

Лекція 2. КЛАСИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Універсальність ЕОМ. Кодування інформації. Кодування чисел. Системи числення. Системне та прикладне програмне забезпечення. Системи програмування. Етапи створення програми. Файлова система ОС

Універсальність ЕОМ. Кодування інформації. Ми декілька разів наголошували на тому, що ЕОМ працює під керівництвом програм. Насправді, це дуже важлива теза, бо без програми навіть найдосконаліший процесор буде «мертвим» – він не матиме, що виконувати. Так само і будь-яка програма без комп'ютера принесе небагато користі. Повноцінну ЕОМ створюють тільки в результаті поєднання апаратури і належних програм. В інформатиці затвердились відповідні терміни: *hardware* (жорстка машина) означає апаратну частину комп'ютера, *software* (м'яка машина) – програмну. Можна сказати, що *hardware* є «тілом» комп'ютера, а *software* – його «душею».

Завдяки реалізації другого принципу фон Неймана ЕОМ може виконувати будь-які програми, складені з команд, зрозумілих процесорові. Саме тому сучасні комп'ютери є *універсальними* пристроями. Та сама машина може допомагати створювати тексти, графіку, виконувати обчислення чи провадити медичні дослідження залежно від встановлених на неї програм.

Як інформація різних видів (числова, текстова, графічна, звукова тощо) зберігається в пам'яті комп'ютера? Невже він настільки «ерудований», що володіє математичними знаннями, багатьма мовами, нотною грамотою? Звичайно, ні! Але цього і не треба. Для відображення різноманітної інформації в електронному вигляді застосовують кодування.

Кодуванням називають заміну літер довільного алфавіту літерами чи послідовностями літер деякого фіксованого, стандартного алфавіту. Якщо ці послідовності мають однакову довжину, то код називають нормальним.

Одним із всесвітньовідомих кодів є азбука Морзе. Її використовували телеграфісти для кодування текстової інформації. Стандартний алфавіт у ній містить тільки дві літери: «·» і «–» (крапку і тире). У табл. 2.1 наведено приклади кодування окремих літер. Видно, що коди мають різну довжину.

Апаратне влаштування електронної машини таке, що є змога розрізнити тільки дві можливості: тече струм, чи ні; намагнічено ділянку магнітного носія, чи ні тощо. Тому стандартний алфавіт ЕОМ також складається тільки з двох літер. Люди домовились позначати їх «0» і «1». Отже, всю різноманітну інформацію, з якою працює комп'ютер, у тім числі і його програми, кодують за допомогою послідовностей нулів та одиниць.

Для різних видів інформації розроблено різні коди. Наприклад, для кодування текстів використовують стандарти ASCII, ISO, KOI8, Unicode тощо. Найдавнішим з них є ASCII (American Standard of Code Information and Interchange). Для кодування однієї літери він використовує послідовність з восьми нулів та одиниць. Тому довжина одного коду – 1 байт, і є змога закодувати 256 різних літер. До стандарту занесено коди великих і малих латинських літер, цифр, спеціальні знаки, керуючі символи (основна таблиця, коди від 0 до 127), символи псевдографіки, літери національних алфавітів (додаткова таблиця, коди від 128 до 255).

Кодування графіки доволі трудомістке, адже все зображення треба поділити на складові частини і кожен з них якось позначити. Сьогодні використовують два головні підходи: растровий і векторний. *Растровий* формат трактує графічне зображення як прямокутний масив дрібних цяток (пікселів), описаних числовим значенням кольору та

Таблиця 2.1.
Кодування в азбуці
Морзе

Літера	Код
е	·
т	–
а	· –
н	– ·
к	– · –
о	– – –
с	· · ·
ш	– – – –

яскравості. *Векторний* формат виокремлює більші частини зображення і описує їхні контури за допомогою математичних формул. Ось деякі з форматів кодування графіки.

- *BMP* (Bitmap) – растровий формат, створений компанією Microsoft для використання в ОС Windows для подання зображень у ресурсах програми (іконки, меню, кнопки, панелі тощо). Підтримує тільки RGB модель з глибиною кольору до 24 біт (модель Red Green Blue використовують для опису кольорів, що випромінюються, наприклад, відтворених монітором).
- *TIFF* (Tagged Image File Format) – універсальний растровий формат фірми Aldus для зберігання сканованих зображень. Реалізовано практично на всіх платформах, тому його часто використовують для перенесення зображень. Підтримує монохромні, індексовані, півтонові зображення, зображення в моделях RGB і CMYK з каналами у 8 та 16 біт (модель Cyan Magenta Yellow black призначено для опису відбитих кольорів, наприклад, відтворених кольоровим принтером).
- *JPEG* (Join Photographic Expert Group) – у цьому растровому форматі вперше реалізовано новий спосіб стиску графіки з втратою якості. З зображення вилучають ту його частину, яка не сприймається (чи слабко сприймається) людським оком, у результаті чого перетворене зображення займає набагато менше місця. Зручний для передавання файлів каналами зв'язку. Підтримує RGB і CMYK та різні ступені стиску. Головний недолік – на рисунках з чіткими границями і великими заповненими ділянками сильно простежується дефект стиску.
- *GIF* (Graphic Interchange Format) – створений фірмою CompuServe спеціально для передавання у глобальних мережах. Формат зорієнтовано на компактність. Він підтримує зберігання декількох зображень у схожому на шари розташуванні. Браузери здатні демонструвати такі зображення циклічно-послідовно, реалізуючи таким способом просту анімацію.

Мова опису сторінок PostScript – основа усіх видавничих систем – за своєю суттю належить до *векторних форматів*. Вона дає змогу описувати векторні та імпортовані растрові зображення, шрифти, параметри растрування та керування кольором. Кожен сучасний принтер має апаратний або програмний інтерпретатор PostScript. Приклади векторних форматів такі.

- *EPS* (Encapsulated PostScript) – спрощений PostScript для опису не цілої сторінки, а групи об'єктів. Підтримує всі моделі кольору.
- *PDF* (Portable Document Format) – універсальний векторний формат фірми Adobe System для електронного поширення документів. Створені різними програмами публікації можна зберігати у цьому форматі та переглядати на різних комп'ютерах за допомогою freeware (програма, що розповсюджується безплатно) Acrobat Reader. Сучасні принтери безпосередньо друкують документи у PDF форматі.

Кодування чисел. Системи числення. Детальніше розглянемо кодування числової інформації. Для цього ознайомимось спочатку з поняттям системи числення.

Системою числення (нумерації) називають спосіб позначення і запису чисел. Розрізняють *позиційні* та *непозиційні* системи. Так звана римська система для позначення чисел використовує літери: літера «I» має значення «один», «V» – «п'ять», «X» – «десять», «L» – «п'ятдесят», «C» – «сто», «D» – «п'ятсот», «M» – «тисяча». Значення кожної літери не залежить від її розташування в записі числа. Така система є *непозиційною*.

