VILNIAUS PEDAGOGINIS UNIVERSITETAS
FIZIKOS IR TECHNOLOGIJOS FAKULTETAS

Aušra Kynienė

C KALBŲ ABC

Metodinė priemonė

Leidykla

VILNIUS, 2004

Darbas apsvarstytas ir rekomenduotas skelbti Teorinės fizikos ir informacinių technologijų katedros posėdyje 2004 m. spalio mėn. 06 d., protokolo Nr. 2.

Recenzavo: *prof. Joana Lipeikienė*
dr. Artūras Acus

TURINYS

PRATARMĖ

„*C kalbos ABC*" metodinio leidinio paskirtis – supažindinti su C kalbos programavimo pagrindais. Knygelė skirta visiems, kurie nori išmokti šiuolaikinės programuotojų kalbos.

Programuojant, kaip ir užsiimant bet kuria kita veikla, patirtis įgyjama dirbant, todėl šioje metodinėje priemonėje pateikiama daug užduočių dirbti savarankiškai. Ši metodinė priemonė šiek tiek praplėsta: pateikiamos ir sudėtingesnės temos, kurių gali prireikti, kuriant sudėtingesnes programas. Čia visiškai nekalbama apie grafikos kūrimo elementus.

Leidinyje gausu pateiktų programų pavyzdžių, tačiau dalis programų yra nebaigtos ir, norint, kad šios programos veiktų, pačiam skaitytojui reikės jas papildyti. Leidinio pabaigoje pridedamas priedas, kuriame patekiama C kalboje naudojamų funkcijų santrauka.

Šis leidinys studentams turėtų padėti suprasti C kalbą ir pramokti patiems šia kalba rašyti programas. Norintiems daugiau susipažinti su C ir C++ kalba rekomenduojama paskaityti literatūros sąraše minimas knygas [1–5].

C KALBOS PRANAŠUMAI

C programavimo kalba – tai didelių galimybių programavimo kalba, vis labiau naudojama visame pasaulyje. Dabartiniu metu ši kalba – pagrindinė profesionalų programavimo kalba, kuri sėkmingai vartojama tiek sisteminei, tiek taikomajai įrangai kurti. Pirmą kartą C kalbą aprašė B. W. Karnighan ir D. M. Ritchie 1978 m. išleistoje knygoje „C programavimo kalba". C kalbos variantas su klasėmis pavadintas C++ kalba. Ši kalba – kaip instrumentas programuotojams – praktikams. Be pastarosios yra ir kitų programavimo kalbų, pvz., Paskalio, t. y. griežto programavimo kalba, Beisiko, kurios sintaksė artima anglų kalbai. Iš esmės C++ yra nauja programavimo kalba, pritaikyta sudėtingoms programų sistemoms ir instrumentinėms programavimo priemonėms kūrti, panaudojant objektinio programavimo technologiją. Ši kalba yra glaudžiai susijusi su klasikine C kalba.

C – šiuolaikinė, efektinga programavimo kalba.

Jos dėka geriausiai išnaudojamos kompiuterio galimybės. C kalba parašytos programos yra kompaktiškos ir greitai vykdomos.

C – mobilioji programavimo kalba. Tai reiškia, jei programa parašyta šia kalba, ji gali būti lengvai, su nedideliais pataisymais arba visai be jų perkeliama į kitas skaičiavimo sistemas, pvz., iš IBM kompiuterio programos veikimą perkelti į UNIX.

C – galinga ir lanksti programavimo kalba. Didelė dalis galingos UNIX ir Windows operacinės sistemos parašyta C kalba. Ja parašytos programos naudojamos fizikiniams ir techniniams uždaviniams spręsti, taip pat animacijai kurti.

C – turi galimybę panaudoti nemažai valdančiųjų konstrukcijų, kurios paprastai asocijuojasi su asembleriu. Asem-

blerio programavimo kalba labai sudėtinga (žr. 1 pav.). Tai skaičiai ir kodai, kuriuos ne specialistui suprasti sunku, kadangi ji rašoma procesoriaus kalba (žr. 2 pav.).

```
B8 23 01 MOV AX,0123
05 25 00 ADD AX,0025
8B D8 MOV BX,AX
03 D8 ADD BX,AX
8B CB MOV CX,BX
2B C8 SUB CX,AX
2B C0 SUB AX,AX
90 NOP
CB RETF
```

1 pav. Asembleriu prašyta programa

```
B8 23 01 05 25 00 8B D8 03 D8 8B CB 2B C8 2B C0
90 CB D0 A2 17 04 1F C3 1E 52 2E 8E 34 00 AE 1A
45 04 B4 41 CD 21 BA 94 04 B4 41 CD 21 5A E8 93
02 C6 06 44 04 00 1F C3 BA 87 80 E8 DA FF 0E 1F
E9 AB F8 9C E8 9D FC 50 52 BA 78 81 E8 C9 FF 0E
1F E8 08 31 5A 58 9D 3D 41 00 75 03 E9 8F F8 E9
12 BA 8E 1E 3A D0 FE 06 44 04 06 57 1E 56 1E 06
1F BE 8D 87 26 C6 06 A3 E5 00 E8 EA F0 1F 72 1F
```

2 pav. Procesoriaus kalba

Asembleriu parašyta programa yra labai didelės apimties, nes kompiuteriui reikia detaliai aiškinti kiekvieną atliekamą veiksmą. Asembleris — tai sisteminė programuotojų kalba, kuri leidžia panaudoti visas kompiuterio galimybes. Fortranas puikiai suskaičiuoja sinusą, tačiau sunkiai pavaizduoja tašką ekrane. C kalba — tai aukšto lygio programavimo kalba, kuri kaip ir asembleris leidžia panaudoti visas kompiuterio galimy-

bes. Šiuo metu C kalba yra sisteminė programuotojų kalba.

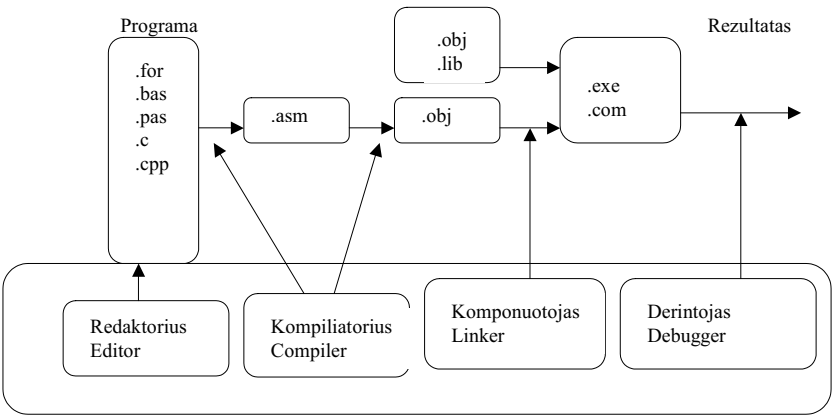
Tai struktūrinė kalba, tačiau ji nėra labai griežta ir per daug nevaržo programuotojo.

PROGRAMAVIMO PROCESAS

Programos rašymas ir parengimas vykdyti yra gana ilgas procesas, kuris schematiškai yra pavaizduotas 3 pav. Iš pradžių pasinaudojus turimu redaktoriumi, rašomas programos tekstas. Čia svarbu neapsirikti renkant programos tekstą ir parinkti bylos vardą, kurioje bus programa. Programos teksto byla – *.cpp arba *.c (pirmoji – C++ kalba, antroji – C).

Toliau parašytą programą kompiliuojame, t. y. mūsų parašytą programą perrašome kompiuterio kalba ir gauname objektinę bylą *.obj. Tai gali būti tik nedidelė programos dalis.

Visus programų gabaliukus į vieną *.exe bylą sujungia kita programa, vadinama komponuotoju (Linker). Ši programos dalis įtraukia ir bibliotekos funkcijas. Tačiau gautoji programa yra klaidinga, todėl ją reikia derinti ir taisyti. Tai atliekama derintojo (Debugger) programa.



3 pav. Borland'o kompiliatorius

OPERATORIAI

„="

Iš pradžių pabandysime pasiaiškinti visiškai paprastą programą. Pabandykite suprasti, ką daro ši programa?

```
#include <stdio.h>

main ( ) /* paprasta programa*/
{
   int num;

   num = 1;
   printf ("Aš paprasta");
   printf ("skaičiavimo mašina. \n");
   printf ("Mano mėgstamiausias skaičius %d todėl, kad tai pirmas skaičius. \n", num);
   return 0;
}
```

Jei manote, kad programa kažką parodys ekrane, tai Jūs esate visiškai teisūs.

Programos rezultatas:

Aš paprasta skaičiavimo mašina.
Mano mėgstamiausias skaičius 1 todėl, kad tai pirmas skaičius.

