В. Х. Власюк

ОСНОВИ ПРОГРАМУВАННЯ АЛГОРИТМІЧНИМИ МОВАМИ

Міністерство освіти і науки України

Вінницький національний технічний університет

В. Х. Власюк

ОСНОВИ ПРОГРАМУВАННЯ АЛГОРИТМІЧНИМИ МОВАМИ

Затверджено Вченою радою Вінницького державного технічного університету як навчальний посібник для студентів спеціальності «Програмне забезпечення автоматизованих систем». Протокол №8 від 27 березня 2003 р.

Вінниця ВНТУ 2004

УДК 681.3.06
В 58

Рецензенти:

В.С. Абрамчук, кандидат фізико-математичних наук, професор
Т.О. Савчук, кандидат технічних наук, доцент
С.В. Юхимчук, доктор технічних наук, професор

Рекомендовано до видання Вченою радою Вінницького державного технічного університету Міністерства освіти і науки України

Власюк В.Х.

В 58 **Основи програмування алгоритмічними мовами.** Навчальний посібник. – Вінниця: ВНТУ, 2004.- 157 с.

В посібнику розглянуті виразні засоби алгоритмічних мов, які є основою програмування, і їх розвиток. Посібник розроблений у відповідності з планом кафедри та програми до дисципліни "Проектування інструментального програмного забезпечення", "Основи програмування і алгоритмічні мови". Для студентів денної та заочної форм навчання.

УДК 681.3.06

ЗМІСТ

Введення в спеціальність

Дисципліна "Основи програмування і алгоритмічні мови" (і цей посібник також) фактично є *введенням в спеціальність*. Тому спочатку даємо множину слів – "остов" мовного середовища, оточення, в якому проходить навчальний процес. Ця множина буде вміщувати слова не тільки з формальних мов, а й природних. Останні, здебільшого іншомовного походження, мають багато значень, отриманих в процесі еволюції їхнього смислу.

Під час навчання поступово розвивається, формується сприйняття смислу слова. Цей розвиток визначається не лише контекстом, а й рядом інших причин. З цих причин може трапитись, що в якомусь використанні (тран)сформоване значення слова буде протилежним значенню, прийнятному до цього використання. Тоді всі інші значення слова будуть сприйматись як неправильні, а висловлювання з ним – як незрозумілі, не зважаючи на те, що тепер це вже значення ніби різних слів.

Не важко знайти подібні тлумачення для більшості слів із даної нижче множини слів – як з природних так і формальних мов.

Далі в фігурних дужках – авторська етимологія. Приклад: {Знак, значення. Звичай, звичайно, звикання. Випадок, жереб.}.

Первісне чи буквальне значення слова "програма" в грецькій мові – "те, що написано, назначено" (рос. предначертание) має подвійний вплив (наперед визначеність і невідворотність) на вибір професії програміста і його долю.

Психологія програміста

Наведемо головні психологічні типи програмістів.

Інженер безумовно заглиблений в реальний світ. Його творчість обмежена законами цього світу. Він, врешті-решт, може робити лише те, що відповідає цим законам. Багато чого йому треба робити і одних лише міркувань не досить [4, С. 159].

Програміст – творець світів, в яких він сам є єдиним законодавцем (також як і творець будь-якої гри). Але світи практично необмеженої складності можна створювати у формі програм для обчислювальних машин (комп'ютерів). Більш того (і саме це є вирішальним моментом), визначені і побудовані таким чином системи виконують власні запрограмовані сценарії [4, С. 160]. Жоден драматург, режисер чи імператор ніколи не мали б такої абсолютної влади і таких неухильно

слухняних підлеглих. Було б вражаюче, якби зауваження лорда Актона, що влада розбещує, виявилось би не застосовне, коли всемогутність досягається так просто. Розтління, спричинене, породжене всемогутністю програміста проявляється у формі, повчальній для сфери значно більш ширшої ніж світ комп'ютерів.

Далі Дж. Вейценбаум описує ознаки "комп'ютерних наркоманів" – *одержимих програмістів* (МТІ 1976 р.). Це явище спостерігається в усьому світі.

Як відрізнити одержимого програміста від простого, відданого своїй справі *працелюбного програміста-професіонала*. В першу чергу, на основі того, що звичайний професійний програміст займається задачею, яка потребує розв'язку, а одержимий програміст розглядає задачу як привід для звертання до комп'ютера. Зазвичай одержимий програміст – чудовий "технар", що відмінно, з усіма подробицями знає як працює його комп'ютер, його периферійне обладнання, операційну систему і тому подібне.

Такий програміст береться за систему, яка, як він вважає, для здійснення потребує знань лише в області комп'ютерів, програмування і тому подібних предметів. "Працювати" – це не те слово, яке він використовує; те, що він робить, він називає "хакеруванням". "To hack", згідно словнику, – це "розсікати безладно, невміло чи ніби багатократними ударами якого-небудь рубаючого інструмента".

В [4, С. 165] відмічається, що не всі *хакери* страждають патологією одержимого програміста. Справді, якби не ця вищою мірою творча робота людей, що гордо називають себе хакерами, то небагато б із тогочасних (витончених) систем з розділенням часу, деяких трансляторів комп'ютерних мов та систем комп'ютерної графіки взагалі існувало б.

Хакер "діє без певної мети", він не здатний поставити перед собою ясно сформульовану довготривалу мету і виробити план її досягнення, оскільки він володіє лише вмінням, а не знанням. Він не має нічого, що він міг би аналізувати чи синтезувати; коротше кажучи, у нього немає предмета для побудови теорій.

Одержимий програміст шукає у комп'ютера не задоволення, а підбадьорення. Такий же невмолимий і неусвідомлений потяг до втішення, характерний для одержимого гравця. Для одержимого гравця гра – це все.

Зловмисники. До появи персональних комп'ютерів, комп'ютери були вірними і надійними помічниками. Потім комп'ютери стали не

передбачувані і небезпечні. Так, хворе самолюбство в одних випадках, бажання виділитись в других і жага помсти в третіх (є ще захист власного продукту) – породили вандалів в поважному середовищі програмістів. Їх винахідливість не знає меж.

Сучасні комп'ютерні віруси – це програми, які не тільки розмножуються і живуть самостійним життям, а й які обманюють, приховуються, вбивають інші програми. "Інтелектуальні" віруси виконані за Stealth-технологією [14, С. 84]. Вірусом називають програму, яка проти бажання користувача комп'ютера виконує дії, що заважають його нормальній роботі. Характерні риси: втручання в інші програми, в пам'ять, зовнішні накопичувачі; шкідливі наслідки: сповільнення роботи комп'ютера, пошкодження програм і даних, спотворення результатів введення-виведення, засмічення оперативної пам'яті і зовнішніх носіїв.

Боротьба з вірусами справа не стільки захоплююча, скільки складна, вимагає особливої уваги, терпіння, вдумливості і твердих знань. Методи боротьби: слідкування, ловля на "живця", обшуки виконуються антивірусними програмами на диску і в пам'яті комп'ютера [14, С. 3].

"Дозволяє" появу зловмисників і "*мисливців*" робота групи користувачів на "персональному" комп'ютері і доступність недосконалостей операційних систем. В світі програмістів повинно бути місце для всіх.

Робота програміста

Термін "гумор", має буквальний переклад ("волога"), який на перший погляд відмінний від двох його значень [7]. У визначенні і використанні алгоритмічних мов (це добре видно з Алголу 68 [8]) важливі всі сторони терміну. Перекладачі, редактори літератури з програмування [10] й самі автори зазначають, що для її читання необхідне почуття гумору.

"Жарт, як і справедливість, народжується разом із мовою" [Льюис К.С. Хроники Нарнии. Племянник чародея. Пер. с англ. – М.: «Сов. Композитор», 1992. – 80 с., С. 51]

В Україні було створено кілька мов програмування. Найбільш відома – *Аналітик* була реалізована на українському комп'ютері "Мир 2", який мав введення-виведення з магнітних карт і паперової стрічки. Мова використовувалась, наприклад, для розв'язання задач статистики і теорії ймовірності. В нашому університеті на початку 70-х мову Аналітик активно використовували В.Д. Дель, П. Сергієнко.

В інституті кібернетики АН Української РСР під керівництвом

К. Ющенка було реалізовано Фортран для комп'ютерів серії "Мінськ" (Фортран, що розроблявся для використання в обчислювальній математиці і обчислювальній фізиці, застосовувався значно ширше. Вінниця, 1972 р. "БИВЦ").

Комп'ютер "Промінь" працював в нашому університеті ще в 1973 році і мав для програмування комутаційну дошку. Програма набиралась на ній пластинками з отворами. "Програмувались" навіть тригонометричні функції. Розробка таких комп'ютерів може бути підтвердженням неочевидності принципу фон Неймана. На кількох кафедрах були комп'ютери "Наірі" (Єреван) з мовою Бейсик.

Автору посібника невідомо, чи були в Україні спроби створити універсальні мови. Назви мов родини мови Сі були переважно однобуквені (C++, C--, C+. Використання знаку "+" (лат. plus – більше) пояснювалось по-різному. Мінус означав подальше зближення з Асемблером). В університеті мова вперше використана 1983 року, як і Паскаль, для ЕОМ СМ 1420.

Інший напрям вибору імені для мови – прізвища. Дуже вдало вибрана назва мови на честь Власа Паскаля. Паска – слово українське (гебрейське Pesach – "перескочення", pasach – "він перескочив", pasachti – "я перескочив"). Розробники мови Ада не наважились взяти прізвище доньки лорда Байрона, Ади Августи, графині Лавлейс (Lavelace). Ада "програмувала" чи описувала нереалізовану аналітичну машину Ч. Бебіджа (Ch. Babbage).

Мова послідовно перейменовувалась таким чином: DoD-1 (Departament of Defense – Міністерство оборони), "Strawman" (Солом'яна людина"), "Woodeman" ("Дерев'яна людина"), "Tinman" ("Олов'яна людина"), "Ironman" ("Залізна людина") і, нарешті, "мова Ада" [15, С. 285]. Важливість назви мови підкреслює ще одна початкова назва мови Ада – Зелена (Green) [15, С. 171].

Особливості праці програміста. На відміну від інших видів розумової праці, наприклад, інженера або письменника, праця програміста найбільш близька до праці математика. Але й тут є суттєва відмінність. В інженера, письменника, математика є можливість створювати внутрішні зорові моделі (нехай не завжди одразу адекватні) результату власної роботи. Абстрактність програмування є його важливою рисою.

Майєрс [9, С. 21] переймається запитаннями: "Чи не є щось унікальне в професії програміста, що приваблює до неї людей за своєю

природою особливо схильних робити помилки? Чи не має програмне забезпечення якісь внутрішні властивості, що пояснюють його особливу піддатливість до помилок? Чи здатні ми якось впливати на ці властивості і чи вони нам не підвладні?"

Далі, досліджуючи причини помилок, приходить до висновку, що помилки в ПЗ цілком і повністю є наслідком недосконалості людського розуму. Засоби представлення проекту на різних рівнях (мови описів і програмування) впливають на процес перекладу. Процес проектування і властивості самого проекту (наприклад, його складність) також суттєво впливають на помилки.

Інший погляд на помилки програмістів (розділяє автор посібника): Із спостережень за роботою програмістів, можна припустити, що помиляються вони не набагато частіше від саперів. Обережність – одна з їх головних особливостей (і нерішучість, забудькуватість).

Л. Ейлер (Leonhard Euler, нім. Eule*f* – сова) мучився від страху перед помилками в своїх роботах. Зараз програміст повідомлення про більшість своїх помилок отримує майже миттєво. В працях математика помилки виявлять тільки творчі нащадки. Помилка програміста може будь-коли несподівано "вибухнути" (іноді в буквальному розумінні, разом із системою (Є випадки, дивись [9]).

Загальна властивість розумової роботи – захоплення "процесора". Ресурси мозку людини, робочого інструменту свідомості, зайняті абстракціями. Як не дивно на перший погляд, роздуми сучасної людини в основному конкретні, а первісної – абстрактні. Хтось (не виключено, відомий християнський автор) називав стародавніх греків і римлян біологічними автоматами, дуже подібними до тварин (що вже казати про решту тогочасного людства).

Сантехнік чи шахтар під час роботи можуть думати про що завгодно. У людини розумової праці робочий час – це вирвані роки, на які вона ніби зникає із реального життя. Французький письменник і філософ-просвітитель (інженер людських душ) Вольтер (Voltaire) (Справжнє ім'я Марі Франсуа Аруе, Arouet) в реальному житті увічнив нашого Ейлера в образі Акакія Акакієвича (а Н. Вірт Ейлера – мовою) [Рюдігер Тіле, Вірт].

Багатозадачний режим роботи мозку потребує відновлення, яке не досягається використанням хімічних чи біологічних речовин. Людство для цього нічого кращого, ніж гумор, винайти поки що не здатне. Навіть жалюгідний гумор програмістів має дуже велике значення.

Здається, Брукс-молодший пропонує програмістові, коли не видно виходу, прорубати стіну і поставити ще одні двері в кімнату, де той працює. Або хоча б перефарбувати, ті, що є, в інший колір. Робити так, поки не стане зрозуміло, в чому справа. Взагалі, зовні стан і поведінка типового програміста були дуже схожі зі станом і поведінкою розумово відсталої людини. Ті ж "кози", бульбашки слини, те ж мугикання тощо. Так нервова система звільняється від надмірного перевантаження.

Особливості марної праці програмістів. Працю будівельників можуть змарнувати війна, стихійні лиха. Працю програміста може знищити навіть мода, не кажучи вже про самих програмістів. Але "добре виконана робота ніколи не пропаде. Завжди хтось знайдеться, що вона йому сподобається і він охоче буде вважати її своєю" (М. Булгаков).

Така неприваблива картина означає, що інженером-програмістом, розробником можна стати, лише ним народившись. Взагалі, все, що є в посібнику, стосується лише розробників і практично не торкається тих, кого автор посібника теж прираховує до програмістів.

1. Слова із мовного оточення

1.1. Навчання

Словник цього розділу невеликий, тому слова в ньому не в алфавітному порядку, а за можливим очевидним смисловим зв'язком. З кількох значень слова взяті лише ті, що цікаві для даного введення. Поділ на підрозділи – умовний. В основному використано довідкове видання [12]. Статті словника іншомовних слів складаються з пояснення слова заголовка, етимологічної довідки (походження іншомовного слова) та дефініції (визначення).

Інструкція (від лат instructio – введення, настанова) – керівні вказівки, детальні настанови, зведення правил для виконання якоїсь роботи.

Кваліфікація (від лат. qualis – який, якої якості і ...фікація (лат. ... ficatio від facio – роблю) – в складних словах означає здійснювання, втілювання) ступінь і вид професійної виучки працівника, наявність в нього знань, уміння, навичок, потрібних для виконання певної роботи {Слова *якість* і *який* однокорінні}.

Професія (від лат. professio – офіційно зазначене заняття,

спеціальність) – рід трудової діяльності, що вимагає певних знань і трудових навичок і є джерелом існування. {Теж facio}.

Спеціалізація (від лат. specialis – особливий) – 1) Оволодіння спеціальними знаннями в певній галузі. 2) Зосередження діяльності на певному заняттi, спеціальності.

Спеціальний (лат. specialis) – 1) Виключно для чогось призначений. 2) Пов'язаний з окремою, відособленою галуззю, властивий тій чи іншій спеціальності.

Спеціальність – 1) Комплекс набутих людиною знань і практичних навичок, що дає їй можливість займатись певним родом занять у якійсь галузі діяльності. 2) Окрема, відособлена галузь науки, техніки, мистецтва тощо. Всяка самостійна професія, основна кваліфікація.

Fach n – (нім.) – предмет (*навчання*), спеціальність. Facharbeiter – кваліфікований робітник. {Фах, теж facio}.

Спеціаліст – представник якоїсь спеціальності, особа, яка володіє професійно певним фахом.

Інженер (франц. ingénieur, від лат. ingenium – здібність, винахідливість) – спеціаліст з вищою технічною освітою {Здібність, вроджена здатність}.

Факультатив (франц. facultatif – необов'язковий, вид лат. facultas – можливість) – понадпрограмний, необов'язковий для відвідування навчальний курс.

Факультет [нім. Facultät, від лат. facultaš (facultatis) – спроможність, здатність] – підрозділ (частина, відділ) вищого навчального закладу, де викладають певні предмети, готують фахівців відповідного профілю.

Універсальний (лат. universalis) – всеосяжний, різнобічний, загальний;

Університет [нім. Universität, від лат. universitas (universitatis) – сукупність].

Кафедра (грец. καθέδρα, букв. сидіння, стілець) – 2) У вузах – основна навчально-наукова група.

Лекція (від лат. lectio – читання) – 2) Виклад програмної теми предмета або проблеми (навчальна, академічна лекції).

Лектор (від лат. lector – читець, читач) – 1) У давніх римлян слуга-читець (невільник). 2) Особа, яка читає лекцію публічно за програмою навчального закладу тощо.

Дисципліна (від лат. disciplina – вчення, виховання, розпорядок) –

2) Окрема галузь наукового знання.

Екзамен (від лат. examen – зважування, дослідження, випробування) – перевірка, іспит з якого-небудь навчального предмета {Іспит, испытание}.

Колоквіум (від лат. colloquium – співбесіда) – 1) Одна з форм навчання. 2) Вид екзамену. 3) Збори, на яких заслуховують і обговорюють наукові доповіді.

Атестація (від лат. attestatio – посвідчення) – визначення кваліфікації працівника, відзив про його здібності, ділові та інші якості.

Атестат (від лат. attestor – підтверджую, посвідчую) – 1) Свідоцтво про закінчення навчального закладу.

Тест (від англ. test – випробування) – коротке стандартне завдання, метод випробування, що застосовується у різних галузях науки для одержання кількісної характеристики певних явищ.

Курс (від лат. cursus – рух, біг) – 3) Період навчання у вищих і середніх спеціальних навчальних закладах. 4) Виклад наукової дисципліни, галузі знання {Курс лекцій}.

Кейф (араб.) – безтурботний, приємний відпочинок.

Студія (італ. studio, від лат. studeo – ретельно вивчаю) {Студент – той хто *ретельно* вивчає. В багатьох мовах – перше значення цього слова}.

Диплом (франц. diplome, від грец. δίπλωμα – лист, складений удвоє) – 1) Офіційний документ про закінчення вищого чи середнього спеціального навчального закладу, про присвоєння вченого ступеня чи звання (франц. chef-d'œuvr||e, букв. головна робота) {дипломна}.

Бакалавр (лат. baccalaureus, від baccalarius – старший студент, клерк, землероб-орендар, з кельт.) – 1) Перший вчений ступінь у ряді країн. 2) Особа, що одержала диплом про закінчення середньої освіти (у Франції).

Магістр (від лат. magister – начальник, вчитель) – 1) У Стародавньому Римі титул деяких службовців. 2) У Візантії високий придворний титул. 3) У Західній Європі голова середньовічного духовно-лицарського ордену. 4) В дореволюційній Росії, Англії і США вчений ступінь. 5) В Західній Європі за середньовіччя викладач гуманітарних наук. Magistra – навчати.

Аспірант (від лат. aspirans (aspirantis) – той, що до чогось прагне) – особа, що готується до науково-педагогічної діяльності.

Кандидат (від лат. candidates – одягнений у біле; в Стародавньому

Римі претендент на державну посаду одягався у білу тогу) – 3) К. наук – перший науковий ступінь, який присуджується після захисту дисертації. 4) У дореволюційній Росії особа, що закінчила з відзнакою курс університету або прирівняного до нього іншого вищого навчального закладу (ліцей, академія) і подала письмову роботу на обрану нею тему.

Дисертація (лат. dissertatio – розвідка, дослідження, від disserto – досліджую) – наукова праця, яку підготовлено для прилюдного захисту на здобуття вченого ступеня.

Доктор (лат. doctor – учитель, наставник, від doceo – навчаю) – 1) Один з учених ступенів; особа якій присвоєно цей ступінь. 2) Лікар.

Асистент (від лат. assistens – присутній, помічник) – 1) Помічник спеціаліста (професора, лікаря, екзаменатора); перше наукове звання. 2) У вузах штатна посада на кафедрі. 3) Помічник прапороносця.

Доцент [від лат. docens (docentis) – навчаючий] вчене звання викладачів вищих навчальних закладів.

Техніка (грец. τεχνικός – вправний, досвідчений, від τεχνη – майстерність, мистецтво) – 1) Сукупність засобів праці. 2) Сукупність прийомів, застосовуваних у якійсь справі (напр., Т. живопису).

Техно ... (від грец. τεχνη – майстерність, мистецтво).

Прагматизм [від грец. πράγμα (πράγματος) – справа, дія].

Практика (від грец. πρακτικά – діяльність).

Теорія (від грец. θεωρια – розгляд, дослідження).

Σκεπτικός – досліджуючий, недовірливий, σκέψις – розгляд, сумнів.

Філологія (грец. φιλολογία – любов до вчених занять) гуманітарна наука, що вивчає мову і літературу (усну і писемну) якогось народу чи групи народів.

Філософія (грец. φιλοσοφία – любов до мудрості) – 2) Методологічні принципи, покладені в основу певної науки, галузі знання (напр., Ф. математики, Ф. права). 3) Переносно – абстрактні міркування, зайве розумування {Букв. філософ – не мудрець} [Паскаль, Б. Тарасов, С. 278]. Засновником (наукового, як системи поглядів) оптимізму був Ляйбніц (~1710 р.) (лат. pessimus – найгірший, optimus – найкращий).

На XXVII Міжнародному геологічному конгресі в 1984 році І.С. Шкловський висловив песимістичну (практично заборонену) думку: "Розумне життя не є вищою формою руху матерії. Розум – це лише один з численних "винаходів" еволюційного процесу, який до того ж приводить

вид, що ним нагороджений, до еволюційного тупика". Окрім відчаю, розпачу, розладу – дрібниць життя, в крайньому песимізмові є оптимістична сторона – життєва мудрість.

Мораль (від лат. mor множина mores – звичаї, *норов, вдача,* поведінка), лат. ethica (від грец. ἦθος – звичай, зазвичай). Мораль належить до основних типів нормативної регуляції дій людини.

Метод (від грец. μέθοδος – шлях дослідження, спосіб пізнання).

Методика (грец. μεθοδικά) – 1) Розділ педагогіки, що вивчає і складає правила і методи викладання якогось навчального предмета. 2) Узагальнення досвіду, способів, прийомів доцільного здійснення будь-якого завдання.

Проблема (від грец. πρόβλημα – задача, утруднення) – складне теоретичне або практичне питання, що потребує розв'язання, вивчення, дослідження.

Сентенція (лат. cententia – думка, судження, вирок) – вислів морально-повчального змісту, коротке напучення.

Сенс (лат. sensus – смисл, від sentio – вважаю) – 1) Значення. 2) Почуття, *замисел.*

Фундамент (від лат. fundamentum – основа) – 2) Переносно – основа, підпора, підвалина.

Фундаментальний (лат. fundamentalis) – 1) Міцний, великий. Переносно – обґрунтований, позитивний. 2) Основний, головний.

Клас (від лат. classis – розряд, група) – 1) Сукупність, розряд, *група* предметів або явищ, що мають спільні ознаки, якість.

Сорт (франц. sort – частина, якість). Нім. Gruppe *f* – загін, підрозділ.

1.2. Інформатика

Алгоритм, алгорифм (лат. algorithmus) –сукупність дій (правил) для розв'язування даної задачі. Від імені математика аль-Хорезмі.

Асемблер (англ. assembler, від assemble – збирати, монтувати) – загально прийнятна назва транслятора з автокоду {Збірка. Ансамбль (франц. ensemble, букв. разом), асамблея}.

Транслятор (лат. translator, від transfero – переношу) – 2) Програма перекладу описів алгоритмів з однієї формальної мови на іншу.

Компіляція (лат. compilatio, букв. – крадіжка, грабіж, від compilo –

грабую) – неоригінальна, несамостійна літературна чи наукова праця, побудована на використанні чужих творів {В код нічого "свого" не додавати}.

Інтерпретація (лат. interpretatio, від interpretor – роз'яснюю, перекладаю).

Тип (франц. type, від грец. τύπος – відбиток, форма, зразок) – 1) ... різновид чого-небудь.

Інформація (від лат. information – роз'яснення) – 1) Повідомлення про щось. 2) Відомості про навколишній світ, події, ситуації, системи.

Документ (від лат. documentum – повчальний приклад, взірець, доказ) – 1) Матеріальний об'єкт, в якому міститься та або інша інформація.

Проект (від лат. projectus – кинутий вперед) – 1) Сукупність документів. 2) Попередній гаданий текст будь-якого документа. 3) План, задум організації, влаштування, застосування будь-чого.

Специфікація (від лат. species – вид, різновид і ...фікація) – 1) Визначення і перелік специфічних особливостей будь-чого.

Специфіка, специфікум (від лат. specificus – особливий) – особливість, властива даному предмету, явищу, істотні ознаки, що відрізняють один об'єкт від іншого.

Ідентифікація (від лат. identicus – тотожний і ... фікація) – ототожнення, прирівнювання, уподібнення.

Система (від грец. σύστημα – утворення, складення) – 6) Сукупність частин, пов'язаних спільною функцією.

Комплекс (від лат. complexus – поєднання, зв'язок) – 1) Сукупність предметів чи явищ, що становлять єдине ціле.

Операція (від лат. operatio – дія, діяння).

Оператор (від лат. operator – виконавець).

Процес (від лат. processus – проходження, просування вперед) – 1) Послідовна зміна предметів і явищ, що відбувається закономірним порядком.

Процесор (англ. processor, від лат. procedo – просуваюсь) – 1) Частина цифрової обчислювальної машини, що реалізує процес складної обробки інформації. 2) Складна логічна програма, що є частиною системи автоматизації програмування {Процес, процесор}.

Програма (від грец. προγραμμα – публічна об'ява, розпорядження, указ) – 1) План діяльності. ... 7) П. в кібернетиці – докладний план дії обчислювальної машини, що складається з послідовних команд

(інструкцій), за якими машина виконує весь процес обчислень.

Ре... (лат. re...) – префікс, що означає зворотну або повторну дію.

Рекурентний [від лат. recurres (recurrentis) – той, що повертається].

Tourist (лат. torus – опуклість, tornāre – обточувати) {return}.

Математика (грец. μαθηματίκή, від μάθημα – *знання*, наука) – наука про кількісні співвідношення і просторові форми дійсного світу {Рекурсія в слові; звучить – "наука про науку"} та ματήματα – науки.

Дієслово μανθάνω – вчусь через розмірковування, μάθημα, μάθησις – наука, вчення.

Автомат (від грец. αυτοματος – самодійний) {сам знає (що робити)}.

Ітерація (лат. iteratio – повторення, від itero – повторюю).

Ієрархія (грец. ίεραρχία, від ίερός – священний, і άρχή – влада) – поділ на вищі і нижчі посади.

Апаратура (від лат. apparatus – устаткування).

Машина (франц. machine, від лат. machina – пристрій, знаряддя, споруда) сукупність механізмів, наприклад, для збирання і обробки інформації.

Інструмент (від лат. instrumentum – знаряддя) – знаряддя або пристрій.

Персона (лат. persona) – особа, поважна особа.

Солідний (франц. solide, від лат. solidus – твердий) – 1) Міцний, твердий, масивний, грунтовний, надійний. – 2) Поважний.

Індекс (лат. index, від indico – вказую) – 1) Список, покажчик, перелік будь-чого. 2) Числовий або буквений покажчик у математичних виразах, запроваджений для того, щоб відрізнити їх один від одного.

Блок (англ. block) 2) Вузол машини з кількох однакових частин, напр. Б. циліндрів у двигуні внутрішнього згорання. б) Скріплений комплект аркушів книжки, підготовлений для оправлення (книжковий Б.).

Модуль (від лат. modulus – міра) – 1) *тех.* Назва важливого коефіцієнта чи величини. 4) В *радіотехніці* – уніфікований функціональний вузол у вигляді взаємозамінного пакета деталей (і простіших М.), що виконує самостійну роботу у приладі чи апараті.

Пакет (нім. Paket, з франц. paquet) – 1) Згорток якихось речей; паперовий мішок для продуктів. **Папір** (πάπυρος, з егип.) {Папір, папка} Рос. **бумага** (від італійського bambagia – бавовна). Польською мовою "папка" – teczka (транскр. [течка]). *Макулатура* (нім. Makulatur, від лат. maculo – забруднюю) – відпрацьований папір, картон, паперові відходи.

Вторинна сировина. 1 т. М. замінює 4 $м^3$ деревини.

... тека (від грец. Θήκη – вмістище, сховище) – у складних словах відповідає поняттю "сховище", напр. *фільмотека*.

Код (франц. code, від лат. codes – звіт законів) – 1) Система символів для передавання, обробки і зберігання (запам'ятовування) інформації. 2) Ключ до способу шифрування чи дешифрування тексту.

Масив (франц. massif – важкий, від лат. massa – шматок, брила).

Гебр. massa' (1) пророцтво (2) тягар.

Йота (арамейське jod) – найменша буквочка (як ' апостроф) (грец. I, ι – іота. Y, υ – іпсілон).

Декларація (лат. declaratio, від declaro – заявляю, оповіщаю).

Елемент (від лат. elementum – первісна речовина) – 1) У давньоримській філософії – стихія (вогонь, повітря, вода і земля). 2) Складова будь-чогось цілого.

Компонент [від лат. componens (componentis) – той, що складає] – складова частина чогось.

Фаза (від грец. φάσίς – поява) – 1) Період, ступінь у розвитку чого-небудь.

Факт (від лат. factum – зроблене) – 1) Дійсна подія, явище) {Дія, дійсна. Подія – прояв дії, після дії}.

Фактор (лат. factor – той, що робить, від facio – роблю) – 1) Умова, рушійна сила, причина будь-якого процесу.

Форма (від лат. forma – зовнішність, устрій) – 1) Зовнішній вигляд, обриси предмета. 2) *філософ.* Всякий зовнішній вигляд будь-якого змісту.

Енергія (від грец. ένέργεμ – діяльність) – загальна міра всіх форм руху.

Активний (лат. activus) – діяльний, енергійний, той що розвивається.

Енергійний (від енергія) – 1) Активний, життєдіяльний, сповнений сил. – 2) Сильнодійний {останні слова – мінімальне порочне коло, серед помічених в посібнику}.

Ефект (лат. effectus – виконання, дія, від efficio – дію, виконую) – 1) Результат, наслідок яких-небудь причин, заходів, дій.

Інтелект (від лат. intellectus – розуміння, розсудок, пізнання) – здатність до мислення, особливо до його вищих теоретичних рівнів.

Моделювання (від франц. modeler – ліпити, формувати) – 1) Метод дослідження явищ і процесів, що грунтується на заміні конкретного об'єкта досліджень (оригінала) іншим, подібним до нього (*моделлю*).

Модель (франц. modele, від лат. modulus – міра) – 6) Схема для пояснення якогось явища або процесу {Модель спеціаліста}.

Симуляція (від лат. simulatio – удавання).

Динаміка (грец. δυναμικός – сильний, від δύναμις – сила) – 2) Хід розвитку, зміна якогось явища. Протилежне – статика.

Situ||ation f(франц., від лат. situs – становище) – 1) місцеположення; розташування; 2) становище, *стан*; 3) ситуація, обстановка; 4) пост, посада. Ситуація – 1) Збіг умов і обставин, що створюють певне становище.

Схема (від грец. σχῆμα – вид, форма) – 1) Спрощене зображення, викладення чогось у загальних, основних рисах.

Норма (від лат. norma – правило, взірець) – 1) Загально визначене, узаконене правило, міра, закон, взірець, звичайний стан.

Стандарт (англ. standart) – 1) Норма, зразок, мірило. 2) Прийнятий тип виробів, що відповідає вимогам за якістю тощо.

Магічний (грец. μαγικος) – 1) Той, що стосується магії. 2) Чудодійний, чарівний.

Магія (від грец. μαγεια – ворожба) – обряди пов'язані з чаклунством, віщуванням, вірою в "уміння" людини викликати надприродні явища. З. М. пов'язані найрізноманітніші повір'я, які є складовою частиною всіх релігійних вірувань.

Фаталізм (фран. fatalisme, від лат. fatalis – наперед визначений долею). **Фатум** (лат. fatum) – доля. **Фея** (франц. fée, від лат. fatum) – 1) У західноєвропейській міфології чарівниця, що може бути доброю чи лихою для людей. 2) Переносно – чарівна жінка.

Психіка (від грец. ψυχή – душа) – функція мозку, що полягає у відображенні об'єктивної реальності.

Психологія (від психо... і ...логія) – 2) Особливості характеру, психіки.

Цикл (від грец. κύκλος – круг, коло, круговерть).

Параметр (від грец. παραμετρέω – розмірюю, відмірюю) – 1) Величина, що зберігає стале значення лише за умов даної задачі. 2) Величина, що нею характеризують якусь властивість, стан, розмір або форму пристрою, робочого тіла тощо.

...метр (від грец. μετρέω – вимірюю).

Портал (італ. portale, від лат. porta – ворота; двері; прохід; отвір)

Порт (лат. portus, вхід, прохід; порт, гавань) Похідні слова, що сходять до слова "порт" – **ford** "брід", **fiord** "довга глибока морська затока в скелях".

...**port** (portāre "носити")

Результант [від лат. resultans (resultantis) – той, що відображує] – алгебраїчний вираз, який використовують знаходячи спільні розв'язки.

Структура (лат. structura – побудова, розміщення, від struo – будую, зводжу) – внутрішня будова чогось, певний взаємозв'язок складових частин цілого {Struo, стрій}.

Декларація (лат. declaratio, від declaro – заявляю, оповіщаю).

Опціон [від лат. optio (optionis) – вільний вибір].

Конструкція (від лат. constructio – побудова, складання) – 1) Будова, план, взаємне розташування частин (споруди, машини, вузла, витвору тощо).

Диск (грец. δίσκος) – 1) Плоский круг, кругла пластина {Пластина, площина} κλαστος – зламаний, потрощений.

Монітор (від лат. monitor – застережливий) – 4) Частина керівної програми операційної системи, яка здійснює керування однією з фаз обчислювального процесу.

Дисплей (англ. display, букв. – показ, демонстрування) – пристрій для введення інформації, який забезпечує візуальне представлення даних.

Англ. слово **window** – вторинне лексико-семантичне утворення: в давні часи йому передувало складне слово, що складається із англосаксонського vindr "вітер" і auga "око", тобто букв. "око вітру". Слово сходить до тих часів, коли ще не було скла, а вікна були отворами.

В слов'ян слово утворилось в результаті метафоричного перенесення слова "око" на отвір в стіні [7, С. 205] {отвір, отворять, отверстие}.

Франц. vitrage m 2) вікна; vent m 1) вітер; brise-vent *m* вітровий щит {брезент}. Нім. Wind m вітер. Теорія розбудови міст рим. архітектора і інженера 2-ї полов. 1 ст. до н.е. Вітрувія (Vitruvius): планувати вулиці так, щоб місто провітрювалось, не застоювалось хворобливе *повітря* [Докладніше у Д. Підоу].

Візуальний (лат. visualis) – спостережуваний неозброєним оком або за допомогою оптичного приладу.

Віртуальний (від лат. virtualis – сильний, здібний) – можливий; той що може або має проявитися.

Вірус (від лат. virus – отрута).

Трюк (франц. truc, букв. – спритність) – 1) Спритний, ефективний, майстерний прийом у сценічному, переважно цирковому мистецтві. 2) Переносно – спритна витівка, несподіваний вчинок, що вражає.

Манія (від грец. μανία – безумство, захоплення) – психічний розлад, при якому у хворого виникають нав'язливі ідеї.

Шифр (франц. chiffre, букв. – цифра, від араб. сіфр – нуль) – 1) Сукупність умовних знаків.

1.3. Тексти

Автор (від лат. autor – письменник).

Ідея (від грец. ἰδέα – початок, основа, первообраз) – 1) Найвища форма пізнання й мислення, яка не тільки відображує об'єкт, а й спрямована на його перетворення. 2) Думка, загальне поняття про предмет чи явище.

