дерева, графи)

Основи побудови структур даних охоплюють різноманітні способи організації та управління даними для ефективної обробки й зберігання інформації. Різні структури даних оптимізовані для певних типів задач і можуть суттєво впливати на продуктивність програм [7]. Розглянемо основи побудови структури даних таких як масиви, списки, дерева, графи.

Для побудови структури даних *Масиви* (Arrays) необхідно пройти кілька етапів, які визначають основні дії і правила, які створюються і використовуються масиви в програмуванні (рис. 2.2).

Рис. 2.2. Етапи побудови структури даних Масиви

Нижче описано базовий алгоритм для побудови масиву:

1. Першим кроком є визначення кількості елементів, які будуть зберігатися в масиві. Розмір масиву повинен бути відомий заздалегідь, оскільки масиви мають фіксовану довжину.

Приклад	Алгоритм
якщо ми знаємо, що потрібно зберігати 10 чисел, то розмір масиву буде 10.	Input: n (розмір масиву) Declare: array[n]

2. Після визначення розміру масиву, його потрібно ініціалізувати. Можна або оголосити масив без значень (пустий масив), або ініціалізувати його за замовчуванням певними значеннями (наприклад, всі елементи дорівнюють нулю).

Приклад	Алгоритм
масив цілих чисел з початковими значеннями: array[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}	For i = 0 to n-1: array[i] = 0 (або інше значення за замовчуванням)

3. Доступ до елементів масиву здійснюється через індекси, де кожен елемент має унікальний номер (починається з 0). Для звертання до елемента масиву достатньо вказати його індекс.

Приклад	Алгоритм
для отримання першого елемента масиву: element = array[0]	Input: i (індекс елемента) Output: array[i]

4. Можна додавати або змінювати значення елементів масиву за допомогою індексів. Присвоєння значень здійснюється за аналогією з доступом.

Приклад	Алгоритм
для встановлення значення в певний елемент масиву: array[2] = 5	Input: i (індекс), value (значення) array[i] = value

5. Для обробки всіх елементів масиву (наприклад, для виведення на екран або зміни значень) використовується цикл. Цей крок дозволяє виконати операції над кожним елементом.

Приклад	Алгоритм
щоб вивести всі елементи масиву: For i = 0 to n-1: Print array[i]	For i = 0 to n-1: Process array[i] (вивести, змінити тощо)

6. У фіксованому масиві неможливо додати нові елементи, оскільки його розмір є постійним. Якщо потрібно збільшити розмір масиву, необхідно створити новий масив більшого розміру та скопіювати в нього елементи старого масиву.

Приклад	Алгоритм
для збільшення масиву на 1 елемент: Create new_array[n+1] For i = 0 to n-1: new_array[i] = array[i]	Input: old_array, new_size Create: new_array[new_size] For i = 0 to old_size-1: new_array[i] = old_array[i]

7. Видалення елементів вимагає аналогічного підходу — створення нового масиву меншого розміру, з копіюванням всіх елементів, за винятком видаленого.

Приклад	Алгоритм
щоб видалити елемент з певного індексу: Create new_array[n-1] For i = 0 to index-1: new_array[i] = array[i] For i = index+1 to n-1: new_array[i-1] = array[i]	Input: index (індекс для видалення) Create: new_array[n-1] For i = 0 to index-1: new_array[i] = array[i] For i = index+1 to n-1: new_array[i-1] = array[i]

8. Після завершення роботи з масивом у деяких мовах програмування (наприклад, C або C++) потрібно звільнити пам'ять, зайняту масивом, щоб уникнути витоків пам'яті.

Приклад	Алгоритм

здійснюється через виклик функції звільнення пам'яті (наприклад, free() в C).	Free array

Масиви є простою, але потужною структурою даних, яка дозволяє ефективно зберігати й обробляти елементи за допомогою індексів. Однак, вони мають фіксований розмір, тому не завжди підходять для задач, де необхідно динамічно змінювати кількість елементів.

Структура даних *Списки* (Linked Lists) являє собою послідовність елементів (вузлів), де кожен вузол містить дані і вказівник (посилання) на наступний вузол. Списки є динамічними, тобто їх розмір може змінюватися під час виконання програми, на відміну від масивів, що мають фіксований розмір. Ось алгоритм побудови однозв'язного списку (рис. 2.3):

Рис. 2.3. Етапи побудови структури даних Списки

1. Вузол є основною складовою списку. Кожен вузол містить два елементи: дані – значення, яке потрібно зберігати в вузлі і посилання на наступний вузол – вказівник, що з'єднує вузли між собою.

У мовах програмування вузол найчастіше реалізується як клас або структура:

class Node:

data (зберігає значення)

next (вказує на наступний вузол або є null для останнього елемента)

Алгоритм	Create new_node: new_node.data = value new_node.next = null

2. Створюємо порожній список, де початково немає жодного вузла. Для цього використовується змінна, яка зберігає посилання на перший вузол (голову списку).

Якщо список порожній, то голова списку (head) дорівнює null.

Алгоритм	Initialize empty list: head = null

3. Додавання елементів може відбуватися кількома способами: на початок списку, в середину або в кінець списку. Розглянемо найбільш поширені методи:

3.1. Додавання елемента на початок списку. При додаванні на початок, новий вузол стає новою головою списку, а його next вказує на стару голову.

Якщо список порожній, то новий вузол просто стає першим елементом.

Алгоритм	Input: new_data Create new_node with new_data If head == null:

	head = new_node Else: new_node.next = head head = new_node

3.2. Додавання елемента в кінець списку. При додаванні до кінця списку, новий вузол приєднується до останнього вузла, а його next залишається порожнім (null).

Потрібно знайти останній вузол, і тільки після цього приєднати новий вузол.

Алгоритм	Input: new_data Create new_node with new_data If head == null: head = new_node Else: Set temp = head While temp.next != null: temp = temp.next temp.next = new_node

4. Видалення вузла з однозв'язного списку передбачає пошук вузла, який треба видалити, і зміщення посилання попереднього вузла так, щоб він вказував на вузол після видаленого.

Якщо видаляється голова списку, просто встановлюється нова голова.

Якщо видаляється будь-який інший вузол, потрібно змінити посилання попереднього вузла.

4.1. Видалення елемента з початку списку. Видаляється перший вузол (голова), і голова змінюється на наступний елемент.

Алгоритм	If head != null: head = head.next

4.2. Видалення елемента з середини або кінця списку. Необхідно знайти попередній вузол, вказівник якого слід змінити, і потім перетворити його next на наступний після видаленого вузла.

Алгоритм	Input: value_to_delete
	If head == value_to_delete:
	  head = head.next
	Else:
	  Set temp = head
	  While temp.next != value_to_delete:
	  temp = temp.next
	  temp.next = temp.next.next

5. Для перегляду всіх елементів списку необхідно пройтись по кожному вузлу, починаючи з голови, і вивести його значення.

Алгоритм	Set temp = head
	While temp != null:
	  Print temp.data
	  temp = temp.next

6. Для пошуку елемента в списку потрібно пройти по всіх вузлах, порівнюючи значення кожного вузла з шуканим елементом.

Алгоритм	Input: value_to_search Set temp = head While temp != null: If temp.data == value_to_search:

	Return temp (або індекс) temp = temp.next

7. Якщо в мові програмування потрібно звільняти пам'ять, необхідно проходити по кожному вузлу і видаляти його.

Алгоритм	Set temp = head While temp != null: next_node = temp.next Delete temp temp = next_node

Списки є гнучкою структурою даних, яка дозволяє динамічно змінювати розмір і ефективно додавати або видаляти елементи. Вони корисні, коли необхідно часто змінювати розмір структури або виконувати операції з елементами на початку або в середині списку. Хоча доступ до елементів є повільнішим порівняно з масивами, списки дозволяють уникати проблем з фіксованим розміром і можуть бути ефективні для певних типів задач.

Структура даних *Дерева* (Trees) – це ієрархічна структура, де кожен елемент (вузол) може мати кілька дочірніх елементів. Найбільш поширеним видом дерева є бінарне дерево, де кожен вузол має не більше двох дочірніх вузлів (лівий і правий). Наведемо алгоритм побудови та основні дії для бінарного дерева (рис. 2.4).

Рис. 2.4. Етапи побудови структури даних Дерева

Основними елементами бінарного дерева є вузол (Node) – базовий елемент дерева. Кожен вузол містить: дані (значення вузла); посилання на лівий дочірній вузол та посилання на правий дочірній вузол.

1. Вузол містить дані і два вказівники на лівого і правого нащадків (дочірні вузли). У бінарному дереві вузол може мати не більше двох дочірніх вузлів.

Алгоритм	Create new_node:
	new_node.data = value
	new_node.left = null # Лівий дочірній вузол
	new_node.right = null # Правий дочірній вузол

2. Початково дерево порожнє, тобто корінь дерева (root) дорівнює null. При додаванні першого елемента дерево починає будуватися від кореня.

Алгоритм	Initialize empty tree: root = null

3. Додавання вузлів у бінарне дерево зазвичай здійснюється за певними правилами, наприклад, для бінарного дерева пошуку (BST):

- Елементи менші за значення поточного вузла додаються в ліве піддерево.
- Елементи більші або рівні додаються в праве піддерево.

Алгоритм	Input: value
	If root == null:
	  root = new_node(value) # Створюємо перший вузол (корінь)
	Else:
	  Insert(root, value)
	
	Function Insert(node, value):
	  If value < node.data:
	  If node.left == null:
	  node.left = new_node(value) # Додаємо новий вузол в ліве піддерево
	  Else:
	  Insert(node.left, value) # Рекурсивний виклик для лівого піддерева
	  Else:
	  If node.right == null:
	  node.right = new_node(value) # Додаємо новий вузол в праве піддерево
	  Else:
	  Insert(node.right, value) # Рекурсивний виклик для правого піддерева

4. Перегляд вузлів дерева може виконуватись різними способами:

– Обхід у глибину (DFS) – прямий (Pre-order): спочатку вузол, потім ліве піддерево, потім праве; симетричний (In-order): спочатку ліве піддерево, потім вузол, потім праве піддерево; зворотний (Post-order): спочатку ліве і праве піддерево, потім вузол.

