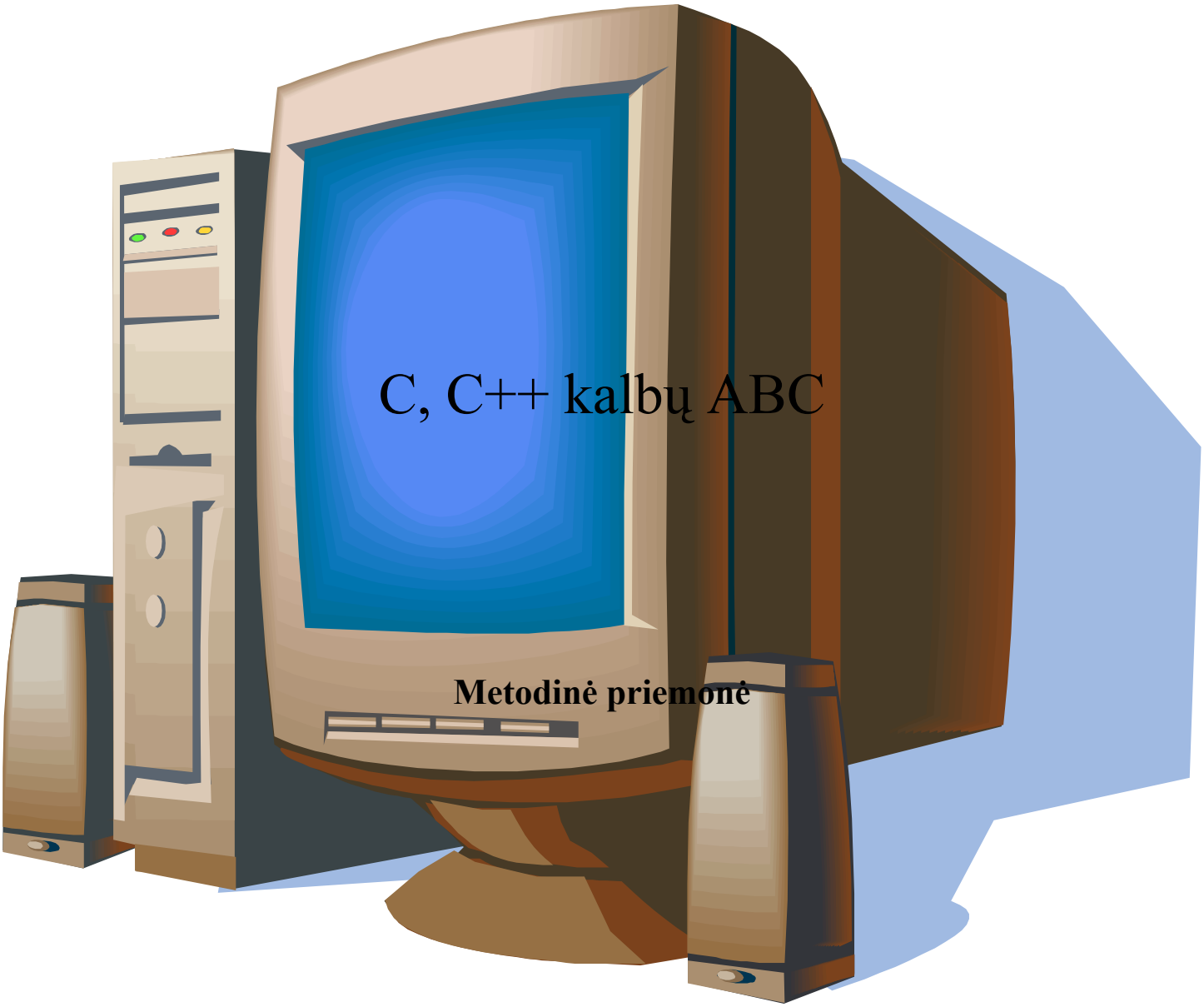
A.Kynienė



C, C++ kalbų ABC

Metodinė priemonė

Vilnius
2004

TURINYS

Pratarmė

"C, C++ kalbos ABC" metodinio leidinio paskirtis – supažindinti su C kalbos programavimo pagrindais. Ji skirta visiems, kurie nori išmokti programuoti šiuolaikine programuotojų kalba.

Programavime, kaip ir bet kuriame moksle, patirtis įgyjama dirbant, todėl šioje metodinėje priemonėje pateikiama daug savarankiškų užduočių. Ši metodinė priemonė šiek tiek praplėsta pradžiamoksliui: pateikiamos ir sudėtingesnės temos, kurių gali prireikti kuriant sudėtingesnes programas. Čia visiškai negalbama apie grafikos kūrimo elementus.

Leidinyje gausu pateiktų programų pavyzdžių, tačiau dalis programų yra nepilnos ir norint, kad šios programos veiktų, pačiam skaitytojui reiks jas papildyti. Gale pateiktame priede sudėta C kalboje naudojamų funkcijų santrauka.

Manau, šis leidinys padės studentui perprasti C kalbą ir pačiam pramokti rašyti programas šia kalba.

C programavimo kalba, tai galinga programavimo kalba, vis labiau naudojama visame pasaulyje. Dabar ši kalba tai pagrindinė profesionalų programavimo kalba, kuri sėkmingai vartojama tiek sisteminės, tiek taikomosios įrangos kūrimui. Pirmą kartą C kalbą aprašė jos autoriai B.W.Karnighan ir D.M.Ritchie 1978 metais išleistoje knygoje „C programavimo kalba“. C kalbos variantas su klasėmis pavadintas C++ kalba. Ši kalba buvo kaip instrumentas programuotojams – praktikams. Be šios kalbos, yra ir kitų programavimo kalbų: Paskalis – griežtas programavimas, Beisikas – jo sintaksė artima anglų kalbai. Iš esmės C++ yra nauja programavimo kalba, pritaikyta sudėtingų programų sistemų ir instrumentinių programavimo priemonių kūrimui, panaudojant objektinio programavimo technologiją. Tačiau šioje kalboje išsaugotas glaudus ryšys ir su klasikine C kalba.

