

# T-106.3101 Ohjelmoinnin jatkokurssi T2 (6 op)

## Luento 5: Standardikirjaston funktioita. I/O, tiedostot.

Sami Liedes

31. tammikuuta 2008  
(päivitetty 31. tammikuuta 2008)

## Tänään luvassa

- ▶ Standardikirjastoista ja -otsikoista
- ▶ `stdio.h`
- ▶ `printf()`
- ▶ Syötteen lukeminen
  - ▶ `scanf()`
  - ▶ `fgets()`
- ▶ Tiedostot
- ▶ `errno`

## ANSI/ISO C -standardikirjasto

- ▶ ANSI/ISO standardoi joukon otsikoita ja kirjastofunktioita
- ▶ C99:ssä näitä on muutamia uusia (usein hyödyllisiä)
  - ▶ Esim. `<stdint.h>` ja vakiomittaiset kokonaislukutyypit (`uint32_t` ym.)
- ▶ Toisaalta ISO-standardifunktiot jättävät tarkoituksella jotain "liian järjestelmäspezifistä" pois
  - ▶ Verkkoliikenne rutiinit
  - ▶ Säikeet, prosessit
  - ▶ Monimutkaisemmat tekstikäyttöliittymäasiat, kuten näytön tyhjennys, kursorin siirtäminen, värit
- ▶ Matematiikkarutiinit eriytetty omaksi kirjastokseen, otsikko on `<math.h>` ja kirjasto saadaan linkitettyä kääntäjän vivulla `-lm`

## POSIX-standardi

- ▶ POSIX = Portable Operating System Interface
- ▶ Joukko hieman järjestelmäspesifisempiä funktioita
- ▶ Hyödyllisiä moderneissa käyttöjärjestelmissä
- ▶ Alun perin kehitetty Unixien yhteensopivuutta varten, mutta myös esim. Windows on nykyään osin POSIX-yhteensopiva
- ▶ Täydentää C-standardikirjastoa
  - ▶ verkkoliikennenerutiinit (<sys/socket.h>)
  - ▶ säikeet ja prosessit (<pthread.h>, fork())
  - ▶ käyttäjänimet, tiedostojen omistajat ja oikeudet, ...

## Tiedonhaku kirjastorutiineista

- ▶ GNU C-kirjaston dokumentaatio:  
<http://www.gnu.org/software/libc/manual/>
  - ▶ Sisältää sekä ISO että POSIX, lisäksi laajennuksia
- ▶ Edellämainittu on melko hyvä, mutta lisää (esim. käyttöesimerkkejä) voi hakea Googalla
  - ▶ Webissä on kuitenkin paljon väärääkin tietoa C:stä
- ▶ Komento  
man 3 *funktio* (joskus 2)
- ▶ Joskus on järkevää käyttää standardikirjaston lisäksi jotain lisäkirjastoja, esim. *ncurses* monimutkaisempiin tekstikäyttöliittymiin
  - ▶ Tällä kurssilla kannattanee tyytyä yksinkertaisiin tekstikäyttöliittymiin ilman *ncurses*ia
  - ▶ Goblin ei tykkää *ncurses*ista

## Syöttö ja tulostus

- ▶ Ohjelmien syötteen lukemisesta ja tulostuksesta käytetään nimeä I/O = Input/Output = syöttö ja tulostus
- ▶ Lukeminen esim. näppäimistöltä tai tiedostosta
- ▶ Kirjoittaminen esim. näytölle tai tiedostoon
- ▶ C-standardikirjaston I/O-rutiinit on esitelty otsikkotiedostossa `<stdio.h>`
- ▶ Sisältää rutiinit tekstin muotoiltuun tulostamiseen ja lukemiseen
- ▶ *standard input* (`stdin`) = oletuksena näppäimistö
- ▶ *standard output* (`stdout`) = oletuksena näyttö
- ▶ *standard error* (`stderr`) = oletuksena näyttö

## stdio.h:n funktioita 1/2

- ▶ Usein funktioista kaksi versiota, esim. `printf()` ja `fprintf()`, joista ensimmäinen käyttää `stdin/stdout`ia ja jälkimmäinen tiedostoa
- ▶ Esim.  
`printf("luku=%d\n", luku)` on täsmälleen sama kuin `fprintf(stdout, "luku=%d\n", luku)`
- ▶ Muotoiltu ja muotoilematon syöttö ja tulostus
- ▶ Rutiinit tiedostojen käsittelyyn
- ▶ Huomattavaa: Usein tulostus on rivipuskuroitu, jolloin se tulee näkyviin vasta rivinvaihdon tulostuksen jälkeen
  - ▶ mahdollista pakottaa, ks. `fflush()` jos tämän kanssa tulee ongelmia

