AutoCAD Shape/Font Definition Files

M. Şahin GÜVERCİN
İnş. Eng.

www.cizimokulu.com

Dedicated to Akya Derin Nazlıoğlu's uncle Alp Eren Güvercin

M. Şahin Güvercin
17.11.2012 - Ankara

TABLE OF CONTENTS

1. Shape/Font Description Files

After creating and compiling your own symbols and fonts, you can use them in your drawings. Shapes are objects that are very similar in use to Blocks. You can place the objects that you will load with the LOAD command from the compiled Shape definitions file (*.shx) at the scales and angles you specify in the desired parts of your drawing with the SHAPE command, and you can change the position, scale and rotation angles of your existing Shape objects.

AutoCAD SHP Fonts are special types of Shape Definition files. Characters (letters) are defined in the same way as Shapes.

Blocks are flexible objects that are much easier to use and apply than shapes. However, Shapes are much more efficient objects to use and store in the drawing by AutoCAD. If there will be a large number of simple drawing pieces and speed is more important, user-defined shapes are much more efficient. While block definitions are contained in the drawing file, shape definitions are contained in a standalone SHX file, so when transferring drawing files in any way, Compiled Shape Definition files (*.shx) must also be transferred together with the drawing file.

AutoCAD Shape and Font files (*.shx) are obtained by compiling Shape Definition Files (*.shp) from AutoCAD. Shape Description files can be created and modified in ASCII format with any Text Editor (such as Notpad).

In the definition files, each Shape or Character (any in the Font) is defined and written in exactly the same shape. In this respect, Font and Shape files are no different from each other. If the file (*.shp) is written to be used as a font, first of all, the font related to (Font Name, Type, Character Size, Character Code Table,... etc) definitions. If the information entered at the beginning is directly the Shape Definition, this file will no longer be a Font File, but a Shape File.

First of all, we have to clarify one thing. Shape and Font files are identical in terms of their creation (except for the first definition is different) and the shape of the definitions, as well as the file type obtained after the Compile process (*.shx). These files, which look the same at first glance, are completely different in how they are used (called) in the AutoCAD environment. Font files can be associated with a Text Style and added to Text, DText, MText,... or Attribute definition commands. Shape definition files, on the other hand, can be loaded first and then used with the SHAPE command. Both the Font file and the Shape file can be used with direct reference within a Line Type definition. Under this topic, Shape Definition files will be discussed first, and the details of Font files will be examined in the following sections.

Don't forget that being able to create your Shape definitions is a very powerful and valuable skill. Acquiring such a skill is an extremely complex task that requires attention and patience.

Shape files (*.shx) can be created from selected objects from the screen by using the MKSHAPE function from the AuoCAD Express Tools menu. Since the files created by this method are directly in SHX format, it is clear to the user that many details will remain dark within the limits of the function. The aim here is to explain all the details of defining Shape by writing code for those who are interested and need it.

In the Shape definition file, each line can be written in the desired length. On long lines, those after 128 characters are not compiled. During the compile process, blank lines and characters to the right of semicolons are not processed. This means that comments can be written in the Shape Definition file using semicolons.

Each Shape Definition consists of a header line followed by (at least) one or more lines. Specification Bytes consist of numeric (integer) values separated by commas and terminated with 0 (zero).

***shapenumber,defbytes,shapename**
specbyte1,specbyte2,specbyte3,...,0

shapenumber : An integer ranging from 1 to 258 (up to 32768 in Unicode fonts) preceded by an asterisk (asterisk). In non-Unicode font files, 256, 257, and 258 are used for symbolic designations such as the Degree sign, plus/minus sign, and Diameter sign. In Unicode fonts, this section is used as hexadecimal numbers such as U+00B0, U+00B1. In font files, shapenumber is the ASCII code number of the corresponding character. In Shape files, any number can be given to that Shape.

Defbytes : The number of Definition bytles (specbytes) used for the definition of Shape. 0 (zero), which will be used to terminate the definition, will also be counted. A maximum of 2000 bytes can be used for each Shape Definition.

Shapename : The name given to the defined Shape. The Shape Name should be written in capital letters. Lowercase Shape Names are ignored and are generally used in Font files to mean the character's tag or description.

specbyte : Shape definition byte. Each definition byte is written as a numeric (integer) code that expresses the vector length, direction, and direction, or special meaning. Definition bytes can be written as Decimal numbers or Base Hexadecimal numbers. If the first character of the code number in question is 0 (zero), the next 2 digits will be read as a Hexadecimal number.

2. Vector lengths and Direction codes

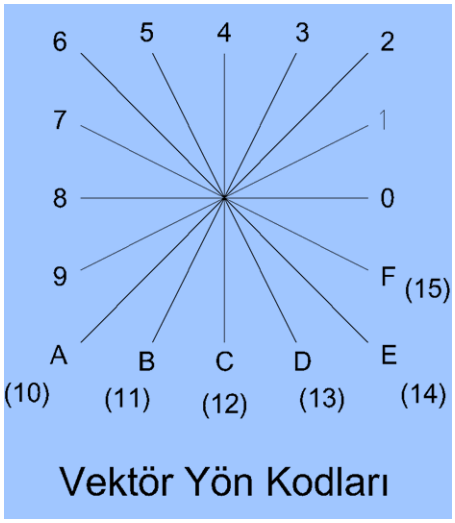
The vector definition, which is the simplest of the Shape Definition Bytes, consists of the code that defines the Vector Length in units and the Direction code of the vector. Each Vector Length and Direction code group consists of a sequence of 3 characters.

The first character must be 0 (zero) so that AutoCAD recognizes the following 2 characters as a Base 16 (Hexadecimal) value.

Second character : Refers to the length of the vector in units. The vector length must be a valid base 16 number in the range 1 through F (15). So basically, we can create vectors with a maximum length of 15 units. Don't let this limitation bother you too much, we will explain how to create vectors of different lengths in the special codes section later.

Third character : determines the direction of the vector.

The figure below shows the Vector Direction codes.



The vectors in all directions shown in the figure are drawn with the same definition of length from the origin. The diagonal vectors (in terms of their projections on the X and Y axis) are extended to coincide with the corresponding X and Y values of the nearest Orthogonal vector. This extension process is similar to the "snap to grid" phenomenon in AutoCAD. Don't worry about the fact that vectors can be created in 16 directions here, in the future we will explain how to create vectors in every desired direction and even draw Arcs and Curves in the special codes section.

Below is an example (randomly selected) sample shape definition named DBOX with number 230.

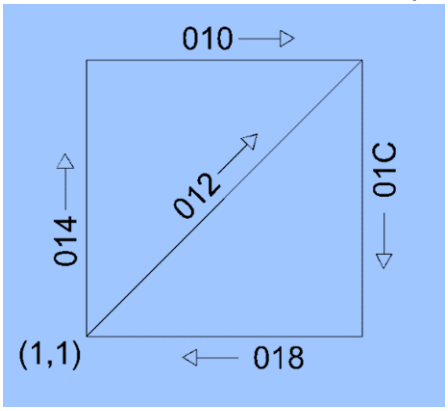
```
*230,6,DBOX
014,010,01C,018,012,0
```

When the definition bytes in the example are applied in the given order, a shape consisting of a box and its diagonal with a width of 1 unit, a height of 1 unit emerges. If we explain the operation of the code step by step;

***230,6,DBOX** -> Shape number is 230, we use 6 bytes in the definition, shape name DBOX

014 -1 unit in > 4 directions
010 -> 1 unit in the 0 direction
01C -> 1 unit in the C direction

018 -1 unit in > 8 directions
012 -1 unit in > 2 directions
0 - End of > shape definition



I would especially like to warn you here, if you have a resistance in your mind as to why the hypotenuse of the triangle with a base and height of 1 unit is 1 instead of root 2, do not continue the subject, read what is explained above again. When I was new to programming languages, I couldn't accept the expression A=A+B, How come? There is no such equality! I couldn't break the resistance in my mind. I didn't find peace until I finally discovered that the equal sign (=) in this expression was used in the sense of assignment, not equality. Here, too, after drawing the square, do not think about how its diagonal is 1 unit. Instead, when I define a diagonal vector with a length of 1 unit from one corner of the square, it is useful to think that this vector goes all the way to the diagonal corner.

3. Special codes in definition bytes and their meanings

There is no difference whether the special codes described here are written in hexadecimal or direct decimal numbers, starting with 0. So in the code, 00C and 12 mean the same thing.

000 : Refers to the end of the definition of Sahape.

001 : Enable the drawing state (Download the pen). By entering code 1 at the beginning of each Shape definition, the drawing state is activated and the specified vector(s) are drawn.

002 : Exit the active drawing state (Remove the pen) When the drawing state is removed from the active state by entering Code 2, the new position is navigated without drawing the specified vector.

003 : Dividing vector lengths by the next byte.

004 : Multiplying vector lengths by the next byte.

Code 003 and Code 004 are followed by an integer scale factor (from 1 to 255) that is 1 byte long. The scale factor (multiplication or quotient) operation is cumulative. That is, if 003.2 is followed by 003.6, the following vectors will be 2x6=12 in 12 long. In Shape definitions and Text Fonts, especially when a child shape is used, the scale factor should be returned to its old value at the end of the shape definition. AutoCAD doesn't return the scale factor on your behalf.

005 : Storing the current position.

006 : Return to the stored position.

With code 005, only 4 positions can be stored. Before a new position can be stored, the required positions must be retrieved with Code 006.

Otherwise, when a new position is saved using code 005 while the position stack is full, an error will not be encountered during the compile process, but when trying to use the defined Shape, the error message Position stack overflow in shape nnn will be received in the Shape definition with nnn.

When the position(s) stacked with code 005 are retrieved using code 006, the position stack will be emptied. Similarly, when more code 006 is applied than the current number of records in the Position stack, no error message will be received during the compile process, and the same error message will be received when attempting to use the defined Shape.

When storing and retrieving positions with code 005 and code 006, it should be noted that the Last In First Out principle applies.

007 : Draws the numbered Shape (the character whose number is entered) given by the next byte.

In non-Unicode font files, the byte following code 007 is an integer that represents the ASCII code of a character between 1 and 255. In Unicode font files, this character number is expressed as 2 bytes between 1 and 65535. In Shape Definition files, it is the shape number defined in the same file and given the number, the shape whose number is written will be drawn as a sub-shape.

When the shape is completed, it returns to the current shape definition. The drawing states defined by code 001 and code 002 and the scale values defined by code 003 and code 004 are not automatically reinstated at the end of the sub-shape.

When using Alt-Shape, the required bytes must be written in the called Alt-Shape to revert the Drawing State, Scale factor changes to their previous state.

Normally, vector definitions can be in 16 predetermined directions (given above in the form of Vector Direction Codes) and up to 15 units long. Although this makes the definition of shape very simple and efficient, limitations are imposed for some cases. By using code 003 and code 004, the vector length limit of 15 units is exceeded. Apart from the normal vectors mentioned using code 008 and code 009, vectors in more free and unlimited directions and directions can be defined.

008 : X-Y displacements are determined by the following 2 bytes.

8 is written as X-displacement, Y-displacement . The substitutions X and Y here are expressed in integers ranging from -128 to +127. For negative values, the - sign is placed at the beginning of the integer, for positive displacements, it is optional to write the + sign. Parentheses can be used to facilitate readability and traceability.

In the case of 8,(-10,3), a displacement of 10 units to the left and 3 units up is defined. When the 2 bytes following code 008 are completed, the shape definition reverts to normal vector mode.

009 : Multiple X-Y displacements are defined following this code, determined by pairs of bytes in series terminated by (0,0).

A series of non-standard vectors can be drawn using code 009. In principle, it can be considered as a multiple application of Code 008. The X-Y displacement pairs that define the series of vectors can be any number of them. The sequence of vectors following code 009 is terminated by the displacement vector (0,0).

9,(3,1),(3,2),(2,-3),(0,0) three non-standard vectors are drawn with the sample code and the Normal Vector state is returned. The series of X-Y substitutions following Code 009 should be terminated with (0,0). When AutoCAD sees the termination byte pair, it will know that the following bytes are either Regular Vector or Special code.

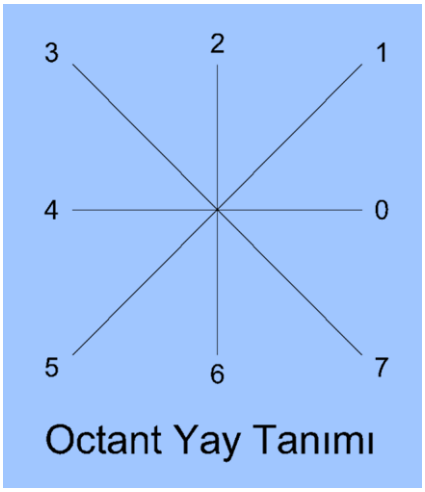
00A : Octant Arc boot with the following 2 bytes.

Octant Arc : Arc 1/8th of a full circle

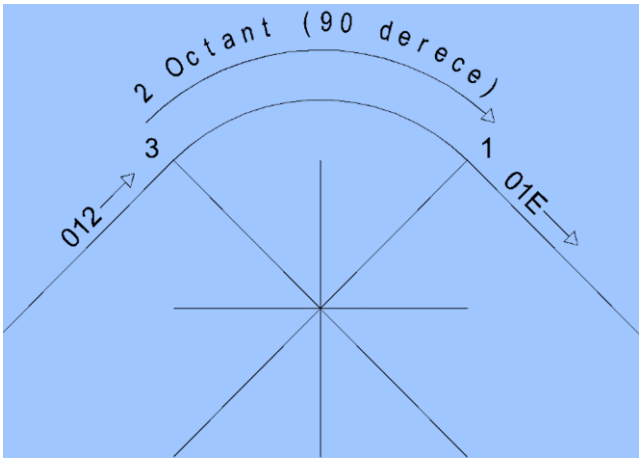
As can be understood from the definition, the process of drawing an arc starting from 45 degrees and one of its multiples, again up to 45 degrees and its multiples. As shown in the figure below, the 1/8 steps are determined to start at 3 o'clock and proceed counterclockwise.

Arc definition; ***10,radius, (-)OSC*** .

The first definition byte radius following the special code 00A (10) is an integer between 1 and 255. The second definition refers to the direction of the arc to be drawn (positive determines counterclockwise, negative value determines clockwise), the initial octant (from 0 to 7), and the length of the arc (in terms of octants). If 0 is entered as the arc length, 8 octants are taken, which means full circle. Parentheses can be used to improve readability and traceability.



... 012,10,(1,-032),01E,... If we take the Shape definition section as an example;



As can be seen in the figure, a vector with a length of one unit to the right, a vector with a radius of 1 unit, clockwise, starting from 3 octants, continuing from 2 octants, and a vector with a length of one unit to the right and down is drawn.

00B : Draw a Bow Fragment determined by the following 5 bytes.

Arc identification with code 00B is the most complex and difficult to understand among the special codes.

It is written as 11,start_offset,end_offset,high_radius,radius,(-)0SC.

start_offset and end_offset values determine how far the start and end of the arc are from the octant boundaries. high_radius refers to the most significant upper 8 bits of the radius value, the high_radius is 0 unless the radius value is greater than 255 units. By multiplying the high_radius value by 256 and adding it to the radius value, the arc with a radius greater than 255 is drawn according to the value found. The radius and end definitions are the same as the octant arc identification with code 00A (described earlier). Arcs with a radius value of up to 255 can be drawn with code 00A, while arcs with a radius value greater than 255 can also be drawn with code 00B.

start_offset is the difference (distance) in degrees from the starting octant (45 degrees and multiples) of the beginning of the arc. The difference in degrees we are talking about is obtained by multiplying by 256 and dividing by 45. If the arc we are going to define starts exactly from one of the octant points, the start_offset value will be 0. end_offset value will be determined by the same logic. It is obtained by subtracting the value in degrees calculated by the same logic from the octant limit at the end of the arc. If the end of the arc is exactly one of the octant points, the end_offset will be 0.

For example, to define an arc with a radius of 3 units, starting at 55 degrees and ending at 95 degrees;

11,(56,28,0,3,012). If we explain the coded values one by one;

start_offset = 56-> ((55 - 45) * 256 / 45) = 56
end_offset = 28-> ((95 - 90) * 256 / 45) = 28
high_radius = 0 -> (radius < 255)
Radius = 3
starting octant = 1-> The beginning of the arc is between 45 º and the next octant (90 º).
ending octant = 2-> The arc end is between the 90 º octant and the next octant (135 º).

It should be noted that operations are performed as integers.

00C : Arc identification with X-Y displacements and Bombe value.

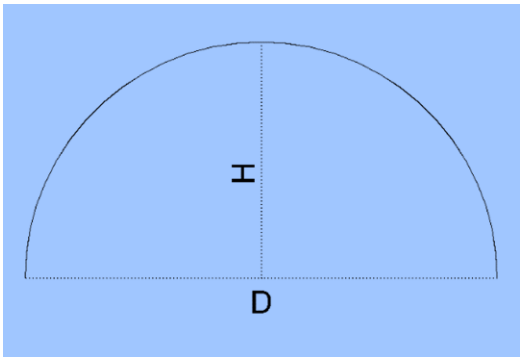
It is very similar to the identification of X-Y displacements with code 008 and code 009 described earlier. Using the special codes code 00C and code 00D, in addition to the X-Y displacements, the dished up is defined as a function of the displacement vector. A single Arc segment can be defined with 00C, while a multiarc can be defined with 00D until terminated by (0,0) definition byte.

The three bytes following the 00C code are used to identify Arc:

00C,X Displacement,Y Displacement,Dished (Bulg)

All three parameters that define the arc segment are integers in the range -127 to +127.

If the X-Y displacements are expressed by D, as seen in the figure below,
if H is the distance perpendicular to the line D of the midpoint of the arc,
The size of the bomb is: ((2 * H / D) * 127).



If the sign of the displacement is negative, the new position is determined by turning clockwise from the current position.

