Turinys

1. pamoka. Algoritmas, jo vykdymas, savybės

Kasdienėje veikloje kiekvienas susiduriame su įvairiomis taisyklėmis, nurodymais, pavyzdžiui: naudojimosi įvairiais įrenginiais ar baldų surinkimo instrukcijomis, patiekalų receptais ir pan.

Yra ir kitokių nurodymų. Pavyzdžiui, draugas kviečia jus į svečius pasidalyti atostogų įspūdžiais. Jis sako: „Išėjęs iš namų pasuk į dešine, paėjėk iki artimiausios autobusų stotelės, įlipk į autobusą Nr. 5, pavažiuok 3 stoteles, išlipk „Žvaigždžių" stotelėje. Ten tave pasitiksiu."

Panašiai taisyklės sudaromos ir matematikos, fizikos, chemijos uždaviniams spręsti. Naudodamiesi jomis, nesunkiai išsprendžiame vienokio ar kitokio tipo uždavinius. Mokydamiesi gimtąją ir užsienio kalbas, išmokstame pagrindines taisykles ir jas taikydami sėkmingai įveikiame gramatikos subtilybes.

Iš pateiktų pavyzdžių matyti, kad taisyklės yra skirtingo pobūdžio, tačiau turi ir bendrų bruožų:

- jas galima aprašyti atskirais aiškiais nurodymais, ką reikia daryti;
- yra pradiniai duomenys (pavyzdžiui: patiekalui pagaminti reikalingi produktai, spintos sudedamosios dalys ir kt);
- gaunamas tam tikras rezultatas (pagaminamas patiekalas, sukonstruojama spinta ir kt.).

Išvardyti bruožai apibūdina algoritmo (lot. *algorithmus* - pagal Vidurinės Azijos matematiko al-Chwarizmi pavardės lotyniškąją formą *Algorithmf)* sąvoką. Šiuo atveju, vadovaudamasis algoritmu, veiksmus atlieka žmogus.

Algoritmu vadinami aiškūs vienareikšmiai nurodymai (sakiniai), kaip turint tam tikrus pradinius duomenis galima gauti reikiamus rezultatus.

Algoritmo sąvoka yra viena iš pagrindinių matematikos ir informatikos sąvokų. Pir mieji algoritmai apibūdino veiksmus, atliekamus su dešimtainės skaičiavimo sistemos skaičiais. Vėliau algoritmo sąvoka pradėta vartoti apibūdinant veiksmų seką, kurią reikia atlikti norint išspręsti uždavinį. Šioje knygoje nagrinėjami matematinio pobūdžio algoritmai. Pateikiame pagrindines jų sąvokas.

Pradinis duomuo - tai iš anksto žinoma reikšmė (paprasčiausiu atveju - skaičius), būtina veiksmams atlikti. Pavyzdžiui, norint apskaičiuoti stačiakampio plotą, būtina žinoti jo ilgį ir plotį.

Rezultatas - tai reikšmė, gauta atlikus visus skaičiavimus.

Tarpinis rezultatas - tai apskaičiuota reikšmė, kuri naudojama tolesniems veiksmams atlikti. Tarpiniai rezultatai padeda programuotojui pasitikrinti, ar parašyta visa programa, ar tik jos dalis, ar programos dalys veikia gerai.

Algoritmu aprašomi veiksmai yra skirti ***vykdytojui.***

Kiekvienam algoritmui būdingos tokios savybės:

Diskretūmas. Algoritmas suskaidomas į baigtinę žingsnių seką. Tik atlikus vieno žingsnio veiksmus

galima pereiti prie kito žingsnio.

Aiškumas. Visus algoritmu aprašomus veiksmus bet kuris vykdytojas turi suprasti vienareikšmiškai. Šioje knygoje pateikiami algoritmai skirti kompiuteriui - *nemąstančiam vykdytojui.* Kad suprastume, ką reiškia ši sąvoka, prisiminkime rašytojo V. Petkevičiaus pasakos vaikams „Sieksnis, Sprindžio vaikas" vieną epizodą. Tėvai, išleisdami sūnų į mokyklą, sako: „Eidamas dairykis. Neskubėk lėkti per kelią, palauk, kol mašina pravažiuos, kad po ratais nepakliūtum." Visą dieną Sieksnio iš mokyklos nesulaukęs tėvas išėjo jo ieškoti ir priėjęs kryžkelę pamatė ilgšį pagriovy stovintį ir garsiai žliumbiantį. Tėvo paklaustas, kas atsitiko, Sieksnis atrėžė: „Sakei, kad neskubėčiau, palaukčiau, kol mašina pravažiuos. Aš visą dieną laukiu, o jos kaip nėr, taip nėr." Ir į mokyklą tą dieną Sieksnis taip ir nenuėjo. Kitą dieną tėvas sūnų pats per kryžkelę pervedė ir paleido: „Žiūrėk man, eik ir nesidairyk!" Sieksnis, tiesiai eidamas, pirmiausia malūną priėjo ir visą dieną malūnininkui talkino. Trečią dieną tėvas pats sūnų į mokyklą nutempė.

Šiame epizode Sieksnį galime laikyti nemąstančiu algoritmo vykdytoju, kuris tiesiogiai, t. y. nemąstydamas ir neanalizuodamas, vykdė tėvo nurodymus.

Rezultatyvumas. Atlikus baigtinį skaičių algoritmo veiksmų, gaunamas rezultatas. Vienas iš galimų rezultatų - uždavinys sprendinių neturi.

Baigtumas. Rezultatas gaunamas įvykdžius baigtinį skaičių algoritmo veiksmų.

Universalumas. Naudojant tą patį algoritmą, sprendžiami visi to tipo uždaviniai, t. y. kiekvienam pradinių duomenų rinkiniui gaunamas teisingas rezultatas.

