C11 Solutions

1.

```
Count := 1;
FOR I in 1 .. 10 LOOP
        If I MOD 2 = 0 THEN
                FOR J in 1 .. 10 LOOP
                        Count:= Count + 2;
                END LOOP;
        ELSE
                FOR J in 1 .. 5 LOOP
                        Count := Count – 1;
                END LOOP;
        END IF;
END LOOP;
```

Count = 76.

Count increments by 20 when I is even and decrements by 5 when I is odd.

2. Write an Ada95 program to implement the Euler's 2nd order integration method? Turn in a hard copy of your algorithm and code listing and an electronic copy of your code.

C 11 part b ALGORTIHM

Eluer's 2^{nd} order integration – use trapeziodal rule.
Area of a trapezoid under curve = .5*(y1+y2)*delta_x

Algortihm:
Ask user for inputs:

- Coefficients of each polynomial term plus constant
- Upper and Lower Bounds of integration
- Step Size

Calculate number of steps = (upper_bound-lower_bound)/step_size and convert it to an integer

Loop from 0 to the number of steps using a for loop, performing euler's second order approximation

- Integral = Integral + .5*(y1+y2)*step_size
- Y1 = Y2
- Y2 = Y2 +Step_Size

Print out results

C11 b code
Solutions

```
with Ada.Text_Io;
use Ada.Text_Io;
with Ada.Integer_Text_Io;
use Ada.Integer_Text_Io;
with Ada.Float_Text_Io;
use Ada.Float_Text_Io;

procedure Second_Order_Euler is
   --procedure to perform euler's second oreder integration method
   --a definite integral is input by the user and is the calculation is
   --performed and returned
   --only takes in polynomials up to 6th order
   --Unified Computers and Programming, Problem C11 b, Fall 2003
   --Author: Howard Kleinwaks, based on an algorithm by Phil Springmann
   --Last Modified: October 5, 2003

   --declare variables
   Order : Integer; --stores order of polynomial
   Upper_Bound : Float;
   Lower_Bound : Float; --numbers to store the bounds of the integral
   First_Order_Term : Float;
   Second_Order_Term : Float;
   Third_Order_Term : Float;
   Fourth_Order_Term : Float;
   Fifth_Order_Term : Float;
   Sixth_Order_Term : Float;
   Constant_Term : Float;
   Integral : Float;
   Step_Size : Float; --input value by user to determine step size to use
   Number_Of_Steps : Float;
   Integer_Number_Of_Steps : Integer; --need integer number of steps to
use in for loop
   Low_Step : Float;
   High_Step : Float; --variables to represent current x-values (x_i and
 x_i+1)

begin -- Second_Order_Euler
   --get input variables
   --take the order of the polynomial and the coefficients
   Put("Please enter the order of the polynomial (between one and six):"
);
   Get(Item => Order);
   New_Line;
   --check order to make sure it is within the proper bounds
   while Order< 1 and Order > 6 loop
      Put("Please enter the order of the polynomial (between one and six
):");
      Get(Item => Order);
      New_Line;
   end loop;

   --get coefficients of the polynomial
   Put("Please enter the constant term:");
   Get(Item => Constant_Term);
   New_Line;
   Put("Please enter the coefficent of the lowest order term:");
   Get(Item => First_Order_Term);
   New_Line;
   Put("Please enter the coefficent of the next lowest order term:");
   Get(Item => Second_Order_Term);
```

```
New_Line;
Put("Please enter the coefficent of the next lowest order term:");
Get(Item => Third_Order_Term);
New_Line;
Put("Please enter the coefficent of the next lowest order term:");
Get(Item => Fourth_Order_Term);
New_Line;
Put("Please enter the coefficent of the next lowest order term:");
Get(Item => Fifth_Order_Term);
New_Line;
Put("Please enter the coefficent of the next lowest order term:");
Get(Item => Sixth_Order_Term);
New_Line;

--get bounds of integration
Put("Please enter the lower bound:");
Get(Item => Lower_Bound);
New_Line;
   Put("Please enter the upper bound:");
   Get(Item => Upper_Bound);
   New_Line;

   --get step size desired from user
   Put("Please enter the step size:");
   Get(Item => Step_Size);
   New_Line;

   --calculate number of steps
   Number_Of_Steps := (Upper_Bound - Lower_Bound)/Step_Size;
   --convert to integer
   Integer_Number_Of_Steps := Integer(Number_Of_Steps);

   --now loop from 0 to the number of steps, performing euler's second o
rder approximation
   --the approximation follows the trapezoidal rule
   --area of a trapezoid = .5*(b1 + b2)*h, where b1 and b2 are the funct
ion values at either end of the step
   --and h is the step size

   --need to initialize the value of integral (the result) and Low_Step
and High_Step
   Integral := 0.0;
   Low_Step := Lower_Bound;
   High_Step := Lower_Bound + Step_Size;

   for I in 1..Integer_Number_Of_Steps loop
      --calculate integral according to following method:
      --Integral := Integral + .5*(f(x)+f(x+1))*Step_Size
      Integral := Integral + 0.5*((Sixth_Order_Term*High_step**6 + Fifth
_Order_Term*Low_Step**5 + Fourth_Order_Term*Low_Step**4
            + Third_Order_Term*Low_Step**3 + Second_Order_Term*Low_Step*
*2 + First_Order_Term*Low_Step+Constant_Term)
         + (Sixth_Order_Term*High_Step**6 + Fifth_Order_Term*High_Step**
5 + Fourth_Order_Term*High_Step**4
            + Third_Order_Term*High_Step**3 + Second_Order_Term*High_Ste
p**2 + First_Order_Term*High_Step+Constant_Term))*Step_Size;
      Low_Step := Low_Step+Step_Size;
      High_Step := High_Step+Step_Size;
   end loop;
   Put("The integration is: ");
```

```
   Put(Integral, Exp => 0);
end Second_Order_Euler;
```