Since the semicircle has a camber value of 127 (or -127), larger arcs can be created using these codes with successive arc definitions.

A valid method is to use 0 for the camber value, and the arc with a curve of 0 is a straight line. (I used and explained this concept in the AutoLISP functions I shared.)

Although a straight line can be defined with code 008, for the straight line definition between multiple arc definitions, a large number of consecutive arc and straight line combinations can be defined without leaving the 00D code, but by entering the curved 0 of the part that will be a straight line. Spring definitions in groups of 3 following the 00D code are terminated with 0 displacement definitions in the form of (0,0). Bombe (bulg) is not entered after the last 0 displacement value. For example, the letter S can be simply defined as;

```
13,(0,5,127),(0,5,-127),(0,0)
```

The zero (0) curved arc definition is very useful for describing straight lines between multiple arc definitions; This method is much more efficient than leaving the definition of a multi-arc and entering a new multi-arc definition after defining a straight line. The number -128 cannot be used in the definitions of spring part and multi-spring.

00D : Multi-Arc definition defined by displacements and camber values.

The 00D special code is a multiple use of the 00C code, its usage and parameters are the same as 00C, as described above. The series of Arc segments identified by the special code 00D must be terminated with the vector (0,0).

00E : The part after this code is only valid for Font files and Vertical Text.

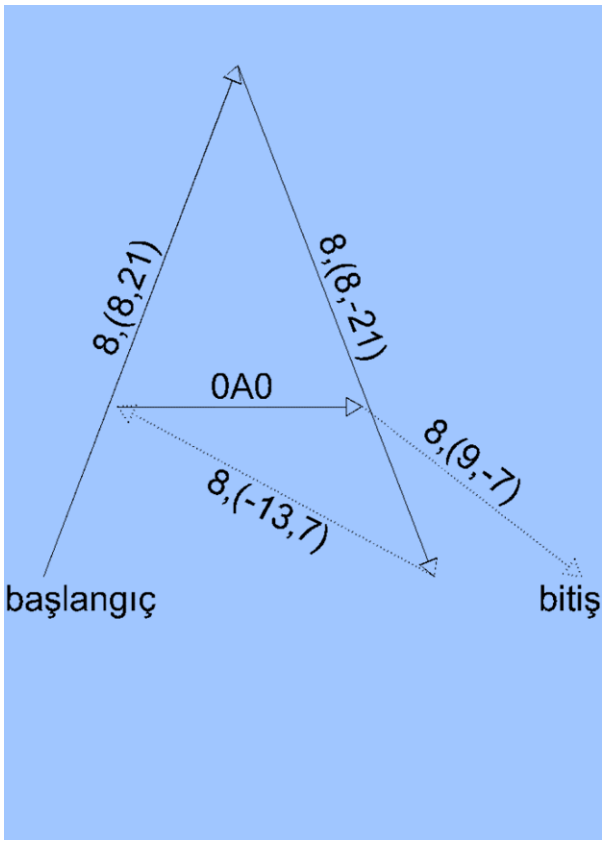
When the special code 00E is encountered in character definitions, the following codes are processed or ignored depending on the direction of the Text. If the text orientation is vertical, these codes are processed, otherwise they are not processed, the next code is processed and continued.

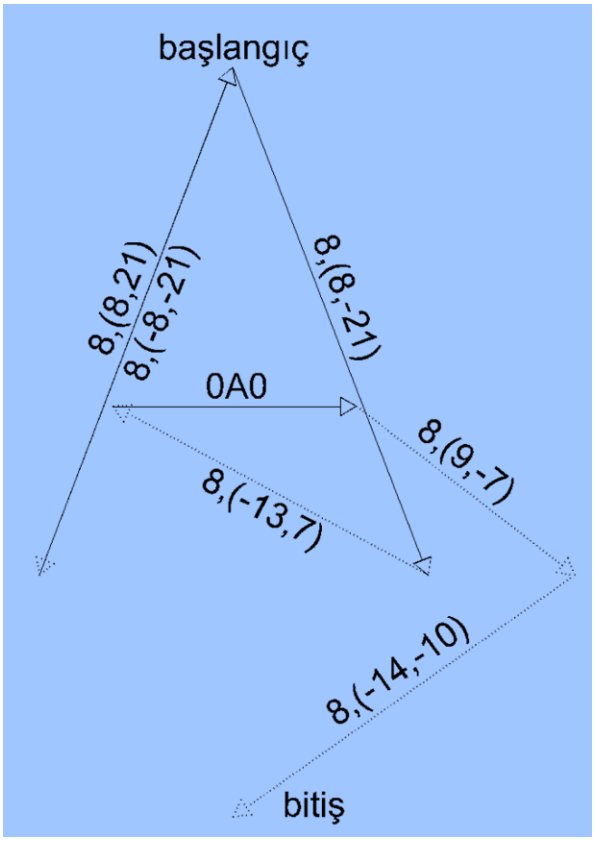
In horizontal text (Horizontal Text, regardless of the font angle), the starting point of the character (according to the direction of reading) is taken as the lower left corner. In vertical text, the starting point is considered the upper middle point of the character. At the end of each character definition, the pen is removed (code 002) to the starting point of the next character. This letter spacing is Right in Horizontal Posts and Downwards in Vertical Posts.

The special code 00E (14) is used to define the start and end points of the character at once, depending on whether the type is Horizontal or Vertical. Thus, the need to define separate characters for Horizontal and Vertical texts is eliminated.

The following sample code and two figures **explain the definition of the start and end points for both Horizontal and Vertical writing for** the capital letter A.

```
*041,27,uca
2,14,8,(-8,-21),1,8,(8,21),8,(8,-21),
2,8,(-13,7),1,0A0,2,8,(9,-7),14,
8,(-14,-10),0
```





4. Using Shape in the Linetype Definition:

Open a blank file,

```
*1,25,OK002,9,(0,1),(0,0),001,9,(0,-2),(2,1),(-2,1),(0,-
2),(0,0),002,9,(2,1),(0,0),0
```

After writing the code, **let's save it with the file name** Ok.shp.

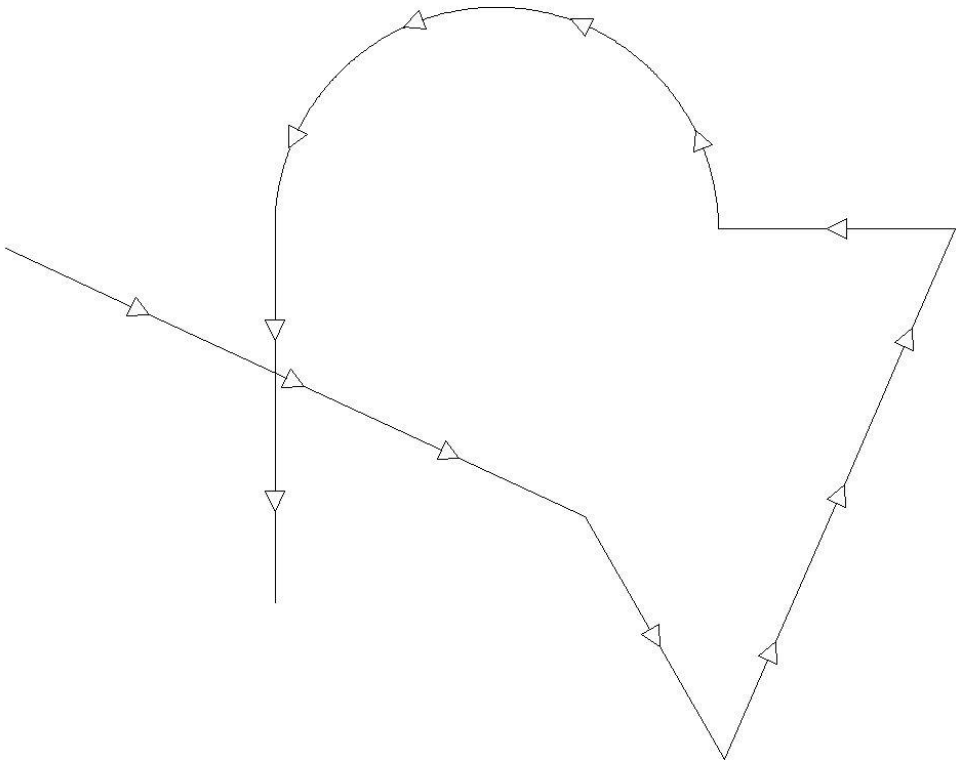
From the AutoCAD command line, **enter Compile<↵, find the Ok.shp** file you created and double-click on it. A compiled **file named Ok.shx** will be created.

We use the following 2 lines in the Linetype Definition (*. Lin extension) file, or by creating a new Linetype file.

```
*Arrow,Arrow Line --->--->---
>A,15,[OK,Ok.shx,s=1.0],-2
```

Our line type called "Line with Arrow" is ready to use.

Arrow sizes and spacing can be set from the LTSCALE or Object LTSCALE properties. The Shape Definition and Linetype Definition values given above can be changed to make drastic changes to the dimensions and ranges.



5. Font Definition Files

As mentioned in the previous sections, type Font definition files begin with a custom definition with the number 0 Shape, which gives the properties of the Font. This initial definition will be slightly different for UNIFONT files, as described in the next section. If Big Font is to be used, the Big Font Definition Line, which is explained in detail in the following sections, will be added to the beginning.

There are many fonts in AutoCAD. Fonts defined with the STYLE command can be written in expanded, collapsed or italicized letters, at the desired height, aligned to the desired point, in horizontal or vertical directions, according to the need, using the TEXT, MTEXT or ATTDEF commands.

In AutoCAD Text Font files, Shape numbers in Shape definitions are written in relation to the ASCII code number or Unicode number of each character. The first 256 Unicode numbers are the same as ASCII codes. Characters with ASCII codes from 1 to 31 are control characters. Only one of them is used in the Font definition file.

```
10 (LF)
```

The line skipping character defines the number of repeated Text commands by going down by the specified unit to the beginning of the next line.

```
*10,5,lf
2,8,(0,-10),0
```

As in the example, code 2 consists of a vector definition that goes down to the line spacing, leaving the active drawing state. Line spacing can be adjusted by changing the definition of character number 10.

As described above, the Font Definition file begins with the definition of Shape 0, which contains information about the font.

```
*0,4,font-name
up,down,mode,0
```

up: Determines how far the capital letters go up from the type starting line in terms of vector length.

Down: Determines how far down the character extensions that extend down the text line go down in terms of vector length.

When writing on a lined sheet of paper, you can think of the lines there as writing lines. With these up and down values, the character sizes and font height of the articles to be written will be defined. To put it in the simplest terms, the character definition will be divided by the value up, multiplied by the text height and the texts at the desired height will be written.

mode: If the font is defined to be used only in the Landscape orientation, its value should be 0, and if it is defined to be used in both the Horizontal and Vertical orientations, the value should be 2. When the Mode value is 2, the special codes 00E (14) described earlier will be processed.

0: Value that determines the end of the definition

Almost all standard fonts that come with AutoCAD have a few additional character definitions for dimensioning and other uses.

%%d Degree symbol (°)
%%p Plus/Minus Tolerance symbol (±)
%%c Circle diameter dimensioning symbol
These symbols and other desired characters can be used in Text by typing ASCII code as %%nnn.

AutoCAD uses characters in type according to ASCII codes (Shape number), not shape name. If the shape name is written in lowercase letters in character definitions, this information is not loaded into memory. For this reason, shape names in Font definitions are usually written as descriptive sections in lowercase letters.

6. Unicode Font Definitions

A single Unicode Font can be used to support a large number of character sets and many languages and platforms. Unicode font definitions have essentially the same format and features as general AutoCAD shape/font definition files, but with a few different features.

The main difference is the font description information written at the beginning.

```
*UNIFONT,6,font-name
up,down,mode,encoding,type,0
```

The font-name, up, down, and mode parameters are the same as regular fonts, as described earlier.

encoding: 0 Unicode
1 Packed multi-byte
2 Shape file

type: Font embedding information. Indicates whether the font is licensed. Licensed fonts cannot be modified.
0 Fonts embeddable
1 Font cannot be embedded
2 Recessed is read alone

0: Value that determines the end of the definition

Another important difference when writing Unicode fonts is that when using the special code 007 alt-shape. The data byte (shape number) following code 007 is taken as 2 bytes, even if it is written as a single byte, and should be calculated as such when writing the number of definition bytes at the beginning of the shape definition.

```
*122,4,bigS
7,83,0
```

Although 3 definition bytes are used in the example definition, the number of definition bytes should be written as 4, since the byte after 7 will be counted as 2.

In the Unicode font definition at the beginning of the file, if 0 is entered as the encoding value, that is, if it is specified that the unicode type will be encoding, the byte after 7 should be written with a length of 2 bytes, if the encoding value is 1 or 2,

the code can be written 1 byte after 7. When calculating the number of definition bytes in each case, the code will be calculated as 2 bytes after code 7.

Apart from these, hexadecimal numbers can be used as character numbers in Unicode font definitions. Although this is not a requirement, it will provide great convenience in the use of cross-references in the form of \U+.

7. Big Font Definition

The first line of Big Font files contains special code that defines how to read 2-byte hexadecimal definition bytes. After the Big Font Definition Line, one of the Normal Font or Unicode Font definition lines will be placed.

It is clear that Unifont files with hundreds, or even thousands, of character definitions will have a different definition and usage than 256-character ASCII codes.

We need to declare both the ASCII code system of 256 characters and how to read and recognize other character codes with numbers larger than these in the first line of the Big Font file.

The first line of the Big Font file;

*BIGFONT nchars,nranges,b1,e1,b2,e2,...is in the form.

where *nchars* is the approximate number of characters in the character set. If we define the number of characters greater than 10% of the current number, the font and drawing file size will increase. In this case, computer performance is unnecessarily affected. In the rest of the Big Font definition line, special character codes (escape codes) are defined. *How many escape codes to use with* nranges, *B1,E1,B2,E2,...* The start and end characters are defined for each range, which we will also use with .

*When using the BIGFONT definition line, character descriptions and other properties are the same as in other AutoCAD font files, except that 65535 characters can be defined.

8. Defining exponent and Index usages

Although there is no superscript and subscript character definition in AutoCAD SHX font files, except for a few characters, they can be easily defined.

The definition of exponential and indexed characters takes place in two steps. The first is to open exponential or indexed characters, and the second is to close them. For these on/off operations, 4 of them (2 for the exponent and 2 for the index) that are not used in the Character Code table can be used. Scale and vertical movement should be defined within the character definition determined in the opening process. In the closure process, on the other hand, scale and vertical movement will need to be restored. When choosing exponential and indexed text opening/closing characters, care should be taken not to change the character definitions used and to make them accessible from the numeric part of the keyboard in the form of Alt + karakter_kodu.

In the font definition to be given in the following sections, characters 082 (130) for indexing, 083 (131) for closing, 084 (132) for exponential text opening, and 084 (133) for closing are used.

Creating a Sample Font Definition File

To explain what has been explained so far in more detail on examples, let's examine the creation of a new font file in detail at each step.

True Type Fonts used in the AutoCAD environment significantly weigh up the drawing files and slow down the system in drawings with dense text, leading to very unpleasant situations. *.shx type fonts are used more quickly and quickly by AutoCAD.

Some unpleasant features in *.shx type fonts come with AutoCAD or can be found in sharing environments.

In the early days, characters defined with ASCII codes from 0 to 255 were used in the computer environment. Due to the differences between countries and languages, separate font files had to be created for each language. In this case, we were defining Turkish characters using empty or useless character numbers in the standard character table. When the keys on our keyboards were pressed, each key produced one of these 256 characters.

In our time, characters (letters) are defined in the computer environment with a system we call UNICODE. In the standard ASCII code system, 256 letters can be defined, while in the Unicode system, 65535 characters can be defined. In Unicode 6.2.0 released in September 2012, the Turkish Lira Symbol (₺) is defined as 020BA, i.e. 8378.

The keys on our computer keyboard now produce character code numbers suitable for the Unicode system. There are problems with old AutoCAD *.shx type fonts, especially Turkish characters, and it has become almost a necessity to use True Type Fonts that slow down the system noticeably.

In fonts commonly used in the AutoCAD environment, even the round parts of the letters are written in small line segments, resulting in unpleasant results, especially in large fonts and if the pen thickness is thin.

For these reasons, in the font we will write, the curved parts of the characters will be defined as arcs and circles, and Unicode encoding will be used to write all the Turkish characters we can use from the keyboard. There is no need for a Big Font definition.

The compiled version of the font definition file, which is given all the details and open codes here, can be downloaded and used from (FaLcon-TR.shx).

9. Unicode Table and Characters to Use

Dec.		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Hex.	0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E	0F
0000	00000																
16	00010															-	
32	00020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	00030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	00040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	He
80	00050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	00060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	he
112	00070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
128	00080	€	CTR	ib	soot	Ub	US				‰						□
144	00090						•				™						
160	000A0			¢	£		¥	¦	§		©		«			®	
176	000B0	°	±	²	³						¹	º	»	1/4	1/2	3/4	
192	000C0								Ç								
208	000D0							Ö	×	Ø				Ü			ß
224	000E0								Ç								
240	000F0							ö	÷	Ø				ü			
256	00100																
272	00110															Ğ	ğ
288	00120																
304	00130	İ	I														
320	00140																
336	00150															Ş	ş
8368	0B020											₺					

%%u: Underlined
%%o: strikethrough
%%d: ° (degree sign)
%%p: ± (plus-minus sign)
%129: peat mark (diameter of reinforced concrete reinforcement)
%130: Base (SuperScript) Startup
%131: Exponent End
%142: Index (subscript) start
%141: index end

As mentioned above, in the Unicode character table, the ₺ symbol is defined by the hexadecimal number 020BA. Your next generation computers will come with the ₺ symbol under the letter T like @ under the same Q on the keyboard and can be used with the AltGr+T key combination. A Windows update has already been released to use Unicode 6.2.0. AltGr+T can be used directly in the font file we have defined, or you can type the ₺ symbol by entering \U+B02A.

```
*UNIFONT,6,FaLcon-EN
66,24,2,0,0,0
```

We will create a UNIFONT type font file as described above.
Letter height: 66 units
The overhanging parts of the letters are 24 units.
Since we will define characters for both horizontal and vertical texts, we enter this value as 2.
Since we will define Unicode 0.
0 because the font can be embedded.
0 to define the end of definition bytes.
AutoCAD takes the font height in the drawing as the value found by dividing the vectors given in the *.shx file by the defined character height and multiplying the value found by the text height in the drawing.

```
*000A,13,[10-sell r_atlama]
2,14,8,(0,100),8,(0,-100),14,8,(-100,0),0
```

The ideal distance between the two lines is taken to be 100 units, which is slightly more than 1,414xletter height. 0A (Our line skipping character 10 defines the distance between columns in vertical fonts.