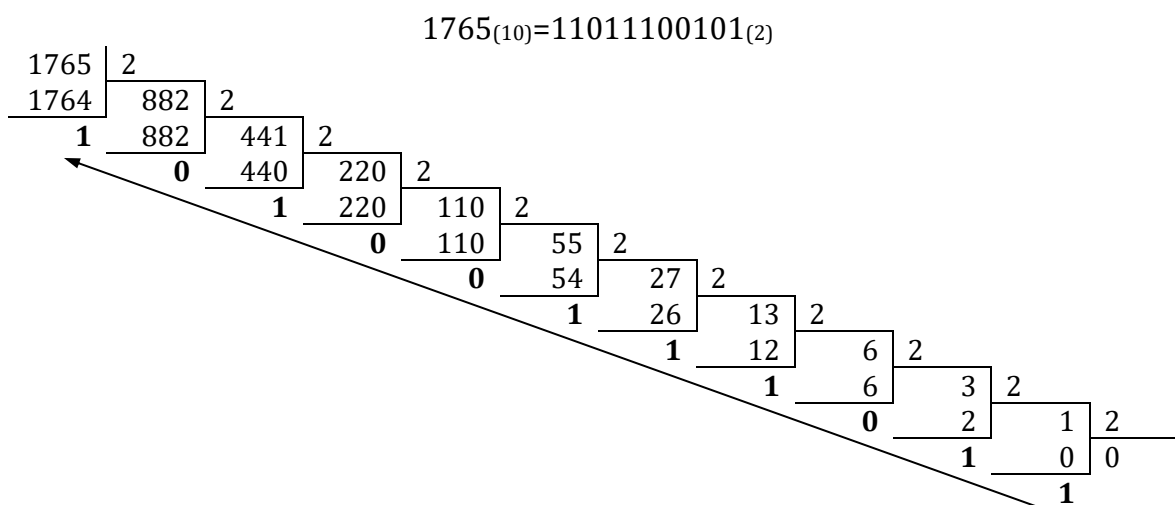
У *позиційній* системі деяку кількість n одиниць (наприклад, десять) молодшого розряду об'єднують в одну одиницю старшого розряду (десять одиниць – один десяток, десять десятків – одна сотня і т. д.). Число n називають *основою* системи числення, а знаки, які використовують для позначення кількостей одиниць у кожному розряді – цифрами (повинно бути n цифр). Звична нам система числення (арабські числа) є десятковою: $n = 10$, десять різних цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Значення цифри залежить від розряду – її місця в записі числа. Наприклад, запис «0001» означає число «один», «0010» – «десять», «0100» – «сто», «1000» – «тисяча». Незначущі нулі тут записано для наочності.

Походження десяткової системи пов'язують з лічбою на пальцях. Деякі народи використовували п'ятіркову систему. У Давньому Вавилоні була розповсюджена шістдесяткова система, рудименти якої знаходимо в поділі години на 60 хвилин і 1 хвилини на 60 секунд, або 1 кутового градуса на 60 minut і 1 минути на 60 секунд.

Ми вже знаємо, що в стандартному алфавіті ЕОМ є тільки два знаки, тому для зображення чисел в «електронній пам'яті» використовується двійкова система. Щоб поррахувати суму двох чисел, навіть звичайний калькулятор виконує такі дії: перетворює обидва доданки з десяткової системи в двійкову, виконує арифметичну дію за правилами двійкової арифметики, одержаний (двійковий) результат перетворює назад у десяткову систему. Як він це робить?

Переведення цілого числа до нової системи виконують послідовним діленням цього числа і одержаних часток на нову основу аж до одержання в частці нуля і записом обчислених остач цифрами нової системи, причому перша остача є наймолодшою цифрою.

Розглянемо приклад переведення числа з десяткової системи в двійкову:



Ділення завершилось, коли частка стала дорівнювати нулю. Обчислені остачі виділено жирним шрифтом. Двійковий запис числа одержано, починаючи з останньої – за напрямом стрілки.

Переведення дробового числа x ($0 < x < 1$) до нової систем виконують послідовним множенням дробової частини (числа та одержаних добутоків) на нову основу і записом обчислених цілих частин цифрами нової системи в порядку одержання. (Завжди множимо тільки дробову частину!)

Розглянемо два приклади:

$0,40625_{(10)} = 0,01101_{(2)}$	$0,7835_{(10)} = 0,110010..._{(2)}$
$\times 2$	$\times 2$
0 40625	0 7835
0 81250	1 5670
1 6250	1 134
1 250	0 268
0 50	0 536
1 0	1 072
	...

У першому з них процес переведення закінчився, коли дробова частина отриманого добутку стала дорівнювати нулю.

У другому прикладі одержуємо нескінченний двійковий дріб. Переведення припиняється тоді, коли заповнено всі двійкові розряди, відведені для запису числа, адже в скінченній пам'яті ЕОМ неможливо записати нескінченну послідовність цифр. У прикладі ми обмежились записом шести цифр, а на практиці їх враховують значно більше.

Щоб перевести до нової системи довільне число, окремо переводять його цілу і дробову частини: $1765,40625_{(10)} = 11011100101,01101_{(2)}$.

Арифметичні дії в двійковій системі виконують так само, як і в десятковій (з нулями та одиницями) за одного винятку: $(1+1)_{(2)} = 10_{(2)}$.

Як виконати обернене переведення з двійкової системи в десяткову? Можна використати ті самі алгоритми, але доведеться працювати за правилами двійкової арифметики і ділити та множити на $1010_{(2)}$ ($1010_{(2)} = 10_{(10)}$), що не дуже зручно. Тому використаємо інший підхід. Будь-яке число в позиційній системі можна подати у вигляді розвинення за степенями основи, наприклад, $1765 = 1 \cdot 10^3 + 7 \cdot 10^2 + 6 \cdot 10 + 5$. Так само можна записати і двійкове число: $1010_{(2)} = 1 \cdot 2^3 + 1 \cdot 2 = 10$. Тепер переведемо в десяткову систему двійкові числа, отримані в попередніх прикладах:

$$11011100101,01101_{(2)} = 2^{10} + 2^9 + 2^7 + 2^6 + 2^5 + 2^2 + 1 + 2^{-2} + 2^{-3} + 2^{-5} = \\ = 1024 + 512 + 128 + 64 + 32 + 4 + 1 + 0,25 + 0,125 + 0,03125 = 1765,40625.$$

$$0,11001_{(2)} = 2^{-1} + 2^{-2} + 2^{-5} = 0,5 + 0,25 + 0,03125 = 0,78125 \approx 0,7835.$$

Бачимо, що в результаті всіх переведень ціле число не змінилося, а друге дробове число тільки наближено дорівнює оригіналові (перше дробове є щасливим винятком). Така неприємність з дробовим числом сталася тому, що ми врахували не всі двійкові цифри його запису, відкинувши «хвіст» нескінченної послідовності. Наведені приклади демонструють головну відмінність між кодуванням цілих і дійсних чисел: *цілі відображаються в пам'яті ЕОМ точно, а дійсні – наближено*.

