Examinationsuppgifter
C - programmering I (FYD400)

Version 1.3
30:e augusti 2013

Version 1.2 2012-09-11 Jonas Einarsson (jonas.einarsson@chalmers.se)
Version 1.15 2012-09-09 Andreas Heinz (andreas.heinz@chalmers.se)
Version 1.1 2010-08-25 Patrik Johansson (patrikj@fy.chalmers.se)
Version 1.0 2009-09-03 Peter Andersson (peand@chalmers.se)
Göteborgs Universitet 2009, 2010

GÖTEBORGS UNIVERSITET

Innehållsförteckning:

I. Introduktion

Välkommen till FYD400 C-programmering I. Kursen ingår i programmet Datorstödd Fysikalisk Mätteknik (DFM), men kan även läsas separat, och har som syfte att lära dig grunderna i programmeringsspråket C. (I påbyggnadskursen FYD410 C-programmering II är syftet att använda C och olika hårdvarugränssnitt för att utveckla styr-, regler- och mätprogram). Kursen går som halvfartskurs på kvällstid. Detta kompendium kompletterar föreläsningarna och innehåller examinationsuppgifterna – de ditt kursbetyg kommer baseras på (se examination nedan), samt lite praktiska tips för att underlätta för både dig och oss. Förutom uppgifterna här rekommenderar vi att du själv löser ett antal övningsuppgifter från kurslitteraturen eller annan källa för att komma igång på bästa sätt.

II. Översiktligt innehåll

C-programmering I fokuserar på grunderna i C och programmering i allmänhet. Du lär dig göra både konsol- och Windowsbaserade applikationer i en integrerad utvecklingsmiljö. Mer detaljerat innehåll fås på hemsidan och via föreläsningarna. Labdelen av kursen, och därmed examinationen, är uppdelad i fyra delar:

1) Allmän kunskap och grunder i C.
2) Filhantering.
3) Grafiska användargränssnitt.
4) Länkning.

Varje del motsvaras av ett kapitel.
Varje del innehåller ett antal examinationsuppgifter.

III. Examination

Följande gäller:

- Kursen examineras helt med examinationsuppgifterna (alltså *ingen* tentamen).
- All examination sker under schemalagd laborationstid (alltså *ej* via inlämning).
- Examination sker kontinuerligt, i form av redovisning av hela uppgifter eller delkapitel – vänta *ej* till sista labtillfället med alla uppgifter!
- Vid examinationen visar du programmets kod och funktion, och svarar på frågor från labbhandledaren om hur uppgiften löstes samt för en diskussion om ev. förbättringar.
- Alla uppgifter har ett poängvärde och du får dessa poäng efter godkänd redovisning.
- Du **måste ha minst en uppgift per delkapitel godkänd** för att få minst betyget Godkänd.
- Ditt slutbetyg på kursen bestäms utifrån följande kriterier:

Betyget Godkänd: är normalbetyget. Programmen skall vara välskrivna och kommenterade och givetvis lösa uppgiften på ett sätt som utnyttjar C och utvecklingsmiljön på ett bra sätt. Dina program ska hantera fel på ett korrekt sätt; som exempel: om användaren ska mata in heltal ska programmet styra så att inte decimaltal går att mata in och/eller ska ett felmeddelande genereras.

Du måste minst kommit upp i **30** redovisade poäng för att få Godkänd (G).

Betyget Väl Godkänd: här krävs lite mer av dig utöver kraven för godkänd;
- att dina program uppvisar en allmänt god programmeringsteknisk kvalitet
- att du följer anmodad kodstandard
- att du kommenterar koden väl
- att lösningarna meningsfullt utnyttjar finesser i både C och utvecklingsmiljön
- att de färdiga programmen är användarvänliga.

Du måste minst kommit upp i **40** redovisade poäng för att få Väl Godkänd (VG).

Examinationens presentation i utvecklingsmiljön:
För att det ska bli lättare att redovisa ska du göra följande:

- Skapa EN arbetsyta ’workspace’ för varje delkapitel.
- Skapa ETT projekt, på den aktuella arbetsytan, för varje deluppgift.
- Döp både arbetsytor och projekt på ett bra sätt.
- Presentera svar till ’essäuppgifter’ i form av textfiler.

IV. Övrig information

Under kursen kommer utvecklingsmiljön LabWindows CVI från National Instruments att användas till att skapa både konsol- och Windowsapplikationer. Denna utvecklingsmiljö finns installerad i ET- och EP-labben (där laborationerna utförs). Utöver detta så kan du testa en egen LabWindows på din egen dator – besök gärna National Instruments hemsida för LabWindows (http://www.ni.com/lwcvi/). Aktuell kurslitteratur får du via föreläsningarna och hemsidan. Det finns även mycket bra material om C att tillgå på internet.

Lite praktiska tips och tricks:

Det är starkt rekommenderat att du sparar en kopia av dina filer på ett eget USB-minne vid varje labbtillfälle.

- Börja med att bekanta dig med utvecklingsmiljön och andra resurser; lokalisera var på den lokala hårddisken som LabWindows/CVI är installerat, syna katalogstrukturen, fundera lite över katalog och filnamn samt hitta katalogerna 'extlib', 'include', 'samples' och 'toolslib'. Gör egna anteckningar för att enkelt hitta tillbaka.

- Kolla in pdf:en ”Getting Started with LabWindows”. Speciellt rekommenderas den del av kapitel tre som handlar om hur man enkelt kan hitta de formella parametrarna till olika funktioner i utvecklingsmiljön. Ännu mer rekommenderas den del som beskriver ’Function Panels’.