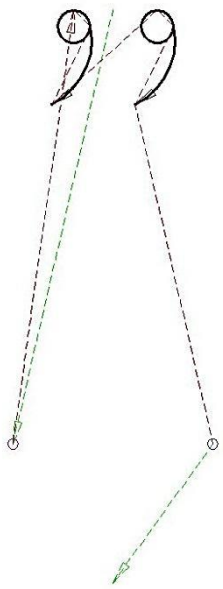
```
*00020,13,[32-space]
2,14,8,(-20,-66),8,(40,0),14,8,(-20,-25),0
```

The space character is defined as movement in a 40-unit horizontal without drawing a line.

```
*00021,29,[33-!]
2,14,8,(-7,-66), 8,(7,66),1,8,(0,-45),
2.8,(0,-15),1.00A,(3,020),
2,14,8,(-7,-25),8,(7,-6),0
```

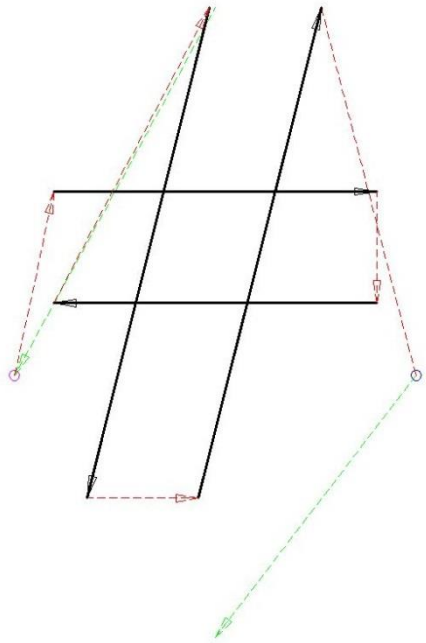
In figures, the vectors indicated by the red dashed line indicate the vectors that make up our character when the pen is removed (the drawing state is inactive), those indicated by the green and dashed lines are the vectors that are valid in the vertical state, and the black and thick lines indicate the vectors that make up our character when the pen is lowered (drawing mode is active). Our first character consists of a straight line in the form of 8,(x,y) and a fractional arc in the form of 00A,(r,0SC) (full circle because the octant number is given as 0, i.e. 8).

```
*00022,47,[34-"]
2,14,8,(-18,-78),8,(11,78),1,00A,
(3,020),2,8,(3,-3),1,12,(-7,-14,-43),
2.8,(19,17),1.00A,(3,020),2.8,
(3,-3),1,12,(-7,-14,-43),
2,14,8,(-18,-25),8,(14,-61),0
```



Our character 022 (34) is created using the special code 00C (12), which uses two full circles drawn with the code 00A, x and y displacements and the camber value.
It has been explained in the previous sections how to calculate the camber value with displacement vectors when using the special code 00C.
To do this process quickly and easily,
In the Autocad environment, the character was drawn in the correct dimensions and dimensions and the following AutoLISP function was used.

```
(defun c:bulg (/bLg DsT DtX DtY P1 P2 Pt1 Pt2 Pt3 Pt4)
  (setq Pt1 (getpoint "\nSpring Starting point: ")
        Pt2 (getpoint "\nArc Endpoint : ")
        Pt3 (getpoint "\nBombe Vertex : ")
        Pt4 (mapcar '(lambda (p1 p2) (/ (+ p1 p2) 2.0)) Pt1 Pt2)
        DtX (itoa (fix (- (car Pt2) (car Pt1))))
        DtY (itoa (fix (- (cadr Pt2) (cadr Pt1))))
        DsT (fix (distance Pt1 Pt2))
        bLg (itoa (fix (*127 (/ (* 2 (distance Pt3 Pt4)) DsT)))))
  (princ (strcat "\n" DtX "," DtY "," bLg)) (prin1))
```

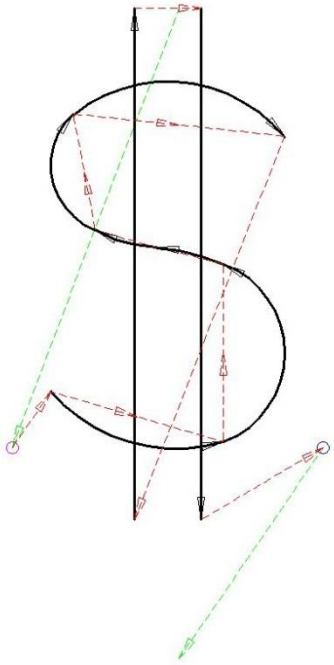


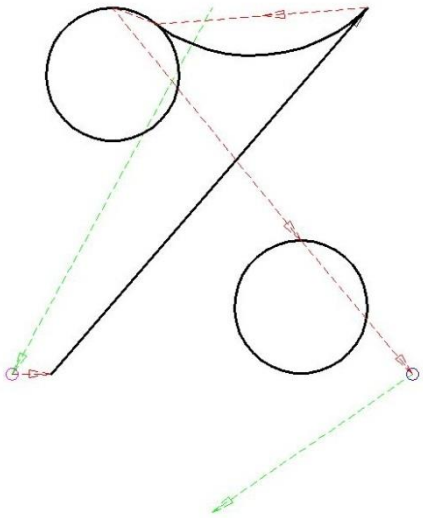
```
*00023,45,[35-#]
2,14,8,(-36,-66),8,(7,33),1,8,(58,0),
2,8,(0,-20),1,8,(-58,0),
2,8,(28,53),1,8,(-22,-88),
2,8,(20,0),1,8,(22,88),
2,14,8,(-34,-47),8,(17,-66),0
```

Only 8,(x,y) vector definitions are used in this character definition, which consists entirely of linear vectors. Since we are creating a font file, the shape name does not matter, so the character code number in decimal numbers and the character itself are written in this section.

```
*00024.55,[$36-]
2,14,8,(-30,-79),8,(7,10),
1,13,(31,-9,37),(0,32,87),(-12,3,11),
(-11,3,-15),(-4,21,-70),(38,-4,-52),(0,0),
2,8,(-27,-69),1,8,(0,92),
2,8,(12,0),1,8,(0,-92),
2,14,8,(-28,-38),8,(22,13),0
```

Successive spring parts are identified in a single group using the special code 00D (13). The series is terminated with a definition byte in the form of arcs (0,0).

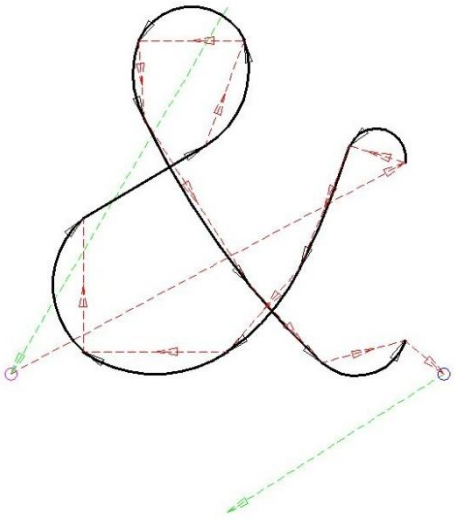




```
*00025,41,[37-%]
2,14,8,(-36,-66),8,(7,0),1,8,(57,66),
12,(-38,-3,-47),2,8,(-8,3),
1.00A,(12.020),
2.8,(34,-42),1.00A,(12,020),
2,14,8,(-36,-25),8,(20,-23),0
```

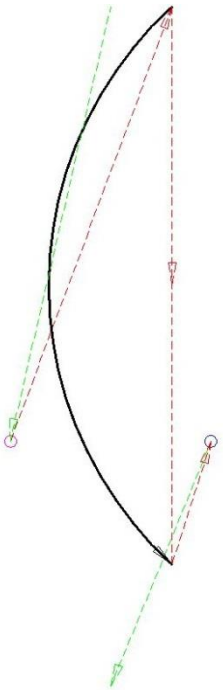
Combination of full circles with special code 00A (10) and spring part with special code 00C (12).

```
*00026,57,[38-&]
2,14,8,(-39,-6),8,(71,38),
1,13,(-10,3,112),(-9,-22,-5),
(-13,-15,-17),(-26,0,-39),
(0,24,-58),(22,13,0),(7,19,45),
(-19,0,80),(1,-14,27),(19,-30,7),
(13,-14,5),(15,4,68),(0,0),
2,14,8,(-39,-25),8,(7,-7),0
```



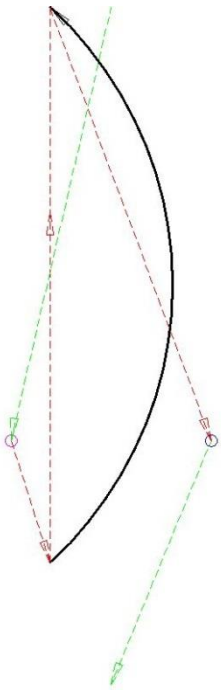
```
*00027,30,[39-']
2,14,8,(-11,-8),8,(11,78),
1.00A,(3,020),2,8,(3,-3),
1,12,(-7,-14,-43),2,14,8,(-10,-25),
8,(14,-61),0
```

Vertical Text vectors were defined with the special code 00E (14) by taking the first part of the previously defined character 022 (34).



```
*00028,22,[40-(]
2,14,8,(-18,-78),8,(29,78),1,12,
(0,-100,56),2,14,8,(-18,-47),8,(7,22),0
```

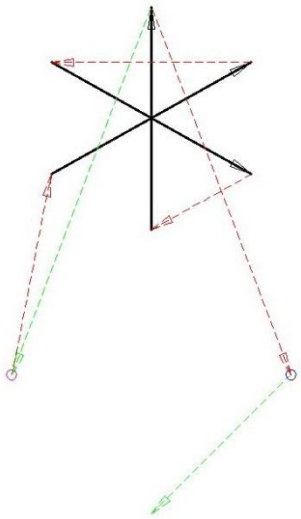
Character 028 (40) consists of a single arc designation.

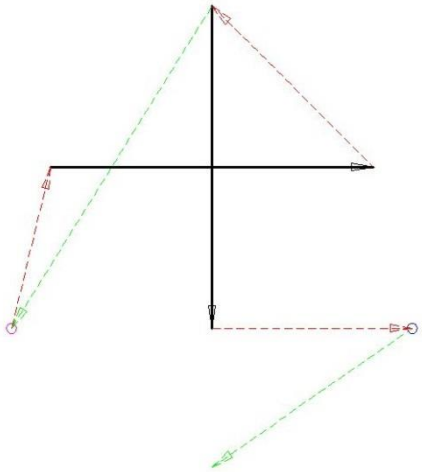


```
*00029,22,[41-)]
2,14,8,(-18,-78),8,(7,-22),1,12,
(0,100,56),2,14,8,(-18,-47),
8,(29,-78),0
```

Symmetry of the previous character.

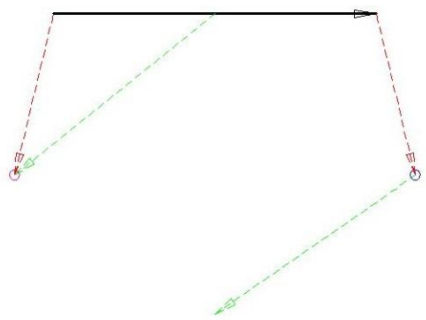
```
*0002A,37,[42-*]
2,14,8,(-25,-66),8,(7,36),1,8,(36,20),
2,8,(-36,0),1,8,(36,-20),2,8,(-18,30),
1,8,(0,-40),2,14,8,(-25,-25),8,(25,-26),0
```





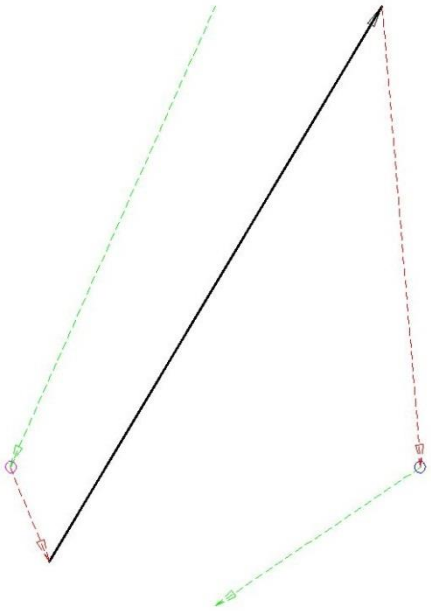
```
*0002B,29,[43-+]
2,14,8,(-36,-58),8,(7,29),1,8,(58,0),
2,8,(-29,29),1,8,(0,-58),
2,14,8,(-36,-25),8,(36,0),0
```

```
*0002C,30,[44-,]
2,14,8,(-11,-6),8,(11,6),1,00A,(3,020),
2,8,(3,-3),1,12,(-7,-14,-43),
2,14,8,(-10,-36),8,(14,11),0
```



```
*0002D,21,[45--]
2,14,8,(-36,-29),8,(7,29),1,8,(58,0),
2,14,8,(-36,-25),8,(7,-29),0
```

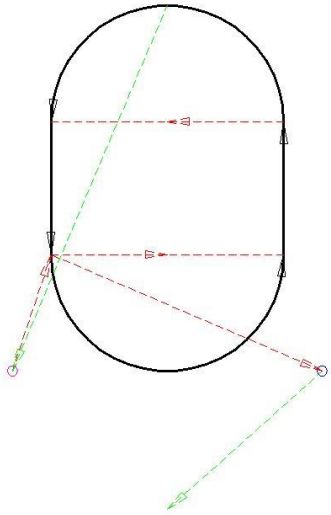
```
*0002E,21,[46-.]
2,14,8,(-10,6),8,(10,6),
1.00A,(3,020),2,14,8,(-10,-25),
8,(10,-6),0
```

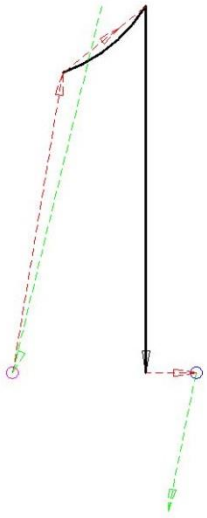


```
*0002F,21,[47-/]
2,14,8,(-37,-83),8,(7,-17),
1,8,(60,100),
2,14,8,(-37,-42),8,(7,-83),0
```

```
*00030,32,[48-0]
2,14,8,(-28,-66),8,(7,21),1,
00C,(42,0,127),8,(0,24),
00C,(-42,0,127),8,(0,-24),
2,14,8,(-28,-27),8,(49,-21),0
```

When defining the arc using the special codes 00C (12) or 00D (13), the Semicircle is drawn when the camber value is given as 127 (or -127).



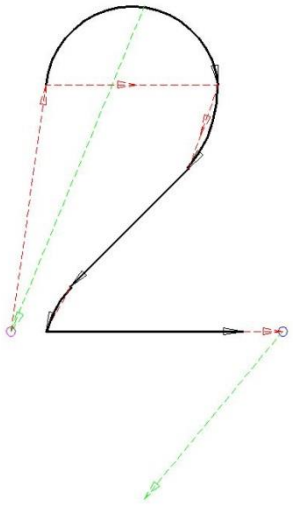


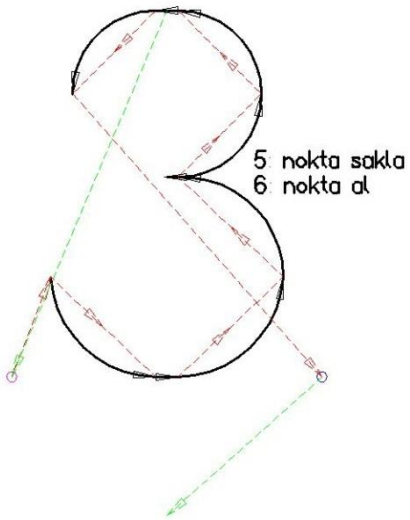
```
*00031,25,[49-1]
2,14,8,(-16,-66),8,(9,54),
1,12,(15,12,21),8,(0,-66),
2,14,8,(-17,-25),8,(9,0),0
```

In terms of aesthetics and readability, the entrance and exit of this character is larger than the others, (9 units instead of 7).

```
**00032,36,[50-2]
2,14,8,(-27,-66),8,(10,50),
1.00D,(35,0,-116),(-6,-17,-27),
(-26,-24,0),(-5,-9,17),(40,0,0),(0,0),
2,14,8,(-27,-25),8,(7,0),0
```

In the last segment of the continuous arcs written with the special code 00D (13), a straight line was defined by giving the bomb 0 with the expression (40,0,0) as we mentioned before, and the character was created by using fewer definition bytes.



