p1

Starta uppgift

Inlämningsdatum 10 nov 2019 av 23.59 **Poäng** 1 **Lämnar in** en filuppladdning
Filtyper py och txt

Table of Contents

1 Förberedelser

1.1 github

Senare i kursen kommer vi att lära oss använda versionshanteringssystemet github. För att kunna generera kurskataloger åt var och en så behöver ni **logga in på KTH:s github** (https://gits-15.sys.kth.se/) med ert KTH:login.

1.2 Få tillgång till kommandoraden.

Under kursen kommer vi jobba med olika unix-verktyg på kommandoraden.

1.2.1 Mac OS X

Mac OS X har de flesta, men inte alla, verktyg/program installerade.

Följ instruktioner på nätet hur man öppnar terminalen (https://www.google.com/search?&q=how+to+open+terminal+mac+os+X) lägg därefter till en genväg till terminalen.

Det finns ett installationssystem som heter homebrew som kan vara värt att titta på för att installera ytterligare verktyg eller uppdatera de existerande programmen.

1.2.2 Linux/ubuntu terminal

Om du har linux är verktygen ofta redan installerade. **Öppna en terminal** (https://www.google.com/search?q=how+to+open+terminal+linux)

Du kan installera de verktyg du behöver på ubuntu med apt-get. Ett tips är att installera paketet build-essential som innehåller flera programmeringsspråk färdigpaketerade.

1.2.3 Windows 10

Installera **ubuntu från windows store** (https://www.microsoft.com/en-us/p/ubuntu/9nblggh4msv6)

Följ instruktionerna som ber dig kryssa i linux subsystem under kontrollpanelen. Installera därefter python3, gcc med kommandot apt-get vilket för övrigt är samma som ubuntu Linux.

Ett tips för både windows 10 och linux/ubuntu är att installera paketet build-essential som innehåller flera programmeringsspråk färdigpaketerade.

Om du vill köra python3 eller C i en utvecklingsmiljö så kommer du att behöva ladda ner python3 och en C kompilator (t.ex. mingw) för windows.

1. windows 10 filsystem

   När du startar ubuntu på windows 10 så ligger alla filer i ett internt filsystem. Det är praktiskt att ha dina labbfiler i windows eget filsystem. Windows filsystem ligger på c:\ men under ubuntu hittar man dem via *mnt/c* för att skapa en genväg dit följ dessa instruktioner

   1. Skapa en katalog för dina labbfiler med kommandot mkdir (make directory)
   2. Skapa sedan en länk (genväg) till katalogen med kommandot ln [från] [till] punkten i exemplet betyder den katalog du står på.
   3. Gå in i katalogen med kommandot cd (change directory).

```
mkdir /mnt/c/tilpro
ln -s /mnt/c/tilpro .
cd tilpro
```

1.3 Introduktion unixterminal och filsystem

Under kursens gång kommer vi använda UNIX terminalfönster. Här följer en kort introduktion. Du behöver inte kunna dessa kommandon nu.

Unix filsystem har sin root i / och hela filsystemet är en trädstruktur därifrån (jmfr instruktionerna ovan).

Det finns **ca 160 terminalkommandon** (https://en.wikipedia.org/wiki/List_of_Unix_commands) i standard UNIX. De flesta har väldigt korta och ointuitiva namn. Här följer några som kan vara värt att lära sig

