Improve CNC Productivity with Parametric Programming

Mike Lynch - CNC Concepts, Inc. - 847-639-8847 - lynch@cncci.com

Companion materials available for download at:

http://www.cncci.com/protect/product10/pmpapres.zip

User name: concept10 Password: machining

Objective: To show how you can improve CNC machine tool utilization through the use of parametric programming.

Outline

- Introduction to parametric programming (FANUC Custom Macro)
- Five application categories
- Computer- and CNC-related features
- Part Family and user created canned cycles examples
- Suggestions to improve productivity
  - Program verification and optimizing
  - Sizing in the first workpiece on Swiss (sliding headstock) lathe
  - Use an edge finder like a spindle probe
  - Streamline tool length measurement on a machining center
  - Take calculations out of sizing adjustments
  - Eliminate program zero assignment
  - Centering the Y axis on live-tooling lathes
  - Use a universal program for jaw boring
  - Facilitate calculations for needed values

Introduction to parametric programming

Parametric programming goes by many names. FANUC (or any control manufacturer that claims to be FANUC-compatible) calls it Custom Macro. Fadal calls it Macro. Okuma calls it User Task. Sodick calls it Q Routine. Some control manufacturers have parametric programming capabilities but have not named it with any special name. The focus in this text will be on FANUC's version, called Custom Macro.

While the variations from one version of parametric programming to another lead to differences in specific usage techniques, the broader applications-for and usage-of parametric programming remain remarkably similar. This is evidenced by the fact that the majority of applications described in this text can be adapted to every version of parametric programming just mentioned. Just as a given software application can be handled by a variety of computer programming languages, so can a given CNC application be handled with different versions of parametric programming.

Comparison to sub-programming

The best way to get comfortable with any complex subject matter is to compare it to simpler topics with which you may already be familiar. Custom Macro is no exception. If, for example, you have worked with the sub-programming functions of your control, you have scratched the surface of what can be done with Custom Macro.

All CNC controls have sub-programming functions to allow commands within the CNC program to be repeated. This minimizes the number of commands that must be given in the CNC program. If, for instance, five identical pockets must be milled in workpieces during the same cutting cycle, it would be cumbersome to program each pocket independently. Instead, you can program just one of the pockets and place the commands in a separate (sub) program. These redundant commands can be executed five times to machine the five pockets, eliminating many cumbersome, lengthy, and error prone commands.

FANUC uses an M98 to call a sub-program. A P word within the M98 specifies the sub-program number. An L word specifies the number of executions of the sub-program. Consider this command:

```
N050 M98 P1000 L5
```

It tells the machine to execute sub-program O1000 five times. As long as program O1000 contains the commands needed to correctly machine one of the pockets, the program’s length can be shortened and the potential for mistakes will be reduced.

Sub-programming techniques can be very helpful. However, if anything changes about the pocked from one pocket to the next (width, height, depth, etc.), you cannot use sub-programming techniques. Without Custom Macro, each pocket must be programmed independently. In addition to giving the programmer the ability to repeat redundant commands, Custom Macro allows *anything* to change from one execution of the Custom Macro program to the next. In the pocket example, if any pocket-attribute changes from one pocket to the next (width, height, depth, etc.) these variations can be easily handled within the Custom Macro program.

In this sense, Custom Macro programming gives a programmer the ability to write a general purpose sub-program. If you have ever found yourself wishing you had the ability to write general purpose sub-programs, you have an application for Custom Macro.

The things that change from one pocket to the next are called *arguments*. Custom Macro lets you name the arguments in a logical manner. You use a G65 command to call the Custom Macro

program. Letters of the alphabet can be included in this command to specify argument values. Consider this G65 command:

```
N050 G65 P1000 X2.0 Y1.5 W4.0 H2.0 D.25
```

G65 is a Custom Macro *call statement*. The P-word specifies the program number of the pocket-milling Custom Macro program. Letter-addresses X and Y specify the lower left hand corner position of this pocket along the X any Y axis. Letter-address W specifies the pocket width, H specifies the height of the pocket, and D specifies the pocket depth.

Notice how logical you can make the entry of input data (*you* name them). Anyone can easily recognize the meanings of letter-addresses X, Y, W, H, and D. If another pocket of a different size must be machined, another G65 command can be easily specified that contains different argument values.

Comparison to canned cycles

All CNC control manufacturers provide a series of programming features to minimize a programmer's work. FANUC-controlled machining centers, for instance, come with a set of hole-machining canned cycles (specified by G73-G89). Some machining center controls also have certain milling canned cycles like circle pocket milling, slot milling, thread milling, and face milling. FANUC-controlled turning centers come with a set of multiple repetitive cycles for rough & finish turning and boring, grooving, hole machining, and threading. It is likely that you are familiar with at least some of these cycles. Let's compare what you know to Custom Macro.

Here are the commands to drill a series of holes on a FANUC-controlled machining center.

```
.
.
.
N065 G54 G90 S400 M03 (Select coordinate system, absolute mode, and start spindle)
N075 G00 X1.0 Y2.0 (Rapid to first hole location)
N080 G43 H01 Z0.1 (Instate tool length compensation, move to Z approach position)
N085 G81 R0.1 Z-0.75 F4.5 (Drill first hole)
N090 X3.0 (Drill second hole)
N095 X5.0 (Drill third hole)
N100 X7.0 (Drill fourth hole)
N105 G80 (Cancel cycle)
N110 G91 G28 Z0 M19 (Return to Z axis reference position)
.
.
.
```

In line N085, the first hole is completely machined based upon FANUC's G81 function and the words included in the command (R, Z, F, etc.). The machine will perform a series of previously planned motions based on the canned cycle’s design. With G81, the machine will, first, rapid a drill to the XY position. Next, it will rapid the drill to the R plane, plunge the drill to the hole bottom, and retract the drill from the hole. So with G81, four movements are generated with one command. With other canned cycles (like peck drilling) even more movements are caused by one command.

Notice how similar the G81 command format is to that of the pocket milling example calling G65 command shown earlier. The R-, Z-, and F-words in the G81 (or any canned cycle) are like the arguments being passed to the Custom Macro program. You can think of all canned cycles as being like Custom Macro programs written and maintained by the CNC control manufacturer.

If your control does not have a needed canned cycle, or if you do not agree with how a given canned cycle functions, you can develop a Custom Macro program to handle the application. In essence, you can create your own canned cycles.

If you have ever wanted the ability to create your own canned cycles, you have an application for Custom Macro.

Comparison to computer programming

If you have had experience with any computer programming language, you already know much of what is available with Custom Macro. There are many computer-related features of Custom Macro that closely resemble those found in computer programming languages. These features include variables, arithmetic, logic, and looping. For now, suffice it to say that most of what can be done in computer programming languages can be done within Custom Macro programs. If you have had no previous computer programming experience, we again recommend that you pick up a beginner's book on computer programming. It will reinforce the presentations we make about computer-related features of Custom Macro.

If you have ever found yourself wishing you could include computer-programming-like commands in your CNC program, you have an application for Custom Macro.

All programs can use Custom Macro functions

You may be wondering if anything special must be done in order in order to utilize Custom Macro commands in a program. When the Custom Macro option has been equipped on a machine, you have access to its function set that extends what you can do with normal G-code level programming. These additional (Custom Macro) functions will be available for use from within CNC programs.

Custom Macro functionality can be utilized from within any CNC program, including main programs and sub-programs. You do not have to do anything special to use Custom Macro functions in a program. Programs can reside in CNC memory or on an external device, like a memory card or flash drive.

FANUC controls will interpret CNC commands a bit differently than Custom Macro (arithmetic and logic) commands. If the SINGLE BLOCK switch is turned on, for example, the machine will stop after every CNC command. Depending on a parameter setting, the machine may not stop after each Custom Macro command.

Application categories

As stated in the preface, there are countless applications for Custom Macro, and almost every CNC user has at least some good applications. In this discussion, we organize all applications for Custom Macro into five basic categories.

Part-families

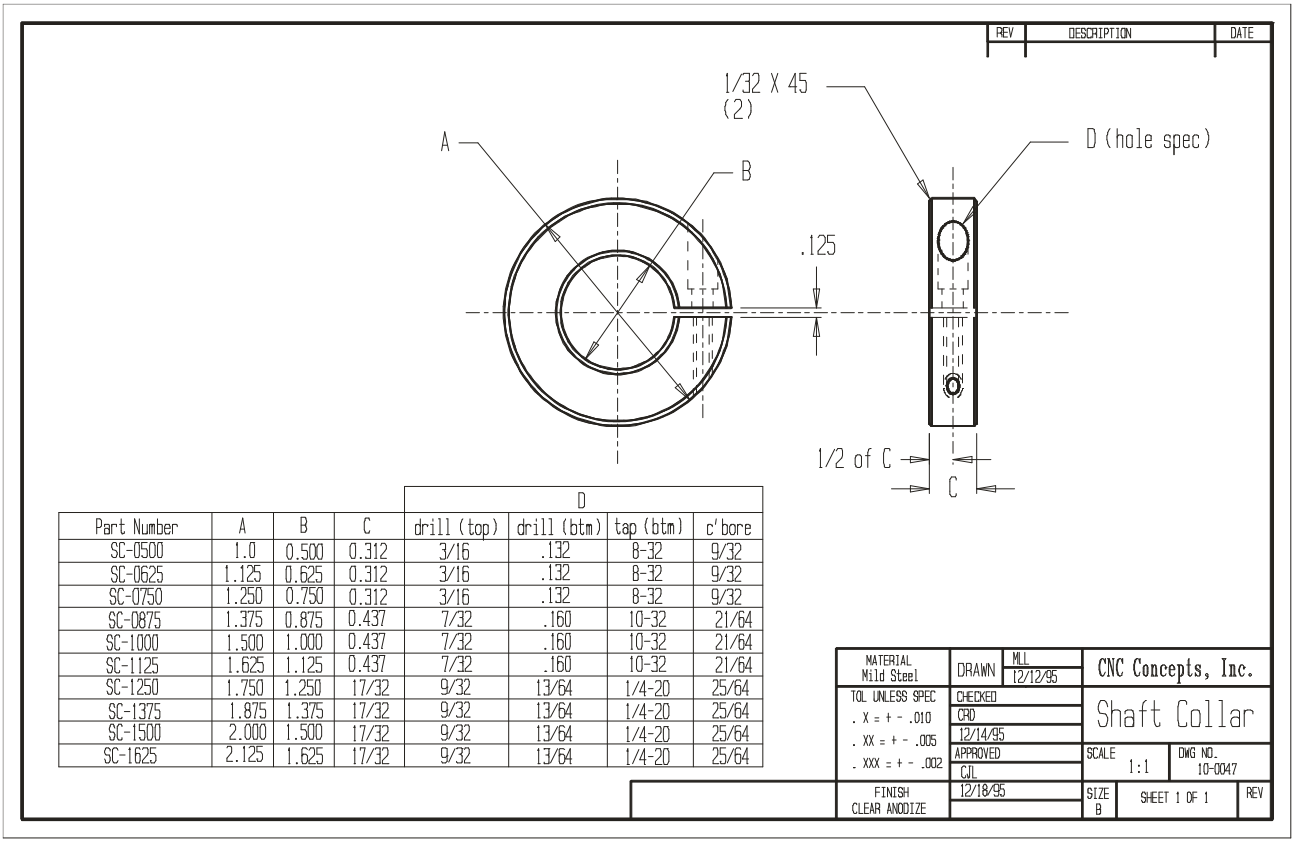
Many CNC users machine a series of very similar workpieces. Groups of similar workpieces are called part-families. Generally speaking, all workpieces in a part-family closely resemble one another and require a similar (if not identical) machining process. In classic part-families, only workpiece size changes.

