1 Stacking Colored Bricks:

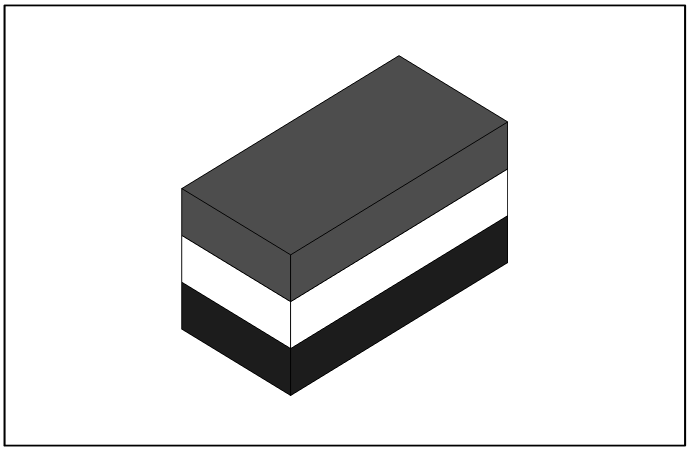
An Introduction to for .. do ... od and if ... then ...fi Commands.

1.1 A Chromatic Brick Placer

Suppose we want to stack ten bricks but we want to alternate the colors. First we need to restore the color option to our "BL" command. Lets modify it to a "CBP" (chromatic brick placer) which allows us to specify the color.

```
CBP:= proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [8, 0, 0]+shift; v3 := [8, 4, 0]+shift;
v4 := [0, 4, 0]+shift; v5 := [0, 0, 2]+shift; v6 := [8, 0, 2]+shift;
v7 := [8, 4, 2]+shift; v8 := [0, 4, 2]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=clr,style=patch, scaling=constrained )
end:

plots[display]([CBP([0,0,0],blue),CBP([0,0,2]
,white),CBP([0,0,4],red)]);
```



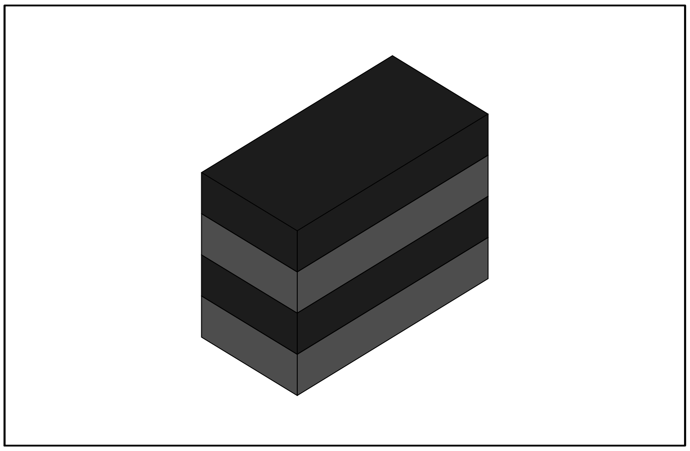
Suppose we want to stack alternating blue and red bricks. The obvious way to do it "by hand" would be to manufacture the bricks one at a time assemble them into a list and use plots[display] to view the result. Indeed lets do a few steps in the process to see what we want to automate. This time we want to assemble the list as we go rather than waiting until the end. This should keep us from having to give a permanent name to each one. We simply give each new brick a temporary name and append it to the list. Lets let "CBS" stand for colored brick stack.

```
CBS:=NULL;tmpbrick:=CBP([0,0,0],red):
CBS:=CBS,tmpbrick:
```

```
tmpbrick:=CBP([0,0,2],blue):
CBS:=CBS,tmpbrick:
tmpbrick:=CBP([0,0,4],red):
CBS:=CBS,tmpbrick:
tmpbrick:=CBP([0,0,6],blue):
CBS:=CBS,tmpbrick:
```

$$CBS :=$$

```
plots[display]([CBS]);
```



Like the other processes we have been working with recently we can see that this is terribly repetetive. All we are doing is repeating the two commands

tmpbrick =CBP([0,0,x], somecolor); CBS:=CBS,tmpbrick;

and we are doing it ten times.