– Обхід у ширину (BFS): рівень за рівнем.

Алгоритм	Function InOrderTraversal(node): If node != null: InOrderTraversal(node.left) Print(node.data) InOrderTraversal(node.right)

5. Пошук у бінарному дереві пошуку (BST) ґрунтується на порівнянні значення з вузлом:

– Якщо шукане значення менше за значення вузла, рухаємось в ліве піддерево.

– Якщо більше – в праве піддерево.

Алгоритм	Function Search(node, value): If node == null: Return False # Елемент не знайдено If value == node.data: Return True # Елемент знайдено If value < node.data: Return Search(node.left, value) # Пошук в лівому піддереві Else: Return Search(node.right, value) # Пошук в правому піддереві

6. Видалення елемента в бінарному дереві пошуку залежить від того, який вузол треба видалити:

– Вузол без дітей: просто видаляємо вузол.

– Вузол з одним дочірнім вузлом: видаляємо вузол і з'єднуємо його дитину з батьківським вузлом.

– Вузол з двома дочірніми вузлами: знаходимо найменший елемент у правому піддереві, замінюємо значення вузла на нього, а потім видаляємо цей найменший вузол.

Алгоритм	Function Delete(node, value):
	If node == null:
	Return node # Якщо вузол не знайдено
	If value < node.data:
	node.left = Delete(node.left, value) # Рекурсивно видаляємо з лівого піддерева
	Elself value > node.data:
	node.right = Delete(node.right, value) # Рекурсивно видаляємо з правого піддерева
	Else:
	If node.left == null:
	Return node.right # Якщо немає лівого вузла, повертаємо правий
	Elself node.right == null:
	Return node.left # Якщо немає правого вузла, повертаємо лівий
	Temp = FindMin(node.right) # Знаходимо найменший вузол у правому піддереві
	node.data = Temp.data # Замінюємо дані вузла
	node.right = Delete(node.right, Temp.data) # Видаляємо найменший вузол
	Return node

Дерева є потужною структурою даних, яка дозволяє ефективно виконувати різноманітні операції, такі як додавання, пошук і видалення елементів. Бінарні дерева пошуку (BST) забезпечують швидкий доступ до даних завдяки своєму ієрархічному характеру, де кожен вузол має не більше двох дочірніх елементів, а розміщення елементів визначається правилами порівняння.

Алгоритми для роботи з *графами*:

1. Обхід в глибину (DFS - Depth First Search). Обхід графа, який починається з вибраної вершини і досліджує якомога глибше, перш ніж переходити до інших вершин.

Алгоритм	

Стартуємо з будь-якої вершини. Відвідуємо вершину і позначаємо її як відвідану. Переходимо до однієї з невідвіданих суміжних вершин. Рекурсивно повторюємо ці дії для кожної вершини. Якщо вершина не має невідвіданих сусідів, повертаємось до попередньої вершини.	# Алгоритм DFS
	def dfs(graph, vertex, visited=None):
	if visited is None:
	visited = set()
	visited.add(vertex)
	print(vertex, end=" ")
	for neighbor in graph[vertex]:
	if neighbor not in visited:
	dfs(graph, neighbor, visited)
	# Приклад використання DFS
	graph = {
	0: [1, 2],
	1: [2],
	2: [0, 3],
	3: [3]
	}
	dfs(graph, 0) # Починаємо з вершини 0

2. Обхід в ширину (BFS - Breadth First Search). Обхід графа рівень за рівнем. Спочатку відвідуються всі сусіди вершини, а потім сусіди їхніх сусідів.

Алгоритм	

Стартуємо з вибраної вершини і додаємо її до черги.
Виймаємо вершину з черги, позначаємо її як відвідану і виводимо.
Додаємо до черги всіх невідвіданих сусідів цієї вершини.
Повторюємо ці дії, поки черга не стане порожньою.
	from collections import deque
	
	# Алгоритм BFS
	def bfs(graph, start_vertex):
	visited = set()
	queue = deque([start_vertex])
	
	while queue:
	vertex = queue.popleft()
	if vertex not in visited:
	print(vertex, end=" ")
	visited.add(vertex)
	queue.extend([neighbor for neighbor in graph[vertex] if neighbor not in visited])
	
	# Приклад використання BFS
	graph = {
	0: [1, 2],
	1: [2],
	2: [0, 3],
	3: [3]
	}
	
	bfs(graph, 0) # Починаємо з вершини 0

Графи є фундаментальною структурою даних, яка моделює зв'язки між об'єктами. Основні способи представлення графів – це матриця суміжності та список суміжності. Операції, такі як додавання вершин і ребер, а також обхід графа (DFS, BFS), є базовими алгоритмами, що широко використовуються в алгоритмічних задачах.

Алгоритми, які описані вище, написані у псевдокоді. Псевдокод – це спосіб опису алгоритмів за допомогою простих, неформальних інструкцій, що робить його зрозумілим для людей, незалежно від того, яку мову програмування вони використовують. Псевдокод не дотримується правил жодної конкретної мови програмування, але використовує структури, які можуть бути легко адаптовані до більшості мов, таких як Python, Java, C++, і т.д.

Якщо ці алгоритми потрібно реалізувати в конкретній мові програмування, то їх можна переписати, враховуючи синтаксис і особливості цієї мови. Так, узагальнимо особливості побудови структур даних мовою програмування Python.

1. Масиви (Arrays) на Python

Масиви в Python реалізовані через list, який є динамічною структурою. Ось основні дії з масивами:

```
# Створення масиву
array = [0] * 10  # Масив з 10 елементів, ініціалізованих нулями

# Доступ до елемента масиву
element = array[0]  # Отримання першого елемента

# Присвоєння значення елементу масиву
array[2] = 5  # Присвоєння значення 5 елементу на позиції 2
```

```
# Виведення всіх елементів масиву
for i in range(len(array)):
    print(array[i])

# Додавання елемента в кінець масиву
array.append(7)

# Видалення елемента з масиву
del array[2]  # Видаляємо елемент на позиції 2
```

2. Списки (Linked Lists) на Python

Однозв'язний список потребує створення класу для вузла і списку. Ось приклад реалізації:

```
# Вузол списку
class Node:
    def __init__(self, data=None):
        self.data = data  # Дані у вузлі
        self.next = None  # Вказівник на наступний вузол

# Однозв'язний список
class LinkedList:
    def __init__(self):
        self.head = None  # Початково список порожній

    # Додавання елемента на початок списку
    def add_to_start(self, data):
        new_node = Node(data)  # Створюємо новий вузол
        new_node.next = self.head  # Новий вузол вказує на стару голову
```

```
        self.head = new_node  # Голова тепер новий вузол

    # Додавання елемента в кінець списку
    def add_to_end(self, data):
        new_node = Node(data)
        if self.head is None:  # Якщо список порожній
            self.head = new_node
        else:
            current = self.head
            while current.next:  # Пошук останнього вузла
                current = current.next
            current.next = new_node  # Додаємо новий вузол в кінець

    # Видалення елемента з початку списку
    def delete_from_start(self):
        if self.head:  # Якщо список не порожній
            self.head = self.head.next  # Змінюємо голову списку

    # Видалення елемента з кінця списку
    def delete_from_end(self):
        if self.head is None:  # Якщо список порожній
            return
        if self.head.next is None:  # Якщо у списку тільки один елемент
            self.head = None
            return
        current = self.head
        while current.next.next:  # Пошук передостаннього елемента
```

```
            current = current.next
        current.next = None  # Останній елемент видаляється

    # Перегляд списку
    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        print(elements)
```

3. Бінарне дерево на Python

Бінарне дерево – це структура даних, де кожен вузол має не більше двох дочірніх вузлів: лівий і правий.

```
# Клас для вузла бінарного дерева
class Node:
    def __init__(self, value):
        self.value = value  # Дані вузла
        self.left = None  # Лівий дочірній вузол
        self.right = None  # Правий дочірній вузол

# Клас для бінарного дерева
class BinaryTree:
    def __init__(self):
        self.root = None  # Ініціалізуємо корінь як порожній

    # Додавання вузла в дерево
```

```
    def add(self, value):
        if self.root is None:
            self.root = Node(value)  # Якщо дерево порожнє, створюємо корінь
        else:
            self._add(self.root, value)

    # Рекурсивна функція для додавання вузла
    def _add(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = Node(value)  # Додаємо новий вузол зліва
            else:
                self._add(node.left, value)  # Рекурсивно додаємо в ліве піддерево
        else:
            if node.right is None:
                node.right = Node(value)  # Додаємо новий вузол справа
            else:
                self._add(node.right, value)  # Рекурсивно додаємо в праве піддерево
    # Обхід дерева (симетричний - InOrder)
    def inorder_traversal(self, node):
        if node is not None:
            self.inorder_traversal(node.left)  # Обхід лівого піддерева
            print(node.value, end=" ")  # Виведення значення вузла
            self.inorder_traversal(node.right)  # Обхід правого піддерева
```

4. Граф (Список суміжності) на Python

Граф – це структура даних, яка представляє зв'язки між об'єктами. Граф може бути орієнтованим або неорієнтованим.

```
# Клас для представлення графа за допомогою списку суміжності
class Graph:
    def __init__(self):
        self.graph = {}  # Ініціалізуємо порожній граф

    # Додавання вершини до графа
    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []  # Додаємо нову вершину зі списком сусідів

    # Додавання орієнтованого ребра між вершинами
    def add_edge(self, vertex1, vertex2):
        if vertex1 in self.graph:
            self.graph[vertex1].append(vertex2)  # Додаємо вершину2 до
списку сусідів вершини1

    # Обхід графа (DFS - Обхід в глибину)
    def dfs(self, vertex, visited=None):
        if visited is None:
            visited = set()
        visited.add(vertex)
        print(vertex, end=" ")

        for neighbor in self.graph[vertex]:
            if neighbor not in visited:
```

```
                self.dfs(neighbor, visited)


        # Обхід графа (BFS - Обхід в ширину)
        def bfs(self, start_vertex):
            visited = set()
            queue = [start_vertex]

            while queue:
                vertex = queue.pop(0)
                if vertex not in visited:
                    print(vertex, end=" ")
                    visited.add(vertex)
                    queue.extend([neighbor for neighbor in self.graph[vertex] if
neighbor not in visited])

        # Виведення графа
        def display(self):
            for vertex in self.graph:
                print(f"{vertex}: {self.graph[vertex]}")
```

Узагальнення побудови можна сконцетрувати в табл. 2.2.

