CC3 Macro

PART 2

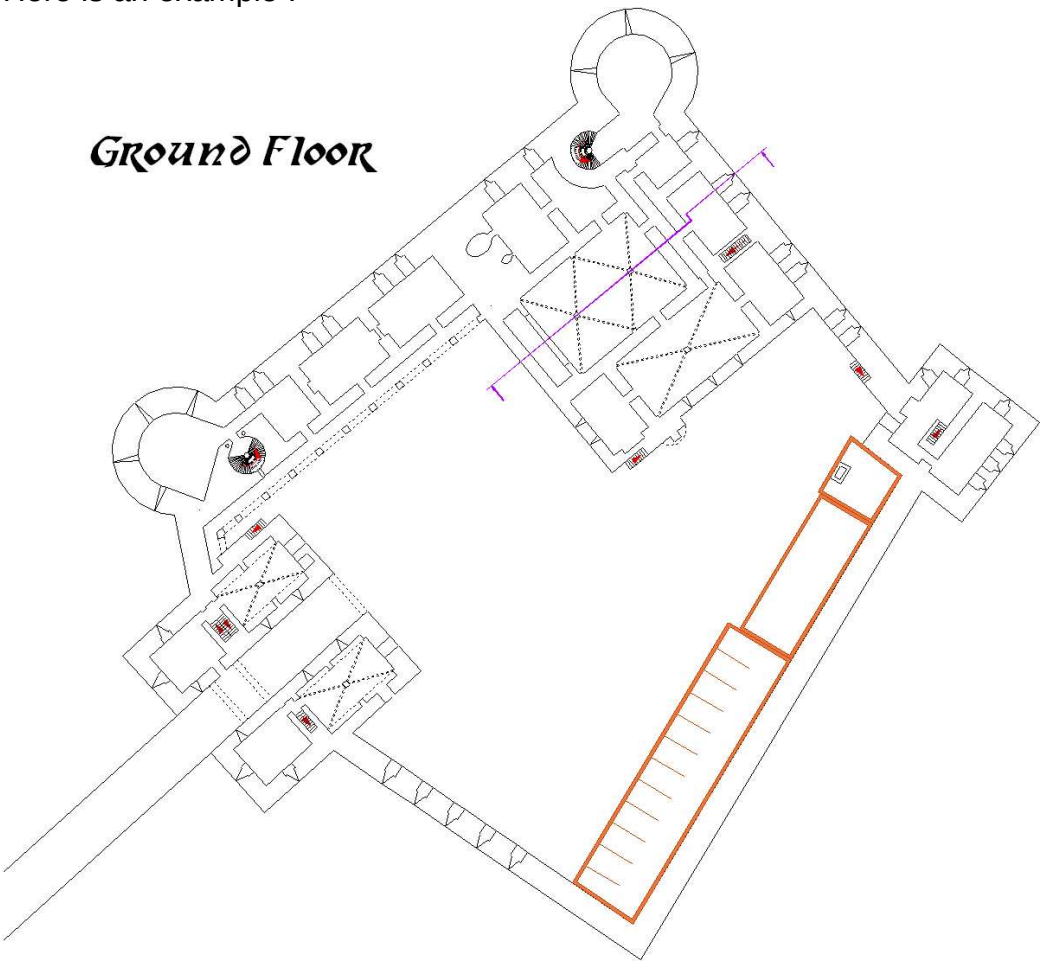
Writing a new macro

Introduction

I assume that you read the first part or that you know enough about CC3 macro writing and CC3 itself to understand part 2.

When I draw building plans I often use the technical style where walls are white polygons outlined in black.