kommando	förklaring
cd	(change directory) byter katalog
ls	(list) listar filer, om inget argument ges så listas filerna i katalogen
man	(manual) skriver ut manualen för ett givet unix-kommando
mkdir	(make directory) skapar en katalog
rmdir	(remove directory) tar bort en katalog
rm	(remove) tar bort en fil permanent (det går att ställa in en varning för att den tas bort permanent)
mv	(move) flyttar en fil (eller byter namn på filen) kan också flytta en hel katalogstruktur.
ssh	loggar in på annan dator
scp	kopierar filer från och till annan dator, det är ofta smidigare att använda ett annat program med grafiskt gränssnitt.
cat	(concatenate) skriver ut innehållet från en eller flera filer i terminalfönstret
more	fungerar som cat men skriver ut en skärm i taget
less	fungerar som more men man kan gå fram och tillbaka och söka i utskriften, avslutas genom att trycka q
wc	(word count) räknar antal tecken, ord och radbrytningar
grep	(globally search a regular expression and print) filtrerar utskrift
touch	ändrar tidstämpeln på en fil, skapar filen om den inte finns
echo	skriver ut i terminalfönstret
tee	splittar utskrift att skriva både i terminalfönstret och på en fil
curl	hämtar en fil över nätet givet en url

Vidare finns det specialtecken med olika funktioner

tecken	förklaring
>	(redirect output) omdirigerar terminalutskrift till en fil som skapas
>>	omdirigerar terminalutskrift och lägger till i befintlig fil
<	tar indata från fil istället för inmatning via tangentbordet
|	(pipe) skickar skärmutskriften till ett annat program

Exempel:

```
man ls
man ls > ls-manual.txt
cat ls-manual.txt | grep list
grep list < ls-manual.txt
cat ls-manual.txt | grep list | tee list-grep.txt
cat list-grep.txt
```

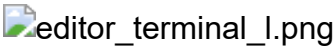
Första raden visar unix-manualen för *ls*. Andra raden sparar ner manualen i filen *ls-manual.txt*. Tredje, fjärde och femte raden filtrerar ut alla rader som innehåller ordet *list* men den femte sparar dessutom ned utskriften i filen *list-grep.txt*

1.3.1 Versionshantering med git

Senare i kursen kommer vi att använda versionhanteringssystemet git som också kan skötas via kommandoraden. För att du ska få tillgång till git måste du surfa in på **KTH:s github (https://gits-15.sys.kth.se/)**. Du loggar in med ditt KTH id (vilket sker per automatik om du redan är inloggad på kth.se) och då blir du registrerad som githubanvändare. Först när du är registrerad på **KTH:s github (https://gits-15.sys.kth.se/)** kan kursledningen generera repositoryn åt dig. Dessa genereras ett par veckor in i kursen.

1.4 Textredigerare (Editor) och utvecklingsmiljö

Det är krav på kursdeltagarna att de ska kunna utveckla program med en textredigerare och terminal. När du behärskar terminal och textredigerare kan du, om du vill, installera en eller flera integrerade utvecklingsmiljöer och skriva dina program i dessa.



Oavsett editor, lär dig åtminstone använda sök, sök och ersätt och hur editorn kan fylla ut/gissa det ord man börjat skriva på. Nedan listas några alternativ på editorer.

1.4.1 Integrerade utvecklingsmiljöer

Integrerade utvecklingsmiljöer (IDE) är sådana som kör koden i verktyget (istället för i en terminal) och kan dessutom ha mer kontroll och debugverktyg och textediteringsförslag medan man skriver koden. Det tar tid att lära sig en utvecklingsmiljö och vi uppmuntrar att ni tar er tiden att lära er en, glöm dock inte bort att ni ska även kunna hantera en terminal.

1. Visual studio code

   Visual studio code är den rekommenderade utvecklingsmiljön. Den finns installerad på skolans datorer. Vi kan tyvärr inte ge extensiv hjälp med att konfigurera utvecklingsmiljön.

   När du öppnar ett pythonprogram kommer Visual studio code att föreslå att du installerar en python-mode (extension). Gör så. Det går att köra python-program direkt i visual studio code men det kan kräva konfigurering av operativsystemet, särskilt om du använder windows 10.

   Det går att använda Visual studio code både som editor och utvecklingsmiljö.