Parašytos programos apžvalga

#include <stdio.h> – į programą įtraukiama papildoma byla. Ši byla paprastai yra bet kuriame C kompiliatoriaus

pakete. Gali būti nurodoma ne viena, o kelios reikalingos bibliotekos. Programuotojai tai vadina programos antrašte. Ši eilutė net nėra C kalbos operatorius. # nurodo, kad programai vykdyti reikės C kompiliatoriaus bibliotekos.

main ( ) – funkcijos vardas. Programa, kuri yra parašyta C kalba, pradedama vykdyti nuo funkcijos **main** ( ). Todėl C kalbos visos paprogramės gali turėti įvairius vardus, išskyrus valdančiąją. Paprogramė – tai atskira nepriklausoma programos dalis, į kurią nukreipta pagrindinė funkcija. Skliausteliai po **main** ( ) nurodo, kad tai ne kintamasis, o funkcija. Šiuose skliaustuose gali būti nurodoma šios funkcijos grąžinama informacija. Nurodytoje programoje funkcijai nieko neperduodama, todėl ir ji nieko negrąžina.

/* paprasta programa*/ – komentaras.

Komentarai – tai pastabos, kurios padeda suprasti programos esmę. Komentarai skirti tik programuotojui, kompiliatorius jų nepastebi. Komentarus galima rašyti toje pat eilutėje, kurioje nurodomos operacijos.

{ – funkcijos pradžia.

int num; – kintamojo tipo aprašymas. Aprašydami kintamuosius nurodome jų tipą ir vardą. Šiuo atveju nurodome, kad bus naudojamas kintamasis num, kuris bus sveikas skaičius (**int**).

Programuojant yra būtina nurodyti naudojamų kintamųjų tipą. Programa pradedama rašyti nuo kintamųjų aprašymo, kurie bus naudojami programoje, nurodant jų tipą. Kintamaisiais gali būti ne tik sveikieji skaičiai, bet ir simboliai bei slankaus kablelio skaičiai. Kabliataškiu baigiamas rašyti bet

koks operatorius C kalboje. Kintamųjų tipai gali būti įvairūs: char, int, float, double ir t. t.

Pasirenkant kintamųjų vardus taip pat laikomasi tam tikrų taisyklių: kintamojo pavadinimas gali būti nuo vieno iki septynių daugiau simbolių. Kintamojo vardo pirmasis simbolis būtinai turi būti raidė, o toliau − skaičiai, didžiosios ir mažosios raidės ir „_" simbolis, kuris suprantamas kaip raidė.

Teisingi vardai	**Klaidingi vardai**
Wiggly	$Z^**
cat1	1cat
Hot_Tub	Hot-Tub
_kcaB	don't

num = 1; − priskyrimo operatorius. „=" kintamajam num priskiria vienetą. Apibrėžus kintamojo tipą, kompiuterio atmintyje kintamajam buvo išskirta atminties vieta, o su priskyrimo operatoriumi mes tą vietą užpildėme. Šis operatorius irgi baigiamas kabliataškiu.

printf ("Aš paprasta"); − išvedimo ekrane operatorius.

Šis operatorius ekrane parašo frazę: *Aš paprasta*.

printf ( ) − išvedimo ekrane funkcija. Kad tai yra funkcija, parodo skliaustai. Simbolių eilutė, esanti skliaustuose, perduodama funkcijai **printf** ( ). Ši funkcija peržiūri visus simbolius tarp kabučių ir juos parašo. Tarp kabučių rašomi simboliai − tai funkcijai perduodami argumentai.

\n − kompiliatoriui nurodo į kitos eilutės pradžią. Tai simbolis, kuris atitinka įvedimo klavišo paspaudimą. Simbolis \t − žymi tabuliacijos klavišo paspaudimą ir pan.

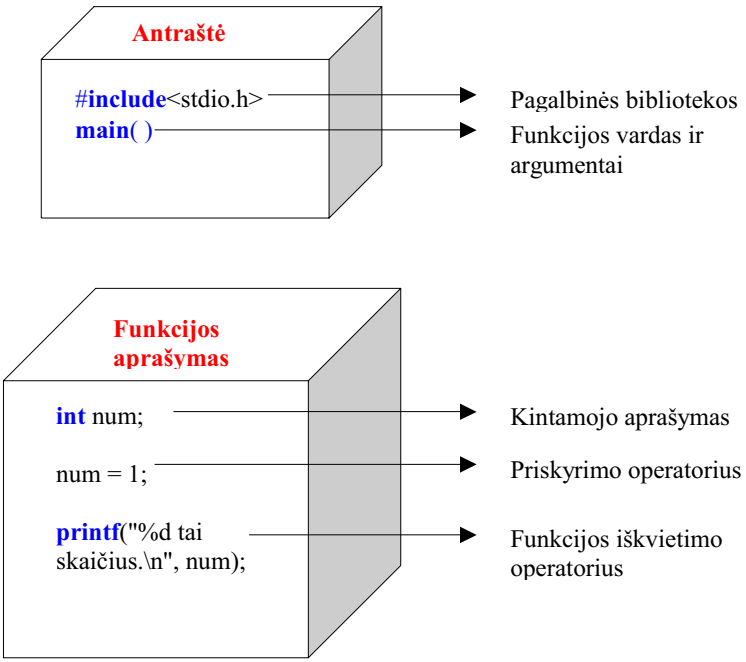
%d − nurodo, kur ir kokiu formatu pažymėti kintamojo

num vertę. % – nurodo, kad šioje vietoje turi būti spausdinamas skaičius, o d – nurodo, kad žymimas sveikasis skaičius turi būti dešimtainėje sistemoje.

} – pagrindinės **main** ( ) funkcijos pabaiga.

PAPRASTOS PROGRAMOS STRUKTŪRA

Susipažinsime su pagrindinėmis C kalbos programos rašymo taisyklėmis. Paprastai programą gali sudaryti viena ar kelios funkcijos, iš kurių viena būtinai turi vadintis **main** ( ). Funkciją pradedame aprašyti nuo antraštės, toliau aprašoma funkcija. Funkcijos struktūros paprasta schema pateikta 4 pav.



4 pav. C kalbos funkcijos struktūra

Programuojant nereikėtų visko rašyti į vieną eilutę. Kompiliatorius parašytoje programoje į tuščias eilutes nereaguoja. Todėl galima detaliai atskirti kiekvieną funkcijos dalį.

Pateiksime šiek tiek sudėtingesnę programą:

```
#include <stdio.h>

main ( ) /* skaičių daugyba*/
{
int a, b;

a = 6;
b = 2 * a;
printf ("%d padauginus iš dviejų gausime %d. \n", a, b);
return 0;

}
```

Šios programos rezultatas*: 6 padauginę iš dviejų gausime 12*. Kaip matome, jei ekrane reikia išvesti kelis kintamuosius, tai po kabučių iš eilės nurodomi kintamieji, kurių reikšmes ekrane norime išvesti.

Užduotys

1. Parašykite programą, kuri ekrane išvestų Jūsų vardą ir pavardę.
2. Parašykite programą, kuri ekrane išvestų Jūsų amžių, kai nurodyti Jūsų gimimo metai.

DUOMENŲ IR KINTAMŲJŲ TIPAI

Duomenų ir kintamųjų tipai bei jų raktinių žodžių paaiškinimas yra pateiktas 1 lentelėje.

1 lentelė

Duomenų tipai ir jų raktažodžiai

Raktiniai žodžiai: **int**, **long**, **short**, **unsigned**, **char**, **float**, **double**.	
int – jei kintamasis sveikasis skaičius su ženklu. **long** arba **long int** – dideli sveikieji skaičiai **short** arba **short int** – nedideli sveikieji skaičiai	Skaičiai nuo –32768 iki 32767 –2147483648 iki 2147483647 –32768 iki 32767
unsigned int, **unsigned long**, **insigned short** – gali būti nulis arba teigiamas sveikasis skaičius	
char – simbolinis kintamasis	
float – teigiamas ir neigiamas slankaus kablelio skaičius.	skaičiai nuo $3{,}4 \cdot 10^{-38}$ iki $3{,}4 \cdot 10^{38}$
double arba **long float** – dvigubo tikslumo arba ilgas dvigubas skaičius	$1{,}7 \cdot 10^{-308}$ iki $1{,}7 \cdot 10^{308}$

Žmogui skirtumas tarp sveikojo skaičiaus ir slankaus kablelio skaičiaus — tai tik skirtingas užrašymo būdas, o kompiuteriui — šių skaičių užrašymas į kompiuterio atmintį. Sveikųjų skaičių pavyzdžiai: 2, —23, 2456. Tuo tarpu 3,14 arba 2/3 jau nėra sveikieji skaičiai.

Pagrindiniai šių skaičių skirtumai:

1. Sveikieji skaičiai neturi trupmeninės dalies, o slankaus kablelio skaičiai gali būti sveiki arba trupmeniniai.
2. Veiksmai su trupmeniniais skaičiais atliekami ilgiau.

Norint, kad ekrane būtų parašytas sveikasis skaičius, reikia rašyti %d, simbolį — %c, slankaus kablelio skaičių — %f.

