VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

KRISTINA LAPIN

PROGRAMAVIMAS PASKALIU IR C

II dalis

MATEMATIKOS IR STATISTIKOS
STUDIJŲ KRYPČIŲ
PIRMO KURSO STUDENTAMS

Vilnius, 2005

TURINYS

Pratarmė

Metodinė priemonė „Programavimas Paskaliu ir C, II dalis“ yra skirta 1 kurso matematikos ir statistikos studijų krypčių studentams. Ji apima Informatikos dalyko antro semestro programą. Ši priemonė papildo doc. dr. S. Ragaišio metodinę priemonę [2] programavimo aspektu. Joje yra pateikiami programavimo sąvokų realizavimo Paskaliu ir C pavyzdžiai. Daugelyje skyrių siūloma susipažinti su [2] priemonėje išdėstyta medžiaga ir toliau nagrinėti čia pateikiamus pavyzdžius.

Ši medžiaga yra skirta programavimo naujokams. Joje stengiamasi kuo paprasčiau, remiantis iliustracijomis ir jas atitinkančiomis kodo eilutėmis išdėstyti dinaminio programavimo metodiką. Labiau pažengusiems programavime, siūlyčiau skaityti kitas, aukštesnį žinių lygį atitinkančias mokymo priemones [1], [4]. Paskutinis skyrius, skirtas simbolių eilučių apdorojimui C kalboje, yra fakultatyvinis. Realizuoti simbolių eilučių apdorojimo užduotį privaloma tik Paskalio kalba. Už C kalbos realizaciją studentui yra suteikiami papildomi taškai.

Šioje mokymo priemonėje yra siūloma šis mokymosi scenarijus:

1) išnagrinėti [2] metodinėje priemonėje pateiktą sąvokos apibūdinimą:

2) išnagrinėti šioje metodinėje pateikiamus realizacijos paaiškinimus,

3) pratybų metu gautą užduotį pradėti rašyti nuo pavyzdžio, pateikto šioje medžiagoje,

4) pertvarkyti pavyzdį, atsižvelgiant į gautą užduotį.

Sąvokų realizacija yra iliustruojama Paskalio ir C programavimo kalbų pavyzdžiais. Sąmoningai stengiamasi pateikti vieną (autorės nuomone paprastesnį) realizavimo būdą ir vengti visų kalbos galimybių ir subtilybių dėstymo.

Ši priemonė leidžia pasiruošti skaityti ir suprasti programavimo literatūrą, skirtą skaitytojui, suprantančiam bazines programavimo sąvokas.

1. DINAMINIS ATMINTIES IŠSKYRIMAS

1.1. STATINIS IR DINAMINIS ATMINTIES IŠSKYRIMAS

Atsižvelgiant į egzistavimo trukmę ir identifikavimo būdą, kintamieji yra skirstomi į statinius ir dinaminius.

Statiniai kintamieji identifikuojami vardais. Šie kintamieji atsiranda, tai yra jų reikšmėms yra skiriama vieta atmintyje, pradėjus vykdyti programą, kurioje jie yra aprašyti. Tokie kintamieji vadinami statiniais, nes jų egzistavimą nusako tų kintamųjų aprašų vieta programos tekste.

Dinaminiai kintamieji sukuriami, vykdant programą, kurioje yra standartinės kintamojo kūrimo (**new**, **malloc**) ir naikinimo (**dispose**, **free**) procedūros. Jie vadinami dinaminiais todėl, kad jų egzistavimo trukmę nusako programos atlikimo procesas. Dinaminiai kintamieji identifikuojami rodyklėmis. Rodyklė – tai rodyklės tipo reikšmė.

> *Kalbant apie statinius ir dinaminius kintamuosius Paskalyje ir C reiktų paminėti tam tikrą šių kalbų terminų nesuderinamumą. Statiniu kintamuoju C kalboje yra vadinamas* ***static*** *atminties klasės kintamasis. Šiame skyriuje statinio kintamojo terminą naudojame Paskalio kalbos prasme, būtent kintamojo, kuriame yra saugoma ne rodyklė į reikšmę, o pati reikšmė.*

Aprašant dinaminius kintamuosius, kintamųjų aprašų skyriuje nurodomas ne pats kintamasis, o rodyklė į jį. Taigi, rodyklė yra įprastas kintamasis, o kintamasis į kurį nurodo rodyklė – dinaminis.

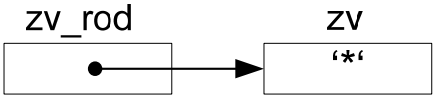
1.2. RODYKLĖS

Naudojant rodykles į kintamąjį, kintamasis yra apdorojamas, operuojant jo adresu.

Žemiau pateiktuose programų fragmentuose iliustruojamas statinių ir dinaminių kintamųjų aprašymas ir naudojimas. Dar kartą pastebėkime, kad šiame kodo fragmente kintamasis **zv** yra statinis Paskalio kalbos terminijos prasme, tai yra turima omenyje, kad jame yra saugoma reikšmė. o ne rodyklė. Kintamasis **zv_rod** yra rodyklė.

Paskalis	C
program stat_dinam; **var zv_rod: ^char;** **zv: char;** **begin** **zv := '*' ;** **zv_rod := addr(zv);** **end.**	**#include <stdio.h>** **int main()** **{** **char *zv_rod, zv;** ***zv_rod = '*';** **zv_rod = &zv;** **}**

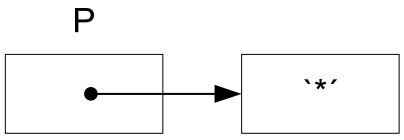
1-1 pav. pavaizduoja situaciją atmintyje, atlikus pateiktame fragmente nurodytus veiksmus.



1-1 pav. Grafiškas kintamųjų būsenos pavaizdavimas, atlikus programėlėje aprašytus veiksmus

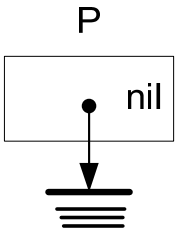
Rodyklės kintamasis gali būti šiose būsenose:

1) Jame gali būti įrašytas dinaminio kintamojo adresas, kuriam yra išskirtas vieta ir priskirta reikšmė (žr. 1-2 pav.).



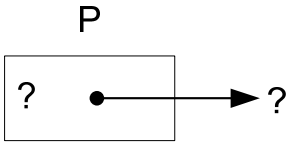
1-2 pav. Apibrėžta rodyklė

2) Jame gali būti specialus tuščias simbolis. Paskalio kalboje šis simbolis yra vadinamas **`NIL`** (žr. 1-3 pav.). C kalboje tuščias simbolis yra **`NULL`**.



1-3 pav. Tuščios rodyklės žymėjimas Paskalyje

3) Jis gali būti neapibrėžtas (žr. 1-4 pav.).



1-4 pav. Neapibrėžta rodyklė

Šioje būsenoje rodyklė yra programos darbo pradžioje, kol nepriskirtas joks adresas ir tuomet, kai atminties ląstelė, į kurią rodo rodyklė, yra atlaisvinama. Rodyklė dar literatūroje gali būti vadinamos nuorodomis.

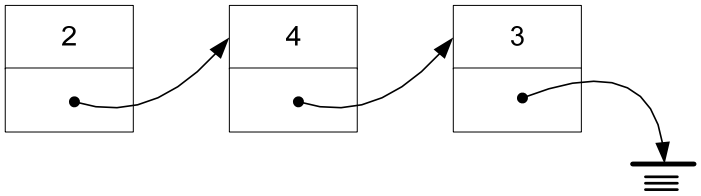
2. DUOMENŲ STRUKTŪROS

2.1. VIENPUSIS SĄRAŠAS

Vienpusis sąrašas yra tokia duomenų struktūra, kurioje kiekvienas, išskyrus paskutinį, sąrašo elementas nurodo einantį po jo elementą (žr. 2-1 pav.). Minimalus žinių rinkinys apie šią duomenų struktūrą yra pateikiamas [2] metodinėje priemonėje:
http://www.mif.vu.lt/~ragaisis/InfIvadas/DuomStr.htm

Rekomenduoju paskaityti ten pateiktą medžiagą, prieš pradedant mokytis programuoti dinamines struktūras.

2.1.1. Sąrašo aprašas



2-1 pav. Vienpusio sąrašo grafiškas pavaizdavimas

Kiekvienas identifikatorius turi būti aprašytas prieš jo panaudojimą kituose aprašuose. Aprašant dinaminius kintamuosius negalima išsiversti ir dinaminės struktūros elementams yra padaryta išimtis. Paskalyje rodyklė į dinaminės struktūros elementą turi būti aprašoma prieš to elemento aprašą. C kalboje taip pat tipo apraše naudojamas rekursinė nuorodą į aprašomą tipą.

Paskalis	C

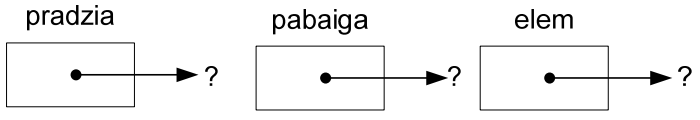
`type rod = ^el;`
`el = record`
`duom : integer;`
`kitas : rod;4`
`end;`
	`struct el {`
	`int duom;`
	`struct el *kitas;`
	`};`

2.1.2. Sukūrimas

Vienpusis sąrašas yra sukuriamas, kai į jį įtraukiamas pirmas elementas. Vienpusiame sąraše paprastai naudojame du kintamuosius-rodykles, kurie rodo į sąrašo pradžią ir pabaigą. Pradžioje yra aprašomos sąrašo pradžios, pabaigos rodyklės. **`elem`** – darbinė rodyklė, reikalinga darbui su sąrašu.

Paskalis	C
`var pradzia, pabaiga, elem : rod;`	`struct el *pradzia, *pabaiga, *elem;`

Kol dar rodyklėms nepriskirtos reikšmės, jos yra neapibrėžtos (žr. 2-2 pav.).



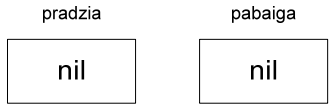
2-2 pav. Aprašyta nuoroda, kuriai dar nepriskirta jokia reikšmė

2) Tuščio sąrašo sukūrimas

Tuščias sąrašas yra sukuriamas, kai pradžios ir pabaigos rodyklėms yra priskiriamas tuščias simbolis:

Paskalis	C
`pradzia := nil;` `pabaiga := nil;`	`pradzia = NULL;` `pabaiga = NULL;`

Po šių veiksmų kintamųjų būsenas pavaizduoja 2-3 pav.



2-3 pav. Tuščias sąrašas

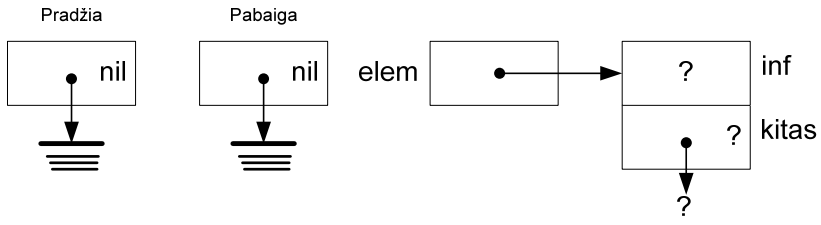
3) Pirmajam sąrašo elementui išskiriama atmintis.

Paskalyje naujas elementas yra sukuriamas, naudojant procedūrą **`new()`**. Žemiau pateikiamas iliustracijoje pavaizduotos situacijos aprašas Paskaliu. C kalboje, naudojant atminties išskyrimo funkciją **`malloc()`**, reikia prijungti antraščių failą **`stdlib.h`**, kuriame yra šios funkcijos prototipas (pirmavaizdis).

Paskalis	C

`new ( elem );`	`#include <stdlib.h>`
	...
	`elem = (struct el *) malloc ( sizeof ( struct el ) );`

Po šių veiksmų kintamųjų būsenas grafiškai pavaizduoja 2-4 pav. Jame rodyklėje **`elem`** jau patalpintas pirmam sąrašo elementui išskirtos atminties adresas.

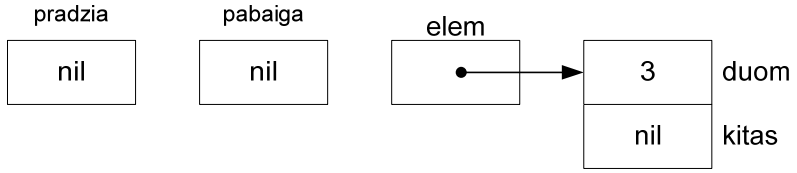


2-4 pav. Grafiškas kintamųjų būsenų pavaizdavimas

4) Pirmajam elementui priskiriamos reikšmes.

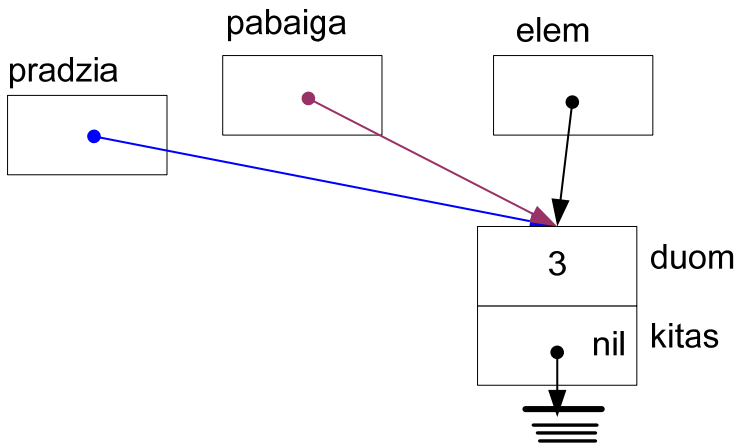
Paskalis	C
`elem^.duom := 3;` `elem^.kitas := nil;`	`elem -> duom = 3;` `elem -> kitas = NULL;`

Šis kodo fragmentas pakeitė dinaminio kintamojo būseną (žr. 2-5 pav.). Dinaminio kintamojo laukams yra priskirtos reikšmės.



2-5 pav. Dinaminio kintamojo laukams yra priskiriamos reikšmės

5 žingsnis. Nukreipiame sąrašo pradžios ir pabaigos rodykles į pirmą sukurtą elementą (žr. 2-6 pav.).



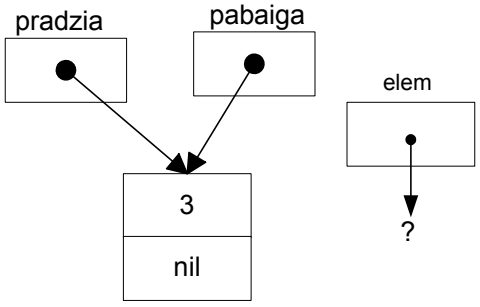
2-6 pav. Sąrašo pradžios ir pabaigos rodyklių susiejimas su sukurtu elementu

2-6 pav. rodyklių spalva atitinka kodo eilutės spalvą:

Paskalis	C
`pradzia := elem;` `pabaiga := elem;`	`pradzia = elem;` `pabaiga = elem;`

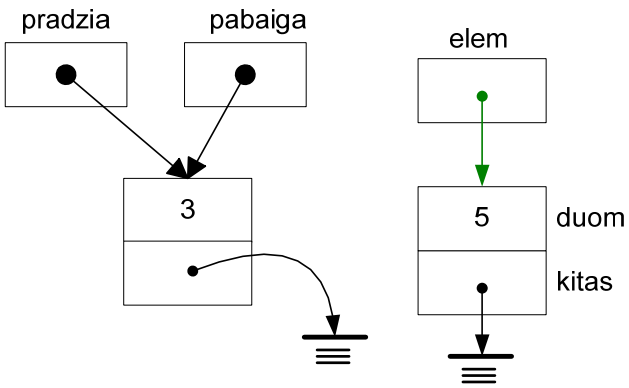
2.1.3. Prijungti elementą prie sąrašo pabaigos

Pradinė būsena: sąraše yra vienas elementas, į kurį yra nukreiptos sąrašo pradžios ir pabaigos rodyklės. Pagalbinio elemento **elem** reikšmė neapibrėžta (žr. 2-7 pav.).



2-7 pav. Elemento prijungimo prie sąrašo pradinė padėtis

1) Atminties išskyrimas naujam elementui ir reikšmių priskyrimas.



2-8 pav. Reikšmių naujam elementui priskyrimas

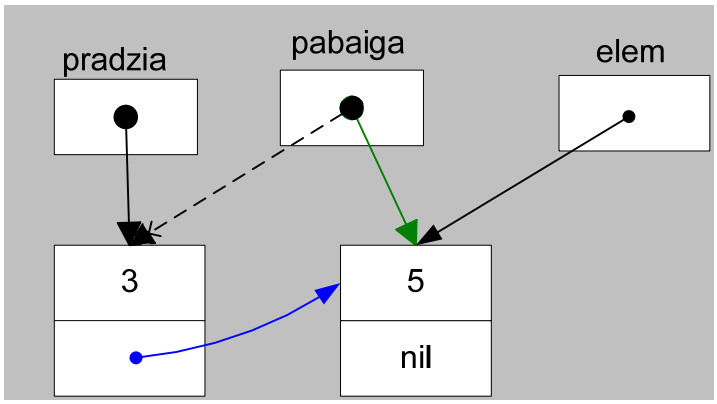
Paskalis	C

`new ( elem );`
`elem^.duom := 5;`
`elem^.kitas := nil;`
	`elem = (struct el *) malloc( sizeof(struct el));`
	`elem->duom = 5;`
	`elem->kitas = NULL;`

3) Elemento susiejimas su esamu sąrašu.

Sąrašo galo nuoroda nukreipiama į prijungiamą elementą. Buvęs paskutinis elementas taip pat turi rodyti į prijungiamą elementą.

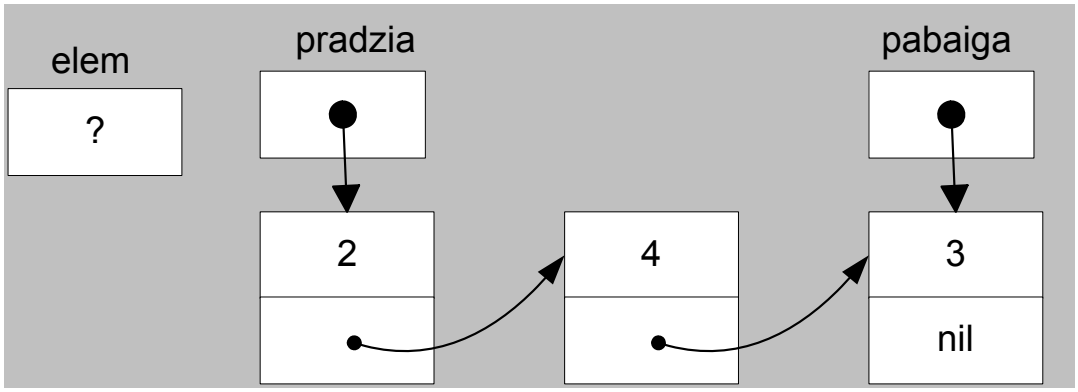
Paskalis	C
`pabaiga^.kitas := elem;` `pabaiga := elem;`	`pabaiga->kitas = elem;` `pabaiga = elem;`



2-9 pav. Sąrašo pabaigos nuorodos nukreipimas į naująją sąrašo pabaigą

2.1.4. Įterpti elementą į sąrašo pradžią

1) Pradinė būsena: sąraše jau turi būti bent vienas elementas. 2-10 pav. pavaizduotas sąrašas su trimis elementais.