Ідеал (франц. idéal, від грец. ἰδέα – ідея, первообраз) – взірець досконалості, кінцева, найвища мета прагнень; *ідеальний* образ, що визначає спосіб мислення і діяльності людини чи суспільного класу.

Лінгвістика (франц. linguistique, від лат. lingua – мова) наука про мову; мовознавство. Польською "мова" – język (транскр. [є^{н}зик]).

В українській і білоруській мовах слово "язик" давно втратило своє друге значення.

Ляйбніц називав ідеографічне (знак – поняття) письмо Китаю "пасиграфією", тобто "письмо для всіх", письмо поза мовою {письмо світу}. Він вказав шляхи до побудови універсального письма і мови (думки), які стануть "азбукою людської думки" [А. Кондратов, С. 219]. Його роботи з пасиграфії містили корисні для інформатики (кібернетики) і математичної логіки ідеї.

Символ (від грец. σύμβολον – знак, прикмета, ознака) – 1) Умовне позначення будь-якого предмета, поняття або явища {Прикмета, мітка}.

Характер (грец. χαρακτήρ – ознака, риса, особливість).

Лексема (від грец. λεξις – мовний зворот) – слово-тип, структурний елемент мови; словникова одиниця з усіма її формами і значеннями.

Лексика (від грец. λεξικος – словесний, словниковий) – 1) Сукупність слів якоїсь мови чи діалекту (говірки). λεξ – закон.

Граматика (грец. γραμματική, від γράμμα – літера, написання)

частина лінгвістики, що вивчає будову мови (γράμμα – риска, *лінія*).

Синтаксис (грец. σύνταξίς – побудова, зв'язок, з'єднання) – розділ граматики, що вивчає правила поєднання слів у речення в певній мові.

Семантика (від грец. σημαντικος – означальний) – 1) Розділ мовознавства, що вивчає значення слів та виразів і зміну цих значень. Інша назва – *семасіологія*. 2) Значення (слова, виразу). В металогіці – розділ, що вивчає відношення виразів логічної мови до позначуваних ними об'єктів і змісту, який вони виражають.

Семасіологія (від грец. – σημασία – означення, смисл і ... логія).

Морфема (від грец. μοφή – форма) – *мовозн.* значеннєва частина слова (корінь, префікс, суфікс, закінчення).

Омоніми (від грец. όμώνυμος – однойменний) – слова або форми слів, що вимовляються однаково, але мають різне значення (наприклад, дверний ключ, журавлиний ключ, скрипковий ключ, ключ води). Слова омоніми даються в одній статті і відмічені цифрами з дужками. Якщо ж слова різного походження, то біля слова-заголовка вгорі ставиться порядкова цифра.

Синоніми (від грец. συνώνυμος – однойменний) – близькі за значенням, але різні за звуковим оформленням слова.

Концепція (від лат. conceptio – сприйняття) {Головне, голова} – 1) Система поглядів на певне явище; спосіб розуміння, тлумачення якихось явищ, основна ідея {Основа основ} будь-якої теорії.

Парадигма (від грец. παράδειγμα – приклад, взірець) – 1) Система форм словозмінювання одного й того самого слова чи групи слів. 2) Сукупність філософських, загальнотеоретичних і метатеоретичних основ науки. 3) Той чи інший приклад або показовий випадок концепції чи теоретичного підходу [Великий тлумачний словник сучасної української мови].

Діалект (від грец. διάλεκτος – розмова, говір, наріччя) – місцева говірка {Діаграма}.

Діа (грец. δια ...) – префікс, що означає наскрізний рух, проникнення, розподілену дію, розділення, взаємність, посилення, завершеність.

Ди, ді ... (грец. δί ..., від δίς – двічі).

Глоса (лат. glossa – слово, що потребує тлумачення, від грец. γλῶσσα – мова, говір(ка), чужа мова, незрозуміла мова) – 1) Невживане або незрозуміле слово чи вираз, що трапляються у тексті, й пояснення до них.

Глосарій (від лат. glossarium – словник глос) – тлумачний словник застарілих і маловживаних слів.

Префікс (від лат. praefixus – прикріплений спереду) приставка в слові.

Пре... (від лат. prae – попереду) – префікс, що означає передування.

Про... (лат. pro...) – префікс, що означає "для", "на боці", "в інтересах".

Сюжет (франц. sujet, від лат. subjectum – підкладене) ланцюг подій, образне відтворення задуму. **Сюжет** (фран. sujet – предмет) ряд зв'язаних подій, що послідовно розвиваються, і складають безпосередній зміст.

Суб'єкт (від лат. subjectum – підкладене) носій певного роду діяльності, джерело активності, спрямованої на об'єкт. У граматиці підмет (рос. подлежащее). У логіці – частина судження, що є поняттям про предмет, на який спрямована думка.

Об'єкт (від лат. objectus – предмет) – 3) *мовозн.* Другорядний член речення, додаток. 1) Матеріальний предмет пізнання і практичного впливу з боку людини (суб'єкта). Будь-який предмет думки, дослідження, художнього відображення.

В німецькій мові: Gegenstand m 1) предмет, річ; 2) тема.

Stehen – стояти, знаходитись. Gegen – 1) навпроти, проти, супротив; 2) до, в напрямку, порівняно {Об'єкт – супротивник, ціль}.

Фран.: Sujet – підлеглий, тема, предмет; jet – кидання, шпурляння; sous – під; su – з, на. Object – предмет, мета.

Предикат (від лат. praedicatum – сказане) – 1) В логіці – один з двох термінів судження, а саме той, в якому щось говориться про предмет мови (суб'єкт). В мат. логіці П. часто розуміють як пропозиційну функцію, тобто як функцію, що визначена на якійсь предметній області і значеннями якої є висловлювання або їхні істиннісні значення (*мовозн.* Присудок).

Німецькою: суб'єкт, сюжет Inhalt m; Fabel f; Stoff m об'єкт.

Subjekt – підмет (подлежащее), Prädicat – присудок (сказуемое).

Термін (лат. Terminus) – 1) В давньоримській міфології – божество меж, кордонів. 2) Слово, або словосполучення, що виражає певне поняття якоїсь галузі. У логіці – необхідна складова частина судження (суб'єкт і предикат) чи силогізму (дедуктивний умовивід, з двох суджень зумовлене ними третє – висновок).

Поезія (від грец. ποίησις – творення, творчість) – 1) Мистецтво слова, що художніми образами впливає на уяву (фантазія лат. phantasia, з грец.

φαντασία – уява) і почуття читача. 4) Переносно – чарівність, привабливість. **Поет** (ποίητης – творець, автор).

Стиль (лат. stilus, тут – паличка для письма) – 3) У мові – сукупність мовних засобів і прийомів, вибір яких зумовлений змістом, характером і метою висловлювання.

Етимологія (грец. ετυμολογια, від ετυμον – істина і λογος – слово, вчення) – 1) Розділ мовознавства, що вивчає походження слів. 2) Пояснення походження якогось слова зіставленням його з спорідненими словами тієї самої або іншої мови. {Походження слів "німці" і "слов'яни" мають подібні пояснення}.

За словами Вольтера, етимологія – це наука, в якій голосні нічого, а приголосні майже нічого не значать [13, С. 24].

1.3.1. Текст. Споріднені слова

Текст (від лат. textum – тканина, зв'язок, *побудова*) – 1) Відтворені на письмі або друком авторська праця, висловлювання, документи, пам'ятки тощо. 4) Друкарський шрифт, кегль якого дорівнює 20 пунктам (7,52 мм) {рос. ткань повествования – тканина оповіді, повісті}.

Кегль, кегель (нім. Kegel) – розмір шрифту в друкарських пунктах (0,376 мм). Американський пункт 0.352(7) мм 1/72 дюйма, 1pica = 12pt = 4.23(3) mm, пункт Дідо 0.376065 мм.

Лат. textura – тканина, зв'язок, будова {Тканини, текстиль, в'язання}. Споріднене – **техніка** (див. "техніка").

{Тикання човником у ручному верстаті, *утік*, *тканина*}.

(Лат. Textile – тканина, від texo – тку) {В'язання (наприклад, з інтерпретації контексту)}.

Матерія (від лат. materia – речовина) – 2) Тканина.

Мануфактура (лат. manufactura, від manus – рука і factura – обробіток, виготовлення) – 3) Застаріла назва тканин, продукції текстильної промисловості.

Німецькою: тканина Stoff m; Gewebe n, ткати weben vt. Ткацький верстат Webstuhl m. Ткнути stoβen* vt; weisen* (mit dem Finger).

(vt – перехідні дієслова, * – сильного і неправильного відмінювання)

Павук Spinne f. Павутиння Spinngewebe n. Мереживо Spitzen pl;

Англійською: тканина 1. (*матерія*) fabric, material; cloth;

Фабрика (лат. fabrica – майстерня).

Web n (of spider) павутина, weawe vt ткати, плести, снувати.

Wave vt 1) махати, розмахувати; 2) звиватись; wave n хвиля, змах.

Іспанською: tejedor m ткач, tejer vt 1) ткати; 2) в'язати; tejedo m 1) тканина ж; 2) в'язання с; tela f тканина, матерія, telar m ткацький верстат, telaraña f павутиння.

Румунською: тканина f tesătură, ткати a ţese.

Брезент (від голл. presenning) – густа лляна, напівлляна або бавовняна тканина, просочена водотривкими і протигнильними речовинами (див. франц.).

Велюр (франц. velours – оксамит, від лат. villosus – волохатий) – 1) Назва кращих сортів тканин (драп, оксамит, плюш) з мякою ворсованою лицьовою поверхнею (див. франц.).

1.3.2. Основа

Кажуть, що основа просвічує крізь предмет. На Україні основа виготовлялась з конопель, льону. Глосарій цього процесу: плоскінь, матірка, бережина, кострица, ґрадіво, куделя, кукла, кужіль, коловороток, верстат, колода, основа, ряд, покожушок, рядно, полотно (лат. linum), (житня) шліхта (нім. Schlichte), шліхтування {Г. Шліхтінг}.

Основа – поздовжні нитки полотна (утік, піткання – поперечні), готувалась снуванням нитки на вертикальний "барабан", який обертали по кілька разів то в один, то в інший бік {Таким чином, снування є основою всіх основ}.

Лінія (лат. līnea – льняна нитка, līnum – [грец. λινον] льон, лат. linio – мащу, натираю).

Лінь (голл. lijn) – тонкий канат (<25мм) з конопл. волокон вищої якості.

Тканина (і нитки) **"вісімка"** (8), далі 10, 12, 13, 14, за кількістю пучків в основі, по 30 ниток в пучку.

Чеською "основа" – zaklad. latka – 1) тканина, матеріал. kostra – остов, каркас.

База (франц. base, від грец. βάσις – основа) – 1) Основа, фундамент, опора будь-чого (βάσανος – пробний камінь).

Грецькою мовою – bat "той, що йде" < bainō "ходжу, вилажу". База, "основа", букв. "те, на що ступаю, опираюсь" [7, С. 34].

Польською "основа" – 2) podctawa; 3) *грам.* osnowa; 4) *текст.* osnowa. Założyciel – засновник {Підмурок}.

Німецькою "основа" – 1. Grundlage *f*, Basis f; 2. *текст.* Kette *f*; 3. *грам.* Stamm *m*. Грунт – Boden, Grund *m* {Підгрунтя}.

Угорською "основа" 1) alap; 2) szótö, alapszo. Слово – 1) szó; 2) beszed {Сотати, бесіда}. Словник – szótár.

Лема (λῆμμα – надходження, *засновок*) – *матем.* допоміжне твердження. **Дилема** (διλημμα – подвійний *засновок*) – 2) Переносно – необхідність вибору між двома небажаними можливостями. {*Висновок* – неологізм, нове слово}.

Контекст (від лат. contextus – тісний зв'язок, з'єднання) – уривок тексту із закінченою думкою, який дає змогу точно визначити смисл окремого слова або виразу, що входить до цього лінгвістичного оточення.

Пролог (грец. πρόλογος – передмова).

Епілог (від грец. επίλογος – висновок, післямова).

Епіграф (від грец. έπίγραφή – заголовок, надпис).

Унікальний (від лат. unicus – єдиний) – єдиний в своєму роді, винятковий, рідкісний {Одинак. Рід, рідкісний}.

Коректура, коректа (від лат. correctura – виправлення) {Рихтувати. Прямий, правий, напрям}.

Корекція (від лат. correctio – виправлення, поліпшення) {Кращий}.

Тривіальний (лат.trivialis, від trivium – перехрестя шляхів) – звичайний, буденний. Пересічний. {Сьогодні – ниньки. Вечора, днині, заутра. Ніколи не кажи "завтра"}.

Ініціал (від лат. initialis – початковий).

Ілюзія (лат. illusio, від illudo – висміюю, обманюю) –1) Оманливе, хибне сприймання людиною дійсності. 2) Нездійснена мрія, необгрунтована надія.

Парадокс (від грец. παράδοξος – несподіваний, дивний).

Інтервал (від лат. intervallum – проміжок, відстань).

Абзац (нім. Absatz, букв. – уступ) – 1) Відступ у початковому рядку тексту. 2) Пов'язана за змістом частина тексту від одного відступу до іншого.

Нім.: **Kólon** n –s, -s u Kola дві крапки {Крапки в стовпчик}.

Еліпсис (від грец. έλλειψις – опущення) – 1) *мовозн.* Пропуск у реченні слова чи словосполучення, зрозумілого з контексту, для вираження енергійності, схвильованості, складних переживань, розгубленості, а також

для уникнення повторів {Крапки в рядок}. Грец. λεῖψις – елейпсіс, недостача (minus), ὕπαρξις – існування, буття, в множ. – майно (plus).

Фрагмент (франц. fragment, від лат. fragmentum – уламок, шматок).

Монографія (від моно... і ...графія) – ґрунтовна наукова праця, в якій досліджується одне питання, одна тема.

Журнал (франц. journal, від jour – день).

Параграф (грец. παράγραφος, букв. написаний поруч) – 1) Частина тексту, що містить закінчене положення, правило. Позначається знаком § з порядковим номером. 2) Назва самого знака §. 3) Фігура у катанні на ковзанах.

Курсив [від лат. cursive (littera) – скоропис] – 1) Скорописний почерк у єгипетській, грецькій, латинській та інших системах письма. 2) Друкарський похилий шрифт для набірного виділення в тексті; деякі літери К. мають накреслення, подібне до рукописного.

Курсорний (від лат. cursor – бігун, скороход) – поспішний;

Фігура (лат. figura – образ, вид) {Образний}.

Літеральний (від лат. littera – літера) – буквальний.

Прото... (від грец. πρῶτος – перший) – у складних словах означає першість, первинність чого-небудь.

Апостроф (грец. απότροφοσ, букв. – відвернений, звернений на бік) – знак, яким в українській мові відокремлюють йотовані від губних та "р" ; в інших мовах ставиться замість пропущеної голосної.

Апо.. (грец. άπο...) префікс, що означає віддалення, завершення, заперечення, припинення, перетворення.

Строфа (від грец. στροφος – поворот, зміна) – у віршуванні ритмічно завершена частина твору, практично не більша за 14 рядків.

Катастрофа (від грец. καταστροφή – переворот, кінець, загибель) – 2) Центральна подія в драматичному творі, що передує розв'язці й зумовлює її.

Дефіс (нім. Devis – риска, від лат. divisio – розчленування) – знак переносу, з'єднання складних слів та скорочення слова, напр., б-ка.

Редактор (франц. redacteur, від лат. redactus – впорядкований) {Діє червоним (олівцем)}.

Кома[2] (від грец. κόμη – волосся) – 1) Один з недоліків зображень, утворюваних оптичними системами. К. дає пляму, форма якої нагадує синтаксичний знак – кому {волосинку, вію}. 2) *астр.* Газова туманність

оболонки ядра комети, що разом з ним створює голову комети.

Франц. virgule f – кома.

Комета (грец. κομήτης, букв. – волохатий, від κόμη – волосся).

Франц. vill||eux, –euse – мохнатий, волохатий (частина слова до рисок – стала частина словникового гнізда); velu, -e – волохатий, кошлатий; velou||rs – оксамит, бархат {Велюр}.

Ноти (від лат. nota – письмовий знак).

Літера (англ. letter від lettre – "буква", лат. littera від lītera "буква") Лат. lītera утворене від лат. lino "мурзаю", "мащу". Семантичний розвиток від "мащу", до "мазок", "помарка", "накреслений знак", "буква" [7].

Шрифт (нім. Schrift, від schreiben – писати) – 1) Накреслення, написання літер. 2) Друкарський матеріал.

1.3.3. Книга

В [7, С. 62] гот. bōka "**буква**" в множині bōkō "щось написане", "**книга**" семантичний розвиток прагерманського *bōkō – бук. * – знак вказує, що дана мовна форма припустима, не зареєстрована в джерелах. Значення науково відновлене.

Зарубки в давніх германців робились на корі бука. З появою писемності писали знаки – букви на букових дощечках. З готів bōkō в потрібний час потрапила до слов'ян (*боукъви*) але скоро була замінена давнішим словом "кънига" {Автор посібника вважає, що перша необхідність у виготовленні і поширенні "книг" виникла при дворі (подвір'ї) короля. Від їх назви – "королівські букви" в слов'ян для книг залишилось тільки "королівські", що перетворилось в "книги". Княжі}.

Нім. : **König** *m* (e) s, – e король;

Ф. Кьоніг створив першу практично придатну друкарську машину (плоско друкуючу) в 1810 р. {*Белл* А.Г. і телефон 1876 р.}

Кеннінг – текст давньоісландської поезії і одночасно формула його породження за формальними правилами з одного слова {"ідеї"} [19, С. 62].

1.3.4. Книга в інших мовах

Угорською мовою "книга" – könyv, румунською – carte, польською – książka (транскр. [ксьо^{Н}шка]), ksiega {ксьонз – книжник}, чешською – kniha, латиською – grāmata, французькою – livre *m*, іспанською – libro *m*, латиною – liber, libelli – книги.

Грецькою мовою chartēs "листок папіруса" [7, С. 75] {Карта}.

{З простого порівняльного аналізу кількох гебраїзмів випливає, що переклад слова "Вавілон" – "Велике місто". В подібний спосіб, як гебрейське або арамейське слово, "біблія" матиме переклад – "Велика збірка". Грецьке слово (βιβλίο – книга) виникло під впливом співзвуччя і, можливо, особливостей утворення множини в грецькій мові (в укр. транскрипції – "та бібліа") від назви прадавніх "дощечок" – "табличок"}.

Перейменування міста Губла (гебр. Гебал, нині Джубейль) в Бібл в певній мірі спричинено співзвуччям Бублос – Губла. Через цей торгівельний і культурний центр Фінікії греки возили папір(ус) з Єгипту (грецькою папір – "біблос", давніше – "бублос") [Ернст Добльгофер]. Походження алфавіту зокрема пов'язується і з писемністю міста Губла.

1.3.5. Книга. Споріднені слова

Французькою livrer *vt* 1) поставляти; видавати видання, відпускати, вручати; 2) виказувати, зраджувати.

bouquin *m розм.* стара книжка. cartable *m* 1) папка для малюнків.

carton *m* – 2) папка. libelle – пасквіль, libraire – книготорговець {Вольності}.

libre *adj* – 1) Вільний, незалежний, добровільний.

Іспанською librar – 1) позбавляти, звільняти; 2) видавати (указ, декрет); 3) видавати (вексель, чек).

Carta *f* – 1) лист; 2) *грамота*. cartilla *f* 1) азбука, буквар; 2) білет, документ.

Арамейське safar, гебрейське sofer – *писар*, *книжник*, учений, учитель.

Софізм (від грец. σοφιστής – судження, придумане розумно, хитро) – навмисне хибно зроблений умовивід, який має видимість істинного.

Софіст (гр. σόφισμα) – 1) В Стародавній Греції розумна, винахідлива людина; платний вчитель мудрості й красномовства.

Брошура (франц. brochure) друковане видання невеликого об'єму (в міжнародній практиці від 5 до 48 сторінок).

Французькою мовою Broche *f* 3) шило; 4) спиця, дротик. Brocher *vt* 1) *полігр.* Брошурувати; 2) ткати золотом, шовком. Broncher 1) спотикатися.

Кодекс (лат. codex, див. "код") 2) В Стародавньому Римі форма книги із скріплених разом навоскованих дощечок для письма чи папірусних листів. Сучасна книга зберігає форму кодекса у вигляді

книжного блоку. 3) Навоскована дощечка для письма, книга.

Блок 2) скріплені з одного боку *зошити.*

Книга – видання об'ємом більше 48 сторінок. Книга 4-3 тис. до н.е. – сувій, 2-4 ст. – кодекс, 2 ст. до н.е. – з пергаменту (від назви міста), в Європі з 13 ст. – з паперу. Корея – перший друк книги, ксилографія, з 704 по 751 рр. [Енциклопедичний словник].

Хрестоматія (грец. χρηστομάθεια, від χρηστός – корисний і μάθος – знання) навчальна книга, що складається з літературно-художніх або науково-популярних творів чи уривків з них, підібраних за програмою певного курсу.

Дискримінант [від лат. discriminans (discriminantis) – розрізняючий].

Етап (від франц. étape – перегін, перехід).

Аксіома (грец. ἀξίωμα – значуще, прийняте положення, від ἀξόω – вважаю гідним) – 1) Твердження теорії, що приймається без доведення, як вихідне, що є підставою для доведення інших тверджень цієї теорії.

Вандали (лат. Vandali) – давньогерманські племена, відомі жорстокими війнами з Римом; у 455 р. В. оволоділи Римом і знищили в ньому безліч творів мистецтва. Переносно – неуки, варвари, руйнівники культурних цінностей.

Варвари (лат. barbari, від грец. βάρβαροι – іноземці) – 1) У давніх греків і римлян назва чужоземців. 2) Переносно – жорстокі, грубі люди, руйнівники культурних цінностей {brutal, Brut}.

Редукція (від лат. reductio – повернення, відновлення) – 1) Процес або дія, що спричинює зменшення, спрощення чогось.

Денотат (від лат.denotatus – позначений, визначений) – об'єкт; те, що можна назвати певним іменем.

Афонія (від грец. ἀφωνία – німота, безмовність) {Німий – ні мовний. Афон – німота (безмолвие)}.

Аргумент (лат. argumentum, від arguo – показую, виявляю) – 2) *матем.* А. *функції* – незалежна змінна величина.

Функція (від лат. functio – виконання, звершення).

Аспект (від лат. aspectus – погляд, вид) – 1) Точка зору, з якої сприймається або оцінюється те чи інше явище, предмет, подія; перспектива в якій вони виступають.

Перспектива (франц. perspective, від лат. perspicio – бачу наскрізь, ясно бачу, уважно розглядаю) – 4) Переносно – види, плани на майбутнє.

Альтернатива (франц. alternative, від лат. alterno – чергую, змінюю) – необхідність вибору між двома можливостями, що виключають одна одну; кожна з цих можливостей {чи тільки дві? – наліво, направо, прямо}.

Принцип (від лат. principium – начало, *основа*) – 1) Первоначало; те, що лежить *в основі* певної теорії науки. 2) Внутрішнє переконання людини; основне правило поведінки.

Дедукція (лат. deductio, від deduco – відводжу, виводжу).

Фанатик (від лат. fanaticus – несамовитий, шалений).

Абстракція (від лат. abstractio – віддалення) – 3) Метод наукового пізнання, що полягає в мисленому виділенні суттєвих, найістотніших рис, відношень сторін предмета.

Абсурд (лат. absurdus – немилозвучний, безглуздий) – безглуздя, нісенітниця. Це перше слово довідника [7]. Вираз ab surdum означав "від глухого", тобто "те, що людина могла почути від глухої людини", "щось недочуте", і, в наслідок, "спотворене, безглузде".

Раціональний (лат. rationalis – розумний) – 2) Розумовий, доцільний.

Елімінація (від лат. elimino – винесення за поріг) – виключення, видалення; *матем.* виключення невідомих із системи рівнянь. **Елімінування** – видалення.

Ко..., ком..., кон... (лат. co..., com..., con) – префікс, що означає об'єднання, спільність, сумісність.

(лат. **catena** – ланцюг).

Вектор (від лат. vector – той, що несе) – 1) Величина, що характеризується розміром і напрямом. 2) Напрямлений прямолінійний відрізок.

Скаляр (від лат. scalaris – ступінчатий) – величина, яка повністю визначається своїм числовим значенням.

Прогрес (від лат. progressus – рух уперед, розвиток). Протилежне – *регрес*. **Регрес** (від лат. regressus – зворотний рух).

Каркас (франц. carcasse, від італ. carcassa) – кістяк, несуча основа будь-якого виробу, що складається з комбінації лінійних елементів, скріплених між собою.

Механіка (від грец. μηχανική – наука про машини).

Автономія (грец. αὐτονομία – незалежність).

...ном (від грец. νομή – частина, частка, νόμος – звичай, закон).

Атрибут (від лат. attributum – додане) – 1) Невід'ємна властивість об'єкта. 2) Означення (член речення). 3) Переносно – істотна ознака,

властивість чого-небудь.

Момент (від лат. momentum – рух, поштовх, мить) – 3) Обставина, окрема сторона якого-небудь явища.

Мнемоніка, мнемотехніка (від грец. μνημονικόν τέχνημα – мистецтво запам'ятовування) – система особливих засобів, що полегшують запам'ятовування і збільшують обсяг пам'яті.

Історія (від грец. ἱστορία – оповідь про минулі події, дослідження, знання) –1) Всякий процес розвитку в природі і суспільстві. (ἱστορίκος-дослідницький, науковий).

Семіотика (грец. σημειωτικος – пов'язаний із знаком, від σημεῖον – знак) - 1) Наука про різні системи знаків, які використовують для передачі інформації.

Фізика (від грец. φυσικά – єство, природа).

Замість висновків. Скільки є мов відомо: "І розділив на сімдесят і на дві мови, ... " [20, С. 10/11], [Ветхий завіт Книга буття 11, 1-9 Вавілонське змішання мов].

Зі словами складніше. Гебрейське devar – слово (діло). Грецькою ῥῆμα - слово, річ, випадок; ρήματα – слова, речі, події. "Спочатку було слово" давньогрецькою мовою – Εν ἀρχῇ ἦν ὁλόγος [Євангліє від Іоана 1,1] ("... і Слово було Бог"). В [2, С. 469], як і має бути для формальних мов, – це порожнє слово. "Спочатку було слово, але це не було фіксоване число бітів" (Р.С. Бартон) [2, С. 354] – думка, яка поділяється багатьма. В філософії це слово дало напрям, який називається фізичною семіотикою. Розглядається розвиток Всесвіту як процес обміну інформацією з допомогою електромагнітного поля між усіма елементами матерії. Навіть тоді, коли ще *не було* ні простору, ні часу, ні матерії в їх теперішньому розумінні.

{Для слова "врем'я" Яременко дає пояснення, а не переклад сучасною мовою [20]. Переклад назви [20] буде такий: "Повість історичних, героїчних літ"}.

Деякі "мертві" мови ефективніші, "живіші" усіх живих. Подібне можна говорити і про алгоритмічні мови, не помилившись при цьому {в "масі"} ні на йоту.

2. Виразні засоби алгоритмічних мов

2.1. Сім основних елементів програмування

Більшість програм створюється для розв'язання якої-небудь задачі. Розв'язок задачі досягається завдяки обробці інформації чи даних. Тому програміст мусить знати:

- як ввести інформацію в програму (введення);
- як зберігати інформацію в програмі (дані);
- як вказати правильні команди для обробки даних (операції);
- як передати дані із програми назад користувачеві (виведення).

Він може впорядковувати команди таким чином, щоб:

- деякі із них виконувались тільки, якщо виконується деяка умова або ряд умов (умовне виконання);
- інші виконувались повторно деяке число разів (цикли);
- треті виділялись в окремі частини, що можуть бути виконані в різних місцях програми (підпрограми).

Таким чином, перераховані всі сім основних елементів програмування: введення, дані, операції, виведення, умовне виконання, цикли і підпрограми. Цей список не є вичерпним, але він включає ті елементи, які звичайно притаманні всім програмам і мовам програмування [Borland Int.].

Багато мов програмування, серед яких Паскаль і Сі, мають ще додаткові засоби. Для швидкого ознайомлення з мовою досить вивчити, як реалізовані в мові ці сім елементів, і на їх основі будувати програми. В розділі представлені і ті виразні засоби мови, які були додані в її відповідну реалізацію. Практично весь розділ присвячено мові Паскаль, найважливішій "паперовій" мові, тобто мові запису алгоритмів.

2.2. Характерні особливості мови Паскаль

Створенню алгоритмічних мов і теорії програмування завдячуємо видатним людям [Бэкус Дж. В., Бауэр Ф.Л., Грин Дж., Кэтц С., Мак-Карти Дж., Наур П., Перлис Э.Дж., Рутисхаузер Х., Замельсон К., Вокуа Б., Уэгстейн Дж., Ван-Вэнгаарден А., Вуджер М. Алгоритмический язык Алгол 60. Пересмотренное сообщение, пер. с англ. М, : Мир., – 1965].

Мова Паскаль (попередній опис 1968 року) мала продовжити лінію

мов Алголу 60 і Алголу W, а саме концептуального спрощення з педагогічними цілями [Н. Вірт]. Інший напрям розвитку (започаткований того ж 1968 року) розглянемо в наступному розділі.

Важливо, що мова Паскаль проектувалась з врахуванням простоти написання відповідного транслятора. В результаті (звичайно, без сучасних розширень) розробка такого транслятора майже не перевищує трудомісткості гарної дипломної роботи випускника вузу (Д.Б. Подшивалов).

2.2.1. Розділ описів. Розділ операторів

Програма для комп'ютера, записана мовою Паскаль, як і багатьма іншими мовами, містить дві основні частини: опис дій, які необхідно виконати, і опис даних, з якими оперують ці дії. Дії описуються за допомогою операторів, а дані – за допомогою описів і визначень.

Програма складається із заголовка і "тіла" програми, що називається блоком. В 1962 році Замельсон вперше ввів блоки в мови програмування, як складені оператори, кілька операторів в операторних дужках, але в блоці є ще описи. В заголовку дається ім'я програми і перераховуються її параметри. Параметрами виступають змінні, які містять аргументи програми – вхідні дані і результати обчислень.

Блок складається з шести розділів, які розташовують в такій послідовності (для мови Турбо Паскаль (ТП, Turbo Pascal, TP) послідовність довільна і кількість розділів теж):

Блок: розділ опису міток (label);
розділ визначення констант (const);
розділ визначення типів (type);
розділ опису змінних (var);
розділ опису процедур і функцій (procedure, function);
розділ операторів (begin … end.).

Будь-який з перерахованих розділів блока може бути відсутнім, (чи всі) крім останнього.

Приклад програми: begin end. {Розмір виконуваного коду, файла .exe – 1632 байти}. В мові Турбо Сі – main(){} – 5257, Борланд Сі – 7684, Фортрані – end – 7671. Не зважаючи на текст, його вміст, компілятор в усіх випадках ставить у виконуваний код ділянки потрібні для обробки помилок на етапі виконання, з повідомленнями, такими як "стек переповнений", "незавантажений емулятор" тощо.

2.2.2. Складений оператор. Порожній оператор

Складений оператор (не плутати з поняттям "складний оператор") – це сукупність операторів, взята в операторні дужки begin – end. Всі оператори, що складають складений оператор, виконуються послідовно.

В мові Паскаль крапка з комою використовується як розділювач операторів і не входить до їх складу. Тому після оператора перед end не потрібно ставити ";". Наприклад, складений оператор:

Begin x:=a+b; c:=x-r end

Все ж в даному випадку ";" після другого оператора не призведе до помилки, а буде означати, що третім оператором цього складеного оператора є пустий. Але, як побачимо далі, ";" після оператора може призводити і до неприємностей.

Перший оператор є оператором присвоювання значення змінній. Цузе і Замельсон писали так: a+b ⇒ x [2, С. 489].

2.2.3. Складені типи даних

Дані – це загальне поняття для того, з чим оперує комп'ютер. Абстрагування від деталей внутрішнього представлення досягається головним чином за рахунок введення концепції *типу даних*. В мові є прості типи для простих, елементарних даних і типи для груп даних.

Подібно тому, як складений оператор складається з інших операторів, складені типи – із інших типів. Говорять про типи компонентів і про методи утворення або про структуру складного типу.

Змінні, що складаються з декількох компонентів, називаються змінними складеного (або структурованого) типу. Для змінних цього типу важливими є такі характеристики:

1) спосіб утворення, тобто структура складеного типу;

2) тип компонентів, що входять в структуру.

В подальшому будемо розглядати 4 різних складених типи змінних, що відрізняються за структурою і типом компонентів:

1) регулярний тип (масиви, array);

2) комбінований тип (записи, record);

3) файловий тип (файли, file);

4) множинний тип (множини, set).

Крім вказаних чотирьох структурованих типів в мові Паскаль існує ще один – покажчиковий тип (^, pointer, тип посилань), який утворюється

за певними особливими правилами. Мова ТП є ще більшою мірою мовою типів.

Введення складених (структурованих) типів даних значно підвищує ефективність засобів мови Паскаль, робить оператори мови більш потужними, створює ширші можливості обробки даних.

2.3. Типи даних мови Паскаль

2.3.1. Прості типи даних, порядкові типи

З кожною змінною програми зв'язаний один і тільки один тип. Він визначає діапазон значень, яких може набувати дана змінна, а також операції, що можуть з нею виконуватися.

В мові Паскаль є чотири стандартні типи даних: цілий (integer), дійсний (real) логічний (boolean) і символьний (char).

Типи змінних описуються в розділі опису змінних (var).

Типи всіх змінних повинні вказуватись. Константи теж належать до певного типу, їх тип легко визначається компілятором. Типізовані константи є фактично ініціалізованими змінними вказаного типу.

Приклад опису даних:

```
Var   lich, ind, nomsym: integer;
znach, maxznach: real;
endstr: boolean;
sym, sent: char;
const eil: single =2.71848;
```

Цілий тип (діапазон значень, внутрішнє представлення, операції, функції).

В ТР є п'ять стандартних типів:

Тип пам'яті	Діапазон	Розмір
shorting	-128..127	1 байт
integer	-32768..32767	2 байти
longint	-2147483648..2147483647	4 байти
byte	0..255	1 байт
word	0..65535	2 байти

Спроба обчислити вираз, значення якого виходить за межі діапазону, призводить до помилки підчас виконання програми.

Наступні операції, застосовані до цілих операндів (тобто змінних, констант, над якими здійснюються операції), дають цілі значення:

+ , - , * , div (частка) і mod (взяття залишку).

Значення m mod n визначене тільки для n>0.

Якщо m>=0, то m mod n = m-((m div n)*n),

а для m<0 m mod n = m-((m div n)*n)+n.

Таким чином, значення m mod n завжди додатне.

Значення цілочисельного виразу може бути присвоєне як змінній змінних цілого, так і дійсного типів. У іншому випадку ціле значення виразу перетворюється до дійсного типу і присвоюється змінній дійсного типу. Це так зване *неявне перетворення типу.*

Функції *pred* і *succ* мають завжди цілочисельні аргументи і дають цілочисельне значення:

pred(i) – дає попереднє ціле значення (i-1);

succ(i) – дає наступне ціле значення (i+1).

Функції *sin*, *cos*, *arctan*, *ln*, *exp* і *sqrt* можуть працювати з цілочисельними аргументами, але результат одержується завжди дійсний.

Функції *trunc* і *round* працюють тільки з дійсними аргументами, але приводять до цілочисельного результату:

trunc(r) – r – дійсне, результат – його ціле число. Дріб відкидається: trunc(3.7)=3.

Round(r) – r – дійсне, результат – округлене ціле.

Round(r) для r>0 означає trunc(r+0.5), а для r<0 – trunc(r-0.5).

2.3.2. Логічний тип (діапазон значень, операції, функції)

Логічні змінні можуть мати одне з двох значень: true або false (істина або хибність). Як і maxint, ці значення належать до наперед визначених констант, тобто констант, які не потребують опису. Логічні змінні найчастіше використовуються для керування послідовністю виконання операторів програми.

