Godkendelsesopgave 1 i "Styresystemer og multiprogrammering"

1 Generelt

Denne ugeopgave stilles fredag den 4. februar 2011 og skal afleveres senest mandag den 14. februar 2011 klokken 23:59:59. Den kan løses af grupper på op til 2 personer (3 kan tillades, men frarådes). Besvarelsen af opgaven vil resultere i enten 0, 1/2 eller 1 point. Pointene uddeles efter følgende retningslinjer:

- 0 point: besvarelsen har flere store mangler.
- 1/2 point: besvarelsen opfylder i store træk kravene, men har flere mindre mangler.
- 1 point: besvarelsen opfylder kravene til opgaven med kun få eller ingen mangler.

Det er en betingelse for at gå til eksamen på kurset at man har opnået mindst 4 point i alt.

Besvarelsen skal indleveres elektronisk via kursushjemmesiden (Absalon). Besvarelsen bør ske ved aflevering af en enkelt fil. Brug 'zip' eller 'tar.gz' til at samle flere filer. Opgavenummer og navne på gruppemedlemmer skal fremgå tydelig af første side i besvarelsen. Når I indleverer bør efternavne på alle gruppemedlemmer indgå i filnavnet - desuden skal opgavenummer fremgå af navnet:

```
efternavn1-efternavn2-efternavn3-G1.<endelse>
```

Jeres aflevering skal være i et format der kan læses på DIKUs systemer uden problemer (således bør formatet f.eks. ikke være MS Word dokumenter). Se i øvrigt "Krav til G-opgaver" siden på kursushjemmesiden under "G-opgaver" menupunktet.

I skal kun aflevere én fuld besvarelse per gruppe. Personen, som afleverer skal markere sine gruppemedlemmer via Absalon.

2 Denne uges tema: Pegere

Formålet med denne uges opgaver er at se nærmere på dynamisk lagerallokering i C, samt brugen af pegere (pointers) til både data og funktioner. Løsningerne må ikke anvende en statisk øvre grænse for antallet af elementer med mindre det er angivet i opgaven.

Aflevereingen skal indeholde en rapport på 1-3 sider der dokumenterer hver delopgave. I skal også huske at kommentere jeres kildetekst så den er let at forstå.

Introduktion til opgaverne

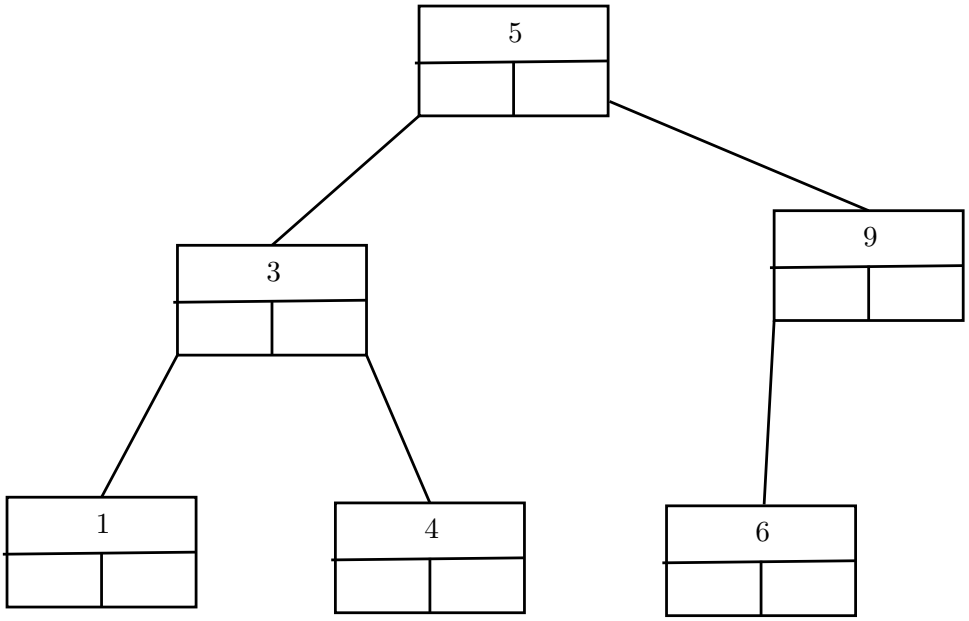
I denne uges G-afleverering skal I implementere et antal funktioner som behandler og konstruerer binære søgetræer.

Et binært træ består af knuder hvor hver knude indeholder en peger til "venstre"- og "højre"-undertræ samt et dataelement. I denne opgave (d.v.s. hele G1) fortolkes en `NULL`-pointer som et tomt træ. Følgende er definitionen af datastrukturen, som skal bruges i de næste to opgaver (afsnit 2.1 og 2.2): (defineres også i filen `bintree.h`)

```
typedef struct tnode_t {
  int data;
  struct tnode_t *lchild;
  struct tnode_t *rchild;
} tnode_t;
```

2.1 Binære søgetræer

Et binært søgetræ er et binært træ hvor knuderne i træet er arrangeret i rækkefølge. For hver knude gælder det at alle knuder i venstre undertræ er mindre-end eller lig-med knuden ($\leq$), og alle knuder i højre undertræ er større end ($>$) knuden.



