Basic Programming

The Core Perl Syntax

garth@ggilmour.com

Introducing Scalar Variables

- A scalar variable is a single box
  - It is prefixed by the dollar sigil
- Perl is a weakly typed language
  - A box may hold a number, a string or a memory address
  - If the scalar holds an address it is known as a ‘reference’
- Type conversions occur automatically
  - So in the expression ‘$var1 = $var2 + $var3’ both types are converted to numbers before being added together
- As with all variables sigils are created on demand
  - You don’t need to declare them separately

garth@ggilmour.com

Scalar Variables and Operators

- In strongly typed languages operators are overloaded
  - So ‘var1 + var2’ would add the variables if they were numbers but concatenate them if they were strings
  - If the types weren't matched there would be a compiler error
- Weak typing means Perl cannot support overloading
  - Instead there must be an operator for each operation
  - In the case of addition:
    - The ‘+’ operator means add as numbers
    - The ‘.’ operator means concatenate
    - Conversions are made as required

garth@ggilmour.com

Operators Commonly Used in Perl

Description	Number Version	String Version
Addition	$var1 + $var2	$var1 . $var2
Equality	$var1 == $var2	$var1 eq $va2
Ordered Comparison	>, <, <=, >=	lt, gt, le, ge
Power Of	$var1 ** 3	$var1 x 3
Bitwise Comparison	&, |, ^ (NB work differently for numbers and strings)	
Logical	&&, ||, ! and, or, not (Lower precedence)	
Conditional	$var1 = $var2 ? 12 : 14;	
Range	1..4	‘D’ .. ‘Z’

garth@ggilmour.com

```
$num1 = 42;
$num2 = "42";

$result = $num1 + $num2;
print "adding numbers gives $result", "\n"x2;

$result = $num1 . $num2;
print "adding strings gives $result", "\n"x2;

$result = $num2 ** 3;
print "42 to the power of 3 is $result", "\n"x2;

$result = $num1 x 3;
print "42 concatenated with itself three times is $result", "\n"x2;

if($num1 == $num2) {
	print '$num1 and $num2 are equal as numbers',"\n"x2;
}
if($num1 eq $num2) {
	print '$num1 and $num2 are equal as strings',"\n"x2;
}
```


garth@ggilmour.com

String Values in Detail

- Strings may be placed in single or double quotes
  - They have different meanings and are not interchangeable
- Single quotes are not treated specially
  - The interpreter sees them as a plain sequence of characters
- Double quotes cause variable interpolation
  - Perl searches for sigils in the string and replaces them with the value of the variable (creating it if required)
- Note that you can also use ‘backtick’ quotes
  - These surround a string to be run as an OS command
  - E.g. ‘ $var1 = `ls -al` ’ runs the UNIX list command and stores the results in the variable ‘$var1’

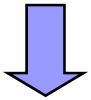
garth@ggilmour.com

```
$var1 = "abc";
$var2 = 123;
$var3 = ["ab","cd","ef"];

print 'Values are $var1, $var2 and $var3 \n';
print "Values are $var1, $var2 and $var3 \n";

$path = `set path`;

print "Value of path environment variable is:\n $path";
```



```
Values are $var1, $var2 and $var3 \nValues are abc, 123 and ARRAY(0x225e28)
Value of path environment variable is:
PATH=c:\jdk1.5.0_05\bin;C:\Perl\site\bin;C:\Perl\bin;c:\ruby\bin;C:\WINDOWS\syst
em32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
```


garth@ggilmour.com

What is Truth in Perl?

- Truth is a source of confusion in Perl
  - Many Perl functions return boolean values
  - By Perl does not have a boolean type
- Three things in Perl count as false
  - The empty string
  - The string or number “0”
  - An undefined variable
- You can obtain the undefined value by:
  - Using a variable that has not been initialized
  - Passing a variable as an argument to ‘undef’
  - Using the return value from ‘undef’

garth@ggilmour.com

```
$var1 = "0";              # The string "0" counts as false
$var2 = "";               # The empty string also counts as false
$var3 = "AB";             # Other string values are true
$var4 = -12;              # Other numerical values are true
$var5;                    # Undefined values are false
$var6 = undef();          # Values set to undef are false

printTruth('$var1',$var1);
printTruth('$var2',$var2);
printTruth('$var3',$var3);
printTruth('$var4',$var4);
printTruth('$var5',$var5);
printTruth('$var6',$var6);

undef($var4);  # Release storage space for var1 so it becomes undefined
printTruth('$var4',$var4);

sub printTruth {
	my ($varName,$varValue) = @_;
	if($varValue) {
		print "$varName is true\n";
	} else {
		print "$varName is false\n";
	}
}
```


garth@ggilmour.com

Special Scalar Variables

- The Perl interpreter automatically creates variables
  - These variables are represented by symbols rather than names
- This is a further source of confusion when learning Perl
  - The ‘English’ module renames the variables more clearly