C kalbos privalumai

C – šiuolaikinė programavimo kalba.
C – efektinga programavimo kalba. Ji leidžia geriausiai išnaudoti kompiuterinius resursus. 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 perkelti programos veikimą į UNIX.
C – galinga ir lanksti programavimo kalba. Didelė dalis galingos UNIX ir Windows operacinės sistemos parašyta C kalba. C kalba parašytos programos naudojamos fizikiniams ir techniniams uždaviniams spręsti, taip pat naudojamos ir animacijai kurti.
C – turi galimybę panaudoti eilę valdančiųjų konstrukcijų, kurios paprastai asocijuojasi su asembleriu. Asemblerio programavimo kalba labai sudėtinga (1pav.), tai skaičiai ir kodai, kuriuos suprasti ne specialistui sunku, kadangi ji rašoma procesoriaus kalba (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.

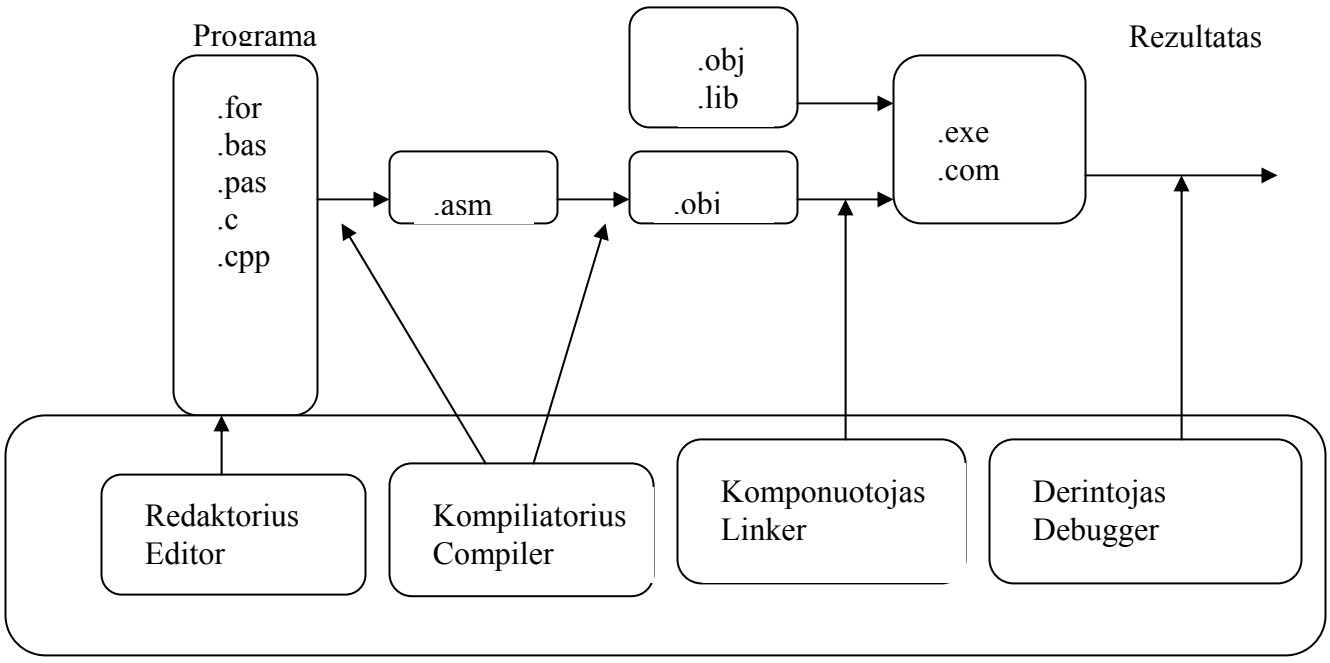
Asembleriu parašyta programa yra labai didelės apimties, nes kompiuteriui reikia aiškinti kiekvieną norimą atlikti veiksmą detaliai. Žinoma, programuojant asembleriu galima pasinaudoti visais kompiuterio resursais. Asembleris – tai sisteminių programuotojų kalba, kuri pilnai leidžia panaudoti visus kompiuterio resursus. Fortranas – puikiai suskaičiuoja sinusą, tačiau sunkiai pavaizduoja tašką ekrane. C kalba – tai aukšto lygio programavimo kalba, kuri leidžia panaudoti visus kompiuterio resursus kaip ir asembleris. Šiuo metu C kalba – yra sisteminių programuotojų kalba.
C – patogi kalba. Tai struktūrinė kalba, tačiau ji nėra labai griežta ir per daug nevaržo programuotojo.

Programavimo procesas

Programos rašymas yra gana ilgas procesas, kuris schematiškai yra pavaizduotas 3 pav. Pirma, 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 (pirmasis C++ kalba, antrasis – C).

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

Galiausiai, visus programų gabaliukus sujungia į vieną *.exe bylą kita programa, vadinama komponuotoju (linker). Ši programos dalis kartu įtraukia ir bibliotekines funkcijas. Tačiau gautoji programa yra su klaidomis, todėl ją reikia derinti ir taisyti. Tai atliekama derintojo (Debugger) programos pagalba.



3 paveikslas. Borland'o kompiliatorius

Operacijos

“=”

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

```
#include <stdio.h>

main ( ) /* paprasčiausia 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;
}
```

Na jei manote, kad programa kažką atspausdins 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 jau yra bet kuriame C kompiliatoriaus pakete. Programuotojai tai vadina programos antrašte. Ši eilutė net nėra C kalbos operatorius. # - nurodo, kad programos vykdymui bus reikalinga pasinaudoti C kompiliatoriaus biblioteka.

main ( ) – funkcijos vardas. Programa, kuri yra parašyta C kalboje, pradedama vykdyti nuo funkcijos **main ( )**. Todėl C kalboje, visos paprogramės gali turėti įvairius vardus, išskyrus valdančiąją. Paprogramė – tai atskira nepriklausoma programos dalis, į kurią kreipiasi pagrindinė funkcija. Skliausteliai po **main ( )** nurodo, kad tai ne kintamasis, o funkcija. Šiuose skliaustuose gali būti nurodoma šios funkcijos grąžinama informacija. Kadangi nurodytoje programoje mūsų funkcija nieko neturi grąžinti, tai jie yra tušti.