Užduotys

Studentas parašė programą, kurioje gausu klaidų. Suraskite jas.

```
#include <stdio.h>

main ( )
(
   float g; h;
   float tax, rate;

   g = e21;
   tax = rate * g;
)
```

SIMBOLINĖS EILUTĖS, FUNKCIJOS *PRINTF ( )* IR *SCANF ( )*

Šiame skyriuje pabandysime išsiaiškinti, kaip reikia apibrėžti simbolines konstantas ir kaip dirbti su simbolinėmis eilutėmis. Iš pradžių pateiksime programos pavyzdį ir pabandysime suprasti, ką ji daro:

```
/* dialogas*/
#define DENSITY 1200 /*žmogaus kūno tankis*/
#include<stdio.h>
#include<conio.h>
#include<string.h>

main ( )
{
   float weight, volume;
```

```
    int size, letters;
    char name[40];

    clrscr ();
    printf ("Labas! Koks Jūsų vardas? \n");
    scanf ("%s", name);
    printf ("%s, kokia Jūsų masė?\n", name);
    scanf ("%f", &weight);
    size = sizeof (name);
    letters = strlen (name);
    volume = weight/DENSITY;
    printf ("Nuostabu, %s, Jūsų tūris %2.2f kubiniai metrai.\n", name, volume);
    printf ("Be to, Jūsų vardas sudarytas iš %d raidžių. \n", letters);
    printf ("Ir jis kompiuterio atmintyje užima %d baitų.\n", size);
    getch ( );

    return 0;
}
```

Programos veikimo rezultatas:

```
Labas! Koks Jūsų vardas?
Viktorija
Viktorija, koks Jūsų svoris?
64
Nuostabu, Viktorija, Jūsų tūris 0,053 kubiniai metrai
Be to, Jūsų vardas sudarytas iš 9 raidžių,
Ir jis kompiuterio atmintyje užima 40 baitų.
```

Peržiūrėsime, kas gi nauja atsirado šioje programoje:

1. Čia buvo apibrėžtas masyvas, kuriame saugomas vartotojo įvestas vardas.

2. Vedant ir išvedant simbolinę eilutę buvo naudota išvedimo specifikacija %s.

3. Buvo apibrėžta konstanta DENSITY.

4. Įvesta įvesties funkcija **scanf** ( ), kuri nuskaito vedamus duomenis.

5. Įvestos eilutės ilgiui nustatyti naudota funkcija **strlen** ( ).

6. Masyvo dydžiui nustatyti panaudota nauja funkcija **sizeof ( )**.

scanf ( ) ir **printf** ( ) funkcijų viduje įvesta ir norima išvesti informacija rašoma kabutėse, kurios nurodo eilutės pradžią ir pabaigą. Simbolinei eilutei įvesi panaudotas masyvas — tai kompiuterio atmintyje išskirta vieta, kurioje šalia patalpinama tarpusavyje logiškai susijusi vienodo tipo informacija (žr. 5 pav.).

V	I	K	T	O	R	I	J	A	\0		

5 pav. Eilutės išdėstymas masyve. Kiekvienas masyvo elementas užima 1 baitą. \0 – C kalboje žymi eilutės pabaigą

Pateiktame pavyzdyje yra apibrėžtas 40 elementų masyvas, kiekviename masyvo elemente galima patalpinti vieną simbolį. Laužtiniai skliaustai rodo, kad *name* kintamasis yra masyvas, kuris turi 40 elementų, o **char** nurodo, kad elementų tipas yra simbolinis:

char name[40];

Simbolis %s nurodo funkcijai **printf** ( ) žymėti simbolinę eilutę. Funkcija **scanf** ( ) skaito simbolius iš klaviatūros, kol

sutinka tarpo, tabuliacijos klavišo ir įvedimo klavišo simbolį. Duomenims įvesti arba nuskaityti C kalboje yra ir daugiau funkcijų, pvz.: **gets** ( ), **getchar** ( ).

Rašant programas, dažnai tenka susidurti su konstantomis, kurių nuolat prireikia skaičiuojant. Todėl konstantos yra apibrėžiamos pradžioje **#define**, paskui nurodomas konstantos vardas, o po tarpo nurodoma konstantos reikšmė. Jei tai simbolinė konstanta, tai jos reikšmė nurodoma kabutėse:

#define DENSITY 1200 – skaitinė konstanta;

#define PHRAISE "Štai ir aš" – simbolinė eilutė – konstanta.

Kiekvienoje programos vietoje, kur sutinkamos nurodytos konstantos, į jų vietą iš karto įrašomos konstantų reikšmės.

Funkcijos **scanf** ( ) ir **printf** ( ) – įvedimo ir išvedimo funkcijos. Norint, kad šios funkcijos išvestų arba įvestų kintamąjį, reikia nurodyti kintamųjų formatą. Toliau pateikti išvedimo / įvedimo formatai ir išvedamos / įvedamos informacijos tipas:

Formatas	**Išvedamos / įvedamos informacijos tipas**
%d	dešimtainis sveikasis skaičius;
%c	vienas simbolis;
%s	simbolių eilutė;
%e	slankaus kablelio skaičius, išvedimas eksponentiniu formatu;
%f	slankaus kablelio skaičius, dešimtaininis išvedimas;
%g	naudojama vietoj f ir e, jeigu jis yra trumpesnis;
%u	dešimtainis sveikasis skaičius be ženklo;

%o aštuntainės sistemos skaičius be ženklo;
%x šešioliktainės sistemos skaičius be ženklo.

Galima tarp % ir simbolio nurodyti skaičius:

%4d – spausdinamam skaičiui skirti 4 simboliai, pildoma iš dešinės į kairę
_ _ _ 5;

%4.5f – skaičius po taško nurodo išvedimo tikslumą, t. y. kiek skaičių po kablelio išvesti ekrane;
%ld – atitinka long duomenų tipą;
%-10d – išvedimui skirta 10 simbolių, spausdinama iš kairės į dešinę.

Įrašant duomenis klaviatūra, kaip matėme **scanf ( )** funkcijoje, nurodant kintamąjį, kuriam bus priskirta nuskaityta reikšmė, naudojamas simbolis &. Šis simbolis reiškia nuorodą į kintamąjį, arba kintamojo adresą. Jei kintamasis apibrėžtas *p, tai kintamojo adresas bus p; jei kintamasis apibrėžtas p, tai kintamojo adresas bus &p.

Užduotys

1. Parašykite programą, kuri paklaustų Jūsų vardo, pavardės, gimimo metų. Po to išvestų Jūsų amžių, suskaičiuotų raides ir pasakytų, kiek vietos kompiuterio atmintyje užima Jūsų duomenys.
2. Suraskite klaidas:

```
define B oi-oi
define X 10

main ( )
   {
```

```
int age;
char name;

printf ("Koks Tavo vardas?");
scanf ("%s", name);
printf ("nuostabu, %c, kiek jums metų?\n", name);
scanf ("%f", age);
xp = age + X;
printf ("%s! Jums tikriausiai %d ?\n", B, xp);
}
```

VEIKSMAI, REIŠKINIAI IR OPERATORIAI

Šiame skyrelyje aptarsime duomenų apdorojimo operacijas: sudėtį, atimtį, daugybą ir dalybą; sužinosime apie ciklą. Ciklas – tai eilė veiksmų, kurie nuolatos yra kartojami. Vienas iš ciklo operatorių – **while**. Panagrinėkime programos pavyzdį:

```
/*batų dydis*/
   #define OFFSET 7,64
   #define SCALE 0,325

main ( )
{
/* perskaičiuoja batų dydį į pėdos dydį coliais*/

float shoe, foot;

printf ("Batų dydis Pėdos dydis\n");
shoe = 3,0;
while (shoe < 18,5)
   {
```

```
    foot = SCALE*shoe + OFFSET;
    printf ("%13.2f %16.2f coliai\n", shoe, foot);
    shoe = shoe + 1,0;
    }
printf("Jei Jums ši avalynė tinka, nešiokite ją. \n");
return 0;
}
```

Programos veikimo rezultatas:

```
Batų dydis Pėdos dydis
3,01          8,61 coliai
4,0           8,94 coliai
..            ..
17,0          13,16 coliai
18,0          13,49 coliai
Jei Jums ši avalynė tinka, nešiokite ją.
```

while ciklo darbas: sąlyga, kuri turi būti tenkinama, nurodoma skliaustuose. Kol sąlyga shoe < 18,5 bus tenkinama, tol ciklas bus vykdomas. Pradinis rezultatas − shoe = 3,0. Apskaičiuotas pėdos dydis parašomas, o shoe kintamojo reikšmė padidinama vienetu: shoe = shoe +1,0;. Ciklo pradžią ir pabaigą žymi riestiniai skliaustai {}. Tarp jų esantys operatoriai kartojami tol, kol ciklo sąlygos yra tenkinamos. Kai sąlyga netenkinama, tada išeinama iš ciklo ir vykdoma kita komanda **printf** ( ). Šią programą galima pakeisti, jei vietoj SCALE reikšmės parašysime 1,8, o vietoj OFFSET − 32,0, Tuomet gausime programą, kuri temperatūrą žymi ne pagal Celcijaus, o pagal Farenheito skalę.

PAGRINDINIAI VEIKSMAI

Pagrindinės operacijos yra skirtos aritmetiniams veiksmams atlikti.

