Max and Programming

There are several questions that come up from time to time on the Max discussion list. Is Max a real programming language? if so how do I do [loop, switch, bitmap, recursion] and other programming tricks I already know how to do.

Is Max a Programming Language?

Computer programming texts generally spend a bit of time explaining the hierarchy of languages. There is at least:

- Machine language -- this is a list of numbers that instructs the CPU in what to do. People can't make much sense out of this, it's just a list of numbers. (There is actually an even more primitive language called microcode built in to the CPU that translates some of the machine language). Each processor operation requires two or more numbers.
- Assembly language -- this gives memorable names (mnemonics) to the numbers that represent operations. Names like CMPX and MOVH. Just to keep it obscure, actual numbers are usually represented in hexadecimal. Experts can read this stuff, but prefer not to. One line of assembly is translated into a single instruction for the processor.
- High level language -- still pretty terse, but at least the symbol for addition is usually +. That symbol can result in 30 lines of assembler. High level languages have a certain amount of intelligence built in. They can figure out (from context) whether the character '1' should be treated as an integer, floating point number, the address of something, or a character. In high level languages the numbers can be given names, like in algebra.

Max goes one level higher -- large chunks of code are represented by named boxes on the screen, and lines between the boxes show how information flows. Programs are often designed with something similar called flowcharts. The difference is, with flowcharts the blocks represent code that will be written. In Max, the code works already.

In further discussion, the books make other kinds of distinction between high level languages:

- In compiled languages, complete programs are converted into machine language before they can run. The form people can read is called source code, and this is rendered into object code for the computer. There has to be a special program to do this, called a compiler. Programs are actually written in a text editor.
- In interpreted languages, each line is decoded from the text file and executed as it is encountered. If you think about it, this means that there actually is a program running to run your program. It's a combination of text editor and compiler called an interpreter. This is a bit slower, but more flexible than compiled programs. It actually seems faster to write code, because the interpreter responds to each line as you type it. Of course if anyone else wants to run your program, they have to also have a copy of the interpreter.

Another distinction is between procedural and object oriented languages:

- Procedural languages are simple in concept - they work like a cake recipe; first a list of ingredients (data), then a series of instructions that are carried out in strict order. Ingredients are taken off their place on the shelf, and finished things are put back -- the program knows where things are supposed to be, not what they actually are.
- Object Oriented Languages are a little harder to explain. Instead of data in slots, the metaphor is like a bunch of people, each of whom knows some things, and knows how to manipulate the things they are given. The objects receive messages that tell them what their data is and what to do with it. What sends the messages? Other objects. Object oriented programs would be horribly complicated to write if it weren't pretty easy to derive new objects from existing ones.

So, how does Max fit into this? Max is a program that interprets actions of compiled objects that are coded in a procedural language. Since Max uses icons to represent chunks of high level language, Max is a meta-language. You can program Max in two ways. You can connect the existing objects, and you can write and compile your own objects in a high level language called C.

Most of learning Max is learning the behavior of the objects. The Max tutorials make a good start on that, and for the objects not covered, the help files usually give enough information for you to figure out their behavior with a bit of experimentation. The point of this particular essay is to look at the issue of interconnecting objects of any sort to handle various kinds of problem.

Passing Messages

Since Max is object oriented, you are never given access to raw data in the way you are in say, C programming[1]. You will encounter data in arguments to objects and in messages.

Arguments are easy. Some objects start out containing data, and you specify the data as you create the object. Type it right after the name.

Messages are entered in message boxes. A message box is an object whose name you don't type - you pick it from the palette and just type in the arguments. The arguments become the message it will send when told to.

Messages have two parts -- the message name, and the message arguments. Some messages have names but no arguments, like bang. Some messages seem to have no names, because the name is optional. These are the messages int, float, and list. The name is optional because it's easy to figure out what is meant. A number with a decimal point is a float, a number without is an int. Anything with several arguments, the first of which is a number, is a list.

[1] Where you have to type things like this: double pi = 3.14;

Any argument that is not a number is a symbol. In other languages, you have something called characters or text. Max does not have these, except at the edges. Characters and text are converted to and from ints by a few special objects. The ints associated with text are in the ascii code, and generally just called ascii. Symbols are easy to create, you just type groups of letters separated by spaces. [2] When Max encounters a symbol, it assigns a pointer[3] to it, and just passes that pointer around in messages. Saves time.

If a symbol is the first element in a message, it is taken as the name of the message. It may or may not have arguments. Yes, there is a message called Symbol. It needs one argument, which has to be a symbol.