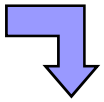
Variable Name	Description
$]	The version of Perl supported by this interpreter
$0	The name of the file containing the current script
$^O	The name of the operating system
$_	The current item (used in input, output and loops)
$/	The line separator used when reading text (default it newline)

garth@ggilmour.com

Special Scalar Variables

```
print "This is verson $] of Perl\n";
print "Running on the $^O operating system\n";
print "The current script is $0 \n";

@myarray = ("ab","cd","ef","gh");
print "Elements are:\n";
foreach(@myarray) {
    print "\t $_ \n";
}
```



```
This is verson 5.008008 of Perl
Running on the MSWin32 operating system
The current script is C:\perl\specialScalars.pl
Elements are:
        ab
        cd
        ef
        gh
```


garth@ggilmour.com

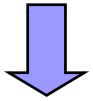
Reading Text Into a Scalar Variable

- Reading text into a scalar is simple
  - The expression '$line = <INPUT>' reads a line of text from INPUT and stores it in the scalar 'line'
- The symbol within the angle braces is a handle
  - Handles are links to resources outside your program
  - The 'STDIN' and 'STDOUT' handles are created automatically
  - We will see how to open and close handles later…
- To write data use the 'print' function
  - This takes a file handle as an optional first parameter
  - The default handle is STDOUT

garth@ggilmour.com

```
open(INPUT,"MuchAdoAboutNothing.txt");

$count = 1;
foreach(<INPUT>) {
    print("$count:\t $_");
    $count++;
}
```



```
1:          Much Ado About Nothing
2:          A comedy by William Shakespear
3:
4:          Act 1, Scene 1
5:          Before LEONATO'S house.
6:            Enter LEONATO, HERO, and BEATRICE, with a Messenger
7:
8:          LEONATO
9:            I learn in this letter that Don Peter of Arragon
10:           comes this night to Messina.
```


garth@ggilmour.com

Conditionals and Iteration in Perl

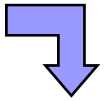
- Perl supports the standard ‘if’ conditional
  - Note that the ‘elif’ keyword is used instead of ‘else if’
- The ‘unless’ is an ‘if’ in reverse
  - So ‘if(!done()) { … }’ becomes ‘unless(done()) { … }’
  - This is convenient once you get used to it
- It is possible to place the test after the action
  - E.g. ‘$a = 12 if $b < $c’ or ‘$a = 12 unless $b >= $c’
- The C/C++ ‘switch’ keyword is not supported
  - Although there are ways of simulating it if required
- The rarely used ternary conditional operator is available
  - E.g. ‘$var1 = ($var2 == $var3) ? 17 : 19’

garth@ggilmour.com

```
print "Enter a number\n";
$number = <STDIN>;
chomp($number);

if($number < 10) {
    print "$number is less than 10\n";
} elsif($number < 20) {
    print "$number is less than 20\n";
} elsif($number < 30) {
    print "$number is less than 30\n";
} else {
    print "$number is greater than 30\n";
}

unless($number % 2 == 0) {
    print "$number is odd\n";
}
```



```
Enter a number
17
17 is less than 20
17 is odd
```


garth@ggilmour.com

Conditionals and Iteration in Perl

- The standard loops are supported
  - Perl provides ‘while’, ‘do … while’ and ‘for’ loops
  - With the same syntax and semantics as C/C++
- Variations of the ‘while’ loops are available
  - The ‘until’ and ‘do … until’ loops avoid the need to negate the conditional, but are not always intuitive
- The ‘for’ loop is extended in two ways:
  - It can be used with ranges rather than counters
    - E.g. ‘for(1..4) { … }’ or ‘for(1..$max) { … }’
  - The ‘foreach’ loop iterates over arrays
    - We will meet it later when introducing data structures