Двійковий запис числа доволі громіздкий. Кількість двійкових цифр дуже швидко збільшується зі збільшенням числа. Для зручності запису люди вигадали «допоміжну» шістнадцяткову систему, тобто систему з основою $n=16$. Така система використовує шістнадцять різних цифр – на додачу до звичних десяти арабських цифр ще шість літер латинського алфавіту: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Таблиця 2.2.

Відповідність між цифрами двійкової та шістнадцяткової систем

(16)	(2)	(10)	(16)	(2)
0	0000		8	1000
1	0001		9	1001
2	0010	10	A	1010
3	0011	11	B	1011
4	0100	12	C	1100
5	0101	13	D	1101
6	0110	14	E	1110
7	0111	15	F	1111

Оскільки $16 = 2^4$, то між цифрами шістнадцяткової системи і тетрадами (четвірками) цифр двійкової системи існує взаємно-однозначна відповідність. Її подано в табл. 2.2.

Щоб перевести двійкове число до шістнадцяткової системи, його розбивають на тетради, починаючи від двійкової коми, і кожен тетраду замінюють відповідною шістнадцятковою цифрою. В разі потреби до двійкового числа можна дописати незначущі нулі зліва та справа (щоб доповнити першу та останню тетради):

$$11011100101,01101_{(2)} = 0110\text{'1110}\text{'0101,0110}\text{'1000}_{(2)} = 6E5,68_{(16)}.$$

Так само легко переводять число з шістнадцяткової системи в двійкову: кожен цифру замінюють відповідною тетрадою і відкидають незначущі нулі. (Прочитайте попередній приклад справа наліво.)

Чи можна десяткове число перевести одразу в шістнадцяткову систему? Звичайно! Використаємо вже відомі нам алгоритми, тільки ділити і множити будемо на 16.

1765	16		
16	110	16	
16	96	6	16
16	14	0	0
05		6	

	0	40625
	6	50000
	8	0

Замінімо остачу 14 на цифру E, і результат готовий: $1765,40625_{(10)} = 6E5,68_{(16)}$.

Щоб повернути шістнадцяткове число в десяткову систему також використовують розвинення за степенями основи: $6E5,68_{(16)} = 6 \cdot 16^2 + 14 \cdot 16 + 5 + 6 \cdot 16^{-1} + 8 \cdot 16^{-2} = 1536 + 224 + 5 + 0,375 + 0,03125 = 1765,40625$.

Окремо розглянемо *кодування від'ємних чисел*. Люди розрізняють додатні та від'ємні числа за допомогою знаків «+» та «-». В алфавіті ЕОМ є тільки два символи, тому «+» позначають нулем, а «-» – одиницею, причому в записі числа визначено спеціальне місце для знака – знаковий розряд. Для кожного числа в пам'яті відведено певну кількість байт: 1, 2, 4 тощо. Байт складається з 8 біт (чи двійкових розрядів: bit від англійського Binary digiT). Біти пронумеровано справа наліво. Найстарший з бітів є знаковим, усі інші – значущими. До речі, кількість виділених байтів (а отже, і бітів) визначає діапазон значень, які можна буде записати у пам'ять. У 1 байт можна записати цілі з діапазону від -128 до 127, в 2 байти – від -32768 до 32767 і т. д. (Байт містить 1 знаковий біт і 7 значущих, $2^7=128$; в двох байтах 15 значущих біт, $2^{15}=32768$.)

Код числа, в якому значущі розряди містять двійковий запис модуля числа, а знаковий розряд за допомогою «0» чи «1» задає знак числа (відповідно, «+» чи «-»), називають *прямим кодом*. Додатні числа завжди зображають у прямому коді.

Натомість від'ємні зображають у *доповняльному коді*, спеціально для цього створеному. Щоб отримати доповняльний код від'ємного цілого числа, треба: 1) перевести в двійкову систему модуль цього числа, заповнивши при цьому незначущими нулями ВСІ відведені для числа біти – матимемо двійковий код; 2) замінити в двійковому коді всі «0» на «1», а «1» на «0» – матимемо обернений (інверсний) код; 3) додати «1» до оберненого коду за правилами двійкової арифметики – отримаємо доповняльний код.

Нехай, наприклад, для цілого відведено 2 байти. Тоді для числа -1765 отримаємо: 1) двійковий код 0000`0110`1110`0101; 2) обернений код 1111`1001`0001`1010; 3) доповняльний код 1111`1001`0001`1011.

$$1765_{(10)} = 0000011011100101_{(2)} = 06E5_{(16)},$$

$$-1765_{(10)} = 1111100100011010_{(2)} = F91A_{(16)}.$$

Бачимо, що в знаковому розряді (першому зліва) додатного числа стоїть «0», а від'ємного – «1». Використання доповняльного коду полегшує процесорові виконання арифметичних операцій: віднімання виконується як додавання від'ємного. Наприклад, десяткове віднімання $7823 - 1765 = 6058$ із застосуванням доповняльного коду в двійковій системі еквівалентне такому додаванню:

$$\begin{array}{r}
 + \quad 0001111010001111 \\
 \quad 1111100100011010 \\
 \hline
 10001011110101010
 \end{array}$$

Найстарша цифра отриманої суми (перша зліва одиниця) виходить за межі розрядної сітки, тому втрачається (потрапляє в прапорець перенесення процесора). Знаковий розряд суми дорівнює нулю, тому вона додатня. Остаточо: $1011110101010_{(2)} = 6058_{(10)}$.

З доповняльним кодом пов'язана ще одна важлива особливість використання цілих чисел. Проілюструємо її на прикладі. Скільки отримаємо в результаті такого додавання:

127+1? Не поспішайте з відповіддю, бо вона залежить від того, якого розміру пам'ять використано для результату. Якщо 2 байти, то отримаємо, як і сподівалися, 128. Якщо ж 1 байт, то $(127+1)_{(10)} = (01111111+00000001)_{(2)} = 10000000_{(2)}$. Останнє число містить «1» у знаковому розряді і є доповняльним кодом для -128. Отож, якщо для зберігання цілого зарезервовано пам'ять недостатнього розміру, то комп'ютер може відобразити його неправильно.