2. Alternativa utvecklingsmiljöer
   - JetBrains tillhandahåller **pycharm** (https://www.jetbrains.com/pycharm) med studentlicens som är en avancerad utvecklingsmiljö
   - **Xcode** (https://developer.apple.com/xcode/) finns enbart för Mac-användare.
   - IDLE följer med när man installerar python för windows, vi rekommenderar inte att IDLE används.

1.4.2 Alternativa textredigerare (editorer)

1. Atom och Sublime

   Atom är en kostnadsfri avancerad editor som dessutom finns installerad på skolans datorer.

   Sublime är en ny avancerad editor som kan köras kostnadsfritt men med en påminnelseruta som poppar upp och påminner att det går att köpa produkten till en engånskostnad. Det finns ett flera tillägg till Sublime.

2. Nano

   Nano är en mycket enkel editor som kan köras på unixterminal. Den har en liten menyrad längst ner där kortkommandon listas. Den kan vara bra om man t.ex. vill logga in via terminalen (med ssh) på en annan dator och göra små ändringar.

3. Emacs och Vim

   Det finns två gamla kostnadsfria avancerade och nybörjarovänliga editorer, **vim** och **emacs**. Båda kan ta lång tid att lära sig.

   Emacs är en lispinterpretator och lite av ett operativsystem i sig. Man startar en instans av emacs och har igång den hela tiden. Det finns ett rikt utbud av tillägg till emacs och på senare tid har tillägget org-mode med vars hjälp enkelt kan skriva rapporter m.m. blivit populärt.

   Vim är en snabbstartad editor som körs i terminalen. Vim har på senare tid fått ett uppsving med nya användare på KTH.

2 Programmingsuppgift i python, spårutskrifter, variabelnamn, buggrapportering

Öppna detta pythonprogram i en webbläsare. (http://www.csc.kth.se/utbildning/kth/kurser/DD1321/20/p1.php) Markera innehållet (ctrl-a) och kopiera (ctrl-c). Klistra in programmet i en editor och spara programet som **p1.a.py**

2.0.1 Att köra programmet

Läs igenom föreläsningsanteckningarna F1-F2.

Öppna en terminal, gå till den katalogen där programmet sparades och skriv

```
python3 p1.a.py
```

Programmet uppmanar dig att uppdatera, kör igen

```
python3 p1.a.py --update
```

Kursens schema skrivs ut i terminalen. Programmet är trasigt. Schemat verkar vara förskjutetoch en del annat är konstigt. Öppna programmet i en editor. Observera du behöver **inte** förstå programmet i dess helhet nu, först vid kursens sista föreläsningar har vi gått igenom programmet i dess helhet.

2.0.2 Spårutskrifter

Programmet har en klass med ointuitiva medlemsvariabler med grekiska bokstäver. Leta upp i koden var dessa tilldelas värden och gör en spårutskrift för att se vad det är som lagras i medlemsvariablerna.

En spårutskrift skriver ut variablens namn och värde samt var någonstans i koden utskriften skett.

```
# Exempel på spårutskrifter
print("någonstans i koden: m.group(1) =", m.group(1) )
print("någonstans i koden: klass_instans.ΞΘΩPBΨZ =", klass_instans.ΞΘΩPBΨZ)
```

Din första uppgift är att byta namn på alla de grekiska medlemsvariablerna genom att använda sök och ersätt i editorn. Byt till namn som man intuitivt förstår. Använd spårutskrifter för att ta reda vad variabeln innehåller. Byt därefter även namn på klassen. Spara denna nya version som p1.b.py

Du kan ändra på namnen senare om du kommer på bättre namn. Det kan finnas fler variabelnamn som kan förbättras.

2.0.3 Ändra i programmet

I funktionen som läser från fil finns följande rader.

```
#file_content = infil.readlines()
file_content = infil.read()
```

Ändra så att programmet läser med read istället för readlines()()

```
#file_content = infil.readlines()
file_content = infil.read()
```

Följ programmet, vad händer med det som returneras? Någonstans kommer programmet att gå sönder eftersom filinnehållet inte längre kommer som en stor sträng utan som en lista/vektor av rader.