3.

Algorithm:

1. Initialize the counter to 1
2. Initialize Sum to 0
3. While (counter <= 10) loop
    i. Get a number from the user
    ii. Add Number to Sum
    iii. Increment the Counter
4. Compute the average by dividing sum by 10
5. Display computed average to the user

Code Listing

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Compiling: c:/docume~2/joeb/desktop/16070/codeso~1/average_with_while.adb (source file time stamp: 2003-10-02 02:41:10)

```
 1. ---------------------------------------------------
 2. -- Program to find the average of 10 numbers using
 3. -- a While Loop
 4. -- Programmer : Joe B
 5. -- Date Last Modified : October 01, 2003
 6. ---------------------------------------------------
 7.
 8.
 9. with Ada.Text_Io;
10. with Ada.Float_Text_Io;
11.
12. procedure Average_With_While is
13.    Counter : Integer :=1; -- initialize counter to 0
14.    Sum : Float :=0.0; -- initializise sum to 0
15.    Num : Float;-- variable used to get input from the user
16. begin
17.    while (Counter <= 10) loop
18.       -- get input from the user
19.       Ada.Text_Io.Put("Please Enter A Number : ");
20.       Ada.Float_Text_Io.Get(Num);
21.       Ada.Text_Io.Skip_Line;
22.       -- compute sum
23.       Sum := Sum + Num;
24.       -- increment the counter
25.       Counter := Counter +1;
26.
27.    end loop;
28.
29.    Ada.Text_Io.Put("The Average of Numbers is :");
30.    Ada.Float_Text_Io.Put(Sum/10.0);
31.
32. end Average_With_While;
```

32 lines: No errors

C12

1.

Algorithm

a. Get the number from the user
b. If the number is >= 1 then
    i. For I in 1 .. number loop
        - Factorial := Factorial * I;
c. Else
    i. Display Cannot Compute Factorial
d. Display Computed Factorial to the User

Code Listing

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Compiling: c:/docume~2/ joeb /desktop/16070/codeso~1/factorial_with_iteration.adb (source file time stamp: 2003-10-02 03:57:26)

```
 1. ------------------------------------------------------
 2. -- Program to find the factorial of a number using
 3. -- iteration.
 4. -- Programmer : Joe B
 5. -- Date Last Modified : October 01, 2003
 6. ------------------------------------------------------
 7.
 8.
 9. with Ada.Text_Io;
10. with Ada.Integer_Text_Io;
11.
12. procedure Factorial_With_Iteration is
13.    Factorial : Integer := 1; -- initializise factorial to 1
14.    Num       : Integer;      -- variable used to get input from the user
15. begin
16.    -- get the number from the user
17.    Ada.Text_Io.Put("Please Enter A Number : ");
18.    Ada.Integer_Text_Io.Get(Num);
19.    Ada.Text_Io.Skip_Line;
20.
21.    if Num >=1 then
22.
23.       -- compute factorial
24.       for I in 1 .. Num loop
25.          Factorial:= Factorial * I;
26.       end loop;
27.
28.       -- display the computed factorial to the user
29.       Ada.Text_Io.Put("The Factorial of ");
30.       Ada.Text_Io.Put(Integer'Image(Num));
31.       Ada.Text_Io.Put("is : ");
32.       Ada.Integer_Text_Io.Put(Factorial);
33.
34.    else
```

```
35.       Ada.Text_Io.Put("Cannot Compute Factorial");
36.    end if;
37.
38. end Factorial_With_Iteration;
```

38 lines: No errors

2.

