**Tallinna Tehnikaülikool**

**Tarkvaratehnika õppetool**

**Teodor Luczkowski**

**Programmeerimiskeeled C ja C++**



**Sisukord**

[Sissejuhatus 4](#_Toc340488538)

[1. C/C++ programm 5](#_Toc340488539)

[1.1. Üldised keelereeglid 5](#_Toc340488540)

[1.2. Programmi struktuur 6](#_Toc340488541)

[1.3. Näiteprogramm 8](#_Toc340488542)

[2. Deklaratsioonid 14](#_Toc340488543)

[2.1. Muutujate ja konstantide deklareerimine 14](#_Toc340488544)

[2.2. Massiivid ja viitmuutujad 19](#_Toc340488545)

[3. Avaldised 25](#_Toc340488546)

[4. Keelestruktuurid. Juhtlaused 31](#_Toc340488547)

[4.1. Valikud 31](#_Toc340488548)

[4.2. Kordused 33](#_Toc340488549)

[4.3. Juhtlaused 38](#_Toc340488550)

[5. Funktsioonid 42](#_Toc340488551)

[5.1. Funktsiooni deklareerimine ja kirjeldamine 42](#_Toc340488552)

[5.2. Programmi parameetrid 48](#_Toc340488553)

[6. Failide kasutamine 49](#_Toc340488554)

[6.1. C standardfunktsioonide teek stdio.h 50](#_Toc340488555)

[6.2. Klass fstream 56](#_Toc340488556)

[7. Viidatehnika 58](#_Toc340488557)

[7.1. Viida väärtus 58](#_Toc340488558)

[7.2. Mitmesed viidad 61](#_Toc340488559)

[8. Tekstitöötlus 64](#_Toc340488560)

[8.1. Tekstistringid programmis 64](#_Toc340488561)

[8.2. Standardfunktsioonid tekstistringidega 67](#_Toc340488562)

[8.3. Teisendusfunktsioonid 72](#_Toc340488563)

[8.4. Märgifunktsioonid 74](#_Toc340488564)

[8.5. Klass string 76](#_Toc340488565)

[9. Mälu kasutamine 78](#_Toc340488566)

[9.1. Muutuja skoop (kasutuspiirkond) 78](#_Toc340488567)

[9.2. Staatilised muutujad 80](#_Toc340488568)

[9.3. Mälu dünaamiline eraldamine ja vabastamine 81](#_Toc340488569)

[9.4. Andmekooslused 85](#_Toc340488570)

[10. Standardfunktsioonide teegid 90](#_Toc340488571)

[10.1. Ajafunktsioonid (time.h) 91](#_Toc340488572)

[10.2. Üldfunktsioonid (stdlib.h) 98](#_Toc340488573)

[11. Eelprotsessor 102](#_Toc340488574)

[12. Veatöötlus 107](#_Toc340488575)

[13. Objektiklassid 112](#_Toc340488576)

[13.1 Objektiklassi deklareerimine 112](#_Toc340488577)

[13.2.Tehete üledefineerimine 116](#_Toc340488578)

[13.2. Uute klasside tuletamine 118](#_Toc340488579)

[13.3. Staatilised klassiliikmed 122](#_Toc340488580)

[13.4 Abstraktne objektiklass. 126](#_Toc340488581)

[Lisad 130](#_Toc340488582)

[Lisa 1. Tehted ja nende täitmise järjekord prioriteetide tabelina 130](#_Toc340488583)

[Lisa 2. Vorminguelemendid 132](#_Toc340488584)

[Lisa 3. Funktsioon ‘kuva’ failis tapid.h 134](#_Toc340488585)

# Sissejuhatus

Programmeerimiskeel C ja tema objektorienteeritud edasi­arendus C++ on populaarsed programmeerijate seas. Pea­põhjuseks on asjaolu, et keele loojad Brian Kerninghan ja Dennis Ritchie olid ise rahutu loomuga programmeerijad, kellele olemas­olevad keeled ühel või teisel põhjusel ei meeldinud. Nii lõid nad oma nägemusele ja soovidele vastava programmeerimiskeele. C/C++ populaarsust kinnitab asjaolu, et ta baasil on loodud mitmeid teisi programmee­rimiskeeli.

Tõsiselt informaatikult eeldatakse vähemalt kahe-kolme programmee­rimiskeele valdamist. Üheks neist sobib kindlasti C++ ja/või temaga paljuski sarnane Java. Antud materjal on mõeldud abiks C/C++ program­meerimiskursust läbivale tudengile. Lugejalt eelda­takse elementaarteadmisi program­meerimise alustest.

C keele loomisel oli üheks eesmärgiks universaalsus suva­lises programmeerimiskeskkonnas. C++ suurim erinevus C-st on objektorienteeritud (OO) programmeerimis­tehnoloogia toetamine. Saab kirjeldada objektide klasse, millesse kuuluvad objektid omavad ühesuguseid andmeid ja oskavad täita samu tegevusi, OO mõistes meetodeid. Kuigi objektiklasside loomist on kirjeldatud päris lõpus, 13. jaotises, on mõningaid C++ standardklasse kasutatud juba esimestes näidetes. Näiteks andmeedastus programmi ja tema kasutaja vahel on objekte kasutades märgatavalt lihtsam ja seetõttu kasutusel enne OO teemani jõudmist.

C/C++ võtmesõnad on muust tekstist eristamiseks esitatud kirjas Courier,ingliskeelsed terminid *kaldkirjas*, prog­rammi­tekstid kirjastiilis Arial Narrow ja ka programmide väljund on samas kirjas või näidatud ekraanitõmmisena.

 “**NB!**”-ga on tähistatud C/C++ keelereeglid ja tehete tähistused, mis erinevad tava­pärasest või teistes program­meeri­miskeeltes kasutusel olevatest ja millega eksitakse seetõttu kõige sagedamini.

Materjalis toodud programminäited ja -lõigud on koos­tatud ja testitud Code::Blocks 10.05 keskkonnas. Programmid on loodud konsoolirakendusena.

# 1. C/C++ programm

Esimeses jaotises antakse kiire ülevaade sellest, mida on vaja teada C/C++ programmitekstist arusaamiseks. Näiteprogramm jao­tises 1.3 on valitud võimalikult lihtne, kuid pisut keerulisem kui ainult maailma tervitava teksti kuvamine.

1.1. Üldised keelereeglid

Suur- ja väiketähti eristatakse. C/C++ programm on enamasti väike­tähtedes, suurtähti kasutatakse reeglina vaid konstantide nimedes.

Programmi kujundus on suhteliselt vaba. Nii on lubatud tühikud, tabulatsioonid, reavahetused iga eraldaja ees ja järel. Heaks tooniks on programmi tekstis taandridade kasu­tamine ehk ”treppimine”, sest muidu on raske määrata programmi­plokkide algust ja lõppu. Paljudes program­meerimis­keskkondades on selline ”treppimine” võimalik tellida automaatsena (*Auto indent*).

Kommentaarid on kas märkide /\* ja \*/ vahel või pärast kahte kaldkriipsu (//). Esimesel juhul võib kommentaar olla mitmel real, kahe kaldkriipsu puhul on kommentaar antud rea lõpuni. Kui rida algab kaldkriipsudega, on kogu rida mõeldud programmi teksti kommenteerimiseks.

Iga kirjeldav ja täidetav lause lõppeb semikooloniga. Look­sulgudesse paigutatud laused moodustavad lausete ploki ehk liitlause.

1.2. Programmi struktuur

Lihtne C/C++ programm koosneb järgmistest osadest:

1. Eelprotsessori korraldused #include.

#include <failinimi> lisab programmi tekstile standard­funktsioonide teegi (sõnast biblioteek, *library*) sisu, milles on funktsioonide kirjeldused (prototüübid). Kuna C/C++ ei sisalda sisend-väljundlauseid, siis tuleb andmete edastamiseks kasutada standard­funktsioone. Seetõttu on vähemalt ühe sisend-väljund­funktsioonide kirjeldusi sisaldava teegi lisamine vajalik praktiliselt igale programmile.

2. Peaprogramm int main(void)

C/C++ peaprogramm on funktsioon nimega ‘main’. Selle funktsiooni väärtuseks on tavaliselt täisarvuna (int) esitatav veakood. Kui veakood on 0, siis programmi töö lõppes normaalselt. Võtmesõna void näitab parameetrite puudumist, vajadusel saab peaprogramm neid aga kasutada.

3. Programmi (funktsiooni) tekst looksulgude ‘{‘ ja ’}’ vahel. Tekstis paiknevad:

3.1. Muutujate deklaratsioonid.

3.2. Täidetavad laused, mis moodustavad programmi algo­ritmi.

3.3. Programmi lõpulause return 0. Veakoodi 0 üldlevinud tähendus on programmi täitmise vigadeta lõpetamine.

CodeBlocks loob uue C++ konsoolrakenduse projekti avamisel faili, milles sisalduvad juba vajalikud käsud elemen­taarse programmi käivitamiseks (tühjad read on eemaldatud):

#include <iostream>

using namespace std;

int main()

{

 cout << "Hello world!" << endl;

 return 0;

}

Sellise programmi käivitamisel kuvatakse tulemus konsooli­aknas:



Loodud aken jääb avatuks kuni suvalise klahvi vajuta­miseni. Mitte kõik C/C++ programmeerimiskeskkonnad pole nii vastutulelikud, et jätavad aega programmi töö tulemustega tutvumiseks – nad sulgevad konsooliakna esimesel võimalusel ehk viimase programmikäsu täitmise järel. Konsooliaken sulgub kohe ka siis, kui CodeBocksi projekti käivitamisel \bin\Debug-nimelisse sisukorda salvestatud exe-fail käivitada väljaspool program­meerimis­keskkonda, näiteks Windows Exploreris. Alljärgnev programm jätab seevastu konsooliakna igal juhul avatuks kuni suvalise klahvivajutuseni.

**Näiteprogramm 1.1. Mall.cpp.** Programm algab nii C++ kui C sisend-väljundfunktsioonide teekide lisava eel­protsessori käsuga #include. Üks fail on #include-käsuga lisatud veel, milleks on C mittestandardne fail ”tapid.h”. Selle faili sisuks on funktsioon ”kuva”, mille tekst asub lisas 3. Vajalik on see funktsioon selleks, et muuta enne kuvamist täpitähtede koode, mis on erinevad tava- ja konsooliaknas. Eelprotsessori käskudele järgnevad C/C++ programmi algus- ja lõpulaused.

Programm ‘Mall’ väljastab teksti "Vajuta mõnele suvalisele klahvile..." ja ootab luba programmi töö lõpetamiseks. Sellisena sobib ta kõigi lihtsamate programmide aluseks (malliks):

#include <iostream> // C++ sisend-väljund

#include <stdio.h> // C standardne sisend-väljund

#include <conio.h> // C konsoolifunktsioonid

#include <tapid.h> // funktsioon kuva muudab täpitähtede koode

using namespace std;

int main()

{

 // programmi algoritm

 printf(kuva("Vajuta mõnele suvalisele klahvile..."));

 getch();

 return 0;

}

Järgnev aken ei sulgu enne lubavat klahvivajutust ka siis, kui ta programm käivitatakse mitte programmeerimiskeskkonnast, vaid tavalise exe-failina (Mall.exe):



## 1.3. Näiteprogramm

Alljärgnevas programmis on olemas kõik põhikomponendid – andmete küsimine, tulemuste väljastamine, kordus ja valik. Järgmistes jaotistes käsitletakse neid teemasid põhja­likumalt.

**Näiteprogramm 1.2. SadaAastat.cpp**. Programm küsib nime ja vanust ning väljastab nende andmete põhjal isiku vanuse iga järgneva aasta kohta kuni saja aastaseks saamiseni. Nulliga lõppevate sünnipäevade puhul lisatakse teade juubeli kohta.

#include <iostream>

using namespace std;

int main(void)

{

 int vanus; // vanus täisaastates

 int i; // muutuja järgmiste aastate arvutamiseks

 const int aasta=2013; // käesolev aasta

 char nimi[60]; // märgimassiiv nime tähtede jaoks

 cout << "Mis Su nimi on > ";

 cin >> nimi;

 cout << "Tere, " << nimi << "!" << endl;

 cout << "Kui vana sa oled > ";

 cin >> vanus;

 cout << "Järgmisel aastal saad juba " << ++vanus

 << "-aastaseks. " << endl;

 for(i=aasta+2;vanus<100;i++)

 {cout << i << ". aastal saad " << ++vanus

 << "-aastaseks. " <<endl;

 if(!(vanus%50))cout << "SUUR JUUBEL!" << endl;

 else if(!(vanus%10))cout << "Juubel!" << endl;}

 return 0;

}

Programmi töö tulemus on selline, kui nimeks sisestakse Bruno ja vanuseks 18:

Mis Su nimi on > Bruno

Tere, Bruno!

Kui vana sa oled > 18

Järgmisel aastal saad juba 19-aastaseks.

2015. aastal saad 20-aastaseks.

Juubel!

2016. aastal saad 21-aastaseks.

2017. aastal saad 22-aastaseks.

...

2094. aastal saad 99-aastaseks.

2095. aastal saad 100-aastaseks.

SUUR JUUBEL!

Järgnevad selgitused näiteprogrammile on sissejuhatuseks keelereegleid käsitlevatele jaotistele.

**Muutujate deklaratsioonid**

Kõik programmis kasutatavad muutujad tuleb deklareerida. Muutuja deklaratsioonis näidatakse andmetüüp ning muutuja nimi. Põhilised andmetüübid on täisarv (int, long), murdarv (float, double) ja märk (char). Sama­tüübi­liste muutujate nimed võib paigutada samasse lausesse. Nimede koosseisus on tavaliselt väiketähed ja numbrid; nimedes on lubatud ka suurtähed ja mõned erimärgid, nagu näiteks allkriips (\_).

Muutujatega sarnaselt deklareeritakse konstandid, mille väärtust programmis muuta ei saa. Tunnuseks on võtmesõna const deklaratsiooni ees.

Muutuja või konstandi algväärtustamisel näidatakse algväärtus deklareerimisel võrdusmärgi järel. **NB!** Kui algväärtust pole antud, sisaldab muutuja mäluväli juhuslikku väärtust. Konstanti ei saa jätta algväärtuseta.

Massiiv on samatüübiliste muutujate kogum, deklareerimisel näidatakse nurksulgudes ära elementide arv massiivis.

int vanus; // vanus täisaastates

int i; // muutuja järgmiste aastate arvutamiseks

const int aasta=2013; // käesolev aasta

char nimi[60]; // märgimassiiv nime tähtede jaoks

Deklareeritud on täisarvulised muutujad ‘vanus’ ja ‘i’, käesolev aasta on kirjeldatud konstandina ja nime jaoks on deklareeritud piisavalt pikk märgimassiiv ’nimi’. Käesolevat aastat pole tegelikult hea konstandina programmi teksti kirjutada, aastanumbrit tuleks küsida arvuti kellalt. Kuidas seda teha, on kirjas jaotises 10.1.

**Sisend-väljund**

Kui on vaja väljastada mõni küsimus või muu tekst või sisestada vastus, on seda kõige hõlpsam teha C++ objektide cin ja cout ning tehete << ja >> abil:

cout << "Kui vana sa oled > ";

cin >> vanus;

Väljastuseks saab kasutada ka C funktsiooni printf, nagu näites Mall.cpp. Tema väljakutsumine on keerulisem, kuid samas saab kasutada mitmeid vormindamisvõimalusi (täpsemalt lisas 2). Näiteks seitsmeaastase isiku puhul kuvab lause:

cout << i << ". aastal saad " << ++vanus

 << "-aastaseks. " <<endl;

arvutuste tulemused nii:

2015. aastal saad 9-aastaseks.

2016. aastal saad 10-aastaseks.

siis printf-funktsioonis saab näidata tekstis kohad, kuhu ja mis kujul andmed teksti sisse paigutada. Saab moodustada joondatud veergudega väljundit ja määrata väljundväljade suurused. Näiteks lause:

printf("%d. aastal saab %s %3d-aastaseks.\n", i, nimi, ++vanus);

väljastab samad andmed joondatuna:



Reavahetust võib väljastamisel nõuda kas nimega ’endl’ või märgikonstandiga ’\n’.

Andmeid saab sisestada lisaks objekti ’cin’ kasutamisele ka standardfunktsiooniga ’scanf’. Nagu funktsiooni ’printf’ puhul, tuleb näidata kõigepealt vorming, seejärel muutuja aadress, kuhu andmed sisestatakse. Käsu

cin >> vanus;

asemel võiks sama hästi olla:

scanf(”%d”, &vanus);

**Avaldised**

Avaldisi kasutatakse enamasti omistuslauses, mille üldkuju on:

muutuja=avaldis;

Lisaks tavalistele aritmeetikatehetele + - \* / on aval­dises kasutatavate tehete seas olemas veel:

* jääk %
* täisarvu suurendamine ühe võrra ++
* täisarvu vähendamine ühe võrra --
* väärtuse muutmine vastava aritmeetikatehtega += -= \*= /=

Avaldis

++vanus

tähendab muutuja ’vanus’ suurendamist ühe võrra enne tema väärtuse väljastamist.

**Korduvad tegevused (tsükkel, silmus)**

Korduvaid tegevusi on võimalik kirja panna kolmel erineval moel, erinevad nad ka kordust alustava võtmesõna poolest. Kui korduste arv on programmi vastavas kohas teada, siis kasutatakse enamasti tsükli­muutujaga kordust:

for(algusavaldis; kordustingimus; lõpuavaldis) tegevus;

Kõigepealt täidetakse algusavaldis (tavaliselt tsükli­muutuja alg­väärtusta­mine), siis kontrollitakse kordus­tingimust. Kui vastav avaldis on tõene (väärtus pole 0), täidetakse tegevus ja seejärel lõpuavaldis. Jätkatakse seni, kuni tingimuse väärtus muutub nulliks.

 for(i=aasta+2;vanus<100;i++)

Tsüklimuutuja ‘i’ algväärtuseks saab konstandi ’aasta’ väärtusest kahe võrra suurem täisarv, kordustingimuseks on tema võrdlemine sajaga. Iga korduse sammu lõpus suurendatakse tsüklimuutuja väärtust ühe võrra. Korduvalt täidetav tegevus koosneb sageli mitmest lausest ja on seetõttu vormistatud look­sulgudes oleva liitlausena.

**Valikud**

Kõige lihtsam valik on vaid ühe võtmesõnaga if. **NB!** võtmesõna ‘then’ C-s puudub.

if(tingimus)tegevus;

Kui tingimus on tõene (pole 0), täidetakse tegevust määrav (liit)lause.

if(!(vanus%50))cout << "SUUR JUUBEL!" << endl;

Vaid siis, kui vanuse 50-ga jagamise jääk on null, väljasta­takse if-lauses oleva tekst "SUUR JUUBEL!" . Loogilist eitust märgitakse C-keeles hüüumärgiga (!).

# 2. Deklaratsioonid

Deklaratsioonidega näidatakse, milliseid andmeid programmis kasuta­takse. Muutujate deklareerimine on C/C++ keeles kohus­tuslik.

## 2.1. Muutujate ja konstantide deklareerimine

Muutuja deklaratsioon koosneb andmetüübi võtmesõnast ja muutuja(te) nime(de)st. Samas lauses saab anda muutujatele ka algväärtuse, näiteks nii:

int limiit=10;

Ühe deklaratsioonilausega võib kirjeldada mitut sama tüüpi muutujat, nende nimede eraldajaks on koma.

Tähtsamateks andmetüüpideks on täisarv, murdarv ja märk. Kasutusel on küll erinevad võtmesõnad, kuid nad vaid täpsustavad eraldatava mälu soovitavat suurust. Kuna C/C++ programmid on mõeldud töötama mitmes keskkonnas, pole keeles määratud täpset mälumahtu deklareeritud muutujatele.

Täisarvu andmetüübid on int ja long. Suure tõe­näosusega on mõlemad 4-baidised, kuid selles kindel olla ei tohi. Tehe sizeof annab teada, kui palju mälu muutujal tegelikult on. Võtmesõna unsigned muutuja deklaratsioonis tähen­dab, et esimest bitti ei loeta märgiks ja muutuja väärtust käsitletakse alati positiivsena.

Nagu järgneva programmi väljund näitab, peab program­meerija ise vastutama muutujate õige väärtustamise ja kasuta­mise eest:

int main()

{

 int tavaline;

 const int suurim=0x7FFFFFFF;

 unsigned int margita;

 long pikk;

 unsigned long margitapikk;

 printf("int on %d baiti, long %d baiti\n",

 sizeof(int), sizeof(long));

 tavaline=-1;

 margita=tavaline;

 printf("tavaline int = %d, märgita on ta %u\n",

 tavaline, margita);

 printf("Suurim märgiga täisarv on %d.\n", suurim);

 pikk=2500000000; // see arv on liiga suur

 margitapikk=pikk;

 printf("pikk= %ld, märgita on ta %u\n", pikk, margitapikk);

}

väljastatakse:

int on 4 baiti, long 4 baiti

tavaline int = -1, märgita on ta 4294967295

Suurim märgiga täisarv on 2147483647.

pikk= -1794967296, märgita on ta 2500000000

Murdarvud (ujupunkt-, reaalarvud) kirjeldatakse kui float- või double-tüüpi muutujaid, erinevad nad täpsuse ehk tüvekohtade arvu poolest. Kui täisarvu puhul tuleb arvestada, et tema suurim väärtus on pisut üle kahe miljardi ehk vähem kui elanike arv maailmas või Eesti riigi eelarve, siis murdarvudega seda probleemi pole. Lühema murdarvu puhul võib väärtus olla vahemikus 10 astmes +/- 38, pikema korral +/-308. Küll aga võib esimese puhul usaldada vaid esimest seitset numbrit, teise puhul ülimalt viitteist.