Figur 1: Eksempel på binært søgetræ.

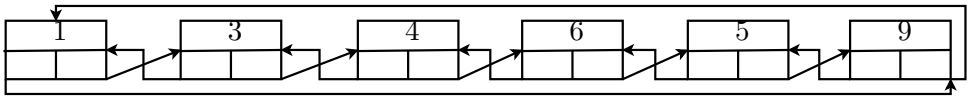
Filen `bintree.h` indeholder erklæringer af følgende funktioner:

```
void insert(tnode_t **, int);
void print_inorder(tnode_t *);
int size(tnode_t*);
int *to_array(tnode_t*);
```

Opgave 1 Implementér hver af de ovennævnte funktioner i en fil `bintree.c` sådan at implementationen opfører sig som beskrevet nedenfor.

`insert(tree, data)` Et kald til `insert` skal indsætte `data` elementet på rette plads i træet sådan at træet stadig er et binært søgetræ. Bemærk typen for `insert`: træet gives som en peger til en peger på træet.

`print_inorder(tree)` Denne funktion skal udskrive elementerne i søgetræet sådan at venstre undertræ udskrives før elementerne i højre undertræ og data feltet udskrives efter venstre



Figur 2: Eksempel på dobbelt-hægtet cirkulær liste.

undertræ, men før højre (dvs. en "inorder" rækkefølge). Eksempel: lad `t` være en peger til træet i figur 1. Resultatet af `print_inorder(t)` skal være at tallene udskrives i følgende rækkefølge: $1, 3, 4, 5, 6, 9$.

`size(tree)` Denne funktion skal returnere antallet af elementer i træet. Eksempel: lad `t` være en peger til træet i figur 1. `size(t)` skal da returnere tallet 6.

`to_array` Denne funktion skal returnere en peger på en tabel (array) som indeholder alle data-elementerne fra træet.

2.2 Konvertering til dobbelt-hægtet liste

I denne opgave skal I konvertere et binært søgetræ til en cirkulær dobbelt-hægtet liste. Definitionen af strukturen for dobbelt-hægtede lister ligner til forveksling den for binære træer (defineret i `dlist.h`):

```
typedef struct dlist_t {
  int data;
  struct dlist_t *prev;
  struct dlist_t *next;
} dlist_t;
```

I en dobbelt-hægtet liste har elementer en peger til det næste og det forrige element i listen. I en *cirkulær* dobbelt-hægtet liste må ingen af elementernes næste eller forrige pegere pege på `NULL`.

Den tomme liste repræsenteres ved `NULL` værdien. I en dobbelt-hægtet cirkulær liste med netop ét element peger `next` og `prev` på elementet selv. Hvis `dl` er er peger på en liste med netop et element så vil det gælde at `dl->next == dl` og `dl->prev == dl`

Opgave 2 Implementér funktionen `tree2list` som er erklæret i `dlist.h`. Funktionen tager et binært søgetræ og konstruerer en cirkulær dobbelt-hægtet liste og returnerer er peger til det mindste element i listen. Elementerne i listen skal være sorteret i en stigende rækkefølge. Hvis `t` er træet fra figur 1 skal resultatet være en peger til elementet som indeholder 1 i figur 2.

2.3 Tilknytning af ekstern data

I denne opgave skal begrebet "funktionspegere" benyttes (se evt. K&R afsnit 5.11).

I de forrige opgaver indeholdt knuderne i træet ikke andet data end et tal. Det er ikke specielt nyttigt i praksis. I en mere realistisk sammenhæng ville man gerne kunne indsætte "vilkårlige" typer data i et søgetræ. Vi ændrer derfor på strukturen således at den bliver:

```
typedef struct tnode_t2 {
  void *data;
  struct tnode_t2 *lchild;
  struct tnode_t2 *rchild;
} tnode_t2;
```

For at kunne indsætte elementer i et søgetræ af typen `tnode_t2` mangler vi nu en måde at sammenligne to elementer for at afgøre om et nyt element skal indsættes til venstre eller til højre. Til at hjælpe med det formål ændrer vi derfor `insert` funktionen. Den nye indsættelsesfunktion tager en ekstra parameter som er en peger til en funktion der kan sammen ligne to `void *` værdier. Funktionen `insert2` er erklæret i `bintree.h`:

```
void insert2(tnode_t2 **, void*, int (*comp)(void*, void*));
```

Den sidste parameter har typen `int (*comp)(void*, void*)`. Det betyder at når der forekommer et kald `insert2(t,da,fun)` er `fun` en peger på en funktion som tager to `void *` argumenter og som returnerer en `int`. I `insert2` kan man således skrive `(*comp)(data1, data2)` for at sammenligne to data-elementer. Hvis `data1` er mindre-end (men ikke lig-med) `data2` skal `comp` returnere -1 og hvis `data1` er lig-med `data2` skal `comp` returnere 0 og ellers 1.

Opgave 3 Implementér funktionen `insert2`.