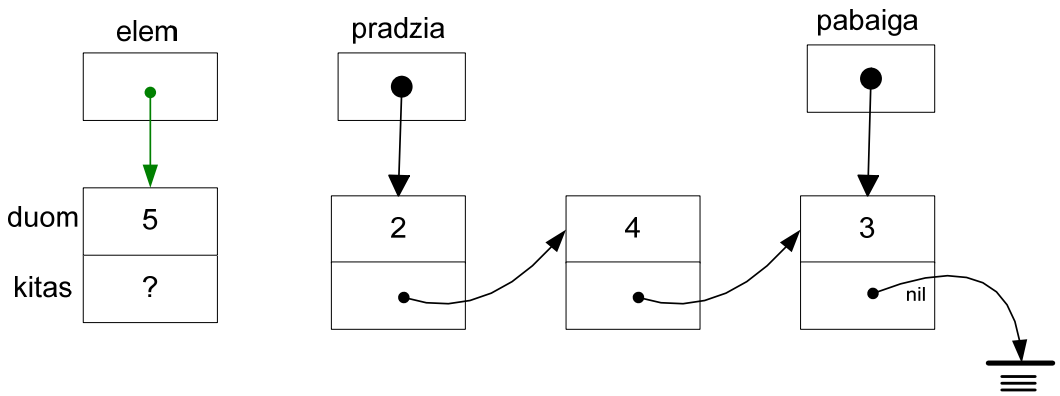


2-10 pav. Naujo elemento įterpimo į sąrašo pradžią pradinė padėtis.

2 žingsnis. Atminties išskyrimas naujam elementui ir reikšmės priskyrimas

Paskalis	C
`new ( elem );` `elem^.duom := 5;`	`elem = (struct el *) malloc( sizeof(struct el));` `elem->duom = 5;`

Šiame etape išskiriame vietą naujam sąrašo elementui. Jo duomenų laukui priskiriame reikšmę (žr. 2-11 pav.).

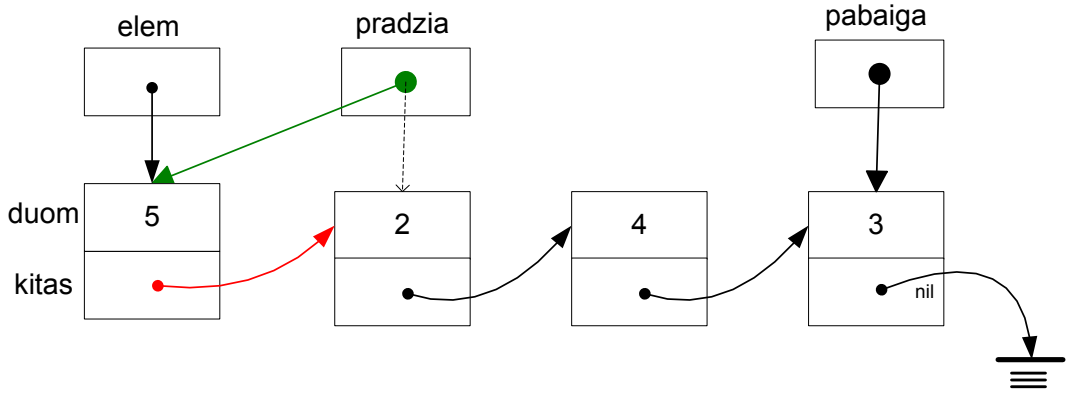


2-11 pav. Prijungiamo elemento sukūrimas ir reikšmės priskyrimas

3 žingsnis. Elementas prijungiamas prie sąrašo.

Paskalis	C
`elem^.kitas := pradzia;` `pradzia := elem;`	`elem->kitas = pradzia;` `pradzia = elem;`

Šiame kodo fragmente aprašoma situacija grafiškai pavaizduojama 2-12 pav.

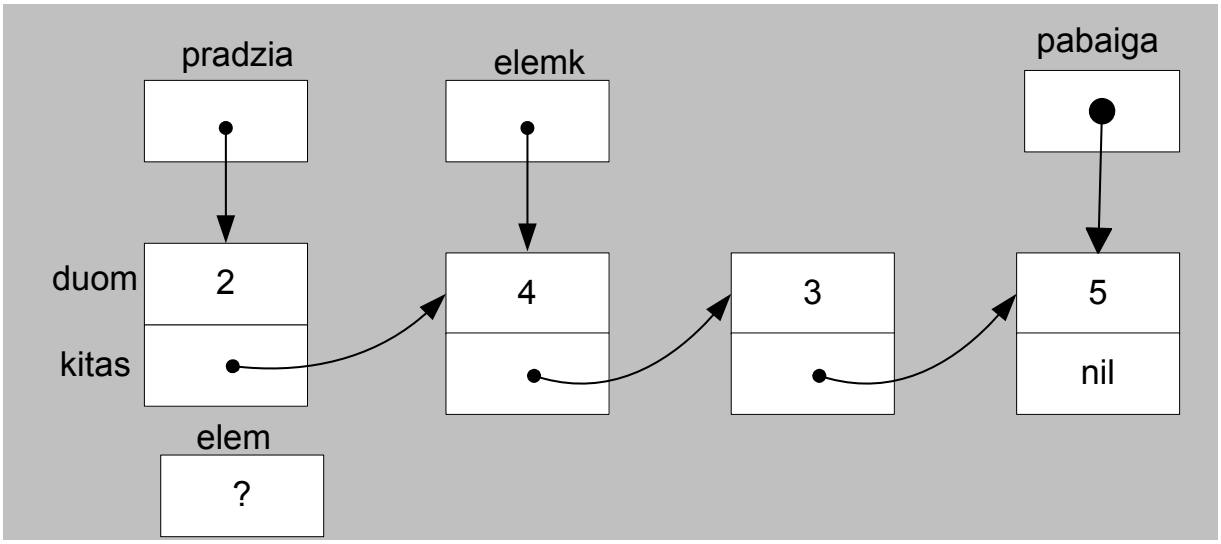


2-12 pav. Sukurto elemento susiejimas su sąrašu

2.1.5. Elemento įterpimas sąrašo viduje, po k-ojo elemento

Pradinė būsena: į k-ąjį elementą turi rodyti papildoma rodyklė **elemk** (žr. 2-13 pav.) . Kaip nukreipti rodyklę į numeriu nurodytą elementą bus aiškinama 2.1.9 skyrelyje.

```
var elemk: rod;
```

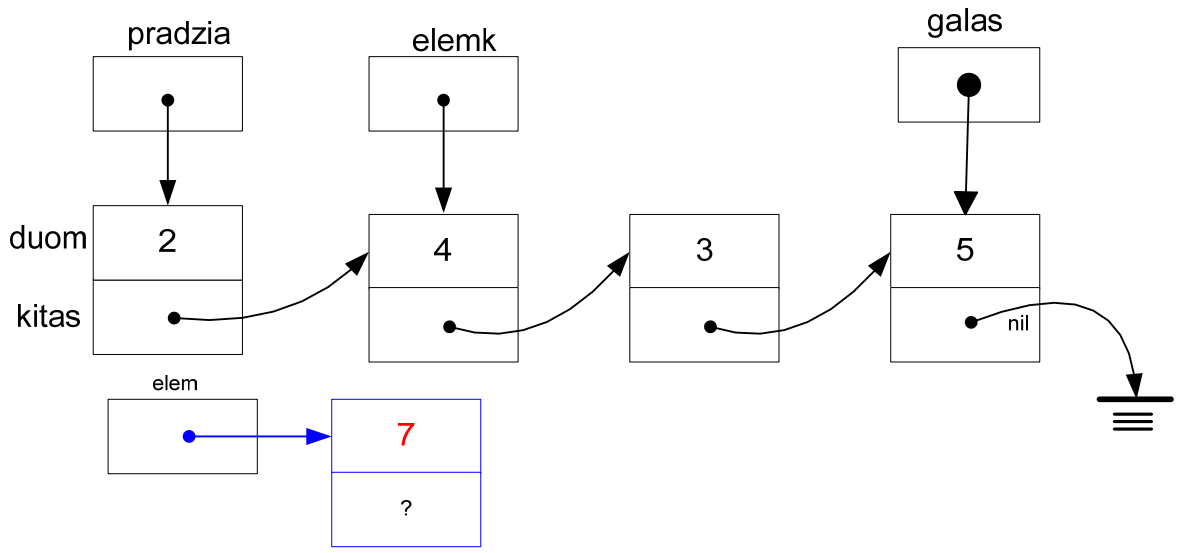


2-13 pav. Elemento įterpimo po k-tojo elemento pradinė padėtis

2 žingsnis. Išskirsime atmintį naujam elementui.

Paskalis	C
`new ( elem );` `elem^.duom := 7;`	`elem = (struct el *) malloc (sizeof(struct el));` `elem -> duom := 7;`

Po veiksmų, aprašytų šiame kodo fragmente yra gauta situacija, kurią iliustruoja 2-14 pav.

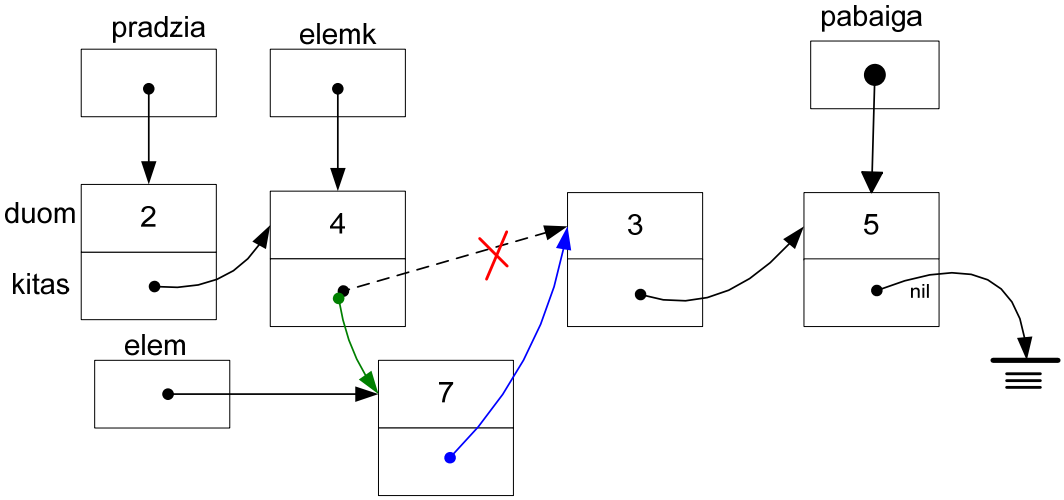


2-14 pav. Įterpiamo elemento kūrimas

3 žingsnis. Susiejame naują ir k+1 – jį elementą ir k - ąjį su nauju elementu.

Paskalis	C
`elem^.kitas := elemk^.kitas;` `elemk^.kitas := elem;`	`elem -> kitas = elemk->kitas;` `elemk -> kitas = elem;`

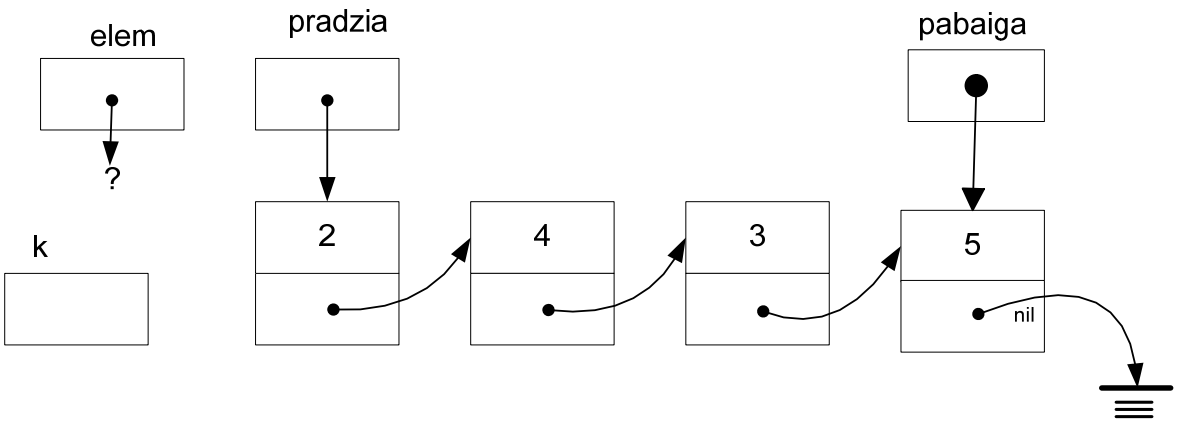
Šias kodo eilutes iliustruoja 2-15 pav.



2-15 pav. Sukurto elemento įterpimas po k-tojo elelemto: susiejimas su k ir k+1 sąrašo elelemtais

2.1.6. Panaikinti elementą iš sąrašo pradžios

Pradinė būsena. Turime sąrašą, kuriame yra bent vienas elementas. Naudosime pagalbinį elementą **`elem`** ir statinį kintamąjį **`k`**, skirtą išsaugoti naikinamo sąrašo elemento reikšmę (žr. 2-16 pav.).

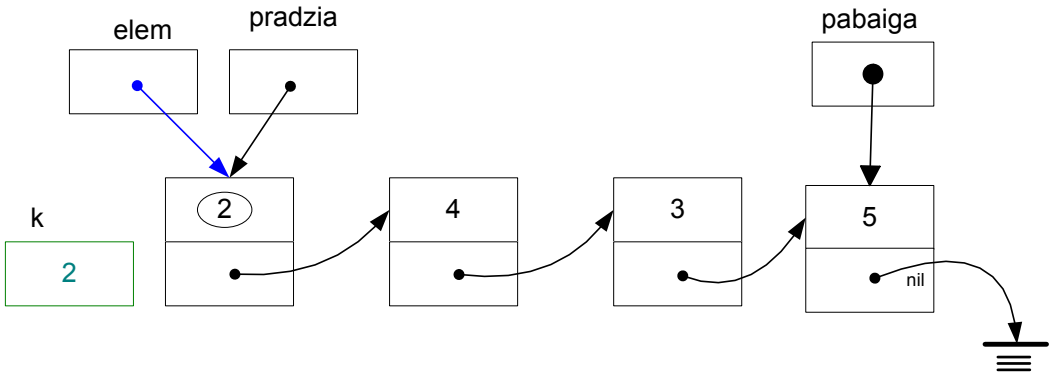


2-16 pav. Elemento naikinimo iš sąrašo pradžios pradinė padėtis

2 žingsnis. Išsaugojame naikinamo elemento duomenį pagalbiniame kintamajame **`k`**. Į išmetamą elementą nukreipiame pagalbinę rodyklę **`elem`**.

Paskalis	C
`k := pradzia^.duom;` `elem := pradzia;`	`k = pradzia->duom;` `elem = pradzia;`

Šias kodo eilutes iliustruoja 2-17 pav.

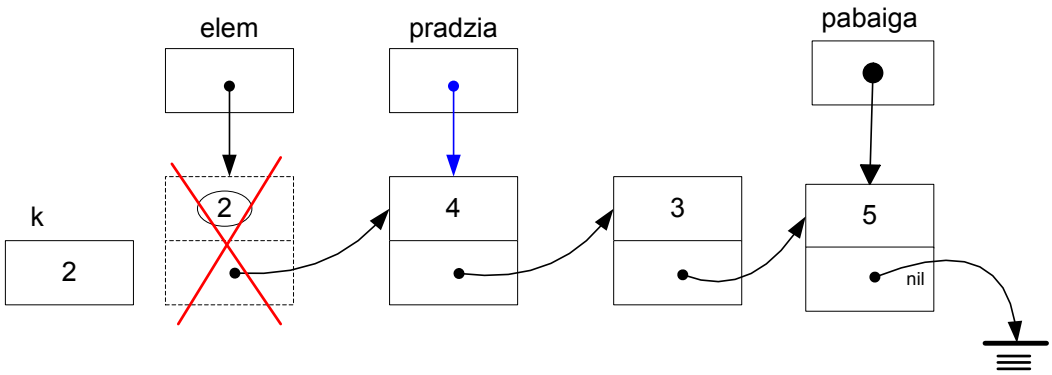


2-17 pav. Elemento naikinimo iš sąrašo pradžios paruošiamieji veiksmai

3 žingsnis. Pradžios rodyklę perkeliame į antrąjį elementą, panaudodami pirmo elemento lauko **kitas** reikšmę. Toliau atlaisviname pirmojo elemento užimamą atmintį.

Paskalis	C
`pradzia := elem^.kitas;` `dispose ( elem );`	`pradzia = elem->kitas;` `free ( elem );`

Atlikus šias kodo eilutes, rodyklė į sąrašo pradžios rodyklė yra nukreipta į naująją sąrašo pradžią, o buvusio pirmojo sąrašo elemento užimama atmintis yra atlaisvinta. Pagalbinė rodyklė **elem** dabar yra neapibrėžta. Šia situaciją iliustruoja 2-18 paveikslas.

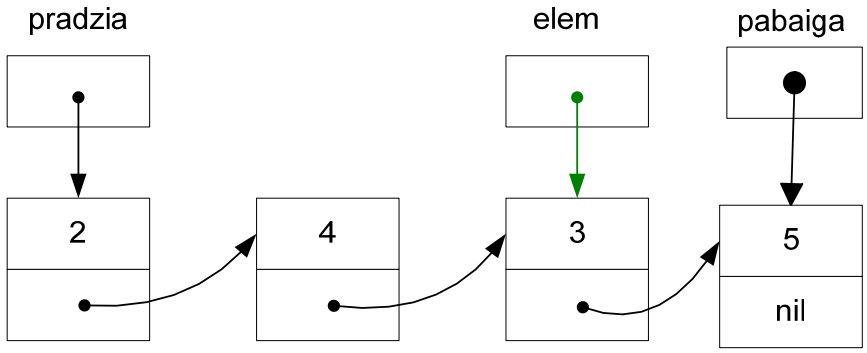


2-18 pav. Elemento naikinimas iš sąrašo pradžios

2.1.7. Elemento panaikinimas sąrašo pabaigoje

Jei paprasčiausiai išmesti paskutinį elementą, į kurį rodo **pabaiga**, tai pamesime sąrašo vientisumą. Todėl iš pradžių reikia rasti priešpaskutinį elementą ir į jį nukreipti rodyklę **elem** (kaip tai padaryti, aprašyta 2.1.10 sk.).

Elemento naikinimo sąrašo pradžioje pradinė padėtis pavaizduota 2-19 paveiksle.

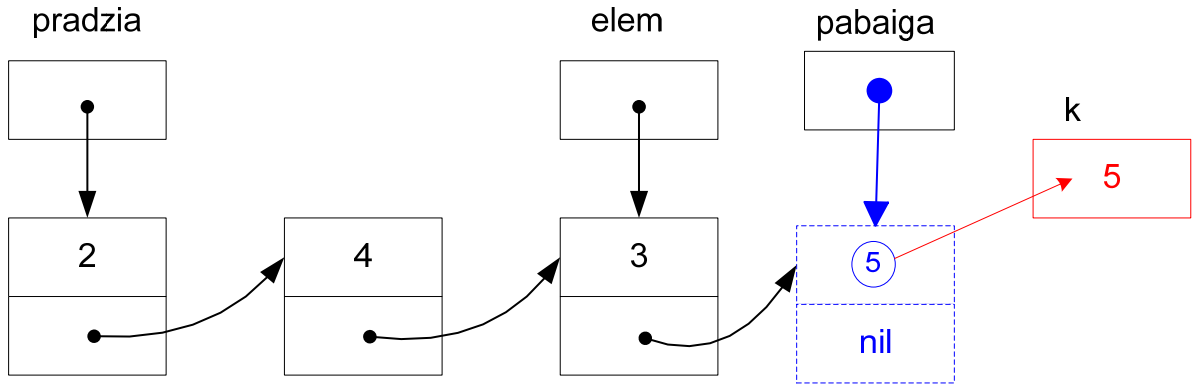


2-19 pav. Elemento naikinimo sąrašo pabaigoje pradinė padėtis

2 žingsnis. Dabar galima naikinti paskutinį elementą, išsaugojant, jei reikia, duomenį kintamajame.

