Vilniaus universitetas

Informatikos ir informatinio mąstymo mokomoji veikla

Duomenų struktūros

Mokyklos pedagogika

Kuriame Lietuvos ateitį
2014–2020 metų Europos Sąjungos fondų investicijų veiksmų programa

Vilniaus universitetas

Informatikos ir informatinio mąstymo mokomosios veiklos sukurtos įgyvendinant projektą „Aukštųjų mokyklų tinklo optimizavimas ir studijų kokybės gerinimas Šiaulių universitetą prijungiant prie Vilniaus universiteto“ (Nr. 09.3.1-ESFA-V-738-03-0001), finansuojamą iš Europos socialinio fondo lėšų pagal 2014–2020 metų Europos Sąjungos fondų investicijų veiksmų programos 9 prioriteto „Visuomenės švietimas ir žmogiškųjų išteklių potencialo didinimas“ įgyvendinimo priemonę Nr. 09.3.1-ESFA-V-738 „Aukštųjų mokyklų tinklo tobulinimas“.

Metodinė medžiaga „Duomenų struktūros“ skirta Mokyklos pedagogikos studijų programos moduliui „Informatikos didaktika“. Tikslinė grupė – būsimi pagrindinio ugdymo informatikos mokytojai. Medžiaga siejasi su informatikos ir matematikos Bendrosiomis programomis, pateikiamas teorinis temos pagrindimas mokytojui, aptariamos pagrindinės srities sąvokos. Supažindinama su algoritmavimo ir programavimo praktikoje sutinkamomis duomenų struktūromis ir jų ypatumais.

Šios veiklos autorė Sigita Turskienė

Redagavo: Viktoras Dagys

Vilnius, 2022

Tikslai

- Supažindinti mokytojus su algoritmavimo ir programavimo praktikoje sutinkamais pagrindiniais duomenų tipais bei duomenų struktūromis.
- Padėti suprasti, kaip įvairios duomenų struktūros gali būti efektyviai naudojamos duomenims saugoti.
- Aptarti realias situacijas, kuriose galima panaudoti vieną ar kitą duomenų struktūrą.

Pabandysime atsakyti į klausimus:

- Kokių tipų duomenis gali apdoroti kompiuteris?
- Kaip galima juos sujungti į sudėtingesnes duomenų struktūras?
- Kiek vietos reikia įvairių duomenų tipų duomenims saugoti kompiuterio atmintinėje?
- Kokios duomenų struktūros dažniausiai yra sutinkamos praktiniuose uždaviniuose?

Ryšys su bendrosiomis programomis

Informatika:

Algoritmai ir programavimas:

B2. Naudojasi algoritmavimo, programavimo kalbos konstrukcijomis, programavimo aplinkomis. (Informatikos bendrosios programos projektas, https://www.emokykla.lt/bendrasis/bendrosios-programos/bendrojo-ugdymo-programu-projektai-derinami)

B2.3. Kuria ir (ar) naudoja programai reikalingas duomenų struktūras (taip pat ir abstrakčiąsias). Naudojasi įvairiais duomenų šaltiniais (pvz. tekstiniais failais, jutikliais, internetu). (Informatikos bendrosios programos projektas, https://www.emokykla.lt/bendrasis/bendrosios-programos/bendrojo-ugdymo-programu-projektai-derinami)

Matematika:

C. Problemų sprendimas. C1. Modeliuoja įvairaus konteksto suprantamas ir prasmingas situacijas: skaido problemą į dalis, nustato jų tarpusavio santykį, suformuluoja matematinį klausimą/užduotį. (Matematikos bendrosios programos projektas, https://www.emokykla.lt/upload/EMOKYKLA/BP/PDF/matematika/Matematikos%20BP%20projektas%202021-03-31.pdf)

6.12.1.7. Skaičių sekos:

> https://www.emokykla.lt/upload/EMOKYKLA/BP/PDF/matematika/Matematikos%20BP%20projektas%202021-03-31.pdf

6.8.1.1. Realieji skaičiai:

> https://www.emokykla.lt/upload/EMOKYKLA/BP/PDF/matematika/Matematikos%20BP%20projektas%202021-03-31.pdf

Ugdomi įgūdžiai

Gebėti parinkti ir sukurti tinkamas duomenų struktūras sprendžiant konkrečius uždavinius.

Teorinis pagrindimas

1. Įvadas

Viena iš kompiuterio programų paskirčių yra išsaugoti ir pateikti **duomenis** kompiuterio vartotojui kaip galima patogesne forma.

Kas yra duomenys?

Duomenys – tai sąvoka, kuri siejama su tuo, kas duota, kuo remiantis daromos išvados arba priimami sprendimai.

Kiekvienos kompiuterio programos veikimo rezultatas irgi yra duomenys. Praktikoje yra sutinkami įvairios prigimties duomenys: skaičiai, simboliai, lentelės, sekos, tekstai, garsai, paveikslėliai ir t. t. Natūralu, kad duomenis, skirtus apdoroti kompiuteriu, patogu suskirstyti į tam tikrus tipus pagal jų savybes. Pavyzdžiui, visi realieji skaičiai priklauso realiųjų skaičių tipui, simboliai priklauso simboliniam tipui, visi sveikieji skaičiai priklauso sveikųjų skaičių tipui ir t. t.

Toliau pakalbėsime apie duomenų tipus, sutinkamus programavimo kalbose.

2. Duomenų tipai

Spręsdami uždavinius pastebime, kad kiekvienas duomenų tipas apibūdina dvi aibes:

1) galimų duomenų reikšmių,
2) operacijų su to tipo duomenimis.

Taigi šios dvi aibės nusako dvejopą duomenų tipo paskirtį bei panaudojimą.

Pagrindinė duomenų tipo paskirtis yra informuoti programavimo kalbos transliatorių, kiek vietos jis turi paskirti kompiuterio atmintinėje kintamojo reikšmei saugoti. Nuo galimų reikšmių aibės dydžio (t. y. aibei priklausančių elementų skaičiaus) priklauso duomeniui reikalingos vietos kiekis kompiuterio atmintinėje (žr. 2 lentelę). Pirmųjų kompiuterių atmintinė buvo labai maža. Todėl programuotojai turėjo ją labai taupyti spręsdami didesnės apimties uždavinius.

Su skirtingų tipų duomenimis atliekamos skirtingos operacijos. Su kiekvieno duomenų tipo reikšmėmis galima atlikti tik tam tikras operacijas, pavyzdžiui, su sveikaisiais skaičiais atliekamos aritmetinės (sudėtis, atimtis, daugyba, dalyba) ir lyginimo operacijos. Dar gali būti dalmens radimo ir liekanos radimo operacijos.

Kai žinomas duomenų tipas, galima patikrinti, ar su duotais duomenimis bus galima atlikti norimas operacijas. Tai įgalina išvengti daugelio klaidų kuriamoje programoje.

2.1. Duomenų grupės

Pagal **reikšmių struktūrinimo laipsnį** duomenų tipai, pavyzdžiui, *C++* kalboje skirstomi į dvi klases:

1) bazinius (paprastuosius),
2) išvestinius (struktūrinius).

Paprastieji duomenų tipai yra kelių tipų:

- diskretieji (sveikasis, loginis, simbolinis);
- tolydusis (realusis tipas);
- tuščiasis (*void*).

Apie paprastuosius duomenų tipus

Paprastieji duomenų tipai – tai duomenys, kurių reikšmės yra atskiros, pavienės. Šie tipai turi bazinius vardus ir jų reikšmės yra nedalomos.

Pavyzdžiui, *C* programavimo kalboje yra šie paprastieji tipai: *int*, *long*, *short*, *char*, *double*, *float*, *void*.

