UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	IN 110 — Algoritmer og datastrukturer
Eksamensdag:	18. mai 1992
Tid for eksamen:	9.00 – 15.00
Oppgavesettet er på 8 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle trykte og skrevne

Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Merk:

Oppgavene kan naturlig løses i den rekkefølge de står, men det er ikke noe i veien for å hoppe fram og tilbake, bare man har lest de foregående oppgavene nøye.

Programmer skal skrives enten i Simula eller Pascal. Oppgavene er formulert ut fra at programmene skal skrives i Simula. De som vil bruke Pascal skal erstatte de angitte klassedeklarasjoner med tilsvarende record-deklarasjoner, samt gjøre andre opplagte småjusteringer. Du behøver ikke gi noen fullstendig dokumentasjon av programmene, men du skal skrive noen få linjer som gir leseren nøkkelen til forståelse av programmet. Du kan anta at leseren kjenner problemstillingen i oppgaven meget godt.

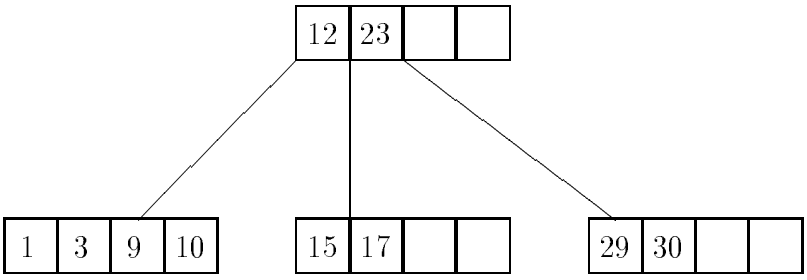
Alle steder der det er spørsmål etter et program (eller en prosedyre eller algoritme) skal man skrive dette helt ut, og *ikke* bare henvise til liknende programmer i f.eks. boka.

Lykke til !
Olav Lysne og Anne Salvesen

(Fortsettes side 2.)

1 B-trær 10%

Vi har følgende B-tre:



1-a

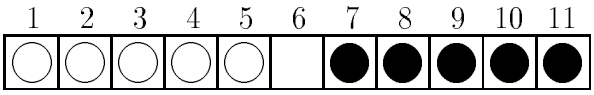
Vis hvordan dette treet blir seende ut etter at man har satt inn tallet 2.

1-b

Hvordan vil treet se ut dersom vi i stedet for å sette inn tallet 2 fjerner tallet 29.

2 Spill 45%

Vi skal i denne oppgaven ta for oss et enkelt spill, og lage noen prosedyrer for å analysere dette. Selve spillet er ment for én person, og foregår på et brett som har et antall felter som ligger etter hverandre. Disse feltene er nummerert $1, 2 ...., \texttt{lengde}$. Idet spillet begynner er det plassert $2n$ brikker, n hvite og n svarte, på brettet, slik at de hvite ligger på de n feltene lengst til venstre, og de svarte ligger på de n feltene lengst til høyre. Det skal være nøyaktig ett ledig felt, nemlig det midterste, nummerert $n + 1$. Vi har derfor $\texttt{lengde} = 2n + 1$. For lengde $= 11$ vil situasjonen ved starten av spillet være følgende:

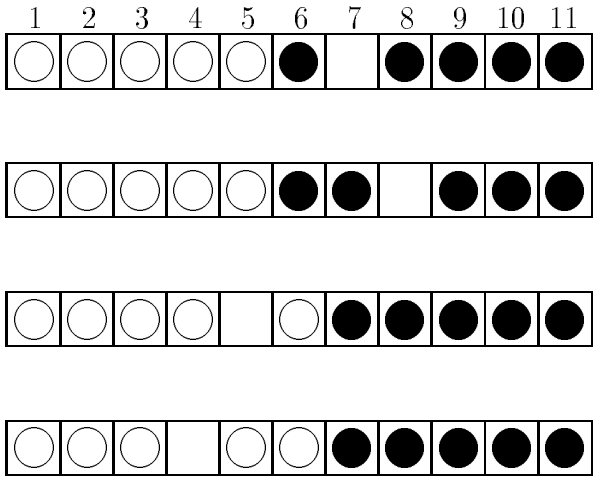


(Fortsettes side 3.)