Paskalis	C
`k := pabaiga^.duom;` `dispose ( pabaiga );`	`k = pabaiga->duom;` `free ( pabaiga );`

Šiose kodo eilutėse išsaugotas paskutinio sąrašo elemento duomuo ir atlaisvinta jo užimama atimintis:

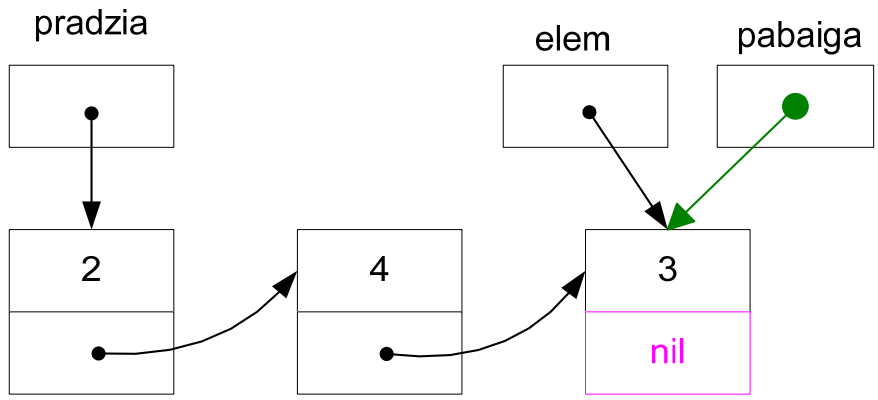


2-20 pav. Atminties atlaisvinimas, naikinant elementą

3 žingsnis. Sąrašo pabaigos rodyklę reikia nustatyti į naują sąrašo pabaigą, tai yra į buvusį priešpaskutinį elementą ir nustatyti jam tuščią adresą laukui **kitas**.

Paskalis	C
`pabaiga := elem;` `pabaiga^.kitas := nil;`	`pabaiga = elem;` `pabaiga->kitas = NULL;`

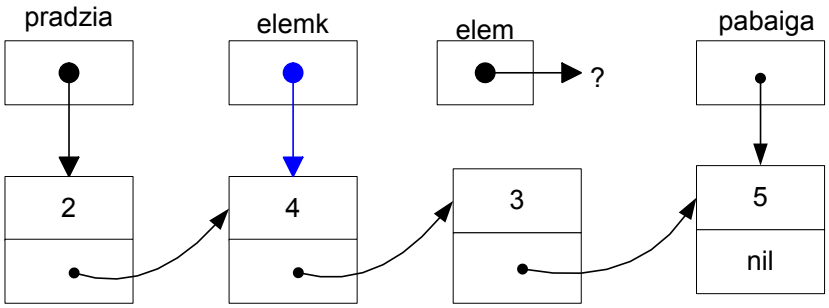
Šias kodo eilutes iliustruoja 2-21 pav.



2-21 pav. Sąrašo pabaigos rodyklės nukreipimas

2.1.8. Panaikinti elementą iš sąrašo vidurio

Pradinė padėtis. Prieš elemento naikinimą reikia nustatyti rodyklę **elemk** į elementą, stovintį *prieš* naikinamą elementą (kaip tą atlikti aprašyta 2.1.9 sk.).

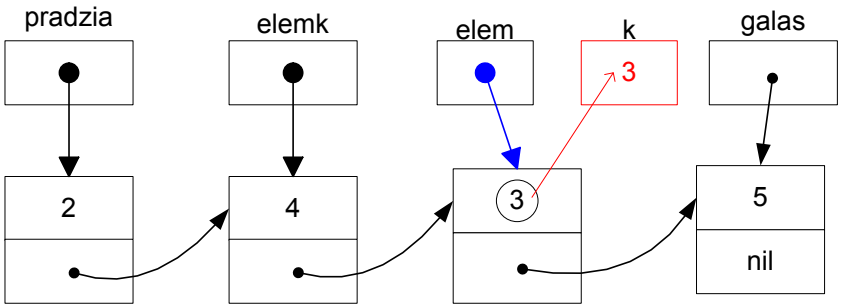


2-22 pav. Elemento naikinimo iš sąrašo vidurio pradinė padėtis

2 žingsnis. Papildoma nuoroda **elem** nustatoma į naikinamą elementą ir išsaugojama naikinamo elemento reikšmė.

Paskalis	C
`elem := elemk^.kitas;` `k := elem^.duom;`	`elem = elemk -> kitas;` `k = elem -> duom;`

Šiose kodo eilutėse atliekamus veiksmus iliustruoja 2-23 paveikslas.

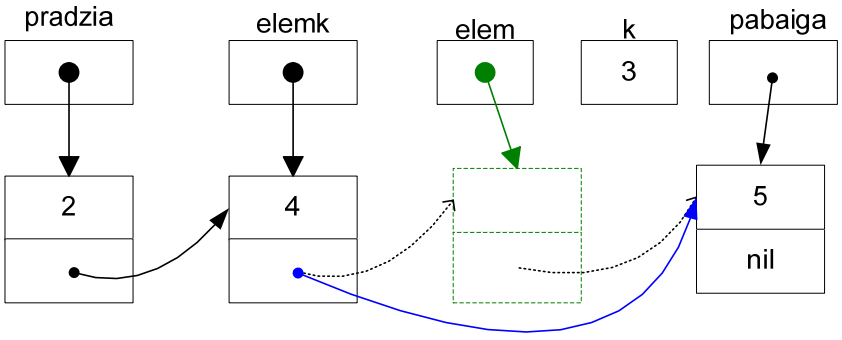


2-23 pav. Papildomos rodyklės į naikinamą elementą nustatymas

3 žingsnis. Naikiname elementą.

Paskalis	C
`elemk^.kitas = elem^.kitas;` `dispose ( elem );`	`elemk->kitas = elem->kitas;` `free ( elem );`

Šias kodo eilutes iliustruoja 2-24 pav. Atlaisvinus naikinamo elemento užimamą atmintį, rodyklė tampa neapibrėžta.



2-24 pav. Elemento naikinimas iš sąrašo vidurio

2.1.9. Nukreipti rodyklę į k-ąjį sąrašo elementą

Neįmanoma vienu operatoriumi nustatyti rodyklę į i-tąjį elementą, kadangi turime tik rodykles, rodančias į pirmąjį ir paskutinį elementus. Todėl reikia paskaičiuoti k elementų nuo sąrašo pradžios, nuosekliai perkeliant rodyklę nuo vieno elemento prie kito.

Atrodytų, kad čia tiktų paprasčiausias ciklas, tačiau jis nerekomenduojamas naudoti:

Paskalis	C

`elem := pradzia;`
`for i := 2 to k do`
`  elem := elem^.kitas;`
	`elem = pradzia;`
	`for ( i = 2; i <= k; i++ )`
	`  elem = elem->kitas;`

Šiame cikle nėra tikrinamas atvejis, kai sąraše yra mažiau nei k elementų. Todėl bendru atveju, reikia tikrinti, ar dar nepasiekta sąrašo pabaiga. Paskalyje šis ciklas galėtų atrodyti taip:

```
elem := pradzia;
i = 2;
while ( i < k ) and ( elem <>nil ) do
  begin
    elem := elem^.kitas;
    inc ( i );
  end;
if i = k
  then writeln('Rodyklė nukreipta į', k, ' -ąjį elementą.')
else writeln('Sarase yra mažiau nei',  k, , elementu.');
```

Pateikiame šį kodo fragmentą C kalba:

```
elem = pradzia;
i = 2;
while ( i < k && elem != NULL)
   {
     elem = elem->kitas;
     i++;
   }
if( i == k )
     printf ("Rodyklė nukreipta į %d-ąjį elementą.\n", k);
else printf("Sąraše yra mažiau nei %d elementų.", k);
```

2.1.10. Rodyklės nukreipimas į priešpaskutinį elementą

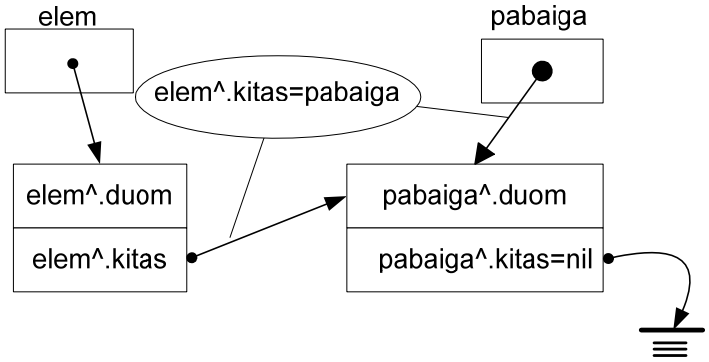
Šis veiksmas reikalingas paskutinio sąrašo elemento pašalinimui. Kadangi rodyklės vienpusiame sąraše rodo tik kitą elementą, tai ateiti prie priešpaskutinio elemento galima tik einant nuo pradžios. Kaip eiti per sąrašą, jau buvo nagrinėta. Liko tik patikslinti ciklo sąlygą. Tai padaryti galima dviem būdais.

1 būdas. Einamojo elemento lauką **`kitas`** galima lyginti su rodykle **`pabaiga`**, kuri rodo į paskutinį elementą:

Paskalis	C

`elem = pradzia;`
`while elem^.kitas <> pabaiga do`
`elem := elem^.kitas`
	`elem = pradzia;`
	`while ( elem->kitas != pabaiga)`
	`elem = elem -> kitas;`

Šios sąlygos tikrinimą iliustruoja 2-25 pav.



2-25 pav. Priešpaskutiniojo elemento paieška: tikrinama sąlyga, ar kitas elementas nėra paskutinis

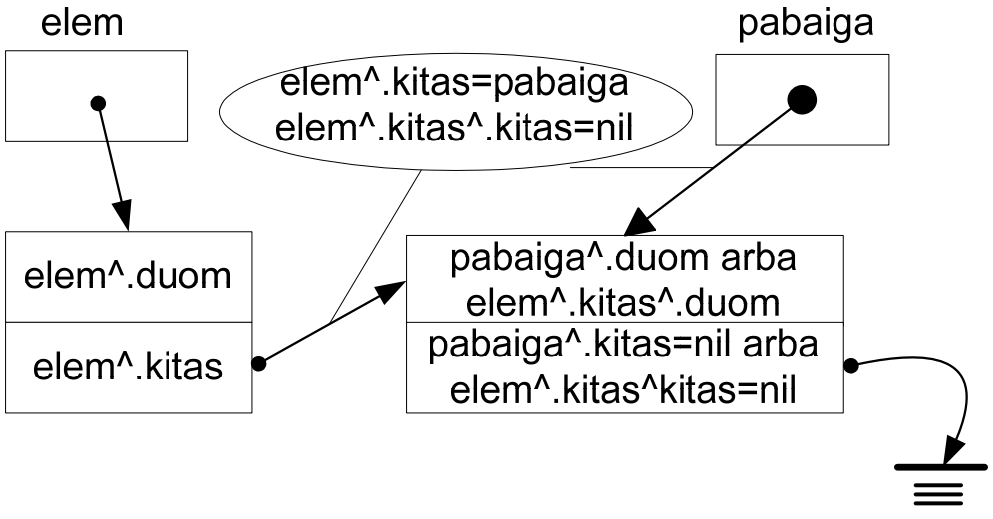
Po ciklo rodyklė **`elem`** yra nukreipta į priešpaskutinį elementą.

2 būdas. Galime tikrinti, ar kito elemento laukas nėra tuščias adresas **`nil`** arba **`NULL`**.

Paskalis	C

`elem = pradzia;`
`while elem^.kitas^.kitas <> nil do`
`elem := elem^.kitas`
	`elem = pradzia;`
	`while ( elem->kitas->kitas != NULL)`
	`elem = elem->kitas;`

Šį kodo fragmentą iliustruoja 2-26 pav.



2-26 pav. Tikrinama sąlyga, ar kito elemento lauko „kitas“ reikšmė yra „nil“

Šiame variante naudojame rodyklę **`el^.kitas^.kitas`**, kuri gana sunkiai suprasti. Šis atvejis yra universalesnis ta prasme, kad nėra naudojama rodyklė **`pabaiga`**.

2.1.11. Atspausdinti vienpusio sąrašo elementus nuo pradžios

Šis veiksmas atliekamas analogiškai tik ką nagrinėtam atvejui. Pereinamas sąrašas ir atspausdinamas kiekvieno elemento duomuo:

Paskalis	C

`elem = pradzia;`
`while elem <> nil do`
`begin`
`writeln ( elem^.duom );`
`elem := elem^.kitas`
`end;`
	`elem = pradzia;`
	`while ( elem != NULL )`
	`{`
	`puts ( elem->duom );`
	`elem = elem^.kitas;`
	`}`

2.1.12. Atspausdinti sąrašo elementus nuo pabaigos

Tai netrivialus atvejis vienpusiam sąrašui. Sprendimas galėtų būti rekursinė procedūra:

Paskalis	C

`procedure PrintList ( elem:rod );`
`begin`
`if elem <> nil then`
`begin`
`PrintList ( elem^.kitas );`
`writeln ( elem^.duom )`
`end`
`end;`
 	
kviečiame procedūrą taip:
`PrintList ( pradzia);`
	`void PrintList( struct el *elem )`
	`{`
	`if (elem != NULL)`
	`{`
	`PrintList( elem->kitas );`
	`printf( "%d\n", elem->duom )`
	`}`
	`}`
	
	kviečiame funkciją taip:
	`PrintList ( pradzia);`

2.2. Dvipusis (dvikryptis) sąrašas

2.2.1. Aprašas

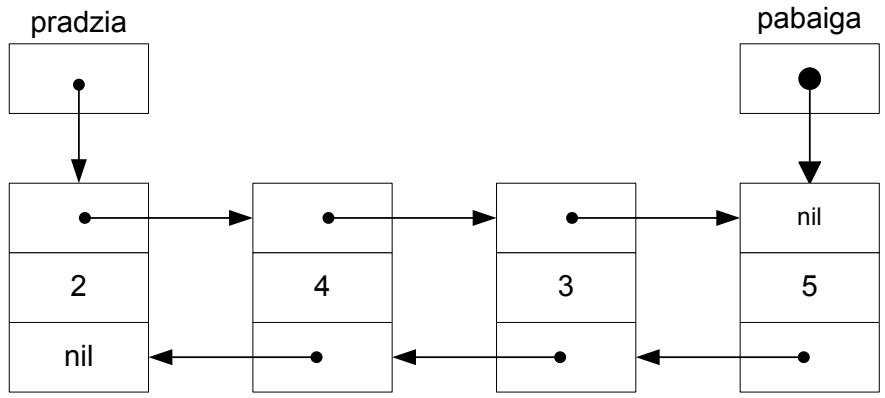
Dvipusis sąrašas – tai struktūra, kurioje yra dvi rodyklės:

Paskalis

```
type dvipSar  = ^dvipElem;
     dvipElem = record
                  priekin: dvipSar;
                  duom   : integer;
                  atgal  : dvipSar;
                end;
```

C

```
struct dvipElem {
                  struct dvipElem *priekin;
                  int duom;
                  struct dvipElem *atgal
                }
```



2-27 pav. Dvipusis sąrašas

Darbui su dvipusiu sąrašu skirti šie kintamieji:

- Rodyklė į sąrašo pradžią: **`pradzia,`**
- Rodyklė į sąrašo pabaigą: **`pabaiga,`**
- Rodyklės į elementus, su kuriais yra atliekami veiksmai: **`elem`** ir **`elemk`**.

Paskalis

```
var pradzia, pabaiga, elem, elemk: dvipSar;
```

C

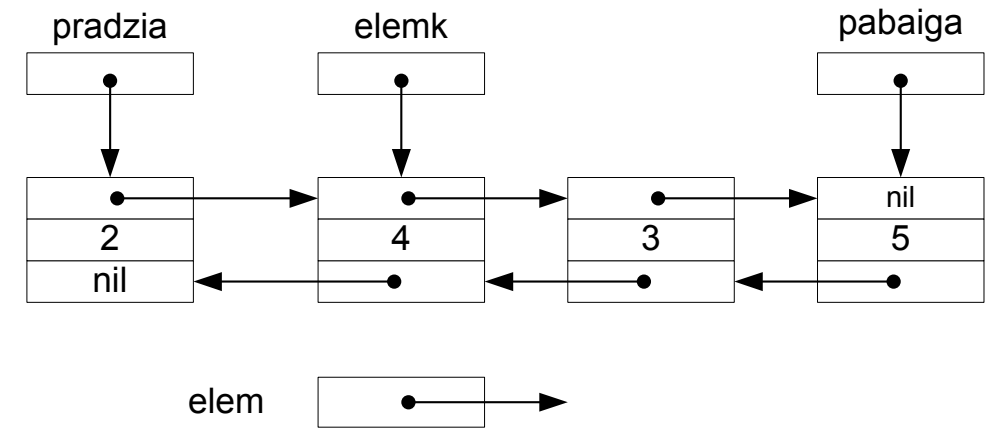
```
struct dvipElem *pradzia, pabaiga, *elem, *elemk;
```

Operacijų rinkinys yra toks pat, kaip ir vienpusio sąrašo atveju. Jos yra realizuojamos analogiškai. Panagrinėsime naikinimo ir įterpimo į sąrašo vidų operacijas, nes jose reikia tvarkyti dvi nuorodas vietoje vienos, kaip buvo vienpusio sąrašo atveju.

2.2.2. Įterpti elementą dvipusio sąrašo viduje po k-ojo elemento

Pradinė būsena: į k-ąjį elementą turi būti nukreipta pagalbinė rodyklė **`elemk`** (kaip tą atlikti vienpusiam sąrašui aprašyta 2.1.9 sk.). Dvipusiame sąraše rodyklę kitas reikia pakeisti rodykle **`priekin`**.

Ši situacija yra pavaizduota 2-28 pav.

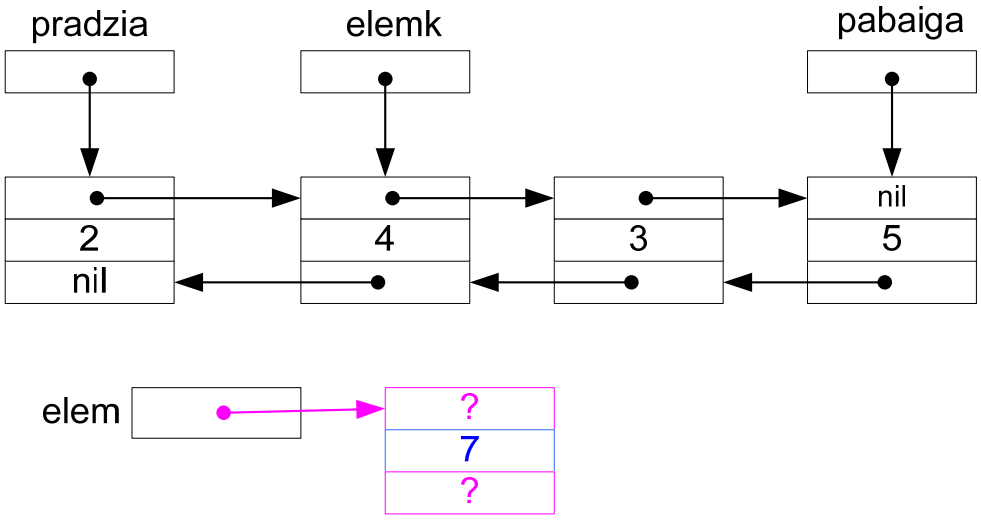


2-28 pav. Elemento įterpimo į sąrašo vidurį pradinė padėtis

2 žingsnis. Sukuriame elementą

Paskalis	C
`new ( elem );`	`elem = (struct dvipElem *) malloc ( sizeof ( struct dvipElem ) );`
`elem^.duom := 7;`	`elem -> duom = 7;`

Šias kodo eilutes iliustruoja 2-29 pav.

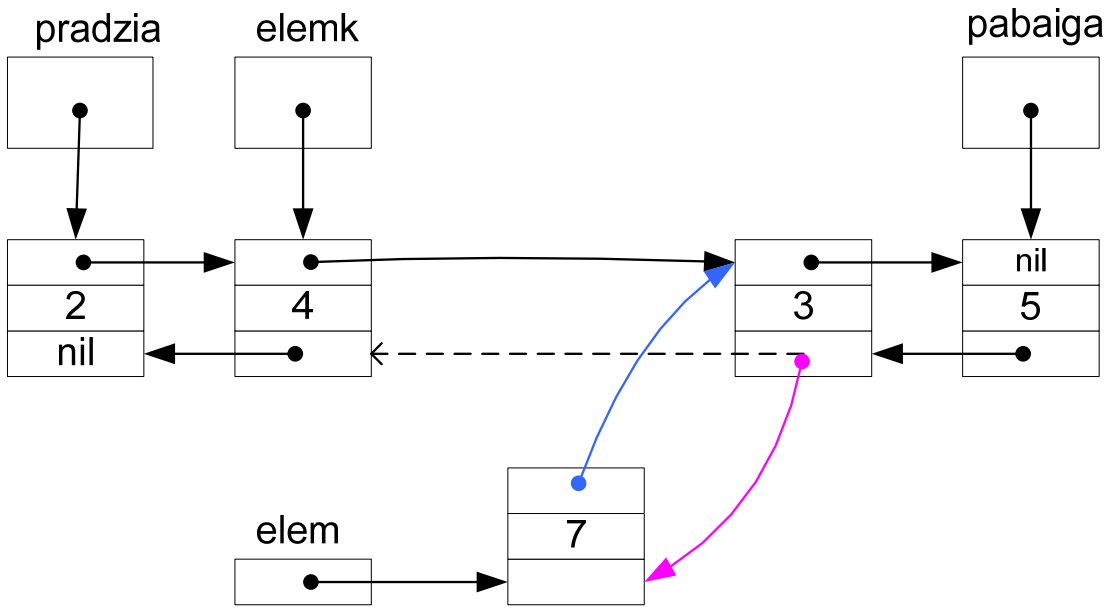


2-29 pav. Elemento įterpimo į dvipusio sąrašo vidurį parengimas: įterpiamo elemento formavimas

3 žingsnis. Įstatome sukurtą elementą po k-ojo elemento ir susiejame su k+1 elementu.

Paskalis	C
`elem^.priekin := elemk^.priekin;` `elemk^.priekin^.atgal :=elem;`	`elem->priekin = elemk->priekin;` `elemk->priekin->atgal = elem;`

Šis kodo fragmentas yra iliustruojamas 2-30 pav.



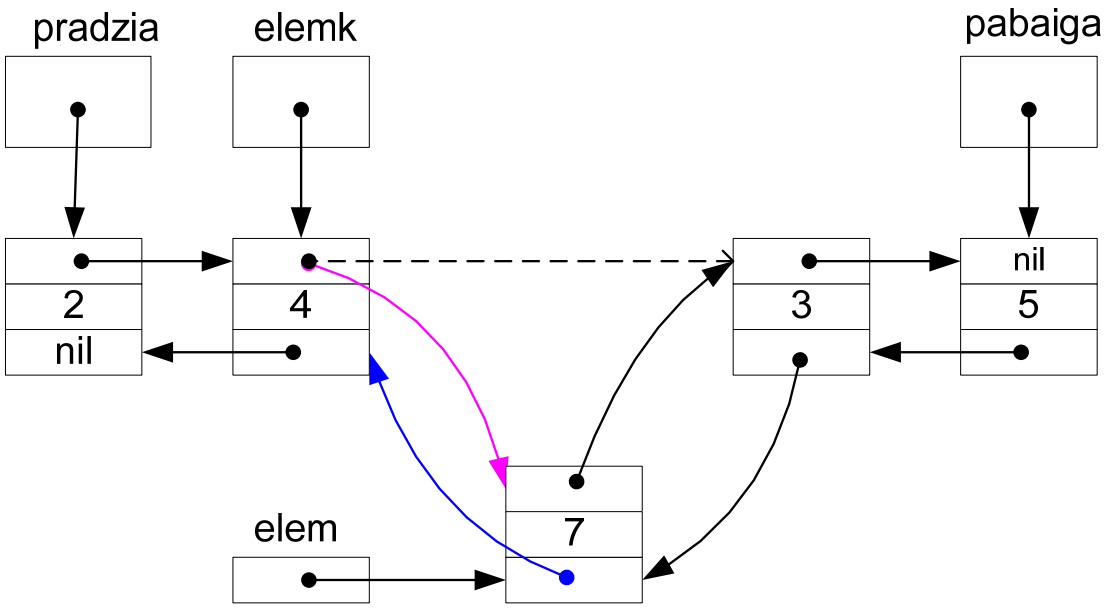
2-30 pav. Elemento susiejimas su k+1-uoju elementu

4 žingsnis. Susiejame sukurtą elementą su k-tuoju elementu.

Paskalis	C

`elem^.atgal:= elemk;`
 	
`elemk^.priekin := elem;`
	`elem->atgal = elemk;`
	
	`elemk->priekin = elem;`

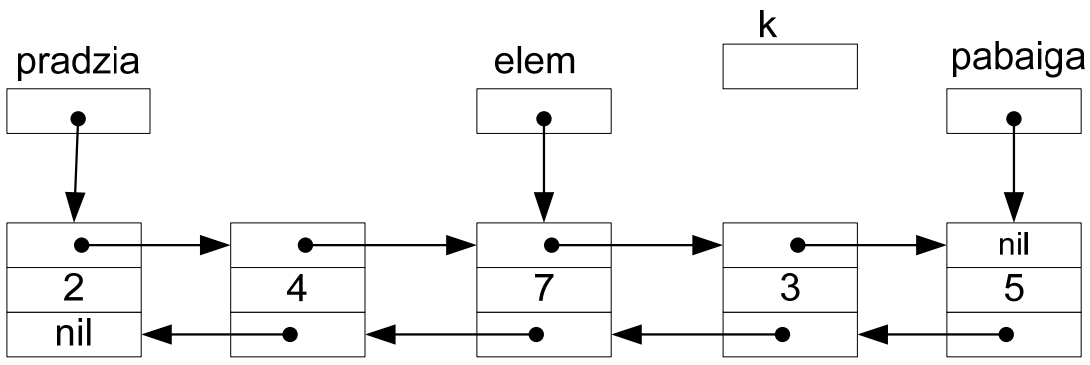
Šis kodo fragmentas yra iliustruojamas 2-31 pav.



2-31 pav. Įterpiamo elemento susiejimas su k-tuoju sąrašo elementu

2.2.3. Panaikinti k-ajį elementą iš sąrašo vidurio

Pradinė padėtis. Papildoma rodyklė **`elem`** turi būti nukreipta į naikinamą k-tąjį elementą (kaip tą atlikti vienpusiam sąrašui yra aprašyta 2.1.9 sk., dvipusiam sąrašui reikia pakeisti rodyklę **`kitas`** į rodyklę **`priekin`**).



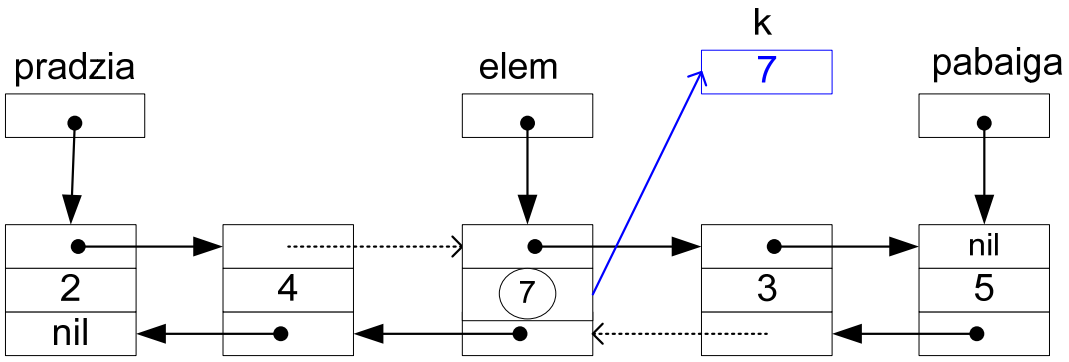
2-32 pav. Elemento naikinimo iš sąrašo vidurio pradinė padėtis

2 žingsnis. Jei toliau programoje bus reikalinga naikinamo elemento reikšmė, išsaugojame k-ojo naikinamo elemento duomenį.

Paskalis	C

`var k : integer;`
…
`k := elem^.duom;`
	`int k;`
	…
	`k = elem->duom;`

Šį kodo fragmentą pavaizduoja **Error! Reference source not found.**



2-33 pav. Naikinamo elemento duomens išsaugojimas

3 žingsnis. Nukreipiame (k+1)-ojo elemento rodyklę atgal į k-1 elementą, o k-1 elemento rodyklę priekin į (k+1) -ąjį elementą.

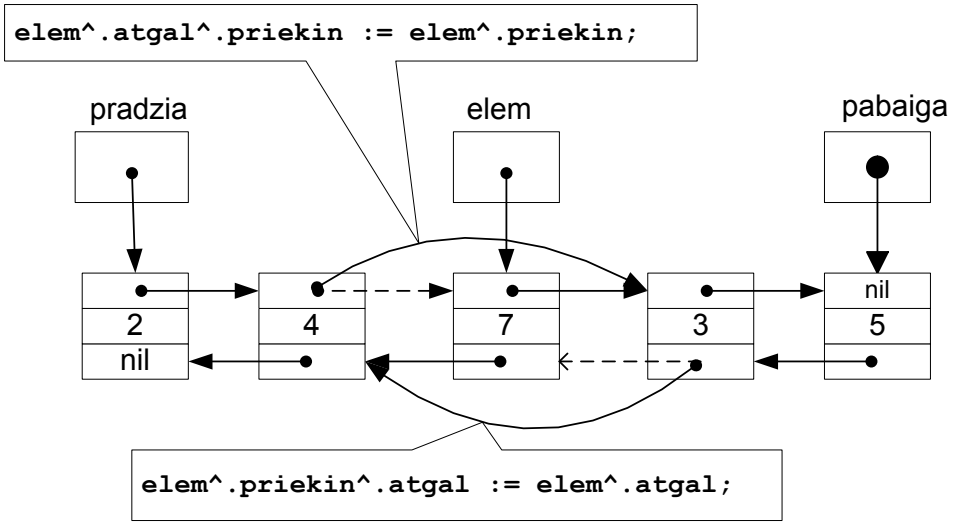
Paskalis

`elem^.priekin^.atgal := elem^.atgal;`
 	
`elem^.atgal^.priekin := elem^.priekin;`

C

`elem->priekin->atgal = elem->atgal;`
 	
`elem->atgal->priekin = elem->priekin;`

Pastarąjį kodo fragmentą iliustruoja 2-34 pav.

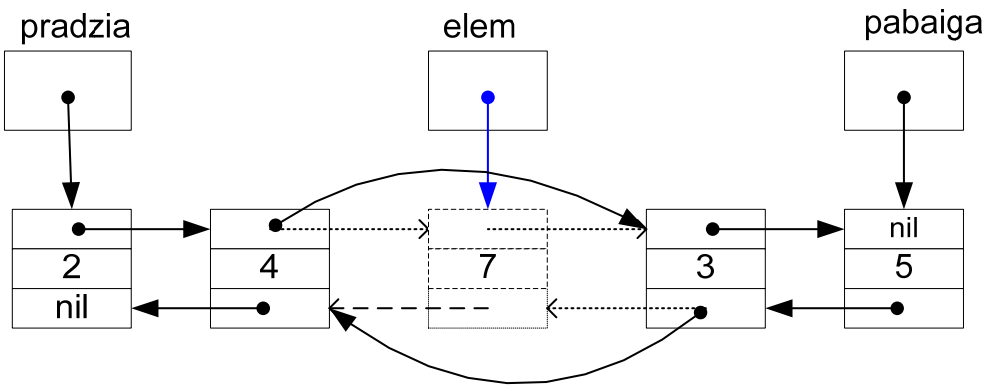


2-34 pav. Elemento naikinimas iš sąrašo vidurio

4 žingsnis. Atlaisviname naikinamo elemento atmintį.

Paskalis	C
`dispose ( elem );`	`free ( elem );`

Šis veiksmas yra iliustruojamas 2-35 pav. Po šio veiksmo rodyklė **`elem`** tampa neapibrėžta.



2-35 pav Galutinis elemento naikinimo veiksmas: atminties atlaisvinimas.

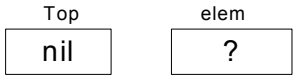
2.3. Stekas

Stekas yra duomenų struktūra, kurioje elementai gali būti įterpiami arba naikinami, tik steko viršūnėje (angl. *top*). Daugiau apie šią duomenų struktūra rekomenduojama paskaityti [2] metodinėje priemonėje. Toliau yra nagrinėjama steko realizacija.

Darbui su steku yra naudojama į steko viršūnę – **`Top`**. Apdoroti einamąjį elementą reikalinga rodyklė **`elem`**. Aprašai pateikti skirsniuose 2.1.1, 2.1.2.

2.3.1. Sukurti steką su vienu elementu

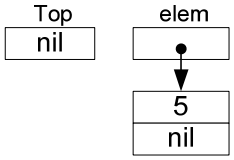
a) tuščias stekas



2-36 pav. Tuščio steko kūrimas

Paskalis	C
`Top := nil;`	`Top = NULL;`

b) Išskirsime atmintį pirmam elementui ir priskiriame duomenį.



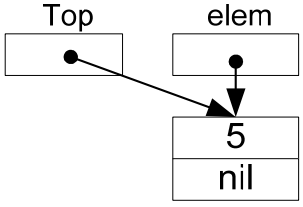
2-37 pav. Naujo elemento įtraukimo į steką paruošimas

2-37 pav. pavaizduoja situaciją, sukurtą šiuo kodo fragmentu:

Paskalis	C

`new ( elem);`
`elem^.duom  := 5;`
`elem^.kitas := nil;`
	`elem = (struct el *) malloc(sizeof(struct el));`
	`elem->duom  = 5;`
	`elem->kitas = NULL;`

c) Steko viršūnę nukreipiame į sukurtą elementą.



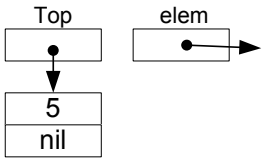
2-38 pav. Steko viršūnės susiejimas su sukurtu sąrašo elementu

Šis kodo fragmentas iliustruoją pateiktą iliustraciją (žr. 2-38 pav.).

Paskalis	C
`Top := elem;`	`Top = elem;`

2.3.2. Įdėti elementą į steką (operacija *push*)

Pradinė padėtis (žr. 2-39 pav.). Steke yra jau vienas elementas. Įdėsime į jį antrą elementą.

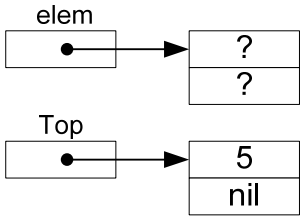


2-39 pav. Operacijos push pradinė padėtis

Išskiriame vietą naujam sąrašo elementui

Paskalis	C
`new ( elem);`	`elem = (struct el *) malloc ( sizeof ( struct el ) );`

Šis kodo fragmentas yra pavaizduotas 2-40 pav.

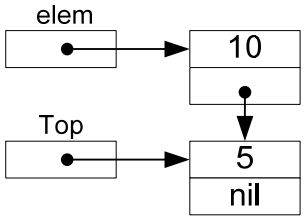


2-40 pav. Naujo elemento, įterpiamo į steką, sukūrimas

c) Priskiriame reikšmę ir susiejame su steku.

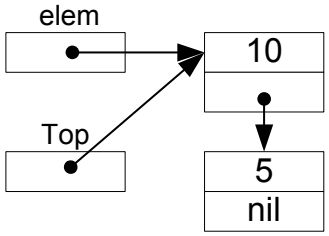
Paskalis	C
`elem.duom := 10;` `elem^.kitas := Top;`	`elem -> duom = 10;` `elem -> kitas = Top;`

Šis kodo fragmentas yra iliustruojamas 2-41 pav.



2-41 pav. Įterpiamo elemento susiejimas su einamąją steko viršūne

d) perkeliame steko viršūnę



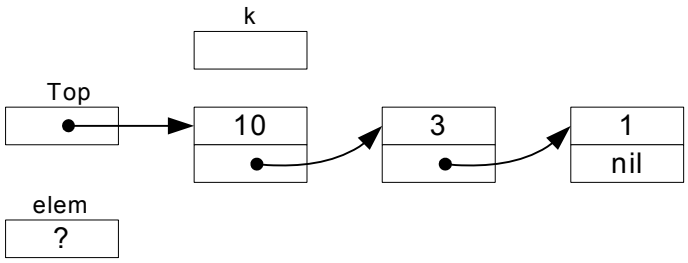
2-42 pav. Steko viršūnės susiejimas su naująja viršūne

Nuorodos perkėlimą atlieka šis kodo fragmentas

Paskalis	C
Top := elem;	**Top = elem;**

2.3.3. Panaikinti steko elementą (operacija pop)

a) Pradinė padėtis: steke yra keli elementai (žr. 2-43 pav.).

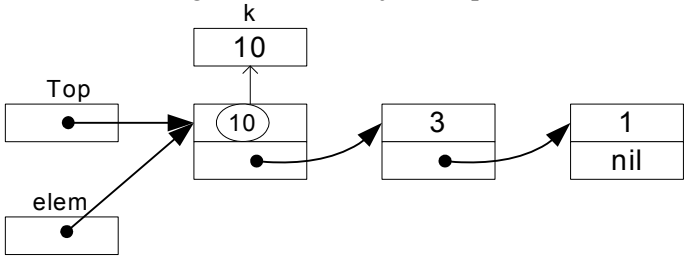


2-43 pav. Steke yra keli elementai

b) Imame duomenį iš steko viršūnės į kintamąjį k ir nukreipiame pagalbinį elementą į steko viršūnę.

Paskalis	C
k := Top^.inf; **elem := Top;**	**k = Top->inf;** **elem = Top;**

Šį kodo fragmentą iliustruoja 2-44 pav.

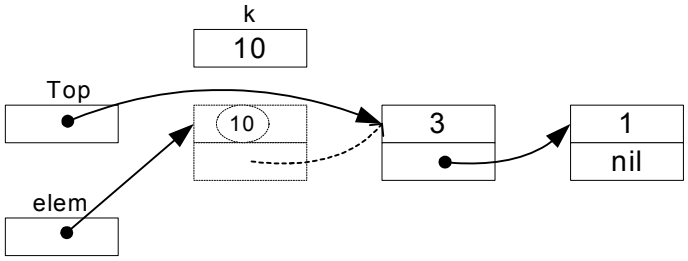


2-44 pav. Nuorodos nukreipimas į naikinamą steko viršūnę

c) Perkelti steko viršūnės rodyklę į antrąjį elementą ir atlaisviname atmintį.

Paskalis	C
`Top := elem^.kitas;` `dispose ( elem );`	`Top = elem->kitas;` `free ( elem );`

Šį kodo fragmentą iliustruoja 2-45 pav.



