Föreläsningsöversikt

- Genomgång inlämningsuppgift 1B.
- Genomgång inlämningsuppgift 1A.
- Binära sökträd i SML.
- Kommentarer om inlämningsuppgift 2.

Uppgift 1B

Implementera ett program som tar tre parametrar på kommandoraden. De första två parametrarna är flyttal, den tredje är en sträng `"add"` eller `"mul"`. Beroende på utseendet på den tredje strängen skall de två talen antingen adderas eller multipliceras. Resultatet skall skrivas ut till `stdout`.

För att lösa uppgiften skall ni använda funktionen `strtod()` från `<stdlib.h>` och funktionen `strcmp()` från `<string.h>`. Använd `man`-kommandot för att hitta dokumentationen för dessa funktioner, eller titta i er lärobok.

Programhuvudet. Vi har `#include`–deklarationer för de bibliotek vi behöver.

```
/*
 * Inlämningsuppgift 1B.
 */
#include <stdio.h>  /* always needed */
#include <stdlib.h> /* for strtod() */
#include <string.h> /* for strcmp() */
```


Början av `main()`. Vi använder versionen med `argc` och `argv` eftersom vi kommer att titta på kommandoradsparametrarna. Alternativet `main(void)` ger ej tillgång till kommandoraden.

```
int main(int argc, char *argv[])
{
    double x, y;

    if (argc != 4) {
        printf("fel antal parametrar\n");
        return 1;
    }
```

Kroppen av `main()`. Vi konverterar de två första strängarna till flyttal, och kollar sedan vilket kommando den tredje strängen innehåller.

```
    x = strtod(argv[1], NULL);
    y = strtod(argv[2], NULL);
    if (strcmp(argv[3], "add") == 0)
        printf("%f\n", x+y);
    else if (strcmp(argv[3], "mul") == 0)
        printf("%f\n", x*y);
    else {
        printf("fel kommando\n");
        return 1;
    }
    return 0;
}
```


Uppgift 1A

Implementera en procedur som multiplicerar en $k \times m$ matris A och en $m \times n$ matris B, och skriver resultatet till en $k \times n$ matris C vars element beräknas enligt

$$c_{ij} = \sum_{p=1}^{m} (a_{ip} \times b_{pj})$$

för $i = 1, 2, \ldots, k$ och $j = 1, 2, \ldots, n$.

Matriselementen skall vara flyttal i dubbel precision. Dimensionerna k, m och n är konstanter, men de ska enkelt kunna ändras i källkoden.

Programhuvudet. Använd `#define` för att introducera konstanterna `k`, `m` och `n`.

```
/*
 * Inlämningsuppgift 1A.
 */
#include <stdio.h>

#define k 2
#define m 3
#define n 2
```


Procedur för matrismultiplikation.

```
void matmult(double A[k][m], double B[m][n], double C[k][n])
{
    int i, j, p;

    for(i = 0; i < k; ++i)
        for(j = 0; j < n; ++j) {
            double sum = 0;
            for(p = 0; p < m; ++p)
                sum += A[i][p] * B[p][j];
            C[i][j] = sum;
        }
}
```

Procedur för utskrift av C–matrisen.

```
void printC(double C[k][n])
{
    int i, j;

    for(i = 0; i < k; ++i) {
        for(j = 0; j < n; ++j)
            printf("\t%.1f", C[i][j]);
        printf("\n");
    }
}
```


Procedur för initialisering av A– och B–matriserna.

```
void initAandB(double A[k][m], double B[m][n])
{
    A[0][0] = 0; A[0][1] = 1; A[0][2] = 2;
    A[1][0] = 3; A[1][1] = 4; A[1][2] = 5;
    B[0][0] = 6; B[0][1] = 7;
    B[1][0] = 8; B[1][1] = 9;
    B[2][0] = 10; B[2][1] = 11;
}
```

