In [ ]:
```
#Problem 1-1
using JuMP, Cbc

Data
t = time required to fill gas in each aircraft for j in J
g = gallon required for each airplane i in I
aircrafts = 1:15
sites = 1:7


Decision Variables
xj , Binary variable to decided whether to fill gas

Objective
maximize gallon distribution
sum(gas*xj)



Constraints
cover of each aircraft in each area
        e.g. aircraft 2 is covered by site 1 and 2, so site 1 and site 2 will go together in a constraint
        for aircraft 2 because either location can be used to fill up aircraft 2.
        I don't really know how to say this in words, but if you look at the code it will make hopefully make sense

time spent filling aircraft <= 175
coverage for each plane i in sites
```

In [9]:
```
#Problem 1-2
using JuMP, Cbc

m=Model()

sites = [:1,:2,:3,:4,:5,:6,:7]
time = Dict(zip(sites, [50, 38, 45, 53, 42, 59, 48]))
gas = Dict(zip(sites, [100,130,250,310,220,350,190]))
# gas is calculated by adding the gas required for each plane in each site, example
# site 1 covers planes 1 2 and 4 and 1 2 and 4 require 20, 20 and 60. 20+20+60 = 100, and do for each site

#aircrafts = Dict(zip(sites, [[1,2,4],[2,3,5],[4,7,8,10],[5,6,8,9],[8,9,12],[7,10,11,12,15],[12,13,14,15]]))
#gas = Dict(zip(sites, [[20,20,60],[20,30,80],[60,90,90,110],[80,40,90,100],[90,100,30],[90,110,60,30,60], [30,70,30,60

@variable(m, x[sites], Bin)
```

```
@objective(m, Max, sum(gas[i]*x[i] for i in sites))

@constraint(m, x[1] >= 1)          #aircraft 1 coverage
@constraint(m, x[1] + x[2] >= 1)   #aircraft 2 coverage
@constraint(m, x[2] >= 1)          #aircraft 3 coverage
@constraint(m, x[1] + x[3] >= 1)   #aircraft 4 coverage
@constraint(m, x[2] + x[4] >= 1)   #aircraft 5 coverage
@constraint(m, x[4] >= 1)          #aircraft 6 coverage
@constraint(m, x[3] + x[6] >= 1)   #aircraft 7 coverage
@constraint(m, x[3] + x[4] + x[5] >= 1) #aircraft 8 coverage
@constraint(m, x[4] + x[5] >= 1)   #aircraft 9 coverage
@constraint(m, x[3] + x[6] >= 1)   #aircraft 10 coverage
@constraint(m, x[6] >= 1)          #aircraft 11 coverage
@constraint(m, x[6] + x[7] >= 1)   #aircraft 12 coverage
@constraint(m, x[7] >= 1)          #aircraft 13 coverage
@constraint(m, x[7] >= 1)          #aircraft 14 coverage
@constraint(m, x[6] + x[7] >= 1)   #aircraft 15 coverage

@constraint(m, limit[i in sites], time[i]*x[i] <= 175) #time constraint to fill gas

set_optimizer(m, Cbc.Optimizer)
optimize!(m)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.8
Build Date: Jan  1 1970

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 1550 - 0.00 seconds
Cgl0004I processed model has 0 rows, 0 columns (0 integer (0 of which binary)) and 0 elements
Cbc3007W No integer variables - nothing to do
Cuts at root node changed objective from -1550 to -1.79769e+308
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 se
conds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                1550.00000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.01
Time (Wallclock seconds):       0.01

Total time (CPU seconds):       0.01   (Wallclock seconds):       0.01
```

In [8]:

```
#Problem 1-2
#using JuMP, Cbc

#m=Model()

#aircrafts = [:1,:2,:3,:4,:5,:6,:7,:8, :9, :10, :11, :12, :13, :14, :15]
#gas = Dict(zip(aircrafts, [20, 20,30,60,80,40,90,90,100,110,60,30,70,30,60]))
#time = Dict(zip(aircrafts, [50, 88, 38, 95, 91, 53, 104, 150, 95, 104, 59, 107, 48, 48, 107]))


#@variable(m, x[aircrafts], Bin)


#@objective(m, Max, sum(gas[i]*x[i] for i in aircrafts))
```

```
#@constraint(m, x[1] >= 1)          #aircraft 1 coverage
#@constraint(m, x[1] + x[2] >= 1)   #aircraft 2 coverage
#@constraint(m, x[2] >= 1)          #aircraft 3 coverage
#@constraint(m, x[1] + x[3] >= 1)   #aircraft 4 coverage
#@constraint(m, x[2] + x[4] >= 1)   #aircraft 5 coverage
#@constraint(m, x[4] >= 1)          #aircraft 6 coverage
#@constraint(m, x[3] + x[6] >= 1)   #aircraft 7 coverage
#@constraint(m, x[3] + x[4] + x[5] >= 1) #aircraft 8 coverage
#@constraint(m, x[4] + x[5] >= 1)   #aircraft 9 coverage
#@constraint(m, x[3] + x[6] >= 1)   #aircraft 10 coverage
#@constraint(m, x[6] >= 1)          #aircraft 11 coverage
#@constraint(m, x[6] + x[7] >= 1)   #aircraft 12 coverage
#@constraint(m, x[7] >= 1)          #aircraft 13 coverage
#@constraint(m, x[7] >= 1)          #aircraft 14 coverage
#@constraint(m, x[6] + x[7] >= 1)   #aircraft 15 coverage

#@constraint(m, limit[i in aircrafts], time[i]*x[i] <= 175)

#set_optimizer(m, Cbc.Optimizer)
#optimize!(m)
```

In [ ]:

In [ ]:

Problem 2-1

Geoff, the Owner of Freedom Skate Shop On State wants to build as many skateboards as he can to sell in the skateshop. In order to build a skateboard, Geoff needs **one Board**, **two trucks**, a **pack of wheels** and a **pack of bearings**. W It takes **30 minutes** for Geoff to build a skateboard. Geoff Has **8** hours of labor in order to build these skateboards. Each skateboard sold will earn Geoff a profit of **80 Dollars**. If Geoff has leftover parts, he can sell them individually for extra profit. Each board can be sold for **40 Dollars**, each truck can be sold for **22 Dollars**, each wheel pack can be sold for **12 Dollars**, and each bearing pack can be sold for **10 Dollars**. Occasionally, Geoff will recieve **Professional Decks** and **Old School Decks** in his orders, and can sell professional ones for **60 Dollars**, and Old School Decks for **100 Dollars**. But he can't sell both at the same time, as there will be no more space on his board display if he does so.

Write an integer programming model that will maximize Geoff's revenue, using the table below to help you write it

Item Name	Amount	Sell Price
Board	20	40

Item Name	Amount	Sell Price
Trucks	37	22
Wheels	12	12
Bearings	14	10
Professional Board	10	60
Old School Board	5	100
Complete Skateboard	-	80

Problem 2-1 Answer

Decision Variables

xp, number of pro decks to sell xo, number of old school decks to sell yp, binary decision to decide to sell pro decks yb, binaru decision to decide to sell old school boards cb, complete boards

Objective

Max xp+xo+cb + sum(xi for i in I) I being skateboard items

Constraints

sum(items i in I) >= cb #make a complete skateboard

0.5cb <= 8 #time constraint to make a skateboard

5yo >= xo #binary to sell old school boards

10yp >= xp #binary to sell professional boards

cb,xp,xo >=0 #all non zero constants

Problem 2-2

Use the JuMP package in IJulia to solve this problem!

In [1]:

```
#Problem 2-2
using JuMP, Cbc

items = [:1, :2, :3, :4] #1 being board, 2 is trucks,  3 is wheels, 4 is bearings
materials = [20, 37, 12, 14]
sell = [40, 22, 12, 10]

m=Model()

@variable(m, xp >=0) #Pro boards
@variable(m, yp, Bin) #binary to decide to sell pro boards

@variable(m, xo >= 0) #Big funny boards
@variable(m, yo, Bin) #binary to decide to sell big funny boards

@variable(m, cb >=0) #Complete deck

@variable(m, x[items] >=0)



@objective(m, Max, 60xp + 100xo + 80cb + sum(sell[i] for i in items))



@constraint(m, items[i in items], sum(x[i]) >= cb)
@constraint(m, 0.5cb <= 8)

@constraint(m, 5yo <= xo)
@constraint(m, xo <= 5yo)
@constraint(m, 10yp <= xp)
@constraint(m, xp <= 10yp)


@constraint(m, cb >=0)
@constraint(m, xp >=0)
@constraint(m, xo >=0)

set_optimizer(m, Cbc.Optimizer)
optimize!(m)
```

```
Welcome to the CBC MILP Solver
Version: 2.10.8
Build Date: Jan  1 1970

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 2380 - 0.02 seconds
Cgl0004I processed model has 4 rows, 4 columns (4 integer (2 of which binary)) and 8 elements
Cutoff increment increased from 1e-05 to 19.9999
Cbc0038I Initial state - 0 integers unsatisfied sum - 0
Cbc0038I Solution found of -2380
Cbc0038I Before mini branch and bound, 2 integers at bound fixed and 2 continuous
Cbc0038I Mini branch and bound did not improve solution (0.06 seconds)
Cbc0038I After 0.06 seconds - Feasibility pump exiting with objective of -2380 - took 0.00 seconds
Cbc0012I Integer solution of -2380 found by feasibility pump after 0 iterations and 0 nodes (0.06 seconds)
Cbc0001I Search completed - best objective -2380, took 0 iterations and 0 nodes (0.06 seconds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from -2380 to -2380
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 se
conds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                2380.00000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             0.07
Time (Wallclock seconds):       0.07

Total time (CPU seconds):       0.09   (Wallclock seconds):       0.09
```

In [ ]: