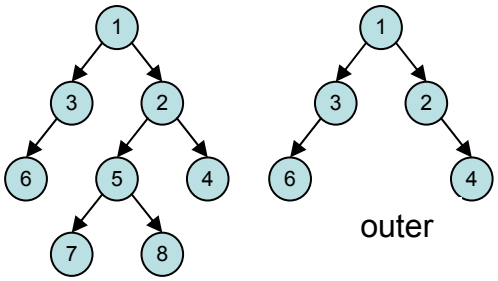
Aufgabe 1 **12 Punkte**

In der Vorlesung wurden binäre Bäume eingeführt, die als Knoteninformation integer-Werte aufnehmen können:

```
type tree=^node;
     node=record
            l:tree;
            x:integer;
            r:tree
     end;
```




```
function gennode(al:tree;ax:integer;ar:tree):tree;
var result:tree;
begin
  new(result);result^.x:=ax;result^.l:=al;result^.r:=ar;
  gennode:=result;
end;
```

Wir bezeichnen als äußere Knoten eines Baumes die Wurzel und alle ihre Nachfolger, die ausschließlich über l-Kanten oder alternativ ausschließlich über r-Kanten erreichbar sind. Alle anderen Knoten heißen innere Knoten.

Schreiben Sie eine Funktion `function outer(t : tree): tree;` die einen Baum erzeugt, der nur die äußeren Knoten von t enthält.

```
function lsucc(t : tree):tree;
begin
 if t=nil then lsucc := nil else
 lsucc := gennode(lsucc(t^.l),t^.ax.nil);
end;

function rsucc(t : tree):tree;
begin
 if t=nil then rsucc := nil else
 rsucc := gennode(nil,t^.x, rsucc(t^.r));
end;


function outer(t : tree):tree;
begin
 if t=nil then outer := nil else
 outer := gennode(lsucc(t^.l), t^.x, rsucc(t^.r)) ;
end;
```

Aufgabe 2 **12 Punkte**

Aus der Vorlesung kennen Sie die Unit `LispUnit.pas`, die Listen implementiert. Darin sind z.B. definiert:

```
function cons(ahead,atail:list):list;
function head(a:list):list;
function tail(a:list):list;
function isatom(a:list):booleaz;

function copy(l:list):list;
begin
 if l=nil then copy:=nil
 else copy:=cons(head(l),copy(tail(l)));
end;

function append(l1,l2:list):list;
begin
 if l1=nil then append:=copy(l2)
 else append:=cons(head(l1),append(tail(l1),l2));
end;
```

Schreiben Sie eine rekursive Funktion **`function odd(l:list):list;`** die eine Liste der Elemente aus der Liste l erzeugt, die an ungerader Position in l stehen.

```
function odd(l:list):list;
begin
 If l=nil then odd:=nil else
 odd := cons(head(l),odd(tail(tail(l))));
end;
```

Aufgabe 3 **12 Punkte**

Sortieren Sie die Zahlenfolge

3,14,2,1,20,4,7,5,19,6,8,12,11,9,13,15,17,16,10,18

nach der Methode des natürlichen 3-Band-Mischens. Zeichnen Sie die Bandinhalte nach jedem Durchlauf auf!

B1 = 3,14,1,20,5,19,11,16
B2 = 2,4,7,6,8,12,9,13,15,17,,10,18

M = 2,3,4,7,14,1,6,8,12,20,5,9,13,15,17,19,,10,11,16,18

B1 = 2,3,4,7,14, 5,9,13,15,17,19
B2 = 1,6,8,12,20,10,11,16,18

M = 1,2,3,4,6,7,8,12,14,20,5,9,10,11,13,15,16,17,18,19

B1 = 1,2,3,4,6,7,8,12,14,20
B2 = 5,9,10,11,13,15,16,17,19

M = 1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20

Aufgabe 4 **12 Punkte**

a) Definieren Sie ein Prolog-Prädikat **`letztes(E,L),`** das wahr ist, falls E das letzte Element in der Liste L ist.