**NB!** Ujupunktarvude ebatäpsuse tõttu võib nende väärtuste võrdlemine anda soovimatu tulemuse. Vaid nulliga võrd­lemine on üldiselt ohutu, teiste väärtuste puhul pole täpne võrdlemine hea mõte. Näiteks selline programm:

 const float algvaartus=0.0;

 float kolmandik;

 const float vordlus1=0.333333;

 const float vordlus2=0.33333333;

 kolmandik=1.0/3.0;

 if(algvaartus)cout << "Pole null. "; else cout << "On null. ";

 if(kolmandik==vordlus1)cout << "On sama väärtus. ";

 else cout << "Pole sama väärtus. ";

 if(kolmandik==vordlus2) cout << "On sama väärtus. ";

else cout << "Pole sama väärtus. ";

väljastab:

On null. Pole sama väärtus. On sama väärtus.

Teine, negatiivne teade on põhjustatud konstandi ’vordlus1’ ebatäpsusest. Kuuest tüvekohast ei piisa, kaheksast aga küll, nagu konstandi ´vordlus2’ puhul.

**Näiteprogramm 2.1. KolmHinnet.cpp.** Programm küsib kolme hinnet ja väljastab nende keskmise. Kesk­mise leidmine on tehtud eri täpsusega murdarvudega, väljund peegeldab seda erinevust:

#include <stdio.h> // standardne sisend-väljund

#include <iostream> // C++ sisend-väljund

using namespace std;

int main(void)

{

 int hinne, i;

 float hinded=0, keskmine;

 double keskminetapne;

 int n=3;

 for(i=1;i<=n;i++){

 printf("Sisesta %d. hinne > ", i);

 cin >> hinne;

 hinded+=hinne;

 }

 keskmine=hinded/n;

 printf("Keskmine hinne on %15.12f\n", keskmine);

 keskminetapne=hinded/n;

 printf("Täpsemalt on see hinne %15.12lf\n", keskminetapne);

 return 0;

}

Hinnete 4, 3 ja 4 puhul on programmi väljundiks:

Sisesta 1. hinne >4

Sisesta 2. hinne >3

Sisesta 3. hinne >4

Keskmine hinne on 3.666666746140

Täpsemalt on see hinne 3.666666666667

Ühte märki sisaldava muutuja võtmesõnaks on char ja ta sisaldab selle märgi koodi. Kuna märgikoodi käsitletakse nagu täisarvu, on C/C++ programmides tavalised aritmeetika­tehted märgikoodidega. Märk võib olla tsüklimuutuja ja osaleda aval­distes, nagu järgmises programmis:

int main()

{

 char mark;

 int tahti=0;

 for(mark='A'; mark<='Z'; mark++,tahti++)printf("%c", mark);

 printf("\nTähestikus (A-Z) on %d tähte\n", tahti);

 mark='F';

 printf("F on tähestikus %d. täht\n", mark-'A'+1);

}

Väljund:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Tähestikus (A-Z) on 26 tähte

F on tähestikus 6. täht

Märgi algväärtustamisel näidatakse märk apostroofide (**NB!** mitte jutumärkide) vahel. Erimärgi ees on tagurpidi kaldkriips, vajalikud erimärgid on:

\n – reavahetus;

\” – jutumärk;

\t – tabulatsioon;

\0 – nullbait, kus kõik bitid on nullid. Nullbait lõpe­tab teksti.

Konstandi deklaratsioonis on võtmesõna const, tema algväärtus­tamine on kohustuslik. Seda väärtust programmis enam edaspidi muuta ei saa. Algväärtuse esitamisel ja ka mujal programmis saab soovi korral kasutada kaheksand- ja 16ndsüsteemi. Esimesel juhul on väärtuse ees tavaline null (0), teisel juhul lisaks x-täht (0x). Murdarve saab esitada komakohaga (punkt programmis) või e-esituses 10 astmega:

int main()

{

 const double mlrd=1e9;

 const int rgb=0xFF;

 const int kaheksa=010;

 printf("Konstantide väärtused on %14.2lf; %d ja %d.\n",

 mlrd, rgb, kaheksa);

 return 0;

}

Väljund:

Konstantide väärtused on 1000000000.00; 255 ja 8.

## 2.2. Massiivid ja viitmuutujad

Massiiv on samatüübiliste andmeelementide hulk. Andme­elemendi asu­koha massiivis määrab indeks, esimese indeksi väärtus on alati 0. Ühemõõtmelise massiivi dekla­reerimisel näidatakse andmetüüp, massiivi nimi ja nurk­sulgudes andmeelementide arv. **NB!** Suurim indeks on elementide arvust ühe võrra väiksem. Näiteks järgmine deklaratsioon kirjeldab üheksat täisarvust koosnevat massiivi ‘eur’:

int eur[9];

Kirjeldatud massiivis on elemendid eur[0]...eur[8]. Massiivi elemendi poole pöördumisel näidatakse tema indeks nurksulgudes nagu deklaratsioonilauses.

Massiivi elemente on lubatud deklareerimisel alg­väärtustada, omista­tavad väärtused paiknevad loendina looksulgudes, väärtusi eraldavad komad:

int eur[9]={1,2,5,10,20,50,100,200,500};

Kui väärtustatakse kõik massiivi elemendid, võib nende kokku­lugemise jätta translaatori hooleks ja elementide arvu deklarat­sioonis mitte näidata.

Massiivi pikkuse saab teada tehtega sizeof. Nii väljastab järg­mine programmilõik massiivi kogupikkuse, elementide arvu ja nende väärtused:

 int eur[]={1,2,5,10,20,50,100,200,500};

 int n;

 printf("Massiivi pikkus %d baiti, elemente: %d\n",

 sizeof(eur), n=sizeof(eur)/sizeof(int));

 printf("Massiivi elemetide väärtused:\n");

 for(int i=0;i<=n;i++)printf("%d ", eur[i]);

Meelega on ülaltoodud programmis tehtud C progammides levinud viga. Tsükli­muutuja lõppväärtus on näites võrdne elemen­tide arvuga massiivis, ta peab jääma aga ühe võrra väiksemaks. Mingit veateadet sellega tavaliselt ei kaasne, lihtsalt väljastatakse massiivile järgneva mäluvälja juhuslik väärtus:

Massiivi pikkus 36 baiti, elemente: 9

Massiivi elemetide väärtused:

1 2 5 10 20 50 100 200 500 28

Kõigi massiivielementide poole pöörduva kordus­spetsifi­katsioon õige kuju on:

for(int i=0;i<n;i++)...;

**Näiteprogramm 2.2. Eurod.cpp.** Programm küsib raha­summat täiseurodes ja väljastab, milliste rahasedelite ja -müntide abil saab sisestatud summat tasuda. Kasutatud on eespool kirjeldatud massiivi, mida tsüklis läbitakse tagurpidi, st. alustatakse viimasest elemendist ehk suurimast rahaühikust ja jõutakse lõpuks esimese elemendini ehk üheeuroseni. Täisarvude jagamise ja jäägitehte abil vähendatakse sisestatud summat, kuni ta muutub nulliks.

#include <stdio.h> // standardne sisend-väljund

int main(void)

{

 int i, n, summa;

 int eur[]={1,2,5,10,20,50,100,200,500};

 printf("Sisesta summa eurodes > ");

 n=sizeof(eur)/sizeof(int);

 scanf("%d", &summa);

 printf ("%d euro maksmiseks tuleb anda:\n", summa);

 for(i=n-1;i>=0;i--){

 if(summa>=eur[i]){

 printf("%d %d-euro%s\n", summa/eur[i], eur[i],

 i>1?"st":"st mynti");

 summa%=eur[i];

 }

 }

 return 0;

}

Väljund 3229-eurose summa puhul:



Tekstistring on märgimassiiv, massiivi viimane element on 0-bait (‘\0’). **NB!** Stringi pikkus on märkide arvust ühe, lõpu­märgi võrra suurem. Seda tuleb stringi deklareerimisel arves­tada. Järgnevalt on deklareeritud kaks stringi märgi­massiivina. Esimene deklaratsioon on selles mõttes vale, et lõpumärk on algväärtustamisel näitamata jäänud:

char riik1[]={'E', 'S', 'T'};

char riik2[4];

lausete:

printf("Sisesta vastane > ");

for(int i=0; i<3; i++)scanf("%c", &riik2[i]);

riik2[3]='\0';

printf("Täna kohtuvad: %s - %s.\n", riik1, riik2);

täitmise tulemusena väljastatakse sisestuse ‘ESP’ puhul midagi taolist:

Sisesta vastane > ESP

Täna kohtuvad: ESTw- ESP.

Kuna esimese stringi ‘riik1’ lõputunnus puudub, väljastatakse vormingu­elementi ‘%s’ kasutades märke kuni esimese ettesattuva nullbaidini. Tulemus on ettearvamatu, ta ei pruugi olla selline nagu ülaltoodud väljundis. Veelgi hul­lem, ta võib paljudel kordadel näida õigena ja seetõttu ei ole viga kergesti leitav. Tegemist on ühe tüüpili­sema vigade klassi “Ei saa olla, mul ta töötab!” kuuluva eksimusega C-programmides. Stringi korrektne lõpetamine tagatakse kas omistuslausega, nagu

riik2[3]='\0';

või korrektselt lõpetatud stringi algväärtustamisega märgi­massiivi deklaratsioonis:

char riik1[]={'E', 'S', 'T', '\0'};

Tekstistringi algväärtustamine on palju lihtsam, kui kasutada stringi­konstanti jutumärkides, mida reeglina ka kasutatakse. Lisaks kirjutus­mugavusele tagatakse nii stringi lõputunnuse automaatne lisamine:

char riik1[]="EST";

Mitmemõõtmelised massiivid on massiivide massiivid, nende deklareerimisel kasutatakse iga mõõtme jaoks eraldi nurksulge:

andmetüüp massiivi\_nimi[elementide\_arv1][elementide\_arv2]...;

C-s on kõik tekstistringide massiivid kahemõõtmelised, sest string ise on märgimassiiv. Massiivi elementide algväär­tustamisel tuleb seda asjaolu looksulgude kirjutamisel arvestada. Järgmine deklaratsioon kirjeldab kahemõõtmelist märgimassiivi, mis koosneb kolmest nelja­märgilisest ühe­mõõt­melisest massiivist ja algväärtustab kõik massiivi ‘baltiriik’ kuuluvad 12 märki:

char baltiriik[3][4]={

{'E', 'S', 'T', '\0'},

{'L', 'A', 'T', '\0'},

{ 'L', 'T', 'U', '\0'},

};

Lause

for(int i=0; i<3; i++)printf("%s\n", baltiriik[i]);

väljastab:

EST

LAT

LTU

Viitmuutuja sisaldab mäluvälja aadressi. Enamasti vastab see aadress mõne teise, viidatava muutuja mäluvälja aadressile. Viit­muutuja deklareerimine toimub viidatavale andetüübile vastava võtme­sõnaga, viitmuutuja ees on tärn (\*), muutuja aadressi ees see­vastu ‘&’-märk. Järgnevalt on deklareeritud viitmuutuja ‘p’, millele omistatakse täis­arvulise muutuja ‘krediidilimiit’ aadress. Viidatava väärtuse poole pöördumisel on viitmuutuja ees tärn nagu deklaratsiooniski:

int krediidilimiit=1000;

int \*p;

p=&krediidilimiit;

printf("Krediidilimiit oli: %d eurot.\n", \*p);

(\*p)\*=2; // viidatavat väärtust suurendatakse kaks korda

printf("Krediidilimiit on nüüd: %d eurot.\n", \*p);

Väljastatakse:

Krediidilimiit oli: 1000 eurot.

Krediidilimiit on nüüd: 2000 eurot.

Viitmuutujaid kasutatakse kõige sagedamini koos mas­sii­vi­dega, stringimassiivide korral on viitmuutujate kasutamine lisaks massiivi indeksitele või nende asemel valdav. Seda teemat on põjalikumalt käsitletud 7. jaotises ”Viidatehnika”.

# 3. Avaldised

Tavaline avaldise kasutuskoht on omistuslause paremal pool, kuid C/C++ lubab suvalise keerukusega avaldisi kasutada pea kõik­võimalikes kohtades programmi tekstis, ka iseseisva lausena. Seal­juures loetakse iga avaldise nullist erinevat väärtust tõeseks, kui avaldist saab käsitleda tin­gimusena. Avaldistes saab kasutada järgmisi tehteid, mis on alljärgnevalt jaotatud klassidesse:

**1. Aritmeetikatehted.** Tavatehted on liitmine, lahutamine, korru­tamine ja jagamine ning ka nende tähistus on tavaline ( + - \* /). **NB!** Täisarvude jagamisel tekkiva jäägi saab arvutada selle jaoks veidi kummalisena tunduva tehtemärgiga ’%’. C-le on omased suurendamise (++) ja vähendamise tehe (--), mis tähendab muutuja suurendamist või vähendamist ühe võrra. Sealjuures on oluline, kas tehtemärk on muutuja ees või järel. Esimesel juhul osaleb avaldise arvutamisel muutuja alg-, teisel juhul lõppväärtus. Seda on lihtsam selgitada näite varal:

int arv1=1000;

int arv2=25;

int tulemus;

int tehe=0;

tulemus=arv1++; // 1. tehe

printf("%d. tehte tulemus on %d, arv1=%d\n",

++tehe, tulemus, arv1);

tulemus=--arv2; // 2. tehe

printf("%d. tehte tulemus on %d, arv2=%d\n",

 ++tehe, tulemus, arv2);

Esimese tehte puhul väljastatakse kõigepealt muutuja ‘arv1’ alg­väärtus, seejärel tema väärtust suurendatakse. Muutuja ‘arv2’ väärtuse vähendamist tehakse vastupidises järjestuses. Muutuja ‘tehe’ väärtust suurendatakse enne tema väärtuse väljastamist:



Aritmeetikatehet saab kombineerida omistusega, kui muutuja nimi muidu korduks omistuslause mõlemal pool. Nii on avaldis arv1+=arv2 samaväärne avaldisega arv1=arv1+arv2, sama­moodi ka teiste aritmeetikatehete korral:

arv1\*=arv2; // tehe 3

printf ("%d. tehte tulemusena on arv1=%d; arv2=%d\n",

++tehe, arv1, arv2);

väljastatakse:

3. tehte tulemusena on arv1=24024; arv2=24

**2. Loogikatehted**. Tavalised tehtemärgid on võrdlustehetel suurem (>), väiksem (<), suurem-võrdne (>=) ja väiksem-võrdne (<=). **NB!** Tehe “võrdne” tuleb aga kirjutada kahe võrdusmärgiga (==), üks võrdusmärk tähistab vaid omis­tamist! Tavapärasest erinev märgi tähendus on tavaline viga C-programmis, mistõttu paljud translaatorid hoiatavad võimaliku vea eest järgmises if-lauses olevas tingimuses:

arv1=1;

if(arv1=0)printf("arv1=%d (null)\n", arv1);

else printf("arv1=%d (mittenull)\n", arv1);

Tulemus

arv1=0 (mittenull)

võib tunduda uskumatuna, kuid nulli omistamisel on avaldise väärtuseks null ehk mittetõene. Soovitud tulemuse annab ühe võrdusmärgi lisamine if-lause tingimusele:

if(arv1==0)printf("arv1=%d (null)\n", arv1);

Enamasti kasutatakse nulliga võrdlemisel aga eituse tehet (!arv), nii on veakindlam ja tingimus sageli paremini mõistetav, pealegi veel kahe märgi võrra lühem!

Tehte “mittevõrdne” tähistuseks on märgikombinatsioon ‘!=’, mitte ‘<>’, nagu paljudes teistes program­meeri­mis­keeltes. Keerulisemate tingimuste koostamisel tuleb enamasti kasutada loogikatehteid JA (&&) ning VÕI – kaks püstkriipsu (||) **NB!** Ka nende tehete puhul tuleb kasutada kahte märki, sest sama märk ühekordselt kirjutatuna on teistsuguse tähendusega tehtemärk, mis samuti igas avaldises lubatud. Loogilise eituse tähiseks on hüüumärk (!).

**3. Bititehted.** Muutuja mäluväljas olevaid bitte saab nihutada paremale (>>) või vasakule (<<), näidates nihu­tata­vate biti­positsioonide arvu. Loogikatehted saab teha bitthaaval, tehtemärkideks on siis ‘~’ (eitus), ‘&’ (kon­junktsioon) ja ‘|’ (disjunktsioon). Mõned näited bititehete tulemustest:

int rgb=0xFF;

int mask=0x110;

printf("Eitus:%x\n", ~rgb);

printf("Võrdlus maskiga: %x; %x\n", rgb&mask, rgb|mask);

printf("Bait vasakule:%x\n", rgb<<8);

Väljastatakse:

Eitus:ffffff00

Võrdlus maskiga: 10; 1ff

Bait vasakule:ff00

**4. Tingimustehe**. Avaldis koosneb sisuliselt kahest avaldi­sest ja tingimusest. Kui viimane on tõene, valitakse arvuta­miseks esimene avaldis, vastasel juhul teine. Exceli IF-funktsiooni kasutamis­kogemu­sega informaatikule on selline lähenemine tuttav:

(tingimus) ? (avaldis1, kui tingimus on tõene) :

(avaldis2, vastasel juhul);

Märkidest on kohustuslikud küsimärk ja koolon, sulud aitavad parandada avaldise loetavust.

Näiteks programmilõik küsib testi tulemust ja vähemalt 60 punkti saamisel väljastab teate testi edukast läbimisest:

int punkte, sooritus;

printf("Testis saadud punkte > ");

scanf("%d", &punkte);

sooritus=(punkte>=60)? 1 : 0;

if(sooritus)printf("Test edukalt läbitud\n");

Sama hästi saaks tingimustehte asemel kirjutada if-lause:

if(punkte>=60)sooritus=1; else sooritus=0;

Lause on küll veidi pikem, kuid annab samuti õige tulemuse. Suuremaks erinevuseks tingimustehte ja if-lause vahel on asjaolu, et tingimustehet saab kasutada igal pool avaldise koosseisus, ka seal, kus tema kasutamine võib tunduda ootamatu. Seda põhjusel, et tingimus­tehet sisaldavale avaldisele samaväärne if-lause ei pruugi antud kohas olla lubatud. Järgmine koodilõik väljastab isikukoodi alusel tema soo (paarisarvuga algavad isikukoodid kuuluvad naistele). Kümne­kohaline isikukood ei pruugi mahtuda 4-baidisesse täisarvu, seetõttu on ta kirjeldatud märgimassiivina ja esimese arvukoha leidmiseks kasutatakse märgikoodide lahutamist:

char ikood[11];

printf("Sisesta isikukood > ");

scanf("%s", ikood);

printf("Sugu: %c\n", (ikood[0]-'0')%2 ? 'M' : 'N');

Nagu näha, saab tingimustehet kasutada funktsiooni parameetrite väärtustamiseks. Tingimustehte otstarbekuse kohta olgu teinegi näide. Selleks on taas funktsiooni ’printf’ poole pöördumine, mis sobib asendama vastavat rida näite­programmis 2.2. Käändelõpud saab õigeks nii:

printf("%d %d-euro%s%s\n", mitu, eur[i],

 mitu>1 ? "st" : "ne",

 i>1? "" : mitu>1 ? " mynti" : " mynt");

Programm väljastab sel juhul sama sisestatud summa (3229) korral:



Sisestades summaks vaid ühe euro, väljastab programm aga:



Avaldise tavaline kasutuskoht on omistuslauses, mille üldkuju on

muutuja=avaldis;

omistuslause vasakut poolt tähistab C-keeles termin *lvalue (left value)*. Omistus­lause väärtus on avaldise väärtus. Omistuslauseid võib ühendada, näiteks nii: i=j=1;

Tüübiteisendus toimub automaatselt, kui avaldises osale­vad eri tüüpi andmed. Nii muudetakse float-arv vajadusel double-arvuks, täisarv murdarvuks, märgikood täisarvuks. Tagamaks avaldise soovitud tulemust, on kindlam vajalik teisendus ise ära näidata, kirjutades soovitud andmetüübi võtmesõna sulgudes avaldise komponendi ette. Kui programminäites 2.1 oleks hinnete summa kirjeldatud täisarvuna (mis ta sisuliselt ka on):

int hinded=0;

siis tuleks murdarvulise keskmise arvutamiseks tüübiteisendus paigutada keskmise hinde arvutamise avaldisse:

float keskmine;

//...

keskmine=(float)hinded/n;

Tehete prioriteedid määravad tehete täitmise järjekorra avaldises. Tehete täitmise järjekord on üldiselt vasakult paremale, vaid omista­misel paremalt vasakule. Lisas 1 on toodud tabelina kõigi C++ tehete täitmise järjestus.

# 4. Keelestruktuurid. Juhtlaused

Struktuurprogrammeerimise põhimõte nõuab, et programmi struktuur koosneks tegevuste jadadest, valikutest ja kordus­test. C-keeles on olemas piisaval arvul keelekonstruktsioone sobivaima valiku või korduse kirjapanekuks.

## 4.1. Valikud

**Ühene valik** sisaldab ühe tingimusavaldise ning üks või kaks tegevust. Kui valikulist tegevust ei saa kirja panna ühe lausena, siis tuleb nad paigutada looksulgudesse liitlausena. Esimest tegevust täidetakse siis, kui tingimus on tõene (avaldise väärtus erineb nullist), teist vaid siis, kui tingimusavaldise väärtus on null:

if(tingimus)tegevus;

if(tingimus)tegevus1; else tegevus2;

**NB!** Tingimus peab kindlasti olema sulgudes ja võtmesõna ‘then’ C-keeles puudub. if-lause on niivõrd arusaadav ja juba korduvalt varasemates näidetes kasutatud, et pikemat kommen­teerimist nähtavasti ei vaja.

**Mitmene valik** omab eeliseid siis, kui valikuid on vähemalt kolm või rohkem.

switch(avaldis){

case väärtus1: tegevus1

case väärtus2: tegevus2

...

default: tegevusx

}

Avaldise väärtuse järgi valitakse “sisenemiskoht”, millest alates alusta­takse tegevuste täitmist. Lauseid täidetakse kuni switch-lause lõpusuluni, kui teisiti pole näidatud. Enamasti aga on programmi loogika selline, et pea iga leitud väärtusele vastava tegevuse lõpus määratakse break-lausega “väljumis­koht”, millega lõpeb antud keele­konstruktsiooni täitmine.