Siekiant geriau išnaudoti kompiuterių, kuriems sudaroma programa, aparatūrines savybes, dauguma paprastųjų duomenų tipų turi vieną ar kelias modifikacijas. Pavyzdžiui, *C++* kalboje sveikųjų skaičių tipo modifikacijos yra: *short*, *long*, *unsigned*, *signed*.

Jei duomens tipo modifikacija naudojama be paprastojo tipo, tai standartiškai yra *int* tipas. Konkrečios modifikacijos savybės ir užimamas dydis priklauso pavyzdžiui, nuo *C++* programavimo kalbos kompiliatoriaus. Labai dideliems skaičiams su ženklu vaizduoti naudojamas *long* tipas.

Sveikiesiems teigiamiems skaičiams skirtas beženklis tipas *unsigned*. Beženklis (*unsigned*) tipas leidžia dvigubai padidinti galimų teigiamų sveikųjų skaičių reikšmių intervalą. Pavyzdžiui, *unsigned short* tipo reikšmių intervalas yra [0, 65535], o *short* tipo galimų reikšmių intervalas yra [–32768, 32767]. Beženkliai sveikieji skaičiai naudojami loginiams dydžiams, bitų masyvams ir kitokiems panašaus pobūdžio duomenims laikyti. Beženklių skaičių modifikacijos gali būti: *unsigned char*, *unsigned short*, *unsigned int*, *unsigned long*.

1 lentelėje pateiktos *Go* (Golang) programavimo kalbos sveikojo tipo modifikacijos ir jų įgyjamos reikšmės.

1 lentelė. *Go* programavimo kalbos sveikojo tipo modifikacijos

Duomenų tipas	**Galimos reikšmės**
uint8	0 iki 255
uint16	0 iki 65535
uint32	0 iki 4294967295
uint64	0 iki 18446744073709551615
int8	-128 iki 127
int16	-32768 iki 32767
int32	-2147483648 iki 2147483647
int64	-9223372036854775808 iki 9223372036854775807

Baziniai duomenų tipai sutinkami visose programavimo kalbose. Dažniausiai tai būna:

simbolinio, sveikojo, realiojo, loginio, vardinio, atkarpos tipai, kartais tam priskiriamas ir rodyklės (nuorodos) tipas.

2 lentelė. *Microsoft Visual Studio* baziniai duomenų tipai ir jų ribinės reikšmės

Tipas	***Matmuo (baitais)***	***Reikšmių sritis***
bool	1 baitas	true (1) arba false (0)
char	1 baitas	nuo –128 iki 127
unsigned char	1 baitas	nuo 0 iki 255
int	4 baitai	nuo –2147483648 iki 2147483647
short	2 baitai	nuo –32768 iki 32767
long	4 baitai	nuo –2147483647 iki 2147483647
unsigned long	4 baitai	nuo 0 iki 4294967295
unsigned short	2 baitai	nuo 0 iki 65535
unsigned int	4 baitai	nuo 0 iki 4294967295
float	4 baitai	nuo -1.18×10^{-38} iki $3.4 \times 3 \times 10^{38}$
double	8 baitai	nuo $-2.2.3 \times 10^{-308}$ iki 1.79×10^{308}

Skirtingi duomenų tipai yra reikalingi atvaizduoti **skirtingo dydžio skaičiams** ir **skirtingo tikslumo skaičiavimams.** Skaičiavimuose yra naudojami įvairaus tikslumo duomenys. Didesniems ir tikslesniems skaičiams reikia daugiau vietos atmintinėje, jiems apdoroti reikia ir daugiau procesoriaus laiko. Duomenų tipo užimamas atmintinės dydis yra matuojamas **baitais** arba **bitais** (2 lentelė).

Reikia atsiminti, kad duomenų tipui atmintinė neišskiriama, o išskiriama tik to tipo kintamajam ar konstantai. Kartu šis dydis nusako atitinkamo tipo reikšmių kitimo ribas (žr. 1 lentelę).

Kiekviena konkreti programavimo kalbos realizacija apibrėžia savus apribojimus ribinėms konstantų reikšmėms, todėl 2 lentelėje nurodytos reikšmės konkrečiai kompiuterio operacinei sistemai gali skirtis.

2.2. Duomenų tipų įvairovė programavimo kalbose

Specializuotose programavimo kalbose, pavyzdžiui, *Mathematica* ir *Maple* yra integruotas racionalusis duomenų tipas, skirtas pateikti racionaliuosius skaičius be apvalinimo, pavyzdžiui, 1/5 ir -11/15, ir atlikti su jais aritmetinius veiksmus. Racionalųjį tipą turi ir *Julia*, *Haskell* programavimo kalbos.

Ada programavimo kalboje yra vardinis tipas, kuris naudojamas naujiems diskretiesiems duomenų tipams kurti ir žymimas *type*. *Ada* kalboje yra 2 rūšių realiųjų duomenų tipai:

1) slankaus kablelio (*float*) (tikslumui nurodyti naudojamas požymis *digits*),
2) fiksuoto kablelio (*fixed*) (skaitmenų kiekiui po kablelio nurodyti požymis *delta*).

Groovy programavimo kalboje yra vardinį duomenų tipą primenanti konstrukcija *enum*, savo esme labiau panaši į *C/C++* analogą nei į *Ada type*.

Loginis duomenų tipas programavimo kalbose yra įvardijamas įvairiai, pavyzdžiui, žodeliu *bool Rust*, *C++*, *Go* programavimo kalbose ar žodeliu *boolean Pascal* programavimo kalboje. Pastebėsime, kad *C* kalba neturi loginio duomens tipo, bet *C++* kalba jau turi loginį duomenų tipą.

Rodyklės tipas leidžia konstruoti sudėtingas dinamines duomenų struktūras iš paprastųjų ir struktūrinių tipų.

Programavimo kalbose tekstams vaizduoti ir apdoroti yra numatyti du duomenų tipai:

- simbolinis tipas (*char*) apibrėžiantis vieno simbolio duomenį;
- simbolių eilutės (*string*), su kuriomis patogiau dirbti nei su pavienėmis teksto raidėmis. Galimi raidžių junginiai: žodžiai, sakiniai ar tiesiog simbolių rinkiniai. Jos gali saugoti daug simbolių. Priklausomai nuo programavimo kalbos, eilutėms gali būti taikomi ilgio limitai, taip pat nustačius vieną reikšmę, vėliau gali būti neįmanoma jos pakeisti ilgesne. Dauguma modernių programavimo kalbų šių apribojimų neturi.

C kalboje yra specialus duomenų tipas *void*, kuris naudojamas pažymėti, kai:

- funkcija neturi parametrų funkcijos apraše,
- funkcija nepateikia rezultato.

Sudėtingesniuose uždaviniuose ne visada patogu naudoti ilgus paprastųjų duomenų tipų vardus. Pavyzdžiui, *C++* kalboje *unsigned short int*. Operatoriumi *typedef* galima duomenų tipą pervardinti trumpesniu vardu. Operatoriaus *typedef* sintaksė yra tokia:

typedef kintamojoTipas naujasTipoVardas

Konstrukcija *typedef* naujo duomenų tipo nesukuria, bet suteikia naują vardą jau egzistuojančiam duomenų tipui. Reikia prisiminti, kad naujas duomenų tipo vardas turi būti ne tik patogus, bet ir prasmingas.

Duomenų tipų pervardinimo pavyzdžiai *C* kalboje:

1. // Pervardinamas *unsigned short int* tipas:
   typedef unsigned short int bezenklis;
   bezenklis x;

2. // Pervardinamas *float* tipas:
   typedef float realus;
   realus y[10];

Beveik visose programavimo kalbose yra galimybė transformuoti duomenis iš vieno duomenų tipo į kitą tipą.