The program does not work when you have numbers greater than 12 because the value of the factorial is larger than Integer'Last

3.

Algorithm

From the series, you can see that any number in the series is a sum of the previous two numbers, i.e. $number_n = num_{n-1} + number_{n-2}$

Main Program

1. Get the number of Fibonacci numbers from the user.
2. If the number is <= 0, then
    a. Display cannot generate numbers
3. else
    a. for I in 1 .. Number
        i. Call the recursive function Fibo
        ii. Display Value returned by Fibo

Fibo Function

1. If num < 2
    a. Return 1
2. Else
    a. Return (Fib(n-1)+ Fib(n-2));

Code Listing

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Compiling: c:/docume~2/ joeb /desktop/16070/codeso~1/fibo_with_recursion.adb (source file time stamp: 2003-10-02 14:42:40)

```
1. ------------------------------------------------------
2. -- Program to find N fibonacci numbers using
3. -- recursion.
```

```
-- Programmer : Joe B
-- Date Last Modified : October 01, 2003
-------------------------------------------------------


with Ada.Text_Io;
with Ada.Integer_Text_Io;

procedure Fibo_With_Recursion is
   function Fibo (
         Num : Integer )
     return Integer is
   begin
      if (Num < 2) then
         return 1;
      else
         return(Fibo(Num-1) + Fibo(Num-2));
      end if;
   end Fibo;

   Number : Integer;

begin
   -- get the number from the user
   Ada.Text_Io.Put("Please Enter The Number Of Fibonacci Numbers : ");
   Ada.Integer_Text_Io.Get(Number);
   Ada.Text_Io.Skip_Line;

   if Number <=0 then
      Ada.Text_Io.Put("Cannot Generate Required Numbers");

   else
      for I in 1 .. Number loop
         Ada.Integer_Text_Io.Put(Fibo(I));
      end loop;
   end if;

end Fibo_With_Recursion;
```

41 lines: No errors

Note: While we are generating the numbers recursively, we are not displaying them as they are generated. Can you guess why?

The recursive function Fibo uses two recursive calls within the program. If you trace the program, you will see that the values are computed twice by the recursive function. If you try to display the values inside the recursive function, the values will be repeated.

C13

1. Solve the following recurrence equation using the iteration method. Show all the steps in your derivation.

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Substitute the value of T(n) from the recurrence equation:

aT(n/b) + cn

$\Rightarrow$ a(aT((n/b)/b) + c(n/b)) + cn

$\Rightarrow$ $a^2T(n/b^2) + cn(a/b) + cn$

$\Rightarrow$ $a^2T(n/b^2) + cn((a/b) + 1)$

$\Rightarrow$ $a^2(aT((n/b^2)/b) + cn/b^2) + cn((a/b) + 1)$

$\Rightarrow$ $a^3T(n/b^3) + cn(a^2/b^2) + cn((a/b) + 1)$

$\Rightarrow$ $a^3T(n/b^3) + cn((a^2/b^2) + (a/b) + 1)$

…

$\Rightarrow$ $a^kT(n/b^k) + cn((a^{k-1}/b^{k-1}) + (a^{k-2}/b^{k-2}) + \ldots + (a^2/b^2) + (a/b) + 1)$

When $k = \log_b n$,

$\Rightarrow n = b^k$

T(n) $= a^kT(1) + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)$

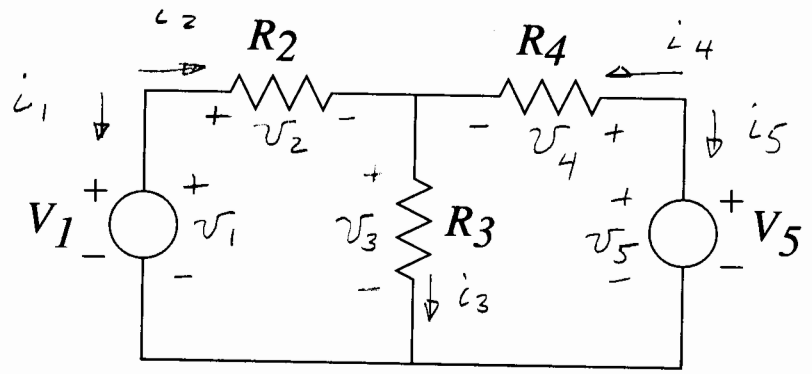
$= a^kc + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)$

$= ca^k + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)$

$= cna^k/b^k + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1)$

$= cn(a^k/b^k + ... + a^2/b^2 + a/b + 1)$

1. The labeling is arbitrary, except for the voltage sources. Also, each current arrow must be drawn <u>into</u> the "+" terminal, or out of the "−" terminal.