Since everything is a number, how does Max keep floats, ints and symbols sorted out? All data in Max is contained in structures[4] called atoms. Part of the structure is the number, the other part is what type it is. Occasionally you will see notes in the Max window referring to argument type.

So, messages are passed from object to object in Max, and when an object receives a message, it does what it knows how to do.

Responding to Messages

Each object has a set of messages it can respond to. You can see the set for an object by shift-option-click on the object. Exactly what happens is explained in the manual, or in the Help File (option click) for that object. In general, messages to an object set internal data for later operations or cause immediate operations. Because the convention for passing messages is right before left, the right inlets of object are the data setters, and the left inlet causes the action.

It's worth taking a moment to really appreciate the effects of right to left action. In figure 1 we have three simple actions- the word bang will be printed in the Max window, preceded by the letter A, B or C.

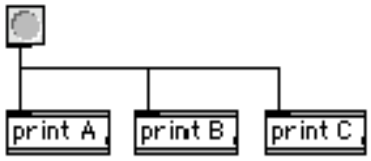

Figure 1.
The result of the configuration shown is:

C : bang
B : bang
A : bang

[2] If you want a symbol that includes a space, enclose the whole thing with smart single quotes as you type (‘option- right bracket’)..
[3] Pointer is a programming term that means the location of something.
[4] That's another programming term. It means bunch of numbers that always go together.

The C print object, being on the right, gets to go first. Now look at figure 2.

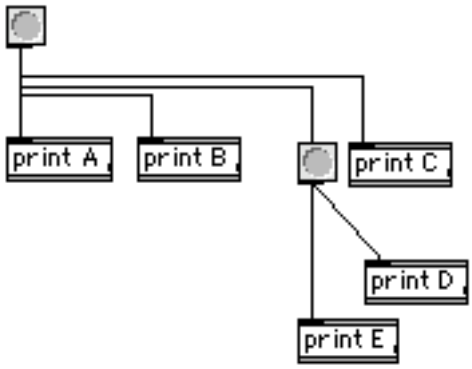

Figure 2.
Adding more buttons splits the path of the messages. The right hand button sends a message to print C, then to another button. That button sends its message to print D, and then print E. The result is:

C : bang
D : bang
E : bang
B : bang
A : bang

The entire right hand fork of the patch occurs before print B. With the slight change given in figure 3, we get:

C : bang
E : bang
D : bang
B : bang
A : bang

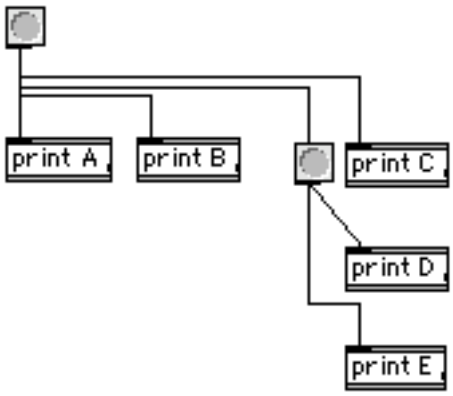

Figure 3.

It's really common for the movement of one object to seriously change the behavior of a patch. To prevent this, use the trigger object with one destination per outlet.

Since we generally want objects to work with the most recent data, the convention that the left inlet triggers actions make sense. Likewise, if an object sends more than one message, they will come from different outlets, the right before the left.

Right to left ordering fails when the messages are sent through the send and receive objects. There is no way to tell which receive object will respond first. If it matters, you must use one send for each receive, each pair with a unique name.

Stack Overflow

The fact that all actions in Max patches are triggered by message passing raises the possibility of feedback situations. The interface won't let you connect the outlet of an object directly back to one of its own inlets, but you can do this indirectly through another. If the message sent back finds its way to the inlet that triggers itself, the resulting feedback loop will cause a stack overflow. Max will halt in this condition, and not restart until you have chosen resume in the Edit Menu.

It sometimes makes sense to connect the result of an operation back to the beginning, but it should be to a right inlet, not one that causes action. The calculation should be triggered by a controlled source such as Uzi or metro.

Iteration

The question most traditionally trained programmers usually ask first is "How do I do iterations in Max?" At first glance this seems rather difficult, because iteration in most languages is done by conditional loops of one sort or another. In Max we have Uzi, which gives a predefined number of bang messages. This takes care of most cases, where the condition for ending the loop is simply how many times it has executed.

Figure 4.
If you need a loop that can be shortened by a conditional expression, you can send the message break back up to Uzi when the condition is satisfied.