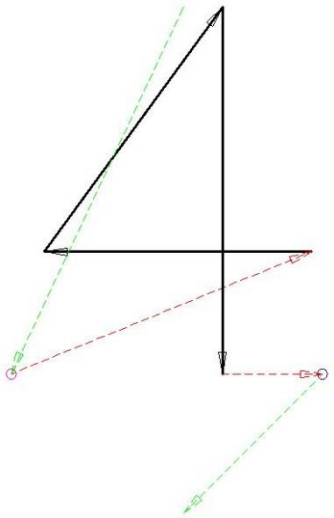


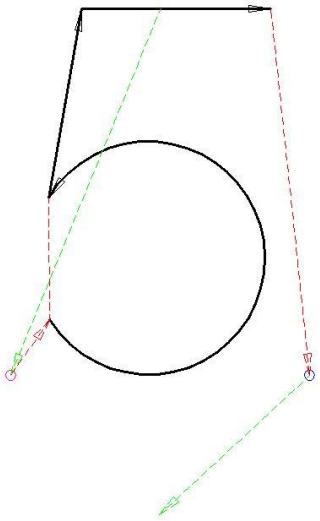

```
*00033,53,[51-3]
2,14,8,(-28,-66),8,(7,18),1,00D,
(19,-18,50),(4,0,0),(19,18,50),
(-19,18,54),(0,0),5,8,(-2,0),
6.00D,(15,15,53),(-15,15,53),
(-4,0,0),(-15,-15,53),(0,0),
2,14,8,(-28,-25),8,(46,-51),0
```

In addition to the zero-curved spring parts, the current point was stored with code 005 (5), then the process was continued by returning to the stored point with code 006 (6), thus using fewer bytes.

```
*00034,27,[52-4]
2,14,8,(-31,-66),8,(54,22),
1,9,(-48,0),(32,44),(0,-66),(0,0),
2,14,8,(-31,-25),8,(23,0),0
```

Instead of using a large number of code 008s, consecutive linear vectors were defined as a single series with the special code 009.

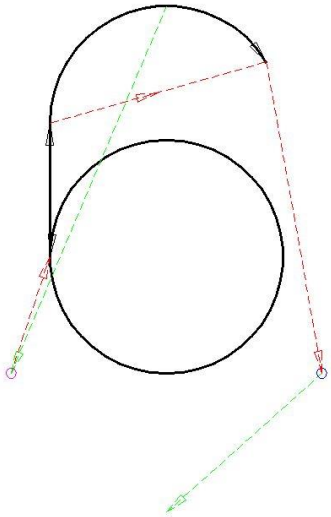


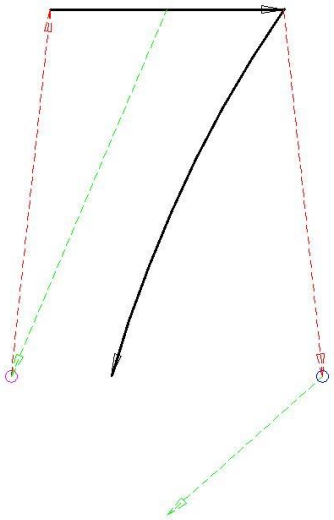


```
*00035,30,[53-5]
2,14,8,(-27,-66),8,(7,10),1,00B,
(182,79,0,21,040),8,(6,34),8,(34,0),
2,14,8,(-27,-25),8,(7,-66),0
```

Drawing of an arc with special code 00B (11).
Starting angle: 212°,
Starting octention: 4 (180°)
End angle: 149°, from start 8=0
Start_offset: ((212-180)*256)/45)=182
End_offset: ((149-135)*256)/45)=79
Radius: 21 units<256 =>high bit 0

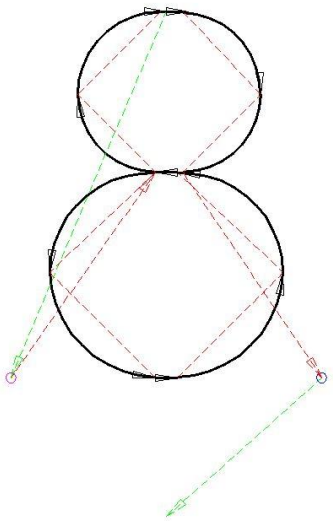
```
*00036,28,[54-6]
2,14,8,(-28,-66),8,(7,21),1.00A,
(21,040),8,(0,24),00C,(39,11,-97),
2,14,8,(-28,-25),8,(10,-56),0
```

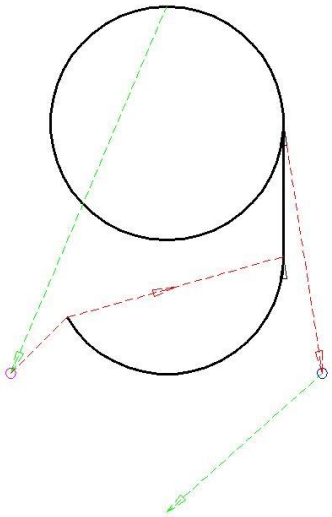




```
*00037,25,[55-7]
2,14,8,(-28,-66),8,(7,66),
1,8,(42,0),00C,(-31,-66,10),
2,14,8,(-28,-25),8,(38,0),0
```

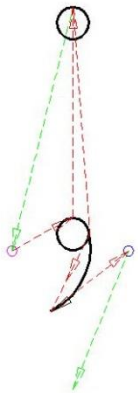
```
*00038,54,[56-8]
2,14,8,(-28,-66),8,(26,37),1,00D,
(-19,-18,54),(19,-19,50),(4,0,0),
(19,19,50),(-19,18,54),(-4,0,0),
(-14,14,-51),(15,15,-53),(4,0,0),
(14,-15,-53),(-14,-14,-51),(0,0),
2,14,8,(-28,-25),8,(25,-37),0
```





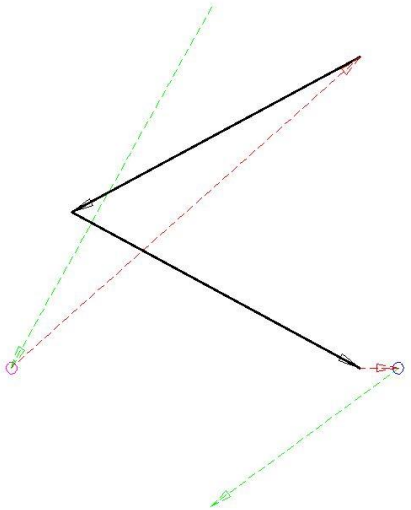
```
*00039,28,[57-9]
2,14,8,(-28,-66),8,(10,10),1,00C,
(39,11,97),8,(0,24),00A,(21,000),
2,14,8,(-28,-25),8,(7,-45),0
```

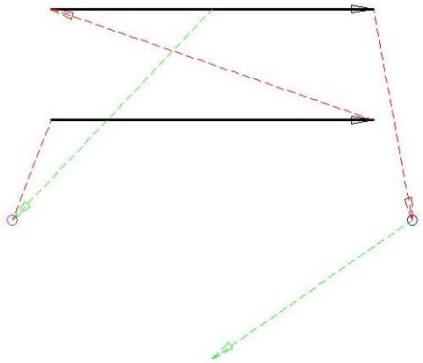
```
*0003A,29,[58-:]
2,14,8,(-10,-44),8,(10,6),1,00A,
(3.020),2.8,(0.38),1.00A,(3.020),
2,14,8,(-10,-25),8,(10,-44),0
```



```
*0003B,38,[59-;]
2,14,8,(-11,-44),8,(11,6),1.00A,
(3.020),2.8,(0.38),1.00A,(3.020),
2,8,(3,-41),1,12,(-7,-14,-43),
2,14,8,(-10,-25),8,(14,11),0
```

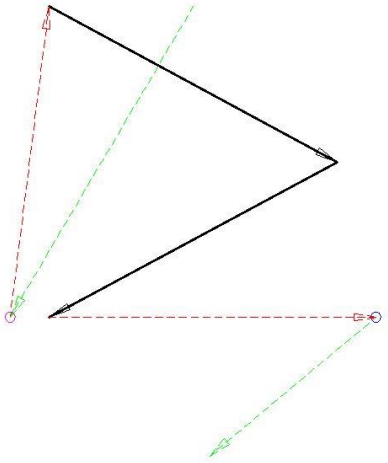
```
*0003C,24,[60-<]
2,14,8,(-33,-56),8,(63,56),
1,8,(-52,-28),8,(52,-28),
2,14,8,(-33,-25),8,(7,0),0
```

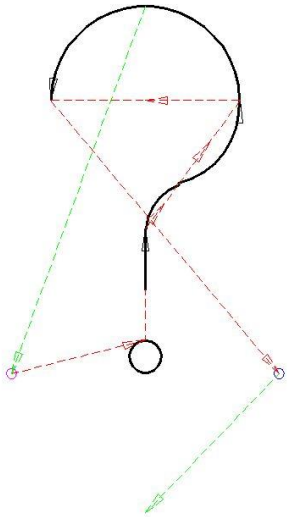




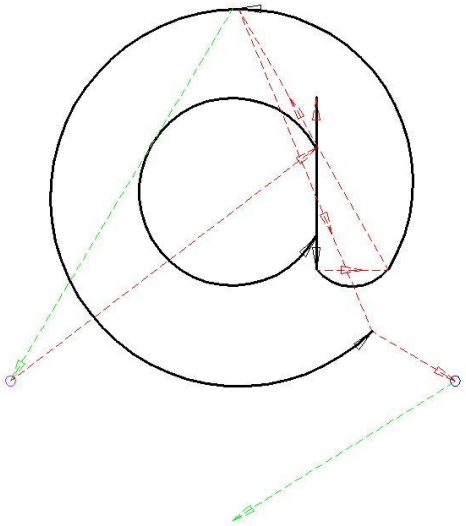
```
*0003D,29,[61-=]
2,14,8,(-36,-8),8,(7,18),
1,8,(58,0),2,8,(-58,20),1,8,(58,0),
2,14,8,(-36,-25),8,(7,-38),0
```

```
*0003E,24,[62->]
2,14,8,(-33,-56),8,(7,56),
1,8,(52,-28),8,(-52,-28),
2,14,8,(-33,-25),8,(59,0),0
```

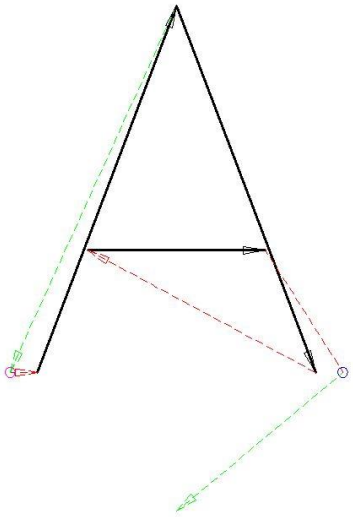




```
*0003F,41,[63-?]
2,14,8,(-24,-66),8,(24,6),1.00A,
(3,020),2,8,(0,9),1,8,(0,11),00D,
(6,8,-39),(11,15,46),(-34,0,127),
(0,0),2,14,8,(-24,-25),8,(41,-49),0
```



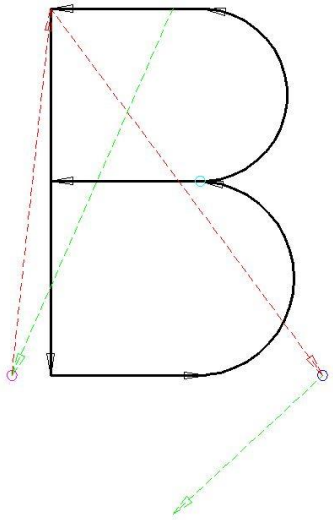
```
*00040,44,[64-@]
2,14,8,(-40,-67),8,(55,42),1,00B,
(159,96,0,17,000),2,8,(0,25),
1,8,(0,-31),13,(13,0,60),(-27,47,75),
(0,0),00A,(34,025),2,14,8,(-39,-26),
8,(15,-9),0
```

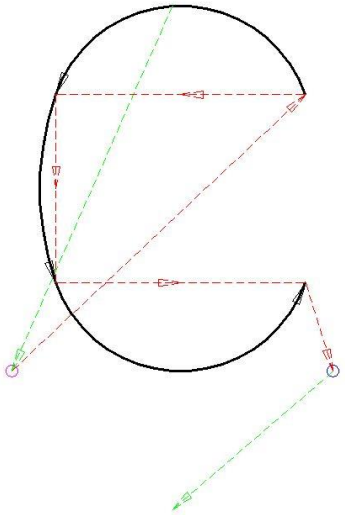


```
*00041,32,[65-A]
2,14,8,(-30,-66),8,(5,0),
1,8,(25,66),8,(25,-66),
2,8,(-41,22),1,8,(32,0),
2,14,8,(-30,-25),8,(14,-22),0
```

```
*00042,40,[66-B]
2,14,8,(-29,-66),8,(7,66),
1,8,(0,-66),8,(28,0),12,(0,35,115),
5,8,(-28,0),6,12,(0,31,120),8,(-28,0),
2,14,8,(-27,-25),8,(49,-66),0
```

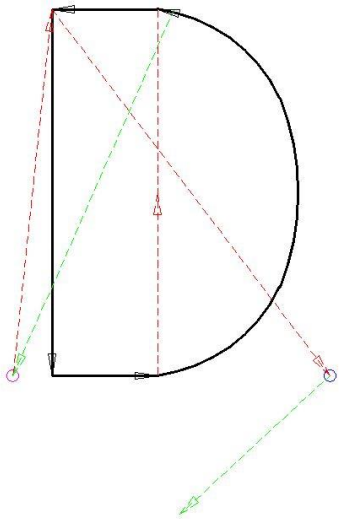
The location stored with code 5 was retrieved with code 6.

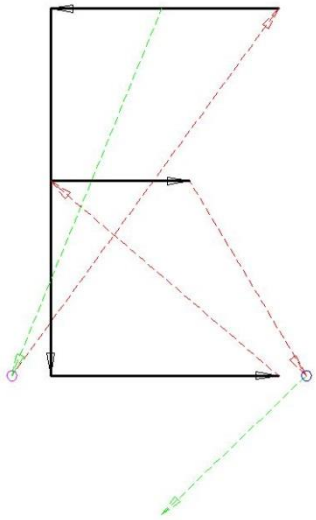




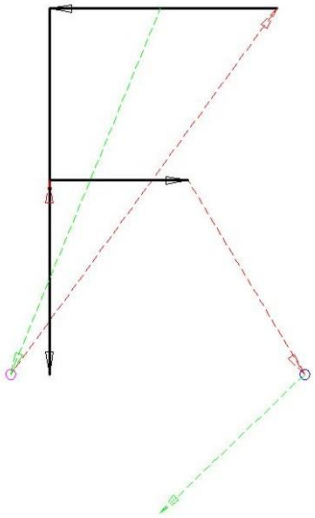
```
*00043,30,[67-C]
2,14,8,(-29,-66),8,(53,50),
1,13,(-45,0,90),(0,-34,22),
(45,0,90),(0,0),
2,14,8,(-29,-25),8,(5,-16),0
```

```
*00044,39,[68-D]
2,14,8,(-29,-66),8,(7,66),
1,8,(0,-66),8,(19,0),13,(22,16,33),
(0,34,24),(-22,16,33),(0,0),
8,(-19,0),2,14,8,(-27,-25),8,(50,-66),0
```

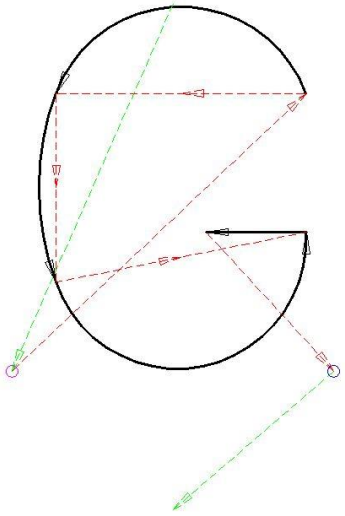




```
*00045,35,[69-E]
2,14,8,(-27,-66),8,(48,66),
1,9,(-41,0),(0,-66),(41,0),
(0,0),2,8,(-41,35),1,8,(25,0),
2,14,8,(-26,-25),8,(21,-35),0
```

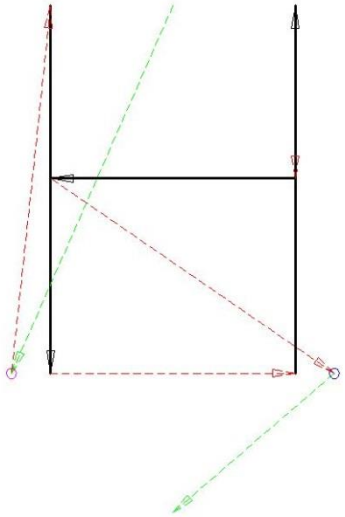


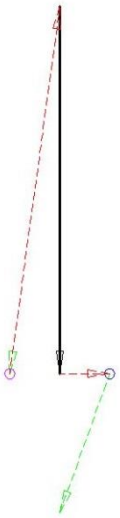
```
*00046,32,[70-F]
2,14,8,(-27,-66),8,(48,66),
1,8,(-41,0),8,(0,-66),
2,8,(0,35),1,8,(25,0),
2,14,8,(-26,-25),8,(21,-35),0
```



```
*00047,33,[71-G]
2,14,8,(-29,-66),8,(53,50),
1,13,(-45,0,88),(0,-34,23),
(45,9,111),(-18,0,0),(0,0),
2,14,8,(-29,-25),8,(23,-25),0
```

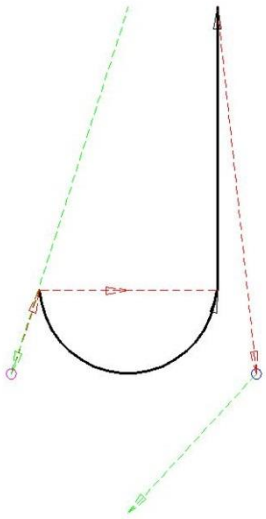
```
*00048,37,[72-H]
2,14,8,(-29,-66),8,(7,66),
1,8,(0,-66),2,8,(44,0),
1,8,(0,66),2,8,(0,-31),1,8,(-44,0),
2,14,8,(-29,-25),8,(51,-35),0
```