1. Priskyrimo veiksmas: „=".

C kalboje šis ženklas nereiškia lygybės. Jis pažymi kintamajam priskirtą reikšmę, pvz.,

```
Bmw = 2003;
```

Toks užrašas reiškia, kad kintamajam Bmw priskirta reikšmė 2003, t. y. kintamojo vardas – Bmw, kintamojo reikšmė – 2003.

Pažiūrėkime į pavyzdį:

```
i = i + 1;
```

Matematiniu požiūriu tai būtų neteisinga, o C kalboje tai reiškia, kad reikia paimti kintamąjį vardu i ir jo vertę padidinti vienetu. Trumpai tai galima užrašyti: i++, t. y. pats kaip ir i = i + 1.

2. „+" – tai sumavimo veiksmas.

Sudedami du dydžiai, esantys operatoriaus kairėje ir dešinėje pusėje. Pvz., **printf** ("%d", 4 + 20);

Ekrane bus išvesta *24*.

Sumuojami dydžiai gali būti ir kintamieji, ir konstantos.

3. „–" – tai atimties veiksmas.

Atima kintamųjų arba konstantų reikšmes: iš kairėje ženklo pusėje stovinčio kintamojo reikšmės atimama dešinėje ženklo pusėje stovinčio kintamojo reikšmė.

4. „−" − tai ženklo pakeitimo veiksmas.

Pakeičia kintamojo ženklą priešingu.

5. „* " − tai daugybos veiksmas.

Padaugina kintamuosius ar konstantas. Specialaus operatoriaus laipsniui kelti C kalboje nėra.

6. „ / " − tai dalybos operatorius.

Operatoriaus kairėje pusėje esančio kintamojo reikšmė yra dalijama iš kintamojo reikšmės, esančios operatoriaus dešinėje pusėje.

Toliau pateiktos programos pavyzdys parodo, kaip atliekama dalybos operacija ir kuo skiriasi sveikųjų skaičių dalyba nuo slankaus kablelio skaičių dalybos:

```
/*dalybos  pavyzdžiai*/
...

main ( )
   {
   printf ("sveikųjų skaičių dalyba: 5/4 tai %d \n", 5/4);
   printf ("sveikųjų skaičių dalyba: 6/3 tai %d \n", 6/3);
   printf ("slankiuoju kableliu: 7,0/4,0 tai %2.2f \n", 7,0/4,0);
   return  0;
   }
```

Programos veikimo rezultatas:

sveikųjų skaičių dalyba: 5/4 tai 1

sveikųjų skaičių dalyba: 6/3 tai 2
slankaus kablelio skaičių dalyba: 7,0/4,0 tai 1,75

7. „%" – tai liekanos operatorius.

Jei turime užrašą 13%5, tai rezultatas bus 3: 13, kurį galima užrašyti 2*5 + 3, liekana yra 3. Tai ir yra operatoriaus veikimo rezultatas.

8. „++" ir „– –" – tai didinimo ir mažinimo operatoriai.

„++" – padidina kintamojo vertę 1, o „– –" sumažina vertę 1. Šiems operatoriams vykdyti yra galimi keli variantai. Tai priklauso nuo to, kur šie operatoriai rašomi: prieš funkciją ar po jos. Veikimo rezultatas tas pats, tik reikšmės pakeitimo laikas skirtingas.

Paanalizuokime toliau pateiktą programą:

```
/* sumavimas*/

main ( ) /*operatoriaus rašymas iš kairės ir iš dešinės*/
{
int a = 1, b = 1;
int aplus, plusb;

   aplus = a++;
   plusb = ++b;
   printf ("a aplus b plusb");
   printf ("%3d %5d %5d %5d \n", a, aplius, b, pliusb);
  return 0;
}
```

Programos veikimo rezultatas:

A aplus b plusb
2 1 2 2

Abiejų kintamųjų reikšmės padidėjo vienetu, tačiau kintamajam *aplus a* reikšmė buvo priskirta prieš padidinimą: kintamajam priskirta sena reikšmė a ir tik po to a reikšmė padidinama; o *plusb* – po padidinimo: kintamojo reikšmė padidinama, o tik po to ta reikšmė priskiriama naujam kintamajam.

Analogiškai veiksmai atliekami ir su mažinimo operatoriumi: „– –". 2 lentelėje pateikiamos C kalbos operacijos ir jų atlikimo tvarka.

2 lentelė

Aritmetiniai veiksmai ir jų atlikimo tvarka

Aritmetiniai veiksmai	
+	Prie kairėje pusėje stovinčio kintamojo pridedama dešinėje stovinčio kintamojo reikšmė.
–	Iš kairėje stovinčio kintamojo atimama dešinėje stovinčio kintamojo reikšmė.
–	Pakeičia dešinėje šio ženklo pusėje stovinčio kintamojo ženklą.
*	Padaugina kintamuosius.
/	Kairėje pusėje esantį kintamąjį padalija iš dešinėje pusėje esančio kintamojo.
%	Išveda skaičiaus liekaną, kai dalijama šio operatoriaus kairėje stovinčio kintamojo reikšmė iš šio operatoriaus dešinėje pusėje esančio kintamojo reikšmės.
++	Kintamojo vertę padidina vienetu.
– –	Kintamojo vertę sumažina vienetu.
Veiksmai (išdėstyti atlikimo tvarka)	**Kaip vykdoma operacija**
() {} -> .	Iš kairės į dešinę
! ~ ++ – – – * & sizeof( )	Iš dešinės į kairę
* / %	Iš kairės į dešinę
+ –	Iš kairės į dešinę
<< >>	Iš kairės į dešinę
< < = > > =	Iš kairės į dešinę
= = ! =	Iš kairės į dešinę
&	Iš kairės į dešinę
^	Iš kairės į dešinę
|	Iš kairės į dešinę
&&	Iš kairės į dešinę
||	Iš kairės į dešinę
?:	Iš kairės į dešinę
= + = – = * = / * % =	Iš dešinės į kairę
,	Iš kairės į dešinę

Dabar pabandysime pasiaiškinti anksčiau neminėtas operacijas.

+ = − prideda dešinėje esantį kintamąjį prie kairėje esančio kintamojo: a + = 2, tai atitinka a = a + 2.

Analogiškai atliekami ir tokie veiksmai: − =, * =.

< = − mažiau arba lygu;
= = − lygu;
! = − nelygu;
&& − loginis veiksmas reiškiantis ir;
|| − loginis veiksmas reiškiantis arba;
! − loginis veiksmas ne.

Užduotys

1. Parašykite programą, kuri pavaizduotų skaičiaus x nuo 1 iki 10 vertę, jo kvadratą, kubą ir 1/x:

x	x^2	x^3	1/x
1	1	1	1
2	4	8	0,50
3	9	27	0,33
..	..	..	..
10	100	1000	0,10

2. ASCII kodų lentelė. Tai standartinė kompiuteryje naudojamų simbolių lentelė. Operatorius **printf** ("%4d %c", 97, 97) išspausdina ekrane tokius simbolius: 97 a. Mat, pirmasis 97 skaičius spausdinamas kaip sveikasis, o antrasis − kaip simbolis, pažymėtas numeriu 97. Išspausdinkite visus 256 simbolius.

3. Parašykite programą, kuri pažymėtų skaičius nuo 1 iki 16 dešimtainėje, aštuntainėje, šešioliktainėje sistemose.

4. Parašykite programą, kuri paprašytų, kokio skaičiaus kvadratų sumą norite sužinoti. Pvz., 4 − 1*1 + 2*2 + 3*3 + 4*4 = 30. Ekrane bus išvesta 30.

LOGINĖS OPERACIJOS

Raktažodžiai: **if, else, switch, break, case, default**

Operacijos:

> > = <= < = = ! = && || / :?

OPERATORIUS IF

Pasiaiškinsime paprastą programą:

```
/*eilučių skaičiavimas*/

#include <stdio.h>

void main ( )
{
   int ch;
   int lin = 0;

   while ((ch = getchar( )) ! = EOF)
   if (ch = = '\n')
   lin + +;

   printf ("suskaičiavau %d eilutes\n", lin);
}
```

Pagrindinį darbą šioje programoje atlieka operatorius:

```
   if (ch = = "\n")
   lin + +;
```

Šis operatorius kompiuteriui nurodo didinti kintamojo lin vertę vienetu, jei klaviatūros simbolis yra nauja eilutė „\n". Jei klaviatūros simbolis neatitinka naujos eilutės simbolio, tai toliau, vykdant operatorių **while,** imamas kitas simbolis. Operatorius **getchar ( )** yra skirtas vienam simboliui įvesti iš klaviatūros. Vienam simboliui išvesti naudojamas operatorius **putchar** ( ).