## stdio.h:n funktioita 2/2

<code>printf()</code> , <code>fprintf()</code>	muotoiltu tulostus
<code>fgets()</code>	kokonaisen rivin lukeminen
<code>puts()</code> , <code>fputs()</code>	muotoilematon tulostus
<code>getchar()</code> , <code>getc()</code>	lukee yhden merkin
<code>putchar()</code> , <code>putc()</code>	kirjoittaa yhden merkin
<code>sprintf()</code> , <code>snprintf()</code>	muotoiltu tulostus merkkijonoon
<code>scanf()</code> , <code>fscanf()</code>	muotoillun tekstin ja lukujen lukeminen
<code>sscanf()</code>	<code>scanf()</code> merkkijonosta
<code>feof()</code>	kertoo, onko tiedoston loppu saavutettu
<code>fopen()</code>	avaa tiedoston
<code>fclose()</code>	sulkee tiedoston
<code>fflush()</code>	pakottaa kirjoitetun mutta puskuroidun datan tiedostoon tai näytölle asti

## puts() ja putchar() – Merkkijonon ja merkin tulostus

- ▶ puts() tulostaa muotoilematonta tekstiä
- ▶ puts("Heippa, maailma!");
- ▶ Toisin kuin printf(), puts() tulostaa loppuun rivinvaihdon
  - ▶ Miksi? (ei voi tietää)
  - ▶ fputs() sen sijaan ei tulosta rivinvaihtoa...
- ▶ putchar() - tulostaa yksittäisen merkin
- ▶ putchar('c');

## printf() – Muotoiltu tulostus

- ▶ Tekstin muotoiltuun tulostamiseen
- ▶ Ottaa ensimmäiseksi parametriksi *muotoilun*
- ▶ Loput parametrit ovat muotoilussa käytettyjä muunnosmääreitä vastaavat arvot (muuttujat)

- ▶ Esim.

```
printf("luku on %d\n", luku);  
printf("Opiskelijan nimi on \"%s\"\n", nimi);
```

## printf():n muunnosmääreitä

Tärkeimmät muunnosmääreet:

<i>Määre</i>	<i>Merkitys</i>
%s	nollatavuun päättyvä merkkijono ( <i>char *</i> )
%d (tai %i)	<i>int</i> -tyyppinen kokonaisluku
%hd	<i>short</i> -tyyppinen kokonaisluku
%ld	<i>long</i> -tyyppinen kokonaisluku
%f	<i>double</i> , muotoa -1.234
%e	<i>double</i> 10-potenssimuodossa, esim. -2.234e+08
%g	valitsee sopivamman %e:stä ja %f:stä
%lf, %le, %lg	samat <i>long double</i> lle
%c	<i>char</i> (yksittäinen merkki)
%%	%-merkki

## Muunnosmääreiden tarkenteita

- ▶ %-merkin ja määrään väliin voidaan tarvittaessa lisätä tarkenteita
- ▶ Kokonaisluku kertoo kentän leveyden, esim. "%5d"
- ▶ Miinusmerkki: kenttä täytetään vasemmalta alkaen
- ▶ `printf("|%6d| ja |%-6d|\n", 123, 123)`  
→ `"|_ _ _ 123|_ _ja_|123_ _ _|"`
- ▶ Tarkenne 0 (nolla): täytetään välilyöntien sijaan nolilla (esim. "%06d" → "000123")
- ▶ *double*ille haluttu tarkkuus: piste ja kokonaisluku (esim. "%.5f" → "123.45679", "%.2f" → "123.46")
- ▶ Muitakin hienouksia löytyy, halutessasi katso esim. manuaalisivut