**Näideprogramm 4.1. Keelevalik.cpp**. Mitmene valik sobib hästi lihtsate, numbervalikuga menüüde realiseeri­miseks. Näiteks kasutatava keele valik võib toimuda all­järg­neva programmiga:

#include <iostream>

using namespace std;

int main()

{

 char \*kysi, jah, ei, vastus;

 int keel;

 cout << "1 - Eesti\n2 - English\n3 - Russki\n";

 cin >> keel;

 switch(keel){

 case 1 : kysi="Kas jätkata"; jah='J'; ei='E'; break;

 case 2 : kysi="Continue"; jah='Y'; ei='N';break;

 case 3 : kysi="Prodolzat"; jah='D'; ei='H';break;

 default : kysi="1/0"; jah='1'; ei='0';}

 printf("%s? (%c/%c) > ", kysi, jah, ei);

 cin >> vastus;

 if(vastus==ei)return 1; // Katkestamine

// Siit algab tegelik programm

return 1;

}

Kui sisestada valiku vastusena ühe, on ekraanipilt selline:

1 - Eesti

2 - English

3 – Russki

1

Kas jätkata? (J/E) >

default-osa on siin selleks, et vale vastuse sisestamisel (pole 1, 2 ega 3) ei jääks muutujad väärtustamata**:**

1 - Eesti

2 - English

3 – Russki

4

1/0? (1/0) >

Negatiivne vastus lõpetab programmi töö koodiga 1:



##

## 4.2. Kordused

Korduseid on C-s kolm – for-kordus on sobivaim tsükli­muutuja kasutamise puhul, do-kordus järel- ja while-kordus eelkontrolliga korduse jaoks.

**Tsüklimuutujaga korduse** tavaline kuju on

for(algusavaldis; tingimusavaldis; lõpuavaldis) tegevus;

Sulgudes paiknev kordusspetsifikatsioon sisaldab kolme kom­ponenti. Kõik komponendid ja isegi tegevus võivad puududa, kuid kaks semikoolonit peavad igal juhul olemas olema. Algusavaldist täidetakse vaid üks kord, seejärel kontrollitakse tingimusavaldise väärtust. Edasi korratakse tegevust ja lõpu­avaldist seni, kuni tingimus saab mitte­tõeseks. Algus- ja lõpuavaldis võivad koosneda mitmest alam­avaldisest, siis eraldatakse nad üksteisest komadega. Näiteks järgmine for-kordus väljastab rea ”a8 b7 c6 d5 e4 f3 g2 h1”:

int i;

char c;

for(i=8,c='a';i>0;printf("%c%d ",c++,i--));

Kui tsüklimuutujat kasutatakse vaid for-lause sees, siis on küllalt levinud tema deklareerimine kordus­spetsi­fikatsioonis (**NB!** lubatud keeles C++, mitte C-s). Kõige tavalisem tsüklimuutujaga korduse algus, kus tegevusi täide­takse n korda tsüklimuutuja väärtustega nullist n-1-ni, on selline:

for(int i=0; i<n; i++)

Alljärgnev programm kuvab malelaua ruutude tähistused:

int main()

{

 for(int i=8;i>0;i--,printf("\n"))

 for (char c='a'; c<='h';printf("%c%d ",c++, i));

 return 0;

}



**Näiteprogramm 4.2. Apollo11.cpp**. Programm esitab küsimuse ja lubab pakkuda teatud arv kordi (antud näites viis korda) õiget vastust. Viimase leidmise hõlbustamiseks teata­takse iga vale vastuse korral, kummas suunas eksiti.

#include <iostream> // C++ sisend-väljund

using namespace std;

int main(void)

{

 const int oige=1969; // õige vastus

 const int limiit=5; // katsete lubatud arv

 int katseid=0; // katsete arv

 int i, vastus=0;

 cout << "Mis aastal astus esimene inimene Kuu pinnale?\n";

 for(i=0; i<limiit && vastus!=oige; i++){

 cout << i+1 << ". pakkumine > ";

 cin >> vastus;

 if(vastus>oige) cout << "See toimus varem\n";

 else if(vastus<oige) cout << "See toimus hiljem\n";

 }

 if(vastus==oige) cout << "See oli õige vastus\n";

 else cout << "Õige vastus on: "<< oige << ". aastal\n";

 return 0;

}

**Järelkontrolliga korduse** puhul täidetakse tegevus vähemalt üks kord, tingimuse väärtus määrab, kas ja mitu korda seda tegevust korrata. Programmeerija peab tagama, et mingil tsüklisammul muutub tingimuse väärtus mittetõeseks. Vastasel juhul on tegemist “lõputu tsükliga”, levinud ja tülika veaga programmide silumisel.

do tegevus while(tingimus);

Kui eelmise näite puhul loobuda vastuste piirarvust ja küsida seni, kuni saadakse õige vastus, siis võib korduse kirja panna selliselt:

 int katseid=0;

 do{

 cout << katseid+1 << ". pakkumine > ";

 cin >> vastus;

 if(vastus>oige) cout << "See toimus varem\n";

 else if(vastus<oige) cout << "See toimus hiljem\n";

 }while(vastus-oige);

Lõputingimuse avaldises lahutatakse sisestatud vastusest õige vastus, tulemus on nullist erinev järelikult suvalise vale vastuse puhul. Täpselt sama tulemuse annab ühe märgi võrra pikem tingimus (vastus!=oige).

Iseküsimus on see, kas tegemist on kuigi kasutajasõbraliku program­mee­rimisstiiliga. Küsimuse esitamist ei saa loobuda enne õige vastuse sisestamist. Ainus väljapääs on katkestus­klahvidele (Ctrl-C, Ctrl-Break) vajutamine, kuid siis ei saa teada õiget vastust.

**Eelkontrolliga korduse** puhul kontrollitakse tingimust enne, kui alustatakse korduva osa täitmist, seetõttu on võtmesõna while selles keelekonstruktsioonis kõige ees:

while(tingimus)tegevus;

Programmiloogika nõuab sel juhul kõigi vajalike andmete olemasolu tingimuse esmakordsel kontrollimisel ja sellest tulenevalt võib olla vajadus mõne lause ümberpaigutamiseks. Taas sama näite puhul lubab järgmine kordus loobuda vastamisest juba enne kordusesse sisenemist, kui vastuseks sisestada null (see ei saa kindlasti olla õige vastus):

 cout << "Mis aastal astus esimene inimene Kuu pinnale?\n";

 cout << "Loobumiseks sisesta 0\n";

 cin >> vastus;

 while(vastus && vastus-oige){ // vastus pole 0 ja erineb õigest

 if(vastus>oige) cout << "See toimus varem\n";

 else if(vastus<oige) cout << "See toimus hiljem\n";

 cout << "Proovi uuesti > ";

 cin >> vastus;

 }

 if(vastus==oige) cout << "See oli õige vastus\n";

 else cout << "Õige vastus on: "<< oige << ". aastal\n";

 return 0;

Vastates kohe ‘0’, ei hakata teda õige vastusega võrd­lemagi, sest kordustingimus ei ole täidetud:

Mis aastal astus esimene inimene Kuu pinnale?

Loobumiseks sisesta 0

0

Õige vastus on: 1969. aastal

## 4.3. Juhtlaused

**Suunamislause** goto jätkab programmi täitmist märgendiga tähistatud lauselt. Märgend vastab samadele nõuetele nagu muutuja nimigi ja kirjutatakse programmis suunatava lause ette. Märgendit ja täidetavat lauset eraldab koolon. Kuigi struktuurprogrammeerimine ei soosi goto-lause kasutamist ja harva erandina lubatakse vaid suunamist programmi­tekstis edasi (allapoole), võib goto olla mõnel juhul igati mõistlik. Seda eriti mitmetasemeliste ehk hõlmavate korduste puhul ja ka siis, kui kasutajapoolse vea tõenäosus on väga väike. Keele küsimise näiteprogrammis 4.1. on väheusutav, et vastamisel eksitakse. Soovides kontrollida sisestust ja vajadusel korrata küsimust, võib programmiteksti täiendada märgrndiga ja goto-lausega:

 int keel=0;

kysiuuesti:

 cout << "1 - Eesti\n2 - English\n3 - Russki\n";

 cin >> keel;

 if(keel<1 || keel>3)goto kysiuuesti;

**Katkestuslause** break katkestab korduse (while, do, for) või switch-lausesse paigutatud tegevuste täitmise. Mitmese korduse puhul kasutatakse break-lauset peaaegu alati iga valikulise tegevuse lõpulausena. Korduse puhul kasu­­ta­takse katkestuslauset vaid siis, kui korratav tegevus on vormistatud liitlausena ning break tähendab sisuliselt programmi töö jätkamist liitlause lõpusulule järgnevast lausest. Vahel võib programmeerimise teha lihtsamaks “lõputu tsükli” kasutamine, nagu on tehtud järgnevas näiteprogrammis.

**Näiteprogramm 4.3. Kysitlus.cpp**. Veebiportaalides on levinud tava esitada “päeva küsimus”. Järgmine programm soovib kõigepealt teada küsimuse sõnastust, mis eeldab vastusevariante Jah/Ei (tavaliselt on veel lisaks “Ei tea” vms). Kuna küsimuse tekstis võivad olla funktsiooni ‘printf’ vormingut eksitavad eraldajad (nagu näiteks tühik), on sisestus­funktsiooniks seekord ‘gets’. Nimetatud funktsioon edastab kõik enne Enter-klahvi sisestatud märgid ja reavahetuskoodi parameetrina näidatud mäluväljale. **NB!** Programmeerija peab tagama, et mälu oleks sisestatud teksti jaoks piisavalt.

Küsimuse esitamise järel küsitakse korduvalt vastuse­variante kuni nulli sisestamiseni. “Lõputu tsükli” while(1) ja tema sees paikneva katkestamise:

if(!vastus)break;

asemel võib tsükkel alata ka näiteks kordus­spetsifikatsioonideta for-lausega for(;;). Kuna protsendi­märk omab ’printf’ vormingus eri tähendust, on mugavam see märk deklareerida märgi­konstandina.

#include <iostream> // C++ sisend-väljund

using namespace std;int main(void)

{

 char kysimus[80];

 int jah=0, ei=0, vastus;

 const char protsent='%';

 cout << "Sisesta päeva küsimus > ";

 gets(kysimus);

 while(1){

 cout << kysimus << "\n\tVasta nii: 1=Jah, 2=Ei, 0=Lõpeta\n";

 cin >> vastus;

 if(!vastus)break;

 switch(vastus){

 case 1 : jah++; break;

 case 2 : ei++; break;

 default : cout << "Seda vastust ei arvestata!\n";

 }

 cout << "Senine vastuste jaotus: ";

 printf("Jah - %d(%6.2f%c), Ei - %d(%6.2f%c)\n",

 jah, (float)jah/(jah+ei)\*100, protsent,

 ei, (float)ei/(jah+ei)\*100, protsent);

 }

 return 0;

}

Sisestades ühe tähtsaima küsimuse juunikuus ja vastused 1, 2, 3, 1 ja lõpuks 0, on kuvaril:

Sisesta päeva küsimus > Kas jaanilaupäeval sajab?

Kas jaanilaupäeval sajab?

Vasta nii: 1=Jah, 2=Ei, 0=Lõpeta

1

Senine vastuste jaotus:

Jah - 1(100.00%), Ei - 0( 0.00%)

Kas jaanilaupäeval sajab?

Vasta nii: 1=Jah, 2=Ei, 0=Lõpeta

2

Senine vastuste jaotus:

Jah - 1( 50.00%), Ei - 1( 50.00%)

Kas jaanilaupäeval sajab?

Vasta nii: 1=Jah, 2=Ei, 0=Lõpeta

3

Seda vastust ei arvestata!

Senine vastuste jaotus:

Jah - 1( 50.00%), Ei - 1( 50.00%)

Kas jaanilaupäeval sajab?

Vasta nii: 1=Jah, 2=Ei, 0=Lõpeta

1

Senine vastuste jaotus:

Jah - 2( 66.67%), Ei - 1( 33.33%)

Kas jaanilaupäeval sajab?

Vasta nii: 1=Jah, 2=Ei, 0=Lõpeta

0

**Jätkamislause** continue katkestab tsüklisammu täitmise ja alustab uut. Tsüklimuutuja kasutamise korral tähendab see järgmise väärtusega tsükli­sammu alustamist. Näiteks järg­mised käsud testivad täisarvu jagumist 2, 4 ja 8-ga ning katkestavad jäägi arvutamise, kui tulemus on juba ette teada:

 int i, n;

 for(i=2;i<=16;i++){

 n=2;

 if(i%n)continue;

 cout << endl << i << " jagub " << n << "-ga ";

 n\*=2;

 if(i%n)continue;

 cout << "ning ka " << n << "-ga ";

 n\*=2;

 if(i%n)continue;

 cout << "ning ka " << n << "-ga ";

 }

Konsooliaknas kuvatakse:



**Lõpulause** return lõpetab funktsiooni töö ja edastab välja­kutsunud funktsioonile või operatsioonisüsteemile funkt­siooni lõpu­väärtuse:

return avaldis;

# 5. Funktsioonid

Moodulprogrammeerimise põhimõte nõuab, et programm koosneks iseseisvatest, korduvalt kasutatavatest kompo­nentidest, millel on võimalikult täpselt määratletud eesmärk. C-s ei jaotata mooduleid pea- ja alamprogrammideks ning funktsioonideks nagu paljudes teistes programmeerimis­keeltes. Kõik C moodulid on funktsioonid, tähtsaimat moodulit eristab teistest vaid tema nimi – funktsioonist ‘main’ alustatakse programmi täitmist.

Veidigi keerulisema ehk mitut funktsiooni sisaldava C/C++-programmi tüüpiline struktuur on:

* Preprotsessori käsud #include ja #define (lähemalt jaotises 11).
* Globaalmuutujate deklaratsioonid (lähemalt jaotises 9).
* Peaprogramm – funktsioon main, mis sisaldab teiste funktsioonide prototüüpe.
* Teiste funktsioonide definitsioonid ehk koodid.

## 5.1. Funktsiooni deklareerimine ja kirjeldamine

Funktsiooni deklareerimine on ühtlasi funktsiooni proto­tüübiks:

andmetüüp funktsiooninimi(parameetriloend);

andmetüüp näitab, kuidas käsitleda funktsiooni väärtust. Kui funktsioonil väärtus puudub, asendab seda võtmesõna void. Sama võtmesõna kasutatakse ka parameetriloendi puudumisel, kuid enamasti pole see loend tühi. Tavaliselt koosneb ta parameetrite andme­tüüpidele vastavatest võtme­sõnadest kas koos või ilma para­meetrite nimedeta. Parameetrite nimed loendis aitavad paremini mõista funktsiooni olemust, kuid kohustuslikud nad prototüübis pole. Standardfunktsioonide prototüüpe pole vaja ise kirjutada, nad paiknevad tavaliselt teekideks kutsutatavates ‘.h’-laiendiga failides ja lisatakse programmi teksti eelprotsessori #include-käsuga.

Funktsiooni väljakutsumine toimub omistuslauses, avaldise koosseisus või eraldi lausena. Viimast kasutatakse vaid juhul, kui funktsioonil väärtus puudub või seda programmis ei vajata. Viimasel juhul seega nii:

funktsiooninimi(parameetriloend);

**NB!** Sulud on kohustuslikud ka parameetrite puudumisel.

Funktsiooni kirjeldamise ehk definitsiooni esimene rida sarnaneb funktsiooni prototüübiga, kuid neil on kaks olulist erinevust. Esiteks on kirjelduses parameetrite nimed kohustuslikud ja teiseks on parameetriloendi lõpetava sulu järel looksulgudes funktsiooni tekst, mitte prototüüpi lõpetav semikoolon:

andmetüüp funktsiooninimi(parameetriloend)

{

 funktsiooni algoritm

 return funktsiooniväärtus;

}

Andmeedastus funktsioonide vahel toimub kas globaalmuutujate vahendusel (jaotis 9) või parameetrite teel. Parameetrid tagavad parema kontrolli andmete kasutamise üle funktsioonis ja seetõttu on nad andmevahetuseks eelista­tumad. Pealegi vastab tugevam kontroll andmete üle objektorienteeritud programmeerimise vaimule.

Funktsioonide kirjutamisel on oluline teada, mille poolest erinevad väärtus- (*by value*) ja viidaparameetrid (*by reference*). Esimesel juhul tehakse parameetri väärtusest KOOPIA ning edastatakse funkt­sioonile, teise juhul edasta­takse parameetri AADRESS. Hea tava näeb ette, et funktsioon parameetrite väärtust üldjuhul ei muuda. Siis pole parameetri edastamisviisil kuigi olulist tähtsust. Kuid teatud tingi­mustel on vaja, et funktsioon omaks juurdepääsu väljakutsuva funktsiooni muutuja mäluväljale ja saaks seda ka muuta. Nii poleks näiteks sisestusfunktsioonist ’scanf’ suurt kasu, kui ta ei saaks sisestatud andmeid paigutada väljakutsuva funktsiooni muutujate mälu­väljadesse.

**Näiteprogramm 5.1. Vahetus.cpp**. Erinevust kahe para­meetrite edastamisviisi vahel näitab järgmine programm, milles on kõigepealt kahe funktsiooni ‘koopiad’ ja ‘viidad’ prototüübid:

#include <stdio.h>

int main()

{

 void koopiad(int, int); // väärtusparameetrid

 void viidad(int\*, int\*); // viidaparameetrid

 int esimene=2016, teine=2020;

 printf("Esimene on aadressil %p\n", &esimene);

 printf("Teine aadressil %p\n", &teine);

 printf("1. Esimene=%d, Teine=%d\n", esimene, teine);

 koopiad(esimene, teine);

 printf("2. Esimene=%d, Teine=%d\n", esimene, teine);

 viidad(&esimene, &teine);

 printf("3. Esimene=%d, Teine=%d\n", esimene, teine);

}

void koopiad(int a, int b)

{

 int abi;

 abi=a; a=b; b=abi;

 return;

}

void viidad(int\* a, int\* b)

{

 int abi;

 abi=\*a; \*a=\*b; \*b=abi;

 return;

}

Mõlemad funktsioonid vahetavad parameetrite väärtused. Erinevad nad teineteisest parameetrite edasta­mis­viisi poolest, mis tingib ühtlasi ka erineva pöördumise parameetrite poole.

Programmi väljundist näeb, mis aadressil paiknevad muutujad ‘esimene’ ja ‘teine’ ning millised on nende väärtused enne ning pärast funktsioonide ‘koopiad’ ja ‘viidad’ poole pöördumist:



Kindlasti peab teadma seda, et muutuja väärtus edastatakse väärtus-, massiiv aga viidaparameetrina. Selline lahendus on igati loogiline, sest massiivist koopia tegemine igakordsel funktsiooni poole pöördumisel pole mõistlik mitmel põhjusel.

**Näiteprogramm 5.2. Keno.cpp**. Programm väljastab 20 juhuslikku ja sealjuures erinevat täisarvu vahemikus 1-64. Programmi koosseisus on mitmed funktsioonid, mida võib kasutada suvaliste täisarvuliste massiivide korral, nagu:

printa – väljastab massiivi esimese n elemendi väärtused;

nulli – nullib massiivi esimese n elemendi väärtused;

olemas – annab teada, kas otsitav väärtus on massiivi esimese n elemendi seas olemas (1, kui on; 0, kui pole);

jarjesta – järjestab kasvavalt massiivi algusest n elemendi väärtused.

Kuna massiiv edastatakse viidaparameetrina, siis funktsioonid ’nulli’ ja ’jarjesta’ saavad muuta massiivi elementide sisu. Juhuslikud arvud saadakse standard­funktsioonide ’srand’ ja ’rand’ abil. Esimene neist käivitab juhuslike arvude generaatori algväärtusega, teine annab juhusliku täisarvu. Lähemalt neist kahest funktsioonist jaotises 10. 2.

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#define N 20

#define MAKS 64

using namespace std;

void printa(int[], int);

void nulli(int[], int);

int olemas(int, int[], int);

void jarjesta(int[], int);

int main()

{

 int numbrid[N];

 int i, loosimine, juhuslik;

 printa(numbrid, N);

 nulli(numbrid, N);

 printa(numbrid, N);

 cout << "Loosimise number > ";

 cin >> loosimine;

 srand(loosimine);

 for(i=0;i<N;i++){

 do

 juhuslik=rand()%MAKS+1;

 while (olemas(juhuslik,numbrid, N));

 numbrid[i]=juhuslik;

 }

 printa(numbrid, N);

 jarjesta(numbrid, N);

 printa(numbrid, N);

 return 0;

}

void nulli(int a[], int n){

 for(int i=0;i<n;a[i++]=0);

}

void printa(int a[], int n){

 for(int i=0;i<n;printf("%d ", a[i++]));printf("\n");

}

int olemas(int x, int a[], int n){

 for(int i=0;i<n;i++)if(x==a[i])return 1;

 return 0;

}

void jarjesta(int arvud[], int n){

 int korras=0;

 int i, abi;

 while (!korras){

 korras=1;

 for (i=0; i<n-1; i++)

 if(arvud[i]>arvud[i+1]){

 abi=arvud[i];

 arvud[i]=arvud[i+1];

 arvud[i+1]=abi;

 korras=0;

 }

 }

 return;

}

Neljal korral väljastatakse massiivi ’numbrid’ sisu funktsioo­niga ’printa’. Esimene väljastus näitab, et algväärtustamata massiivi elementide väärtused on tõesti juhuslikud, teine näitab, et funktsioon ’nulli’ on teinud seda, milleks ta mõeldud on. Kolmas väljastus kuvab 20 erinevat täisarvu ja pärast nende järjestamist on lihtsam veenduda, et valitud numbrid on tõesti erinevad:

##

##

## 5.2. Programmi parameetrid

Funktsioonile ‘main’ edastatakse programmi käivitamisel tema nimi, parameetrite arv ja parameetrite massiivi algus­aadress. Neid andmeid võib kasutada, kuid ei pea. Senistes näidetes tähendas võtmesõna void main-funktsiooni alguses, et programm parameetreid ei vaja. Kui aga alustada pea­programmile vastavat funktsiooni selliselt:

int main(int argc, char \*argv[])

on ‘argc’ väärtus programmile edastatud parameetrite arvust ühe võrra suurem, ‘argv’ on viidamassiiv programmi nimele ja tema para­meetritele. Tegemist ei ole võtmesõnadega, kuid sellised parameetrite tähistused C/C++-programmis on üldlevinud. Järgneva programmi nimeks on ‘MainArg’.

int main(int argc, char \*argv[])

{

 printf("Parameetreid %d\n",argc-1);

 printf("Programmi nimi: %s\n", argv[0]);

 for (int i=1;i<argc;i++)

 printf("%d. parameeter: %s\n", i, argv[i]);

 return 0;

}

Windowsi käsuaknas pole tähtis, kas faili nimedes (seega ka programmide nimedes) kasutada suur- või väiketähti:



#

# 6. Failide kasutamine

C/C++ keel ei sisalda sõltumatuse ja universaalsues huvides andmete edastamise käske, andme­edastus on realiseeritud standardfunktsioonide või C++ puhul vastavate objekti­klasside abil.

