УВОДЗІНЫ

§ 1. Алгарытм і яго ўласцівасці

Тэрмін «алгарытм» паходзіць ад прозвішча матэматыка IX ст. Мухамеда ібн Муса аль-Харэзмі, які сфармуляваў правілы чатырох асноўных арыфметычных дзеянняў. Менавіта гэтыя правілы спачатку і называліся алгарытмамі, але пазней алгарытмам сталі называць любы спосаб вылічэнняў, адзіны для некаторага класа зыходных даных.

Прыклад 1.1. Алгарытм «Рэшата Эратасфена» дазваляе атрымаць простыя лікі, якія не пераўзыходзяць N.

1. Выпішам запар усе натуральныя лікі ад 2 да N.

2. Возьмем першы лік 2 і закрэслім кожны другі лік, пачынаючы адлік з наступнага за двойкай ліку.

3. Возьмем першы незакрэслены лік, большы за 2 (лік 3), і закрэслім кожны трэці лік, пачынаючы адлік ад ліку, які стаіць пасля 3 (улічваючы і раней закрэсленыя лікі).

4. Працягнем дзеянні датуль, пакуль першы незакрэслены лік не акажацца большым за $\sqrt{N}$.

5. У выніку не закрэсленымі акажуцца ўсе простыя лікі, якія не пераўзыходзяць N, і толькі яны.

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~

У інфарматыку паняцце алгарытму прыйшло з матэматыкі.

Алгарытм — дакладна вызначаная сістэма зразумелых выканаўцу прадпісанняў, фармальнае выкананне якіх дазваляе атрымаць рашэнне задачы для любога дапушчальнага набору зыходных даных за канечны лік крокаў.

Прыведзенае азначэнне не з'яўляецца азначэннем у матэматычным сэнсе слова, паколькі ў ім выкарыстаны іншыя нявызначаныя паняцці — «сістэма прадпісанняў», «фармальнае выкананне» і інш. Гэта апісанне паняцця «алгарытм» раскрывае яго сутнасць. (Разгледзьце прыклад 1.1.) Апісанне можна ўдакладніць, паказаўшы агульныя ўласцівасці, якія характэрны для алгарытмаў. Да іх належаць: дыскрэтнасць, дэтэрмінаванасць (пэўнасць), зразумеласць, выніковасць, канечнасць, масавасць.

Дыскрэтнасць. Алгарытм разбіваецца на асобныя дзеянні (крокі). Выкананне чарговага дзеяння магчыма толькі пасля завяршэння папярэдняга. Пры гэтым набор прамежкавых даных канечны і атрымліваецца па пэўных правілах з даных папярэдняга дзеяння. **Каманда** з'яўляецца спецыяльным указаннем для выканаўца, якое прадпісвае выканаць кожнае асобнае дзеянне. Каманды, што залічваюць да сістэмы каманд выканаўца,

называюць простымі; іншыя каманды могуць быць вызначаны праз простыя.

Дэтэрмінаванасць. Калі алгарытм неаднаразова ўжыць у дачыненні да адных і тых жа зыходных даных, то кожны раз павінны атрымлівацца адны і тыя ж прамежкавыя вынікі і адзін і той жа выхадны вынік. Дадзеная ўласцівасць азначае, што вынік выканання алгарытму вызначаецца толькі ўваходнымі данымі і камандамі самога алгарытму і не залежыць ад выканаўца алгарытму.

Зразумеласць. Алгарытм не павінен змяшчаць каманд, сэнс якіх выканаўца можа ўспрымаць неадназначна. Запіс алгарытму павінен быць выразным, поўным і зразумелым. У выканаўцы не павінна ўзнікаць неабходнасці ў прыняцці якіх-небудзь самастойных рашэнняў.

Выніковасць. Пры дакладным выкананні каманд алгарытму вынікам павінен быць адказ на пытанне задачы. Калі спосаб атрымання наступных велічынь з якіх-небудзь зыходных не прыводзіць да выніку, то павінна быць вызначана, што трэба лічыць вынікам выканання алгарытму. У якасці аднаго з магчымых адказаў можа быць усталяванне таго факту, што задача не мае рашэння.

Канечнасць. Рэалізуемы па зададзеным алгарытме працэс павінен спыніцца праз канечны лік крокаў і выдаць шуканы вынік. Гэта ўласцівасць цесна звязана з уласцівасцю выніковасці.

Неабходнасць пабудовы фармальнага азначэння алгарытму прывяла да з'яўлення ў 20—30-х гг. XX ст. тэорыі алгарытмаў. Для вызначэння рознымі матэматыкамі былі прапанаваны:

- машына Цьюрынга;
- машына Поста;
- нармальны алгарытм Маркава.

Існавалі і іншыя азначэнні алгарытму. Пасля было даказана, што ўсе яны эквівалентныя.

Алан Мэтысан Цьюрынг (1912—1954) — англійскі логік і матэматык, які зрабіў істотны ўплыў на развіццё інфарматыкі. Прапанаваная ім у 1936 г. абстрактная вылічальная машына Цьюрынга дазволіла фармалізаваць паняцце алгарытму, якое выкарыстоўваецца ў тэарэтычных і практычных даследаваннях.

Эміль Леон Пост (1897—1954) амерыканскі матэматык і логік. Вядомы сваімі працамі па матэматычнай логіцы. Прапанаваў абстрактную вылічальную машыну — машыну Поста (1936).

Прыклад 1.2. Часта рэцэпты гатавання якіх-небудзь страў называюць алгарытмамі. У дадзеным выпадку парушаецца ўласцівасць дэтэрмінаванасці, паколькі пры гатаванні якой-небудзь стравы рознымі людзьмі вынік можа быць розным (напрыклад, ён можа залежаць ад таго, на якой пліце гатавалі, ці ад якасці прадуктаў). Акрамя таго, у рэцэптах часта бываюць фразы «пасаліць на смак», «дабавіць 2—3 лыжкі цукру» і г. д., якія парушаюць уласцівасць зразумеласці.