/* paprasčiausia programa*/ - komentaras.

Komentarai – tai pastabos, kurios padeda suprasti programos esmę. Komentarai skirti tik programuotojui, kompiliatorius juos ignoruoja. Komentarus galima rašyti toje pat eilutėje kur ir nurodomos operacijos.

{ - funkcijos pradžia.

int num; - operatorius kintamojo tipui aprašyti arba kitaip, tai paprasčiausias raktinis žodis. Nurodome, kad bus naudojamas kintamasis num, kuris bus sveikas skaičius (int).

Programuojant yra būtina nurodyti naudojamų kintamųjų tipą. Programa rašyti pradedama nuo kintamųjų, kurie bus naudojami programoje, nurodant jų tipą. Kintamaisiais gali būti ne tik sveiki skaičiai, bet ir simboliai bei slankaus kablelio skaičiai. Kabliataškiu baigiamas rašyti bet koks operatorius C kalboje. Kintamųjų tipai gali būti įvairūs.

Pasirenkant kintamųjų vardus taip pat laikomasi tam tikrų taisyklių: kintamojo pavadinimas gali būti nuo vieno simbolio iki septynių. Kintamojo vardo pirmasis simbolis būtinai turi būti raidė, o sekantys – 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 buvo išskirta kintamajam atminties vieta, o su priskyrimo operatoriumi, mes tą vietą užpildėme. Šis operatorius irgi baigiamas kabliataškiu.

printf (" Aš paprasta"); - išvedimo į ekraną operatorius.

Šio operatoriaus pagalba ekrane atspausdinama frazė: *Aš paprasta*.

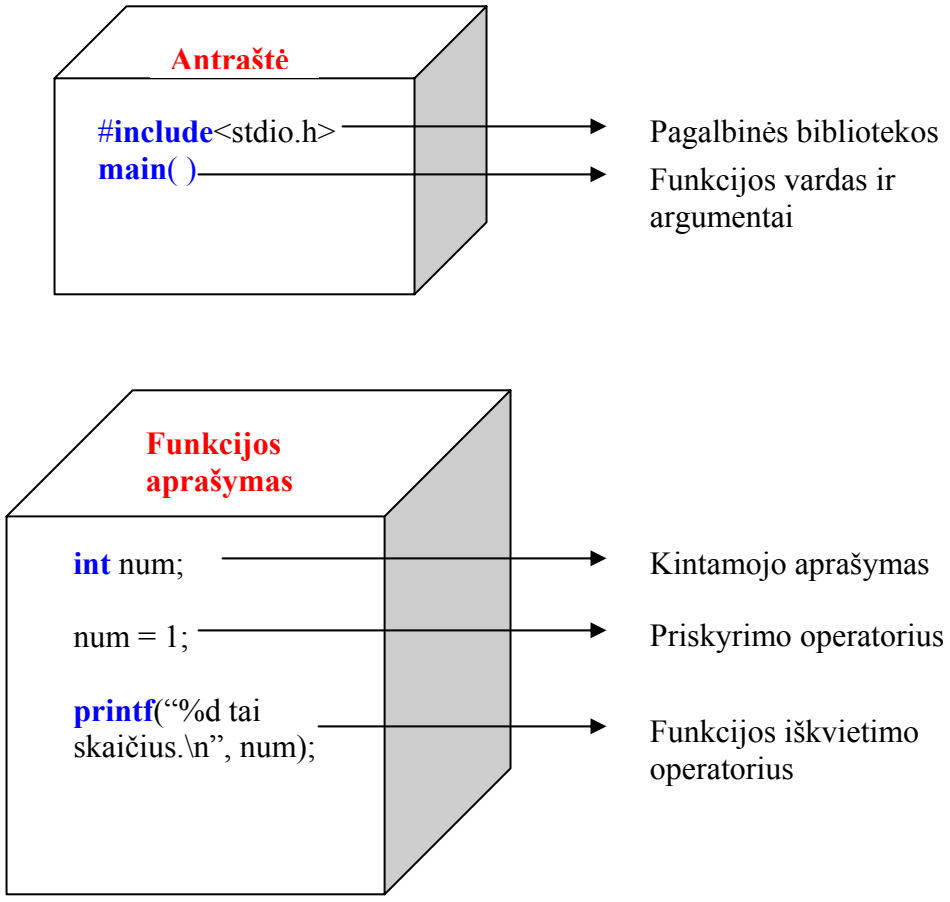
printf( ) – išvedimo į ekraną funkcija. Kad tai funkcija, rodo skliaustai. Simbolių eilutė esanti skliaustuose, perduodama funkcijai **printf**( ). Ši funkcija peržiūri visus simbolius tarp kabučių ir juos atspausdina. Tarp kabučių rašomi simboliai, tai funkcijai perduodami argumentai.
\n – nurodo kompiliatoriui į kitos eilutės pradžią. Tai simbolis, kuris atitinka įvedimo klavišo paspaudimą. Simbolis \t – atitinka tabuliacijos klavišo paspaudimą ir pan.

%d – nurodo kur ir kokiu formatu atspausdinti kintamojo num vertę. % - nurodo, kad šioje vietoje turi būti spausdinamas skaičius, o d – nurodo, kad spausdinamas skaičius turi būti dešimtainėje sistemoje.

} – pagrindinės main( ) funkcijos pabaiga.

Paprastos programos struktūra

Susipažinsime su pagrindinėmis taisyklėmis rašant programą C kalba. Paprastai, programą gali sudaryti viena ar keletas funkcijų, kurių viena būtinai turi vadintis **main( )**. Funkciją pradedame aprašyti nuo antraštės, po to seka pačios funkcijos aprašymas. Funkcijos struktūros paprasčiausia schema pateikta 4 pav.



4 paveikslas. Funkcijos struktūra C kalboje.

Programuojant nereikėtų rašyti visko į vieną eilutę. Kompiliatorius, parašytoje programoje, tuščias eilutes ignoruoja. 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 padauginus iš dviejų gausime 12*. Kaip matome, jei į ekraną reikia išvesti kelis kintamuosius, tai po kabučių nurodomi iš eilės kintamieji, kurių vertes norime išvesti į ekraną.

Užduotys

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

Duomenų ir kintamųjų tipai

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

1 Lentelė. Duomenų Tipai ir jų raktiniai žodžiai.

Raktiniai žodžiai: **int**, **long**, **short**, **unsigned**, **char**, **float**, **double**.	
int - jei kintamasis bus sveikas skaičius su ženklu. **long** arba **long int** – dideli sveiki skaičiai **short** arba **short int** – nedideli sveikieji skaičiai	Skaičiai nuo –32768 iki 32767
unsigned int, **unsigned long**, **insigned short** – gali būti nulis arba teigiamas sveikas skaičius	
char – simbolinis kintamasis	

float – teigiamas ir neigiamas slankaus kablelio skaičius.	Skaičiai, didesni nei sveikieji
double arba **long float** – dvigubo tikslumo arba ilgas dvigubas skaičius	

Žmogui skirtumas tarp sveiko skaičiaus ir slankaus kablelio skaičiaus, tai tik skirtingas užrašymo būdas. Kompiuteriui – tai šių skaičių užrašymas į kompiuterio atmintį. Sveikų skaičių pavyzdžiai: 2, -23, 2456. Tuo tarpu 3,14 arba 2/3 jau nėra sveikieji skaičiai. Pagrindiniai šių skaičių skirtumai:

1. Sveiki skaičiai neturi trupmeninės dalies, tuo tarpu slankaus kablelio skaičiai – gali būti sveiki arba trupmeniniai.
2. Naudojant slankaus kablelio skaičius, galima apimti didesnį skaičių diapazoną.
3. Veiksmai su trupmeniniais skaičiais atliekami ilgiau.

Norint, kad ekrane būtų atspausdintas sveikas skaičius, reikia rašyti %d, simbolis - %c, slankaus kablelio skaičius - %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. Pradžioje 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, kokia Jūsų masė?
64
Nuostabu, Viktorija, Jūsų tūris 0.053 kūbiniai metrai.
Be to, Jūsų vardas sudarytas iš 9 raidžių,
Ir jo patalpinimas kompiuterio atmintyje užima 40 baitų.
```

Peržiūrėsime, kas gi naujo 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 tarp kabučių, kurios nurodo eilutės pradžią ir pabaigą. Simbolinės eilutės įvedimui panaudotas masyvas – tai kompiuterio atmintyje išskirta vieta, kurioje greta vienas kito patalpinama tarpusavyje logiškai susijusi vienodo tipo informacija (5 pav.).

V	I	K	T	O	R	I	J	A	\0		

5 paveikslas. 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( )** spausdinti simbolinę eilutę. Funkcija **scanf( )** skaito simbolius iš klaviatūros, kol sutinka tarpo, tabuliacijos klavišo ir įvedimo klavišo simbolį. Duomenų įvedimui arba nuskaitymui C kalboje yra ir daugiau funkcijų, pvz.: **gets( )**, **getchar( )**.

Rašant programas, dažnai tenka susidurti su konstantomis, kurių nuolat prireikia skaičiavimuose. Todėl konstantos yra apibrėžiamos pradžioje **#define**, po to nurodomas konstantos vardas ir 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 konstanta.

Programoje kiekvienoje vietoje, kur bus sutiktos nurodytos konstantos, į jų vietą iš karto bus įrašytos konstantų vertės.

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

Formatas	**Išvedamos/įvedamos informacijos tipas**
%d	dešimtainis sveikas 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 sveikas 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 tašku nurodo išvedimo tikslumą, t.y. kiek skaičių po kableliu išvesti į ekraną
%ld – atitinka long duomenų tipą.
%-10d – išvedimui skirta 10 simbolių, spausdinama bus iš kairės į dešinę.
Įrašant duomenis iš klaviatūros, kaip matėme **scanf( )** funkcijoje nurodant kintamąjį, kuriam bus priskirta nuskaityta vertė naudojamas simbolis &. Šis simbolis reiškia rodyklę į kintamąjį arba kitaip, kintamojo adresą. Jei kintamasis apibrėžtas *p, tai rodyklė į jį bus p; jei kintamasis apibrėžtas p, tai rodyklė į jį bus &p.

Užduotys

1. Parašykite programa, kuri paklaustų Jūsų vardo, pavardės, gimimo metų. Po to išvestų Jūsų amžių, suskaičiuotų raides ir pasakytų kiek vietos 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);
    }
```

