TENTAMEN

Programmering Grundkurs (HI1900)

Skrivtid 13:15-18:15

Fredagen 14 januari 2011

Tentamen består av 8 sidor

Hjälpmedel
Förutom dator med installerad *Code::Blocks*,
Utforskaren, *Acrobat reader* och *Notepad* (inga andra program),
den kurslitteratur som använts under kursen, samt egna anteckningar,
programlistningar och böcker. Dock inga egna disketter, CD-ROM eller USB-minne.

- Under `W:\PROV\C` finns program- och datafiler som kan komma till användning vid lösandet av uppgifterna. Kopiera över dessa till ditt konto.
- Till alla uppgifter ska ett program levereras i form av källkod (C eller CPP-fil). Dina bidrag lägger du i en katalog i roten på `H:`. Katalogen ska ha samma namn som prefixet i din *mailadress*. Exempelvis för Kalle Kula: `HDI02KEKA`. Namnen på lösningarna ska ges `UPPG1.C` till `UPPG8.C`. De är endast dessa filer som kommer att bedömas.

Rättningen görs genom att programmen körs ett antal gånger för olika indata. Om resultatet överensstämmer med det förväntade bedöms programmet som korrekt och ger 2 poäng. Om ett program ej kan kompileras utan fel, är det knappast troligt att det kommer att ge några poäng. I det fall där programmet läser från och eller skriver till en fil, testas programmet oftast med en annan fil än den bifogade.

Betygsgränser (HI1900):

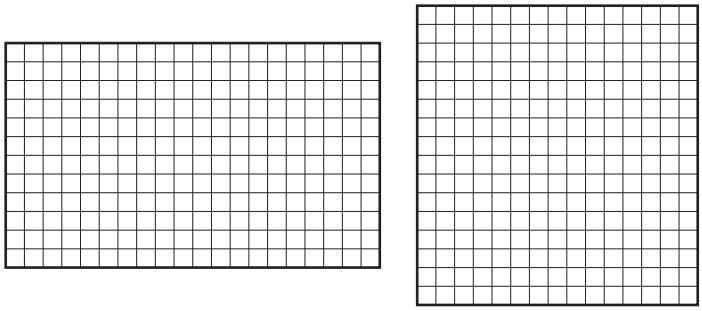
Poäng	14-16	12-13	10-11	8-9	7	6
Betyg	A	B	C	D	E	Fx

Resultatet anslås på kursens hemsida `ingforum.haninge.kth.se/c` i kodat skick.

Lycka till!

Håkan Strömberg

Uppgift 1. Den ändrade rektangeln



Figur 1:

I en rektangel är den längre sidan a meter längre än den kortare. Om man ökar den kortare sidan med b meter och samtidigt minskar den längre med c meter, får man en ny rektangel med *samma area*.

Skriv ett program som tar emot uppgifter om a, b och c och som bestämmer arean hos rektangeln. Alla indata och även arean kan förväntas vara heltal.

```
a ? 8
b ? 4
c ? 5
Arean är 240
```

Lösning:

```
#include <stdio.h>
int main(void){
  int a,b,c,x;
  printf("a ? "); scanf("%d",&a);
  printf("b ? "); scanf("%d",&b);
  printf("c ? "); scanf("%d",&c);
  x=b*(a-c)/(c-b);
  printf("Arean är %d m^2\n",x*(x+a));
}
```

H I första uppgiften testas endast inmatning från tangenbord, utskrift till skärm och tilldelningssats. Uppgiften kräver matematik där man kan ställa upp en förstagradsekvation: Antag att den kortare sidan är x meter. Vi kan nu teckna arean på två sätt

$$\begin{aligned} x(x+a) &= (x+b)(x+a-c) \\ x^2+ax &= x^2+ax-xc+bx+ba-bc \\ cx-bx &= b(a-c) \\ x &= \frac{b(a-c)}{c-b} \end{aligned}$$

Uppgift 2. Slantsingling

Adam har n enkronor. Han kastar upp dem i luften, alla på en gång. När de landat plockar han bort de, där bilden av Karl den XVI Gustav, hamnat uppåt. Han singlar sedan de återstående mynten och plockar åter bort de som hamnat med kungen uppåt. Proceduren fortsätter så tills alla mynt är bortplockade.

