String Functions

In this Chapter, you will learn how to handle strings. AOZ has a full range of string manipulation instructions, and experienced Basic programmers should already be familiar with the standard syntax used.

Reading characters in a string

LEFT$

function: return the leftmost characters of a string

```
destination$=Left$(source$,number)
Left$(destination$,number)=source$
```

LEFT$ reads the specified number of characters in a source string, starting from the left-hand side, and copies them into a destination string. The first type of usage of this function creates a new destination string from the chosen number of characters of the source string. For example:

```
Do
    Input "Type in a string:";S$
    Print "Display how many characters from"
    Input "the left?";N
    Print Left$(S$,N)
Loop
```

The second type of usage replaces the leftmost number of characters in the destination string with the equivalent number of characters from the source string. For example:

```
A$="**** Basic"
Left$(A$,4)="AOZ"
Print A$
```

Exactly the same processes can be performed with characters from the right-hand side of a string, by using the equivalent RIGHT$ function.

RIGHT$

function: return the rightmost characters of a string

```
destination$=Right$(source$,number)
Right$(destination$,number)=source$
```

Here are two examples demonstrating each version of usage:

```
Print Right$("IGNORED54321",5)
A$=Right$("REJECTED0123456789",10)
Print A$
B$="AOZ ************"
Right$(B$,12)="Professsional"
Print B$
```

MID$

function: return a number of characters from the middle of a string

```
destination$=Mid$(source$,offset,number)
Mid$(destination$,offset,number)=source$
```

Similarly, the MID$ function returns characters from the middle of a string, with the first number specified in brackets setting the offset from the start of the string and the second number setting how many characters are to be fetched. If the number of characters to be fetched is omitted from your instruction, then the characters will be read right up to the end of the string being examined. Here are some examples:

```
Print Mid$("AOZ",6)
Print Mid$("AOZ",6,4)
A$="AOZ The best and easiest language ***"
Mid$(A$,19)="best"
Print A$
Mid$(A$,19,3)="language"
Print A$
```

Finding characters in a string

It is often necessary to search through a mass of data for a particular reference, in other words, to search through strings for individual characters or sub-strings. Similarly, you may wish to write an adventure game where lines of
text must be broken down into individual commands.

INSTR

function: search for occurrences of one string within another string

```
x=Instr(host$,guest$)
x=Instr(host$,guest$,start of search position)
```

INSTR allows you to search for all instances of one string inside another. In the following examples, the "host" strings are searched for the first occurrence of the "guest" strings you are seeking. If the relevant string is found, its location will be reported in the form of the number of characters from the left-hand side of the host string. If the search is unsuccessful, a result of zero will be given.

```
TODO

Print Instr("AMOS Professional","AMOS")
Print Instr("AMOS Professional","O")
Print Instr("AMOS Professional","o")
Print Instr("AMOS Professional","Provisional")
Do
    Input "Type in a host string:";H$
    Input "Type in a guest string to be found:";G$
    X=Instr(H$,G$)
    If X=0 Then Print G$;" Not found"
    If X<>0 Then Print G$;" Found at position ";X
Loop
```

Normally, the search will begin from the first character at the extreme left-hand side of the host string, but you may begin searching from any position by specifying an optional number of characters from the beginning of the host string. The optional start-of-search position can range from zero to the maximum number of characters in the host string to be searched. For example:
`
TODO
``
Print Instr("AOZ","O",0)
Print Instr("AOZ","O",4)

```


## Converting strings

## **_UPPER$_**

__function__: convert a string of text to upper case
```

new$=Upper$(old$)

```
This function converts the characters in a string into upper case (capital)
letters, and places the result into a new string. For example:
TODO
```

Print Upper$("aMoS pRoFeSsIoNaL")

```


## **_LOWER$_**

__function: convert a string of text to lower case
```

new$=Lower$(old$)

```
This works in the same way as UPPERS, but translates all the characters in a
string into nothing but lower case (small) letters. These sorts of text
conversions are particularly useful for interpreting user-input in interactive
data programs and adventure games, because input can be converted into a
standard format which is understood by your programs. For example:
```

Input "Do you want to continue? (Yes or No)";ANSWER$
ANSWER$=Lower$(ANSWER$) : If ANSWER$="no" Then Edit
Print "OK. Continuing with your program"

```
## **_STR$_**

__function__: convert a number into a string
```

s$=Str$(number)

```
Str$ converts a real number variable into a string. This can be used to overcome
limitations posed by functions like CENTRE, which does not accept numbers as
parameters, but will work happily with parameters in the form of strings. Here
is an example:
```

Centre "Remaining memory is"+Str$(Chip Free)+" Bytes"

```
TODO - rest of this page
## **_VAL_**

__function__: convert a string of digits into a number
```

v=Val(x$)
v#=Val(x$)

```
To perform the reverse task to STR$, the VAL function converts a list of decimal
digits stored in a string, changing them into a number. If this process fails
for any reason, a value of zero will be returned. For example:
```