Operacijos, išraiškos ir operatoriai

Čia aptarsime duomenų apdorojimo operacijas: sudėtis, atimtis, daugyba ir dalyba. Pirmą kartą susidursime su ciklu. Ciklas – tai eilė veiksmų, kurie nuolatos yra kartojami. Dabar pasiaiškinsime vieną iš ciklo operatorių: **while**. Pažiūrėkime programos pavyzdį:

```
/*batų dydis*/
    #define OFFSET 7.64
    #define SCALE 0.325

main ( )
{
```

```
/* perskaičiuoja batų dydį į pėdos dydį ciliais*/

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: sąlygą, kuri turi būti tenkinama nurodoma skliaustuose. Kol ši sąlyga shoe<18,5 bus tenkinama, tol ciklas bus vykdomas. Pradinis rezultatas shoe=3.0 tenkinamas, tai tuomet apskaičiuojamas pėdos dydis, jis atspausdinamas ir shoe kintamojo vertė 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, tai išeinama iš ciklo ir vykdoma sekanti komanda: **printf**( ). Šią programą galima pakeisti, jei vietoj SCALE vertės parašysime 1.8, o vietoj OFFSET – 32.0, gausime programą kuri temperatūrą iš celcijaus skalės perveda į farenheitus.

Pagrindinės operacijos

Pagrindinės operacijos yra skirtos aritmetiniams veiksmams.

1. Priskyrimo operacija: =

   C kalboje šis ženklas nereiškia lygu, jis reiškia, kad kažkokiam kintamajam turi būti priskirta vertė:

   ```
   Bmw = 2003;
   ```

Toks užrašas reiškia, kad kintamajam bmw priskirta vertė 2003., t.y. kintamojo vardas – bmw, kintamojo vertė – 2003.

Pažiūrėkime pavyzdį:

i=i+1;.

Matematiniu požiūriu tai yra visiška nesąmonė, o C kalboje tai reiškia, kad reikia paimti kintąmąjį vardu i ir jo vertę padidinti vienetu. Trumpai tai galima užrašyti: i++ , tai tas pats kaip ir i=i+1.

2. Sumavimo operacija: +

+ - sudedami du dydžiai, esantys iš kairės ir dešinės operatoriaus pusės. Pvz.:
printf(“%d”, 4+20);
Į ekraną bus išvesta - *24*.
Sumuojami dydžiai gali būti ir kintamieji ir konstantos.

3. Atimties operacija: -

Atima kintamųjų vertes arba konstantų vertes: iš kairės ženklo pusės stovinčio kintamojo vertės atimama dešinėje ženklo pusėje stovinčio kintamojo vertė.

4. Ženklo pakeitimo operacija: -

Pakeičia kintamojo ženklą priešingu.

5. Daugybos operatorius:*

Sudaugina kintamuosius ar konstantas. Specialaus operatoriaus kvadratui C kalboje nėra.

6. Dalybos operatorius: /

Kairėje pusėje esančio kintamojo vertė prieš operatorių yra dalinama iš kintamojo vertės, esančios už operatoriaus dešinėje pusėje.

Žemiau pateiktos programos pavyzdys pademonstruos kaip atliekama dalybos operacija ir kuo skiriasi sveikų skaičių dalyba nuo dalybos su slankaus kablelio skaičiumi.

```
/*dalybos pavyzdžiai*/
...

main( )
  {
  printf(“sveikų skaičių dalyba: 5/4 tai %d \n”, 5/4);
  printf(“sveikų 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:

sveikų skaičių dalyba: 5/4 tai 1
sveikų skaičių dalyba: 6/3 tai 2
slankiuoju kableliu: 7.0/4.0 tai 1.75

7. Dalybos pagal modulį operatorius: %

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

8. Didinimo ir mažinimo operatoriai: ++ ir - -

++ - padidina kintamojo vertę 1, o - - sumažina vertę 1. Šių operatorių vykdymo yra galimi keli variantai. Tai priklauso nuo to, kur šie operatoriai rašomi: prieš funkciją ar po jos. Veikimo rezultatas tas pats, tik vertės pakeitimo laikas skirtingas. Paanalizuokime žemiau 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");
    return 0;
}
```

Programos veikimo rezultatas:

A aplus b plusb
2 1 2 2

Abiejų kintamųjų vertės padidėjo vienetu, tačiau kintamajam aplus a vertė buvo priskirta prieš padidinimą: kintamajam priskirta sena vertė a ir tik po to a vertė padidinama; o plusb – po padidinimo: kintamojo vertė padidinama, o tik po to ta vertė priskiriama naujam kintamajam.
Analogiškai veiksmai atliekami ir su mažinimo operatoriumi: --. Žemiau pateiktoje lentelėje pateikiamos C kalbos operacijos ir jų atlikimo tvarka.

2 Lentelė. Aritmetiniai veiksmai ir jų atlikimo vyresniškumas.

Aritmetiniai veiksmai	
+	Prie kairėje pusėje stovinčio kintamojo pridedama dešinėje stovinčio kintamojo vertė
-	Iš kairėje stovinčio kintamojo atimama dešinėje stovinčio kintamojo vertė
-	Pakeičia dešinėje pusėje šio ženklo stovinčio kintamojo ženklą
*	Sudaugina kintamuosius
/	Kairėje pusėje esantį kintamąjį padalina iš dešinėje pusėje esančio kintamojo
%	Išveda liekaną skaičiaus, kai dalinama kairėje šio operatoriaus stovinčio kintamojo vertė iš dešinėje šio operatoriaus pusėje esančio kintamojo vertės
++	Kintamojo vertę padidina vienetu
--	Kintamojo vertę sumažina vienetu
Veiksmai (išdėstyti veiksmų 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 neminėtas anksčiau operacijas.
+= - prideda dešinėje esantį kintamąjį prie kairėje esančio kintamojo: a+=2, tai atitinka a=a+2.
Analogiškai atliekami ir sekantys veiksmai: -=, *=.

<= - mažiau arba lygu.
== - lygu
!= - nelygu
&& - loginis veiksmas reiškiantis ir.
|| - loginis veiksmas reiškiantis arba.
! – loginis veiksmas ne.

Užduotys

1. Parašykite programą, kuri atspausdintų 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) atspausdina ekrane tokius simbolius: 97 a. Mat, pirmasis 97 skaičius spausdinamas, kaip sveikasis, o antrasis - kaip simbolis, pažymėtas numeriu 97. Atspausdinkite visus 256 simbolius.
3. Parašykite programą, kuri atspausdintų skaičius nuo 1 iki 16 dešimtainėje, aštuntainėje, šešioliktainėje sistemose.
4. Parašykite programą, kuri paprašytų kokio skaičiaus norite sužinoti kvadratų sumą. Pvz.:4 – 1*1+2*2+3*3+4*4=30. Į ekraną bus išvesta 30.

Loginės operacijos

Raktiniai žodžiai

if , else, switch, break, case, default

Operacijos

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

Operatorius if

Pasiaiškinsime paprasčiausią 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 nurodo kompiuteriui didinti kintamojo lin vertę vienetu, jei iš klaviatūros paimtas simbolis yra nauja eilutė "\n". Jei iš klaviatūros paimtas simbolis neatitinka simbolio nauja eilutė, tai toliau vykdant operatorių **while** – imamas sekantis simbolis. Operatorius **getchar( )** yra skirtas vieno simbolio įvedimui iš klaviatūros. Vieno simbolio išvedimui naudojamas operatorius **putchar( )**.

EOF, tai "-1", kas atitinka bylos pabaigą. 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:

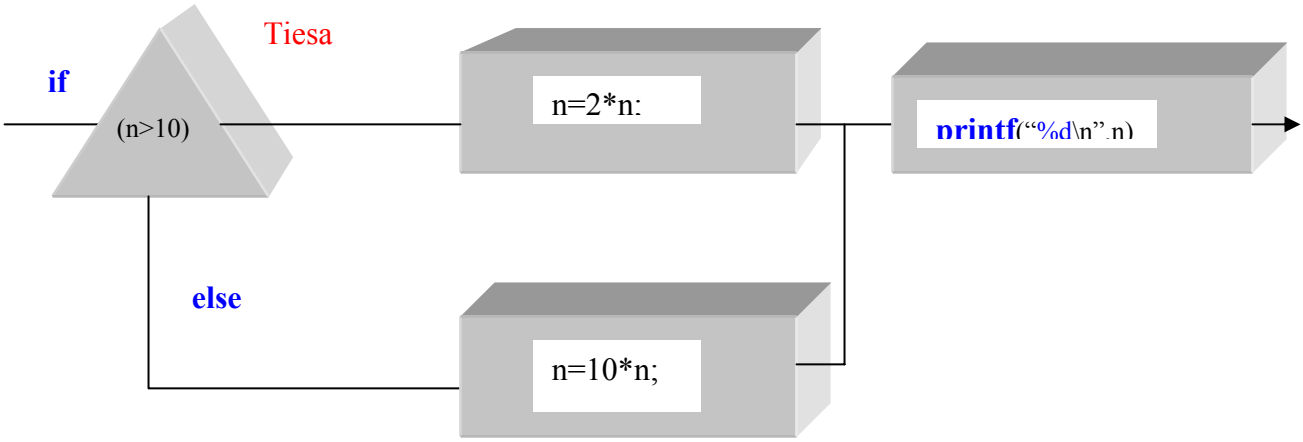
```
#include <stdio.h>


