Datorövning 1

A) Kör igång MicrosoftVisualC++-miljön, välj File | New | Project och skapa ett nytt tomt Win32 Console Project med namnet Strtest. Bläddra och skapa en ny katalog M:\PCSA\Metod. Ditt projekt hamnar nu i katalogen M:\PCSA\Metod\Strtest\. Ta fram Solution Explorer med View och addera den nya filen Strang.c till projektets Source Files. I denna fil ska du skriva funktionen strlangd, som tar en sträng som parameter och returnerar strängens aktuella längd. Du får inte använda den färdiga funktionen strlen. Spara filen och kompilera filen separat genom att i Solution Explorer markera filen Strang.c och med höger musknapp välja Compile. Addera sedan en ny fil Strtest.c till Source Files och skriv där ett huvudprogram, som läser in en sträng på max 30 tecken och skriver ut strängens längd genom att anropa funktionen strlangd. Kompilera separat och tänk på att deklarationen (huvudet) för funktionen strlangd måste skrivas in före main. Länka ihop projektet med Project | Build och kör det med Debug | Start.

B) Välj File | Open Project och öppna projektet Strtest. Utöka filen Strang.c med en ny funktion strkopia, som tar två strängar som parametrar och kopierar den andra strängen tecken för tecken till den första. Du får ej använda den färdiga funktionen strcpy. Kompilera filen separat. Komplettera huvudprogrammet med ett anrop av strkopia samt skriv ut den kopierade strängen. Istället för att skriva funktionernas deklarationer direkt i samma fil som huvudprogrammet, skriver du in dessa i en headerfil strang.h, som du sedan inkluderar i huvudprogrammet. Filen skapar du genom att addera en Headerfil till projektet. Länka och kör.

*C) Skapa ett projekt medtest som förutom huvudprogrammet *medtest.c*, som läser in och skriver ut data för en medlem, innehåller den abstrakta datatypen medlem i filerna *medlem.c* och *medlem.h* där headerfilen ska innehålla specifikationen

```
typedef struct
{
   int nr;
   char namn[30];
} medlem;
void las_medlem(medlem *mp);
void skriv_medlem(medlem m);
```

**D) Kopiera alla filer från kurssidans Studmetod till din aktuella katalog. Skriv sedan ett program (se sid 22 i kompendiet), som använder *nollfunk.h* och *nollfunk.c* , för att numeriskt lösa ekvationen:

$$e^x = 2 - x \quad \text{(Svar : 0.443)}$$

**E) Testa funktionsbiblioteket sort, som finns i filerna *sort.h* och *sort.c* i din katalog, genom att initiera en vektor av medlemmar enligt C) som sorteras efter nr. Du måste tillfoga en jämförelsefunktion för medlemmar.

Datorövning 2

A) Skriv ett program som läser in ett heltal, tilldelar en pekare adressen för detta heltal och skriver ut heltalet med hjälp av pekaren.

B) Skriv funktionen *medsprid* som beräknar medelvärdet och spridningen (största - minsta) för en vektor av flyttal och som anropas från ett huvudprogram enligt:

```
#include <stdio.h>
int main()
{
   float med, sprid, vek[5] = {2.3, 1.2, 3.4, 5.6, 4.5};

   medsprid(vek, 5, &med, &sprid);
   printf("Medel = %.2f\nSpridning = %.2f\n", med, sprid);
   return 0;
}
```

C) Skriv ett program som dynamiskt allokerar minne för ett flyttal, läser in ett värde till detta tal och skriver ut talet multiplicerat med 3.

D) Skriv ett program som dynamiskt allokerar minne för två strängar på maximalt 30 tecken vardera, läser in ditt förnamn till den första och ditt efternamn till den andra. Byt sedan pekare, så att den första pekaren kommer att peka på efternamnet och den andra på förnamnet. Testa genom att skriva ut. Använd en temporär pekare, så att du inte tappar bort något namn.

*E) Skriv ett program som dynamiskt skapar mätposter av typen :

```
struct mes
{
   int nr;
   float x;
   struct mes *next;
};
```

läser in värden nr och x till dessa samt länkar ihop dessa med hjälp av nextpekarna. Inlänkningen ska upprepas så länge det inlästa nr-värdet ej är 0. Då inlänkningen avslutats skrivs de ingående x-värdena ut följt av medelvärdet.