Ne visi duomenų tipai vienodai rekomenduotini vartoti. Tai lemia ne tik jų panaudojimo galimybės (pavyzdžiui, vardinis duomenų tipas), bet ir jų sudėtingumas, galimybės tarpusavyje sąveikauti, paprastumas. Nors, pavyzdžiui, daug yra funkcijų darbui su simbolių eilutėmis (*char* tipo), paveldėtomis iš *C* kalbos, tačiau veiksmai su jomis sudėtingi ir nėra patogūs. Todėl patogiau dirbti su *C++* kalbos *string* klase, skirta darbui su simbolių eilutėmis.

3. Struktūriniai duomenų tipai

Pradžioje aptarsime, kodėl programuojant nepakanka paprastųjų duomenų tipų ir tenka naudoti sudėtingesnius duomenų tipus, vadinamus **struktūriniais duomenų tipais** arba trumpiau – **duomenų struktūromis**.

Daugeliu praktinių atvejų kompiuterinės programos konstrukcija reikalauja duomenų tipų, kurių neįmanoma išreikšti programavimo kalbos paprastaisiais duomenų tipais. Praktikoje sutinkame daug uždavinių, kurių duomenys yra pateikiami įvairiais rinkiniais.

Pavyzdžiui,

- teksto eilutė yra grupė ASCII simbolių, kur kiekvienas simbolis yra koduojamas vienu baitu;

- kompleksinis skaičius sudarytas iš poros realiųjų skaičių, kurių vienas atitinka realiąją dalį, kitas – menamąją;
- duomenys banke apie klientus pateikiami lentele.

Visuose minėtuose pavyzdžiuose formuojami duomenų rinkiniai.

Šiuolaikiniame pasaulyje susiduriame su įvairaus sudėtingumo užduotimis. Vienos iš jų yra lengvai išsprendžiamos, kitų sprendimui reikia daugiau laiko. Kai kurias užduotis galima spręsti išvardinant visus galimus sprendimo variantus, po to parengti geriausią sprendimą. Todėl programuotojams labai svarbu žinoti, kaip greičiau, sugaištant mažiau laiko, išrašyti visus sprendimo variantus.

Elementų išvardijimas turi būti sisteminis, nei vienas elementas negali būti pamestas arba pakartotas kelis kartus, aiškiai išdėstytas. Turi būti paprasta ir kitiems programuotojams, ir mėgėjams suprasti, kur turi būti kiekviena kito elemento parinktis. Tai yra svarbu ir dėl galimybės automatizuoti procesą.

Paprastieji kintamieji aprašo pavienes objektų savybes. Praktikoje yra uždavinių, kai būtina aprašyti savybių grupę. Reikalinga vardų sisteminimo ir duomenų grupavimo priemonė. Turi būti galimybė duomenų rinkinį peržiūrėti kelis kartus.

Kaip spręsti šią problemą?

Tokiais atvejais patogu turėti tam tikro tipo kintamąjį,

1) kurio viduje būtų keli kintamieji;
2) kuris išoriškai būtų formuojamas kaip vienas kintamasis.

Daugelis objektų yra išreiškiami (ar apibūdinami) kaip tam tikru būdu sudaryta dydžių visuma. Pavyzdžiui, taško padėtį erdvėje nusako trys koordinatės, tiesinių lygčių sistemą – koeficientų matrica, šachmatų langelį – raide ir skaičius. Tokie sudėtiniai dydžiai informatikoje vadinami **struktūriniais duomenimis**. Aukštosios matematikos kurse jie vadinami vektoriais ir matricomis, o programavime – masyvais.

Aprašydami uždaviniuose įvairius taikomųjų sričių objektus, dažnai juos apibūdiname ne pavieniais parametrais, o jų grupėmis, rinkiniais. Rinkinius vaizduoti skaliariniais tipais yra nepatogu, patogiau specialiomis duomenų rinkinių vaizdavimo priemonėmis – duomenų struktūromis.

Kompiuterių moksle duomenų struktūra – tai būdas laikyti ar grupuoti duomenis taip, kad darbas su jais būtų kiek įmanoma efektyvesnis.

Duomenų struktūra – duomenys, logiškai jungiantys keletą paprastųjų duomenų tipų (reikšmių) arba kelias paprastesnes duomenų struktūras.

https://lt.wikipedia.org/wiki/Duomen%C5%B3_strukt%C5%ABra

Struktūriniams duomenims yra taikomos specialios apdorojimo operacijos ir algoritmai, kurie palengvina įvairių praktinių uždavinių sprendimą.

Duomenų struktūrizavimas prasidėjo atsiradus vadinamosioms aukšto lygio programavimo kalboms. Duomenų tipizavimas pirmą kartą įvestas *Fortran* kalboje (1953). *Algol-60* kalboje buvo išplėstas masyvo panaudojimas neribojant indeksų ir matmenų. *Cobol* kalboje (1961) įvesti tipai simbolių eilutei, įrašui, failui saugoti. *Simula-67* buvo pirmoji kalba, kurioje įvestas tipas *klasė*. Vėliau *Pascal* kalboje taip pat įvestas tipų sisteminimas.

Taigi į duomenų struktūrą galima žiūrėti kaip sudėtinį duomenų tipą, sudarytą iš kitų duomenų reikšmių, logiškai sujungtų kuriuo nors būdu į visumą. Galima jungti keletą paprastų duomenų tipų arba kelias paprastesnes duomenų struktūras.

Duomenų struktūros tarpusavyje skiriasi jas sudarančiomis duomenų reikšmėmis ir jų sujungimo būdu.

Sveikieji ir realieji skaičiai, saugomi kompiuterio atmintinėje taip pat gali būti traktuojami kaip paprastosios duomenų struktūros. Surūšiuotas kintamųjų sąrašas – tipinis struktūrizuotų duomenų pavyzdys.

Konkrečios duomenų struktūros, dažnai sutinkamos programavimo kalbose: **masyvas** ir **įrašas**, rečiau naudojamos – **failas**, **seka**, **aibė**, **sąrašas**. Didesnės apimties problemoms spręsti naudojamos rekursinės duomenų struktūros: **eilė**, **dėklas** *(stekas)*, **medis.**

Duomenų struktūrų yra labai įvairių. Vienos iš jų geriau tinka vieniems atvejams, kitos kitiems. Yra duomenų struktūrų hierarchija – iš paprastųjų duomenų reikšmių sudaromos struktūrinės reikšmės, iš šių – sudėtingesnės struktūrinės reikšmės ir t. t.

Struktūrinius duomenų tipus kuria programuotojas. Struktūrinių duomenų tipų pavyzdžiai *C++* kalboje yra:

- masyvas [ ],
- struktūra (*struct*),
- junginys (*union*),
- klasė (*class*).

Kiekviena programavimo kalba turi savą **duomenų tipų ir struktūrų sistemą.** Jų skirtumai priklauso nuo programavimo kalbos paskirties.

Specializuotos programavimo kalbos *Matlab* duomenų tipų hierarchijos viršuje yra masyvo (*array*) tipas, nes kalba skirta sudėtingiems skaičiavimams atlikti. Sveikojo tipo duomenims laikyti gali būti skiriami vienas, du, keturi ar aštuoni baitai. Vieno ar kito sveikojo tipo didžiausią ar mažiausią reikšmę galima nustatyti *intmin* ir *intmax* funkcijomis. Tai patogu įvairiems matematiniams skaičiavimams atlikti ir įvairios paskirties matematiniams modeliams kurti.

Išsamiau aptarsime dažniausiai programavimo kalbose sutinkamas duomenų struktūras: masyvus, rinkinius, alternatyvas.