Таблиця 2.4

Основи побудови структур даних

Масиви (Arrays)	**Списки**	**Бінарне дерево**	**Графи**
Створення масиву	Для реалізації списку	Створюється клас вузла, де	Граф представлений як

відбувається за допомогою списків (list). Основні операції (додавання, доступ, видалення) виконуються через стандартні методи списків.	створюється клас Node для вузлів і клас LinkedList для управління списком. Можливі операції: додавання на початок і в кінець, видалення з початку і кінця, виведення елементів списку.	зберігається значення вузла, а також посилання на лівого і правого нащадків. Додавання нових вузлів здійснюється рекурсивно, залежно від їх значення (менші значення йдуть у ліве піддерево, більші – в праве). Симетричний обхід дерева (InOrder) виводить вузли у порядку зростання.	словник, де ключами є вершини, а значеннями – списки сусідів кожної вершини. Додавання вершин і ребер дає змогу будувати граф, а обхід в глибину (DFS) і в ширину (BFS) дозволяють досліджувати граф.

Отже, структури даних відіграють ключову роль у програмуванні та алгоритмічних задачах, дозволяючи ефективно організовувати, зберігати та обробляти інформацію. Масиви та списки забезпечують базову, але ефективну схему зберігання послідовних даних, що робить їх корисними для простих задач з лінійною структурою. Дерева надають гнучкішу ієрархічну організацію даних, що зручно для моделювання відносин типу батько-нащадок, а графи представляють складні зв'язки між об'єктами, використовуючи їх для побудови маршрутів, мереж та рішень, пов'язаних із взаємодією множинних об'єктів. Кожен тип структури даних має свої переваги і недоліки, і вибір відповідної структури залежить від конкретних задач, що стоять перед програмістом або дослідником.

Висновки до розділу 2

У другому розділі «Вивчення структур даних у ЗЗСО» проведено аналіз найбільш поширених структур даних та представлено особливості їхньої побудови.

Показано, що вивчення структур даних є важливою частиною програмування, оскільки допомагає зрозуміти, як створювати більш ефективні програми. Від того, яку структуру даних обрати для зберігання інформації, залежатиме, як швидко програма виконає завдання. Структури даних вчать мислити логічно, будувати правильні алгоритми, і краще розуміти, як працюють комп'ютерні програми на базовому рівні. Тому розуміння структур даних робить навчання програмуванню більш глибоким та цілісним

Розглянуто основи побудови таких структур даних як масиви, списки, дерева, графи.

Розділ 3.
МЕТОДИЧНІ ОСОБЛИВОСТІ ВИВЧЕННЯ ТЕМИ «МОВА ПРОГРАМУВАННЯ ТА СТРУКТУРА ДАНИХ» В УМОВАХ ПРОФІЛЬНОЇ ШКОЛИ

3.1. Профільна школа та особливості навчання інформатики у ній

Профільна освіта в Україні впроваджується в старших класах середньої школи (10-11 класи) і передбачає вибір навчальних предметів за інтересами. Профільна освіта – це різновид диференційованої освіти, що враховує освітні потреби, нахили та здібності учнів і створює умови для навчання старшокласників відповідно до їхнього професійного самовизначення, що забезпечується зміною цілей, змісту, структури та організації освітнього процесу [39].

Профільне навчання в Україні впроваджується та унормовується різними законодавчими документами, серед яких: Закон України "Про освіту" [40]; Закон України "Про загальну середню освіту" [41]; Національна доктрина розвитку освіти [42]; Державний стандарт базової і повної загальної середньої освіти [43] та ін. Згідно із Концепцією профільного навчання у старшій школі [44] профільне навчання тлумачиться як вид диференціації та індивідуалізації освіти, який за рахунок зміни структури, змісту та організації освітнього процесу дає змогу повною мірою враховувати інтереси, нахили, можливості учнів, їхні здібності та створювати умови для навчання відповідно до навчально-професійних інтересів, намірів соціального та професійного самовизначення старшокласників.

Цілі профільної освіти є (рис. 3.1).

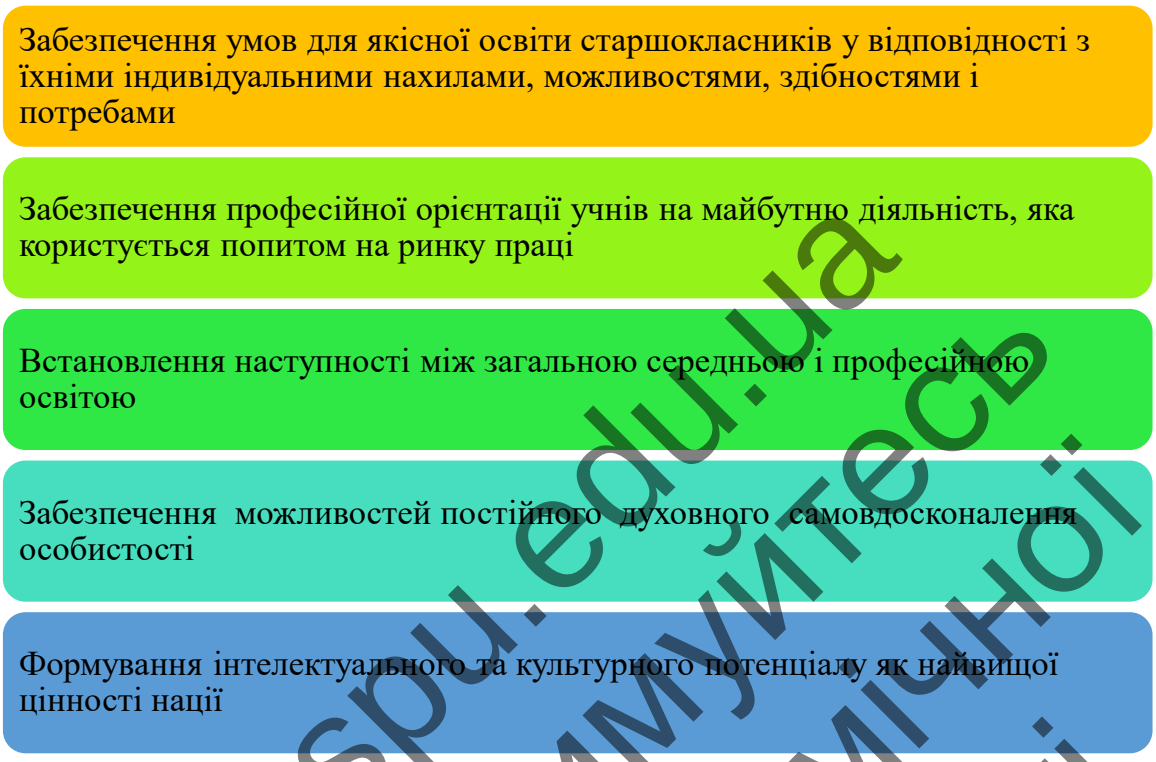

Рис. 3.1. Цілі профільного навчання

Основними завданнями профільної освіти є:

– створити умови для врахування та розвитку навчальних, пізнавальних і професійних інтересів, нахилів, здібностей і потреб старшокласників у процесі загальної освіти;

– забезпечення наступності між загальною середньою та професійною освітою, забезпечення можливості професійного працевлаштування;

– сприяти професійній орієнтації та самовизначенню старшокласників, соціалізації учнів незалежно від місця проживання, стану здоров'я тощо;

– проводити психолого-педагогічну діагностику для визначення готовності до прийняття самостійних рішень, пов'язаних із професійним розвитком;

– сприяння розвитку творчої самостійності, системи ідей, ціннісних орієнтацій, дослідницьких навичок і здатності надати випускникам школи можливості для успішної самореалізації;

– продовження цілісного розвитку учня як цілісної особистості, здібностей, талантів, духовності та культури; формування громадянина України, здатного робити усвідомлений соціальний вибір.

Профільне навчання базується на предметах двох типів – базові і профільні [45]. Базові предмети є обов'язковими для учнів усіх профілів (постійний елемент). Ці предмети реалізують цілі та завдання загальної середньої освіти. Зміст освіти та освітні вимоги до старшокласників визначаються Державними стандартами основної та повної загальної середньої освіти. У них виокремлено шість базових предметів (українська мова та література, іноземні мови, історія України та всесвітня історія, математика, природознавство та фізична культура), на які в 10 та 11 класах відводиться три години на тиждень.

Профільні предмети – це предмети, що реалізують мету, завдання та зміст кожного профілю. Спеціалізовані предмети передбачають більш повне освоєння концепцій, законів і теорій, використання інноваційних методів навчання, організацію дослідницької та проєктної діяльності, а також практику професійного навчання учнів. Профільні предмети також забезпечують прикладну спрямованість навчання за рахунок інтеграції когнітивних знань і методів та їхнього застосування в різних сферах діяльності, включно з професіями. Вони обираються з переліку, встановленого Міністерством освіти і науки України.

Профільні предмети можуть вивчатися по 5-10 годин на тиждень у 10 та 11 класах, залежно від кількості обраних учнем профільних предметів. Кількість навчальних годин може бути збільшена шляхом додавання додаткових годин до навчального плану.

Аналіз розподілу навчального часу з предмета «Інформатика» показує такі результати. У природничо-наукових фізико-математичних, екологічних,

географічних, біолого-фізичних, біолого-хімічних, усіх галузях суспільних наук, лінгвістиці, спорті та однойменній технічній галузі на вивчення предмета в 10 та 11 класах відводиться 1 година на тиждень. На математичну, фізичну, біоінженерну, хіміко-технічну, агрохімічну, фізико-хімічну, художньо-естетичну галузі природничих наук та математику відводиться одна година на тиждень у 10 класі та дві години на тиждень в 11 класі. Найбільша кількість годин на вивчення інформатики припадає на інформаційні технології в технічних дисциплінах – по п'ять годин на тиждень.