2. There are two loops, one on the left, one on the right. KVL gives

$$-v_1 + v_2 + v_3 = 0 \qquad (1)$$

$$-v_3 - v_4 + v_5 = 0 \qquad (2)$$

3. Write KCL for upper left, upper middle, and upper right nodes:

$$i_1 + i_2 = 0 \qquad (3)$$

$$-i_2 + i_3 - i_4 = 0 \qquad (4)$$

$$i_4 + i_5 = 0 \qquad (5)$$

4. For the voltages sources,

$$v_1 = V_1 = 4\ \text{V} \qquad (6)$$

$$v_5 = V_5 = 6\ \text{V} \qquad (7)$$

For the resistors,

$$v_2 = i_2 R_2 = 4 i_2 \quad (8)$$
$$v_3 = i_3 R_3 = 6 i_3 \quad (9)$$
$$v_4 = i_4 R_5 = 12 i_4 \quad (10)$$

5. There are 10 unknowns (5 i's, 5 v's), and 10 equations, so we should be able to solve.

To solve, substitute (6) – (10) into (1) and (2):

$$4 i_2 + 6 i_3 = 4 \quad (11)$$
$$-6 i_3 - 12 i_4 = -6 \quad (12)$$

(Note that I've dropped the units for now.)

Now, begin reducing equations:

$$(3) \Rightarrow i_1 = -i_2 \quad (13)$$
$$(4) \Rightarrow i_3 = i_2 + i_4 \quad (14)$$
$$(5) \Rightarrow i_5 = -i_4 \quad (15)$$

Plug these into (11), (12) to obtain

$$4 i_2 + 6(i_2 + i_4) = 4 \quad (16)$$
$$-6(i_2 + i_4) - 12 i_4 = -6 \quad (17)$$

Simplifying,

$$10\,i_2 + 6\,i_4 = 4 \quad (18)$$

$$6\,i_2 + 18\,i_4 = 6 \quad (19)$$

Solve for i_2, i_4 using Cramer's rule or Gaussian elimination. The result is

$$\boxed{\begin{aligned} i_2 &= 0.25\ A \\ i_4 &= 0.25\ A \end{aligned}} \quad (20), (21)$$

Plug these back into (13) – (15):

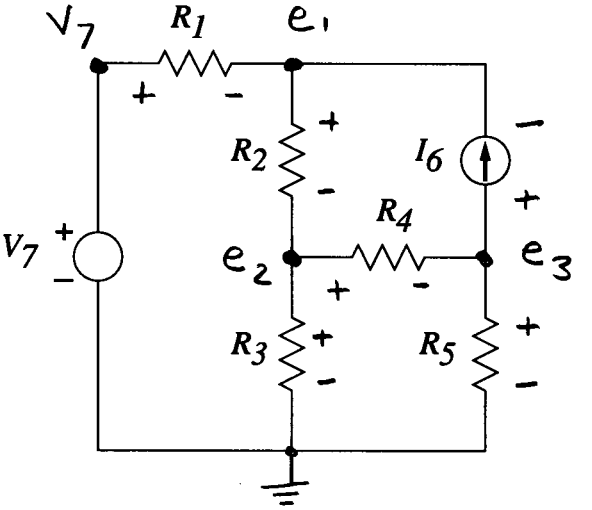
$$\boxed{\begin{aligned} i_1 &= -0.25\ A \\ i_3 &= 0.5\ A \\ i_5 &= -0.25\ A \end{aligned}} \quad (22), (23), (24)$$

Use constitutive laws [(6)–(10)] to find voltages:

$$\boxed{\begin{aligned} v_1 &= 4\ V \\ v_2 &= 1\ V \\ v_3 &= 3\ V \\ v_4 &= 3\ V \\ v_5 &= 6\ V \end{aligned}}$$

To check our answer, it is easily verified that these values satisfy KVL, KCL, and the constitutive laws.

1. The first steps are to label the node potentials (to allow solution by the node method) and to label each element with +/- signs (so we can talk about branch currents, voltages).



Next, we write KCL at each node with unknown potential. This can be done "by inspection," as in class:

$$e_1: \quad \left(\frac{1}{R_1}+\frac{1}{R_2}\right)e_1 - \frac{1}{R_2}e_2 = \frac{1}{R_1}V_7 + I_6$$

$$e_2: \quad -\frac{1}{R_2}e_1 + \left(\frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4}\right)e_2 - \frac{1}{R_4}e_3 = 0$$

$$e_3: \quad -\frac{1}{R_4}e_2 + \left(\frac{1}{R_4}+\frac{1}{R_5}\right)e_3 = -I_6$$

Plugging in values, we have that:

$$\frac{5}{6}e_1 - \frac{1}{2}e_2 = 6$$

$$-\frac{1}{2}e_1 + \frac{7}{6}e_2 - \frac{1}{3}e_3 = 0$$

$$-\frac{1}{3}e_2 + \frac{4}{3}e_3 = -5$$

(I've dropped units here.) We can solve by Cramer's rule, Gaussian elimination, calculator, etc. The result is

$$e_1 = 9\ V$$
$$e_2 = 3\ V$$
$$e_3 = -3\ V$$

The branch voltages are just the difference in node potentials across each element:

$$v_1 = -6\ V$$
$$v_2 = 6\ V$$
$$v_3 = 3\ V$$
$$v_4 = 6\ V$$
$$v_5 = -3\ V$$
$$v_6 = -12\ V$$
$$v_7 = 3\ V$$

The branch currents are found by applying the constitutive laws:

$$\boxed{\begin{aligned} i_1 &= v_1/R_1 = -2\ A \\ i_2 &= v_2/R_2 = 3\ A \\ i_3 &= v_3/R_3 = 1\ A \\ i_4 &= v_4/R_4 = 2\ A \\ i_5 &= v_5/R_5 = -3\ A \\ i_6 &= I_6 = 5\ A \end{aligned}}$$

Note that the constitutive law for the voltage source,

$$v_7 = V_7, \text{ for all } i_7$$

gives no information about i_7. To find i_7, apply KCL at the V_7 node:

$$i_7 + i_1 = 0$$

$$\Rightarrow \boxed{i_7 = +2A}$$

2. Find the net power dissipated by the circuit:

$$P = \sum_n i_n v_n$$

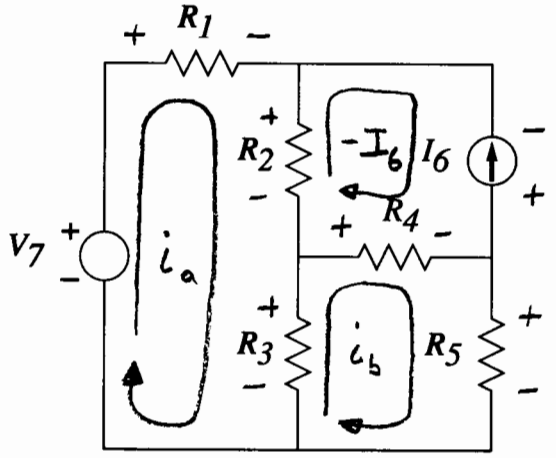
$$= (-2)(-6) + (3)(6) + (1)(3) + (2)(6) + (-3)(-3) + (5)(-12) + (2)(3)$$

$$\Rightarrow P = 12 + 18 + 3 + 12 + 9 - 60 + 6 = 0\,W$$

$$\boxed{P = 0\ W}$$

Note that the current source supplies power (−60 W), and the voltage source absorbs power (+6 W).

The first steps are to label the loop currents (to allow solution by the loop method) and to label each element with +/- signs (so we can talk about branch currents, voltages).