- Inne i LabWindows finns ett stort antal exempel som levereras med utvecklingsmiljön. Sök och testkör en del så upptäcker du vad du verkligen kan göra med C.

- För att få behålla kommandotolkens fönster öppet även efter det att programmet nått sitt slut, vilket kan vara *mycket* praktiskt, kan man använda **KeyHit**() – en av flera stödfunktioner i LabWindows eget win32-API.
För att kunna använda dessa stödfunktioner måste ett bibliotek inkluderas:

```
#include <utility.h>
```

I slutet av programmet lägger man till:

```
while ( !KeyHit( ) ) ;
```

Fönstret är nu öppet tills användaren trycker ner en (nästan valfri) tangent.

(Hur funkar det? Jo, while-satsen gör att programmet "väntar" i en, till synes, oändlig loop tills användaren trycker ner en (nästan valfri) tangent – vissa registreras inte av **KeyHit**(). Observera den tomma satsen ';' i slutet av uttrycket.)

Och kom ihåg:
Om du stöter på frågor under kursens gång så utnyttja gärna de tillfällen som finns, dvs i anslutning till föreläsningar och handledningstillfällen. Viss hjälp går att få via e-post, men det är mycket bättre att du frågar vid de schemalagda tillfällena. Kom ihåg, det finns inga dumma frågor bara dumma svar :-)

Lycka till!

Kapitel 1: Allmän kunskap och grunder i C

Mål: Du ska lära dig använda utvecklingsmiljön, förstå grundläggande programmeringsstruktur och kunna skriva enkla C-program som läser in data från tangentbordet, utför beräkningar och presenterar resultatet med enklare formattering.

1.1 Utvecklingsmiljö, förbehandling, ASCII och kommentarer

1.1.1 'Hello world' – givetvis! - 1p

a) Skriv ett eget program som skriver ut

Låt oss göra lite matte:
*5 * 10 = 50*

(på svenska) i kommandotolkens fönster.

b) Förklara översiktligt vad stegen som sker från källkod till körbart program innebär:

- Preprocessor ("förbehandlaren")
- Kompilator
- Länkare

1.1.2 ASCII - 1p

Förståelse ASCII i C:

a) vad är det hexadecimala värdet för tecknet @ ?

b) vilka två heltal utgör gränserna för alla versaler i ASCII-1?

c) vilken datatyp används i C för att representera ASCII-tecken?

d) skriv ett eget program som skriver ut:

I C kan vi göra en massa saker med escape-sekvenser:
vi har tab
eller vi kan aktivera ett larm

(programmet ska larma vid exekvering)

1.1.3 Kommentarsyntax - 1p

Vilka av följande rader programkod är rätt formaterade? Rätta till de felaktiga och förklara varför/hur de var fel.

a) `/* printf("Skriv ut..."); */`
b) `// printf("Skriv */ ut...");`
c) `printf("// Skriv ut...");`
d) `printf(//"Skriv ut...");`
e) `printf(/* Skriv ut... */"");`
f) Hitta ett alternativ till "**while** (!**KeyHit**( )) ;" (se sedan 4). Du behöver detta om du har inte tillgång till "utility.h".

1.2 Variabler, inläsning och utmatning

1.2.1 Inmatning och utmatning av tal – 1p

Skriv ett program som läser in ett heltal, multiplicerar det med två och beräknar hur många siffror resultatet har. Om användaren matar in felaktig data (dvs, inte ett heltal) skall programmet skriva ut ett meddelande och avslutas. Tips: använd funktionen `scanf`

1.2.2 Formaterad utmatning av tal – 1p

Skriv ett program som läser in två heltal, låt oss kalla dem P och Q. Använd funktionen `printf` för att producera följande utskrifter från programmet:

- Talet P på hexadecimal form,
- Talet P/Q på flyttalsform med exakt 3 decimaler,
- Talet P%Q på heltalsform
- Talet Q*P på tiopotensform (”scientific notation”),
- Talet Q med nollpadding till totalt 9 siffror (ex: 747 skrivs 000000747)

Tips: Angående heltalsdivision, vad är skillnaden på P/Q och (float)P/Q?

1.3 Styruttryck, operatorer och satser

1.3.1 Operatorer - 1p

Vad blir resultaten av följande operationer?
Lös detta moment utan dator och visa steg för steg för varje uttryck.
Antag att följande definitioner finns:
int a=1, b=2, c=3;
a) a & ~b & ~c
b) a & ~b & c
c) a && b & c
d) a ^ b & c
e) a & b & ~c
f) a | b & c

Tips: vad är den binära representationen av talen 1, 2 och 3?

1.3.2 Fler operatorer - 1p

Beräkna a för hand utan dator och visa steg för steg. Ta reda på vilka operatorer som beräknas först!

```
int a = (!(1==0)+(0==1)*1)+(1||1&&0);
```

1.3.3 **while-** och **case**-satser - 2p

Skriv ett program som låter användaren manipulera ett heltal genom en meny. Talet är från början lika med 0, och användaren erbjuds följande val i en meny:
1) Addera 1
2) Multiplicera med 2
3) Subtrahera 3
4) Avsluta programmet

Vid val 1-3 utförs operationen, och användaren kommer tillbaka till menyn. Vid val 4 avslutas programmet. Om användaren matar in något annat så skall programmet hantera det på något lämpligt vis. Tips: använd en `case`-sats för användarens val i menyn, och en `while`-sats för att repetera programmet tills användaren väljer att avsluta. Detta är ett vanligt mönster som du kan återanvända i andra program!

