Tentamen *:58/2I4122 Programmering i C 2004-11-20

Denna tentamen består av fyra uppgifter som tillsammans kan ge maximalt 30 poäng.

För godkänt resultat krävs minst 18 poäng. Väl godkänt ges för speciellt bra lösningar. För KTH-studenter krävs minst 18 poäng för betyget 3, minst 23 poäng för betyget 4 och minst 27 poäng och bra lösningar för betyget 5.

Observera:

- skriv tydligt, oläsbar lösning medför 0 poäng
- börja varje uppgift på nytt blad
- skriv endast på ena sidan av pappret
- skriv namn på varje inlämnat blad
- redovisa eventuella antaganden (som givetvis inte får strida mot uppgiftstexten)
- tänk på att det kan passa dig bättre att lösa uppgifterna i annan ordning

Tillåtna hjälpmedel:

- boken "Programmeringsspråket C" eller "The C Programming Language" av Kernighan/Ritchie i valfri upplaga
- kurskompendium *:58/2I4122 Programmering i C HT01 eller senare (rött omslag)

LYCKA TILL !

Uppgift 1 (Inläsning, stränghantering - 6 poäng)

I ett program där man hanterar information om artiklar i olika butiker har man följande posttyp:

```
#define MAX_SHOPS 10
typedef struct item{
   char *name;
   int price;
   char *shops[MAX_SHOPS];
   int shop_count;
} Item;
```

`name` ska peka ut artikelnamnet, `price` är priset i kronor, `shops` är en maximerad array av `char`-pekare som ska peka ut namn på de butiker där artikeln säljs och `shop_count` anger hur många av pekarna i `shops` som används.

I programmet vill man kunna läsa in data till sådana poster, varvid användaren förväntas mata in data för en artikel på en rad, enligt exemplet

```
Hammare 127 Kista, Nacka, Malmö, Göteborg
```

Först kommer artikelns namn (ett ord) sedan priset, sedan en lista av butiknamn, separerad med komma och/eller mellanslag. Artikelnamnet och priset måste finnas med (annars är inmatningen ogilltig) och priset måste vara ett heltal. Listan av butiksnamn kan vara tom.

Skriv funktionen `readItem()` som tar en sådan `Item`-post som argument, läser in data från användaren (från `stdin`) och lägger dem i posten. Om inmatningen är felaktig skall ett felmeddelande ges (vad som var fel behöver inte anges) och ny inmatning begäras. Funktionen skall inte returnera något utan uppdatera sitt argument.

Uppgift 2 (Bitmanipulering, preprocessormacros - 6 poäng)

Skriv två preprocessormacros `PUT(b,uli)` och `GET(uli,nr)` som kan användas för att stoppa in resp. ta ut byte-värden i/från en `unsigned long int` (sådan teknik används ibland för att kunna skicka flera byte-värden som ett argument till biblioteksfunktioner).

Macrot `PUT` ska stoppa in byte-värdet `b` i heltalsvariabeln `uli` (`unsigned long int`), det ska kunna anropas flera gånger på samma heltalsvariabel:

```
unsigned long int param;
PUT('a', param); /* 'a' har stoppats in som byte 0 */
PUT('b', param); /* 'b' har stoppats in som byte 1 */
PUT('c', param); /* 'c' har stoppats in som byte 2 */
```

Macrot `GET` ska sedan kunna användas för att få ut ett visst byte-värde:

```
char ch = GET(param, 2); /* ch har fått värdet 'c' */
```

Du får själv bestämma i vilken ordning byte-värden ska lagras i heltalsutrymmet, se bara till att `PUT` och `GET` stämmer överens, d.v.s. om argumentet `nr` till `GET` är lika med *x* så ger `GET` det byte-värde som stoppades in vid *x*-te anropet av `PUT`.

Du ska inte kontrollera om det finns plats för fler bytevärden vid `PUT` eller om `nr`-argumentet till `GET` är rimligt, om programmeraren som använder dessa macros gör fel för hon skylla sig själv.