Yra daug uždavinių, kuriems spręsti nėra tikslių algoritmų. Pavyzdžiui, reikia mašinomis gabenti daug įvairaus dydžio tuščių dėžių, kurias galima dėti vienas į kitas. Mašinų su kroviniu skaičius turi būti kuo mažesnis. Net žmogus, galintis intuityviai spręsti tokį uždavinį, ne iš karto gaus geriausią rezultatą. Jeigu dėžių daug, o j mašinas telpa nevienodas jų skaičius, tuomet neįmanoma perrinkti visų galimų sprendimo variantų ir tenka pasirinkti vieną kurį nors geresnį rezultatą. Tokio tipo uždaviniams spręsti kuriami algoritmai vadinami ***euristiniais.***

Algoritmai gali būti pateikiami skirtingais būdais:

Užrašomi ***žodžiais.*** Šis būdas naudojamas, kai norima labai aiškiai nurodyti atliekamus veiksmus. Užrašomos komandos gali būti numeruojamos arba veiksmai aprašomi kaip pasakojimas.

Vaizduojami ***grafiškai*** - dažniausiai ***blokinėmis (simbolinėmis) schemomis*** arba ***struktūrogramomis.***

Vartojami grafiniai simboliai apibrėžia tam tikro tipo veiksmą. Simbolinėse schemose grafinius simbolius jungiančios linijos rodo, kokia tvarka tie veiksmai atliekami. Sutarta, kad linijos eina iš viršaus į apačią ir iš kairės į dešinę. Visais kitais atvejais linijos gale braižoma rodyklė, nurodanti tolesnių veiksmų kryptį. Struktūrogramoje veiksmų vykdymo tvarka nusakoma grafiniais simboliais.

Užrašomi ***pseudokodu.*** Vartojami žodžiai, artimi natūraliai kalbai. Pseudokodu patogu užrašyti algoritmus, kai norima trumpiau ir suprantamiau atskleisti jų esmę - vadovėliuose, straipsniuose.

Užrašomi ***programavimo kalba.***

2. Pamoka. C kalba

Denis Riči (Dennis Ritchie) 1970-aisiais sukūrė C kalbą. Pradinis jos tikslas buvo rašyti sistemos lygio programas, pavyzdžiui, operacines sistemas, bet tuo pačiu metu tai buvo kalba, kurią galėjo kiekvienas išmokti ir naudoti. Prieš jos atsiradimą operacinės sistemos buvo rašomos asembleriu, o tai buvo labai daug pastangų reikalaujanti užduotis net ir palankiausiomis sąlygomis. Dėl savo pradinės paskirties ji turi daug bendro su asemblerio kalba, kuri, kaip minėjome ankstesniajame skyrelyje, gali būti naudinga, kai iš sistemos stengiamasi išspausti viską, kas įmanoma.

Privalumai

C kalba labiausiai tinka rašant mažas ir labai sparčias programas. Kaip minėta anksčiau, ją labai paprasta susieti su asembleriu. Be to, ji labai standartizuota, taigi platformos pakeitimai nėra tokie pastebimi C kalboje, lyginant ją su kitomis kalbomis. Daug kalbos aspektų yra nepriklausomi nuo platformos, nors iš esmės esate priversti rašyti vartotojo sąsajas kiekvienai platformai, kurią planuojate naudoti. Tai nėra labai sudėtingas procesas, dėl to C kalba dažnai pasirenkama rašant

kelioms platformoms.
Trūkumai
Paprastai C kalbos sintaksei reikia skirti šiek tiek laiko ir tai gali būti ne pats geriausias pradedančiojo programuotojo pasirinkimas. Jame nenaudojama objektinio programavimo technika, o dėl to asmenys, pripratę prie objektinio programavimo (OOP), gali susidurti su sunkumais.
Papildoma informacija
Apie C kalbą prirašyta tiek daug dokumentacijos ir knygų, kad vien tik joms išvardyti prireiktų viso skyriaus. Turėdami tai omenyje, galite peržiūrėti diskusijų grupes ar knygynus internete, pavyzdžiui, „Barnes and Noble" (www.bn.com) ir ten paieškoti populiarių knygų šia tema.
C++ kalba
C + + kalba -- tai objektinio programavimo C kalbos įpėdinė. Nesusipažinusiems su OOP pasakysime, jog tai paprasčiausiai reiškia, kad programos kuriamos iš objektų. Teoriškai šis programavimo tipas leidžia kurti programą panaudojant savas ar kitų sukurtas bibliotekas, susiejant jas taip, kaip to reikia jums. C + + kalba turi daug bibliotekų, kuriose rasite viską— nuo garso iki grafikos ir duomenų bazių. Dažnai tai daug paprastesnis programavimo sprendimas, bet daug žaidimų kūrėjų nenaudoja C++ kalbos, nes ji dažnai padidina žaidimo kodą, dėl ko žaidimas sulėtėja. Akivaizdu, kad ne to tikisi dauguma žaidimų programuotojų.
Kaip minėta anksčiau, tam tikros kalbos ar aplinkos gynėjai greitai nurodys kitų aplinkų trūkumus. Ginčas „C prieš C++", ko gero, ir šiuo metu vyksta diskusijų grupėje ar interneto pokalbių kanale. Paprastai sutiksite asmenų, kurie bus vienoje ar kitoje pusėje, nors retkarčiais sutiksite ir tokių, kurie mėgsta abi kalbas ir žino abiejų jų privalumus bei trūkumus. Nepriklausomi diskusijų nariai paprastai atkreips dėmesį, kad C++ kalbą paprasčiau naudoti ir kad su ja susijęs papildomo kodo kiekis nėra toks didelis, kad dėl jo reikėtų šios kalbos atsisakyti.
Si didelė problema slepia mažesnę: daugelis bando nuspręsti, kurią kalbą jie turėtų išmokti pirmiausiai — C ar C++. Vėlgi, nėra paprasta pasirinkti. Nors C kalbą, ko gero, paprasčiau išmokti, jos mokydamiesi neišmoksite objektinio programavimo, todėl jums teks mokytis naujo programavimo būdo, jei norėsite naudoti vieną naujesnių programavimo kalbų.