Bolts, screws, nuts, washers, and pins, for example, are made in a variety of sizes to suit the needs of industry. The hex-shaped sockets a hand tool manufacturer makes are made in various sizes to accept changing bolt and nut sizes. The rings a piston ring manufacturer makes are made in various sizes and used with a variety of piston sizes. The list of common part-families is virtually endless.

How are your drawings dimensioned?

Some companies utilize variable dimensioning techniques for dimensioning a family-of-parts. A design engineer will dimension values that change with a letter of the alphabet. Any person viewing the drawing will determine the value of a given dimension by referencing a chart included on the drawing. By knowing the workpiece's part number, anyone can find the values of each variable dimension. The next illustration shows an example of this kind of part-family dimensioning.

If your company uses variable dimensioning techniques, you have a part-family application for Custom Macro.



				D			
Part Number	A	B	C	drill (top)	drill (btm)	tap (btm)	c'bore
SC-0500	1.0	0.500	0.312	3/16	.132	8-32	9/32
SC-0625	1.125	0.625	0.312	3/16	.132	8-32	9/32
SC-0750	1.250	0.750	0.312	3/16	.132	8-32	9/32
SC-0875	1.375	0.875	0.437	7/32	.160	10-32	21/64
SC-1000	1.500	1.000	0.437	7/32	.160	10-32	21/64
SC-1125	1.625	1.125	0.437	7/32	.160	10-32	21/64
SC-1250	1.750	1.250	17/32	9/32	13/64	1/4-20	25/64
SC-1375	1.875	1.375	17/32	9/32	13/64	1/4-20	25/64
SC-1500	2.000	1.500	17/32	9/32	13/64	1/4-20	25/64
SC-1625	2.125	1.625	17/32	9/32	13/64	1/4-20	25/64

A part-family application for Custom Macro

Notice that dimensions are specified with letters (A, B, C, etc.). Custom Macro often allows you to designate the value of each argument with the same letters that are used on the drawing. For example, here is one way to specify that part number SC-0875 is to be made from the previous illustration:

```
N060 G65 P1000 A1.375 B0.875 C0.437 D0.1875
```

In this example, notice how arguments A, B, C, and D directly correspond to print dimensions (D specifies the counter-bored hole-diameter, which in turn, determines how the rest of the hole must be machined). Though there may be other changing attributes to be handled for this application (speed & speed variations, tool station numbers, etc.), this example command should nicely stress how easy it can be to specify which workpiece in a family is being machined.

The more workpieces in a part-family, the easier it is to justify Custom Macro programming techniques. Keep in mind, however, that Custom Macro programs do take longer to write than conventional CNC programs. From a strictly programming-time-based justification standpoint (not considering program maintenance issues), it may be difficult to justify writing a Custom Macro program for part-families having but a few workpieces. Even for simple part families, it can take

from 3-10 times longer to write the Custom Macro program than it will take to write one hard-and-fixed CNC program for a workpiece in the part-family.

User-created canned cycles

Even if your company's products contain no part-families, it is likely that you have at least some similar machining operations required on multiple workpieces. Custom Macro can dramatically streamline the programming of repeated machining operations. Many machining operations are similar in nature and can be easily handled by using Custom Macro programming techniques. Here are some examples you should easily recognize.

Machining centers:

Thread milling
Round pocket milling
Rectangular pocket milling
Circle milling
Face milling
Keyway milling
Slot milling
Hole-machining (drilling, tapping, reaming, boring, etc.)
Hole-pattern machining (bolt-hole circle, grid pattern, window pattern, etc.)

Turning centers:

Groove necking
Tapping (some turning centers do not have this cycle)
Deep hole peck drilling (some turning centers do not have this cycle)
Knurling

If you have ever found yourself wishing that your control's canned cycles worked differently, you have a user-created canned cycle application for Custom Macro.

In addition to modifying the method by which your current canned cycles work, you have the ability to create your own canned cycles with Custom Macro. Most machining centers, for example, do not have a canned cycle for thread milling. If you perform thread milling on a regular basis and if your machine does not have a thread milling cycle, you are likely writing many tedious, redundant, and error prone commands. With Custom Macro, you can create your own thread milling canned cycle.

In similar fashion, most turning center controls do not include an adequate canned cycle to machine grooves. If you must neck grooves in many workpieces and if your machine does not have a canned cycle for machining grooves, again, you must write many tedious, redundant, and error prone commands. With Custom Macro, you can create your own grooving cycle.

Many companies perform unusual machining operations that are specific only to their own products and manufacturing processes, and no control manufacturer will consider creating canned cycles for machining operations that are not helpful to the majority of their users. Relatively few machining center users, for example, machine dovetails. This machining operation normally requires a number of successive milling passes with a dovetail cutter.

Utilities

This application category tends to be the most overlooked. Even people who consider themselves well versed may not be aware of the countless utility applications for Custom Macro. There is a good reason why utility applications are not so well known. Most incorporate lesser known *CNC*-

related features of Custom Macro. And before you can incorporate any new technique, of course, you must know it is possible to do so. While we discuss CNC-related features of Custom Macro in during lessons six and seven, we wish to at least introduce you to some of the remarkable possibilities now.

Utility applications can reduce setup and program verification time, the can reduce production-run time, they can catch mistakes, they can make a CNC machine easier and safer to run, they can emulate costly options, and in general, they can in some way facilitate CNC machine tool usage. In fact, any specific problem you are having with CNC machine tool utilization can probably be facilitated in some manner through the use of a utility Custom Macro program. You say this sounds like a pretty bold statement? Let's look at some specific examples of utility applications that reinforce these claims.

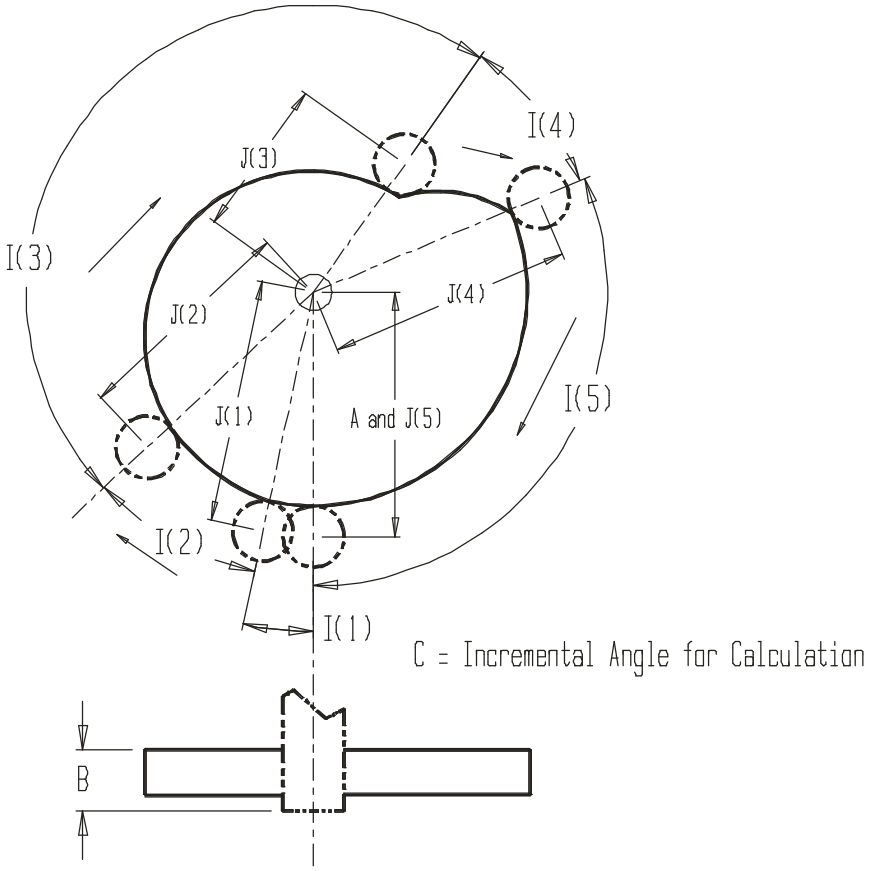
Complex motions and geometric shapes

Custom Macro programs can perform basic arithmetic functions like equality (substitution), addition, subtraction, multiplication, and division, as well as certain higher level functions like sine, cosine, tangent, square root, and rounding (among many others). Arithmetic, combined with the ability to perform logic and looping, give the Custom Macro programmer the power to create Custom Macro programs capable of machining highly complex geometric shapes. Indeed, any shape that can be defined with an arithmetic calculation can be machined with a Custom Macro program.

While Custom Macro programs for standard shapes like spheres, pyramids, ellipses, and splines can be created with relative ease, the difficulty of writing Custom Macro programs for applications in this category is directly related to the complexity of the workpiece to be machined. When it comes right down to it, it is usually easier to utilize a computer aided manufacturing (CAM) system for complex work. However, you should know that Custom Macro programming for complex shapes (when feasible) has two advantages over even the most sophisticated CAM systems.

First, G-code programs for complex shapes that are generated by CAM systems tend to be very long. It is not unusual for a program that simply machines a spherical shape to include hundreds (if not thousands) of commands. The comparable sphere milling Custom Macro program (shown in lesson eight) will consist of about fifty commands.

Second, Custom Macro programming gives you the ability to create motions that are specific to your own application/s. The Custom Macro program for the circular cam shown in the next illustration, for example, is very easy to use. While there are CAM systems capable of creating CNC programs to machine a circular cam, there are also many that cannot, and of those that do, none make it as simple as a Custom Macro program developed specifically for this purpose.



A circular cam machined by a Custom Macro program.

No computer aided manufacturing system we know of makes it as simple for the user to input the values needed to machine the circular cam as this example command using Custom Macro's argument assignment number two. (The Custom Macro program for this application is shown in lesson seven.)

N010 G65 P1002 A1.75 B-0.5 C0.1 I12.0 J1.75 I35.0 J1.625 I175.0 J1.1 I25.0 J1.75 I114.0 J1.75

Driving accessory devices

As stated earlier, there are certain devices associated with CNC machine tools that require a higher level of programming than can be found with standard G-code level manual programming. Most touch-probing systems like spindle probes, for example, are programmed at G-code level yet require sophisticated programming functions. When a touch-probe stylus contacts a surface, the program must have a way to locate the surface just contacted. Once located, it is likely that some calculations will be necessary to determine if the surface is where it should be. If it is not, possibly an alarm should be sounded. Possibly an offset should be adjusted. Possibly the machine should be made to stop.