Прыклад 1.3. Задача можа мець рашэнне, але сфармуляваць алгарытм рашэння гэтай задачы не заўсёды атрымліваецца. Калі чалавеку прапанаваць фатаграфіі жывёл, то ён дастаткова хутка зможа падзяліць іх на дзве групы: дзікія і свойскія. Аднак сфармуляваць алгарытм, паводле якога ён гэта зрабіў, на сённяшні дзень не ўяўляецца магчымым. Некаторыя задачы класіфікацыі сёння паспяхова рашаюцца сістэмамі штучнага інтэлекту з дапамогай метадаў машыннага навучання. Аднак характэрнай рысай такіх метадаў з'яўляецца не прамое рашэнне задачы, а працэс навучання ў ходзе аналізу мноства падобных задач.

Прыклад 1.4. Спосабы праверкі алгарытму на правільнасць работы:

- матэматычны доказ;
- выкарыстанне спецыяльна падабраных тэстаў.

Масавасць. Алгарытм прыдатны для рашэння любой задачы з некаторага класа задач, г. зн. пачатковая сістэма велічынь можа выбірацца з некаторага мноства зыходных даных, якое называецца **абсягам прымянімасці алгарытму.**

Для практычнага рашэння задач на камп'ютары найбольш істотная ўласцівасць масавасці. Як правіла, каштоўнасць праграмы для карыстальніка будзе тым вышэйшай, чым большы клас аднатыпных задач яна дазволіць рашаць.

Калі распрацаваная паслядоўнасць дзеянняў не валодае хоць бы адной з пералічаных вышэй уласцівасцей, то яна не можа лічыцца алгарытмам. Для разумення ўласцівасцей алгарытму разгледзьце прыклады 1.2 і 1.3.

Для аднаго і таго ж алгарытму могуць існаваць розныя формы запісу: тэкставае апісанне, блок-схема, машына Цьюрынга і інш. Незалежна ад формы запісу любы алгарытм можа быць прадстаўлены з выкарыстаннем базавых алгарытмічных канструкцый: **паслядоўнасць, цыкл** і **галінаванне.**

У межах тэорыі алгарытмаў адбываецца аналіз розных алгарытмаў рашэння задачы для выбару найбольш эфектыўнага (аптымальнага). Распрацоўка інструментаў для аналізу эфектыўнасці алгарытмаў — адна з задач тэорыі алгарытмаў.

Любы алгарытм трэба правяраць на правільнасць работы (прыклад 1.4).

? **1.** Што такое алгарытм?
2. Якія ўласцівасці характарызуюць алгарытм?
3. Якія базавыя алгарытмічныя канструкцыі выкарыстоўваюцца пры складанні алгарытмаў?

Практыкаванні

1 Пракаменціруйце асноўныя ўласцівасці алгарытму для рэшата Эратасфена.

2 Аль-Харэзмі склаў алгарытмы арыфметычных дзеянняў яшчэ ў IX ст. Складзіце алгарытмы выканання арыфметычных дзеянняў у слупок. Праверце для складзеных алгарытмаў асноўныя ўласцівасці.

3 Прывядзіце прыклады алгарытмаў, якія валодаюць названымі прыметамі.

1. Выкарыстоўваецца толькі алгарытмічная канструкцыя *паслядоўнасць*.
2. Прысутнічае алгарытмічная канструкцыя *галінаванне*.
3. Прысутнічае алгарытмічная канструкцыя *цыкл*.
4. Выкарыстоўваюцца падпраграмы.

4 Апішыце паслядоўнасць дзеянняў для рашэння задачы «Воўк, каза і капуста».

Аднойчы селяніну спатрэбілася перавезці праз раку ваўка, казу і капусту. У селяніна ёсць лодка, у якой можа змясціцца, акрамя самога селяніна, толькі адзін аб'ект — ці воўк, ці каза, ці капуста. Калі селянін пакіне без нагляду ваўка з казой, то воўк з'есць казу; калі селянін пакіне без нагляду казу з капустай, каза з'есць капусту. Як селяніну перавезці на іншы бераг і ваўка, і казу, і капусту ў цэласці і захаванасці?

Ці будзе атрыманая паслядоўнасць дзеянняў алгарытмам? Якая ўласцівасць алгарытму не выконваецца? Ці магчыма перафармуляваць задачу так, каб аналагічная паслядоўнасць дзеянняў стала алгарытмам?

5 Ёсць два пясочныя гадзіннікі на 3 мін і на 8 мін. Як з іх дапамогай адмераць 7 мін? Вызначыце сістэму каманд выканаўца, які можа рашаць гэту задачу, і складзіце для яго паслядоўнасць дзеянняў, што прыводзіць да адказу. Якія алгарытмічныя канструкцыі былі выкарыстаны пры рэалізацыі? Ці можна лічыць атрыманую паслядоўнасць дзеянняў алгарытмам? Якая ўласцівасць алгарытму не выконваецца? Ці магчыма перафармуляваць задачу так, каб аналагічная паслядоўнасць дзеянняў стала алгарытмам?

§ 2. Мовы праграмірaвання

2.1. Высокаўзроўневыя мовы праграмірaвання

Любы алгарытм разлічаны на канкрэтнага выканаўцу, які мае сваю сістэму каманд. Алгарытм для камп'ютара павінен быць запісаны з дапамогай каманд, якія камп'ютар можа выконваць.

Першай высокаўзроўневай мовай праграмірaвання, якая была рэалізавана практычна, стаў у 1949 г. кароткі код (Short Code). Аперацыі і пераменныя ў гэтай мове праграмірaвання кадзіраваліся двухсімвальнымі спалучэннямі.