Skriv ett program som frågar efter hur många enkronor Adam har och som sedan 10000 gånger simulerar den ovan beskrivna proceduren. Programmet ska därefter ange ett medelvärde av antalet kast han måste göra innan alla mynten plockats bort. Ett körningsexempel:

```
Hur många mynt ? 1000
I medeltal behöver han göra 11.29 kast.
```

Lösning:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void){
  int forsok,antal,slant;
  int nkast=0,nantal,gubbe,n;
  printf("Antal slantar ? ");
  scanf("%d",&antal);
  srand(time(0));
  for (forsok=1;forsok<=10000;forsok++){
    nantal=antal;
    while (nantal>0){
      nkast++;
      n=0;
      for (slant=1;slant<=nantal;slant++){
        gubbe=rand()%2;
        if (gubbe)
          n++;
      }
      nantal=nantal-n;
    }
  }
  printf("I medeltal krävs %.2f\n",nkast/10000.0);
}
```

I uppgiften testas loop och användandet av slumptal.

7-8 Programmet tar emot antalet slantar som ska användas vid simuleringen.

9 Slumptalsserien ges ett 'slumpmässigt' startvärde som bestäms av datorns klocka.

10-22 För varje varv i denna loop utförs en av 10000 försök.

11 I början av försöket sätts `nantal` det antal mynt, `antal`, som kastas första gången.

12-21 Så länge det finns mynt kvar ska vi fortsätta att kasta alla mynt vi har kvar.

13 Vi ökar på räknaren för antalet kast.

14 `n` håller reda på hur många av de kastade mynten som visar 'gubbe' (och som senare ska tas bort) och nollställer `n` innan vi börjar singla slantarna.

15-19 Vi singlar nu ett mynt i taget

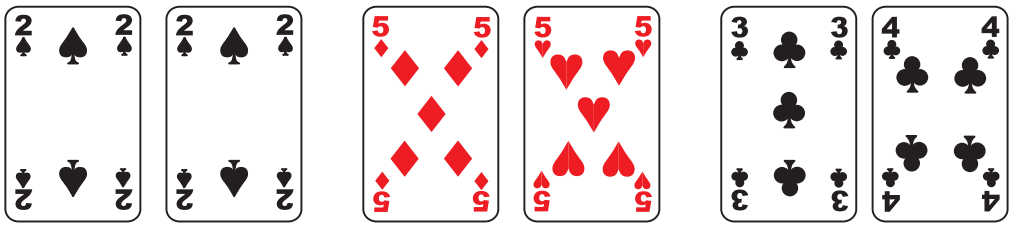
16 Här får vi ett av slumptalen 0 eller 1.

17-18 Om `gubbe` är lika med 1, kom Karl Gustav upp, och detta mynt ska tas bort. Därför ökar vi `n` med 1.

20 Nu drar vi bort `n` från `nantal` och nästa gång ska vi alltså kasta `nantal` mynt.

23 När vi hamnar här är alla 10000 försöken gjorda och vi vet hur många kast vi behövt göra. Genom att dividera `nkast` med 10000 får vi det efterfrågade medelvärdet.

Uppgift 3. Patiens



Identiska kort
3 poäng

Samma valör
2 poäng

Samma färg
1 poäng

Här introducerar vi ”Haningepatiensen”. Vi startar med två vanliga kortlekar, som vi blandar var och en för sig. De två lekarna läggs på bordet med baksidan upp. Nu vänder man samtidigt ett kort från varje lek upprepade gånger tills alla kort i de två lekarna vänts upp. Man får nu poäng för varje uppvänt par enligt följande:

- Om två uppvända **kort är identiska** får man 3 poäng.
- Om två uppvända kort **har samma valör** får man 2 poäng.
- Om två uppvända kort **har samma färg** får man 1 poäng.
- I övriga fall ges 0 poäng.

Det är den totala poängen för de 52 paren av kort vi är ute efter.

Skriv ett program som utför patiensen och beräknar den totala poängen. Korten i de två lekarna finns på filerna `lek1.txt` och `lek2.txt`. Var och en av filerna består av 52 rader. På varje rad presenteras ett kort genom en sträng bestående av två tecken. Det översta kortet i leken finns på första raden i filen och så vidare. Det första tecknet talar om vilken färg kortet har och det andra vilken valör enligt följande

Första tecknet	Betyder
S	♠
H	♡
D	♢
C	♣

Första tecknet	Betyder
2...9	2...9
T	10
J	Knekt
Q	Dam
K	Kung
A	Äss

Ett körningsexempel