garth@ggilmour.com

```
print "Enter a positive number\n";
$max = <STDIN>;
chomp($max);

if($max <= 0) {
	die("Number must be positive!");
}
print "Demo of while loop\n";
print "\tNumbers from 0 to $max are:\n";
$count = 0;
while($count <= $max) {
	print "\t\t$count\n";
	$count++;
}
print "Demo of do..while loop\n";
print "\tNumbers from 0 to $max are:\n";
$count = 0;
do {
	print "\t\t$count\n";
	$count++;
} while($count <= $max);
```

```
print "Demo of until loop\n";
print "\tNumbers from 0 to $max are:\n";
$count = 0;
until($count > $max) {
	print "\t\t$count\n";
	$count++;
}
print "Demo of do..until loop\n";
print "\tNumbers from 0 to $max are:\n";
$count = 0;
do {
	print "\t\t$count\n";
	$count++;
}until($count > $max);

print "Demo of for loop v1\n";
print "\tNumbers from 0 to $max are:\n";
for($count=0; $count<= $max; $count++) {
	print "\t\t$count\n";
}
print "Demo of for each loop v3\n";
print "\tNumbers from 0 to $max are:\n";
for(0..$max) {
	print "\t\t$_\n";
}
```


garth@ggilmour.com

Conditionals and Iteration in Perl

- Loops can optionally have a continue block
  - E.g. ‘while($a < 12) { … } continue { … }’
- This is used with the loop control operators
  - A call to ‘last’ immediately exits the loop
    - Without executing the continue block
  - A call to ‘next’ skips the remaining statements in this iteration
    - But the continue block is executed before the loop condition is re-evaluated and (if true) the next iteration begins
  - A call to ‘redo’ restarts the current iteration
    - The continue block is not executed
    - The loop condition is not checked

garth@ggilmour.com

Basic Perl I/O

Using the Console and Files

garth@ggilmour.com

Basic Perl I/O

- Perl I/O is based around handles
  - Links provided by the OS to data sources and sinks
- Handles for the console are built-in
  - ‘STDIN’ and ‘STDOUT’ represent the command prompt
- Input is read via the ‘< >’ operator
  - So ‘$data = <STDIN>;’ reads a line from the console
  - Use the ‘chomp’ function to remove the newline
- Output is written via the ‘print’ function
  - If the first argument is not a handle then STDOUT is used
  - A comma should not be placed after the handle

garth@ggilmour.com

Testing File Paths

- Perl provides built in operators for testing file paths
  - E.g. ‘if(-e $file && -T $file) { print "$file exists and is a text file"; }’
  - You should always check a file before opening it

File Test Operator	Description
-e	File exists
-r	File is readable
-w	File is writable
-z	File has zero size
-s	Returns file size
-T	File is a text file
-B	File is a binary file
-S	File is a socket

garth@ggilmour.com

Opening and Reading From Files

- The ‘open’ function is used to create a file handle
  - The first argument is the symbol we want to represent the handle
- Files are opened in a particular mode
  - As indicated by the character(s) before the filename
  - The default is to open for reading

Function	Description
open(HANDLE, “myfile.txt”) open(HANDLE, “<myfile.txt”)	Open file for reading
open(HANDLE, “>myfile.txt”)	Open file for writing (truncating if necessary)
open(HANDLE, “>>myfile.txt”)	Open file for appending
open(HANDLE, “+<myfile.txt”)	Open file for reading and updating

garth@ggilmour.com

Opening and Reading From Files

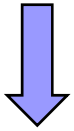
- The standard form of ‘open’ could cause problems
  - If the handle name was already in use (e.g. as a subroutine)
  - If we were trying to open a file called ‘>myfile.txt’
- There are two ways around this
  - There is a three argument form of ‘open’
    - The mode(s) are passed as separate arguments
  - The handle can be stored in a scalar variable
    - This is known as an indirect filehandle
- Once you have a file opened you can:
  - Read lines from the file using the ‘< >’ operator
  - Write to the file using the print method

garth@ggilmour.com

```
open(INPUT,"input.txt");
open(OUTPUT,">output.txt");

$count = 0;
while($line = <INPUT>) {
    print OUTPUT ++$count, "\t", $line;
}
print "Processed $count lines\n"
```

This short interval was sufficient to determine d'Artagnan on the part he was to take. It was one of those events which decide the life of a man; it was a choice between the king and the cardinal--the choice made, it must be persisted in. To fight, that was to disobey the law, that was to risk his head, that was to make at one blow an enemy of a minister more powerful than the king himself. All this young man perceived, and yet, to his praise we speak it, he did not hesitate a second. Turning towards Athos and his friends, "Gentlemen," said he, "allow me to correct your words, if you please. You said you were but three, but it appears to me we are four."