Exempel på utskrift:

```
Talet är 0, välj en operation:
1) Addera 1
2) Multiplicera med 2
3) Subtrahera 3
4) Avsluta programmet

>3
Talet är -3, välj en operation:
1) Addera 1
2) Multiplicera med 2
3) Subtrahera 3
4) Avsluta programmet

>1
Talet är -2, välj en operation:
1) Addera 1
2) Multiplicera med 2
3) Subtrahera 3
4) Avsluta programmet
```

```
>4
Programmet avslutas.
```

1.3.4 Växelkassa – 2p

När du handlar i en affär och betalar kontant får du kanske växel tillbaka. I regel får affärsbiträdet hjälp av kassaapparaten med att beräkna summan man ska få tillbaka, men inte alltid vilka sedlar och mynt som ska lämnas tillbaka. Skriv ett program som beräknar den växel biträdet ska ge tillbaka i samband med ett köp. Programmet ska, förutom att presentera beloppet kunden får tillbaka avrundat till närmsta 50-öring, även bestämma vilka, och antalet, sedlar och mynt. Kunden ska få så få sedlar och mynt som möjligt tillbaka. Programmet ska kunna ge växel tillbaka med sedlar av valörerna 1000, 500, 100, 50 och 20 samt mynten 10, 5, 1 och 50-öring. Du kan anta att det alltid finns tillräckligt antal av de sedlar och mynt som krävs.

1.3.5 Formattering och gränser - 1p

Skriv ett program som beräknar och skriver ut (med lämplig formatering) resultat från funktionen:

$$f(x) = 4x^3 + 3x^2 - 5x - 10$$

Programmet ska beräkna funktionen med följande gränser:
i) Samtliga heltal x [-15,15]
ii) Samtliga x [-1,1] med en upplösning i x på 0.1

1.4 Funktioner och funktionsprototyper

1.4.1 Funktioner - 2p

Du skall nu skriva ett program som sträcker sig över flera källkodsfiler. I de flesta projekt organiseras kod i separata funktioner, i olika filer, efter syfte. Den som sedan vill använda sig av funktionerna *inkluderar* koden med hjälp av preprocessordirektiv. Du kommer nu skapa ett enkelt sådant *bibliotek* för konvertering mellan enheter.

a) Skapa filerna convert.h, convert.c samt main.c. I convert.c skapar du en funktion som tar emot avstånd mätt i meter, och returnerar avståndet i millimeter. Placera en funktionsprototyp i convert.h. Skriv sedan ett enkelt program i main.c som använder sig av funktionen ur convert.h.
b) Förklara hur instruktioner till förbehandlaren kan förhindra att samma deklarationsinformation inkluderas dubbelt under kompilering
c) Förklara skillnaden mellan följande två preprocessordirektiv:

```
#include <Test.h>
#include ”Test.h”
```

1.4.2 Funktionsargument – pass by value - 1p

Skriv ett program med en funktion `dubblera`, som dubblerar ett tal:

```
void dubblera(int x)
{
	x = 2*x;
}
```

Gör en `main`-funktion som 1) initierar ett heltal till något värde, 2) skriver ut det, 3) använder funktionen `dubblera` på det och 4) skriver ut talet igen.

a) Vad ger ditt program för svar, och varför?
b) Ge ett förslag på hur man kan göra en `dubblera`-funktion istället.

1.5 Pekare och adresser

Pekarvariabler är helt enkelt heltal som kan tolkas som en viss position i datorns minne: en adress. Alla variabler finns lagrade någonstans i minnet, och har en adress. En pekare används för att referera till olika ställen i minnet. Har man en pekare kan man referera till det den pekar till med *-operatorn. Vice versa kan man med en variabel få veta dess adress med &-operatorn. Följande uppgifter syftar till att praktisera dessa

1.5.1 Pekare, avreferering och adresser – 3p

a) Om vi har deklarationerna
```
int heltal = 10;
int *pekare_till_heltal = &heltal;
```
Vilka av följande uttryck resulterar i en utskrift av talet 11? (och varför? Testa!)
```
1)  printf("%i\n", pekare_till_heltal + 1);
2)  printf("%i\n", heltal+1);
3)  printf("%i\n", *(&heltal)+1);
4)  printf("%i\n", *(pekare_till_heltal + 1));
5)  printf("%i\n", *pekare_till_heltal + 1);
6)  printf("%i\n", &heltal + 1);
```
b) Du har många gånger använt funktionen `scanf(”%d”, &heltal);` Vad gör &-tecknet egentligen så att `scanf` fungerar som den gör?

c) Gör ett program med funktionen `dubblera` som i uppgift 1.4.2, men använd en pekare för att få det att fungera. `dubblera` har nu alltså deklarationen `void dubblera(int *x)`

1.5.2 Flyttande pekare – 1p

Gör ett program med en meny liknande uppgift 1.3.3. Ditt program skall skapa en sträng i minnet:
```
char text[] = "Gothenburg University!";
```
samt en pekare som pekar på början av strängen:
```
char* my_pointer = text;
```
Låt nu användaren manipulera pekaren med menyvalen, och skriv ut vad pekaren pekar på. Exempel på utskrift kan vara:

```
Strängen är: ”Gothenburg University!”.
Pekaren pekar på ”G”, välj en operation:
1) Plus 1
2) Minus 1
4) Avsluta programmet

> 1
Strängen är: ”Gothenburg University!”.
Pekaren pekar på ”o”, välj en operation:
1) Plus 1
2) Minus 1
4) Avsluta programmet

> 1
Strängen är: ”Gothenburg University!”.
Pekaren pekar på ”t”, välj en operation:
1) Plus 1
2) Minus 1
4) Avsluta programmet

>4
Programmet avslutas.
```

1.5.3 Minnesadresser - 1p

Pekarvariabler är helt enkelt heltal som kan tolkas som en viss position i datorns minne: en adress. Anledningen till att vi inte bara använder heltal är att pekare har en egen matematik.

Skriv ett program som skapar tre pekarvariabler: en int*, en char* och en double*. Initiera dem till adressen 0 och skriv ut dem som tal med `printf` (använd `%i`). Addera sedan 1 till alla tre variablerna och skriv ut dem igen.

Vad får du för resultat? Varför är det på detta sätt?

1.5.4 Fält - 1p

Att deklarera ett fält av variabler (en *array)* är i princip ett sätt att allokera en viss mängd minne och få en pekare till det första elementet. Förklara vad som händer i de två programsegmenten nedan och ange speciellt var det finns fel som går igenom en kompilering men ändå inte gör det man kan tänka sig var det förväntade.

a)
```
double dblField [100];
. . .
dblField = dblField + 10;
```

b)
```
double dblField [100], *pDblField;
. . .
pDblField = dblField + 10;
```

1.5.5 Dynamisk allokering av minne – textsträngar - 2p

Med fält (arrayer) måste du bestämma dig för en storlek när du skriver din källkod. Med funktionerna `malloc` och `realloc` kan man allokera en valfri mängd minne och få en pekare till det. Allt minne man allokerar på detta sätt måste man sedan manuellt frigöra med funktionen `free.` Använd dessa verktyg för att skapa ett program som accepterar indata av godtycklig längd. Efter inmatning har skett ska programmet dela upp den inmatade informationen i två separata textsträngar:

```
char *str1, *str2;
```

str1 ska innehålla alla tecken med jämnt index från ursprungstexten medan
str2 ska innehålla alla tecken från ojämna positioner.
Skriv ut de två resulterande strängarna till kommandotolkens fönster.

Tips: `malloc` med släktingar arbetar alltid med mängden i *byte*. Använd specialfunktionen `sizeof` för att ta reda på hur många byte en viss datatyp är. Exempelvis `sizeof(double)`, eller `sizeof(char)`.

1.5.6 Sortering – 2p

Skriv ett program som med olika funktioner sorterar 10 inmatade textsträngar först med avseende på längd och därpå i bokstavsordning. Presentera resultatet med lämplig formatering. Berätta för användaren vilka begränsningar som finns i programmet.

1.5.7 Textsträngar forts - 2p

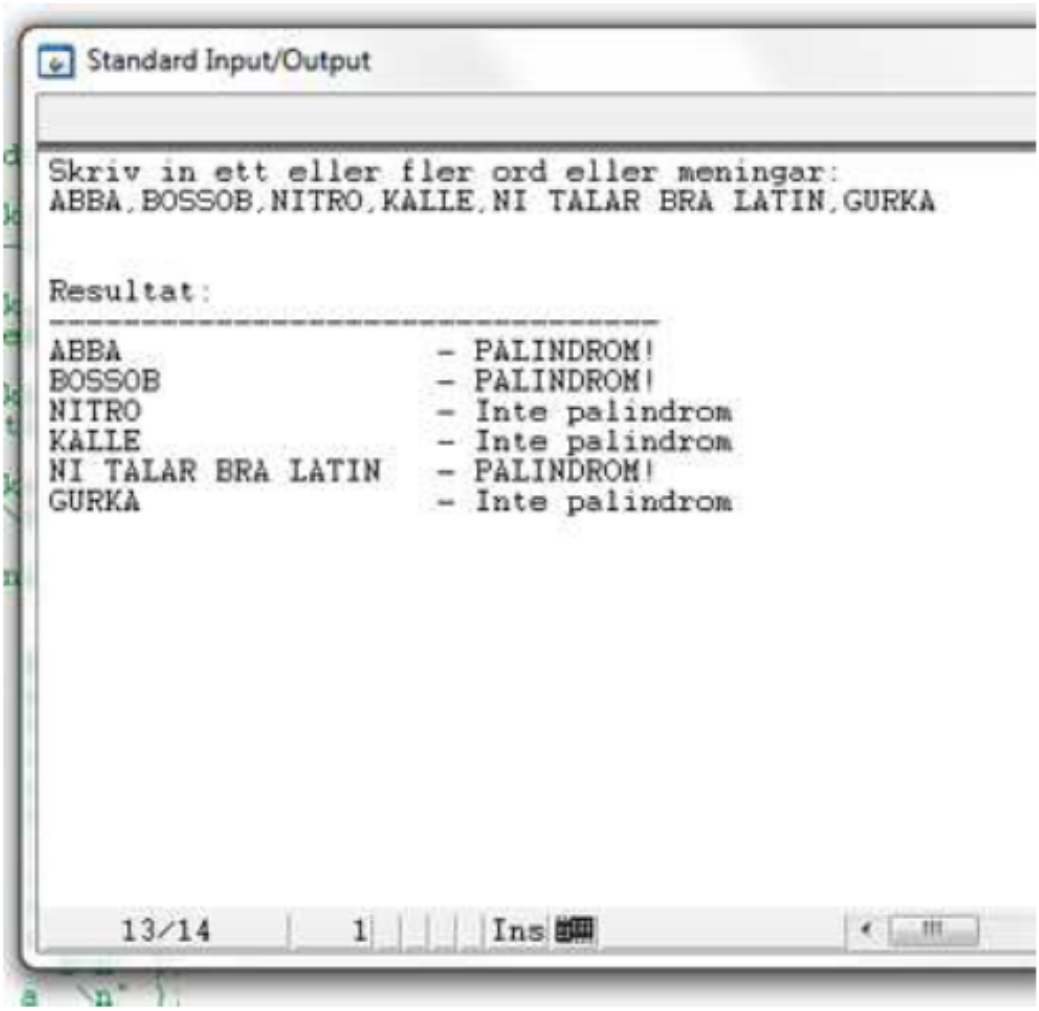
Skriv ett program som avgör om en sträng är ett palindrom eller inte. Programmet ska kunna ta ett enstaka ord som inmatning men även meningar med upp till tio ord.
När användaren trycker ’Enter’ ska programmet avgöra om det matats in ett eller flera ord samt presentera resultatet på ett tilltalande vis (tex i en tabell med två kolumner, se exempel).
Programmet *ska* använda dynamisk minnesallokering.