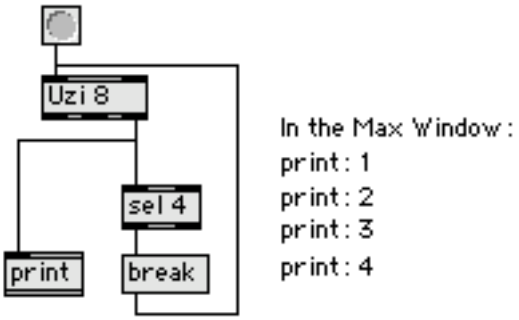

Figure 5.
You can chain Uzis in much the same way that traditional loops are nested. The lower Uzis will cycle faster than the upper Uzi.

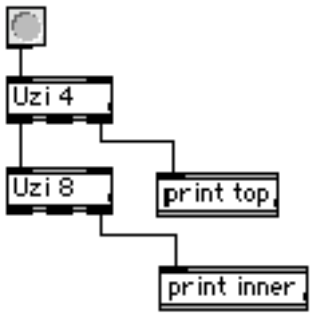

Figure 6.
One oddity of Uzi is that the index output starts at 1 and counts up. If you need 0 based values, you can process the index. For more flexible control of the count, there is loop, part of my Lobjects. Loop uses feedback with internal buffering to prevent stack overflow.

More often, we need iterations at musical rates. This is usually done with metro, which produces a bang at specified intervals. Metro is also easily switched off with a conditional expression. The period of metro is set by a number message. Any change to this period will usually take effect after the end of the current cycle. (After the next bang.) The exception is when the change of period is a consequence of the metro's own bang output. This is explored a bit further in the essay on Max & Rhythm.

Conditionals

Most conditional operations in Max are done by using the truth of some comparison object to route messages around the patch. One thing to keep in mind is that the simple conditional objects like < and = = always give some output when a message is received. Very often, the output has to be modified by a **select** if you want a bang on true or by **change** if you only want output on the change in truth.

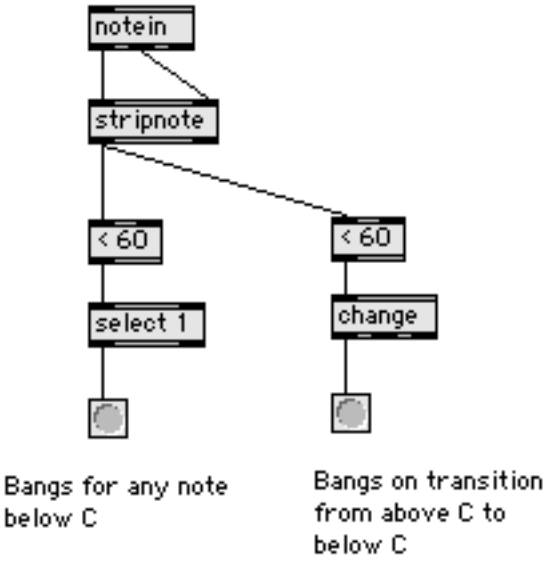

Figure 7.

To conditionally affect the operation of the patch, use the comparison objects to control various gates and switches.

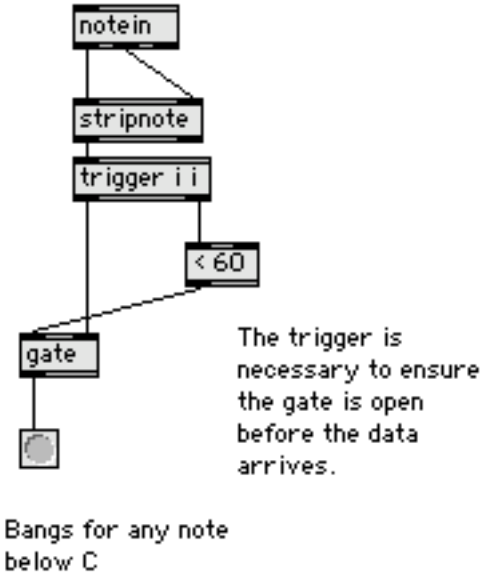

Figure 8.

For more complex conditional operation without a lot of boxes, there is the if/then object. This takes a statement of the type:

If {expression} then {message} [else message]

If and then are required, else is optional. The expression is a C type conditional expression. The expression can refer to data from inlets by the tokens $iN or $fN

where N is the inlet number. Note that the expression is evaluated on receipt of a message in inlet 1, but that message does not have to be part of the expression.