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



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

Sąlyginės operacijos

Kai kurias sąlygines komandas mes jau naudojome anksčiau, dabar susipažinsime su visomis galimomis sąlyginėmis operacijomis C kalboje.

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, sąlygines komandas negalima naudoti eilučių lyginimui. 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 paskaičiuoti, 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, || tai loginė komanda "arba". C kalboje yra trys loginės komandos:

Operacija	Reikšmė
&&	ir
||	arba
!	ne

Pažiūrėkime kaip jos vykdomos. Tarkime turime du sąlyginius 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 bus teisingos abi sąlygos.
3. !a teisinga, jei a sąlyga yra neteisinga.

Paimsime 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 skaičiaus absoliutinės vertės radimui:

```
x=(y<0)? –y : y;
```

tai atitiktų užrašymą:

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

Operatoriai switch ir break.

Jei reikia pasirinkti vieną galimą variantą iš eilės duomenų, tuomet nepatogu naudoti **if-else** konstrukciją. Daugeliu tokių atveju pravartus yra operatoriaus **switch** panaudojimas. Pateiksime programos pavyzdį, kuri nuskaito raidę ir išveda 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 norint nutraukti 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 **swicht** operatoriuje nurodytos raidės ir veiksmų, ką toliau daryti su duomenimis. **default** – bus naudojama visiems anksčiau nenurodytams atvejams. **break**, skirtas išėjimui iš **swicht** operatoriaus.

Užduotys

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

Ciklai ir kiti programos valdymo būdai

Raktiniai žodžiai

While, **do**, **for**, **break**, **continue**, **goto**

Operacijos

+= -= *= /= %=

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

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

Šio ciklo viduje turime nurodyti kaip turi keistis sąlygoje 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!*”.

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

/*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 operatorių, tuomet jie turi būti atskirti {} skliaustais. To nereikia, jei ciklui priklauso tik vienas operatorius.

for ciklo privalumai:

1. Sąlygos ir žingsniai užrašomi iš karto.
2. Galima žingsnį ne tik didinti (a++), bet ir mažinti (a--).
3. Galima ciklo žingsnį keisti bet kokiu nurodytu dydžiu: a+=13 (a=a+13).
4. Galima dirbti ne tik su skaičiais, bet ir su simboliais. Dirbant su 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. Galima palikti ciklo aprašymo vietas tuščias, svarbu, kad negalima praleisti kabliataškio.

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

šis ciklas bus vykdomas be galo ilgai, kadangi tuščia sąlyga yra visada teisinga.

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

bus vykdoma iki 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 cikle **while**, iš kurio išeinama tik šio operatoriaus dėka. Šis operatorius tikrai netinka su **if** operatoriumi. Jei taip atsitinka, kad jis reikalingas, tuomet būtina peržiūrėti parašytos programos algoritmą, kad nereikėtų naudoti šio operatoriaus.

Operatorius **continue** gali būti naudojamas visuose cikluose išskyrus **switch**. Šie operatoriai geriausiai praverčia, kai reikia sutrumpinti **if-else** sąlygos veikimą.

Operatorius **goto** – pats prasčiausias. Jo nedera naudoti C++ kalboje. Šio operatoriaus naudojimas – prasto programavimo požymis.