3.Pamoka. Privalumai

Dėl to, kad C++ programavimo kalba objektinė, ji daug paprasčiau naudojama, be to, kai kuriems žmonėms patinka tai, kad yra labai daug galimų naudoti bibliotekų. Ko gero, daug paprasčiau tvarkyti programas, parašytas C++ kalba, o ne C, ypač kai jos tampa didelės ir sudėtingos, kokie dažniausiai ir būna žaidimai. Ji šiek tiek lengviau perkeliama nei C kalba. Didžioji dalis komercinių žaidimų yra parašyti C ar C++ kalba. Be to, C++ kalbai egzistuoja begalė IDE, pavyzdžiui, Visual C++ (žr. 6.1 pav.) ir C++ Builder.

6.1 pav. Visual C++ IDE teikia kūrėjams pranašumų.

Trūkumai

Priklausomai nuo situacijos, C++ kalba gali būti lėtesnė už C. Be to, gali prireikti šiek tiek laiko, kol priprasite prie jos, jei nesate susipažinę su struktūriniu programavimu.
Papildoma informacija
Kaip ir C kalbos atveju, yra pernelyg daug šaltinių, kad galėtume apsiriboti sąrašu. Internete bet kuriame knygų pardavimo tinklalapyje peržvelkite sąrašus ir paieškokite knygų apie savo naudojamą C++ kalbos versiją. Be to, pasiieškokite programavimo kursų, kurie tikrai naudingi pradedantiesiems programuotojams.

4. Pamoka. Programavimo koncepcijos ir priemonės

Seniausias tradicijas turi procedūrinio programavimo koncepcija, kurioje vadovaujamasi natūraliu požiūriu, kad programos yra išsamūs kompiuteryje vykdomų procesų aprašymai. Todėl daroma prielaida, kad pagrindinis programavimo tikslas yra kompiuterio valdymui pritaikyta forma parengti ir aprašyti uždavinių sprendimo algoritmus. Tokių aprašymų poreikiams buvo skirta dauguma iki 1980 metų sukurtų programavimo kalbų (Fortan, C, Algol, Pascal ir kitos). Šios priemonės taip pat plačiai naudojamos ir moderniose šiuolaikinėse programavimo kalbose.
Programavimo priemonių efektyvumą ir populiarumą lemia tai, kaip šios priemonės pritaikytos sudėtingiems uždaviniams aprašyti. Sprendžiant šią problemą buvo sukurti struktūrinio programavimo principai, rekomenduojantys, kaip sudėtingų procesų aprašymus skaidyti į mažesnes nepriklausomai aprašomas dalis ir jas sujungti j vientisą programą. Tokios struktūrinės dalys C++ kalboje vadinamos funkcijomis.
Pagalbines funkcijas ypač naudinga išskirti tada, kai jos aprašo daugeliui uždavinių būdingus procesus ir skaičiavimus. Populiariausių tipinių funkcijų aprašymai kaupiami pagalbinių programavimo priemonių bibliotekose, kurios išplečia jas naudojančių programavimo kalbų galimybes ir parengiamos taip, kad jomis būtų galima naudotis nežinant vidinės funkcijų struktūros ir sudarymo principų. Pakanka žinoti tiktai funkcijų paskirtį, darbo rezultatus ir sąsajai su jomis naudojamas priemones.
Taip pat aktualu turėti galimybę patiems susikurti savo poreikiams pritaikytų pagalbinių priemonių rinkinius, kuriuos būtų galima naudoti daugeliui programų kurti. Sprendžiant šią problemą parengti modulinio programavimo ir vidinės pagalbinių priemonių struktūros paslėpimo (encapsulation) principai, sukurta abstrakčių tipų koncepcija. Vadovaujantis ja pagrindinis dėmesys programose skiriamas duomenų tipų aprašymui ir jų naudojimui modeliuojant konkrečių uždavinių sprendimą.
Abstrakčių tipų aprašymai sudaromi iš sąsajos ir jos realizavimo dalių. Sąsajoje nurodoma, kokios yra šių duomenų savybės ir kokius veiksmus jiems galima taikyti, o realizavimo dalyje aprašoma, kaip visos šios savybės realizuojamos. Universalios paskirties duomenų tipus, kurie naudojami sprendžiant daugumą uždavinių ir vadinami elementariais, realizuoja programavimo kalbų kompiliatoriai. Tokių tipų pavyzdžiai yra sveikųjų ir realiųjų skaičių tipai, loginiai duomenys.
Taip pat reikalingos priemonės, kurios leistų vartotojams patiems suprojektuoti siaurai specializuotus, pritaikytus atskirų uždavinių poreikiams, duomenų tipus. Tokių tipų aprašymus pageidautina parengti ir tvarkyti nepriklausomai nuo juos naudojančių programų ir susieti su jomis tiktai kompiliavimo metu. Taip parengtuose aprašymuose paslepiama vidinė duomenų tipų struktūra, jie apsaugomi nuo nepageidautinų pakeitimų, palengvinama programų priežiūra, sudaromos sąlygos modifikuoti aprašymus taip, kad jie tiktų ir senesnes atmainas naudojančioms programoms. Abstraktūs duomenų tipai taip pat yra puiki pagalbinių programavimo priemonių bibliotekų kūrimo priemonė.
Pritaikant anksčiau parengtas abstrakčias struktūras naujų programų poreikiams, dažnai pakanka joms suteikti tiktai keletą, naujų savybių. Naudojant procedūrinio programavimo technologiją, kai atskirai aprašomos duomenų struktūros ir joms taikomos funkcijos, net ir nežymiai pakeitus duomenų struktūras, tenka perrašyti daugumą jas apdorojančių funkcijų. Tokių perrašymų padeda