Таблиця 2.3. Додавання у шістнадцятковій арифметиці

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Таблиця 2.4. Множення у шістнадцятковій арифметиці

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

У всіх позиційних системах, незалежно від основи числення, діють ті самі алгоритми виконання арифметичних дій, які ми вивчили у школі. Додавання, віднімання, множення та ділення можна виконувати у стовпчик, тільки враховувати таблиці додавання та множення відповідної арифметики, наприклад, $(1+1)_{(2)}=10_{(2)}$, $(7+3)_{(16)}=A_{(16)}$, $(9 \times 5)_{(16)}=2D_{(16)}$.

Системне та прикладне програмне забезпечення. Повернемось до обговорення програм, які керують роботою комп'ютера.

Усю сукупність програм, написаних для комп'ютерів одного типу (комп'ютерів, процесори яких «розуміють» ту саму систему команд) називають *програмним забезпеченням* (ПЗ). Багатство ПЗ здебільшого і визначає можливості машини. Для персональних комп'ютерів створено надзвичайно багато різноманітних програм. ПЗ класифікують за призначенням. Виділяють два головні класи:

- системне програмне забезпечення;
- прикладне програмне забезпечення.

До *системного ПЗ* належать.

1. Операційні системи (ОС).
2. Системи керування файлами.
3. Інтерфейсні оболонки для взаємодії користувача з ОС.
4. Системи програмування.
5. Утиліти.

Операційною системою називають комплекс керуючих програм, які, з одного боку, відіграють роль інтерфейсу між апаратурою комп'ютера і задачами користувача, з іншого – призначені для підвищення ефективності використання ресурсів обчислювальної системи та організації надійних обчислень. ОС встановлено на кожному комп'ютері, вона завантажується автоматично одразу після його увімкнення. Будь-яка прикладна програма працює під керуванням ОС. ОС – єдина програма, що взаємодіє з апаратурою комп'ютера.

Головні завдання ОС: отримання і опрацювання команд користувача; завантаження в ОП програм та їхнє виконання; забезпечення роботи систем керування файлами (а також систем керування базами даних); забезпечення багатозадачного режиму; виконання всіх операцій введення та виведення даних; розподіл пам'яті та часу процесора між задачами, що виконуються одночасно, організація обміну повідомленнями між ними; захист однієї програми від впливу іншої, забезпечення цілісності даних; опрацювання помилок, які виникають під час виконання програм; підтримка роботи систем програмування.

Призначення *системи керування файлами* – організація зручного доступу до даних, організованих як файли. Саме завдяки їй замість низькорівневого звертання до даних за допомогою зазначення конкретної фізичної адреси потрібного нам запису використовується логічний доступ за допомогою зазначення імені файлу та номера запису. Система керування файлами (СКФ) входить до складу кожної ОС, і водночас є самостійною її частиною: багато сучасних ОС можуть працювати з кількома СКФ. Так само, одна СКФ може працювати в різних ОС. Відому систему FAT (file allocation table) було реалізовано для різних ОС: FAT16 для MS DOS, super-FAT для OS/2, FAT для Windows NT.

Система FAT16 на сьогодні застаріла. Їй на зміну прийшли системи VFAT (virtual FAT) та FAT32 (повноцінна 32-розрядна файлова система). Вони підтримують довгі імена файлів, працюють з дисками великого розміру, мають ще низку важливих удосконалень і переваг. Система HPFS (High Performance File System) вперше з'явилась в OS/2. Розроблена спеціалістами IBM та Microsoft для багатозадачного режиму. Система NTFS (New Technology File System) відрізняється від попередників підвищеною надійністю, безпекою, швидкістю доступу, особливо при роботі з великими дисками; підтримує стандарт POSIX (Portable operating system for computing environments – міжнародний стандарт машинно-незалежного інтерфейсу комп'ютерного середовища); має засоби емуляції інших файлових систем; гнучко пристосовується до конкретних умов.

Головне завдання *інтерфейсної оболонки* – надати користувачеві зручні засоби взаємодії з ОС. Наприклад, різноманітні варіанти графічного інтерфейсу XWindow в системах сім'ї UNIX, PM Shell чи Object Desktop в OS/2 з графічним інтерфейсом Presentation

Manager. Графічний інтерфейс сім'ї ОС компанії Microsoft реалізують програмні модулі з іменем Explorer.

Система програмування є інструментом створення нових програм. Вона містить найперше такі компоненти: транслятор з відповідної мови, бібліотеки стандартних програм, редактор текстів, компоувальник, налагоджувач тощо. Будь-яка система програмування може працювати тільки у відповідній ОС, для якої вона і створена.

Утилітами називають службові програми, за допомогою яких можна обслуговувати саму ОС, готувати носії даних, перекодовувати дані, оптимізувати розташування даних на носіях, відновлювати втрачені файли тощо. Зрозуміло, що утиліти можуть працювати лише у відповідному операційному середовищі.

Дати вичерпну класифікацію *прикладного ПЗ* надзвичайно складно через те, що таких програм написано і використовують дуже і дуже багато. Спробуємо перелічити хоча б найпоширеніші.

1. Редактори текстів і видавничі системи.
2. Графічні редактори.
3. Табличні процесори.
4. Інтернет-браузери та поштові системи.
5. Системи керування базами даних.
6. Антивірусні програми.
7. Навчальні програми, тренажери-імітатори, оперативні довідкові системи.
8. Електронні перекладачі, системи розпізнавання текстів.
9. Інтегровані системи наукових розрахунків і проектування.
10. Музичні редактори, мультимедійні програвачі.
11. Експертні системи.
12. Спеціалізовані програми керування виробничими процесами, дослідями та експериментами тощо.

До цього скромного переліку можна було б додати ще декілька сотень категорій вживаних програм. Ми ж коротко охарактеризуємо вже згадані.

Головне завдання *редактора текстів* – надати користувачеві засоби для зручного створення та редагування файлів з текстовою інформацією. Потреба у цьому виникає кожного разу, коли ми оформляємо звіт, готуємо листа, реферат чи статтю, пишемо електронного листа, набираємо початковий текст програми тощо. Тому редактори текстів найпоширеніші програми. Вони різняться своєю спеціалізацією та переліком можливостей, часто входять до складу більших програмних комплексів: систем програмування, проектування, поштових систем тощо. Найпростіші редактори (наприклад, Блокнот чи Notepad зі стандартних програм ОС Windows) дають змогу набрати сам текст, перемістити чи розмножити його фрагмент, виконати автоматичну заміну слова чи словосполучення, зберегти його на диску – цього достатньо, щоб, наприклад, створити текст нової програми. Потужні редактори текстів (їх ще називають текстовими процесорами) мають набагато ширші можливості (наприклад, Microsoft Word): великий набір стилів, шрифтів, способів форматування абзаців, колонтитулів; засоби автоматичного створення списків, посилань, змісту; інструменти побудови таблиць, схем, діаграм; можливості впровадження зовнішніх об'єктів (формул, рисунків, звуків, відеокліпів) – всього не перелічити. *Видавнича система* разом з усіма можливостями потужного редактора текстів має засоби для задоволення потреб поліграфічного виробництва: макетування видань різноманітних форматів, виготовлення форм тощо.