Існують три логічні операції, які після застосування до логічних операндів, приводять до результату типу *boolean*. Це операції: *and* – логічна кон'юнкція; *or* – логічна диз'юнкція; *not* – логічне заперечення.

В ТР додається ще одна операція *xor* (виключне або) – додавання за модулем 2.

Наприклад, якщо описати Var x, y, z: boolean; тоді можна записувати вирази, які будуть мати логічні значення

x and y or z not y and x x and (y or z)

Послідовність виконання логічних операцій: not, and, or. Для зміни послідовності використовуються дужки.

Таблиця істинності логічних операцій (x і y – логічні змінні)

x	y	not x	x and y	x or y	x xor y
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Логічний (бульовий) результат дають також операції відношення (порівняння): <, <=, =, <>, >=, >, in. Операнди цих операцій можуть бути дійсного, цілого а також логічного типу. Величини логічного типу можна порівнювати, оскільки відомо, що false<true.

Функція odd має цілочисельний аргумент. odd(*i*) дає true, якщо *i* непарне, і false, якщо *i* – парне. Якщо x – змінна логічного типу, то оператор процедури write(x) приведе до виведення 'TRUE' або 'FALSE', залежно від значення змінної.

Процедура read не може мати аргументів бульового типу.

2.3.3. Символьний тип (діапазон значень, операції, функції)

Значенням змінної символьного типу є символ – елемент деякої скінченної і упорядкованої множини символів. В кожній обчислювальній системі така множина є – вона необхідна для обміну інформацією із системою. На жаль немає єдиної стандартної множини символів на всіх комп'ютерних системах, різні системи відрізняються як самими символами, так і порядковими номерами. Але стандартом мови Паскаль вимагається, щоб цим множинам були притаманні такі властивості:

- кожний символ повинен мати порядковий номер;
- десяткові цифри впорядковані за зростанням і йдуть одна за одною;

- букви впорядковані за алфавітом, але їх номери не обов'язково йдуть підряд.

Найширше застосовується (на всіх персональних комп'ютерах) множина символів – ASCII (аскі) – American Standard Code for Information Interchange.

В програмах символьні константи, визначені своїми значеннями, беруться в апострофи: 'a', '*', '3', ' ' ' ' (сам апостроф треба записувати двічі).

Для відображення заданої множини символів на порядкові номери і навпаки існують дві функції ord і chr – функції відображення: ord(C) – дає порядковий номер символу C в заданій упорядкованій множині символів; chr(i) – дає символ з номером *i* в цій множині.

Допускається використовувати запис символу, вказуючи його внутрішній код, якому передує символ # (#35): #7 – bel, #10 – fl, #13 – cr, #39 – ' (апостроф), #90 – 'Z', #97 – 'a'.

2.4. Арифметичні і логічні вирази

2.4.1. Оголошення нетипізованих констант. Константні вирази. Подвійна дія оператора присвоювання

Основними об'єктами з яких конструюється виконувана частина програми є константи, змінні і звертання до функцій. Кожний з цих об'єктів характеризується своїм значенням і належить до деякого типу даних.

За допомогою знаків операцій, дужок з них можна скласти вирази, які фактично є правилами отримання нових значень. В загальному випадку вираз складається із декількох об'єктів (операндів) і знаків операцій, а тип його значення визначається як типом операндів, так і видом операцій, що застосовані до них.

Константи визначаються в розділі, що починається зі службового, зарезервованного слова const. Тип нетипізованної константи визначається способом запису її значення. При оголошенні констант в якості значень допускаються вирази:

```
Type prl=(e, f, g, h); const c1='A'; c2=$1ff; i=1; j=2; sij=[i+j, j-i];
prc=[g, e, succ(e)];   h10=hi(c2);   sb=odd(50);   s='Рядок-константа';
sw=swap(c2); k=length(s); NumCh=ord('z') - ord('A')+1;
```

```
MaxReal=MaxInt div SizeOf(real); p=trunc(pi);
begin
writeln(h10:10, sb:10, sw:10, k:5, NumCh:5, MaxReal:10, p:5)
{16 FALSE 65281 15 58 5461 3}
end.
```

З прикладів видно, якого виду функції можуть бути використані в простих константних виразах.

Подвійна дія оператора присвоювання (':=') полягає:

- в обчисленні значення виразу;
- набутті змінною цього значення (воно має бути в межах допустимого діапазону).

Константний вираз – це вираз, який може бути обчислений компілятором фактично без виконання програми. Тому знак '=' в декларації константи не є оператором присвоювання. Значення буде в імені ще до виконання програми.

2.4.2. Операції відношення (порівняння), логічні операції, побітові (бінарні) операції. Логічні вирази

В операціях відношення (порівняння) мають брати участь однотипні операнди. Виняток операнди типів Integer і Real. Результат застосування операції відношення до будь-яких операндів має тип boolean.

При порівнянні даних типу boolean враховується внутрішня домовленність ТР: ord(false)=0, ord(true) =1. Логічні операції not – не, and – і, or – або, xor – виключне або (або-або). Логічні операції застосовні до операндів цілого і логічного типів.

Мова Сі, як і Алгол 68, в певному розумінні є мовою, основаною на виразах. В Сі оператори присвоювання, умовні оператори, функції мають значення [15, с. 66] (Точніше, в мові Сі – операції присвоювання. SizeOf і "," – кома – операції також).

В мові Сі 1<2<3 і 3<2<1 – коректні вирази, що мають ненульові значення. Логічні вирази схожі з арифметичними і можуть включати операцію ", – кома" {Кома – операція Фортрану, Алголу 68}.

До цілих данних як побітові використовуються – і shl j – зсув *і* вліво на *j* двійкових розрядів, і shr j – зсув *і* вправо на *j* двійкових розрядів. Пріоритет – як мультиплікативних операцій. Всі логічні операції мають вищий пріоритет, ніж операції відношення, тому в складних логічних виразах необхідно розставляти дужки (порівняння беруться в дужки).

Приклади:

writeln(7 or 5:4, 7 and 5:4, 7 shr 2:4,

6 or 5:4, 6 and 5:4, 1 shl 3:4); { 7 5 1 7 4 8 }

В мові Сі логічні операції позначаються двома символами, а побітові – одним. Причина цього пояснюється метою створення мови Сі – використання для розробки операційної системи UNIX (в мові Фортран операція степеня – "**". В мові Симула позначення операцій двома символами було введено для об'єктів).

2.4.3. Таблиця пріоритетів, старшинства операцій

Розглядається три групи операцій: арифметичні, логічні та відношення (порівняння). Встановлено пріоритетність цих операцій в межах кожної групи. Встановимо загальну пріоритетність:

Операція	Класифікація
в дужках	Найвищий пріоритет
not	Логічне заперечення (перший пріоритет)
*, /, div, mod, and	Мультиплікативні операції (2-й пріоритет)
+, -, or	Адитивні операції (третій пріоритет)
=, <>, <, <=, >, >=, in	Відношення (найнижчий пріоритет)

2.5. Оператори управління

2.5.1. Структура розгалуження. Реалізація в мові Паскаль

Часто в програмі необхідно вказати декілька варіантів можливих дій так, щоб вибір одного з них здійснювався уже під час виконання програми. В мові Паскаль структура розгалуження реалізується за допомогою умовного оператора, який дає можливість вибирати одну з двох дій залежно від результату обчислення логічного виразу. Цей логічний вираз називається умовою або предикатом.

Загальний вигляд умовного оператора:

if <логічний вираз> then <оператор> else <оператор>

Якщо значення умови, яка задається булевим виразом, є true, то виконується оператор, що стоїть за then, якщо значення умови дорівнює false, то виконується оператор, що стоїть після else. Залежно від результату булевого виразу, виконується один з альтернативних операторів.

Прикладом, коли логічний вираз в операторі if має складнішу структуру, може бути задача визначення, чи можна побудувати трикутник з відрізків a, b, c:

```
if (a+b>c) and (a+c>b) and (b+c>a) then
writeln (' трикутник побудувати можна')
else writeln (' трикутник побудувати не можна')
```

Умовний оператор може і не мати конструкції else – така форма називається скороченою:

if <булевий вираз> then <оператор>;

Вважається, що односторонній вибір майже завжди неправильний. Багатосторонній є правильним лише тоді, коли враховані абсолютно всі ознаки випадків. Тому в ТР є else (Навіть так: else;).

Майерс [9, С. 141] ставить "зустрічне", протилежне питання: "А чи потрібно уникати else? Наприклад, щоб уникнути вкладених розгалужень".

Оператори, що стоять після then або else, самі можуть бути умовними операторами, тоді маємо вкладену конструкцію умовного оператора. При "вкладених" умовних операторах будь-яка частина else, що зустрілась в послідовності операторів відповідає найближчій до неї "зверху" частині then умовного оператора.

В мові Паскаль символ ";" – розділювач операторів. Формально він не є кінцем оператора, як, наприклад, в Сі, де всі оператори окрім складеного закінчуються ";" (В мові Сі цей символ "перетворює" вираз в оператор). Тому в Паскалі наявність ";" перед else оператора if є помилкою.

2.5.2. Оператор безумовного переходу. Мітки. Основа структурного програмування

Умовний оператор if – then – else вибирає один з двох можливих напрямків виконання програми залежно від виконання умови. Інакше він називається оператором умовного переходу. В програмі може виникнути потреба в переході до виконання деякого відрізка програми незалежно від жодної умови. Такий перехід реалізується в мові Паскаль за допомогою оператора безумовного переходу:

goto <мітка>

Тут мітка – це число без знака (0..9999), чи правильне ім'я, описане в розділі label, ідентифікатор, ім'я оператора. Оператор goto передає управління оператору, який помічено відповідною міткою. Перед

оператором може бути їх декілька. Після кожної має бути двокрапка.

Вживання оператора goto в мові Паскаль небажане, оскільки порушує структурну цілісність і наочність програми. Використовують його в крайніх випадках – наприклад, для виходу із деякого складеного оператора, якщо виникає особлива ситуація в програмі (Д. Кнут заперечує обмеження використання goto для будь-якої мови).

Можна довільно передавати управління в середині складеного оператора, виходити з нього, але входити можна тільки через початок. Не можна передавати управління з однієї гілки умовного оператора в другу.

За допомогою операторів if і goto можна промоделювати всі види циклів мови Паскаль.

2.5.3. Загальна характеристика структури повторення. Моделювання всіх типів циклів за допомогою умовного оператора і оператора безумовного переходу, його недоліки

Розв'язуючи деякі задачі, доводиться багатократно здійснювання обчислення за одними і тими ж залежностями, але для різних значень величин, що входять в ці залежності. Такий процес називається циклічним. Ділянки програми, які реалізують такий циклічний процес, називаються циклами, а змінні, що змінюються в циклі, – циклічними змінними. Алгоритм циклічної структури повинен містити:

1) підготовку циклу – задання початкових значень змінних циклу;
2) тіло циклу – дії, що виконуються в ньому;
3) модифікацію значень змінних циклу перед кожним новим його повторенням;
4) керування циклом – перевірку умови продовження циклу і перехід на початок циклу чи вихід із циклу залежно від виконання умови.

Прикладом простого циклічного процесу є табуляція функції – обчислення її значень y = f(x) для різних значень x в інтервалі $[X_1..X_n]$ з кроком h. Організувати такий цикл можна за допомогою операторів присвоєння, умовного і безумовного переходу і операторів повторень (циклу).

```
x:=x0;                    Тут явно відображені
15:{ обчислення f(x) }    всі етапи організації
x:=x+hx;                  циклу : підготовка, тіло,
```

```
if x<=xn                    модифікація і перевірка
then goto 15;               і продовження.
```

В мові Паскаль, є декілька (три) можливостей безпосередньо організувати циклічний процес. Це оператори циклу з *передумовою*, з *постумовою* і оператор циклу з *параметром*. Перші два використовуються коли невідома кількість повторень, а третій – коли кількість повторень відома. Оператора останнього типу немає в мові Сі.

```
{ Обчислити k членів послідовности An=An-1/n, A1=B}
label nc;
const  k=9; b=10000;
var  n: 1..k;  a: array[1..k] of real;
begin
a[1]:=b; for n:=2 to k do a[n]:=a[n-1]/n;
writeln(' Послідовність');

{МайжеМодель циклу repeat (з допущенням)}
n:=1;
nc: writeln(' a[',n:2,'] = ',a[n]:8:3);
n:=n+1;
if n<=k then goto nc;
end.

{ Обчислити суму з точністю до епсілон = 10E-9: }
{ S = 2 + 3/2! + 4/3! + .. + (n+1)/n!,              }
{ доки ще виконується умова (n+1)/n!>eps        }
label  nc, kc;
const  eps=1e-9;
var   s, n1, fc: real;
begin
S:=0; n1:=2; fc:=1;
writeln(' Сума : ');
while n1/fc>eps do
begin
s:=s+n1/fc;
fc:=fc*n1; n1:=n1+1;
end;
```

```
writeln('S = ',s:10:8); {4.43656366}
writeln;

S:=0; n1:=2; fc:=1;
{МайжеМодель циклу while (з відмовою)}
nc:
if n1/fc<=eps then goto kc;
s:=s+n1/fc;
fc:=fc*n1; n1:=n1+1;
goto nc;
kc:
writeln('S = ', s:10:8);
end.
```

Тут слід звернути увагу на послідовність визначення виразних засобів мови програмування.

2.5.4. Цикл з передумовою (з відмовою)

Часто доводиться мати справу із циклічними обчисленнями, коли число повторень циклу наперед невідоме, зате задана деяка умова закінчення циклу. Для таких випадків у мові Паскаль існує два типи операторів – цикл з передумовою і цикл з постумовою.

Загальний вигляд оператора циклу з передумовою

while <умова> do <оператор>,

де <умова> – це будь-який логічний вираз, <оператор> – довільний оператор мови Паскаль. Оператор виконується в циклі доти, поки умова має значення true. Як тільки значення умови стане false, здійснюється вихід з циклу і виконується наступний оператор. Оператор в циклі з передумовою може не виконуватись зовсім, якщо умова рівна false на самому початку.

Слід наголосити, що структура циклу з передумовою передбачає перевірку умови завершення циклу перед виконанням оператора в циклі, що обумовлює можливість не виконання цього оператора жодного разу. Потрібно врахувати, що при обчисленні умови значення змінних чи індексів може вийти за межі значень для яких виконується тіло циклу.

2.5.5. Цикли з параметром

Якщо число повторень наперед відоме, то доцільно використовувати структуру циклу з параметрам. Доцільно – але не обов'язково, оскільки ця структура може бути замінена іншою – наприклад, циклом з передумовою.

Це число повторень обчислюється з допомогою спеціальної змінної, що називається параметром циклу. Для неї відоме початкове і кінцеве значення, а також крок її зміни. Керування циклом здійснюється шляхом порівняння поточного значення параметра циклу з кінцевим.

Загальний вигляд оператора циклу з параметром:

for <параметрциклу>:=<вираз1> [to | downto] <вираз2>
do <оператор>;

де <параметрциклу> – змінна будь-якого ординального типу; <вираз1> і <вираз2> – вирази, тип яких повинен збігатися з типом змінної параметра циклу (обчислюються раз, перед виконанням циклу); <оператор> – будь-який оператор Паскаля.

Оператор виконується для кожного значення параметра циклу від значення <вираз1> до <вираз2> включно. При цьому під час використання службового слова to значення параметра циклу зростає за правилом пц:=succ(пц), під час використання службового слова downto зменшується за правилом пц:=pred(пц). Отже, коли параметр циклу – змінна цілого типу, то крок рівний 1 для to і рівний -1 для downto.

Використовуючи оператор циклу з параметром слід враховувати такі правила:

1. Заборонено змінювати всередині циклу, тобто в тілі циклу, значення <параметрциклу>, <вираз1>, <вираз2>;
2. Заборонено входити в цикл, обминувши оператор for, оскільки значення <параметрциклу>, <вираз1> і <вираз2> будуть невизначені.
3. Цикл не виконується взагалі, якщо початкове значення більше (а у випадку downto – менше) від кінцевого;
4. По закінченні виконання циклу значення змінної <параметрциклу> не визначене і не може бути використане для подальших обчислень;
5. Після службового слова do може стояти тільки один оператор; якщо в циклі треба виконати групу операторів, то їх беруть в операторні дужки begin-end;
6. Із складеного оператора можна вийти за допомогою оператора безумовного переходу goto. При цьому останнє значення <параметрциклу> зберігається.

2.5.6. Цикл з постумовою (з післяумовою, з допущенням)

Другим способом організації циклічного процесу, коли число повторень невідоме, є використання оператора циклу з постумовою. Загальний вигляд цього оператора

Repeat <оператор 1>; <оператор 2>; ... <оператор n> until <умова>;

де <умова> – це логічний вираз; <оператор 1> ... <оператор n> – будь-які оператори мови Паскаль.

Дія оператора з постумовою подібна до дії оператора з передумовою (while), але перевірка умови закінчення циклу здійснюється після (кожного) виконання тіла циклу. В зв'язку з цим цикл з постумовою виконується як мінімум хоча б один раз. В циклі виконуються всі оператори, записані між словами repeat і until, що відіграють роль операторних дужок. Оператори виконуються, доти, доки умова рівна false.

Хоча вибір оператора циклу здійснюється в відповідності з логікою задачі, рекомендується використовувати цикл з передумовою.

2.5.7. Характеристика ітераційних обчислень і циклів. Інваріанти циклу

В процесі різних обчислень доводиться будувати послідовність значень $g_1, g_2, \ldots g_n$, що обчислюються в результаті повторення тіла циклу. Ці значення утворюють збіжну послідовність, що прямує до деякої границі a $\lim g_n = a$, $n \to n_{гр}$.

Це звичайно цикли з невідомим наперед числом повторень. Кожне наступне значення послідовності визначається через попереднє g_{n-1} і з врахуванням збіжності є більш точним наближенням до результату, тобто границі a. Цикли, що реалізують таку послідовність наближень, називаються ітераційними циклами.

Цикл в алгоритмі завжди має три частини, а саме:

- ініціалізацію (чи підготовку), в якій прокладаються шляхи для наступних прогонів;
- перевірку умови, чи досягнута мета циклу;
- тіло циклу, яке містить дії, що поступово наближають цикл до його мети.

При кожному прогоні по циклу всі задіяні в ньому змінні приймають нові значення. Хоча при кожному прогоні вони змінюються, певні відношення між ними залишаються. Ці відношення називаються

інваріантами циклу.

Оператор **break** перериває виконання оператора циклу. Керування передається наступному оператору. Оператор **continue** передає керування на перевірку умови виконання циклу.

Етапи складання циклу:

1. З'ясовується мета циклу.

2. Обмірковується ряд проміжних станів:

- потрібна мета циклу відноситься до їх ряду,

- один з таких станів легко створити і він придатний в якості "початкового" для циклу,

- перехід від одного проміжного стану до іншого можна здійснити виконанням операторів програми.

2.5.8. Оператор вибору (варіанта). Ключ вибору, список вибору можливих продовжень програми. Вибір за замовчуванням

Оператор дозволяє вибрати одне з кількох можливих продовжень програми. Параметром, за значенням якого здійснюється вибір, служить так званий ключ вибору – вираз будь-якого порядкового типу.

Оператор варіанта є узагальненням умовного оператора: він дає можливість виконати один з декількох операторів (в умовному – з двох) залежно від значення деякого виразу. Цей вираз (ключ вибору) називається ще селектором. Загальний вигляд оператора варіанта (оператора управління, який використовується для зміни послідовності виконання операторів):

```
case <селектор> of
<список міток 1> : <оператор 1>;   {тут перехід до виконання }
                                   {наступного за case оператора }
<список міток 2> : <оператор 2>;   { ...          ...     }
. . . . . .
<список міток N> : <оператор N>;   { ...          ...     }
else          <список операторів>  { в решті випадків }
end;
```

Список вибору складається з однієї або більше конструкцій (N)

<константи вибору (список міток)> : <довільний оператор>;

Список міток – це список розділених комою можливих значень селектора або одне його значення. В ТР мітки можуть задаватись і як

діапазон. Ці константи повинні мати той же тип, що і селектор. Вони називаються мітками варіанта. Мітка варіанта – це не мітка оператора: це константа порядкового типу або діапазон порядкового типу або їх список через кому. Оператор варіанта вибирає для виконання той оператор, одна з констант якого рівна значенню виразу селектора. Після виконання вибраного оператора керування передається на кінець оператора case.

Приклад оператора варіанта:

```
case j of 1: x:=0; 2: x:=sin(X); 3: x:=exp(x); 4: x:=ln(x) end;
```

В мові TP допускається конструкція з else:

```
case color of red: x : = y = 2; blue: x : = y; white: x : = y/2
else writeln('помилка'); writeln('y=', y)
end;
```

Для альтернативи else допускається довільна кількість операторів на відміну від помічених операторів, де може бути тільки один. Перед else оператора case може бути символ ";". У відповідності зі стилем мови Паскаль, звичайно, оператор case записується "східцями".

2. 6. Дійсний тип (діапазон значень, операції, функції)

В TP є такі стандартні типи дійсних чисел:

Тип	Діапазон значень	Кількість розрядів мантиси	Розмір пам'яті (в байтах)
real	2.9E-39..1.7E38	11–12	6
single	1.5E-45..3.4E38	7 – 8	4
double	5.0E-324..1.7E308	15–16	8
extended	3.4E-4932..1.1E4932	19–20	10
comp	-2E+63..+2E+63-1		8

Змінні дійсного типу real можуть набувати значень з підмножини дійсних чисел. Константи дійсного типу подаються в програмі з десятковою крапкою і з порядком. Можливі значення дійсних змінних характеризуються двома характеристиками – діапазоном і точністю, які й обмежують підмножину дійсних чисел, що можуть оброблятися.

Наприклад, діапазон чисел, що обробляються: $(\pm)10^{-75} .. 10^{75}$ і

точність 10 десяткових цифр. Тобто діапазон обмежується максимальним порядком, а точність – максимальною кількістю десяткових цифр процесора.

Кожна операція з дійсними значеннями вносить похибку. Під час виконання великих програм похибка накопичується і результат може бути спотворений. Тому треба вибирати ефективні алгоритми, контролювати проміжні результати під час розв'язування задач на комп'ютері (наприклад, вибір головного елемента під час розв'язування систем лінійних алгебричних рівнянь).

Такі операції дають дійсний результат, якщо хоча б один з операндів дійсного типу (другий може бути цілого типу) : + , - , * , /. Під час ділення обидва операнди можуть бути цілі, а результат дійсний. Якщо типи операндів змішані, перед виконанням операції всі величини приводяться до дійсного типу (автоматично).

Значення виразу цілого типу може бути присвоєне змінній дійсного типу (відбувається неявне перетворення ціле → дійсне). Проте забороняється присвоювати результат дійсного типу змінній цілого типу. Функції trunk і round використовуються для переведення дійсного значення в ціле.

Функції abs(r) і sqr(r) дають дійсний результат, якщо аргумент дійсний. Функції sin, cos, arctan, exp, sqrt, int за будь-якого дійсного чи цілого аргументу дають дійсний результат.

У дійсних величин не існує попереднього чи наступного значення.

2.7. Типи, що оголошуються. Переоголошення типу

Крім стандартних типів (визначених в системі програмування), в мові Паскаль є можливості самостійно встановлювати тип даних. Це дає змогу, з одного боку, складати наочну програму, а з іншого – передавати компіляторові більше інформації про програму, за допомогою якої компілятор може ретельніше перевірити синтаксичні помилки і сформувати ефективнішу робочу програму.

Нові типи визначаються в розділі визначення типів, який починається словом *type*.

Наприклад: type cilyj = integer;

Тут введено новий тип cilyj, який ідентичний типу integer.

Тип може бути переоголошений. Наприклад: type real = single;

2.7.1. Діапазонний, інтервальний тип. Правила декларування, вирази в границях

Тип-діапазон є підмножина свого базового типу (host), яким може бути довільний порядковий тип, окрім типу-діапазону. Границі типу-діапазону розділяються *одним* спеціальним символом – двома послідовними крапками:

<мінімальне значення>..<максимальне значення>

Правила оголошення (декларування):

1. Два символи .. розглядаються як один і пропуски (інтервали) між ними недопустимі.

2. Необхідно, щоб ліва границя діапазону не перевищувала праву.

3. Ім'я типу – правильний ідентифікатор. Функція ord поверне значення порядкового номера в базовому типі. Функція pred на нижній границі дасть помилку, хоча ця границя може бути всередині базового типу. Аналогічно, помилка при виконанні функції succ на верхній границі.

4. Значення границь задаються константними виразами.

Приклади:

```
Const x=50; y=60;
Var a: 1..10; b: 0..30;
Type days=(pn, vv, sr, ch, pt, sb, nd);
indx = 1..20; –  { діапазон типу byte; }
budni = pn..pt; – { діапазон типу days; }
bukv = 'a'..'z';-   { діапазон типу char; }
scal =2*(x-y)..(x+y)*2; { Конст. вирази }
```

Компілятор TP збудований просто, тому, наприклад, нижню межу діапазону не слід починати з дужки.

Нижче в розділі змінних використані ці явно оголошені типи:

```
var
i, j: indx; rd, dn: budni; sym: bukv;
```

Всі операції, що застосовуються до змінних базового типу, можуть застосовуватися до відповідного діапазону:

```
j+i*2        rd<dn        pred(cum)<>'d'
```

Всі функції базового типу можуть застосовуватись до діапазону. Значення функції необов'язково буде належати до діапазону.

Наприклад: sqr(i) не можуть входити в діапазон indx (якщо, звичайно, це значення не присвоюється змінній типу indx, наприклад j).

Застосування типу діапазонів покращує наочність програми

(проблема наочна), економне використання пам'яті, а також передає інформацію компіляторові для перевірки правильності програми – перевірки діапазону значень змінних.

2.7.2. Переліковий тип

Переліковий тип визначається як впорядкований набір ідентифікаторів, заданий шляхом їх перелічення. Тобто для визначення перелікового типу задається іменований список значень, що можуть набувати змінні цього типу.

Імена, які перелічуються в дужках, називаються константними іменами. Вони іменують числа від 0 до n-1, де n – кількість імен в дужках. Приклад опису типів:

```
type
color = (red,black,white,blue,orange,green);
var
x, y, z:color;
```

Тоді можна виконувати операції

x:=black; z:=blue;

Забороняються змішані (різнотипні) присвоювання.

Ім'я значення не може належати більш як до одного типу.

Категорично забороняється різним іменам типів надавати однакові описи.

Єдина операція, яка може виконуватись над змінними перелікового типу, це операція відношення. Результат отримуємо булевий. Зрозуміло, що обидва компоненти відношення мають однаковий тип. Впорядкованість в перелікових типах визначається послідовністю, в якій значення перераховані під час визначення типу.

Запишемо умовно – type T = (W410, W420, ..., W4n0);

Тоді істинність

W4i0<>W4j0, для i<>j – ознака відмінності,

W4i0<W4j0, для i<j – ознака впорядкованості.

Для визначених вище типів вираз white<red буде мати значення false.

Впорядкованість змінних перелікового типу дозволяє застосовувати функції pred і succ: pred(w420) = w410; succ(blue) = orange; Перший елемент списку не має попереднього, а останній – наступного значення, тобто pred(w410) і succ(green) тут невизначені.

Функція ord теж має аргументом змінну перелікового типу і видає ціле число, що є порядковим номером значення змінної у списку визначення. Причому перше значення у цьому списку має порядковий номер нуль, друге – один і т.д. Тобто ord(black) = 1;

У стандартній версії мови Паскаль (і TP) не дозволяється вводити і виводити на зовнішні пристрої значення перелікових типів.

Вперше перелікові типи введені в мові Паскаль, де є і їх піддіапазони. Тип сприяє виявленню багатьох помилок на етапі трансляції. Пізніше Bell Labs ввела цей тип в Сі (із заданням початкового числового значення першого імені), але без його контролю.

2.8. Структуровані типи

2.8.1. Однорідні структури – масиви. Попередньо визначені масиви (оперативної пам'яті, портів)

Масивом називається впорядкований набір даних однакового типу. Прикладами масивів можуть бути: рядок тексту, компонентами якого є окремі символи; вектор, що є масивом дійсних чисел; матриця, як масив векторів. Кожному простому значенню, що утворює масив, тобто кожному компоненту, відповідає сукупність номерів, що визначає місце цього компонента в загальній послідовності. Ці номери називаються індексами. Основними характеристиками сукупності значень, що утворюють масив, є:

ім'я, вимірність (тобто число компонентів), тип компонентів і типи індексів. Ім'я – це повна змінна типу масив, її значенням є весь масив. Загальна форма опису масиву така

type ім'я=array[<список_індексних_типів>] of <тип_компонентів>;

Індексним типом може бути довільний порядковий тип, крім longint і типу-діапазону з базовим longint. Список індексних типів записується через кому. Типом компонентів може бути довільний тип. Важливо, що час доступу до будь-якого компонента масиву не залежить від значення індексу. Тому масив вважається структурою *прямого доступу*.

Приклад опису масиву

```
type
mt = array[0..5, -2..2, char] of byte;
ti=1..10;
vector=array[ti] of real;
bom=array[ti] of boolean;
```

Тоді деякі змінні можна описати так:

Var x, y, v:vector; c: bom;

Вибір окремого компонента масиву здійснюється вказанням ідентифікатора масиву, за яким в квадратних дужках йде індексний вираз. Індексний вираз повинен давати значення, що є в діапазоні, визначеному типом індексу. Наприклад, для описаних вище типів змінних можна писати (визначивши перед тим i, j, cp):

... i:=5; v[i-j]:=7.5; j:=2; c[cp]:=false;

Ім'я з індексами в квадратних дужках – це окрема змінна, оскільки її значення є не весь масив, а його *окремий компонент*. В пам'яті комп'ютера елементи масиву розміщуються так, що під час переходу до старших адрес найбільш швидко змінюється самий правий індекс масиву.

В TP *попередньо визначені масиви*: mem, memW, memL і port, portW. Перші дозволяють безпосередньо звертатись до оперативної пам'яті за фізичними (абсолютними) адресами, інші використовувати для прямого керування пристроями введення-виведення (для доступу до портів комп'ютера).

Суфікси означають тип. Індексація при звертанні до елементів пам'яті має спеціальний вигляд: кожний індекс складається з двох виразів типу word; перший дає сегментну частину адреси, другий – зміщення. Вирази розділяються двокрапкою.

Наприклад:

Mem[$0000:$1000]:=0;

DataMem:=MemW[Seg(p):Ofs(p)]; { p – змінна }

MemLong:=MemL[64:SizeOf(real)];

Індексами масивів типу port мають бути вирази типу byte, що вказують номер потрібного порту. Присвоєння елементу масиву призведе до запису в порт, а використання у виразі – до читання з порту. Ці масиви не можна передавати як параметри процедурам і функціям, не можна використовувати без індексних виразів.

Процесор посилає дані чи інформацію керування в певний порт із заданим номером, а порт відповідає посиланням в МП даних чи інформації про свій стан.

2.8.2. Особливості роботи з масивами

Якщо змінні-масиви A і B однакового типу, то можливе присвоювання A:=B. Таке присвоювання є скороченим записом покомпонентного присвоювання.

Для задання значень елементам масиву, крім операторів присвоювання, можна використати оператори введення даних. В мові Паскаль введення-виведення масивів поелементне, тобто вводяться і виводяться окремі елементи масиву.

При роботі з масивами найчастіше виникають помилки, що пов'язані з виходом індексу за допустимі межі. Частина таких помилок виявляється компілятором. Наприклад:

```
Var M: array[1..8] of real;
... M[9]:=1.1;
```

Але якщо індекс є змінною, а не константою, то ця помилка виникає на етапі виконання програми. При трансляції компілятор вставляє в об'єктний код динамічні перевірки на належність поточних значень індексів допустимому діапазону. Ці перевірки збільшують розмір об'єктного коду, але роблять програму надійною.

В TP є можливість керувати процесом створення коду з допомогою директив компілятора. Директива { $R+ } в тексті програми приводить до створення об'єктного коду, в якому передбачені перевірки на належність індексів до допустимого діапазону їх значень.

Директива {$R-} виключає режим і приводить до "полегшеного" коду, що не містить відповідних перевірок. Цей режим прийнятий за замовчуванням. Директиви використовуються на етапах розробки, налагодження і тестування програми.

Іноді для найнебезпечніших фрагментів залишають перевірки. Для цього фрагмент беруть "в дужки":

```
{ $R+ }
Фрагмент програми
{ $R- }
```

2.8.3. Використання багатовимірних масивів

Оскільки, як було сказано, тип компонента масиву може бути довільним, то він може бути й масивом. В результаті отримуємо багатовимірний масив.

Приклад опису такого масиву:

Var M:array[A..B] of array[C..D] of T;

де T – раніше описаний тип (або стандартний).

Тоді j-компонента (типу T) i-го компонента масиву M = M[i][j]. Частіше використовується для багатовимірних масивів скорочена форма опису

Var M: array[A..B, C..D] of T;

а компоненти позначають M[i, j]. І в цьому випадку буде правильне використання M[i][j]. В мові Сі дужки [] є окремою операцією, тому не можна два індекси записати в одних дужках.

Константи, у визначенні індексних типів масиву при його *неявному* опису (тут: A, B, C, D) повинні бути визначені перед їх вживанням для опису масиву. Це можуть бути і константні вирази, як межі типу-діапазону.

Наведено приклад двовимірного масиву. Однак, звичайно, T в свою чергу може також мати тип масиву.

Приклад. Обчислити

$$z = (x_1 \Rightarrow x_2) \Rightarrow (x_2 \Rightarrow \neg x_1),$$

представивши імплікації $\Rightarrow$ у формі $\neg x_1 \vee x_2$, а коньюнкцію через дизьюнкцію $\neg(x_1 \vee \neg x_2)$.

```
{Таблиця істинності}
type b=boolean;
const t=true; f=false; im:array[b,b] of b=((t,t),(f,t));
st=#13#10#13#10#13#10;
var z:array[b,b] of b; i,j:b;
begin
writeln(st);
for i:=f to t do begin
for j:=f to t do begin
z[i,j]:=im[im[i,j], im[not j,not i]];
write(i:10, j:10,z[i,j]:15)
            end;
writeln      end
end.
```

2.9. Рядковий (стрінговий) тип

Рядковий тип реалізований в ТР і є узагальненням символьних масивів, дозволяючи на відміну від них динамічно змінювати довжину рядка. Загальний вигляд опису рядкового типу

Type <ім'я>=string<[<макс. розмір>]>;

де <макс. розмір> – ціле число <= 255. Якщо [] з розміром відсутні, то він має максимальне значення за замовчуванням – 255 для ПК.

Наприклад: type Line=string[80]; Var R:Line;

Тепер змінна R типу Line може як значення приймати будь-яку послідовність символів (типу char) довжиною від 0 до 80. Це значення може бути присвоєне або введене. Використовується операція *конкатенації* (зчеплення, з'єднання).

Наприклад:

writeln(R+'складає іспит 20-го січня');

Байти відповідно індексуються. Байт з нульовим індексом, як символ з кодом k, містить фактичну поточну довжину рядка, наступні k елементів – поточне значення рядка, і k+1..N – вільні. Тому можливе таке визначення довжини поточного значення рядка, як в операторі: writeln(ord(R[0]));

Частіше для визначення довжини рядка використовується функція Length, параметром якої є вираз рядкового типу.

Наприклад:

writeln(Length(' складає іспит 20-го січня'));

виведе число 26 – цілого типу (byte).

Крім операції конкатенації над змінними рядкового типу визначені операції порівняння:

writeln('ПАскаль'<'Пас':8); {true}

{'A' < 'a' = #$80 < #$a0}

В ТР до рядкових типів застосовний ряд *функцій*:

1) *concat*([s1, s2, ..., sn]: string): string – злиття довільного числа рядків. Якщо довжина результуючого рядка > 255, то відсікаються байти справа;

2) *copy*(s: string; Index: integer; Count: integer): string – повертає частину рядка s довжиною Count, починаючи з символу, що має номер Index;

3) *Insert*(source: string; var s: string; Index: integer) – процедура вставляє рядок source в рядок s, починаючи з позиції Index;

4) *Length*(s: string): integer – повертає довжину рядка s;

5) *Pos*(Substr, s: string): byte – результатом цієї функції є номер позиції, з якої в рядку s розміщений рядок substr.