Here is an example :



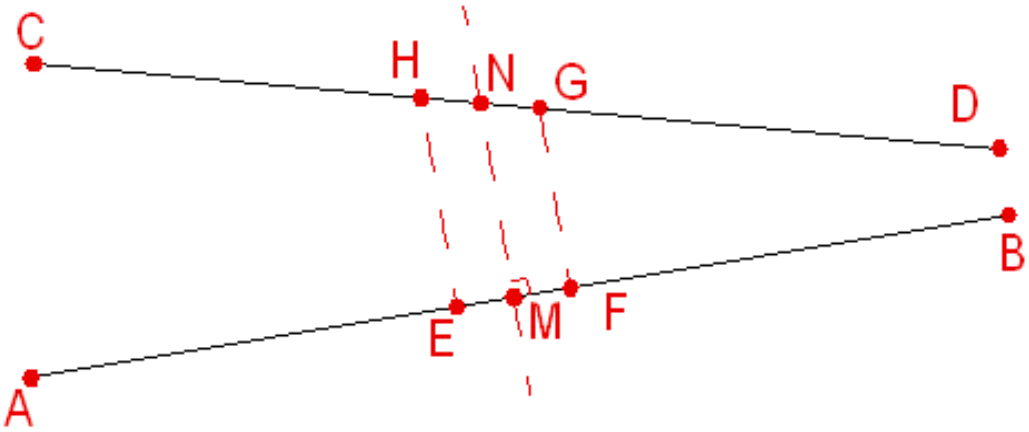
Usually, I draw walls without opening and then add door, windows, arrowslits, whatever...

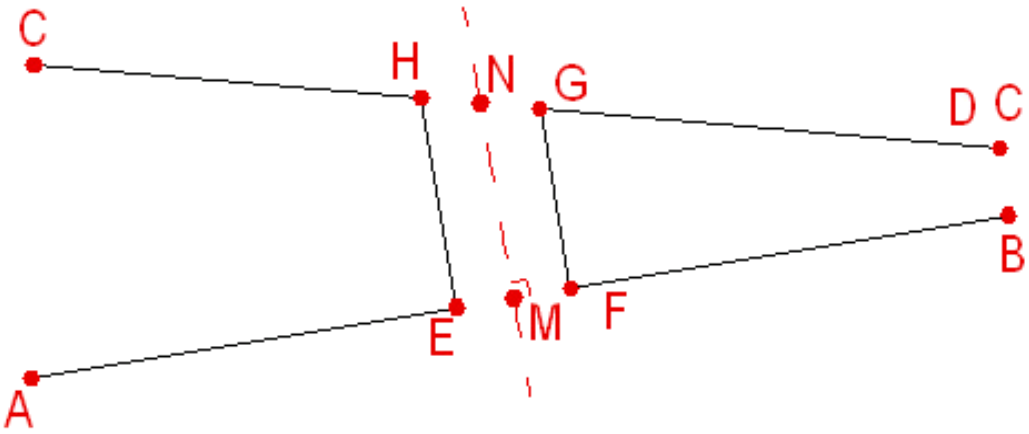
That implies an extensive use of SPLIT and TRIM commands, so why not design a macro doing it for me ?

This part will be about writing the DOOR macro.

The difference between doors and windows is that a door leaves a gap in a wall whereas windows only add lines between the two sides of the wall (otherwise you won't see the difference).

Look at that :





The picture above shows the two sides of the wall (in black).
The picture below shows what I want the macro to do (in black).
What's in red won't appear, but will have to be used in the macro.

I want the door to be centered on point M on the bottom line and perpendicular to that line. Usually, the second line would be parallel to the first, but let's make it more general.

How would you do that using CC3 ?

1. Start a line at M using ON modifier, perpendicular (F12) to the bottom line.
2. OFFSET this line on both sides half the door width.
3. Split both wall lines (at M and N for example)
4. TRIM lines TO INTERSECTION.

The point to select

I decided to center my door on M. I could have started at A with only one offset to do but I find it better to work with symetry.

Seems easy enough, eh ?

I learned a lot analysing macros written by other people. Here I propose you to build the macro one step at a time so this work will be (I hope) easier for you.

1. Starting the macro, SELSAVE and SELREST

In part1, when I had to change the selection method, I restored it with a SELBYD command. It's standard (isn't it ?).

But what if, for some obscure reasons, it was not SELBYD that was active before the macro ? Two commands come handy : SELSAVE and SELREST!