Feature types

There are computer-related and CNC-related features in Custom Macro. They provide tremendous programming capabilities to help improve CNC productivity.

Computer related features

These are features that are similar to those found in computer programming languages.

Variables

One of the most important features of any programming language, including any version of parametric programming, is the ability to incorporate variables. Though the syntax and structure for variable usage changes from one version of parametric programming or computer programming to another, the general applications remain remarkably similar.

If you have experience with computer programming, you know what variables are. The use and application of variables in Custom Macro are much the same as they are in any programming language. However, people that have no computer programming experience may find it somewhat difficult to understand the concept of variables. For this reason, we begin with two simple analogies to help you understand variables.

Arguments

Arguments specify input data (values that can change from one time the Custom Macro program is used the next). They tell the Custom Macro program how to behave *right now* - or *this time*. They are especially helpful in part-family and user-created canned cycle applications. There are actually two ways to use arguments in Custom Macro based on which of these two these application categories is being developed.

Local variables

Local variables have two purposes. The primary purpose for local variables is to represent letter-address arguments that are specified in a G65 command. The secondary purpose is for use in general purpose calculations (just like *common variables* which are described a little later).

Local variables with argument assignment number one

As you now know, when a Custom Macro program is called with a G65 command, arguments take the form of letter-addresses. And again, allowable letter-address arguments with argument assignment number one include:

A, B, C, D, E, F, H, I, J, K, M, Q, R, S, T, U, V, W, X, Y, and Z

Though it may be a little confusing, you are not allowed to directly reference an argument within the Custom Macro program with its letter-address. If you try to do so, the machine will confuse the argument name (A, B, C, X, Y, Z, etc.) with the actual CNC meaning for the letter-address.

For this reason, you must reference arguments in the Custom Macro program by their corresponding *local variables*. Like all variable types in Custom Macro (except letter-address arguments), local variables begin with a pound sign and are specified with a number. The next table shows the list containing all allowable letter-address arguments and their corresponding local variable numbers. This is the most popular form of argument assignment, called argument assignment number one.

Letter	Local Variable
A	#1
B	#2
C	#3
D	#7
E	#8
F	#9
H	#11

I	#4
J	#5
K	#6
M	#13
Q	#17
R	#18
S	#19
T	#20

U	#21
V	#22
W	#23
X	#24
Y	#25
Z	#26

Letter-address arguments and their representing local variables when using argument assignment number one.

When a G65 command is executed, two things will happen. First, all local variables that have corresponding letter-address arguments in the G65 command will be set. Second, the Custom Macro specified by the P word will be executed. During the execution of the Custom Macro, local variables will contain the values of arguments specified in the G65 command. Within the Custom Macro, local variables must be used whenever you need to reference argument values specified in the G65 calling command.

As an example, consider this command:

N025 G65 P1000 X0. Y0. Z0. W5.0 H3.0 T0.5 D1.0 F5.0 (Mill right side)

It first sets the local variables corresponding to letter-addresses X, Y, Z, W, H, T, D, and F. In this case, local variables will be set as follows:

- #24 (for X) set to 0
- #25 (for Y) set to 0
- #26 (for Z) set to 0
- #23 (for W) set to 5.0
- #11 (for H) set to 3.0
- #20 (for T) set to 0.5
- #7 (for D) set to 1.0
- #9 (for F) set to 5.0.

Second, the machine will execute program O1000. When the programmer needs to reference the value of argument X within the Custom Macro program, #24 must be used.

Common variables

Common variables are introduced during the discussion of arguments used in part-family applications. Again, common variables range in the #100 series and at least fifty are available, from #100 through at least #149. It is likely that your machine has more. If your machine has two hundred common variables, for instance, they will range from #100 through #199.

As the name implies, common variables remain active even after the execution of the Custom Macro program. Most FANUC CNCs are set to retain common variable values (again, in the #100 series) until the power to the machine is turned off, at which time they are set back to vacant.

Permanent common variables

These variables range in the #500 series. FANUC CNCs will have at least ten permanent common variables (from #500 through #509). Most have many more, possibly ranging from #500 through #999. As the name implies, these variables are retained even after the power is

turned off. In this sense, they are much like tool offsets. And like tool offsets, most CNCs let you modify the values of permanent common variables through the MDI panel keyboard and display screen.

Since some CNCs have so few permanent common variables (again, as few as ten), be sure to reserve their use for those applications when it is necessary to retain data from day to day. One example is a with utility Custom Macro program for part counting. Say you wish to set up a part counter. The machine will run 750 workpieces and then halt the machine (this application is shown in lesson six). As you begin running production, your Custom Macro based part counter will begin counting workpieces.

It isn't very likely that you will finish all 750 workpieces before the end of the day/shift. It will more likely take several days to complete the production run. In this case, you will need your part counting Custom Macro program to remember where it left off from day to day (after the power is turned back on). If you store the current part count in a #500 series permanent common variable, the part count value will not be lost when the power is turned off.

System variables

Many of the CNC-related features of Custom Macro are accessed with system variables. These variables range from #1000 through about #12000. Many can also be referenced by names that are easier to remember than the system variable numbers (like [#_DATE] instead of #3011. System variables allow you to access many CNC functions, including current axis position, current values of offsets, alarm generation, and much more.

Arithmetic

Anything that can be done on a scientific calculator can be done in a Custom Macro program.

Basic functions

As stated, Custom Macro allows equality (substitution), addition, subtraction, multiplication, and division. The characters used to represent each basic arithmetic function are universally accepted, they are used by all computer programming languages. The equal sign (=) is used for equality, the plus sign (+) for addition, the minus sign (-) for subtraction, the asterisk (*) for multiplication, and the slash code (/) for division. The use of these basic arithmetic functions should be familiar, since applications for their use are shown in lesson two.

Advanced functions

As stated, most versions of parametric programming allow much more than simple equality, addition, subtraction, multiplication and division. Here we introduce those functions supported by Custom Macro.

Trigonometry functions

Custom Macro provides enough trigonometry functions to handle even the most complex right-angle trigonometry problems. The next table shows Custom Macro trigonometry functions including their names, syntax, a simple example, and the answer:

Function	Syntax	Example	Value of #100

Sine	#i=SIN[#j]	#100 = SIN[30.0]	0.5
Cosine	#i=COS[#j]	#100 = COS[30.0]	0.866025
Tangent	#i=TAN[#j]	#100 = TAN[30.0]	0.577350269
Arc Sine	#i=ASIN[#j]	#100 = ASIN[0.5]	30.0
Arc Cosine	#i=ACOS[#j]	#100 = ACOS[0.866025]	30.0
Arc Tangent (method 1)	#i=ATAN[#j]	#100 = ATAN[0.577350269]	30.0
Arc Tangent (method 2)	#i=ATAN[#j]/[#k]	#100 = ATAN[0.5]/[0.866025]	30.0
Arc Tangent (method 3)	#i=ATAN[#j,#k]	#100 = ATAN[0.5,0.866025]	30.0

Trigonometry functions in the current version of Custom Macro.

Rounding functions

Custom Macro allows the rounding of real numbers (values that include a decimal portion). ROUND will round a number to the next closest integer (up or down). FIX will round a number down to the next lower integer. FUP will round a number up to the next higher integer.

The next table shows the rounding function, their syntax, an example, and the answer.

Function	**Syntax**	**Example**	**Value of #100**
Round	#i=ROUND[#j]	#100 = ROUND[3.2]	3.0
Round down	#i=FIX[#j]	#100 = FIX[3.8]	3.0
Round up	#i=FUP[#j]	#100 =FUP[3.2]	4.0

Rounding functions in Custom Macro.

Other arithmetic functions

There are other arithmetic functions available in Custom Macro. Some may be quite important to you while others may be seldom, if ever, used. An understanding of mathematics is required to understand them all, and your own ingenuity may inspire times when they can be useful. We will introduce each, show its syntax, and if possible, provide an example of its use.

Square root

As the name implies, this function (specified with the Custom Macro word SQRT) returns the square root of the specified value or expression. Consider this example:

#100 = 5.0 (Short side of triangle)
#101 = 8.0 (Long side of triangle)
#102 = SQRT[[#100 * #100] + [#101 * #101]] (Hypotenuse of triangle)

All higher level functions (including square root) allow arithmetic expressions to be performed internal to the brackets of the function. In this case, everything in brackets will be calculated prior to the square root function. You may recognize this example as Pythagorean's Theorem (calculating a triangle's hypotenuse length by applying the square root to the side-adjacent squared plus the side-opposite squared).

Absolute value

This function, specified with ABS, returns the *magnitude value* (without polarity) of any value or expression specified within its brackets. While it may not be the most appropriate use, absolute function has the effect of ensuring that a value that cannot be negative.

Power

This function provides the ability to multiply a number times itself a specified number of times. Here is the syntax.

```
#100 = POW[4.0,2.0]
```

Natural logarithm and exponent using base e (2.718…)

Though we do not provide any applications (frankly because I cannot think of any) for logarithms and exponents using base e in Custom Macro), it is possible to use them within Custom Macro programs. Again, with your situation and ingenuity, you may come up with suitable applications. So you should at least know it is possible to incorporate natural logarithms and exponents in your Custom Macro programs.

The syntax for natural logarithm is as follows:

```
#100 = LN[3.0]
```

Binary and binary coded decimal conversions

We do not provide any applications for these functions, but they are available should you find a need. The function BIN converts a binary coded decimal value to binary format. The function BCD converts a binary value (made up of eight digits of zeros and ones) to binary coded decimal format.

Reading parameter values

This is a relatively new feature in Custom Macro and is not available in all older versions (like Custom Macro B). We consider the ability to read parameter values from within Custom Macro programs to be a CNC-related feature of Custom Macro. However, FANUC includes this function in the list of arithmetic functions so we include a brief introduction here. Rest assured that more information will be provided when we address CNC-related features of Custom Macro in lesson six.

Priority of arithmetic operations

As stated earlier, you must understand the order by which arithmetic operations are performed in order to develop expressions that combine arithmetic operations. Custom Macro follows the same priority as any computer programming language. Now that we have shown all of the arithmetic functions, we can show the complete priority of arithmetic operations from highest priority to lowest.

1) Operations in brackets

2) Higher level functions (SIN, COS, TAN, SQRT, etc.)

3) Multiplication then division

4) Addition then subtraction

While it will not have any bearing on the outcome of the expression, the control will always work through the expression at a given level (1-4) from left to right. If performing at level three (multiplication then division), for example, it will perform multiplication from the left to right in the expression. Then it will perform division from left to right.

As stated earlier, if you are in doubt about operation execution order, you can always use brackets to force the order of execution you need. Remember that when you do so, however, you will be wasting nest levels if you "force" the machine to perform the calculation in the same order it would have otherwise done.

Logic and program flow control

The general flow (order of execution) for a normal CNC G-code level program is from beginning to end. When a program is activated, the control will read, interpret, and execute the very first command in your program. It will then proceed to the second command. Read, interpret, and execute. Then it will move on to the third command, then the next, and then the next. It is by this sequential order that CNC programs are normally executed.