Майже всі профілі відводять однакову кількість годин на вивчення інформатики. Виняток становлять деякі профілі з природничих наук і математики, на які відводиться більше годин в 11 класі. Здебільшого це пов'язано з необхідністю вивчення та використання інформаційних технологій, що базуються на математичному моделюванні та обчисленнях. Збільшення годин на інформатику в художньо-естетичному профілі пов'язане з необхідністю вивчення тем, пов'язаних із використанням комп'ютерної графіки та вебдизайну.

У старших класах (усі дисципліни, крім «Інформаційних технологій») інформатика викладається на двох рівнях: стандартному (одна година на тиждень у 10 та 11 класах) та предметному (одна година на тиждень у 10 класі та дві години на тиждень у 11 класі). Навчальний план для вивчення інформатики на кожному з цих рівнів був затверджений наказом МОН.

Профільна школа надає можливість учням поглиблено вивчати певні дисципліни, зокрема інформатику, що є особливо актуальним у сучасному світі, де інформаційні технології стають невід'ємною частиною кожної сфери життя. Вивчення інформатики у профільній школі дозволяє учням не лише опанувати базові навички програмування, алгоритмічного мислення та роботи з даними, але й підготуватися до подальшої професійної діяльності в IT-сфері. Специфіка навчання інформатики у профільних класах полягає в акценті на більш складні теми, такі як алгоритми, структури даних, сучасні мови

програмування, що сприяє розвитку ключових компетенцій учнів, необхідних для майбутньої професійної підготовки.

Отже, вибір профілю навчання дозволяє учням зосередитися на тих дисциплінах, які є найбільш важливими для їхнього подальшого розвитку, і забезпечує глибші знання та навички в обраній галузі. Особливості профільного навчання включають індивідуальний підхід до кожного учня, більш інтенсивне вивчення профільних предметів, а також використання сучасних методик і технологій викладання. Це дозволяє створити умови для ефективного навчання та підготовки до вступу у заклади вищої освіти.

Розглянемо надалі навчальні програми предмету інформатика для вивчення у старших класах ЗЗСО. Нині в старших класах ЗЗСО існує дві навчальні програми вибірково-обов'язкового курсу «Інформатика» [48]: стандартна та профільна. Стандартний рівень вивчення інформатики в 10-11х класах складається з двох частин: базового модуля та елективного (варіативного) модуля. Базовий модуль є основою навчання, його зміст може бути розширено за рахунок елективних модулів. Вивчаючи розділи базового модуля, учні завершують формування предметних і ключових компетентностей у галузі використання сучасних ІКТ на рівні, визначеному чинними базовим і повним державними стандартами загальної середньої освіти [19]. Базовий модуль є основою інформатики в старшій школі.

Елективні (варіативні) модулі розширюють курс інформатики. Вчителі вільні у виборі, зважаючи на освітню програму закладу освіти, потреби дітей, їхні індивідуальні інтереси та здібності, особливості району проживання, навчальні й технічні ресурси, наявні в кабінеті інформатики, та необхідне програмне забезпечення.

Профільне навчання в програмах з інформатики може бути реалізовано двома способами: шляхом розширення змісту окремих тем або шляхом вибору інших профільних навчальних завдань.

Поєднання двох модулів викладання інформатики в 10-11 класах забезпечує гнучкість і свободу вибору та редагування матеріалів, необхідних для досягнення цілей навчання та виховання учнів.

Особливості обох навчальних програм з інформатики для учнів 10-11 класів ЗЗСО подано на рис. 3.2-3.3.

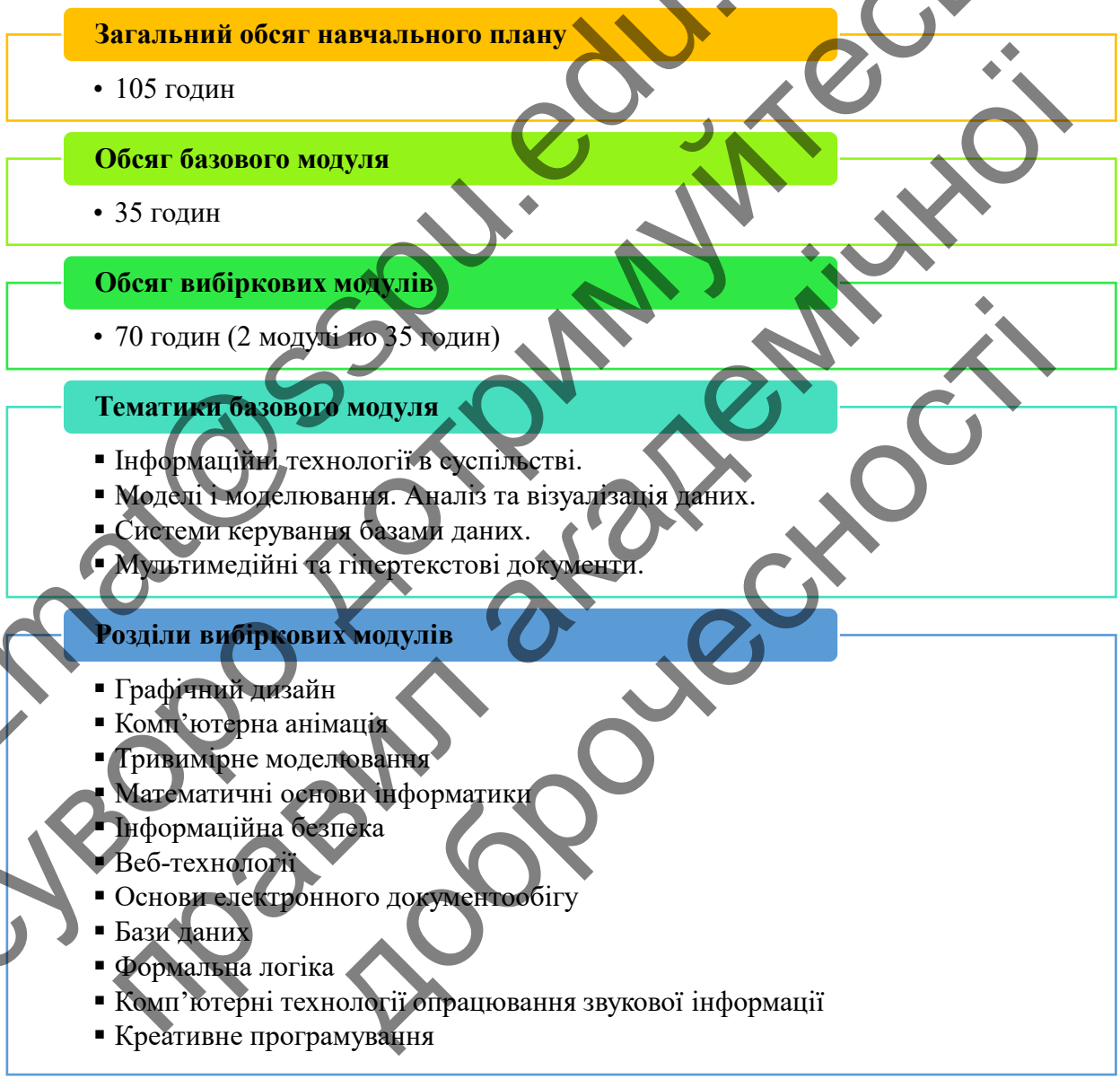

Рис. 3.2. Характеристика навчальних програм з інформатики (рівень стандарт)

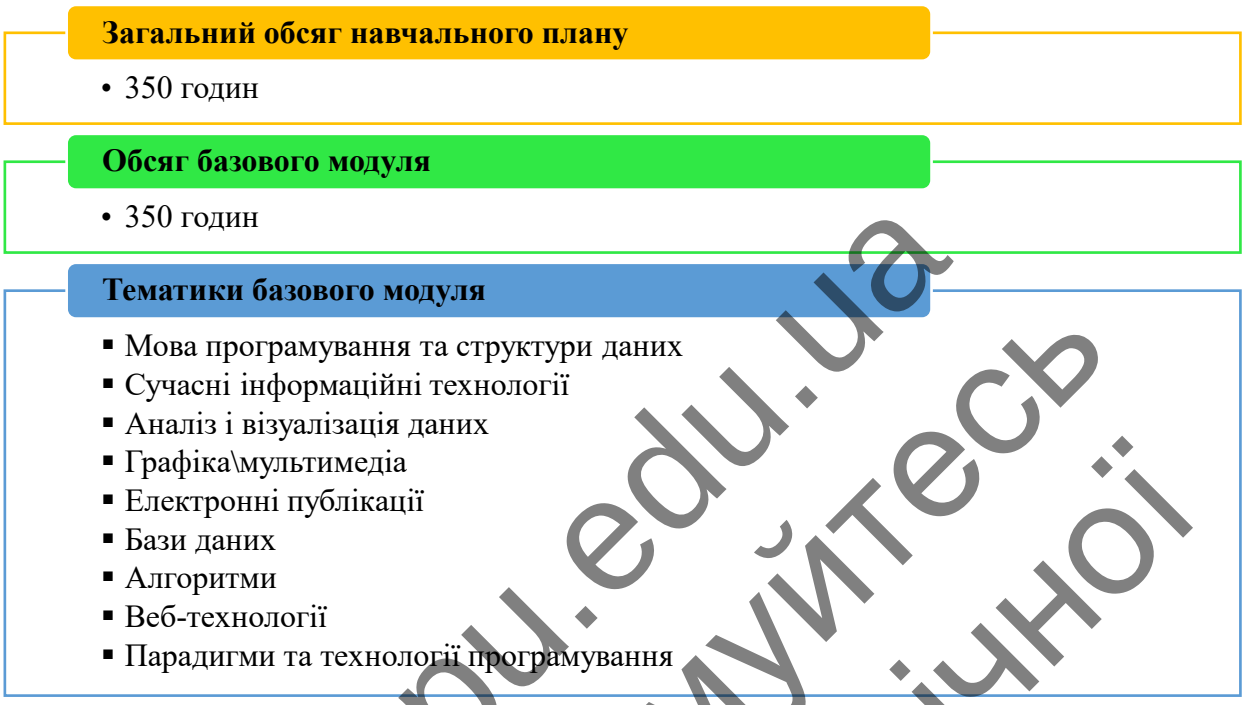

Рис. 3.3. Характеристика навчальних програм з інформатики (профільний рівень)

Згідно з рис. 3.2-3.3 бачимо, що існує значна різниця в обсязі викладання інформатики на обох рівнях. Це пов'язано з тим, що на профільному рівні інформатика вивчається п'ять годин на тиждень, або 175 годин на рік, у той час як на стандартному рівні інформатика вивчається лише одна година на тиждень у 10 класі та дві години на тиждень в 11 класі.