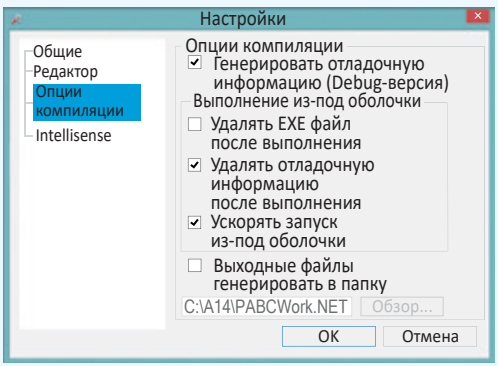
Прыклад 2.1. Некаторыя мовы праграміравання высокага ўзроўню:

- C++;
- Python;
- Pascal (Delphi);
- C#;
- Java;
- JavaScript;
- Perl;
- Fortran;
- VisualBasic;
- Lisp.

Прыклад 2.2. Пры трансляцыі праграмы адбываецца пераўтварэнне тэксту з адной мовы на іншую. Адрозніваюць наступныя віды трансляцыі: кампіляцыя і інтэрпрэтацыя.

Кампілятар — транслятар, што пераўтварае зыходны код з якой-небудзь мовы праграміравання на машынны. У выніку ствараецца файл, які можа быць выкананы непасрэдна ў аперацыйнай сістэме.

Асяроддзе праграміравання PascalABC мае ўбудаваны кампілятар, опцыі якога можна наладзіць, выканаўшы каманду **Сервис → Настройки**:



Інтэрпрэтатар — транслятар, які можа працаваць двума спосабамі:

- чытаць код і выконваць яго адразу (чыстая інтэрпрэтацыя);
- чытаць код, ствараць у памяці прамежкавае ўяўленне кода (байт-код ці р-код), выконваць прамежкавы паказ кода.

Чыстая інтэрпрэтацыя ўжываецца для моў JavaScript, VBA, Lisp і інш. Прыклады інтэрпрэтатараў, якія ствараюць байт-код: Perl, PHP, Python і інш.

Устройствам, якое выконвае каманды ў камп'ютары, з'яўляецца працэсар. Сістэму каманд працэсара называюць машыннай мовай або машынным кодам. Кожная каманда працэсара запісваецца ў двайковым кодзе. Для запісу гэтых каманд у сімвальнай форме выкарыстоўваюць мову Асэмблер. Асэмблер з'яўляецца мовай нізкага ўзроўню, паколькі змяшчае каманды, што адлюстроўваюць машынны код.

У мовах праграміравання высокага ўзроўню выкарыстоўваюцца каманды, якія аб'ядноўваюць паслядоўнасці машынных каманд. Праграмы, напісаныя на такіх мовах, прасцейшыя для разумення чалавекам, паколькі для абазначэння каманд выкарыстоўваюць словы натуральнай мовы (часцей за ўсё англійскай).

Мовы праграміравання высокага ўзроўню, або высокаўзроўневыя мовы праграміравання (прыклад 2.1), былі распрацаваны для таго, каб сутнасць алгарытму магла не залежаць ад апаратнай рэалізацыі камп'ютара. Для пераўтварэння тэксту праграмы, напісанай на мове высокага ўзроўню, у элементарныя машынныя каманды выкарыстоўваюцца спецыяльныя праграмы — **транслятары** (прыклад 2.2). Напрыклад, у тэксце праграмы дастаткова напісаць «while», а ўжо транслятар мовы перавядзе гэту каманду ў паслядоўнасць машынных кодаў. Прынцыпы работы транслятараў залежаць ад апаратнага і праграмнага забеспячэння камп'ютара.

Мовы праграміравання высокага ўзроўню з'яўляюцца фармальнымі

штучнымі мовамі. У кожнай мовы праграміравання можна вылучыць два складальнікі: сінтаксіс і семантыку. **Сінтаксіс** (граматыка мовы) — сукупнасць правіл для напісання праграмы. **Семантыка** — сэнсавы бок мовы (вызначае сэнсавы змест моўнай канструкцыі).

Тэкст праграмы на мове высокага ўзроўню ўяўляе сабой звычайны тэкставы файл. Для яго «чытання» і ператварэння ў паслядоўнасць машынных каманд выконваецца аналіз тэксту праграмы — праверка на адпаведнасць сінтаксічным правілам і семантыцы дадзенай мовы праграміравання. У выпадку выяўлення неадпаведнасці кампілятар можа выдаваць паведамленні пра памылкі (прыклад 2.3).

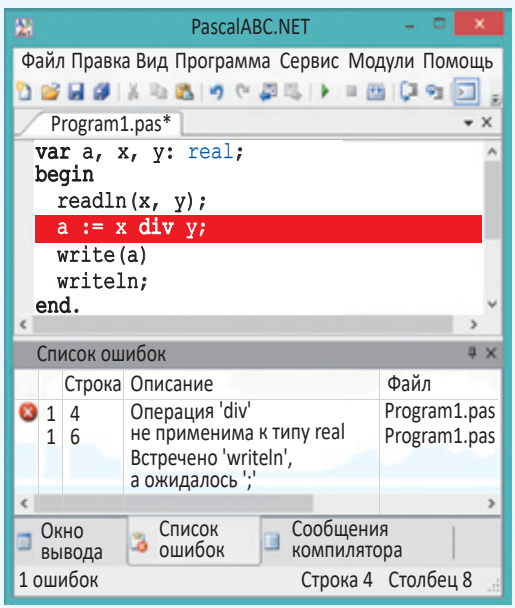
За час існавання вылічальных машын было створана больш за 8 тыс. моў праграміравання, і штогод з'яўляюцца новыя. Некаторыя з іх належаць да вузкаспецыяльных, іншыя выкарыстоўваюцца мільёнамі праграмістаў па ўсім свеце.