išvengti objektinio programavimo technologija, kurioje abstraktiems tipams aprašyti naudojamos klasėmis vadinamos sintaksinės struktūros. Naudojant šią technologiją, iš kiekvienos klasės galima sukurti ištisą išvestinių klasių šeimą, paveldinčią joms kurti naudojamos klasės savybes. Visos vienos šeimos klasės taikomus veiksmus aprašančios funkcijos gali būti polimorfinės, prisitaikančios prie atskirų klasių savybių. Galinga objektinių programų kūrimo priemonė yra šabloninės klasės, lengvai pritaikomos įvairių tipų duomenims apdoroti.

Šiuo metu plačiausiai naudojamos procedūrinio ir objektinio programavimo koncepcijomis pagrįstos technologijos ir programavimo kalbos, kurios geriausiai atitinka įvairius praktinio programavimo poreikius bei kompiuterių įrangos suteikiamas galimybes. Atskirose taikomosiose srityse taip pat naudojamos funkcinio ir loginio programavimo technologuos. Jose naudojamos priemonės leidžia parengti programas nesigilinant į tai, kaip jos bus vykdomos kompiuteryje. Todėl programų tekstai būna kompaktiški, gerai atspindi sprendžiamų uždavinių ypatumus, tačiau jų vykdymo efektyvumas nėra didelis.

Pavyzdžiui, funkcinio programavimo kalbose uždaviniams aprašyti naudojamas nedidelis rinkinys elementarių funkcijų, iš kurių konstruojamos sprendžiamų uždavinių poreikiams pritaikytos naujos sudėtinės funkcijos. Nėra tokių procedūriniame ir objektiniame programavime vartojamų sąvokų kaip duomenų tipas, kintamasis ir reikšmės priskyrimo veiksmas. Funkcija suvokiama kaip tam tikru būdu skaičiuojamų reikšmių aibė. Populiariausia funkcinio programavimo kalba yra LISP, naudojama sprendžiant rekursinėmis funkcijomis aprašomus dirbtinio intelekto uždavinius.

Loginio programavimo kalbose uždavinių sprendimui aprašyti naudojamos matematinės logikos priemonės. Programose šiomis priemonėmis aprašomos pradinių duomenų ir pageidaujamų rezultatų savybės, o kaip šie rezultatai turi būti gaunami, privalo rūpintis programavimo kalbos transliatorius. Todėl tokios programavimo kalbos siaurai specializuotos, dažniausiai naudojamos ekspertinėms sistemoms kurti. Plačiausiai naudojama loginio programavimo kalba Prolog.

Savikontrolės klausimai

Apibūdinkite procedūrinio, objektinio, funkcinio ir loginio programavimo koncepcijas.

Iš kokių dalių sudaromi abstrakčių duomenų tipų aprašymai?

Kuo objektinio programavimo technologija pranašesnė už procedūrinį programavimą?

Kokiems uždaviniams aprašyti dažniausiai naudojamos loginio programavimo kalbos?

5. pamoka. Taikomojo ir sisteminio programavimo poreikiai

Kompiuterių programos pagal savo paskirti skirstomos į dvi pagrindines grupes: taikomąsias ir sistemines. Taikomosios programos leidžia vartotojams naudoti kompiuterius sprendžiant jiems aktualius uždavinius, o sisteminėmis programomis vadinamos kompiuterių darbą ir bendros paskirties paslaugas organizuojančios operacijų sistemos bei įvairios naujų programų kūrimo priemonės.

Seniausiai kompiuteriai naudojami moksliniams skaičiavimams, kuriuose sudėtingi duomenų apdorojimo veiksmai taikomi paprastoms duomenų struktūroms. Dažniausiai duomenims aprašyti naudojamos struktūros yra vektoriai ir matricos, kuriems taikomi kartojami (cikliniai), jų elementų analizės, skaičiavimo bei atrankos veiksmai. Atsižvelgiant į šiuos poreikius, buvo kuriamos abstrakčių duomenų apdorojimo procesų aprašymams pritaikytos programavimo kalbos (Fortran, Algol), parengtos pagrindinės procedūrinio programavimo priemonės.

Pradėjus kompiuterius plačiau taikyti verslo informacijai apdoroti ir įvairioms informacinėms sistemoms kurti, tapo reikalingos tekstinių duomenų apdorojimo, įvairių tipų duomenis jungiančių sudėtingų struktūrų tvarkymo, spausdintų ataskaitų ir draugiškos kompiuterio vartotojo sąsajos rengimo priemonės. Jos buvo rengiamos tiek kuriant specializuotas kalbas (Cobol), tiek modifikuojant populiarias universalios paskirties kalbas (Pascal).

Masiškai paplitus kompiuteriams, jų taikymo sritys, įvairių tipų duomenų apdorojimo poreikiai ir reikalavimai vartotojo sąsajai taip pat sparčiai kinta. Tapo aktualūs programų modifikavimo, šabloninio ir komponentinio programavimo, grafinių ir multimedijos sistemų kūrimo klausimai.