На профільному рівні немає модулів за вибором, а всі основні теми з інформатики вивчаються на рівні стандарт.

Що стосується списку тем у двох навчальних програмах, то слід зазначити, що деякі теми дублюють одна одну, а деякі взагалі не вивчаються. Наприклад, розділ «Інформаційні технології в суспільстві» базового модуля рівня стандарт має певну схожість з розділом «Сучасні інформаційні технології» профільного рівня; розділ «Аналіз та візуалізація даних» присутній взагалі в обох навчальних програмах; «Системи керування базами даних» базового модуля рівня стандарт є певним прототипом вибіркового модуля «Бази даних» цього ж рівня і розділу «Бази даних» профільного рівня.

На рівні стандарту елементи програмування та структур даних вивчаються у вибірковому модулі «Креативне програмування» [48]. На профільному рівні [47] вивчення цих тем відбувається відразу в трьох розділах: «Мова програмування та структури даних», «Алгоритми», «Парадигми та технології програмування».

Прикметно, що у змісті матеріалу розділу, що стосується вивчення програмування не вказано конкретної мови програмування. Це означає, що вибір мови залишається за вчителем, і він готує конспекти уроків та інструкції для лабораторних робіт відповідно до обраної мови та змісту матеріалу, зазначеного в навчальній програмі.

Розглянемо детально зміст теми у навчальній програмі для вивчення інформатики у 10-11 класах профільної школи [47]. Так, у 10 класі починається навчальний рік з вивчення розділу «Мова програмування та структури даних». Цей розділ налічує 18 тем для вивчення, серед яких: мова програмування; класифікація та складові мов програмування; особливості середовища розробки та ін. (рис. 3.4).

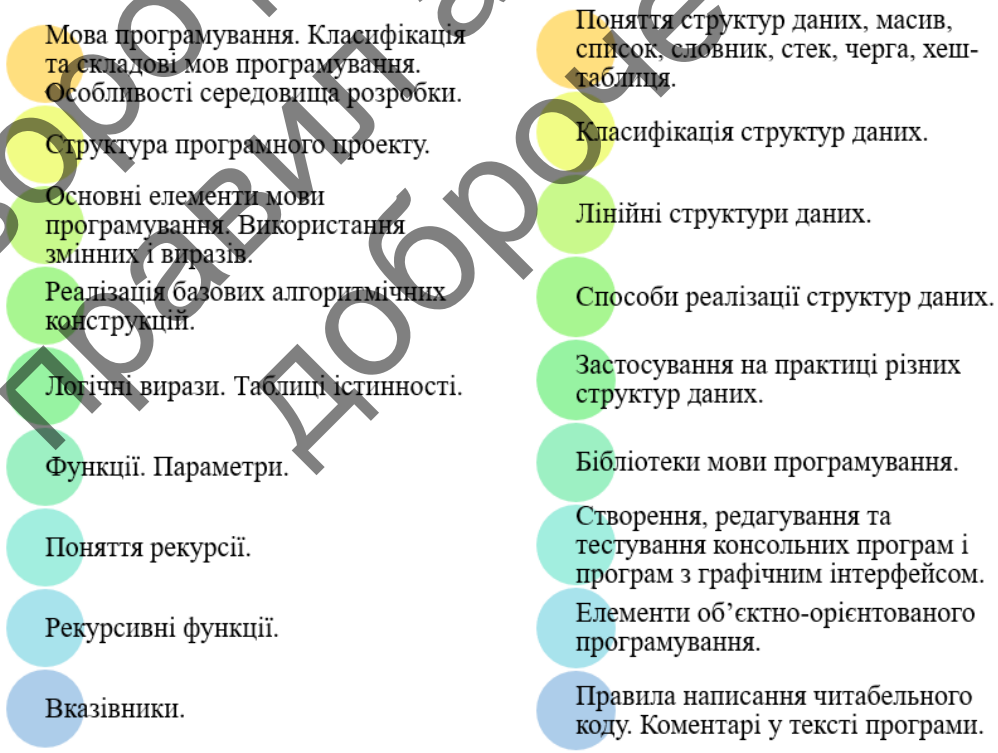

Рис. 3.4. Зміст розділу «Мова програмування та структури даних»

Відповідно до навчальної програми, після опрацювання цього розділу діти повинні розуміти призначення мов програмування та їхні елементи; уміти наводити приклади середовищ програмування та підтримуваних ними мов; розуміти концепції консольного режиму виконання програм і графічних інтерфейсів; уміти наводити приклади типів даних і пояснювати їхнє призначення; уміти пояснити поняття об'єктів, класів як об'єктних типів даних, подій та обробників подій; розглядати деякі типи програмних проєктів як подієво-орієнтовані та об'єктно-орієнтовані середовища; пояснювати поняття логічних виразів, розуміти таблиці істинності, застосовувати логічні функції та складові логічні вирази; уміти пояснити різницю між формальними та реальними параметрами; пояснювати поняття масивів, списків, словників, стеків, черг і хеш-таблиць та наводити їхні приклади; вміти розпізнавати, розрізняти та класифікувати різні структури даних; уміти пояснити зручність використання та характеристики конкретних структур даних у тому чи іншому алгоритмі тощо.

В 11 класі школярі вивчають ще два розділи, пов’язані з програмуванням – «Алгоритми» та «Парадигми та технології програмування».

Розділ «Алгоритми» зосереджений на вивченні основних принципів і методів розробки алгоритмів для вирішення різноманітних задач. Він охоплює такі ключові аспекти, як побудова алгоритмів, їх аналіз та оптимізація. Учні дізнаються про різні типи алгоритмів, такі як сортування, пошук, та графові алгоритми, і вчаться використовувати їх на практиці для створення ефективних програм. Ця тема є критично важливою для розвитку навичок логічного мислення та проблемоорієнтованого підходу.

Вивчення алгоритмів тісно пов'язано з програмуванням та структурами даних, утворюючи основу для створення ефективного і надійного програмного забезпечення (рис. 3.5).

Програмування	Структури даних
• Алгоритми визначають послідовність дій, які комп'ютер повинен виконати для розв'язання задачі. Вони є центральним елементом програмування, оскільки код, що реалізує алгоритм, забезпечує його виконання на машинному рівні. Знання алгоритмів дозволяє програмістам писати код, який ефективно вирішує конкретні проблеми, з урахуванням обмежень часу та пам'яті.	• Алгоритми часто взаємодіють зі структурами даних, оскільки ці структури визначають спосіб зберігання та організації даних у пам'яті. Правильний вибір структури даних може значно покращити ефективність алгоритму. Наприклад, пошукові алгоритми можуть використовувати хеш-таблиці або дерева для швидкого доступу до елементів, тоді як сортувальні алгоритми потребують оптимальних структур для ефективного впорядкування даних.

Рис. 3.5. Зв'язок розділу «Алгоритми»
з програмуванням і структури даних

Зміст даного розділу сконцентрований у наступних темах (рис. 3.6).

- Методи проектування алгоритмів. Методи представлення алгоритмів. Кодування алгоритмів. Поняття складності алгоритмів.
- Основні поняття теорії чисел: системи числення, робота з великими числами, факторизація чисел
- Алгоритми сортування. Квадратичні алгоритми сортування. Сортування вставками, сортування підрахунком, сортування злиттям
- Алгоритми пошуку. Бінарний пошук, тернарний пошук, пошук з поверненням.
- Обробка рядків. Основні поняття теорії графів. Способи представлення графів. Пошук у ширину та глибину; визначення найкоротшого шляху в графі, алгоритм Дейкстри, алгоритм Флойда-Уоршелла.
- Динамічне програмування. Жадібні алгоритми. Базові поняття обчислювальної геометрії.
- Векторний добуток; напрямок повороту; визначення площі многокутника; побудова опуклої оболонки.

Рис. 3.6. Зміст розділу «Алгоритми»

Таким чином, алгоритми, програмування та структури даних утворюють тісно взаємопов'язану тріаду, що забезпечує розробку ефективних програмних рішень.

Розділ «Парадигми та технології програмування» охоплює основні концепції та підходи до програмування, які використовуються у створенні програмного забезпечення. Він включає вивчення різних програмних парадигм, таких як процедурне, об'єктно-орієнтоване та функціональне програмування, а також ознайомлення з сучасними технологіями та інструментами, які допомагають розробникам писати ефективний та надійний код. Цей розділ є фундаментальним для розуміння різноманітності методів програмування та їх практичного застосування.

У 11 класі даний розділ представлений такими темами (рис. 3.7).