With Custom Macro programming, you have the ability to change the order of program execution. You can have Custom Macro programs make tests to determine which of two or more possible outcomes it will follow. You can even make the machine repeat a series of commands until some criteria is/are satisfied. We'll call this powerful ability to change the order of program execution program flow control.

There are several commands in Custom Macro that are related to program flow control. In this lesson, we will introduce program flow command types and then show examples of when they are used.

Statement labels

Some program flow commands require a unique marker to designate a position in the program that is to be used to control program flow. We call these markers *statement labels*. Statement labels specify points in the program where the CNC is told to continue executing the program.

FANUC uses sequence numbers (N words) as statement labels. This text shows a sequence number in almost every command of most programs for the purpose of documentation. However, we recommend in your own Custom Macro programs, that you only include sequence numbers when they are required as statement labels. This will make them stand out. It will be easier to find a statement label being referenced. It will also reduce the length of your Custom Macro programs, conserving some CNC memory.

To further simplify statement label usage when they are required, use small sequence number (like N1, N2, N3, etc.) near the beginning of the Custom macro program. Use sequence numbers around fifty in the middle of the program. Use sequence numbers close to 99 near the end of the program. And if possible, keep sequence numbers in ascending order. This way, when you need to find a given statement label in a lengthy Custom Macro program, you will know where to look.

Unconditional branching

As the name implies, the unconditional branching Custom Macro command will cause the machine to branch (jump) from one location in the program to another location in the same program. The location to which branching is done must be marked with a statement label (sequence number). You specify an unconditional branching statement in Custom Macro with a GOTO statement. Here is the syntax for the GOTO statement:

.
.
GOTO 50 (Branch to statement label N50. Note no N word)
.
.
.
N50 G00 X . . . (Statement label N50)

Conditional branching

Just like conditional branching in any computer programming language, conditional branching in Custom Macro provides a decision-making capability. The Custom Macro programmer can make the program perform tests. The program can be executed differently based upon test results.

The purposes for testing are extremely varied and we cannot over-stress the importance of conditional branching for testing purposes. You can test just about anything within your Custom Macro program. And based upon the results of your test, you can make the program execute differently. This feature has fantastic, nearly limitless, implications.

Syntax for conditional branching statements

Before we show specific applications for conditional branching, you must understand its syntax. Custom Macro uses an IF statement for conditional branching. And Custom Macro as it is equipped on current FANUC controls allows great flexibility regarding how IF statements can be written.

Older versions of Custom Macro (like Custom Macro B), on the other hand, do not support all of the variations we show. You may be limited to the most basic (universal) format for IF statements. Rest assured that even though it is more difficult to create the needed conditional branching (IF) statements with only the most universal IF statement, all conditional branching applications can be handled.

The conditional expression

All IF statements require at least one *conditional expression*. The result of the conditional expression/s will be either true or false. If the result is true, the Custom Macro command to the right of the IF statement is executed. If the result is false, the command after the IF statement is executed. This is how Custom Macro programs can make decisions. With each conditional branching statement, the result will cause the program to do one of two things, again: execute what is to the right of the IF statement or execute the next command.

Conditional expressions require *conditional operators*. There are six conditional operators in Custom Macro:

1) Less than (LT)
2) Less than or equal to (LE)
3) Equal to (EQ)
4) Greater than (GT)
5) Greater than or equal to (GE)
6) Not equal to (NE)

The six conditional operators allow great flexibility in the method by which you develop conditional branching commands. In most cases, the same test can be written in several ways (as many as six ways, corresponding to the number of available logical operators). While this flexibility is good, it can sometimes be confusing, especially when you are testing with the not equal to (NE) conditional operator. Always remember that the test within the conditional branching statement is to determine whether the conditional expression is true or false. When testing against a negative condition, it is easy to make evaluation mistakes.

The universal conditional branching command format

Newer FANUC control models allow much more flexibility with conditional branching than older models. You must consult your FANUC manual to determine whether newer features are available. You can rest assured that all FANUC controls allow this universal conditional branching command format.

As stated, if the conditional branching statement is true, the command to the right of the statement is executed. With the universal conditional branching command format, you are limited to placing a GOTO statement to the right of the IF statement.

Consider this example: You wish to write a grooving Custom Macro program for your turning center that can use right hand or left hand tooling. You wish to include a special argument in the call statement, say H for hand, to specify the hand of tooling style. Your documentation criteria states that if H is set to one, right hand tooling is being used. If H is set to zero (or any value other than one), left hand tooling is being used.

If, of course, right hand tooling is being used, the spindle must be running in a forward (M03) direction. If left hand tooling is being used, the spindle must be running in a reverse direction (M04). Here is a portion of the Custom Macro program that performs the appropriate test and starts the spindle in the correct direction. Letter-address H is represented by local variable #11 in the Custom Macro program. Study these commands that will be included in the Custom Macro program when the spindle is started:

```
.
.
.
IF [#11 EQ 1.0] GOTO 5
M04
GOTO 6
N5 M03
N6 . . .
.
.
.
```


Note that brackets ([]) must be used to enclose the conditional expression. This particular conditional branching statement uses the equal-to (EQ) logical operator, which makes it quite easy to evaluate the results of the test (true or false). If a person using this Custom Macro includes H1.0 in the call statement to the Custom Macro (the G65 command), the machine will evaluate the conditional branching statement as true. In this case, the GOTO statement to the right of the IF statement will be executed. In N5, the M03 will start the spindle in the forward direction (as is required for right hand tools).

On the other hand, if the person using this Custom Macro program sets the value of H to zero (or anything other than one), the IF statement's conditional expression will be evaluated as false. Now the command following the IF statement will be executed, starting the spindle in the reverse direction (as is required for left hand tools).

Note the need for the GOTO 6 statement. If the conditional expression is evaluated as false, and the spindle starts in the reverse direction, we need the CNC to skip the M03 command (the other possibility of the IF statement). This GOTO statement keeps the CNC from starting the spindle in the forward direction.

CNC related features (system variables)

Most CNC-related features of Custom Macro programming are handled with system variables. Like any other variable in Custom Macro, one way to reference them is with a pound-sign and a number (like #2001). But unlike other variables that can be used to represent just about anything, each system variable has a fixed function, and it can only be used to reference its related function.

System variables allow you to access many machine functions right from within the Custom Macro program. Offsets, axis position, alarm generation, and current machine states are among the many accessible machine functions.

Access to tool offsets

Custom Macro provides access to all tool offset registers within a FANUC control. Though the specific system variable number/name may vary among control models, rest assured that you have access to every tool offset register. You have the ability to read and write values to and from tool offset registers, meaning you can access their values for use within a Custom Macro program as well as enter them from within a program.

You probably already know that most controls give you the ability to write to the tool offset registers using standard G-code. G10 is used for this purpose. However, there is no standard CNC function (without Custom Macro) that allows you to *read* the value stored within a tool offset register and use it within your CNC program. This ability to read offset register values has some pretty important implications. But before we show applications, we will first show the related system variables and their syntax.

No.	GEOM (H) Number	GEOM (H) Name	WEAR (H) Number	WEAR (H) Name	GEOM (D) Number	GEOM (D) Name	WEAR (D) Number	WEAR (D) Name
1	#2001	[#_OFSHG[1]]	#2201	[#_OFSHW[1]]	#2401	[#_OFSDG[1]]	#2601	[#_OFSDW[1]]
2	#2002	[#_OFSHG[2]]	#2202	[#_OFSHW[2]]	#2402	[#_OFSDG[2]]	#2602	[#_OFSDW[2]]
.								

.								
199	#2199	[#_OFSHG[199]]	#2399	[#_OFSHW[199]]	#2599	[#_OFSDG[199]]	#2799	[#_OFSDW[199]]
200	#2200	[#_OFSHG[200]]	#2400	[#_OFSHW[200]]	#2600	[#_OFSDG[200]]	#2800	[#_OFSDW[200]]

System variable numbering/naming for a 0iD control when there are four registers per offset and 200 total offsets. (Tool Compensation Memory Type C - 200 offsets)

Access to current position

This CNC related feature of Custom Macro gives you the ability to determine the position of at any axis at any time (during the execution of a program). You can access current position in four ways:

1) Relative to the program zero point (absolute position **without** compensation)
2) Relative to the program zero point (absolute position **with** compensation)
3) Relative to the machine's reference (zero return) position (machine position), with compensation
4) Absolute position after skip signal of G31 (used with touch probes), with compensation

We show the related system variables and their syntax before showing implications of when position access can be helpful.

Unlike system variables related to offset access that provide read and write access, position-related system variable only provide read capabilities. You cannot write to them.

For a three axis machining center

Here are the system variables for a three axis machining center:

Current absolute position not modified by instated compensation
#5001 or [#_ABSIO[1]] - X axis position in current coordinate system

#5002 or [#_ABSIO[2]] - Y axis position in current coordinate system

#5003 or [#_ABSIO[3]] - Z axis position in current coordinate system

Current absolute position modified by instated compensation
#5041 or [#_ABSOT[1]] - X axis position in current coordinate system

#5042 or [#_ABSOT[2]] - Y axis position in current coordinate system

#5043 or [#_ABSOT[2]] - Z axis position in current coordinate system

Current machine position:
#5021 or [#_ABSMT[1]] - X position relative to the machine’s reference (zero return) position

#5022 or [#_ABSMT[2]] - Y position relative to the machine’s reference (zero return) position

#5023 or [#_ABSMT[3]] - Z position relative to the machine’s reference (zero return) position

Skip signal position after touch-probe contact (absolute position):
#5061 or [#_ABSKP[1]] - X position in current coordinate system at the instant of probe stylus contact

#5062 or [#_ABSKP[2]] - Y position in current coordinate system at the instant of probe stylus contact

#5063 or [#_ABSKP[3]] - Z position in current coordinate system at the instant of probe stylus contact

Alarm generation

Error trapping is the process of finding conditions that require the stoppage of a program's execution before something bad can happen. During our discussion of conditional branching in lesson four, we introduce the alarm generation function of Custom Macro:

There are the two ways to specify the alarm generating system variable:

1) #3000
2) [#_ALM]

If the control executes a #3000 (or a [#_ALM]) command, an alarm will be sounded. Like any program storage (PS) alarm, the alarm generated by the alarm-generating system variable will put the machine in a true alarm state. You must press the RESET button to clear the alarm. Here is the syntax for the alarm-generating system variable shown with both the number and the name:

#3000 = 100 (ARGUMENT MISSING IN CALL)

or

[#_ALM] = 100 (ARGUMENT MISSING IN CALL)

Again, if the control executes either of these commands, it will generate an alarm. The value (100) is an alarm number (of your choosing) and can range from 0 to 200 (some FANUC control models allow other alarm numbers to be specified). The message in parentheses (all upper case) is the alarm documenting message. It must be 60 characters and numbers or less. If this command is executed, the display will switch to the alarm page and this alarm message will appear on the control screen:

MC-100 ARGUMENT MISSING IN CALL

With an understanding of the alarm-generating system variable, let's look at a few more examples of alarm generation (unrelated to error trapping). Then we will show how to minimize program execution time required for error-trapping.