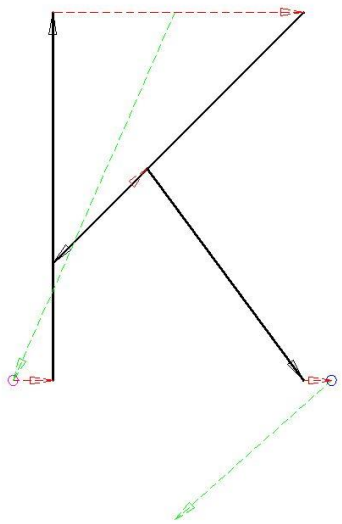




```
*00049,21,[73-I]
2,14,8,(-9,-66),8,(9,66),
1,8,(0,-66),2,14,8,(-9,-25),8,(9,0),0
```

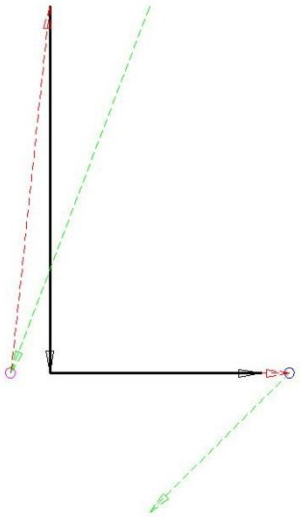
```
*0004A,25,[74-J]
2,14,8,(-21,-66),8,(5,15),
1,12,(32,0,119),8,(0,51),
2,14,8,(-23,-25),8,(7,-66),0
```

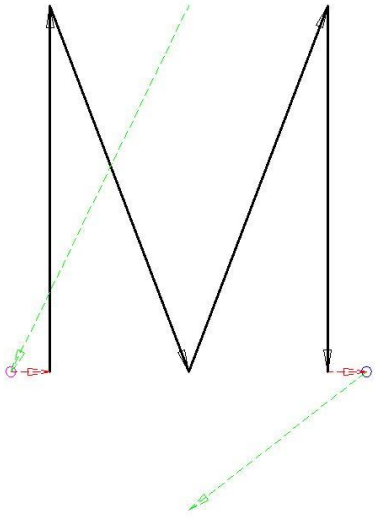




```
*0004B,37,[75-K]
2,14,8,(-29,-66),8,(7,0),1,8,(0,66),
2,8,(45,0),1,8,(-45,-45),
2,8,(17,17),1,8,(28,-38),
2,14,8,(-28,-25),8,(5,0),0
```

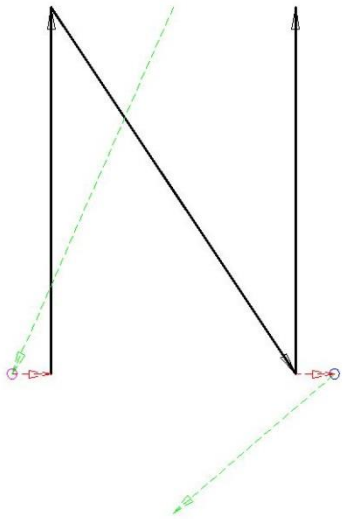
```
*0004C,24,[76-L]
2,14,8,(-25,-66),8,(7,66),
1,8,(0,-66),8,(38,0),
2,14,8,(-25,-25),8,(3,0),0
```

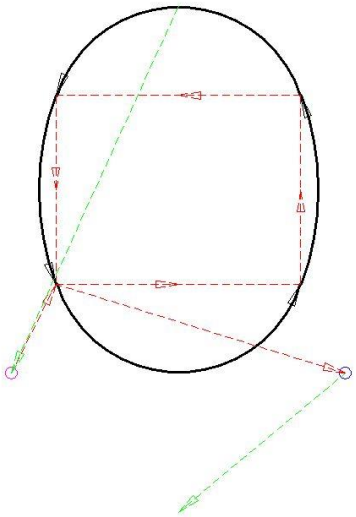




```
*0004D,29,[77-M]
2,14,8,(-32,-66),8,(7,0),1,9,
(0,66),(25,-66),(25,66),(0,-66),
(0,0),2,14,8,(-32,-25),8,(7,0),0
```

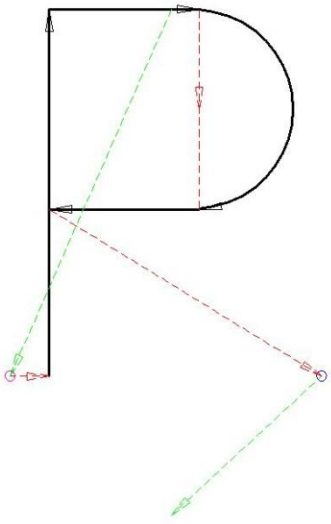
```
*0004E,27,[78-N]
2,14,8,(-29,-66),8,(7,0),1,9,
(0,66),(44,-66),(0,66),(0,0),
2,14,8,(-29,-25),8,(7,-66),0
```

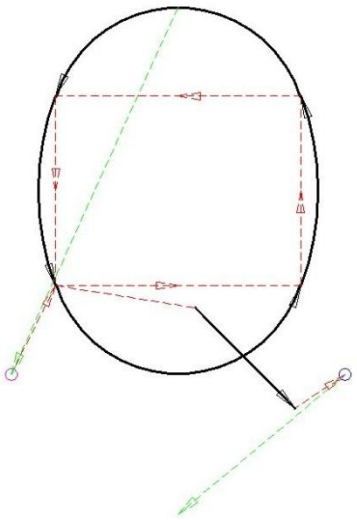




```
*0004F,33,[79-O]
2,14,8,(-30,-66),8,(8,16),1,13,(44,0,92),
(0,34,23),(-44,0,92),(0,-34,23),(0,0),
2,14,8,(-30,-25),8,(52,-16),0
```

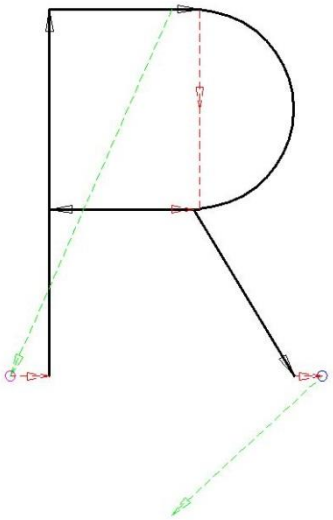
```
*00050,31,[80-P]
2,14,8,(-29,-66),8,(7,0),1,8,(0,66),
8,(27,0),12,(0,-36,-120),8,(-27,0),
2,14,8,(-27,-25),8,(49,-30),0
```

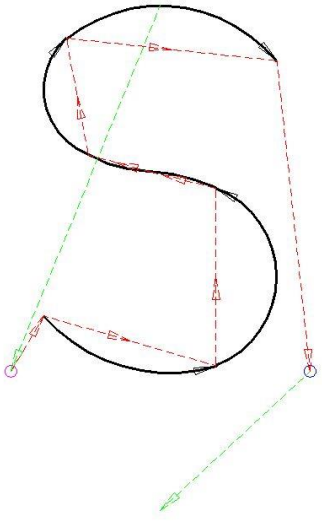




```
*00051,41,[81-Q]
2,14,8,(-30,-66),8,(8,16),1,13,
(44,0,92),(0,34,23),(-44,0,92),
(0,-34,23),(0,0),2,8,(25,-4),1,8,
(18,-18),2,14,8,(-30,-25),8,(9,6),0
```

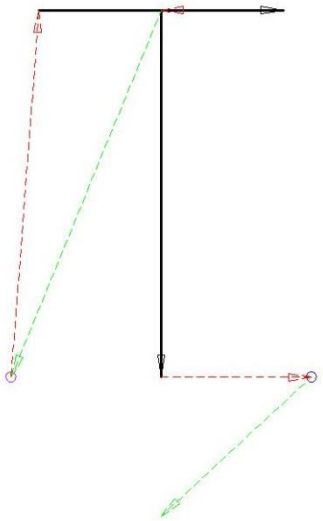
```
*00052,39,[82-R]
2,14,8,(-29,-66),8,(7,0),1,8,
(0,66),8,(27,0),12,(0,-36,-120),
8,(-27,0),2,8,(26,0),1,8,(18,-30),
2,14,8,(-27,-25),8,(5,0),0
```

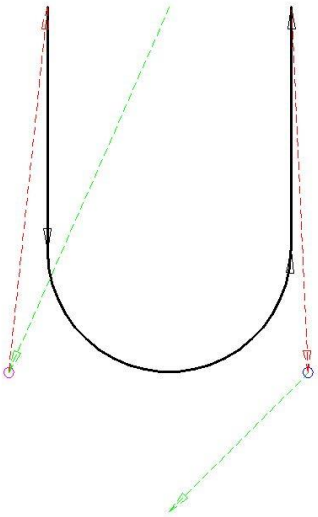




```
*00053,39,[83-S]
2,14,8,(-27,-66),8,(6,10),
1,13,(31,-9,37),(0,32,87),(-12,3,11),
(-11,3,-15),(-4,21,-70),(38,-4,-52),
(0,0),2,14,8,(-27,-25),8,(6,-56),0
```

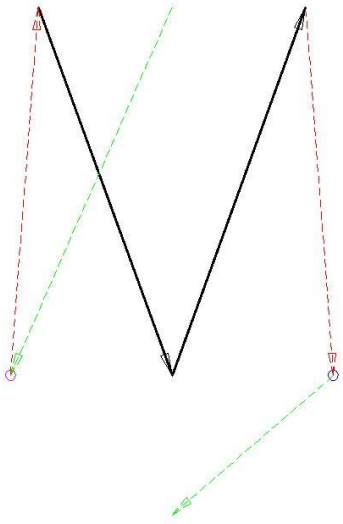
```
*00054,29,[84-T]
2,14,8,(-27,-66),8,(5,66),
1,8,(44,0),2,8,(-22,0),1,8,(0,-66),
2,14,8,(-27,-25),8,(27,0),0
```

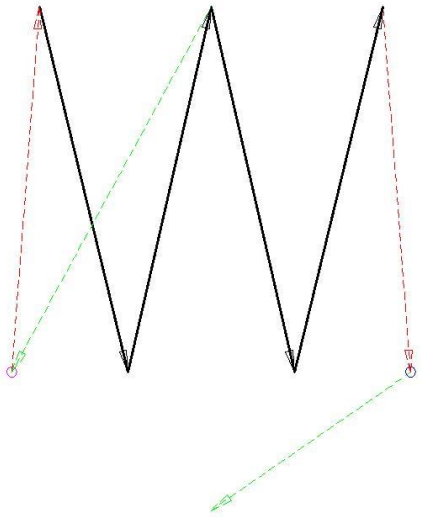




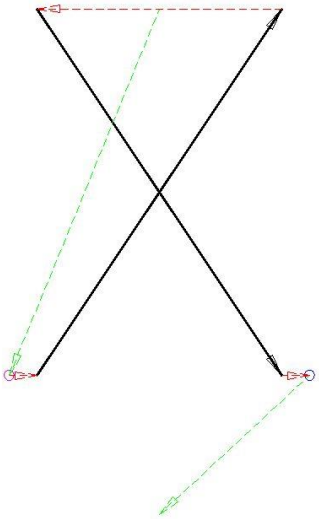
```
*00055,28,[85-U]
2,14,8,(-29,-66),8,(7,66),
1,8,(0,-44),12,(44,0,127),8,(0,44),
2,14,8,(-29,-25),8,(7,-66),0
```

```
*00056,24,[86-V]
2,14,8,(-29,-66),8,(5,66),
1,8,(24,-66),8,(24,66),
2,14,8,(-29,-25),8,(5,-66),0
```

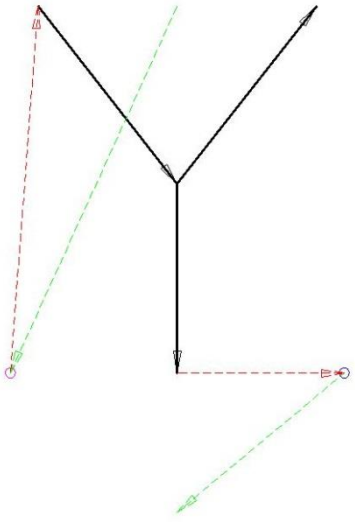




```
*00057,30,[87-W]
2,14,8,(-36,-66),8,(5,66),1,8,
(16,-66),8,(15,66),8,(15,-66),
8,(16,66),2,14,8,(-36,-25),
8,(5,-66),0
```

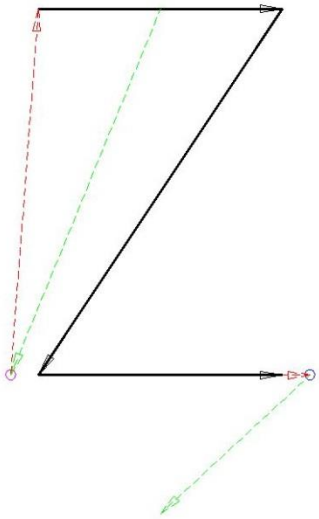


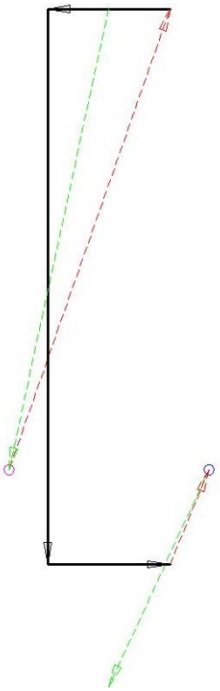
```
*00058,29,[88-X]
2,14,8,(-27,-66),8,(5,0),1,8,
(44,66),2,8,(-44,0),1,8,(44,-66),
2,14,8,(-27,-25),8,(5,0),0
```



```
*00059,29,[89-Y]
2,14,8,(-30,-66),8,(5,66),1,8,
(25,-32),5,8,(25,32),6,8,(0,-34),
2,14,8,(-30,-25),8,(30,0),0
```

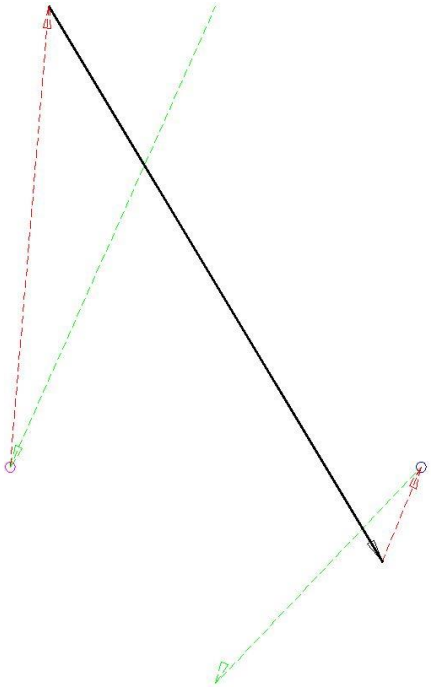
```
*0005A,27),[90-Z]
2,14,8,(-27,-66),8,(5,66),
1,9,(44,0),(-44,-66),(44,0),(0,0),
2,14,8,(-27,-25),8,(5,0),0
```

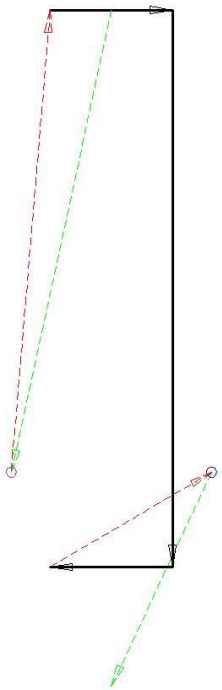




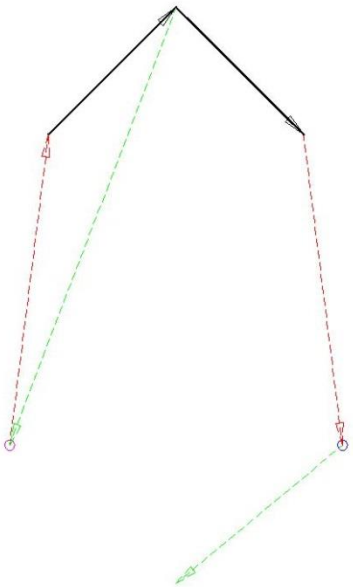
```
*0005B,27,[91-[]
2,14,8,(-18,-83),8,(29,83),
1,9,(-22,0),(0,-100),(22,0),(0,0),
2,14,8,(-18,-42),8,(7,17),0
```

```
*0005C,21,[92-\]
2,14,8,(-37,-83),8,(7,83),1,8,(60,-100),
2,14,8,(-37,-42),8,(7,17),0
```

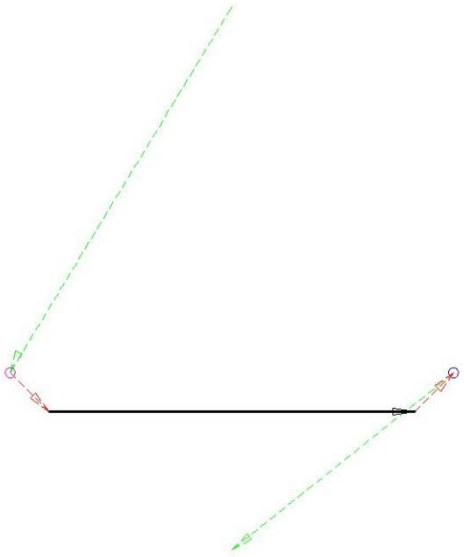




```
*0005D,27,[93-]]
2,14,8,(-18,-83),8,(7,83),
1,9,(22,0),(0,-100),(-22,0),(0,0),
2,14,8,(-18,-42),8,(29,17),0
```