Процедури перетворення.

str(x[: width[: decimals]], st) діє аналогічно процедурі writeln,

val(st, x, code) {code типу integer, має значення 0, якщо перетворення рядка в число успішне, інакше – це номер позиції з помилковим символом}.

delete(var s: string; Index, Count: integer) – ця процедура вилучає з рядка s частину цього рядка довжиною Count, починаючи з позиції Index;

Є також модуль Strings в TP для рядків типу PChar, подібного до типу рядків в мові Сі.

2.10. Множинний тип

2.10.1. Конструктор множини. Типізовані константи-множини. Операції над множинами

Множини – це набори однотипних об'єктів, певним чином пов'язаних між собою. Характер цих зв'язків контролюється програмістом.

Множина – *невпорядкований* набір різних об'єктів однакового типу. В мові Паскаль допускаються тільки скінченні множини, причому всі елементи множини повинні бути одного типу, який визначений в мові Паскаль. Тип елементів множини називається *базовим* типом. Як базовий, може бути довільний *ординарний* тип (тобто будь-який простий, за винятком real).

Опис типу: <ім'я типу> = set of <базовий тип>;

Для задання значення (наприклад, змінній чи константі) використовують *конструктор* множини: в довільному порядку список специфікацій елементів множини, відокремлених один від одного комами. Список береться в квадратні дужки ([] – порожня множина). Специфікаціями елементів можуть бути константи чи вирази базового типу, а також тип-діапазон того ж базового типу.

Реалізоване *компактне зберігання* множин. На кожний елемент резервується один біт пам'яті. Для опису: type ts=set of 0..15; виклик Writeln(SizeOf(ts)) дасть 2.

Приклади задання множин:

[4 , 6 , 12 , 9] – множина, елементами якої є цілі числа;

['l' , 'm' , 'p' , 'q' , 'r'] – множина, елементами якої є символи;

[m , 10] – множина, що містить значення змінної цілого типу m і цілого числа 10.

Якщо елементи множини утворюють діапазон значень базового типу, то ці елементи можна задавати скорочено аналогічно до задання діапазонного типу.

Наприклад, множину [3, 4, 5, 6, 7] можна представити: [3..7]. Частину елементів можна перерахувати, а частину задавати як діапазон: [3, 4, 5..100]. Специфікаціями елементів можуть бути константи чи вирази базового типу, а також тип-діапазон того ж базового типу.

Константи типу множини

Type dg=set of 0..9; lt=set of 'A'..'Z';

Const evdig: dg=[0, 2, 4, 6, 8];

vow: lt=['A', 'E', 'I', 'O', 'U', 'Y'];

hexdg: set of '0'..'z' =['0'..'9', 'A'..'F', 'a'..'f'];

Типізовані константи ініціалізуються тільки один раз – на початку виконання програми. При кожному новому входженні в процедуру чи функцію локально описані типізовані константи заново не ініціалізуються. Типізовані константи використовуються як змінні відповідного типу.

Операції над множинами в мові Паскаль практично збігаються з операціями в теорії множин (і для наочності можна будувати відповідні діаграми). Це в першу чергу операції об'єднання, перетину і різниці множин.

Нехай A і B – вирази однакового множинного типу, тобто це множини, елементи яких належать до одного базового типу.

Тоді:

- об'єднанням A+B множин A і B є множина, елементи якої входять хоча б до однієї множини A чи B (A∪B). Наприклад, A є [1,3,6,7], а B є

множина [2,4,6,7], то A+B – [1,2,3,4,6,7]; перетином A*B множин A і B є множина, елементи якої одночасно входять в множину A і в множину B (A∩B). Для тих же A і B перетин A*B буде [6,7];

- різницею A-B множин A і B називається множина, що складається з елементів множини A, які не входять в множину B (A\B). Якщо взяти ті множини A і B, що були використані раніше, то A-B дає множину [1,3];

- Використовуючи операції з множинами можна будувати вирази множинного типу. Пріоритетність виконання операцій та ж, що і при обчисленні арифметичних виразів: в дужках, потім *, після чого + і -.

Операції відношення над множинами і результати, які одержуються при відповідних значеннях A і B.

A=B (A = B) – множини A і B збігаються.

A<>B (A≠B) – A і B не збігаються.

A<=B (A⊆B) – всі елементи A належать до B.

A>=B (A⊇ B) – всі елементи B належать до A.

x in A (x ∈ B) – елемент x входить в A.

Остання бінарна операція серед наведених операцій відношення відрізняється тим, що перший операнд належить до базового типу, а другий – до множинного типу, побудованого на основі цього базового. Особливо слід зауважити, що операції порівняння > (більше) і < (менше) над операндами множинного типу не використовуються.

Приклади використання операцій відношення:

Нехай R:=[4, 7, 9, 11]. Тоді 5 in R рівне false [7, 11]<=R рівне false

[4, 7, 8, 11]=R дає false, а [7, 11]<=R дає true

Бажано, щоб всі операції з множинами виконувались швидко. Тому в деяких трансляторах обмежений максимальний розмір множин розміром слова відповідної машини. (Зокрема в Turbo Pascal допускається як базовий тип будь-який дискретний (ординальний тип), число елементів якого не перевищує 256, причому значення елементів для цілих типів повинні лежати в діапазоні від 0 до 255. (В ранніх трансляторах з мови Паскаль були випадки з цим обмеженням ще гірші).

Множини в системній мові для проекту SUE – цінна структура даних (мова SUE була створена для розробки операційних систем) [9, С. 285].

2.11. Організація комбінованих типів

2.11.1. Неоднорідні структури, записи зі сталою частиною. Розділи і поля запису. Типізовані константи-записи з константами-покажчиками

Спрощена форма визначення запису включає вказання імені запису, імен окремих компонентів (чи списку однотипних і відповідних їм типів даних: Type <ім'я_запису> = record

{Розділи – список полів одного типу}

<список_імен_компонент_1>: <тип1>;

<список_імен_компонент_2>: <тип2>;

. . . . . .

<список_імен_компонент_N>: <типN>

end;

Розглянемо приклад. Потрібно задати інформацію про студентів групи itp у вигляді: прізвище студента і оцінки на іспитах з 4-х дисциплін. Обчислити середній бал кожного студента. Для опису цих даних використаємо записи.

Компоненти однакового типу можуть об'єднуватися і записуватися через кому із вказанням їх спільного типу. Дані, наведені в прикладі, можна описати як запис таким чином:

Type STD = record prizv: string[24]; B1,B2,B3,B4:2..5; SB: real end;

Тут змінна SB має зміст середнього балу, B1, B2, ...; B4 – бали з відповідних дисциплін; ідентифікатор prizv позначає рядок для зберігання прізвища студента. Змінна STD – ім'я запису – це структура для зберігання інформації про одного студента.

Якщо, як в даному прикладі, в пам'яті треба зберігати інформацію про 24-х студентів групи, то вводиться масив записів itp1:

var itp1:array[1..24] of STD;

Компонент запису (елемент масиву записів) знаходиться за ім'ям запису (масиву) і ім'ям цього компонента, розділених крапкою (кваліфікація імені). Компоненти в записах називаються **полями**. З компонентами записів можна виконувати операції згідно їх типу.

В мові Паскаль немає жодної операції, яка сприймала б запис як єдиний об'єкт. Однак з допомогою оператора присвоювання можна пере-

силати значення одного запису в інший. Наприклад, враховуючи опис типу STD можна ввести змінні: Var Z1,Z2: STD; Тоді в програмі можна записати оператор Z1:=Z2;

2.11.2. Принцип доступу до полів запису і його спрощення. Оператор приєднання

Загальний вигляд оператора приєднання

with <ім'я_запису, ..., ім'я запису> do <оператор S>

Застосувавши оператор приєднання, до компонентів запису всередині оператора S можна звертатися вже тільки по імені поля, не вживаючи імені запису. Так оператор введення масиву itp1 може бути записаний

```
for i:=1 to M do
with itp1[i] do read(prizv,B1,B2,B3,B4);
```

Колізії в операторі приєднання розв'язуються зліва направо. Тобто, загальний вигляд оператора – це об'єднання вкладених операторів with в один.

2.11.3. Використання записів з варіантною частиною. Можливості перетворення типів. Використання імен

Загальний вигляд опису типу запису з варіантами:

```
Type  <ім'я_запису> = record
<ім'я_поля_1>: <тип>;
 . . . . . .
<ім'я_поля_N>: <тип>;
case [<ім'я >] | [<ім'я >: <ім'я_типу>] | <ім'я_типу> of
 <конст_1>: (<список_полів_1>);
 <конст_2>: (<список_полів_2>);
 . . . . . .
 <конст_M>: (<список_полів_M>)
end;
```

Необов'язкове <ім'я> можна використати відповідно до смислу вирішуваної задачі і пов'язати з константами варіантів. Взагалі константи варіантів не пов'язані з ключем, можуть бути однакових або різних типів.

Для прикладу наведемо опис типу "фігура", якою може бути одна з геометричних фігур: точка, пряма або коло. Оскільки інформація про кожну з таких фігур різна, використаємо запис з варіантами. Перед тим опишемо деякі потрібні типи.

```
Type  coord = record  absc, ordi: real end;
forma=(tchk, prjam, kolo);
figure = record FIG: forma;
case forma of
tchk  : (locat: coord);
prjam: (koefA, koefB, koefC: real);
kolo  : ( centr: coord; radius: real)
          end;
{Особливості задання констант варіантів і ключа:}
rec1 = record a:byte; b:word; f:text end;
rec2 =  record c:longint;
case x: byte of
1: (d:word);
2: (e: record
   case Boolean of
   3  : (f:rec1);
   3  : (g:single);
  '3' : (c:word)
     end)
        end;
var p: rec1;   r: rec2;
```

2.11.4. Типізовані константи-записи з варіантними полями

Для констант-записів з варіантними полями вказується тільки один з можливих варіантів констант. Для однієї константи – один варіант

```
person1:forma(country:'Польща');
```

а для іншої – другий варіант

```
person2:forma(BirthPlace:'Київ');
```

2.12. Організація доступу до фізичних файлів (наборів даних)

2.12.1. Оголошення змінних файлового типу

Існує багато задач, коли кількість компонентів певного типу (будь-якого з відомих уже нам) наперед визначити неможливо, вона визначається в процесі виконання програми. Для роботи з такими даними в мові Паскаль передбачено файловий тип, що представляє собою структуру

даних, яка є певною послідовністю компонентів упакованого типу і невизначеної наперед довжини.

Окремо розглянемо файли послідовного доступу. Послідовний доступ передбачає можливість рухатися по файлу тільки послідовно, починаючи з першого компонента. При цьому завжди доступним є тільки наступний елемент. Якщо ж треба звернутися до якогось із попередніх елементів, то необхідно вернутись на початок і послідовно пройти всі елементи до потрібного.

Файловий тип є єдиним засобом зв'язку програми із зовнішнім середовищем. Тільки через файли в мові Паскаль можливо передавати в програму вхідні дані і передавати з програми результати.

Особливості файлів:

1. У файла є ім'я.

2. Файл містить компоненти одного типу.

3. Тип компонентів файла довільний, окрім файлового.

4. Довжина файла, що створюється, завчасно не обумовлюється.

Вона обмежена тільки ємністю зовнішньої пам'яті.

Файловий тип чи *змінну файлового типу* можна задати одним з трьох способів:

<ім'я> = file of <тип>;

<ім'я> = text;

<ім'я> = file;

І в залежності від способу декларації можна виділити три види файлів: *типізовані, текстові, нетіпизовані.*

Приклади оголошень:

```
type
product = record
name: string;
code: word;
cost: comp
end;
text80= file of string[80];
var
f1: file of char;
f2: text; f3: file;
f4: text80;
f5: file of product;
```

fa: array[1..10] of text;

2.12.2. Підготовчі та завершальні операції з файлами

Операції цієї групи реалізуються процедурами ASSIGN, CLOSE, FLUSH, RESET, REWRITE.

а) ASSIGN – призначена для встановлення зв'язку між фізичним файлом на магнітному носії і файловою змінною. Звертання до процедури:

Assign (<ім'я_файлової_змінної>, '<зовнішнє_ім'я_файла>').

Ім'я фізичного файла будується за правилами MS-DOS, тобто може містити ім'я диска, ланцюг каталогів і повне ім'я файла (з розширенням). Імена поточного диска чи каталога можна не вказувати. Ім'я фізичного файла – це рядок символів і тому береться в апострофи. Наприклад:

Assign (f, 'E:\dani\fl.dat')

б) RESET і REWRITE – призначені для відкриття файлів:

Reset(<ім'я_файлової_змінної>)

Rewrite(<ім'я_файлової_змінної>)

Файлова змінна повинна до відкриття уже бути зв'язаною з конкретним дисковим файлом з допомогою процедури Assign.

Під відкриттям файла розуміють:

- вказання напряму обміну інформацією із зовнішнім носієм (для читання чи для запису);
- пошук (і створення для виведення, якщо такий файл не існує) файла на зовнішньому носії;
- утворення системних буферів для обміну інформацією;
- встановлення вікна (покажчика початку потрібної підмножини набору даних) файла на його початок, тобто на нульовий елемент.

в) FLUSH – для завершення обміну без закриття файла:

Flush (<ім'я_файлової_змінної>)

"скидає" з буфера останні елементи в файл під час записування.

Close теж виконує цю функцію, крім інших.

г) CLOSE – для завершення дій з файлом:

Close(<ім'я_файлової_змінної>)

При цьому ліквідуються всі внутрішні буфери, утворені під час відкриття цього файла. Після цього файлову змінну можна зв'язати з іншим дисковим файлом з допомогою Assign. Слід зауважити, що по завершенні програми всі файли автоматично закриваються. Однак для гарантії краще використати Close.

2.12.3. Операції введення-виведення для типізованих файлів

Ці дії виконують дві операції-процедури, які реалізують читання і запис інформації у файл:

Read – читання;

Write – запис.

а) Процедура read (f, v) присвоює змінній *v* значення компонента файла *f*, на яку вказує вікно, і пересуває вікно (покажчик) на наступну позицію файла.

Може бути декілька параметрів процедури read(f, v1, v2, ...). Компоненти файла послідовно присвоюються цим змінним. Після читання кожного компонента вікно файла зміщується на одну позицію.

б) Процедура write (f, x) записує в файл *f* в позицію, на яку вказує вікно, чергового компонента, який рівний значенню виразу *x*, а вікно зсувається на наступну позицію файла. Зрозуміло, що тип виразу чи змінної x повинен збігатися з типом компонентів файла *f*.

2.12.4. Послідовність роботи з типізованими файлами. Прямий доступ

Як тип компонентів може бути завдання або ім'я будь-якого типу, крім файлового або такого, що містить в собі файловий тип. Наприклад, не може бути оголошений файловим типом набір даних, компонентами якого є записи, що містять в якості одного або декількох полів компонента файлового типу.

Приклад опису файлового типу

```
type
vect = array [1..10] of char;
F=file of vect;
g=record nnd: string; nf: f end;
```

Тут F описаний як файл, компонентами якого є масиви 10-ти символів. Цей же файл може бути введений і в розділі опису змінних:

Var FL: file of vect; a: array[1..5] of f; якщо тип vect визначений. Якщо ні, то тип визначається разом з оголошенням змінної:

Var FL: file of array [1..10] of char;

Слід зауважити, що Turbo Pascal допускає використання так званих безтипових файлів, для яких тип компонентів не встановлюється (такі файли розглянемо дещо пізніше).

Над значеннями файлового типу не визначені жодні арифметичні чи логічні операції, навіть операції присвоювання чи порівняння (на відміну від змінних комбінованого типу, для яких дозволені операції присвоювання значень змінних одного і того ж комбінованого типу).

Для файлів допускаються тільки операції з компонентами файлів. Множина операцій визначається типом компонентів. Зате передбачено цілий ряд операцій над файлами у вигляді стандартних процедур. Зокрема широкий набір цих процедур в TP, тому не будемо обмежуватися розглядом тільки засобів стандартної версії мови.

Процедури assign, reset, rewrite, read, write, seek. Функції filesize, filepos.

Приклад використання:

```
{ Типізовані файли}
type rcf = record nam: string[10]; kol: integer; ves: real end;
var namf: file of rcf; rab: rcf;
begin
assign(namf,'rezves');
rewrite(namf);
 with rab do
 begin
 nam:='Оля'; kol:=1000;  ves:=128;  write(namf, rab);
 nam:='Саша'; kol:=300;  ves:=64;  write(namf, rab);
 nam:='Віктор'; kol:=100; ves:=1.5; write(namf, rab);
 end;
reset(namf);
read(namf, rab);                  { на перший запис }
writeln(rab.nam);                 { 'Оля' }
seek(namf, FileSize(namf)-2); { Якщо -1, то покажчик }
read(namf, rab);                  { буде після остан.запису }
writeln(rab.nam);                 { 'Саша' }
close(namf)
end.
{03 4f ab ef  00 00 00 00  00 00 00 e8  03 88 00 00
 00 00 00 04  91 a0 e8 a0  00 00 ... – початок файлу rezves}
```

2.12.5. Процедури введення-виведення для стандартних (текстових) файлів

Введення і виведення значень змінних здійснюється за допомогою операторів процедур read і write. Близькими до них є оператори readln і writeln (read line, write line). Відрізняються вони тим, що після введення чи відповідно виведення величин, що вказані як параметри, здійснюється перехід на новий рядок (стандартні файли – текстові).

Наприклад:

readln(A, B, C, D) – зчитує A, B, C, D з одного рядка і переходить до читання з наступного.

writeln(A); – виведення значення A;

writeln; – пропуск рядка;

writeln(B); – виведення значення B.

У процедур виведення формальний параметр є параметром-значенням. Тому, звичайно, в операторі виклику write (і writeln) може бути константа (взагалі вираз):

writeln('Введіть дані');

Процедури мають *змінну кількість параметрів.*

2.12.6. Нетипізовані файли. Довжина запису. Високошвидкісні процедури

Оголошуються реченням file і відрізняються тим, що для них не вказаний тип компонентів. Це робить їх сумісними з будь-якими іншими файлами. З іншого боку, дозволяє створювати високошвидкісні процедури обміну даними між диском і пам'яттю. При ініціалізації нетипізованного файла процедурами reset чи rewrite *другим параметром* можна вказувати довжину (в байтах) запису, наприклад

Var f:file;

Begin assign(f, 'vyf.dat'); reset(f, 512);

В другому параметрі можна використати вираз типу word. Якщо довжина запису не вказана, вона приймається рівною 128. В TP немає ніяких обмежень на довжину, окрім вимоги додатності і обмеження максимальної довжини типу (word – 64 K).

Для забезпечення максимальної швидкодії потрібно задавати довжину запису, кратну довжині фізичного сектора диска (512 K).

Кластер – 4-8 суміжних сектори вінчестера (2 – гнучкого диска) може бути прочитаний чи записаний за один оберт диска. Найвища швидкість обміну – при довжині запису рівній довжині кластера.

Але, для невідомої можливої реальної довжини запису (наприклад, при стисненні-розширенні) аргументом вказується один байт.

Високошвидкісні процедури

blockread(<файлова змінна>,<буфер>,<N>, [,<NN>]);

blockwrite(<файлова змінна>,<буфер>,<N>, [,<NN>]);

<буфер> – ім'я змінної, яка візьме участь в обміні даними з дисками;

<N> – кількість записів, які мусять бути прочитані чи записані за одне звертання до диска;

<NN> – необов'язковий параметр, що містить після виходу з процедури кількість фактично оброблених записів.

За одне звертання – N*recs байтів, recs – довжина запису нетипізованного файла.

Передача йде з першого байта змінної <буфер>. Якщо довжина цієї змінної недостатня для розміщення N*recs байтів, чи на диску не залишиться місця, виникає помилка введення-виведення, яку можна заблокувати, вказавши необов'язковий параметр <NN> типу word.

Якщо при запису не залишається місця, то покажчик файла зміщується на <NN> записів.

Приклад зі стандартної допомоги TP:

```
uses Crt,Dos;
{program CopyFile;}
var
 FromF, ToF: file;
 NamIn, NamOut: string;
 NumRead, NumWritten: Word;
 Buf: array[1..2048] of Char;
begin
If ParamCount>0 then
NamIn:=ParamStr(1)
 else
begin
writeln(#7,'Немає імен файлів');
halt(111)
end;
```

```
If ParamCount>1 then
NamOut:=ParamStr(2)
 else
NamOut:='$$$';
 Assign(FromF, NamIn); { Open input file }
 Reset(FromF, 1); { Record size = 1 }
if IOResult<>0 then { Перевірка існування }
begin
writeln(#7,'Не існує файл ',NamIn);
halt(222)
end;
 Assign(ToF, NamOut); { Open output file }
 Rewrite(ToF, 1); { Record size = 1 }
 Writeln('Copying ', FileSize(FromF), ' bytes...');
 repeat
  BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
  BlockWrite(ToF, Buf, NumRead, NumWritten);
 until (NumRead = 0) or (NumWritten <> NumRead);
 Close(FromF);
 Close(ToF);
end.
```

2.12.7. Можливості обробки помилок введення-виведення

Можна організувати самостійно контроль помилкових ситуацій. Для цього з допомогою директиви {$I-} відключається автоматична перевірка, тоді при виникнені помилки робота програми не припиняється, а код помилки запам'ятовується системою. З допомогою стандартної функції IOResult без параметрів можна одержати цей код і побудувати подальші дії залежно від його значення, наприклад:

```
Assign (f,'d:\myfile');
{$I-}
Reset(f);
{$I+} { Включення автоматичного контролю }
if IOReset<>0 then
write ('Помилка при відкритті файла');
```

При виникненні помилки введення-виведення у випадку відключеного автоматичного контролю (директива { $I-}) всі подальші операції з

будь-яким файлом ігноруються до звертання до функції IOResult. Тому правильним є звертання до IOResult відразу після операції з файлом. Функція IOResult повертає код помилки в програму і обнулює цей код. Якщо ж помилки немає, то IOResult повертає в програму нуль.

2.12.8. Використання текстових файлів. Процедури доступу. Кінець рядка, кінець файла

В мові Паскаль окремо виділяються текстові файли, яким відповідає стандартне ім'я text.

Наприклад:

```
type T = text;
```

Тип text є стандартним і тому в програмі не описується. Додатково для текстових файлів визначені такі процедури.

Append – відкриває файл для запису в кінець його – дописування. Звертання: Append(<ім'я_файлової_змінної>) Процедура не очищує файл, а тільки встановлює вікно файла в його кінець. Процедуру зручно використовувати для додавання нових рядків в кінець вже існуючого текстового файла.

SetTextBuf – визначає буфер для обміну з текстовим файлом. Використовується для прискорення обміну з файлом. При відкриванні файла з файловою змінною зв'язується системний буфер розміром 128 байт, через який здійснюється обмін. Цей буфер виділяється автоматично і доступ до нього неможливий. Процедура SetTextBuf дозволяє сформувати його не в системній області, а в самій програмі і за необхідності задати його розмір. Звертання:

SetTextBuf(var F: text; var Buf; Size: word)

Тут F – файлова змінна (<ф. з.>), файл не повинен бути відкритий; Buf – змінна для розміщення буфера (нетипізований параметр); Size – розмір буфера в байтах. Цей параметр необов'язковий. При його відсутності за замовчуванням встановлюється розмір 128 байт.

Для текстових файлів не використовується процедура Seek, оскільки довжина рядків невизначена. Зате використовуються функції SeekEoLn і SeekEoF.

Бульова функция EOLN(<ф. з.>) – повертає true, якщо в вхідному текстовому файлі маркер досяг кінця рядка. Якщо <ф. з.> відсутня, то функція перевіряє стандартний файл input.

2.12.9. Процедура write

Формат процедури

write (<ф.з.>, <список виведення>),

де <список виведення> – один чи більше виразів типу char, string, boolean, а також будь-якого цілого або дійсного типу, не структурного.

Формат вивідного параметра:

OutExpr [:MinWidth [:DecPlases]], де MW та DP – вирази типу word.

[] означають, що параметри можуть бути відсутні.

MW – мінімальна ширина поля, в яке будуть записуватися всі символьні представлення значення OutExpr. Якщо параметр відсутній, то приймається MW=23. Якщо MW<10, то приймається MW=10.

DP – кількість десяткових знаків у дробовій частині дійсного числа. Якщо параметр DP не вказаний, то дійсні числа виводяться в експоненціальному форматі.

Процедура Writeln повністю індентична процедурі write, за винятком того, що рядок символів завершується CR і LF. Окрім <списку виведення> в файл передається маркер EOLN. При виведенні на екран – курсор встановлюється на початок наступного рядка.

Приклад:

```
{Програма читає два цілі числа та використовує їх як параметри при виведенні різних дійсних чисел}
Program UsingWrite;
var
MW, DP, i: word; a: real;
begin
repeat
write ('MinWidth, DecPlase =?');
readln (MW, DP);
if MW<>0 then
begin
      a:=sqr(pi);
      for i:=1 to 5 do
      begin
      writeln ( 'exp:', a:MW, 'fix:', a:MW:DP);
      a:=sqr(a)
      end;
end;
```

```
until MW=0
end.
```

Подібний приклад (можливої зміни формату виведення, роботи з рядками символів) мовою Сі:

```
#include <stdio.h>
#include <string.h>
#define n 5
main()
{ int i = 5;
char a[79], b[79];
char* ic=a; char* jc=b;
ic="%d";
b[0]='\0'; a[1]='\0';
printf(ic, i);
 ic=strcat(ic,"++");  //Використання ненадійності. Модифікація
}                     // програми
```

2.12.10. Особливості работи функцій EOLN та EOF з дисковими файлами та логічними пристроями

Для логічного пристрою неможливо передбачити, яким буде результат читання символа. При читанні функція EOLN повертає true, якщо останнім прочитаним символом *був* CR чи Ctrl-Z.

При читанні з диску EOLN повертає true в випадку, якщо наступним прочитаним символом *буде* CR або Ctrl-Z.

Приклад. {[Довгаль С.И. и др. ТурбоПаскаль 6.0, С. 55]
З файла Input вводиться текст, який містить символи від знаку "+" до лівої квадратної дужки "[". Роздрукувати символи тексту в послідовності кодів ASCII (з тих символів, що повторюються виводити тільки один).}

```
{program Sort(input,output);}
type
 ts='+'..'['; { +[ AZ az->2B5B 415A 617A}
var
   s:char;
 Sets: set of ts;
   i:ts;
begin
```

```
    Sets:=['[','+',char($2b)];
    read(s);
    while not eof do
     begin { До <Ctrl+z>}
     while not eoln do {Введення рядків}
      begin {До <enter>}
{Символи, які не належать діапазону ts, не додаються в Sets}
     Sets:=Sets+[s]; {Дужки обов'язкові}
      read(s)
      end;
  readln
    end;
for i:='+' to '[' do
if i in Sets
  then write(i)
  else;
  writeln
end.
```

Якщо не буде оператора readln, який скидає символи $0A, $0D з буфера клавіатури (очистка буфера від eoln), то програма зациклиться в операторі while not eof і пропускатиме оператор while not eoln (там хибне значення). Покажчик буде залишатись в кінці рядка.

Бульова функція **seekEoLn** (<ф.з.>) пропускає всі пропуски (інтервали), знаки табуляції до маркера eoln або до першого значущого символу і дає результат true, якщо маркер знайшовся.

Бульова функція **seekEoF** (<ф.з.>) пропускає всі інтервали, знаки табуляції,маркери кінця рядка до маркера файла чи до першого значущого символу; результат true – якщо маркер знайшовся.

Процедура read пристосована до введення чисел, "перескакує" маркери кінця рядка, тобто фактично весь файл розглядається як один довгий рядок, який містить текстові представлення чисел.

Приклад програми введення масиву даних.

```
Const N=1000;
Var f: text; m: array [1..N] of real; i: integer;
begin
      assign (f, 'prog.dat'); reset (f);
```

```
i:=1;
while not eof(f) and (i<=N) do
begin read (f,m[i]); inc(i) end;
...
```

Процедуру readln використовують для введення текстових рядків.

Бульова функция EOLN (<ф. з.>) – повертає true, якщо в вхідному текстовому файлі маркер досяг кінця рядка. Якщо <ф. з.> відсутня, то функція перевіряє стандартний файл input.

2.12.11. Операції переміщення покажчика файла

Використовуються дві процедури Seek і Truncate і додатково три функції FileSize, FilePos і Eof.

Процедура Seek змінює положення покажчика файла, встановлюючи його на елемент файла з заданим номером. Звертання до процедури:

Seek (<ім'я_файлової_змінної>,<номер_елемента>)

Другий параметр – ціле число типу longint. Після виконання процедури Seek подальше читання чи запис можна здійснювати з позиції, на яку переміщений покажчик файла. Початкове значення покажчика 0.

Функція FileSize повертає число елементів файла.

Функція FilePos повертає номер елемента, на який встановлене вікно.

Обидві функції мають один параметр – ім'я файлової змінної.

Приклади:

Seek(f, FilePos(f)+1) – пропуск одного елемента;

Seek(f, 0) – встановлення покажчика на початок;

Seek(f,FileSize(f)) – встановлення покажчика за останнім елементом, для дописування в файл.

Функція Eof перевіряє кінець файла. Оскільки число компонентів в файлі ніде не фіксується, при обробці файлів важливо мати можливість визначати, чи не вийшло вікно за межі файла. Для цього в мові Паскаль використовується логічна функція eof(f). Значення цієї функції рівне true, якщо вікно вказує на маркер кінця файла, тобто на позицію, що слідує за останнім компонентом файла. В протилежному випадку значення функції eof(f) рівне false. Якщо значення функції eof(f) рівне true, то звертання до процедури читання з цього файла приводить до помилки.

Процедура Truncate(f) відсікає частину файла, починаючи з позиції вікна файла, включно з нею.

2.12.12. Спеціальні операції з файлами. Процедури пошуку в директорії. Тип запису з файловою інформацією

Процедури, які призначені для дій з елементами файлової системи MS-DOS:

Erase – вилучення файла на диску;
Rename – перейменування файла;
Chdir – встановити поточний каталог;
Mkdir – створити новий (під-) каталог;
Rmdir – вилучити пустий підкаталог.

Тип запису з файловою інформацією

```
type
SearchRec = record
file: array[1..21] of byte;  { Резервується ОС. Не користувача }
Attr: byte;                  { Атрибути. Формуються з констант }
time: longint;
Size: longint;               { Розмір в байтах }
Name: String[12] end;
```

2.13. Динамічна пам'ять

Використання динамічних змінних значно розширює можливості написання великих програм, оскільки об'єм пам'яті під статичні змінні дуже обмежений. Під динамічну пам'ять відводиться практично весь об'єм оперативної пам'яті крім пам'яті для коду програми і статичних змінних.

Однак часто такий об'єм динамічної пам'яті надлишковий, часом і недопустимий, оскільки програма не може викликати іншу програму, розміщену в EXE-файлі. Інші програми можуть взагалі не використовувати динамічну пам'ять.

Керування розмірами динамічної пам'яті. В TP для встановлення розмірів динамічної пам'яті використовується директива компілятора $M. Ця директива розміщується на початку програми і має три параметри. Перший визначає максимальний розмір пам'яті під стек локальних змінних, два інші задають мінімальний і максимальний розміри динамічної пам'яті. Приклад:

```
{ $M 16384, 1024, 65000 }
```

Мінімальний розмір встановлює той мінімум, нижче якого програма на може виконатися. Якщо він рівний нулю, то програма буде

виконуватись завжди. Третій параметр встановлює максимальний об'єм пам'яті, який може бути використаний під динамічні змінні. Іншими словами розмір динамічної пам'яті встановлюється, виходячи з реальної наявності пам'яті, на початку роботи програми, але не перевищує максимального розміру.

За замовчуванням встановлюються наступні параметри директиви

{ $M 16384, 0, 655360 }

тобто під стек виділяється 16К, під динамічну пам'ять – вся вільна оперативна пам'ять, доступна операційній системі.

2.14. Покажчики

2.14.1. Типізовані покажчики. Покажчик Nil. Типізовані константи-покажчики

Значенням покажчикового типу є покажчик на деякий програмний об'єкт – динамічну структуру даних, за якою здійснюється доступ до цього об'єкта. Після стрілки "^" вказується тип цього динамічного об'єкта. Важливо пам'ятати, що цей тип може вказуватися тільки з допомогою імені типу – стандартного чи раніше описаного, але в жодному разі не з допомогою безпосереднього задання типу.

Наведемо приклад опису змінних типу покажчик:

```
Type vect = array[1..10] of char; mas = ^vect;
Var p: ^real; q: ^integer; m: mas;
```

Далі розглянемо, як вводяться самі динамічні об'єкти. Введення змінної типу покажчик не приводить до покажчика на якийсь об'єкт, ця змінна не містить навіть значення порожнього покажчика nil. Опис змінної типу покажчик вводить тільки деяку статичну змінну, яка може містити покажчик на об'єкт. Сам же динамічний об'єкт ним не визначається.

Для породження динамічного об'єкта, зв'язаного з деякою змінною типу покажчик, в мові Паскаль використовується стандартна процедура (функція) new. Фактичним параметром цієї процедури може бути тільки змінна типу покажчик.

В результаті виконання оператора виклику new(m) в процесі виконання програми створюється новий динамічний об'єкт, тип якого вказаний при описі змінної типу покажчик, що є фактичним параметром процедури new. Крім цього змінній типу покажчик (в даному випадку m) присвоюється посилання на новостворений динамічний об'єкт.

Є лише одна нетипізована константа типу покажчик (іноді називають

константою типу посилань) – nil (пусто). Бажано ініціалізувати покажчик значенням nil, якщо інше не передбачено.

Приклад з документації TP:

```
Type NamePtr = ^NameRec;
NameRec = record
            Next: NamePtr;
            Name: string[31];
            End;
Const
NameList: NamePtr = nil;
NoName: NameRec = (Next: nil; Name: '');
```

2.14.2. Робота з динамічними змінними. Функції і процедури для роботи з купою і типізованими покажчиками

При роботі з динамічними змінними необхідно дотримуватись правила: звільняти виділені області пам'яті під динамічні змінні, якщо вони в подальшій роботі не використовуються. Інакше використання динамічних змінних приводить до швидкого "засмічення" динамічної пам'яті і її переповнення.

Ще одна особливість пов'язана із стековим принципом розміщення статичних змінних, якими можуть бути змінні типу покажчик.

Розглянемо приклад:

```
program Noreference;
    type anketa=record
                ......
                end;
    aan=^anketa;
procedure GetAnketa;
var an: aan;
begin an:=New(aan) end;
begin writeln(MemArail); GetAnketa; writeln(MemArail); end.
```

Тут після виходу з процедури GetAnketa втрачається доступ до змінної *an* типу покажчик, а значить і до динамічного об'єкта, на який вона вказує. В той же час об'єкт типу anketa залишається в динамічній пам'яті, відповідне місце не звільняється, що підтверджує результат виведення розміру вільної динамічної пам'яті (Heap-області) перед і після виконання

процедури. В описаному прикладі маємо справу з втратою покажчика на динамічний об'єкт.

Для уникнення таких випадків треба при виході з блока (процедури), в якому описані локальні змінні, що є покажчиками на відповідні динамічні об'єкти, або знищувати ці об'єкти, якщо вони надалі непотрібні, з допомогою процедури Dispose, або зберегти покажчики на них, присвоївши ці покажчики, наприклад, глобальним змінним.

Heap-область – стекоподібна структура ОП, пам'ять в якій динамічно розподіляється при відповідних програмних запитах.

При виділенні пам'яті для динамічної змінної система керування Heap-областю пересуває покажчик вгору, адреса зростає. При звільненні, навпаки, зменшується. При цьому в середині зайнятої пам'яті з'являються вільні ділянки. На верхній границі ОП є список записів з даними про наявність вільного простору в Heap-області.