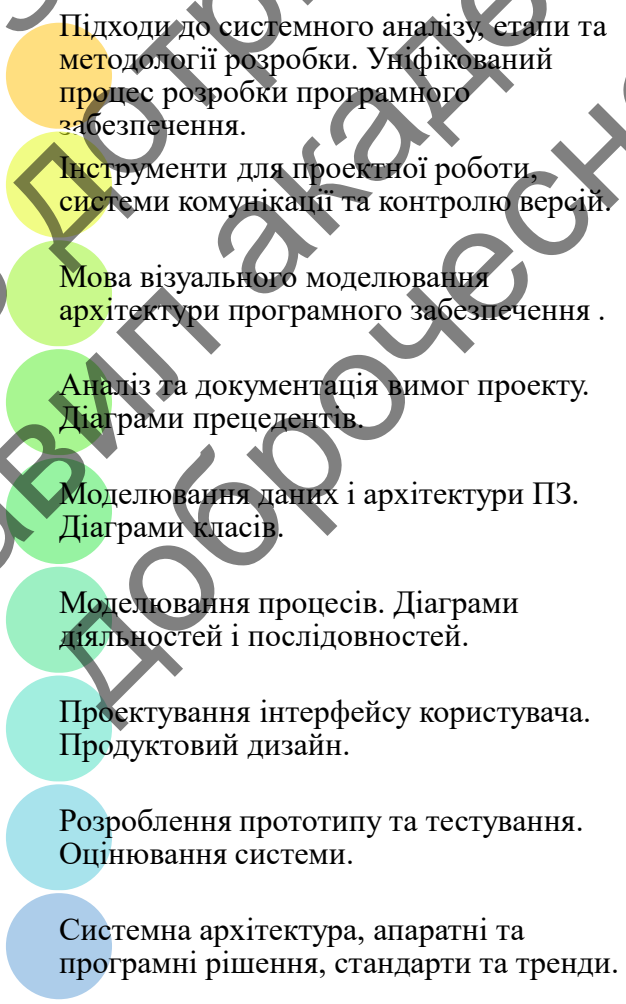

Рис. 3.7. Зміст розділу «Парадигми та технології програмування»

«Парадигми та технології програмування» є основою для розуміння та ефективного використання структур даних, що забезпечує створення якісного та продуктивного програмного забезпечення. На рис. 3.8 відображено зв’язок даного розділу із програмуванням та структури даних.

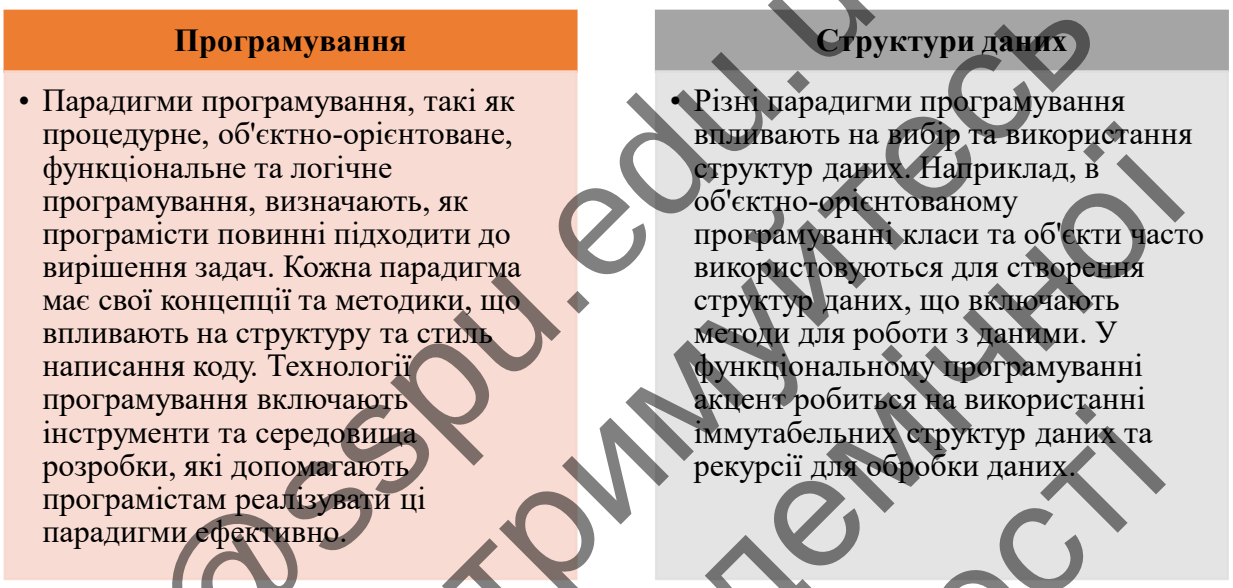

Рис. 3.8. Зв’язок розділу «Парадигми та технології програмування» з програмуванням і структури даних

Вивчення розділів «Програмування та структури даних», «Алгоритми» та «Парадигми та технології програмування» в старших класах профільної школи у комплексі надає учням фундаментальні навички, які є невід'ємними для будь-якого програміста. Разом вони формують цілісне розуміння процесу розробки програмного забезпечення, від основних принципів написання коду і роботи з даними до складніших підходів та методів розв'язання задач. Це дозволяє створювати ефективні, надійні та масштабовані рішення, що відповідають сучасним вимогам технологій.

Розглянемо вивчення програмування та структур даних у підручнику з інформатики для профільних класів у старших класах ЗЗСО. Проаналізуємо підручник авторів В. Д. Руденко, Н. В. Речич, В. О. Потієнко [22].

Тут для вивчення програмування є розділ «Мова програмування та структури даних». Зміст даного розділу відображено у вивченні 8 тем, поділеним на параграфи (рис. 3.9).

Зміст

Рис. 3.9. Зміст розділу «Мова програмування та структури даних» у [22]

Зі змісту видно, що автори підручника рекомендують вивчати мову програмування Python. Її вивченню присвячено майже усі теми розділу.

Цей розділ зазвичай охоплює основні концепції програмування та структури даних, які є фундаментальними для розуміння процесів розробки програмного забезпечення. Він покликаний навчити учнів базовим принципам

програмування та використання структур даних для ефективної організації та обробки інформації.

У підручнику передбачено вивчення теоретичного матеріалу (рис. 3.10), що підкріплюється запитаннями для перевірки знань (рис. 3.11) та завданнями для самостійного виконання (рис. 3.12).

Рис. 3.10. Приклад подання теоретичного матеріалу в підручнику

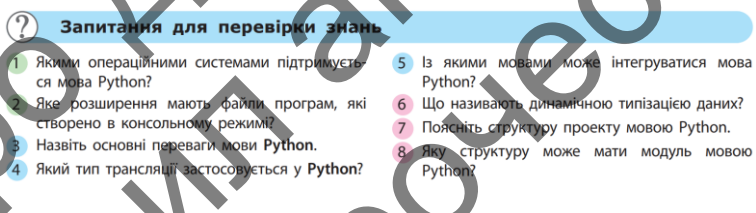

Рис. 3.11. Приклад блоку запитання для перевірки знань в підручнику

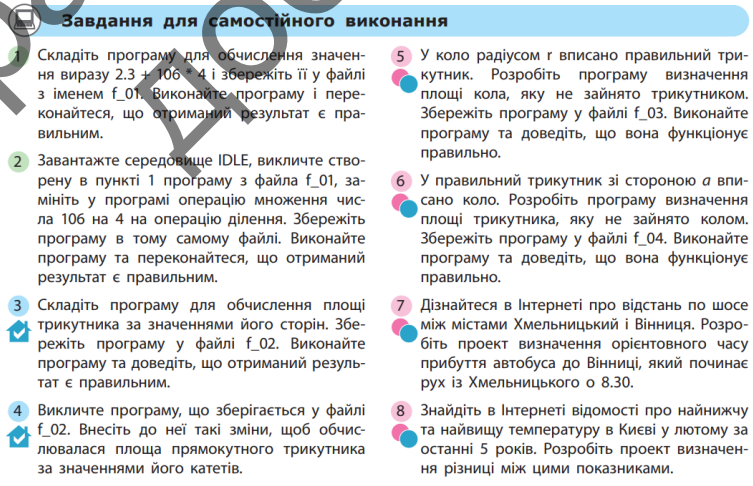

Рис. 3.12. Приклад блоку завдання для самостійного виконання

Завершуючи аналіз програм і підручника з інформатики профільної школи, можна зазначити, що сучасні навчальні матеріали забезпечують учнів якісними знаннями та навичками, необхідними для успішної діяльності в інформаційному суспільстві. Підручники і програми відповідають вимогам часу, включають інноваційні методики та технології навчання, що сприяє розвитку логічного мислення, алгоритмічних здібностей та творчого підходу до вирішення задач. Одним з ключових аспектів є інтеграція різних мов програмування і структур даних, що дозволяє учням отримати всебічну підготовку і бути конкурентоспроможними на ринку праці.

3.2. Типові задачі теми та аналіз типових помилок у їх розв’язуванні

Типові задачі з програмування охоплюють широкий спектр тем: від базових операцій з даними до побудови складних алгоритмів. Розглядаючи типові задачі, можна глибше зрозуміти підходи до їх розв’язання, а також найбільш поширені помилки, з якими стикаються учні. Аналіз цих помилок дозволяє виявити прогалини у знаннях і зробити процес навчання більш ефективним, адже учні отримують можливість не лише опанувати базові концепції програмування, але й навчитися уникати й коригувати помилки. Це сприяє розвитку алгоритмічного мислення і формує ґрунтовну базу для подальшого вивчення складніших тем.

Типові задачі – це завдання, що часто зустрічаються у певній області чи контексті і мають стандартні підходи до їх вирішення [23]. В інформатиці та програмуванні типові задачі можуть включати сортування масивів, пошук елементів у даних, обробку тексту, роботу з файлами і базами даних, реалізацію алгоритмів на графах тощо. Ці задачі допомагають навчатися застосовувати теоретичні знання на практиці і закріплюють основні принципи та методи розв'язання задач у конкретній галузі.

Проаналізувавши завдання з підручника «Інформатика» для 10-11 класів профільної школи були визначені типові задачі з теми «Програмування і структури даних».

1. Перевірка парності числа

Тема: Розгалуження

Опис: Написати програму, яка отримує на вхід ціле число і виводить повідомлення, чи є число парним або непарним.

Мета: Навчитися використовувати умовні оператори для прийняття рішень.

Програмний код розв'язання задачі, написаний мовою Python:

```
def is_even(number):
    if number % 2 == 0:
        return "Число парне"
    else:
        return "Число непарне"

# Приклад виклику
print(is_even(7))
```

Типова помилка при розв'язуванні задачі: Використання оператора = замість == у виразі умови.

Аналіз типової помилки: часто учні плутають оператор присвоєння (=) та порівняння (==). Це може призвести до логічних помилок, коли програма працює не так, як очікується, або виводить некоректний результат.

2. Визначення максимального і мінімального елементів масиву

Тема: Масиви, цикли

Опис: Створити програму, яка приймає масив чисел і знаходить максимальний і мінімальний елементи.

Мета: Розвинути навички роботи з масивами та циклами.