I vilken funktion kraschar programmet?

Ändra tillbaka så att programmet fungerar igen. Men **kommentera i get_file_content** att OUT är en sträng och i funktionen som kraschade att IN förväntas vara en sträng.

Spara denna version som p1.c.py

2.0.4 Kodkommentarer

Försök kommentera de två funktionerna som du arbetade med ovan. Använd spårutskrifter för att ta reda på vad som skickas in och vad som returneras. Kommentera parametrarna under rubriken IN och vad som returneras under UT och skriv även vad du tror funktionen gör.

Du behöver inte kommentera de funktioner eller parametrar du inte förstår nu, vi kommer att återvända till labben senare. Spara den kommenterade koden som p1.d.py

2.0.5 Hårdkodning

I huvudprogrammet (main) är strängen "DD1321.htm" hårdkodad, den anger namnet på den lokala fil där schemat sparas i HTML-format. Inför en ny global variabel och sätt värdet på variabeln till "DD1321.htm". Ta bort filen från disk, kör igen (med *-update*) och se till att programmet fungerar som det ska.

Strängvärdet "DD1321.htm" är fortfarande hårdkodat men nu finns det bara på ett ställe i koden.

2.0.6 Buggrapport

Den globala variablen url innehåller länk till schemat. Öppna länken i en webbläsare. Jämför schemat med utskriften och skriv ner minst tre, principiella avvikelser.

Gå vidare med nästa uppgift, du kommer att komma tillbaka till programmet senare i kursen.

3 Programmering i C

3.1 Kompilering och exekvering

Skriv in följande C-program i en editor. Spara filen som p1.21.c

```
#include <stdio.h>

int main() {
  printf("Hej välkommen till 21-spelet\n");
  printf("Vi börjar från 0 och ökar i tur och ordning med antingen 1 eller 2, den som först kommer till 2
1 vinner!!!\n");
  printf("Exempel:\n du säger   1\n jag säger  2\n du säger   4\n jag säger  6\n du säger   8\n jag säger
10\n du säger  12\n jag säger 14\n du säger  18\n jag säger 20\n du säger 21 och vinner!!!\n\n");
  for (int summa = 3; summa <= 21; summa += 3) {
    printf("Vilket tal säger du? ");
    int tal = 0;
    while ( 1 ) {
      scanf("%d", &tal);
```

```
      if (summa - tal == 1 || summa - tal == 2)
        break;
      else {
        printf("Hallå där, du kan bara öka med 1 eller 2\n");
        printf("Vi var på %d, Vilket tal säger du? ", summa - 3);
      }
    }
    printf("Jag säger %d\n", summa);
  }
  printf("Jag vann !!!\n");
}
```

Kompilera filen i en termimal genom att skriva

```
> gcc p1.21.c
```

Observera att du i terminalen måste stå i katalogen där du sparat filen **p1.21.c**

När du kompilerar filen så bildas en exekverbar (körbar) fil som heter **a.out**

Kör (exekvera) programmet genom att skriva

```
> ./a.out
```

3.2 Kompileringsalternativ

vill man att den körbara filen ska heta något annat (t.ex. 21spel.out) så måste man ange det vid kompilering med flaggan -o

```
> gcc -o 21spel.out p1.21.c
> ./21spel.out
```

ofta vill man ha debug info när man kompilerar, det gör man med flaggan -g

```
> gcc -g -o 21spel.out p1.21.c
```

3.3 printf

Googla på **printf**, vad betyder %d på raden

```
printf("Vi var på %d, Vilket tal säger du? ", summa - 3);
```

3.4 scanf

Funktionen **scanf** tar hand om det som matas in av användaren. Vad händer om man matar in något konstigt, t.ex. några bokstäver istället för ett tal?

3.5 programanalys

Programmet vinner alltid, analysera programmet och förklara varför. Du behöver inte kunna redogöra för detaljer i programmet.