1.5.8 Fält av pekare - 1p
Skriv ett valfritt program som använder sig av ett fält av pekare till funktioner som alla returnerar ett 32-bitars heltal för att lösa en fritt vald matematisk uppgift.

Följande deklaration finns:

int (*ipFunks [3]) (**void**);

Låt programmet skriva sina resultat till kommandotolkens fönster.



1.5.9 Säker kopiering – 2p
På nittiotalet var en vanligt förekommande metod vid datorintrång att använda ’buffer overflow errors’. Det som utnyttjades var att om för mycket info (”för långa datapaket”) skickades under vissa steg under t.ex. inloggning hamnade en del av paketet direkt i minnet och exekverades – intrång var därmed möjligt. Här ska information kopieras på ett säkert sätt som omöjliggör denna typ av attack.

Skriv ett program med en eller flera funktioner som kopierar information från en sträng till en annan. Funktionen ska dels acceptera godtyckliga teckensträngar av typen (**char** *) eller (**char** [])) som är nollterminerade men även icke-terminerade strängar av samma typer.

Då överlagrade funktioner inte existerar i C måste problemet med stränglängd lösas på annat sätt. Ni väljer fritt vilken metod som avgör huruvida den mottagna strängen är nollterminerad eller ej.

All samlad funktionalitet som beskrivs i detta moment ska vara tillgängligt från ett och samma funktionsanrop. Er deklaration (och definition) ska se ut så här:

char *SafeCopyString (**char** *dest, **const char** *origin, size_t count);

Funktionen ska returnera pekaren till den resulterande strängen och om det är nödvändigt ska den också signalera till användaren att ett fel inträffat under kopieringen. Tänk igenom vilka specialfall det finns. Det är *inte* tillåtet att använda standardfunktionen **strncpy** () eller liknande funktioner. Testa programmet med alla olika aktuella parametrar.

1.6 Poster (structs)

En `struct` är ett sätt att gruppera flera variabler av olika typer till en ny, sammansatt, typ. Denna typ kan sedan användas som alla andra typer: man kan deklarera variabler, man kan ha pekare till dem, man kan använda `sizeof` på dem, och så vidare.

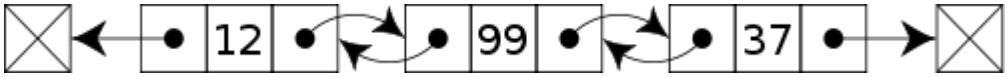
Många vanliga datastrukturer bygger på att man definerar en struct för en *nod* som innehåller dels data, dels pekare till andra likadana noder. På så vis kan man bygga upp en lista, eller ett träd.

Hantering av `struct` har lite speciell syntax (se kommentar nedan) så läs på om det och lös någon eller båda av följande uppgifter.

1.6.1 Länkad lista – 2p

Utgå från följande deklaration av en nod i en dubbelt länkad lista:

```
struct node {
        double data;
        struct node *next;
        struct node
*previous;
};
```



Skriv ett program som har funktioner för att

a) Skapa en ny nod (dvs, en kedja med en ensam nod),
b) lägga till en nod i slutet av en kedja av noder,
c) ta bort en nod på en viss plats i en kedja av noder,
d) gå igenom en kedja och skriva ut alla värden.

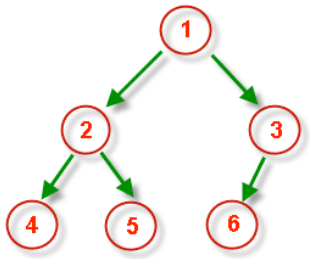
Och demonstrera denna funktionalitet. Ditt program måste allokera noderna dynamiskt, och frigöra minnet för dem efteråt.

1.6.2 Binärträd - 2p

Utgå från följande deklaration av en nod i ett binärträd:

```
struct node {
        int data;
        struct node *parent;
        struct node *left;
        struct node *right;
};
```

Skriv ett program som skapar trädet i figuren till höger. Skapa en funktion som givet en rot-nod skriver ut trädet på något lämpligt sätt. Ditt program måste allokera noderna dynamiskt, och frigöra minnet för dem efteråt.