В результаті пам'ять фрагментується, перетворюється в *сукупність* {купу (укр. куп||а 1. heap, pile;)} ділянок зайнятої і вільної пам'яті різних розмірів.

2.14.3. Можливості перетворення покажчикових типів. Небажані перетворення. Робота з покажчиками в купі і в сегменті даних

Вказівні типи вважаються різними відповідно до загальних правил, хоча значення покажчикових змінних визначають адреси в оперативній пам`яті. Так, типи, описані як type p= ^real; q= ^integer; визначають різні множини значень і змінні var s: p; j: q; не можуть передавати одна одній значення, наприклад шляхом присвоєння. Якщо ж var s, j: p; то можна записати s:=j.

Для покажчикових типів допускаються конструкції явного перетворення. Наприклад, задано такі описи:

Type rac=record re, im: real end; rrac=^rac; ll=^longint;

Var v1: rrac; v2: ll;

Тут v1, v2 є покажчиками на значення різних типів. Але конструкція приведення дозволить трактувати їх покажчиками на однаковий тип:

```
rrac(v2)^.re:=5.7;
ll(v1)^:=729364;
```

```
type
                    Небажані перетворення
var
  i,j:^integer; r:^real;
```

```
begin
new(i);        {i:=HeapOrg; HeapPtr:=HeapOrg+2}
j:=i;          {j:=HeapOrg;}
dispose(i);    { HeapPtr:=HeapOrg}
new(r);        {r:=HeapOrg; HeapPtr:=HeapOrg+6}
r^:=pi;        {Функція pi дає число π}
writeln(j^)
{Виведення числа 8578 – внутрішнє подання частини pi як цілого }
end.
```

Робота в сегменті даних

```
{ Покажчики. Динамічна пам'ять }
type uk1=^real; dme=1..3;
var
 a: array[dme, dme] of real;
 k: byte;
 um: uk1;
begin
 writeln(' Вільна динамічна пам'ять=',MemAvail:10);
 um:=addr(a);
 prab:=@a;
 {um:=prab;}
 writeln('seg=',Seg(um^):5,' ofs=',Ofs(um^):5);
 for k:=1 to 9 do
 begin
 um^:=k;
 um:=ptr(Seg(um^), Ofs(um^)+sizeOf(real));
 writeln('seg=', Seg(um^):5, ' ofs=', Ofs(um^):5)
 end;
 writeln(a[1,1]:3:0, a[2,3]:3:0, a[3,3]:3:0);
 {1 6 9 – масив в пам'яті розміщується рядками }
end.
```

2.14.4. Нетипізовані покажчики. Процедури виділення і звільнення динамічної пам'яті. Адресні функції і унарна операція. Сумісність з типізованими покажчиками

Можна оголосити покажчик і не зв'язувати його при цьому з яким-небудь конкретним типом даних; для цього використовується службове

слово pointer:

var pp: pointer;

Оголошений таким чином покажчик називається нетипізованим. З допомогою такого покажчика зручно динамічно розміщувати дані, структура яких змінюється під час роботи програми.

Передавати значення можна тільки між покажчиками, пов'язаними з одним і тим же типом даних чи нетипізованими:

var p1, p2:^integer; p3:^real;

begin ... p1:=p2; pp:=p3; p1:=pp;

Для роботи з нетипізованими покажчиками використовуються процедури

getmem(p,size) – резервування пам'яті,

freemem(p,size) – звільнення пам'яті,

де p {:pointer} – змінна нетипізованного покажчика, який буде вказувати (чи вказував) на відповідний фрагмент динамічної пам'яті.

size {:word} – розмір в байтах частини купи, що виділяється чи звільняється.

Унарна операція @ застосовна до операндів довільного типу і повертає результат типу pointer, що містить адресу операнда.

Аналогічний результат вертає функція Addr(x), де x – будь-який об'єкт програми (ім'я змінної, процедури, функції).

За допомогою вбудованої функції ptr(seg, ofs: word){:pointer} за сегментом і зміщенням можна створити значення покажчика, сумісного з покажчиком будь-якого типу. Разом з вбудованими функціями ofs, seg, dseg вона є базою адресної арифметики.

Концепцію посилань запропонував Н. Вірт в 1965 році, коли викладав курс системного програмування в Стенфордському університеті. Його асистентом був А. Шоу [А. Шоу].

Оголошення константи типу покажчик звичайно використовує константний адресний вираз, щоб вказати значення.

```
type
Direction = (Left, Right, Up, Down);
StringPrt = ^String;
NodePtr = ^Node;
Node = record Next: NodePtr;  Symbol : StringPrt; Value: Direction end;
const
S1: string[4] = 'DOWN'; S2: string[2] = 'UP';
```

```
S3: string[5] = 'RIGHT'; S4: string[4] = 'LEFT';
  N1: Node = (Next: nil; Symbol:@S1; Value: Down);
  N2: Node = (Next: @N1; Symbol:@S2; Value: Up);
  N3: Node = (Next: @N2; Symbol:@S3; Value: Right);
  N4: Node = (Next: @N3; Symbol:@S4; Value: Left);
  DirectionTable: NodePtr = @N4;
```

В мові Сі можливий вираз *++*argv, коли char** argv (покажчик на масив покажчиків на символ). Д. Рітчі (один з розробників Сі) зауважує, що такий запис нагадує АПЛ [15, с. 167].

2.15. Процедури і функції

Першими програмними інструментами, які економили труд програмістів, були підпрограми. На думку Д. Кнута, можливо, що ще Ч. Беббідж помітив їх доцільність в звязку зі своєю аналітичною машиною (проте тільки в тих межах, в яких ця ідея застосовна до перфокарточного керування верстатом *Жаккарта*, оскільки він виконував керуючі функції в машині) [9, С. 30]. {Карта. J.M. Jacquard'a машина роздільно керувала кожною ниткою основи або невеликою їх групою}.

Все ж, мабуть, в Ж.М. *Жаккара* (1801 р.), а спочатку в шовкоткацькому верстаті Ж. де Вокансона (1741 р.) і його автоматах була перфострічка, її прообраз [2, С. 672]. Г. Холлерит (IBM) в 1890 році використав перфокарту для перепису населення США.

Х. Голстайн і Дж. фон Нейман запропонували в 1946 році детальну методику переміщення команд і виконання зв'язку між підпрограмами. Використання і конструювання дуже гнучкої бібліотеки підпрограм було головною темою першої книги з програмування (ИЛ, М., 1953).

2.15.1. Використання принципу локалізації

Труднощі, що виникають при збіганні ідентифікаторів, усуваються в мові Паскаль завдяки блочній структурі програми і принципу локалізації, суть якого полягає в тому, що імена, які вводяться в певній процедурі, мають силу тільки в межах цієї процедури.

Опис процедури може містити 3 види ідентифікаторів: формальні параметри, глобальні і локальні ідентифікатори.

Глобальні ідентифікатори – це ідентифікатори, які не є

формальними параметрами і не описані в тілі процедури. Глобальні ідентифікатори в тілі процедури можуть мати той самий зміст, що й до моменту входу в процедуру.

Таким чином, з допомогою глобальних ідентифікаторів здійснюється безпосередній зв'язок процедури з блоком, що її охоплює, обминаючи формальні параметри.

Локальні ідентифікатори – це ідентифікатори, які не є формальними параметрами, але описані в тілі процедури.

Принцип локалізації полягає в тому, що якщо якийсь ідентифікатор описаний в тілі процедури, то всередині процедури цей ідентифікатор приймає зміст згідно з описом. Якщо ж цей ідентифікатор був описаний ще й поза тілом процедури – в блоці, що її охоплює, то дія попереднього опису тимчасово (до виходу з процедури) припиняється.

Зокрема, якщо ідентифікатор в попередньому описі, був іменем деякого програмного об'єкта, то цей об'єкт стає недоступним з тіла процедури. При виході з процедури втрачає свою дію опис в тілі процедури і цей ідентифікатор знову приймає той зміст, що він мав до входу в процедуру (якщо він був описаний поза процедурою).

Якщо ідентифікатор, локалізований в тілі процедури, є іменем програмного об'єкта, то при виході з процедури він припиняє своє існування, а при вході в процедуру породжується знову.

2.15.2. Формальні параметри процедур. Розділи const і var в списку формальних параметрів. Параметри-значення

В заголовку блока (процедури чи функції) оголошується ім'я блока і формальні параметри, якщо вони є. Для функції крім того вказується тип результату, який вона повертає. За заголовком розміщене тіло процедури, яке складається із розділу описів і розділу виконуваних операторів. Всі імена, що описані всередині блока, локалізуються в ньому, тобто вони ніби невидимі із зовні блоку. Таким чином, стосовно операторів, що використовують звернення до блоку, він трактується як "чорна скринька", в якій реалізується той чи інший алгоритм.

Заголовок процедури має вигляд

[{$x+}] procedure<ім'я> [(<список формальних параметрів>)];

function <ім'я> [(<список формальних параметрів>)]:<тип>;

Формальні параметри, якщо вони є, відокремлюються один від одного крапкою з комою. (В операторі виклику фактичні параметри-

аргументи наводяться через кому). Однотипні – розділяються комою і об'єднуються в підписки. В межах блока список формальних параметрів розглядається як своєрідне розширення розділу описів. Їх можна використовувати в будь-яких виразах. Через параметри здійснюється налагодження алгоритма блоку на конкретну задачу.

Щоб запобігти випадковій зміні параметра в тілі процедури він оголошується в розділі const

```
function c(const n:byte):word;
 begin
 c:=n*n;
 end;
 var k:byte;
        begin i:=9;
        writeln(c(i div 3)); readln
        end.
```

При звертанні, як і для параметра-значення, може бути вираз. Вживання кодового слова var в оголошенні формальних параметрів дає параметр-змінну, яку можна використати для повернення результату. Завжди рекомендується не зловживати глобальними зв'язками, а передавати через параметри-змінні:

```
procedure Inc2(var a:integer; b:integer);
begin
a:=a+b
end;
```

Якщо параметр оголошений як параметр-змінна, то при виклику йому має відповідати фактичний параметр в вигляді змінної потрібного типу. Передається сама змінна, а не її копія.

Опис *всіх* формальних параметрів як параметрів-змінних не бажаний з двох причин:

1. Неможливий виклик блоку з фактичними параметрами в вигляді виразів.

2. Головне, в блоці можливе випадкове використання формального параметра для тимчасового зберігання проміжного результату, як параметра циклу і т. п., тобто ненавмисне пошкодження фактичної змінної.

Не рекомендується використовувати параметри-змінні в заголовку функції. Якщо результат її роботи не єдине значення, необхідно декомпозувати алгоритм на декілька блоків чи використати процедуру.

Побічний ефект підпрограми полягає в модифікації стану підпрограми, що її викликала, шляхом присвоювань елементам, відмінним від явно передаваних, фактичних параметрів.

2.15.3. Параметри-масиви і параметри-рядки

Оголошення змінних в списку формальних параметрів (функції чи процедури) нічим не відрізняється від оголошення їх в розділі опису змінних, за винятком однієї суттєвої деталі:

типом будь-якого формального параметра може бути тільки стандартний чи раніш оголошений тип.

Не можна оголосити таку процедуру:

```
procedure S(a: array[1..10] of real);
```

так як в списку формальних параметрів фактично оголошується (неявно) тип-діапазон, що вказує границі індексів масиву.

Звичайно, при передачі елемента немає проблем, але якщо передається весь масив, то необхідно спочатку описати його тип:

```
type
    at = array[1..10] of real;
    procedure S(a: at);
```

Оскільки рядок є своєрідним масивом, його передача в блок здійснюється подібним чином

```
type
    inty = string[15];
    outy = string[30];
        function St(s:inty):outy;
```

В мові Паскаль неможливо використовувати в блоках масиви з межами, що "плавають", зміни індексів. Якщо є розроблена програма, що обробляє матрицю 10×10, то для обробки матриці розміром 9×11 ми змушені перевизначити тип, тобто перекомпілювати всю програму.

Розробники TP включили деякі засоби, що пом'якшують ці недоліки:

1. Спеціальна опція VAR – STRING checking в середовищі TP дозволяє встановлювати режим компіляції, при якому відключається контроль відповідності довжин фактичного і формального параметра-рядка (стан опції *relaxed* вимикає, а *strict* вмикає контроль). Це дозволяє розв'язати проблему передачі блокові рядка довільної довжини.

Приклад з відповідною опції *директивою* компілятора {$v-}:

```
Type styp=string[20];
```

```
Var s: string[6];
        Function Pas(var S: styp): string; begin Pas:=s+'.pas' end;
begin {$v-} s:='myfile'; writeln(pas(S)) end.
```

Більший розмір відсікається до розміру формального параметра. Звичайно, контроль (мовою Паскаль, чи по опції) вмикається тільки при передачі рядка, який оголошено як формальний параметр-змінна.

Для масивів є кілька варіантів розв'язання проблеми з деякими ускладненнями тексту програми. Наприклад, з використанням нетипізованих параметрів-змінних.

2.15.4. Типізовані і нетипізовані параметри-змінні

Якщо параметр оголошений як параметр-змінна, то при виклику йому має відповідати фактичний параметр в вигляді змінної потрібного типу. Якщо параметр визначений як параметр-значення, то перед викликом блока це значення обчислюється, отриманий результат розміщується в тимчасову, допоміжну пам'ять і передається блоку. Якщо ж він визначений як параметр-змінна, то передається сама змінна, а не її копія:

```
const
a: integer=5;   b: integer=7;
{$x+}
var a,b: integer;
function Ink(var a:integer; b:integer):integer;
begin
a:=a+a; b:=b+b; writeln(a:3,b:3)
end;
begin
writeln(a:3,b:3); Ink(a,b); writeln(a:3,b:3); readln
        {5 7 10 14 10 7}
end.
```

Щоб відрізнити параметр-змінну від параметра-значення, перед ним в списку формальних параметрів записується службове слово var. Після формального параметра-змінної, як звичайно, вказується його тип.

На відміну від формального параметра-значення, для якого фактичним параметра може бути будь-який вираз відповідного типу, для формального параметра-змінної фактичним параметром може бути тільки змінна відповідного типу.

Параметр-змінна в розділі var оголошення формальних параметрів в заголовку блока є нетипізованим, якщо не вказано його тип. Відповідний фактичний параметр може бути змінною будь-якого типу.

```
{Процедура міняє місцями в оперативній пам'яті два однотипних об'єкти, в даному випадку – масиви}
var v1, v2: array[1..10] of integer; i: byte; logic: boolean;
    procedure SwapVar(var a,b; size: longint);
    type tp=array[1..MaxInt] of byte;
    var v1: tp absolute a; v2:tp absolute b; i:longint; Tmp:byte;
    begin
    for i:=1 to size do
        begin Tmp:=v1[i];  v1[i]:=v2[i]; v2[i]:=Tmp end
    end;
begin {writeln(SizeOf(boolean));}
for i:=1 to 10 do begin v1[i]:=0;  v2[i]:=i end;
SwapVar(v1,v2,SizeOf(v1));
for i:=1 to 10 do  writeln(v1[i]:3)
end.
```

2.15.5. Принцип використання нетипізованих параметрів-змінних

Якщо параметр визначений як параметр-змінна, то при виклику блока в блок передається сама змінна, а не її копія. Якщо тип формального параметра не вказаний в заголовку блока, то вважається що цей параметр нетипізований, а відповідний йому фактичний параметр може бути змінною довільного типу.

Звичайно такий параметр використовується в випадку, коли тип даних не суттєвий для алгоритму задачі, точніше, контролюється програмістом. Такі ситуації виникають при різних копіюваннях з однієї області пам'яті в іншу. В посднанні з механізмом суміщення в пам'яті нетипізовані параметри-змінні зручно використовувати для передачі процедурі масивів змінних розмірів.

```
{$x+}  {Функцію можна викликати як процедуру}
procedure SuM(var a,b,c; n:word);
var
                                {Пам'ять не виделяється навіть в стеку }
{ Двовимірні масиви використовуються як одновимірні }
```

```
x: array[0..0] of word absolute a;      {Накладання без виділення }
y: array[0..0] of word absolute b;      {пам'яті}
z: array[0..0] of word absolute c;
{ Тут суттєво тільки нижнє значення індексного типу-діапазону }
i, j: integer;
begin
for i:=0 to n-1 do
for j:=0 to n-1 do
z[i*n+j]:=x[i*n+j]+y[i*n+j];
end;
const
a0:array[1..2,1..2] of word=((2,3),(2,3));
b0:array[1..2,1..2] of word=((4,5),(4,5));
var
c0:array[1..2,1..2] of word;
begin
Sum(a0,b0,c0,2);
writeln(c0[1,1],c0[1,2],c0[2,1],c0[2,1])
end.
```

При звертанні до процедури покажчики на масиви a0, b0, c0 розміщуються в стеку і передаються по посиланню. Тому локальні змінні x, y, z процедури SuM, які суміщенні з параметрами a, b, c, є фіктивними змінними, розмір яких в їх оголошенні не впливає на розмір фактично виділеної пам'яті.

Нетипізованими можуть бути тільки параметри-змінні.

2.15.6. Використання покажчиків в параметрах і адресної арифметики

Паскаль – це мова, в якій не передбачено операцій з матрицями, не існує можливостей для їх введення і дії з покажчиками складніші ніж, наприклад, в Сі. Але все ж виразних засобів ТР досить для задовільної роботи з матрицями.

Найбільш універсальний засіб – роботу з покажчиками і використання адресної арифметики продемонструємо на прикладі процедури додавання двох матриць:

```
procedure SumMatr(pa,pb,pc:pointer; n:integer);
var si, i, j: integer;
```

```
type pint=^integer;
function pr(p: pointer): pint;
begin pr:=ptr(seg(p^),ofs(p^)+(i*n+j)*si) end;
begin
si:=SizeOf(integer);
for i:=0 to n-1 do
for j:=0 to n-1 do
pr(pc)^:=pr(pa)^+pr(pb)^
end;
const n=2;
a:array[1..2,1..2] of word=((2,3),(2,3));
b:array[1..2,1..2] of word=((4,5),(4,5));
var
c:array[1..2,1..2] of word;
begin
SumMatr(@a,@b,@c,n);
writeln(c[1,1],c[1,2],c[2,1],c[2,2]);
end.
```

Функція pr повертає покажчик на цілий тип – адресу (i, j)-елементу матриці. Формальний параметр – нетипізований покажчик.

У виклику аргументом – фактичним параметром буде адреса матриці-масиву, отримана за допомогою унарної операції @ взяття адреси і передана через параметри-значення процедури SumMatr.

Адреса (i, j)-елемента обчислюється за сегментом і зміщенням (вбудовані функції seg, ofs) першого елемента матриці і за індексами i, j. Вбудована функція ptr повертає значення типу pointer, сумісне з покажчиком будь-якого типу.

Особливістю функції зі значенням типу покажчик є можливість використання її виклику в лівій частині оператора присвоювання значення змінній, ім'я якої утворюємо приєднанням справа до імені покажчика символа '^'(В данному випадку – до виклику функції pr).

Деяке ускладнення тексту програм – плата за універсальність, за використання адресної арифметики.

2.15.7. Побічна рекурсія. Використання випереджувального опису

Якщо процедура чи функція безпосередньо викликає сама себе, то така рекурсія є простою рекурсією. Але трапляються ситуації, коли процедура чи функція викликає одну або декілька проміжних процедур чи функцій, які в кінці кінців викликають першу (Д. Пойа: B. Pascal, Œuvres, Paris, 1819, v. 3).

При такому взаємному звертанні немає можливості виконати правило мови Паскаль, що всі процедури і функції повинні бути визначені до їх виклику. В цьому випадку використовується *випереджувальний* опис процедур і функцій, що полягає в тому, що: процедура чи функція містить опис тільки свого заголовка, за яким ставиться ім'я FORWARD; опис тексту цієї процедури чи функції розміщується далі, в будь-якому місці розділу опису процедур і функцій, без повторення списку формальних параметрів.

Приклад:

```
program pr;
var a, b: real;
                procedure p1(x:real); forward;
        procedure p2(y: real);
        begin ... p1(a); ... end;
                procedure p1;
                begin ... p2(b); ... end;
begin ... p2(a); ... p1(b); ... end.
```

2.16. Процедурні типи

2.16.1. Константи процедурного типу. Процедурні змінні. Обов'язковість дальньої моделі пам'яті. Оболонки для стандартних процедур

Константа процедурного типу повинна вказувати ідентифікатор процедури чи функції, сумісний за присвоюванням з типом константи. Приклад:

```
type
        Wri = procedure(X: integer);
        ErrorProc = procedure(ErrorCode: Integer);
var
        W: Wri;
```

```
        x, y: integer;
procedure DefaultError(ErrorCode: Integer); far;
begin
        Writeln('Error ', ErrorCode, '.');
end;
const
        ErrorHandler: ErrorProc = DefaultError;
procedure WriteInt(Number: Integer); far;
begin
        write(Number);
end;
begin
        x:=5;
        W:=WriteInt;
        W(x);
end.
```

2.16.2. Параметри-процедури і параметри-функції

Як формальні параметри в мові Паскаль, крім параметрів-значень і параметрів-змінних, допускається використання також імен процедур і функцій.

Параметри-процедури в списку формальних параметрів в авторській версії мови Паскаль вказуються після службового слова procedure. Наприклад

```
procedure PR(i, j: integer; var z: real; procedure P);
```

Параметри-функції в списку формальних параметрів вказуються після службового слова function з вказанням типу функції:

```
procedure PM(i, j: integer; var z:real; function F: real);
```

В TP дещо інакше організоване використання параметрів-процедур і параметрів-функцій. Необхідно попередньо визначити процедурний тип:

```
type
Proc=procedure(T:real);
Func=function(x,y:real):real;
```

Тоді, вказуючи серед формальних параметрів ім'я процедури чи функції, необхідно після двокрапки вказати відповідне ім'я типу.

Приклад:

```
{$F+}
type
    Func=function(x, y: real): real;
function f1(a, b: real): real;
    begin f1:=a+b end;
function f2(c, d: real): real;
    begin f2:=c*d end;
function fun (x,y:real;f:Func):real;
    begin fun:=f(x,y) end;
begin
writeln('+ ',fun(1,2,f1)); writeln('* ',fun(2,3,f2))
end.
```

Отже, при виклику процедури чи функції, що містить в якості формальних параметрів параметри-процедури чи параметри-функції, на їх місце здійснюється підстановка відповідних імен фактичних процедур і функцій. Якщо при цьому процедури і функції, що фігурують як формальні параметр, мають в свою чергу параметри, то ці параметри можуть бути тільки параметрами-значення.

2.16.3. Параметри процедурного типу. Процедурний тип в оголошенні структурного типу

Можна розглядати процедури і функції як об'єкти, які можна присвоїти змінним і які можуть слугувати в якості параметрів. Процедурні типи роблять це можливим. Вони оголошуються як заголовки процедур без імені

```
type
    proc = procedure;
    SwapProc = procedure(var x, y: Integer);
    MathFunc = function(x: real): real;
    DeviceFunc = function(var F:text):Integer;
MaxFunc = function(A,B:real; f : MathFunc): real;
```

Імена параметрів в оголошенні процедурного типу чисто "декоративні" – вони не впливають на сенс оголошення.

ТР не дозволяє оголошувати функції зі значеннями процедурного типу.

Головне призначення – передача фактичними параметрами процедур і функцій в зверненні до інших процедур і функцій.

Наступна програма демонструє використання параметра процедурного типу для виведення трьох таблиць різних арифметичних функцій:

```
program Tables;
type
    Func = function(X, Y: Integer): Integer;
{$F+}
function Add(X, Y: Integer): Integer;
begin
    Add := X + Y;
end;
Function Multiply(X, Y: Integer): Integer;
begin
    Multiply := X * Y;
end;
function Funny(X, Y: Integer): Integer;
begin
    Funny := (X + Y) * (X - Y);
end;
{$F-}
procedure PrintTable(W, H: Integer; Operation: Func);
var
    X, Y: Integer;
begin
    for Y := 1 to H do
    begin
    for X := 1 to H do Write(Operation(X,Y ) : 5);
    Writeln;
    end;
Writeln;
end;
begin
    PrintTable(10, 10, Add);
    PrintTable(10, 10, Multiply);
    PrintTable(10, 10, Funny);
end.
```

Аналогічний приклад мовою Сі:

```
#include <stdio.h>
add(int x, int y) {return x+y;}
mult(int x, int y) {return x*y;}
void tabl(int w, int h, int(*func)(int, int))
{int x, y;
for(y=1; y<=h; y++){
for(x=1; x<=w; x++) printf("%4d ",func(x, y));
printf("\n");     }
}
void main(void){ tabl(2,3,add);}
```

Процедурний тип може бути присутнім в оголошенні структурного типу:

```
type
        GoToProc = procedure(X, Y: Integer);
        ProcList = array[1..10] of GoToProc;
        WindowPtr = ^WindowRec;
        WindowRec = record
                        Next: WindowPtr;
                        Header: String[31];
                        Top, Left, Bottom, Right: integer;
                        SetCursor: GoToProc;
                        end;
var
P: ProcList;
W: WindowPtr;
procedure go(X, Y: Integer); far;
begin
        gotoXY(x, y); writeln(' Pos=',x,y)
end;
begin
        ClrScr;
        P[3]:=go;
        W^.SetCursor:=go;
{Справжні виклики процедур}
```

```
P[3] (1,1);
W^.SetCursor(10,10);
readln
end.
```

2.16.4. Використання процедурних типів в виразах

Використання процедурної змінної в операторі чи у виразі означає виклик процедури чи функції, що зберігається в змінній.

Виняток: коли ТР бачить процедурну змінну в лівій частині оператора присвоювання, він знає, що права частина представляє процедурне значення

```
type
    IntFunc = function: integer;
var
    f: IntFunc;
    N: Integer;
    function ReadInt: Integer; far;  { Дальня модель пам'яті }
    var                              { Атрибут для організації }
    I: Integer;                      { міжсегментних зв'язків –}
    Begin                            { крім зміщення є ще сегмент}
        read(i); Read Int:=i
    end;
begin
    f:=ReadInt; N:= ReadInt: Writeln(N, f)
end.
```

Перший оператор присвоює процедурне значення (адресу) ReadInt процедурній змінній f. Другий оператор викликає ReadInt і присвоює змінній N повернуте цією функцією значення.

Дії розрізняються за типом використаної в лівій частині оператора присвоювання змінної.

Синтаксис умовних виразів.

```
if f = ReadInt then writeln('- дорівнює ');
```

– виклик f і ReadInt і порівняння.

```
if @f = @ReadInt then writeln('- дорівнює ');
```

– порівняння процедурних значень в f і ReadInt.

Оператор адреси @ застерігає компілятор від виклику і перетворює аргумент в покажчик.

@f перетворює процедурну змінну f в змінну нетипізованного покажчика, що містить адресу, а @ReadInt повертає адресу функції ReadInt, після чого можна порівняти два значення покажчиків для визначення, чи посилається f на ReadInt.

Щоб отримати адресу процедурної змінної, а не адресу, що в ній зберігається, потрібно використати подвійний оператор адреси. @P перетворить P в змінну нетипізованного покажчика; @@P поверне адресу змінної P.

В наступному прикладі Д. Рітчі мовою Сі: char (*x1())[]; x1 є функцією, яка значенням вертає покажчик на масив символів. Серед синтаксичних конструкцій *, ( ), [ ] приорітет операції * є найменшим. * – доступ за покажчиком, ( ) – застосування функції, [ ] – доступ до елементів масиву мовою Сі [15, С. 260].

2.17. Перетворення типів

2.17.1. Явне перетворення типів з використанням імені типу. Приведення типів

Існує загальний механізм перетворення типів, згідно якого перетворення досягається застосуванням імені (ідентифікатора) стандартного типу чи типу визначеного користувачем як ідентифікатора функції перетворення до виразу перетворюваного (іншого) типу. Це так зване автовизначене перетворення типів, при якому автоматично, разом з типом визначається і функція перетворення, з тим же ім'ям.

Наприклад, допустимі такі виклики функцій

```
Type  Mytyp=(a, b, c, d);
Var   i: integer; v: Mytyp; p:pointer; e: longint;
begin
v:=Mytyp(2); a:=12345; p:=pointer(longint(e)+$ef)
i:=integer('D');
end.
```

Може відбутись зміна довжини виразу (Останнє присвоювання).

Може також здійснюватись перетворення в лівій частині присвоювання

```
type
      byt=array[1..2] of byte;
      int=array[1..2] of integer;
      rec=record x, y: integer end;
```

```
var
    vb: byt; vin: int; vre: rec;
begin
    byt(vin[1])[2]:=0; {Присвоювання як перетвореному типу}
    int(vre)[1]:=256;
```

В області пам'яті, яку займає змінна деякого типу, розміщується значення виразу іншого типу. Контроль довжини здійснюється для результату перетворення.

Приведення типу здійснюється як *неявне* перетворення типів.

Наприклад, в виразах із цілочисельних і дійсних змінних, при суміщенні в пам'яті даних різних типів (записи з варіантними полями, типізовані покажчики, що вказують на одну адресу. Одна і та ж область пам'яті трактується за змістом як область даних то одного, то іншого типу.

Може бути і явне розміщення різнотипних даних в одній і тій же області пам'яті за допомогою атрибута absolute:

```
var
b: byte absolute $0000:$0055;
w: longint absolute 128:0;
x: real;
y: array[1..3] of integer absolute x; {суміщення по перших байтах}
```

2.17.2. Можливості неявного перетворення типів. Розміщення даних різних типів в одній області. Абсолютні змінні

В мові Паскаль типи даних задаються статично, і, загалом, недопускається їх зміна в процесі виконання програми. Однак Турбо Паскаль допускає в певних межах такі перетворення, які необхідно задавати в явному вигляді.

Є три способи задання перетворення типів: неявні перетворення, використання стандартних функцій і явні перетворення. Перші два визначені в стандартній версії мови Паскаль, а третій в TP.

Спосіб неявного перетворення допускається тільки в операції присвоювання, коли, наприклад, дійсній змінній присвоюється ціле значення.

В другому способі для перетворення типів використовуються такі функції: odd, trunc, round, ord, chr. Правила їх використання розглядалися при вивченні відповідних типів даних, з якими вони оперують.

Явне перетворення типів в TP представляє собою спеціальну конструкцію. Загальний вигляд перетворення типу такий:

< тип > (< змінна >).

Тобто для надання змінній іншого типу необхідно взяти її в дужки, вказати перед дужками ім`я її нового типу.

Розглянемо цей спосіб приведення типу змінної на прикладі. Нехай деяка змінна описана, як Var i: byte; Присвоєння їй значення i:='m' призведе до помилки – невідповідність типів. Таке присвоювання можна здійснити тільки використовуючи конструкцію явного перетворення типів:

char(i):='m'.

Ця конструкція може використовуватись у всіх позиціях, куди входить змінна. Слід враховувати, що розмір змінної (тобто кількість байтів для розміщення змінної) повинен збігатися з розміром типу, ім`я якого вказане перед дужками.

```
type
    ByteRec = record
                Lo, Hi : Byte;
                end;
    WordRec = record
                Lo, Hi: Word;
                end;
PtrRec = record
            Ofs, Seg: Word;
            end;
    BytePtr = ^Byte;
var
    B: Byte; W: Word; L: LongInt; P: Pointer;
```

Для процедурних типів

```
type
    Func = function(X: Integer) : Integer;
var
    F: Func; N: Integer;  Pt: Pointer;
```

Змінні можна описати директивою absolute так, що вони будуть розміщені в пам'яті за певною адресою.

```
CrtMode : Byte absolute $0040 : $0049;
```

Перша константа означає базовий сегмент, а друга визначає зміщення всередині цього сегмента. Обидві константи не виходять за межі

діапазону від $0000 до $FFFF (від 0 до 65535).

```
var
    Str: String[32];
    StrLen: Byte absolute Str;
        {Містить довжину Str (нульовий байт)}
begin
    W:= $1234;
    B:= ByteRec(W).Lo;
    ByteRec(W).Hi:= 0;
writeln(B:7,W:7); readln;
    L := $01234567;
    W := WordRec(L).Lo;
writeln(L:7,W:7); readln;
    B := ByteRec(WordRec(L).Lo).Hi;
writeln(B); readln;
    B := BytePtr(L)^;
writeln(B); readln;
    P := Ptr($40,$49);
    W := PtrRec(P).Seg;
  Inc(PtrRec(P).Ofs,4);
writeln(W:7,Ofs(P^):7,' Значення за адресою ',p^:7); readln;
    ErrorProc(Pt):=ErrorHandler;
    ErrorProc(Pt)(314);
    ErrorHandler(314);
 readln;
 F := Func(P);      { присвоювання процедурного значення із P в F }
 N:=1;
N := F(N);          { виклик функції через F }
writeln(N); readln;
Func(P) := F;       { присвоювання процедурного значення із F в P }
N := Func(P)(N);    { виклик функції через P }
writeln(N); readln;
@F := P;            { присвоїти значення покажчика із P в F }
P := @F;            { присвоїти значення покажчика із F в P }
end.
```

Сентенція: Є речі, про які не варто говорити, але необхідно писати.

3. Розвиток виразних засобів

Дуже давно вхожі в практику викладання концепції багатомовного або, правильніше кажучи, надмовного підходу до навчання програмуванню. Звичайна побудова курсу [10] така, що він охоплює методологічні положення інформатики, використовуючи як певного роду мову специфікацій неформально описані мовні конструкції "високого рівня".

Ілюзія простоти програмування, далі гіркий досвід зіткнення з "порогом складності", за яким відчувалась неможливість контролю створюваних об'єктів або процесів, шлях, яким проходили всі. Паралельне дослідження мов потрібне для розуміння смислу елементів і методів програмування. Програмний контекст різних мов, які мають різну філософію, створює саме поняття, показує ті його сторони, яких не видно із статті цього поняття в описі мови.

В [10] історія інформатики поділяється на предінформатику, протоінформатику, інформатику. З предінформатики нас цікавлять такі події.

В 1623 році професор орієнталістики (сходознавства) в Тюбінгені, любитель астрономії і математики, *Шиккард* писав Кеплеру, що сконструював лічильну машину, яка автоматично виконує додавання і віднімання, множення і ділення [Б. Тарасов, С. 80]. Винахідник помер підчас епідемії чуми. Схеми і креслення, стосовні цієї машини, були знайдені в 1957 році в одній з бібліотек ФРН. Єдиний дослідний екземпляр її згорів в 1624 році. Вчені його не бачили і машина ніяк не вплинула на подальший розвиток механізації обрахунків.

Паскаль – справжній і повністю самостійний винахідник арифметичної машини, що виконувала додавання і віднімання (1642 р.). Її принцип лічильника обертів в наш час використовується в таксометрі (лат. taxo – оцінюю). Паскаль – син дворянина, королівського урядовця, конструював машину для батька і фактично винайшов касовий апарат [2, С. 661].

Ляйбніц в 1671 році вдосконалив результати Паскаля. Його лічильна машина виконувала чотири операції, а також піднесення до степеня і добування квадратного та кубічного коренів. Готфрід Вільгельм Ляйбніц є засновником інформатики (як і значної частини математики). В 1679 році він створив проект машини, яка б працювала в двійковій системі числення, розробив двійкову арифметику.

Leibniz Gottfried Wilhelm народився в Ляйпцігу в сім'ї слов'янина, вихідця з Польщі. Його прізвище Лубенець пізніше трансформувалося в Ляйбніц. Батько Ляйбніца, юрист за освітою, був професором моральної філософії в Ляйпцігському університеті [О.І. Бородін, А.С. Бугай, 1973], [Укр. матем. журнал.- 1967.- т. 19.- №2].

Протоінформатика із працями Тюрінга, Клині, Поста є предметом інших дисциплін і відповідної літератури.

Фундаментальні поняття інформатики українською, англійською і французькою мовами:

комп'ютер (computer, ordinateur);
програмне забезпечення (software, logiciel);
технічне забезпечення (hardware, matériel) [10, С. 34].