## Muuta printf():stä

- ▶ Palauttaa tulostettujen merkkien määrän
- ▶ Ei tulosta rivinvaihtoa, haluttaessa `\n` lisättävä itse
- ▶ Keskenäisen rivin pakotus näytölle asti:  
`fflush(stdout)`
- ▶ Sama merkkijonoon: `sprintf()`, `snprintf()`
  - ▶ `snprintf()` ottaa lisäksi kohdemerkkijonon maksimikoon
  - ▶ Turvallisuussyistä hyvä käyttää `snprintf():ä`
- ▶ Jos haluat tulostaa käyttäjältä saadun merkkijonon, älä anna sitä `printf():n` muodoksi. Käytä `puts()`, `fputs()` tai `printf("%s", mjonon);`
  - ▶ Merkkijonossahan voi olla esim. %-merkkejä

## fgets() – Rivin lukeminen

- ▶ Syötteen lukemiseen on olemassa fgets() ja getchar()
  - ▶ On olemassa myös gets(), mutta se on aina vaarallinen (koska se ei ota merkkijonon maksimipituutta), käytä aina fgets(mjono, koko, stdin)
- ▶ char \*fgets(char \*kohde, int koko, FILE \*tiedosto)
- ▶ Lukee merkkijonoon kohde *korkeintaan koko-1 merkkiä* ja laittaa loppuun nollatavun (joten kohteen koko tulee olla vähintään *koko* tavua)
- ▶ Palauttaa osoittimen *kohde*, virhetilanteessa tai tiedoston lopussa erityisen "null-osoittimen" NULL
- ▶ Muuten lukeminen loppuu rivinvaihtoon, *joka myös tallennetaan kohteeseen*

## getchar() – Yhden merkin lukeminen

- ▶ Syötettä on mahdollista lukea myös merkki kerrallaan: `getchar()`
- ▶ Palauttaa yhden merkin tai arvon EOF (syötteen loppu)
- ▶ Useimpiin sovelluksiin `getchar()` on liian hienosäätöä, mutta joskus syötettä on mielekästä tulkita merkki kerrallaan
- ▶ Nopeampaa voi olla lukea merkkijono kerrallaan ja käsitellä sitä, tai binääritiedoston tapauksessa lohko kerrallaan (`fread()`)
  - ▶ Mutta jos nopeudella ei ole väliä, älä tee asioista turhan vaikeita (Google: KISS principle)

## scanf() – Muotoillun syötteen lukeminen

- ▶ printf():n kaltainen funktio muotoillun syötteen lukemiseen
- ▶ Ottaa myös ensimmäiseksi parametriksi muotoilun
- ▶ Loput parametrit ovat muuttujia, joihin luetaan
  - ▶ Kuitenkin, täytyy antaa *osoitin muuttujaan* (&muuttuja), jotta scanf() voi kirjoittaa siihen
  - ▶ Paitsi merkkijonomuuttujat, joille ei tarvita &-merkkiä
- ▶ Esim.

```
scanf("%d", &luku);          /* lukee kokonaisluvun */  
scanf("luku = %d", &luku);  /* hyväksyy syötteen muotoa  
                             "luku = 123" */  
scanf("sana = %8s", mjono); /* lukee yhden sanan,  
                             max. 8 merkkiä */
```

## scanf():n muunnosmääreitä

- ▶ Useimmat muunnosmääreet samoja kuin printf():ssä
- ▶ double-määreet (%f, %e, %g) kaikki keskenään ekvivalentteja
- ▶ 9 merkin lukeminen välilyönneistä välittämättä:  
scanf("%9c", m\_jono);
  - ▶ Ei lisää nollatavua!
- ▶ Tiettyjen merkkien lukeminen:  
scanf("%[0-9a-fA-F]", m\_jono);
  - ▶ Lukee merkeistä 0-9, a-f ja A-F koostuvan merkkijonon
  - ▶ Komplementti: ^ (hattu) [-merkin jälkeen
  - ▶ Esim.  
scanf("%[^!]", m\_jono); /\* lukee !-merkkiin asti \*/  
scanf("%10[^!]", m\_jono); /\* korkeintaan 10 merkkiä \*/

## Muuta scanf():stä

- ▶ Palauttaa onnistuneiden sijoitusten lukumäärän (voi olla myös 0) tai arvon EOF (syötteen loppu)
- ▶ %n: tähän asti luettujen merkkien määrä
  - ▶ Ei ole muunnos, eikä (ilmeisesti) kasvata paluuarvoa (tehtyjen muunnosten määrä)
  - ▶ Standardi epäselvä tältä osin, parempi ettei oletta mitään
- ▶ Sama merkkijonosta: sscanf ( )
- ▶ Monesti on kätevää lukea kokonainen rivi kerrallaan (fgets ( )) ja parsia sitä sscanf ( ):llä

