KTH – STH

TENTAMEN

`HI1024:TEN2` - *Praktisk tentamen*
Tid: Fredagen den 21 oktober 2011, 8.15-13.15

Gamla kurskoder: HI1900, 6E2950, etc.
Examinator: *Johnny Panrike*
Rättande lärare: *Nicklas Brandefelt*, *Johnny Panrike* och *Peter Steiner*
Tentamen konstruerad av de som rättar tentan.

Tentamen består av **5** sidor (inklusive denna sida.)

Instruktioner. Alla uppgifter är värda 4 poäng styck. Tentan har en **A**-del och en **B**-del. A-delen innehåller tre uppgifter och B-delen innehåller två uppgifter. För att få minsta godkända betyg, E, ska man klara 8 poäng på A-delen. Om man inte klarat A-delen rättas inte B-delen. Om man klarat A-delen har man minst betyget E. B-delens uppgifter kan då ge ett högre betyg.

Tillåtna hjälpmedel: Vad som helst i *pappersform*. Böcker, utskrifter, egna anteckningar etc. Dock inga som helst elektroniska hjälpmedel som USB-minnen, CD-skivor eller liknande.

- På `W:\` finns det tillhörande filer för tentamen
- Man behöver inte ta hänsyn till `ÅÄÖ` någon av uppgifterna.
- Svaren till uppgifterna ska lämnas i form av *C*-program med namnen `UPPG1.C`, `UPPG2.C`, `UPPG3.C`, `UPPG4.C`, `UPPG5.C`. Inga andra filer kommer att beaktas vid rättningen (förutom de filer som hör till olika resultat som ska produceras av de program ni skriver).

För att ett program ska rättas måste det kunna köra (och därmed definitivt kompileras.) Rättningen går till som så att programmet provkörs för några olika indata, ofta andra än de som ges på själva tentamen. Programmet ska ge korrekt resultat vid dessa körningar. Vidare ska även programmets form och innehåll uppfylla de krav som ställts i uppgiften, helt enkelt till exempel om man ska skriva en funktion så ska en funktion finnas i svaret som man ger in.

Alla uppgifter ger 4 poäng. Man kan få delpoäng för halvt fungerande program, men man får aldrig poäng för program som inte ens kan kompilera och köra. På samma sätt ges inga poäng för program som kraschar. (Till exempel om man glömmer adressoperatorn, `&`, då den behövs till `scanf()`.)

Uppgifterna i Del A examinerar också det som benämnts som *kärnkompetensen* i programmering. Vid varje uppgift anges vilket delområde av kärnkompetensen som de examinerar.

Del A

1 Stoppa in ett tal (4p) *[strukturering]*

Skriv en funktion som stoppar in ett heltal på rätt plats i en redan sorterad lista.
Funktionen skall ha tre in-parametrar: en array, antal element som fältet är fyllt med, och ett tal som ska stoppas in.
Funktionen kan utgå ifrån att det finns plats för talet i array:en, låt exempelvis arrayen ha 1000 platser.
Skriv kod i main som testar funktionen.

Vi studerar ett exempel på anrop:

```
int nytt_tal=21, tal_lista [100]={1,10,45,65}, antal=4;
stoppa_in(tal_lista, &antal, nytt_tal);
```

Detta anrop resulterar i att tal_lista innehåller: `{1,,10,21,45,65,..}` och `antal` får värdet 5.

Mall för main med hjälpfunktion för att skriva ut listan:

```
#include <stdio.h>

#define STORLEK 100

void print(int tal_lista[], int antal){
	int i;
	printf("\n\n<Lista> {");
	for(i=0; i<antal; i++){
		printf(" %d", tal_lista[i]);
		if(i<antal-1)
			printf(",");
	}
	printf(" } <\\Lista>\n");
}

int main(void){
	int nytt_tal, tal_lista [STORLEK]={1,10,45,65}, antal=4;

	printf("\nStartup...");
	while(1){
		printf("\n Ange ett tal: ");
		scanf("%d", &nytt_tal);
		if(nytt_tal!=0){
			stoppa_in(tal_lista, &antal, nytt_tal);
		}else
			break;
		print(tal_lista, antal);

	}
	printf("\nShutdown...\n");
}
```

2 Fordon (4p)

[representation/omvandling]

I denna uppgift arbetar vi med följande definitioner:

```
#define PERSONBIL 0
#define LASTBIL 1
#define MOTORCYKEL 2

struct motorfordon
{
  int typ;
  int arsmodell;
  char marke[20];
  float vikt;
};
```

En textfil `fordon.txt` med blandade fordon är given. Uppgiften består i att sortera dessa fordon i fyra nya textfiler: `bilar.txt`, `lastbilar.txt`, `motorcyklar.txt` och `veteranfordon.txt`. I `fordon.txt` beskrivs ett fordon på en rad i taget och en rad ser då ut så här:

`ÅRSMODELL:TYP:MÄRKE:VIKT`

`ÅRSMODELL` är heltalet som motsvarar årsmodellen på fordonet.
`TYP` är ett av heltalen 0, 1 och 2 (enligt definitionerna ovan).
`MÄRKE` är fordonets märke (som tex *Volvo* eller *Saab*).
`VIKT` är fordonets vikt med 1 decimals noggrannhet.