SELSAVE stores the selection method, and SELREST RESTores it to what SELSAVE SAVEd.

Here is the beginning of the macro :

```
MACRO DOOR1
SELSAVE
SAVESETTINGS
:MacroDone
GETSETTINGS
SELREST
ENDM
```

ECOFF ?

I usually keep ECho ON while designing a macro, so I can keep an eye of what's going on when an error occurs (it goes to fast to read anyting else). When there's an error, the command line usually shows last command so it's easier to localise the error point in the code.

Note the symetry of it...
Don't forget the blank line at the end !

This macro does nothing, right ?
However, this is a good way to begin a macro because you won't forget those important steps.

2. Selecting the first line.

The first line is the one the door opening will be perpendicular to. To select a line, we will use the GE command.

Do you remember how GE works ? It doesn't really store an entity but the point of selection.

To illustrate that try this

```
MACRO DOOR2
ECOFF
SELSAVE
SAVESETTINGS
GE vline1 ^DSelect first line
GP vpoint ^DSelect another point outside line
COLOR 2
LINE vline1;vpoint;
:MacroDone
GETSETTINGS
SELREST
ECON
ENDM
```

Variable names

Because I'm the lazy guy, I tend to use small variable names.

Il will drop the 'var' prefix from part on and only use v now.

If this won't be a tutorial-like, I would probably have chosen vl1 instead of vline1.

Capitals or not ?

CC3 does not make the differences between ECOFF and ecoff.

It's just another tip : commands in capital and variables not. It makes the code easier to read.

Note line 8 : `LINE vline1;vpoint;` (don't forger the semicolon)

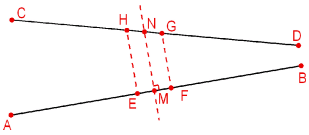
:MacroDone ?

The MacroDone label has no utility here. Once again it is added to prepare the macro.

`vline1` should be an entity variable, right ? `LINE` has no trouble with it and takes this variable as a point variable...

I hope this is clear enough. I needed some time to understand that.

The good side of it is that we selected our M point and our line in one click !



3 Points E and F

Those points are on the first line, each at half the door width from M.

First we ask the width of the door and then we place E and F on the line using polar coordinates :

```
MACRO DOOR3
ECOFF
SELSAVE
SAVESETTINGS
GE vline1 ^DSelect first line at insertion point
IFERR MacroDone
GP PM vline1
GOLAYER TOERASE
SELBY1
CHANGEL vline1 TOERASE
GE vline2 ^DSelect second line
IFERR MacroDone
CHANGEL vline2 TOERASE
GOLAYER JDRWALL
COLOR 2
LWIDTH 1
GV vdoorw 2.5
GV vdoorw ^DEnter Door Width: (2.5')
IFZ vdoorw MacroDone
GBRNG vAngle1 % 0 vLine1 % 100 vLine1
GP PE ref PM <vAngle1,vdoorW/2
GP PF ref PM <vAngle1+180,vdoorw/2
LINE % 100 vLine1;PE;
LINE % 0 vLine1;PF;
:MacroDone
GETSETTINGS
SELREST
ECON
ENDM
```

GP PM vline

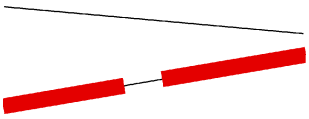
As I said before, an entity is sometimes considered as an x,y value and here I use that to *officialise* point M.

COLOR 2
LWIDTH 1

Of course we wouldn't want that for the macro but it's a good way to show what it does through design.

E and F

Angle 1 is from the 0% side so E will be on the **other** side. Adding 180° to the angle changes side (relative to M).



result

4 H Point

Now we're going to do real math. If you're allergic to math you can skip this section but most macros involve some math...

(EH) and (FG) lines are perpendicular to the (AB) line and H and G are on the (DC) line. Let's look at H first as G will be similarly found.

Scalar/dot product

. represeents the dot product sometimes known as scalar product.

Points and vectors

If O is the 0;0 point, an x,y M point has the same coordinates as $\overrightarrow{OM}$.