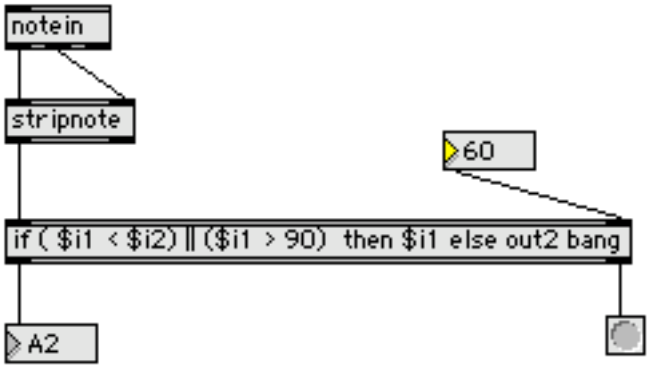

Figure 9.

It's important to remember that what follows then is a message, not an expression. A message may include the input tokens to pass the data through. A message may be prefaced by out2 to direct it to a right outlet.

Jump

A switch or case operation can be done with the route object. Prepend the selector into a list containing the data. Then route will direct it where it should go.

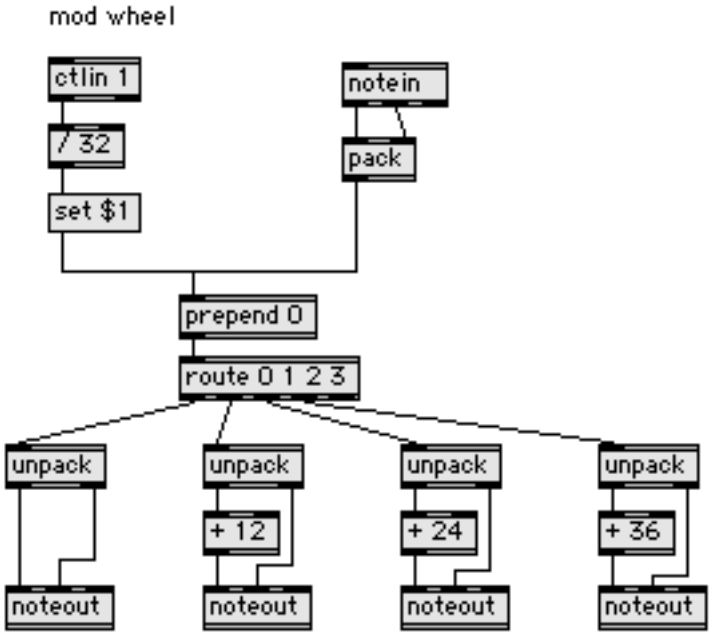

Figure 10.

Recursion

Even though you can name a patcher and then refer to it in other patchers, you can't refer to it in itself. That's because patchers are stored as files, and when a patcher is loaded, all of the patchers it contains are loaded too. When Max attempts to load a patcher that contains itself, it has to load the copy of the patcher it contains, then the copy contained in that, which goes on infinitely and crashes the system. You can write a recursive routine in Java to run in the mxj object.

Data Storage and Retrieval

Programmers are used to directly manipulating data with statements like

theVariable = 0; or Array[index] = data;

In Max, the data for objects is usually associated with inlets. For instance, the counter object has inlets for count maximum, direction flag, and reset value. If there is no inlet far a particular parameter, there may be a message that sets it.

Of course there are many situations where we need to store data for later use.

For simple variables use the int and float objects. These will hold a value applied to the right inlet and send it when banged on the left. The number boxes can work in a similar manner, as the set message will change the value without sending it along.

An entire list of values can be stored by setting a message box, or with the llist Lobject.

The preset object can memorize the value of number boxes and other user interface objects and restore them.

Bulk data storage is done with the coll and table objects. These may be thought of as arrays: the data is applied to the right inlet, and the index to the left. To get data back, only an index is needed. Both of these objects have an extensive set of additional features, including direct user access and the ability to save data in files.