Stop with message

The stop with message system variable is similar to the alarm-generating system variable. But instead of placing the machine in alarm state (requiring the cancelation of the cycle), the stop with message system variable places the machine in program-stop state (just like an M00 word). As with M00, it is possible to make the machine continue executing the program by pressing the CYCLE START button.

There are the two ways to specify the stop with message system variable:

1) #3006
2) [#_MSGSTP]

If the control executes a #3006 (or a [#_MSGSTP]) command, it will place the machine in program-stop state. You can reactivate the cycle by pressing the CYCLE START button, continuing from the point of the stop with message command.

Here is the syntax for the alarm-generating system variable shown with both the number and the name. Notice how similar it is to the alarm generating system variable:

#3006 = 100 (TURN PART AROUND IN CHUCK)

or

[#_MSGSTP] = 100 (TURN PART AROUND IN CHUCK)

Again, if the control executes either of these commands, it will place the machine in program-stop state. The value (100) is a message-number of your choosing and can range from 0 to 200 (some FANUC control models allow other message numbers to be specified). The message in parentheses (all upper case) is the documenting message. It must be 60 characters and numbers or less. If this command is executed, the display will switch to the message page and this message will appear on the control screen:

MS-100 TURN PART AROUND IN CHUCK

Suppressing single block and waiting for auxiliary functions

System variable #3003 or [#_CNTL1] allows you to take control of two important machine functions, single block and whether or not M-codes are awaited. If #3003 or [#_CNTL1] is set to 0 (the initialized state), single block is not suppressed and the machine will wait for auxiliary functions (M-codes) to be completed before going on to the next command.

If #3003 or [#_CNTL1] is set to 1, single block will be suppressed and the control will wait for the completion of M-codes to continue with the program. If #3003 or [#_CNTL1] is set to 2, single block is not suppressed, and the control will not wait for the completion of auxiliary functions. If set to 3, single block is suppressed and M-codes will be awaited.

Newer versions of Custom Macro additionally allow you to control these functions separately. [#_M_SBK] controls single block. Single block can be suppressed by setting this system variable to one (zero for not suppressed).

[#_M_FIN] controls whether or not M-codes are awaited (set to zero for awaited, one for not awaited).

Access to timer and clock (date/time)

Another set of system variables gives you the ability to time events and determine current date and time from within Custom Macro programs. The next table shows the related system variable numbers and names.

Type	Number	Name
Milliseconds	#3001	[#_CLOCK1]
Hours	#3002	[#_CLOCK2]
Date	#3011	[#_DATE]

Time of day	#3012	[#_TIME]

System variables related to times and date/time.

Timing events with the timers

The clock and times are always running. If you will be timing an event with either of the timers, you must first reset the timer (to zero). Here are commands that reset the milliseconds timer:

#3001 = 0 (Reset milliseconds timer)

or

[#_CLOCK1] = 0 (Reset milliseconds timer)

Here are the commands that reset the hours timer:

#3002 = 0 (Reset hours timer)

or

[#_CLOCK2] = 0 (Reset hours timer)

Modal G-code access for machining centers

The next table shows the system variables related to modal G-code information in buffered commands on machining centers (FANUC 0iD). Reference the Custom Macro section of the FANUC manual to find system variable names and numbers for active and interrupted commands.

Modal information	**Number**	**Name**
G-code group 1 (G01-G03)	#4001	[#_BUFG[1]]
G-code group 2 (G17-G19)	#4002	[#_BUFG[2]]
G-code group 3 (G90-G91)	#4003	[#_BUFG[3]]
G-code group 4 (G22-G23)	#4004	[#_BUFG[4]]
G-code group 5 (G93-G95)	#4005	[#_BUFG[5]]
G-code group 6 (G20-G21)	#4006	[#_BUFG[6]]
G-code group 7 (G40-G42	#4007	[#_BUFG[7]]
G-code group 8 (G43-G44)	#4008	[#_BUFG[8]]
G-code group 9 (G80, G81, and all canned cycles for drilling)	#4009	[#_BUFG[9]]
G-code group 10 (G98-G99)	#4010	[#_BUFG[10]]
G-code group 11 (G50-G51)	#4011	[#_BUFG[11]]
G-code group 12 (G66-G67)	#4012	[#_BUFG[12]]
G-code group 13 (G96-G97)	#4013	[#_BUFG[11]]
G-code group 14 (G54-G59)	#4014	[#_BUFG[14]]

G-code group 15 (G61-G64)	#4015	[#_BUFG[15]]
G-code group 16 (G68-G69)	#4016	[#_BUFG[16]]
G-code group 17 (G15-G16)	#4017	[#_BUFG[17]]
G-code group 19 (G40.1-G42.1)	#4019	[#_BUFG[19]]
G-code group 20 (G160-G161)	#4020	[#_BUFG[20]]
G-code group 22 (G50.1-G51.1)	#4021	[#_BUFG[21]]
G-code group 34 (G80.1-G81.1)	#4022	[#_BUFG[22]]

System variables related to buffered modal G-code states.

The value contained in each G-code related system variable is the G-code number for the instated G-code. For group number one, for example, if linear interpolation mode (G01) is currently instated, the value of #4001 or [#_BUFG[1]] will be 1 (the numerical value of G01)

Productivity improving suggestions

Here are some specific suggestions for improving CNC productivity with parametric programming.

1: Reduce program verification and optimizing efforts

Simply incorporating part-family and user created canned cycle applications will save program verification and optimizing time. Since these programs work much like canned cycles (and when was the last time when you questioned whether or not a built-in canned cycle "worked" [like G81 for drilling]), once verified, you can rest assured that your parametric programs will run properly.

2: Sizing in the first workpiece on a Swiss-type (sliding headstock) lathe

Trial machining involves taking each tool in the program and *making* it cut to size prior to going on to the next tool. For each tool that machines a critical surface (with a tight tolerance), this involves 1) making an adjustment (usually to an offset) to force excess stock to be left on the machined surfaces, 2) machining under the influence of this adjustment, 3) stopping the machine and measuring after the tool has machined, 4) readjusting based upon the measurement, and 5) rerunning the tool. If these techniques are used for each critical tool, the first workpiece machined will be a good one.

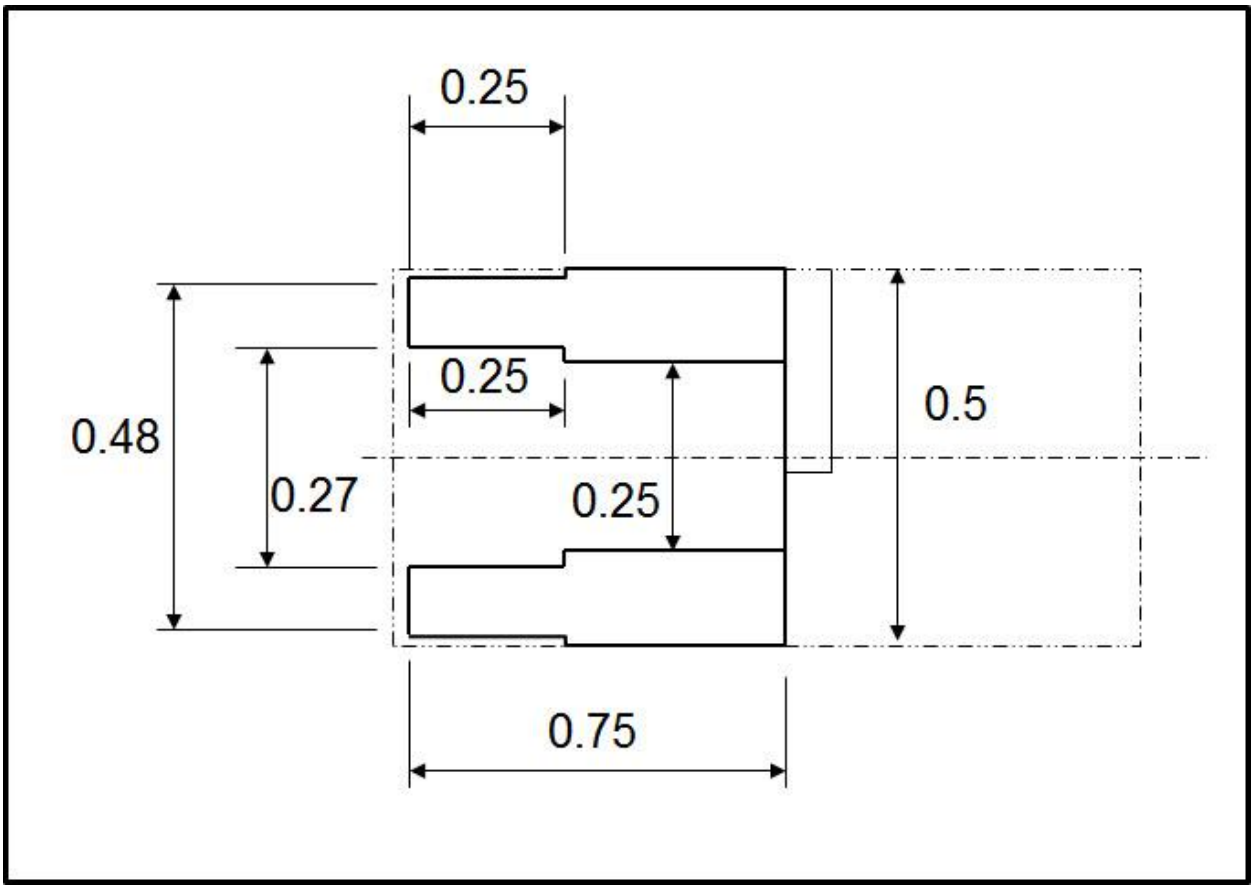
There is at least one type of turning center that is not appropriate for trial machining. By their very nature, sliding headstock – also called Swiss-type – turning centers almost never allow cutting tools to remachine the workpiece. Additionally, many of these machines (at least the ones I've seen) require relatively crude techniques for assigning program zero, meaning each cutting tool will almost surely be off by a few thousandths of an inch after initial setup.

Traditionally, the setup person must run the entire workpiece (at least the entire operation for one of the spindles) before they can begin measuring to determine what adjustments must be made. For jobs that have more than a couple of tools, this can be extremely difficult and very confusing (for example, when a bored hole isn't going deep enough, do you go deeper with the boring bar or move the facing tool out?). What a setup person does with one adjustment will surely affect

other workpiece attributes. It is not uncommon for even an experienced setup person to run several workpieces before they get it right. Novices can take hours doing so.

While it may not be possible to trial machine, it is possible to run a *test workpiece*. The test workpiece can be very simple, making it obvious as to which tool has machined a given surface. Measurements will be taken with standard measuring tools. And if custom macro techniques are used, getting ready to run the test workpiece will take no more than a minute or so. While I won't show the entire application in this short article, I'll show enough for you to get the idea. If you've had any experience with custom macro B, you should be able to take it from there.

The drawing shows the test workpieces – at least as it will be in our example.