Програмний код розв'язання задачі, написаний мовою Python:

```
def find_min_max(arr):
    min_val = min(arr)
    max_val = max(arr)
    return min_val, max_val

# Приклад виклику
array = [3, 5, 2, 8, 6]
print(find_min_max(array))
```

Типова помилка при розв'язуванні задачі: Забування ініціалізувати змінні для мінімального та максимального значень перед циклом.

Аналіз типової помилки: Учні можуть пропустити ініціалізацію або вибрати неправильні початкові значення для змінних. Це призводить до того, що програма не знаходить правильний максимум чи мінімум, особливо коли працює з від'ємними числами або нулями.

3. Пошук простих чисел у заданому діапазоні

Тема: Цикли, умови

Опис: Написати програму, яка виводить усі прості числа в заданому діапазоні.

Мета: Закріпити використання вкладених циклів та умов для визначення простих чисел.

Програмний код розв'язання задачі, написаний мовою Python:

```
def is_prime(num):
    if num < 2:
        return False
```

```
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

def primes_in_range(start, end):
    primes = [num for num in range(start, end + 1) if is_prime(num)]
    return primes

# Приклад виклику
print(primes_in_range(10, 30))
```

Типова помилка при розв'язуванні задачі: Пропуск випадку, коли num < 2 або неправильне використання діапазону для перевірки дільників.

Аналіз типової помилки: Учні часто не враховують, що числа менше двох не є простими, або ж перевіряють діапазон дільників до самого числа, а не до його квадратного кореня, що знижує ефективність.

4. Зворотний порядок символів у рядку

Тема: Робота з рядками, цикли

Опис: Написати програму, яка приймає рядок і виводить його у зворотному порядку.

Мета: Навчитися працювати з рядками та ітераціями для створення нових послідовностей символів.

Програмний код розв'язання задачі, написаний мовою Python:

```
def reverse_string(s):
    return s[::-1]
```

```
# Приклад виклику
print(reverse_string("Програмування"))
```

Типова помилка при розв'язуванні задачі: Використання циклу без врахування індексації, що призводить до виключення першого або останнього символу.

Аналіз типової помилки: Учні можуть не враховувати коректне обчислення індексів або неправильне застосування операції зрізу (slicing), що може дати неповний або некоректний результат.

5. Підрахунок слів у реченні

Тема: Рядки, масиви

Опис: Створити програму, яка отримує на вхід речення і виводить кількість слів у ньому.

Мета: Розвинути навички роботи з текстовими даними, використання методів розбиття рядка на слова.

Програмний код розв'язання задачі, написаний мовою Python:

```
def count_words(sentence):
    words = sentence.split()
    return len(words)

# Приклад виклику
print(count_words("Програмування розвиває логічне мислення"))
```

Типова помилка при розв'язуванні задачі: Неправильний підрахунок через відсутність перевірки на порожній рядок.

Аналіз типової помилки: Якщо рядок порожній або містить лише пробіли, метод split() повертає порожній список, що може призвести до

помилкових результатів. Учні часто не перевіряють цей випадок, що призводить до неправильних результатів.

6. Реалізація черги (Queue)

Тема: Структури даних (черга)

Опис: Написати програму, яка імітує чергу з можливістю додавання та видалення елементів.

Мета: Зрозуміти базові принципи роботи черги та навчитися реалізовувати її основні операції.

Програмний код розв'язання задачі, написаний мовою Python:

```
from collections import deque

queue = deque()

def enqueue(item):
    queue.append(item)

def dequeue():
    if queue:
        return queue.popleft()
    else:
        return "Черга порожня"

# Приклад використання
enqueue(1)
enqueue(2)
enqueue(3)
print(dequeue())  # Видалить 1
print(dequeue())  # Видалить 2
```

Типова помилка при розв'язуванні задачі: Відсутність перевірки на порожність черги перед видаленням елементу.

Аналіз типової помилки: Учні можуть не враховувати випадок, коли черга порожня, що спричиняє помилку при виконанні операції видалення (dequeue). Важливо вивчити, як обробляти такі ситуації, щоб уникнути помилок виконання.

7. Сортування масиву методом вставки

Тема: Алгоритми сортування

Опис: Реалізувати алгоритм сортування методом вставки для масиву чисел.

Мета: Розібратись із принципами сортування та навчитися використовувати алгоритми для впорядкування даних.

Програмний код розв'язання задачі, написаний мовою Python:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

# Приклад виклику
array = [12, 11, 13, 5, 6]
print(insertion_sort(array))
```

Типова помилка при розв'язуванні задачі: Некоректний обхід масиву, що призводить до відсутності сортування останнього елемента або виникнення нескінченних циклів.

Аналіз типової помилки: Якщо учні не дотримуються умов виходу з циклу або пропускають інкремент індексу, це може призвести до помилок сортування або нескінченних циклів.

8. Факторіал числа за допомогою рекурсії

Тема: Рекурсія

Опис: Написати функцію, яка обчислює факторіал заданого числа рекурсивно.

Мета: Закріпити принципи рекурсії на прикладі розрахунку факторіалу.

Програмний код розв'язання задачі, написаний мовою Python:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Приклад виклику
print(factorial(5))
```

Типова помилка при розв'язуванні задачі: Відсутність базового випадку для рекурсії, що призводить до переповнення пам'яті.

Аналіз типової помилки: Часто учні забувають додати базовий випадок (наприклад, n == 0), що є необхідним для припинення рекурсії. Це призводить до того, що програма працює нескінченно або завершується з помилкою.

9. Перетин двох множин

Тема: Структури даних (множини)

Опис: Написати програму, яка приймає два списки чисел та виводить їх перетин, тобто спільні елементи.

Мета: Навчитися працювати з множинами та використовувати їх для знаходження спільних елементів.

Програмний код розв'язання задачі, написаний мовою Python:

```
def intersection(list1, list2):
    return list(set(list1) & set(list2))


# Приклад виклику
list1 = [1, 2, 3, 4]
list2 = [3, 4, 5, 6]
print(intersection(list1, list2))
```

Типова помилка при розв'язуванні задачі: Використання списків замість множин, що спричиняє дублікати у результаті.

Аналіз типової помилки: Учні можуть не використовувати множини для знаходження спільних елементів, що призводить до дублювання елементів у результаті, особливо якщо в обох списках є повторювані значення.

10. Перевірка симетричності дерева (дерево рішень)

Тема: Структури даних (дерева)

Опис: Написати програму, яка перевіряє, чи є двійкове дерево симетричним.

Мета: Закріпити навички роботи з деревоподібними структурами даних і розвинути розуміння алгоритмів обробки дерев.

Програмний код розв'язання задачі, написаний мовою Python:

```
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
```

```
        self.right = right


def is_symmetric(root):
    def is_mirror(left, right):
        if not left and not right:
            return True
        if not left or not right:
            return False
        return (left.value == right.value and
                is_mirror(left.left, right.right) and
                is_mirror(left.right, right.left))

    return is_mirror(root, root)

# Приклад дерева для перевірки
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(2)
root.left.left = TreeNode(3)
root.left.right = TreeNode(4)
root.right.left = TreeNode(4)
root.right.right = TreeNode(3)

print(is_symmetric(root))  # Поверне True, якщо дерево симетричне
```

Типова помилка при розв'язуванні задачі: Некоректне порівняння вузлів з використанням значень замість рекурсивного обходу.

Аналіз типової помилки: Учні можуть не зрозуміти, що для перевірки симетричності потрібно рекурсивно порівнювати піддерева, а не лише значення вузлів. Також важливо враховувати випадок, коли один вузол має нащадка, а інший – ні, що робить дерево несиметричним.

Розглянуті задачі охоплюють широкий спектр тем, з якими учні стикаються на уроках інформатики. Аналіз типових помилок допомагає зрозуміти, де учням може знадобитися більше уваги та вправ. Це також сприяє розвитку уважності до деталей і допомагає покращити загальну логіку побудови алгоритмів та розв'язання задач.

Висновки до розділу 3

У третьому розділі «Методичні особливості вивчення теми «Мова програмування та структура даних» в умовах профільної школи» проаналізовано навчальні програми з інформатики профільної школи, наведено типові задачі теми та проведено аналіз типових помилок учнів при їх розв'язуванні.

За результатами аналізу навчальних програм і підручників з інформатики з'ясовано, що вони відповідають вимогам часу, включають інноваційні методики та технології навчання, що сприяє розвитку логічного мислення, алгоритмічних здібностей та творчого підходу до вирішення задач. Виявлено можливість інтеграції різних мов програмування і структур даних, що дозволяє учням отримати всебічну підготовку і бути конкурентоспроможними на ринку праці.

Представлено типові задачі для 10-11 класів профільної школи з теми «Програмування і структури даних»: розгалуження; масиви, цикли; цикли, умови; робота з рядками, цикли; рядки, масиви; структури даних (черга); алгоритми сортування; рекурсія; структури даних (множини); структури даних (дерева). До кожної із задач подано аналіз типових помилок.

ВИСНОВКИ

У кваліфікаційній роботі представлено особливості вивчення теми «Мова програмування та структура даних» у профільній школі. Проведене дослідження засвідчує досягнення мети, вирішення поставлених завдань та уможливлює такі **висновки**.

1. З розвитком ІТ удосконалюється зміст і вимоги до результатів навчання інформатики. Сьогодні актуальним є вивчення мови програмування та структур даних у профільній школі. Серед найбільш популярних мов програмуванням, які вивчаються у ЗЗСО - Python, Scratch, C++, Java, JavaScript, Ruby, Pascal та Object Pascal. Ці мови програмування можуть бути рекомендовані для вивчення у шкільному курсі інформатики профільної школи.

2. Для програмування важливе розуміння різних структур даних, тобто структур, які передбачають певну організацію та зберігання даних у пам'яті комп'ютера для ефективного доступу й оброблення. Серед поширених структур даних виділяють масиви, списки, стеки, черги, хеш-таблиці, дерева, графи. Вони мають специфічну побудову, а тому використовуються для вирішення різних (специфічних) завдань.