Графічними редакторами називають програми для редагування зображень в електронному форматі – так званих цифрових зображень. Конкретні формати ми вже згадували. Редактор має набір інструментів для створення нових графічних композицій і для опрацювання зображень, отриманих за допомогою сканера, цифрової фото- чи відеокамери. Мабуть, кожен користувач ОС Windows знайомий з простим графічним редактором Paint, який входить до складу її стандартних програм. З «серйозних» редакторів назовемо Adobe Photoshop та CorelDRAW. Photoshop – визнаний лідер серед систем опрацювання растрової графіки. У спектрі його застосування: сканування та ретушування

фотографій, виготовлення професійних зображень для книг і журналів, підготовка зображень для публікації в Інтернет, для розповсюдження в електронному форматі PDF тощо. Однією з найпотужніших систем векторної графіки є CorelDRAW.

Поширеним способом подання інформації (числової, текстової) є таблиця. За допомогою таблиць зручно виконувати не надто громіздкі обчислення: зведення торговельного балансу, хід виконання запланованих заходів тощо. Спеціально для опрацювання інформації у вигляді таблиць використовують *табличні процесори*. Їхні головні завдання – виконання табличних розрахунків, побудова за заданими числами діаграм і графіків, ведення простих баз даних. Одна з відомих програм цієї категорії – Microsoft Excel.

Документи, опубліковані у всесвітній мережі Інтернет, написані спеціальною мовою – HTML (HyperText Markup Language). Ця специфікація дає змогу об'єднувати в одному документі текст, графіку, мультимедійні елементи, посилання на інші документи, розташовані в довільному місці мережі. *Інтернет-браузер* (WEB-браузер, програма-оглядач) розуміє метамову і вміє відобразити на екрані монітора HTML-документ. Браузер дає змогу «мандрувати» у всесвітній павутині від одного документа до іншого. *Поштову систему* використовують для обміну електронними листами. Вона має засоби для створення, надсилання, отримання, зберігання кореспонденції: текстів і різноманітних вкладень.

Система керування базою даних (СКБД) – комплекс програм для швидкого і зручного отримання інформації з бази даних, для її поповнення та модифікації. СКБД невтомно працюють у всіх сферах економіки, культури, спорту.

Комп'ютерними *вірусами* називають окрему категорію програм, головною ознакою яких є здатність самостійно відтворюватись, розмножуватись. Нешкідливі віруси нічого іншого і не роблять, але таких абсолютна меншість. Інші віруси завдають значної шкоди: класичні віруси знищують чи псують системні файли, вміст дисків зараженої машини, поштові хробаки перевантажують канали зв'язку непотрібними пересиланнями, трояни відкривають шлях для комп'ютерних злодіїв – жах. *Антивірусні програми* покликані запобігти проникненню вірусу на ваш комп'ютер, або вилікувати його, якщо це вже трапилось.

Комп'ютерна *навчальна програма* надає користувачеві додаткові можливості для успішного оволодіння новими знаннями та вміннями. Якісна програма широко використовує можливості комп'ютера для демонстрації прикладів у динаміці (в русі чи розвитку), для інтерактивного контролю за ходом навчання, для імітації різноманітних пристроїв і машин (від автомобіля до космічного корабля) тощо.

Електронний перекладач автоматично перекладає текстовий документ з однієї мови на іншу. На жаль, ці програми поки що далекі від досконалості, і результати їхньої роботи лише віддалено нагадують справжній переклад. Найлегше вони справляються з технічними текстами. Оперативні перекладачі надають переклад слова чи фрази, зазначеної користувачем на екрані. Це особливо зручно, коли працюєш в Інтернеті, з іншомовною навчальною літературою чи технічною документацією. *Система розпізнавання* перетворює графічне зображення, найчастіше відскановане, до текстового формату. Її використовують для швидкого переведення друкованого тексту до електронного вигляду, для перевірки результатів тестування тощо.

Системи наукових розрахунків пристосовано для швидкого створення складних обчислювальних програм з широкого набору стандартних алгоритмів. Такі системи містять потужні засоби, які реалізують числові та аналітичні методи розв'язування математичних задач, будують дво- та тривимірні графіки тощо. *Системи проектування* надають конструкторові зручні та наочні інструменти для побудови нових пристроїв, механізмів, будинків ландшафтів. Ще до втілення ідеї в життя можемо побачити на екрані монітора змодельований прототип, виконати усі потрібні розрахунки, навіть провести початкові випробування.

Експертні системи допомагають швидко приймати правильні рішення у відповідальних ситуаціях, коли доводиться враховувати багато різноманітних факторів. Доведено, що в такій ситуації (багато вхідної інформації і мало часу) людина помиляється у чотирьох випадках з п'яти. Запобігти цьому допомагають «незворушні» комп'ютерні експертні системи. Головні галузі їхнього використання – охорона здоров'я, військова справа, керівництво надскладними промисловими об'єктами. Експертна система складається з бази знань, машини виводу і бази даних. Базу знань створюють програмісти разом зі спеціалістами конкретної прикладної галузі, формалізуючи та структуруючи їхні знання і досвід – це найважча частина роботи. Машина виводу – це програма, що сприймає запити користувача системи, ставить додаткові запитання і на підставі отриманих відповідей та знань, закладених в базі, формулює відповіді-рекомендації на запит. Інформацію про вирішені проблеми система накопичує в базі даних, щоб враховувати її в майбутньому, тобто для самонавчання і самовдосконалення.

Системи програмування. Етапи створення програми. Ми згадали досить багато різноманітних програм. Кожна з них пройшла певний шлях від задуму до остаточного втілення в життя. Які ж головні етапи створення комп'ютерної програми? Які «інструменти» використовує програміст у своїй роботі?

На самому початку історії розвитку ЕОМ перші програми записували лише в кодах процесора. Одну команду задавав довгий ланцюжок нулів та одиниць, який містив код операції, адреси операндів і результату. В одному рядку програми – код однієї команди. Ціла програма могла містити тисячі і десятки тисяч рядків. Мабуть, зайвим буде пояснювати, що праця програміста була надзвичайно копіткою, вимагала вичерпного знання влаштування машини і кодів усіх її команд. І написання програми, і виправлення помилок займало дуже багато часу. Щоб задовольнити постійно зростаючі потреби у нових програмах, потрібні були дієві способи підвищення продуктивності праці програміста, нові виразні засоби для зображення та структурування складних алгоритмів. Саме тому було винайдено мови програмування. *Мова програмування* – формальна мова зі строго визначеними синтаксичними та семантичними правилами для опису даних і алгоритмів їхньої переробки. Вона відіграє роль посередника між системою команд машини і звичайною мовою людей.

Розрізняють мови програмування низького рівня, високого рівня, декларативні мови. Мова *низького рівня* (її ще називають *мовою асемблера*) близька до машинної. Одній асемблерній команді відповідає одна машинна команда, і навпаки. Тому між системою команд процесора і мовою його асемблера існує взаємно однозначна відповідність. Мову асемблера відрізняє від машинної використання мнемонічних позначень для операцій та імен для даних, наприклад, ADD – додати, SUB (subtract) – відняти, CMP (compare) – порівняти тощо. Мова *високого рівня* ближча до звичайної мови. Записані нею інструкції «обчислити вираз», «виконати розгалуження, повторення, підпрограму» тощо подібні до речень наукового тексту. Одній інструкції мови високого рівня здебільшого відповідає ціла група машинних команд. Декларативна мова описує сутності проблеми і їхні взаємозв'язки за допомогою абстракцій високого рівня.

Сьогодні немає жодного комп'ютера, який би безпосередньо сприймав хоча б одну з мов програмування – як і раніше, всі машини послуговуються власною. Щоб ЕОМ могла виконати деяку програму, треба спочатку перекласти її на машинну мову. Таку роботу виконують спеціальні програми – *транслятори*. Зверніть увагу (!): програма допомагає створювати нові програми. Трансляція текстів програм можлива завдяки універсальності комп'ютерів і другому принципу фон Неймана – принцип збереженої програми, – про які ми вже говорили раніше. Справді, комп'ютерній програмі байдуже, що є її вхідними даними, то чому б це не міг бути текст іншої програми? Насправді для цього немає жодних застережень. Цей текст може бути текстом нового транслятора. Звичайно, найперші транслятори було написано машинною мовою.

Розрізняють два способи трансляції: компіляція та інтерпретація.

Компілятор перекладає весь початковий текст програми, написаної мовою високого чи низького рівня, у машинні коди і записує проміжний код програми – він ще не зовсім готовий до виконання, бо потребує налаштування остаточних адрес розташування в пам'яті даних програми, зв'язків між підпрограмами (у тім числі з модулями зовнішніх бібліотек) тощо. Таке налаштування також виконує спеціальна програма – *редактор зв'язків*, або *лінкер* (від англійського *linker*), який і створює остаточний (виконуваний) код програми. Надалі готову програму можна переносити на різні машини і виконувати її там, навіть, якщо на новій ЕОМ не встановлено компілятор.

Варто зауважити, що назва «компілятор» закріпилася за компіляторами з мов високого рівня. Компілятор мови низького рівня називають *асемблером*.

Як ми вже зазначали, компіляція не єдиний спосіб виконання програм. Його альтернативою є інтерпретація. Програма *інтерпретатор* імітує на звичайній ЕОМ машину, яка розуміє команди мови високого рівня. Для цього інтерпретатор для кожної інструкції з програми будує відповідну послідовність машинних команд і одразу ж надає комп'ютерові для виконання. Інтерпретована програма виконується одразу ж, без затрат часу на попереднє компілювання, редагування зв'язків, обов'язкове виправлення всіх синтаксичних помилок. Такий спосіб досить зручний для створення нових програм – швидка робота і поступове виправлення. Проте інтерпретація має суттєві недоліки: на нову машину програму можна перенести тільки разом з інтерпретатором; інтерпретатор ніде не запам'ятовує побудованих послідовностей машинних команд, тому змушений будувати їх знову і знову для тієї самої повторюваної інструкції, хоча б вона виконувалась і сотні разів, а це дуже сповільнює виконання.

Після всіх пояснень можемо дати такі загальні означення:

Компілятор – це програма для машини *A*, яка перетворює програму для машини *B* у програму для машини *A*.

Інтерпретатор – це програма для машини *A*, яка виконує на ній команди програми для машини *B*.

Тепер охарактеризуємо кожен з етапів створення програми.

1. *Формулювання задачі*. Програміст повинен дуже добре зрозуміти, що саме він програмує, що є вхідними даними, а що – результатом, у яких форматах вони відображатимуться, які вимоги ставлять щодо продуктивності майбутньої програми тощо. Усі ці деталі спільно з'ясовують замовник і виконавець. Нерідко постановку задачі уточнюють і в ході розробки програми, бо існування проекту майбутньої програми покращує розуміння проблеми.
2. *Побудова моделі* реального процесу, системи, об'єкта. Моделлю можуть бути діаграми, математичні абстракції, формалізовані словесні описи тощо.
3. *Проектування алгоритму* тісно пов'язане з попереднім кроком. У ході проектування визначають структури даних, які відображатимуть деталі моделі, та структури керування, що реалізують взаємодію між ними. Ефективність, зручність у користуванні, надійність майбутньої програми на три чверті визначається досконалою завершеністю саме цих трьох етапів.
4. *Набір початкового тексту програми* виконують за допомогою будь-якого зручного редактора текстів і зберігають його у файлі. Цьому текстовому файлові призначають тип відповідно до мови програмування, наприклад, тип '*pas*' – програма мовою Pascal (чи Object Pascal), '*for*' – мовою Fortran, '*c*' – мовою C (чи C++), '*bas*' – мовою Basic тощо. Спеціалізований редактор текстів виконує синтаксичний аналіз тексту програми «на льоту» – у ході набирання – виокремлює ключові слова, константи, підказує можливі способи опрацювання структур даних (об'єктів) тощо. Це полегшує написання програми, дає змогу ще на початковому етапі виявляти помилки.
5. *Компіляція програми*. Ми щойно обговорили, навіщо потрібен цей крок. Він зможе відбутися тільки тоді, коли текст програми не міститиме синтаксичних помилок: транслятор спочатку виконує синтаксичний розбір тексту, в ході якого виявляються помилки, якщо вони є (саме тому початківці хибно трактують трансляцію як перевірку

синтаксису). Транслятор завжди повідомляє про виявлені суперечності і намагається діагностувати причину помилки. Діагностичне повідомлення є лише підказкою для програміста і свідчить про те, які синтаксичні правила порушено. Деякі транслятори припиняють роботу після виявлення першої помилки, інші – опрацьовують увесь текст до кінця і звітують про всі виявлені помилки. Важко сказати, який спосіб ліпший, бо наявність наступних помилок може бути зумовлена першою. За відсутності синтаксичних помилок транслятор створює новий файл типу '*obj*' з проміжним, так званим об'єктним, кодом програми. Об'єктний код є універсальним (стандартним) для більшості різних трансляторів. Винятком є Borland Pascal Compiler, який використовує власний проміжний код (мабуть, для збільшення ефективності).