Masyvai

Jau anksčiau minėjome kas yra masyvas ir kaip jį reikia apibrėžti. Dar kartą grįšime prie jų, kadangi jie yra svarbūs programavimo procese. 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 ciklo pagalba. 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 atspausdinimas*/
…
```

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 priskirti i-tąjį masyvo elementą.
Svarbu: užpildymui nurodomas masyvo 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 išrūšiuokite didėjančia tvarka ir išveskite į ekraną.
3. Parašykite programą, kuri sudėtų dviejų masyvų elementus (masyvai sudaryti iš 10 elementų ir 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. Į ekraną turi būti išvesti pradiniai masyvai bei jų sumos, bei dalybos masyvai.

Kaip teisingai naudotis funkcijomis

Raktinis žodis

return
Programavimas C++ kalba paremtas funkcijų naudojimu. Mes jau naudojome funkcijas **printf( )**, **scanf( )**, **getch( )**, **putchar( )**, **strlen( )**. Šios funkcijos yra sisteminės, bet mes esame sukūrę ir savų funkcijų – **main( )**. Programos vykdymas visada prasideda komandomis, kurios yra **main( )** funkcijoje, kuri gali kreiptis ir į kitas funkcijas. Dabar išsiaiškinsime kaip patiems sukurti funkcijas, į kurias galėtų kreiptis **main( )** funkcija ir kitos sukurtos funkcijos.

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

Tarkime, norime parašyti programą, kuri: įvestų skaičių rinkinį, jį išrūšiuotų ir rastų vidutinę vertę. Minėtą programą galima užrašyti sekančiai:

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

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

        return 0;
}
```

Aišku, kad pagrindinė funkcija kreipiasi į funkcijas **readlist( )**, **sort( )** ir **average( )**, kurios atlieka joms nurodytus veiksmus ir pagrindinei funkcijai grąžina rezultatą. Naudojant pagalbines funkcijas, galima pagrindinį dėmesį skirti programos struktūrai negaištant laiko jų detalėms.

Ką reikia žinoti apie funkcijas? Aišku, kaip jas reikia aprašyti, 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 iškarto kreipiasi į **starbar( )** funkciją, kuri atspausdina simbolius, toliau **main( )** funkcija kreipiasi į sistemines funkcijas, kurios atspausdina tekstą ir galiausiai – vėl į **starbar( )**, kuri atspausdina simbolius. Kuriant pagalbinę funkciją, ji rašoma pagal tokias pačias taisykles kaip ir **main ( )** funkcija. Čia abi funkcijos buvo užrašytos į vieną *.cpp bylą. Paprastai, geriau rašyti atskiras funkcijas į atskiras bylas, kurias būtų galima panaudoti kitose programose.

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

Pasiaiškinsime žemiau pateiktą programą:

```
/*absoliutinės vertės*/
#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 gražina rezultatą pagrindinei funkcijai, kuri gautąjį rezultatą išveda į ekraną.

Globalieji (išoriniai) ir lokalieji (vietiniai) kintamieji

Paprastai, 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. Pažiūrėkime programą, kuri sukeičia kintamųjų vertes:

```
/*kintamuju 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 vertės\n", x, y);
        interchange(&x, &y);
```

```
        printf("sukeistos x= %d ir y= %d vertės \n", x, y);
}

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

        return *u, *v;

}
```

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

Globalieji kintamieji apibrėžiami prieš **main( )** funkciją 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 vertes, jas pakeičia jų suma ir skirtumu bei atspausdina jas.
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. Vaizdumo dėlei, pasiaiškinsime kaip parašyti funkciją faktorialui skaičiuoti. Faktorialą galima apskaičiuoti dviem būdais:

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

arba

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

Antrasis faktorialo užrašymo būdas, tai yra faktorialo lygtis. Pagal pirmąjį apibrėžimą faktorialo skaičiavimo programą Jūs jau rašėte. Dabar parašysime faktorialo skaičiavimo funkciją pagal antąjį apibrėžimą:

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

Kaip matome, funkcijos užrašymas sutrumpėja ir čia nėra jokio ciklo. Ši funkcija kreipsis pati į save tol, kol n bus lygus 0. Rekursinės funkcijos naudingos, kai reikia programuoti veiksmus su nežinomu operacijų skaičiumi.

Užduotis

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

Masyvai ir rodyklės

Jau žinome, kas yra masyvai ir kaip apibrėžti statinį (apibrėžto dydžio) masyvą. Apibrėžto masyvo elementams galima iš karto priskirti vertes:

```
#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, days[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 vertes:

```
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]);
}
```