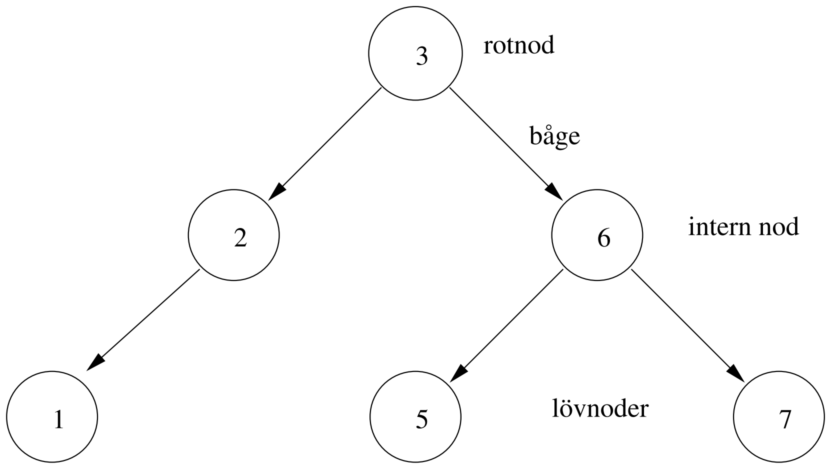

Huvudprogram. Deklarera matriserna och låt sedan de andra procedurerna göra allt jobb.

```
int main(void)
{
    double A[k][m], B[m][n], C[k][n];
    initAandB(A, B);
    matmult(A, B, C);
    printC(C);
    return 0;
}
```

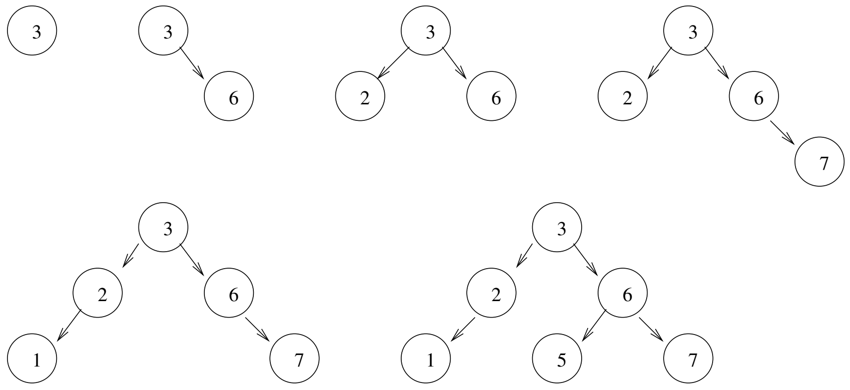

Binära sökträd

- Ett *träd* är en datastruktur bestående av *noder* och riktade *bågar*. En nod har bågar till 0 eller flera andra noder. Om nod N_1 har en båge till nod N_2 så kallas N_1 för *föräldranod* och N_2 för *barnnod* eller *delträd*. Varje nod har *exakt en* föräldranod, utom startnoden som kallas *roten*. En nod utan barn kallas *löv*.
- Ett *binärt* träd är ett träd där antalet delträd under en nod är högst 2.
- Ett binärt *sökträd* är ett binärt träd där varje nod innehåller en *nyckel*. Om en nod har nyckel K så skall alla noders nycklar i det vänstra delträdet vara mindre än K, och alla noders nycklar i det högra delträdet vara större än K.

Exempel på binärt sökträd



Exempel: insättning 3, 6, 2, 7, 1 och 5



Binärt sökträd i SML

```
(* bintree.sml -- an SML implementation of
   simple binary search trees *)

(* A tree is either empty, or a node with a
   key and references to two other trees. *)

datatype tree
  = EMPTY
  | NODE of {key: int, left: tree, right: tree}
```



```
(* Print the keys in a tree, using an
   inorder traversal: left, key, right. *)

fun printTree(tree) =
  case tree
    of EMPTY => ()
     | NODE{key,left,right} =>
        (printTree(left);
         print(Int.toString(key)); print "\n";
         printTree(right))
```

```
(* Insert a key in a tree and return the new tree. *)
fun insert(tree, newKey) =
  case tree
    of EMPTY =>
        NODE{key=newKey, left=EMPTY, right=EMPTY}
     | NODE{key,left,right} =>
        if newKey < key then
          NODE{key=key, right=right,
               left=insert(left, newKey)}
        else if newKey > key then
          NODE{key=key, left=left,
               right=insert(right, newKey)}
        else (* key = newKey *)
          tree
```



```
(* Sort and print a list of integers using a binary tree. *)

fun sortAndPrint2(intList, tree) =
  case intList
    of [] => printTree(tree)
     | first::rest =>
         sortAndPrint2(rest, insert(tree, first))

fun sortAndPrint(intList) =
  sortAndPrint2(intList, EMPTY)
```


Kommentarer om Uppgift 2

- Realisera typen `tree` i C. Jämför ex.vis. med hur listor representeras i C och justera lite.
- Konvertera `printTree`. Använd rekursion.
- Konvertera `insert`. Antingen kan ni följa SML–koden (rekursera, skapa nod, skapa nya noder på vägen upp) eller så kan ni *iterera* ner i trädet och stoppa in den nya node mha en tilldelning. Det senare är den normala programmeringsstilen i C.
- För inläsning av heltal, använd `scanf()` och `%d`–formatet. Läs manualen (`man scanf`) eller se lämplig lärobok för exempel.