## feof() – Onko syöte luettu loppuun?

- ▶ feof ( ) -funktio kertoo, onko syöte luettu loppuun
- ▶ feof(stdin)
- ▶ feof(tiedosto)
- ▶ Joskus kätevä, monesti tosin myös itse lukufunktiot palauttavat arvoja jotka kertovat syötteen loppumisesta

## Tiedostojen avaaminen ja sulkeminen

- ▶ Tiedosto avataan `fopen()`-funktiolla
- ▶ `fopen()` palauttaa *tiedostokahvan*, joka on tyyppiä `FILE *`
- ▶ `FILE *fopen(const char *nimi, const char *tila);`
- ▶ Tila voi olla muun muassa
  - ▶ "r" (avataan lukemista varten)
  - ▶ "r+" (avataan lukemista ja kirjoittamista varten)
  - ▶ "w" (avataan kirjoittamista varten)
  - ▶ Lisäksi tilan perään voidaan lisätä kirjain 'b', jolloin tiedostoa käsitellään binääritiedostona
- ▶ `fopen()` palauttaa `NULL` epäonnistuessaan
- ▶ Tiedosto suljetaan `fclose()`-funktiolla
- ▶ Kuten muisti, myös tiedostokahvat voivat vuotaa (ja loppua)

## I/O tiedostoille

- ▶ Aiemmin mainituille luku- ja kirjoitusfunktioille on olemassa vastineet, jotka ottavat parametriksi tiedoston (FILE \*)
- ▶ `fprintf()`, `fscanf()`, `fgets()`
- ▶ Esimerkki, joka kirjoittaa tiedostoon "Heippa, maailma!":

```
1 #include <stdio.h>
2 int main() {
3     FILE *fp = fopen("heippa.txt", "w"); /* "w" = kirjoitus */
4     if (!fp) /* NULL = 0 = epätosi C:ssä */
5         puts("Tiedoston avaaminen epäonnistui!");
6     else {
7         fputs("Heippa, maailma!\n", fp);
8         fclose(fp);
9     }
10    return 0;
11 }
```

## Tiedostojen kelaaminen

- ▶ Tiedostoja voi myös kelata
- ▶ Tekstitiedostoille ei välttämättä ole aina selvää, mitä tämä tekee (paitsi alkuun tai loppuun kelaaminen)
- ▶ Esimerkiksi `fseek()`, `ftell()`
- ▶ Lisätietoja manuaalista

## errno - Virhekoodi

- ▶ `errno` - yksi harvoista standardikirjastossa määritellyistä globaaleista muuttujista
- ▶ Esittely löytyy otsikosta `<errno.h>`
- ▶ Koska C:ssä ei ole poikkeuksia, täytyy virhetilanne osoittaa funktion paluuarvolla
- ▶ Monet standardikirjaston funktiot asettavat virhetilanteen sattua muuttujaan `errno` virhekoodin
- ▶ Virhekoodeja ovat mm.
  - ▶ `EPERM` - Ei oikeuksia (tiedostoon tmv.)
  - ▶ `ENOENT` - Yritettiin avata tiedostoa jota ei ole

## perror() - Ihmisten luettava virheteksti

- ▶ Ihmisluettavan virhetulostuksen voi tehdä perror()-funktioilla (<stdio.h>)
- ▶ Tulostaa errno:n arvoa vastaavan virheilmoituksen virheilmoituksille tarkoitettuun stderr-virtaan
- ▶ Funktio ottaa parametriksi merkkijonon johon virhe liittyy
- ▶ perror("tiedosto"); saattaa tulostaa esim.  
tiedosto: No such file or directory
- ▶ Myös omia virheilmoituksia voi tulostaa stderr-virtaan:  
fprintf(stderr, "Tiedosto \"%s\" on rikki.\n", nimi);

## Seuraavalla luennolla ti 5.2.

- ▶ Kaksiulotteisia taulukoita
- ▶ "Puolidynaamisia" taulukoita (`int *taulu[10]`)
- ▶ Komentoriviparametrien käsittely
- ▶ Linkitetty lista
- ▶ Funktio-osoittimia
- ▶ *Huomaa*: to 7.2. ei luentoja  
(katso luentoaikataulu kurssin WWW-sivuilta)