2-45 pav. Naikinamo elemento atminties atlaisvinimas

2.4. Eilė

Eilės apdorojimui mažiausiai reikia dviejų rodyklių (nuorodų):

- į eiles pradžią, **`e_pradzia`**
- į eilės pabaiga, **`e_pab`** .

2.4.1. Sukurti eilę

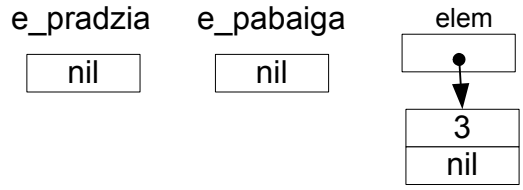
a) Pradinė padėtis: eilės pradžios ir pabaigos rodyklėms yra priskiriamas tuščias simbolis.

Paskalis	C
`e_pradzia := nil;` `e_pabaiga  := nil;`	`e_pradzia = NULL;` `e_pabaiga  = NULL;`

b) Sukuriame pirmą eilės elementą ir priskiriame jam duomenis.

Paskalis	C

`new ( elem );`
`elem^.duom  := 3;`
`elem^.kitas :=nil;`
	`elem =(struct el *) malloc(sizeof(struct el));`
	`elem->duom  = 3;`
	`elem->kitas = NULL;`

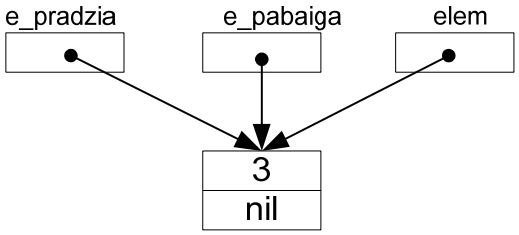


2-46 pav. Pirmo eilės elemento sukūrimas

c) Nukreipiame pradžios ir pabaigos rodykles į sukurtą elementą

Paskalis	C
`e_pradzia := elem;` `e_pabaiga := elem;`	`e_pradzia = elem;` `e_pabaiga = elem;`

Šiuos veiksmus iliustruoja



2-47 pav. Eilė su vienu elementu

2.4.2. Įdėti elementą į eilę

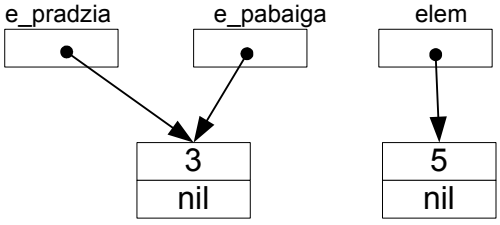
a) Pradinė padėtis: eilėje jau yra bent vienas elementas (žr. **Error! Reference source not found.**).

b) Sukuriame naują elementą ir jo laukams priskiriame reikšmes.

Paskalis	C

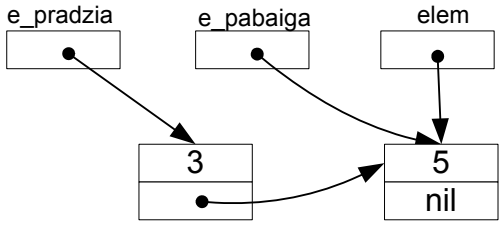
`new ( elem );`
`elem^.duom := 5;`
`elem^kitas := nil;`
	`elem = (struct el*) malloc( sizeof(struct el) );`
	`elem->duom  = 5;`
	`elem->kitas = NULL;`

Šio kodo fragmento veiksmai yra pavaizduoti 2-48 pav.



2-48 pav. Naujo elemento sąrašui paruošimas

c) Nukreipiame paskutinio elemento nuorodą į naują elementą.

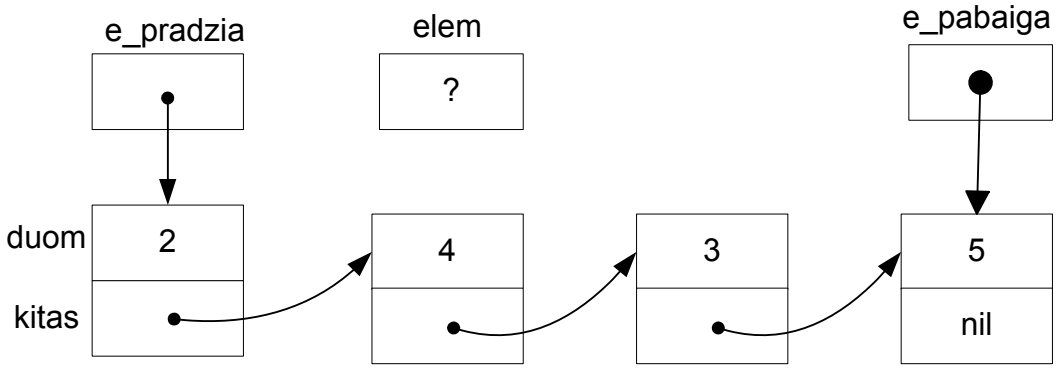


2-49 pav. Elemento prijungimas prie eilės

Paskalis	C
`e_pabaiga^.kitas := elem;` `e_pabaiga := elem;`	`e_pabaiga->kitas = elem;` `e_pabaiga = elem;`

2.4.3. Panaikinti eilės elementą

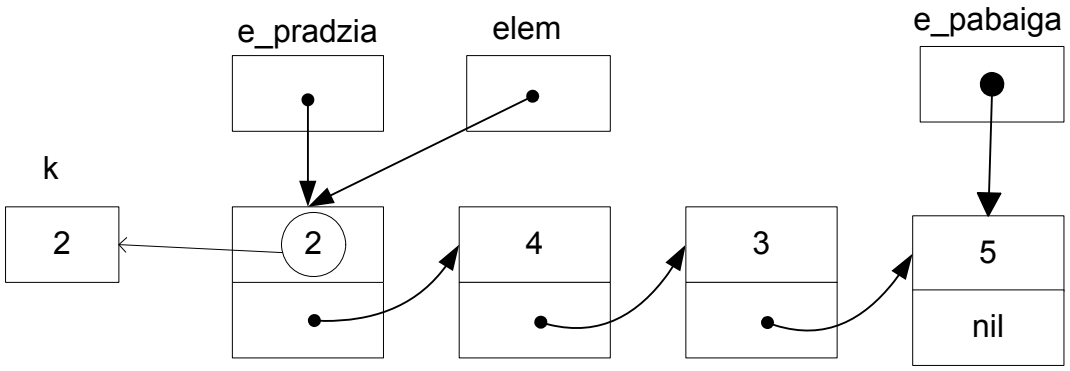
a) Pradinė padėtis: eilėje yra keli elementai (žr. 2-50 pav.):



2-50 pav. Eilė su keliais elementais

b) Išsaugome išmetamo elementa duomenį ir pagalbinę rodyklę nustatome į išmetamą elementą.

Paskalis	C
`k := e_pradzia^.duom;` `elem := e_pradzia;`	`k = e_pradzia->duom;` `elem = e_pradzia;`

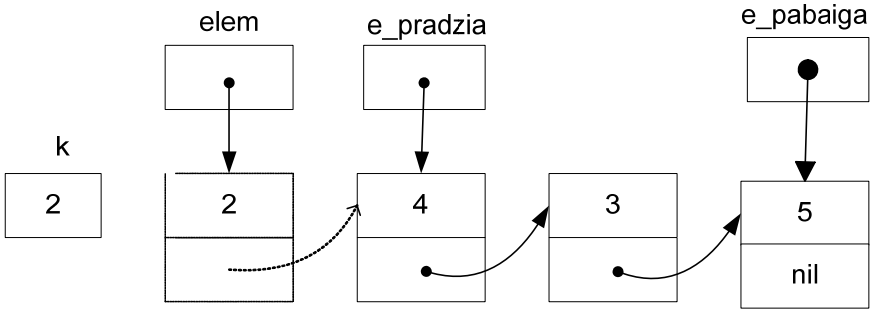


pav. 2-51Naikinamo eilės elemento reikšmės išsaugojimas

c) Perstatome pradžios rodyklę **e_pradzia** į elementą, einantį po elemento, naudojant lauką **kitas**. Po to atlaisviname atmintį, naudojant elementą **elem**.

Paskalis	C
e_pradzia := elem^.kitas; **dispose (elem);**	**e_pradzia = elem->kitas;** **free(elem);**

Pradžios rodyklės perkėlimą iliustruoja 2-52 pav.



2-52 pav. Pradžios rodyklės nustatymas

2.5. Pavyzdžiai

2.5.1. Vienpusio sąrašo sukūrimas ir spausdinimas

Remiantis šiais pavyzdžiai, rekomenduojama pradėti mokytis programuoti dinaminius sąrašus.

2.5.1.1. Paskalio pavyzdys

```
{*********************************************************}
{*   Užduotis nr. XX                                     *}
{*   Sukurti vienpusį sąrašą ir jį atspausdinti.         *}
{*                                                       *}
{*   Atliko: Vardas PAVARDĖ, kursas, grupė, pogrupis     *}
{*                                                       *}
{*********************************************************}
program sukurti_sarasa;
 type sar = ^el;
```

```
      el  = record
         duom : integer;
         kitas : sar;
      end;
              { Kuria sąrašą ir grąžina abi rodykles. }
procedure Sukurti_sar ( var pr, pab : sar );
   var g : sar;
       x : integer;
begin  { Sukurti_sar }
 pr  := nil;    { 1. Is pradžių sukuriame tuščia sąrašą. }
 pab := nil;
 writeln('Kuriame sarasa. Iveskite sveikus skaicius.');
 writeln('Noredami baigti, iveskite 0.');
 readln ( x ); { 2. Formuojame sąrašą}
 while x <> 0 do
    begin
      if pr = nil
      then { 2.1 Pirmasis elementas }
        begin
          new( g );
          g^.duom  := x;
          g^.kitas := nil;
          pr       := g;
          pab      := g;
          readln( x );
        end
      else  { 2.2 Kiti elementai prijuniami į pabaigą. }
        begin
          new( g );
          g^.duom    := x;
          g^.kitas   := nil;
          pab^.kitas := g;
          pab        := g;
          readln( x );
        end
    end;
 writeln('Sarasas sekmingai sukurtas');
end; { Sukurti_sar }

procedure Spausdinti( pr : sar );
 var s : sar;
begin  { Spausdinti }
   writeln('Spausdiname sarasa:' );
   s := pr;
   while s <> nil do
     begin
       write( s^.duom, ' ');
       s := s^.kitas;
     end;
   writeln;
end; { Spausdinti }

procedure Naikinti_sar( var pr, pab : sar );
 var s : sar;
begin  { Naikinti_sar }
   writeln('Naikiname sarasa nuo pabaigos.');
   while pr <> nil do
   begin
      s := pr;
      if pab <> pr
      then
```

```
        begin {Nukreipiame rodyklę i priešpaskutinį elementą.}
          while s^.kitas <> pab do
                s := s^.kitas;
          writeln('  Naikiname ', s^.kitas^.duom);
          pab := s;
          dispose( s^.kitas );
        end
      else
        begin
          writeln('  Naikiname pirma: ', s^.duom);
          dispose( s );
          pr  := nil;
          pab := nil;
        end;
   end;
end; { Naikinti }
var pradzia, pabaiga : sar;
begin
   Sukurti_sar  ( pradzia, pabaiga );
   Spausdinti   ( pradzia );
   Naikinti_sar ( pradzia, pabaiga);
end.
```

2.5.1.2. C kalba:

```
/*********************************************************/
/*   Uzduotis nr. XX                                     */
/*   Sukurti vienpusi sarasa ir ji atspausdinti.         */
/*                                                       */
/*   Atliko: Jonas Jonaitis                              */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <stdlib.h>

  struct el {    int duom;
                 struct el *kitas;
            };
/* Kadangi reikia grąžinti dvi rodykles, būtent pradzią ir pabaigą,
naudosime parametrus-kintamuosius. Pradžia ir pabaiga yra rodyklės į
sąrašo elementą, o į funkcija tuomet nueina jų adresai, tai yra
elemento rodyklės rodyklė arba adreso adresas. */

---

void Sukurti_sar    ( struct el ** , struct el** );
void Spausdinti_sar ( struct el * );
void Naikinti_sar   ( struct el ** , struct el** );

int main()
{
   struct el *pradzia, *pabaiga, *elem;

   Sukurti_sar    ( &pradzia, &pabaiga );
   Spausdinti_sar ( pradzia );
   Naikinti_sar   ( &pradzia, &pabaiga );
   return 0; /* tik Windows */
}

/* Iterpti elementa i saraso pradzia. */

void Sukurti_sar ( struct el * *pr, struct el * *pab )
{
  struct el *g;
  int reiksme;

  puts ("Iveskite saraso elementus. Noredami baigti, iveskite 0:" );
  *pr  = NULL;
  *pab = NULL;
  scanf("%d", &reiksme);
  while( reiksme != 0 )
   {
     if ( *pr == NULL)
       {
       g = (struct el *) malloc(sizeof(struct el));
       g->duom  = reiksme;
       g->kitas = NULL;
       *pab = g;
       *pr  = g;
       }
     else
       {
       g = (struct el *) malloc(sizeof(struct el));

---
```

```
        g->duom  = reiksme;
        g->kitas = NULL;
        (*pab)->kitas = g;
        *pab = g;
        }
      scanf( "%d", &reiksme );
   }
   puts("Sarasas sekmingai sukurtas");
}

/* Spausdinti nuo pradzios */
void Spausdinti_sar (struct el *elem)
{
    int i=1;

    puts("****************************************");
    puts("Spausdiname sarasa:");
    while ( elem != NULL )
      {
      printf( "%d elementas yra %d\n", i, elem->duom );
      elem = elem->kitas;
      i++;
      }
}

/*Naikinti sarasa */
void Naikinti_sar ( struct el * *pr, struct el * *pab )
{
   struct el *einam;

    puts("****************************************");
    puts("Naikiname sarasa nuo pabaigos:");
    while (*pr != NULL)
     {
      einam = *pr;
      if ( *pab != *pr )
        {
         while ( einam->kitas != *pab )
               einam = einam->kitas;
         *pab = einam;
         printf(" %d ", einam->kitas->duom );
         free ( einam->kitas );
        }
      else
        {
          printf( " %d. \n ", einam->duom );
          free ( einam );
          *pr  = NULL;
          *pab = NULL;
        }
     } /* while */
}  /* Naikinti_sar */
```

Modulis realizuoja abstraktaus duomenų tipo sąvoką. Apie šią sąvoką rekomenduoju paskaityti trumpai ir aiškiai išdėstytą medžiagą [2]. Modulis yra savarankiška programos dalis, kuri yra atskirai kompiliuojama. Moduliai yra skirti didelę programą skaidyti dalimis.

3.1. Modulių kūrimas Paskalyje

Moduliai prijungiami, naudojant sakinį

```
Uses modulio_vardas;
```

Modulis susideda iš

- modulio antraštės,
- apibrėžimų dalies – interfeiso,
- veiksmų dalies – realizacijos,
- inicializavimo dalies.

Modulio teksto skirstymas į dalis, iš kurių viena yra matoma vartotojui, sudaro galimybę naudoti modulinio programavimo ir abstrakčiųjų duomenų tipų idėjas.

3.1.1. Antraštė

```
unit modulio_vardas;
```

3.1.2. Apibrėžimų dalis

Apibrėžimo dalyje yra talpinamos tik funkcijų ir procedūrų antraštės. Programa gali naudotis tik tomis funkcijomis ir procedūromis, kurios yra paskelbtos apibrėžimų dalyje.

```
Interface
  uses      modulių_sąrašas;   { naudojamų modulių sąrašas }
  const     konstantų_sąrašas; { naudojamų konstantų sąrašas }
  type      tipų_sąrašas;      { naudojamų naudotojo tipų sąrašas }
  var kintamųjų_sąrašas;       { naudojamų modulio kintamųjų sąrašas}

  procedure Proc1(parameterai); {procedūrų ir funkcijų antraščių skyrius}
  function Func(): tipas;
```

3.1.3. Veiksmų dalis

```
implementation
   {naudojami moduliai}
   funkcijų ir procedūrų aprašai;
```

Veiksmų modulio dalį sudaro funkcijų ir procedūrų aprašai. Programos jų nemato, tik jais naudojasi. Veiksmų dalyje antraštes galima praleisti, jas galima sutrumpinti iki pavadinimo, nes parametrų sąrašas buvo nurodytas apibrėžimų dalyje. Jei antraštė nėra trumpinama, jis turi sutapti su pateikta apibrėžimo skyriuje.

Veiksmų dalyje gali būti aprašytos ir tokios procedūros ir funkcijos, kurių antraščių nėra apibrėžimų dalyje. Tuomet jos gali būti naudojamos tik modulio viduje. Išorinėje programoje jos negali būti naudojamos.

3.1.4. Inicializavimo dalis

Inicializavimo dalis yra panaši į programos sakinių dalį. Ji atliekama prieš vykdant programos, kurioje yra naudojamas modulis, sakinius. Jeigu programoje yra keli moduliai, inicializavimo dalys yra įvykdomos tokia tvarka, kokia moduliai yra išvardinti **uses** sakinyje.

```
begin
   sakiniai
end.
```

3.1.5. Transliavimas

Transliuodami modulius Turbo Paskaliu reikia nustatyti **destination disk**. Atsiranda failas su išplėtimu **.tpu**. Po to transliuojamas pagrindinis failas. Transliuodami kompiuterinėse klasėse pradžioje transliuojame modulį:

```
Fpc funkc.pas
```

Kataloge atsiras failas su išplėtimu .ppu. Po to pagrindinės programos failą ir paleidžiame pagrindinę programą.

3.2. C

3.2.1. Antraščių failų kūrimas

Antraščių failuose yra talpinami kintamųjų ir funkcijų aprašai. Jie turi išplėtimą **.h.** Antraščių failų pavyzdžius jau žinome, būtent **stdio.h**, **stdlib.h** ir kitus. Kadangi antraščių faile yra tik funkcijų prototipai (pirmavaizdžiai), o jų apibrėžtys yra kitur, tai jų vardai turi būti pažymėti kaip **extern**.

Funkcijos yra talpinamos faile su išplėtimu **.c**. Šiame faile negali būti funkcijos **main()**!

Pagrindinėje programoje šios funkcijos yra naudojamos, kai prijungiamas antraščių failas, panaudojant prekompiliatoriaus komandą:

```
#include "pvz.h"
```

Standartiniai antraščių failai yra apskliaudžiami **<stdio.h>**, o naudotojo **"example.h"**. Apskliaudimo būdas pasako, kur yra failas, standartiniuose kataloguose ar darbiniame naudotojo kataloge.

Kompiliatorius verčia programą iš programavimo kalbos į mašininę. Kaip transliuojamos programos iš kelių failų? Ryšių redaktorius (angl. *linker*) – tai programa, kuri yra paleidžiama po kompiliacijos. Ji sujungia visas programos dalis į vieną ir pagamina vykdomąjį modulį.

3.2.2. Kelių failų transliavimas, naudojant gcc

Prieš pagrindinės programos kompiliavimą reikia sutransliuoti visas sudedamąsias dalis į objektinius modulius su išplėtimu **.o**.

```
gcc –c example.c –o example.o
gcc –c main.c –o main.o
```

Toliau sutransliuotos dalys yra sujungiamos ir sukuriamas vykdomas modulis **example**:

```
gcc example.o main.o –o example
```

Jei kažkuris failas buvo pakeistas, reikia jį pertransliuoti ir sujungti su kitomis dalimis iš naujo:

```
gcc -c main.c -o main.o
gcc main.o example.o -o example
```

3.2.3. Transliavimas kitur

Naudotojas turi sukuria projektą (angl. *Project*) ir įtraukti naudojamus failus. Juos reikia atskirai sutransliuoti (tai gali būti atliekama automatiškai), po to surišti (link) ir įvykdyti.