Spillet går ut på å få de hvite og de svarte brikkene til å bytte plass, slik at de svarte brikkene ligger på venstre side og de hvite brikkene ligger på høyre side. Brikkene skal flyttes en og en etter følgende regler

- Brikkene kan ikke flyttes lenger enn to felt av gangen, og de kan bare flyttes til det til enhver tid ledige feltet.
- Hvite brikker kan kun flyttes mot høyre, og sorte brikker kan kun flyttes mot venstre.

Etter ett flytt er det altså fire mulige situasjoner:



Et brett som inneholder en spillsituasjon kan representeres med en character array

```
CHARACTER ARRAY brett(1:lengde);
```

Cellene i arrayen inneholder en av tegnene **H**, **S** og **L** for henholdsvis hvit brikke, svart brikke og ledig.

2-a

Anta at du har følgende klasse-deklarasjon som kan benyttes til å bygge en liste av mulige flytt i spillet:

(Fortsettes side 4.)

```
CLASS flytt;
BEGIN
  INTEGER fra, til;
  ref(flytt) neste;
END;
```

De to variablene `fra` og `til` skal inneholde hvor i arrayen `brett` en brikke flyttes henholdsvis fra og til.

Skriv prosedyren

```
ref(flytt) PROCEDURE finnflytt(brett);
CHARACTER ARRAY brett;
```

som tar en spillsituasjon i arrayen `brett` som parameter, og som returnerer med en peker til en liste av `flytt`-objekter som angir hvilke flytt det er mulig å gjøre fra den gitte situasjonen. Du kan anta at arrayen `brett` går fra 1 til `lengde`.

2-b

La oss nå tenke oss at vi ønsker å studere alle de mulige forløpene av spillet. Vi kunne da tenke oss disse lagt ut som et tre hvor rotnoden inneholder en representasjon av brettet slik det er ved begynnelsen av spillet. Under rotnoden er det fire sub-noder som angir situasjonene etter ett flytt. Fra hver av disse fire nodene går det maksimalt 4 kanter (muligens færre) til noder som angir situasjonen etter 2 flytt osv. Dermed vil man kunne finne hvert enkelt spillforløp som en sti fra rotnoden og nedover i treet.

Dersom `lengde` er rimelig stor, vil det treet som er skissert over bli for stort til å kunne lagres eksplisitt. I stedet ønsker vi oss en prosedyre som er i stand til å gi oss den informasjonen vi ønsker fra dette treet uten å lagre det i sin helhet.

Anta at du har

```
CHARACTER ARRAY brett(1:lengde);
```

som en globalt deklarert datastruktur, og at du har prosedyren

```
PROCEDURE bytt(i,j); INTEGER i,j;
```

(Fortsettes side 5.)

som bytter om innholdet i posisjon `i` og `j` i arrayen `brett`. Lag prosedyren

```
PROCEDURE generer(i); INTEGER i;
```

som skriver ut alle mulige situasjoner man kan ha etter nøyaktig `i` flytt. Du skal anta at ved kallet av prosedyren inneholder `brett` starttilstanden i spillet. Dersom det er flere spillforløp som leder til samme situasjon skal denne situasjonen skrives ut én gang for hvert forløp. Hvert prosedyrekall skal ikke lagre annen informasjon om tidligere spilltilstander enn det som til enhver tid ligger i arrayen `brett`. Du kan benytte deg av prosedyren `finnflytt` fra oppgave 2-a selv om du ikke har programmert denne.

2-c

Siden det til enhver tid vil være kun ett felt det er mulig å flytte til, vil et spill (en sekvens av flytt) kunne beskrives som en sekvens av tall mellom 1 og `lengde`. Hvert tall i denne sekvensen angir hvilket felt det ble flyttet fra. Vi skal i denne oppgaven lage en prosedyre som finner og skriver ut et spesielt spillforløp.

Anta at vi har en spesiell spillsituasjon, og at vi lurer på om det er mulig å komme fram til denne situasjonen med mindre enn `k` trekk. Lag en prosedyre

```
PROCEDURE finnspill(situasjon,k);
CHARACTER ARRAY situasjon; INTEGER k;
```

som skriver ut en sekvens av trekk som er kortere enn `k`, som leder fram mot den spillsituasjonen som er beskrevet i `situasjon`, og som er slik at det ikke er noen kortere sekvens som også leder fram til den samme situasjonen. Dersom ingen slik sekvens finnes skal ingen ting skrives ut. Sekvensen av trekk skal skrives ut som en sekvens av tall som angir hvilke felt det ble flyttet fra.