What we need is the ability to tell the computer to do something a fixed number of times - in this case 10 times. The basic format for the command that does this is

for VARIABLE from STARTINGNUMBER to STOPPING NUMBER
do
"sequence of Maple Commands"
od;

You can always spot the end of a do statement with do spelled backwards. od

To have Maple print "n!" for n from 1 to 10 one could do the following

```
for n from 1 to 10 do n!  od;
```

To have Maple print "n" followed by "n!" for n from 1 to 10 one could do any of the following

$$1$$

$$2$$

$$6$$

$$24$$

120

720

5040

40320

362880

3628800

```
for n from 1 to 10 do [n,n!]  od;
```

[1, 1]

[2, 2]

[3, 6]

[4, 24]

[5, 120]

[6, 720]

[7, 5040]

[8, 40320]

[9, 362880]

[10, 3628800]

or

```
for n from 1 to 10 do print(n,n!)  od;
```

1, 1

2, 2

3, 6

4, 24

5, 120

6, 720

7, 5040

8, 40320

9, 362880

10, 3628800

or

```
for n from 1 to 10 do cat(`The factorial of
`,n,` is ` ,n!,`.`) od;
```

The factorial of 1 is 1.

The factorial of 2 is 2.

The factorial of 3 is 6.

The factorial of 4 is 24.

The factorial of 5 is 120.

The factorial of 6 is 720.

The factorial of 7 is 5040.

The factorial of 8 is 40320.

The factorial of 9 is 362880.

The factorial of 10 is 3628800.

To build an expression sequence called SS containing the squares of the integers from 62 to 73 we could do the following

```
SS:=NULL;
for w from 62 to 73 do SS:= SS,w^2 od;
```

$$SS :=$$

$$SS := 3844$$

$$SS := 3844,\ 3969$$

$$SS := 3844,\ 3969,\ 4096$$

$$SS := 3844,\ 3969,\ 4096,\ 4225$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184,\ 5329$$

If as is the case here we really don't care to see all of the intermediate steps we can suppress the intermediate display by terminating with a colon ":". Change the semicolons to colons. After, type an "SS;" to see the final sequence.

```
SS:=NULL:
for w from 62 to 73 do SS:= SS,w^2 od;
```

$$SS := 3844$$

$$SS := 3844,\ 3969$$

$$SS := 3844,\ 3969,\ 4096$$

$$SS := 3844,\ 3969,\ 4096,\ 4225$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900$$

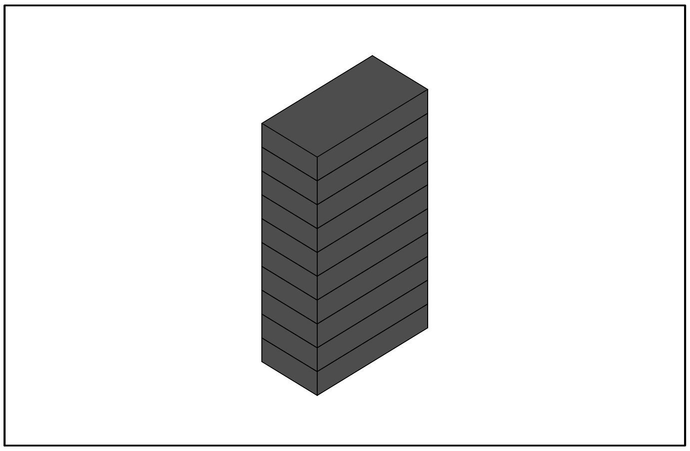
$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184$$

$$SS := 3844,\ 3969,\ 4096,\ 4225,\ 4356,\ 4489,\ 4624,\ 4761,\ 4900,\ 5041,\ 5184,\ 5329$$

So, for instance to build a sequence of ten red blocks like the one we did with "seq" we could simply

```
bricks:=NULL: for t from 0 to 9 do
bricks:=bricks,BL([0,0,2*t]) od:plots[display]([bricks]);
```