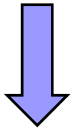
```
1	This short interval was sufficient to determine d'Artagnan on the
2	part he was to take.  It was one of those events which decide the
3	life of a man; it was a choice between the king and the
4	cardinal--the choice made, it must be persisted in.  To fight,
5	that was to disobey the law, that was to risk his head, that was
6	to make at one blow an enemy of a minister more powerful than the
7	king himself.  All this young man perceived, and yet, to his
8	praise we speak it, he did not hesitate a second.  Turning
9	towards Athos and his friends, "Gentlemen," said he, "allow me to
10	correct your words, if you please.  You said you were but three,
11	but it appears to me we are four."
```


garth@ggilmour.com

```
open(INPUT, '<', "input.txt");
open(OUTPUT, '>', "output.txt");

$count = 0;
while($line = <INPUT>) {
    print OUTPUT ++$count, "\t", $line;
}
print "Processed $count lines\n"
```

```
This short interval was sufficient to determine d'Artagnan on the
part he was to take.  It was one of those events which decide the
life of a man; it was a choice between the king and the
cardinal--the choice made, it must be persisted in.  To fight,
that was to disobey the law, that was to risk his head, that was
to make at one blow an enemy of a minister more powerful than the
king himself.  All this young man perceived, and yet, to his
praise we speak it, he did not hesitate a second.  Turning
towards Athos and his friends, "Gentlemen," said he, "allow me to
correct your words, if you please.  You said you were but three,
but it appears to me we are four."
```



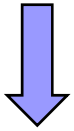
```
1   This short interval was sufficient to determine d'Artagnan on the
2   part he was to take.  It was one of those events which decide the
3   life of a man; it was a choice between the king and the
4   cardinal--the choice made, it must be persisted in.  To fight,
5   that was to disobey the law, that was to risk his head, that was
6   to make at one blow an enemy of a minister more powerful than the
7   king himself.  All this young man perceived, and yet, to his
8   praise we speak it, he did not hesitate a second.  Turning
9   towards Athos and his friends, "Gentlemen," said he, "allow me to
10  correct your words, if you please.  You said you were but three,
11  but it appears to me we are four."
```


garth@ggilmour.com

```
open($input, '<', "input.txt");
open($output, '>', "output.txt");

$count = 0;
while($line = <$input>) {
    print $output ++$count, "\t", $line;
}
print "Processed $count lines\n"
```

This short interval was sufficient to determine d'Artagnan on the part he was to take. It was one of those events which decide the life of a man; it was a choice between the king and the cardinal--the choice made, it must be persisted in. To fight, that was to disobey the law, that was to risk his head, that was to make at one blow an enemy of a minister more powerful than the king himself. All this young man perceived, and yet, to his praise we speak it, he did not hesitate a second. Turning towards Athos and his friends, "Gentlemen," said he, "allow me to correct your words, if you please. You said you were but three, but it appears to me we are four."



```
1   This short interval was sufficient to determine d'Artagnan on the
2   part he was to take.  It was one of those events which decide the
3   life of a man; it was a choice between the king and the
4   cardinal--the choice made, it must be persisted in.  To fight,
5   that was to disobey the law, that was to risk his head, that was
6   to make at one blow an enemy of a minister more powerful than the
7   king himself.  All this young man perceived, and yet, to his
8   praise we speak it, he did not hesitate a second.  Turning
9   towards Athos and his friends, "Gentlemen," said he, "allow me to
10  correct your words, if you please.  You said you were but three,
11  but it appears to me we are four."
```


garth@ggilmour.com

Opening and Reading From Files

- Lines from a file should be read using a ‘while’ loop
  - A ‘for’ loop causes the interpreter to create a list of all the lines from the file, which is then iterated over
- File handles should be closed via ‘close’
  - Handles stored as scalars are automatically closed when the variable goes out of scope, but you may want to do this earlier
- You should verify calls to ‘open’, ‘print’ and ‘close’
  - Both return a boolean value to indicate success or failure
  - You can throw an error using the ‘die’ or ‘croak’ functions
    - We will cover these in depth later in the course