3.3. Pavyzdys

Pateiksime paprastą programėlę, susidedančią iš dviejų failų:

3.3.1. Pascal

Pagrindinė programa:

```
program programa_su_moduliu;
  uses funkc;
  var x:integer;

  begin

     x:=5;
     writeln(x,' kvadratu yra ', square(x));
     writeln(x,' pakeltas 7 laipsniu yra ', power(x,7):7:0);
     writeln(x,' faktorialas yra ', factorial(x));
  end.
```

Modulis:

```
unit funkc;

interface

   function square(x:integer):integer;
   function power(x,y:integer):real;
   function factorial(x:integer):longint;

implementation

   function square(x:integer):integer;
   begin
      square:=x*x
   end;

   function power(x,y:integer):real;
     var ans:real;
         temp,p:integer;
   begin
      temp:=x;
      ans:=x;
      for p:=y-1 downto 1 do
        ans:=ans*temp;
      power:=ans;
   end;

   function factorial (x:integer):longint;
     var temp:integer;
   begin
      for temp:=x-1 downto 1 do
        x:=x*temp;
      factorial:=x
   end;
end.
```

3.3.2. C

Pagrindinė programa:

```
#include <stdio.h>
#include "funkc.h"

int main() {
  int x = 5;

  printf("%d kvadratu yra %d.\n", x, square(x));
  printf("%d pakeltas 7 laipsniu yra %.0f.\n", x, power(x,7));
  printf("%d faktorialas yra %.0f.\n\n", x, factorial(x));

  /* %f yra naudojama isvesti float ir double tipo elementus.
       power() ir factorial() funkcijos grazina sveikus skaicius,
       todel naudojame %.0f */
  return 0;
}
```

Failas **funkc.h**, kuriame yra funkcijų antraštės (prototipai):

```
#ifndef FUNK_H
#define FUNK_H

extern int square(int x);
extern double power(int x, int y);
extern double factorial(int x);
#endif
```

Failas **funkc.c**, kuriame yra funkcijų aprašai:

```
#include <stdio.h>
// Kadangi tipas yra standartinis, prijungti funkc.h nereikia. Jei modulis
// ir tipą, tuomet būtina prijunti antraščių bylą #include "funkc.h"


int square(int x) {
  return (x * x);
}

double power(int x, int y) {
  double ans = x;
  int temp = x;
  int p;

  for(p=y-1 ; p!=0 ; p--) {
    ans*=temp;
  }
  return ans;
}

double factorial(int x) {
  int temp;
  for(temp=x-1 ; temp!=0 ; temp--) {
    x*=temp;
  }
  return x;
}
```

4. Naudingi standartiniai modulial

4.1. DOS modulis (Paskalyje)

Šį modulį sudaro priemonės, leidžiančios TP programuose naudotis operacinės sistemos galimybėmis. Tai įvairios veiksmų su diskais, failais, pertraukimų apdorojimo operacijos ir pan.

4.1.1. Veiksmai su data ir laiku

Pateikia sisteminę datą. Savaitės diena yra nurodoma skaičiais nuo 0 iki 6:

```
GetDate (var Metai, Menuo, Diena, sav_diena :word);
```

Pateikia sistemos laiką:

```
GetTime(var Val, Min, Sek, Sek100: word);
```

4.1.2. Veiksmai su failais

4.1.2.1. Paieška

Kartais reikia patikrinti, ar diske yra ieškomi failai, kiek jame yra laisvos vietos ir panašiai. Procedūra FindFIrst randa pirmąjį failą kataloge, atitinkantį nurodytas savybes:

```
Procedure FindFirst( Kelias:string; Savybes: word;
                                   var Results: SearchRec);
```

Parametras *Kelias* nurodo paieškos katalogo kelią ir ieškomo failo pavadinimo šabloną, kuriame galima naudoti simboliu * ir ?, pavyzdžiui, '***.***'

Parametras *Savybes* nurodo ieškomųjų failų savybes. Jų reikšmės yra aprašytos modulio DOS konstantomis:

```
const    ReadOnly=$01;              Directory=$10; {Katalogai.}
      Hidden=$02;                        Archive=$20;   {Supakuotos
      bylos.}
      SysFiles=$04;              AnyFile=$3F;
      VolumeID=$08;
```

Jei reikia kelių savybių, tuomet jos yra sumuojamos, kaip antai **Hidden+Sysfiles**

In addition to producing reports, this template can be used to create proposals and workbooks. To change the text or graphics, the following suggestions are provided.

Paieškos rezultatas yra pateikiamas SearchRec tipo kintamojo **Results** laukuose. Pastarojo tipo struktūra yra aprašoma taip:

```
Type     SearchRec=record
file : array [1..21] of byte;    {OS duomenys.}
attr : byte;                                    {Požymių kodų
sumos.}
time : longint;                                 {Sukūrimo data
ir laikas.}
size : longint;                                 {Failo dydis
bitais.}
name : string[12];                          {Failo vardas ir
prievardis.}
```

Kartoti paiešką su tais pačiais parametrais:

```
Procedure FindNext(var Results: SearchRec);
```

Ar paieška yra sėkmingai vykdoma rodo kintamojo **DosError** reikšmė. Juo remiantis galima sužinoti klaidos priežastis. Jei paieška yra sėkinga, tai **Doserror** reikšmė yra lygi **0.**

```
program katalogo_turinys;
   uses Dos;
   var info: SearchRec;
       i : integer;
       e : string;
   begin
        e := '*.*';  `\*.*` DOSe Pagrindinis disko katalogas.}
        Findfirst(e, anyfile, info);  {Pirmojo failo paieska.}
        writeln('Katalogas: ':30, e );
        i := 0;
        while DosError = 0 do
           begin
              inc(i);
              if i=24 then
                 begin
                    write('Kita lenteles dalis',
                          '--- po Enter paspaudimo.');
                    Readln;
                    i := 0;
                 end;
              writeln(info.name:12, info.attr:3, info.size:8);
              FindNext(info);
           end;
   end.
```

Šios programos išvestis Linux yra tokia:

```
                  Katalogas: *.*
          . 16       0
         .. 16       0
  DIR_MANO.C 32     966
       a.out 32    4976
dir_mano.rez 32      45
    SYSTEM.C 32     448
  system.rez 32     289
     DIR.PAS 32     737
      stat.c 32    2489
         dir 32  123520
     dir.rez 32     518
PARAMSTR.PAS 32     438
 dir_stand.c 32     852
  TIME_KML.C 32    1254
 DOSDEMO.PAS 32    1042
dir_stand.rez 32     157
       dir.o 32    2304
Time_kml.rez 32      86
   direk.rez 32     467
```

Katalogą, kuriame yra tam tikras failas, randa funkcija

```
Function Fexpand(<Failas>:Pathstr): PathStr;
```

Ši funkcija dalinį bylos kelią išplėčia iki pilno kelio.

Pilno kelio pavadinimas dalimis skaido procedūra:

```
Procedure Fsplit (pilnas_failo_pavadinimas : PathStr;
                                  var kelias : DirStr;
                                  var vardas : Namestr;
                                  var prievardis : Extstr; );
```

Procedūra pilną failo vardą (su katalogais) suskaido į dalis, kurias grąžina parametrai **kelias**, **vardas** ir **prievardis**.

4.1.2.2. Failų savybes ir jų keitimas

```
Procedure GetFAttr (var failo_kintamasis; var savybes: word);
Procedure SetAttr  (var failo_kintamasis; var savybes: word);
```

Failų savybės yra koduojamos taip pat, kaip ir FindNext atveju. Failas negali būti atvertas.

Failo paieška tam tikrame katalogų rinkinyje:

```
function FSearch(Failo_vardas: PathStr; katalogu_rinkinys ;
                                        string):PathStr;
```

Failų tvarkymo funkcijose iir procedūrose yra naudojami šie tipai:

```
Type Pathstr = string[79];   {pilnas failo vardas}
     DirStr = string[67];    {katalogas}
     Namestr = string[8];    {failo vardas}
     ExtStr = string[4];     {prievardis}
```

4.1.2.3. Duomenys apie disko talpą

```
function DiskFree(diskas : byte):longint;
```

Ši funkcija grąžina laisvos atminties kiekį baitais. Disko A: numeris 0, B: – 1, C: – 2 ir t.t.

```
function Disksize (diskas : byte): longint;
```

Funkcija grąžina disko talpą baitais.

```
function ParamStr(indeksas : word);
```

Šui funkcija grąžina veikiamosios programos vardas.

```
program failo_vardas;
   uses Dos;
   var pilnas_vardas : string;
       katalogas : DirStr;
       vardas : NameStr;
       prievardis : extStr;
   begin
       pilnas_vardas := ParamStr(0);
       Fsplit(pilnas_vardas, katalogas, vardas, prievardis);
       writeln('Veikiamasis katalogas yra: ', katalogas);
       writeln('Programos failo vardas : ', vardas);
       writeln('Failo prievardis yra : ', prievardis);
   end.
```

Šios funkcijos išvestis:

```
Veikiamasis katalogas yra: Informatika/konsole/
Programos failo vardas : paramstr
Failo prievardis yra :
```

4.1.2.4. Daugiau Dos modulio galimybių demonstruojantis pavyzdys

```
program DosDemo;
{ modulio DOS galimybiu demonstracija }

uses
  Crt,Dos;
```

```
var
  ch : char;

procedure Dir;
{ parodo veikiamojo katalogo turini }

var
  katalogas : DirStr; { Modulio Dos tipas }
  failoSav : SearchRec; { Modulio Dos tipas }

begin
  GetDir(0,katalogas); {Koks yra veikiamasis katalogas?}
  writeln('Veikiamasis katalogas: ',katalogas);
  writeln;
  FindFirst('*.*',Archive,failoSav); { randa pirma faila }
  while DosError = 0 do
    begin
      writeln(failoSav.Name);
      FindNext(failoSav) { iesko kitu failu}
    end;
  writeln;
  writeln(DiskFree(0),' laisvu baitu is ',DiskSize(0),
          ' galimu.')
end;

procedure Parodyk_laika;
{ rodo sistemini laika }

var
  val,min,sek,simt : word;

begin
  writeln;
  write('Siuo metu yra: ');
  repeat
    GetTime(val,min,sek,simt);
    write(val:2,':',min:2,':',sek:2,'.',simt:2);
    GotoXY(25,WhereY)
  until KeyPressed
end;

begin { programa }
  ClrScr;
  Dir;
  ch:=ReadKey;
  Parodyk_laika
end. { programa DOSDemo }
```

Programos išvestis mano kompiuteryje (Borland Pascal aplinkoje neveikia!!)

```
Veikiamasis katalogas: /users/moroz/Informatika/konsole

DIR_MANO.C
a.out
dir_mano.rez
SYSTEM.C
system.rez
DIR.PAS
dir
dir.o
dir.rez
PARAMSTR.PAS
paramstr
paramstr.o
DOSDEMO.PAS
dosdemo.rez
dosdemo.o
dosdemo
```

```
1306564608 laisvu baitu is 10569627648 galimu.

Siuo metu yra: 11:31: 7.52
```

4.2. Standartinės C funkcijos darbui su laiku ir data

ANSI C funkcijų aibei priklauso keletas funkcijų su laiku, kurių aprašai yra patalpinti time.h antraščių faile.

time_t duomenų tipas skirtas kintamiesiems, kuriuos naudoja laiko funkcijos.

Sisteminį laiką sekundėmis grąžina funkcija

```
time_t time (time_t *time);
```

Lokalus ir Grinvičo laikas yra gaunamas, naudojant funkcijas

```
struct tm *localtime (const time_t * timer);
struct tm * gmtime (const time_t *timer);
```

Tuomet laikas yra verčiamas į įprastą pavidalą funkcijomis

```
char * asctime (const struct tm *tblock);
char * ctime (const time_t *time);
```

Ctime verčia laiką į 26 simbolių eilutę, kurios pabaigoje yra naujos eilutės simbolis '\n' ir eilutės pabaigos simbolis '\0'. Jos formatas yra pateikiamas toliau pavyzdyje.

Funkcija **strftime** formatuoja laiko išvestį.

Pavyzdys, iliustruojantis laiko funkcijas:

```
/*********************************************************************
 * Tikslas: demonstruoti time.h funkcijas.
 * Failas: time_kml.c
 *********************************************************************/

#include <stdio.h>
#include <time.h>
int main()
   {
   time_t now;               /* naudojame time_t tipa  time modulio  */
                        /* Kalendorinis laikas yra sekundziu skaicius nuo
                           1/1/1970         */

   now = time((time_t *)NULL);             /* gauna sistemini laika ir
                               * patalpina ji kintamajame 'now' */

   printf("\n%s", ctime(&now)); /* Vercia 'now' i simb. eilute su

   '\n' */
   /*******************************************************/
   /* dar vienas budas suzinoti laiko ir datos informacija     */

   time(&now);
   printf("%s", ctime(&now));       /* Format data in 'now'          */
  /*******************************************************/
/* trecias budas: */

   time(&now);
   printf("%s", asctime(localtime( &now  )));

/*************************************************************/
   {
   struct tm *l_time;
   char string[20];
```

```
time(&now);
l_time = localtime(&now);
strftime(string, sizeof string, "%d-%b-%y\n", l_time);
printf("%s", string);
}
 return 0;
}
```

Programos išvestis mano kompiuteryje yra tokia:

```
Mon Mar 29 12:57:24 2004
Mon Mar 29 12:57:24 2004
Mon Mar 29 12:57:24 2004
29-Mar-04
```

4.3. Operacinės sistemos komandų vykdymas C programoje

Funkcija, kviečianti operacinės sistemos komandą:

```
int system (const char *komanda);
```

Jei kreipinys yra sėkmingas, tuomet funkcija grąžina 0. Jei įvyko klaida, tuomet yra grąžinama reikšmė -1, o klaidos kodai yra priskiriami kintamiesiems ENOENT, ENOMEM, E2BIG, ENOEXEC.

Funkcijos veikimo pavyzdys:

```
/*************************************************************
 *
 * Tikslas: Isbandyti kreipinio i sistema funkcija.
 *
*************************************************************/
#include <stdlib.h>
main()
   {
     system("ls -l"); {return 0; DOS versijoje}
   }
```

Šios programėlės išvestis mano kompiuteryje:

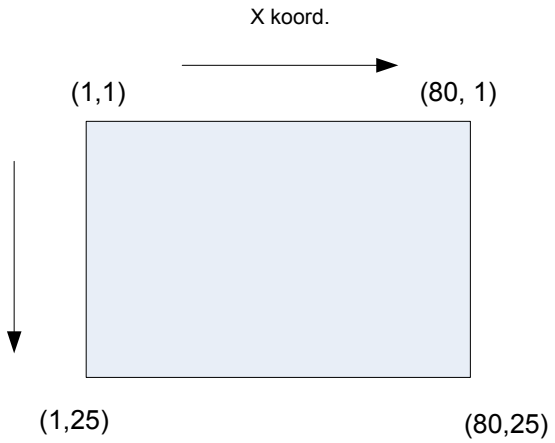
```
total 12
-rw-rw-r--  1 moroz  moroz   966 Kov 26 08:47 DIR_MANO.C
-rw-rw-r--  1 moroz  moroz   448 Kov 26 08:53 SYSTEM.C
-rwxrwxr-x  1 moroz  moroz  4304 Kov 26 08:54 a.out
-rw-rw-r--  1 moroz  moroz    45 Kov 26 08:48 dir_mano.rez
-rw-rw-r--  1 moroz  moroz     0 Kov 26 08:54 system.rez
```

4.4. Tekstinės grafikos modulis Crt

Crt modulyje realizuotos procedūros ir funkcijos, valdančios klaviatūrą, išvedimo išdėstymą, spalvas ir garsus. Šio modulio privalumas – didesnė duomenų pateikimo sparta. Programos, naudojančios operacinėw sistemos priemonės veikia lėčiau.

4.4.1. Ekranas ir spalvos

Ekraną sudaro 25 eilutės ir 80 stulpelių (4-1 pav.).



4-1 pav. Ekrano koordinatės tekstinėke grafikoje

Spalvų kodavimo fragmentas pateikiamas 4-1 lentelėje.

4-1 lentelė. Spalvos tekstinėje grafikoje

Kodas	Konstanta	Spalva	Kodas	Konstanta	Spalva
0	Black	Juoda	...		
1	Blue	Mėlyna	13	Lightmagenta	Šviesi violetinė
2	Green	Žalia	14	Yellow	Geltona
3	Cyan	Žydra	15	White	Balta
4	Red	Raudona			

4.4.2. Naudingos funkcijos ir procedūros

Fono spalva nustatoma iš 8 spalvų (nuo 0 iki 7) naudojant:

```
procedure TextBackground (spalva:byte);
```

Teksto spalva nustatoma naudojant:

```
procedure TextColor ( Spalva:Byte );
```

Teksto spalvos gali būti pasirenkamos iš visų nuo 0 iki 15 spalvų. +emiau pateikiamas pavyzdys, iliustruojantis teksto spalvą ir fonų naudojimą.

```
program TestTextColor;
   uses Crt;
   var TColor,BColor:byte;
begin
   for BColor:= Black to LightGray do
   begin
            TextBackGround(BColor);
            ClrScr;
            For TColor := Black to White do
               begin
                 TextColor(TColor);
                 Writeln('Bandome spalva');
               end;
            TextColor(Blue+Blink);
            Writeln('Bandome mirksejima');
            readln;
```

```
    end;
    NormVideo;
end.
```

Galima apibrėžti vidinį langą, kurš galijma išskirti fonu:

```
procedure Window (x1, y1, x2, y2);
```

čia x1, y1 – viršutinio kairiojo kampo koordinatės, x2, y2 – apatinio dešiniojo kampo koordinatės.

Programėlė su tekstiniais langais:

```
program tekstas_fonas;
  uses Crt;
  procedure Langas (x1, y1, x2, y2,fonas:integer);
    begin
       Window(x1,y1,x2,y2);
       TextBackground(fonas);
       ClrScr;
    end;
  var Tekstas, Fonas, ekranas: integer;
      kAS:string;
  begin
    Ekranas:= TextAttr;
clrscr;
    write('Nurodykite teksto spalva (1..15): ');
    readln(Tekstas);
    write('Nurodykite lango fono spalvos koda (1..7): ');
    readln(Fonas);
    Langas(5,2,75,8,Fonas);
    If Tekstas = Fonas
       then
         begin
         writeln('Spalvos vienodos ');
         writeln('Teksto spalva bus:', Fonas+1 );
         Tekstas:= Fonas+1;
       end;
   Writeln('Rainas, su uodega ir usais, ');
   Writeln(' vaiksto, kada nori ir kur nori.');
   Readln(kas);
   Langas(10,10,70,20,Fonas);
   TextColor(Tekstas);
   GotoXY(10,5);
   if (kas='katinas') or (kas='Katinas') then
      writeln('Puiku! Atspejote')
    else writeln(' Neatspejote');
   readln;
   TextAttr:= Ekranas;
   Clrscr;
end.
```

5.1. Apibrėžtis

Procedūra arba funkcija, kuri kreipiasi į save tiesiogiai (iš savo pačios bloko) arba netiesiogiai (per kitas procedūros ir funkcijos), vadinama rekursine.

5.2. Savybės

Rekursiniai algoritmai turi bendrų savybių. Rekursinė funkcija pateikia paprasčiausio arba *pagrindinio atvejo* sprendimą. Jei funkcija yra kviečiama pagrindiniu atveju, ji tiesiog grąžina rezultatą. Jei funkcija yra kviečiama sudėtingesniam atvejui, funkcija užduoties sprendimą dalina į dvi dalis: į kurią turi sprendinį bei į sudėtingesnę užduotį, kai yra kviečiama nauja funkcijos kopija su paprastesniu variantu.

Šis procesas yra vadinamas *rekursijos žingsniu*. Kiekvienam rekursijos žingsniui, kol algoritmas dalina užduotį į paprastesnes, prieš buvę iškvietimai yra atverti ir laukia, kol rekursija pasieks pagrindinį atvejį. Kai yra pasiekiamas pagrindinis atvejis, tuomet grąžinamas rezultatas iškvietusiai rekursijos kopijai. Toliau yra grąžinimų seka, kol į pagrindinę funkciją yra grąžinamas rezultatas. Šią metodiką iliustruosime pavyzdžiais.

5.3. Rekursinių procedūrų formos

Rekursinių procedūrų struktūra gali būti 3 rūšių:

1. Veiksmai atliekami prieš rekursinį iškvietimą:

```
procedure Rec;
   begin
        sakiniai;
        if sąlyga then Rec;
   end;
```

2. Veiksmai, atliekami po rekursinio iškvietimo. Tuomet veiksmai yra atliekami, grįžtant iš rekursinio grįžimo:

```
procedure Rec;
   begin
        if sąlyga then Rec;
        sakiniai;
   end;
```

3. Veiksmai atliekami tiek prieš, tiek po rekursinio iškvietimo

```
procedure Rec;
   begin
        sakiniai1
        if sąlyga then Rec;
        sakiniai2;
   end;
```

```
procedure Rec;
   begin
        if sąlyga then
            begin
               sakiniai1;
               Rec;
               Sakiniai2;
            end
   end;
```

Rekursija ir iteracija

Tiek rekursija, tiek iteracija reiškia veiksmų kartojimą:

- iteracija kartojimo scenarijų naudoja išreikštiniu būdu,
- rekursija – neišreikštiniu būdu, pakartotinai iškviesdama funkcijo kopijas.

Tiek rekursija, tiek iteracija atlieka pabaigos tikrinimą:

- iteracija baigiasi, kai ciklo kartojimo sąlyga yra patenkinama.
- rekursija yra baigiama, yra pasiekiamas pagrindinis atvejis.

Abiem atvejais rezultatas yra pasiekiamas pažingsniui:

- iteracija keičia ciklo skaitliuką,
- rekursija gamina vis paprastesnį atvejį.

Abiem atvejais kartojimai gali būti amžini:

- iteracija gali patekti į amžiną ciklą, nes ciklo skaitliukas nepasiekia ribinių reikšmių
- rekursija gali tęstis amžinai, jei žingsnis nesuprastina užduoties iki pagrindinio atvejo.

Rekursijos trūkumai:

- jis daug kartų inicijuoja funkcijos iškvietimo mechanizmą ir eikvojami tam skirti resursai, būtent procesoriaus laikas ir atmintis;
- kiekvienam iškvietimui kintamiesiems yra skiriama atmintis ir tai gali reikalauti daug atminties išteklių;

Iteracija paprastai yra atliekama vienos funkcijos ribose, todėl šių resursų ji nereikalauja. Kaip rinktis kartojimo aprašymo būdą?

- Rekursiniai algoritmai yra geresni tuomet, kai jie natūraliau aprašo užduoties sprendimo eigą. Kai kada iteracijos sprendinys gali būti neakivaizdus, tuomet geriau yra rinktis rekursinį algoritmą.
- Jei yra reikalaujama, kad užduoties sprendimas būtų efektyvus, tuomet yra renkamasi iteraciją.

Pavyzdžiai

5.3.1. Faktorialo skaičiavimas

Rekursiją žmonija pažįsta nuo seno.

Panagrinėsime funkciją $n! = 1 * 2 * ... * n$

Tokią formulę galima aprašyti paprasčiausiu ciklu, pavyzdžiui,

```
program Faktorial;
var fact : Longint;
    n, i : integer;
begin
   write('Iveskite sveika skaiciu:');
   readln (n);
```

```
   fact := 1;
   for i := 2 to n do
       fact := fact * i;
   writeln(n,'! = ', fact)
end.
```

Faktorialo sąvoką galima apibrėžti ir kitaip: $0! = 1 \quad \forall n > 0 \quad n! = n * (n-1)!$

Remiantis šia apibrėžtimi, perrašysime rekursiškai šią formulę. Pradžioje pateikiamas kodas Paskalio kalba:

```
program rekursinis_faktorialas;
  var n : integer;

  function Fact( i:integer ) : longint;
  begin
    if i = 1 then Fact := 1
       else Fact := i * Fact(i-1);
  end;
begin
  write('Iveskite sveika skaiciu: ');
  readln(n);
  writeln(n,'! =', Fact(n))
end.
```

Rekursinis faktorialo skaičiavimo algoritmas, aprašytas C kalba

```
#include <stdio.h>
long Fact(long);
int main()
{
  int n;
  printf("\n Iveskite sveika skaciu:");
  scanf("%d", &n);
  printf ("%d! = %d\n", n, Fact(n));
   return 0;
}
long Fact(long skaicius) {
   if ( skaicius <=1 )   return 1;
   else  return (skaicius*Fact(skaicius-1));
}
```

5.3.2. Vienkrypčio sąrašo spausdinimas atvirkščia tvarka

Šis pavyzdys buvo pateiktas duomenų struktūrų 2.1.12 skirsnyje.

5.3.3. Medžio apėjimo rekursiniai algoritmai

Pavyzdyje yra pateikiamas medžio kūrimas ir apėjimas KVD tvarka.

5.3.3.1. Paskalio pavyzdys

```
Medžio apėjimo rekursinių algoritmų pavyzdžiai:
Program sukurti_medi;
 type tree = ^treeNode;
      treeNode = record
         leftPtr  : tree;
         data     : integer;
         rightPtr : tree;
```

```
      end;

 procedure InsertNode( var treePtr : tree; value : integer);
 begin
    if treePtr = nil then
       begin
          new ( treePtr );
          if treePtr <> nil then
             begin
                treePtr^.data     := value;
                treePtr^.leftPtr  := nil;
                treePtr^.rightPtr := nil;
             end
          else
            writeln(value, 'Neitrauktas, nes truksta atminties.');
       end
    else
       if value < treePtr^.data then
          insertNode (treePtr^.leftPtr , value)
       else
          if value > treePtr^.data then
             insertNode ( treePtr^.rightPtr, value )
          else write('Kartojasi. ');
 end;

Procedure inOrder( var treePtr : tree ); /* apeisime KVD tvarka */
begin
   if treePtr <> nil then
     begin
        inOrder( treePtr^.leftPtr );
        write (  treePtr^.data, ' ');
        inOrder( treePtr^.rightPtr );
     end
end;

var i, item : integer;
    rootPtr : tree;

begin
   rootPtr := nil;
   writeln('============================');
   writeln( 'Skaiciai, saugomi medyje:' );
   randomize;
   for i:=1 to 10 do
     begin
        item := random(15);
        write ( item, ' ' );
        insertNode ( rootPtr, item )
     end;
   writeln;
   writeln( 'Apeiname medi:' );
   inOrder( rootPtr );
   writeln;
end.
```

5.3.3.2. C kalba:

```
#include<stdio.h>
#include<string.h>
#include<time.h>
```

```
#include<stdlib.h>

struct treeNode {
      struct treeNode *leftPtr;
      int data;
      struct treeNode *rightPtr;
      };
typedef struct treeNode TreeNode;
typedef TreeNode *TreeNodePtr;

void insertNode ( TreeNodePtr *, int );
void inOrder ( TreeNodePtr );

int main()
{
   int i, item;
   TreeNodePtr rootPtr = NULL;

   srand( time(NULL) );
   puts("----------------------");
   printf("\nSkaiciai saugomi medyje: ");
   for ( i = 1; i <= 10; i++ ) {
      item = rand() %15;
      printf ( "%3d ", item );
      insertNode ( &rootPtr, item );
   }
   printf ( "\n\n Apeiname medi:" );
   inOrder( rootPtr );
   return 0;
}

void insertNode ( TreeNodePtr *treePtr, int value )
   {
    if ( *treePtr == NULL ) {
      *treePtr = ( TreeNodePtr ) malloc ( sizeof ( TreeNode ));
      if ( *treePtr != NULL) {
         ( *treePtr )->data     = value;
         ( *treePtr )->leftPtr  = NULL;
         ( *treePtr )->rightPtr = NULL;
      }
      else
         printf("%d neitrauktas i medi, nes truksta atminties.\n", value );
   }
   else
      if ( value < ( *treePtr )->data )
          insertNode (&((*treePtr)->leftPtr), value);
      else if ( value > ( *treePtr )->data )
            insertNode (&((*treePtr)->rightPtr), value);
         else printf("Duplicated");
}

void inOrder(TreeNodePtr treePtr)
{
   if (treePtr != NULL) {
      inOrder (treePtr->leftPtr);
      printf("%3d",treePtr->data );
      inOrder (treePtr->rightPtr);
   }
}
```

5.3.4. Naujos reiškmės įtraukimas į dvejetainį paieškos medį

5.3.4.1. Paskalio kalba

```
Program sukurti_medi;
 type medis=^virsune;
      virsune = record
         kaire  : medis;
         duom   : integer;
         desine : medis;
      end;

 procedure Iterpti_virsune(var virs : medis; reiksme : integer);
 begin
    if virs = nil then
       begin
          new( virs);
          if virs <> nil then
             begin
                virs^.duom   := reiksme;
                virs^.kaire  := nil;
                virs^.desine := nil;
             end
          else
            writeln(reiksme, ' neitraukta, nes truksta atminties.');
       end
    else
       if reiksme < virs^.duom then
          Iterpti_virsune (virs^.kaire, reiksme)
       else
          if reiksme > virs^.duom then
             Iterpti_virsune (virs^.desine, reiksme)
          else write('dubliuojasi ');
 end;

Procedure Infix_KVD(var virs : medis);
begin
   if virs <> nil then
     begin
        Infix_KVD ( virs^.kaire);
        write ( virs^.duom , ' ' );
        Infix_KVD ( virs^.desine);
     end
end;

var i, skaic:integer;
    saknis:medis;

begin
   saknis := nil;
   writeln('============================');
   writeln('Skaiciai, saugomi medyje:');
   randomize;
   for i:=1 to 10 do
     begin
        skaic := random(15);
        write(skaic,' ');
        Iterpti_virsune( saknis, skaic)
     end;
   writeln;
   writeln('Apeiname medi KVD tvarka:');
```

```
   Infix_KVD(saknis);
   writeln;
end.
```

Programos vykdymo rezultatas:

```
==============================
Skaiciai, saugomi medyje:
2 13 4 12 8 3 13 dubliuojasi 12 dubliuojasi 2 dubliuojasi 12
dubliuojasi
Apeiname medi KVD tvarka:
2 3 4 8 12 13
```

5.3.4.2. C kalba

```
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<stdlib.h>

struct virsune {
      struct virsune *kaire;
      int duom;
      struct virsune *desine;
      };

typedef struct virsune MedzioVirsune;
typedef struct virsune * MedzioVirsRod;

void Iterpti_virsune(MedzioVirsRod *, int);
void Infix_KVD (MedzioVirsRod);

int main()
{
   int i, skaic;
   MedzioVirsRod saknis = NULL;

   srand(time(NULL));
   puts("-----------------------------------------------");
   printf("\n Skaiciai saugomi medyje: ");
   for ( i = 1; i <= 10; i++ ) {
      skaic = rand() %15;
      printf("%3d ", skaic);
      Iterpti_virsune(&saknis, skaic);
   }
   printf("\n\n Apeiname medi KVD tvarka:");
   Infix_KVD(saknis);
   printf("\n");
   return 0;
}

void Iterpti_virsune(MedzioVirsRod *virs, int reiksme)
   {
    if ((*virs) == NULL) {
      (*virs) = (MedzioVirsRod) malloc(sizeof(struct virsune));
      if ((*virs) != NULL) {
```

```
         (*virs)->duom  = reiksme;
         (*virs)->kaire = NULL;
         (*virs)->desine= NULL;
      }
      else
         printf("%d neitrauktas i medi, nes truksta atminties.\n",
reiksme);
   }
   else
      if ( reiksme < ( *virs )->duom )
       Iterpti_virsune (&((*virs)->kaire), reiksme);
      else if ( reiksme > ( *virs )->duom )
            Iterpti_virsune (&((*virs)->desine), reiksme);
         else printf("Dubliuojasi");
}

void Infix_KVD(MedzioVirsRod virs)
{
   if (virs != NULL) {
      Infix_KVD (virs->kaire);
      printf("%3d",virs->duom );
      Infix_KVD (virs->desine);
   }
}
```

6. Rūšiavimo suliejimu (angl. mergesort) algoritmas

Populiariausi rikiavimo algoritmai yra išsamiai išdėstyta doc. dr. Sauliaus Ragaišio metodinėje medžiagoje: http://www.mif.vu.lt/~ragaisis/InfIvadas/PaiesRus.htm,

Šiame skyriuje pateikiamas algoritmo aprašas, kurio nėra minėtoje metodinėje priemonėje.

Rūšiavimo suliejimu algoritmas sudaro surūšiuotą seką, rūšiuodama dvi puses ir jas suliedama. Panašiai, kaip greito rūšiavimo algoritmas, rūšiavimo suliejimu algoritmas yra paremtas principu „skaidyk ir valdyk“.

6.1. Algoritmo aprašas

Žingsniai

1. *Jei n<2, tai rinkinys jau surūšiuotas ir darbas baigiamas.*
   Priešingu atveju
   a. *Rūšiuoti kairiąją masyvo pusę.*
   b. *Rūšiuoti dešiniąją masyvo pusę.*
   c. *Sulieti kairę ir dešinę puses.*

1 c) žingsnį aprašo suliejimo algoritmas:

Pradinės sąlygos

First, Last, Middle (kairiosios dalies paskutinio elemento indeksas), Key [First..Last], kuriame karioji ir dešinioji dalys jau surūšiuotos, tai yra yra tenkinamos sąlygos:
Key[First] ≤ Key[First+1] ≤… ≤ Key[Middle] ir Key[Middle] ≤ Key[Middle+1] ≤… ≤ Key[Last].

Galinės sąlygos

Visas masyvas surūšiuotas: Key[First] ≤ Key[First+1] ≤… ≤ Key[Last]

Žingsniai

1. Nukopijuoti Key[First. . Last] į pagalbinį masyvą Pagalb[First..Last]

2. Indeksams i ir k priskirti reikšmę First, indeksui j priskirti reikšmę Middle+1.

3. Kol i ≤ Middle ir j ≤ Last

   Jei Pagalb[i] yra mažesnis už Pagalb[j],

   tai kopijuoti Pagalb[i] į Key[k], i ir k padidinti vienetu;

   priešingu atveju, kopijuoti Pagalb[j] į Key[k], j ir k padidinti vienetu;

4. Jei trečiame žingsnyje liko kairiosios pusės nenukopijuota dalis, tuomet nukopijuoti ją į masyvo pabaigą Key[k..Last] (jei trečiame žingsnyje liko dešiniosios dalies elementai, jie jau atėjusiame masyve stovėjo savo vietose ir jų nereikia judinti).

6.2. Algoritmo sudėtingumo vertinimas

Vertinant rūšiavimo laiką, įprastai vertiname rakto lyginimo operacijų skaičių. Pažymėkime jį T(N). Pagrindiniu rekursijos atveju,

T(N) = 0, kai n<2.

Kitais atvejais, T(N) = T(N/2) + T(N/2) + N, n>1.

Pirmas dėmuo reiškia kairės pusės raktų lyginimo operacijų skaičius, antras – dešinės pusės rūšiavimo rakto lyginimo operacijų skaičius, trečias – operacijos, reikalingos dviejų masyvų suliejimui.

Paskaičiuokime 16 elementų masyvo rakto lyginimo operacijų skaičiui T(16):

T(16) = 2T(8)+ 16
T(8) = 2T(4)+ 8
T(4) = 2T(2)+ 4
T(2) = 2T(1)+ 2
T(1) = 0

Toliau grįžtame ir įstatome reikšmes

T(1) = 0
T(2) = 2T(1) + 2 = 0 + 2 = 2
T(4) = 2T(2) + 4 = 2*2 + 4 = 8
T(8) = 2T(4) + 8 = 2*8 + 8 =24
T(16) = 2T(8) + 16 = 2*24 + 16 = 64

Bendru atveju,

$$T(n) = 2 * T(\frac{n}{2}) + n = 2 * (T(\frac{n}{4}) + \frac{n}{2}) + n = 4 * T(\frac{n}{4}) + 2 * n$$

Pažymėkime k = log n – tiek lygčių turime iki pagrindinio atvejo T(1):

$$T(n) = n * T(1) + n * \log n = n * \log n + n = n * (\log n + 1)$$

Taigi, vertinant apytikriai gauname laiko sudėtingumą $O(n * \log n)$ – vienas optimaliausių iš visų nagrinėtų algoritmų.

Šį algoritmą galima realizuoti tiek rekursiškai, tiek iteraciškai.

7. Simbolių eilutės

Simbolių eilučių apdorojimas Paskalyje yra išdėstytas doc. dr. Sauliaus Ragaišio metodinėje medžiagoje: http://www.mif.vu.lt/~ragaisis/InfIvadas/SimbEilutes.htm, kuria rekomenduoju naudotis.

C kalba simbolių eilučių apdorojimas yra atliekamas, remiantis rodyklėmis. Todėl šis skyrius yra pateikiamas II dalyje, išnagrinėjus rodyklės tipą.

7.1. Simboliai ir eilutės

Simbolis – tai `int` tipo reikšmė, apskliausta apostrofais, pavyzdžiui, `'a'` yra simbolio a kodas, o `'\n'` yra naujos eilutės simbolio kodas.

Simbolių eilutė – simbolių seka, kuri yra apdorojama kaip vientisas objektas. Simbolių eilutėje gali būti raidės, skaitmenys, specialūs simboliai, kaip antai +, -, *, /, $ ir kiti.

7.2. Simbolių eilučių aprašymas

Simbolinės eilutės yra vienmačio masyvo poaibis. Simbolių eilutės skiriasi nuo įprasto simbolių masyvo tuo, kad ji baigiasi nulio simboliu ‘\0’. Jei simbolių eilutė yra kuriama, naudojantis tam skirtomis funkcijomis, šis simbolis yra įrašomas automatiškai. Jei naudotojas pats formuoja eilutę, tuomet jam pačiam teks pasirūpinti įrašyti pabaigos simbolį.

Simbolių eilutę galima aprašyti keliais būdais:

- simbolinė konstanta,
- simbolių (char) masyvu,
- nuoroda į simbolį,
- simbolių eilučių masyvu.

7.2.1. Simbolių konstanta

Simbolinė konstanta yra rašoma dvigubose kabutėse. Eilutės simboliai ir pabaigos simbolis yra rašomas nuosekliai atminties ląstelėse, pavyzdžiui,

Vardas

P	E	T	R	A	S	\0

Kompiliatorius suskaičiuoja, kiek atminties reikia skirti tokiai konstantai ir atitinkamai išskiria jai vietos. Jei simbolių eilėje reikia panaudoti dvigubos kabutės simbolį, tuomet prieš jį yra rašomas pasviręs brūkšnys \“.

Vardas – tai nuoroda, kuri rodo į simbolinės konstantos pirmąjį simbolį.

7.2.2. Simbolių eilutė

Eilutė gali būti priskiriama tiek simbolių masyvui, tiek nuorodai į simbolį,

```
char spalva[]="balta";
```

Šis aprašas sukuria masyvą spalva, kuriame yra penki elementai: ‘b’, ‘a’, ‘l’, ‘t’, ‘a’, ‘\0’.

Kitas aprašas:

```
const char * rod_spalva="balta";
```

sukuria kintamąjį **rod_spalva**, kuris rodo į eilutę **"balta"**.