3. За аналізом навчальних програм з інформатики профільної школи встановлено, що на вивчення інформатики відводиться 5 годин на тиждень, а тема «Мова програмування та структура даних» включає вивчення «Мова програмування. Класифікація та складові мов програмування. Особливості середовища розробки. Структура програмного проекту. Основні елементи мови програмування. Використання змінних і виразів. Реалізація базових алгоритмічних конструкцій. Логічні вирази. Таблиці істинності. Функції. Параметри. Поняття рекурсії. Рекурсивні функції. Вказівники» та «Поняття структур даних, масив, список, словник, стек, черга, хеш-таблиця. Класифікація структур даних. Лінійні структури даних. Способи реалізації структур даних. Застосування на практиці різних структур даних. Бібліотеки мови програмування. Створення, редагування та тестування консольних

програм і програм з графічним інтерфейсом. Елементи об’єктно-орієнтованого програмування. Правила написання читабельного коду. Коментарі у тексті програми». Чинні підручники передбачають вивчення наведених понять.

4. За результатами дослідження виділено типові задачі теми «Мова програмування та структура даних» та представлено аналіз типових помилок учнів при їх розв’язуванні. Розглянуто розв’язування задач, які пов’язані з розгалуженнями, масивами, циклами, рядками, різними структурами даних (черга, множина, дерево), алгоритмами сортування, рекурсією.

Подальшого дослідження потребують особливості вивчення теми «Мова програмування та структура даних» в умовах дистанційної освіти та з використанням штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Java – Енциклопедія мов програмування. URL: http://progopedia.ru/language/java/

2. Schildt Herbert. C++ The Complete Reference Third Edition. Osborne McGraw-Hill, 1998. 1008 с.

3. Stroustrup Bjarne. The C++ Programming Language. Addison-Wesley, 1997. 910с.

4. The Good and the Bad of Java Programming, 2019. URL: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/

5. The Making of Python. URL: https://www.artima.com/intv/python.html

6. TIOBE Index. URL: https://www.tiobe.com/tiobe-index/

7. Алгоритми і структури даних: лабораторний практикум для студентів напряму підготовки "Телекомунікації і радіотехніка" / Уклад.: М.А. Скулиш, С.В. Суліма. К.: КПІ ім. Ігоря Сікорського, 2021. 109 с.

8. Вакалюк Т.А. Програмування мовою Pascal. Навчально-методичний посібник для студентів фізико-математичного факультету. Житомир: ФО-П Левковець Н.М., 2016. 232 с.

9. Горошко Ю., Костюченко А., Шкардибарда М. Використання ВПЗ у процесі вивчення основ програмування. Інформатика та інформаційні технології. 2012. №1. С. 22–25.

10. Дудка О. М., Власій О. О. Особливості вивчення програмування на Scratch. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2017. № 26. С. 81-87.

11. Кобильник Т.П., Когут У.П., Жидик В.Б. Методичні аспекти вивчення основ алгоритмізації і програмування мовою Python у шкільному курсі інформатики у старших класах. Фізико-математична освіта, 2021. Випуск5(31). С. 36-44.

12. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002. 83 с.

13. Маковський Д. Ю. Особливості мови програмування Ruby та фреймворку Ruby on Rails. Актуальні питання сучасної інформатики, 2016. С. 100-105.

14. Міцкан Л., Вербицька Т., Базурін В. Порівняльний аналіз мов Python і Free Pascal як перших мов програмування для учнів 8 класу. Актуальні питання природничо-математичної освіти : збірник наукових праць, 2017. № 2 (10). С. 130–139.

15. Нова українська школа | Веб-ресурс НУШ. URL: https://nus.org.ua/

16. О Ruby. URL: https://www.ruby-lang.org/

17. Пенко В., Пенко О. Використання візуалізації на різних етапах вивчення дисципліни «Програмування». Освіта. Інноватика. Практика, 2023. Том 11, № 2. С. 31-39. DOI: 10.31110/2616-650X-vol11i2-005

18. Підручник Pascal - основи програмування. URL: http://pascal.dp.ua/rozdl-pershiy.html.

19. Про затвердження Державного стандарту базової і повної загальної середньої освіти. URL: https://zakon.rada.gov.ua/laws/show/1392-2011-%D0%BF

20. Рубець А. До питання про вибір мови програмування при вивченні теми «Мова програмування та структура даних» у старшій школі. Студентська звітна конференція: Матеріали результатів наукових досліджень молодих науковців. Суми: Вид-во фізико-математичного факультету СумДПУ імені А.С. Макаренка, 2024. Випуск 18. С. 42-43.

21. Рубець А. Методичні особливості вивчення теми «Мова програмування та структура даних» у старшій школі. Збірник праць студентів фізико-математичного факультету СумДПУ імені А.С.Макаренка. Суми: Вид-во фізико-математичного факультету СумДПУ імені А.С. Макаренка, 2024. Випуск 18. С. 109-114.

22. Руденко В. Д., Речич Н. В., Потієнко В. О. Інформатика (профільний рівень) : підруч. для 10 кл. закл. загал. серед. освіти. Харків : Вид-во «Ранок», 2019. 256 с.

23. Семко Л. Прикладні задачі у навчанні інформатики в гімназії. Матеріали II Міжнародної науково-практичної інтернет-конференції «Ресурсно-орієнтоване навчання в «3D»: доступність, діалог, динаміка, 2022. С. 22-23.

24. Сучасна інформатика (сила практики в теорії): Структура програми. URL: https://alextexnok.blogspot.com/p/normal-0-21-false-false-false-uk-x-none_29.html

25. Татарчук Д.Д. Алгоритмічна мова Паскаль: Навчальний посібник для студентів бакалаврату напрямку електроніка. ІВЦ “Політехніка”, 2006. 85 с.

26. Юрченко А.О., Самойленко Л.О. Мови програмування, які вивчаються у ЗЗСО. Діджиталізація в Україні: інновації в освіті, науці, бізнесі: Матеріали міжнародної науково-практичної конференції, 16-18 вересня 2019 року, Бердянськ, 2019. С. 38-47.

27. Юрченко А.О., Семеніхіна О.В., Хворостіна Ю.В., Удовиченко О.М. Навчання програмувати в старшій школі крізь призму чинних навчальних програм. Фізико-математична освіта. 2019. Випуск 2(20). Ч.2. С. 47-54.

28. Жалдак М. І., "Інформатика – фундаментальна наукова дисципліна".Комп’ютер у школі та сім’ї. № 2, с. 39-43, 2010.

29. Мінтій І. С., "Форми організації навчання для формування компетентностей з програмування". Педагогіка вищої та середньої школи, №41, с.91-96, жовтень, 2014.

30. Співаковський О.В., "Майбутнє шкільної інформатики". Науковий часопис НПУ імені М. П. Драгоманова. № 3, с. 226-234, 2010.

31. Іщеряков С.. Вчити програмування треба в школі чи університеті? Освітня політика. Портал громадських експертів URL: http://osvita.ua/school/54063/

32. Rudenko, Y., Drushlyak, M., Osmuk, N., Shvets, O., Kolyshkin, O., Semenikhina, O. (2022). Problems of Teaching Pupils of Non-Specialized Classes to Program and Ways to Overcome Them: Local Study. International journal of computer science and network security, 22, 1, 105-112. doi 10.22937/IJCSNS.2022.22.1.16

33. Тихоненко О. О. Методичні підходи до формування та розвитку алгоритмічного мислення в учнів початкових класів на уроках інформатики. Вісник Чернігівського національного педагогічного університету. Серія: Педагогічні науки. 2015. № 125. С. 371–374.

34. Базурін В. М. Середовища програмування як засіб навчання учнів основ програмування. Інформаційні технології і засоби навчання. 2017. № 59, вип. 3. С. 13–27.

35. Шаран О. В., Шаран В. Л., Кулинич М. М. Особливості використання електронних освітніх ресурсів у процесі розвитку алгоритмічного мислення молодших школярів. Педагогіка формування творчої особистості у вищій і загальноосвітній школах. 2021. № 78. С. 130–134.

36. Василиків І., Романчук Р. Особливості формування алгоритмічного мислення молодших школярів на уроках інформатики. Молодь і ринок. 2021. № 2/188. С. 90–94

37. Семеніхіна О.В., Руденко Ю.О. Проблеми навчання програмувати учнів старших класів та шляхи їх подолання // Інформаційні технології і засоби навчання. – 2018. – Том 66. – №4. – С. 54-64.

38. Кобильник Т.П., Когут У.П., Жидик В.Б. Методичні аспекти вивчення основ алгоритмізації і програмування мовою Python у шкільному курсі інформатики у старших класах. Фізико-математична освіта, 2021. Випуск5(31). С. 36-44.

39. Буйдіна О. О. Профільне навчання як модель зовнішньої диференціації в освіті: реалії та перспективи. Проблеми сучасної педагогічної освіти. Сер. : Педагогіка і психологія: зб. статей. Ялта: РВВ КГУ, 2014. Вип.44. Ч.2. С.52-59.

40. Закон України "Про освіту". URL: https://osvita.ua/legislation/law/2231/

41. Закон України "Про загальну середню освіту". URL: https://osvita.ua/legislation/law/2232/

42. Національна доктрина розвитку освіти. URL: https://osvita.ua/legislation/other/2827/

43. Державний стандарт базової і повної загальної середньої освіти. URL: https://osvita.ua/legislation/Ser_osv/28030/

44. Про затвердження Концепції профільного навчання у старшій школі. URL: https://osvita.ua/legislation/Ser_osv/37784/

45. Шиян Н. Профільне навчання учнів у загальноосвітній школі сільської місцевості. Педагогічні науки. 2010. Вип. 1. С. 12-20.

46. Python on Android. URL: https://www.damonkohler.com/2008/12/python-on-android.html

47. Навчальна програма з інформатики (профільний рівень) для 10-11 класів загальноосвітніх шкіл, затверджена Наказом Міністерства освіти і науки № 1407 від 23 жовтня 2017 року. URL: https://mon.gov.ua/ua/osvita/zagalna-serednya-osvita/navchalni-programi/navchalni-programi-dlya-10-11-klasiv

48. Навчальна програма з інформатики (рівень стандарту) для 10-11 класів загальноосвітніх шкіл, затверджена Наказом Міністерства освіти і науки № 1407 від 23 жовтня 2017 року. URL: https://mon.gov.ua/ua/osvita/zagalna-serednya-osvita/navchalni-programi/navchalni-programi-dlya-10-11-klasiv