Sprendžiant šiuos uždavinius, sukurtos programavimo kalbos Simula ir Smalltalk ir parengta objektinio programavimo technologija. Dabar objektinio programavimo priemonės įdiegtos visose plačiau naudojamose programavimo kalbose (C++, Java, Delphi, Visual Basic, Php ir kitose).
Dauguma programavimo kalbų skirta taikomajam programavimui, todėl ilgą laiką sisteminiam programavimui buvo naudojama tiktai specialiai šiam tikslui pritaikyta Asemblerio kalba. Tai labai galinga programų kūrimo priemonė, nes ja tiesiogiai aprašomi kompiuterio valdomi fiziniai procesai. Tačiau, norint sėkmingai ją naudoti, būtina gerai žinoti tiek pačią Asemblerio kalbą, tiek fizinių procesų organizavimo kompiuteriuose principus. Visa tai reikalauja daug pastangų ir aukštos kvalifikacijos, todėl stengiamasi sukurti tokias programavimo kalbas, kurios tiktų ne tik taikomajam, bet ir sisteminiam programavimui. Pirmoji plačiai paplitusi tokia kalba buvo C.
Savikontrolės klausiniai
Apibūdinkite taikomųjų ir sisteminių programų paskirtis.
Kokios programavimo priemonės naudojamos taikomajam programavimui skirtose progra
mavimo kalbose?
Kas būdinga Asemblerio kalbai?
Kas būdinga C kalbai?

6. Pamoka. C ir C++ kalbos

C kalba sukurta kartu su populiaria operacijų sistema Unix — buvo naudojama ją projektuojant. Pabrėžiant pradinę C kalbos paskirtį, ji dargi buvo vadinama kompiliatorių kompiliatoriumi. Plintant operacinei sistemai Unix, kalbos populiarumas sparčiai augo. Dabar tai pagrindinė profesionalų programavimo kalba, sėkmingai naudojama tiek sisteminei, tiek taikomajai programinei įrangai kurti. Pirmą kartą C kalbą aprašė jos autoriai B. W Karnighanas ir D. M. Ritchie 1978 m. išleistoje knygoje „C programavimo kalba" (The C Programming Language). Nuo 1983 m. kalboje C naudojamos programavimo priemonės reglamentuojamos ANSI (American National Standards Institute) standartais. Kalbos populiarumą ir taikymo galimybes labai padidino firmos Borland International sukurta asmeniniams kompiuteriams skirta integruota programų kūrimo aplinka Turbo C. Dabar C kalbos transliatorius turi kone visos operacijų sistemos.
Skirtingai nuo kitų programavimo kalbų, kalba C nėra skirta vienai konkrečiai programavimo technologijai. Tai programavimo įrankių ir naujų programavimo technologijų vystymo priemonė. Kuriant naujus programavimo įrankius ir naujas programų konstravimo technologijas, bazinės kalbos priemonės papildomos naujomis, kurios kaupiamos pagalbinėse bibliotekose. Kadangi tokioms bibliotekoms sudaryti ir tvarkyti geriausiai tinka objektinio programavimo metodologija, buvo parengtas kalbos variantas, pavadintas C kalba su klasėmis. Jis buvo pradėtas kurti 1980 m firmoje AT&T vadovaujant B. Stroustrupui. Kalbos aprašymas viešai paskelbtas 1983 metais. Tada jai buvo pritaikytas ir pagal C kalbos sintaksę sudarytas pavadinimas C++, kuris rodo, kad tai nauja C reikšmė.
Iš esmės C++ yra nauja programavimo kalba, pritaikyta sudėtingoms programų sistemoms ir instrumentinėms programavimo priemonėms kurti naudojant objektinio programavimo technologiją. Tačiau šioje kalboje išsaugotas glaudus ryšys ir su klasikine C kalba. Naujos C++ versijos parengiamos taip, kad C kalba išliktų jos poaibiu ir būtų galima C kalba sudarytus pagalbinius modulius įtraukti į naujai kuriamas programas.
C++ kalba, kaip ir mūsų bendravimo kalbos, nuolat tobulinama, vystoma. Atskiros kalbos realizacijos, jos dialektai skirti kompiliatorių, kurie pritaiko šia kalba parengtas programas įvairioms operacijų sistemoms ir įvairiems pagalbinių priemonių rinkiniams, poreikiams. Populiariausi yra Borland C++ bei Microsoft C++ kompiliatoriai Windows operacijų sistemai ir AT&T
kompiliatorius operacijų sistemoms Unix ir Linux. Taip pat plačiai naudojami laisvai platinami atvirojo kodo kompiliatoriai, pritaikyti abiejų pagrindinių operacijų sistemų šeimų (Windows ir Unix) poreikiams.