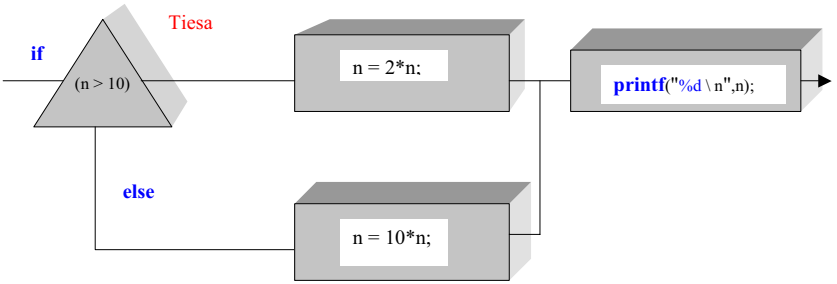
EOF – tai „-1". Paprastai automatiškai tokiu simboliu yra pabaigiama byla.

Šiame pavyzdyje operatoriui **if** priklauso tik vienas veiksmas lin++, todėl šis veiksmas baigiamas kabliataškiu. Jei operatoriui priklausytų daugiau veiksmų, juos reiktų atskirti skliaustais {}. Galima patobulinti programą, kad ji skaičiuotų simbolius ir eilutes:

```
#include <stdio.h>

void main ( )
{
   int ch;
   int lin = 0, sim = 0;

   while ((ch = getchar ( )) ! = EOF)
{
   sim++;
if (ch = = '\n')
lin++;
}

   printf ("suskaičiavau %d eilutes ir %d simbolius \n",
lin, sim);
}
```

Operatoriaus **if** galimybes galima išplėsti panaudojus operatorių **else**. Naudojant konstrukciją **if else**, tikrinama ope-

ratoriaus **if** sąlyga ir, jei ji netenkinama, tuomet vykdomi **else** operatoriaus veiksmai:



Jei nebūtų operatoriaus **else**, tai iš karto būtų spausdinama. Po operatoriaus **else** galima vėl kartoti operatorių **if**. Tokiu būdu galima patikrinti didesnį sąlygų skaičių.

SĄLYGINĖS OPERACIJOS

Kai kurias C kalbos sąlygines komandas jau naudojome anksčiau, dabar susipažinkime su visomis galimomis šios kalbos sąlyginėmis operacijomis:

Operacija	**Reikšmė**
<	mažiau
< =	mažiau arba lygu
= =	lygu
> =	daugiau arba lygu
>	daugiau
! =	nelygu

Sąlyginės komandos naudojamos su **if** ir **while** operatoriais. Atkreipiame dėmesį — jokiu būdu lyginimui negalima naudoti operatoriaus =, nes tai yra priskyrimo operatorius. Deja, sąlyginių komandų negalima naudoti eilutėms lyginti.

Jei dirbama su trupmeniniais skaičiais, patartina naudoti tik < ir > lyginimo komandas.

LOGINĖS OPERACIJOS

Dažnai tenka panaudoti ne tik sąlygines operacijas, bet ir logines. Pavyzdžiui, mums reikia suskaičiuoti, kiek byloje yra tuščių simbolių, t. y. tarpo simbolių, naujos eilutės ir tabuliacijos klavišo.

```
/*simbolių skaičius*/
 #include <stdio.h>

main ( )
{
   int sim;
   int t = 0;

   while ((sim = getchar( ))! = EOF)
         if (sim = =' ' || sim = = "\n" || sim = = "\t")
         t++;
   printf ("Iš viso yra %d tuščių simbolių", t);
   return 0;
}
```

Taigi || yra loginė komanda „arba". C kalboje yra trys loginės komandos:

Operacija	Reikšmė
&&	ir
||	arba
!	ne

Pažiūrėkime, kaip jos vykdomos. Tarkime, turime du sąly-

ginius operatorius a ir b:

1. a&& b sąlyga bus teisinga ir vykdoma, kada bus teisingos a ir b sąlygos.

2. a||b teisinga, jei yra teisinga a arba b sąlyga, arba teisingos abi sąlygos.

3. !a teisinga, jei a sąlyga yra neteisinga.

Paimkime keletą konkrečių pavyzdžių:

5 > 2 && 4 > 7 – neteisinga, nes viena sąlyga nėra patenkinta.

5 > 2 || 4 > 7 – teisinga, nes viena sąlyga yra patenkinta.

!(4 > 7) – teisinga, nes 4 ne didesnis už 7.

SĄLYGINĖ OPERACIJA: ?:

Ši operacija – tai trumpas **if – else** operatoriaus užrašymas. Pateiksime operatoriaus išraišką absoliutinei skaičiaus reikšmei rasti:

```
x = (y < 0)? – y : y;
```

Tai atitiktų užrašymą:

```
if (y < 0)
   x = –y;
else
   x = y;
```

OPERATORIUS *SWITCH*

Jei reikia pasirinkti vieną galimą variantą iš eilės duomenų, tuomet nepatogu naudoti **if–else** konstrukciją. Tada pra-

vartu naudotis operatoriumi **switch**. Pateiksime pavyzdį programos, kuri nuskaito raidę ir nurodo iš tos raidės prasidedančio gyvūno pavadinimą.

```
/*gyvūnas*/

void main ( )
{
char ch;

printf ("Įveskite bet kokią raidę, o aš išvesiu gyvūno pavadinimą iš Jūsų nurodytos raidės. \n");
printf ("Jei norite darbą nutraukti, įveskite  # simbolį.\n");

while ((ch = getchar ( ))!="#")
   {
   if ( ch!=" ") /*nelygu tarpo simboliui*/

         switch  (ch)
         {
         case "a": printf ("avis, naminis gyvūnas.\n");
         break;
         case "b": printf (" barsukas, laukinis gyvūnas.\n");
         break;
         ...
         default: printf ("tokio gyvūno nežinau.\n");
         break;
         }

   else
         printf ("Jūs įvedėte tarpo klavišą. Įveskite raidę arba, norėdami baigti darbą, simbolį #");
   }
}
```

Tikriausiai aišku, kad parašytoji programa nuskaito iš klaviatūros įvestą simbolį. Patikrina, ar tai ne tarpo simbolis arba pabaigos simbolis „#". Jei sąlygos tenkinamos, tuomet ieško **switch** operatoriuje nurodytos raidės ir veiksmų, ką toliau daryti su duomenimis. **default** – bus naudojama visiems anksčiau nenurodytiems atvejams; **break** skirtas išeiti iš **switch** operatoriaus.

Užduotys

1. Parašykite programą, kuri tikrintų, ar iš klaviatūros įvestas skaičius yra teigiamas, ar neigiamas. Neigiamas skaičius turi būti paverstas teigiamu ir išvestas ekrane, tai pranešant, o teigiamas – tik pranešant, kad tai teigiamas skaičius.
2. Parašykite programą, kuri skaičiuotų nurodyto skaičiaus faktorialą.
3. Parašykite programą, kuri pateiktų meniu. Meniu būtų galima pasirinkti: ar išvesti nurodyto skaičiaus kubą, ar kvadratinę šaknį, ar faktorialą.

CIKLAI IR KITI PROGRAMOS VALDYMO BŪDAI

Raktažodžiai: **while**, **do**, **for**, **break**, **continue**, **goto**

Operacijos:

+= -= *= /= %=

Priminsime, kad ciklo **while** bendriausia užrašymo forma yra:

while (sąlyga)
  Operatorius (veiksmai, kurie bus atliekami)

Šiame cikle turime nurodyti, kaip turi keistis į sąlygą įeinantis kintamasis. Kitaip ciklas bus begalinis, ir iš jo niekada neišeisime.

Ciklas **for** − jame iš karto nurodomos ciklo pradinės ir galinės sąlygos, ciklo atlikimo žingsnis.

```
....
for (a = 1; a < = 10; a++)
printf (" man puikiai sekasi!");
....
```

Šios programos fragmento vykdymas: ekrane dešimt kartų pasirodys frazė „*man puikiai sekasi!*".

Pateikiame su šiuo ciklo operatoriumi parašytą programą, kuri skaičiuoja skaičių kubus nuo 1 iki 6.

```
/*kubai*/

#include <stdio.h>
#include <conio.h>

void main ( )
{
   int a;

   for (a = 1; a < = 6; a++)
   printf ("%5d %5d \n", a, a*a*a);
}
```

Jei į ciklą įeina ne vienas, o keletas sakinių, tuomet jie turi būti atskirti skliaustais {}. To nereikia, jei ciklui priklauso tik vienas sakinys.

for ciklo pranašumai:

1. Sąlygos ir žingsniai užrašomi iš karto.

2. Žingsnį galima ne tik didinti (a++), bet ir mažinti (a--).

3. Ciklo žingsnį galima keisti bet kokiu nurodytu dydžiu: a += 13 (a = a + 13).

4. Galima naudotis ne tik skaičiais, bet ir simboliais. Naudojantis simboliais, jie nurodomi tarp kabučių:

```
...
for (a = "a"; a<= "z"; a++)
printf ("simbolis %c atitinka %d skaičių.\n", a, a);
...
```

5. for ciklo viduje galima naudoti aritmetinius veiksmus:

```
...
for (a = 1; a*a*a < = 216; a++)
printf ("%5d %5d \n", a, a*a*a);
...
```

6. Žingsniui keisti galima panaudoti kokią norime algebrinę išraišką.

