In [23]:

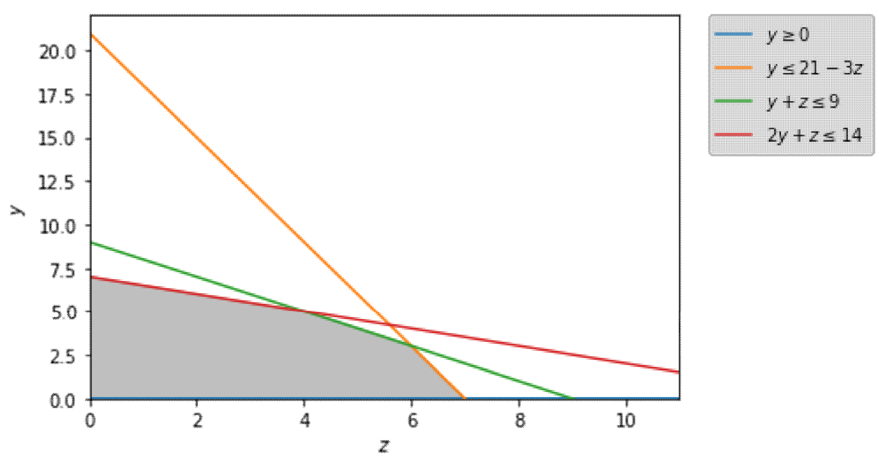
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Construct lines
# z >= 0
x = np.linspace(0, 20, 2000)
y = np.linspace(0, 20, 2000)
# y >= 0
y1 = (z*0)
# y <= 21 - 3z
y2 = (21-3*z)
# y <= 9 - z
y3 = 9-z
# 2y <= 14 - z
y4 = (14-z)/2

# Make plot
plt.plot(x, y1, label=r'$y\geq0$')
plt.plot(x, y2, label=r'$y \leq 21 - 3z$')
plt.plot(x, y3, label=r'$y +z\leq 9$')
plt.plot(x, y4, label=r'$2y+z\leq 14$')
plt.ylim((0, 22))
plt.xlim((0, 11))
plt.xlabel(r'$z$')
plt.ylabel(r'$y$')

# Fill feasible region
y5 = np.minimum(np.minimum(y2, y3), y4)
#y6 = np.maximum(y1, y3)
plt.fill_between(x, y5,  color='grey', alpha=0.5)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
#plt.scatter(xi,yi, color='black' )
```

Out[23]: <matplotlib.legend.Legend at 0x21e97f89108>



Q2:5 intersection points [u,v,w,y,z]:

```
[21,9,14,0,0]
[0,0,2,3,6]
[14,2,0,7,0]
[0,2,7,0,7]
[5,0,0,5,4]
```

Q3, Q4 see below:

In [68]:
```python
import numpy as np
import sympy
from sympy import *
M=Matrix([[1,0,0,1, 3, 21], [0,1,0, 1, 1, 9], [0,0, 1, 2, 1, 14]])
print("Matrix : {} ".format(M))

# Use sympy.rref() method
M_rref = M.rref()

print("The Row echelon form of matrix M(u,v,w,y,z) and the pivot columns : {}".format(M_rref))
```

```
Matrix : Matrix([[1, 0, 0, 1, 3, 21], [0, 1, 0, 1, 1, 9], [0, 0, 1, 2, 1, 14]])
The Row echelon form of matrix M and the pivot columns : (Matrix([
[1, 0, 0, 1, 3, 21],
[0, 1, 0, 1, 1,  9],
[0, 0, 1, 2, 1, 14]]), (0, 1, 2))
```

In [81]:
```python
#M.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
print("M(yuvwz)=")
M.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
```

```
M(yuvwz)=
```

Out[81]:
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 3 & 21 \\ 1 & 0 & 1 & 0 & 1 & 9 \\ 2 & 0 & 0 & 1 & 1 & 14 \end{bmatrix}$$

In [80]:
```python
M2= M.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
M2_rref = M2.rref()
print("The Row echelon form of matrix M(y,u,v,w,z) and the pivot columns : {}".format(M2_rref))
```