**F) Gör om programmet ovan i E) så att det istället för att läsa in värden från tangentbordet läser från textfilen xindata.txt, som du själv skrivit rader med nr- och x-värden i. Programmet ska sedan fortsätta med att fråga efter ett nr-värde och med en sökfunktion söka efter motsvarande x i listan. Låt sökfunktionen ha nr och lista som parameter och returnera x.

Datorövning 3

A) Skriv ett program som från tangentbordet läser en textrad tecken för tecken och stoppar in tecknen i en LIFO-lista. Efter instoppningen tas tecknen bort från listan och skrivs ut på skärmen (ska bli omvänt). Använd den abstrakta datatypen lifo, i filerna *lifo.h* och *lifo.*c, vilka du hämtar från kurssidans Studmetod. Anpassa lifo.h till char och inkludera den till programmet. Infoga lifo.c till projektet.

B) Skapa en textfil med ett antal ordnade heltal enligt exempelvis:

123 156 245 312 367 489 534..

Skriv ett program som läser talen från filen, stoppar in dessa på en stack och slutligen skriver ut talen i omvänd ordning på skärmen.

C) Skriv ett program som från tangentbordet läser in ett antal reella tal (avsluta med 0) till en FIFO-lista, samtidigt som medelvärdet beräknas. Plocka sedan bort talen från listan och skriv ut alla tal som avviker mer än 10% från medelvärdet. Använd den abstrakta datatypen fifo i filerna *fifo.h* och *fifo.c*, för att hantera listan

D) Skapa en textfil med ett antal reella tal enligt:

5.43 5.67 5.34 …..

Skriv sedan ett program som läser filen och stoppar in talen i en tvåvägslista av FIFO-typ och skriver ut listan på skärmen. Använd den abstrakta datatypen twolist i filerna *twlolist.h* och *twolist.c* för att hantera listan.

*E) Skapa en abstrakt datatyp för personer med namn och telefonnummer (även riktnummer alltså en sträng) och med operationer för att läsa från tangentbord och skriva på skärm. Skriv sedan ett program som läser in personer (avsluta med bara RETURN genom att låta läsfunktionen returnera 0 då tom sträng läses för namn) från tangentbordet, skriver in personerna på en binärfil telreg.dat och som avslutande test skriver ut filen på skärmen

*F) Skriv ett program som läser den binära filen telreg.dat och stoppar in personerna, sorterade efter namn, i en tvåvägslista. Skriv sedan ut hela listan på skärmen.

**G) Komplettera programmet i uppgift F) så att den innan listan skrivs ut, frågar efter en persons namn och om personen ifråga finns i listan plockar bort denna från listan.

**H) Ändra programmet i uppgift G) så att den istället för att ta bort personen, som lästs in, frågar efter nytt telefonnummer och uppdaterar personen innan den nya listan skrivs ut.

**I) Skriv ett program som läser den binära filen telreg.dat och stoppar in personerna sorterade efter telefonnummer i en tvåvägslista. Låt sedan programmet fråga efter ett riktnummer och på skärmen skriva ut alla personer med detta riktnummer.

Datorövning 4

A) Implementera den abstrakta datatypen komplex i filen *komplex.c* då specifikationen, som du skriver in i filen *komplex.h* ser ut som:

```
typedef
struct
{
   float re;   /* Realdel */
   float im;   /* Imaginärdel */
} komplex;

void las_komplex(komplex *kp); /* Läser på formen 3.4+4.5j */
void skriv_komplex(komplex k); /* Skriver på formen 3.4+4.5j */
komplex add_komplex(komplex k1, komplex k2); /* k1 + k2 */
komplex sub_komplex(komplex k1, komplex k2); /* k1 - k2 */
komplex mul_komplex(komplex k1, komplex k2); /* k1 * k2 */
komplex div_komplex(komplex k1, komplex k2); /* k1 / k2 */
```

Skriv sedan ett huvudprogram i *impedans.c*, som läser in två komplexa impedanser och skriver ut ersättningsimpedansen för parallellkopplingen.