7. Vietas ciklui aprašyti galima palikti tuščias, svarbu, kad nebūtų praleistas kabliataškis.

```
...
for (; ;)
printf ("pakibau...\n");
...
```

Šis ciklas bus vykdomas labai ilgai, kadangi tuščia sąlyga visada yra teisinga.

```
...
a = 2;
for (n = 3;a < = 25;)
a = a*n;
...
```

Tai bus vykdoma, kol a bus mažiau arba lygu 25.

8. Galima nurodyti ne vieną, o kelias pradines sąlygas. Jos tarpusavyje turi būti atskirtos kableliu.

```
...
for(a = 2, b = 0; b < 100; a* = 2)
   b + = a;
...
```

Kiti valdantieji operatoriai

break, continue, goto

Operatorius **break** naudojamas daugiausiai iš visų trijų operatorių. Jį jau sutikome **while** cikle, kuris baigiamas tik šio operatoriaus dėka. Operatorius **break** tikrai netinka naudoti su **if** operatoriumi. Tačiau jei atsitinka taip, kad operatorių **break** reikia naudoti, tuomet būtina peržiūrėti parašytos programos algoritmą, kad šio operatoriaus nereikėtų.

Operatorius **continue** gali būti naudojamas visuose cikluose, išskyrus **switch**. Jis skirtas peršokti kai kurias ciklo iteracijas. Šie operatoriai geriausiai praverčia, kai reikia sutrumpinti **if**−**else** sąlygos veikimą.

Operatorius **goto** yra prastas. Jis C kalboje nenaudojamas. Šio operatoriaus naudojimas − prasto programavimo požymis.

MASYVAI

Jau anksčiau minėjome, kas yra masyvas ir kaip jį reikia apibrėžti. Dar kartą prie jo grįšime, kadangi programavimo procese jis yra svarbus. Pvz.,

```
float a[20];
```

Kintamojo aprašymas reiškia, kad turime masyvą a, kurį sudaro dvidešimt elementų. Pirmas masyvo elementas yra a[0], antras − a[1] ir t. t. Paskutinis masyvo elementas − a[19]. Kadangi masyvo tipas yra **float**, tai kiekvienas masyvo elementas irgi bus **float** tipo. Masyvai gali būti bet kokio anksčiau nurodyto tipo. Masyvo užpildymas ir jo elementų skaitymas atliekamas ciklu. Svarbu neužmiršti, kad masyvo pirmas elementas yra a[0].

```
...
for (i = 0; i < = 19; i++) /*masyvo užpildymas*/
scanf ("%lf", &a[i]);
for (i = 0; i < = 19; i++)
printf ("%f", a[i]);  /*masyvo elementų išspausdinimas*/
...
```

Masyvų elementus galima lyginti, atlikti aritmetinius veiksmus:

```
...

if (a[i] > b)
    b = a[i];
...
```

Jei masyvo i-tasis elementas didesnis už b, tuomet b kintamajam reikia priskirti i-tąjį masyvo elementą.

Svarbu: masyvui užpildyti nurodomas adresas: &a[i].

Masyvo dydį galima apibrėžti pasinaudojus konstantomis:

```
...
#define MAX 45

main ( )
{
 int a[MAX];
...
}
```

Užduotys

1. Parašykite programą, kuri išvestų daugybos lentelę.
2. Parašykite programą, kuri leistų vartotojui užpildyti masyvą iš 6 elementų. Užpildyto masyvo elementus išdėstykite didėjančia tvarka ir išveskite ekrane.
3. Parašykite programą, kuri sudėtų dviejų masyvų elementus (masyvai sudaryti iš 10 elementų; sudėtis turi būti: pirmo masyvo pradžia sudedama su antro masyvo pabaiga) ir rezultatą surašytų į trečią masyvą, o padalintus masyvo elementus sudėtų į ketvirtą masyvą. Ekrane turi būti pavaizduoti pradiniai masyvai, jų sumos ir dalybos masyvai.

KAIP TEISINGAI NAUDOTIS FUNKCIJOMIS?

Raktažodis: **return**

Programuojant C kalba naudojamės funkcijomis. Anksčiau jau buvo pasinaudota **printf ( )**, **scanf ( )**, **getch ( )**, **putchar ( )**, **strlen ( )** funkcijomis. Šios funkcijos yra sisteminės, bet

mes esame sukūrę ir savų funkcijų, pvz., **main ( )**. Programa visada prasideda komandomis (jos yra **main ( )** funkcijoje), kuriomis galima kreiptis ir į kitas funkcijas. Dabar išsiaiškinkime, kaip patiems sukurti funkcijas, į kurias galėtų būti nukreipta **main ( )** funkcija ir kitos sukurtos funkcijos.

Funkcija – tai savarankiška programos dalis, skirta konkrečiam veiksmui atlikti. Pvz., funkcija **printf ( )** išveda informaciją ekrane. Naudojant funkcijas, nereikia dar kartą programuoti besikartojančių veiksmų. Jei programoje kokį nors veiksmą reikia kartoti keletą kartų, tą veiksmą užtenka aprašyti funkcija ir, reikalui esant, į ją kreiptis. Be to, šią funkciją galima naudoti ne tik vienoje programoje, bet ir kitose.

Tarkime, norime parašyti programą, kuri įvestų skaičių rinkinį, jį suskirstytų ir rastų vidutinę reikšmę. Minėtą programą galima užrašyti taip:

```
...
main ( )
{
    float list[50];

    readlist  (list);
    sort (list);
    average (list);

    return  0;
}
```

Aišku, kad pagrindinė funkcija nukreipta į funkcijas **readlist ( )**, **sort ( )** ir **average ( )**, kurios atlieka veiksmus, o pagrindinei funkcijai pateikia rezultatą. Naudojant pagalbines funkcijas, pagrindinį dėmesį galima skirti programos struktūrai

negaištant laiko jų detalėms.

Apie funkcijas reikia žinoti, kaip jos turi būti aprašomos, kaip į jas kreiptis ir kaip nurodyti ryšį tarp programos ir parašytos funkcijos.

PAPRASTOS FUNKCIJOS KŪRIMAS IR JOS PANAUDOJIMAS

Parašysime programą, kuri spausdintų firminį blanką ir sukursime naują funkciją, kuri brėžtų 65 simbolius „*".

```
/*firminio blanko viršus*/
#define Name "Vilniaus pedagoginis universitetas"
#define Address "Studentų 39"
#define Vieta "Vilnius"
#include <stdio.h>

void starbar ( );

void main ( )
{
   starbar ( );
   printf ("%s\n", Name);
   printf ("%s\n", Address);
   printf ("%s\n", Vieta);
   starbar ( );
}

/*funkcija starbar ( )*/
#include <stdio.h>
#  define Riba 65
```

```
void starbar ( )
{
   int count;
         for (count = 1; count < = Riba; count + +)
               putchar ("*");
         putchar ("\n");
}
```

Programos veikimo rezultatas:

```
****************************************
```

Vilniaus pedagoginis universitetas
Studentų 39
Vilnius

```
****************************************
```

Į funkciją **starbar ( )** mes kreipėmės iš funkcijos **main ( )**, nurodydami tik reikalingos funkcijos vardą. Kaip veikia programa: pirma, ji iš karto kreipiasi į **starbar ( )** funkciją, kuri pažymi simbolius, toliau **main ( )** funkcija kreipiasi į sistemines funkcijas, kurios atspausdina tekstą ir galiausiai — vėl į **starbar ( )**, kuri spausdina simbolius. Kuriant pagalbinę funkciją, ji rašoma pagal tokias pačias taisykles kaip ir **main ( )** funkcija. Čia abi funkcijos buvo užrašytos į vieną bylą *.cpp. Paprastai geriau rašyti atskiras funkcijas į atskiras bylas, kurias būtų galima panaudoti kitose programose.

Mūsų parašytoji programa nieko neturi grąžinti funkcijai **main ( )**, todėl priekyje buvo nurodomas jos tipas **void**. Jei programa turėtų grąžinti reikšmę, reiktų nurodyti, kokio tipo reikšmę funkcija grąžins, taip pat grąžinamo argumento reikšmę: **int starbar** (**int** a); tai reikštų, kad funkcija **starbar ( )** grąžins sveiką skaičių ir apdorojimui gaus taip pat sveiką skaičių a. Tuomet sukurtos funkcijos viduje reikia pasinaudoti operatoriumi **return**, kuris sukurtai funkcijai nurodo, ką ji turi grąžinti.

Pasiaiškinsime toliau pateiktą programą:

```
/*absoliutinės reikšmės*/
#include <stdio.h>

int abs (int x);
void main ( )
{
int a = 10, b = 0, c = −22;
int d, e, f;
   d = abs(a);
   e = abs(b);
   f = abs(c);
   printf ("%d %d %d\n", d, e, f);

}

int abs (int x)
{
   int y;

   y = (x < 0)? − x : x;
   return y;
}
```

Programos veikimo rezultatas:

10 0 22

Pagrindinė funkcija kreipiasi į kitą funkciją, kuri skaičiuoja skaičiaus modulį. Suskaičiavusi modulį **return** komanda pateikia rezultatą pagrindinei funkcijai, kuri pastarąjį pavaizduoja ekrane.