Next, we write KVL around each loop with unknown current. This can be done "by inspection":

$$i_a: \quad (R_1+R_2+R_3)\,i_a - R_3\,i_b = -R_2 I_6 + V_7$$

$$i_b: \quad -R_3\,i_a + (R_3+R_4+R_5)\,i_b = -R_4 I_6$$

Plugging in values, we have that

$$8\,i_a - 3\,i_b = -7$$
$$-3\,i_a + 7\,i_b = -15$$

This 2x2 set of equations can be solved by Cramer's rule, etc., to obtain

$$i_a = -2\,A$$
$$i_b = -3\,A$$

The branch currents are just the algebraic sum of loop currents:

$$i_1 = +i_a = -2A$$
$$i_2 = i_a + I_c = 3A$$
$$i_3 = i_a - i_b = 1\,A$$
$$i_4 = i_b + I_c = 2\,A$$
$$i_5 = i_b = -3\,A$$
$$i_6 = I_c = 5A$$
$$i_7 = -i_a = 2A$$

The voltages are found by applying the constitutive laws:

$$v_1 = i_1 R_1 = -6\,V$$
$$v_2 = i_2 R_2 = 6\,V$$
$$v_3 = i_3 R_3 = 3\,V$$
$$v_4 = i_4 R_4 = 6\,V$$
$$v_5 = i_5 R_5 = -3\,V$$
$$v_7 = V_7 = 3\,V$$

Note that the constitutive law for the current source,

$$i_6 = I_6, \text{ for all } v_6$$

gives no information about v_6. To find v_6, apply KVL around I_6 loop:

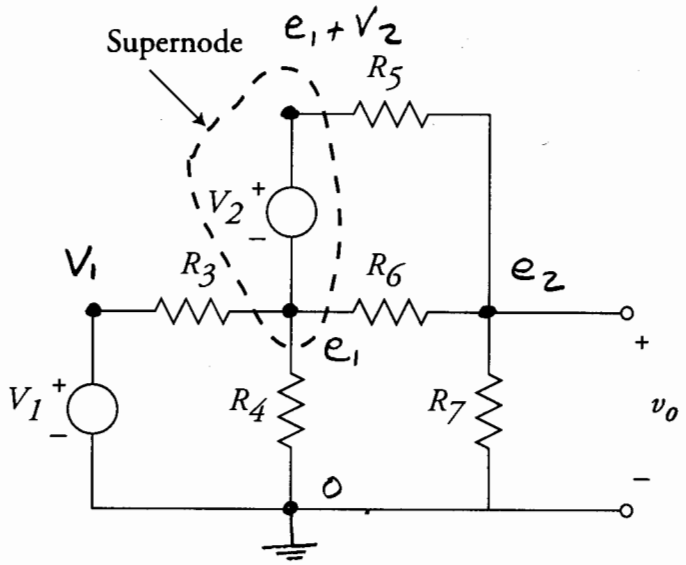
$$-v_4 - v_2 - v_6 = 0$$

$$\Rightarrow \quad v_6 = -v_2 - v_4$$
$$= -12\,V$$

$$\boxed{v_6 = -12\,V}$$

Of course, these values agree with those of §3.

The nodes are labeled as below:



Note that 4 of the nodes (ground, V_1, e_1, e_2) are labeled normally. The 5th node is labeled as $e_1 + V_2$, not e_3, since there is a known potential difference across V_2.

To start, apply KCL at e_2:

$$e_2: \quad \frac{e_2 - 0}{R_7} + \frac{e_2 - e_1}{R_6} + \frac{e_2 - (e_1 + V_2)}{R_5} = 0$$

$$\Rightarrow -\left(\frac{1}{R_5} + \frac{1}{R_6}\right) e_1 + \left(\frac{1}{R_5} + \frac{1}{R_6} + \frac{1}{R_7}\right) e_2 = \frac{V_2}{R_5}$$

This result is typical of the node method in simpler problems.

Next, apply KCL at nodes e_1 and $e_1 + V_2$:

e_1: $\frac{e_1 - V_1}{R_3} + \frac{e_1 - 0}{R_4} + \frac{e_1 - e_2}{R_6} - i_2 = 0$

$e_1 + V_2$: $\frac{e_1 + V_2 - e_2}{R_5} + i_2 = 0$

Note that the constitutive law for the V_2 source gives no information about i_2. However, we can eliminate i_2 by <u>adding</u> the two equations above:

supernode: $\left(\frac{1}{R_3} + \frac{1}{R_4} + \frac{1}{R_5} + \frac{1}{R_6}\right) e_1 - \left(\frac{1}{R_5} + \frac{1}{R_6}\right) e_2 = \frac{1}{R_3} V_1 - \frac{1}{R_5} V_2$

Plugging in values, we have

$$2.5\, e_1 - e_2 = 2.5$$
$$-e_1 + 2e_2 = 2.5$$

Solving for e_1, e_2, we have

$$e_1 = 1.875\ V \qquad e_2 = 2.1875\ V$$

Since $v_0 = e_2 - 0$,

$$\boxed{v_0 = 2.1875\ V}$$