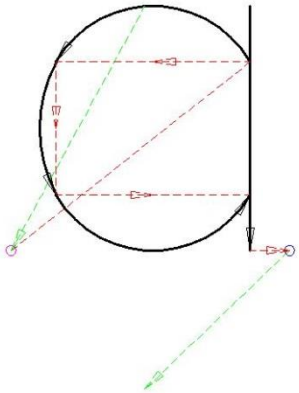


```
*0005E,24,[94-^]
2,14,8,(-30,-79),8,(7,56),
1,8,(23,23),8,(23,-23),
2,14,8,(-30,-25),8,(7,-56),0
```



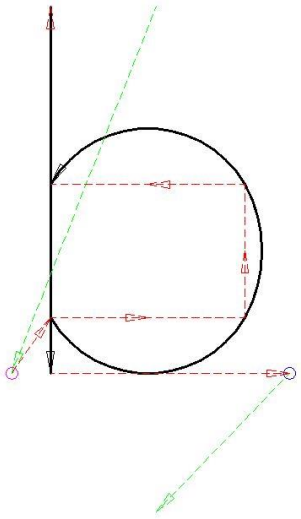
```
*0005F,21,[95-_]
2,14,8,(-40,-66),8,(7,-7),1,8,(66,0),
2,14,8,(-40,-32),8,(7,7),0
```

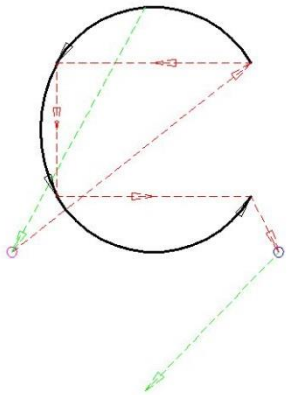
```
*00060,30,[96-']
2,14,8,(-10,-78),8,(10,78),1,00A,
(3,020),2,8,(-3,-3),1,12,(7,-14,43),
2,14,8,(-11,-25),8,(7,-61),0
```



```
*00061,38,[97-A]
2,14,8,(-24,-44),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(0,34),1,8,(0,-44),
2,14,8,(-26,-25),8,(7,0),0
```

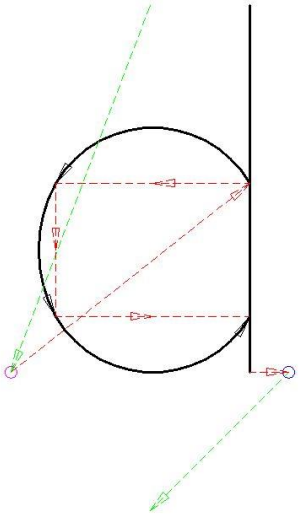
```
*00062,38,[98-b]
2,14,8,(-26,-66),8,(7,10),1,13,
(35,0,73),(0,24,32),(-35,0,73),
(0,0),2,8,(0,32),1,8,(0,-66),
2,14,8,(-24,-25),8,(43,0),0
```

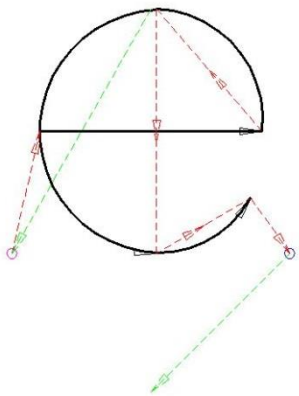




```
*00063,30,[99-c]
2,14,8,(-24,-44),8,(43,34),
1,13,(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,14,8,(-24,-25),8,(5,-10),0
```

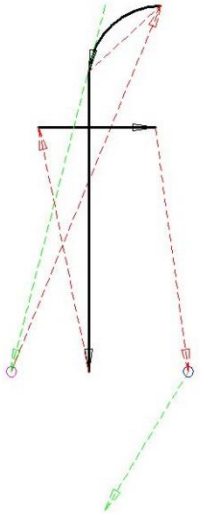
```
*00064,38,[100-d]
2,14,8,(-25,-66),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(0,56),1,8,(0,-66),
2,14,8,(-25,-25),8,(7,0),0
```

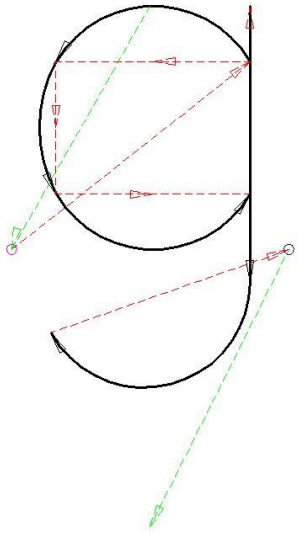




```
*00065,33,[101-e]
2,14,8,(-25,-44),8,(5,22),1,00D,
(40,0,0),(-19,22,55),(0,-44,122),
(17,10,37),(0,0),
2,14,8,(-25,-25),8,(7,-10),0
```

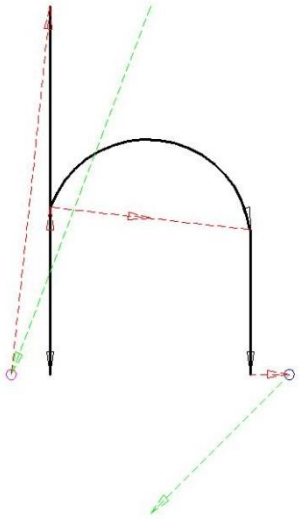
```
*00066,33,[102-f]
2,14,8,(-17,-66),8,(28,66),
1,12,(-13,-12,50),8,(0,-54),
2,8,(-9,44),1,8,(21,0),
2,14,8,(-16,-25),8,(6,-44),0
```

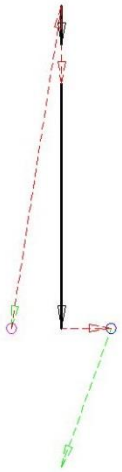




```
*00067,42,[103-g]
2,14,8,(-25,-52),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(0,34),1,8,(0,-50),00C,
(-36,-9,-98),2,14,8,(-25,-50),
8,(43,15),0
```

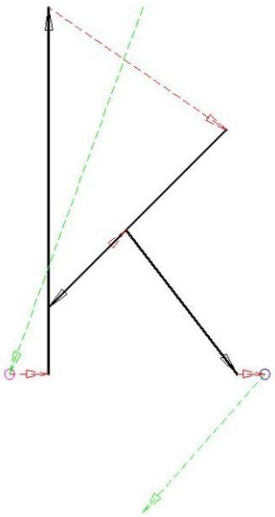
```
*00068,33,[104-h]
2,14,8,(-25,-66),8,(7,66),1,8,
(0,-66),2,8,(0,30),1,12,
(36,-4,-100),8,(0,-26),
2,14,8,(-25,-25),8,(7,0),0
```





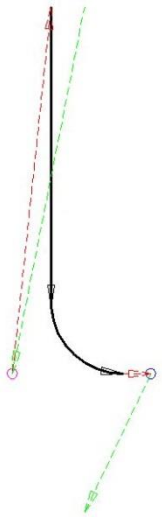
```
*00069,29,[105-i]
2,14,8,(-9,-58),8,(9,58),
1,8,(0,-7),2,8,(0,-7),1,8,(0,-44),
2,14,8,(-9,-25),8,(9,0),0
```

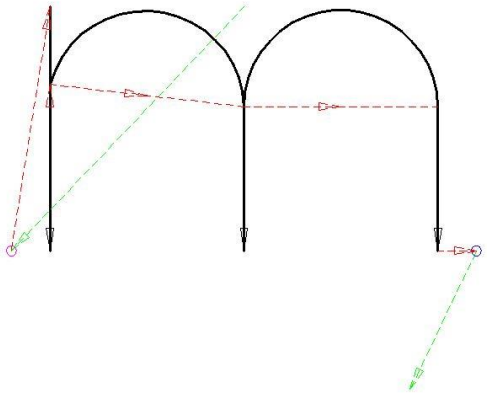
```
*0006A,33,[106-j]
2,14,8,(-11,-58),8,(5,-20),
1,12,(13,12,50),8,(0,52),
2,8,(0,7),1,8,(0,7),
2,14,8,(-14,-50),8,(7,-58),0
```



```
*0006B,37,[107-k]
2,14,8,(-24,-66),8,(7,0),1,8,
(0,66),2,8,(32,-22),1,8,(-32,-32),
2,8,(14,14),1,8,(20,-26),
2,14,8,(-22,-25),8,(5,0),0
```

```
2,14,8,(-13,-66),8,(7,66),
1,8,(0,-54),12,(13,-12,50),
2,14,8,(-12,-25),8,(5,0),0
```

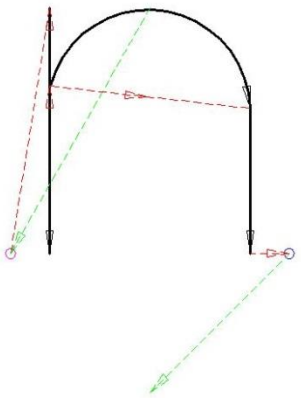


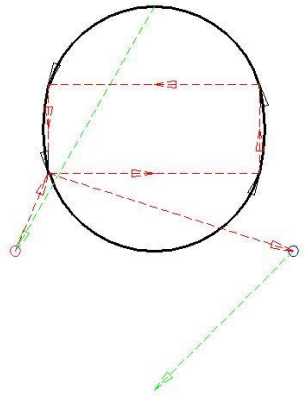


```
*0006D,42,[109-m]
2,14,8,(-42,-44),8,(7,44),1,8,
(0,-44),2,8,(0,30),1,12,(35,-4,-110),
5,8,(0,-26),6,12,(35,0,-127),
8,(0,-26),2,14,8,(-42,-25),8,(7,0),0
```

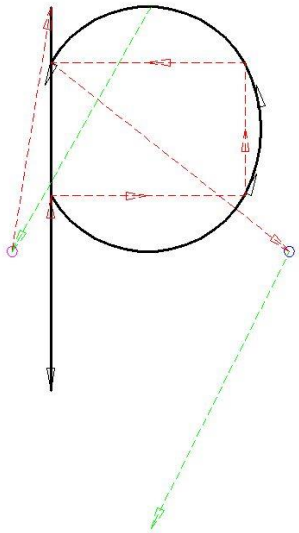
A location was stored using code 005, and the stored location was returned using code 006.

```
*0006E,33,[110-n]
2,14,8,(-25,-44),8,(7,44),
1,8,(0,-44),2,8,(0,30),1,12,
(36,-4,-110),8,(0,-26),
2,14,8,(-25,-25),8,(7,0),0
```

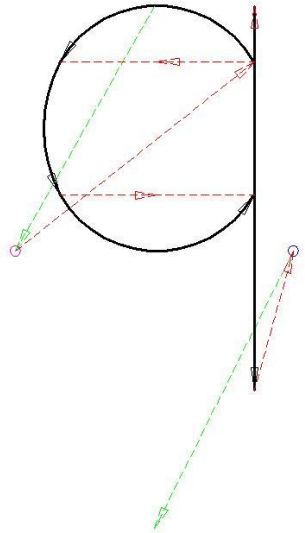




```
*0006F,33,[111-O]
2,14,8,(-25,-44),8,(6,14),1,13,
(38,0,94),(0,16,16),(-38,0,94),
(0,-16,16),(0,0),2,14,8,(-25,-25),
8,(44,-14),0
```

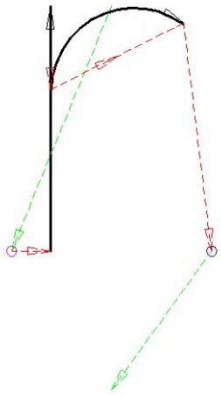


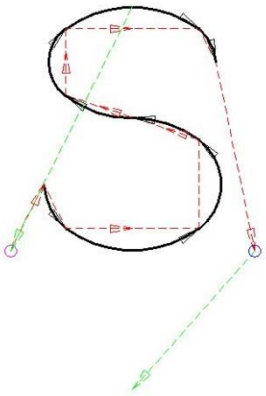
```
*00070,37,[112-p]
2,14,8,(-25,-44),8,(7,44),1,8,
(0,-69),2,8,(0,35),13,(35,0,73),
(0,24,32),(-35,0,73),(0,0),
2,14,8,(-25,-50),8,(43,-34),0
```



```
*00071,38,[113-q]
2,14,8,(-24,-44),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(0,34),1,8,(0,-69),
2,14,8,(-26,-50),8,(7,25),0
```

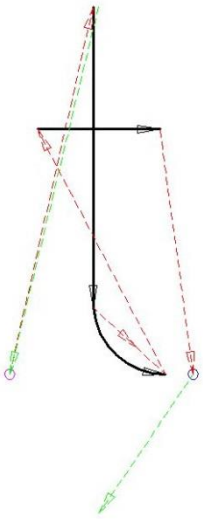
```
*00072,29,[114-r]
2,14,8,(-18,-44),8,(7,0),1,8,
(0,44),2,8,(0,-15),12,(24,12,-78),
2,14,8,(-18,-25),8,(5,-41),0
```

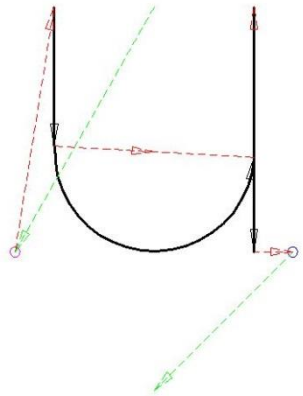




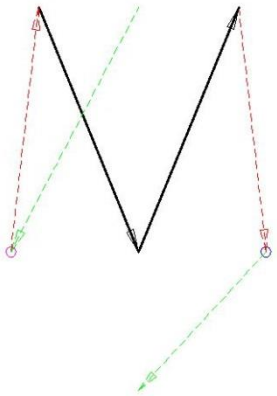
```
*00073,45,[115-s]
2,14,8,(-22,-44),8,(6,12),1,13,
(4,-8,25),(24,0,42),(0,16,63),
(-12,4,20),(-12,4,-23),(0,12,-63),
(24,0,-42),(3,-6,-25),(0,0),
2,14,8,(-22,-25),8,(7,-34),0
```

```
*00074,33,[116-t]
2,14,8,(-16,-66),8,(15,66),1,8,(0,-54),
12,(13,-12,50),2,8,(-23,44),1,8,(22,0),
2,14,8,(-17,-25),8,(6,-44),0
```

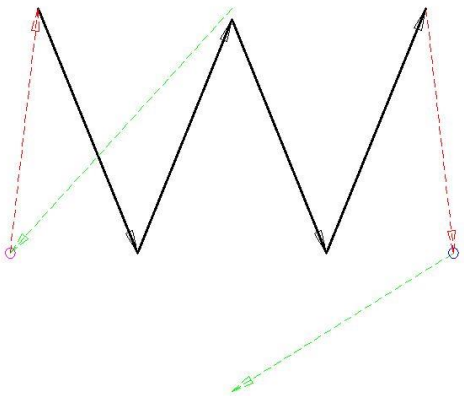




```
*00075,33,[117-u]
2,14,8,(-25,-44),8,(7,44),
1,8,(0,-25),12,(36,-2,127),
2,8,(0,27),1,8,(0,-44),
2,14,8,(-25,-25),8,(7,0),0
```

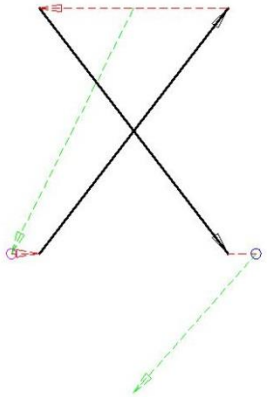


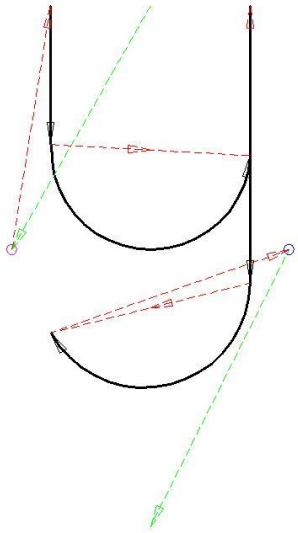
```
*00076,25,[118-V]
2,14,8,(-23,-44),8,(5,44),
1,9,(18,-44),(18,44),(0,0),
2,14,8,(-23,-25),8,(5,-44),0
```



```
*00077,29,[119-w]
2,14,8,(-40,-44),8,(5,44),1,9,
(18,-44),(17,42),(17,-42),(18,44),
(0,0),2,14,8,(-40,-25),8,(5,-44),0
```

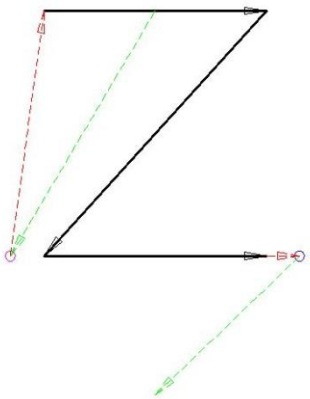
```
*00078,29,[120-x]
2,14,8,(-22,-44),8,(5,0),1,8,
(34,44),2,8,(-34,0),1,8,(34,-44),
2,14,8,(-22,-25),8,(5,0),0
```

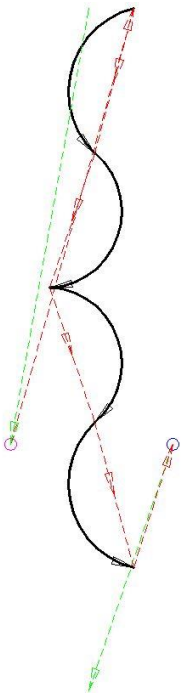




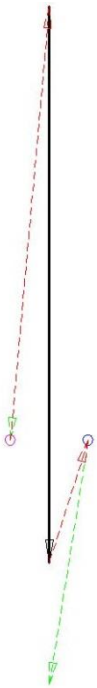
```
*00079,37,[121-y]
2,14,8,(-25,-44),8,(7,44),1,8,
(0,-25),12,(36,-2,127),2,8,(0,27),
1,8,(0,-50),00C,(-36,-9,-98),
2,14,8,(-25,-50),8,(43,15),0
```

```
*0007A,27,[122-Z]
2,14,8,(-26,-44),8,(6,44),
1,9,(40,0),(-40,-44),(40,0),(0,0),
2,14,8,(-26,-25),8,(6,0),0
```