Uppgift 3 (Kommandoradsargument, enkel filhantering - 6 poäng)

Skriv ett program som kan läsa en eller flera textfiler och skriva ut tecken från varje fil tills man har träffat på (och skrivit ut) ett visst antal av ett visst tecken (avskiljartecknet).

T.ex. om programmet heter `upg3` och följande kommando ges:

```
upg3 –d; -n15 –outfil.txt infil.txt
```

så ska programmet skriva ut text från filen `infil.txt` till filen `utfil.txt` tills det har träffat på och skrivit ut 15 semikolon.

Om det finns färre förekomster av avskiljartecknet i en infil så skrivs hela filen ut.

Infilsnamn, utfilsnamnet, avskiljartecknet och antalet ges på kommandoraden.
Utfilsnamn, avskiljartecknet och antalet ges som flaggor:

`-o`*`filnamn`*	utskriften skall göras till en fil med namnet *`filnamn`* – innan denna flagga påträffats görs utskriften till `stdout`
`-d`*`tecken`*	*`tecken`* är avskiljartecknet, innan denna flagga har man inget avskiljartecken utan skriver helt enkelt ut angivet antal tecken (`d` står för *delimiter*)
`-n`*`heltal`*	*`heltal`* är antalet avskiljartecken som ska träffas på för att avsluta behandligen av denna infil, innan denna flagga påträffats gäller 10 förekomster av avsklijartecknet
`-h`	namnet på varje infil ska skrivas ut på egen rad innan texten från infilen (`h` står för *headers*)

Flaggorna kan förekomma flera gånger på kommandoraden och gäller alla efterföljande filer eller tills en ny flagga med samma innebörd påträffats.

Om inga infilsnamn påträffats på kommandoraden skall input tas från `stdin`.

Uppgift 4 finns på nästa sida!

Uppgift 4 (Generella moduler, dynamiska arrayer, funktionspekare - 12 poäng)

Denna uppgift handlar om att skriva en modul för en mycken enkel datastruktur `PrioQueue`, som skall implementera en prioritetskö för element av en godtycklig typ.

En prioritetskö är en datastruktur där man kan stoppa in element och man kan ta ut element, och det är alltid det största elementet som tas ut (och returneras) först.

Modulen skall innehålla en funktion `add()` som tar en `PrioQueue` och en pekare till ett godtyckligt objekt som argument och stoppar in objektet i datastrukturen, samt en funktion `del()` som tar en `PrioQueue` som argument och tar bort och returnerar pekare till det största av de objekt som ingår i datastrukturen. Den skall även innehålla en funktion `empty()` som tar en `PrioQue` som argument och anger om denna är tom samt en funktion `init()` som skapar, initierar och returnerar ett `PrioQueue`-objekt.

Det ställs inga krav på effektiv implementering. Datastrukturen **skall** implementeras som en omallokerbar array med pekare till elementobjekten.

Observera att tillämpningsprogram som skall använda `PrioQueue` måste kunna ange hur man avgör om ett objekt är större än ett annat eftersom det inte är klart vad "störst" betyder för ett godtyckligt objekt.

Modulen som skall implementera `PrioQueue` beskrivs av följande headerfil `prioqueue.h` (`init()`-prototypen är inte fullständig, argumenten skall du bestämma själv):

```
#ifndef _PRIOQUEUEH_
#define _PRIOQUEUEH_

typedef struct pqstruct *PrioQueue;

PrioQueue init(/* Eventuella argument bestämmer du själv */ );
void add(PrioQueue pq, void *object);
void *del(PrioQueue pq);
int empty(PrioQueue pq);

#endif
```

a) Skriv modulen `PrioQueue.c`

b) Antag följande deklarationer:

```
typedef struct person{
   char *namn;
   int age;
} Person;

Person *make(char *namn, int age){
   Person *p = malloc(sizeof(Person));
   p->namn = malloc(strlen(namn)+1);
   strcpy(p->namn, namn);
   p->age = age;
   return p;
}
```

Skriv ett litet program som skapar några `Person`-objekt, stoppar in dem i en `PrioQueue` och tar sedan ut objekten och skriver ut dem så att de äldsta personerna skrivs ut först.