Aufgabe 1 (Sichtbarkeit von Variablen) *30 Punkte*

Gegeben sei folgendes Programmfragment:

```
1:  var s, t, u: integer;
2:  procedure A(s : integer, k : integer);
3:     var u, v : integer;
4:
5:     procedure B(s : integer, m : integer);
6:        var n : integer;
7:
8:        procedure C( o : integer);
9:           var p : integer;
10:       begin
11:          ...
12:       end C;
13:
14:    begin
15:          ...
16:    end B;
17:
18:       procedure D(o : integer, s : integer);
19:          var u : integer;
20:       begin
21:          ...
22:       end D;
23:
24: begin
25:    ...
26: end A;
27:
28: procedure E(v : integer)
29:    var w : integer;
30: begin
31:    ....
32: end E;
33:
34: ...
```

(a) Geben Sie die Sichtbarkeits- und die Gültigkeitsbereiche für alle vorkommenden Variablen an. *10 Punkte*

(b) Betrachten Sie die Aufruffolge[1]: *20 Punkte*
main→E→ E→A→B→B→D→B→E
Geben Sie Schnappschüsse der Display-Verwaltung (ähnlich wie im Kurstext) an.

1. Sie können davon ausgehen, dass es sich um eine gültige Aufruffolge handelt, d.h. entsprechende Aufrufe innerhalb der Prozeduren existieren.

20 Punkte

Aufgabe 2 (3-Adress-Programme)

Gegeben sei der folgende Programmtext

```
program main

  var
    res : integer;
    swi : bool;

  const
    integer a := 9;
    integer b := 7;



  procedure calc(flag: bool, x: integer, y: integer) : integer;

    var res : integer
    begin
      res := 0;

      if flag = true
        res := x * y;
      else
        if x <= y
          res := compute(x,y) + 1;
        else
          res := compute(y,x) + 2;

      return res;
    end calc


  procedure compute (u: integer, v: integer) : integer

    var val : integer;
    begin
      val := 0;

      if u >= 0
        begin
          u := u - 1;
          val := compute(u,v) + v ;
        end
      else
        val := val + 42;

      return val;
    end compute

begin
  swi := false;
  res := calc(swi, a, b) + compute(b, a);
...
end main
```

Übersetzen Sie Hauptprogramm und Prozeduren in 3-Adress-Code unter Verwendung der im Kurstext beschriebenen Befehlsklassen. Geben Sie zudem die Inhalte der Symboltabellen (Variablen bzw. Parameter und Prozeduren) an. Das Alignment sei für alle Typen 4. Die Startadressen für die Funktionen können, ebenso wie die Labels, frei gewählt werden. **Hinweis**: Die booleschen Werte sollen hier auf die Integer-Werte 0 = *false* und 1 = *true* abgebildet werden. Boolesche Operationen haben dann als Parameter ebenfalls Integer-Werte. Die *static_size* ergibt sich unter Berücksichtigung des Headers (16 Bytes) für den Prozedurrahmen (ohne Parameter und lokale Variablen). Integer benötigen 4 Bytes. Für die Darstellung der booleschen Werte veranschlagen wir nur 1 Byte. Gegebenenfalls müssen zusätzliche Variablen zur Zwischenspeicherung eingeführt werden.

Aufgabe 3 (myPS - Semantische Prüfungen 2) *50 Punkte*

In der letzten Kurseinheit wurden erste semantische Prüfungen in den Compiler für die Sprache *myPS* eingebaut. Nun sollen weitere Prüfungen stattfinden, die verhindern, dass ungültige Ergebnisse produziert werden. Im einzelnen sind dies:

- Bei Zuweisungen mit sofortiger Bindung muss allen Variablen, die auf der rechten Seite der Zuweisung verwendet werden, bereits ein Wert zugewiesen worden sein. Eine Variable gilt als definiert, wenn ihr ein Wert unabhängig von der Art der Bindung zugewiesen wurde.
- Rekursive Abhängigkeiten in Zuweisungen mit später Bindung führen zu einer Endlosrekursion, wenn der erzeugte Code durch einen Postscriptinterpreter verarbeitet wird. Eine entsprechende Prüfung soll nun bereits durch den *myPS*-Compiler durchgeführt werden.
- Auf eine Variable, die in einer `for`-Schleife verwendet wird, darf nur lesend zugegriffen werden.

Damit haben wir bereits einen vollständigen Compiler für die Sprache *myPS* implementiert. Um die Nutzung dieser Sprache zu vereinfachen, sollen in den nächsten Aufgaben noch einige Erweiterungen zur Sprache hinzugefügt werden. Die ersten Erweiterungen sind:

- Die beiden `write`-Befehle können bislang jeweils nur einen einzigen String verarbeiten. Um diesen Befehl etwas bequemer verwenden zu können, sollen nun auch Folgen von Strings akzeptiert werden, die jeweils durch ein ’+’ Zeichen miteinander verbunden werden. Somit läßt sich der Code

```
write("Ergebnis:"); write(num2string(erg));
```

nun vereinfacht als

```
write("Ergebnis:" + num2string(erg));
```

schreiben.

- Auch die Deklaration von Variablen erlaubt bislang nur eine einzige Variable. Nun soll es auch möglich sein, mehrere Variablen auf einmal mit dem gleichen Typ zu deklarieren. Die Variablen innerhalb der Liste sollen durch Kommata abgetrennt werden.
- Rotationen finden bisher stets um den Nullpunkt statt. Es soll ein weiterer `rotate`-Befehl bereitgestellt werden, der neben dem Rotationswinkel auch die Angabe des Rotationspunktes ermöglicht. Die Syntax des neuen Befehls lautet:

  ```
  rotate(point; angle, term).
  ```

- Neben der Ausgabe als „normale“ Postscript-Datei sollen nun auch sogenannte Encapsulated Postscript-Dateien erzeugt werden können. Innerhalb der *myPS*-Datei soll zu diesem Zweck eine *Bounding Box* angegeben werden können, die die Größe der Zeichenfläche beschreibt. Der dazugehörige Befehl muß zwischen den Deklarationen und dem Symbol `start` stehen und hat die folgende Syntax:

  ```
  bbox ux uy ox oy
  ```

  Dabei beschreibt der Punkt (`ux`, `uy`) die linke untere und der Punkt (`ox`, `oy`) die rechte obere Ecke des entsprechenden Rechtecks. Alle Werte sind vom Typ `Int`. Tritt diese Anweisung nicht in einer *myPS* Datei auf, so soll die übliche Postscript-Datei erzeugt werden.