Algoritmid ja andmestruktuurid
IFI6228.DT
6 EAP

Kursuse ülesehitusest

Neli tundi nädalas:

- 2x45 min loengut või harjutust kõigile korraga;
- 2x45 min praktikumi eraldi rühmades.

Lõppeb kirjaliku eksamiga:

- paberil – teooriaküsimused + ülesannete lahendamine;
- eksami eelduseks on kontrolltöö semestri teises pooles, millega näitate oma arusaamist C-keelest.

Kogu hinne tuleb eksamist.

Kursuse sisust

Tutvus mitmete huvitavate andmestruktuuride ja algoritmidega:

- pinu, järjekord, erinevad puud, graaf, paisktabel;
- sorteerimine, otsimine, puu- ja graafialgoritmid;
- algoritmimise strateegiad ja algoritmide keerukus.

Tutvus C-keelega, sh viitmuutuja mõiste ja kasutamisega, mis võiks natuke rohkem panna mõtlema sellele, mis arvuti sees toimub.

Materjalid

Kursuseprogrammist leiate kaks raamatut:

V. Leppikson, *Programmeerimine C keeles.* 1997

J. Kiho, *Algoritmid ja andmestruktuurid.* 2003.

Ka järgnevad on huvitavad:

R. Sedgewick, *Algorithms in C* (Parts 1-5). 2002.

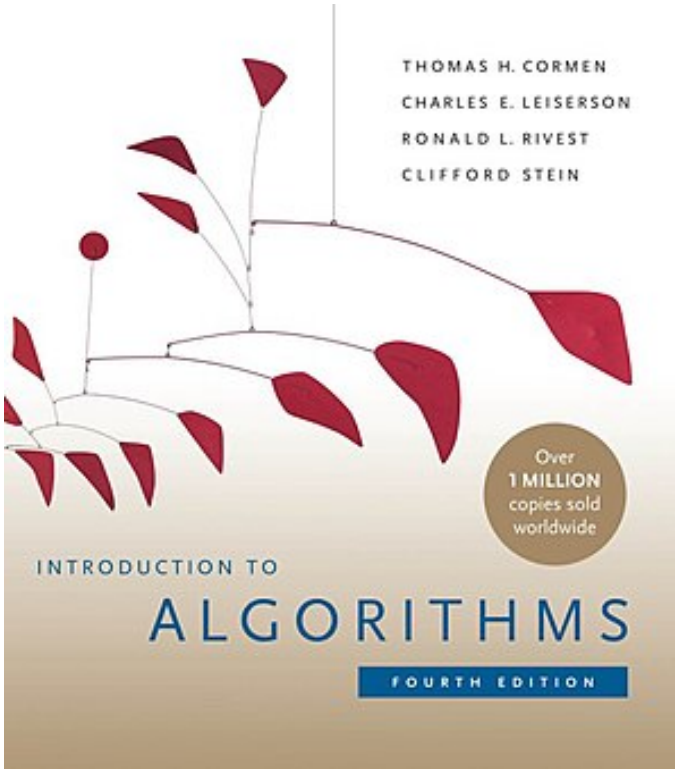
A. Isotamm, *Programmeerimine C-keeles ("Algoritmide ja andmestruktuuride" näiteil).* TÜ, 2009

Klassikalised raamatud

Donald Knuth, *The Art of Computer Programming* Volumes 1-3 (1968 – 1973), Volume 4: 2008

T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, First Ed 1990 (Fourth Ed 2022) MIT õpik (viidatakse kui CLRS)

Ja internetist leiab lõpmata palju materjali.



Soovitused ellujäämiseks

Tutvu enne loengut materjaliga.

Peale loengut loe üle loengumaterjal (mitte ainult slaidid), mõtle läbi ja püüa mõisteid meelde jätta ning nendest aru saada.

Peale harjutustundi vaata üle tunnis tehtud ülesanded ja lõpeta see, mis pooleli jäi.

Peale praktikatundi lõpeta tunnis pooleli jäänud ülesanded. Vaatan neid hea meelega üle!

Proovi saada sõbraks C keelega.