1.2 Exercises: DO .. OD

1.2.1 *Exercise: Modify the following to produce a command sequence which prints out the first 100 powers of 2.*

```
for s from 0 to 5 do print(x,2*x) od;
```

1.2.2 *Exercise: Write a command sequence which prints out* $[n,\ (1+\frac{1}{n})^n]$ *for the first 100 values of n.*

1.3 Conditional statements

Now we would like to make a stack of differently colored bricks. We will use our "CBP" command in a do .. od constructionl but we need to add commands within the "loop" to tell Maple how to color the tiles.

Suppose we want to start with a red brick on the bottom and then alternate with blue blocks from then. We need a way to issue the following instructions to the computer.

"if t is even then make the brick red otherwise make it blue".

We do this with the if ... then ... else .. fi command. Just as we end the do statement with do spelled backwards an if..then .. statement is ended with if spelled backwards.

Like the do..od command the if..fi command is best explained with an example. Lets do the alternating red-blue bricks. There are lots of possible ways to do this. We will be very pedantic at first.

We need one additional tool. First recall that an integer is even if it yields a remainder of zero when divided by 2; it is odd if it leaves a remainder of 1 under the same circumstances. If we enter the command "a mod b;" Maple returns the remainder upon division of "a" by "b".

```
6 mod 2;
```

$$0$$

```
17 mod 2;
```

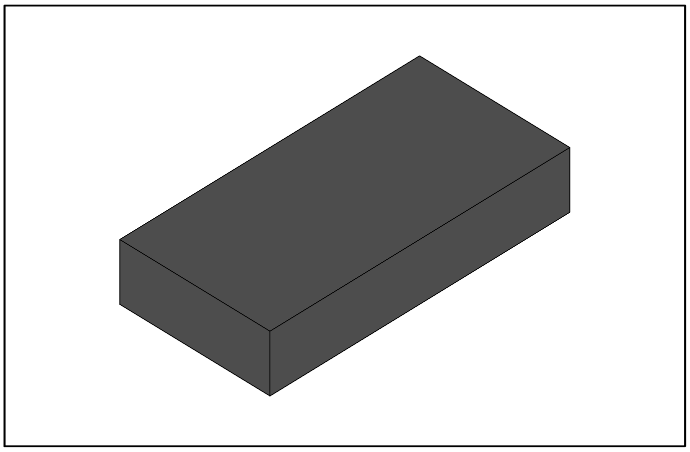
$$1$$

Now let "T" have some integral value and lets test whether its even or odd and make a red or blue brick accordingly

```
T:=17;
```

$$T := 17$$

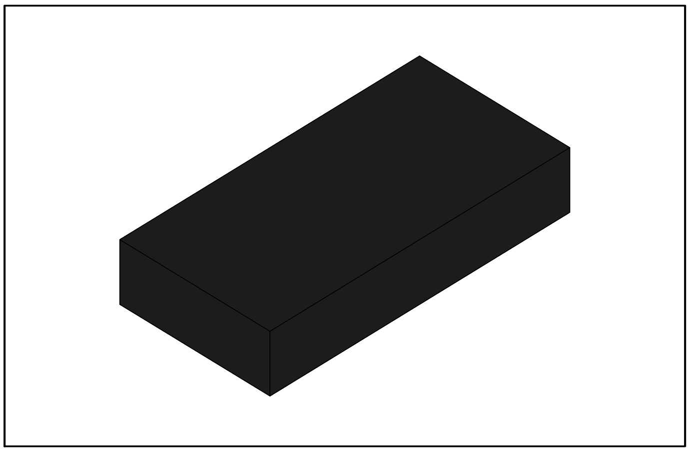
```
if T mod 2 = 1 then CBP([0,0,0],red) else
CBP([0,0,0],blue) fi;
```



```
T:=18;
```

$$T := 18$$

```
if T mod 2 = 1 then CBP([0,0,0],red) else
CBP([0,0,0],blue) fi ;
```

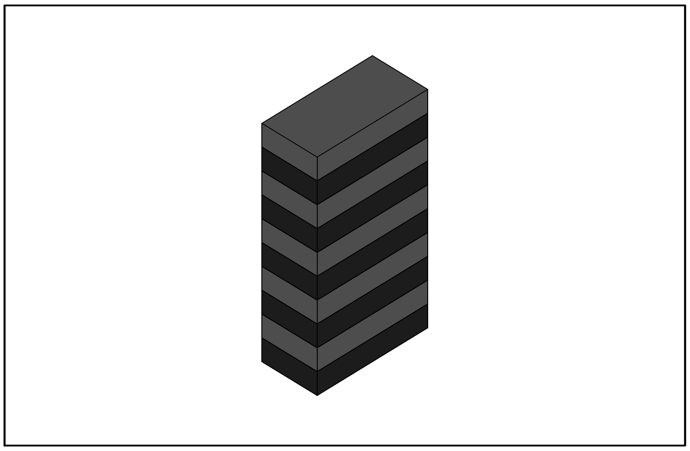


Note that it is often convenient to enclose expressions of the form **a mod b** in parenthesis. This is optional but it tends to make Maple code easier to follow.

Be especially careful to distinguish the **equality** = symbol in the above as opposed to the assignment symbol := . Although these look similar they are miles apart. The statement T := 17; assigns the value 17 to the name T. The equation T mod 2 = 1 is tested in the if .. fi statements above. Now all we have to do is modify the "body" of our "do..od" command which stacked ten red bricks

```
bricks:=NULL:
for t from 0 to 9 do
if (t mod 2) = 1 then
tmpbrick:=CBP([0,0,2*t],red) else
```

```
tmpbrick:=CBP([0,0,2*t],blue) fi:
bricks:=bricks,tmpbrick; od:
plots[display]([bricks]);
```

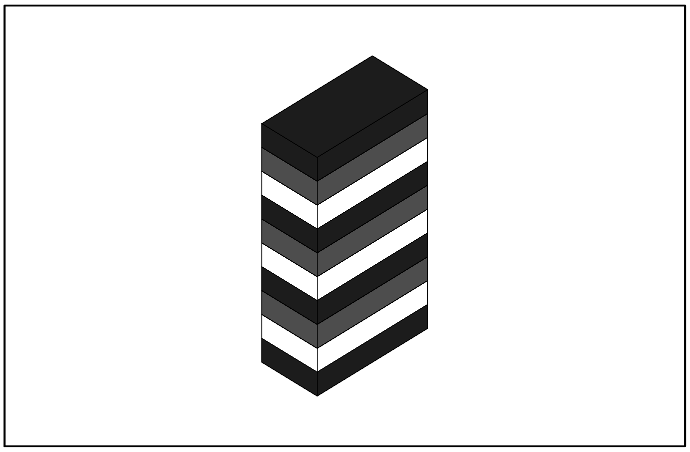


It is no harder to use alternating red, white, and blue bricks. We just need to know how to have Maple choose among three alternatives. The format is

if ...then...elif.. then ...elif ...then...fi;

Here is an example. Recall that the possible remainders upon division by 3 are 0,1, and 2.

```
bricks:=NULL:
for t from 0 to 9 do
if (t mod 3) = 2 then
tmpbrick:=CBP([0,0,2*t],red) elif (t mod 3) = 1 then
tmpbrick:=CBP([0,0,2*t],white) elif (t mod 3) = 0 then
tmpbrick:=CBP([0,0,2*t],blue) fi:
bricks:=bricks,tmpbrick; od:
plots[display]([bricks]);
```



The Boolean operators 'and' and 'or' can be used the with the if then elif then else ficonstruction in expressions like the following defintion.

```
compare:=proc(a,b)
if (a<0) or (b>1) then print( a , ‘is less than 0 or ‘,b,‘ is greater
than 1‘)
elif (a=0) and (b=0) then print(‘both are zero‘)
elif (a>0) then print(‘the first is positive‘)
else print(‘the other tests fail‘) fi:
end:
```

```
compare(-1,0);
```

$$-1,\ \textit{is less than 0 or },\ 0,\ \textit{is greater than 1}$$

1.3.1 *Exercise: Is it possible to find numbers a and b so that 'the other tests fail' is the output?*

1.4 Exercises: If .. elif .. fi; and if .. else.. fi

1.4.1 *Exercise: The following procedure "tosses" a coin and reports "heads" or "tails". Modify it to have the computer toss two coins and report (heads,heads), (heads, tails), etc, Note that the procedure has no inputs - this is perfectly ok. Note also that the "rand" function used actually returns a procedure (called tmp) in below which itself has no variables.*

```
toss:=proc(x)
local coin:  coin:=rand(2):
if coin()=0 then print(‘heads‘) else print(‘tails‘) fi:
end:
```

1.4.2 *Exercise: Make a procedure which throws a six-sided die - that is one which returns a random digit from 1 to 6 (hint: die:= rand(1..6);)*

1.4.3 *Exercise: Adapt the following procedure to draw a red square if its input is between 0 and 1, a blue rectangle if its input is between 3 and 4, and a green triangle otherwise.*

```
drawit:=proc(n) local sq, tri, ans;
sq:=plots[polygonplot]([[0,0],[1,0],[1,1],[0,1]],
color=red,thickness=4);
tri:=plots[polygonplot]([[0,0],[1,0],[1,1]], color=blue,thickness=3);
if (n>=0) and (n<=2) then ans:=sq:
elif (n>=3) and (n<=5) then ans:=tri;
else ans := plots[polygonplot]([[0,0],[2,0],[2,1],[0,1]],
thickness=3);
fi:
ans;
end;
```

1.5 A vertical chromatic brick placer

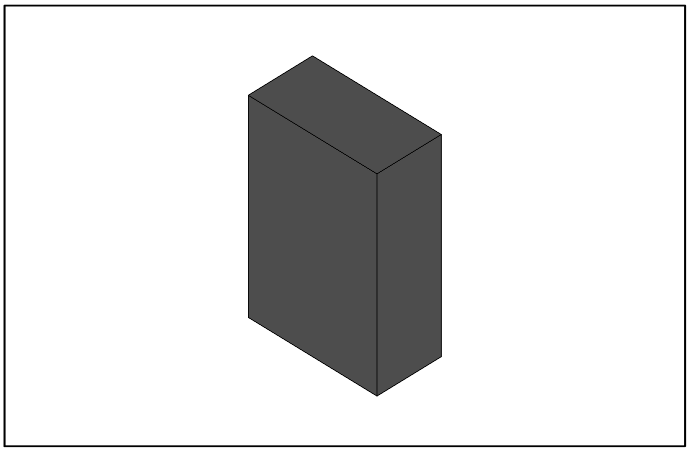
If we had vertical bricks we could make more elaborate patterns. Lets cook up a quick vertical chromatic brick placer called 'VCBP'.

```
VCBP := proc (shift,clr )
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0] + shift; v2 := [0, 0, 8] + shift; v3 := [0, 4, 8] +
shift;
v4 := [0, 4, 0] + shift; v5 := [2, 0, 0] + shift; v6 := [2, 0,
8]+shift;
v7 := [2, 4, 8]+shift; v8 := [2, 4, 0]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color = clr, style = patch, scaling =
constrained )
end:
```

We can test it with

```
VCBP([0,0,0],red);
```

Now we can begin to make more elaborate placement. Note that we are using both CPB and VCPB below - a plotting error will reult if either is not compiled



```
bricks:=NULL:
END:=0:
for t from 0 to 19 do
if (t mod 3) = 2 then tmpbrick:=VCBP([END,0,0],red);
END:=END+2;
elif (t mod 3) = 1 then tmpbrick:=VCBP([END ,0,0],white);
END:=END +2;
elif (t mod 3) = 0 then tmpbrick:=CBP([END,0,0],blue);
END:=END+8;
fi:
bricks:=bricks,tmpbrick; od:

plots[display]([bricks]);
```

```
Error, (in plots/object2plot) bad object to display, string
```

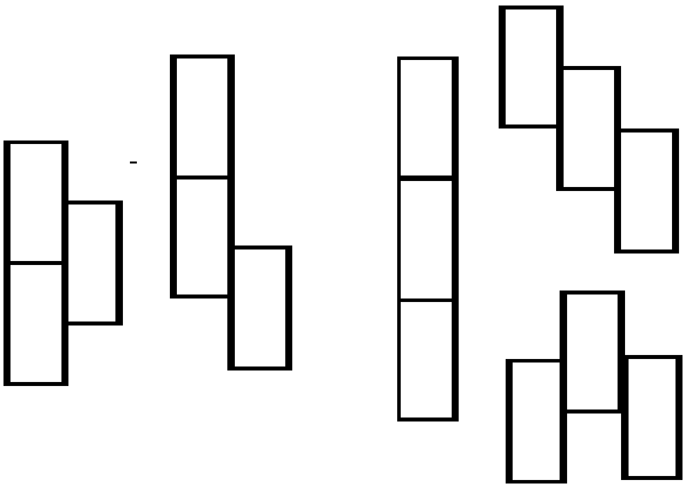
1.6 Exercises: Colored Brick Stacking

1.6.1 *Exercise: Make a stack of 15 bricks that alternate in color: white, red, blue, yellow - with a white brick on the bottom.*

1.6.2 *EXERCISE: Do the previous problem, making the stack end with a white brick on the top.*

1.6.3 *Exercise: Use a "do..od" command to stack sixteen bricks in eight layers such that each layer has two side-by-side bricks with each brick on every layer but the first lying across both bricks on the level below it.*

1.6.4 *EXERCISE: Place nine bricks in a line on the base level such that except for the ends each horizontal brick is flanked by two vertical bricks and each vertical brick is flanked by two horizontal bricks.*



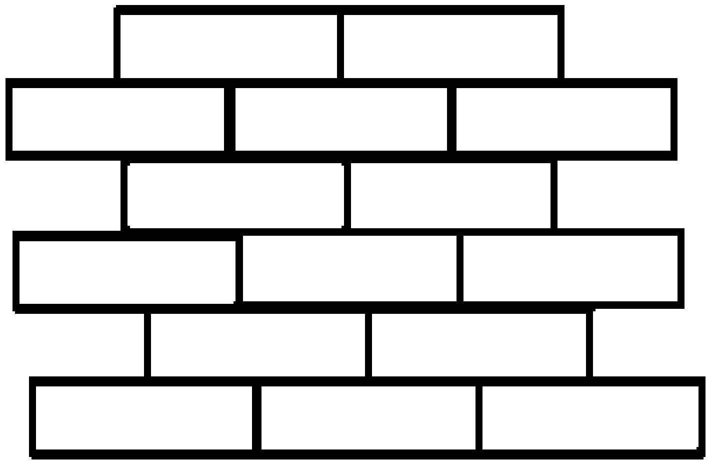
1.6.5 *EXERCISE: A puzzle called Brick by Brick which is marketed by the Binary Arts Corporation of Alexandria, Virginia consists of five pieces, each made of three bricks The left diagram shows a two dimensional sketch. The object of the puzzle is to assemble the five pieces in to various shapes. Pieces may be inverted or rotated 180 degrees. Thus for instance the upper left piece could be used in either of the orientations indicated at the right. One of the pieces has four orientations.*

1.6.5.1 *a. Make a three dimensional representation of each of the possible orientations of each piece.*

1.6.5.2 *b. Make a set of procedures, one for each possible orientation of one of the pieces, which will place any piece wherever you want it.*

1.6.5.3 *c. Merge your procedures into a single procedure which inputs the name of the piece, the orientation, where you want it, and a color.*

1.6.5.4 *d. Assemble the following pattern. Make one solution with the bricks all the same color and a second with each of the five pieces a different color. .*



1.6.5.5 *e. Is this arrangement possible?*

1.6.5.6