```
Totalpoängen blev 16
```

Lösning:

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void){
5    FILE *fil1,*fil2;
6    char kort1[3],kort2[3];
7    int i,tot=0;
8
9    fil1=fopen("lek1.txt","rt");
10   fil2=fopen("lek2.txt","rt");
11   for(i=1;i<=52;i++){
12     fscanf(fil1,"%s",kort1);
13     fscanf(fil2,"%s",kort2);
14     if(strcmp(kort1,kort2)==0)
15       tot=tot+3;
16     else
17       if (kort1[1]==kort2[1])
18         tot=tot+2;
19       else
20         if (kort1[0]==kort2[0])
21           tot++;
22   }
23   fclose(fil1);
24   fclose(fil2);
25   printf("Slutpoängen=%d\n",tot);
26 }
```

I uppgiften testas att man kan läsa data från fil, att man kan läsa in en textsträng, ta fram ett tecken från strängen och jämföra med ett annat tecken. Att man kan hantera `if - else`.

9-10 Vi öppnar de två filerna för läsning.

11-22 Vi ska nu läsa in och jämföra 52 kort

12-13 Här sker inläsningen. Ett kort från varje fil.

14-15 Om korten är identiska ökar vi räknaren `tot` med 3. I `tot` kommer så småningom slutpoängen att finnas.

17-18 Annars om korten har samma valör ökar man `tot` med 2

20-21 Annars om korten har samma färg ökar man `tot` med 1

23-25 Vi stänger filerna och skriver ut resultatet.

Uppgift 4. Vokaler och konsonanter

Figur 2: *Fyra av de eftersökta orden*

Här är vi på jakt efter de ord i svenska språket där varannan bokstav är en *vokal* och varannan en *konsonant*, som till exempel de fyra orden i figur 2.

Skriv ett program som tar reda på hur många sådana ord det finns på filen `ord.txt`. Här en tabell över vokaler och konsonanter:

Vokaler	a e i o u y
Konsonanter	b c d f g h j k l m n p q r s t v w x z

OBSERVERA: Vi undviker vokalerna å ä ö i detta problem. Filen `ord.txt` inleds med ett tal `n` som anger hur många ord filen innehåller. Därefter följer `n` rader med ett ord på varje. Alla med < 40 bokstäver `a...z`.

Ett körningsexempel:

```
Det finns 3839 sådana ord
```

Lösning:

```c
#include <stdio.h>
#include <string.h>

int vokal(char b){
  char vokaler[]="aeiouy";
  int i;
  for(i=0;i<strlen(vokaler);i++)
    if(vokaler[i]==b)
      return 1;
  return 0;
}

int main(void){
  FILE *fil;
  int i,j,antal,v,ok,n=0;
  char ord[40];
  fil=fopen("saol3.txt","rt");
  fscanf(fil,"%d",&antal);
  for(i=1;i<=antal;i++){
    fscanf(fil,"%s",ord);
    v=vokal(ord[0]);
    ok=1;
    for(j=1;j<strlen(ord);j++)
      if(v!=vokal(ord[j]))
        if(v==1)
          v=0;
        else
          v=1;
      else
        ok=0;
    if(ok)
      n++;
  }
  printf("Det finns %d ord\n",n);
}
```

I stort klarar man sig med de kunskaper som krävs för uppgift 3, Att läsa en fil och hantera strängar. Det kan vara lämpligt men ej nödvändigt att använda en funktion som avgör om en bokstav är vokal eller konsonant. Svårigheten i uppgiften består i att logiskt formulera 'varannan vokal och varannan konsonant'

17-18 Vi öppnar filen och tar reda på hur många ord det finns på filen.

19-33 I den här loopen ska vi läsa in ett ord i taget och ta reda på om det har efterfrågade egenskaper.

20 Vi läser in ett ord

21 Om ordet inleds med en vokal får `v` värdet `1` annars `0`. Vi använder oss här av funktionen `vokal`, som vi återkommer till.

22 `ok` är en flagga som har värdet `1` så länge ordet är godkänt och det är det ju innan vi börjar testa.

23-30 För varje bokstav i ordet ska vi nu testa om ordet är godkänt. För att ta reda på längden av ordet använder vi `strlen`.

24 Om aktuell bokstav inte är av samma 'sort' (vokal eller konsonant) som förra så är ordet fortfarande godkänt.

25-28 `v` togglar (slår om) nu från `1` till `0` eller tvärt om, för att ha rätt mode inför nästa bokstav som ska jämföras.

29-30 Om aktuell bokstav är av samma 'sort' i 24 är ordet underkänt och flaggan `ok` slår om till `0`.

31-32 Om programmet tagit sig igenom loopen $23-30$ och `ok` fortfarande är `1` har vi hittat ett godkänt ord och ökar därför `n`, som håller reda på antalet godkända ord.

34 Vi skriver ut resultatet.

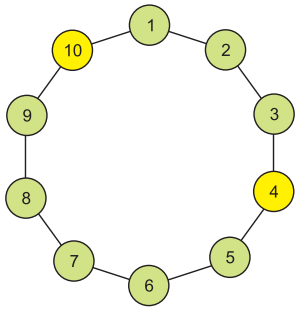
4 Funktionen `vokal` tar emot ett tecken (bokstav).

5 I strängen `vokaler` lagrar vi de 6 vokalerna.

7-9 I loopen jämför vi varje bokstav i `vokaler` med `b`, den 'inskickade'. Om villkoret blir sant är `b` en vokal och vi kan returnera `1`.

10 Om hela loopen genomlöps utan att likhet uppstår kan vi nu returnera `0`, alltså att `b` är en konsonant.

Uppgift 5. Ringlek



Figur 3:

Adam ställer upp $n \leq 100$ personer i en ring. Personerna är numrerade medurs från 1 till n. Han startar på person 1 och räknar personerna medurs. Var tredje person får lämna ringen.

Exempel med $n = 10$. Den första som får lämna ringen är person nummer 3, därefter $6, 9, 2, 7, 1, 8, 5$. Nu återstår två personer 4 och 10.

Skriv ett program som inledningsvis frågar efter antalet personer i ringen och därefter tar reda på var han ska placera sig själv och sin vän Bertil, för att de ska bli sist kvar. I exemplet ovan ska han alltså vika platserna 4 och 10 för sig själv och Bertil. Ett körningsexempel;

```
Antal personer i ringen ? 40
De två eftertraktade platserna är 13 och 28
```

Lösning:

```
#include <stdio.h>
int next(int r[],int n,int p){
  do{
    p++;
    if (p>n)
      p=1;
  }while(r[p]==1);
  return p;
}

int main(void){
  int antal,utraknade=0,ring[101]={0},pekarut=0,i;
  printf("Hur många i ringen ? ");
  scanf("%d",&antal);

  while (utraknade<antal-2){
    for (i=1;i<=3;i++)
      pekarut=next(ring,antal,pekarut);
    ring[pekarut]=1;
    utraknade++;
  }

  printf("\nVännenas placering ");
  for (i=1;i<=antal;i++)
    if (ring[i]==0)
      printf("%d ",i);
}
```

I denna uppgift ska man kunna hantera en array och förstå hur man kan med hjälp av den skapa en 'ring'.

H Arrayen `ring` är ursprungligen dimensionerad för att kunna hålla reda på `100` personer i ringen. `ring` nollställs vid deklarationen. Detta innebär att alla ännu finns kvar i ringen. Vi kommer att använda index `1` till `antal`, där `antal` är det antal som från början finns i ringen. När någon blir uträknad kommer vi att placera `1` på motsvarande plats i `ring`. När det endast finns två `0`:or kvar i arrayen är vi klara och kan skriva ut platsen där dessa `0`:or finns.