Ülevaade C-keelest ehk selgitusi keelekonstruktsioonidest võrdluses Pythoniga

Interpreteeritav / kompileeritav

Python on interpreteeritav keel.

C on kompileeritav keel:

- Alati tekib masinkoodis programm (Windowsis laiendiga .exe, Linuxis nimega a.out).
- Masinkoodis programmi saab eraldi käivitada.

Mis on erinevus interpreteeritavast keelest?

Kompileerimisprotsess

Leksiline analüüs – analüsaator töötleb koodi sümbol-haaval, tuvastab koodis lekseemid ja asendab need märkidega.

Süntaksi analüüs – ka parsimine, ehitatakse süntaksipuu, tuvastatakse laused ja programmi vastavus programmeerimiskeele grammatikale.

Semantiline analüüs – uuritakse, kas on järgitud keelenõudeid (sõltub keelest!): näiteks kas omistuslaues on andmetüübid omavahel vastavuses või kas muutujad on enne kasutamist kirjeldatud / deklareeritud.

Optimeerimine - sõltuvalt kompilaatori "osavusest" võib järgneda koodi optimeerimine, muutes näiteks lausete järjekorda mälukasutuse vähendamiseks

Kompileerimisprotsess

Koodi genereerimine – võetakse vahekoodis programm ja tõlgitakse konkreetse arvuti masinkoodi.

Selle käigus võidake koodile lisada "koodijupikesi" teistest masinkoodis programmidest (nt erinevad teegid).

Linkimine – ka nii võidakse nimetada eelnevat tegevust ja sõltub kompilaatorist, kas seda osa kuidagi välja tuuakse.

Lisa saad lugeda siit:
https://www.tutorialspoint.com/compiler_design/compiler_design_phases_of_compiler.htm

Eelnevast jutust on oluline kõrva taha panna, et kompileerimisel väljatoodavad vead võivad tulla erinevatest faasidest ning ühe vea likvideerimise järel võib ilmneda viis uut viga. Ja see on OK!

Kompilaator ja IDE

C-keelseid programme saab arendada erinevates töökeskondades. Sobib näiteks CodeBlocks (GNU GPL), Visual Studio, ...

Toetab GCC kompilaatoreid (GNU Compiler Collection) – erinevate keelte kompilaatorite kogum, mis on valminud GNU projekti käigus.

C-keele kompilaator oli esimene sellest seeriast (1987).

Kasutame C-keelt (mitte C++ või C#!!).

Katsetame koodi kirjutamist ja töötlemist ka Linuxis.

Erinevate IDEde korral peab olema ettevaatlik, sest C-keelel on mitmeid dialekte, st väikeseid erinevusi süntaksis, funktsioonide kasutuses jms.

C keele ajaloost

Loodud Dennis Ritchie poolt 1969-72. aastatel.

Kasutatud OS-i Unix ümberkirjutamiseks (eelmine versioon oli Assembleris).

1978. a. C keelt kirjeldav raamat:

"The C Programming Language", autorid Brian Kernighan ja Dennis Ritchie

https://en.wikipedia.org/wiki/C_(programming_language)#/media/File:The_C_Programming_Language,_First_Edition_Cover.svg

Raamatus olev C kirjeldus ei ole üheselt mõistetav

Standardiseeritud 1989 ANSI poolt (nn ANSI C, C89), 1990 ISO poolt (C90). Viimane standard ISO/IEC 9899:2018 (nn C17). Ettevalmistamisel on järgmine versioon, eeldatavasti C23.