Існуюць розныя падыходы да класіфікацыі моў праграміравання. Паводле ступені адрознення семантыкі мовы ад машыннага кода мовы дзеляць на нізкаўзроўневыя і высокаўзроўневыя. Часта мовы праграміравання падзяляюць на кампіляваныя і інтэрпрэтаваныя, аднак такое дзяленне ўмоўнае, паколькі для любой традыцыйна кампіляванай мовы (такой, як Паскаль) можна напісаць інтэрпрэтатар.

У 1951 г. амерыканскі вучоны Грэйс Мюрэй Хопер (1906—1992) стварыла першы ў свеце кампілятар і ўвяла сам гэты тэрмін.

Кампілятар ажыццяўляў функцыю аб'яднання каманд, выконваў вылучэнне памяці камп'ютара і пераўтварэнне каманд высокага ўзроўню (у той час псеўдакодаў) у машынныя каманды.

Прыклад 2.3. Семантычная (радок 4) і сінтаксічная (радок 6) памылкі ў асяроддзі PascalABC.



Роберт В. Флойд (1936—2001) — амерыканскі вучоны ў галіне тэорыі вылічальных сістэм. Лаўрэат прэміі Цьюрынга. Упершыню тэрмін «парадыгма праграміравання» быў выкарыстаны Р. Флойдам у 1978 г. у час атрымання прэміі Цьюрынга: «Калі прагрэс мастацтва праграміравання ў цэлым патрабуе пастаяннага вынаходства і ўдасканалення парадыгм, то ўдасканаленне мастацтва асобнага праграміста патрабуе, каб ён пашыраў свой рэпертуар парадыгм».

Флойд адзначаў, што парадыгмы праграміравання не з'яўляюцца ўзаемавыключальнымі, яны могуць спалучацца, узбагачаючы інструментарый праграміста.

Прыклад 2.4. Асноўная ідэя структурнага праграміравання заключаецца ў тым, што праграма павінна мець простую структуру, добра чытацца і лёгка мадыфікавацца. Структураванасць кода падтрымліваецца дзякуючы падпраграмам, якія выклікаюцца з іншых падпраграм. Структурнае праграміраванне падтрымліваюць такія мовы, як Pascal, Go, C і шмат якія іншыя.

Прыклад 2.5. Асаблівасць моў працэдурнага праграміравання складаецца ў тым, што задачы разбіваюцца на крокі і рашаюцца крок за крокам. Рашэнне для кожнага асобнага кроку афармляецца ў выглядзе асобнай працэдуры. Да працэдурных моў залічваюць: C, Pascal, Lua і інш.

2.2. Парадыгмы праграміравання

У гісторыі развіцця моў праграміравання можна вылучыць розныя парадыгмы праграміравання — сукупнасць ідэй і паняццяў, якія вызначаюць стыль праграміравання. Паміж парадыгмамі і мовамі праграміравання няма цвёрдай сувязі. Парадыгма паказвае адзін з магчымых спосабаў выкарыстання сродкаў мовы праграміравання для напісання кода праграмы. Часта мова праграміравання, створаная ў межах адной парадыгмы, праз некаторы час мадэрнізуецца, пашыраецца і пачынае выкарыстоўвацца ў межах іншай парадыгмы.

Разгледзім некаторыя парадыгмы праграміравання.

Структурнае праграміраванне — парадыгма праграміравання, у аснаве якой ляжыць уяўленне праграмы ў выглядзе блокаў іерархічнай структуры. Распрацавана ў канцы 1960-х — пачатку 1970-х гг. У адпаведнасці з дадзенай парадыгмай любая праграма складаецца з трох базавых кіруючых структур: *галінаванне, цыкл* і *паслядоўнасць*; акрамя таго, выкарыстоўваюцца падпраграмы (прыклад 2.4). Распрацоўка праграмы вядзецца пакрокава, метадам «зверху ўніз» (сыходнае праграміраванне): спачатку вызначаюцца мэты рашэння задачы, а затым ідзе дэталізацыя кожнага кроку, які, стаўшы асобнай задачай, таксама можа дэталізавацца.

Працэдурнае праграміраванне — парадыгма праграміравання, пры якой каманды, што выконваюцца паслядоўна, можна сабраць у падпраграмы

з дапамогай механізмаў самой мовы (прыклад 2.5). Гэта канцэпцыя праграмiравання «знізу ўверх», або канцэпцыя ўзыходнага праграмiравання, — распрацоўка праграм пачынаецца з распрацоўкі падпраграм (працэдур, функцый), у той час як прапрацоўка агульнай схемы не скончылася.

Парадыгмы структурнага і працэдурнага праграмiравання заснаваны на падпраграмах. Розніца заключаецца ў парадку іх распрацоўкі: «зверху ўніз» ці «знізу ўверх».

Функцыянальнае праграмiраванне — парадыгма праграмiравання, у якой працэс вылічэння тлумачыцца як вылічэнне значэнняў функцый у матэматычным разуменні. Засноўваецца функцыянальнае праграмiраванне на вылічэнні вынікаў функцый ад зыходных даных і вынікаў іншых функцый і не прадугледжвае відавочнага захоўвання стану праграмы (прыклад 2.6).

Аб'ектна-арыентаванае праграмiраванне (ААП) — парадыгма праграмiравання, заснаваная на ўяўленні праграмы ў выглядзе сукупнасці аб'ектаў і адлюстраванні іх узаемадзеяння. Канцэпцыя ААП дазваляе аб'яднаць даныя і алгарытмы іх апрацоўкі ў адзіную структуру, якую называюць класам. Кожны аб'ект з'яўляецца экземплярам якога-небудзь класа (прыклад 2.7). У ААП праграма ўяўляе сабой набор аб'ектаў, якія маюць стан і паводзіны. Стан і паводзіны аб'екта могуць змяніцца ў выніку падзеі. Праграма ў цэлым — гэта аб'ект. Для

Прыклад 2.6. У аснове функцыянальных моў ляжыць лямбда-вылічэнне (λ-вылічэнне) — матэматычная тэорыя для фармалізацыі паняцця вылічальнасці. Да іх залічваюць: Haskell, Lisp, F# і інш.

Прыклад 2.7. У праграме *тэкставы рэдактар* аб'ектамі могуць быць абзац тэксту, меню, кнопкі і г. д. У класе, які апісвае кнопку, змяшчаюцца даныя пра памер кнопкі, яе знешні выгляд, алгарытмы, якія дазваляюць «націснуць» кнопку ці «навесці на яе мыш» і інш.

Канкрэтная кнопка, напрыклад Ч, з'яўляецца аб'ектам. Такая падзея, як «пстрычка мышшу па такой кнопцы», зменіць напісанне тэксту. Падзея «двайная пстрычка мышшу па абзацы тэксту» вылучыць яго і г. д.

Да аб'ектна-арыентаваных моў залічваюць:

- C++;
- Java;
- Delphi;
- Python;
- Ruby і інш.

Развіццё аб'ектна-арыентаванага праграмiравання часта звязваюць з паняццямі «падзея» (падзейна-арыентаванае праграмiраванне) і «кампанент» (кампанентнае праграмiраванне).

Асяроддзе PascalABC дае магчымасць ствараць аконныя дадаткі. Пры распрацоўцы інтэрфейсу праграмы выкарыстоўваюцца візуальныя кампаненты, а праграмны код заснаваны на падзейным праграмiраванні. Ствараць такія дадаткі вы навучыцеся ў 11-м класе.

Прыклад 2.8. Мультыпарадыгменныя мовы праграміравання часцей за ўсё падтрымліваюць працэдурную (структурную), аб'ектна-арыентаваную і функцыянальную парадыгмы: C++, Python, JavaScript, Ruby, C#.

Існуюць і іншыя класіфікацыі і спосабы параўнання розных моў праграміравання[1].

Прыклад 2.9. Вучэбная мова забяспечвае прастату, выразнасць і лёгкачытальнасць канструкцый мовы. Як вучэбныя мовы праграміравання распрацоўваліся: Basic, Pascal, Logo, Scratch.

Прыклад 2.10. Некаторыя эзатэрычныя мовы служаць для праверкі матэматычных канцэпцый (Thue, Unlambda), іншыя ствараюцца для забавы. Часта яны парадыруюць «сапраўдныя» мовы праграміравання ці з'яўляюцца абсурдным увасабленнем канцэпцый праграміравання (Smetana, Var'aq, FiM++ і інш.).

Прыклад 2.11. Псеўдакод алгарытму знаходжання сумы квадратаў першых n натуральных лікаў.

```
ввод n;
S = 0
нц для i от 1 до n
  S = S + i * i
кц
```

Службовыя (ключавыя) словы — зарэзерваваныя словы, якія маюць спецыяльныя значэнні для кампілятара. Іх нельга выкарыстоўваць як ідэнтыфікатары ў праграмах.

выканання сваіх функцый яна звяртаецца да аб'ектаў, якія ў яе ўваходзяць. Яны, у сваю чаргу, могуць звяртацца да іншых аб'ектаў, рэалізоўваць свае метады ці рэагаваць на падзеі. Аб'ектна-арыентаванае праграміраванне асабліва важна пры рэалізацыі буйных праектаў.

Большасць сучасных моў праграміравання з'яўляюцца **мультыпарадыгменнымі** — падтрымліваюць адразу некалькі парадыгм праграміравання (прыклад 2.8).

Асобна разглядаюць такія класы моў праграміравання, як **вучэбныя** (прыклад 2.9) і **эзатэрычныя** мовы праграміравання (прыклад 2.10).

Часта для запісу алгарытмаў ужываюць **псеўдакод** — мову апісання алгарытмаў, якая выкарыстоўвае ключавыя словы моў праграміравання, але прапускае дэталі, неістотныя для разумення алгарытму (напрыклад, апісанні пераменных). Галоўная мэта выкарыстання псеўдакода — забяспечыць разуменне алгарытму чалавекам, зрабіць апісанне больш успрымальным, чым зыходны код на мове праграміравання. Пры напісанні псеўдакода можа выкарыстоўвацца лексіка рускай мовы (прыклад 2.11).

2.3. Асноўныя структурныя элементы мовы праграміравання

Для запісу элементаў мовы праграміравання выкарыстоўваецца алфавіт. **Алфавіты** большасці моў праграміравання блізкія адзін аднаму па сін-

[1] https://ru.wikipedia.org/wiki/Сравнение_языков_программирования (дата доступу: 10.02.2019).

таксісе і, як правіла, выкарыстоўваюць літары лацінскага алфавіта, арабскія лічбы і агульнапрынятыя спецсімвалы (знакі прыпынку, знакі матэматычных аперацый і параўнанняў, раздзяляльнікі, службовыя словы). Большасць распаўсюджаных моў праграмiравання змяшчаюць у сваім алфавіце наступныя элементы:

- літары — {AaBbCcDd...};
- лічбы — {0 1 2 3 4 5 6 7 8 9};
- знакі арыфметычных аперацый — {× / + -...};
- знакі параўнання — {< > = ...};
- раздзяляльнікі — {. , ; : ( ) { } [ ]... };
- службовыя словы — {**if while for** і г. д.};
- каментарыі — любы набор сімвалаў і інш.

Нягледзячы на значныя адрозненні паміж мовамі, шмат якія фундаментальныя паняцці ў большасці моў праграмiравання падобныя паміж сабой (прыклад 2.12). Паводле вядомай формулы Н. Вірта «Алгарытмы + структуры даных = праграмы», разглядаючы мову праграмiравання, трэба гаварыць пра спосабы запісу каманд алгарытму з дапамогай аператараў і арганізацыі работы з данымі (прыклад 2.13).

Аператары

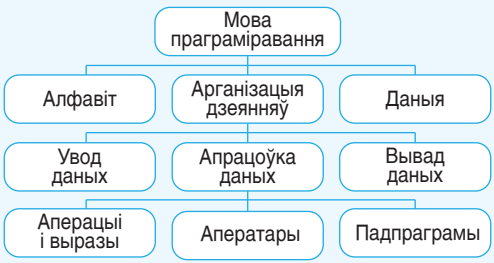
Адным з асноўных паняццяў усіх моў праграмiравання з'яўляецца аператар, які ўяўляе сабой скончаную фразу мовы і з'яўляецца прадпісаннем на выкананне пэўных дзеянняў па апрацоўцы даных. Праграма будуецца з аператараў гэтак жа, як тэкст літаратурнага твора фарміруецца са сказаў.

Прыклад 2.12. Некаторыя службовыя словы (у алфавітным парадку) у розных мовах праграмiравання.

Pascal	Python	C++

`const,`
`do,`
`else,`
`for,`
`if,`
`then,`
`var,`
`while`
	`def,`
	`elif,`
	`else,`
	`for,`
	`if,`
	`return,`
	`while`
		`const,`
		`do,`
		`else,`
		`for,`
		`if,`
		`return,`
		`while`

Прыклад 2.13. Структурная схема працэдурнай мовы праграмiравання.



Выразы будуюцца з велічынь (канстант і пераменных), функцый, дужак, знакаў аперацый і г. д. Тып выразу вызначаецца вынікам вылічэнняў. Выразы могуць прымаць лікавыя, лагічныя, сімвальныя, радковыя і іншыя значэнні.

Выраз 5 + 3 з'яўляецца лікавым, а выраз $A + B$ можа мець самы розны сэнс у залежнасці ад таго, што стаіць за ідэнтыфікатарамі A і B (калі A і B — радкі, то вынік — радок, які атрымаўся пры канкатэнацыі зыходных радкоў).

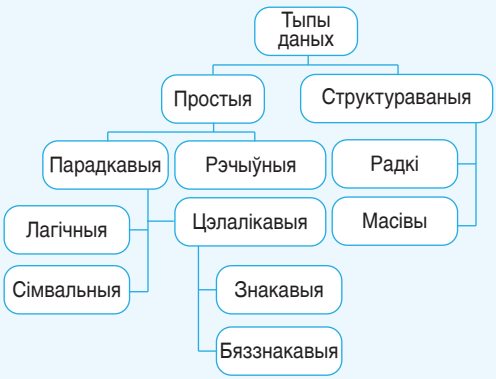
Прыклад 2.14. Запіс аператара цыкла (пошук сумы квадратаў першых n натуральных лікаў).

Pascal:
```
for var i := 1 to n do
  s := s + i * i;
```

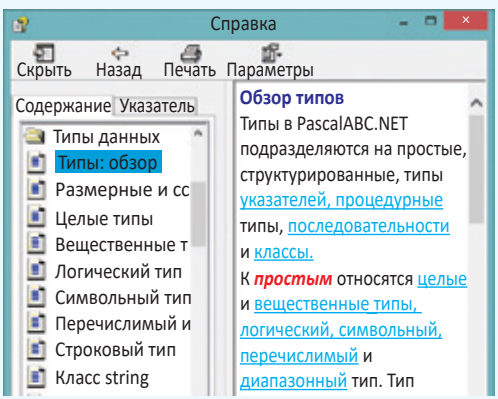
Python:
```
for i in range(n + 1):
    s += i * i
```

C++:
```
for (int i = 1; i <= n; i++)
    s += i * i;
```

Прыклад 2.15. Класіфікацыя даных у мове праграмірования.



Падобная структура тыпаў даных уласцівая шмат якім мовам праграмірования. З іншымі тыпамі даных, якія выкарыстоўваюцца ў мове PascalABC, можна пазнаёміцца ў даведачнай сістэме.



Вылучаюць наступныя аператары: аператар прысвойвання, аператар умоўнага пераходу (галінавання), аператар цыкла (прыклад 2.14), аператар выбару, састаўны аператар, часам выкарыстоўваюць пусты аператар, аператар безумоўнага пераходу і інш.

Усе аператары мовы ў тэксце праграмы аддзяляюцца адзін ад аднаго з дапамогай відавочных ці невідавочных раздзяляльнікаў (у Pascal такім раздзяляльнікам з'яўляецца «;»). Аператары выконваюцца ў тым парадку, у якім яны запісаны ў тэксце праграмы. Парадак выканання аператараў можа быць зменены толькі пры дапамозе кіруючых аператараў: галінавання, цыкла і інш.

Даныя

Большая частка аператараў прызначана для апрацоўкі велічынь. Велічыня характарызуецца тыпам, імем і значэннем. Тыпы даных могуць быць простымі і структуравanымі (прыклад 2.15). Велічыня простага тыпу ў кожны момант мае адно значэнне. Велічыня структураванага тыпу складаецца з велічынь іншых тыпаў. Напрыклад, радок складаецца з сімвалаў, кожны асобны сімвал радка мае сваё значэнне (код). Самым распаўсюджаным структураваным тыпам даных з'яўляецца масіў, з якім вы пазнаёміцеся ў гэтым вучэбным годзе.

Усім аб'ектам у мовах праграмірования (перамennым, функцыям, працэдурам і інш.) даюцца імёны. Імя аб'екта ў праграме называюць **ідэнтыфікатарам** (ад слова «ідэнтыфікаваць»).

Ідэнтыфікатарам з'яўляецца любая канечная паслядоўнасць літар і лічбаў, якая пачынаецца з літары (прыклад 2.16). Імя можа змяшчаць знак падкрэслівання «_». Выкарыстоўваць службовыя словы мовы ў якасці ідэнтыфікатара забараняецца ў большасці моў праграмiравання.

Велічыні могуць быць пастаяннымі (канстанты) і пераменнымі. **Пераменная** можа прымаць некаторае значэнне ў выніку выканання каманды ўводу ці з дапамогай аператара прысвойвання. У ходзе выканання праграмы значэнні пераменнай могуць неаднаразова змяняцца. Пасля апісання пераменная атаясамліваецца з некаторым блокам памяці, змесціва якога з'яўляецца яе значэннем. Пераменная захоўвае значэнне, якое адпавядае яе тыпу (напрыклад, пераменная цэлага тыпу не можа прымаць значэнне рэчаіснага ліку). Гэта значэнне можа здабывацца з памяці для выканання з ім аперацый, якія адпавядаюць тыпу пераменнай.

Падпраграмы

Алгарытм, які рэалізуе рашэнне асобнай часткі асноўнай задачы, называюць **дапаможным**, а яго запіс на мове праграмiравання — **падпраграмай**. Падпраграмы могуць быць рэалізаваны ў выглядзе функцый ці працэдур.

Прыклад 2.16. Імёны пераменных задае праграміст. Існуюць рэкамендацыі, як можна (трэба) называць пераменныя ў кодзе:

- імя пераменнай павінна быць зразумелым, наглядным і адлюстроўваць сутнасць абазначанага аб'екта (sum, number, count_of_positive);
- уводзіць пераменным кароткія імёны (*s*, *i*, *n*) можна ў тым выпадку, калі яны выкарыстоўваюцца ў невялікім фрагменце кода і іх ужыванне відавочна.

Некаторым ідэнтыфікатарам загадзя прадпісаны пэўны сэнс, напрыклад sin, cos, sqrt, abs — імёны матэматычных функцый.

Шмат якія кампаніі распрацоўваюць свае правілы па афармленні кода, дзе прапісаны таксама і правілы называння пераменных. У кампаніі Microsoft выкарыстоўваюць так званую «венгерскую натацыю»[1]. Вядомымі з'яўляюцца таксама:

- «вярблюджая натацыя»;
- «змяіная натацыя»;
- «шашлычная натацыя»[2].

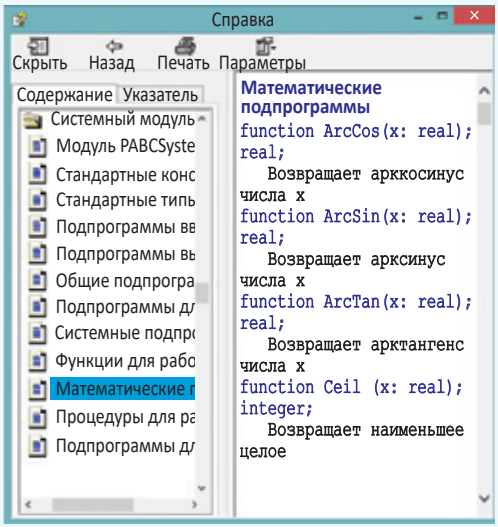
Правілы афармлення кода, распрацаваныя ў некаторых кампаніях, з'яўляюцца адкрытымі і могуць выкарыстоўвацца іншымі распрацоўшчыкамі. Напрыклад, у Google распрацаваны styleguide («гід па стылі») для розных моў праграмiравання (C++, Java, Python, Lisp і інш.)[3].

[1] https://ru.wikipedia.org/wiki/Венгерская_нотация#cite_note-hunganotat-1 (дата доступу: 25.02.2019).

[2] https://ru.hexlet.io/blog/posts/naming-in-programming (дата доступу: 25.02.2019).

[3] http://google.github.io/styleguide/ (дата доступу: 25.02.2019).

Прыклад 2.17. Спіс стандартных функцый мовы праграміравання PascalABC можна паглядзець у даведачнай сістэме:



Прыклад 2.18. Працэдуры ў мовах праграміравання.

У мове VisualBasic працэдуры аб'яўляюцца як sub (скарачэнне ад англ. *subroutine* — падпраграма).

У мове C++ няма асобных канструкцый для апісання працэдур. Іх ролю выконваюць функцыі, якія маюць тып *void* (англ. «пустата»).

На сайце Tiobe[1] штомесяц публікуецца рэйтынг моў праграміравання.

Рэйтынг складаецца на аснове падліку вынікаў пошукавых запытаў, якія змяшчаюць назву мовы, у Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon.

Функцыя апісвае працэс вылічэння пэўнага значэння, залежнага ад некаторых аргументаў, таму для функцыі заўсёды паказваецца тып значэння, якое вяртаецца. Для кожнай мовы высокага ўзроўню распрацавана бібліятэка стандартных функцый: арыфметычных, лагічных, сімвальных і г. д. (прыклад 2.17). Функцыі (як стандартныя, так і тыя, што задаюцца праграмістам) выкарыстоўваюцца ў праграме ў выразах.

Часта выкарыстоўваюць падпраграмы, якія не вяртаюць пэўнае значэнне, а ўяўляюць сабой самастойны этап апрацоўкі даных. У мове Pascal іх называюць **працэдурамі**, у іншых мовах яны могуць называцца па-іншаму ці не мець уласнай назвы і апісвацца гэтак жа, як функцыі (прыклад 2.18).