It goes like that :

$\overrightarrow{EH} . \overrightarrow{EM} = 0$ (1)
$\overrightarrow{DH} = k \times \overrightarrow{DC}$ (2) *k* is a real number

CC3 has no special input for vectors and it doesn't need it. Vectors are essentially defined (here!) by their x,y coordinates so for the system is looks like a point.

Coordinates notating

It's conventionnal to note x_E the first coordinate of E and y_E the second coordinate.

Dot product again

Analyticaly, the dot product of *x,y* vector and *x',y'* vector is $x \times x' + y \times y'$

Vectors

Analytically, $\overrightarrow{AB}$ is the $x_B - x_A, y_B - y_A$ vector.

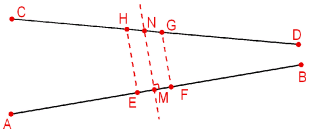
Let's go analytical

$(x_H - x_E)(x_M - x_E) + (y_H - y_E)(y_M - y_E) = 0$ (1)

$$\begin{cases} x_H - x_D = k(x_C - x_D) \\ y_H - y_D = k(y_C - y_D) \end{cases}$$ (2)

from (2) we get

$$\begin{cases} x_H = x_D + k(x_C - x_D) \\ y_H = y_D + k(y_C - y_D) \end{cases}$$ (3) inserted into (1)



$(x_D + k(x_C - x_D) - x_E)(x_M - x_E) + (y_D + k(y_C - y_D) - y_E)(y_M - y_E) = 0$ and then

$k[(x_C - x_D)(x_M - x_E) + (y_C - y_D)(y_M - y_E)] = -(x_D - x_E)(x_M - x_E) - (y_D - y_E)(y_M - y_E)$

thus

$$k = \frac{(x_E - x_D)(x_M - x_E) + (y_E - y_D)(y_M - y_E)}{(x_C - x_D)(x_M - x_E) + (y_C - y_D)(y_M - y_E)} = \frac{\overrightarrow{DE} . \overrightarrow{EM}}{\overrightarrow{DC} . \overrightarrow{EM}}$$

Wow. Got our *k*. Now it's easy to get H from (3) above.

Wake up !

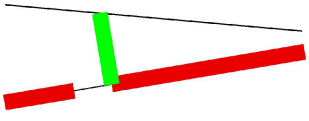
That was the easy part. No kidding. CC3 does not know the dot product (well, we could write a macro for it !).

First we have to know all those x and y values. For that, we have GETX and GETY commands.
Syntax is : `GETX varName PointVar`

Then we have to compute k , x_H and y_H .

Here are the lines to add to our macro :

```
GETX vxC % 0 vLine2
GETY vyC % 0 vLine2
GETX vxD % 100 vLine2
GETY vyD % 100 vLine2
GETX vxE PE
GETY vyE PE
GETX vxF PF
GETY vyF PF
GETX vxM PM
GETY vyM PM
GV vdot1 (vxC-vxD)*(vxM-vxE)+(vyC-vyD)*(vyM-vyE)
```



result

```
GV vdot2 (vxE-vxD)*(vxM-vxE)+(vyE-vyD)*(vyM-vyE)
GV vk vdot2/vdot1
GV vxH vxD+vk*(vxC-vxD)
GV vyH vyD+vk*(vyC-vyD)
GP PH vxH,vyH
COLOR 1
LINE PE;PH;
```

(see doors.mac for the whole macro, here we have door4)

5 What if... (#1)

,,,the two lines are perpendicular, or almost perpendicular ?

In that case, H will be very far away (theoritically at the infinite if lines are really perpendicular).

We almost have what we need here. One of the first things students learn about the dot product of two vectors is that it is equal to zero only if one vector is nil OR if vectors are perpendicular.

Did you note that I made the macro compute the two dot products in two variables ? I could have written

```
GV vk (vxE-vxD)*(vxM-vxE)+(vyE-vyD)*(vyM-vyE)/((vxC-vxD)*(vxM-vxE)+(vyC-vyD)*(vyM-vyE))
```

even if it's a very long command.
Using two variables was not innocent, I confess.

If the two lines are perpendicular, so are $\overrightarrow{DC}$ and $\overrightarrow{EM}$ because D and C are two points of line 2 and E and M two points of line 1.
We can then use vdot1 in a test.

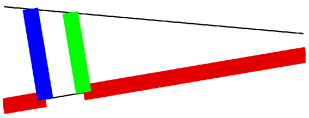
```
IFZ vdot1 MacroDone
```

would ensure that no perpendicular lines are selected. But was if they are almost perpendicular, but not quite ?

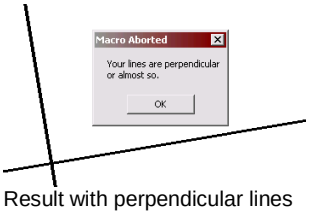
```
IFN vdot1-0.001 MacroDone
```

Looks tempting, but alas! dot products are signed ! The test must then not be on vdot1 but on it's absolute value (see part 1 page 7). The code will then be :

```
GV absdot1 vdot1
IFP absdot1 AbsoluteDone
GV absdot1 -absdot1
:AbsoluteDone
IFN absdot1-(vdoorw/1000) Lperp
...
GO MacroDone
:Lperp
MSGBOX Macro Aborted
Your lines are perpendicular
or almost so.
<blank line>
:MacroDone
...
```



Result of door5 macro. Note that each step is in another color. That's something I encourage my students to do while drawing geometry. It helps to read the picture and clears the mind. Of course, one the macro is complete, this will dissapear as well as the line thickness.



Result with perpendicular lines

vdoorw/1000

The absolute value of vdot1 is tested against vdoorw/1000 instead of 0.001.

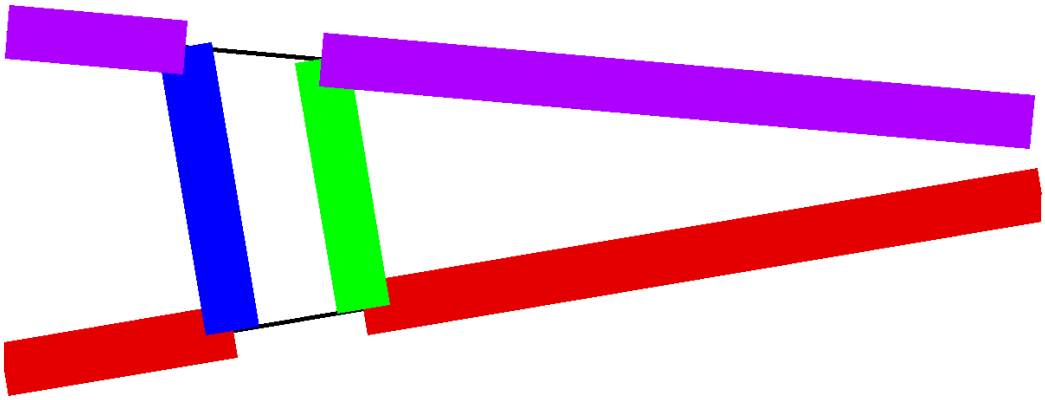
One never knows at which scale the user works. If the scale is very very low, vdoorw would logically also be so.

It also means that drawing lines AE and FB are placed after the test or the macro will indeed be aborted but the first lines would have been drawn anyway.

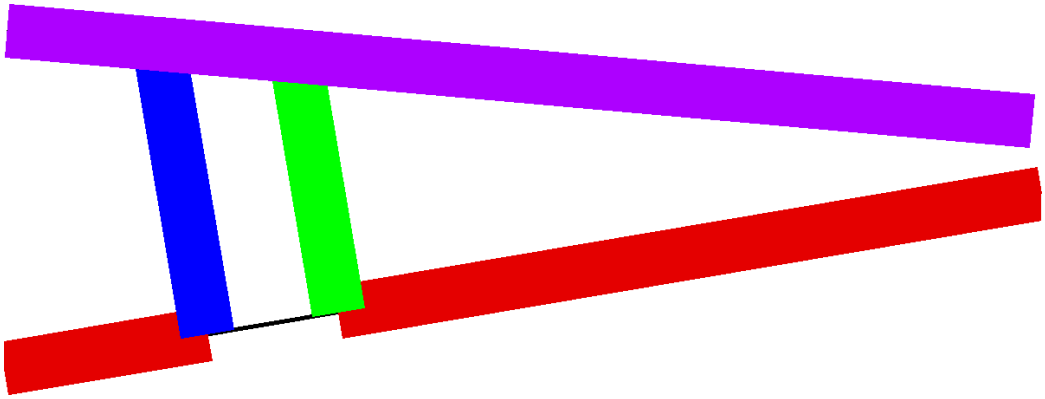
The door5 included in the doors.mac file has this already built in along with point G whose math is similar to point H.

6 Almost done. Or ?
We just have now to trace lines DH and CF. That's now kids play :

```
GP PC % 0 vline2
GP PD % 100 vline2
LINE PD;PH;
LINE PC,PG;
```

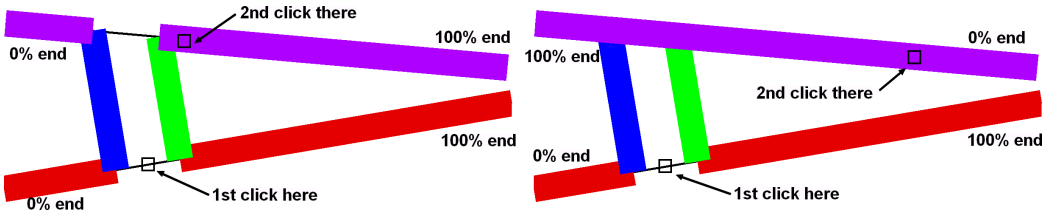


Yipee ! Lets make another try :



What the H### ?

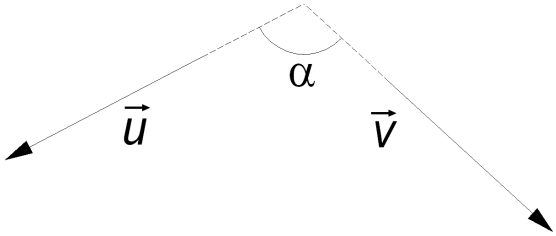
7 What if... (#2)
C and D are at the wrong ends ?
Remember that the % modifier responds to the nearest end after selection. For the second example, I deliberately choose to select line two at the wrong end :



You could say « let it be », the user would know. That's ok if you're the user but if you share your macro, it could be me! So what ?

Well, dot product rescues us once again. I know, more math. Can't help it...

The non analytical version of the dot product is that it yields the product of the lenghts of the two vectors times the cosine of the angle made by those two vectors :



$$\vec{u}.\vec{v}=(\text{length of } \vec{u}).(\text{length of } \vec{v}).\cos(\alpha)$$

That means that if angle is between 90° and 180°, t he dot product will be less than 0. All we need now to get D and C right is a dot product of two vectors of line1 and line 2.

But we already have that ! It's vdot1 ! But because D and C were assumed to be in the position shown in first drawing, it's if vdot1 is more than 0 then D and C must be switched and that's it :

```
IFN vdot1 DCok
GP vBufP PD
GP PD PC
GP PC vBufP
:DCok
```

The macro door is now ready. I just added an askbox at the end to make erasing line1 and line2 an option.

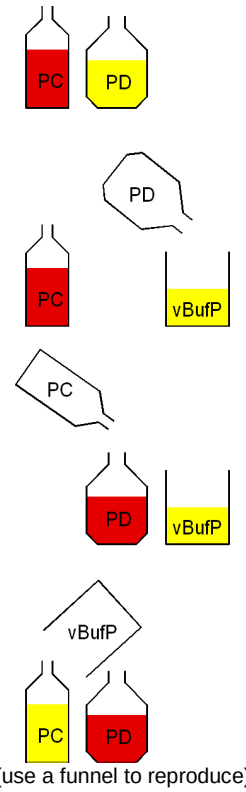
The room next page took less than a minute. If, like I did, you use a polygon, explode it before running the macro otherwise all the polygons will be erased ! (How I love GE !)