Du skal anta at denne prosedyren skal kjøres på en maskin med lite internminne, slik at det ikke er plass til å lagre alle spillsituasjonene etter `k` trekk på en gang. Du skal derfor ikke bruke en eksplisitt kø.

Hint:
Modifiser prosedyren `generer` fra oppgave 2-b, slik at den bruker en global hjelpearray til å holde rede på hvilke trekk som leder frem mot den situasjonen som til enhver tid betraktes. Benytt denne prosedyren i søket.

(Fortsettes side 6.)

2-d

Kunne man løst oppgaven over uten å benytte den hjelpearrayen som omtales i hintet? Forklar!

3 Grafer 45%

Vi skal fremdeles holde oss til det spillet som ble beskrevet i den forrige oppgaven (les evt. fram til oppgave 2-a). Nå skal vi imidlertid tenke oss at vi har et brett som er så lite at alle spillsituasjonene kan lagres eksplisitt.

Anta at vi har klassedeklarasjonen

```
CLASS situasjon;
BEGIN
  CHARACTER ARRAY brett(1:lengde);
  REF(situasjon) ARRAY nestesituasjon(1:4);
END;
```

som kan benyttes til å lagre alle spillsituasjonene i en graf. Arrayen `nestesituasjon` skal inneholde pekere til de noder som angir spillsituasjonene etter ett flytt fra denne situasjonen. Dersom en node har færre enn fire etterfølgersituasjoner skal dette markeres ved at noen av pekerene er NONE.

3-a

Skriv en prosedyre

```
PROCEDURE generer(n);
REF(situasjon) n;
```

som oppretter en graf for alle spillsituasjoner som kan forekomme fra situasjonen i noden `n`. En node i grafen skal være entydig identifisert med en spillsituasjon, slik at ingen situasjon skal lagres to ganger. Du kan anta at arrayen `brett` til å begynne med inneholder den samme spillsituasjonen som noden `n` inneholder. Du kan også benytte funksjonsprosedyren

(Fortsettes side 7.)

```
REF(situasjon) finnode(c); CHARACTER ARRAY c;
```

som returnerer med `NONE` hvis det ikke finnes en node med spillsituasjon `c` i grafen og som returnerer med en peker til denne noden hvis den finnes. Du skal ikke programmere `finnode`.

3-b

Har du et forslag til hvordan nodene i grafen kan representeres for å lette jobben for `finnode`? Gjør en kort vurdering av et par mulige metoder.

3-c

I den grafen vi laget i oppgave 3-a har vi lagret informasjon om alle mulige spillforløp. Vi sier at den noden i grafen som tilsvarer *vinnersituasjonen* er den som inneholder det brettet hvor alle de hvite og de svarte brikkene har byttet plass. Vi sier at en node er *unødvendig* dersom den ikke er med i noen vei som leder fram til vinnersituasjonen. Altså er en node i grafen unødvendig hvis den ikke tilsvarer vinnersituasjonen og enten

1. Den ikke har noen utkanter, eller
2. Den kun har utkanter til unødvendige noder.

Lag en prosedyre

```
BOOLEAN PROCEDURE strip(n);
ref(node) n;
```

Som fjerner alle unødvendige noder fra grafen med startsituasjon `n`. Hvert prosedyrekall skal returnere med `FALSE` dersom parameteren `n` viste seg å peke på en unødvendig node, og `TRUE` hvis den peker på en nødvendig node.

3-d Kan Puffes

Fra oppgave 3-c fikk vi en graf som representerer alle vinnende sekvenser av trekk i spillet. Vi skal i denne oppgaven ta utgangspunkt i denne strippede grafen. Til nå har vi sett på

(Fortsettes side 8.)

det spillet som starter med alle de hvite brikkene til venstre, og alle de svarte brikkene til høyre. Dersom vi snur alle kantene i grafen vår vil vi få en graf med alle vinnende sekvenser for det motsatte spillet, hvor vi starter med de hvite brikkene på høyre side, og de svarte på venstre. Forklar hvorfor den snudde grafen tilsvarer det motsatte spillet.

Programmer prosedyren

```
PROCEDURE snu(n); REF(situasjon) n;
```

som snur alle kantene i grafen slik at de går den andre veien.

Av plasshensyn skal du ikke opprette noen nye noder, men du kan utvide klassen `situasjon` med en variabel `merket`.

(Slutt på oppgavesettet)