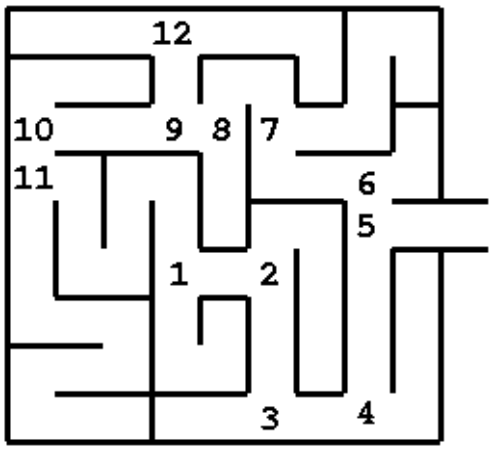
```
letztes(E,[E]).
letztes(E,[_|T]):-letztes(E,T).
```

b) Definieren Sie ein Prolog-Prädikat **`max(M,L),`** das wahr ist, falls M das größte Element in der Liste L ist. Tipp: 1. Klausel: Das Maximum einer Liste aus einem Element. 2. Klausel: Das head-Element der Liste ist das Maximum, falls es größer oder gleich dem Maximum des Tails der Liste ist. 3. Klausel: Das head-Element ist nicht das Maximum, falls es kleiner als das Maximum des Tails der Liste ist.

```
max(A,[A]).
max(A,[A|B]):- max(C,B), A>=C.
max(C,[A|B]):- max(C,B), A<C.
```

Aufgabe 5 **12 Pkte**

Ein Labyrinth ist ein Geflecht von Kreuzungen, die jeweils aus drei Abzweigungen zu einer weiteren Kreuzung, einer Sackgasse oder dem Ausgang bestehen (siehe Bild):




```
type kreuzung =
 array[1..3] of integer;
type labyrinth =
 array[1..12] of kreuzung;
```

Dabei sei der Integer-Wert in Kreuzung entweder der Index der nächsten Kreuzung (1 – 12), -1 (Sackgasse) oder 0 (Ausgang).

Wir suchen stets den Weg aus dem Labyrinth von ausgangspunkt an der Kreuzung 1 aus.

Erweitern sie die Prozeduren **`try()`** und **`zurueck()`** so, dass alle direkten Wege (d.h ohne Schleifen) aus dem Labyrinth `l` gefunden werden. Folgende Variabeln und Prozeduren stehen zur Verfügung:

```
var l: labyrinth; // das Labyrinth
    besucht: array[1..12] of boolean; // schon besucht
    loes: array[1..12] of integer;  // aktuelle Loesung
    laenge: integer; // Laenge der aktuellen Loesung

procedure initLabyrinth; // initialisiert alle Felder
begin
 l := ((-1,2,11),(-1,1,3),(-1,2,4),(-1,3,5),(0,4,6),(-1,5,7),
     (-1,6,8),(-1,7,9),(8,10,12),(-1,9,11),(-1,1,10),(-1,-1,9));
end;

procedure ausgabe; // gibt die aktuelle Loesung aus

procedure besuche(k: integer); // Besuchen der k-ten Kreuzung
begin
    laenge := laenge + 1;
    loes[laenge] := k;
    besucht[k] := true;
end;
```

Ab hier ihre Aufgabe:

```
procedure zurueck(k: integer); // Besuch der k-ten Kreuzung
begin                          // rueckgaengig machen
    besucht[k] := false;       // wie man leicht sieht, ist
   //loes[laenge] := 0;        // das einfach die Umkehrung
    laenge := laenge ñ 1;      // der Prozedur besuche


end;

procedure try(k: integer); // suche alle direkten Wege
var i: integer;            // ab der k-ten Kreuzung
begin
    if (k = 0) then begin
       ausgabe; // der Ausgang ist erreicht, also ausgeb



    end
    // Kreuzung noch nicht passiert und keine Sackgasse
    else if (( not besucht[k]) AND (k > 0)) then begin
        besuche( k );
        for i := 1 to 3 do begin
            try(l[k][i]); // Versuche in alle drei
                          // Richtungen unternehmen



        end;
        zurueck( k );
    end;
end;
```

Aufgabe 6 **12 Punkte**

Stellen sie den folgenden Graphen durch eine symmetrisch verkettete Adjazenzliste dar!