3.1. Masyvas

Istoriškai pirmasis struktūrinis duomenų tipas buvo masyvas (jis tuomet turėjo būti paprastasis, nes struktūrizavimas dar nebuvo galimas).

Labai dažnai tenka dirbti su daug vienodo tipo kintamųjų reikšmių. Paprasčiau visą tą elementų rinkinį galime pavadinti vienu vardu x, o atskirus to rinkinio elementus sunumeruoti: x_1, x_2, x_3, ... Tokį duomenų rinkinį vadiname masyvu. Masyvai sudaromi iš nustatyto skaičiaus elementų.

Informacijos masyvas. Kas tai?

Nupirkta n įvairių prekių. Yra žinoma prekės vieneto kaina Eurais ir pirktų vienetų skaičius. Kokia viso pirkinio kaina?

Pasižymime i-osios prekės kainą p_i, jos kiekį k_i, ieškoma bendra kaina S bus:

$$S = p_1k_1 + p_2k_2 + \ldots + p_ik_i + \ldots + p_nk_n$$

Matematikoje tokią formulę įprastai užrašome trumpiau:

$$S = \sum p_ik_i, \quad (i = 1 .. n) \qquad (1)$$

Išanalizavę (1) lygybę pastebime, kad daugybos veiksmas kartojamas n kartų su vis kitais skaičiais, tarpusavyje nesusijusiais jokia priklausomybe, išskyrus tarpusavio išsidėstymą. Tai reiškia, kad mums duota ne $2n$ skaičių, o dvi n ilgio sutvarkytos eilutės: viena eilutė – prekių kainos, kita eilutė – prekių kiekiai. Tokias eilutes vadinsime masyvais (pranc. *massif* – sunkus, lot. *massa* — luitas, gabalas).

Masyvas – daug to paties tipo reikšmių jungimas į vieną sudėtinę reikšmę. Jo komponentai vadinami masyvo elementais, jie pasiekiami naudojant indeksus.

Pavyzdyje eilutės koeficientai sudaro sutvarkytų elementų rinkinį, kurį pavadinsime masyvu.

Indeksas *i* nurodo masyvo elemento eilės numerį, o *p* ir *k* – masyvų pavadinimus.

Taigi sunumeruotų vienodo tipo kintamųjų rinkinys vadinamas masyvu.

Apibendrinsime: masyvas – tai duomenų struktūra, kuri tenkina šias savybes:

- visi elementai yra to paties tipo;
- elementų skaičius yra fiksuotas ir jo negalima keisti;
- kiekvienas masyvo elementas turi savo indeksą;
- masyvo elementus išrenkame tiesiogiai pagal jų indeksus.

Taigi masyvas – tai vieno tipo duomenų tuo pačiu vardu rinkinys, kuriame svarbi elementų išdėstymo tvarka.

Masyvo analogais matematikoje gali būti: seka, vektorius, matrica.

Masyvuose labai patogu saugoti vienarūšius duomenis. Pavyzdžiui, moksleivių klasės trimestro pasiekimų rezultatus, moksleivių sąrašą ar kitus statistinius duomenis.

Kaip masyvas aprašomas?

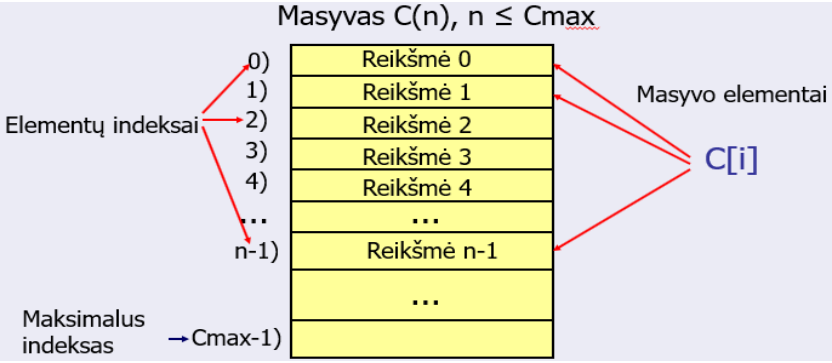
Skirtingose programavimo kalbose masyvai aprašomi panašiai: nurodomas masyvo vardas, elementų tipas, elementų skaičius. Pavyzdžiui, *C* kalboje masyvo aprašas yra:

tipas masyvo_vardas[Cmax]

Masyvo dydis *CMax* nurodomas konstanta (jos reikšmę būtų patogu keisti), pavyzdžiui *C* kalboje:

const int Cmax = 100;

Kompiuterio atmintinėje vienmačio masyvo elementai išdėstomi nuosekliai vienas paskui kitą (1 pav.).



1 pav. Vienmatis masyvas

Kas yra masyvo elementas?

Tai indeksuotas kintamasis, turintis masyvo vardą ir indeksą, pavyzdžiui, C(1), C(2),...

Indeksas nusako elemento eilės numerį masyve, tai yra diskretus dydis.

Elemento reikšmė išrenkama indeksu. Masyvo elementas reiškiniuose vartojamas kaip paprastas kintamasis.

Nors **masyvo reikšmė** yra sudėtinė – sudaryta iš daug jo elementų reikšmių, tačiau gali būti traktuojama kaip viena reikšmė. Masyvo reikšmę galima:

1) priskirti tokio paties masyvo tipo kintamajam,
2) perduoti parametru funkcijai, procedūrai ar kitokiam moduliui.

Kokius veiksmus galima atlikti su masyvais ir masyvo elementais?

- Atliekami veiksmai su masyvo elementais priklauso nuo masyvo elementų tipo.
- Su masyvais galima tik priskyrimo operacija (tik vienodo tipo masyvams).

Kokius tvarkymo veiksmus galima atlikti su masyvo elementais?

- Rikiavimą
- Šalinimą
- Įterpimą
- Reikšmių paiešką
- Grupavimą
- Keitimą

3.1.1.Dvimačiai masyvai

Dažnai pradiniai duomenys yra pateikiami lentele. Pavyzdžiui, moksleivių pasiekimų lentelė, parduotuvės darbo ataskaita ir pan.

Yra daug uždavinių, kurių duomenis patogu surašyti į lenteles:

1. Daugianario koeficientus galima surašyti tiesine lentele kartu nurodant narį.
2. Aritmetinės ar geometrinės progresijos elementus saugoti lentele.
3. Į tiesines lenteles galima surašyti ne tik skaitinius duomenis, bet įvairios prigimties informaciją: abėcėlės raides, kelio ženklus, mokinių pavardes ir kt.

Pavyzdžiui, aritmetinės progresijos narių duomenys užrašyti tiesine lentele:

a_1	a_2	a_3	a_4	a_5	...	a_8
1	5	10	15	20	...	35

Pagal sekos numerį galima greitai surasti reikiamą sekos narį lentelėje. Pastebime, kad seka apibūdinama dviem reikšmėmis: nurodoma sekos nario reikšmė ir nusakomas jos numeris toje sekoje. Turime sutvarkytą duomenų rinkinį – masyvą. Lentelei galima suteikti vardą.

Patogu tokią informaciją saugoti ne keliuose to paties tipo masyvuose, bet viename dvimačiame masyve.

Formuojant dvimatį masyvą svarbu, kokia informacija saugoma eilutėse, o kokia – stulpeliuose:

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

Dvimatis masyvas – tai masyvų masyvas, turintis du matmenis: eilučių skaičių ir stulpelių skaičių.

Dvimatį masyvą galime vaizduoti kaip vienmačių masyvų masyvą, t. y. masyvą, kurio elementai yra kiti vienmačiai masyvai. Privalumas – racionaliau naudojama atmintis, nes vienmačiai masyvai gali užimti skirtingus atminties sektorius.

Dvimačio masyvo aprašas nurodo vardą, elementų tipą, eilučių ir stulpelių skaičių:

Tipas Vardas [n] [m];

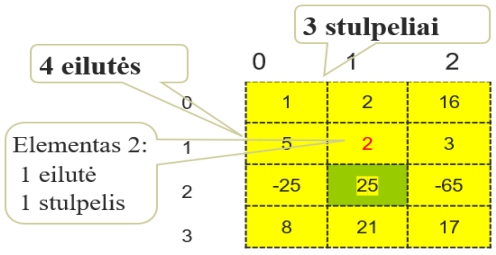
čia *n* ir *m* konstantos (sveikieji teigiami skaičiai)

Masyvo elementų indeksacija gali būti, pavyzdžiui, C kalba:

i = 0, *n* - 1 // eilučių indeksas,
j = 0, *m* – 1 // stulpelių indeksas.

Dvimačio masyvo elementas lentelėje aprašomas eilutės ir stulpelio numeriais (2 pav.). Masyvo elementas žymimas:

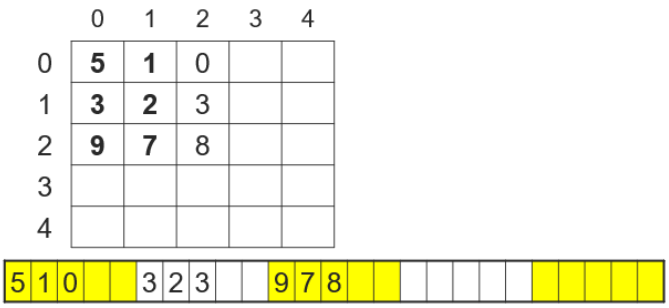
Masyvo_vardas [*indeksas*] [*indeksas*]



2 pav. Dvimačio masyvo elemento padėtis

Kaip dvimačio masyvo elementai laikomi kompiuterio atmintinėje?

Masyvo elementai kompiuterio atmintinėje naudojantis skirtingomis programavimo kalbomis gali būti išdėstomi skirtingai: eilutėmis arba stulpeliais. Pavyzdžiui, *C* kalboje elementai atmintinėje išdėstomi eilutėmis (3 pav.).



3 pav. *C* kalbos dvimačio masyvo išdėstymas kompiuterio atmintinėje

Masyvo elementai apibrėžiami naudojant indeksus:

- Vienmačio masyvo: A[0]; A[1]; A[2]
- Dvimačio masyvo: B[0][0]; B[0][1]; B[1][2]
- Trimačio masyvo: C[0][0][0]; C[0][0][1]; C[1][2][1]

Masyvų aprašai programavimo kalbose skiriasi nedaug. Pavyzdžiui, masyvo aprašo pavyzdys *Rust* programavimo kalboje:

let masyvo_vardas: [tipas, dydis]

Masyvą naudoti tinka tuo atveju, kai iš anksto žinome, kiek bus elementų ir kad mums su jais nereikės atlikti daug sudėtingų operacijų. Būna uždavinių, kuriuose masyvus panaudoti sunku ar neįmanoma. Tada praverčia kitos standartinės duomenų struktūros.

Užduotys savarankiškam darbui

1. Sakykime, kad masyvas A, kurio ilgis *n*, yra sveikojo skaičiaus skaitmenų seka. Raskite duotąjį skaičių *c*.
2. Sudarykite masyvą, kuriame būtų saugomos 2021 m. kiekvieno mėnesio vidutinė, žemiausia ir aukščiausia temperatūros.
3. Dvimatis masyvas gali būti vartojamas duomenų apie moksleivių pažangumą saugojimui, jo eilutės skiriamos atskirų moksleivių pažymiams aprašyti. Tokio masyvo taikymo galimybės yra labai ribotos. Jos būtų platesnės, jeigu papildomai dar būtų sudarytas informacinis masyvas, kuriame galėtume pagal moksleivio skaitinį kodą surasti jį aprašančius informacinius duomenis: vardą, pavardę, adresą, klasę.

3.2. Rinkinys

Realius sąrašus ir lenteles galima pavaizduoti rinkiniais. Su jo elementais galima atlikti įvairius veiksmus: nustatyti rinkinio ilgį; surasti elementą, kurio indeksas žinomas; paimti rinkinio dalį ir kt.

Rinkinys yra platesnė sąvoka negu lentelė. Lentelės vaizduoja vieno tipo sutvarkytų reikšmių visumą. Rinkinio reikšmės gali būti skaitinės, tekstinės, simbolinės.

Rinkinys – tai dydis, vaizduojantis baigtinę sutvarkytą visumą bet kokio tipo reikšmių. Rinkiniai konkrečiose programavimo kalbose vadinami skirtingai, pavyzdžiui, įrašu *Pascal* kalboje, struktūra *C* kalboje.

Kitais žodžiais galime pasakyti, kad rinkinys (įrašas, struktūra) – tai tiesioginis kelių duomenų tipų grupavimas.

Struktūros (įrašai)

Masyvai yra skirti vienodo tipo duomenų rinkiniams aprašyti. Sprendžiant daugelį įvairių praktinių uždavinių, tokių duomenų struktūrų nepakanka, nes tenka apdoroti iš įvairių tipų elementų sudarytus duomenų rinkinius. Aptarsime kelis pavyzdžius.

- Nagrinėjant moksleivių pažangumą, reikalingi duomenys, kurie apibūdina tiek moksleivį, tiek jo laikytų egzaminų įvertinimus. Moksleivio vardas ir pavardė yra simbolių eilutės, o egzaminų įvertinimai aprašomi skaičiais.
- Data apibūdinama trimis reikšmėmis: metais, mėnesiu, diena. Visus tris datos komponentus pavadinome skirtingais vardais, tad nepatogumo nėra. Jei sprendžiant uždavinį duota daug datų, tuomet atsiranda kelis kartus daugiau vardų ir sunkiau susigaudyti tarp jų.
- Kortos apibūdinamos spalva ir verte.
- Laikas apibūdinamas valandomis, minutėmis, sekundėmis.
- Žmogaus struktūra apibūdinama galūne (ranka, koja), puse (kairė, dešinė), pirštais (nykštys, smilius, didysis, bevardis, mažasis) ir t. t.
- Racionaliąją trupmeną apibūdina: skaitiklis, vardiklis (1..*maxint*).

Šie pavyzdžiai rodo, kad vieną objektą apibūdina kelios skirtingo tipo savybės, tad patogiau jas jungti į vieną sudėtinę reikšmę ir pavadinti vienu vardu. Taigi, reikia patogesnio būdo įvairaus tipo duomenims grupuoti, negu masyvas.

Struktūriniu duomenų tipu aprašomi nauji duomenų tipai, kurių reikšmės sudaromos iš kitų, jau žinomų duomenų tipų reikšmių. Tokio rinkinio reikšmę sudaro visų jo komponentų reikšmės drauge. Tuo skiriasi rinkinys nuo alternatyvos – alternatyvos reikšmę sudaro tik vieno jos elemento reikšmė.

Struktūros duomenų tipu (angl. *struct*) vadinamas įvairių duomenų tipų elementų rinkinys. Struktūros tipo kintamasis saugo kompiuterio atmintinėje tarpusavyje logiškai susietus duomenis.

Žmogus turi 20 pirštų. Jeigu naudotumėmės duomeniu, kurio reikšmės būtų kuris nors pirštas, tai turėtume sugalvoti 20 skirtingų pirštų vardų. Struktūros tipas leidžia turėti mažiau vardų.

Struktūrų aprašų sintaksė programavimo kalbose skiriasi. Pavyzdžiui, *C* kalboje struktūros tipas aprašomas taip:

```
struct Vardas
{
        tipas1 : kintamasis;
        tipas2 : kintamasis;
        ......
};
```

Rust kalboje struktūros tipas aprašomas panašiai:

```
struct Vardas
{
    kintamasis : tipas1
    kintamasis : tipas2
    ...
}
```

Riestiniuose skliaustuose išvardinami duomenų elementai, kurių kiekvienas turi savo tipą ir vardą. Atkreipiame dėmesį, kad sukuriamas naujas duomenų tipas nurodytu vardu. Toliau programoje jį galima naudoti kaip ir bet kurį kitą tipą. Svarbu yra skirti, kur yra struktūros tipo vardas ir kur yra tokios struktūros objektas (kintamasis). Vieno struktūros tipo objektų galime paskelbti daug.

Kokios operacijos atliekamos su struktūromis?

- Struktūrų priskyrimas – tai vienintelis veiksmas.
- Netaikomos palyginimo operacijos.
- Lyginti galima tik pagal elementus.

Kokie veiksmai atliekami su struktūros laukais?

Struktūrų laukams taikomos visos jų tipams leidžiamos operacijos.

Ar masyvas gali būti struktūros elementu?

Taip, masyvas gali būti struktūros elementu. Indeksai rašomi po masyvo elemento, o ne po struktūros.

Ar gali būti struktūrų masyvai?

Taip. Realiuose uždaviniuose gana dažnai yra naudojami struktūrų masyvai. Jie apibrėžiami taip pat, kaip ir kitų tipų masyvai. Pavyzdžiui, *C* kalboje gali būti statiniai ir dinaminiai struktūrų masyvai.

3.3. Alternatyva

Tai viena paprastesnių duomenų struktūrų, todėl daugelyje programavimo kalbų nepateikiama išreikštiniu pavidalu.

Programavimo praktikoje sutinkamos situacijos, kai reikalinga duomenų struktūra, kurioje būtų aprašyti:

1) keli laukai (kaip ir rinkinyje),
2) jos reikšmė būtų sudaryta tik iš vieno komponento ir būtų žinoma, iš kurio būtent komponento ji sudaryta.

Kas sudaro alternatyvos reikšmę ?

- Alternatyvos tipo reikšmė gaunama iš vieno apraše paminėto tipo reikšmės.
- Tai vieno kurio nors lauko reikšmė.

Pastebėsime, kad rinkinio reikšmę sudaro keli komponentai drauge.

Kaip alternatyva realizuojama kompiuterio atmintinėje?

Alternatyvos komponentai gali būti įvairių tipų. Kadangi vienu metu saugoma tik vieno komponento reikšmė, tai šiai daliai reikalingą atmintinės kiekį lemia tas komponentas, kuriam reikia daugiausiai vietos.

Aptarsime alternatyvos realizaciją *C* programavimo kalboje. Ši kalba turi junginio (angl. *union*) tipą.

Junginys – tai atminties sritis, kuri naudojama saugoti skirtingo tipo kintamiesiems. Junginys leidžia interpretuoti vieną ir tą patį bitų rinkinį skirtingai. Jo paskelbimas analogiškas struktūros paskelbimui, vietoje žodelio *struct* vartojamas žodelis *union*, tačiau savo esme jie yra visiškai skirtingi dalykai.

Junginio tipas leidžia interpretuoti toje pačioje fizinėje saugojimo struktūroje esančius duomenis keliais skirtingais būdais. Jis dažnai naudojamas atminties taupymo tikslais.

Pateiksime junginio aprašo sintaksę *C* kalba:

```
union vardas
{
    tipas1 elemento_vardas1;
    tipas2 elemento_vardas2;
   tipas3 elemento_vardas3; . . .
 };  objektų_vardai;
```

Nors laukai ir turi vardus, tačiau tipų kontrolės nėra – programuotojas turi pats pasirūpinti iš junginio išimto komponento tipo nustatymu. Taigi alternatyva realizuota tik iš dalies.

Pascal ir *Ada* kalbos neturi atskiro alternatyvos tipo konstrukcijos, tačiau tokį tipą galima realizuoti įrašu su variantine dalimi [3].

Apibendrindami galime sakyti, kad alternatyvos tipas yra kelių duomenų sąjunga, o alternatyvos tipo reikšmė – vieno kurio nors jos komponento reikšmė, kurią galima išimti.

Apskaičiuokite, kiek reikšmių turi tokio bendro tipo alternatyva:

```
T = (
        | A: boolean;
        | B: 0..5;
        | C: (gruodis, sausis, vasaris)
    )
```

Atsakymas: 2+6+3 (11).

4. Pagrindinės dinaminės duomenų struktūros

Aptarsime pagrindines dinamines duomenų struktūras, kurios taikomos sprendžiant didelius praktinius uždavinius.

Dinaminės duomenų struktūros skiriasi nuo kitų duomenų struktūrų dinamiškumu. Jų komponentų skaičius gali didėti arba mažėti.

Pirmiausia aptarsime sąrašą.

Sąrašas – tai duomenų struktūra, kurioje duomenys surašyti tam tikra tvarka.

Sąrašų pavyzdžiai:

1) darbuotojų sąrašas, kuriame nurodyti vardai, pavardės, pareigos, atlyginimas,
2) mokinių sąrašas, kuriame nurodyti vardai, pavardės, gimimo datos, klasė.

Sąrašai aprašo duomenų struktūras, kurios modeliuoja realius gyvenimo objektus ar reiškinius.

Sąrašas – tai objektų išvardijimas. Sąraše esantys objektai – sąrašo elementai. Iš sąrašo galima paimti bet kurį elementą, įtraukti naują elementą, šalinti esamą.

Daugeliu atvejų viena sąrašo eilutė charakterizuoja objektą, kuriam aprašyti reikalingi keli duomenų tipai. Pavyzdžiui, darbuotojų sąrašo vardai ir pavardės priskiriami eilutės tipo kintamiesiems, o parduodamų prekių kiekiai – realiojo tipo kintamiesiems.

Sąrašas nėra vien tik paprastas jį sudarančių įrašų rinkinys. Tie įrašai privalo būti susieti tarpusavyje, kad būtų galima nustatyti jų eilės tvarką sąraše. Todėl kiekviename sąrašo įraše, vadinamame mazgu, turi būti nuoroda į tolesnę grandį.

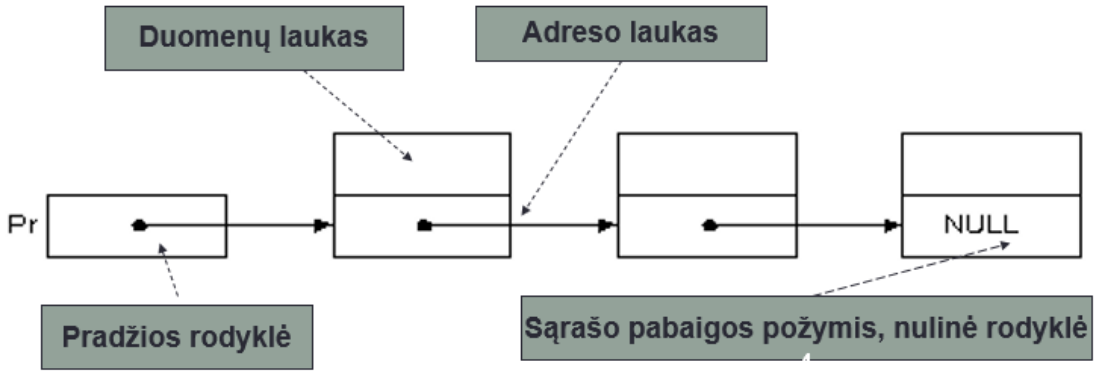
Dinaminis sąrašas sudaromas iš įrašų, kurių kiekvienas turi rodyklę į tolesnį sąrašo narį.

Tiesinis vienkryptis dinaminis sąrašas:

Tai vieno tipo elementų seka, kurioje kiekvienas sekos elementas saugo:

1) savo duomenis,
2) kito sekos elemento adresą.

Laikoma, kad iš pradžių sąrašas būna tuščias. Šiuo atveju sąrašas – tai pradžios rodyklė ir rodyklėmis susietų vienodų elementų seka. Iš pradžios rodyklės paeiliui galima pereiti visus elementus. Sąrašo sandaros grafinis vaizdas yra pateiktas 4 pav.

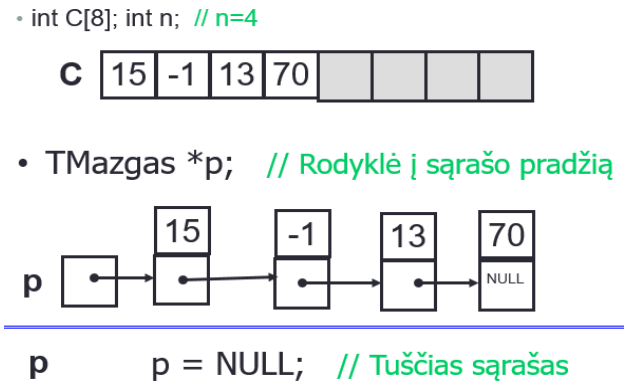


4 pav. Tiesinio vienkrypčio sąrašo sandara.

Vienas tokio sveikųjų skaičių sąrašo narys (mazgas) pavyzdžiui, *C* programavimo kalba gali būti užrašomas taip:

```
struct TMazgas
{
 int duom;
 TMazgas *sek;
};
```

Galime pabandyti palyginti masyvo ir tiesinio vienkrypčio sąrašo sandaros grafinius vaizdus:



5 pav. Masyvo ir tiesinio vienkrypčio sąrašo sandara

Dirbant su dinaminiais sąrašais reikia mokėti:

- Suformuoti sąrašą;
- Peržiūrėti (išspausdinti) sąrašą;
- Ieškoti sąraše mazgo (rasti reikiamą mazgą);
- Įterpti naują mazgą;
- Pašalinti mazgą.

4.1.Tiesiosios ir šakotosios dinaminės duomenų struktūros

Uždaviniams spręsti naudojamos šios pagrindinės dinaminės struktūros:

- Tiesiosios dinaminės duomenų struktūros: tiesinis dinaminis sąrašas, jo modifikacijos (dvikryptis, ciklinis), specialios paskirties struktūros (dėklas, eilė, dekas).
- Šakotosios duomenų struktūros: dvejetainis medis, Bajero medis ir kt.

Dėklas (angl. *stack*) – tai duomenų struktūra, kai duomenys dedami iš eilės tokia tvarka, kokia pateikiami, o paimti galima tik paskiausiai įdėtą duomenį; tik jį paėmus, bus galima paimti prieš tai buvusį duomenį ir t. t.

Dėklas angliškai apibūdinamas santrumpa LIFO (*last in, first out*)– atitinka lietuvišką patarlę „kas pirmas į maišą, paskutinis iš maišo“.

Tai labai dažnai naudojama duomenų struktūra. Ji dažnai naudojama ir buityje:

- Padėklų krūva valgykloje: padėklas, kuris paskutinis padedamas ant viršaus, bus paimtas pirmas, o apatinis padėklas, kuris buvo padėtas pirmas, bus paimtas paskutinis.
- Kaladėlių bokšto statyba ir ardymas.

- Knygų krovimas į siaurą dėžę (vienu bokštu) ir išėmimas iš jos.
- Traukinys su vagonais: vagonai prijungiami arba atskiriami.

Dėklai naudojami įvairių algoritmų realizacijose kuriant programas.

Kita dažnai naudojamų duomenų struktūrų yra **eilė.**

Eilė (angl. *queue*) – tai tokia duomenų struktūra, į kurią duomenys dedami jų gavimo tvarka (į eilės pabaigą), o išimamas pirmasis į eilę įdėtas duomuo (esantis eilės pradžioje).

Eilė angliškai apibūdinama santrumpa FIFO (*first in, first out* – atitinka posakį „*kas pirmas įeina, tas pirmas išeina*“.

Eilę galima apibrėžti kaip tiesinę duomenų struktūrą, iš kurios **pradžios elementai šalinami ir į kurios galą įrašomi nauji elementai.**

. Daug eilės pavyzdžių sutinkame realiame gyvenime:

- Eilės taisyklę naudojame buityje: taip aptarnaujami klientai parduotuvėse, įstaigose.
- Eilę galime įsivaizduoti kaip pirkėjų eilę stovinčią prie kasos. Naujai atėjęs pirkėjas stoja į eilės galą, o aptarnaujamas pirkėjas, esantis eilės pradžioje.
- Stovime eilėje norėdami patekti į mokyklinį autobusą: pirmas eilėje laukiantis vaikas į autobusą įlips pirmas, o vėliausiai atėjęs vaikas atsistos į eilę paskutinis.
- Dokumentų, pasiųstų spausdinti, eilė spausdintuve.
- Klaviatūros buferis - klavišų paspaudimai yra laikomi eilėje ir kt.
- Kai kada eilės yra **prioritetinės**, pvz., poliklinikoje pirmiausia aptarnaujami karščiuojantys ligoniai, po to, ligoniai, turintys lengvatų, ir tik vėliau visi likę pacientai. Kiekvienos grupės ligoniai vėl sudaro atskiras eiles.

Kokios gali būti atliekamos operacijos?

Priimama, kad iš pradžių eilė yra tuščia - tai pradinė eilės būsena. Tam programavimo kalbose yra speciali operacija. Pavyzdžiui, C kalboje yra rodyklė *NULL*.

Eilė yra specifinis sąrašas su apribotomis operacijomis. Galimos operacijos:

1. Sukurti tuščią eilę.
2. Patikrinti, ar eilė tuščia.
3. Patikrinti, ar eilė pilna.
4. Įdėti (angl. *enqueue*) naują elementą į eilę.

5. Išimti (angl. *dequeue*) elementą iš eilės.
6. Gauti pirmo eilės elemento duomenis, neišimant jo iš eilės.
7. Gauti eilės elementų skaičių.
8. Sunaikinti eilę.

Dėklas ir eilė skiriasi vienas nuo kito tik elemento padėjimo į sąrašą operacija.

Eilė yra viena iš svarbių duomenų struktūrų, naudojamų programose ir kompiuterių procesuose.

Praktinė užduotis. Padėklų krūvelė

Mokyklos valgykloje yra dviejų spalvų padėklai:

1) balti pradinukams,
2) raudoni vyresnių klasių mokiniams.

Kai valgykloje vyko remonto darbai, buvo galima sudėti tik vieną padėklų krūvelę. Mokinukai laukdami pietų stovi eilėje, o virėjai turi sudėti padėklus taip, kad lėkščių sudėjimas į padėklus atitiktų eilėje laukiančius mokinius. Šiuo atveju gali būti pateikta mokinių eilė ir padėklų krūvelė. Viena iš užduočių gali būti patikrinti, ar padėklų krūvelė sukrauta gerai.

Šiame uždavinyje mokinukai, stovintys eilėje, gali būti pavaizduoti eilės duomenų struktūra, o padėklų krūvelė – dėklo duomenų struktūra.