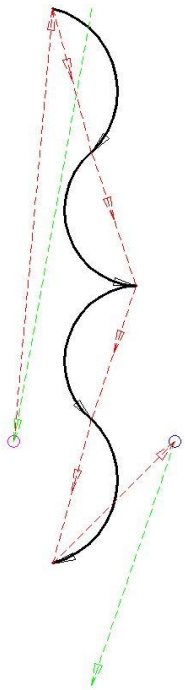




```
*0007B,33,[123-{]
2,14,8,(-14,-78),8,(22,78),1,13,
(-7,-26,75),(-8,-24,-86),(8,-24,-86),
(7,-26,75),(0,0),2,14,8,(-15,-47),
8,(7,22),0
```

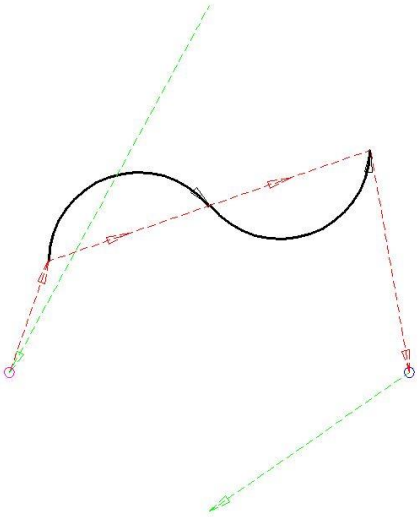


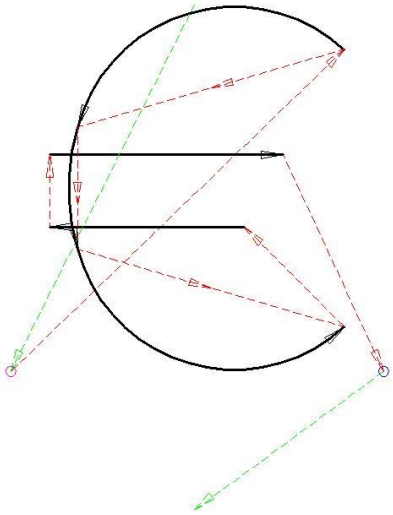
```
*0007C,21,[124-|]
2,14,8,(-7,-78),8,(7,78),1,8,(0,-100),
2,14,8,(-7,-47),8,(7,22),0
```



```
*0007D,33,[125-}]
2,14,8,(-14,-78),8,(7,78),1,13,
(7,-26,-75),(8,-24,86),(-8,-24,86),
(-7,-26,-75),(0,0),2,14,8,(-15,-47),
8,(22,22),0
```

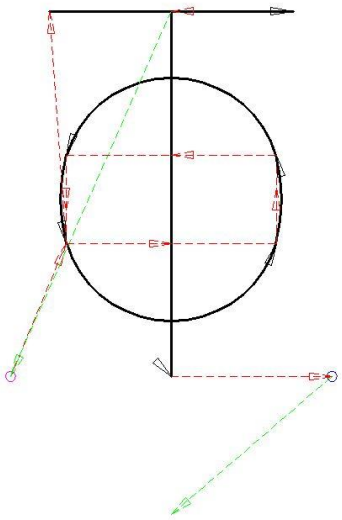
```
*0007E,26,[126-~]
2,14,8,(-36,-66),8,(7,20),
1,12,(29,10,-90),12,(29,10,90),
2,14,8,(-36,-25),8,(7,-40),0
```





```
*00080,46,[128-€]
2,14,8,(-33,-66),8,(60,58),1,13,
(-48,-14,72),(0,-22,17),(48,-14,72),
(0,0),2,8,(-18,18),1,8,(-35,0),2,8,
(0,13),1,8,(42,0),2,14,8,(-34,-25),
8,(18,-39),0
```

```
*00081,49,[129-tø]
2,14,8,(-29,-66),8,(10,24),1,13,
(38,0,94),(0,16,16),(-38,0,94),
(0,-16,16),(0,0),2,8,(-3,42),
1,8,(44,0),2,8,(-22,0),1,8,(0,-66),
2,14,8,(-29,-25),8,(29,0),0
```



```
*00082,10,[130-super_on ]
2,004,7,003,10,8,(0,60),1,0
```

No letters are written with the character number 082 (130). Letter entries are shifted up, vector lengths multiplied by 7 using code 004, vector lengths divided by 10 using code 003. In this way, the font heights are made 7/10 = 0.7 and the exponential text (super-script) situation is started.

```
*00083,10,[131-super_off ]
2,8,(0,-60),004,10,003,7,1,0
```

After the letter entries are restored to their normal level, the vector length is multiplied by 10 by Code 4, divided by 7 using Code 003 to restore the font height. That is, the exponential post case is terminated.

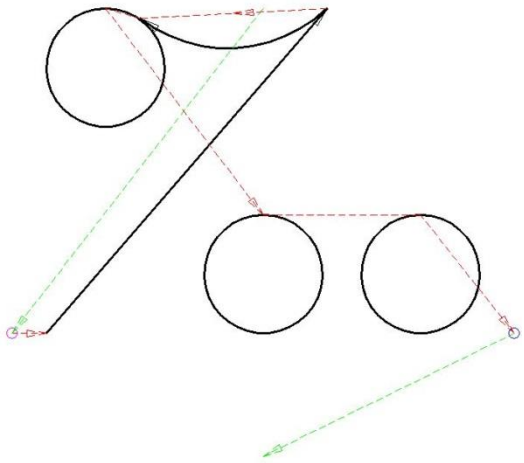
```
*00084,10,[132-sub_on]
2,004,7,003,10,8,(0,-20),1,0
```

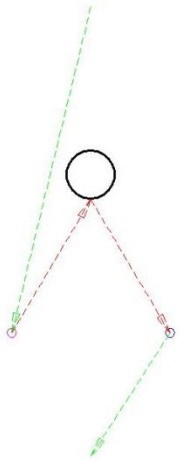
As in the case of exponential writing, operations are applied and the sub-script state is passed.

```
*00085,10,[133-sub_off ]
2,8,(0,20),004,10,003,7,1,0
```

The indexed text status is terminated.

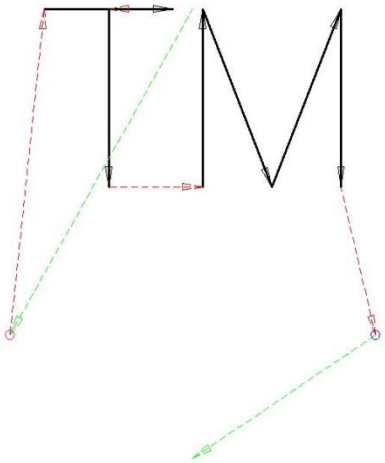
```
*00089,49,[137-‰]
2,14,8,(-51,-66),8,(7,0),1,8,
(57,66),12,(-38,-2,-47),2,8,(-7,2),
1.00A,(12,020),2.8,(32,-42),1.00A,
(12,020),2,8,(32,0),1.00A,(12,020),
2,14,8,(-51,-25),8,(19,-24),0
```

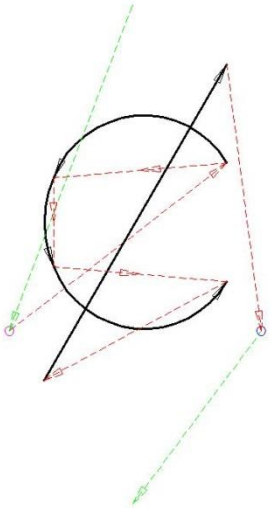




```
*00095,21,[149-•]
2,14,8,(-16,-66),8,(16,27),1.00A,
(5,060),2,14,8,(-16,-25),8,(16,-27),0
```

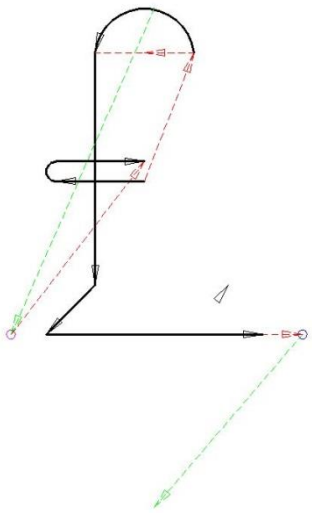
```
*00099,45,[153-™]
2,14,8,(-37,-66),8,(7,66),1,8,(26,0),
2,8,(-13,0),1,8,(0,-36),2,8,(19,0),
1,9,(0,36),(14,-36),(14,36),(0,-36),
(0,0),2,14,8,(-37,-25),8,(7,-30),0
```

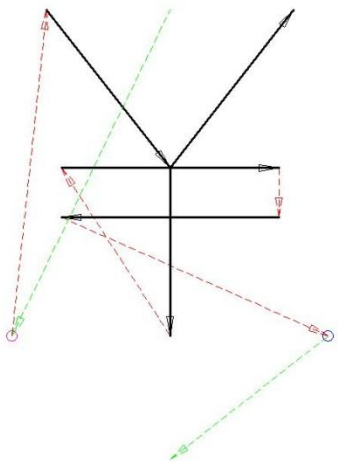




```
*000A2,38,[162-¢]
2,14,8,(-25,-66),8,(44,34),1,13,
(-35,-3,80),(0,-18,25),(35,-3,80),
(0,0),2,8,(-37,-20),1,8,(37,64),
2,14,8,(-26,-35),8,(7,-54),0
```

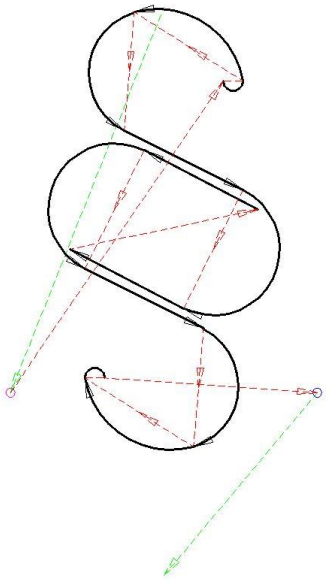
```
*000A3,46,[£163-]
2,14,8,(-29,-66),8,(27,35),1,8,
(-18,0),12,(0,-4,127),8,(18,0),
2,8,(10,26),1,12,(-20,0,114),9,
(0,-47),(-10,-10),(44,0),(0,0),
2,14,8,(-30,-35),8,(7,0),0
```





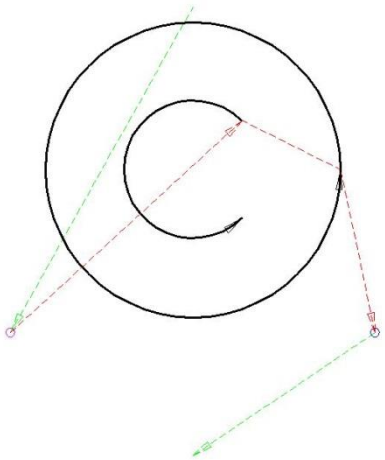
```
*000A5,45,[165-¥]
2,14,8,(-32,-66),8,(7,66),1,8,(25,-32),
5,8,(25,32),6,8,(0,-34),2,8,(-22,34),
1,8,(44,0),2,8,(0,-10),1,8,(-44,0),
2,14,8,(-32,-25),8,(54,-24),0
```

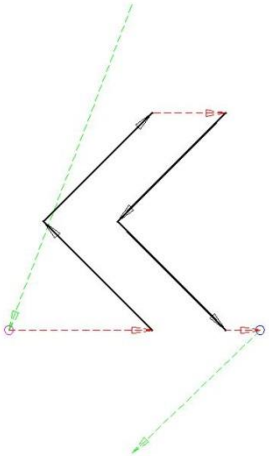
```
*000A6,29,[166-¦]
2,14,8,(-7,-66),8,(7,66),1,8,
(0,-35),2,8,(0,-16),1,8,(0,-35),
2,14,8,(-7,-45),8,(7,20),0
```



```
*000A7,65,[167-§]
2,14,8,(-31,-78),8,(43,63),1,13,
(4,0,127),(-22,14,56),(-2,-24,86),
(24,-12,0),(-12,-24,-127),(-23,12,0),
(0,0),2,8,(38,8),1,13,(-23,12,0),
(-12,-24,127),(24,-12,0),(-2,-24,-86),
(-22,14,-56),(4,0,-127),(0,0),
2,14,8,(-31,-37),8,(43,-3),0
```

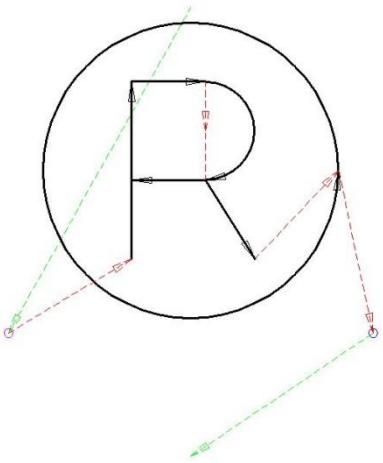
```
*000A9,31,[169-©]
2,14,8,(-37,-66),8,(47,43),5,1,
00A,(14,016)6,2,8,(20,-10),1,00A,
(30,000),2,14,8,(-37,-25),8,(7,-33),0
```

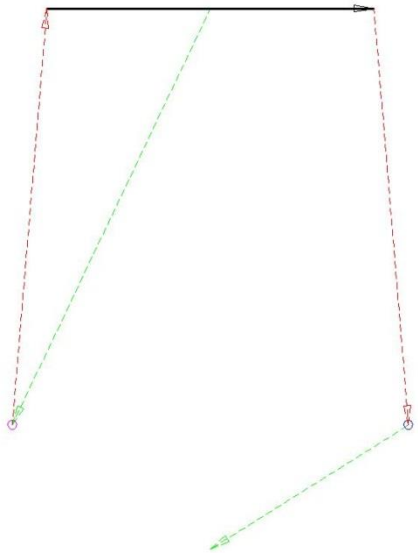




```
*000AB,35,[171-«]
2,14,8,(-25,-66),8,(29,0),1,8,
(-22,22),8,(22,22),2,8,(15,0),
1,8,(-22,-22),8,(22,-22),2,14,8,(-26,-
25),8,(7,0),0
```

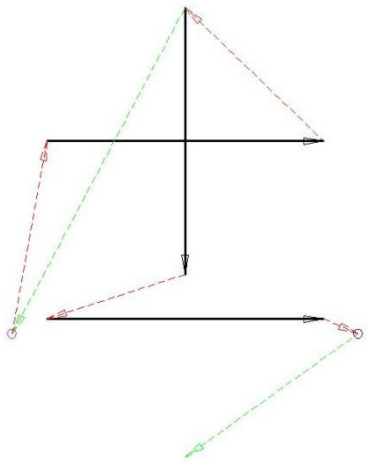
```
*000AE,47,[174-®]
2,14,8,(-37,-66),8,(25,15),1,8,
(0,36),8,(15,0),12,(0,-20,-127),
8,(-15,0),2,8,(15,0),1,8,(10,-16),
2,8,(17,18),1.00A,(30,000),
2,14,8,(-37,-25),8,(7,-33),0
```





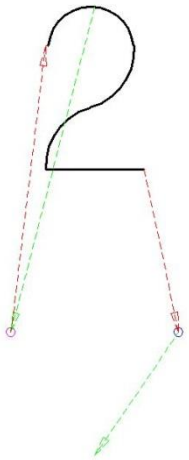
```
*000AF,21,[175-¯]
2,14,8,(-40,-84),8,(7,84),1,8,(66,0),
2,14,8,(-40,-25),8,(7,-84),0
```

```
*000B0,21,[176-°]
2,14,8,(-14,-66),8,(14,66),1.00A,
(7,020),2,14,8,(-14,-25),8,(14,-66),0
```

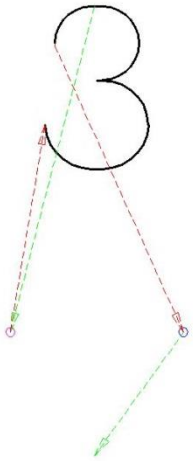


```
*000B1,37,[177-±]
2,14,8,(-35,-66),8,(7,39),
1,8,(56,0),2,8,( -28,27),1,8,(0,-54),
2,8,(-28,-9),1,8,(56,0),
2,14,8,(-35,-25),8,(7,-3),0
```

```
*000B2,40,[178-²]
2,14,8,(-17,-66),8,(7,58),3,2,8,
(1,0),1,13,(35,0,-116),(-18,-25,-45),
(-18,-25,45),(40,0,0),(0,0),
4,2,2,14,8,(-17,-25),8,(7,-33),0
```



Here, the 2nd character 033 (50) could be called directly using code 007. To avoid both the confusion in the case of vertical text and half-and-half (non-integer) values, code 007 was not used, but instead code 003 halved the vector lengths and copied the bytes of character 2 outside the input and output definitions.