Exempel på en Volvo personbil från 1979:
`1979:0:Volvo:1500.8`

Alla veteranfordon ska alltså klumpas ihop i filen `veteranfordon.txt` och de övriga ska sorteras i de nämnda filerna. OBS: Ytterligare en förändring i formatet ska också ske, istället siffrorna 0, 1 och 2 ska texterna **`"PERSONBIL"`**, **`"LASTBIL"`**, **`"MOTORCYKEL"`** och "**`VETERANFORDON`**" användas så att Volvon från filen `fordon.txt` ovan representeras i filen `bilar.txt` av raden:

`1979:PERSONBIL:Volvo:1500.8`

Ledning:
Vid inläsning av textsträngen som beskriver ett fordon kan man läsa in en rad till flera variabler samtidigt genom följande sats:

```
fscanf(fil_pekare, "%d:%d:%s:%f", . . . );
```

Där det står "`. . .`" ska ni ange de variabler som värdena ska läsas in till. Formatsträngen "`%d:%d:%s:%f`" är anpassad till indatats format i textfilen `fordon.txt`.

(Det blir lite tårta på tårta att det står "`PERSONBIL`" på varje rad i en fil som heter `bilar.txt`, men det får vi leva med.)

3. Punkter rummet

[Sortering, representationsformer och strukturering.]

Punkter i rummet kan beskrivas med tre koordinater organiserade i en *struct*, så här:

```
struct punkt {
  double x, y, z;
};
```

Avståndet till origo kan bildas genom uttrycket `sqrt(x*x+y*y+z*z)`. Ett antal punkter i rummet är givna i en binärfil (på `W:\`) baserat på structen ovan. Skriv ett program som läser in dessa i en array och sorterar på avstånd från origo, närmaste punkten ska komma först och punkten längst från origo ska komma sist. Resultatet ska skrivas ut på skärmen så här:

```
Punkt 1: (x,y,x) Avstånd: d
Punkt 2: (x,y,x) Avstånd: d
Punkt 3: (x,y,x) Avstånd: d
...
```

Här står de kursiverade bokstäverna för de tal som ska förekomma i utskriften hörande till det riktiga programmet. Det räcker att skriva ut punkternas koordinater (x, y och z) med två decimaler. (Alltså `printf("... %.2f ...")`.) Likadant för avståndet.

Ledning: Nedanstående program användes för att skapa binärfilen:

```
#include <stdio.h>
#define MAX 5

struct punkt {
  double x, y, z;
};

int main() {
  int i; FILE * punktfil;
  struct punkt enpunkt;

  punktfil = fopen("punkter.dat","wb");
  for(i=0;i<MAX;i++)
  {
    printf("Punkt %d.\n", i+1);
    printf("x: "); scanf("%lf", &(enpunkt.x));
    printf("y: "); scanf("%lf", &(enpunkt.y));
    printf("x: "); scanf("%lf", &(enpunkt.z));
    fwrite(&enpunkt, sizeof(struct punkt), 1, punktfil);
  }
  fclose(punktfil);

}
```

Funktionen `sqrt()` finns i bibliotket `math` som måste inkluderas för att kunna användas.

Del B

4. Binär sökning (4p)
Binär sökning kan man göra när man har ett sorterat material. Om vi t.ex har en lista med sorterade heltal och vill veta om ett visst nummer finns i listan så tittar vi i mitten och ser om det är för stort eller för litet. Om det är för stort vet vi att numret finns i den vänstra delen av vår array om det är för litet vet vi att den finns i den högra delen. Nu gör vi samma sak igen men med denna del osv.

Ex:

Låt oss söka efter 15 i följande lista: Vad står det i mitten av listan? Jo, **21**.
2 13 15 **21** 33 45 101
15<21 så vi tittar i den vänstra delen: Vad står det i mitten av vänstra listan? Jo, **13**.
2 **13** 15
15>13 så vi tittar i högra delen: Vad står i mitten? Jo, **15**.
15
15=15 så vi har hittat talet!

Skriv en funktion som tar en array med heltal och en `int langd` som innehåller längden på arrayen och en `int tal` som innehåller talet som eftersöks. Funktionen skall sedan returnera index för platsen där den hittat talet. Den ska använda en binär sökning för att göra detta. Du kan utgå från att varje tal endast finns en gång i vår array.

Funktionsdeklaration:

```
int sokning(int v[], int langd, int tal);
```

Exempel på fullständigt `main` som testar funktionen:

```
int main(void)
{
     int i;
     int v[10]={0,1,4,7,12,21,22,23,30,40};
     int index=sokning(v,10,4);
     printf("Index: %d,",index);
}
```

Vid rättningen kommer andra siffror och arraystorlekar att användas.

5. Flygplatssimulering (4p)

På en flygplats med tre landningsbanor är det 5% sannolikhet att det kommer ett nytt plan som vill landa varje minut. Om det finns en ledig landningsbana får planet landa direkt. Om det inte finns en landningsbana får planet köa tills det finns en ledig. Att genomföra en landning för ett plan tar 20 minuter sedan är banan ledig igen.

Skriv ett simuleringsprogram som simulerar en dag (12h) och tar reda på hur länge det plan som fått vänta längst har fått vänta.