Kiekvieno mokinuko suporavimas su padėklu, nekeičiant išsidėstymo tvarkos, gali būti vadinamas atvaizdavimu. Kiekvienai sekai (padėklų krūvelei ir mokinukų eilei) yra apibrėžtas pirmasis elementas (aukščiausiai esanti padėklas ir kairiausias mokinukas), jie turi sutapti. Toliau kiti elementai poruojami atitinkamai iš eilės. Tokiu būdu nustatomas atitikimas tarp padėklų ir mokinukų.

Su dėklu galima atlikti keletą veiksmų (operacijų):

1. Galima pašalinti aukščiausiai esantį elementą.
2. Galima įterpti naują elementą į dėklą, padedant viršuje.

Tačiau iš sąlygos nežinome, kaip buvo sudarytos sekos: jos galėjo būti sudarytos ir iš viršaus į apačią (t. y. pirmasis padėklas krūvelėje buvo padėtas pirmasis, antroje vietoje esantis buvo padėtas antrasis ir taip toliau), tai būtų vadinama eile, jai galimos šios operacijos:

1) galima pašalinti priekinį elementą,
2) galima įterpti naują elementą eilės gale.

Tą patį galima pasakyti ir apie mokinukus: jie formuoja eilę: kairysis buvo pirmas, dešinysis – paskutinis, jis eilę paliktų paskutinis. Gali būti ir kad kairysis mokinukas eilėje atsirado paskutinis, o ją paliktų pirmas.

Programuojant labai svarbu teisingai parinkti ir suformuluoti konkrečias pradinių duomenų struktūras bei instrukcijas kuriamai programai. Jei pavyzdyje padėklai būtų išdėstomi eile, tai pirmas mokinukas gautų vėliausiai padėtą padėklą (tai dėklo duomenų struktūra).

Gebėjimas suprasti duomenų pateikimo taisykles – tvarką – yra labai svarbus programavimo įgūdis.

4.2. Medžiai. Dvejetainis medis

Programavime naudojami įvairūs duomenų kodavimo būdai. Iki šiol nagrinėtos duomenų struktūros (pvz., dėklas, eilė) iš esmės buvo vienmatės – kompiuterio atmintinėje jos buvo dėstomos nuosekliai – fizine ar logine prasme.

Tokias struktūras gana paprasta programuoti ir naudoti. Tačiau jos gali būti ne visada efektyvios, t. y. algoritmas, veikiantis jų pagrindu, gali būti lėtas arba per daug sudėtingai valdomas. Todėl verta aptarti ir sudėtingesnes – dvimates, trimates duomenų struktūras.

Viena iš tokių duomenų struktūra yra **medis**.

Sąrašas, kurio elementus ryšio tvarka negalima išdėstyti vienoje eilėje, vadinamas netiesiniu. Populiariausi ir paprasčiausi netiesiniai dinaminiai sąrašai yra medžio tipo.

Medžio sąvoka yra labai svarbi programavime. Ji vartojama duomenims sutvarkyti taip, kad juos būtų galima greitai surasti.

Pabandysime keliais būdais apibrėžti medžio sampratą.

Medis yra tokia hierarchinė duomenų struktūra (šakotųjų duomenų struktūra), kurioje:

- į kiekvieną elementą (viršūnę), išskyrus vieną, vadinamą medžio šaknimi, yra tik vienintelė nuoroda iš kito elemento (viršūnės),
- iš kiekvieno elemento (viršūnės) gali būti viena arba daugiau nuorodų į kitus elementus (viršūnes),
- iš šaknies einant nuorodomis galima pasiekti visas kitas viršūnes.

Medis – tai duomenų struktūra, vaizduojama iš viršaus į apačią, kur elementai vadinami viršūnėmis. Viena viršūnė išskiriama iš kitų, ji vadinama medžio šaknimi. Kiekvienoje viršūnėje judama skirtingais keliais, priklausomai nuo kitos ypatybės reikšmės. Toks procesas padeda sudaryti struktūruotą visų galimų variantų sąrašą.

Papildomai galima paskaityti:

https://inf-knyga.nmakademija.lt/lt/latest/11_medžiai.html

Sąrašų medžiai grafiškai braižomi apverstai – šakomis žemyn (6 pav.).

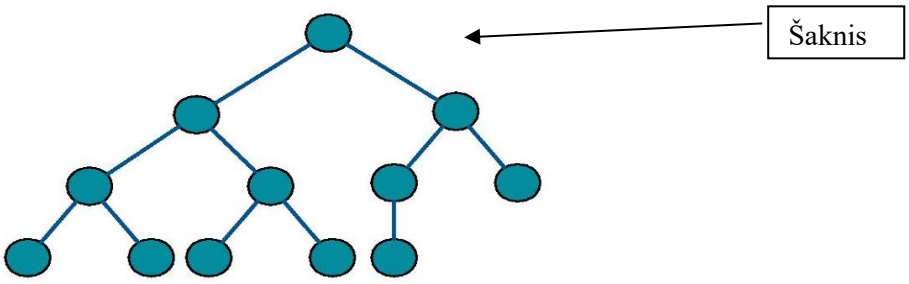
Galimų situacijų nagrinėjimas, pavaizdavus jas medžiu, yra bendra kompiuterinių žaidimų kūrimo strategija. Kai kuriems žaidimams sprendimų medžiai gali būti tokie dideli, kad jų neįmanoma sudaryti, tada ieškoma dalinių sprendimų aprašant tik dalį sprendimo medžio.

Praktikoje sutinkami ir konstruojami įvairūs medžiai: dvejetainiai, AVL, 2-3-4, raudoni-juodi, Bajero ir kt.

Elementarus šakotosios struktūros atvejis yra dvejetainiai medžiai.

Dvejetainis medis – tai duomenų struktūra, kai duomenys aprašomi viršūnėmis ir šakomis ir iš kiekvienos viršūnės išeina ne daugiau kaip po dvi šakas, išskyrus pirmąją.

Medžio struktūra pradedama konstruoti pradiniu mazgu – **š**aknimi (68 pav.). Toliau jungiamos kitos šakos.



6 pav. Dvejetainio medžio duomenų struktūros sandara

Dvejetainiame medyje yra trijų rūšių mazgai:

1) pradinis mazgas arba **šaknis** (angl. *root node*),
2) šakos mazgas – **šaka** (angl. *branch node*),
3) pabaigos mazgas – **lapas** (angl. *leaf node*).

Visi mazgai taip pat gali būti vadinami pagal šakos kryptį arba paveldėjimo kryptį:

1) kairysis mazgas arba dešinysis mazgas (angl. *left*, *right*),
2) tėvinis mazgas arba dukterinis mazgas (angl. *parent*, *child*).

Dvejetainių medžių elementai aprašomi taip pat, kaip ir dvikrypčių sąrašų elementai, ryšio dalyje formuojant dvi nuorodas.

Duomenys medyje turi aiškią hierarchiją, paieška ir elementų įterpimas dvejetainiuose medžiuose yra efektyvesni, medžio gabalus galima labai lengvai kilnoti.

Apibendrinsime: medis – gana efektyvi struktūra duomenims įvardinti, kitaip sakant, adresuoti.

Išsamiau apie dvejetainius medžius skaitykite:
http://www.techmat.vgtu.lt/konspektai/Algoritmu%20teorija/Paskaita5.pdf

Literatūra

1. Baniulis, K., Tamulynas, B. Duomenų struktūros. Technologija, Kaunas, 2003.
2. Dagienė, V., Grigas, G. Programavimo kalbų teoriniai pagrindai. 2007. https://www.scribd.com/document/46457604 6/Dagien%C4%97-V-Grigas-G-2007-Programavimo-kalb%C5%B3-teoriniai-pagrindai.
3. Grigas, G. Duomenų tipai. Žuvėdra, Vilnius, 1997.