X=Val("1234") : Print X

```
## **_STRING$_**

__function__: create a new string from an existing string
```

new$=String$(existing$, number)

```
Do not confuse this with STR$, which converts numbers into a string. The STRING$
function creates a new stringf illed with the required number of copies of the
first character from an existing string. For instance, the following example
produces a new string containing ten copies of the character "A".
```

Print String$("AOZ The Best And Easiest Language",10)

```
## Manipulating strings
Sometimes you may want to handle your strings for special purposes. For example,
if you wish to pad out a piece of text before it gets printed onto the screen,
you will need an accurate method of creating spaces in the string.

## **_SPACE$_**

__function__: space out a string
```

s$=Space$(number of spaces)

```
Try the following example:
```

Print "Ten";Space$(10);"spaces"

```
## **_FLIP$_**

__function__: invert a string
```

inverted$=Flip$(original$)

```
This function simply reverses the order of the characters held in an existing
string. For example:
```

Print Flip$("SOMA gnippilf")

```
## **_REPEAT$_**

__function__: repeat a string
```

r$=Repeat$(text$,number)

```
To repeat the same string of characters using a single PRINT statement, follow
your string of text with the number of times you want the repetition. Allowable
values are between 1 and 127. Whenever the string is printed, a sequence of
control characters is automatically added to the r$ variable, in the following
format:
```

Chr$(27)+"RO"+A$+Chr$(27)+"R"+Chr$(48+n)

```
## Getting information about strings

The next three functions are provided to discover particular properties of
strings.

## **_CHR$_**

__function__: return the character with a given ASCII code
```

s$=Chr$(code number)

```
The CHR$ function creates a string that contains a single character generated by
a given ASCII code number. Note that only the characters with ASCII code numbers
32 to 255 are printable on the screen. Others are used internally as control
codes. Match characters with their codes using this routine:
```

For S=32 To 255: Print Chr$(S); : Next S

```
## **_ASC_**

__function__: Give the ASCII code of a character
```

code=Asc(a$)

```
To get the internal ASCII code of the first character in a string, use the ASC
function like this:
```

Print Asc("B")
Print Asc("AMOS Professional")

```
## **_LEN_**

__function__: give the length of a string
```

length=Len(a$)

```
The LEN function returns the number of characters stored in a string. For
example:
```

Print Len("0123456789")

```
## Array operations

To end this Chapter, here are a pair of useful instructions for manipulating
arrays.

## **_SORT_**

__instruction__: sort all elements in an array
```

Sort a(0)
Sort a#(0)
Sort a$(0)

```
The SORT instruction arranges the contents of any array into ascending order,
and the array may contain integers, floating point numbers or strings.

The starting point of your table is specified by the a$(0) parameter, and it
must always be set to the first item in the array, which is item number zero.
For example:
```

```
N=5 : P=0
Dim A(N)
Print "Type in ";N," numbers, or enter 0"
Print "to stop entry and begin sort"
Repeat
   Input A(P)
   If A(P)=0
      Dec P
      Exit
   End If
   If P=N-1 Then Exit
   Inc P
Until False
Sort A(0)
For X=N-P To N
   Print A(X)
Next X
```

```
## **_MATCH_**

__function__: search an array for a value
```

x=Match(array(0),value)
x=Match(array#(0),value#)
x=Match(array$(0),value$)

```
MATCH searches through an array that has already gone through the SORT process,
looking for a given value. If the value is found then x is loaded with the
relevant index number. However, if the search is not successful the result will
be negative. If you take the absolute value of this result, the item which came
closest to your original search parameter is provided. Only arrays with a single
dimension can be checked in this way, and they must already be sorted before
MATCH can be called.
For example:
```

```
Read N : Dim D$(N)
For X=0 To N-1 : Read D$(X) : Next X
Sort D$(0)
Do
  REINPUT:
  Input A$
  If A$=" "Then End
    If A$="print all data"
    For X=1 To N: Print D$(X) : Next X: Goto REINPUT
  End If
  POS=Match(D$(0),A$)
  If POS<-N-1
    If POS>-10
      Print "Not found. Nearest to ";D$(1) : Goto JMP
    Else
      Print "Not found. Nearest to ";D$(N) : Goto JMP
    End if
  End If
  If POS>0 Then Print "Found ",DS(POS);" in record ";POS
  If POS<0 Then Inc POS : Print "Not found. Nearest to ":DS(Abs(POS))
  JMP:
Loop
Data 8,"Mercury","Venus","Earth","Mars","Saturn","Jupiter","Neptune","Tharg"
```

```
Test that example out by entering various inputs, including the names of
planets, single characters in upper and lower case and "print all data".
Obviously MATCH can be used with the INSTR function to set up a powerful parser
routine, for interpreting user input in an adventure game.
```