garth@ggilmour.com

```
open($input, '<', "input.txt") or die "Can't open input";
open($output, '>', "output.txt") or die "Can't open output";

$count = 0;
while($line = <$input>) {
	print($output ++$count, "\t", $line) or die "Can't write to file";
}
print "Processed $count lines\n";

close($input) or die "Can't close input";
close($output) or die "Can't close output";
```


garth@ggilmour.com

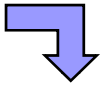
File Handles and Globbing

- You can place a pattern inside the '< >' operator
  - In which case the pattern is 'globbed'
  - E.g. '@dirs = <../*>;'
- To avoid confusion use the 'glob' function
  - E.g. '@dirs = glob('../*');'
- Globbing is often used to change file properties
  - 'chmod' and 'chown' change a files access rights and ownership
  - E.g. 'while(glob("*.pl")) { chmod(O777, $_); }'
  - E.g. 'while(glob("*.pl")) { chown($user_id,$group_id, $_); }'

garth@ggilmour.com

```
@exampleDirectories = glob('..\*');

print "Perl example files are: \n";
foreach $dir (@exampleDirectories) {
	@perlFiles = glob($dir . '\*.pl');
	foreach(@perlFiles) {
		#characters preceding a slash preceding
		#  letters ending in '.pl'
		m/.*\\(\w+\.pl)/;
		print "\t$1 in $dir \n";
	}
}
```



```
Perl example files are:
	arrayFunctions.pl in ..\arrays
	arraysAndLists.pl in ..\arrays
	days.pl in ..\arrays
	forEach.pl in ..\arrays
	fork.pl in ..\concurrency
	threads.pl in ..\concurrency
	customerDB.pl in ..\databases
	arraysOfArrays.pl in ..\dataStructures
	checkingErrors.pl in ..\files
	indirectHandles.pl in ..\files
	basicHashes.pl in ..\hashes
	extraSyntax.pl in ..\hashes
```


garth@ggilmour.com

Arrays in Perl

Creating and Using Lists

garth@ggilmour.com

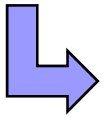
Introducing Arrays in Perl

- In other languages arrays cannot change size
  - Hence they must be supplemented with data structures
  - E.g. the C++ STL or the Collections libraries in Java and C#
- In Perl arrays can grow and shrink as required
  - So there is no need for a separate ‘vector’ or ‘LinkedList’ type
  - If the array is of size 10 and you try to store something in box 100 then the size is automatically changed
- Often arrays are created implicitly
  - E.g. ‘$myarray[9] = “abc”’ would create an array called ‘myarray’ with ten boxes, all of which were undefined apart from the last

garth@ggilmour.com

```
$myarray[8] = "string in 9th box";
$myarray[10] = "string in 11th box";

$count = 0;
foreach(@myarray) {
        print $count++,": $_\n";
}
```



```
0:
1:
2:
3:
4:
5:
6:
7:
8: string in 9th box
9:
10: string in 11th box
```


garth@ggilmour.com

Support for Arrays in Perl

- Arrays are declared with the '@' sigil
  - E.g. '@myarray = ("abc", 123, "def", 456, "ghi", 789)'
- Normally we create an array based on a list of values
  - The 'qw' operator can be used to avoid quoting
  - E.g. '@myarray = qw(abc 123 def 456 ghi 789)'
- Lists can also be initialized based on arrays
  - E.g. '($val1,$val2,$val3) = @myarray' copies the values in the first three boxes into the scalar variables
  - '($val1) = @myarray' is an idiom for grabbing the first value
    - It is equivalent to '$val1 = $myarray[0]'

garth@ggilmour.com

Support for Arrays in Perl

- Note that lists are only used for grouping
  - Unlike arrays their existence is only ever temporary
- The '@' sigil is not used when indexing
  - Instead of '@myarray[1]' we write '$myarray[1]'
  - This is because the value we are accessing is a scalar
- Arrays can be created based on slices of other arrays
  - E.g. '@array2 = @array1[1..3]' creates a new array called 'array2' holding copies of boxes 2,3 and 4 in 'array1'

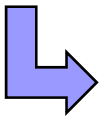
garth@ggilmour.com

```
@tstArray = qw(abc def ghi jkl mno pqr);

($first) = @tstArray;

($a,$b,$c) = @tstArray;

print "First element is: $first\n";
print "First three elements are: $a $b $c\n"
```



```
First element is: abc
First three elements are: abc def ghi
```


garth@ggilmour.com

Special Features of Arrays

- Perl makes assumptions about your use of arrays
  - Almost anything you might write is meaningful
  - Even if the meaning is not what you intended
- If you put an array in a scalar context the size is used
  - '$var = @myarray' stores the size of 'myarray' in 'var'
  - '$var = @myarray - 1' would store the index of the last box
- You can easily copy values into another array
  - E.g. '@myarray3 = (@myarray1,@myarray2)' would mean 'myarray3' contained copies of the values in the other two arrays
  - It does not create a multidimensional array

garth@ggilmour.com

Special Features of Arrays

- There is an easy way to find the last index
  - For ‘@myarray’ it is stored in the variable ‘$#myarray’
- Arrays can be shrunk if required
  - The special variable ‘$#myarray’ is not immutable
  - So ‘$#myarray -= 2’ removes the last two boxes of ‘myarray’
- There are two ways to empty out an array
  - By assigning its size to -1
    - E.g. ‘$#myarray = -1’
  - By assigning it to an empty list
    - E.g. ‘@myarray = ()’

garth@ggilmour.com

```
@workDays = ("Monday","Tuesday","Wednesday","Thursday","Friday");
@weekendDays = qw(Saturday Sunday);
@days = (@workDays,@weekendDays);

$numDays = @days;
$firstDay = $days[0];
$lastDay = $days[$#days];

print "There are $numDays days in a week\n\n";
print "$firstDay is the first day and $lastDay is the last day \n";

print "\nThe other days are:\n";
foreach $day (@days) {
	unless($day eq $firstDay or $day eq $lastDay) {
		print $day, "\n";
	}
}
print "\nThe last four days are:\n";
@lastDays = @days[3..6];
foreach $day (@lastDays) {
	print $day, "\n";
}
```


garth@ggilmour.com

Iterating Over Arrays

- The easiest way to loop over an array is ‘foreach’
  - This iterates over a list of values and assigns each to a scalar variable
  - You can specify the scalar or use the built-in ‘$_’
- The ‘foreach’ keyword is just an alias for ‘for’
  - Which one you use is a matter of style

```
@myarray = qw(abc def ghi jkl);

print "Loop one\n";
foreach $item (@myarray) {
    print "\t",$item,"\n" ;
}
print "Loop two\n";
foreach (@myarray) {
    print "\t",$_,"\n" ;
}
print "Loop three\n";
for $item (@myarray) {
    print "\t",$item,"\n" ;
}
print "Loop four\n";
for (@myarray) {
    print "\t",$_,"\n" ;
}
```


garth@ggilmour.com

Functions for Working With Arrays

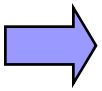
- Perl provides powerful functions for manipulating arrays
  - The ‘push’ and ‘pop’ functions add and remove from the end
  - The ‘unshift’ and ‘shift’ functions do the same from the start

Function Name	Description
push	Add a new box to the end of the array
pop	Remove a box from the end of the array
unshift	Add a new box to the start of the array
shift	Remove the first box in the array
join	Join all the values in the array into a string, separated by a delimiter
split	Create an array by splitting a string into a sequence of sub-strings, using a regular expression to specify the delimiter token(s)

garth@ggilmour.com

```
@myarray1 = qw(abc def ghi);

$val1 = pop(@myarray1);
print "\nJust popped $val1, contents now:\n";
foreach(@myarray1) {
	print "\t$_\n";
}
push(@myarray1,"zzz");
print "\nJust pushed zzz, contents now:\n";
foreach(@myarray1) {
	print "\t$_\n";
}
$val1 = shift(@myarray1);
print "\nJust shifted $val1, contents now:\n";
foreach(@myarray1) {
	print "\t$_\n";
}
unshift(@myarray1,"AAA");
print "\nJust unshifted AAA, contents now:\n";
foreach(@myarray1) {
	print "\t$_\n";
}
```



```
Just popped ghi, contents now:
		abc
		def

Just pushed zzz, contents now:
		abc
		def
		zzz

Just shifted abc, contents now:
		def
		zzz

Just unshifted AAA, contents now:
		AAA
		def
		zzz
```


garth@ggilmour.com