Етапи в історії програмного забезпечення:

- перші комп'ютери Ц2 і Ц3 німецького інженера К. Цузе, готові в 1939 і 1941 рр., але знищені в кінці війни;
- ЕНІАК Д. Маучлі, Дж. П. Еккерта (Пенсільванський університет, 1943-1946 рр.);
- перше використання індексного регістра (Манчестерський університет, 1947-1948 рр.);
- транзистор в 1948 р. Дж.Бардін, У.Браттейн, В.Шоклі (Белл Тел Лаб)
- МЭСМ С.О. Лебедєва в Києві (1951 р.);
- виготовлення ІБМ 701 з 1953 р.;
- і для неї створення Фортрану в 1954 р. (Дж. Бекус);
- перший опис Алголу Дж. Бекусом 1959 р., перша формальна специфікація синтаксису мови програмування;
- опис Коболу в 1959 р., в результаті співробітництва американського уряду, користувачів і конструкторів (Common Business Oriented Language);
- перша система з віртуальною пам'яттю (Атлас, Манчестерський університет, 1962 р.)

Малі алгоритмічні мови Паскаль і Сі схожі і мають спільного пращура – мову Алгол 68. Якщо трохи довше службове ім'я, то і довший код – помилкова думка, яка їх розрізняє. Початкова версія мови визначена А. ван Вейнгаарденом та ін. (Майу Б. Дж., Пек Дж. Є. Л., Костер К. Г.), співавтором "Переглянутого повідомлення" є Ліндсі Ч. Х. [8, С. 223].

Книга [2] побудована на прикладах мовами Алгол 68 і Паскаль. Авторами мови Алгол 68 передбачені спеціальні засоби впорядкованого створення національних варіантів мови, які б зберігали смисл програм і полегшували їх транслітерацію. Так автором посібника був зроблений *український варіант* прикладів програм із [8, 11].

Алгол 68 завершував історичний період розвитку програмування, відкритий такими класичними мовами, як Фортран, Кобол, Ліспі і Алгол 60. В остаточну форму мови ймовірно були внесені зміни. Передбачалось ввести в мову символ **"ніщо"** і розглядати як віртуальний описувач, який "не виробляє значення". Буде **проц ніщо**, а не **проц** [8, С. 7].

Представимо ті виразні засоби цієї мови, які вже є у її нащадків або можуть з'явитись.

3. 1. Альтернативні представлення символів

Українська мова має яскраво виражені риси *синтетичної* мови, в якій зв'язок між членами речення здійснюється насамперед шляхом узгодження роду, числа і відмінка, меншою мірою – з допомогою службових слів, при мінімальній ролі порядку слів в реченні.

Англійська ж мова має ще більш виражені риси *аналітичної* мови – повна відсутність відмінкових і родових закінчень чи артиклів (є лише ознаки статі у деяких назвах людей і тварин), зв'язок слів у реченні на основі їх порядку і з допомогою службових слів.

Все це сильно зближує структуру англійської мови з мовами програмування взагалі і з Алголом 68 зокрема. Більш близькі також до мов програмування правила словотворення і скорочення в англійській мові. В силу всіх цих причин перекладачі навели самі програми, як звичайно, в російській версії Алголу 68, зберігши, однак, вивідні тексти англійською мовою (коментар перекладачів [8, С. 395]).

В мові Паскаль вибрано представлення символів, яке близьке до природного. В мові Сі використано коротші представлення.

Варіанти Алголу [8, С. 366]:

Альтернативні представлення символів

все	)	10	\
із	[	**мінус**	-:=
початок	(	**діл**	/:=
кінець	)	∨	**або**
ніл	°	≠ ¬=	**нрв** (~=)

тояк	**|:**	**÷**	**цдл**
:=	**..=**	**приц**	**÷:=** #присвоювання ціле#
від	**→**	**приза**	**÷::=** #присвоювання залиш#
;	**.,**	**/:=**	**діл**
"	**лап**	**×:=**	**мнж**
прагмат	**пр**	**множ :=**	**(*:=)**
заув	**за #**	**↑ ** **	**вгору ^ (××)**
<	**мнш**	**нигр**	**└ entier**
якщо	**(**	**вегр**	**┌**
то	**|**	**=**	**рів**
інакше	**|**	**іі ⊥**	**(!)**
іняк	**|:**	**@**	**з**
див	**) все ощкя**	# або **прагмат** [8, С. 172]	

Тих, що в дужках, представлень немає в "Повідомленні про мову", але їх легко вивести.

В дещо іншій формі нижче подаються альтернативні представлення символів, зроблені пізніше [11, С. 216]:

Контекст	*Альтернативи*
зображення дійсного	e 10 \
зображення символьного або рядка	_ пропуск
Умовне речення	**якщо** ... **то** ... **іняк** ... **інакше** ... **все** (... | ... |: ... | ...)
Варіантне або порівнювальне речення	**виб** ... **в** ... **абвиб** ... **або** ... **див** (... | ... |: ... | ...)
Вирізка	**з** @
Зауваження	**заув** ... **заув** *ѣ*... *ѣ k* ... *k* # ... #
Відношення однойменності	:=: **є**
Перехід	**на** (нічого)

пусто (empty) [8, С. 385].

Змішування різнорідних представлень в одній парі парних символів не допускається [8, С. 386].

В Алголі 68 прагмат зовні схожий з зауваженнями, межами його є два символи **прагм**. Який текст може бути в середині прагмата, визначає

тільки реалізація; інформація, що він передає, в самій мові не визначена. Наприклад,

прагм тільки трансляція **прагм**

вказує, можливо, що програма не повинна виконуватись після трансляції. **прагм** символи служать для тимчасового виходу за межі мови [8, С. 112].

Виділені жирним шрифтом символи називаються індикантами. В рукописних і машинописних (старих) програмах індиканти звичайно підкреслюються. Єдині символи, однак не можна розбивати пропусками (пробілами) і границями рядків [11, С. 14].

Деякі представлення символів доводилося змінювати, щоб вони стали придатними для конкретних трансляторів і машин. Так, якщо не допускаються малі букви, то *ціл* може виглядати як ЦІЛ, а ціл як 'ЦІЛ', якщо, звичайно, апостроф використовується в мові тільки як обмежувач для індикантів. Справа самого програміста з'ясувати, які варіанти застосовуються в його обчислювальній системі, і слідувати їм.

Специфіка реалізації

Залапковані (закавыченные) слова використовуються для представлення певного символу:

'початок' 'початок **початок** початок ПОЧАТОК

В тексті слова через риску (дефіс) можуть представляти залапковані слова і відповідно певні символи в одинарних лапках (або в апострофах) [8, С. 15].

Від однієї реалізації до іншої змінюється також обстановка. Цей термін стосується того, наприклад, які найбільші значення можуть зберігатись в машині, а також деяких деталей мови, точне визначення яких покладено на реалізацію.

Приблизна відповідність термінів

Алголу 68	**і інших мов**
блок (range)	блок, область дії (block, scope)
вид (mode)	тип даних (date, type)
замкнуте речення (close clause)	складений оператор, блок (compound statement, block)
ім'я (name)	комірка, змінна (location, variable)
індикант (indicant)	службове слово, ключове слово (key word)
масив (multiple value)	масив (array)
основа (unit)	вираз, оператор (expression, statement)
послідовне речення (series)	послідовність операторів

	(statement sequence)
приведення (coercion)	перетворення типу (type conversion)
присвоювання (assignation)	оператор присвоювання (assignment statement)
формула (formula)	вираз (expression)
фраза (phrase)	оператор (statement)
обмін (transput)	введення-виведення (input/output, i/o)
виконання (elaboration)	виконання, обчислення (execution, evaluation)

Мисленe виконання програми ("ручне") концептуально корисніше [8, С. 111].

3.2. Створення нових видів

Можна визначити новий 'індикатор виду' {індекс, індикатор} з допомогою опису

вид стовпчик = [1:n] **дійсн**;

вид матриця = [1:n,1:n] **дійсн**;

Тоді в контексті цих описів-виду отримуємо

стовпчик a; Еквівалентно [1:n] **дійсн** a;

матриця A; Еквівалентно [1:n,1:n] **дійсн** A;

Не можна написати, не допускається

вид матриця = [1:n] **стовпчик**;

Мультизначення з рухомими межами.

Межі, які можна змінювати після опису мультизначення, називаються рухомими (**рух**)

[m: n **рух**] **дійсн** fa; Значення *n* можна потім змінити

[m **рух**: n **рух**] **дійсн** faf;

Особливо корисні у випадку рядків, вбудованих в стандартний опис мови. Рядок, тобто величина виду *рядки* є мультилітера з рухомою верхньою межею

вид рядки = [1: 0 **рух**] **літ**;

рядки S;

Рядок спочатку порожній. При присвоюванні верхня-границя отримує значення [8, С. 28].

Значення умовного речення.

Умовне-речення є основою (основним реченням):

Якщо p **то** 31415927·10-7 **інакше** 27182818·10-7 **все**

після виконання має дійсне значення свого речення-типу-то чи свого речення-типу-інакше в залежності від значення своєї умови. Якщо умовне-речення виробляє змінну, то її можна використати в отримувачі (зліва від символа-присвоїти):

Якщо p **то** x **інакше** y **все** :=3.1415927

або

(p|x|y):=3.1415927

Послідовне-речення може мати значення, хоча не є основою. Його значенням є значення його завершальної (останньої по порядку) основи.

Приклад програми

почат

ціл чис:=0, дод:=0, від:=0, **дійсн** абсум:=0, x;

знову: абсум **плюс якщо** чис +:=1; чит(x); x>0;

то дод +:=1; +x

інакше від +:=1; -x;

все ;

якщо x ≠ 0 **то** *знову* **інакше** *готово* **все**;

готово: друк((чис, дод, від, абсум))

кінець

Паралельні речення. Схожі на сумісні (які без слова **пар**)

пар (x:=1; y:=2; z:=3) Речення має смисл, коли є 3 процесори (стільки в цьому реченні є операторів). Більш реальний приклад використовує переваги синхронізації, і її контроль. Для взаємодії паралельних процесів встановлюються семафори. Є спеціальний вид, призначений для цих цілей [8, С. 194]:

сем ціле1, ціле2, ціле3

В семафорах є імена цілих, доступ до яких дозволяється з допомогою спеціальних унарних операцій.

Для *порівняння імен*, а імена в Алголі 68 є значеннями, є знаки-тотожності:

:=: (або є) і :≠: (або нє)

Покажчики (змінні імена) – ідентифікатори, які є іменами імен.

Покажчики задаються описами [8, С. 51]

ім'я тип ідентифікатор;

наприклад: **ім'я дійсн** xx;

або в строгій формі

ім'я імені тип ідентифікатор = **лок ім'я** тип;

наприклад: **ім'я імені дійсн** xx = **лок ім'я дійсн**;

В машинно-орієнтованому програмуванні (зокрема в мовах асемблерів) ім'я імені відомо як покажчик.

Імена як значення, які мають ідентифікатори

тип ідентификатор;

наприклад: **дійсн** y;

або в строгій формі

ім'я тип ідентифікатор = **лок** тип;

наприклад: **ім'я дійсн** xx = **лок дійсн**;

Побічна адресація (в певній комірці пам'яті знаходиться адреса іншого значення)

Приклад: xx:=y;

Щоб присвоїти значення імені, яке присвоєне покажчику, використовується ядро [8, С. 55].

Ядро – термін виду, в прикладі – **ім'я** тип.

Приведення:

(ім'я тип: покажчик):= ідентифікатор;

Наприклад: (**ім'я дійсн** : xx):= y;

Якщо в xx є адреса x, то це присвоювання тотожне x:=y;

В бібліотеках-вступах використання *купи* схоже з використанням стека [8, С. 337].

Ланки: генератори [8, С. 253].

'Генератори' застосовуються для того, щоб ділянки пам'яті, на яких можна розміщувати значення, були доступні для користувача. Вони виробляють імена цих ділянок. Генератор складається із фактичного-описувача, перед яким стоїть **лок**, або **глоб**, або *нічого* (і в цьому випадку розуміється *глоб*). Локальні генератори особливо корисні при створенні трикутних і інших незвичайних мультизначень:

(П14) **початок**

[1: 0 **рух**] **ім'я** [ ] **дійсн** трикутник;

вид масив = [1: 0 рух] дійсн; ¢ для економії чорнил ¢

трикутник:= (**лок масив**:= 1,

лок масив:= (1, 2),
лок масив:= (1, 2, 3),
лок масив:= (1, 2, 3, 4));
для і **до** 4 **цикл** *друк*(трикутник[i][i]) ¢ друкує діагональ ¢
кінець

Поза блоком П14, як трикутник, так і іменовані ним масиви щезнуть.

Зауваження, коментарі в дужках із пари символів ¢.

Глобальні генератори розміщують в купі.

Процедурні види рівноправні зі всіма іншими видами [11, С. 122]. Можна описати масиви підпрограм (що задаються, наприклад, записом масивів з текстами підпрограм). При звертанні до окремої процедури в цьому випадку використовується вирізка із масивів процедур, яка супроводжується, якщо потрібно, фактичними параметрами.

Інший приклад. Процедура, що складає таблицю значень деякої цілочислової функції f для аргументів, які приймають значення між двома даними цілими *i* і *j*:

проц *таб* = (**проц** (**ціл**) **ціл** *f*, **ціл** *i*, *j*) **пуст**:
для k **від** *i* до *j* **цк** (*k*, f(*k*), *нов рядок*) **кц**

У виклику може бути ідентифікатор виду **проц(ціл) ціл** або текст підпрограми, наприклад:

таб((**ціл** x) **ціл**: 2*x, -5, 5)

Процедури можуть виробляти мультизначення [8, С. 216], а також структурні значення [8, С. 346, С. 199].

Можна визначити *процедурну змінну* **проц (ціл) ціл** vp. Можна задати початкове її значення раніш описаним ідентифікатором процедури або текстом підпрограми. Таким чином можна написати

проц vp := (**ціл** n) **ціл**: (**ціл** *i*, *j* := 0;
до *n* **цк** *чит*(); **плюспр** *i* **кц**; *j*)

3. 3. Визначення операцій

Підпрограми можна зв'язати не тільки з ідентифікаторами процедур, а й з символами операцій. Операції представляються *індікантами*, наприклад **абс** чи **плюспр**, або спеціальними символами, наприклад + або ⇐ [11, С. 124].

Визначення власних операцій їх описом.

Опис операції подібний до опису тотожності для процедури з

заміною **проц** на **оп** [11, С. 125].

Так опис

```
оп макс = ([ ] ціл a) ціл: (
    ціл m:= a [нигр a];
    для i від нигр a + j до вегр a цк
      якщо a[i]>m то m:=a[i] все кц;
    m)
```

визначає унарну операцію **макс**, призначену для знаходження найбільшого елемента не порожнього одновимірного цілочислового масиву. Після цього формула

макс ([] **ціл** a = (-5, 10, 50, 7, 0); a)

видасть значення 50.

Робота з частинами масивів [11, С. 74].

Підмасиви, які йменуються *вирізками*, є самостійними масивами і мають одиницю як нижню межу, а верхні індекси відповідно перераховуються. В кінці відрізка можна помістити символи L або @L, де L – цілочисловий вираз, який дає бажане значення нижньої межі.

Приклад. Якщо *a* описане [2:20] **ціл** a, то вирізки нижче зліва йменують масиви з межами, вказаними справа.

a[5:10 *з* 5]	≡	[5:10]
a[: 5 *з* 40]	≡	[40:43]
a[15: *з* -10]	≡	[-10:-5]
a[*з* 1]	≡	[1:19]

Якщо один символ представляє кілька бінарних операцій, то всі вони мають один і той же пріоритет [8, С. 125]. Наступна версія для = теж буде мати пріоритет 4:

```
оп = = ([ ] ціл a, b) лог:
    якщо ціл m = вегр a[з 1], = вегр b[з 1]; m = n
    то лог f := істина;
        для і до m поки f цк
            якщо a[з 1] [i] ≠b [з 1] [i]
            то f := хиби все
        кц;
    інакше хиби все;
```

З допомогою опису пріоритету можна встановити пріоритет нового символу бінарної операції або перевизначити пріоритет вже існуючих

бінарних операцій [11, С. 126].

Опис пріоритету

пріо символ = цифра,

де цифра в межах від 1 до 9.

В Алголі 68 можна визначити новий вид

вид рац = ст (ціл *чис, знам*)

і вжити в ряді застосувань як альтернативу для **дійсн** [11, С. 147].

Унарний мінус

оп - = (**рац** a) **рац** : (-чис **із** a, знам **із** a)

Добуток двох раціональних чисел p/q і s/t є ps/qt. Результат треба спростити, розділивши чисельник і знаменник на найбільший спільний дільник.

проц нсд = (**ціл** a, b) **ціл**:

(b = 0 | **абс** a | **нсд** (b, a **мод** b))

Тепер операція множення запишеться як [11, С. 148]

оп *= (**рац** a, b) **рац**: (

ціл чс = чис **із** a * чис **із** b,

зн = знам **із** a * знам **із** b:

ціл нд = нсд (чс, зн);

(чс ÷ нд, зн ÷ нд))

Оскільки запис структури не може бути операндом, доцільно визначити *операцію* для *побудови* раціонального числа із пари цілих чисел

пріо = 9;

оп р = (**ціл** a, b) **рац** : (

ціл чс:= a; **зн**:= b;

виб знак зн +2 ¢ – оператор варіанту ¢

в (чс := -чс; зн := -зн),

(*друк* (*нов рядок*, "незаконне ÷число")); стоп)

або пропуск

див

ціл нд = нсд(чс, зн);

(чс÷нд, зн÷нд))

Тоді зручніше всього визначити *ділення* як

оп / = (**рац** a, b) **рац**:

a * знам **із** b р чис **із** b,

а даний раніше опис *множення* замінити на

оп * = (**рац** a, b) **рац** :

(чис **із** a * чис **із** b) **р** (знам **із** a * знам **із** b)

Коли потрібно для видачі в замкнутому-реченні кілька завершальних основ, можна використати завершувач. *Завершувач* – це символ **вихід** (або .); основа, що передує йому, об'являється (за означенням) завершальною основою замкнутого речення [8, С. 30].

Завершувач можна розглядати як замінювач символу-закрити –).

Приклад:

```
p:= (дійси x, y, z;
      до n цикл ( чит ((x, y)); z:=(x+y)/2 – sqrt (x × y);
            якщо z>1 то стоп все);
      хиби вихід
      стоп: друк ((x, y)) ;
      істина)
```

Тіло процедури може бути із основою – замкнутим-реченням

```
      проц  ¢ ніщо – тобто не виробляє значення [8, С. 37] ¢
            (дійси чит
            друк(z:=(x+y)/2 – sqrt (x × y))
            );
```

Можна записувати тіло процедури в дужках символів **початок** і **кінець** (замість "(" і ")").

Набір операцій над *раціональними числами* може бути важливий там, де суттєву роль відіграють такі операції, як додавання, віднімання, множення, ділення і підстановки *степеневих рядів* [8, С. 335]. Відсічений степеневий ряд (тобто многочлен) можна описати, наприклад, як

вид стряд = [0: -1 **рух**] **рац**;

Можна визначити операції

оп += (**стряд** P, Q) **стряд**; § підпрограма додавання §

. . .

Комплексну функцію, визначену такими степеневими рядами, можна описати таким чином:

```
      проц ФУН = ( стряд P, компл z) компл;
            (ціл n = вегр P; компл значення:= ді P[n];
            для і від n-1 через -1 до 0 цикл
                  значення:= значення z + ді P[i]; значення);
```

де операція

(16) **оп ді = (рац r) дійсн чг/зг**; [8, С. 333]

Операції над комплексними операндами [8, С. 67]

вид компл = структ (**дійсн** re, im);

оп ⊥ = (**дійсн** a, b) **компл** : (a, b);

оп re = (**компл** a) **дійсн** : re **від** a;

оп im = (**компл** a) **дійсн** : im **від** a;

оп абс = (**компл** a) **дійсн** : sqrt (re a↑2+ im a↑2);

оп спряж = (**компл** a) **компл** : **re** a ⊥ - **im** a;

оп = = (**компл** a, b) **лог** : **re** a = **re** b ∧ **im** a = **im** b;

оп ≠ = (**компл** a, b) **лог** : ¬(a = b);

оп + = (**компл** a) **компл** : a;

оп - = (**компл** a) **компл** : -**re** a ⊥ - **im** a;

оп + = (**компл** a, b) **компл** : (**re** a + **re** b) ⊥ (**im** a + **im** b);

оп - = (**компл** a, b) **компл** : (**re** a - **re** b) ⊥ (**im** a - **im** b);

оп × = (**компл** a, b) **компл** : (**re** a × **re** b - **im** a × **im** b)

⊥ (**re** a × **im** b + **im** a × **re** b);

оп / = (**компл** a, b) **компл** : (**дійсн** d = **re** (b × **спряж** b);

компл n = a × **спряж** b;

(**re** n/d) ⊥ (**im** n/d));

оп ↑ = (**компл** a, **ціл** b) **компл** :

(**компл** p: = 1;

до абс b **цикл** p:=p × a;

(b≥0 | p | 1/p));

Унарна операція **іі** = (або ⊥) (англійською – **i**)

оп іі = (**ціл** a) **компл**: (0, a);

Залишивши англійське позначення x ⊥ y (або x **i** y) можна записати x + **i**y.

Операції, об'єднані з присвоюванням [8, С. 68].

оп плюс = (**ім'я дійсн** a, **дійсн** b) **ім'я дійсн**: a:=a + b; ¢ або + := ¢

оп мінус = (**ім'я дійсн** a, **дійсн** b) **ім'я дійсн**: a:=a - b; ¢ або - := ¢

оп множ = (**ім'я дійсн** a, **дійсн** b) **ім'я дійсн**: a:=a × b; ¢ або × := ¢

Перший формальний параметр має бути ім'ям, оскільки ми бажаємо присвоїти йому значення і те значення, що виробляється, теж має бути

описане як ім'я.

В мові ці операції описані для всіх арифметичних операндів (видів **ціл, дійсн** і **компл**).

Операції над рядками.

```
вид рядки = [1: 0 рух] літ;
оп < = (рядки a, b) лог:
    (ціл m = вегр a, n = вегр b;
        p = (m<n | m | n), ціл i := 1; лог c;
        якщо p < 1
        то    n ≥ 1
        інакше знов: якщо c:=a[i] = b[i] то якщо
        p ≥ (i плюс 1)
        то знов все
        якщо c то m < n інакше a[i] < b[i] все все
        все
оп + = (рядки a, b) рядки:
    (ціл m = вегр a, n = вегр b; [1: m+n] літ c;
    c[1: m]:= a; [m+1, m+n]:= b;
оп плюс = (ім'я рядки a, рядки b) ім'я рядки: a:= a + b;
```

Приведені в бібліотечному вступі [8] набори операцій та приклади (над векторами і матрицями в E_n [8, С. 336], аналітичного диференціювання [8, С. 356] та інші) можуть бути корисні в різноманітних застосуваннях, в математиці, фізиці, хімії і астрономії. Набір операцій дає програмісту можливість писати прозору і безпосередню алгоритмічну прозу на його власному професійному жаргоні [8, С. 329].

3.4. Структурний тип

Структури і покажчики в ПЛ/1. Структура ПЛ/1 подібна до "запису" Алголу W. Алгол W задуманий Віртом і Хоаром в 1966 р., використовувався в університетах [10, С. 66]. Є, однак, суттєва відмінність: об'ява запису Алголу W ніколи не визначає об'єкт, а визначає тільки "шаблон" для класу об'єктів, що можуть створюватись в будь-якій кількості підчас виконання програми. В ПЛ/1 все залежить від *атрибуту розподілу пам'яті*, відповідного об'яві структури [10, С. 259].

В об'яві структури кожному рівню присвоюється номер, наприклад

```
declare 1 personne automatic,
/* або static, або controlled або based(p) */
2 nom character(16),
2 age binary fixed,
2 adresse,
3 ville character(10),
3 rue character(20),
3 numero character(3);
```

Атрибут *automatic* можна не вказувати. При цьому атрибуті визначається один об'єкт, а не тип, але цей об'єкт створюється і знов ініціалізується при кожному виконанні блоку. При "статичному" розподілі теж створюється об'єкт але він зберігається від одного звертання до блоку чи процедури, яким він належить, до наступного.

При "керованому" (*controlled*) способі розподілу пам'яті визначається тип. Створення об'єкту здійснюється оператором *allocate*, а ліквідується оператором *free*. *Allocate* створює нову версію, яка маскує попередню. *Free* знищує останню створену версію і забезпечує доступ до попередньої, якщо вона є. Наприклад [10, С. 261]:

```
allocate personne;
free personne;
```

При "базованому" способі, нарешті, ситуація схожа з положенням об'єктів типу *record* в Алголі W. Як і в попередньому випадку об'ява визначає лише модель даних. Однак в цьому випадку дані доступні через відповідний покажчик. Якщо в Алголі W тип reference(ім'я запису) вказує тільки на запис, то покажчик типу pointer ПЛ/1 може вказувати на будь-які дані

```
declare p pointer;
```

Результатом інструкції allocate personne є створення "версії" структури personne, на яку направлений покажчик, зв'язаний з нею об'явою. Ця версія може бути пойменована в програмі з допомогою позначення

```
p -> personne.
```

Доступ до поля структури здійснюється з допомогою уточнення імені

```
p -> personne.adresse.rue
```

В Алголі W немає ієрархічних рівнів. Є тільки один рівень структурування і може бути написано: nom(personne) [10, С. 253].

Можна зіставити *паралельно оператори* [10, С. 267]:

Алгол W	**ПЛ/1**
record A(integer B;	dcl 1 A based(Z),
real C);	2 B fixed bin,
reference (A) Z;	2 C float bin;
:	:
Z:=A;	allocate A;
:	:
B(Z):=1;	Z->B=1;
:	:
Z:=NULL;	Z=NULL;

В Фортрані 77 спочатку структур не було. З'явились вони в четвертій версії (Фортран 90). *Моделювання* структур можливе з допомогою оператора *equivalence*. В Фортрані обчислення адреси об'єкту можна виконати з допомогою стандартних функцій *loc*, *locnear*, *locfar*. Значення цих функцій еквівалентне даним типу покажчик в Сі [3, С. 128].

Приклад:

```
real var, var1
integer*2 x
integer*4 y
x=locnear(var)
y=locfar(var1)
```

В мові Сі опис аналогічних адрес можна виконати з допомогою покажчиків таким чином:

```
float near *x;
float far *y;
```

3.5. Зв'язок програм різними мовами

В ТР є вбудований Асемблер, тобто асемблерівський текст вміщуємо в паскалівську оболонку:

```
var
 i,n:integer; ar:array[1..10] of integer;
procedure sort(n:integer;var a);
assembler;
label m1,m2,m3;
asm
```

```
{ Сортування масиву цілих додатних чисел. Метод бульбашок }
        push    bp              {Змінився покажчик стека на +2}
        mov     bp,sp
        push    si
        lds     bx,[bp+6]       { Занесення адреси масиву в ds:bx}
        mov     dx,[bp+10]      { Занесення кількості елементів в dx }
        dec     dx              { Допоміжна область для n – два слова }
m1:     xor     si,si
        mov     cx,dx
m2:     mov     ax,[bx+si]
        cmp     [bx+si+2],ax    { Порівняння (n+1)-го елемента з n-м }
        jge     m3
        xchg    ax,[bx+si+2]    { Обмін (n+1)-го елемента з n-м }
        xchg    ax,[bx+si]
m3:     inc     si
        inc     si
        loop    m2              { Перехід до вибору наступного елемента }
        dec     dx
        cmp     dx,1
        jne     m1
        pop     si              { Вихід із процедури і відновлення стека }
        mov     sp,bp
        pop     bp              { ret – тут не потрібне! }
        end;

begin
n:=10;
for i:=1 to n do ar[i]:=random(99);
for i:=1 to n do write(ar[i]:4); writeln; readln;
sort(n,ar);
for i:=1 to n do write(ar[i]:4); writeln; readln;
end.
```

Між формальними параметрами, які є іменами підпрограм Фортран 77 і покажчиками Сі немає повної відповідності. Однак використовуючи функцію *loc*, можна організувати передачу фактичного параметра, що є іменем підпрограми, із Фортрану 77 в Сі.

Приклад.

```
с Програма мовою Фортран 77
interface to subroutine cifor [c] (x)
integer*4 x
end
external procf                    с підпрограма Фортрану 77
integer*4 x                       subroutine procf
x=loc(procf)                      ...
call cifor(x)                     end
:
end
/* Функція Сі */
void cifor(arg)
void (fortran *arg) (void);
{... arg(); ...}
```

В [3, С. 138] інтерфейс Фортрану з програмою мовою Асемблер показано на практично такому ж прикладі, як наведений вище для вбудованого в Турбо Паскаль Асемблеру. В інтерфейсі вказано [*value*]. Це означає, що параметр передається чи приймається за значенням.

Завдання атрибутів С чи PASCAL приводить до передачі параметрів за значенням. Атрибут REFERENCE дозволяє встановити такі угоди, щоб передавались адреси параметрів, а не їх значення [3, С. 122]. Після того як інтерфейс з програмою встановлений, до неї можна звертатись як до звичайної програми Фортрану.

Приклад.

```
Interface to real*8 function integ[C] (A, B)
Real*4 A
Real*8 B [REFERENCE]
end
Interface to subroutine midl [PASCAL] (M, N)
Integer*2 M [near, REFERENCE]
Integer*4 N [near, REFERENCE]
End
Program test
Real*8 integ, sum
```

```
Integer*2 I
Integer*4 J
Real*4 X
Real*8 Y
...
sum=integ(X, Y)
...
call midl(i, j)
...
end
```

В мові Ада програма може бути відкомпільована як єдине ціле, тобто як один компонент, або передана компілятору як послідовність компонентів, що компілюються. Їх може бути декілька. Кажуть, що компільовані компоненти належать бібліотеці програм [16, С. 105 – 106].

компіляція ::= {компонент, що компілюється}

компонент, що компілюється::=

[обмеження_видимості] [**separate**] тіло_компонента

Роздільно компільований компонент, який використовує який-небудь роздільно компільований модуль, повинен містити ім'я цього модуля в своєму списку видимості.

Головною програмою мовою Ада (в загально прийнятому смислі) є один із компільованих компонентів, що представляє тіло підпрограми. Засоби задання і виконання цієї програми виходять за рамки визначення мови Ада.

Машинні команди можна включити в програму мовою Ада, якщо звернутись до відкритої процедури, тіло якої містить лише кодовані оператори [16, С. 138]:

кодований_оператор::=кваліфікований_вираз;

Кожна машинна команда виглядає як значення запису, типу, що визначає відповідну команду. Опис таких комбінованих типів звичайно є в передвизначеному пакеті на кожній машині. Будь-яка процедура, що містить кодований оператор, має складатись тільки з цих операторів. Реалізація має передбачати залежні від машини вказівки, що задають угоди відносно регістрів і викликів.

Приклад:

```
M: MASK;
    procedure SET_MASK is
        use INSTRUCTION_360;
        pragma INLINE;
    begin
        SI_FORMAT(CODE=>SSM, B=>M'BASE, D=>M'DISP);
    -- M'BASE і M'DISP – передвизначені атрибути,
    -- задані в реалізації
    end;
```

3.6. Друга універсальна мова

Назва "Programming Language number 1" або "Мова програмування №1", скорочено **ПЛ/1**, досить добре відбиває претензії творців цієї мови: мова всього на всього про те, щоб замінити Фортран, Кобол, Алгол і всі інші мови новою (спочатку було запропоновано скорочення NPL), яка об'єднала б усі їх переваги і не мала б їх обмежень.

Мова була створена в 1963-1964 рр., в результаті зусиль ІБМ і асоціацій користувачів ІБМ "Share" і "Guide". ПЛ/1 намагається об'єднати можливості числової обробки і дій з даними, якими оперують в задачах керування виробництвом [10, С. 9]. Мова займала солідну частину "ринку", особливо в економічних розрахунках.

Мова радше різношерста ніж універсальна. Трудомістка, доля в довгостроковій перспективі залишалась, проте, невизначеною. Враховуючи великий об'єм мови майже всі програмісти самі обмежували її деякою підмножиною.

Особливості виразних засобів. В ПЛ/1 є тип **bit** – бітовий рядок. Логічне значення обробляється у формі бітового рядка **bit** довжиною 1. Тип **picture** (шаблон) дозволяє обробляти формати введення-виведення, а **file** – роботу з файлами. Є два особливі типи **label** (мітка) і **pointer** (покажчик).

Атрибут основи числового типу вказує, чи потрібно значення розглядати в двійковій системі числення (**binary**), чи в десятковій (**decimal**). С два атрибути виду – **real** і **complex**. Атрибути представлення – з фіксованою комою (**fixed**) чи з мантисою і порядком (**float**). Є також атрибут розрядності. Для чисел з фіксованою комою – два числа [**p, q**], що

вказують загальну кількість цифр і кількість цифр справа від коми (і для числа з плаваючою комою – загальну кількість).

Алгоритмічна універсальна мова програмування Ада об'єднала практично всі основні можливості ряду найбільш досконалих мов (Алгол-68, ПЛ/1, Паскаль та інші). Вона враховує всі тенденції, що намітились в подальшому розвитку мов програмування.

Мову можна створити на науковій основі, майже "вивести" її з аксіом дедуктивними методами. На цьому принципі реалізована частина Алголу-68, занадто складна. Мова Паскаль виявилась настільки невеликою, витонченою і майже універсальною, що виникла впевненість в близькій методичній побудові універсальної мови. Кобол – мова обробки ділової інформації, Фортран – наукових розрахунків, Ада – для програмування найрізноманітніших систем управління і, взагалі, будь-яких складних програмних систем. Як і Алгол-68, остання – універсальна мова і є завершенням певного етапу в розвитку мов програмування.

Три питання в сучасній проблематиці алгоритмічних мов: визначення типів даних і абстрактні дані, механізми і смисл розпаралелювання і, нарешті, модульне програмування. У всіх трьох областях є прихильники строгих (їх можна назвати статичними) рішень, що передбачують статичний контроль правильності використання механізмів мови на стадії трансляції, і прихильники вільних "динамічних" рішень, де контроль або відкладається на час виконання програми, або взагалі відкидається. Коли розробляли мову Ада, по першому і останньому питаннях притримувались максимально строгих статичних принципів.

Коли вимоги недостатньо послідовні і точні, або з протиріччями, досить ймовірно, що строгість призведе до абсурдності [16, С. 8].

Пакет-модуль – це головний механізм для визначення логічно зв'язаних понять. Пакети можна, наприклад, використати для введення спільної групи типів і даних, групи родинних процедур чи внутрішніх типів і відповідних операцій. Деякі частини модуля можуть бути приховані від користувача, що забезпечує доступ тільки до логічних властивостей, заданих в пакеті.

Кожний компонент програми складається з двох частин: описової частини, яка визначає логічні поняття, використані в даній програмі, і послідовності операторів, які визначають виконувані нею дії.

Описова частина встановлює відповідність між іменами і описуваними поняттями. В цій частині вводяться також імена і параметри

інших підпрограм, задач-модулів і пакетів-модулів, що використовуються в даному компоненті програми.

Задачі-модулі аналогічні пакетам-модулям з тією лише різницею, що вони можуть виконуватись паралельно. Задачі можна реалізувати з допомогою кількох процесорів, або чергуючи виконання на одному процесорі.

Оператори описують дії, що потрібно виконати. Оператор присвоювання вказує, що поточне значення змінної мусить бути замінене новим значенням. Оператор виклику підпрограми вимагає її виконання після встановлення відповідності між вказаними у виклику фактичними параметрами і формальними параметрами даної підпрограми.

Умовний оператор і оператор варіанту дозволяють виділити із послідовності операторів який-небудь один. Виділення проводиться на основі значення умови чи виразу, з якого починаються згадувані оператори. Оператор циклу є основним ітераційним механізмом мови. Він містить послідовність операторів, які слід виконати до моменту, визначеного специфікацією повторення, або до виконання оператора виходу [16, С. 13].