Kommentar om syntax: Vissa programmerare gillar att förenkla syntaxen för struct med hjälp av en typedef, såhär:

```
struct min_typ_t
{
        double data;
        struct min_typ_t *next;
};
typedef struct min_typ_t min_typ;
```

Efter det kan man använda ”`min_typ`” utan att behöva skriva ”`struct`” hela tiden. Vilket sätt ni väljer är en smaksak, det är ingen rätt eller fel i sammanhanget. Notera att i själva deklarationen av struct är inte typedef giltig ännu, så självreferenser måste alltid göras explicit.

Kapitel 2: Filhantering

Mål: Du ska förstå och kunna hantera filer; läsning, skrivning och kopiering. Du ska lära dig att utnyttja standardfunktioner och strömmar på ett bra sätt.

2.1 Standardfunktioner och filströmmar

2.1.1 Filer - 1p

Det kan ibland vara lite svårt att avgöra vad som finns i en fil och/eller i minnet. Här en övning som testar din förmåga att separera begreppen och att hålla tungan rätt i mun.
Konstruera ett program som först skapar tre matriser:

0.0 1.0	0.0 1.0	0.0 -1.0
-1.0 0.0	1.0 0.0	1.0 0.0

När matriserna är skapade ska programmet skriva ner var och en av dessa till en egen fil med namnen matris1.bin, matris2.bin och matris3.bin. Om filerna redan finns ska programmet skriva över dessa. Data skall skrivas till filerna i binärformat. Efter att samtliga matriser är sparade till fil ska programmet skriva ut matriserna som finns i minnet till **stdout** varpå de minnesresidenta matriserna skall nollställas. Programmet ska nu ånyo ladda in respektive matrisdata till rätt matris och sedan skriva ut den nyligen inlästa informationen till **stdout**.

2.1.2 Textanalys - 1p

Att söka i texter är ju ett mycket generellt programmeringsproblem.
Här ska du skapa ett program som kan avgöra om ett inmatat ord finns i en godtycklig fil. Om ordet hittas ska programmet meddela hur många bytes in filen ordet hittades. Om ordet finns mer är en gång skall samtliga positioner presenteras. Utskrift ska ske till **stdout**.

2.1.3 Mönstermatchning - 1p

Att känna igen mönster i data är användbart för säkerhet, komprimering och kryptering.
Här ska ett program skrivas som kan känna igen mönster i 9 filer fördelade på 3 typer (och kataloger) med ändelserna .1, .2. och .3. Du får filerna av en handledare (tillsammans med två ytterligare kataloger för uppgift 2.1.5)
Målet är att nya filer, som har samma mönster/format som de i de tre katalogerna ska kännas igen och identifieras som av formaten .1 .2 och 3. via att öppna och analysera innehållet (ej ändelsen...).
Vid examination är det bara handledare som kör ditt program. Du har inga möjligheter att ge ytterligare instruktioner.

2.1.4 Komprimering - 2p

Använd en godtycklig text på ca 2-3 sidor som du sparar som en fil och skriv sedan ett program som hittar de fem ord som förekommer mest frekvent. Programmet ska sedan ersätta dessa med varsin ’kod’ som (helst) tar mindre utrymme i minnet. Skriv ut resultatet till en ny fil och meddela i kommandofönstret hur mycket plats som sparats. Skriv en funktion för att dekomprimera filen och kontrollera att originaltexten återfås.

2.1.5 Mönstermatchning forts - 2p

Detta moment kräver att uppgift 2.1.3 är klar och godkänd. Upplägget är samma som i 2.1.3 men nu ska du använda filerna i de sista två katalogerna (med ändelserna .4 och .5)
Formaten är givetvis något svårare att upptäcka; planera dina algoritmer noga innan du påbörjar själva implementationen. Skapa gärna separata program vars enda syfte är att testa olika vägar att analysera informationen. Statistiska modeller kan förmodligen hjälpa dig att upptäcka mönster snabbt.

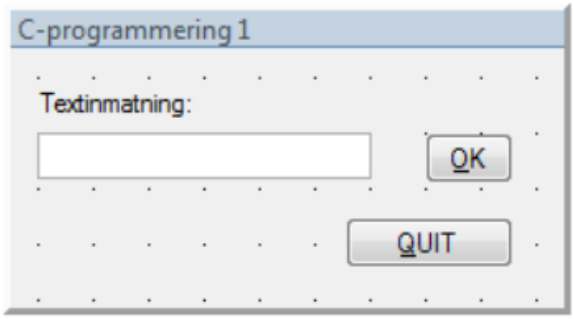
Kapitel 3: Grafiska användargränssnitt

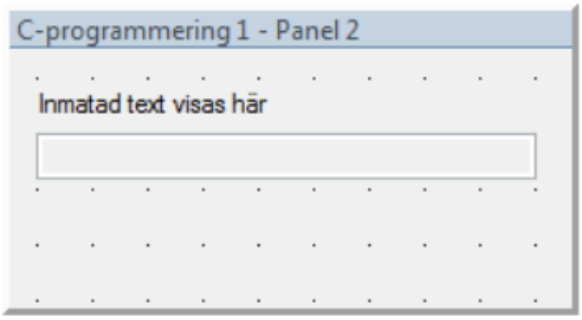
Mål: Att komplettera din kunskap och färdighet i de logiska operationerna i C-språket med grafiska gränssnitt för att underlätta användande. Att upptäcka möjligheterna med inbyggda objekt i LabWindows.
För att komma igång; se kapitel 2 i ”Getting Started...”