13-14 Vi tar reda på hur många det ska finnas i ringen.

16-21 För varje varv i loopen räknar vi ut en person. Så länge `utraknade` är mindre än `antal`−`2` fortsätter räknandet.

17-18 Vi ska hitta tre personer som finns kvar i ringen. Den tredje ska plockas bort. Här har vi använt en funktion för att leta rätt på `0`:or (personer) i arrayen. Funktionen `next` tar emot `ring`, `antal` och `pekarut`, den person som just nu pekas ut.

3-7 Loopen letar reda på första `0`:an som kommer efter index `pekaut` i arrayen. Om `p` blir större än `n` (`antal`) fortsätter letandet från index `1`.

8 När vi hittat en `0`:a returnerar vi numret på den person som pekas ut.

18 Det returnerade värdet hamnar i `pekarut`, som går in i nästa funktionsanrop.

19 När loopen är färdig pekar `pekarut` på den person som ska lämna ringen. Vi sätter därför in en `1` på den platsen i `ring`.

20 Vi räknar upp antalet uträknade.

24-26 När vi hamnar här vet vi att det finns två `0`:or kvar i arrayen. Genom en loop kan vi nu ta reda på var och skriva ut dessa platser.

H Det är på intet sätt nödvändigt med en funktion i detta program. Även om det känns 'knöligt' i början, att skriva funktioner, så kommer det att underlätta framöver för den som vill lära sig programmera.

Uppgift 6. Järnvägen

En järnvägssträcka startar i station A. Närmaste station heter B och så vidare upp till antalet $n \leq 10$ (upp till bokstaven J). Innan färden startar känner man till antalet passagerare som köpt biljett mellan olika destinationer. Här ett exempel

	Till					
Från	A	B	C	D	E	F
A	–	83	10	12	63	57
B	–	–	76	78	39	54
C	–	–	–	88	43	30
D	–	–	–	–	2	14
E	–	–	–	–	–	56

Vi ser att 83 personer kommer att resa från station A till station B. Att 30 personer kommer att resa mellan C och F.

Skriv ett program som tar reda på *mellan* vilka stationer det befinner sig flest passagerare ombord på tåget.

Filen `tåg.txt` inleds med en rad som anger antalet stationer $n \leq 10$. På nästa rad anges antalet sträckningar m, som har bestämts genom formeln $m = \frac{n(n-1)}{2}$. Därefter följer m grupper. Första raden i varje grupp anger med en bokstav (versal) från vilken station biljetten köpts. På nästa rad en bokstav (versal), som anger till vilken station biljetten köpts. På den tredje raden i gruppen, antalet biljetter som sålts mellan dessa två stationer. Körningsexempel:

```
Största antalet passagerare, 464, fanns på tåget mellan C och D
```

Lösning:

```
#include <stdio.h>

int main(void){
  int tab[10][10];
  int i,j,n,n2,a,sum=0,station,max=0;
  char t[2],f[2],s1,s2;
  FILE *fil;
  fil=fopen("tåg3.txt","rt");
  fscanf(fil,"%d",&n);
  fscanf(fil,"%d",&n2);
  for (i=1;i<=n2;i++){
    fscanf(fil,"%s",f);
    fscanf(fil,"%s",t);
    fscanf(fil,"%d",&a);
    tab[f[0]-'A'][t[0]-'A']=a;
  }
  fclose(fil);
  for (station=0;station<n-1;station++){
    sum=0;
    for (i=0;i<=station;i++)
      for (j=station+1;j<n;j++)
        sum=sum+tab[i][j];
    if (sum>max){
      max=sum;
      s1=station+'A';
      s2=station+'B';
    }
  }
  printf("Mellan %c och %c fanns %d ombord\n",s1,s2,max);
}
```

I första hand testar uppgiften kunskaper om att kunna finna ett maximum. Dessutom har man nytta av att kunna hantera en tvådimensionell array (matris). Man bör dessutom kunna översätta bokstaven A till 0, B till 1 och så vidare, till tal som ska användas som index i matrisen.