*B) En stack är en kö av typen LIFO. En statisk stack som kan innehålla max 100 tecken kan avbildas som en abstrakt datatyp enligt :

```
typedef struct
{
   int top;       /* aktuellt index i vek */
   char vek[100];/* vektorn som innehåller stackelementen */
} stack;

void clear(stack *sp);          /* tömmer stacken */
void push(stack *sp, char ch); /* stoppar in ch på stacken */
char pop(stack *sp);     /* returnerar det översta elementet */
int is_empty(stack s);          /* är stacken tom? */
int is_full(stack s);           /* är stacken full?*/
```

Skriv in specifikationen i filen *vekstack.h*, implementera funktionerna i filen *vekstack.c* och skriv ett huvudprogram i *stackmai.c,* som läser in en text som stackas tecken för tecken och skrivs ut baklänges på skärmen.

**C) Gör den abstrakta datatypen stack mer generell så att även andra datatyper än char kan stackas. Testa genom att stacka komplexa tal, enligt ovan, istället för tecken.

Datorövning 5

A) Skriv ett program som läser in ett heltal n och genom att anropa en rekursiv utskriftsfunktion, med n som parameter, skriver ut talen enligt :

1
2
..
n

Kör programmet i Debuggern och studera hur n varierar.

B) Fibonacci-funktionen f för heltal definieras rekursivt enligt:

$$f(n) = \begin{cases} 1 & \text{om } n = 1 \text{ eller } 2 \\ f(n-2) + f(n-1) & \text{om } n > 2 \end{cases}$$

Skriv ett program som innehåller en rekursiv funktion för beräkning av det n:te fibonaccitalet. Testa funktionen genom att läsa in ett tal och skriva ut motsvarande fibonaccital.

*C) Skriv ett program som läser in personerna från binärfilen telreg.dat, som du skapade i datorövning 3E), till ett sökträd, frågar efter en persons namn och om personen finns i trädet skriver ut personen med namn och telefonnummer. Använd dig av din tidigare skapade abstrakta datatyp person, som du kompletterar samt av *tree.h* och *tree.c* som finns i din katalog.

*D) Skriv ett program enligt C) ovan men som istället för ett sökträd använder en länkad hashtabell enligt *linkhash.h* och *linkhash.c* i din katalog.

*E) Skriv ett program enligt D) ovan men som istället för en länkad hashtabell använder sig av en öppen adresserad hashtabell enligt *openhash.h* och *openhash.c* i din katalog.

**F) Utöka den abstrakta datatypen tree med en rekursiv funktion som beräknar antalet noder i trädet. Testa funktionen genom att skriva ut ditt telefonregister och under personerna skriva ut antalet personer.

**G) Utöka den abstrakta datatypen linkhash med en funktion som söker upp angiven nyckel och om den finns i hashtabellen tar bort motsvarande länk.

Datorövning 6

A) Skriv ett program som läser in ett heltal och med en funktion *udda* testar om heltalet är udda. Låt funktionen med lämplig bitoperator testa bit nummer 0 i heltalet.

B) Skriv ett program som läser in ett tecken, skriver ut tecknets ASCII-kod i binär form på skärmen och räknar och skriver ut antalet satta bitar i koden. Använd funktioner för *utskrift* och *antal*.

*C) När man sänder 7-bitars tecken brukar man använda den 8:de biten som en kontrollbit eller paritetsbit. Skriv ett program som läser in ett tecken, räknar antalet satta bitar i teckenkoden och sätter rätt paritetsbit om udda paritet ska gälla, dvs antalet satta bitar i det sända tecknet ska vara udda.

*D) Att multiplicera ett tal med 2 är detsamma som att skifta bitarna i talet ett steg till vänster. Skriv ett program som läser in ett tal och det antal gånger som talet ska multipliceras med 2 samt skriver ut resultatet. Använd en funktion som tar tal och antal som parametrar och returnerar resultatet.

**E) Antag att du i en byte vill hålla reda på dag (1 - 7) och månad (1 - 12). Skriv ett program som läser in dag och månad som heltal och skapar en motsvarande byte. Låt sedan programmet fortsätta med att från byten hämta dag och månad och skriva ut dessa. Använd funktioner för omvandlingarna.

**F) Använd ett bitfält på 1 byte för att hantera punkter i ett koordinatsystem. Använd de 4 minst signifikanta bitarna till punktens x-koordinat (0-15) och de resterande bitarna till y-koordinaten (0-15). Slumpa sedan två punkter samt beräkna avståndet mellan dem. Använd funktioner för att skapa en punkt samt beräkna avståndet mellan två punkter i koordinatsystemet.