Pernešamumo pastaba. Jei eilutė yra inicializuojama char * apraše, tai taip inicializuota eilutė kai kuriose sistemose gali būti talpinama pastovioje atminties dalyje. Jei eilutę bus keičiama programoje, tai siekiant pernešamumo, inicializuoti eilutę reiktų simbolių masyve:

```
char spalva[]={'b', 'a', 'l', 't', 'a', '\0'};
```

Šiuo atveju masyvo ilgis yra nustatomas automatiškai. Kompiliatorius paskaičiuoja simbolių kiekį ir išskiria atitinkamą vietą. Jei yra nurodomas masyvo elementų skaičius apraše, tuomet reikia nepamiršti ir vietą **NULL** simboliui.

Dažnos klaidos:

- Pamirštama išskirti vietos simboliui **NULL**.
- Bandoma spausdinti eilutę, kurios pabaigoje nesuformuotas **NULL** simbolis.

> *Patarimas. Jei yra aprašomas simbolių masyvas, reikia išskirti jam tiek vietos, kiek jos prireiktų maksimalaus ilgio eilutei ir **NULL** simboliui. C kalba įsimena bet kurio ilgio eilutes. Jei eilutė yra ilgesnė nei aprašytas masyvo ilgis, tai netelpantys simboliai perrašys duomenys, esantys už masyvo ribų ir programos veiksmai taps neprognozuojami.*

4-1 pavyzdys. *Eilučių inicializavimo būdai.*

```
#include <stdio.h>
int main()
{
  char eil1[30]="Juodas katinas"; /* konstanta */
  char eil2[]="Baltas katinas";   /* konstanta */
  char *eil3="Pilkas katinas";    /* kintamasis */

  printf("eil1=%s\n", eil1);
  printf("eil2=%s\n", eil2);
  printf("eil3=%s\n", eil3);
  return 0;
}
```

Šios programos rezultatai yra tokie:

```
eil1=Juodas katinas
eil2=Baltas katinas
eil3=Pilkas katinas
```

Pirmieji du aprašymo būdai analogiški masyvams. Trečias aprašas galimas tik simbolių masyvams. Šiuo atveju **eil3** yra kintamasis, nuoroda į eilutė. Visais atvejais eilutės vardas yra nuoroda į pirmąjį eilutės simbolį. Kadangi **eil1** ir **eil2** yra konstantos, tai priskyrimas **eil1 = eil2** yra negalimas, nes konstantų reikšmių negalima keisti. Tačiau kitas priskyrimas **eil3 = eil2** yra teisingas, nes **eil3** yra kintamasis.

Analogiškai dvimačiam masyvui yra naudojamas eilučių masyvas.

```
char *spalvos[5]=”raudona spalva”, “melyna spalva”, “zalia spalva”,
“oranzine spalva”, “juoda spalva”;
```

Šiuo atveju, šį simbolių masyvą sudaro penkios nuorodos į simbolių eilutes. Pirmoji nuoroda yra **spalvos[0]**, ir ji rodo į pirmą eilutę, antroji – **spalvos[1]** ir rodo į antrą eilutę ir panašiai. Jei simbolių eilutės yra aprašomos taip: **char [5][20]**, tuomet visos eilutės bus vienodo ilgio. Atmintis tokiu atveju yra naudojama neefektyviai, nes bus nurodomas dydis pagal ilgiausią eilutę.

7.2.3. Eilutės įvedimas (standartinės funkcijos iš stdio.h)

Eilutės įvedimas apima atminties išskyrimą ir reikšmes priskyrimą. Vieta atmintyje yra išskiriama statiškai

```
char vardas[20]:
```

arba dinamiškai

```
char * vardas;
```

Pastaruoju atveju, vieta yra išskiriama dinamiškai, naudojant funkcijas **malloc** arba **calloc**,

```
  vardas = malloc(20);
```

Eilutės yra įvedamos, naudojant funkcijas **scanf()** ir **gets()**.

Funkcija **scanf()** įveda simbolių eilutes, nurodant joms formatą **%s**. Kadangi eilutės vardas yra jos pradžios adresas, todėl nenaudojame & prieš įvedamą vardą:

```
  scanf( "%s", word);
```

Funkcija **gets()** skaito simbolius tol, kol neaptinka naujos eilutės požymio **'\n'**. Šis simbolis atsiranda, paspaudus ENTER klavišą. Ši funkcija prie įvestų simbolių prideda nulio simbolį.

4-2 pavyzdys. *Funkcijos gets() naudojimo paprastas pavyzdėlis.*

```
#include<stdio.h>
int main()
{
   char name[81];
   printf("Kuo Jus vardu? ");
   gets(name);
   printf("Sveiki, %s \n", name);
   return 0;
}
```

Programos rezultatas:

```
Kuo Jus vardu? Ona
Sveiki, Ona
```

Funkcija **gets()** turi daugiau galimybių. Jei funkcija **gets()** aptinka failo pabaigos požymį **EOF**, tai ji per vardą grąžina **NULL** reikšmę. **NULL** reikšmė apibrėžta faile **<stdio.h>** ir lygi 0. Todėl funkcija dažnai naudojama tokiose konstrukcijose:

```
       while(gets(name)!= NULL)
```

Todėl funkcija dažnai naudojama tokiose konstrukcijose:

Funkcija **sscanf(char *s, const char *format, …)** ekvivalenti scanf, išskyrus tai, kad įveda iš masyvo s, o ne iš klaviatūros, pavyzdys:

7.2.4. Eilučių išvedimas

Eilutėms išvesti yra skirtos dvi standartinės funkcijos **puts()** ir **printf()**. **Printf()** jau buvo naudojama, todėl aptarsime **puts()**. Ji turi tik vieną argumentą, būtent nuorodą į išvedamą eilutę. Pavyzdys iliustruoja **puts()** naudojimo galimybes.

4-3 pavyzdys. *Puts() naudojimo ilistracija.*

```
#include<stdio.h>              /* isvestis.c */
#define DEF "Eilute #define."
int main()
{
   static char eil1[]="Inicializuota eilute.";
        char *eil2="Inicializuota rodykle.";
   puts("Funkcijos puts() argumentas ");
   puts(DEF);
   puts(eil1);
   puts(eil2);
   puts(&eil1[3]); /* nuo ketvirto elemento */
   puts(eil2+6);
}
```

Šios programos veiklos rezultatas yra toks:

```
Funkcijos puts() argumentas
Eilute #define.
Inicializuota eilute.
Inicializuota rodykle.
cializuota eilute.
lizuota rodykle.
```

Šis pavyzdys iliustruoja, kad tiek tekstas tarp kabučių, tiek simbolių eilučių vardai, yra nuorodos. Antroje eilutėje nuo pabaigos yra spausdinama, pradedant ketvirtuoju simboliu ir iki eilutės pabaigos nulio simbolio. Pastarasis simbolis yra keičiamas naujos eilutės simboliu. Todėl, kiekvienas **puts()** išvedamas tekstas yra spausdinamas iš naujos eilutės.

Taigi, funkcija **puts()** spausdina iki nulio simbolio **'\0'**. Todėl simbolių masyvams ši funkcija netinka, kadangi jame nėra eilutės pabaigos požymio.

Išvedant eilutes, naudojant funkciją **printf()**, nurodomas **%s** formatas. Jei reikia perkelti žymeklį į naują eilutę, reikia nurodyti naujos eilutės simbolį **'\n'**:

```
printf("%s\n", SimbEilutė);   =     puts(SimbEilutė)
```

Pailiustruosime **sscanf** ir **sprintf** funkcijas.

4-4 pavyzdys. *Funkcijų sscanf ir sprintf naudojimo pavyzdėlis*

```
#include<stdio.h>
int main()
{
   char str[]="31351 74.545",
      eil[80];
   int x;
   double y;
   puts("Demonstruojame eiluciu ivedimo ir isvedimo operacijas.");
   sscanf(str, "%d%lf", &x, &y);
   printf("%s\n%s%6d\n%s%8.3f\n",
        "Reiksmes, kurios buvo issaugotos masyve:",
        "sveika:",x, "ilga reali:",y);
```

```
   sprintf(eil, "integer:%6d\ndouble:%8.2f", x, y);
   printf("%s\n%s\n", "Formatuota isvestis, issaugota masyve yra:", eil);
   return 0;
}
```

Programos rezultatas:

```
Demonstruojame eiluciu ivedimo ir isvedimo operacijas.
Reiksmes, kurios buvo issaugotos masyve:
sveika: 31351
ilga reali:  74.545
Formatuota isvestis, issaugota masyve yra:
integer: 31351
double:   74.55
```

Po vieną simbolį įveda funkciją **`getchar()`**, o išveda **`putchar()`**.

7.2.5. Kitos String.h funkcijos

Internete galima pasiskaityti

http://cermics.enpc.fr/~ts/C/FUNCTIONS/function.ref.html

7.2.6. Eilučių apdorojimo operacijos

char * strcpy (char *rez, char *prad)	Kopijuoja eilutė **prad** į eilutę **rez**.
char * strncpy (char *rez, char *prad, unsigned kiek)	Kopijuoja eilutės **prad** kiek simbolių į eilutę **rez**.
char * strcat (char *rez, char *prad)	Sujungia dvi eilutes
char * strncat (char *rez, char *prad, unsigned maxlen)	Prijungia eilutės **prad** **maxlen** simbolių prie eilutės **rez**
int strlen (char *eil)	Paskaičiuoja eilutės simbolių skaičių be nulio simbolio.

4-5 pavyzdys. *Keleto lentelėje pateiktų funkcijų naudojimas.*

```
#include<stdio.h>
#include<string.h>

int main()
{
   char x1[]="Laimingu ",
      x2[]="Naujuju metu",
      *x3="Viskas";
   char     x[25], y[25], z[17];
/*     char x[25], *y, z[17]; luzimas */
     unsigned ilgis;

   strcpy(x,x1);
   x3=strcpy(x1, x2);
   printf("x1=%s\nx2=%s\nx3=%s\n",x1,x2, x3);
   strcpy(y,x);
   printf("\n%s%s\n%s%s\n",
        "x simboliu eilute:", strcat(x,x2 ),
        "y simboliu eilute:", y);
   strncpy(z,x,16);
   z[16]='\0';
   printf("z simboliu eilute: %s\n", z);
   ilgis=strlen(x);
   printf("x eilutes ilgis: %d\n",ilgis );
   printf("y eilutes ilgis: %d\n",strlen(y) );
   printf("z eilutes ilgis: %d\n",strlen(z) );
   return 0; }
```

Šios programos rezultatas yra tokie:

```
x1=Naujuju metu
x2=Naujuju metu
x3=Naujuju metu

x simboliu eilute:Laimingu Naujuju metu
y simboliu eilute:Laimingu
z simboliu eilute: Laimingu Naujuju
x eilutes ilgis: 21
y eilutes ilgis: 9
z eilutes ilgis: 16
```

7.2.7. Eilučių palyginimas

int strcmp (char *s1, char *s2)	Rezultatas 0, jei s1==s2; <0, jei s1<s2; >0, jei s1>s2.
int strncmp (char *s1, char *s2, int maxlen)	Rezultatas 0, jei s1==s2; <0, jei s1<s2; >0, jei s1>s2. Lyginami tik pirmi maxlen simboliai

4-6 pavyzdys. Keleto lentelėje pateiktų funkcijų naudojimas.

```
#include<stdio.h>       /* palyg.c */
#include<string.h>

int main()
{
   char x1[]="Laimingu ",
      x2[]="Naujuju metu";
/*   char   x[25], y[25], z[17];*/
     char   x[25], *y, z[17];
     unsigned ilgis;

   strcpy(x,x1);
   strcpy(y,x);
   printf("%s%s\n%s%s\n",
        "x simboliu eilute:", strcat(x,x2 ),
        "y simboliu eilute:", y);
   strncpy(z,x,16);
   z[16]='\0';
   printf("z simboliu eilute: %s\n", z);
   printf("\nLyginame x ir y: %d\n", strcmp(x,y) );
   printf("Lyginame z ir x: %d\n", strcmp(z,x) );
   printf("Lyginame y ir z pirmus 5 simbolius: %d\n", strncmp(y,z,5) );
   return 0;
}
```

Šios programos rezultatai:

```
x simboliu eilute:Laimingu Naujuju metu
y simboliu eilute:Laimingu
z simboliu eilute: Laimingu Naujuju

Lyginame x ir y: 78
Lyginame z ir x: -32
Lyginame y ir z pirmus 5 simbolius: 0
```

7.2.8. Paieška eilutėse

char *strchr (char *prad, char s)	Eilutėje **prad** ieško pirmojo simbolio lygaus s. Jei neranda, grąžina **NULL**.
char *strrchr(char *eil, char c)	Peržiūri eilutę **eil**, kol pasirodo paskutinis simbolis **c**. Jei neranda, grąžina **NULL**.

size_t strcspn(char *eil1, char *eil2)	Grąžina ilgį **eil1** dalies nuo eilutės pradžios, kurioje nėra simbolių iš **eil2**.
size_t strspn(char *eil1, char *eil2)	Grąžina ilgį **eil1** dalies nuo eilutės pradžios, kurioje yra simbolių iš **eil2**.
char *strpbrk(char *eil1, char *eil2)	Peržiūri eilutę **eil1** tol, kol aptinka pirmąjį simbolį, esantį **eil2**. Jei randa, tai funkcija grąžina nuorodą į rastą simbolį eilutėje **eil1**. Jei neranda, grąžina **NULL**.
char *strstr(char *s1, char *s2);	Ieško eilutės **s2** eilutėje **s1**. Jei rado, grąžina nuorodą į rastą eilutę eilutėje **s1**. Jei neranda, grąžina **NULL**.

4-7 pavyzdys. Keleto lentelėje pateiktų funkcijų naudojimas.

```
#include<stdio.h>
#include<string.h>
int main()
{
   char *eilute="Cia yra tekstas",
  *kita="et";
   char simbolis='a';

   if (strchr(eilute, simbolis)!=NULL)
     printf("Simbolis \'%c\' rastas eiluteje \"%s\"\n", simbolis, eilute);
   else
     printf("Simbolis \'%c\'nerastas eiluteje \"%s\"\n", simbolis, eilute);
   printf("Eilutes \"%s\" segmento ilgis, kuriame nera eilutes \"%s\"
simboliu ilgis yra %d\n",
        eilute, kita, strcspn(eilute, kita));
   return 0;
}
```

```
Simbolis 'a' rastas eiluteje "Cia yra tekstas"
Eilutes "Cia yra tekstas" segmento ilgis,
 kuriame nera eilutes "et" simboliu
 ilgis yra 8
```

7.2.9. Kai kurios simbolių apdorojimo funkcijos, iš ctype.h

int isdigit (int c)	True – jei **c** skaitmuo, priešingu atveju – false.
int isalpha (int c)	True – jei **c** raidė, priešingu atveju – false.
int isalnum (int c)	True – jei **c** raidė arba skaitmuo, priešingu atveju – false.
int islower (int c)	True – jei **c** mažoji raidė, priešingu atveju – false.
int isupper (int c)	True – jei **c** didžioji raidė, priešingu atveju – false.
int tolower (int c)	Jei **c** didžioji raidė, tai funkcija grąžina atitinkamai mažąją raidę. Jei mažoji, tai grąžina tą pačią reikšmę.

int toupper (int c)	Jei **c** mažoji raidė, tai funkcija grąžina atitinkamai didžiąją raidę. Jei didžioji, tai grąžina tą pačią reikšmę.
int isspace (int c)	True – jei **c** tarpas, priešingai – false.
int iscntrl (int c)	True – jei **c** valdantis simbolis, priešingai – false.

Pavyzdžiui,

```
a='2';
printf("%s \n", isdigit(a)? "a yra skaitmuo":"a nėra skaitmuo");
```

Fragmento vykdymo rezultatas:

```
a yra skaitmuo
```

7.2.10. Kitos funkcijos

int stricmp(char *s1, char *s2)	Rezultatas 0, jei s1==s2; <0, jei s1<s2; >0, jei s1>s2. Netikrinamas raidžių registras
int strnicmp(char *s1, char *s2, int maxlen)	Rezultatas 0, jei s1==s2; <0, jei s1<s2; >0, jei s1>s2. Lyginami tik pirmi **maxlen** simboliai. Netikrinamas raidžių registras.
char *strlwr(char *eil)	Perveda visus **eil** simbolius į apatinį registrą (mažąsias raides)
char *strupr(char *eil)	Perveda visus **eil** simbolius į viršutinį registrą (didžiąsias raides)
char *strdup(char *eil)	Naudojant funkcija **malloc** eilutės **eil** kopijai yra skiriama vieta.
char *strset(char *eil, char ch)	Užpildo eilutę **eil** simboliais **ch**.
char *strnset(char *eil, char ch, unsigned kiek)	Užpildo kiek eilutės **eil** simbolių simboliais **ch**.

Pavyzdžiai

4-8 pavyzdys. *Eilučių kopijavimas ir spausdinimas*

```
program eiluciu_apdorojimas;
  var x1,x2,y:string;
      x : string[25];
      z : string[17];
      ilgis:byte;
  begin
  x1:='Laimingu ';
  x2:='Naujuju metu!';
  x:=Copy(x1,1,length(x1));
  y:=Copy(x,1,length(x));
  x:=Concat (x,x2);
```

```
  writeln('x simboliu eilute: ',x);
  writeln('y simboliu eilute: ',y);
  z:=Copy(x,1,16);
  writeln('z simboliu eilute: ',z);
  writeln('X simboliu eilutes ilgis: ',length(x));
  writeln('Y simboliu eilutes ilgis: ',length(y));
  writeln('Z simboliu eilutes ilgis: ',length(z));
  end.
```

4-9 pavyzdys. *Eilučių ilgio nustatymas ir spausdinimas*

```
#include<stdio.h>
#include<string.h>

int main()
{
   char x1[]="Laimingu ",
      x2[]="Naujuju metu",
      *x3="Viskas";
   char     x[25], y[25], z[17];
/*     char x[25], *y, z[17]; luzimas */
     unsigned ilgis;

   strcpy(x,x1);
   x3=strcpy(x1, x2);
   printf("x1=%s\nx2=%s\nx3=%s\n",x1,x2, x3);
   strcpy(y,x);
   printf("\n%s%s\n%s%s\n",
        "x simboliu eilute:", strcat(x,x2 ),
        "y simboliu eilute:", y);
   strncpy(z,x,16);
   z[16]='\0';
   printf("z simboliu eilute: %s\n", z);
   printf("x eilutes ilgis: %d\n",strlen(x) );
   printf("y eilutes ilgis: %d\n",strlen(y) );
   printf("z eilutes ilgis: %d\n",strlen(z) );
   return 0;
}
```

Šiame skyriuje pateikiamos dažniau naudojamos operacijos su simbolių eilutėmis. Darbui su eilutėmis C kalboje yra sukurti plėtiniai, kuriuos naudoti patogiau, nei pateikiamas būdas. Tačiau šis būdas yra pernešamas įvairiuose C kalbos versijose.

Literatūra

1. Vladas Tumasonis. *Paskalis ir Turbo Paskalis 7.0*. Ūkas, Vilnius, 1993.

2. Saulius Ragaišis. Informatikos įvadas: metodinė priemonė. 2001. http://www.mif.vu.lt/~ragaisis/InfIvadas/index.html

3. Jonas Blonskis, Vytautas Bukšnaitis, Jonas Smolinskas, Antanas Vidžiūnas. *Programavimo praktikumas: programavimas su Turbo Paskaliu*. „Smaltijos“ leidykla, Kaunas, 2000.

4. H. M. Deitel, P. J. Deitel. *C: how to program.* Third edition. Prentice Hall, 2001.

5. С. А. Абрамов, Г.Г. Гнездилова, Е. Н. Капустина, М. И. Селюн. *Задачи по программированию.* Наука, Москиа, 19887