Table

Table is probably easiest to use. Although it has two inlets, the list method is the most reliable way of setting data. Here the table is storing current velocity for all notes:

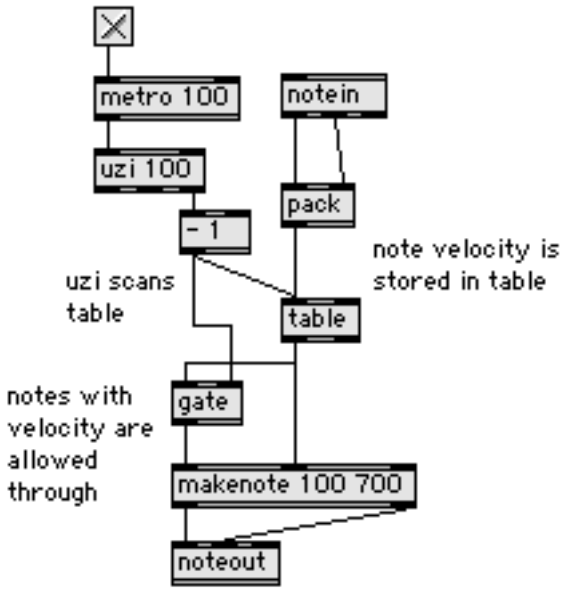

Figure 11.

Table can store data or not store data with the patcher. Note that if you don't specify, there will be a potentially annoying prompt.

Coll

Coll is a dynamic data structure that can mimic a linked list or a multidimensional array. Since memory is allocated only when data is stored, it does not need to be pre dimensioned and keeps a remarkably small footprint.

Coll as array

Each store operation creates a new address to specify an address, prepend it to the data. The address by itself reads the stored data. For a multi dimensional array such as [i] [j] use an expression to combine the two indices.

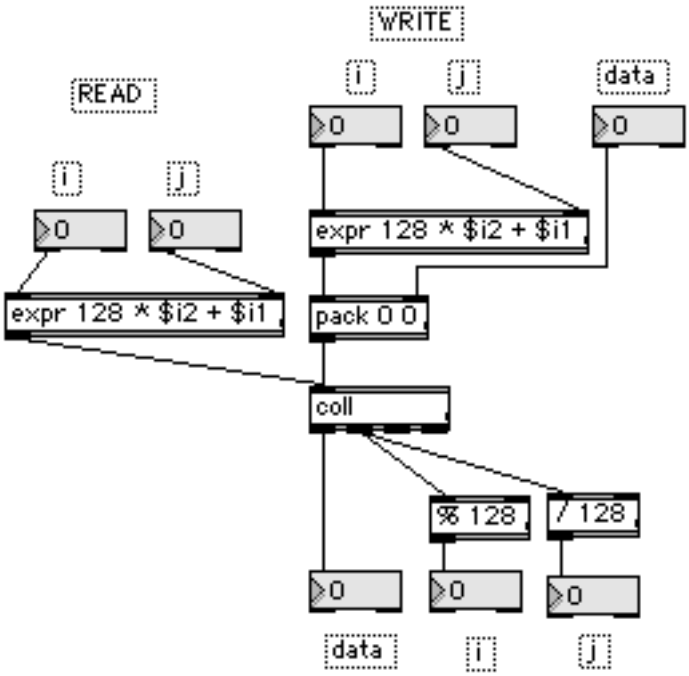

Figure 12.

Of course, since each coll address can refer to a list of data, there is not all that great a need for multi dimensional arrays. You can access individual members of a stored list with the nth message (note that nth counts from 1 in this case), or use sub to write there.

Since there's no need for the addresses in a coll to be contiguous, you can use any sort of calculation to make addresses. See the tutorial on Max & MIDI Time Code to see an example.

Coll as Linked List

The elements of a coll are linked as well as indexed and there is a full set of commands for traversing the contents. The insert function is a bit flaky, and there is no append to the end function so you should maintain your own addressing scheme for writing data.

Colls and Symbol Addressing

Colls can also reference data by symbols. This makes coll something of an enum evaluator. There's not a lot of general symbol support in Max, but the trick of connecting message boxes directly to a coll simplifies making user interfaces. Since the menu object can be set to send symbols instead of numbers, it's easy to edit and reorganize menus if the position of an item does not matter.

For simple array operations with arbitrary amounts of data, there is Larray in the Lobjects. Ldumpster is a specialized Larray designed for system exclusive work.

Strings

Max does not support strings as such. The only text tools available are symbols and lists of ascii values.

Symbols are optimized for rapid processing. The symbol message only contains a pointer, and usually matching the pointers is all that is required. Only the message box and the Max window actually deal with the contents of symbols. Thus the original concept was that symbols would be predefined and there would be no character level manipulation of them.

The spell and sprintf objects will convert symbols to lists of ascii and vice versa. Some techniques for working with these are in the tutorial Max and ACSII.

Coding in Java

Java and Javascript are supported in the mxj, js and jui objects.

Coding in C

You can write your own objects in C this is done in Xcode or Visual C. Instructions and template projects are in the Software Development Kit currently at
http://www.cycling74.com/twiki/bin/view/ProductDocumentation/MaxUBSDK