C keel

C keel:

- on üldotstarbeline *(general-purpose)*
- on imperatiivne *(imperative)*
- toetab struktuurprogrammeerimist *(structured programming)*
- muutuja skoop e kehtivuspirkond on staatiline, seotud koodiosaga *(lexical variable scope)*
- võimaldab rekursiooni *(recursion)*
- kasutab staatilist andmetüübi kontrolli *(static type system)*

hello world

Sellisel kujul ilmus maailma kuulsaim(?) programm esimest korda Brian Kernighan'i ja Dennis Ritchie (kutsutakse K&R) C-keele raamatus "The C Programming Language".

```
#include <stdio.h>

main()

{

    printf("hello, world\n");

}
```

hello world

Järgnevat varianti võib lugeda standardile paremini vastavaks:

```
#include <stdio.h>

int main(void)

{

    printf("hello, world\n");

    return 0;

}
```

Teine näide K&R raamatust

```
/* print Fahrenheit-Celsius table
      for f = 0, 20,..., 300 */
main()
{
    int lower, upper, step;
    float fahr, celsius;
    lower = 0;    /* lower limit of table */
    upper = 300; /* upper limit */
    step = 20;    /* step size */

    fahr = lower;
    while (fahr <= upper) {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf ("%4.0f %6.1f\n", fahr, celsius);
        fahr=fahr+step;
    }
}
```

Andmetüübid ja muutujad

C-s tuleb iga muutuja jaoks määrata andmetüüp (muutuja deklareeritakse / defineeritakse)

Andmetüübid on (osaline valik):

- Täisarvud: `int` (4 baiti); formaat `%d`
- Ujukomaarvud: `float` (4 baiti), `double` (8 baiti); formaadid `%f, %lf`
- Sümbol: `char` (1 bait, sõltub kooditabelist); formaat `%c`

Deklareerime kaks 4-baidist täisarvulist muutujat:

```
int arv1, arv2;
```

Andmetüübi täpsustused

Tavaliste andmetüüpide ees võime kohata veel järgmisi keelesõnu: `long`, `long long`, `short`, `signed`, `unsigned`.

Täpsed suurused sõltuvad arvutisüsteemist.

`long long int` on tavaliselt 2x suurem kui `int`, `short int` aga 2x väiksem (vastavalt 32, 64 või 16 bitti)

Ainult postiivsete (märgita) arvude kasutamiseks `unsigned int`

Sisestusel ja trükkimisel kasuta õigeid formaate!

Sisend ja väljund

Andmete sisselugemise ja trükkimise funktsioonid paiknevad `stdio` **teegis** *(library).*

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

Trüki küsimus `printf()`-ga ja vastust loe `scanf()`-ga

```
int a1, a2;
printf("Sisesta kaks arvu ");
scanf("%d %d", &a1, &a2);
printf("Arvud on %d ja %d.\n", a1, a2);
```

Sisend ja väljund

```
scanf("%d %d",&k1,&k2);
```

Ootab sisendiks kahte tühikuga eraldatud täisarvu.

`&` on muutuja muutmälu (RAM) aadressi tähis ja tema tähendusest räägime täpsemalt edaspidi. Funktsioon tahab teada muutuja asukohta mälus, mitte tema nime.

Tuleta meelde, et üks muutuja omadustest on tema mäluaadress, mille järgi arvuti / programm tema muutmälust üles leiab.

Näide ...

... formaadi kasutamisest `scanf()`-ga:

```
int pikk_m,pikk_cm;
printf("Sisesta pikkus ");
scanf("%d m %d cm",&pikk_m,&pikk_cm);
```

Ootab sisendit:

```
12 m 50 cm
```

NB! Ebakorrektne formaadikasutus põhjustab muutujasse vale väärtuse lugemise. Sealjuures ei pruugi tekkida täitmisaegset viga.

Aritmeetikatehted ja omistamine

Toimub Pythonist tuttaval viisil (tehete jrk jms).

Puudub astendamistehe, seda asendab funktsioon `pow(arv,aste)`.

Ruutjuure leidmiseks `sqrt()`, vajalik lisada `#include <math.h>`

Jagamistehte tulemus ja andmetüüp sõltub operandide tüüpidest (`int/int -> täisarvuline` jagamine).

Unaaroperaator `++` on inkrement ja `--` dekrement (suurendamine ja vähendamine 1 võrra)

Tüübiteisendused

Kasutada saab tüübiteisenduse *(type cast)* operaatorit.

```
f1 = (float)i1 / i2;
```

Tehte tegemiseks muudab operaator `(float)` muutuja `i1` väärtuse ujukomaarvuks, toimub tavaline jagamine.