Kiekvienai programavimo kalbai aktuali joje naudojamų priemonių standartizavimo problema. Plačiausiai naudojami C++ kalbos standartai rengiami ir tvirtinami tarptautinėje standartų organizacijoje ISO (International Standards Organization) ir JAV nacionaliniame standartų institute ANSI (American National Standards Institute). Naudojant programose tik standartines priemones, galima pasiekti, kad jos nepriklausytų nuo kompiuterio įrangos, galėtų būti tvarkomos įvairiais kompiliatoriais ir dirbtų aptarnaujamos įvairių operacinių sistemų.
Knygoje išsamiai aptariami programų aprašymo ANSI/ISO C++ standarto priemonėmis klausimai. Nuo 1999 m visi nauji C++ kompiliatoriai palaiko visas šiame standarte numatytas priemones. Tačiau, nagrinėjant atskirus specialius klausimus, pavyzdžiui, grafinės vartotojo sąsajos projektavimą, atsiriboti nuo konkrečių realizacijų savybių ir naudoti vien tik ANSI/ISO priemones sunku ir net netikslinga. Tokie klausimai plačiau aptariami knygos prieduose.
Savikontrolės klausimai
Kodėl C kalba buvo vadinama kompiliatorių kompiliatoriumi?
Kuo skiriasi C++ kalba nuo C kalbos?
Kodėl rengiami programavimo kalbose naudojamų priemonių standartai?
II. C++ laibos leksika
Programavimo kalbos leksika nurodo, iš kokių elementų ir kaip reikia parengti programų tekstą. Pagrindinė leksinė struktūra yra įvairius nurodymus bei veiksmus, kuriuos turi atlikti kompiuteris, aprašantys sakiniai. Sakinių struktūra priklauso nuo jų paskirties. Pavyzdžiui, jie gali aprašyti programos sukuriamus kintamuosius bei jų savybes, konkrečių reikšmių suteikimą kintamiesiems, kreipinius į tipinius skaičiavimus aprašančias funkcijas, veiksmų atlikimo tvarką valdančius operatorius ir panašiai. Paprasti sakiniai vienas nuo kito atskiriami kabliataškiais, o iš kelių figūriniuose skliausteliuose {} įrašytų paprastų sakinių sudaromi sudėtiniai sakiniai, kurie vadinami programos blokais. Su tokiais blokais kompiliatorius elgiasi taip, lyg tai būtų vienas paprastas sakinys. Sakiniai sudaromi iš žodžių ir įvairius veiksmus, žodžių paskirtį bei skyrybą žyminčių simbolių. Žodžiais nurodomi programose naudojamų objektų vardai ir apdorojamų duomenų reikšmės. Vardai sudaromi iš lotyniškos abėcėlės raidžių ir skaitmenų, bet būtinai privalo prasidėti raide. Raide taip pat laikomas nuleisto brūkšnelio simbolis. Žodžių ribas nurodo tarpai ir skyrybos arba veiksmus žymintys simboliai. Vardų rašymo pavyzdžiai:
X, x, s2, suma, Pirma_Suma
Kiekvienoje programavimo kalboje naudojama grupė specialiems žymėjimams skirtų žodžių (vardų), kurie vadinami pagrindiniais, o jei juos kitiems žymėjimo tikslams naudoti draudžiama.

7. Pamoka. Kintamasis, kintamojo reikšmė

Programos atlieka veiksmus su duomenimis (pvz.: sveikaisiais, realiaisiais skaičiais; simboliais; simbolių eilutėmis). Duomenys gali būti pastovūs (pvz., gimimo data) ir kintami (pvz., amžius). Duomenys, kurie atliekant programą nekinta, vadinami ***konstantomis.***

Kintamieji skirti duomenims, kurių reikšmės atliekant programą keičiasi, atmintyje laikyti. Jų ***vardai*** sudaromi iš raidžių ir skaitmenų, tačiau pirmas ženklas turi būti raidė. Parenkant kintamųjų vardus reikia stengtis, kad jie atitiktų duomenų paskirtį. Pavyzdžiui, skaičiuojant trijų skaičių sumą, juos atitinkančius kintamuosius patogu žymėti raidėmis a, b, c, o sumą - vardu suma. Kintamieji gali būti aprašomi bet kurioje programos vietoje, bet prieš naudojant veiksmuose. Dažniausiai jie aprašomi funkcijos main () pradžioje. Kiekvienam kintamajam reikia nurodyti, kokio tipo duomenis jis turi laikyti atmintyje. Duomenų tipas apibrėžia tam tikrą aibę reikšmių ir veiksmų, kuriuos galima atlikti su tomis reikšmėmis.

Lentelėje pateikiami dažniausiai naudojami duomenų tipai:

Tipo pavadinimas	Galimos reikšmės
int	Sveikasis skaičius iš intervalo [-2147483648, 2147483647]
double	Realusis skaičius iš intervalo [-2.23xl(T^{308}, 1 79xl0^{308}]
char	Vienas simbolis, pavyzdžiui: 'A', '8', '+' , arba skaičius iš intervalo [-128, 127]

bool	Loginė reikšmė: true (1) arba falše (0)

Kintamieji aprašomi taip:

```
duomenųTipas kintamojoVardas;
```

Kintamojo duomenų tipą nustato programuotojas. Jeigu yra keletas to paties tipo kintamųjų, tuomet juos galima išvardyti viename sakinyje, atskiriant vardus kableliu.

Kintamųjų aprašymo pavyzdys:

```
double a, b, c, y;    // realiojo tipo kintamieji
int i, k, a2;         // sveikojo tipo kintamieji
char simb;            // simbolinio tipo kintamasis
```

Vykdant programą, kiekvienam kintamajam kompiuterio atmintyje skiriama tiek vietos, kiek reikia nurodyto tipo duomenims laikyti. Kintamiesiems ***reikšmės*** (duomenys) suteikiamos duomenų ***įvedimo*** (skaitymo) ***sakiniais,*** kai jos imamos iš išorinių įrenginių (klaviatūros, diskinių ir kitų ilgalaikės atminties įrenginių), arba ***priskyrimo sakiniais*** programos tekste taip:

```
duomenųTipas kintamojoVardas = pradinėReikšmė;
```

Pavyzdžiui:

```
double pi = 3.1415;
char a = 'A'; int k
= 5 / 3; bool
status = falše; int
k = 5, b = 15;
```

Kintamojo galiojimo pradžia - jo aprašymo vieta. Kintamasis galioja tame programos bloke, kuriame jis yra aprašytas. Programos bloką sudaro programavimo kalbos sakinių seka, parašyta tarp riestinių skliaustų {} . Jis skiriasi nuo sudėtinio sakinio tuo, kad jame yra ir sakiniai, ir kintamųjų aprašai. Blokas dažniausiai naudojamas funkcijoms užrašyti.

Sudėtinis sakinys - tai tarp riestinių skliaustų { } parašyta sakinių seka.