While we've shown a dimensioned workpiece, every dimension may change from job to job, so we'll need an easy way of specifying the related sizes. Additionally, we're setting up for five cutting tools (facing tool, turning tool, drill, boring bar, and cut off tool) but there may be times when internal tools are not required in the job. Conversely, there may be times when additional tools are required (like grooving tools, threading tools, and all kinds of live tools). With a relatively simple main program, you can easily define the test workpiece and operations you want to perform. Here is an example:

```
O0001 (Main program)
N1 G65 P6000 D0.5 Z1.0 (define workpiece)
N2 G65 P6001 T1.0 S1.0 (facing tool)
N3 G65 P6001 T2.0 S2.0 D0.48 Z0.25 (turning tool)
```


N4 G65 P6001 T5.0 S7.0 Z.75 (0.25 drill)
N5 G65 P6001 T6.0 S8.0 D0.27 Z0.25 (boring bar)
N6 G65 P6001 T7.0 S3.0 D-0.01 (cut-off tool)
N7 M30 (end of program)

Notice that two custom macro programs will be required, O6000 (the part definition custom macro) and O6001 (the cutting operations custom macro). Here is what the variables mean in each:

O6000 variables:

D: Test part stock diameter
Z: Test part length

O6001 variables:

T: tool type
1: facing tool
2: turning tool
3: external threading tool
4: center drill (stop after this tool)
5: drill
6: boring bar
7: cut-off tool
8: threading tool
9: internal threading tool
10: end mill (cross tool)
11: cross drill (cross tool)
S: Station number
Z: Z depth of machining for certain tools
D: diameter to machine

Program O0001 (the main program) will be modified for each job. In the line N1, the part is being defined as 0.5 in in diameter (stock diameter) and 0.75 in long.

In line N2, the facing operation is being done. Notice the code number for tool type. Type 1 (in our example) is a facing tool. The S word specifies that it is in station one. No additional variables are required, since the previous command specifies the stock diameter and this tool will always face to Z0 and to the center of the workpiece.

In line N3, the turning is being done. Additional variables include D for the diameter to turn (0.48 in our case) and Z for the length of the turned diameter (0.25 in our case).

In line N4, drilling is being done. The Z in this command is the depth to which to drill (not including drill point).

Line N5 does the boring – with the same variables as were used for turning (D and Z).

Finally, line N6 does the cutoff. D specifies the diameter to which the cutoff tool will go.

While we're not showing programs O6000 and O6001, you should be able to see how quickly and easily a setup person can modify program O0001 to make it ready to machine any size test workpiece. Once this workpiece is completed, they will measure the various attributes and

adjust all offsets accordingly. And if the test part is machined perfectly, so will the actual workpiece that uses these tools.

One last point. We've only shown the workpiece as it has been machined in the main spindle. If the machine has a sub-spindle (as most Swiss-type turning centers do), similar techniques can be used after the actual workpiece has been machined in the main spindle.

More on the test-part program

Since it is almost always necessary to run an entire workpiece before the setup person can begin taking measurements and making sizing adjustments, it can be very difficult to figure out what must be done with offsets in order to get workpiece attributes to size. This problem is further compounded by the sheer number of cutting tools that can be used on Swiss-type machines. It can be difficult to determine, for example, if one tool is machining too deep or another is machining too shallow.

This complexity often results in the need to run many workpieces in order to get one that passes inspection. This can be very frustrating, time consuming, and wasteful. Material costs may not be substantial (unless you run exotic materials), but the unpredictable amount of time it takes to get a part to pass inspection can be devastating, especially when you consider include the time it takes to *inspect* each workpiece before adjustments can be made.

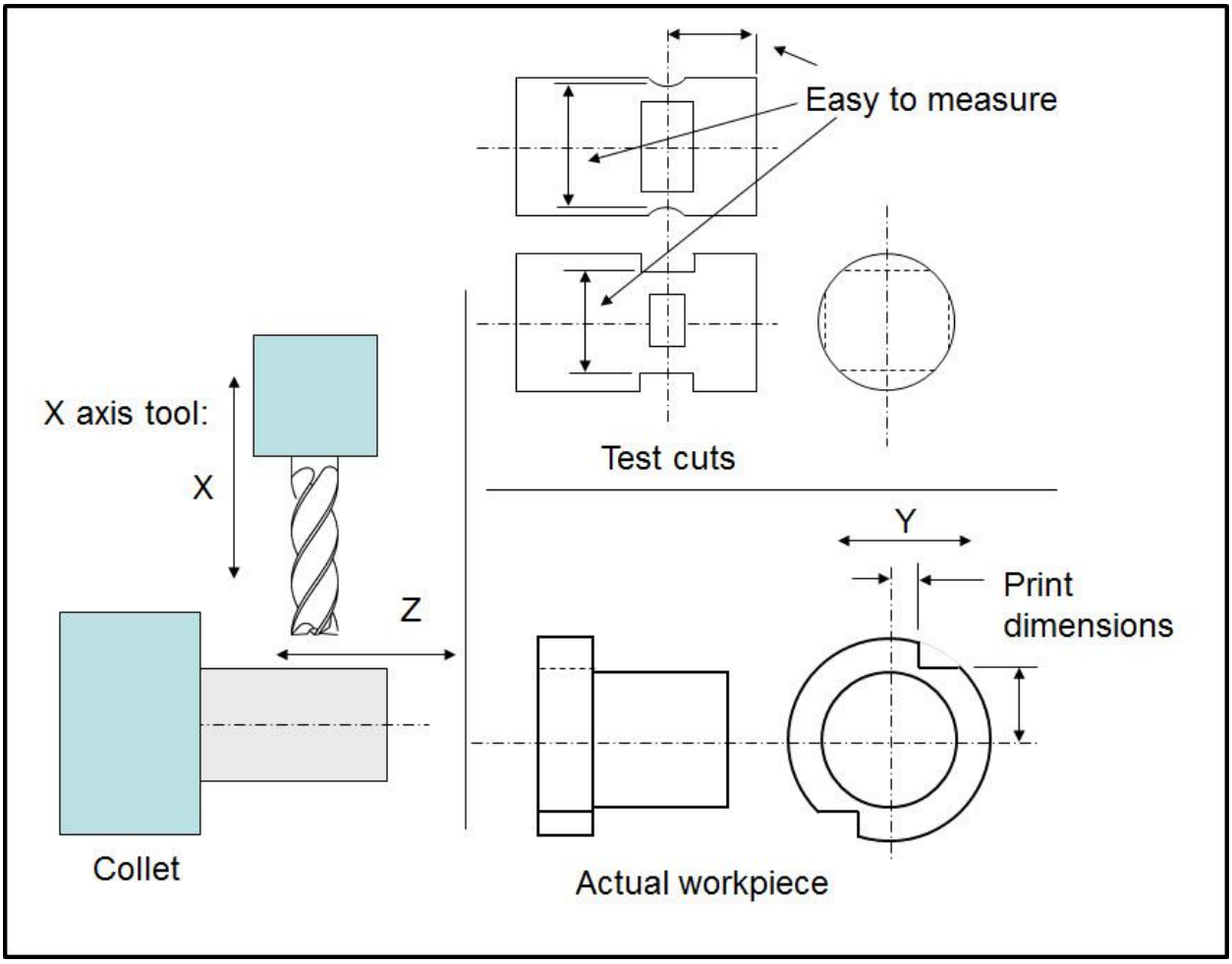
My suggestion was to develop a *universal test part program* that will be used during every setup to size in all of the cutting tools. The premise is that if a cutting tool is machining an attribute correctly on the test part, it will also machine actual workpiece attributes correctly. This, combined with the idea that each cutting tool will machine independent surfaces on the test part that are easy to measure, make's the test part program easy to use.

Frankly speaking, this idea been met with mixed reviews. Many programmers don't want to devote the time to develop the test part program. And many setup people feel that it's just as difficult to make a good test part as it is to make a good workpiece. While I still contend that the test part program will simplify and speed the process of running the first good workpiece, it's not of any value if no one wants it.

You still have to deal with the problems of frustration, lost time, wasted material, and unpredictable setup times. If your setup people are machining countless "practice workpieces" (I heard one manager say *buckets of them*) before they get one to pass inspection, I urge you to rethink your current methods.

My second suggestion is similar to my first. But instead of developing an entire test part program to handle all of the cutting tools the machine can use, concentrate on the most difficult ones. Using custom macro B, you can develop a series of test cuts for those cutting tools that give your setup people the most trouble. If you are doubtful about whether this method will work, start with *just one tool*. Once you're sure that this technique simplifies the needed adjustments as much as I say it will, you can add more.

Here's an example. For many setup people, one troublesome type of tool is an end mill mounted parallel to the X axis. This kind of tool must often machine in the Y axis. With many machines, it can be difficult to perfectly center the end mill along the Y axis during setup.



The test-cut custom macro program will include commands to machine two sets milled surfaces – one flat and one rounded. The example program below machines them 0.050 smaller than the bar diameter. A comparator (or other measuring devices) can be used to easily determine if the Y axis is centered (and if not, how much it is off), if the tool is machining to the correct depth in X (bar diameter minus 0.05 inch), and if the tool is machining in the correct Z position (0.25 from part end to center of slot). With these values, the setup person can perfectly align the tool with offset adjustments and the Y axis program zero assignment before an actual part is run.

Here are the related programs. Notice that I'm only showing motions – certain machine functions, like live tooling mode and main spindle mode selections are omitted.

The main program begins with some notes to point the setup person to the right place in the program. The cross end mill portion of the program begins on line N005. This will allow you to use only one program for all test cutting operations. Under line N005, there are some variables for the setup person to specify. The program is then run from line N005 (in our example). An M30 ends each set of test cuts.

The program also uses your facing tool and cutoff tools. These tool are commonly kept in the machine on a permanent basis, so there won't be anything additional required to set them up. Each segment of the program will face the part, perform the test cuts, and then cut off the part (about an inch long).


```
O0001 (TEST CUTTING)
(TEST CUTS INCLUDED IN THIS PROGRAM)
(N005: CROSS END MILL)
(N010: CROSS DRILL)
(N015: THREAD MILL)
N005 (CROSS END MILL)
(SET VARIABLES)
#100 =3 (TOOL STATION NUMBER)
#101 =0.25 (END MILL DIA)
#102 =0.5 (BAR DIAMETER)
#103 =0.25 (Z POSITION FOR SLOTS)
#104 =900 (RPM)
#105 =2.75 (IPM)
#106 =1 (FACING TOOL STATION)
#107 =4 (CUTOFF TOOL STATNION)
(DISTANCE ACROSS FLATS: BAR DIA MINUS 0.05)
(DISTANCE BETWEEN ROUNDS: BAR DIA MINUS 0.05)
(SLOTS CENTERED AT Z POSITION)
(MACHINING)
M98 P1000 (SUBPROGRAM FOR FACING)
(MUST SELECT LIVE TOOLING MODE - M CODE?)
T[#100*100+#100]
C0 (START ON C ZERO SIDE)
G98S#104 (SPEED FOR LIVE TOOL)
G0Y[#102/2+.5+0.1]Z-#103 (LARGEST ENDMILL 0.5)
X-0.25
G1Y[#102/2-0.025+#101/2]F#105 (MILL ONE ROUND)
G0Y[#102/2+.5+0.1]
G0X[#102-0.05]
G1Y-[#102/2+.5+0.1] F#105 (MILL ONE FLAT)
G0X-0.25
G1Y-[#102/2-0.025+#101/2] (MILL SECOND ROUND)
G0Y-[#102/2+.5+0.1]
X[#102+.2]
C180
X[#102-0.05]
G1Y[#102/2+.5+0.1] (MILL SECOND FLAT)
G0 X[#102+.2]
Y0
M98 P1001 (SUBPROGRAM FOR CUTOFF)
M30
N010 (CROSS DRILL)
.
.
```



```
M30
N015 (THREAD MILL
.
.
M30

O1000 (FACING SUBPROGRAM)
T[#105+#105*100]
(MUST SELECT MAIN SPINDLE MODE - M CODE?)
G97S1000 M03
G0X[#102+.1]Z0
G1X-0.06F0.004
G0X[#102+0.1]
X1.0 Z0.1
M99

O1001 (CUTOFF SUBPROGRAM)
T[#106+#106*100]
(MUST SELECT MAIN SPINDLE MODE - M CODE?)
G97S1000 M03
G0X[#102+.2]Z-1.0
G1X-0.125F0.004
G0X[#102+0.2]
X1.0 Z0.1
M99
```