Складені (структурні) типи дозволяють визначити об'єкти, що складаються зі зв'язаних між собою компонентів. В російській мові слово "структура" вимагає прямого доповнення, тому в [16] прийнято терміни відповідно "комбінований" і (для масивів) "регулярний". Масиви є об'єктами з індексованими компонентами одного типу, а записи – об'єкти з пойменованими компонентами, що, можливо, належать до різних типів.

Використовуючи варіанти в комбінованому типі, можна визначити різні структури записів. До абсурду може привести положення, що складається з поняттям типу для запису, бо введення типу для запису з варіантами структури фактично є ще одним механізмом введення "автономних" типів. Тому в мові починає гіпертрофовано розростатись апарат дискримінантів (міток варіантів).

В цьому випадку логічно було б взагалі усунути механізм варіантів в записах. Мова спроститься, проте стане менш гнучкою. В розглянутому варіанті мови механізм дискримінантів складний, проте особливої гнучкості не додає (зауваження редактора [16] А.Д. Подшивалова).

Джерела. Спрощення розробки сталось завдяки наявності ряду вдалих проектів розвитку (приблизно з тією ж метою) мови Паскаль – мов Euclid, Lis, Mesa, Modula і Sue. Багато концепцій і синтаксичних

конструкцій ввійшли як складові в мову Ада. Меншою мірою вплинули вже на той час існуючі мови Алгол-68 і Симула і останні проекти Alphard і Clu.

Контекстно-вільний синтаксис мови Ада описується з допомогою модифікованих форм Бекуса. При чому:

a) слова з малих букв – синтаксичні поняття;
b) жирним позначаються зарезервовані слова;
c) в квадратних дужках розміщуються елементи, які можуть бути відсутні;
d) елементи, що повторюються, беруться в фігурні дужки;
e) курсивом до назви синтаксичної конструкції може додаватись деяка синтаксична інформація. Такі слова еквівалентні самому синтаксичному поняттю [16, С. 15].

Приклад [16, С. 60]:

Умовний_оператор ::=
    if умова **then**
        Послідовність_операторів
    {**elsif** умова **then**
        Послідовність_операторів}
    [**else**
        Послідовність_операторів]

Умова ::=
    Вираз {**and then** вираз}
    | вираз {**or else** вираз}

Будь-який оператор може бути помічений ідентифікатором, взятим в подвійні кутові дужки, наприклад, <<HERE>>. Мітки використовуються в операторах виходу(exit) і переходу(goto). Як і ::=, | – символ метамови.

Блоки [16, С. 74]

Блоком є послідовність операторів, якій може передувати описова частина:

Блок ::=
    [**declare** описова_частина]
    begin послідовність_операторів
    [**exception** {реакція_на_виняткову_ситуацію}]
    end [ідентифікатор]

Приклад поміченого блоку:

```
<<SWAP>>
    declare
        TEMP: INTEGER;
    begin
        TEMP:=V; V:=U; U:=TEMP;
    end SWAP;
```

Блоки дають можливість без використання апарата процедур робити описи локальними по відношенню до деякої послідовності операторів. Блок можна розглядати як безіменну процедуру, неявно викликану в тому місці, де знаходиться її визначення (тобто блок) [16, С. 66].

(Побічний ефект, тобто присвоєння значень нелокальним змінним, для *функції* не дозволяється).

Суміщення підпрограм – одне й те ж позначення підпрограми в декількох специфікаціях різних підпрограм [16, С. 72].

Суміщення операцій [16, С. 73]

Позначення функції з допомогою рядка символів використовується для розширення смислу відповідної операції. Суміщення операцій в усьому аналогічне суміщенню підпрограм, за винятком того, що рядок символів повинен позначати одну з існуючих в мові операцій. Операцію відношення /= (нерівність) явно суміщувати не можна, оскільки будь-яке суміщення операції = неявно означає суміщення і операції /=.

Рекомендується при суміщенні зберігати загально прийнятний математичний смисл. Приклад:

```
function "*"(X, Y: MATRIX) return MATRIX
```

Приклад *підпрограми-функції* [16, С. 71]:

```
function DOT_PRODUCT (X, Y: VECTOR) return REAL is
    SUM: REAL:= 0.0;
begin
        assert X'FIRST=Y'FIRST;
        assert X'LAST=Y'LAST;
        for I in X'FIRST .. X'LAST loop
            SUM:= SUM+X(I)*Y(I);
        end loop
        return SUM;
end DOT_PRODUCT;
```

Оператор контролю *assert* приводить до обчислення умови і, якщо

вона хибна, до створення ситуації ASSERT_ERROR.

Оператор циклу вказує, що послідовність операторів в деякому основному циклі потрібно виконати по черзі нуль або більше разів. Виконання закінчується або при вичерпанні специфікації повторення, або при виконанні в основному циклі оператора виходу:

оператор_циклу ::= [специфікація_повторення] основний_цикл

основний_цикл ::=

loop

послідовність_операторів

end loop [ідентифікатор];

специфікація_повторення ::=

for параметр_цикла **in** [reverse] дискретний діапазон

| **while** умова

параметр_цикла ::= ідентифікатор

В умові наперед визначені атрибути – мінімальне і максимальне значення типу чи підтипу. Опис для типу масиву, використаного вище:

type VECTOR **is array**(INTEGER) **of** REAL;

Коментар в мові Ада починається з подвійного мінуса **--**.

Дійсні наперед визначені типи short_float, float, long_float [16, С. 30]. При обмеженні *відносної* точності використовуються плаваючі типи з плаваючою точністю і при обмеженні абсолютної точності – фіксовані (з фіксованою крапкою):

визначення_дійсного_типу ::=

обмеження_точності ::=

digits простий_вираз [обмеження_діапазону]

delta простий_вираз [обмеження_діапазону]

Приклади:

type COEF **is digits** 10 **range** –1.0..1.0;

type VOLT **is delta** 0.125 **range** 0.0..255.0;

type FRAC **is delta** 0.00001 **range** 0.0..1.0;

type MASS **is new** FLOAT **digits** 7 **range** 0.0..1E30;

Кваліфікація виразів [16, С. 51] в мові Ада використовується для явного вказання типу виразу, для включення його в даний підтип і, якщо випадок не підпадає під перераховані, для приведення значення до іншого типу:

кваліфікований_вираз ::=

марка_типу(вираз) | марка_типу складене_значення.

Одна константа може відноситись до різних типів. Коли тип не визначається із контексту, потрібно її кваліфікувати, як нижче у виклику звертання до суміщеної підпрограми, в заданні діапазону параметра циклу:

```
type MASKING_CODE is (FIX, DEC, EXP, SIGNIF);
type INSTR_CODE is (FIX, CLA, DEC, TNZ, SUB);
PRINT(MASKING_CODE (DEC));          -- DEC типу MASKING_CODE
PRINT(INSTR_CODE(DEC));             -- DEC типу INSTR_CODE
I in INSTR_CODE(FIX) .. INSTR_CODE(DEC) -- необхідна кваліфікація
I in INSTR_CODE range FIX .. DEC    -- кваліфікація вилучається із
                                       контексту
```

Операції належності **in** і **not in** перевіряють чи належить дане значення будь-якого типу заданому діапазону, підтипу чи обмеженню.

Перетворення типу [16, С. 52]

Приклад перетворення похідного типу:

```
type A_FORM is new B_FORM;
X:= A_FORM;
Y:= B_FORM;
X:= A_FORM(Y);
```

Перша описова частина в **специфікації модуля** називається *видимою частиною* [16, С. 76]. Елементи, описані в ній, можна зробити доступними ("видимими" з іншого компонента програми з допомогою декларації використання в цьому компоненті):

Декларація_використання ::= **use** ім'я модуля{, ім'я модуля }

Якщо в видимій частині є специфікація підпрограми чи іншого модуля, то тіла специфікованих об'єктів мають входити в описову частину тіла даного модуля. Елементи описані в тілі є недоступними зовні.

Приклад пакета [16, С. 78]:

```
package RATIONAL_NUMBERS is
type RATIONAL is
        record
                NUMERATOR: INTEGER;
                DENOMINATOR: INTEGER range 1 .. INTEGER'LAST;
        end record;
function "=" (X, Y: RATIONAL) return BOOLEAN;
function "+" (X, Y: RATIONAL) return RATIONAL;
function "*" (X, Y: RATIONAL) return RATIONAL;
```

Функція "=" пере визначає наперед визначену рівність

```
package body RATIONAL_NUMBERS is
procedure SAME_DENOMINATOR (X,Y: in out RATIONAL) is
    begin
        -- приведення X, Y до спільного знаменника
    end;
function "=" (X, Y: RATIONAL) return BOOLEAN is
U,V: RATIONAL;
begin
    U:=X;
    V:=Y;
    SAME_DENOMINATOR (U, V);
    return U.NUMERATOR=V.NUMERATOR;
end "=";
function "+" (X, Y: RATIONAL) return RATIONAL is
... end "+";
function "*" (X, Y: RATIONAL) return RATIONAL is
... end "*";
end RATIONAL_NUMBERS;
```

Будь-який *параметр підпрограми* може належати до одного з трьох *класів*: **in, out, in out**. Останній є універсальним, розглядається як локальна змінна, допускає можливість використання значення відповідного фактичного параметра і присвоювання йому нового значення (значення-результат) [16, С. 68].

Діапазон кожного індексу масиву має бути відомий в момент обробки опису цього масиву (або при його розміщенні, коли розглядаються типи посилань). Вирази, що визначають діапазон, *не обов'язково статичні*, можуть залежати від обчислюваних значень. Такі масиви називаються **динамічними** [16, С. 34].

Статичним називається вираз, значення якого не залежать від будь-яких динамічно обчислюваних значень змінних. Статичний вираз обчислюється підчас компіляції [16, С. 53].

опис_підтипу ::= **subtype** ідентифікатор **is** марка_типу[обмеження]

визначення похідного типу [16, С. 26]

визначення_похідного_типу ::= **new** марка_типу[обмеження]

Типи посилань. Генеровані об'єкти не зустрічаються в явних описах, тому з допомогою імен їх позначати неможливо. Для доступу до них

використовуються значення посилань, які вертає генератор [16, С. 39].

визначення_типу_посилань ::= **access** тип

Ім'я **null** не позначує ніякого об'єкта і належить до кожного з типів посилань.

Допускаються *рекурсивні типи і взаємно залежні типи* [16, С. 41].

Приклад взаємно залежних типів посилань:

```
type CAR;  -- попередній неповний опис типу
type PERSON is access
    record
        NAME: STRING(1..20);
        AGE: INTEGER range 0..130;
        SPOUSE: PERSON;
        VEHICLE: CAR;
    end record;
type CAR is access  -- повний опис
    record
        NUMB: INTEGER;
        OWNER: PERSON;
    end record;
MY_CAR, YOUR_CAR: CAR;
HPERS: PERSON := null;
```

Генератор створює новий об'єкт і значення посилання, що вказує на цей об'єкт:

генератор ::= **new** кваліфікований_вираз

Приклад: DOUBLE := **new** PERSON(ME.**all**);.

Складені значення визначають значення масивів і записів [16, С. 34].

Приклад:

```
B: TABLE := (5, 4, 8, 1, others => 20);
C: TABLE := (2 | 4 | 10 => 1, others => 0);
```

-- В останньому операторі 2, 4, 10 – варіанти індексів, 1 – значення відповідних елементів.

```
type PERIPHERAL is
    record
        STATUS: (OPEN, CLOSED);
        UNIT    : constant (PRINT, DISK, DRUM);
        -- дискримінант
    case UNIT of
```

```
            when PRINT=>
                LINE_COUNT: INTEGER range 1.. PAGE_SIZE;
            when others =>
                CYLIND: CYLINDR_IND;
                TRACK : TRACK_NUM;
        end case;
        end record;
```

Затримана ініціалізація для константи UNIT [16, С. 24].

Наперед визначене оточення

Наперед визначені ідентифікатори мови зібрані в пакеті STANDARD, який неявно передається всім програмам. Типи і інші поняття, що включені в пакет STANDARD, навмисне даються в мінімальному об'ємі, оскільки решту пакетів, наприклад комплексну арифметику, можна виразити засобами самої мови. Звичайно при реалізації такі пакети можна включати і в стандартне оточення [16, С. 166].

Також можна вибрати найбільш ефективний для даної машини спосіб реалізації наперед визначених функцій, операцій і процедур. Тому тіло відповідного пакета (STANDARD) не наводиться.

Не всі наперед визначені поняття можна повністю описати з допомогою понять самої мови. Наприклад, *переліковий тип* BOOLEAN, хоч і можна визначити, задавши константи FALSE і TRUE, проте його взаємозв'язок з *умовами* засобами мови виразити не можливо. {Умови, умовляти}. Через це наведений текст пакета не дає повністю семантику понять, що визначаються.

З допомогою вказівки [16, С. 91]

pragma ENVIRONMENT (ім'я_модуля {, ім'я_модуля};

в передвизначене оточення можуть бути включені і інші модулі. Ідентифікатори в видимих частинах цих модулів вважаються описаними в програмі на найвищому зовнішньому рівні і обмеження видимості до них не стосуються.

3.7. Робота з масивами. Функціональна мова АПЛ

В ПЛ/1 в деяких випадках можна виключити явне написання циклів. Так, для масиву, об'явленого table(10, 30), і вектора vect(10) оператор

```
vect = table(*, 1)
```

переписує перший стовпчик масиву в вектор.

В мові ПЛ/1 є ще можливість передати транслятору роботу з написання деяких циклів. Мова йде про використання псевдоіндексів вигляду iSUB [10, С. 343].

Нехай A і B – двовимірні масиви. Можна присвоїти масиву B транспонований масив A (тобто рядки B є стовбчиками A):

```
declare a(10, 10) ...,
        b(10, 10) ...,
        c(10, 10) ... defined a(2SUB, 1SUB)
...
b=c;
```

"SUB" є скороченням від "subscript" ("індекс").

Створена транслятором програма, коротша за числом операторів і за часом виконання, ніж програма, що створена із відповідних явних циклів. Важливо, що необережне використання цих способів може призвести до небажаних результатів, причини яких важко виявити.

Мова APL пропонує, значно більш методично, ніж ПЛ/1, можливість працювати з масивами цілком без використання циклів [10, С. 344].

Обробка масивів засобами поширених мов програмування здійснювалась переважно покомпонентно. Однак бажано виконувати операції над цілими масивами, і довільними підмасивами, чи змінювати структуру масиву. Саме ці принципи обробки даних покладені в основу мови АПЛ.

Програми АПЛ приблизно в 5–10 разів коротші ніж ПЛ/1. Є можливості для створення і переструктурування масивів даних, а також визначення і, за необхідності, перевизначення типу і структури змінної під час виконання програми.

Мова є ідеальним засобом для організації діалогу з комп'ютером. АПЛ входила до стандартного забезпечення IBM/360 і 370 та інших і широко на них використовувалася.

Початкова версія АПЛ (APL – A Programming Language) {Array} була запропонована К.Е. Айверсоном (IBM, США) в 1962 році.

Всі операції мови мають *рівні пріоритети* і виконуються справа наліво. Програма – це послідовність нумерованих рядків-виразів. Основними результатами виконання рядка є присвоювання і/або вказівки на безумовний, умовний чи альтернативний переходи. Переходи зображуються боковими стрілками.

Реалізація на комп'ютері початкової версії АПЛ надзвичайно складна. В кінці 60-х були розроблені лінійні АПЛ-нотації, а операції були поширені на багатовимірні масиви. Був також розроблений спеціальний термінал, клавіатура якого містить більше число знаків алфавіту мови.

АПЛ є однією із самих підходящих мов програмування для реалізації на векторних і матричних комп'ютерах. Стиль програмування мовою АПЛ найбільш близький до *стилю функціонального програмування* [17, С. 7].

Мова АПЛ розроблялася як діалогова, тому має ряд засобів, розрахованих на інтерактивну обробку, наприклад, для задання термінового виконання виразів. Всі визначені в мові конструкції представляються посередництвом фіксованої сукупності символів.

В мові немає службових слів. Основна конструкція – вираз. Скаляри і вектори є первинними даними. Вони можуть бути числового або символьного типу.

Тип і структури даних визначаються із їх зовнішнього вигляду чи семантикою функцій, що їх породжують. В АПЛ відсутні статичні засоби для опису ідентифікаторів. В рамках однієї програми в різні моменти її виконання один і той же ідентифікатор може бути іменем змінної або іменем підпрограми. Змінні утворюються підчас роботи програми.

Тип змінної може бути неодноразово змінений при виконанні програми, тобто зв'язок між ідентифікаторами і даними встановлюється або перевизначається автоматично.

Основними функціями є арифметичні операції, операції порівняння, елементарні математичні функції, присвоювання, створення багатовимірного масиву, виведення на термінал, деякі узагальнені операції над масивами з лінійної алгебри та інші. Звертання до основних функцій має інфіксну форму. Всі основні операції мають рівні пріоритети.

Функції, що визначаються, суть підпрограми. Структура звертання до них аналогічна структурі звертання до основних функцій.

З допомогою виразів будуються самостійні одиниці мови – оператори. Розрізняють два види операторів: виконувані і визначальні. Перші є однорядковими виразами, що виконуються безпосередньо після їх введення. Після виконання текст такого оператора стає недоступним, тобто втрачається. Результат виконуваного оператора виводиться на екран, або присвоюється змінній.

Визначальний оператор забезпечує можливість опису нової функції, що визначається. При її опису вказуються її характеристики: вид, ім'я,

аргументи (один чи два), локальні змінні. Тіло задається послідовністю рядків-операторів, кожен з яких є виразом, або оператором розгалуження, або коментар.

Оператор розгалуження використовується тільки всередині функції, що визначається, і слугує для опису альтернатив і циклічного повторення обчислень.

В результаті виконання визначального оператора обчислювальна система отримує інформацію про створену функцію. Звертання до функції може бути з будь-якого виразу. Допускаються *рекурсивні* звертання до функцій, що визначаються.

Оскільки мова АПЛ інтерактивна, то припускається, що після виконання кожного оператора програми, керування передається користувачеві {MathCAD, Math Lab}.

3.7.1. Деякі базові функції

Функція структури ρF, де ρ – символ функції, F – будь-яке дозволене в АПЛ значення [17, С. 17].

1. **Структура:** ρF – обчислює вектор вимірності аргумента.

Приклади: структури ρ '' ρ 1.5 0 -1 -2

0 4

Якщо X – змінна, значенням якої є матриця символів

ABCD

EFGH, то ρX

I J K L 3 4 (Результат на екран виводиться під виразом).

Вектором вимірності змінної X є числовий вектор 3 4. Для визначення числа вимірів значення необхідно застосувати функцію структури два рази ρρF, що дасть 2.

2. **Перебудова:** VρF – призначена для створення масивів. Лівий аргумент – вектор вимірностей створюваного масиву, правий містить елементи, з яких формується масив.

Приклад: Вираз 5 ρ 1 2 3 дає результат: 1 2 3 1 2.

3. **Розгортка:** ,F – перетворює аргумент в вектор.

Приклад: ,6.5 дає 6.5, де ρ ,6.5 результат 1.

Якщо аргумент представляє багатовимірний масив, то результатом функції є вектор, побудований з компонентів масиву, вибраних в лексикографічному порядку (зміни індексів).

4. **Конкатенація:** G , F служить для формування нового значення

шляхом конкатенації лівого і правого аргументів. Аргументи повинні бути одного і того ж типу.

Приклад: 'SOL','AR' результат SOLAR. Виконана конкатенація двох символьних векторів.

5. **Генератор індексів:** ιS призначений для побудови числового вектора, який представляє собою послідовність з перших S чисел натурального ряду. Приклад: ι7 – 1 2 3 4 5 6 7.

6. **Редукція** (reduction): ⊕/F – складна функція, яка застосовується до компонентів векторів або багатовимірних масивів. Коли аргумент є числовим вектором, а ⊕ – символом арифметичної функції, то виконання редукції полягає в застосуванні функції ⊕ до компонентів вектора справа наліво. Приклади: +/2 4 3 дає 9; ÷/ 2 4 3 результат 1.5.

Редукція, той же символ що й стиснення – /, але замість логічного вектора фігурує символ операції. Так, +/A означає суму компонентів, а /A – визначає найбільший компонент.

7. **Присвоювання:** <змінна> ←A. Змінна – це ім'я (ідентифікатор), надане деякому значенню. A – довільний вираз.

Приклади:

A1 ← 3.14. Змінній A1 присвоюється значення скаляра 3.14. Вектором вимірності змінної ρA1 є порожній вектор @.

A1 ← 'A←A+C'. Перевизначається змінна A1. Їй присвоюється значення вектора з п'яти компонентів. Попереднє значення A1 втрачається.

```
A2 ← 2 3 ρ 1.5
A2
1.5 1.5 1.5
1.5 1.5 1.5
      A4 ← ρA4 ← 'A+B'
      A4
3
```

3.7.2. Змінні

Нехай P ← 2 3 5 7. Змінна з індексами йменує значення, яке є окремим компонентом чи підмасивом змінної, що індексується.

```
P[2]          P[ 1 2 3 4]  - еквівалентно P[], оскільки порожній
 3              2 3 5 7     індекс замінюється виразом ι4.
```

Тобто, P[] ≡ P[⍳4] ≡ P[1 2 3 4]. Символ метамови ≡ використовується для вказання еквівалентності виразів мови АПЛ.

Змінна P[1 1 1 1] вказує на новий одновимірний масив з п'яти компонентів, кожен з яких рівний першому компоненту змінної P.

Нехай E – змінна, значенням якої є матриця чисел

11 12 13 14

21 22 23 24

31 32 33 34.

Змінні E[2;] і E[2; 1 2 3 4] вказують на один і той же підмасив змінної E (одновимірний числовий масив з чотирьох компонентів, оскільки E[2;] ≡ E[2; ⍳4] ≡ E[2; 1 2 3 4]).

E[;] ≡ E[⍳3; ⍳4] – змінна вказує на двовимірний масив.

Змінна E[; 4 3 2 1] вказує на нову матрицю зі зворотним порядком елементів в рядках.

Індекси змінної з індексами можуть бути багатовимірними масивами.

Приклад: (2 3 5 7) [2 3⍴⍳4] Перебудова вектора ⍳4 в вимірність 2 3.

Результат: 2 3 5 Потім індексація масиву, заданого в

7 2 3 круглих дужках

Індексація за зростанням: ⍋V. Будує вектор із індексів компонентів вектора RA (правого аргументу). Індекси впорядковані таким чином, що вказують на компоненти RA в порядку їх зростання. Аргумент функції має числовий тип. Таким чином, результатом виразу RA [⍋RA] є вектор, який складається із тих же компонентів, що і RA, але впорядкованих за зростанням [17, С. 46]. Приклади:

⍋2 0 –1 0 0 –20

3 2 4 5 1

2 0 –1 0 0 –20 [⍋2 0 –1 0 0 –20]

-20 –1 0 0 0 2

(*Індексація за спаданням*: ⍒V)

Вибірка: V↑F – служить для вибірки підмасиву із RA. Компоненти LA (лівого аргументу) вказують ті індекси вимірів RA, які використовуються при вибірці; V – будь-яке ціле число або вектор цілих чисел. Результат типу RA.

P← 2 3 5 7

-3↑P Вибираються в результат | LA (| – абсолютне значе-

```
3 5 7          ння) останніх компонентів. Результат є вектором.
6↑P            При (|LA>ρ,RA)∧(LA>0) додаткові компоненти
2 3 5 7 0 0    результату R прирівнюються нулю.
```

Формула для випадку (ρ,RA ≥ |LA) ∧ (LA<0):

R ← (,RA) [((ρ,RA) + LA) + ι | LA]. Другий доданок в дужках – від'ємний. Індексний масив починається не з одиниці, (ρ,RA) + LA).

Викреслювання: V⍀F. Служить для вибірки підмасиву із RA, викреслюванням деяких його компонентів. Наприклад, якщо LA<0, то беруться перші, без останніх |LA компонентів [17, С. 52]:

R ← (,RA) [ι (ρ,RA + |LA)].

Приклади функцій, що визначаються [17, С. 84].

```
∇ SUM V; SP                    ∇ R← SUM1 V; SP; U
[1] VSP← (0 < ,V) / ,V         [1] VSP← (U← 0 < ,V) / ,V
[2] SP← + /VSP                 [2] SP← + /VSP
[3] R← SP ÷ + / 0 < ,V         [3] R← SP ÷ + /U
[4] ∇                          [4] ∇
```

Два приклади. В *лівому* наведена визначувана функція з іменем SUM без явного результату, з одним формальним аргументом V і локальною змінною SP. В тілі функції використовуються нелокальні змінні VSP і R.

Вважаємо V будь-яким числовим не порожнім значенням. Перший оператор обчислює вектор VSP із додатних компонентів масиву V. Другий – суму компонентів VSP, третій – середнє арифметичне компонентів VSP.

В *правому* прикладі – функція з явним результатом SUM1. Вона містить локальні змінні R, V, SP і U. Змінна U використовується для збереження проміжних результатів, щоб не повторювати вибірку (0 < ,V).

Мова діалогова і позначення коментарів не має.

Сортування числового масиву в спадному порядку.

```
∇ SORT F
[1] → (0 = ρρF) / 4          Якщо – числовий скаляр, то на рядок 4
[2] → (0 ≠ ×/F) BEGIN        Масив не порожній – на мітку BEGIN
[3] → 0 = ρ□ ← 'ARGUMENT ERROR'   Все-таки порожній
[4] BEGIN: S ← (⍒F) ρ(,F) [⍒,F]   Вектор ,F впорядковується і
[5] ∇                        перебудовується в структуру F
```

Функція SORT без явного результату з одним формальним аргументом F використовує нелокальну змінну S і мітку BEGIN.

⍞ ← F – виведення.

Компактний запис першого алгоритму [17, С. 85]:

∇ R ← SUM2 V; U

[1] R← (+ / U / ,V) ÷ + /U ← 0 < ,V

[2] ∇

Використання:

R←10 VSP ← 15

Виклик функції:

SUM 2 –1 –1 3 1 0

Результат: VSP R

2 3 1 2

Стиснення (compressing): U/F – призначене для вибірки підмасиву із масиву RA. Які елементи RA мають додаватись в результуючий масив, вказує LA. Якщо (ρρRA) ≥ 1, то LA може бути логічним скаляром або вектором. В цьому випадку R (результат) є масивом з тим же числом вимірів, що й RA [17, С. 54]. Смисл операції пояснюється прикладом:

A ←2 1 5 6 4, B ←0 1 1 0 1.

Результат B/A буде 1 5 4

Розширення (expansion). Операція позначається символом \ .

A ← 2 1 5, B ← 1 0 0 1 0 1

Результат операції B\A є 2 0 0 1 0 5

A ← 'abcde', B ← 1 0 1 0 1 1 0 1.

Результат операції B\A є 'a b cd e'

Розширення визначення скалярної унарної функції на масиви полягає в тому, що функція виконується окремо над кожним компонентом масиву. Якщо обидва аргументи скалярної бінарної функції – масиви, то функція визначена лише тоді, коли збігаються вектори вимірностей, тобто (ρRA) = (ρLA). В цьому випадку виконання покомпонентне [17, С. 38].

Якщо один аргумент скаляр, він розглядається як фіктивний масив з вимірністю іншого аргументу і однаковими значеннями.

Оператор розгалуження має дві форми [17, С. 82]:

→ <мітка > і → <вираз>

Мітка – номер рядка. Відповідна форма – перехід на мітку. Якщо вираз – порожній вектор, то порядок виконання операторів не змінюється. Якщо значення виразу нульове, то функція завершує роботу. Коли ціле

додатне, тоді передача керування на оператор з таким номером рядка. Для позначення логічних "хибність" і "істина" служать 0 і 1 [17, С. 12].

Обчислення і друк простих чисел (ніладична функція, без явного результату) [17. С. 90]:

```
∇ PP; T                                  Довжина послідовності в
[1] P← 1 ⋄ T ← 1                         нелокальній змінній END
[2] PT: → (ρP ≥ END) / PT1               Істина – перехід на мітку
[3] ADD: → ( ∨ / 0 = P | T ← T + 2 / ADD "або" по індексу P
[4] P ← P, T                             конкатенація
[5] → PT
[6] PT1: P                               PT, ADD, PT1 – мітки
[7] ∇                          [17, С. 41] G | F – різниця, R = G mod F.
```

Книга [5] призначена бути навчальним посібником і посібником для самостійного вивчення наслідків, спричинених комп'ютерами для теорії ймовірностей і математичної статистики. В ній є програми мовою АПЛ з результатами роботи. Автори підкреслюють, що більшість алгоритмів в курсі написані мовою АПЛ, яка є найбільш підходящою, поширеною і доступною. Сукупність програм в курсі не може інтерпретуватись як бібліотека.

В додатку перекладача (С.М. Єрмаков) наведені короткі відомості про мову АПЛ. Говориться, що мова АПЛ/360 є універсальною мовою високого рівня, [5, С. 186], що застосовується для роботи в системі з розподілом часу в діалоговому режимі.

3.8. Передача інформації між програмними модулями

В Алголі W *процедура* визначається **об'явою процедури**, яка синтаксично відіграє ту ж роль, що і звичайні об'яви (змінних, масивів) і з'являється на початку блоку після, але перед першим оператором.

Заголовок процедури [10, С. 169]

procedure ім'я_проц (об'ява форм_параметрів)

Об'ява форм_параметрів дозволяє уточнити ім'я, тип і спосіб передачі, який може бути **value** (аргумент), **result** (результат), **value result** (модифікований параметр) або взагалі не вказано. Спосіб вказується після типу перед ім'ям.

(**comment**: так пишуть зауваження, коментар;)

Підпрограми в ПЛ/1 об'являються у вигляді "блоків-процедур" з міткою, що є ім'ям процедури. Слово **procedure** супроводжується списком параметрів, якщо вони є; тип процедури і типи параметрів об'являються в тілі, якщо це потрібно. Якщо підпрограма має тип "вираз" {Функція}, то тип її результату вказується в заголовку процедури атрибутом returns(), наприклад,

```
minimum: procedure (x, y) returns (binary fixed);
        if x<y then return(x);
        else return(y);
end minimum;
```

Тип результату є real binary fixed(15,0). Можна опустити returns і за замовчуванням отримаємо той же результат. Проте використовувати цю можливість рекомендується ще менше ніж в Фортрані, де підпрограми явно позначаються як **subroutine** або **function**.

В способи передачі інформації між програмними модулями такі мови як Паскаль або Алгол 68 не додали нічого нового [10, С. 175].

З трьох мов тільки Алгол W дозволяє вказати, що формальний параметр повинен бути переданий значенням. Для цього використовується покажчик способу передачі value [10, С. 178].

В ПЛ/1 спосіб передачі можна вказати в момент виклику. Якщо другорядні атрибути формальних і фактичних параметрів різні, то здійснюється передача значенням. Передача значенням – важливий засіб підвищення надійності програм.

Передача результату здійснюється в момент повернення.

В Алголі W можуть бути вирази-блоки, умовні вирази, які вертають значення [10, С. 182].

Фрагмент процедури-значення, значенням якої є останнє значення, отримане в її тілі

```
...
if B ¬= 0 then
        begin
        X1 = -C/B;
        1
        end
else   if C=0 then –1 else 0
```

Значення-результат відрізняється від передачі за "адресою" у

випадку, коли для однієї змінної є і передача за адресою і спільне використання [10, С. 186].

В ПЛ/1 потрібно взяти змінну в дужки, щоб реалізувати ефект передачі за значенням:

call P(X) – виклик за адресою X,

call P((X)) – виклик зі значенням змінної .

Передачі іменем, яка введена в Алголі 60 і є в Алголі W, відповідає справжня текстуальна підстановка фактичного аргументу замість формального. Цей неефективний (тільки з порівняння трьох мов) спосіб був популярний на час появи Алголу 60. Майже всі випадки його використання можуть бути зведені до передачі за адресою або передачі підпрограми як параметра.

```
begin comment: [10, С. 190]
    comment: обчислення максимуму будь-якого цілого масиву;
    integer array a( 0:: 27);
    integer array b( -2::5, 1::10);
    integer array c( 0::4, 0::4, 0::4, 0::4, 0::4);
    integer maxa, maxb, maxc;
    integer i, j, k, l, m;
    real procedure maxmat(integer el, inf, sup, ind);
    begin integer max;
    max:=-maxinteger; comment: вбудована змінна;
    ind:=inf;
    while ind<=sup do
        begin
        if el>max then max:=el;
        ind:=ind+1
        end;
    max
    end maxmat;
    ...
    ініціалізація матриць a, b, c ...
    ...
    maxa:= maxmat(a(i), 0, 27, i);
    maxb:= maxmat(maxmat(b(I, j), 1, 10, j), -2, 5, i);
    maxc:= maxmat(
            maxmat( ,
```

```
                maxmat(
                    maxmat(
                        maxmat(c(I, j, k, l, m,), 0,4, m),
                    0, 4, l),
                0, 4, k),
            0, 4, j),
        0, 4, I);
    write("Максимуми трьох матриць:", maxa, maxb, maxc)
    end.
```

Макрооператор – це наказ, адресований програмі перетворення тексту, дія якої логічно відтворює дію транслятора (навіть якщо фізично ця програма об'єднана з транслятором [10, С. 191].

В Фортрані є примітивна форма макросу – арифметичні функції.

Сімдесяті роки були роками розквіту препроцесорів, що перекладали в Фортран тексти програм (в 1976 році їх було більше 70-и офіційно врахованих [10, С. 193]. Потім, коли є помилки, створені системою тексти потрібно вичитувати. Такі тексти не дуже придатні для читання.

Приклади програм в одній із книг Кернігана з інструментального програмного забезпечення були написані мовою Ратфор, яка є структурованим діалектом мови Фортран, реалізованим як препроцесор [11, С. 235], а в наступній книзі перекладені мовою Паскаль.

Приклад препроцесорної обробки мовою ПЛ/1(стиль збережено):

```
/* 5.07.78 ЗАПАД ЭВМ КОМПЛЕКС */
BIR: PROC OPTIONS(MAIN);
DCL Y CHAR(200) INIT((40)'CARTA');
      BY BIT(100) INIT((50)'01'B);
DCL T(8) DEC FIXED (8,3);
      B(7) DEC FIXED (4,3),Z980 CHAR(9),
      (SH(8),MIN(8),SE(8),MSE(8)) DEC FIXED(3);
DCL CH CHAR(100) INIT((50)'AB');
      %DCL R CHAR, RU CHAR, J FIXED,J1 FIXED;
%R='          IF I<0 THEN   BI=BIT(I,50)     CH=CHAR(BY
,50);  CH=SUBSTR(Y.50,50),IN=INDEX(Y,"CARY"),CALL F;';
      N=1000;
      SUBSTR(Y,100,40)='CARY';
      Z(1)=TIME;
      %DO J=2 TO 8;
```

```
        DO I=1 TO N;
        %J1=19*(J-2)-1;
        %RU=SUBSTR(R,J1,18);RU;
        END;
        Z(J)=TIME;
        %END
        F:PROC;
        END;
        DO I=1 TO 8;
    GET STRING(Z(I)) EDIT(SH(I),MIH(I),SE(I),MSE(I))(3F(2),F(3));
        END;
        T=MSE*0.001+SE+MIH*60.0+SH*3600;
        DO I=1 TO 6;
        B(I)=T(I+2)-T(I+1)+T(1)-T(2);
        END;
    PUT EDIT('ПРОДОЛЖИТЕЛЬНОСТЬ РАБОТЫ ОПЕРАТОРОВ',
    'И ВСТРОЕННЫХ ФУНКЦИЙ (МСЕК)','IF    ',B(1),
    'CALL  ',B(6),'BIT   ', B(2),'CHAR  ',B(3),
    'INDEX ',B(5),'SUBSTR',B(4)),
    (SKIP,X(20),A,SKIP,X(20),A,6(SKIP,X(20),A,X(5),F(6,3)));
        END BIR;
```

Передача масивів. Об'ява відмінних від констант границь в Фортрані дозволена лише в точному контексті, тобто в підпрограмі. Насправді границі масиву фіксовані. Вони лише уточнюються програмою, що викликає [10, С. 194]. В Алголі 68 формальні границі будуть тільки порожні [8, С. 53].

В ПЛ/1 і Алголі W, якщо потрібно передбачити, щоб деякі границі були невизначені, в об'яві формального параметра-масиву замінюють відповідні специфікації розміру на зірочки {Чи *n* Фортрану, чи * ПЛ/1 – дія та ж сама}. В Фортрані передача підмасивів стовпчиками. Для передачі підмасивів в ПЛ/1 вказуються піддіапазони або значення індексів. Розширюють можливості псевдоіндекси iSUB.

В Фортрані здійснюється передача тільки адреси масиву чи підмасиву. В Алголі W і ПЛ/1 складніше. Передається "дескриптор", який містить окрім адреси інформацію (число вимірів, границі, крок зміни індексів), яка забезпечує доступ до довільного елемента фактичного

параметра за формальним параметром і списком індексів.

В цих мовах програміст менше зв'язаний конкретним способом розміщення масивів в пам'яті. Платити потрібно втратою гнучкості підпрограм. Проте передана інформація багатша і система краще керує сумісністю параметрів і завершеністю їх обробки в програмі.

Формальний параметр може відповідати підпрограмі. Транслятор виконує синтаксичну перевірку відповідності між числом фактичних параметрів і їх типами і специфікаціями відповідних формальних параметрів [10, С. 202]. В ПЛ/1 формальний параметр може бути об'явлений з атрибутом

entry(тип1, тип2, ..., тип *n*),

який вказує, що мова йде про підпрограму, і уточнює тип її параметрів; атрибут *returns*(тип) уточнює (визначає) тип результату. Відповідний фактичний параметр повинен бути процедурою сумісного типу.

В Фортрані і Алголі W більш примітивно в цьому відношенні. В Фортрані відповідний фактичний параметр повинен бути об'явлений таким, що має тип *external*, тобто підпрограмою. В Алголі W формальний параметр-"підпрограма" позначується з допомогою *procedure* або *mun procedure*.

Наприклад:

```
Long real procedure integrale
    (long real value A, B,
        long real procedure fonction);
```

Виклик може мати вигляд

```
write(integrale(0, 1, longsin));
```

3.9. В пошуках кращої мови

Визначення Н. Віртом поняття "*програмне забезпечення*". Термін використовується для позначення сукупності програм, в якій однозначно з точністю до найменших деталей визначається дія всієї системи і її переходи із одного стану в інший. Програми, що складають цю сукупність, є константами в тому розумінні, що не залежать ні від яких "несприятливих обставин", що можуть виникнути в їх оточенні.

Із цього виходить, що в ПЗ не можуть виникнути неполадки, які є наслідком непередбачених подій чи старіння. Такі неполадки можуть бути лише результатом помилок, допущених при проектуванні його логічної

структури. Необхідна заміна терміну "надійність" на "коректність" у випадках, коли ми говоримо про ПЗ [15, С. 146].

Ступінь «ненадійності» ПЗ необхідно визначити як ймовірність, з якою будуть виконуватись його "некоректні" частини. Але ця міра не є властивістю ПЗ [15, С. 147].

Не слід шукати зовнішні причини. Причини тільки в неадекватності нашого мислення і в невмінні взаємодіяти при розробці. Усвідомлення цього приводить до наслідку — шлях воістину нескінченний, оскільки людині властиво помилятись.

Потрібні точні міркування і інструменти для проведення їх.

Серед таких інструментів в першу чергу необхідно назвати формальні мови, з допомогою яких ми виражаємо свої думки і різні абстракції, а також забезпечуємо передачу своїх думок і різних абстракцій іншим людям. Необхідно покращити програмістські інструменти.

Мова програмування не є лише засобом, з допомогою якого "кодують" деякий алгоритм. Звичайно, хороша мова — це ціла "система мислення", що включає абстраговані поняття, характерні для тієї чи іншої області людської діяльності [16, С. 12]. Така система служить не тільки для "передачі" алгоритму в комп'ютер, але й допомагає самому процесу алгоритмізації задачі. Тому й не ставав процес створення нових мов програмування.

Підмови. Якщо ми опускаємо деякі особливості мови або накладаємо додаткові обмеження, то отримуємо підмову [8, С. 383].

Надмови. Якщо ми додаємо нові особливості або визначаємо результати виконання програм, що залишились поки що невизначеними, то отримуємо надмову. Будь-яка конкретна реалізація мови буде, мабуть, містити варіації обох видів, так що краще було б говорити про "риси підмови" і "риси надмови".

Вже мови Фортран і Кобол досить великі (не кажучи про ПЛ/1) Програміст сам визначає підмножину [15, С. 296].

Аксіома Флона. Не існує і ніколи в майбутньому не буде мови, на якій писати погані програми хоч трохи буде важче, ніж хороші [15, С. 58].

Мова Паскаль мала великий вплив на подальший розвиток систем програмування [15, С. 219]. Це краща мова загального призначення для цілей системного програмування. Транслятор з мови Паскаль був повністю реалізований на мові Паскаль. Мова Асемблера використовувалась в мінімальному розмірі. Написані нею програми мали розмір всього близько

500 машинних слів. Транслятор був довжиною 7000 рядків і написаний за 16 місяців одним програмістом 1973 – 1975 р [15, С. 350].

Однак, з досвіду написання вище згаданих книг Керніган (якого можна вважати якщо не одним із творців мови Сі, то вже ж, щонайменше, таким, що вніс суттєвий вклад в його створення і розвиток) пише: "Мова Паскаль не придатна для написання реальних програм" [15, С. 204]. Керніган – один з розробників системи UNIX.

Елегантність мови Паскаль уявна. Відомий приклад з комплексними числами [15, С. 323].

Програмування на мові Асемблер чи, теж саме машинною мовою. При розкрутці на новій обчислювальній системі може знадобитись програмування в кодах ядра компілятора, а також дуже малого ядра операційної системи, враховуючи певні особливості архітектури, наприклад, механізм переривань [9, С. 146].

В операційній системі UNIX (розуміючи її в широкому сенсі) мова Сі повністю замінила мову Асемблера за винятком 1000 слів, які написані на мові Асемблера [9, С. 163].

Велика операційна система Multics реалізована на PL/1 (Лаб Белл, Дженерал Електрик і MTI, 1969 р.). Тільки 5% на машинній мові. Могло б бути і ще менше, якби розробники мали ПЛ/1 раніше [9, С. 147].

Ефективність досягається розумним вибором алгоритмів і структур, а не мікроефективністю, використанням машинної мови. Програмування полягає в роботі з даними, а не з тонкощами машини.

Трюки – ті особливості мови, які можуть бути незрозумілі читачеві вашої програми.

Традиційний стиль програмування на мові Сі, що нав'язується цією мовою програмування (тобто, використання *побічних* ефектів (ненадійності) при обчислюванні виразів, і покажчиків, взаємна заміннність масивів і покажчиків робить програми складними з точки зору можливого доведення їх правильності. *Підлий трюк* – це використання ненадійності.

Є мови, з яких нарешті виключили тип-покажчик.

Мова Сі була розроблена для програмістів, що звикли до використання Асемблера міні-комп'ютера PDP-11 [15, С. 163]. В нас вона з'явилась на аналогах цих комп'ютерів. Одним з базових принципів, використаних при розробці мови Сі, було забезпечення зручності, ефективності і гнучкості контролю над використаними апаратними засобами. Знання цього робить зрозумілим включення в мову Сі операцій,

що забезпечують можливість маніпулювання бітами всередині слів, і відсутність таких типів як множини і рядки [15, С. 273].

Типова небезпека. Помилку в наступному операторі мовою Сі може бути діагностовано лише побічно – за помилковими результатами і ніяк не на етапі трансляції: sin(1); [15, С. 166].

При доступі до поля структури p може бути не тільки покажчиком, а й просто змінною цілого типу p->memb. Основним поясненням, якщо не вибаченням, цього недоліку Сі може бути безтипова природа його попередників – мов BCPL (Річардсон 1969 р.) і Бі (Джонсон і Керніган).

В PL/1 реалізована спроба зберегти деяку суміш з Фортрану. Наприклад, ідентифікатори, які починаються з i, ..., n – за замовчуванням імена цілих двійкових чисел.

В PL/1 оператор END (позначає кінець групи операторів, наприклад циклу, групи DO, блоку BEGIN). Є *кратні* оператори END, коли замість кількох потрібних операторів END ставлять один, решту доставляє компілятор [9, С. 291]. В мові їх не може бути. Вони заохочують неохайність в програмуванні. Компілятор не має здогадуватись, що намірився зробити програміст.

Не може бути змінних типу мітка, інакше програма в тій чи іншій формі буде змінювати себе.

Виразна потужність. Структура програми має бути явно видима.

Мови програмування загального призначення легко можуть бути адаптовані для розв'язання самих різних задач, можливо шляхом визначення нових підпрограм і типів даних. Великі програмні комплекси звичайно містять набір підпрограм-примітивів, що реалізують базовий набір операцій над об'єктами, які використовуються, що відображують специфіку вирішуваної задачі. Такий набір підпрограм-примітивів фактично є спеціалізованим діалектом мови програмування, що використовується.

Мови програмування загального призначення повинні забезпечити легкість побудови їх спеціалізованих діалектів.

Ядро мови програмування відбиває її філософію, а решта в основному слідує цій філософії [15, С. 267] .

Одна з особливостей структурного програмування полягає в тому, що хід виконання програми має бути ясним з самого тексту. Звичайні блок-схеми впливають справді погано, бо вони допускають можливість

недисциплінованої і довільної структури програм. Ті ж, хто вважає, що *блок-схеми* все-таки важливі, Ф. Пейган рекомендує прийняти яку-небудь нестандартну графічну символіку, більш відповідну хорошій програмній структурі, особливо в зв'язку з представленням циклів [11, С. 8].

Автори [10] залишили 4 елементи блок-схем, щоб використати для пояснення структур керування, що вводяться в книзі. "Важливо відмітити, що автори не бажають захищати використання блок-схем загального вигляду". Блок-схеми були в той час допоміжним засобом документування програм. Блок-схеми – це один з об'єктів комбінаторики, можливо ще і тому вони так довго обговорювались в програмуванні.

Насправді ж програми мають ту ж, чи навіть кращу, внутрішню виразність, що і блок-схеми, і дають змогу бачити структуру програми в самому тексті, що дозволяє позбавитись від необхідності такого типу документації [10, С. 98].

Взагалі існує думка, що умовний оператор (в мові Сі є умовна *операція* – "?:") повинен виробляти значення. Тоді, наприклад, наступний оператор буде правильним:

I:=if I=7 then 1 else I+1; (поки що цього немає в Паскалі).

В ПЛ/1 комбіноване керування циклом з допомогою лічильника і умовного виразу [10, С. 134]:

do *лічильник* = *початзнач* to *границя* by *крок* while (*умова*) тіло *циклу* end

При керуванні з допомогою лічильника може бути кілька границь і кроків через кому (do j = 7 to 12, 19 to 26; [10, С. 342]).

В Алголі 68 загальна конструкція циклу має вигляд

для лічильник **від** початзнач **крок** р **до** границя **поки** умова

цк оператор1; оператор2; ... операторп **кц**

Індексовані переходи в Фортрані. Фортран має оператор, що нагадує варіантний [10, С. 140]:

goto(мітка1, мітка2, ... мітка *n*) ціле значення.

Таблиці переходів є засобами техніки програмування мовою Асемблера. Є оператор умовного переходу за цілим значенням, який враховує однаковий одиничний розмір пам'яті наступних кількох операторів [10, С. 142].

В мові ПЛ/1 є масиви міток. Якщо об'явлено

declare tabmet(100) label;

можна дати значення елементам цього масиву (наприклад,

tablmet(53) = M;), де M – мітка в програмі. Після чого можна виконати оператор goto tablmet(i);

Такий вид засобів обробки відноситься до сумнівних і повністю забороняється, якщо результатом роботи повинна бути ясна й надійна програма. Поняття подібного переходу відповідає практиці програмування в машинних мовах (або мовами рівня Асемблера), яка використовується трансляторами для перекладу операторів типу **case of** або індексованого **goto**; мова йде про таблицю переходів.

"Таблиця переходів" – це сукупність команд – безумовних переходів (однакової довжини в пам'яті), які дозволяють працювати з переходами на бажану мітку. Наводиться приклад з представленням оператора **case і of** Алголу W [10, С. 143].

Подальший розвиток ідеї – *таблиці рішень*, показано на прикладі лексичного аналізатора, керованого з допомогою таблиць [10, С. 111, 115].

Автори [15] стверджують, що всі програми, які написані деякою мовою програмування, просякнуті філософією, використаною при розробці цієї мови (спірне твердження) [15, С. 289].

Н. Вірт підкреслює, що в багатьох мовах є введення в мову явно машинно-орієнтованих засобів замість того, щоб вводити в них добре зрозумілі абстракції і об'єкти, наприклад [15, С. 338]:

1. Список міток, який дозволяє індексований перехід за призначенням (*goto* Фортрану і оператор-перемикач Алголу 60).

2. Використання адрес, покажчиків і т.д. і звичайних арифметичних операцій для роботи з ними.

3. Переривання і виняткові ситуації ПЛ/1.

4. Використання бітових рядків як наборів логічних значень, представлених вісімковими числами, в мові Алгол 68.

5. Використання однієї й тієї ж області для розміщення об'єктів, які мають різні типи – оператор *equivalence* мови Фортран.

Ці засоби не допомагають програмістові мислити абстрактно.

Граматики порівнюваних мов (Ада, Сі, Паскаль) мають різний ступінь регулярності, тобто в них по-різному повторно використовуються визначення різних підструктур [15, С. 296].

Представляється очевидним твердження, що мови Фортран, Джовіал і Ада утворюють ряд, в якому регулярність синтаксису зростає, і що чим ця регулярність більша, тим краще. Мова Кобол випадає з цього ряду – в ній повністю відкинута ідея про те, що одноманітність – це добре, і замість

цього для кожної із конструкцій вибрано спеціальний синтаксис, близький до сингаксису *природної мови* [15, С. 297].

"Зручність читання" і "зрозумілість" не завжди враховується при розробці мови, це впливає і на стиль. Наприклад, стиль програмування мовою Алгол 60 "а-ля" мова Ліспт (текст рядком) суттєво погіршує зручність читання [15, С. 229].

В мові Ада враховано, що всі складні оператори повинні бути явним способом обмежені [16, С. 185]:

if … end if

case … end case;

loop … end loop;

Механізм похідних типів дозволяє автоматично виявляти помилки, пов'язані з виконанням семантично без смислу операцій [15, С. 181].

Тільки для масивів опис типу можна зробити безпосередньо при описі об'єкта [15, С. 176]. Фактичні параметри, відповідні формальним змінним, що мають значення за замовчуванням, можуть бути опущені (виклик з іменами параметрів) [15, С. 177].

Ада пристосована для розробки великих програмних комплексів краще ніж будь-яка інша мова. Засоби керування розробкою таких комплексів повинні ввійти в склад середовища підтримки мови Ада (APSE) [15, С. 188].

Ада – перша мова загального призначення, в якій є високо рівневі засоби для організації паралельних обчислень (Алгол – не загального). Але в ній важко написати програми для паралельних векторних обчислень, рекурсивні виклики паралельних програм важко виразити природним чином [15, С. 192].

Паскаль, Евклід, Меза, Альфард, Клу. Одна з головних їх цілей – зробити більш важким написання поганих програм. Реалізація цієї мети призводить до того, що більш важче стає написання будь-яких програм.

Ліквідація семантичного розриву між машинами і мовами. В машині фон Неймана зі словами пам'яті не зв'язується ніякий певний смисл. На відміну цьому в деяких машинах (B6500 "Барроуз" або R2 фірми Rice Researsh) реалізована ідея даних, що самоідентифікуються, тобто кожне слово пам'яті містить і власну самоідентифікацію (1973 р.)

Принципово важливим є метод адресації слів. Крім адресного компонента покажчика (імені чи значення посилання на слово пам'яті)

можуть бути компоненти *повноваження*, *зміщення*.

Компонент *пастка* вводить ідею програмно-орієнтованого (а не системно-) переривання. Всі покажчики є унікальними, що забезпечується включенням при призначенні покажчика в адресне поле відмітки про час в якій-небудь формі. В системі ISPL кожний покажчик містить дозвіл читати/писати.

Концепція *здатність* в SYSTEM 250 визначає для модуля вказаний тип доступу до вказаної області пам'яті.

Машина SYMBOL 2R з мовою високого рівня була спеціально спроектована для особливої мови високого рівня SPL, яка є сумішшю характерних рис ПЛ/1, Алголу і АПЛ [9, С. 307]. Машина без програмної ОС. Для її функцій є апаратно-логічна схема (1972 р.)

Машина В1700 фірми "Барроуз" могла перетворюватись в машину з мовою Кобол, Фортран і т.д. підключенням одного з багатьох мікропрограмних інтерпретаторів. Коли виконується ОС, набором команд машини стає мова, на якій написана ОС.

В БМОК "Ельбрус" було реалізовано багато з розглянутих тут ідей, а також ряд нових.

І коротко про попередників мов СУБД і середовище-орієнтованих мов – мов табличних процесорів. (Оригінальна СУБД Пальма, розроблена в Україні, впроваджувалась в 1979 р.)

Кобол – алгоритмічна мова, орієнтована на задачі обробки даних, їх великих масивів. Програма складається з чотирьох розділів. Розділ даних містить числові і рядкові дані, об'єднані в групи, групи груп і т. д. Структурна ієрархія задається з допомогою номерів рівнів. Вищий рівень присвоюється логічному запису. Файл – сукупність записів.

Розділ процедур задає операції обробки даних у вигляді послідовності речень, іменованих параграфів і секцій, що складаються з наказових, умовних речень і тих, що керують трансляцією. *Ядро* мови призначене для обробки даних в ОП, а *функціональні модулі* призначені для обробки таблиць, доступу до файлів, сортування-злиття, генерації звітів, роботи з бібліотеками, комунікаційними засобами.

Гнучкість вибору *стандартних* підмножин мови основана на виділенні в ядрі і функціональних модулях фіксованих рівнів.

Мова *РПГ* створена на початку 60-х для машини масового використання IBM 1401 [18]. Умова задачі записувалась у вигляді таблиці.

В записах одного файлу могла бути інформація про інший – *зачеплення*. Поняття *пари таблиць*: таблиці аргументів і таблиці функцій. Виконувались дії пошуку, оновлення, поповнення. Можна було підключати підпрограми іншими мовами.

Програміст завжди думає однією мовою, потім перекладає іншою. В суперечках про мову кожен правий по-своєму. Після руйнування Вавілонської вежі пошуки кращої мови неймовірно ускладнились.

Замість висновків. Комп'ютер був і є інструментом і таким залишиться. Текст почав виконувати свою головну роль (передача думок) тільки в останні два тисячоліття, з неминучою появою науки і літератури (Ньютон і його послідовники наблизили до нас минуле щонайменше на 700 років [А.Т. Фоменко]). Це дуже незначний період порівняно з розвитком мови. За цей час мислення і філософія не змінились, головною причиною виникнення Всесвіту для більшості був і є антропний принцип [П. Девис]. Завдяки текстам всі автори завжди будуть сучасниками, а авторів алгоритмів, особливих текстів будуть вважати програмістами мабуть ще тисячоліття.

Фантастична, божевільна, наукова думка (Ернстові Добльгоферові присвячується). Пять-шість тисяч років тому. Десь в Раю чи Межиріччі. Під лагідним весняним сонцем берегом річки йде людина. Зупиняється, піднімає глиняну дощечку, яку загубив школяр, поспішаючи до школи.

І знову ця думка, яка мучить мозок вже багато років. Настане час, буде чудо. Не таке, як залізо, або верстат (трактор, літак, комп'ютер). Буде перевершене чудо писемності. Нехай скептики і песимісти твердять, що чудо як і буде, то не в цьому Всесвіті, і не в цій його частині. Але вже є програма і вона здійсниться. Ні плескіт води, ні крики працівників, ні калатання дверей не можуть вплинути на хід думки.

Післямова

Мови, основні ідеї яких виникли пізніше ніж ідеї основних елементів алгоритмічних мов, назвемо *післямовами.*

Як це видно і з першого розділу, в мовах **obiectum** – пряме запозичення і у формі копії-кальки: *предмет*, де *пред-* є копією латинської приставки ob-, а *-мет* (від метати "кидати") відтворює лат. *-iectum* (від *iacio* – "кидаю"). Подібного роду кальки: (*рос.*) *подлежащее, сказуемое, падеж, склонение, междометие, местоимение, прилагательное, существительное* – копії латинських кальок з давньогрецьких граматичних термінів [13, С. 122].

Програмний об'єкт як подібне з'єднання, тільки даних і програмного коду, з'являється в мові Симула і далі використовується в багатьох мовах (наприклад, клас в мові Паскаль Плюс Н. Вірта).

В наступній програмі для реалізації дуг, як нащадків кіл з їх атрибутами, і переміщення фігур на екрані монітора натисканням клавіш-стрілок використовується модуль Figures (його текст є після тексту програми). З прикладу можна зрозуміти, що таке віртуальні методи, поліморфізм і ієрархія (наслідування) об'єктів.

```
{ Copyright (c) 1989,90 by Borland International }
program FigureDemo;
{ From Chapter 4 the Turbo Pascal 6.0 User's Guide.
  Extending FIGURES.PAS with type Arc.
  Програма використовує модуль Graph і файли .BGI драйверів}
uses Crt, DOS, Graph, Figures;
const PathToDrivers = '\TP7\BGI';
{Директорія, де є *.BGI файли}
type
  Arc = object (Circle)
    StartAngle, EndAngle: Integer;
    constructor Init(InitX, InitY: Integer; InitRadius: Integer;
      InitStartAngle, InitEndAngle: Integer);
    procedure Show; virtual;
    procedure Hide; virtual;
  end;
var
  GraphDriver,  GraphMode, ErrorCode: Integer;
```

```
 AnArc,An: Arc;
 ACircle,Cir: Circle;
{ Декларація методів об'єкта Arc:}
constructor Arc.Init(InitX, InitY: Integer; InitRadius: Integer;
 InitStartAngle, InitEndAngle: Integer);
begin
 Circle.Init(InitX, InitY, InitRadius);
 StartAngle := InitStartAngle;
 EndAngle   := InitEndAngle;
end;
procedure Arc.Show;
begin
 Visible := True;
 Graph.Arc(X, Y, StartAngle, EndAngle, Radius);
end;
procedure Arc.Hide;
var
 TempColor: Word;
begin
 TempColor := Graph.GetColor;
 Graph.SetColor(GetBkColor);
 Visible := False;
 {Вимальовує дугу кольором фону, щоб зробити її невидимою}
 Graph.Arc(X, Y, StartAngle, EndAngle, Radius);
 SetColor(TempColor);
end;

{Головна програма:}
begin
 GraphDriver := Detect; { Визначає, який BGI використовується}
 DetectGraph(GraphDriver, GraphMode);
 InitGraph(GraphDriver, GraphMode, PathToDrivers);
 if GraphResult <> GrOK then
  begin
   Writeln(GraphErrorMsg(GraphDriver));
   if GraphDriver = grFileNotFound then
   begin
```

```
    Writeln('в ', PathToDrivers, '. поміняйте "PathToDrivers"');
    Writeln('константу, що визначає шлях до цих файлів.');
    Writeln;
  end;
  Writeln('Press Enter...');
  Readln;
  Halt(1);
 end;
{ All descendents of type Point contain virtual methods and  }
{ *must* be initialized before use through a constructor call. }
 ACircle.Init(151, 82, 50);
{Ініціалізація координат центру X,Y і радіуса кола}
 AnArc.Init(151, 82, 25, 0, 90);        { ... ще і кутів дуги }
                    {Start angle: 0; End angle: 90 }
{Replace AnArc with ACircle to drag a circle instead of an }
{Press Enter для зупинки переміщення фігур}
 ACircle.Drag(5); { Parameter is # of pixels to drag by }
 AnArc.Drag(5);  { Параметр вказує число пікселів переміщення}
 ACircle.Drag(5);
 Cir.Init(120, 60, 30);  An.Init(140, 100, 70, 0, 180); {Ще коло і дуга }
 An.Drag(5);       Cir.Drag(5);
 CloseGraph;
end.

{ Turbo Figures }
{ Copyright (c) 1989,90 by Borland International, Inc. }
unit Figures;
{ From Chapter 4 the Turbo Pascal 6.0 User's Guide.
 Virtual methods & polymorphic objects.}
interface
uses Graph, Crt;
type
 Location = object
  X,Y: Integer;
  procedure Init(InitX, InitY: Integer);
  function GetX: Integer;
  function GetY: Integer;
```

```
  end;
  PointPtr = ^Point;
  Point = object (Location)
    Visible: Boolean;
    constructor Init(InitX, InitY: Integer);
    destructor Done; virtual;
    procedure Show; virtual;
    procedure Hide; virtual;
    function IsVisible: Boolean;
    procedure MoveTo(NewX, NewY: Integer);
    procedure Drag(DragBy: Integer); virtual;
  end;
  CirclePtr = ^Circle;
  Circle = object (Point)
    Radius: Integer;
    constructor Init(InitX, InitY: Integer; InitRadius: Integer);
    procedure Show; virtual;
    procedure Hide; virtual;
    procedure Expand(ExpandBy: Integer); virtual;
    procedure Contract(ContractBy: Integer); virtual;
  end;

implementation
{ Реалізація методів об'єкта "положення":}
procedure Location.Init(InitX, InitY: Integer);
begin X := InitX; Y := InitY; end;
function Location.GetX: Integer;
begin GetX := X; end;
function Location.GetY: Integer;
begin GetY := Y; end;
{ Реалізація методів об'єкта "точка": }
constructor Point.Init(InitX, InitY: Integer);
begin Location.Init(InitX, InitY); Visible := False; end;
destructor Point.Done;
begin Hide; end;
procedure Point.Show;
begin Visible := True; PutPixel(X, Y, GetColor); end;
```

```
procedure Point.Hide;
begin Visible := False; PutPixel(X, Y, GetBkColor); end;
function Point.IsVisible: Boolean;
begin IsVisible := Visible; end;
procedure Point.MoveTo(NewX, NewY: Integer);
begin Hide; X := NewX; Y := NewY; Show; end;
function GetDelta(var DeltaX: Integer; var DeltaY: Integer): Boolean;
var KeyChar: Char; Quit: Boolean;
begin DeltaX := 0; DeltaY := 0;        { 0 – без змін позиції;}
 GetDelta := True;                     { Є переміщення  }
 repeat
  KeyChar := ReadKey;  {Читання 1-го байта коду натисн. клавіши}
  Quit := True;              {Використано правильний код}
  case Ord(KeyChar) of
    0: begin                { розширений, 2-байтовий код, перший 0}
       KeyChar := ReadKey;  {Читання другого байта коду}
       case Ord(KeyChar) of
        72: DeltaY := -1;  80: DeltaY := 1;
        75: DeltaX := -1;  77: DeltaX := 1;
       {Стрілки: вверх, вниз, вліво, вправо}
        else Quit := False; {Інші розширені коди ігнорувати}
       end; { case }
      end;
   13: GetDelta := False; { CR pressed – не переміщувати}
  else Quit := False;       { Ігнорувати інші натискання клавіш}
  end;  { case }
 until Quit;
end;
procedure Point.Drag(DragBy: Integer);
var DeltaX, DeltaY: Integer;
 FigureX, FigureY: Integer;
Begin  Show; {Показ переміщуваної фігури}
 FigureX := GetX; FigureY := GetY; {Читання початкових координат}
 { Цикл переміщення:}
 while GetDelta(DeltaX, DeltaY) do
 begin                    { Нові координати фігури: }
  FigureX := FigureX + (DeltaX * DragBy);
```

```
    FigureY := FigureY + (DeltaY * DragBy);
    MoveTo(FigureX, FigureY);    { Переміщення }
  end;
end;
{ Реалізація методів об'єкта "коло":}
constructor Circle.Init(InitX, InitY: Integer; InitRadius: Integer);
begin Point.Init(InitX, InitY); Radius := InitRadius; end;
procedure Circle.Show;
begin Visible := True; Graph.Circle(X, Y, Radius); end;
procedure Circle.Hide;
var TempColor: Word;
begin TempColor := Graph.GetColor; Graph.SetColor(GetBkColor);
  Visible := False;  Graph.Circle(X, Y, Radius);
                     Graph.SetColor(TempColor);
end;
procedure Circle.Expand(ExpandBy: Integer);
begin Hide; Radius := Radius + ExpandBy;
if Radius < 0 then Radius := 0; Show; end;
procedure Circle.Contract(ContractBy: Integer);
begin  Expand(-ContractBy); end;
{ No initialization section – Модуль без дій, необхідних до початку
виконання програми, що його використовує}
end.
```

Останнє слово (справжній епілог) як завжди за мовою Сі. Тип в ній називається "класом", а екземпляр – "об'єктом".

```
#include <stdio.h>
/* клас complex реалізує математичні дії з комплексними числами,
скорочена версія. Можна записати в головний файл complex.h [6] */
class complex {
double re, im;              // за замовчуванням, private
public:
complex(double r=0, double i=0)    //конструктор
{re = r; im = i;}    // якщо не вкажемо фактичний параметр
void print(void);
friend complex operator + (complex, complex);
friend complex operator * (complex, complex);
```

```
friend complex operator / (complex, complex);
            };
//Правила, методи
void complex:: print()
{ /*- це визначення елемента complex*/
printf("(%5.2f, %5.2f)\n", re, im);
}
complex operator +(complex a1, complex a2)
{
return complex (a1.re+a2.re,a1.im+a2.im);
}
complex operator *(complex a1, complex a2)
{
return complex    (a1.re * a2.re - a1.im * a2.im,
                   a1.re * a2.im + a1.im * a2.re);
}
complex operator /(complex a1, complex a2)
{// оригінальне рішення [6]
double r = a2.re;  /*(r, i) */
double i = a2.im;
double ti;
double tr;  /*(tr, ti)*/
tr = r<0? -r : r;      // фактично модулі r і i.
ti = i<0? -i : i;
if (tr <= ti)   { ti = r/i; tr = i*(1+ti*ti); r=a1.re; i=a1.im; }
else            { ti=-i/r; tr=r*(1+ti*ti); r=-a1.im; i= a1.re; }
return complex ((r*ti+i)/tr, (i*ti-r)/tr);
}
void main () {
/* обчислення напруги в колі змінного струму */
const complex j(0, 1); // уявна одиниця
const double pi = 3.1415926535897931;
double
L = .03,      //індуктивність у генрі
R = 5000,     // опір в омах
C = .02,      // ємність у фарадах
freq = 60,    // частота в герцах
```

```
omega = 2 * pi * freq;    //частота в радіанах/сек
complex I = 12, Z,        //струм, імпеданс
V;                        //напруга
Z = R + j * omega * L + 1/(j*omega*C);
V = Z*I;
V.print();          //printf("\n %d \n", sizeof(V));
} //Програма видає: (60000.00, 134.13)
```

Якщо нові слова можуть утворюватись об'єднанням інших слів або їх частин, то нові знаки можуть утворюватись модифікацією, розділом на частини вже відомих. Так утворились латинські *цифри-половинки*.

Знаки L, C, M сходять до букв основного для латинського алфавіту західногрецького алфавіту. Ці букви, що позначували придихові звуки (kh, th, ph), були відсутні в латинській мові, стали в ній використовуватись для позначення чисел, а з часом їх написання було пристосоване для написання латинських букв. C(100) стало ототожнюватись з першою буквою латинського слова centum (сто), M (1000), що є видозміненою формою грецької Φ {скромніша *стилізація*} з першою латинською буквою слова mille (тисяча). Знак D(500), ототожнений з латинською буквою, є половиною знаку Φ (1000) також як V (5) є верхньою половиною знаку X (10) [1, С. 892] {а L є нижньою половинкою C}.

Про символіку і походження букви (що позначує і як виникла) можна здогадатись, маючи її назву і графіку. Наприклад, для тета – Θ, θ – символіка *очевидна*. В той же час латинська буква *f* більш витончена ніж φ і не гірше зберегла первісну символіку.

В реальному процесі мовлення {речення} вибір чергового слова при породженні фрази статистично визначається як загальним формуючим принципом (думка, ситуація, психологічний стан), так і особливостями семантики, граматичного оформлення, фонетики, асоціативної хмари вже сказаних слів [19, С. 62]. Осмисленню побутових текстів людина навчається паралельно з оволодінням мовою. В індивідуальній свідомості утворюється потенційне ядро одразу і однозначно зрозумілих текстів.

При осмисленні нових понять, теорій в програмуванні (як і математичних текстів) звикання спочатку є необхідним. Потім воно долається зміною контексту і відповідно реалістичного, внутрішнього або зовнішнього аспектів семантики [19, С. 151]. Спочатку інтуїтивне поняття стає терміном, а далі, можливо, набуває і нового розуміння.

Список літератури

1. Н.Т. Бабичев, Я.М. Боровский. Словарь крылатых латинских слов. – М.: Рус. яз., 1988.
2. Бауэр Ф.Л., Гооз Г. Информатика. Вводный курс: Ч.2 – М.: Мир, 1990.
3. З.С. Брич, Д.В. Капилевич, Н.А. Клепкова. Фортран 77 для ПЭВМ ЕС: Справ. изд. – М.: Финансы и статистика, 1991.
4. Дж. Вейценбаум. Возможности вычислительных машин и человеческий разум. От суждений к вычислениям. – М.: Радио и связь, 1982.
5. У. Гренандер, В. Фрейбергер. Краткий курс вычислительной вероятности и статистики. – М.: "Наука", 1978.
6. С. Дьюхарст, К. Старк. Программирование на С++. – Киев: НИПФ "Диа-Софт", 1993.
7. Л.Ю. Куштенко. Этимологический справочник учителя английского языка. – К.: Рад. шк. 1987.
8. Ч. Линдси, С. ван дер Мюйлен. Неформальное введение в Алгол 68. – М.: Мир, 1973.
9. Г. Майерс. Надежность программного обеспечения. – М.: Мир, 1980.
10. Б. Мейер, К. Бодуэн. Методы программирования: В 2-х томах. Т.1. Пер. с франц. Ю.А. Первина. Под ред. с пред. А.П. Ершова. – М.: Мир, 1982.
11. Ф. Пейган. Практическое руководство по Алголу 68. – М.: Мир, 1979.
12. Словник іншомовних слів /За ред. О.С. Мельничука. – Київ: Поліграфкнига, 1977.
13. Ю.В. Откупщиков. К истокам слова. Рассказы о науке этимологии. Книга для учащихся. – Л.: «Просвещение». – 1968.
14. П.Л. Хижняк. Пишем вирус и ... антивирус. – М.: ИНТО, 1991.
15. Языки программирования Ада, Си, Паскаль. Сравнение и оценка/ Под ред. А.Г. Фьюэра, Н. Джехани. – М.: Радио и связь, 1989.
16. Язык программирования Ада (предварительное описание)/ Пер. с англ. В.М. Курочкина и Д.Б. Подшивалова. – М.: Финансы и статистика, 1981.
17. Н.А. Магариу. Язык программирования АПЛ.- М.: Радио и связь, 1983.
18. Ю.М. Липень, М.С. Марчолин, Э.А. Марук. Программирование на РПГ в ЕС ЭВМ. – М.: "Статистика", 1977.
19. Ю.И. Манин. Доказуемое и недоказуемое. – М.: Советское радио, 1979.
20. Повість врем'яних літ: Літопис (за Іпатським списком) / Пер. з давньоруської, післяслово, комент. В.В. Яременка. – К.: Рад. письменник. – 1990.

Навчальне видання

Василь Харитонович Власюк

ОСНОВИ ПРОГРАМУВАННЯ АЛГОРИТМІЧНИМИ МОВАМИ

Навчальний посібник

Оригінал-макет підготовлено автором

Редактор С.А. Малішевська

Навчально-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 20.02.04 Гарнітура Times New Roman
Формат 29,7x42 1/4 Папір офсетний
Друк різографічний Ум. друк. арк. 8,83
Тираж 90 прим.
Зам. № 2004-38

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