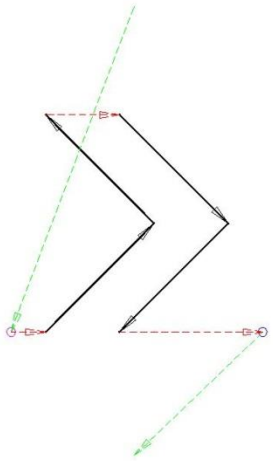


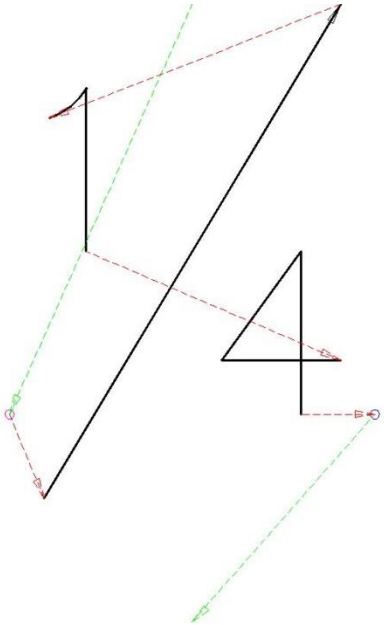
```
*000B3,60,[179-³]
2,14,8,(-17,-66),8,(7,42),3,2,1,
00D,(19,-18,50),(4,0,0),(19,18,50),
(-19,18,54),(0,0),5,8,(-2,0),6,00D,
(15,15,53),(-15,15,53),(-4,0,0),
(-15,-15,53),(0,0),2,8,(0,-1),4,2,
14,8,(-18,-25),8,(26,-58),0
```

```
*000B9,32,[185-¹]
2,14,8,(-11,-66),8,(7,60),3,2,1,
12,(15,12,21),8,(0,-66),2,8,(1,0),
4,2,14,8,(-11,-25),8,(7,-33),0
```

```
*000BA,21,[186-º]
2,14,8,(-16,-66),8,(25,48),
1,00A,(9,000),
2,14,8,(-16,-25),8,(7,-48),0
```

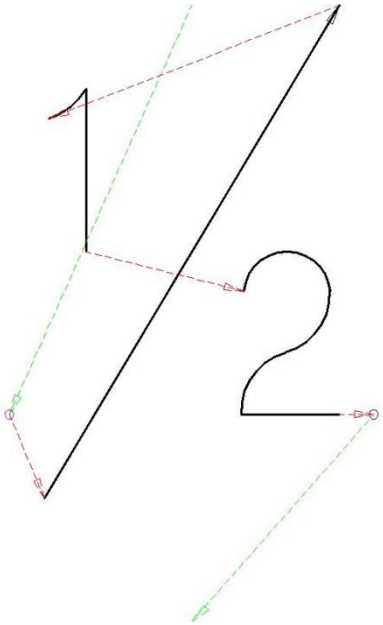
```
*000BB,35,[187-»]
2,14,8,(-25,-66),8,(7,0),1,8,
(22,22),8,(-22,22),2,8,(15,0),
1,8,(22,-22),8,(-22,-22),
2,14,8,(-26,-25),8,(29,0),0
```

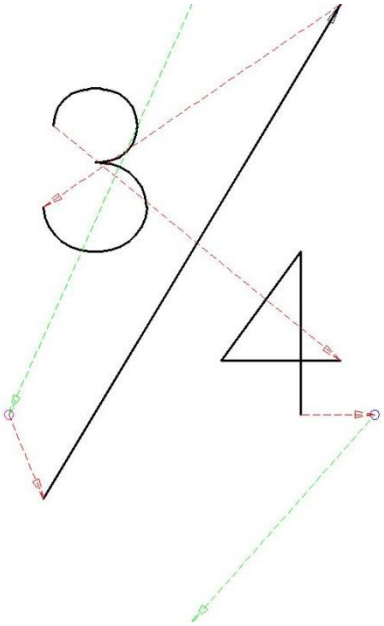




```
*000BC,51,[188-1/4]
2,14,8,(-37,-83),8,(7,-17),
1,8,(60,100),2,8,(-59,-23),
3,2,1,12,(15,12,21),8,(0,-66),
2,8,(103,-44),1,9,(-48,0),
(32,44),(0,-66),(0,0),4,2,
2,14,8,(-37,-42),8,(15,0),0
```

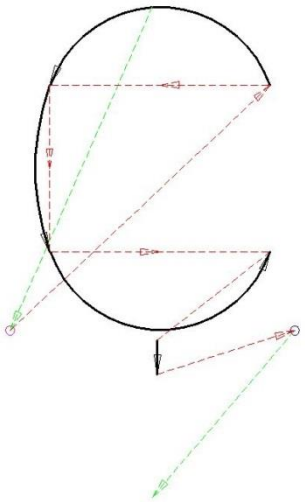
```
*000BD,57,[189-1/2]
2,14,8,(-37,-83),8,(7,-17),1,8,
(60,100),2,8,(-59,-23),3,2,1,12,
(15,12,21),8,(0,-66),2,8,(64,-16),
1,13,(35,0,-116),(-18,-25,-45),
(-18,-25,45),(40,0,0),(0,0),4,2,
2,14,8,(-37,-42),8,(7,0),0
```

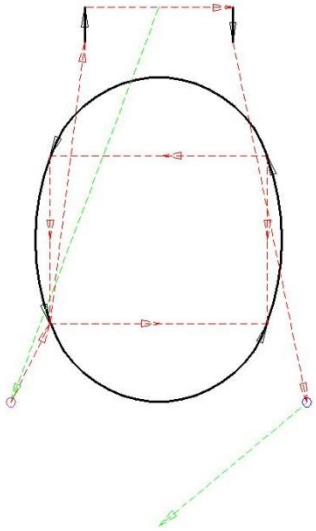




```
*000BE,79,[190-3/4]
2,14,8,(-37,-83),8,(7,-17),1,8,
(60,100),2,8,(-60,-41),3,2,1,00D,
(19,-18,50),(4,0,0),(19,18,50),
(-19,18,54),(0,0),5,8,(-2,0),6,
00D,(15,15,53),(-15,15,53),
(-4,0,0),(-15,-15,53),(0,0),2,8,
(116,-95),1,9,(-48,0),(32,44),(0,-66),
(0,0),4,2,2,14,8,(-37,-42),8,(15,0),0
```

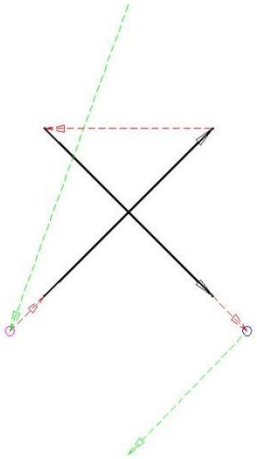
```
*000C7,38,[199-D]
2,14,8,(-29,-66),8,(53,50),1,13,
(-45,0,90),(0,-34,22),(45,0,90),
(0,0),2,8,(-23,-18),1,8,(0,-7),
2,14,8,(-29,-34),8,(28,9),0
```

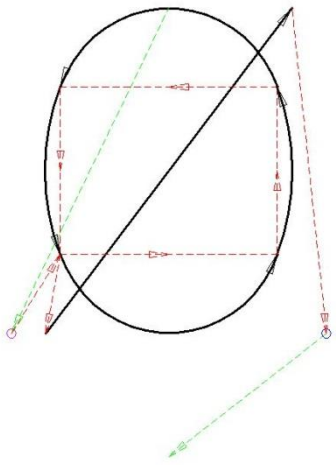




```
*00D6,49,[214-O]
2,14,8,(-30,-80),8,(8,16),1,13,
(44,0,92),(0,34,23),(-44,0,92),
(0,-34,23),(0,0),2,8,(7,57),1,8,
(0,7),2,8,(30,0),1,8,(0,-7),
2,14,8,(-30,-25),8,(15,-73),0
```

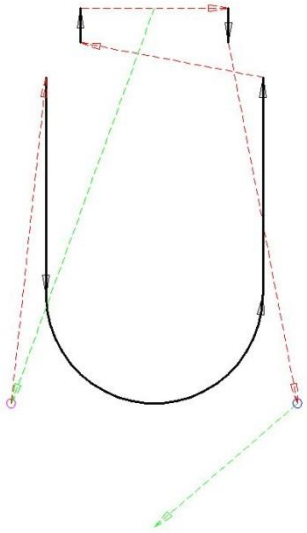
```
*000D7,29,[215-×]
2,14,8,(-24,-66),8,(7,7),
1,8,(34,34),2,8,(-34,0),1,8,(34,-34),
2,14,8,(-24,-25),8,(7,-7),0
```

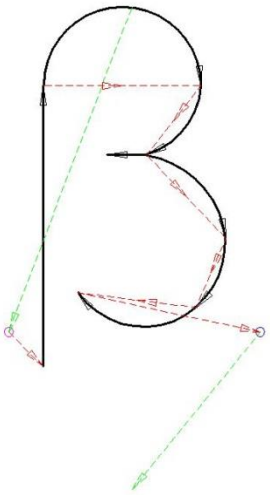




```
*000D8,41,[216-Ø]
2,14,8,(-32,-66),8,(10,16),1,13,
(44,0,92),(0,34,23),(-44,0,92),
(0,-34,23),(0,0),2,8,(-3,-16),
1,8,(50,66),2,14,8,(-32,-25),
8,(7,-66),0
```

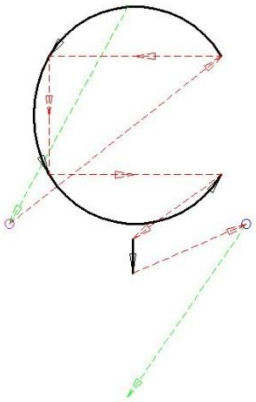
```
*000DC,44,[220-U]
2,14,8,(-29,-80),8,(7,66),1,8,(0,-44),
12,(44,0,127),8,(0,44),2,8,(-37,7),
1,8,(0,7),2,8,(30,0),1,8,(0,-7),
2,14,8,(-29,-25),8,(14,-73),0
```

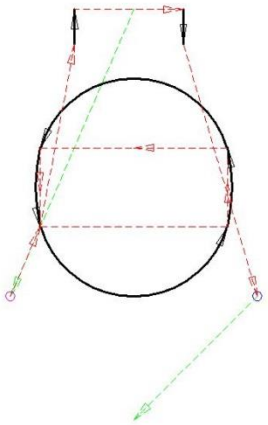




```
*000DF,48,[223-ß]
2,14,8,(-25,-66),8,(7,-),
1,13,(0,57,0),(32,0,-127),
(-11,-14,-41),(0,0),5,8,(-8,0),
6,1,13,(16,-17,-43),(-6,-14,-29),
(-24,3,-57),(0,0),
2,14,8,(-26,-32),8,(37,-8),0
```

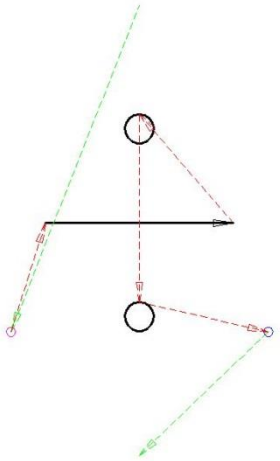
```
*000E7,38,[231-d]
2,14,8,(-24,-44),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(-18,-13),1,8,(0,-7),
2,14,8,(-24,-35),8,(23,10),0
```

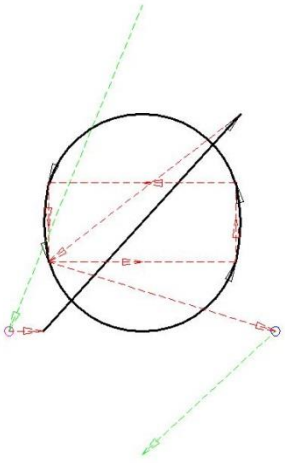




```
*000F6,49,[246-o]
2,14,8,(-25,-58),8,(6,14),1,13,
(38,0,94),(0,16,16),(-38,0,94),
(0,-16,16),(0,0),2,8,(7,37),1,8,(0,7),
2,8,(22,0),1,8,(0,-7),2,14,8,(-25,-
25),8,(15,-51),0
```

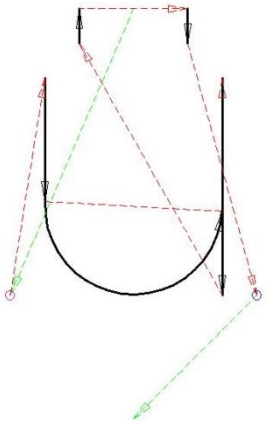
```
*000F7,37,[247-÷]
2,14,8,(-26,-66),8,(7,22),
1.8,(38,0),2.8,(-19.22),1.00A,
(3.020),2.8,(0,-38),1.00A,(3.020),
2,14,8,(-26,-25),8,(26,-6),0
```

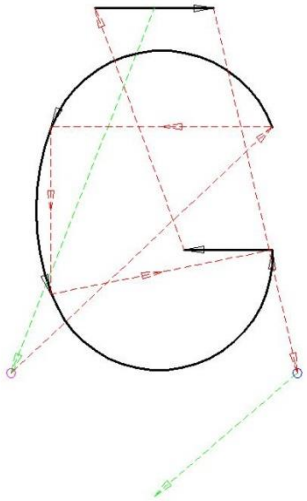




```
*000F8,41,[248-ø]
2,14,8,(-27,-66), 8,(7,0),
1,8,(40,44),2,8,(-39,-30),
1,13,(38,0,94), (0,16,16),(-38,0,94),
(0,-16,16),(0,0),
 2,14,8,(-27,-25),8,(46,-14),0
```

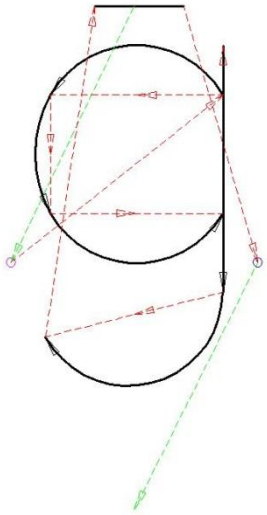
```
*000FC,49,[252-ü]
2,14,8,(-25,-58),8,(7,44),1,8,
(0,-25),12,(36,-2,127),2,8,(0,27),
1,8,(0,-44),2,8,(-29,51),1,8,(0,7),
2,8,(22,0),1,8,(0,-7),
2,14,8,(-25,-25),8,(14,-51),0
```

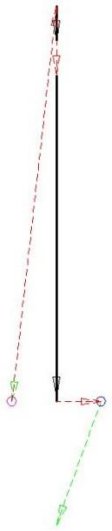




```
*0011E,41,[286-D]
2,14,8,(-29,-74),8,(53,50),1,13,
(-45,0,88),(0,-34,23),(45,9,111),
(-18,0,0),(0,0),2,8,(-
18,49),1,8,(24,0),
2,14,8,(-29,-25),8,(17,-74),0
```

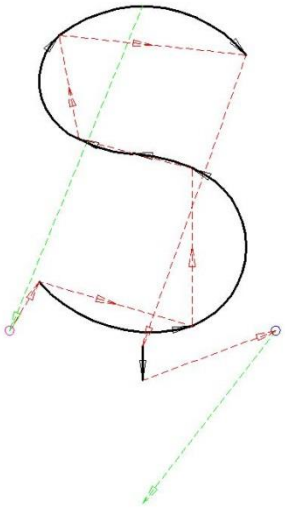
```
*0011F,50,[287-ğ]
2,14,8,(-25,-52),8,(43,34),1,13,
(-35,-0,73),(0,-24,32),(35,0,73),
(0,0),2,8,(0,34),1,8,(0,-50),00C,
(-36,-9,-98),2,8,(10,67),1,8,(18,0),
2,14,8,(-25,-50),8,(15,-52),0
```





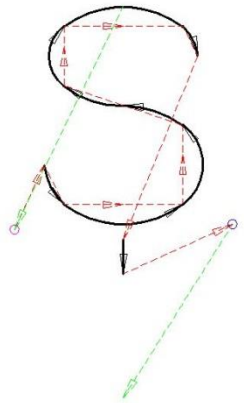
```
*00130,29,[304-I]
2,14,8,(-9,-66),8,(9,80),1,8,
(0,-7),2,8,(0,-7),1,8,(0,-66),
2,14,8,(-9,-25),8,(9,0),0
```

```
*00131,21,[305-I]
2,14,8,(-9,-44),8,(9,44),1,8,(0,-44),
2,14,8,(-9,-25),8,(9,0),0
```



```
*0015E,47,[350-S]
2,14,8,(-27,-66),8,(6,10),1,13,
(31,-9,37),(0,32,87),(-12,3,10),
(-11,3,-14),(-4,21,-70),(38,-4,-52),
(0,0),2,8,(-21,-59),1,8,(0,-7),
2,14,8,(-27,-35),8,(27,10),0
```

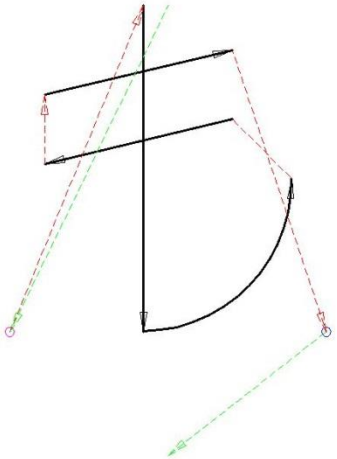
```
*0015F,53,[351-s]
2,14,8,(-22,-45),8,(6,13),1,13,
(4,-8,25),(24,0,42),(0,16,63),
(-12,4,20),(-12,4,-23),(0,12,-63),
(24,0,-42),(3,-6,-25),(0,0),2,8,
(-15,-37),1,8,(0,-7),2,14,
8,(-22,-35),8,(22,10),0
```



```
*02205,4,[8709-ø]
7.0F8.0
```

Identify character 02205 (8709) by calling character 0F8 (248) to make the circle diameter sign (fi) using %%c.

```
*020BA,41,[8378-₺]
2,14,8,(-32,-66),8,(27,66),1,8,
(0,-66),12,(30,31,53),2,8,
(-12,12),1,8,(-38,-9),2,8,(0,14),
1,8,(38,9),2,14,8,(-32,-25),
8,(19,-57),0
```



It should be noted that at least one blank line (Enter) must be entered after the last definition byte (0) of the last character definition of the Font Definition file (*.shp).