6. *Редагування зв'язків.* Лінкер пов'язує в одне ціле відкомпільовані модулі, розташовані в бібліотеках стандартні підпрограми й утворює файл з готовою до виконання програмою. Тип такого файлу – '*exe*'. На цьому етапі можна поєднувати в одну програму модулі, написані різними мовами. Наприклад, Fortran-підпрограми, що реалізують числові методи, С-підпрограми, які реалізують інтерфейс, та частини загального алгоритму, які потребують найефективнішої реалізації, тому написані мовою асемблера. Редактор зв'язків також може виявити помилки у посиланнях на імена підпрограм і форматах їхніх параметрів – їх обов'язково треба виправляти.
7. *Тестування та налагодження.* Кожна відтрансльована програма подібна на правильну – принаймні, нам би так хотілося. На жаль, так є далеко не завжди. Трансляція без помилок засвідчує тільки відсутність помилок у синтаксисі та зв'язках. Щоб виявити можливі логічні помилки, виконують вичерпне тестування програми. Для цього спеціально створюють пакети тестових вхідних даних, які б охоплювали всі типові (правильні та помилкові) комбінації вхідних значень алгоритму. Очікувані (еталонні) результати кожного тестового випробування визначають разом з вхідними даними ще перед тестуванням. У разі незбігання результатів тесту з еталоном шукають помилки в програмі. Це нелегка справа. У пошуках може допомогти покрокове (по одній інструкції) виконання програми та спостереження за поточними значеннями її змінних, встановлення в програмі точок переривання для тимчасової зупинки її виконання, виведення на друк протоколу обчислень тощо.

Для дослідження функціонування і налагодження зовнішніх модулів іноді використовують *дезасемблятор* – програму, яка за виконуваною програмою відтворює її вихідний код. Тобто виконує дію, обернену до трансляції. Текст програми отримують мовою низького рівня – відтворити текст іншою мовою просто неможливо.

Бачимо, що процес побудови програми доволі тривалий і потребує використання багатьох службових програм. Для зручної взаємодії з ними створено *інтегровані середовища програмування* (ІСП) – програми-оболонки, які поєднують в одне ціле спеціалізований редактор текстів, компілятор (можливо, й інтерпретатор), вбудований асемблер, редактор зв'язків, програму-налагоджувач, розвинену довідкову систему, засоби побудови інтерфейсу програми. ІСП до певної міри автоматизує виконання кроків 4–7. Останнім часом з'явилися і використовуються також середовища для моделювання проблем і проектування алгоритмів (кроки 2, 3). Наприклад, UML (Universal Modeling Language) методологія графічного моделювання на основі об'єктно-орієнтованого аналізу використовує визначений набір діаграм. За побудованою моделлю-діаграмою можна автоматично згенерувати значну частину коду алгоритму.

Нагадаємо, що кожен етап побудови програми, починаючи від постановки задачі, треба детально документувати. Описують усі вимоги до програми, усі складові моделі та процеси, які в ній відбуваються. Текст програми ілюструють коментарями, щоб пояснити призначення усіх її частин: змінних, підпрограм тощо. Документація програми повинна містити детальний опис щодо правил її тестування та використання.

Файлова система ОС. Уся інформація на дисках комп'ютера (програми, тексти, числові дані, рисунки) зберігається у файлах. *Файлом* називається іменована цілісна сукупність

даних на зовнішньому носіїв інформації. Про що говорить нам це означення? По-перше, кожен файл має ім'я. По-друге, файл містить сукупність *однотипних* даних. По-третє, файл займає певне *місце* на диску. Яке саме місце – справа операційної системи. Нерідко файл займає навіть декілька несуміжних ділянок диска, але сама ОС захищає користувача від деталей конкретної організації – йому достатньо знати ім'я та розташування файла, а номери доріжок і секторів диска, зайнятих файлом, можуть цікавити хіба що системного програміста у спеціальних випадках.

Ім'я для файла вибирають не довільно, а так, щоб воно нагадувало користувачеві, що саме зберігається у файлі. *Ім'я файла повинно бути змістовним*. Давніші файлові системи використовували винятково короткий формат імені файла, згідно з яким ім'я може містити від однієї до восьми латинських літер, цифр і знаків «-», «_», розташованих у довільному порядку. Сучасний довгий формат допускає використання кирилиці та пропусків, довжина імені до 255 символів – так набагато зручніше давати файлам змістовні назви. Проте завжди знайдеться програма, яка відмовиться розуміти таке ім'я, особливо, якщо у ньому трапляються українські літери «і», «ї», «є», «г». Вибір залишається за користувачем: коротке ім'я вбереже від проблем, а зручне довге завжди можна змінити, якщо якісь проблеми все-таки виникнуть.

Тип файла визначають за типом інформації, що записана в ньому, і зазначають поряд з його іменем як розширення. Розширення від імені відокремлюють крапкою. Воно може містити до п'яти літер. Сьогодні склалися певні традиції у використанні типів файлів: *.txt* – текстовий файл; *.cpp* – текст програми мовою C++; *.bas* – програма на Бейсіку; *.exe* – готова до виконання програма; *.hlp* – довідковий файл; *.bmp* – графічний файл у Bit Map форматі; *.htm* – інтернет-сторінка; *.doc* – документ, створений редактором текстів, можливо, однією з версій Microsoft Word; *.xls* – електронна таблиця; *.tmp* – тимчасовий файл тощо. Приклади повних імен: *message.txt*, *prog1.cpp*, *spider.exe*, «03 - Комп'ютерні мережі.doc», *read.me*.

Отже, файли розташовані на дисках і організовані в деревовидну структуру. Розглянемо її.

Вершиною всієї ієрархії є *Робочий стіл*. Зазвичай на ньому розташовано Інструменти, наприклад, «Мій комп'ютер», «Смітник». (Звичайно, поряд з ними користувач може помістити і деякі файли, але не дуже розумно захищувати Робочий стіл всячиною.) Інструмент «Мій комп'ютер» дає доступ до дисків комп'ютера. В операційній системі вони іменуються великими літерами латинського алфавіту з двокрапкою. У перших моделях комп'ютерів *A:* та *B:* – імена дисководів гнучких дисків (у наш час не використовуються); *C:*, *D:*, *E:*, ... – жорстких, оптичних, мережевих. У більшості ПК є лише один жорсткий диск, але спеціальні програмні засоби дають змогу розбити його на частини, так звані логічні диски, і працювати з кожним з них, як із самостійним диском. Таке розбиття роблять, зокрема, для того, щоб виділити системні диски – диски, призначені для зберігання програмного забезпечення ПК – і диски користувачів. Виконувати будь-які маніпуляції з файлами на системному диску *суворо заборонено!*

Значення імен дисків на деякому комп'ютері може бути, наприклад, таке: *C:* – жорсткий системний; *D:*, *E:* – диски користувачів, *F:* – CD ROM; *G:*, *H:* – мережеві диски.

Кожен магнітний диск комп'ютера повинен бути форматуваним. У ході форматування за допомогою спеціальної програми на диску позначають доріжки (концентричні кола) і сектори, створюють *кореневий каталог*. Кореневий каталог позначають символом «\», наприклад, «*D:*» – кореневий каталог диска *D:*. Записати будь-які дані можна тільки на форматований диск. Форматування виконує виробник диска. Звичайно, кожен диск можна переформатувати (тобто, повторно виконати форматування), проте варто пам'ятати, що під час форматування вся інформація, що раніше була записана на диск, втрачається безповоротно.

Вміло підібрані імена файлів можуть значно полегшити роботу з ПК, допомогти користувачеві зорієнтуватися у великому обсязі різномірної інформації, що зберігається на дисках. Проте сучасний комп'ютер зберігає тисячі файлів. Щоб дати з ними лад, одних імен замало. Каталоги файлів дають змогу структурувати, впорядковувати дисковий простір.

Якщо створюють файл чи змінюють його вміст, то ОС автоматично реєструє в каталозі ім'я, тип, розмір, дату і час модифікації, атрибути доступу до файла.

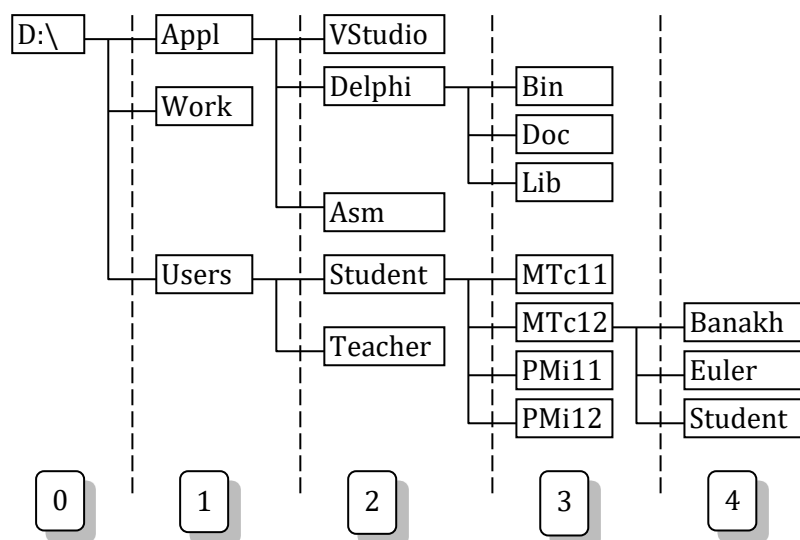


Рис. 2.1. Приклад дерева каталогів. Цифрами позначено рівні:

0 – кореневий каталог; 1, 2, 3, 4 – підкаталоги відповідного рівня (Appl – підкаталог першого рівня, PMi11 – третього рівня тощо)

Каталог (директорія) – це іменована область дискової пам'яті, яку створюють для реєстрації і збереження групи файлів, об'єднаних за деякою ознакою (наприклад, файли студентів однієї групи, файли однієї програмної системи). Кожен файл може бути лише в одному каталозі. Каталог має ім'я (в короткому форматі з 1–8 символів) і може бути зареєстрованим в іншому каталозі. Тоді вони називаються відповідно *підкаталогом* і *надкаталогом*. Підкаталог входить у надкаталог як єдине ціле. На кожному диску завжди є кореневий каталог. У ньому реєструють важливі файли та підкаталоги першого рівня. В підкаталогах першого рівня реєструють файли та підкаталоги другого рівня і т. д. Так утворюється ієрархічна деревовидна файлова система. У конкретній ситуації вона може виглядати так, як на рис. 3.2.

Ланцюжки підлеглих каталогів позначають їхніми іменами, розділеними символом "\". Наприклад, D:\Work; D:\Appl\Delphi\Bin. Зауважимо, що каталоги D:\Users\Student і D:\Users\Student\MTc12\Student – це *різні* каталоги.

В описаній файловій системі можлива ситуація, коли одноіменні, але різні за змістом, файли фігурують у кількох каталогах. Для точної ідентифікації файла треба, крім імені, зазначити його точне розташування – ланцюжок підлеглих каталогів, який називається *маршрутом (шляхом)* до файла. Маршрут можна використовувати як префікс імені файла. У цьому разі він відокремлюється від імені символом "\". Наприклад, C:\Office\word.exe або E:\INA21\RAFAELO\first.txt.

Важливим також є поняття *шаблону імен* (шаблони використовують для позначення групи файлів, наприклад, під час пошуку файла за іменем). У шаблонах імен використовують два спеціальні символи "*" та "?". Перший з них означає довільну кількість довільних символів в іменах чи типах файлів, другий – будь-який один символ. Наприклад,

pr*.* – всі файли з іменами, що починаються на pr;

*.txt – всі файли типу txt;


. – всі файли поточного каталога;

*. – всі файли без типу;

???bas – всі файли типу bas з не більш ніж трілітерними іменами;

a?.* – всі файли, імена яких починаються на "a" і складаються з однієї чи двох літер.

У всіх наведених прикладах ми зазначали розширення імен файлів, але треба зауважити, що у більшості випадків Ви не побачите їх ні у вікні інструмента «Мій комп'ютер», ні у вікні Провідника (Explorer). Замість зареєстрованих у системі розширень ОС відображає біля імен відповідні значки (піктограми, іконки).

Папки (Folders) – це аналоги каталогів. Всі вони мають значок . Папки призначені для зберігання споріднених значків, якими у Windows позначені всі апаратні та програмні компоненти комп'ютера: принтери, файли, ярлики, аплікації, інші папки тощо. Ієрархічна система папок фактично утворює файлову систему комп'ютера. Її структуру можна переглянути засобом Мій комп'ютер або Провідником. Папки відкривають і закривають подвійною фіксацією мишки або клавішею Enter. Їх можна створювати, вилучати, копіювати, переміщувати, переіменовувати.

Ярлики (Shortcuts). Ярлик – це посилання на часто вживаний значок, створене для швидкого доступу до нього. Для одного документа чи пристрою можна використовувати декілька ярликів, розташованих для зручності у різних місцях. Значок ярлика повторює значок-оригінал. Він виглядає як і значок оригіналу, за винятком маленької стрілочки у нижньому лівому кутку. Подвійне клацання мишкою на ярлику призводить до того, що система знаходить відповідний файл і відкриває його (запускає на виконання відповідну програму).