## 6.1. C standardfunktsioonide teek stdio.h

Kahest standardsest andmevoost – sisendist ‘stdin’ ja väl­jundist ‘stdout’ – andmefailide kasutamise korral üldjuhul ei piisa. Põhi­mõtteliselt saab standardvooge ka ümber suunata, kuid enamasti ei saa programm loobuda klaviatuurisisendist ja võimalusest väljastada töö tulemused kuvarile. Iga andmefaili jaoks on mõistlik lisada uus andmevoog, vajaminevad deklaratsioonid selle jaoks on olemas standard­funktsioonide teegis stdio.h.

Teine võimalus on kasutada objektide klassi fstream, seda saab aga teha loomulikult vaid C++-s. Kõigepealt alustame standardse sisend-väljundteegis stdio.h tekstifailide jaoks tagatud võimaluste kirjeldamisega, neistki kaugeltki mitte kõigiga. Järgmises jaotises teeme sama näite fstream-klassi kasutades.

Failikirjeldus paikneb andmestruktuuris nimega FILE (andmestruktuuride kasutamine on kirjeldatud jaotises 9.4). Iga faili poole pöörduv funktsioon vajab ühe para­meetrina viita sellele struktuurile. Viit deklareeritakse nii (**NB!** suurtähed on kohustuslikud):

FILE \*viit\_failile;

Viitmuutuja nime võib vabalt valida. Paljudes näidetes kasutatakse failiviida nime ‘fp’, kuid see pole mitte võtme­sõna, vaid tuletatud sõnadest *file pointer*.

Fail kasutamiseks tuleb ta kõigepealt avada. Toimub see funktsiooni ‘fopen’ väljakutsega, mille prototüüp on

FILE\* fopen(const char\*, const char\*);

Esimene parameeter on faili nimi märgistringina, ta võib sisaldada ka juurdepääsuteed vastavalt operatsioonisüsteemi reeglitele. Võtme­sõna const parameetriloendis tähendab, et funktsioon ei muuda parameetrite väärtust. Ka teine parameeter peab olema märgistring, kuigi ta sisaldab sageli vaid ühte tähte:

* “r” tähendab sisendfaili;
* “w” tähendab väljundfaili (kui selles olid andmed, kirjutatakse nad üle);
* “a” tähendab väljundfaili, mille lõppu lisatakse andmeid.

Iga faili, eriti aga sisendfaili avamisel on kindlasti vaja kont­rollida faili avamise õnnestumist. Failiviida väärtus on NULL, kui avamine ebaõnnestus. Kui failiviit on dekla­reeritud levinud nimega fp:

FILE \*fp;

siis faili ebaõnnestunud avamist saab testida if-lauses lihtsa tingimusega:

fp=fopen(failinimi, "r");

if(!fp)...

Andmete edastamine tekstireana toimub funktsioonidega

char\* fgets (char\*, int, FILE\*);

int fputs (const char\*, FILE\*);

Funktsioonis ‘fgets’ on teine parameeter edastatavate märkide maksimumarv, viitmuutuja char\* on mõlemal juhul edastatav tekstistring. Nagu näha, on neis funktsioonides failiviit viimaseks parameetriks.

Andmete vorminguga edastamine toimub sarnaselt funktsioonidega ‘printf’ ja ‘scanf’, lisandunud on vaid sõnast *file* pärit ‘f’-täht funktsiooni nimes ja esimese parameetrina failiviit:

int fprintf (FILE\*, const char\*, ...);

int fscanf (FILE\*, const char\*, ...);

Funktsioonid andmete edastamiseks ühe baidi (märgi) kaupa:

int fgetc (FILE\*);

int fputc (int, FILE\*);

Edastatav märgi kood on neis funktsioonides täisarvuna.

Sisendfaili lõpu kontroll toimub funktsiooniga ‘feof’. Tema väärtus on tõene, kui ollakse jõutud faili lõppu:

int feof (FILE\*);

Faili sulgemine toimub funktsiooniga ‘fclose’:

int fclose (FILE\*);

**Näiteprogramm 6.1. TekstiFailiSisu.cpp.** Järgmine programm võib olla malliks suvalise programmi jaoks, mis vajab andmeid sisendfailist ja loeb neid sisse reakaupa. Failinimi ’Andmed.txt’ on programmi tekstis kirjas kons­tandina ja tema asendamine pole kuigi keeruline teistsuguste failinimede jaoks. Lisaks on ette nähtud võimalus anda failinimi ette käsureal, siis proovitakse avada etteantud nimega fail. Faili mitteleidmisel antakse veateade ja programm lõpetab töö veakoodiga 1.

#include <stdio.h>

#define MAXRIDA 240

int main(int argc, char \*argv[])

{

 FILE \*fp;

 char rida[MAXRIDA], \*failinimi="Andmed.txt";

 if(argc>1)failinimi=argv[1]; // Failinimi on ette antud käsureas

 fp=fopen(failinimi, "r");

 if(!fp){printf("Faili %s ei leia!\n", failinimi); return 1;}

 while(!feof(fp)){

 rida[0]=0; // rida tühjaks

 fgets(rida, sizeof(rida), fp);

 printf(rida); // failist loetud rea väljastamine

 }

 fclose(fp);

 printf("\n"); // reavahetus (failis ei pruugi lõpureas olla)

 return 0;

}

Kui programmiga TekstiFailiSisu.exe samas sisukorras on mõnerealine tekstifail ’Andmed.txt’ börsifirmade nimedega ja sisukorras ’c:\Andmed\Vallad.txt’ Eesti valdade loend, siis programmi kolm käivitust annavad konsooliaknas sellise väljundi:

****

Faili nimi ei pea olema programmi tekstis ega käsureas, teda võib ka küsida, nagu järgmises programmis.

**Näiteprogramm 6.2. ProgInfo.cpp.** Programm küsib tekstifaili nime ja proovib seda faili avada. Ebaõnnestumise korral antakse vastav teade ja sellega töö ka sisuliselt lõpeb. Kui fail leitakse, siis loetakse ta reahaaval sisse, igat rida läbitakse märkhaaval ja otsitakse teatud märke. Kuna programm eeldab, et tekstifail sisaldab C/C++ programmi teksti, siis otsitakse keele jaoks olulisi märke, nagu ploki/liitlause algus- ja lõpusulgu, lauset lõpetavat semi­koolonit ja kommentaaride algusmärke ‘//’ ja ‘/\*’. Lihtsuse mõttes loendatakse kõik leitud märgid ega analüüsita, kas otsitavad märgid on sees- või väljaspool tekstistringe, märgikonstante ja kommentaare. Tulemusena väljastatakse statistika leitud märkide kohta ja hoiatus, kui sulgude arv pole vastavuses. Sulgude mittevastavus on teatavasti üks raskemini avastatav viga vähegi keerukamates programmides.

#include <iostream>

#include <stdio.h> // standardne sisend-väljund

#define MAXRIDA 240

using namespace std;

int main(void)

{

 char fail[100]; // faili nimi

 char rida[MAXRIDA]; // failirida

 FILE \*fp;

 int algus=0, lopp=0, komm=0, lause=0;

 char mark;

 int pos; // märgi positsioon failireas

 printf("Programmi nimi (koos laiendiga) >");

 scanf("%s", fail);

 fp=fopen(fail, "r");

 if(!fp){

 printf("Faili %s avamine ebaõnnestus!\n", fail);

 return 1;} // jätkamisel pole mõtet

 printf("Info programmi %s kohta:\n", fail);

 printf("HOIATUS! Märgid konstantides võivad ");

 printf("mõjutada statistikat!\n");

 while (!feof(fp)) { // kordus faili lõpuni

 rida[0]=0; // rida tühjaks

 fgets(rida, MAXRIDA, fp); // rea lugemine

 pos=0; mark=rida[pos];

 do{

 mark=rida[pos++];

 switch (mark){

 case ';' : lause++; break;

 case '{' : algus++; break;

 case '}' : lopp++; break;

 case '/' : if(rida[pos]=='/')komm++;

 if(rida[pos]=='\*')komm++;

 } // switch

 }while(mark); // rea lõpumärgini

 } // faili läbimise tsükkel

 printf("Lauseid (semikooloneid) on %d.\n", lause);

 printf("Kommentaare on %d.\n", komm);

 printf("Algussulge on %d; lõpusulge %d.\n", algus, lopp);

 if(algus-lopp) // sulgude arv pole sama

 /\*see kommentaar tekitab segaduse sulgudega :)} \*/

 printf("SOOVITUS - kontrollige looksulge!\n");

 fclose(fp);

 return 0;

}

Kui sisestada programmi nimena tema enda nimi, on tulemuseks:

Info programmi proginfo.cpp kohta:

HOIATUS! Märgid konstantides võivad mõjutada statistikat!

Lauseid (semikooloneid) on 37.

Kommentaare on 13.

Algussulge on 6; lõpusulge 7.

SOOVITUS - kontrollige looksulge!

Näide väljundfaili kasutamisest tuleb jaotises 10.1, kus programmi TekstiFailiSisu.cpp on täiendatud logifailiga, millesse kirjutatakse lisaks failiinfole veel programmi käivi­tamise ja lõpetamise ajad.

## 6.2. Klass fstream

Objektiklass ’fstream’ on kirjeldatud samanimelises failis, nimetatud fail tuleb lisada #include-käsuga ja kasutatav failiobjekt dekla­reerida.

Objektiklassi fstream tähtsamad meetodid on:

open(failinimi, avamisviis) – faili avamine

Faili avamise režiimid on fstream::in sisendfaili jaoks ja fstream::out väljundfaili puhul. Ebaõnnestunud faili avamisel on failiobjekti väärtus NULL.

getline(andmed, makspikkus) – failist loetud reast edastatakse märgid märgimassiivi ‘andmed’, kuid mitte roh­kem kui ‘makspikkus’ väärtusega näidatud.

eof() – tõene, kui on lugemisega jõutud faili lõppu.

flush() – andmepuhvri tühjendamine

close() – faili sulgemine

Väljundfaili saadab andmed meetod: write(andmed, pikkus)

**Näiteprogramm 6.3. TekstiFailiFsisu.cpp.** Kasutaja jaoks käitub alljärgnev programm samamoodi, nagu varem kirjeldatud programm TekstiFailiSisu.cpp, realiseeritud on ta lihtsalt teiste vahenditega:

#include <fstream>

#include <iostream>

#define MAXRIDA 240

using namespace std;

int main(int argc, char \*argv[])

{

 fstream tekstifail;

 char rida[MAXRIDA], \*failinimi="Andmed.txt";

 if(argc>1)failinimi=argv[1]; // Failinimi on ette antud käsureas

 tekstifail.open(failinimi, fstream::in);

 if(!tekstifail){

 cout << "Faili " << failinimi << " ei leia!" << endl; return 1;}

 while(!tekstifail.eof()){

 rida[0]=0; // rida tühjaks

 tekstifail.getline(rida, sizeof(rida));

 cout << rida << endl; // failist loetud rea väljastamine

 }

 tekstifail.close();

 return 0;

}

Tulemus on nimetatud programmide puhul küll sarnane, kuid objektiklasside kasutamise korral pole põhjust imestada, kui loodud exe-fail on kümneid kordi suurem võrreldes vaid C standardvahenditega piirduva programmiga. Seda erinevust kinnitab ka Windows Explorer:



#

# 7. Viidatehnika

Massiivi korral on tavaline, et elementide poole pöördumisel ei kasutata mitte indekseid, vaid viitmuutujat. C/C++-s realiseeritud aadressaritmeetika teeb selle mitte ainult võimalikuks, vaid ka mugavaks. Viitmuutuja massiivi elementidele deklareeritakse sama tüüpi, kui on massiiv. Järgnevalt deklareeritud viitmuutuja ‘raha’ võib (kuid ei pruugi!) hakata viitama massiivi ‘eur’ elementidele.

int eur[]={1,2,5,10,20,50,100,200,500};

int \*raha;

## 7.1. Viida väärtus

Viida väärtustamine toimub tavaliselt omistuslausega. Kuna viitamisel massiivile on viida esimene väärtus tavaliselt massiivi esimese elemendi (indeks 0) aadress, siis programmi kirjutamise mugavuse nimel on omistusel massiivi nimi samaväärne tema esimese elemendi aadressiga. Järgmise kahe omistuslause tulemus on sama ning arusaadavalt eelistatakse reeglina programmideas kasutada teist:

raha=&eur[0];

raha=eur;

Massiivi teistele elementidele viitamiseks kasutatakse aadresstehet ja/või indeksit, nii viitab viitmuutuja ‘raha’ peale järgmise omistuslause täitmist massiivi ‘eur’ seitsmendale elemendile:

raha=&eur[6];

Aadressaritmeetika lubab teha viitmuutujatega aritmeetika­tehteid. **NB!** Liitmisel või lahutatamisel on viit­muutujate jaoks ühikuks alati andmetüübi mäluvälja pikkus! Viit­muutujate kasutamine massiivis olevate andmete poole pöördumisel muutub programmeerijale arusaadavaks ja loomulikuks alates sellest hetkest, kui mõistetakse tava- ja viitmuutuja tehete erinevust. Viitmuutuja suurendamine ühe võrra tähendab viitamist järgmisele massiivi elemendile. Viidale kahe liitmine tähendab viita ülejärgmisele elemendile, mida kinnitab ka koodilõik:

raha=&eur[3];

printf("Aadressil %p on %d-eurone\n", raha, \*raha);

raha+=2; // liidetakse kahe mäluvälja pikkus,// mitte arv '2'!

printf("Aadressil %p on %d-eurone\n", raha, \*raha);

Viida vormingutüübina printf-funktsioonis on tavaliselt ‘%p’ – viida väärtus 16ndsüsteemis. Ülaltoodud kaks printf-funktsiooni väljastavad:



Viitmuutuja õigeks kasutamiseks on tähtsad veel kaks momenti. Esiteks vahe tegemine viitmuutuja väärtuse (milleks on aadress) ja viidatava väärtuse vahel. Viimase poole pöördumisel on viitmuutuja nime ette vaja kirjutada tärn (\*). Teiseks tuleb jälgida tehete prioriteete (lisa 1) ehk nende täitmisjärjekorda. Tehete puhul viitmuutujatega on tähtis see, kas tehe tehakse kõigepealt aadressiga või siis viidatava väärtusega või vastupidi. Alati on parem lisada avaldises kindluse mõttes pigem enam sulge, kui jätta vajalikud avaldisest välja. C/C++ enamasti ei hoiata võimalikust veast, vaid täidab laused nii, nagu kirja pandud. Kui eelnev suurendustehe oleks selline:

\*raha+=2;

tähendab see viitmuutuja ‘raha’ sisaldaval aadressil oleva väärtuse suurendamist kahe võrra:

Aadressil 0022FF44 on 12-eurone

Viitmuutuja suurendamine/vähendamine tähendab viitamist massiivi järgmisele/eelmisele elemendile. Näite­programmis 5.2 oleva funktsiooni ‘jarjesta’ saab viita­mist eelistades kirja panna teisiti. Tulemus on sama ehk esimese n täisarvu alusel kasvavalt järjestatud täisarvude massiiv alates aadressist ‘arvud’:

void jarjesta(int \*arvud, int n){

 int korras=0;

 int \*viit, abi;

 while (!korras){

 korras=1;

 for (viit=arvud; viit<arvud+n-1; viit++)

 if(\*viit>\*(viit+1)){

 abi=\*viit;

 \*viit=\*(viit+1);

 \*(viit+1)=abi;

 korras=0;

 }

 }

 return;

}

## 7.2. Mitmesed viidad

Viit ei pruugi viidata otse andmetele, viidataval aadressil võib paikneda järgmine viit. Tegemist on sel juhul mitmese viidaga, mille deklareerimisel kasutatav võtmesõna lähtub ikkagi viidatavate andmete tüübist, lisandub aga veel üks viitamist tähistav tärn. Järgnevalt on kirjeldatud ja algväärtustatud pikk murdarv ‘pii’, viit temale ‘otse’ ja viimase aadressi sisaldav topeltviit ‘kaudu’:

double pii=3.14159265358979323846;

double \*otse=&pii;

double \*\*kaudu=&otse;

Järgmised väljastused aitavad mõista, mida need muutujad sisaldavad:

printf("Aadressil %p (pikkus %d) on: %p (pikkus %d)\n",

 kaudu, sizeof(kaudu), \*kaudu, sizeof(\*kaudu));

printf("Aadressil %p (pikkus %d) on: %lf (pikkus %d)\n",

 otse, sizeof(otse), \*otse,

 sizeof(\*otse));

Väljund:



Mitmeste viitade kasutamine lihtmuutujate korral pole üldiselt mõistlik. Kuid viidad nagu teisedki andmed võivad omakorda moodustada massiivi ning sellise massiivi elementide poole pöördu­misel saab indeksite asemel kasutada sama hästi viitamist. Viit viidamassiivile on topeltviit, andmete poole pöördumisel saab kasutada mitmeid kombinatsioone viitadest ja indeksitest. Mitmeseid viitu kasutatakse tavaliselt juurdepääsuks mitmemõõtmeliste massii­vide elementidele.

Kõige tavalisem mitmemõõtmeline massiiv C/C++ programmis on stringimassiiv, sest ta on realiseeritud kahemõõtmelise märgimas­siivina. Sealjuures stringide pikkused võivad ja enamasti ongi massiivis erinevad, mis võib raskendada topeltindeksite kasutamist. Kuna iga stringi algusmärgile on võimalik viidata, siis massiivi läbimine ja sealt andmete otsimine viitmuutujate abil võib indeksitest lihtsamgi olla, vähemasti paljude nurksulgude sisestamisest pääseb. Seda kinnitab ka järgmine näide.

**Näiteprogramm 7.1. Linnad.cpp.** Massiivis ‘linn’ on suuremate Eesti linnade nimed, täpsemalt koosneb massiiv viitadest linnade nimedele. Topeltviida ‘linnad’ väärtuseks saab viit esimese linnanime aadressile. Programm küsib linna esitähte ning liikudes topeltviidaga mööda stringimassiivi väljastab leitud linnanimed.

#include <stdio.h> // standardne sisend-väljund

int main(void)

{

 int n; // linnade üldarv

 int loend=0; // leitud linnade arv

 char \*linn[]={"Haapsalu", "Jõgeva", "Jõhvi",

 "Kuressaare", "Kärdla", "Narva", "Otepää",

 "Paide", "Põlva", "Pärnu", "Rakvere",

 "Tallinn", "Tartu", "Valga", "Viljandi"};

 char \*\*linnad=linn; // topeltviit märgimassiivile

 char \*esitaht; // viit märgile

 char otsi; // otsitav algustäht

 printf("Linna algustäht > ");

 scanf("%c", &otsi);

 n=sizeof(linn)/sizeof(char\*); // viitade ehk linnade arv

 for(linnad=linn; linnad<linn+n; linnad++){

 esitaht=\*linnad;

 if(otsi==\*esitaht){

 printf("%s \n",\*linnad);

 loend++;}

 }

 if(loend)printf("Leidsin %d linna\n", loend);

 else printf ("%c-ga algavaid linnu ei leidnud\n", otsi);

 return 0;

}

Sisestades esitatud tähe ‘K’, saame vastuseks:

Linna algustäht > K

Kuressaare

Kärdla

Leidsin 2 linna

Kui küsitud tähega linnanime pole massiivis, antakse ka sellest teada:

Linna algustäht > Y

Y-ga algavaid linnu ei leidnud

Põhimõtteliselt on võimalik viitamiste taset veelgi suurendada, näiteks programmilõik:

int aasta;

int \*v1=&aasta;

int \*\*v2;

int \*\*\*v3;

v2=&v1;

v3=&v2;

\*\*\*v3=2014;

(\*\*v2)++;

(\*v1)++;

printf("Aasta=%d.\n",\*\*\*v3);

väljastab:

 Aasta=2016.

Selliste ja veelgi keerulisemate mitmeste viitade põhja­li­kum käsitlus jääb antud õppematerjali raamidest välja.

# 8. Tekstitöötlus

Viitmuutujate kasutamise oskus lubab suhteliselt lihtsa vaevaga kirja panna enamiku funktsioone, mis on vajalikud tekstitöötluses vajaminevate tegevuste täitmiseks. Stringifunktsioonid on loomulikult ka valmiskujul saadaval, lisaks on nad realiseeritud C++ klassi meetoditena.

## 8.1. Tekstistringid programmis

Tekstistringide ehk märgimassiivide kasutamisel tuleb tagada, et tekstide jaoks oleks piisavalt mälu. Soovitatav on kirjeldada teksti sisaldav märgimassiiv pigem pikemana, kui riskida võimalike vigadega. Mõne baidi kokkuhoidmine pole juba ammu olnud kuigi mõistlik tegevus. **NB!** Viida deklareerimine märgimassiivile ei tähenda veel vajaliku mälu eraldamist! Viitmuutuja tuleb panna viitama piisavalt pikale märgimassiivile. Seda tuleb meeles pidada eriti selliseid stringifunktsioone kasutades, mis muudavad viidana edastatud tekstistringi sisu, nagu näiteks ühe stringi lõppu teise lisamine.