Why 3 GP ?

I could have just used two with :

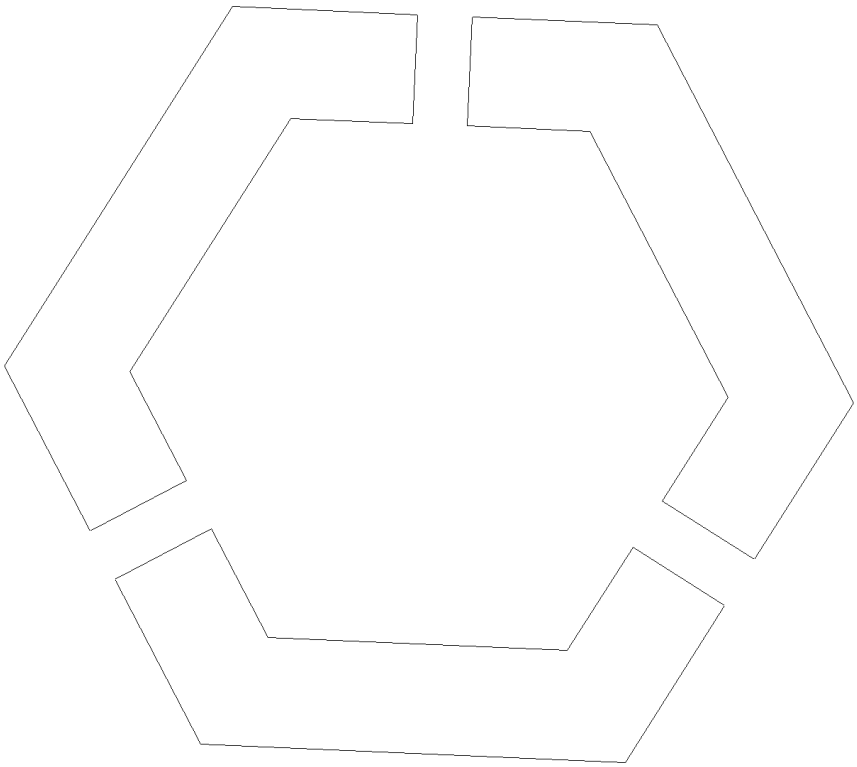
```
GP PC % 100 vLine2
GP PD % 0 vLine2
```

but I wanted to show you an old programmer's trick : swithching two variables. It involves a third one because
`GP PD PC`
would have lost the first value of PD. It's like swapping the content of two bottles, one with strawberry juice and the other with apple juice : you first have to empty one of the bottles in a third container :



(use a funnel to reproduce)

Such variables are often called buffers, hence the name.



That's how I did it :

Rigthclick Polygon tool->regular polygon
Offset by 5 inside
Explode both polygons
Run door macro using F4 (midpoint) to place the door in the middle of the walls.

CONCLUSION

1) To write a macro, start small.

2) Leave ECOFF for the final stage, but SELSAVE and SAVESETTINGS are handy (if you don't forget SELREST and GETSETTINGS at the end)

3) Use debuggers trick as I did with colors and line widths

4) Don't despair. Trials and errors is the way to success.

5) Take out your old math books or go visit your old teacher, he'll be pleased (did I mentionned I teach math in the real live ? Don't forget the cookies).

New commands			
Command	**Effect**	**Syntax**	**Page**
GETX	Extract first coordinate of an x,y point	`GETX varName Point`	5
GETY	Extract second coordinate of an x,y point	`GETY varName Point`	5
SELREST	Restore selection method stored by SELSAVE	`SELREST`	3
SELSAVE	Saves selection method to be restored by SELREST	`SELSAVE`	3

Well, that's not much but part 1 already gave you a lot to work with...

This ends part 2. To be continued...