```
M(yuvwz)=
The Row echelon form of matrix M(y,u,v,w,z) and the pivot columns : (Matrix([
[1, 0, 0,  1/2, 1/2,  7],
[0, 1, 0, -1/2, 5/2, 14],
[0, 0, 1, -1/2, 1/2,  2]]), (0, 1, 2))
```

In [83]:
```python
#M.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
print("M(uyvwz)=")
M2.permute([[0, 1]], orientation='columns', direction='forward')
```

```
M(uyvwz)=
```

Out[83]:
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 3 & 21 \\ 0 & 1 & 1 & 0 & 1 & 9 \\ 0 & 2 & 0 & 1 & 1 & 14 \end{bmatrix}$$

In [85]:
```python
M3= M2.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
M3_rref = M3.rref()
print("The Row echelon form of matrix M(u,y,v,w,z) and the pivot columns : {}".format(M3_rref))
```

```
The Row echelon form of matrix M(u,y,v,w,z) and the pivot columns : (Matrix([
[1, 0, 0, -2, -1, -4],
[0, 1, 0,  1,  1,  9],
[0, 0, 1, -1,  2, 12]]), (0, 1, 2))
```

In [90]:

```
#M.permute([[0, 3],[1,3],[2,3]], orientation='columns', direction='forward')
print("M(uvywz)=")
M.permute([[2,3]], orientation='columns', direction='forward')
```

M(uvywz)=

Out[90]:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 3 & 21 \\ 0 & 1 & 1 & 0 & 1 & 9 \\ 0 & 0 & 2 & 1 & 1 & 14 \end{bmatrix}$$

In [91]:

```
M4= M.permute([[2,3]], orientation='columns', direction='forward')
M4_rref = M4.rref()
print("The Row echelon form of matrix M(u,v,y,w,z) and the pivot columns : {}".format(M4_rref))
```

```
The Row echelon form of matrix M(u,v,y,w,z) and the pivot columns : (Matrix([
[1, 0, 0, -1/2, 5/2, 14],
[0, 1, 0, -1/2, 1/2,  2],
[0, 0, 1,  1/2, 1/2,  7]]), (0, 1, 2))
```

Q5: Solutions all correspondes to vertices. see Q4.

Q6: x1, x2,x3, x4= 5,6,2,1

Q7: Q has to be the point where x5=0 and x2>0. Basically by trying and error.
We have to use x1,x2,x3,x4=5,0,10,7, x5=2.

In [92]:

```
M5=Matrix([[1,0,0,0,1, -3,-4,2, 5], [0,1,0,0, 3, 7, 2,-2,6], [0,0, 1, 0, -4, -2, -7, -2, 2], [0,0,
print("Matrix : {} ".format(M5))
```

```
Matrix : Matrix([[1, 0, 0, 0, 1, -3, -4, 2, 5], [0, 1, 0, 0, 3, 7, 2, -2, 6], [0, 0, 1, 0, -4, -2, -
7, -2, 2], [0, 0, 0, 1, -3, 4, 3, 3, 1]])
```

In [95]:

```
M6= M5.permute([[1,4]], orientation='columns', direction='forward')
# Use sympy.rref() method
M6_rref = M6.rref()
print("The Row echelon form of matrix   and the pivot columns : {}".format(M6_rref))
```

```
The Row echelon form of matrix   and the pivot columns : (Matrix([
[1, 0, 0, 0, -1/3, -16/3, -14/3,   8/3,  3],
[0, 1, 0, 0,  1/3,   7/3,   2/3,  -2/3,  2],
[0, 0, 1, 0,  4/3,  22/3, -13/3, -14/3, 10],
[0, 0, 0, 1,    1,    11,     5,     1,  7]]), (0, 1, 2, 3))
```

Q9. When x5 has a positive coefficient, x1 will decrease; and when x5 has a negative coefficient, then x1 increases.

In [ ]:

```
Q10:
p=2e+6l+4c;
g=1000-8e-10l-7.5c;
m=1-0.03125l;
s=10-0.125l-0.125c.
```

In [125]:

```
latrix([[1,0,0,0, -2,-6,-5, 0], [0,1,0,0, 8, 10, 7.5,1000], [0,0, 1, 0, 0, 0.03125, 0, 1], [0,0, 0, 1
:("The order of variable is p,g,m,s,e,l,c")
'ref = M10.rref()
:("Q10 answer in Matrix : {} ".format(M10.rref))
```

```
The order of variable is p,g,m,s,e,l,c
Q10 answer in Matrix : <bound method MatrixReductions.rref of Matrix([
[1, 0, 0, 0, -2,      -6,    -5,    0],
[0, 1, 0, 0,  8,      10,   7.5, 1000],
[0, 0, 1, 0,  0, 0.03125,     0,    1],
[0, 0, 0, 1,  0,   0.125, 0.125,   10]])>
```

In [126]:

```python
import heapq


'''
   Return a rectangular identity matrix with the specified diagonal entirities, possibly
   starting in the middle.
'''
def identity(numRows, numCols, val=1, rowStart=0):
   return [[(val if i == j else 0) for j in range(numCols)]
               for i in range(rowStart, numRows)]


'''
   standardForm: [float], [[float]], [float], [[float]], [float], [[float]], [float] -> [float], [
   Convert a linear program in general form to the standard form for the
   simplex algorithm.  The inputs are assumed to have the correct dimensions: cost
   is a length n list, greaterThans is an n-by-m matrix, gtThreshold is a vector
   of length m, with the same pattern holding for the remaining inputs. No
   dimension errors are caught, and we assume there are no unrestricted variables.
'''
def standardForm(cost, greaterThans=[], gtThreshold=[], lessThans=[], ltThreshold=[],
                equalities=[], eqThreshold=[], maximization=True):
   newVars = 0
   numRows = 0
   if gtThreshold != []:
      newVars += len(gtThreshold)
      numRows += len(gtThreshold)
   if ltThreshold != []:
      newVars += len(ltThreshold)
      numRows += len(ltThreshold)
   if eqThreshold != []:
      numRows += len(eqThreshold)

   if not maximization:
      cost = [-x for x in cost]

   if newVars == 0:
      return cost, equalities, eqThreshold

   newCost = list(cost) + [0] * newVars

   constraints = []
   threshold = []

   oldConstraints = [(greaterThans, gtThreshold, -1), (lessThans, ltThreshold, 1),
                     (equalities, eqThreshold, 0)]

   offset = 0
   for constraintList, oldThreshold, coefficient in oldConstraints:
      constraints += [c + r for c, r in zip(constraintList,
         identity(numRows, newVars, coefficient, offset))]

      threshold += oldThreshold
      offset += len(oldThreshold)

   return newCost, constraints, threshold


def dot(a,b):
   return sum(x*y for x,y in zip(a,b))

def column(A, j):
   return [row[j] for row in A]

def transpose(A):
   return [column(A, j) for j in range(len(A[0]))]

def isPivotCol(col):
   return (len([c for c in col if c == 0]) == len(col) - 1) and sum(col) == 1
```

```python
def variableValueForPivotColumn(tableau, column):
   pivotRow = [i for (i, x) in enumerate(column) if x == 1][0]
   return tableau[pivotRow][-1]

# assume the last m columns of A are the slack variables; the initial basis is
# the set of slack variables
def initialTableau(c, A, b):
   tableau = [row[:] + [x] for row, x in zip(A, b)]
   tableau.append([ci for ci in c] + [0])
   return tableau


def primalSolution(tableau):
   # the pivot columns denote which variables are used
   columns = transpose(tableau)
   indices = [j for j, col in enumerate(columns[:-1]) if isPivotCol(col)]
   return [(colIndex, variableValueForPivotColumn(tableau, columns[colIndex]))
            for colIndex in indices]


def objectiveValue(tableau):
   return -(tableau[-1][-1])


def canImprove(tableau):
   lastRow = tableau[-1]
   return any(x > 0 for x in lastRow[:-1])


# this can be slightly faster
def moreThanOneMin(L):
   if len(L) <= 1:
      return False

   x,y = heapq.nsmallest(2, L, key=lambda x: x[1])
   return x == y


def findPivotIndex(tableau):
   # pick minimum positive index of the last row
   column_choices = [(i,x) for (i,x) in enumerate(tableau[-1][:-1]) if x > 0]
   column = min(column_choices, key=lambda a: a[1])[0]

   # check if unbounded
   if all(row[column] <= 0 for row in tableau):
      raise Exception('Linear program is unbounded.')

   # check for degeneracy: more than one minimizer of the quotient
   quotients = [(i, r[-1] / r[column])
      for i,r in enumerate(tableau[:-1]) if r[column] > 0]

   if moreThanOneMin(quotients):
      raise Exception('Linear program is degenerate.')

   # pick row index minimizing the quotient
   row = min(quotients, key=lambda x: x[1])[0]

   return row, column


def pivotAbout(tableau, pivot):
   i,j = pivot

   pivotDenom = tableau[i][j]
   tableau[i] = [x / pivotDenom for x in tableau[i]]

   for k,row in enumerate(tableau):
      if k != i:
         pivotRowMultiple = [y * tableau[k][j] for y in tableau[i]]
```

```python
            tableau[k] = [x - y for x,y in zip(tableau[k], pivotRowMultiple)]


'''
   simplex: [float], [[float]], [float] -> [float], float
   Solve the given standard-form linear program:
      max <c,x>
      s.t. Ax = b
           x >= 0
   providing the optimal solution x* and the value of the objective function
'''
def simplex(c, A, b):
   tableau = initialTableau(c, A, b)
   print("Initial tableau:")
   for row in tableau:
      print(row)
   print()

   while canImprove(tableau):
      pivot = findPivotIndex(tableau)
      print("Next pivot index is=%d,%d \n" % pivot)
      pivotAbout(tableau, pivot)
      print("Tableau after pivot:")
      for row in tableau:
         print(row)
      print()

   return tableau, primalSolution(tableau), objectiveValue(tableau)


if __name__ == "__main__":
   c = [2, 6, 4]

#M10=Matrix([[1,0,0,0, -2,-6,-5, 0], [0,1,0,0, 8, 0, 0,1000],[0,0, 1, 0, 0, 0.03125, 0, 1], [0,0,


   A =[ [8, 10, 7.5], [0, 0.03125, 0], [0, 0.125, 0.125,]]
   b = [1000, 1, 10]

   # add slack variables by hand
   A[0] += [1,0,0]
   A[1] += [0,1,0]
   A[2] += [0,0,1]
   #A[3] += [0,0,0,-1]
   c += [0,0,0]

   t, s, v = simplex(c, A, b)
   print(s)
   print(v)
```

```
Initial tableau:
[8, 10, 7.5, 1, 0, 0, 1000]
[0, 0.03125, 0, 0, 1, 0, 1]
[0, 0.125, 0.125, 0, 0, 1, 10]
[2, 6, 4, 0, 0, 0, 0]

Next pivot index is=0,0

Tableau after pivot:
[1.0, 1.25, 0.9375, 0.125, 0.0, 0.0, 125.0]
[0.0, 0.03125, 0.0, 0.0, 1.0, 0.0, 1.0]
[0.0, 0.125, 0.125, 0.0, 0.0, 1.0, 10.0]
[0.0, 3.5, 2.125, -0.25, 0.0, 0.0, -250.0]

Next pivot index is=2,2

Tableau after pivot:
```

```
[1.0, 0.3125, 0.0, 0.125, 0.0, -7.5, 50.0]
[0.0, 0.03125, 0.0, 0.0, 1.0, 0.0, 1.0]
[0.0, 1.0, 1.0, 0.0, 0.0, 8.0, 80.0]
[0.0, 1.375, 0.0, -0.25, 0.0, -17.0, -420.0]

Next pivot index is=1,1

Tableau after pivot:
[1.0, 0.0, 0.0, 0.125, -10.0, -7.5, 40.0]
[0.0, 1.0, 0.0, 0.0, 32.0, 0.0, 32.0]
[0.0, 0.0, 1.0, 0.0, -32.0, 8.0, 48.0]
[0.0, 0.0, 0.0, -0.25, -44.0, -17.0, -464.0]

[(0, 40.0), (1, 32.0), (2, 48.0)]
464.0
```

Q11. Answer as above, not that the order is e=40, l=32, c=48

In [ ]:

```

```