GLOBALIEJI (IŠORINIAI) IR LOKALIEJI (VIETINIAI) KINTAMIEJI

Kintamuosius mes aprašėme funkcijos viduje, todėl tai buvo tos funkcijos lokalieji (vidiniai) kintamieji, kurie nebuvo žinomi kitoms funkcijoms. Todėl vienai funkcijai perduoti kintamojo vertę buvo naudojama komanda **return**.

C kalboje dažnai tenka naudoti globaliuosius kintamuosius, kuriuos naudos keletas funkcijų. Jei kintamasis yra apibrėžtas ir vienoje, ir kitoje funkcijoje tuo pačiu vardu, jį kompiliatorius vis tiek supranta kaip atskirą kintamąjį. Panagrinėkime programą, kuri sukeičia kintamųjų reikšmes:

```
/*kintamųjų sukeitimas*/
#include <stdio.h>
#include <conio.h>

int interchange (int *u, int *v);

void main ( )
{
   int x = 5, y = 10;
   printf ("esamos x = %d ir y = %d reikšmės\n", x, y);
   interchange (&x, &y);
   printf ("sukeistos x = %d ir y = %d reikšmės \n", x, y);
}

int interchange (int *u, int *v)
{
   int a;
   a = *u;
   *u = *v;
   *v = a;
```

```
    return *u, *v;

}
```

Šioje programoje funkcija **main ( )** siunčia ne kintamąjį, o kintamojo adresą &x, &y. Tuo tarpu funkcijos **interchange ( )** kintamieji apibrėžiami kaip rodyklės *u, *v. Rodyklė — tai kintamojo, esančio nurodytu adresu, reikšmė, t. y. a = *u. Tai reiškia kintamajam a priskirti x vertę, nes ši funkcija gavo ne reikšmę, bet adresą. Tokiu būdu, naudojant lokaliuosius kintamuosius, vienai funkcijai galima perduoti kitos reikšmes.

Globalieji kintamieji apibrėžiami prieš funkciją **main ( )** naudojant **extern** komandą (**extern** — nurodo, kad tai bus išorinis, visoms funkcijoms bendras kintamasis).

Užduotys

1. Perrašykite meniu programą pasinaudodami savo sukurtomis funkcijomis skaičiaus kubui, kvadratinei šakniai, faktorialui skaičiuoti.
2. Parašykite funkciją, kuri, gavusi kintamųjų x ir y reikšmes, jas pakeičia jų suma ir skirtumu bei jas išspausdina.
3. Parašykite atskiras funkcijas, kurias naudoja pagrindinė funkcija skaičių sumai, sandaugai ir faktorialui skaičiuoti.

REKURSINĖS FUNKCIJOS

Rekursinė funkcija — tai funkcija, kuri kreipiasi pati į save. Pasiaiškinkime, kaip parašyti funkciją faktorialui skaičiuoti. Faktorialą galima apskaičiuoti dviem būdais:

$n! = 1 \cdot 2 \cdots (n-1) \cdot n$

arba
$n! = n \cdot (n-1)!,\ 0! = 1.$

Antrasis faktorialo užrašymo būdas − faktorialo lygtis. Pagal pirmąjį apibrėžimą faktorialo skaičiavimo programą Jūs jau rašėte. Dabar pateiksime faktorialo skaičiavimo funkciją pagal antąjį apibrėžimą:

```
...
int fact (int n)
{
 if (n = 0) return 1;
else return n*fact (n−1);
}
...
```

Kaip matome, funkcija užrašoma trumpiau ir nėra jokio ciklo. Ši funkcija kreipsis pati į save tol, kol nebus lygu nuliui. Rekursinės funkcijos naudingos, kai reikia programuoti veiksmus su nežinomu operacijų skaičiumi.

Užduotis

1. Parašykite programą, kuri ekrane užrašo dešimt pirmųjų Fibonači skaičių. Fibonači skaičiai tenkina tokią diskretinę lygtį:
$f_n = f_{n-1} + f_{n-2}$, $f_0 = 0$, $f_1 = 1$. Šiai lygčiai reikia parašyti rekursinę funkciją.

MASYVAI IR RODYKLĖS

Jau žinome, kas yra masyvai ir kaip apibrėžti statinį (apibrėžto dydžio) masyvą. Apibrėžto masyvo elementams gali-

ma iš karto priskirti reikšmes:

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

int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};

void main ( )
{
   int i;
   for (i = 0; i < 12; i++)
         printf (" %d mėnuo turi %d dienų. \n", i+1, da-
ys[i]);
}
```

Šiuo atveju masyvas apibrėžtas kaip išorinis, todėl pačioje funkcijoje jo apibrėžti nereikia. Galima ir nenurodyti masyvo elementų skaičiaus. Tuomet pats kompiliatorius suskaičiuos laužtiniuose skliaustuose pateiktas reikšmes:

```
int days [] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

void main ( )
{
   int i;
   for (i = 0; i < sizeof (days) / sizeof (int); i++)
         printf (" %d mėnuo turi %d dienų. \n", i + 1,
days [i]);
}
```

Šiuo atveju pats kompiliatorius apibrėžė masyvo dydį. Cikle **for** norint nurodyti masyvo dydį mes funkcijos **sizeof ( )** pa-

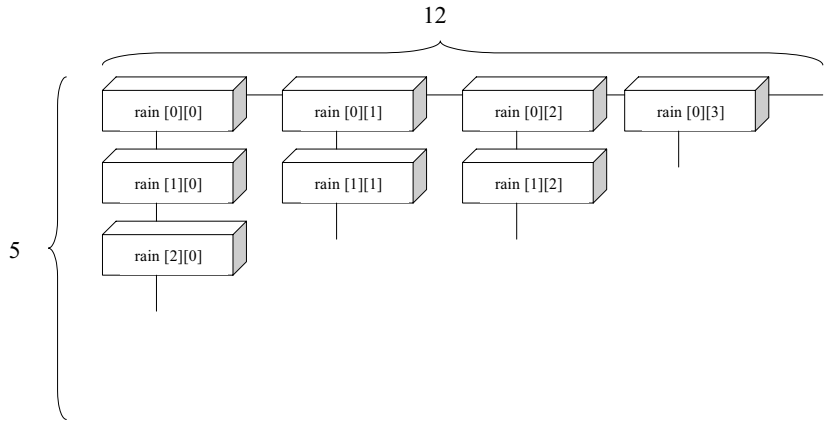
galba masyvo dydį apskaičiuojame baitais. Kadangi masyvo elementai yra sveiko tipo kintamieji, kurie užima du baitus kompiuterio atmintyje, tai norint gauti masyvo skaičių, padalijame jo užimamą vietą iš 2 ir gauname masyvo elementų skaičių.

Siunčiant masyvą funkcijai, elgiamasi taip pat kaip ir su kintamaisiais – naudojant rodykles ir adresus.

Iki šiol buvo naudotas tik vienmatis masyvas. Dažnai prireikia dvimačių arba trimačių masyvų. Pvz.,

float rain [5][12];

Čia apibrėžtas penkių elementų masyvas, kurio kiekvieną elementą sudaro 12 elementų. Tokio tipo masyvą galima pavaizduoti taip:



Keičiant masyvo elementus, elgiamasi taip pat, kaip ir dirbant su vienmačiu masyvu. Reikia pabrėžti, kad C kalbos masyvai yra labai paprasti, todėl galima manyti, kad jų iš viso nėra.

Jei masyvo dydis bus apskaičiuotas tik veikiant programai, tai masyvą reikia apibrėžti kaip rodyklę:

```
....
float *J0;
....
J0 = (float *) malloc (zz * sizeof (float));
....
```

Anksčiau pateiktame pavyzdyje masyvas iš pradžių apibrėžtas kaip rodyklė, veikiant programai apskaičiuota, kiek reikės masyvo elementų zz. Tuomet funkcija **malloc ( )** yra sukuriamas reikiamo dydžio masyvas. Primename, kad zz yra masyvo elementų skaičius, o **sizeof** (**float**) nurodo **float** tipo kintamųjų užimamą vietą kompiuterio atmintyje baitais.

ĮVESTIES IR IŠVESTIES BYLOS C KALBOJE

Teks rašyti programas, kurioms duomenys bus imami iš bylos, o skaičiavimo rezultatai taip pat bus išvesti byloje. Iš pradžių pasiaiškinsime paprastas bylos skaitymo funkcijas **fopen ( ), fclose ( ), getc ( ), putc ( )**.

Panagrinėkime programą, kuri skaito bylos *test* turinį ir jį išveda ekrane.