3.1 Timer, Event, Stripchart och Attribut

3.1.1 Händelsestyrning - 1p

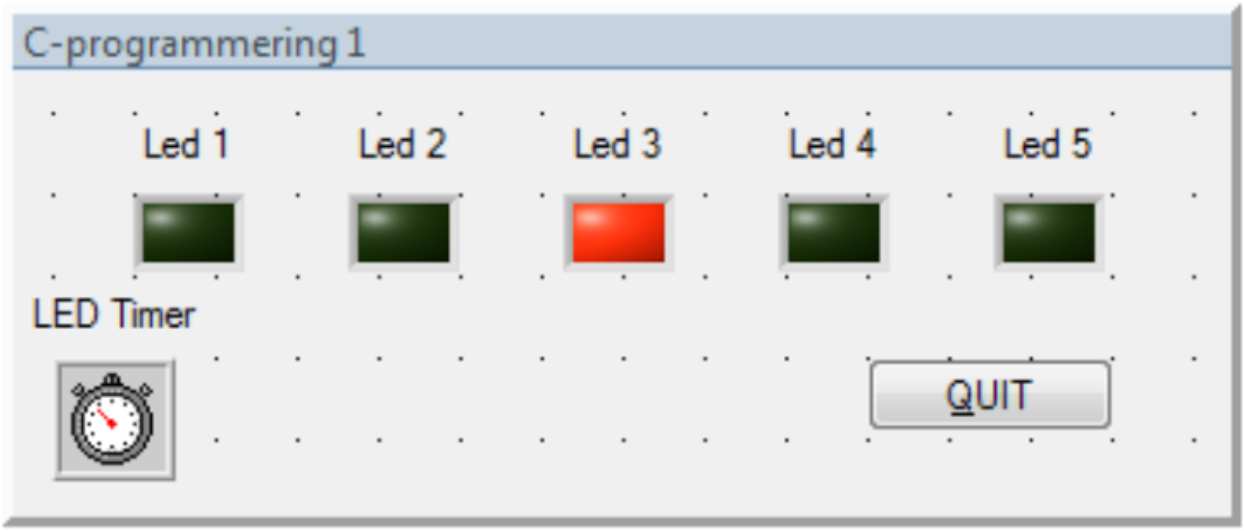
Med hjälp av så kallade 'Events' i LabWindows kan man styra vad som ska hända när användaren gör något speciellt i en grafisk miljö. Skapa ett program där användaren kan mata in en text i ett fält och sedan skicka den till en textkontroll i en annan panel (se exempel) – antingen genom ’Enter’ eller genom vänstermusklick på en OK-knapp. Innehållet i textkontroll två ska inte kunna ändras av användaren.





3.1.2 Timer - 1p

Att kontrollera händelser med en klocka ’timer’ är mycket användbart – här ska du börja lära dig kontrollera timer-funktioner.
Skriv ett program som, för varje timer-tick, tänder och släcker lysdioder på en panel (se exempel). Funktionen ska vara sådan att när klockan tickar första gången tänds dioden längst till vänster, nästa tick så släcks den första dioden och dioden till höger om denna tänds osv tills dess att den sista dioden (längst till höger släcks) – då skall alla dioder vara släckta under ett tick och sen ska cykeln upprepas.

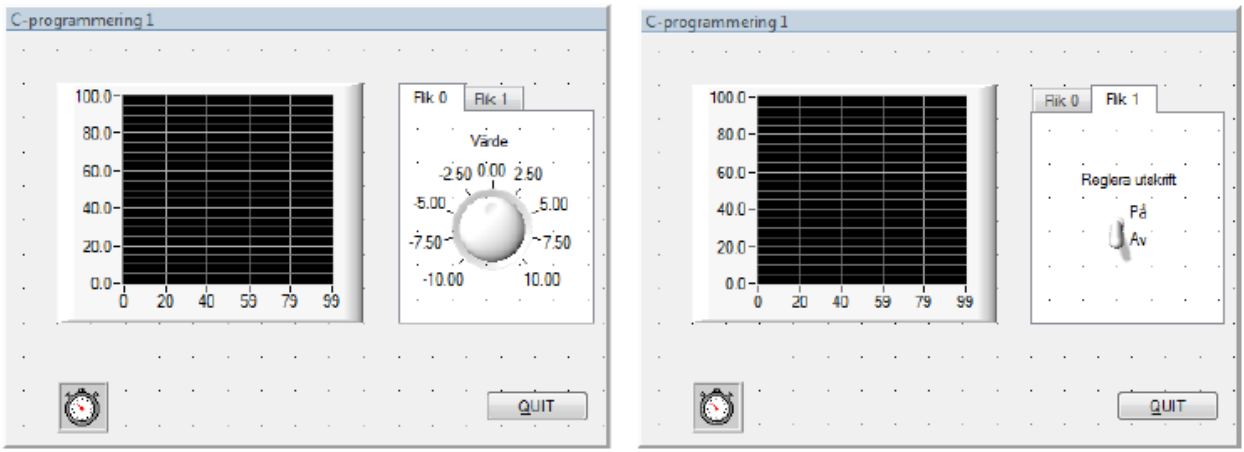


3.1.3 Multipla paneler – effektiv fönsterhantering - 1p
Som du känner igen från ditt operativsystem så öppnas ofta nya paneler beroende av vad du klickar på. Det gäller att programmera på rätt sätt så att inte fönster tappar ”kontakten” med användaren eller programmet.