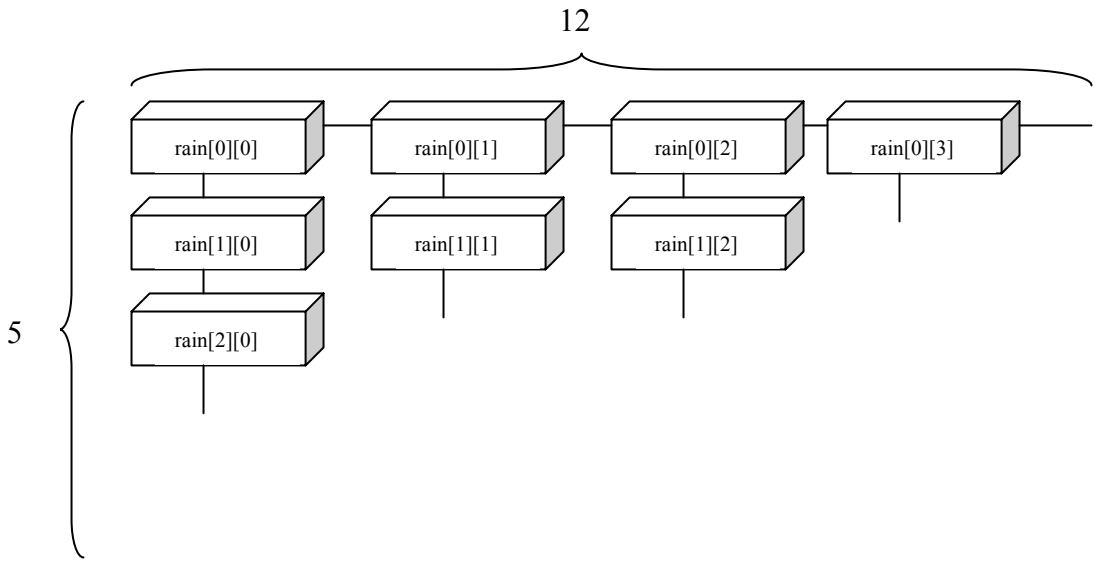
Šiuo atveju pats kompiliatorius apibrėžė masyvo dydį. Cikle **for** norint nurodyti masyvo dydį mes **sizeof( )** funkcijos pagalba suskaičiuojame masyvo dydį baitais. Kadangi masyvo elementai yra sveiko tipo kintamieji, kurie užima du baitus kompiurio atmintyje, tai norint gauti masyvo skaičių, mes padalinome masyvo 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 kiekvieno elementą sudaro 12 elementų. Vaizdžiai tokio tipo masyvą galima pavaizduoti taip:



Keičiant masyvo elementus elgiamasi taip pat kaip ir dirbant su vienmačiu masyvu. Norisi pabrėžti, kad C kalboje masyvai yra labai paprasti. Jie yra tokie prasti, kad yra žymiai geriau galvoti, kad jų nėra iš viso.

Jei masyvo dydis bus suskaičiuotas tik programos eigoje, tai masyvą reikia apibrėžti kaip rodyklę:

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

Aukščiau pateiktame pavyzdyje, pradžioje masyvas apibrėžtas kaip rodyklė, programos eigoje apskaičiuota, kiek reiks masyvo elementų zz. Tuomet, **malloc( )** funkcijos pagalba yra sukuriamas reikiamo dydžio masyvas. Primename, kad zz masyvo elementų skaičius, o **sizeof**(**float**) – nurodo **float** tipo kintamųjų užimamą vietą kompiuterio atmintyje baitais.

Įvesties ir išvesties bylos C kalboje

Dažnai teks rašyti programas, kurios duomenis ims iš bylos ir skaičiavimo rezultatus irgi išves į bylą. Pasiaiškinsime pradžioje paprastas bylos skaitymo funkcijas **fopen( )**, **fclose( )**, **getc( )**, **putc( )**.

Pažiūrėsime programą, kuri skaito bylos *test* turinį ir jį išveda į ekraną.

```
#include<stdio.h>

void main( )
{
    FILE *in;  /* aprašo rodyklę 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 į ekraną*/
    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( )

Mūsų pavyzdyje:
fclose(in);
Atkreipiame dėmesį, 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 funkcijoms **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 į bylą, į kurią siunčia rodyklė out tipo **FILE**.
Mūsų atveju buvo naudota stdout – tai rodyklė į standartinį išvedimą.

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

Kintamojo ir informacijos išvedimo skirtumas yra tik tas, kad išvedant informaciją į bylą 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 dirba panašiai kaip ir funkcijos **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 rodyklę 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 į data*/
fclose(fi);
        }
}
```

Panašiai yra ir su funkcija **fgets( )**, kuri nuo **gets( )** skiriasi papildomu kintamuoju:

```
/* nuskaito bylą 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 bus įvedama perskaityta iš bylos informacija (bus įrašoma į simbolinį masyvą).

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

Skirtumas tarp **gets( )** ir **fgets( )** – **gets( )** keičia naujos eilutės simbolį į ‘\0’, o **fgets( )** – išlaiko šį simbolį. Abi funkcijos sutikę bylos pabaigą EOF, grąžina vertę NULL.

fputs( ) daro panašius veiksmus į **puts( )** funkciją:

```
fputs(“Tu teisus”, fileptr);
```

perduoda eilutę “Tu teisus” į bylą, kurią nurodo rodyklė 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 sveiku 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, skirtą gautų knygų inventorizacijai. Įvedami duomenys turi būti įrašomi į atskirą bylą.
2. Parašykite programą, kuri nuskaitytų iš bylos duomenis: pradžioje turi būti sukurta byla, kurioje būtų nurodyti kokie nors skaičiai. Nuskaičiusi, atspausdintų bylos turinį į ekraną, patikrintų ar nėra nulių, jei yra, juos ištrintų ir naujus skaičius įrašytų į naują bylą.

Literatūra

1. М.Уэйт, С.Прата, Д.Мартин, Язык Си, Москва, Мир, 1988.
2. A. Matulis C, C++, ir OOP .
3. A. Vidžiūnas, C++ duomenų tipai ir struktūros, Kaunas, Smaltija, 2000.
4. A.Vidžiūnas, C++ ir C++ Builder pradmenys, Kaunas, Smaltija, 2002.
5. J.Blanskis ir kiti, C++ praktikumas. KTU, 2001.
6. J.Lipeikienė, Programavimas C++ kalba, VPU leidykla, 2002.

PRIEDAS

Raktiniai C kalbos žodžiai

Programos vykdymo raktiniai žodžiai:
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)
        operatorius1
else
        operatorius2
```

Jei sąlyga teisinga, vykdomas operatorius1, jei klaidinga – operatorius2.

3 būdas

```
if(sąlyga1)
        operatorius1
else if(sąlyga2)
        operatorius2
else
        operatorius3
```

Jei sąlyga1 teisinga, tai vykdomas operatorius1. Jei sąlyga1 klaidinga, o sąlyga2 teisinga, vykdomas operatorius2. Jei abi sąlygos klaidingos, vykdomas operatorius3.

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 1požymis: operatorius1
        case 2požymis: operatorius2
        default        : operatorius3
}
```

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.