3: Use an edge finder like a spindle probe

Just as we can access machine position for purpose of measuring tool lengths, so can we do so for the purpose of measuring the location of the program zero point. For machining center applications, nothing beats a spindle touch-probe for this function. However, if you do not have a spindle probe equipped on your machine, these functions of Custom Macro allow you to use a standard conductivity-type edge finder (which is easier to use than a wiggler type edge finder) to deal with just about any spindle touch-probe application. By *conductivity-type* edge finder, we mean an edge finder that incorporates a light that will illuminate the instant it comes into contact with a metallic surface.

The only limitation is that the task of touching the edge finder to each surface must be performed manually. This lets you develop a series of very helpful utility Custom Macro program to facilitate the measurement of program zero based upon your own specific program zero point specifications. Again, anything that can be done with a spindle touch-probe can be done with an edge finder. Understanding how to program an edge finder in this fashion is a great way to begin learning how to program a spindle touch-probe. It incorporates many of the same functions.

A corner pickup Custom Macro program

This example will locate the lower-left corner of a rectangular workpiece in the X and Y axes. It will also locate the top surface of the workpiece in the Z axis. It will then place the measured values into workpiece coordinate system offset registers.

This Custom Macro program assumes the use of a conductivity-type edge finder. Since this program will probably be used on a pretty regular basis, we will utilize two system constants. The radius of the edge finder (0.100 for our example) is stored in permanent common variable #500 and the length of this edge finder is stored in #501. Once each surface location is determined, the distance from the machine's reference point to the program zero surface will be stored in workpiece coordinate system offset number one. Prior to running this program, the setup person will manually position the edge finder about 0.5 in in each direction from the corner to be picked up (0.5 to the left of the corner in X, 0.5 below the corner in Y, and 0.5 above the corner in Z).

```
O0015 (Program to pick up lower left corner)
G49 (Cancel tool length compensation)
G91 G01 Y0.75 Z-0.75 F30.0 (Move to first touch position)
#3006 = 101 (TOUCH LEFT SIDE IN X)
[#_WZG54[1]] = [#_ABSMT[1]] + #500 (Set X of workpiece coordinate system offset number one)
G91 G01 X-0.2 (Move away in X)
Y-0.75 (Move down in Y)
X0.55 (Move to second touch position)
#3006 = 102 (TOUCH BOTTOM SURFACE IN Y)
[#_WZG54[2]] = [#_ABSMT[2]] + #500 (Set Y of workpiece coordinate system offset number one)
G91 Y-0.2 (Move away in Y)
Z0.75 (Move above top surface in Z)
Y0.55 (Move to third touch position)
#3006 = 103 (TOUCH TOP SURFACE IN Z)
[#_WZG54[3]] = [#_ABSMT[3]] - #501 (Set Z of workpiece coordinate system offset number one)
G91 G01 Z0.5 (Move away in Z)
G54 X0 Y0 (Instate coordinate system, move over corner)
M30 (End of program)
```

Using similar techniques, you can develop any pickup routine that a true probing system is designed to do (any other corner, center of a hole, boss or pin, center of a slot, etc.). And again, if you understand this program, you are well on your way to being able to program a true spindle touch-probe.

4: Streamline tool length measurements on machining centers

While many companies have moved the setup-related task of measuring tool lengths for machining center setups off line (they measure tool lengths for upcoming jobs during the current production run), there are many companies that still measure tool lengths right on the machine during setup. Maybe there is no one available to measure tool lengths during production runs. Or maybe lot sizes are so small and cycle times so short that there is no time to set up and measure tool lengths off line.

Regardless of the reason, if tool lengths must be measured on the machine during setup, you must make it as easy as possible for your setup people to perform the tool length measuring function. Tool length measuring probes make this task about as easy as it can be. However, many companies do not have tool length measuring probes and must rely solely on the setup person to perform tedious and error prone measurements for each tool while the machine is down between production runs. In a sense, the machine is being used as a very expensive height gauge.

If your setup people must manually measure cutting tool lengths on the machine and enter each tool length value into the corresponding tool length compensation offset, you can dramatically facilitate this task with a utility Custom Macro program. This program will take advantage of Custom Macro's ability to access the machine's current position and the ability to enter values into tool length compensation offset registers.

To keep our example simple to follow, we will make some assumptions.

1) We assume that you are using the length of each tool (distance from tool tip to spindle nose) as the tool length compensation offset value.
2) We assume that the offset number for each tool corresponds to the tool station number (offset one for station number one, offset two for station number two, and so on).
3) We assume that all tools being measured will be assembled and loaded into the machine's tool magazine prior to running this Custom Macro program.
4) We assume that the setup person uses a three inch block to touch each tool (possibly the three inch side of a 1-2-3 block). This block is placed directly on the table top prior to tool measurement.
5) We assume that your setup people will always have a series of tools in consecutive tool stations to measure (like from tool station one through tool station ten).

Prior to using this program for the first time, the distance from the table top to the machine spindle nose at the machine's Z axis reference (zero return) position must be measured. This value will be placed in permanent common variable #500.

Here is the Custom Macro program:

```
O0014 (Program number)
#100 = 1 (First tool to measure)
#101 = 15 (Last tool to measure)
#102 = #100 (Initialize counter for loop)
WHILE[#102 LE #101] DO 1 (Test if finished)
G91 G28 Z0 M19 (Return to tool change position, orient spindle)
T#102 M06 (Place current tool in spindle)
#3006 = 101 (TOUCH TOOL TIP TO BLOCK) (Stop with message command)
#[2000 + #102] = #500 - 3 - ABS[#5023] (Set offset)
G91 G01 Z.5 F30. (Move away from block)
#102 = #102 + 1 (Step counter)
END 1 (Go back to WHILE statement)
M30 (End of loop and program)
```

While you should be able to follow most of this Custom Macro program, it does introduce some new functions. First of all, we are using #500 as a system constant (system constants are introduced in lesson two). This value (the distance from the table top to the spindle nose at the

machine's reference position) will be measured and entered but once. The CNC will retain the value of #500 from day to day (it is a permanent common variable) and it will be available whenever this tool length measuring Custom Macro program is used.

The set up person will begin by editing the first two commands, specifying first tool station number and last tool station number for the tools to be measured. They will then place the three inch block on the table under the spindle nose and activate the program. After the loop is initialized and the first execution of the WHILE is found to be true, the G28 command will send the machine to its tool change position. The first tool to be measured will be placed in the spindle and the program will halt due to the stop with message command.

Though it has not been introduced yet, system variable #3006 is very similar to the #3000 alarm generating system variable. However, instead of placing the machine in an alarm state, it simply places the machine in a program stop state, just like M00. But unlike M00, the display screen will automatically change and show the message TOUCH TOOL TIP TO BLOCK on the display screen. An M00 command would work just as well as #3006, but the setup person must be monitoring the program page of the display screen in order to see the message.

At this point, the setup person places the machine in a manual mode (jog, manual, hand-wheel, etc.) and manually brings the tool tip for the first tool down to touch the three inch block. With the tool tip touching the block, they place the machine back in the automatic mode and reactivate the cycle. Consider this command.

```
#[2000 + #102] = #500 - 3 - ABS[#5023]
```

It calculates the tool length compensation value (the tool length from tool tip to spindle nose) and places it into the tool length compensation geometry offset register corresponding to the current tool station number. Remember that system variable #5023 is the distance in Z from the machine's zero return position to its current position. Since this value will be negative (in our case), absolute value function (ABS) is being used to reverse its sign. When this value plus an additional three inches (allowing for the block height) is subtracted from permanent common variable #500, the result will be the tool's length. This value is then placed into the current offset specified with #[2000 + #102].

The program will then back the tool off slightly, step the counter to get to the next tool, and go back to the WHILE statement. This process is repeated for each tool to be measured.

Do you want to learn how to program a touch probe?
The techniques used in the previous example, along with those we are about to show, closely resembles what must be done when programming touch probes. The only real difference is related to making the touch probe come into contact with a surface. Our Custom Macros require that setup people manually cause the contact, meaning lots of manual intervention. With touch probes, contacting is automatic (no manual intervention).

5: Take calculations out of sizing adjustments

Almost every offset entry your operators make requires some kind of calculation to be made before the offset value can be entered. For example, say the target diameter on a workpiece being machined with a turning tool is 3.2342 and its tolerance is plus or minus 0.002 (high limit is 3.23**5**2). After machining with the finish turning tool, the operator finds that the diameter

being turned is 3.2351. Though the workpiece is still within its tolerance band, the operator will need to adjust the offset to bring the workpiece back to its target value.

In order to do so, the operator must subtract the target dimension from the actual dimension (3.2351 - 3.2342 in this case) to come up with the amount of offset change (0.0009 in our case). Though our example involves simple subtraction, even simple calculations do take time, even for experienced operators and opens the door to making mistakes. Offset adjustments of this nature, of course, are made countless times (especially on finishing tools) during a production run.

> **IMPORTANT!** Any time you see your operators using a calculator before they enter an offset value should be taken as a signal that you can do something to help them and minimize offset setting time.

Custom Macro gives you all the tools you need to simplify offset adjustments. You have access to offsets from within programs. You can make calculations to determine the value by which an offset must be adjusted. And you can have a Custom Macro program change an offset value by the appropriate amount based upon the result of a calculation. (By the way, probing systems commonly perform these functions when they are used for in-process gaging applications.)

Given our previous example, the target diameter to be turned on a turning center is 3.2342 inch. Say that it is tool number five is the turning tool that machines this diameter. After machining the operator will normally adjust tool (wear) offset number five to input any discrepancy (by 0.0009 in our previous example). Instead of forcing the operator to calculate the deviating amount (and polarity), wouldn't it be easier for the operator to enter the current machined diameter (3.2351 in our case)?

How it works

Our given technique will allow the operator to do just that, which eliminates the need for a calculation prior to offset entry. But instead of making an adjustment in wear offset number five, we'll pick a secondary offset number in which to enter the measured value. Since most turrets can hold no more than twelve tools, we'll simply add twenty to the tool station number to come up with the secondary offset number. For tool number five, we'll use wear offset number twenty-five.