Konstruera ett program som kan öppna och stänga nya/gamla paneler via en "Huvudpanel". Du får här inte använda dig av utvecklingsmiljöns editor (eller någon form av .uir mall/fil) utan ska skapa huvudpanelen samt alla andra ”underpaneler" genom att använda fördefinierade funktioner i LabWindows. På huvudpanelen ska det finnas en knapp för att skapa en ny panel, stänga den senast skapade panelen, stänga samtliga paneler (utom huvudpanelen) och till sist en knapp för att avsluta hela programmet (och då stänga samtliga öppna paneler).
På varje underpanel ska det ses vilket nummer som den aktuella panelen har och en knapp för att stänga panelen.

3.1.4 Plotta fysikdata - 1p
Skapa ett program som använder sig av en Graph-kontroll för att visa vågformsdata. Du ska ha tre knappar på panelen (Sine, Square och Saw) samt en avslutningsknapp. Programmet ska generera önskade vågformer, hundra punkter, när användaren trycker på respektive knapp (Sinus-, Fyrkant- eller Sågtandsvåg) och visa dessa.

3.1.5 Koppla in/utdata med loggfiler - 2p
Börja med att skapa ett gränssnitt enligt exemplet – obs en panel – två flikar.



Ditt program ska ge användaren möjlighet att använda inställningskontrollen på flik 0 för att ange värdet som skall visas på strip-charten. En timer tickar varannan sekund och vid varje timer-tick ska det aktuella värdet på inställningskontrollen skrivas till en loggfil med namn 'dataXX.dat' i mänskligt läsbart format. XX är ett tal mellan 00 och 99.
Om en loggfil med ett visst nummer redan finns ska programmet skapa en ny loggfil med det närmast lediga numret. Om det finns fler än etthundra loggfiler får du själv avgöra vad som är en lämplig händelse.
På flik 1 finns ett ickefjädrande lägesreglage som anger om utmatnings- och loggningsfunktionerna är aktiva eller ej. I ”Av”-läge skall varekn strip-chart eller loggfiler uppdateras, men återupptas direkt vid ”På” – tänk speciellt på hur du stänger och öppnar loggfilerna.

3.2 Attribut
3.2.1 Attribut - 1p
Börja med att rita upp ett gränssnitt enligt exemplet.

Det färdiga programmet ska kunna styra både position och färg på texten, enklast sker detta via fördefinierade funktioner i LabWindows. Vissa justerar attribut hos panelobjekt direkt, medans andra skapar värden som input till en funktion.

SetCtrlAttribute (...), används just för att göra de justeringar av attribut som behÄovs.
Se kapitel 3 i ”Getting Started...” för information om hur du hittar de formella parametrarna till funktioner som tex SetCtrlAttribute.

3.3 GUI

3.3.1 Egen uppgift – 1-3p

Gör en egen uppgift på GUI i allmänhet - inklusive ett förslag till lösning. Bedöm också svårighetsgraden på uppgiften genom att ge den poäng mellan 1 till 3 p. (Du får inte nödvändigtvis samma poäng vid redovisningen.)

Kapitel 4: Länkning

4.1 Statisk

4.1.1 Statisk länkning - teori - 1p
Förklara utförligt hur statisk länkning påverkar programmet m a p:
- Filstorlek
- Prestanda
- Kompatibilitet/Portabilitet

4.1.2 Statisk länkning - praktik - 3p
I mjukvaruutvecklingsprojekt kan det vara ett krav att det slutliga programmet inte kräver några ytterligare filer utöver den exekverbara huvudfilen (tänk .exe).
För att samla alla programkritiska resurser i en och samma fil måste ett antal aspekter beaktas:

- Hur bygger man ihop samtliga resurser till en och samma fil?
- Hur ansluter man den samlade resursfilen till den exekverbara filen?
- Vilka mjukvaruverktyg krävs för detta?
- Hur kommer upplägget att påverka det färdiga programmet?

Du ska skapa ett program med ett grafiskt gränssnitt, med en huvudpanel med en platshållare för bitmap-bilder samt tre knappar och ett textfält. Du får sex stycken filer att jobba med (från handledare):
- Bildfiler i bitmapformat (5st)
- Textfil

Det färdiga programmets specifikation är:
- Informationen i textfilen och bildfilerna får ej inkluderas i källkoden.
- Det finns endast en fil – det körbara programmet.
- Platshållaren skall kunna visa fem olika bilder, en i taget.
- De tre knapparna bläddrar framåt och bakåt bland bilderna resp. avslutar programmet.
- Vid start laddas textfilen in i textfältet.

Uppgiften kan lösas på många olika sätt.

4.2 Dynamisk

4.2.1 Dynamisk länkning i praktiken - 2p
Här ska du skriva ett program som på enklaste sätt visar på principen genom att två olika funktioners instruktionsförlopp endast ska vara tillgängliga genom två olika, dynamiskt länkade, bibliotek (som du själv byggt).

Programmet har följande specifikation:
- All utskrift sker till kommandotolkens fönster.
- Varje utskriftssteg följs av en radmatning och vagnretur.
- Vid programstart skrivs ”Programstart"
- Programmet anropar sedan en funktion placerad i ett av de två dynamiskt länkade biblioteken. Denna funktion skriver ut "I .dll nummer ett."
- Programmet anropar sedan en funktion placerad i det andra dynamiskt länkade biblioteket. Denna funktion skriver ut "I .dll nummer två."
- Programmet avslutas.