Sudėtinio sakinio pavyzdys:

```
n = n + 1;
suma = suma + n;
```

Aprašant kintamuosius, svarbu nepamiršti taisyklių:

Kintamieji, aprašyti prieš pagrindinę funkciją main (), vadinami ***globaliaisiais.*** Jie galioja visoje programoje.

Kintamieji, aprašyti funkcijoje, vadinami ***lokaliaisiais.*** Jie galioja tik toje funkcijoje, už jos ribų šių kintamųjų reikšmės neišsaugomos.

Jei kintamasis tuo pačiu vardu aprašytas prieš pagrindinę funkciją main () ir funkcijoje, pirmenybė teikiama lokaliajam, t. y. funkcijoje globalusis kintamasis negalioja.

8. Pamoka. Priskyrimo sakinys

Priskyrimo sakinys naudojamas, kai kintamajam reikia suteikti reikšmę programos tekste. Priskyrimo sakinio struktūra tokia:

```
kintamojoVardas = Reiškinys;
```

Čia lygybės ženklas žymi priskyrimo operaciją, o Reiškinys nurodo, kokius veiksmus, kokia tvarka ir su kokiais argumentais reikia atlikti. Kairiojoje priskyrimo operacijos pusėje įrašytas kintamojo vardas nurodo, kam atmintyje suteikiama apskaičiuota reiškinio reikšmė. Pavyzdžiui, priskyrimo sakinys y = x * x; reiškia, kad argumento x reikšmė keliama kvadratu ir rezultatas priskiriamas kintamajam y.

Būtina, kad priskyrimo operacijos kairiojoje pusėje nurodyto kintamojo tipas atitiktų reiškinio, esančio deši-

niojoje pusėje, reikšmės tipą.

Reiškiniuose galima naudoti tik tokius kintamuosius, kurių reikšmės apibrėžtos anksčiau įrašytuose programos sakiniuose.

Skiriami trys reiškinių tipai:

Y ***Aritmetinis.*** Tai reiškinys, kuriame kintamieji ir konstantos jungiamos aritmetinių operacijų ženklais (+, -, *, /, sveikųjų skaičių dalmens liekana %). Tokio reiškinio skaičiavimo rezultatas yra skaičius. Pavyzdžiui:

```
a = 5; b = 7; c = 9; s
= a * b - c;
```

Rezultato tipas priklauso nuo jį sudarančių argumentų tipų ir operacijų, atliekamų su argumentais. Jei visi argumentai yra sveikojo tipo, tai visų operacijų, atliekamų su šiais argumentais, rezultatas yra sveikojo tipo. Tuomet dalybos operacijos (ji žymima pasviruoju brūkšniu /) rezultatas visada yra sveikoji dalybos dalis.

Schemoje pavaizduoti dviejų tipų argumentai, su jais atliekamos operacijos ir rezultatas (l žymi sveikąjį duomenų tipą, R - realųjį).

I	+ * / %	I	=	I
R	+ * /	I	=	R

Santykio. Tai du aritmetiniai reiškiniai, tarp kurių rašomas santykio operacijos ženklas (>,<,<=,>=, ! =, ==). Tokio reiškinio rezultatas yra loginė reikšmė true (tiesa) arba falše (netiesa). Pavyzdžiui,

```
k = 5 != 7 -- 2;
```

Atlikus šį priskyrimo sakinį, kintamojo k reikšmė lygi falše, nes reiškinys 5 *į 5 yra* neteisingas. V ***Loginis.*** Tai reiškinys, kuriame naudojami loginių operacijų ženklai. C++ jie žymimi ženklų grupėmis && (IR), | | (ARBA), ! (NE). Tokio reiškinio rezultatas yra loginė reikšmė true arba falše. Pavyzdžiui:

```
a = true; b = falše; Kintamųjų
```

a ir b reikšmes įrašę į reiškinį

a = !a M b;

gausime

```
a = ! true | | falše
a = falše || falše
a = falše
```

Pateikiame loginių reiškinių teisingumo lentele.

Kintamųjų reikšmės		Loginių reiškinių rezultatai		
a	b	a && b	a ll b	! a
true	true	true	true	falše
true	falše	falše	true	falše
falše	true	falše	true	true
falše	falše	falše	falše	true

Priskyrimo sakinio rašymo forma tokia pat, kaip ir lygybės išraiškos, tačiau šių struktūrų prasmė iš esmės

skiriasi. Lygybėmis patvirtinamas tam tikras faktas, o priskyrimo sakiniai aprašo procesus, kuriems būdinga reikšmių kaita. Taigi taisyklingas ir toks sakinys:

```
a = a + 1;
```

Jis nurodo, kad senoji kintamojo a reikšmė turi būti padidinta vienetu, o rezultatas vėl pavadintas a. Šį veiksmą galima aprašyti lygtimi

(nauja) (sena)

Pavyzdžiui, atlikus sakinių seką

```
a = 5;
a = a + 1;
```

kintamojo a reikšmė lygi 6.