The Custom Macro program will check to see if there is any value in the secondary offset register (other than zero). If there is, the operator has entered the actual size of the workpiece that is deviating from its target size. In this case, the Custom Macro program will make the calculation to determine the offset adjustment value (just as the operator is currently doing manually) and adjust the primary wear offset accordingly (offset number five in our case).

Minimizing repeated commands

If you wish to use this technique for all tools which require frequent offset adjustments (commonly your finish turning tools, threading tools, and grooving tools), you will have quite a few redundant commands in your main program to perform the secondary offset testing and calculations. For high volume work, this may be okay. But the more programs you need to run, the more inconvenient it would be to incorporate these redundant commands right in your main program. To minimize the number required commands, our technique uses a separate program (the Custom Macro program) for this purpose.

In the main program, place this command at the beginning of those tools for which you want to simplify sizing adjustments (typically only the finishing tools). This command must come before the turret index command.

```
.
N050 G65 P8002 T5.0 D3.2342 S3.22 B3.245 (Check to see if offset adjustment is necessary)
N055 T0505 (Finish turning tool)
.
```

In line N050, we call the Custom Macro program and specify the tool station number being used (with T) and the target value for the dimension that is being measured (with D). Note that the Custom Macro program is also going to test the operator's input data to confirm that it is within allowable limits (maybe they measured the wrong diameter or entered the value incorrectly). If their entry is not within the allowable range, an alarm will be sounded. S specifies the small limit for the allowable range and B specifies the big limit. In the Custom Macro program, T is represented by local variable #20, D by #7, B by #2, and S by #19.

Here is the custom macro program.

```
O8002 (Custom macro to calculate and set offsets)
IF [#[2020 + #20] EQ 0 ] GOTO 99 (If operator has not yet entered a value, exit)
IF [#[2020 + #20] GT #19] GOTO 5 (If offset value is greater than small limit, go to N5)
#3000 = 100 (DIMENSION OFFSET TOO SMALL)
N5 IF[ #[2020 + #20] LT #2] GOTO 10 (If offset value is less than big limit, go to N10)
#3000 = 101 (DIMENSION OFFSET TOO BIG)
N10 #[2000 + #20] = #[2000 + #20] + [#7 - #[2020 + #20]] (Adjust primary offset)
#[2020 + #20] = 0 (Set secondary offset back to zero)
N99 M99
```

A note about your current methods

This technique simply builds upon your current methods. If you wish to enter offsets in the same manner you always have been (maybe your setup people or more experienced people wish to do this), you can do so. This technique will not interfere with your current methods. But your entry level operators will likely find this method of offset entry to be easier, faster, and less error prone than your current methods.

6: Quickly center the Y axis of a live-tooling turning center

More and more CNC lathes have live tooling capabilities – especially sliding headstock machines. This allows them to perform the same kinds of machining operations done on milling machines, eliminating the need for secondary operations. Rotating cutting tools, like drills, taps, reamers and end mills can be held in at least two attitudes: parallel to the X axis or parallel to the Z axis. These cutting tools can perform machining operations on faces (Z axis work) or on diameters (X axis work).

With some live tooling machines, each cutting tool is fixed on the X and Z axis centerlines. That is, the tool cannot move in the direction perpendicular to the XZ plane. These machines can only perform machining at the center of the workpiece in X.

More sophisticated live tooling machines *do allow* machining perpendicular to the XZ plane. They have a Y axis (though not all machine tool builders call this axis the Y axis). While the Y

axis often has a rather limited amount of travel (compared to the Y axis of a machining center), this dramatically increases the capability of the machine. Now the machine can perform machining operations that are not centered on the workpiece.

One common frustration with Y axis lathes is that the each cutting tool requires *calibration* during setup. This is because, with many machines, live tooling tool stations are not perfectly aligned with one another. When the center of one of them is at the X axis center, the others won't be. Indeed, many sliding headstock machines actually use the Y axis as the "tool changing mechanism". The Y axis simply moves to change from one cutting tool to the next.

While there may be ways to permanently qualify and record the related center positions, many setup people go through a rather tedious process to calibrate every live tool along the Y axis for every setup. This procedure involves touching off an aligning pin (held in the tool station) on both sides of the workpiece and using the position displays to calculate the center of workpiece location for the tool station along the Y axis. With this value known (the Y axis program zero assignment value for the tool station), the setup person will enter it into the appropriate geometry offset.

This custom macro we provide will dramatically simplify the process:

```
O0001 (CROSS TOOL ALIGN Y AXIS)
N1 #121 = 1 (TOOL NUMBER TO CALIBRATE)

(SELECT TOOL AND JOG PIN TO ABOUT 0.25 ABOVE PART IN X.)

N2 G98
N3 #101 = #5003
N4 G01 Y[#101+0.4] F30.0
N5 U-1.0
N6 #3006 = 100 (TOUCH X PLUS SIDE)
N7 #110 = #5003
N8 G01 Y[#101+0.4]
N9 U1.0
N10 Y[#101-0.4]
N11 U-1.0
N12 #3006 = 100 (TOUCH X MINUS SIDE)
N13 #111=#5003
N14 Y[#101-0.4]
N15 U1.0
N16 #105 = [[#110+#111]/2]]
N17 Y#105
N18 #106 = #5023
N19 #[2300 + #121] = #106
N20 G99
N21 M30
```


The setup person will begin by placing a pin in the cutting tool holder (as they are probably doing currently). The pin can be the same diameter as the cutting tool they'll be using.

They will then alter line N1, entering the tool station number of the tool to be calibrated. Next, they manually jog the machine so that the pin is approximately centered on the workpiece in Y and about 0.2 inch above the workpiece in X.

Next they will activate this program. In line N3, the current Y position is stored. #5003 is the current position in the *third axis*, which for most turning centers is the Y axis. (You must confirm this for your machine before using the custom macro.)

The Y axis will move plus in line N4 to a position that will clear the workpiece diameter. In line N5, the pin will move below center in X (U commands an incremental X movement).

In line N6, the machine will stop and place the message "TOUCH X PLUS SIDE" on the display screen. The setup person will select the handwheel mode and cautiously touch to pin to the workpiece in X. When finished, they will place the mode switch back to automatic mode and press the cycle start button.

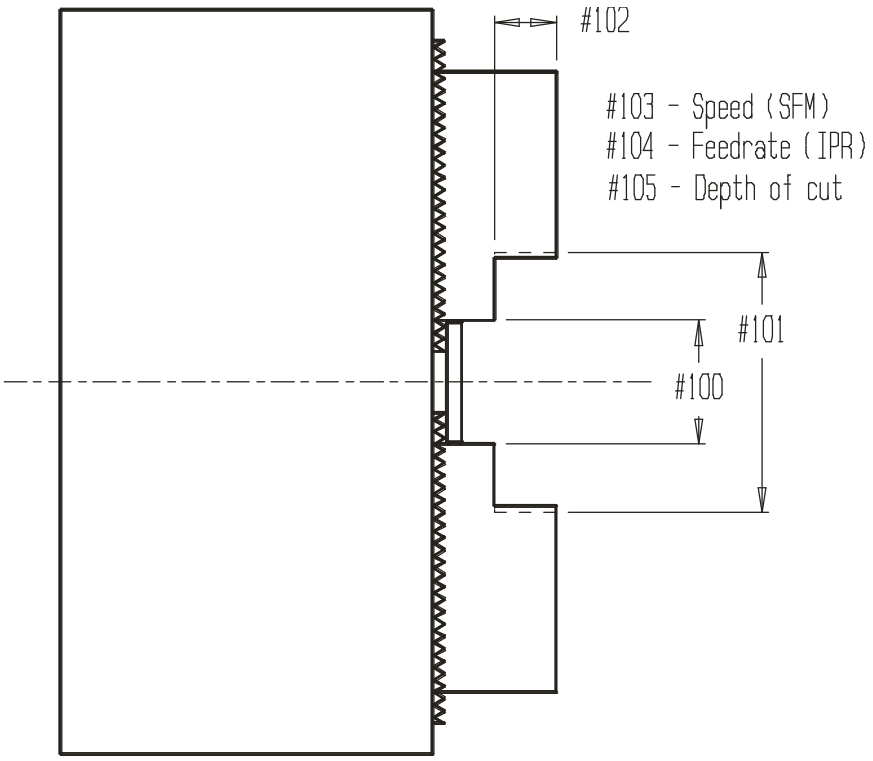
Line N7 stores the current Y position (while the pin is touching) and then the pin will move clear and come back to center in lines N8 through N10.

The process is repeated for the Y minus side in lines N11 through N15. In line N16, the true Y centerline is calculated and the Y axis moves to this position. In line N18, the current Y axis position relative to the Y axis zero return position is determined. Finally, in line N19, this value (Y axis program zero assignment value) is placed in the appropriate geometry offset.

This program can be improved. You may, for example, be able to come up with a way to have the custom macro automatically detect which live tool is in position (possibly by testing the current Y position after center is found). Also, this program is for relatively small parts (up to about 0.5 inch in diameter if a 0.25 diameter pin is used). With a few modifications, it can be made to work for larger part sizes.

7: Facilitate jaw boring on turning centers

Maybe you notice that turning center setup people are taking a great deal of time whenever soft jaws must be bored. They are performing this operation manually, using the machine's hand-wheel. As you know, boring jaws completely manually can be tedious and time consuming. With a Custom Macro program for jaw boring, the setup person will simply specify the size of the jaws to be bored and jaw boring Custom Macro program will bore the jaws. This might even free the setup person to do other things while the jaws are being bored. The next illustration shows the application, including the common variable arguments we will use.



Arguments for jaw boring utility Custom Macro program.

Here is the Custom macro program for the jaw boring utility. It assumes that the boring bar is in station number twelve and that program zero in Z is the face of the jaws.

```
O0023 (Jaw boring utility program)
#100 = 2.0 (Small diameter, boring starts from here)
#101 = 3.0 (Large diameter)
#102 = 0.5 (Depth of boring operation)
#103 = 300.0 (Speed for jaw boring)
#104 = 0.010 (Feed-rate for jaw boring)
#105 = 0.1 (Depth of cut for jaw boring)
T1212 (Index to jaw boring bar)
G96 S#103 M03 (Start spindle)
G98 G01 X#100 Z0.1 F30.0 (Fast feed to position to begin jaw boring)
G99 (Reselect feed per revolution mode)
G71 P1 Q2 U-0.04 W0.005 D#105 F#104 (Rough boring canned cycle)
N1 G00 X#101 (Rapid up to large diameter)
G01 Z-#102 (Machine to bottom of jaw)
N2 X#100 (Feed down face to starting diameter)
G70 P1 Q2 F[#104 * 0.5] (Finish bore)
M30 (End of program)
```

When the setup person needs to bore jaws, they modify the well documented arguments that begin the program. To be efficient, it must be as quick and easy as possible to load the boring bar. Better yet, maybe the jaw-boring boring bar is kept in the machine's turret.

With the program modified and the boring bar in position, they can activate the cycle and the jaws will be bored.