Мова праграміравання — інструмент для рашэння пэўнай задачы. Не існуе адзінай самай лепшай мовы праграміравання. Для рашэння задач рознага роду і ўзроўню складанасці трэба ўжываць розныя мовы і тэхналогіі праграміравання. У найпрасцейшых выпадках дастаткова асвоіць асновы структурнага напісання праграм, напрыклад на мове PascalABC. Для стварэння ж складаных праектаў патрабуецца не толькі свабодна валодаць якой-небудзь мовай у поўным аб'ёме, але і мець уяўленне пра іншыя мовы і іх магчымасці. Як правіла, чым больш складаная задача, тым больш часу патрабуецца на асваенне інструментаў, неабходных для яе рашэння.

[1] https://www.tiobe.com/tiobe-index/ (дата доступу: 26.06.2019).

1. Для чаго прызначаны транслятар?
2. Якія функцыі выконвае кампілятар? Інтэрпрэтатар?
3. Што вызначаецца парадыгмай праграмiравання?
4. З якіх элементаў можа складацца алфавіт мовы праграмiравання?
5. Што ўяўляе сабой аператар мовы праграмiравання?
6. Якія тыпы даных вам вядомыя?
7. Для чаго выкарыстоўваюцца функцыі і працэдуры?

Практыкаванні

1 Напішыце праграмы для рашэння наступных задач.

1. Вызначыце апошнюю лічбу натуральнага ліку N.
2. Два адрэзкі на плоскасці задаюцца каардынатамі сваіх канцоў. Вызначыце, які з іх карацейшы.
3. Знайдзіце суму $1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots + \frac{1}{N^2}$ для зададзенага N.
4. Уводзіцца радок тэксту. Вызначыце, ці з'яўляецца ён паліндромам.
5. Уводзяцца два цэлыя лікі, якія з'яўляюцца лічнікам і назоўнікам дробу. Скараціце дробь, выведзіце атрыманыя лічнік і назоўнік. Падказка: можна выкарыстаць алгарытм Еўкліда.

6*. Дзед Мазай і заяц гуляюць у вельмі простую гульню. Перад імі — гара з N аднолькавых морквін. Кожны з гульцоў у час свайго ходу можа ўзяць з яе любую колькасць морквін, роўную неадмоўнай ступені ліку 2 (1, 2, 4, 8, ...). Гульцы ходзяць па чарзе. Хто возьме апошнюю морквіну, той і выйграе. Складзіце алгарытм, які пры зададзеным значэнні N вызначае пераможцу ў гэтай гульні. Улічвайце, што кожны з гульцоў хоча выйграць і не робіць лішніх хадоў, г. зн. гуляе аптымальна.

2* Прапанаваныя ніжэй алгарытмы запісаны рознымі спосабамі. Вызначыце, што робіць кожны з прапанаваных алгарытмаў, і рэалізуйце іх на мове Pascal.

1. Алгарытмічная мова

```
ввод a
n := Длина(a)
m := 1
b := Извлечь(a, m)
нц для i от 7 до n
  c := Извлечь(a, i)
  b := Склеить(b, c)
кц
вывод b
```

Для рускага слова «энергетика» праграма выводзіць «этика».

2. Python

```
a = int(input())
k = 0
s = 1
while k < a:
  k = k + 1
  s = s + 1.0/k
print(s)
```

Пры $a = 5$ праграма выводзіць 3.2833333333333337.

3. Basic

```
INPUT X
L = 0
M = 0
WHILE X > 0
   M = M + 1
    IF X MOD 3 <> 0 THEN
     L = L + 1
    END IF
   X = X \ 3
WEND
PRINT L
PRINT M
```

Для значэння 5637 праграма выводзіць 6 і 8.

4. C++

```
int F(int x)
{
   return x*x + 16*x + 15;
}
int main()
{
 int a, b;
 cin >> a >> b;
 int M = 0;
 for (int t = a; t <= b; t++)
  if (F(t) > 0)
    M = M + 1;
 cout << M;
 return 0;
}
```

Пры $a = -3$, $b = 5$ праграма выводзіць 6.

3* Адлюструйце любы алгарытм з практыкавання 2 у выглядзе блок-схемы.

Глава 1
АЛГАРЫТМЫ АПРАЦОЎКІ МАСІВАЎ

§ 3. Структураваны тып даных масіў

Упершыню тып даных масіў з'явіўся ў мове Фартран (створана ў перыяд з 1954 па 1957 г. у карпарацыі IBM). Ужо першыя версіі мовы падтрымлівалі трохмерныя масівы (у 1980 г. максімальная размернасць масіва была павялічана да 7). Масівы былі неабходны для стварэння матэматычных бібліятэк, у прыватнасці тых, што змяшчалі працэдуры рашэння сістэм лінейных ураўненняў.

Прыклад 3.1. У 10 А класе 25 навучэнцаў. Вядомы рост кожнага ў сантыметрах. Для захоўвання значэнняў росту можна выкарыстоўваць масіў A, які складаецца з 25 цэлых лікаў. Індэкс кожнага элемента — парадкавы нумар навучэнца са спіса ў класным журнале. Тады запіс A[5] — рост навучэнца пад пятым нумарам.

3.1. Паняцце масіву

У сучасным свеце штосекундна адбываецца апрацоўка вялізнай колькасці даных з дапамогай камп'ютара. Калі неабходна апрацоўваць даныя аднаго тыпу (лікі, сімвалы, радкі і інш.), то для іх захоўвання можна выкарыстаць тып даных, які называецца **масіў**.

Масіў — упарадкаваная паслядоўнасць даных, якая складаецца з канечнага ліку элементаў, што маюць адзін і той жа тып, і абазначаецца адным імем.

Масіў з'яўляецца структураваным (састаўным) тыпам даных. Гэта азначае, што велічыня, апісаная як масіў,