Om data lagras i en matris `tab`, med samma form som figuren i problemtexten, kan vi konstatera att antalet passagerare som är på tåget mellan station A och B är lika med summan av talen som finns i rad A med början från kolumn B till kolumn F. Antalet passagerare mellan station B och station C är summan av talen i de två första raderna A och B, med början från kolumn C och till kolumn F. Alltså handlar det om att upprepade gånger summera en del av talen i matrisen `tab`.

8-10 Vi öppnar filen och läser in antalet stationer `n` och antalet data `n2`.

11-16 En loop som utförs `n2` gånger. I varje varv läses beteckningarna för två stationer in tillsammans med antalet passagerare som löst biljett mellan dessa stationer.
För att översätta till exempel `C` till talet 2 utför vi beräkningen `'C'-'A'`. För två liknande beräkningar får vi index till den plats i `tab` där antalet passagerare ska placeras.

17 Alla data har lästs in och filen kan stängas.

18-28 I denna loop tar vi så reda på det eftersökta maximivärdet. Om det finns `n` stationer finns det `n-1` avstånd mellan närliggande stationer.

19 Den summa som ska beräknas sätts till `0`

20-22 Den dubbelloop som summerar talen i aktuell del av matrisen.

23-27 Här undersöker vi om den just erhållna summan är den hittills största. Om så är fallet sparar vi uppgifter om rekordet och mellan vilka stationer det erhölls.

29 Vi kan skriva ut resultatet

Uppgift 7. Dubbletter

I ett register, i form av en binärfil med namnet `register.dat`, finns 10000 personer med uppgift om namn och personnummer med postbeskrivningen:

```
struct person{
  char namn[48];
  char personnr[12];
};
```

Man befarar nu att det i registret finns flera personer, som av misstag, fått samma personnummer. Skriv ett program som tar reda på hur många dubbletter det finns och dessutom skriver ut dessa personnummer. Ett körningsexempel:

```
110917-8812
150706-7237
000714-2979
Det finns 3 dubbletter
```

Lösning:

```
#include <stdio.h>
#include <string.h>

struct person{
  char namn[48];
  char personnr[12];
};

int main(void){
  FILE *infil;
  struct person p;
  char alla[10000][12];
  int i,j,antal=0;

  infil=fopen("register.dat","rb");
  for(i=0;i<10000;i++){
          fread(&p,1,sizeof(struct person),infil);
          strcpy(alla[i],p.personnr);
  }
  fclose(infil);
  for(i=0;i<9999;i++)
    for(j=i+1;j<10000;j++)
      if(strcmp(alla[i],alla[j])==0){
        printf("%s\n",alla[i]);
        antal++;
      }
   printf("Det finns %d dubbletter\n",antal);
}
```

Traditionsenligt en enkel uppgift som visar att man kan hantera en binär fil. Man ska också förstå hur man i en array finner dubbletter bland elementen.

4-7 Den postbeskrivning som var given i problemtexten.

11 En variabel av typ `struct person` deklareras.

12 I en strängarray gör vi plats för 10000 personnummer.

15 Vi öppnar den binära filen för läsning.

16-19 Genom loopen läser vi in de 10000 posterna från filen och kopierar personnumret till strängarrayen `alla`.

20 Alla data är inlästa och vi kan stänga filen.

21-26 Genom en dubbelloop jämför vi alla par av av personnummer.

23 Då två personnummer är lika skriver vi ut det och ökar samtidigt räknaren `antal`.

27 Vi kan skriva ut hur många dubbletter vi träffat på.

Uppgift 8. Spiral

H	E	J	·	V
R	R	A	R	A
U	T	!	·	D
N	N	U	R	·
S	·	T	E	D

Figur 4: *Hej.vad.det.snurrar.runt!*

Meddelandet i figur 4 får man fram genom att följa rutorna som i en spiral. Början av meddelandet läser man i kvadratens översta rad. Därefter viker man nedåt och läser kolumnen längst till höger. När vi når nedersta raden läser vi baklänges fram till första kolumnen. Sedan är det dags att läsa första kolumnen nedifrån och upp. Men vi stannar nu på andra raden och viker i stället åt höger. Till sist når vi utropstecknet i kvadratens mitt.

Skriv ett program som från filen `spiral.txt` tar emot en kvadrat av tecken och skriver ut det meddelande man får då kvadraten läses som beskrivits ovan. Filen inleds med ett tal n, $3 \leq n \leq 9$, som anger kvadratens sida. Därefter följer n rader med n tecken på varje. Punkt (.) har här använts i stället för mellanslag.

Ett körningsexempel

```
Hej.vad.det.snurrar.runt!
```

Lösning:

```
#include <stdio.h>
int main(void){
  FILE *fil;
  char str[10],tab[11][11];
  int i,j,rikt[4][2]={{0,1},{1,0},{0,-1},{-1,0}};
  int rad=1,kol=0,n,r=0,rbyte=0;

  for(i=0;i<11;i++)
    for(j=0;j<11;j++)
      tab[i][j]='*';

  fil=fopen("spiral.txt","rt");
  fscanf(fil,"%d",&n);
  for(i=1;i<=n;i++){
    fscanf(fil,"%s",str);
    for(j=0;j<n;j++)
      tab[i][j+1]=str[j];
  }
  fclose(fil);

  do{
    while(tab[rad+rikt[r][0]][kol+rikt[r][1]]!='*'){
      rad=rad+rikt[r][0];
      kol=kol+rikt[r][1];
      printf("%c",tab[rad][kol]);
      tab[rad][kol]='*';
    }
    r++;
    if(r>3) r=0;
    rbyte++;
  }while(rbyte<2*n);
}
```

Den sista uppgiften, som förstås får lova att vara lite 'trixig'. Återigen handlar det om att kunna hantera en matris. Denna gång innehållande tecken.

H Vi startar med att läsa första raden. Under det arbetet stegar vi uppåt i kolumnerna. Vi kallar steget $\Delta x = 1$. När vi når kanten av matrisen ska vi läsa nedåt utefter sista kolumnen. $\Delta x = 0$ och $\Delta y = 1$. När vi sedan läser baklänges på sista raden i matrisen sätter vi $\Delta x = -1$ och $\Delta y = 0$. Vi kommer så till första kolumnen och läser nedifrån och upp, med $\Delta x = 0$ och $\Delta y = -1$. Detta beskriver första varvet i spiralen.

4 `tab` kommer att rymma hela spiralen.

5 `rikt` är en matris, 4×2, som innehåller Δy och Δx (i den ordningen), som de har beskrivits ovan.

8-10 Med hjälp av en dubbelloop fyller vi först hela teckenmatrisen med asterisker.

12-13 Vi öppnar filen och läser in matrisens storlek till `n`.

14-18 Vi läser in en rad i taget från filen och flyttar över tecknen till `tab`. Eftersom vi startar med första tecknet i `tab`[1][1] förstår vi att texten kommer omges av en ram med asterisker.

19 All data är nu inläst och filen kan stängas.

21-31 Den dubbelloop som kommer att skriva ut texten.

22 Eftersom `r` från början är lika med 0, kommer vi att från `rikt`[0] få $\Delta y = 0$ och $\Delta x = 1$. När vi lägger dessa tal till `rad` respektive `kol`, får fram den plats i vilken aktuell bokstav finns.

23-24 Vi uppdaterar `rad` och `kol` genom att lägga till aktuellt Δy och Δx.

25 Vi skriver ut tecknet

25 och skriver över det just utskrivna tecknet med en asterisk.

22-27 Loopen snurrar tills vi träffar på en asterisk.

28 Vi byter riktning genom att öka på `r`.

29 Om `r`> 3 har vi fullföljt ett varv och det är dags att börja på nästa, genom att sätta `r`$= 0$.

30-31 Vi räknar antalet riktningsbyten i spiralen och när detta antal uppgår till $2n$ har vi skrivit ut hela meddelandet.

H Det är viktigt att programmet i denna uppgift kan klara av olika storlekar på spiral, eftersom lösningen annars enbart kunde bestå av ett antal `printf`-satser. Här är det kanske svårare att förstå beskrivningen av algoritmen i ord än att direkt läsa och förstå koden.