```
#include <stdio.h>

void main ( )
{
   FILE *in;  /* aprašo nuorodą į bylą*/
   int ch;

   if ((in = fopen ("test","r"))! = NULL)
/*atidaro bylą skaitymui ir tikrina ar yra tokia byla*/
{
```

```
while ((ch= getc(in))!=EOF)
    putc (ch, stdout);
    /* išveda bylą ekrane*/
    fclose (in); /*uždaro bylą*/
}
   else
    printf ("bylos atidaryti negalėjau");
}
```

Bylos atidarymas: fopen ( )

Funkcijai **fopen ( )** reikia perduoti tris duomenis: 1) bylos pavadinimą; 2) nurodyti, kaip naudosime bylą:

"r": bylą skaitysime

"w": į bylą įrašysime duomenis

"a": papildysime bylą

3) rodyklė į bylą:

```
FILE *in;

in = fopen ("test", "r");
```

Dabar in yra rodyklė į bylą „test".

Jei **fopen** ( ) negali atidaryti bylos, ji grąžina vertę „NULL".

Bylos uždarymas: fclose ( )

Pateiktame pavyzdyje tai yra

```
fclose (in);
```

Atkreipiame dėmesį, kad jos argumentas yra rodyklė in, o ne byla test. Ši funkcija, jei bylą pavyko uždaryti sėkmingai, grąžina 0, jei ne − „−1".

Bylos duomenų įvedimas ir išvedimas: getc ( ) ir putc ( )

Šios funkcijos veikia analogiškai kaip ir funkcijos **getchar ( )** ir **putchar ( )**. Skirtumas tik tas, kad reikia pranešti, kokią bylą reikia naudoti. Todėl **getchar ( )**:

ch = **getchar ( )**;
reikia pakeisti:
ch = **getc** (in);
Analogiškai **putc ( )**:
putc (ch, out);

skirta simbolio ch užrašymui byloje, į kurią siunčia **FILE** tipo rodyklė out.

Mūsų atveju buvo naudota **stdout** − tai rodyklė standartiniam išvedimui.

Bylos įvedimas − išvedimas: fprintf ( ), fscanf ( ), fgets ( ), fputs ( )

Kintamojo ir informacijos išvedimo skirtumas yra tik tas, kad išvedant informaciją byloje reikia naudoti rodyklę, kurios tipas yra **FILE**. Pvz:

```
....
FILE *out;
...
out = fopen ("rez.dat", "w"));
....
fprintf (out, "\n j1= %f v2= %d ", J11, v111);
....
```

Funkcijos fprintf ( ) ir fscanf ( ).

Šios funkcijos panašios į funkcijas printf ( ) ir scanf ( ), tik joms reikalingas papildomas argumentas, rodantis siuntimą į bylą. Tai nurodoma pačioje pradžioje. Pvz.:

```
#include <stdio.h>

void main ( )
{
   FILE *fi;  /* aprašo nuorodą į bylą*/
   int age;

   if ((fi = fopen ("test","r"))!=NULL)
{
/*atidaro bylą skaitymui ir tikrina, ar yra tokia byla*/
   fscanf (fi, "%d", &age); /*fi rodo į test*/
   fclose (fi);
fi = fopen ("data","a"); /*papildymas*/
fprintf (fi, "test is %d.\n", age); /* fi nurodo data*/
fclose (fi);
   }
}
```

Panaši yra ir funkcija **fgets ( )**, kuri nuo funkcijos **gets** ( ) skiriasi papildomu kintamuoju:

```
/* bylą nuskaito eilutėmis*/
#include <stdio.h>
#define MAXLIN 80
main ( )
{
   FILE *f1;
   char *string [MAXLIN];

   f1 = fopen ("story","r");
   while (fgets (string, MAXLIN, f1) !=NULL)
   puts (string);
}
```

Pirmas **fgets ( )** funkcijos argumentas yra skaitomos eilutės padėtis. Čia įvedama iš bylos perskaityta informacija (ji bus įrašoma į simbolinį masyvą).

Antrasis argumentas – nurodo skaitomos eilutės ilgį. Trečiasis argumentas nurodo bylą, iš kurios skaitoma informacija.

Skirtumas tarp **gets ( )** ir **fgets ( )** – **gets ( )** keičia naujos eilutės simbolį į „\0", o **fgets ( )** išlaiko šį simbolį. Abi funkcijos bylos pabaigoje EOF grąžina reikšmę „NULL".

Funkcija **fputs ( )** atlieka panašius veiksmus kaip ir funkcija **puts ( )**:

fputs ("Tu teisus", fileptr);

perduoda eilutę „Tu teisus" į bylą, kurią parodo nuoroda **fileptr**.

SIMBOLINIŲ EILUČIŲ PAKEITIMAS

Dažnai perskaitytą simbolinę eilutę reikia pakeisti į atitinkamą skaitinę vertę. Tam yra naudojamos funkcijos **atoi ( )** ir **atof ( )**. Pirmoji funkcija eilutę paverčia sveikuoju skaičiumi, antroji – slankiojo kablelio skaičiumi. Šių funkcijų argumentas yra simbolinio tipo. Atvirkštinės paskirties funkcijos: **itoa ( )** – sveiko tipo skaičių paverčia eilute, **ftoa ( )** – *double* tipo skaičių paverčia eilute.

Užduotys

1. Parašykite programą gautoms knygoms inventorizuoti. Įvedami duomenys turi būti įrašomi į atskirą bylą.

2. Parašykite programą, kuri nuskaitytų bylos duomenis. Iš pradžių turi būti sukurta byla, kurioje būtų nurodyti kokie nors skaičiai, o nuskaičiusi bylos turinį pavaizduotų ekrane,

patikrintų, ar nėra nulių, o jei jie yra, juos ištrintų, naujus skaičius įrašytų į naują bylą.

LITERATŪRA

1. Blanskis J. ir kt. C++ praktikumas. – Kaunas : KTU, 2001.
2. Lipeikienė J. Programavimas C++ kalba. – Vilnius : VPU l-kla, 2002.
3. Matulis A. C, C++ ir OOP. http://www.itpa.lt/baryon/CPP/LCpp.pdf
4. Vidžiūnas A. C++ duomenų tipai ir struktūros. – Kaunas : Smaltija, 2000.
5. Vidžiūnas A. C++ ir C++ Builder pradmenys. – Kaunas : Smaltija, 2002.
6. Уэйт М., Прата С., Мартин Д., Язык Си. – Москва : Мир, 1988.

PRIEDAS

C kalbos raktažodžiai

Programos vykdymo raktažodžiai:
Ciklai:
for while do
Pasirinkimas ir sąlygos:
if else switch case default
Perėjimas:
break continue goto
Duomenų tipai:
char int short long unsigned float double struct union

typedef

Atminties klasės:

auto extern register static

Programos vykdymo valdymas

Operatorius while

Užrašymo forma:

```
while (sąlyga)
   operatorius;
```

Operatorius kartojamas tol, kol sąlyga teisinga.

Pavyzdžiai:

```
while (n++ < 100)
   printf ("%d %d\n", n, 2*n+1);

while (fargo < 1000)
{
   fargo = fargo + step;
   step = 2*step;
}
```

Operatorius for

Užrašymo forma:

```
for (priskyrimas; sąlyga; žingsnis)
   operatorius;
```

Operatorius vykdomas tol, kol tenkinama sąlyga.

Pavyzdžiai:

```
for (n = 0; n < 10; n++)
   printf ("%d %d\n", n, n*2+1);
```

Operatorius do while

Užrašymo forma:
do
    operatorius
while (sąlyga**)**;

Operatorius vykdomas, kol tenkinama sąlyga.

Pavyzdžiai:
do
    scanf ("%d", &num**)**
    while (num! = 20**)**;

Operatoriai if ir else

Užrašymo forma:
1 būdas

if (sąlyga)
    operatorius

Operatorius vykdomas, jei tenkinama sąlyga.

2 būdas

if (sąlyga)
    operatorius 1
else
    operatorius 2

Jei sąlyga teisinga, vykdomas operatorius 1, jei klaidinga

– operatorius 2.

3 būdas

```
if (sąlyga 1)
    operatorius 1
else if (sąlyga 2)
    operatorius 2
else
    operatorius 3
```

Jei sąlyga 1 teisinga, tai vykdomas operatorius 1. Jei sąlyga 1 klaidinga, o sąlyga 2 teisinga, vykdomas operatorius 2. Jei abi sąlygos klaidingos, vykdomas operatorius 3.

Pavyzdžiai:

```
if (a= = 4)
    printf ("tai arklys");
else if (a > 4)
    printf ("tai ne arklys");
else
{
    a++;
    printf ("klaida");
}
```

Operatorius switch

Užrašymo forma:

```
switch (išraiška)
{
    case 1 požymis: operatorius 1
    case 2 požymis: operatorius 2
```

```
    default          : operatorius 3
}
```

default nėra būtinas.

Pavyzdžiai:

```
switch (raidė)
{
    case "a": printf ("as");
    case "b": ...
    case "c":printf ("taip");
    default: printf ("nesiseka");
}
```

Jei paimtas simbolis nėra nei a, nei b, nei c — tai vykdomas **default** operatorius.

Aušra Kynienė
C kalbų ABC

Redagavo Reda Asakavičiūtė
Maketavo D. Petrauskas

SL 605. Tir. 150 egz. 3,75 sp. l. Užsak. Nr. 04-091
Išleido Vilniaus pedagoginis universitetas, Studentų g. 39, LT-08106, Vilnius
Maketavo ir spausdino VPU leidykla, T. Ševčenkos g. 31, LT-03111, Vilnius
Kaina sutartinė