Kui deklareerida maakonna nime jaoks mõeldud muutuja nii:

char maakond[16];

siis eraldatakse mälu 16 märgi jaoks, kuid üks neist peab jääma stringi lõputunnuse jaoks. Tekst ”Lääne-Virumaa” sellesse märgimassiivi mahub, ”Lääne-Viru maakond” enam hästi mitte. Järgmine deklarat­sioon tähendab viitmuutujat maakonna nimele, mis on väärtus­tatud tekstiga ”Harjumaa”:

char \*maakond2="Harjumaa";

Seda teksti korralik C/C++ translaaror muuta ei luba ning selline lause programmis:

gets(maakond2);

peaks andma vea. Viga peaks tekkima ka siis, kui kasutatakse väärtustamata viitmuutujat, mis kindlasti ei sisalda sobiva mäluvälja aadressi (kuigi see pole harukorral välistatud):

char \*maakond3;

gets(maakond3);

Piisava pikkusega korralikult nullbaidiga lõpetatud teksti­stringide jaoks on lihtne kirjutada tavalisi stringi­funktsioone, näiteks ‘pikkus’ stringi pikkuse leidmiseks:

int pikkus(char \*str)

{

 char \*p;

 for(p=str; \*p; p++);

 return (p-str);

}

Funktsioonis liigub viitmuutuja ‘p’ esimese nullbaidini, stringi pikkus (ilma lõpumärgita) saadakse nullbaidi aadressist stringi esimese baidi aadressi lahutamisega. Funktsiooni ‘pikkus’ kasutab järgmine näiteprogramm.

**Näiteprogramm 8.1. Palindroom.cpp**. Programm hindab, kas massiivis ’sonad’ olevad tekstid on sümmeetrilised, st kas nende lugemine vasakult paremale annab sama tulemuse, kui lugedes paremalt vasakule. Sümmeetrilist sõna nimetatakse palindroomiks.

#include <stdio.h> // standardne sisend-väljund

int main(void)

{

 int i, j, marke, pali;

 char \*sonad[]={"sammas", "lammas", "kirik", "keset",

 "laat", "taat", "saippuakauppias", ""};

 char \*\*ps=sonad;

 int pikkus(char\*);

 do{

 pali = 1; // eeldame, et on palindroom

 marke=pikkus(\*ps);

 for (i=0,j=marke-1; i<marke/2; i++,j--) // pool sõna pikkusest

 if (\*(\*ps+i)!=\*(\*ps+j)) //täht algusest ja lõpust

 {pali=0; break;} // erinevad tähed

 printf ("%s %s palindroom\n", \*ps, pali ? "on" : "ei ole");

 ps++; // järgmine sõna

 }while(\*\*ps); // tühi sõna ("") on viimane

 return 0;

}

Programm teatab, et:



##

## 8.2. Standardfunktsioonid tekstistringidega

Stringifunktsioonide prototüübid asuvad teegis string.h. Paljude funktsioonide prototüüpides on kasutusel täisarv tähis­tusega size\_t. Programmide ülekantavuse ja uni­versaalsuse nimel on standardfunktsioonide teekides kirjeldatud mitmed sellised lisatüübid, mida saab samastada kas andmetüübiga int või long vaid ühte programmirida muutes.

Eelmises jaotises toodud funktsiooniga ‘pikkus’ sama tulemuse ehk stringi pikkuse ilma lõpumärki arvestamata annab stringifunktsioon ‘strlen’:

size\_t strlen (const char\*);

Kahe stringi võrdlemine toimub funktsiooniga ‘strcmp’. **NB!** Funktsiooni väärtus on 0 ehk ”mittetõene” siis, kui stringid on erinevad, kuigi võiks eeldada vastupidist. Võib võrrelda ka teatud arv märke alates stringide algusest, funktsiooni nimes ‘strncmp’ on siis lisaks ‘n’-täht:

int strcmp (const char\*, const char\*);

int strncmp (const char\*, const char\*, size\_t);

Järgnevate näidete puhul kasutatakse kahte stringi­konstanti:

char\* linn1="Tallinn";

char\* linn2="Tartu";

Nende võrdlemine funktsiooniga ‘strcmp’ annab tule­museks nullist erineva väärtuse ehk ‘tõese’, funktsiooni ‘strncmp’ väärtus sõltub aga sellest, mitut tähte võrreldakse.

for(int i=1; i<=strlen(linn2); i++)

printf("Esimesed %d tähte on %s\n",i,

 strncmp(linn1, linn2, i)?"erinevad":"samad");

väljastatakse:

Esimesed 1 tähte on samad

Esimesed 2 tähte on samad

Esimesed 3 tähte on erinevad

Esimesed 4 tähte on erinevad

Esimesed 5 tähte on erinevad

Stringi kopeerimisel funktsiooniga ‘strcpy’ tuleb tagada, et esimese stringi jaoks on olemas deklareeritud piisavalt baite. Sama nõue kehtib ka funktsiooni ‘strncpy’ kasutamisel, mis kopeerib näidatud arvu märke ja stringide sidurdamisel funktsiooniga ‘strcat’, kui esimese stringi lõppu lisatakse teine string. Nende funktsioonide tulemuseks on viit tulemus­stringile ehk sama, mis esimese parameetri väärtus.

char\* strcpy (char\*, const char\*);

char\* strncpy (char\*, const char\*, size\_t);

char\* strcat (char\*, const char\*);

Ülaltoodud programmilõigus olevatele lausetele on vaja lisada märgimassiivi ‘liin’ kirjeldus, sest tekstikonstante ‘linn1’ ja ‘linn2’ need funktsioonid muuta ei tohi:

char liin[40];

strcpy(liin, linn1);

strcat(liin, "-");

Ühe märgi otsimiseks stringis on funktsioon ‘strchr’, alamstringi otsimiseks ‘strstr’. Edu korral saab funktsiooni väärtuseks viit kas siis leitud märgile või stringile, vastasel juhul on väärtuseks tühi viit (NULL). Märgi otsimisel edastatakse märgikood täisarvuna, vajaduse korral tehakse teisendus.

char\* strchr (const char\*, int);

char\* strstr (const char\*, const char\*);

Kui lisada eelnevatele lausetele veel mõned printf-funktsioonide poole pöördumised tulemuste väljastamiseks:

printf("Kiirrong %s\n", strcat(liin, linn2));

printf("Linna otsing: %s\n", strstr(liin, linn1));

printf("Linna otsing: %s\n", strstr(liin, "Haapsalu"));

printf("Sihtkoha otsing: %s\n", strchr(liin, '-')+1);

kuvatakse:



Sihtkoha otsimisel annab funktsioon ‘-’-märgi aadressi lähte­stringis, millele liidetakse üks, et viit oleks sihtkoha esimesele märgile. **NB!** Nende funktsioonide puhul tuleb reeglina kontrollida otsingu edukust, näiteks nii:

char \*otsi;

if(otsi=strstr(liin, "Haapsalu")) printf("%s\n", otsi);

 else printf("Linna ei leidnud\n");

Funktsiooni väärtus omistatakse if-lause tingimuses viit­muutujale ‘otsi’, et poleks vaja kaks korda funktsiooni poole pöörduda.

Alamstringideks eraldab funktsioon ‘strtok’. Stringi tükeldamise aluseks on üks või mitu eraldajat, mis edastatakse stringi kujul funktsiooni teise parameetrina:

char\* strtok (char\*, const char\*);

Funktsioon muudab lähtestringi, asendades leitud eraldaja null-baidiga. Kui soovitakse jätkata sama stringi tükeldamist, siis on järgnevatel funktsiooni poole pöördumistel esimese parameetri väärtuseks tühiviit NULL.

**Näiteprogramm 8.2. Tükeldamine.cpp.** Tekstifailis nimega TallinnTartu.txt on andmed Tallinnast Tartusse sõitvate busside kohta. Väljumisaja, liini numbri ja marsruudi vahel on tabulatsioonmärgid (\t). Faili algus on selline:

8:30 4 TALLINN-TARTU

9:00 724 TALLINN-TARTU

9:05 884 TALLINN-TARTU-POLVA-VORU-HAANJA

9:30 716 TALLINN-PAIDE-POLTSAMAA-TARTU-KANEPI-VORU

9:45 112 TALLINN-TARTU-OTEPAA-KAARIKU-SANGASTE-ANTSLA

10:00 724 TALLINN-TARTU

Programmis tükeldatakse rida osadeks kõikide eraldajate alusel, milleks lisaks tabulatsioonimärgile on veel koolon kellaaja sees ja sidekriips sihtkohtade vahel. Igale osale hakkab viitama viitmuutuja nimega ’p’, mille väärtust kasutatakse vajadusel teiste muutujate väärtustamiseks. Funktsioon ’atoi’ teisendab teksti täisarvuks.

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int main()

{

 char \*failinimi="TallinnTartu.txt";

 char \*p, rida[100], \*eraldajad=" \t:-\n";

 FILE \*fp;

 int tund, minut, liin;

 fp=fopen(failinimi, "r");

 if(!fp){printf("Faili %s ei leia!\n", failinimi);return 1;}

 while(!feof(fp)){

 fgets(rida, sizeof(rida), fp);

 p=strtok(rida, eraldajad);

 tund=atoi(p);

 p=strtok(NULL, eraldajad);

 minut=atoi(p);

 p=strtok(NULL, eraldajad);

 liin=atoi(p);

 printf("%02d:%02d. Bussiliin %3d: ",

 tund, minut, liin);

 strtok(NULL, eraldajad); // väljumiskohta ei väljasta

 while(p=strtok(NULL, eraldajad))printf("%s ", p);

 printf("\n");

 }

 fclose(fp);

 return 0;

}

Programmi väljundis on tekstireast eraldatud andmeelemendid lähtefailiga võrreldes veidi teisel moel väljastatud:



##

## 8.3. Teisendusfunktsioonid

Kuna andmete sisestamine-väljastamine toimub tekstina, tuleb arvandmeid teisendada. Seda saab teha üheaegselt andme­edastusega vastavalt vormingule sellistes funktsioonides nagu ‘scanf’ ja ‘printf’. Sarnaste funktsioonidega saab teha tüübiteisenduse tekstandmetega, viit stringile on järgnevate funktsioonide esimeseks parameetriks:

int sprintf (char\*, const char\*, ...);

int sscanf (const char\*, const char\*, ...);

Nende funktsioonide prototüübid on teegis stdio.h. Järgmine programm küsib vastaja nime, millele võib lisada soovi korral vanuse. Kui sisestatud tekstis leitakse tühik, siis eeldatakse, et nimele järgneb vanus täisaastates:

#include <stdio.h>

#include <string.h>

int main()

{

 char rida[80], \*pos;

 char nimi[32];

 int vanus=0;

 printf("Sisesta nimi, soovi korral vanus > ");

 gets(rida);

 if(strchr(rida, ' ')) // tühik sisestusreas

 sscanf(rida, "%s%d", nimi, &vanus);

 else strcpy(nimi, rida);

 printf("Meile vastas ");

 if(vanus)printf("%d aasta vanune ", vanus);

 printf("%s", nimi);

 printf(".\n");

 return 0;

}

Andmed sisestatakse funktsiooniga ‘gets’, mis edastab kõik märgid, sh. eraldajad kuni reavahetusmärgini. Kui sisestatud tekstis leitakse funktsiooniga ‘strchr’ tühik, jaotatakse string ‘rida’ funktsiooniga ‘sscanf’ kahe muutuja vahel, teisendades tühikule järgneva tekstiosa täisarvuks. Väljastus sõltub sellest, kas vanust näidati või mitte:



Kui on vaja teisendada vaid ühte arvu, sobivad selleks teegis stdlib.h kirjeldatud funktsioonid ‘atoi’, ‘atof’, ‘atol’ jt. Viimane täht näitab, mis tüübiks tekstandmed teisendatakse:

int atoi(const char\*);

double atof(const char\*);

Eelnevas näites saab vanuse esitamise korral selle pai­gutada muutujasse ‘vanus’ ka nii (‘pos’ on viitmuutuja märgile, char \*pos;):

if(pos=strchr(rida, ' '))vanus=atoi(pos+1);

## 8.4. Märgifunktsioonid

Et lihtsustada märgi mingisse kategooriasse kuuluvuse kindlaks­tegemist, on prototüüpide teegis ctype.h kirjeldatud mitmed suhteliselt lihtsad funktsioonid. Parameetriks on neil märk ja funktsioon on tõene, kui märk kuulub vastavasse kategooriasse. Funktsiooni nimest on tavaliselt võimalik mõista, mida kontrollitakse:

* isaplha, isupper, islower – tõene, kui märk on täht / suurtäht / väiketäht;
* isdigit – tõene, kui märk on vahemikus ‘0’ – ‘9’;
* isxdigit – tõene, kui märk on tõlgendatav 16-ndsüsteemi arvuna;
* isalnum – tõene, kui märk on täht või number;
* isspace – tõene, kui märk on tühik, tabulatsioon või reavahetuskood;
* isprint, isgraph, isascii – tõene, kui märk on märk on teatud graafilist kuju omavate koodide vahemikus.

Kõigi nende prototüübid on sarnased funktsiooni ‘isalpha’ omaga:

int isalpha(int);

Tähtede puhul kerkib küsimus sellest, kuidas käsitletakse täpitähti või näiteks ‘š’-tähte. Kindlaim viis seda teada saada on järele proovida:

int main()

{

 int taht=0, mark=0, suur=0;

 char \*ps, rida[]="Tšehhi jäähoki MM koondis";

 for (ps=rida; \*ps; ps++){

 if (isascii(\*ps)) mark++;

 if (isalpha(\*ps)) taht++;

 if (isupper(\*ps)) suur++;

 }

 printf("Reas pikkusega %d on %d tavamärki,\n",

 ps-rida, mark);

 printf("sealhulgas %d tähte\n", taht);

 printf("\tneist %d suurtähte\n", suur);

 return 0;

}

Väljund näitab, et täpitähtede olemasolu tekstis nõuab eri­kohtlemist, sest nad on funkstiooni ’isupper’ poolt suurtähtede hulka loetud:

Reas on pikkusega 25 on 22 tavamärki

sealhulgas 22 tähte

neist 6 suurtähte

Pole kuigi keeruline teha täpitähtede jaoks oma funkt­sioonid. Suurtähe tunneb edukalt ära järg­mine funktsioon, kusjuures tähtede A-Z täiendamiseks tuleb nad lisada stringi­­konstandi ’eri’ hulka:

int onsuurtaht(int mark)

{

 char \*eri="ÜÕÄÖŠŽ";

 if (mark>='A' && mark<='Z')return 1;

 for (char \*p=eri;\*p; p++)if(\*p==mark) return 1; // erimärgid

 return 0;

}

Suur- ja väiketähtede teisendusfunktsioonidele kehtib sama jutt. Tähed vahemikus ‘a’-st ‘z’-ni ei tekita probleeme, täpitähtede juhul peab tulemuse üle kontrollima:

int tolower(int);

int toupper(int);

Neist funktsioonidest on abi näiteks siis, kui lubame sisestust nii suur- kui väiketähtedena, nagu alljärgnevalt:

printf("Kas jätkame (J/E) >");

scanf("%c", &vastus);

if(toupper(vastus)=='J')printf("Jätkame...\n");

Nii väikese kui suure ‘j’-tähe sisestamise korral on if-lause tingimus tõene.

## 8.5. Klass string

Küllaltki loomulik, et objektiklasside lisamisega program­meerimiskeelde C++ oli tekstistringi klassi loomine üheks esimeseks ettevõtmiseks. Tülikate märgimassiivide asemel võib kasutada objektiklassi string, milles on rohkelt ära kasutatud võimalust tehete üledefineerimiseks (lähemalt jaotises 13.2).

Klassi string kuuluva objekti deklareerimisel on lubatud algväärtustamine.

Objektiklassis on kirjeldatud järgmised tehted:

= teksti omistamine,

+ tekstide sidurdamine

+= sidurdamine koos omistamisega

[] märk tekstis

Kasutada saab meetodeid:

length – stringi pikkus,

compare – stringide võrdlemine,

substr – alamstring,

find – alamstringi otsimine lähtestringist.

Lihtne programm objektiklassi string võimaluste näita­miseks:

#include <iostream>

using namespace std;

int main()

{

 string eesnimi="James", perenimi="Bond";

 string taisnimi, nimi;

 cout << "Tere, mina olen " << eesnimi << endl;

 taisnimi=eesnimi+" ";

 taisnimi+=perenimi;

 cout << taisnimi << endl;

 for(int i=0;i<perenimi.length();i++)

 cout << perenimi[i] << endl;

 cout << "Mis on Sinu nimi > ";

 cin >> nimi;

 if(!eesnimi.compare(nimi.substr(0,5)))

 cout << "Oo, nimekaim!" << endl;

 return 0;

}



# 9. Mälu kasutamine

Arvuti mälu vajavad nii programm ise kui tema poolt töödeldavad andmed – muutujad ja konstandid. Programmi masinkoodidele mälu eraldamise osas ei pea programmeerija üldjuhul muret tundma. Küll aga peab ta teadma, millal eraldatakse mälu andmetele ja kuidas seda toimingut vajadusel ise juhtida saab. Eristatakse staatilist ja dünaamilist mälujaotust. Esimesel puhul eraldatakse ja vabastatakse mälu vastavalt keelereeglitele, teisel puhul on mälu kasutamine realiseeritud programmis.

## 9.1. Muutuja skoop (kasutuspiirkond)

Muutuja deklareerimise koht määrab ära, millistes programmi osades saab teda kasutada. Väljas-(ees)pool funktsioone kirjeldatud muutuja on globaalne muutuja, tema kasutus­piirkonnaks (*scope,* siit ka termin skoop) on kõik funktsioonid antud programmis. Funktsiooni sees kirjeldatud muutuja kehtib antud ploki (funktsiooni) piires, nii nagu ka liitlauset moodustava ploki sees deklareeritud muutuja skoobiks on vaid antud plokk. Üldjuhul eraldatakse mälu plokki sisenemisel ja vabastatakse korduvkasutamiseks ploki lõpus. Nii on alljärgnevas for-korduses kirjeldatud tsüklimuutuja kehtiv vaid selles korduses, järgmiste lausete puhul antakse transleerimisel viga – muutuja ‘i’ kasutamine ei vasta skoobireeglitele:

char\* linn2="Tartu";

for(int i=0; i<strlen(linn2); i++) printf("%c", toupper(linn2[i]));

printf("Linna nimes on %d tähte\n", i);

Olukorda parandaks muutuja, mis asub väljaspool for-kordust:

int n=strlen(linn2);

for(int i=0; i<n; i++)

 printf("%c", toupper(linn2[i]));

printf("\nLinna nimes on %d tähte\n", n);

Nüüd, kui printf-funktsiooni parameetris on asendatud ‘i’ temast suurema skoobiga muutujaga ‘n’, saab programmi täita ja tulemusena väljastatakse

TARTU

Linna nimes on 5 tähte

Näide muutujate eri ulatusega skoopidest on programm, mis väljastab täisarvu astmed vahemikus 1...8. Globaalne muutuja ‘baas’ määrab, millise täisarvu astmeid väljastatakse. See muutuja kirjelda­takse väljaspool funktsioone ja tema väärtust kasutatakse mõlemas funktsioonis, esimeses neist ta saab ühtlasi algväärtuse. Muutujaid ‘i’ on kaks, üks funktsioonis ‘main’ ja teine, samanimeline tsüklimuutuja, kehtib vaid funktsiooni ‘aste’ korduslause piires:

int baas;

int main(void)

{

 int i;

int aste(int p);

 baas=2;

 for(i=1; i<=8; i++)printf("%d astmel %d teeb %3d\n",

 baas, i, aste(i));

return 0;

}

int aste(int p)

{

 int x=1;

 for(int i=0; i<p; i++)x\*=baas;

 return x;

}

Programm väljastab:

##

##

## 9.2. Staatilised muutujad

Võtmesõna static muutuja deklaratsioonis tähendab seda, et muutuja mälu ja ühtlasi ka tema väärtus jääb alles plokist väljumisel. Ploki järgmisel täitmisel jätkatakse sama väär­tusega.

Funktsioonis ‘lisa’ saab staatiline muutuja ‘x’ algväärtuse mälu esmakordel eraldamisel:

int lisa(void)

{

 static int x=0;

 return ++x;

}

Kuna staatilise muutuja mälu ei vabastata funktsiooni töö lõppedes ega ka nullita, kuvab lause:

for(int i=1; i<=8; i++)printf(" %d", lisa());

arvujada 1 2 3 4 5 6 7 8.

## 9.3. Mälu dünaamiline eraldamine ja vabastamine

Mälu tellimiseks on vajalik viitmuutuja. Tema andmetüüp võib olla määramata (void), kuid kindlasti on parem näidata, mis tüüpi andmetele mälu vajatakse ja edaspidi viitama hakatakse. Nii tagatakse programmi paremale arusaadavusele lisaks õige aadressaritmeetika. C++ tehted new ja delete vastavalt eraldavad ja vabastavad mälu. Mälueraldus toimub omistuslausega, milles näidatakse kas ainult andmetüüp või lisaks andmetüübile andmeelementide arv:

viit = new andmetüüp;

viit = new andmetüüp[n];

Viida väärtus on NULL mälu eraldamise ebaõnnestumisel.

 char \*viit;

 viit=new char[1000000];

 printf("Miljonibaidise märgimassiivi algusaadress on: %p\n",

 viit);

Järgnevast aadressist alates on võimalik kasutada mälu miljonit märki mahutavat mäluruumi:

Miljonibaidise märgimassiivi algusaadress on: 004D0020

Kui eraldatud mäluplokki enam ei vajata, saab ta vabastada ja anda võimalus seda korduvkasutada tehtega delete. Seal­juures näida­takse sama aadress, mida kasutati tehtes new:

delete viit;

**Näiteprogramm 9.1. Vasakult.cpp**. Funktsioon ‘vasa­kult’ eraldab parameetrina etteantud stringi algusest soovitud arvu märke, eraldades nende jaoks vajalikul hulgal mälu. Ta võib jätkata märkide eraldamist poolelijäänud kohast, kui funktsiooni järgmistel välja­kutsumistel on esimese parameetri väärtuseks aadressi asemel näida­tud NULL. Funktsiooni käitumine on veidi sarnane standard­funktsiooni ‘strtok’ omaga, millest on näide jaotises 8.2. Erinevalt funktsioonist ’strtok’ aga ei muudeta lähteteksti, vaid eraldatakse iga kord uus mälu teatud märkide koopia jaoks. Staatiline muutuja ‘sp’ jätab meelde koha, kus eelmine kord märkide eraldamine pooleli jäi.

#include <iostream>

using namespace std;

char \*vasakult(char \*p, int n){

 static char \*sp;

 char \*np;

 int i;

 if(p)sp=p; // kas jätkata eelmisest kohast?

 np=new char[n+1];

 for(i=0;i<n;i++){\*(np+i) = \*sp; sp++;}

 \*(np+i)=0; // lõpetame teksti

 return np;

}

int main()

{

 char \*t="ABCDEFGHIJKLMNOPQRSTUVWZYZ";

 char \*t2="0123456789abcdef";

 int \*a, i, n=3;

 printf("%s\n", vasakult(t, n++));

 printf("%s\n", vasakult(NULL, n++));

 printf("%s\n", vasakult(NULL, n++));

 printf("%s\n", vasakult(NULL, n++));

 printf("%s\n", vasakult(t2, n++));

 printf("%s\n", vasakult(NULL, n++));

 printf("%s\n", vasakult(NULL, n++));

 return 0;

}

Märke väljastatakse sõltuvalt sellest, palju neid küsitakse ja palju neid tekstistringis on. Viimasel funktsiooni väljakutsel sattus alamstringi sisse ka lähtestringi lõputunnus, seetõttu kuvatakse vaid ’f’-täht:

ABC

DEFG

HIJKL

MNOPQR

0123456

789abcde

f

Mälu eraldamiseks ja vabastamiseks on C-keeles olemas ka funktsioonid, mille kirjeldused on failis stdlib.h. Nende väärtus pole seotud konkreetse andme­tüübiga, parameetrina tuleb näidata vajalik mälumaht baitides või ‘calloc’-i jaoks elemendi pikkus ja elementide arv. Funktsioon ‘realloc’ lubab varem eraldatud mäluploki suurust muuta, ‘free’ vabastab mäluploki.

void\* calloc (size\_t, size\_t);

void\* malloc (size\_t);

void\* realloc (void\*, size\_t);

void free (void\*);

Uue mäluploki eraldamisel võib olla kasu funktsioonist ‘memcpy’, mis kopeerib teatud hulk baite ühest mäluosast teise sõltumata seal asuvatest andmetest:

void\* memcpy (void\*, const void\*,size\_t);

**Näiteprogramm 9.1. Malloc.cpp**. Programmis eraldatakse mälu nii funktsioonidega malloc kui realloc. Massiivi sisu väljastab inline-funktsioon ’printa’.

#include <stdio.h>

#include <stdlib.h>

inline void printa(int \*p, int n)

 {int i; for(i=0, printf("%p: ", p);i<n;

 printf("%d ", \*(p+i++)));

printf("\n");}

int main()

{

 int n=5;

 int i, \*a, \*aa;

 a=(int\*)malloc(n\*sizeof(int));

 printa(a, n);

 a[0]=1;

 for(i=1;i<n;\*(a+i)=a[i-1]+i++);

 printa(a, n);

 n\*=2;

 aa=(int\*)malloc(n\*sizeof(int));

 printa(aa, n);

 aa=(int\*)realloc(a,n\*sizeof(int));

 printa(aa, n);

 return 0;

}

Esimese mälueraldusega saadud massiivis on juhuslikud arvud, mis asendatakse viie täisarvuga, mis saadud Fibonacci jada arvutusvalemile pisut sarnase avaldisega. Kaks korda suurema massiiv loomine sisaldab juhuslikke väärtusi, kui seda teha funktsiooniga ‘malloc’. Funktsioon ‘realloc’ säilitab aga massiivi olemasolevad väärtused, kuigi eraldatud mälu aadress on iga kord erinev:



##

## 9.4. Andmekooslused

Lisaks massiivile, mis sisaldab mitut sama tüüpi andme­elementi, saab andmeid koondada kokku ka siis, kui nende tüübid on erinevad. Selleks on C/C++-s mitu võimalust.

**Struktuurid**

Andmestruktuuri kirjelduses näidatakse struktuuri kompo­nendid tavaliste deklaratsioonilausetena nii, nagu moo­dustaksid nad liitlause:

struct nimi1{

 andmetüüp1 muutuja1;

 andmetüüp2 muutuja2;

... }nimi2;

Struktuuril on üks või kaks nime. Esimene neist on enne looksulge, näites ‘nimi1’ ja ta on struktuuri (objekti) üld­nimeks. Teda saab kasutada edaspidi deklaratsioonides kui uue andmetüübi nime. Looksulu järel olev nimi või nimed on programmis kasutatavad andme­struktuuri eksemplarid. Näiteks järgmise lausega:

 struct teaduskond{

 char \*nimi;

 int tudarv;};

on kirjeldatud struktuur, mille üldnimeks on ‘teaduskond’ ning mis sisaldab viida teaduskonna nimele ja tudengite arvu teaduskonnas. Andme­struktuurid võivad moodustada massiive ja neile saab viidata. Struktuuri­massiivi ja viitmuutuja deklareerimisel näidatakse struktuuri üldnimi:

 teaduskond teaduskonnad[8], \*viit;

Andmeelemendi (struktuuri komponendi) poole pöördumine toimub liitnimega. Andmestruktuuri ja tema komponendi vahel on punkt. Struktuurimassiivi puhul on indeks pärast struktuuri, mitte andmeelemendi nime:

teaduskonnad[i],tudarv

Andmestruktuuri dünaamiline loomine toimub tehtega new.

viit=new teaduskond;

Viida olemasolul on andmeelemendile pöördumisel liitnime asemel lihtsam kasutada viitamistehet:

viit->tudarv; // ehk (\*viit).tudarv

**Näiteprogramm 9.3. Teaduskonnad.cpp.** Failis nimega Teaduskonnad.txt on TTÜ tEaduskondade nimed ja tudengite arv. Andmed on eraldatud tabulatsioonimärgiga:

Ehitusteaduskond 1435

Energeetikateaduskond 826

Infotehnoloogia teaduskond 2400

Keemia- ja materjalitehnoloogia teaduskond 679

Majandusteaduskond 3150

Matemaatika-loodusteaduskond 817

Mehaanikateaduskond 1171

Sotsiaalteaduskond 1686

 Failis olevad andmed salvestatakse struktuurimassiivi ’teaduskonnad’ ning väljastatakse järjestatuna tudengite arvu järgi.

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int main()

{

 const int n=8;

 char \*failinimi="Teaduskonnad.txt";

 char \*p, rida[100];

 FILE \*fp;

 int i, ok;

 char\*kuva(char\*);

 struct teaduskond{

 char \*nimi;

 int tudarv;};

 teaduskond \*teaduskonnad[n], \*viit;

 fp=fopen(failinimi, "r");

 if(!fp){printf("Faili %s ei leia!\n", failinimi);return 1;}

 for(i=0;i<n;i++){

 fgets(rida, sizeof(rida), fp);

 strtok(rida, "\t"); // eraldame teaduskonna nime

 p=strtok(NULL, "\n"); // tudengite arvu algab siit

 viit=new teaduskond;

 teaduskonnad[i]=viit;

 viit->nimi=new char[strlen(rida)+1];

 strcpy(viit->nimi, rida);

 viit->tudarv=atoi(p);

 printf("%s, %d tudengit.\n", viit->nimi, viit->tudarv);

 }

 fclose(fp);

 printf("Sorteerime...\n");

 do{

 ok=1;

 for(i=0;i<n-1;i++)

 if(teaduskonnad[i]->tudarv<teaduskonnad[i+1]->tudarv){

 ok=0; // teaduskonnad vales järjestuses

 viit=teaduskonnad[i];

 teaduskonnad[i]=teaduskonnad[i+1];

 teaduskonnad[i+1]=viit;

 }

 }while(!ok);

 for(i=0;i<n;i++)printf("%s, %d tudengit.\n",

 teaduskonnad[i]->nimi, teaduskonnad[i]->tudarv);

 return 0;

}

Väljastatakse TTÜ teaduskondade andmed ja järjestatakse teaduskonnad mull-meetodil (*Bubble sort*) tudengite arvu alusel. Nimetatud meetod vahetab kõrvuti olevate andmete asukoha, kui see pole õiges järjestuses. Esimese kordusega jõuab massiivi lõppu suurim väärtus, sellest ka meetodi nimi. Naabrid jätkavad asukohtade vahetust seni, kuni kõik massiivi elemendid paiknevad oma õigetel kohtadel. Väljundit või ka lähtefaili sisu vaadates selgub pisut üllatav tõsiasi, et üheski TTÜ teadus­konna nimes pole täpitähti:



**Ühendstruktuur**

Väliselt sarnane tavalisele andmestruktuurile, kuid võtmesõnaga union kirjeldatud ühendstruktuuri kompo­nendid jagavad sama mälupiirkonda. Seetõttu määrab ühendstruktuuri pikkuse ära tema pikim komponent, mitte nende pikkuste summa. Millist andme­elementi ühendstruktuur mingil hetkel sisaldab, peab olema võimalik programmis mingil moel otsustada. Järgmine deklaratsioon tähendab, et ‘sisu’ võib olla kas täisarv või viit tekstistringile:

union sisu{

 int arv;

 char \*tekst;};

**Loendustüüp**

Programmi teksti loetavuse parandamiseks võib täisarvulistele väärtustele anda nimed. Loendustüübi deklaratsioon näitab vastavust nimede ja väärtuste vahel. Esimene väärtus jadas on vaikimisi 0 ja iga järgmine eelmisest ühe võrra suurem.

enum nimi{

 väärtus1,

 väärtus2, ...};

Arvväärtuste ja nimede vastavust saab deklaratsioonis näidata, kui vaikimisi nummerdusviis ei sobi. Järgmises kirjelduses samastatakse ‘esmaspaev’ ühega, ‘teisipaev’ kahega jne:

enum toonadal{

 esmaspaev=1,

 teisipaev, kolmapaev, neljapaev, reede};

**Lisatüüp**

Olemasolevale andmetüübile teise, lisatähistuse andmise võimalus tagab programmide parema ülekantavuse. Uus andmetüüp kirjelda­takse lähtuvalt juba olemasolevast:

typedef olemasolev\_tüüp uus\_tähistus;

Paljudes standardfunktsioonides parameetrina esinev täisarvutüüp size\_t võib olla kirjeldatud nii:

typedef long size\_t;

# 10. Standardfunktsioonide teegid

Standardfunktsioone on kasutatud kõigis varasemates näide­tes. Selles jaotises on kokkuvõte enam vajaminevatest standardfunktsioonide teekidest ja lisaks kirjeldatud mõningaid kasulikke funktsioone, mida senistes programmi­näidetes pole esinenud.

Standardfunktsioonide teegid on tekstifailid, mis koosnevad suures osas eelprotsessori käskudest ja funktsioonide prototüüpidest. Enim­kasutatavate funktsioonide kirjeldused on sageli dubleeritud mitmes teegis. Lisaks prototüüpidele on teekides tavalised konstantide, struktuuride, objektiklasside ja makrode kirjeldused ning *inline*-funktsioonid.

Praktiliselt iga C++ translaatoriga on kaasas järgmised failid:

* stdlib.h – üldkasutatavate funktsioonide teek;
* math.h – matemaatikafunktsioonide teek;
* stdio.h – standardsisendi ja -väljundi teek;
* conio.h – vahetu sisendi ja -väljundi (ekraan, klaviatuur) teek;
* iostream.h – C++ klassidel põhineva sisendi ja väljundi teek;
* ctype.h – andmetüüpidega seotud funktsioonide ja makrode teek;
* string.h – stringifunktsioonide teek;
* time.h – ajafunktsioonide teek;
* alloc.h – mälu eraldamise, vabastamise ja kasutamise funktsioonide teek.

## 10.1. Ajafunktsioonid (time.h)

Funktsioon ‘time’ annab arvutikella aja sekundites alates 1970.aasta 1. jaanuari südaööst.

time\_t time (time\_t\*);

Parameeter võib olla kas NULL või ‘time\_t’-tüüpi muutuja aadress. Andmetüüp ‘time\_t’ on teegis kirjeldatud pika täis­arvuna:

typedef long time\_t;

Selline tüübidefinitsioon tähendab seda, et suurim väärtus saab andmetüübina ’time\_t’ kirjeldatud muutujal olla 231. Mida see tähendab, näeme selle jaotise lõpus.

Arvutikellalt saadud sekundeid saab kasutada vähemalt kahel moel. Esiteks saab mõõta mingiks tegevuseks kulunud aega, pöördudes funktsiooni poole tegevuse alguses ja lõpus:

time\_t s1; // algusaeg

long i, j, x;

const long korda=100000;

s1=time(NULL); // algusaeg

for(i=0;i<=korda;i++)

 for(j=0;j<=korda;j++)x=i+j;

printf("Aega kulus: %ld sekundit\n", time(NULL)-s1);

Programmi väljund sõltub arusaadavalt arvuti protsessori jõudlusest. Antud materjal on koostatud arvutil, millel topelttsükli ja 10 miljardi liitmise ja omistuse täitmiseks läks umbes pool minutit:

Aega kulus: 29 sekundit

Alati ei piisa sekundilisest täpsusest aja mõõtmisel. Sel juhul tuleb kasutada funktsiooni ‘clock’, mille mõõtühikuks on CLK\_TCK. Kui suur see mõõtühik on, näeb teegist ‘time.h’:

#define CLOCKS\_PER\_SEC ((clock\_t)1000)

#define CLK\_TCK CLOCKS\_PER\_SEC

Funktsiooni prototüübist selgub, et parameetreid vaja pole:

clock\_t clock (void);

**Näiteprogramm 10.1. Tahestik.cpp**. Programm näitab sajandik­sekundilise täpsusega aega, kui kiiresti sisestatakse kogu tähestik ehk kõik tähed õiges järjestuses 'a'-st 'z'-ini. Klahvivajutuse sisestab failis conio-h kirjeldatud funktsioon ’getche’, mis sisestatud tähe ka kuvab (viimane täht funktsiooni nimes on sõnast *echo*). Hinnang “PERFEKT!” antakse siis, kui 26 tähteklahvile vajutati vähem kui 15 sekundiga. Vastasel juhul jäetakse hinnang andmata.

#include <time.h>

#include <stdio.h>

#include <conio.h>

int main(void)

{

 clock\_t s1, s2; // s1 - algusaeg; s2 – lõpuaeg,

 float aeg; // aeg - nende vahe

char c;

char x='?'; // et algväärtus poleks juhuslikult 'a'

s1=clock(); // stopper käima

for(c='a';c<='z';c++) // tähed a...z

 while(x!=c) // kordame õige tähe sisestamiseni

 {printf("\nVajuta %c>", c);

 x=getche();} // kuvame vajutatud klahvle vastavat märki

s2=clock(); // stopper seisma

aeg=(float)(s2-s1)/CLK\_TCK,

printf ("\nAega kulus: %5.2f sekundit. %s\n",

 aeg, aeg<=15.0 ? "PERFEKT!" : "");

return 0;

}

Väljundi lõpp kujunes selliseks y-tähe sisestamisel meelega tehtud vea tõttu, et näidata programmi käitumist valele klahvile vajutamisel. Muidu oleks aeg olnud õige pisut parem:



Teine funktsiooni ‘time’ väärtuse kasutuskoht on tema edastamine parameetrina mõnele teisele funktsioonile, nagu näiteks ‘localtime’. Nimetatud funktsioon väärtustab temal teadaoleva info põhjal struktuuri ‘tm’ komponendid. Neid komponente on üheksa:

struct tm

{

 int tm\_sec; /\* Seconds: 0-59 (K&R says 0-61?) \*/

 int tm\_min; /\* Minutes: 0-59 \*/

 int tm\_hour;

/\* Hours since midnight: 0-23 \*/

 int tm\_mday;

/\* Day of the month: 1-31 \*/

 int tm\_mon;

/\* Months \*since\* january: 0-11 \*/

 int tm\_year; /\* Years since 1900 \*/

 int tm\_wday;

/\* Days since Sunday (0-6) \*/

 int tm\_yday;

/\* Days since Jan. 1: 0-365 \*/

 int tm\_isdst;

/\* +1 Daylight Savings Time,

 0 No DST, \* -1 don't know \*/

};

Antud kirjeldus on pärit standardteegi ‘time.h’ ühest variandist. ‘K&R’ kommentaaris tähendab viidet C-keele autoritele B. W. Kerninghanile ja D. M. Ritchiele ning nende raamatule ”The C Programming Language”.

**Näiteprogramm 10.2. Aeg.cpp**. Programm väljastab kõik andmed, mida funktsioon ‘localtime’ arvutikellast saadud info alusel ‘tm’-tüüpi struktuuri kannab.

#include <time.h>

#include <stdio.h>

int main(void)

{

 time\_t sek; // arvutiaeg sekundites

 tm \*aeg; // ajastruktuur

 char \*nadalapaev; // viit nädalapäeva nimele

 char \*nadalapaevad[]={"pühapäev","esmaspäev",

 "teisipäev","kolmapäev","neljapäev","reede","laupäev"};

 char \*kuu; // viit kuu nimele

 char \*kuud[]={"jaanuar","veebruar","märts","aprill",

 "mai","juuni","juuli","august","september",

 "oktoober","november","detsember" };

 sek=time(NULL); // arvutiaeg

 aeg=localtime(&sek); // ajastruktuuri väärtustamine

 nadalapaev=nadalapaevad[aeg->tm\_wday];

 kuu=kuud[aeg->tm\_mon];

 printf("Täna on %s, %d. %s %d.\n",

 nadalapaev,aeg->tm\_mday, kuu, 1900+aeg->tm\_year);

 printf("On aasta %d. päev.\n", aeg->tm\_yday);

 printf("Suveaeg %s\n",

aeg->tm\_isdst ? "kehtib." : "ei kehti veel/enam.");

printf("Kell on %02d:%02d:%02d.\n",

aeg->tm\_hour, aeg->tm\_min, aeg->tm\_sec);

return 0;

}

Antud programm väljastas 2012/13 aasta sügissemestri alguspäeval järgmise info:



Funktsioon ’asctime’ annab arvutiaja tekstikujul. Vorming pole meie jaoks just parim võimalikest, kuid sellegipoolest arusaadav. C ajaarvamise algusaeg ja võimalik lõpp kuvatakse nii:

 time\_t algus=0 ,maks;

 maks=(time\_t)2e31;

 printf("Ajaarvamise algus: %s",

 asctime(localtime(&algus)));

 printf("Maksimaalne sekundite arv: %ld sekundit\n", maks);

 printf("Ajaarvamine kestab kuni: %s",

 asctime(localtime(&maks)));

Kohalik aeg on kahe tunni erinev UTC ajast:



**Näiteprogramm 10.3. Logi.cpp**. Tekstifaili lugemise näiteprogrammi 6.1. on täiendatud logifaili andmete kirju­tamisega. Väljund logifaili on ingliskeelne, et sobiks paremini funktsiooni ’asctime’ väljundiga.

#include <stdio.h>

#include <time.h>

#define MAXRIDA 240

int main(int argc, char \*argv[])

{

 FILE \*fp, \*flogi;

 char rida[MAXRIDA], \*failinimi="Andmed.txt";

 char \*logifail="Logi.txt";

 time\_t sek;

 tm\*aeg;

 int veakood=0;

 flogi=fopen(logifail, "a");

 sek=time(NULL); aeg=localtime(&sek);

 fprintf(flogi,"Program started: %s", asctime(aeg));

 if(argc>1)failinimi=argv[1]; // Failinimi on ette antud käsureas

 fprintf(flogi,"File name: %s.\n", failinimi);

 fp=fopen(failinimi, "r");

 if(!fp){printf("Faili %s ei leia!\n", failinimi);

veakood=1; goto lopp;}

 while(!feof(fp)){

 rida[0]=0; // rida tühjaks

 fgets(rida, sizeof(rida), fp);

 printf(rida); // failist loetud rea väljastamine

 }

 fclose(fp);

 printf("\n"); // reavahetus (failis ei pruugi lõpurida olla)

lopp:

 fprintf(flogi,"File opening: %s.\n",

veakood ? "failed" : "success");

 sek=time(NULL); aeg=localtime(&sek);

 fprintf(flogi,"Program ended: %s", asctime(aeg));

 fclose(flogi);

 return veakood;

}

Kui programmi Logi.exe täita samuri nagu tehti programmiga TekstiFailiSisu.exe jaotises 6.1, siis logifaili Logi.txt salvestatakse:



## 10.2. Üldfunktsioonid (stdlib.h)

Üldfunktsioonide teegis stdlib.h on teiste hulgas mälu- ja teisendus­funktsioonide ning operatsioonisüsteemiga suht­lemise funktsioonide ‘system’ ja ‘exit’ prototüübid. Selles jaotises vaatame lähemalt juhuarvude funktsioone ja otsimist-järjestamist.

Funktsioon ‘rand’ annab juhusliku täisarvu vahemikus nullist kuni eel­protsessori nimega RAND\_MAX määratud väärtuseni. Et need väärtused poleks iga programmi käivitamise puhul samad, peab kasutama funktsiooni ‘srand’, mis tagab juhuarvude genereerimise alustamise juhuslikust lähtekohast.

#define RAND\_MAX 0x7FFF

int rand(void);

void srand(unsigned int);

Funktsiooni ‘srand’ parameetriks valitakse juhuslik väärtus, arvuti kella­aeg sekundites on selleks igati sobiv. Et saada juhuarv teatud vahemikus, kasutatakse tavaliselt funktsiooni ‘rand’ väärtuse ja suurima lubatud väärtuse jagamisel saadavat jääki. Kui null juhu­väärtuseks ei sobi, liidetakse jäägile veel üks. Järgmiste käskudega väljastatakse kümme juhuslikku täisarvu vahemikus ühest sajani:

srand(time(NULL));

for (int i=0; i<10; i++)printf("%4d", rand()%100+1);

Tulemus sõltub otseselt arvuti kellaajast, sest funktsiooni ‘time’ väärtus on iga sekundi järel erinev.

Massiivi järjestamine jaoks on olemas funktsioon ‘qsort’, mille nimi esitäht on saadud kiire järjestamise algoritmist (*quick*). Teine kiire meetod, kahendotsimine järjestatud massiivist, on realiseeritud funktsioonis ‘bsearch’ (*binary search*). Otsimisfunktsiooni väärtuseks on NULL, kui otsitavat väärtust massiivist ei leitud, vastasel juhul on funktsiooni väärtuseks viit leitud elemendile. Mõlema funktsiooni parameetrite seas on viit võrdlusfunktsioonile:

void\* bsearch (const void\*, const void\*, size\_t, size\_t,

 int (\*)(const void\*, const void\*));

void qsort (const void\*, size\_t, size\_t,

 int (\*)(const void\*, const void\*));

Viidata saab ka funktsioonidele, nemadki asuvad ju arvuti adresseeritavas mälus. Viitamine on vaadeldavate standard­funkt­sioonide universaalsuse huvides, võrdlusfunktsiooni väärtus otsustab, milline on suvalist tüüpi massiivi elementide õige järjestus. Funktsioon ‘yles’ annab väärtuseks 1, kui esimene täisarvuline parameeter on teisest suurem:

int yles(const void \*p1, const void \*p2){

 if(\*(int\*)p1==\*(int\*)p2)return 0;

 return (\*(int\*)p1>\*(int\*)p2) ? 1 : -1;

}

Kahanevaks sorteerimiseks pole vaja muud, kui koostada teine funktsioon, mis erineb vaid nime ja ühe võrdlustehte poolest, ning kasutada seda funktsiooni ‘qsort’ parameetrina:

int alla(const void \*p1, const void \*p2){

 if(\*(int\*)p1==\*(int\*)p2)return 0;

 return (\*(int\*)p1<\*(int\*)p2) ? 1 : -1;

}

Järjestamisfunktsioonil ‘qsort’ on parameetriteks massiivi algus­aadress, elementide arv, elemendi pikkus ning võrdlus­funktsiooni aadress. Otsimisfunktsioonil ‘bsearch’ on nimetatud parameetrite ees veel viit otsitavale väärtusele.

**Näiteprogramm 10.4. Viking.cpp**. Programm valib kuus juhuslikku mittekorduvat arvu 48-st ja võrdleb neid massiivis ‘pilet’ oleva kuue täisarvuga. Valitud arvud järjestatakse igal massiivi lisamisel funktsiooniga ‘qsort’, funktsioon ‘bsearch’ otsustab, kas funktsiooniga ‘rand’ saadud väärtus on juba valitud arvude hulgas ning hiljem ka seda, millised on kattuvad väärtused massiivides ‘pilet’ ja ‘loos’.

#include <stdio.h> // standardne sisend-väljund

#include <stdlib.h>

#include <time.h>

#define N 6

#define M 48

int main(void)

{

 int loos[N]; // juhuslikud võidunumbrid

 int pilet[]={10,11,23,27,33,36}; // meie pilet

// (TTÜ lähedal peatuvad bussid)

 int i=0, x, \*tabas;

 int yles(const void\*, const void\*);

 srand(time(NULL));

 loos[i++]=rand()%M+1; // esimene võidunumber

 do{ // kordame seni, ...

 x=rand()%M+1; // järgmine võidunumber

 if((int \*)bsearch(&x, loos, i, sizeof(int), &yles))

 continue; // see juba oli

 loos[i]=x; // seda polnud, lisame ja ...

 qsort(loos, i+1, sizeof(int), &yles); // ... järjestame

 i++;

 }while(i<N); // ... kuni kõik numbrid olemas

 printf("Valitud numbrid on:\n");

 for(i=0;i<N;i++)printf("%3d",loos[i]);printf("\n");

 printf("Pileti numbrid tabasid nii:\n");

 for(i=0;i<N;i++){

 tabas=(int \*)bsearch(&pilet[i], loos, N, sizeof(int), &yles);

 printf("%4d %s\n",pilet[i],

 (tabas!=NULL) ? "tabas" : "ei tabanud");

 }

 return 0;

}

int yles(const void \*p1, const void \*p2){

 if(\*(int\*)p1==\*(int\*)p2)return 0;

 return (\*(int\*)p1>\*(int\*)p2) ? 1 : -1;

}

Pärast mitmeid programmi käivitamisi õnnestus saada kahe taba­musega loos:



# 11. Eelprotsessor

Eelprotsessori (preprotsessori) ülesanne on programmi teksti muutmine enne tema transleerimist. Eelprotsessori käsk paikneb eraldi real, käsu võtmesõna ees on ‘#’-märk. Vajadusel saab eelprotsessori käsurida jätkata ‘\’-märgiga. Vajaminevaim eelprotsessori käsk on #include, mis lisab tekstifaili sisu transleeritavasse programmi­teksti:

#include failinimi

failinimi on standardteekide lisamisel märkide ‘<’ ja ‘>’ vahel, teiste failide puhul jutumärkides.

Lisatavaid faile saab moodustada suvalise teksti­redaktoriga. Näiteks salvestades faili lopp.h järgmised read:

#include <stdio.h>

#include <conio.h>

int lopp(int veakood)

{

 printf("Vajuta suvalisele klahvile...");

 getch();

 return veakood;

}

saab loodud faili eelprotsessori include-käsuga lisamisel lõpetada programmi täitmine funktsiooni ’lopp’ poole pöördumisega. Konsooliakent ei suleta sel juhul automaatselt, nagu mõnes teises programmeerimiskeskkonnas või ka programmi (laiendiga .exe) käivitamisel näiteks Explorerist. Malliprogramm (näide 1.1) võib olla tänu failis ‘lopp.h’ olevatele käskudele mõnevõrra lühem:

#include <lopp.h>

int main(void)

{

// muutujate deklaratsioonid

// programmi algoritm

 return lopp(0);

}

Teiseks tähtsamaks eelprotsessori ülesandeks on teksti asendamine, toimub see #define-käsuga:

#define NIMI asendustekst

‘NIMI’ on eelprotsessori muutuja nimi, teistest muutujatest eristamiseks kasutatakse nendes nimedes enamasti suurtähti. Eel­protsessori kasutamine on üheks konstantide kirjeldamise võimalustest, seda on ühes näites varem ka tehtud. Eelprotsessori käsk

#define MAXRIDA 240

tähendab seda, et programmis asendatakse iga string ‘MAXRIDA’ stringiga ‘240’. Tekstiasenduste juures ei toimu mingit keele süntaksi kontrolli, see toimub alles järgmises programmiteksti töötluse etapis. Seetõttu võivad siis väljastatavad veateated olla esimesel pilgul kummalised. Soovides teada saada, millise veateate mingi C või C++ töökeskkond sel juhul annab, võib katsetada näiteks nii, et “eksikombel” lisada eelprotsessori käsu lõppu semikoolon ja seejärel kirjeldada massiiv:

#define MAXRIDA 240;

char rida[MAXRIDA];

Code::Blocs annab massiivi deklareeriva lause kohta lausa neli veateadet, millest osa on lausa desinformeerivad:



Eelprotsessori nimedele saab lisada parameetreid ja koostada makrosid ehk asendatavaid ning muudetavaid programmiteksti lõike. Paljud lihtsad funktsioonid on kirja pandavad makrodena. **NB!** Makrode puhul ei kontrollita asendusel keelereegleid, mistõttu õige tulemuse tagamiseks tuleb hoolega läbi mõelda, milline tuleb muudetud programmi­tekst. Parameetrite kasutamisel on igati põhjen­datud soovitus kasutada avaldistes sulgusid pigem rohkem, kui hädavajalik tundub. Tähtis on see siis, kui parameetrid ise on samuti avaldised ja tehete täitmise järjekord muutub teksti asendamisel oluliseks. Näiteks järgmine makro leiab kahest parameetrist suurema ka siis, kui makro parameetrid on keerukad avaldised:

#define MAX(x,y) (((x)>(y)) ? (x) : (y))

Makrode kasutamise alternatiiviks on *inline*-funktsioon. See on reeglina nii lühike, et vastavad käsud on translaatoril mõistlik paigutada tulemuskoodi iga funktsiooni väljakutse kohas nii nagu makrodki. Järgnevalt on kaks suhteliselt sarnast ülesannet täitvat tekstilõiku. Funktsioon ‘ishinne’ on tõene (1), kui parameeter on vahemikus 0...5. Makro POSIT asendatakse programmis tingimus­lausega, mille väärtus on üks siis, kui parameetri väärtus on ühest suurem:

inline int ishinne(int p) {return (p>=0 && p<=5) ? 1 : 0;}

#define POSIT(p) ((p>1) ? 1 : 0)

Järgmises korduses muutub tsüklimuutuja nullist kuueni. Hinne väljastatakse vaid siis, kui funktsioon ‘ishinne’ on tõene:

 for(int i=0; i<7; i++)

 if(ishinne(i))printf("%d - %s\n", i,

 POSIT(i) ? "sooritatud":"mittesooritatud");

Makro ‘POSIT’ aitab otsustada, kumba teksti väljastada:



Eelprotsessor võimaldab tingimuslikku transleerimist. Transleerita­vasse programmiteksti võetakse laused vaid siis, kui etteantud tingimus on tõene.

#if tingimus1

need laused lähevad programmiteksti, kui tingimus1 on tõene

#elif tingimus2

need laused lähevad programmiteksti, kui tingimus2 on tõene

...

#else

need laused lähevad programmiteksti, kui ükski varasem tingimus polnud tõene

#endif

Tavaliselt kontrollitakse tingimuses seda, kas mingi eelprotsessori nimi on deklareeritud või mitte:

#ifdef nimi

ehk #if defined nimi – tõene, kui eelprotsessori nimi on deklareeritud;

#ifndef nimi

 ehk #if not defined nimi – tõene, kui eelprotsessori nimi on deklareerimata.

Eelprotsessori nime deklaratsiooni on võimalik tühistada:

#undef nimi

Standardfunktsioonide teekides kasutatakse tingimuslikku translee­rimist päris sageli. Nii kontrollitakse enamus lisatavates failides kohe alguses seda, kas see fail on juba varem lisatud. Korduv lisamine põhjustab enamasti vigu. Näiteks failis stdio.h kasutatakse eelprotsessori nime ‘\_STDIO\_H\_’ tuvastamaks, kas faili sisu on juba lisatud. Allkriipsud nimedes on selleks, et oluliselt vähendada juhus­likku kokkulangevust programmis kasutatavate nimedega:

#ifndef \_STDIO\_H\_

#define \_STDIO\_H\_

faili põhisisu

#endif /\* \_STDIO\_H\_ \*/

Eelmises jaotises kirjeldatud teegis ‘time.h’ kontrollitakse andmetüübi ‘clock\_t’ kirjelduse olemasolu järgmiste eelprotsessori käskudega:

#ifndef \_CLOCK\_T\_DEFINED

typedef long clock\_t;

#define \_CLOCK\_T\_DEFINED

#endif

# 12. Veatöötlus

Vaatamata programmi põhjalikule silumisele võivad sellegipoolest tekkida vead ja seda eelkõige valede lähte­andmete tõttu. Programmi kasutaja kõiki võimalikke eksimusi on päris raske ette näha. Vigade avastamiseks ja nende tagajärgede vähendamiseks on veatöötlus­vahendid. Vea­töötlusega programmiploki ette kirjutatakse võtmesõna try:

try {programmikäsud}

Käskude täitmine toimub veakoodi väärtus­tamiseni (tavaliselt plokis väljakutsutavas funktsioonis). Vea tekkimisel “püütakse” ta kinni catch-lausega:

catch (andmetüüp){programmiplokk}

Plokki kuuluvad laused täidetakse näidatud tüüpi veakoodi puhul. Tavaliselt on veakood täisarv. Ilmnenud veast teatamine väljakutsunud funktsioonile toimub võtmesõnaga throw:

throw veakood;

Veatöötluse näiteks on funktsioon ‘minutid’, mille parameetriks on kellaaeg stringina kujul “hh:mm” ja väärtuseks aeg minutites päeva algusest. Funktsioon avastab viis võimalikku viga ja annab neist teada järgmiste veakoodidega:

1 – vale märk,

2 – vale (või vales kohas) eraldaja tundide ja minutite vahel,

3 – tundide arv vale (üle 23),

4 – sisestusreas ülaaru märke,

5 – minutite arv vale (üle 59).

int minutid(char \*p){

 int tund=-1, minut=-1;

 if(\*p>='0' && \*p<='9')tund=\*p-'0'; else throw 1;

 p++;

 if(\*p>='0' && \*p<='9'){tund\*=10; tund+=(\*p-'0'); p++;}

 // tundide arv kahekohaline

 if(\*p!=':')throw 2;

 if(tund <0 || tund>24)throw 3;

 #ifdef TEST

 printf("Tunnid - %d\n", tund);

 #endif

 p++;

 if(\*p>='0' && \*p<='9')minut=\*p-'0'; else throw 1;

 p++;

 if(\*p>='0' && \*p<='9'){minut\*=10; minut+=(\*p-'0'); p++;}

 if(\*p)throw 4; // ülearu märke

 if(minut <0 || minut>59)throw 5;

 #ifdef TEST

 printf("Minutid - %d\n", minut);

 #endif

 return tund\*60+minut;

}

Funktsioonis ja ka järgnevas näiteprogrammis on silumiseks kasutatud väljundlauseid, mis väljastavad silumisinfot sõltuvalt programmi testimistasemest. Väljastuslausete kasutamise puuduseks on asjaolu, et nende lisamine ja eemaldamine on suhteliselt tüütu tegevus. Halvaks stiiliks on silumislausete väljakommenteerimine programmi­tekstist, oluliselt parem on silumismuutuja kasutuselevõtmine ja väljastuslause(te) paigutamine tingimuslausesse:

if(test){printf(...);}

Muutuja ‘test’ (see pole võtmesõna, vaid tavaline muuuja nimi) väärtus otsustab, kas väljastada silumisinfot või mitte. Deklaratsioon kujul:

int test=1;

tähendab, et silumisinfot väljastatakse. Algväärtuse asendamisel nulliga silumisinfot ei kuvata. Antud lahenduse puuduseks on asjaolu, et silumislaused jäävad programmi transleeritud teksti (võtavad mälu) ja silumismuutuja väärtust testitakse igal programmi täitmisel (kulutab aega). Eelprotsessori tingimuslik transleerimine lubab kerge vaevaga lisada või eemaldada silumislaused vaid ühe rea abil:

#define TEST

Silumislaused ümbritsetakse eelprotsessori käskudega #ifdef ja #endif:

#ifdef TEST

printf(...);

#endif

Selliselt on toimitud funktsioonis ’minutid’ – kui eelprotsessori muutuja ’TEST’ on defineerimata, siis tundide ja minutite väärtust ei kuvata. Silumislausete täielikuks väljajätmiseks programmist piisab #define-käsu eemal­damisest ja uuest translee­rimisest.

**Näide 12.1. Vead.cpp.** Testimisprogramm kasutab lauseid try ja catch, et püüda kinni funktsioonis ‘minutid’ avastatavad vead. Nimetatud funktsioon kontrollib, kas parameetrina esitatud andmed on tõesti kujul ‘hh:mm’ ning kas tundide ja minutite arv on loogiline. Täpitähtede kuvamiseks kasutatakse lisas 3 olevat funktsiooni ‘kuva’. Muutuja ‘test’ väärtus määrab, kas kuvada silumisinfot, funktsioonis ‘minutid’ teeb seda eelprotsessori nimi ‘TEST’.

#include <iostream>

using namespace std;

int test=0;

int main()

{

 char rida[100];

 int algus, lopp, vahe; // ajad minutites

 int viga; // veakood

 int minutid(char\*);

 char \*vead[]={"OK", "Vale märk",

"Puudub koolon (või on vales kohas) ",

 "Tundide arv vale", "Ülearu märke", "Minutite arv vale"};

 do{

 viga=0;

 cout << "Sisesta algusaeg kujul tt:mm > " ;

 gets(rida);

 try{

 algus=minutid(rida);}

 catch(int kood){ cout << vead[kood] << endl; viga=1;};

 }while(viga);

 if(test)printf("%algus = %d.\n", algus);

 do{

 viga=0;

 cout << kuva("Sisesta lõpuaeg kujul tt:mm > ") ;

 gets(rida);

 try{

 lopp=minutid(rida);}

 catch(int kood){ cout << vead[kood] << endl; viga=1;};

 }while(viga);

 if(test)printf("lopp = %d.\n", lopp);

 vahe=lopp-algus;

 printf("Sisestatud aegade vahe on %d:%02d.\n",

vahe/60, vahe%60);

 return 0;

}

Nagu näha, pole kellaaegade sisestamisega kuigivõrd võimalik eksida:



Kui programmis muuta ära muutuja ’test’ väärtus ja defineerida eelprotsessori nimi ’TEST’, kuvatakse märga­tavalt rohkem infot programmi töö kohta:



# 13. Objektiklassid

Kui andmestruktuuri kirjeldus algab võtmesõna struct asemel võtmesõnaga class, tähendab deklaratsioon objekti­klassi kirjeldamist. Lisaks andmetele on klassi koosseisus ka funktsioonid, objektorienteerituse mõistes meetodid. Keele­reeglitest palju tähtsam on aga objekt­orienteeritud lähenemis­viis programmi koostamisel. See on mahukas ja üksjagu keeruline valdkond, mille jaoks on eraldi õppeained. Seetõttu klasside kirjeldamise, klassi objektide loomise ja kasutamsie reegleid vaadatakse siin vaid põgusalt.

## 13.1 Objektiklassi deklareerimine

Objektiklassi koosseisus on andmete ja meetodite vaikimisi juurdepääsuviis privaatne (private) ehk neid saab kasutada vaid klassisiseselt. Et kirjeldatud klassist oleks kasu ka mujal programmis, peavad vähemalt mõned klassi meetodid olema avaliku (public) juurdepääsuga. Lihtsaim objektiklass kirjeldatakse järgneva struktuuriga:

class Klass{

 andmed ja meetodid

public:

 meetodid

};

Objekti deklareerimine toimub nagu iga teisegi muutuja deklareerimine, andmetüübiks on klassi nimi:

Klass objekt;

Luuakse objektiklassi kuuluv objekt. Selle loob programmimoodul nimega konstruktor, objekti kustutab destruktor. Konstruktor luuakse ka automaatselt, kuid enamasti soovitakse ise kirjeldada objekti konstruktor, millel on objekti loomiseks vajalikud andmed edastatud para­meetritena. Konstruktori nimi peab olema sama, mis klassi nimi. Destruktoril on samuti sama nimi, mille ees on veel märk ~, mis C/C++ keeles tähendabki eitust. Konstruktoril ja destruktoril puudub väärtus, ka võtmesõna void pole nende ees lubatud. Konstruktoreid (sama nimega) võib olla mitu, kui nad on erinevate parameetritega.

Konstruktorid ja meetodid võivad olla struktuuri deklaratsioonis (*inline*) või järgneda deklaratsioonile iseseisvana koos skoobimääranguga (::):

Klass::meetod(parameetrid){...}

Õpetlik on luua oma versioon klassist ‘string’, milles on kõik vajalik olemas, ülearust on välditud ja lisatud on oskus muuta täpitähtede koode. Minimaalne versioon sellisest klassist sisaldab konstruktori, mis teeb parameetrina etteantud tekstist koopia ning meetodid teksti pikkuse ja sisu väljastamiseks:

class Tekst{

 char \*xsisu;

 int xpikkus(char \*p)

 {int n=0; while(\*p++)n++; return n;}

 char \*kuva(char\*);

public:

 Tekst(char \*p){

 char \*temp;

 xsisu=new char [xpikkus(p)+1];

 temp=xsisu;

 while(\*p)\*temp++=\*p++;

 \*temp=0;}

 int pikkus(void){return xpikkus(xsisu); }

 char \*orig(void){return xsisu;}

 char kuva(void){return kuva(xsisu);}

};

Andmed on reeglina privaatse juurdepääsuga ja nende muutmine toimub vaid vastavaid meetodeid kasutades. Andmete sellist kaitsmist nimetataske kapseldamiseks (*encapsulation*). Tänu andmete kapseldamisele saab vajadusel andmete struktuuri muuta, jättes objekti liidese endiseks. Näiteks objektiklassi ‘Tekst’ kasutaja ei pea teadma, kas teksti pikkus on salvestatud mõnesse andmeelementi või arvutatakse igal korral uuesti, kui meetodi ‘pikkus’ poole pöördutakse.

Meetodi ‘kuva’ realisatsiooni võib tema pikkuse tõttu kirja panna väljaspool klassi, sel juhul algab funktsioon skoobimääranguga:

char\* Tekst::kuva(char \*tekst)

ülejäänud osa on täpselt sama, mis viimases lisas toodud funktsioonil. Peaprogrammis saab kirjeldada klassi Tekst objekti ja väljastada teda originaalis ja muudetud kujul. Meetodi poole pöördumisel eraldatakse objekti nimi ja meetodi oma punktiga:

int main()

{

 Tekst linnaosa("Väike-Õismäe");

 cout << linnaosa.orig() << endl;

 cout << linnaosa.kuva() << endl;

}



Objektide poole pöördutakse sageli viida kaudu, mis dekla­reeritakse nii:

Klass \*viit;

Objekti sel juhul ei looda ega konstruktori parameetrite väärtusi ei nõuta. Neid vajatakse alles siis, kui objekt luuakse new käsuga:

viit=new Klass;

Viida väärtustamine võib toimuda ka olemasoleva objekti aadressiga:

viit=&objekt;

Viitade kasutamisel on objekti meetodite poole pöördumine selline:

viit->meetod(parameetrid);

Objekti kustutamine toimub käsuga delete, mille tulemusena täidetakse objekti destruktor. Näiteks sellised laused on C++ keele süntaksireeglite kohaselt kirja pandud:

 Tekst \*koht;

 koht=new Tekst("Vääna-Jõesuu");

 cout << koht->kuva() << endl;

 delete koht;

## 13.2.Tehete üledefineerimine

Objekti meetodite kasutamine pole keeruline, ent pisut tülikas. C++ lubab meetodeid välja kutsuda ka teisiti, tehtemärkide abil. Arvestada tuleb küll sellega, et operandide arvu ega järjestust muuta ei saa. Tehtena väljakutsutav meetod deklareeritakse võtmesõna operator abil nii:

andmetüüp operator tehtesümbol(parameetrid);

Kui tegevuse jaoks leida sobiv tehe, saab luua elegantse liidesega objektiklasse. Klassi ’string’ objektile saab väärtuse omistada, tema väärtust võrrelda ja tekste sidurdada vastavalt tehetega ’=’, ’==’ ka ’+=’. Selliste tegevuste lisamine klassi ’Tekst’ pole kuigi raske. Hea on luua veel teine objektiklassi konstruktor, mis tekitab tühja tekstistringi. Nii on lihtsam objektiklassi kasutajal kui ka uute meetodite loojal.

 Tekst(void){xsisu=new char[1]; \*xsisu=0;}

Ülaltoodud tehete realisatsioon on selline:

 void operator = (char \*p){

 Tekst uus(p);

 xsisu=uus.orig();}

 int operator ==(const char \*p){

 int i=0;

 while(\*(xsisu+i)){

 if (\*(xsisu+i)!=\*(p+i))return 0; // erinev märk

 i++;}

 return 1; // kõik märgid on samad

 }

 void operator += (char \*p){

 char \*temp, \*p1, \*p2;

 temp=new char[pikkus()+xpikkus(p)+1];

 p1=temp; p2=xsisu;

 while(\*p2){\*p1 = \*p2; p1++; p2++;} // algne tekst

 while(\*p){\*p1 = \*p; p1++; p++;} // lisame teise teksti

 p1=0; // lõpetame uue teksti nullbaidiga

 xsisu=temp;

 }

Kui vaadate tehtemärkide loendit lisas 1, siis võib mõne­legi tehtemärgile leida mõistliku sisu. Näiteks &-tehtega võiks leida etteantud märgi positsiooni tekstis (0, kui sellist märki seal pole) ning tehe /= võib näiteks eraldada tekstist teatud arvu märke:

int operator &(char otsi){

 int i=0;

 char \*p=xsisu;

 while(\*(p+i)){

 if(\*(p+i)==otsi)return i+1; // leidsime märgi

 i++;}

 return 0; // sellist märki tekstis pole

 }

 void operator /=(int n) {

 \*(xsisu+n)=0; // katkestame teksti selles kohas

 }

Järgnevad käsud kasutavad klassi ‘Tekst’ üledefineeritud tehteid:

 Tekst nimi, tervitus ;

 nimi="Mihkel";

 tervitus=nimi;

 tervitus+=", tere tulemast!";

 cout << tervitus.kuva() << endl;

 if(nimi=="Mihkel")cout << "Oled " << nimi.kuva() << endl;

 if(nimi=="Jaanus")cout << "Oled " << nimi.kuva() << endl;

 else cout <<"Ei ole Jaanus\n";

 Tekst meiliaadress("info@ttu.ee");

 int atpos;

 atpos=meiliaadress&'@';

 if(atpos)meiliaadress/=(atpos-1);

 cout << "Kasutajanimi on " << meiliaadress.kuva() << endl;

Tulemus vastab oodatule:



## 13.3. Uute klasside tuletamine

Olemasolevast klassist saab tuletada uusi klasse. Tuletatud (*derived*) klass saab endale ühtlasi kõik baasklassi andme­elemendid ja meetodid. Seda ülekandumist nimetetakse pärimiseks (*inheritance*). Tuletatud klass deklareeritakse koos juurdepääsuviisiga:

class Uusklass: [juurdepääsuviis] Baasklass{

...

};

Juurdepääs baasklassi liikmele ehk komponentidele on määratud kahe juurdepääsuviisidega private ja public. Andmete kapseldamise põhimõte nõuab, et juurdepääs objekti and­metele toimub kontrollitult ehk maksimaalselt piiratult. Baasklassi komponendile omistatud juurdepääsuviis protected edastab juurdepääsu sellele ka tuletaud klassis. Alljärgnevas tabelis on esimeses veerus näidatud tuletatud klassi definitsioonis näidatud juurdepääsuviis baasklassile, teises baasklassi liikme juurdepääsuviis ning kolmandas see, kas ja kuidas saab baasklassi komponenti kasutada.

|  |  |  |
| --- | --- | --- |
| **Baasklass** | **Baasklassi liige** | **Juurdepääs tuletatud klassis** |
| public | public | public |
| public | protected | protected |
| public | private | - |
| private | public | private |
| private | protected | private |
| private | private | - |

Tuletatud klassi konstruktor võib välja kutsuda baasklassi sobiva konstruktori, näiteks nii:

Uus(int p1, int p2):Baas(p1){...}

Baasklasse võib ühel tuletatud klassil keelereeglite järgi olla ka mitu.

**Näide 13.1. Kaubad.cpp.** Kirjeldatud on klass Kaup ja temast on tuletatud klass Sooduskaup. Objekti loomisel näidatakse hind ja kaal grammides, sooduskauba puhul veel soodustuse protsent. Kilohinna arvuta­miseks peab tuletatud klassis olema juurdepääs kauba hinna ja kaalu andmetele, selle annab võtmesõna protected.

#include <iostream>

#include <string.h>

using namespace std;

class Kaup{

protected:

 string xnimetus;

 float xkaal;

 float xhind;

public:

 Kaup (char \*p1, float p2, float p3){

 xnimetus=p1;

 xkaal=p2/1000;

 xhind=p3;

 }

 float kilohind(void){ return xkaal? xhind/xkaal : 0.0; }

 const char \*nimetus(void){return xnimetus.data();}

 float hind(void){return xhind;}

};

class Sooduskaup: public Kaup{

 float xsoodus;

public:

 Sooduskaup(char \*p, float p2, float p3, float p4):

 Kaup(p,p2,p3){

 xsoodus=p4/100;

 }

 float uushind(void){return xhind-xhind\*xsoodus;}

};

int main()

{

 Kaup kaup1("Lihakas sardell", 360, 2.39);

 Sooduskaup kaup2("Verivorst suitsulihaga", 500, 1.95, 33);

 cout << kaup1.nimetus() << ". Hind: " << kaup1.hind() <<

 "; " << kaup1.kilohind() << " EUR/kg. " << endl;

 cout << kaup2.nimetus() << ". Hind: " << kaup2.uushind() <<

 " Tavahind: " << kaup2.hind() << endl;

 return 0;

}

Kaupadeks on 360-grammine sardelli- ja poolekilone soodus­hinnaga verivorstipakend:



Vajadusel laiendatakse juurdepääsu privaatsetele andmetele võtmesõnaga friend. Erakorralise juurde­pääsu võivad saada nii funktsioonid kui objektide klassid.

**Näide 13.3. Arvutiaeg.cpp**. Klassi Arvutiaeg loomisel salvestatakse arvuti kellalt teadasaadud tunnid ja minutid vastavatesse privaatsetesse muutujatesse. Et neid saaks cout-väljundi abil väljastada, tuleb klassi ostream tehe << kuulutada “sõbraks”:

#include <iostream>

using namespace std;

class Arvutiaeg{

#include <time.h>

 int xtund, xminut;

 char\* tekstina(void){

 char \*t=new char[6];

 sprintf(t,"%02d:%02d", xtund, xminut);

 return t;

 }

public:

 Arvutiaeg(void){

 time\_t sek;

 tm \*aeg;

 sek=time(NULL);

 aeg=localtime(&sek);

 xtund=aeg->tm\_hour;

 xminut=aeg->tm\_min;

 }

 friend ostream& operator << (ostream&, Arvutiaeg&);

};

ostream& operator << (ostream&o, Arvutiaeg &p){

 o<< p.tekstina();

 return o;

}

int main(){

 Arvutiaeg kell;

 cout << "Kell on " << kell << "." << endl;

 return 0;

}

## 13.4. Staatilised klassiliikmed

Staatilised võivad olla nii andmed kui meetodid. Andme­elemendist luuakse vaid üks, kõigi klassiobjektide jaoks ühine eksemplar. Staatiline andmeelement eksisteerib ka siis, kui ühtegi selle klassi objekti pole loodud.

Staatiline klassiliige deklareeritakse võtmesõnaga static. Selles võidakse hoida näiteks loodud objektide arvu. Algväärtustamine nõuab skoobimäärangu kasutamist, näiteks:

int Klass::n=0;

Võtmesõna this viitab klassi objektile ning teda kasutatakse vajadusel viidata meetodi käivitanud objektile.

**Näide 13.4. Kysimused.cpp**. Klass Kysimus sisaldab küsimuse muutuvat osa ja vastust ning meetodit ‘vordle’ vastuse võrdlemiseks. Staatilised klassiliikmed sisaldavad ühtse vormingu kõigi küsimuste esitamiseks, loodud objektide arvu ning abil moodustatakse küsimustest list, kus iga küsimus viitab järgmisele. Listi läbimiseks on staatilised viidad, mis sisaldavad küsimustiku esimese, viimase ja esitatava küsimuse aadresse. Peaprogrammis defineeritakse küsimused, esitatakse need ja leitakse õigete vastuste protsendi.

#include <string.h>

#include <iostream>

using namespace std;

class Kysimus{

 static int xn; // objektide arv

 static Kysimus \*xesimene, \*xviimane, \*xesitatav;

 // viidad esimesele ja viimasele küsimusele

 static char \*xvorming;

 // sama vorming kõigile küsimsutele

 char xkysi[80]; // küsimuse muutuv osa

 char xvastus[20]; // õige vastus

 Kysimus \*xjargmine;

public:

 Kysimus(char \*p1, char \*p2){

 strcpy(xkysi, p1); strcpy(xvastus, p2);

 if(!xn)xesimene=this; else xviimane->xjargmine=this;

 xn++;

 xviimane=this;

 }

 Kysimus\* alusta(void){

 xesitatav=xesimene;

 printf(xvorming, xesitatav->xkysi);

 return xesitatav;}

 Kysimus\* jargmine(void){

 xesitatav=xesitatav->xjargmine;

 if(!xesitatav)return NULL;// küsimused on läbi

 printf(xvorming, xesitatav->xkysi);

 return xesitatav;}

 int vordle(char \*p){return !strcmp(xvastus, p);}

 static int kysimusi(void){return xn;}

};

int Kysimus::xn=0;

Kysimus \*Kysimus::xesimene=NULL;

Kysimus \*Kysimus::xviimane=NULL;

Kysimus \*Kysimus::xesitatav=NULL;

char \*Kysimus::xvorming="\n%s.\nMis on teaduskonna tahis > ";

int main()

{

 Kysimus \*kysi;

 int ok=0;

 char vastus[20];

 kysi=new Kysimus("Energeetikateaduskond", "A");

 kysi=new Kysimus("Majandusteaduskond", "T");

 kysi=new Kysimus("Ehitusteaduskond", "E");

 kysi=new Kysimus("Infotehnoloogia teaduskond", "I");

 kysi=new Kysimus

("Keemia- ja materjalitehnoloogia teaduskond", "K");

 printf("Kokku on kysimusi %d.\n", Kysimus::kysimusi());

 kysi=kysi->alusta();

 do{

 gets(vastus);

 if (kysi->vordle(vastus))ok++; // oli õige vastus

 printf("Vastus %s on %s vastus.\n",

 vastus, kysi->vordle(vastus) ? "6ige " : "vale");

 }while(kysi=kysi->jargmine());

 printf("6igete vastuste protsent: %5.1f.\n",

 (float)ok\*100/Kysimus::kysimusi());

 return 0;

}

Programmis omistatakse staatilisele klassiliikmele ’xvorming’ sisu, mille abil küsitakse teaduskondade tähistust. Vastates igale küsimusele eeldusel, et selleks tähistuseks on teadus­konna nime esitäht, saame suhteliselt tagasihoidliku tulemuse:



## 13.5 Abstraktne objektiklass.

Baasklassi võib deklareerida kavatsusega kasutada teda vaid uute klasside tuletamise jaoks, määrates ära nende liidese kohustuslikud komponenedid. Võtmesõnaga virtual kirjeldatud virtuaalne meetod nõuab, et vastav komponent oleks tuletatud klassis realiseeritud.

**Näide 13.5. HTML.cpp.** Elementaarne veebilehe kujun­dus­programm sisaldab virtuaalse klassi Html, millest tuletatakse kõik veebilehel kuvatavate objektide klassid. Neist kolm (tekst, pilt ja viide) on realiseeritud näiteprogrammis. Virtuaalne funktsioon ‘htmltekst’ nõuab, et iga tuletatud klass oskaks lisada sellenimelise meetodiga ennast kuvava teksti HTML-keeles veebilehe faili.

#include <iostream>

using namespace std;

class Html{

protected:

 string xtekst;

public:

 virtual const char \*htmltekst(void)=0;

};

class Tekst:public Html{

public:

 Tekst(char \*p){xtekst=p;}

 const char \*htmltekst(void){

 string t;

 t="<p>"; t+=xtekst; t+="<br>";

 return t.data();

 }

};

class Pilt:public Html{

public:

 Pilt(char \*p){xtekst=p;}

 const char \*htmltekst(void){

 string t;

 t="<img src="; t+=xtekst; t+="><br>";

 return t.data();}

};

class Link:public Html{

 string xaadress;

public:

 Link(char \*p1, char \*p2){xtekst=p1; xaadress=p2;}

 const char \*htmltekst(void){

 string t;

 t="<a href=";

 t+=xaadress; t+=">";

 t+=xtekst;

 t+="</a><br>";

 return t.data();}

};

int main()

{

 FILE \*fp;

 fp=fopen("Veebileht.html", "w");

 Html \*obj[10];

 int n=0;

 obj[n++]=new Tekst("Tere tulemast meie kodulehele!");

 obj[n++]=new Tekst("Selline on TTÜ maskott");

 obj[n++]=new Pilt("c:\\cc++\\julius.gif");

 obj[n++]=new Link("Täpsem info", "http://www.ttu.ee");

 obj[n++]=new Tekst("Tulge teinegi kord!");

 fprintf(fp, "<html><title>KODULEHT</title><body>");

 for(int i=0;i<n;i++){

 fprintf(fp, "%s\n", obj[i]->htmltekst());

 }

 fprintf(fp, "</body></html>");

 fclose(fp);

 return 0;

}

Programmi töö tulemusena salvestatakse faili Veebileht.html järgmine tekst:

<html><title>KODULEHT</title><body><p>Tere tulemast meie kodulehele!<br>

<p>Selline on TTÜ maskott<br>

<img src=c:\cc++\julius.gif><br>

<a href=http://www.ttu.ee>Täpsem info</a><br>

<p>Tulge teinegi kord!<br>

</body></html>

mis veebibrauseris kuvatuna näeb välja selliselt:



# Lisad

## Lisa 1. Tehted ja nende täitmise järjekord prioriteetide tabelina

|  |  |  |  |
| --- | --- | --- | --- |
| **Priori­teedi tase** | **Tehte-märk** | **Tehe** | **Tehe inglise keeles** |
| 1. | () | Funktsiooni väljakutse | Function call |
|  | [] | Massiivi indeks | Array subscript |
|  | -> | Struktuuri element | C++ indirect component selector |
|  | :: | Skoobi määramine | C++ scope access/resolution |
|  | . | Liitnime eraldaja | C++ direct component selector |
| 2.  | ! | Loogiline eitus | Logical negation (NOT) |
|  | ~ | Loogiline eitus bitthaaval | Bitwise (1's) complement |
|  | + | Unaarne pluss | Unary plus |
|  |  - | Unaarne miinus | Unary minus |
|  | ++ | Suurendamine | Preincrement or postincrement |
|  | -- | Vähendamine | Predecrement or postdecrement |
|  | & | Aadress | Address |
|  | \* | Viitamine | Indirection |
|  | sizeof | Mälu suurus | (returns size of operand, in bytes) |
|  | new | Mälu eraldamine | (dynamically allocates C++ storage) |
|  | delete | Mälu vabastamine | (dynamically deallocates C++ storage) |
| 3. |  \* | Korrutamine | Multiply |
|  | / | Jagamine | Divide |
|  | % | Jagamise jääk | Remainder (modulus) |
| 4. | .\*  |  | C++ dereference |
|  | ->\* |  | C++ dereference |
| 5. | + | Liitmine | Binary plus |
|  | – | Lahutamine | Binary minus |
| 6. | << | Bittide nihutamine vasakule | Shift left |
|  | >> | Bittide nihutamine paremale | Shift right |
| 7. | < | Väiksem | Less than |
|  | <= | Väiksem-võrdne | Less than or equal to |
|  | > | Suurem | Greater than |
|  | >= | Suurem-võrdne | Greater than or equal to |
| 8. | == | Võrdne | Equal to |
|  | != | Mittevõrdne | Not equal to |
| 9. | & | Loogiline korrutamine bitthaaval | Bitwise AND |
| 10. | ^ | Välistamine bitthaaval | Bitwise XOR |
| 11. | | | Loogiline liitmine bitthaaval | Bitwise OR |
| 12. | && | Loogiline korrutamine | Logical AND |
| 13. | || | Loogiline liitmine | Logical OR |
| 14.  | ?: | Tingimustehe | (a ? x : y means "if a then x, else y") |
| 15. | = | Omistamine | Simple assignment |
|  | \*= | Läbikorrutamine | Assign product |
|  | /= | Läbijagamine | Assign quotient |
|  | %= | Jäägi omistamine | Assign remainder (modulus) |
|  | += | Juurdeliitmine | Assign sum |
|  | –= | Mahalahutamine | Assign difference |
|  | &= | Loogilise korrutise omistamine | Assign bitwise AND |
|  | ^= | Loogilise välistamise omistamine | Assign bitwise XOR |
|  | |= | Loogilise liitmise omistamine | Assign bitwise OR |
|  | <<= | Nihke omistamine | Assign left shift |
|  | >>= | Nihke omistamine | Assign right shift |
| 16. | , | Avaldiste eraldaja | Evaluate |

##

## Lisa 2. Vorminguelemendid

Vormingut kasutatakse parameetrina funktsioonides, mille nime koosseisus on kas ‘printf’ või ‘scanf’, nagu näiteks ’fprintf’. Vorminguelement algab ‘%’-märgiga ja lõpeb andme­tüübi tähisega, milleks on:

c – märk;

d(i) – täisarv;

e, f, g – murdarv;

o – kaheksandsüsteemi arv;

p – aadress;

s – viit tekstistringile;

u – märgita täisarv;

x – 16ndsüsteemi arv.

Märk ‘l’ tüübitähise ees tähendab andmetüübi pikemat varianti (long, double). Murdarvu on võimalik esitada astme abil (tüübiks e) või komakohaga (f). Vorminguelemendi ‘g’ puhul valitakse loetavam esitusviis sõltuvalt murdarvu väärtusest.

‘%’-märgi ja tüübi tähise vahel võivad olla täpsustused, mida kasuta­takse eelkõige andmete väljastamisel. Täisarvuga saab näidata väljund­välja minimaalse pikkuse, lisades peale punkti teise täisarvu näitab see komakohtade arvu. Muud vormindusmärgid:

- – joondamine vasakule;

+ – märgi kuvamine alati, ka positiivse väärtuse puhul;

0 – väärtuse ees on pikema väljundvälja korral nullid.

Järgnevas programmis on deklareeritud viit tüüpi muutujad ja väljastatud nad erinevate vormingutega.

#include <stdio.h>

int main()

{

 int i=12345;

 float f1=123.45;

 double f2=2e15; // kaks kvadriljoni

 char c='A';

 char \*t="ABC";

 printf("%c.\n",c);

 printf("%d.\n",i);

 printf("%8d.\n",i);

 printf("%08d.\n",i);

 printf("%-8d.\n",i);

 printf("%+8d.\n",i);

 printf("%-8d.\n",i);

 printf("%f.\n",f1);

 printf("%10.2f.\n",f1);

 printf("%-10.2f.\n",-f1);

 printf("%e.\n",f1);

 printf("%g.\n",f1);

 printf("%g.\n",f2);

 printf("%p.\n",t);

 printf("%s.\n",t);

 printf("%5s.\n",t);

 printf("%-5s.\n",t);

 printf("%o.\n",c);

 printf("%x.\n",c);

 return 0;

}

Väljastus on selline:



## Lisa 3. Funktsioon ‘kuva’ failis tapid.h

Funktsioon teeb parameetrina edastatud tekstist koopia ja asendab täpitähtede koodid sobivaks konsooliakna jaoks.

char\* kuva(char \*tekst)

{

 char \*p, \*algus;

 char \*np, \*uus;

 int len=0;

 algus=p=tekst;

 while(\*p){len++;p++;} // teksti pikkus

 np=uus=new char[len+1];

 p=algus;

 while(\*p){

 switch (\*p){

 case 'õ' : \*np=147; break;

 case 'ä' : \*np=132; break;

 case 'ö' : \*np=148; break;

 case 'ü' : \*np=129; break;

 case 'Õ' : \*np=147; break;

 case 'Ä' : \*np=142; break;

 case 'Ö' : \*np=153; break;

 case 'Ü' : \*np=154; break;

 default: \*np=\*p;

 }

 p++;np++;

 }

 \*np=0;

 return uus;

}