NB! `i1` sisaldab jätkuvalt täisarvu!!

Mis on järgmise tehte tulemuseks?

```
(int)29.55 + (int)21.99
```

`(int)` on ka unaaroperaator, tehete järjekorras esimeste seas.

Kas tehte `(int)(29.55 + 21.99)` tulemus on eelnevaga sama?

Võrdlus- ja loogikatehted

C-s on andmetüüp `_Bool`, nn tõeväärtustüüp. Sinna salvestatakse 0 (väär) või 1 (tõene)

Võrdlustehted: `==, !=, >=, <=, >, <`

Nt töötab omistamine: `a = 5 > 4` (a saab vrt 1)

Loogikatehted: `!` (*not*), `||` (*or*), `&&` (*and*)

Tõeseks loetakse mistahes mittenulliline operand, **vääraks** nulliga võrduv operand.

Mõtle loogikatehteid kasutades väärtustele *true* ja *false* ning kirjuta võrdlustehted korralikult välja!

Liitlause

C-s kasutatakse mõistet **liitlause** *(compound statement)* tähistamaks mitmest lausest koosnevat plokki.

Liitlaused paiknevad tavaliselt juhtlausete sees (`if, while, for, ...`)

Liitlaused ümbritsetakse looksulgudega `{}`

Iga lause liitlauses lõppeb semikooloniga `;`

Mis on Pythoni analoog liitlause eristamisel?

Tingimuslause ehk `if`-lause

`if`-lause töötab täiesti harjumuspärasel viisil.

Süntaks:

```
if (tingimus) {
   laused, kui tingimus tõene;
}
else {
   laused, kui tingimus väär;
}
```

`if` ja `else` järel on liitlause.

`else`-osa võib puududa.

Näide

```
...
if (leitud==1) {
   printf("On olemas\n");
}
else {
   printf("Ei ole olemas\n");
}
...
```

Selles näites võiksid looksulud ka puududa, sest liitlauseks on üks lause.

Korduslaused

Korduslauseid *(iteration statement),* tsüklilauseid ehk silmuslauseid on C-keeles kolm:

- `while`-lause (eelkontrolliga tsükkel);
- `do`-lause (järelkontrolliga tsükkel);
- `for`-lause (määratud korduste arvuga tsükkel).

`while`- ja `do`-laused

`while`-lause on tavapärane. Süntaks:

```
while (jätkamise tingimus) {
    korduses täidetavad laused;
}
```

`do`-lauses on tingimus tsükli lõpus. Süntaks:

```
do {
    korduses täidetavad laused
}
while (jätkamise tingimus);
```

`For`-lause

`For`-lause tööpõhimõte erineb oluliselt Pythoni `for`-lause omast. Süntaks:

```
for (avaldis1;avaldis2;avaldis3) {
    tsüklis täidetavad laused;
}
```

`avaldis1` täidetakse üks kord enne tsükli algust

`avaldis2` on tsükli jätkamise (loogika)tingimus, kontrollitakse enne iga kordust

`avaldis3` täidetakse iga korduse lõpus

`For`-lause näide

`For`-lausel võib olla mitmeid kummalisi ja raskesti mõistetavaid variante, tavalisim on järgmine:

```
int i, n=0;
for (i=0; i<n; i++) {
   printf("%d",i);
}
```

Selgitus: enne tsüklit tehe `i=0;` kontroll, kas `i<n`. Kui on, siis trükitakse `i`, suurendatakse `i` 1 võrra (`i++`). Tegevus kordub.

Proovi sama kirja panna `while`-lausega!

Lõpetuseks

Sellega on olulised laused üle vaadatud ja praktikumis saame katsetada lihtsamate (ja võib olla ka tuttavate) ülesannete lahendamisega, harjumaks C-keele süntaksiga.

Enim probleeme tekib arvatavasti andmete formaadiga sisestamisel.

Tegelikult on lisaks `scanf()`-le veel sisestuse funktsioone, mis on andmetüübi-põhised ja mida on soovitav kasutada stringe / märke sisestades.