9. Pamoka. Programos sudarymas

Pradėję spręsti uždavinį pirmiausia turime suvokti, ko iš mūsų norima. Tam reikia išsiaiškinti uždavinio formuluotę ir nustatyti, kokie gali būti programos pradiniai duomenys. Tuomet prognozuojame, kokie turi būti rezultatai.
Programų sudarymas - kūrybinis procesas, reikalaujantis daug triūso ir nuovokos. Tam pačiam uždaviniui spręsti galima sudaryti daug ekvivalenčių programų. Nėra bendrų receptų, kaip sudaryti kiekvieno uždavinio programą. Tačiau galima suformuluoti bendras taisykles, kurios padėtų šį darbą paspartinti, geriau jį atlikti.
Beveik kiekvieną didesnį darbą lengviau įveikti išskaidžius jį į dalis - mažesnius darbus. Tai tinka ir programoms.
Programos sudarymą galima suskirstyti į šitokias dalis - etapus:
uždavinio formulavimas,
pradinių duomenų analizė,
sprendimo metodo (algoritmo) parinkimas ar sudarymas,
programos rašymas,
programos tikrinimas - elementarių klaidų paieška,
programos tobulinimas,
programos derinimas - kontrolinių duomenų išsamaus rinkinio sudarymas,
programos testavimas,
programos bandymas,
10) programos dokumentavimas ir diegimas.
Vieniems uždaviniams svarbesni vieni etapai, kitiems - kiti, kartais kai kurie etapai visai nereikalingi. Vadovėlyje pateikiamų programų sudarymo detaliai neaprašinėsime. Tai užimtų nemažai vietos ir būtų nuobodu skaityti pasikartojančius aiškinimus. Todėl dažniausiai pateiksime paruoštas programas, apibūdinsime kontrolinius duomenis, pavaizduosime rezultatus. O jums teks patiems nagrinėti ir suvokti, kaip ir kodėl šitaip rašoma.
Didelis uždavinys skaidomas į dalis, kurias galima programuoti atskirai. Dažnai didesnei daliai rašoma funkcija arba procedūra. Jei pastaroji vis dar sudėtinga, vėl skaidoma į dalis.
Sudėtingo uždavinio programavimo technologija skaidant jį į mažesnes dalis pademonstruota aštuntajame ir devintajame skyriuose. Mažų, pratimo pobūdžio uždavinių programų dažnai nebūtina skaidyti į dalis. Tačiau sudarant bet kurio uždavinio programą tenka apgalvoti duomenų įvedimą ir rezultatų išvedimą. Šios dalys būdingos kiekvienai programai (jei duomenų daug, tuomet duomenims įvesti ir išvesti geriausia parašyti atskiras procedūras).
Reikia pagalvoti apie programos vaizdumą, patogumą vartotojui, dialogą. Todėl kiekviena programa galėtų turėti savo darbo pradžią ir pabaigą.
Pradžioje būtų galima pristatyti, kokia ši programa, kam ji skirta, kas jos autorius. Ypač reikėtų atkreipti dėmesį į pradinių duomenų nusakymą - kokius reikalavimus jie turi tenkinti.
Programos pabaigoje galima nurodyti, kur ieškoti rezultatų (pavyzdžiui, jei jie pateikiami byloje).

Ypač daug dėmesio reikėtų skirti rezultatų pateikimo aiškumui (kai duomenys rašomi ekrane). Programa gali pranešti, kada baigia darbą ir atsisveikina su vartotoju.
Vadovėlyje pateikiamų pratimų programose paprastai nėra pradžios ir pabaigos veiksmų, išskyrus pradinių duomenų įvedimą ir rezultatų įforminimą. Šie veiksmai programuojami nesunkiai, kiekvienas gali tai padaryti savaip.

10. Pamoka. Blokinė schema.

blokinė schema

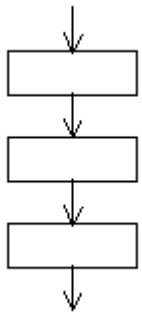
1. Kompiuterio arba kitos sistemos schema, kurioje blokai su pavadinimais vaizduoja pagrindines sudedamąsias dalis, o tarp jų esančios linijos ir rodyklės rodo ryšius tarp dalių.

2. Programavime – algoritmo grafinis vaizdavimas, kai veiksmai rašomi į geometrines figūras, o jų vykdymo eilė nurodoma tas figūras jungiančiomis rodyklėmis.

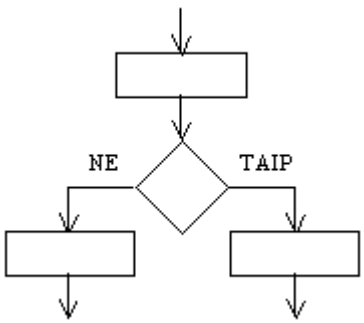
Vartojami ovalai (paprastai reiškiantys veiksmų pradžią ir pabaigą), stačiakampiai (nuosekliems veiksmams aprašyti), rombai (pasirenkamiems veiksmams išreikšti). Šios figūros būna įtrauktos į tekstų rengyklėse esančius autofigūrų rinkinius.

Pagrindiniai algoritmų veiksmai išreiškiami šitokiomis blokinėmis struktūromis:

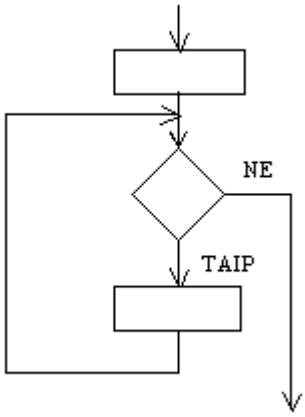
a) nuosekli veiksmų seka:



b) veiksmų pasirinkimas (šakojimas):



c) veiksmų kartojimas:



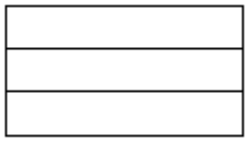
11. Pamoka. Struktūrograma

struktūrograma

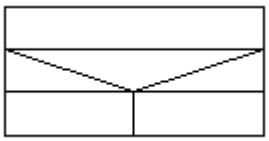
Grafinis algoritmų vaizdavimo būdas. Nuo blokinės schemos skiriasi tuo, kad labiau struktūrizuoja algoritmą.

Pagrindiniai algoritmų veiksmai išreiškimi tokiomis struktūromis:

a) nuosekli veiksmų seka:



b) veiksmų pasirinkimas (šakojimasis):



c) veiksmų kartojimas: