ESCI 386 – IDL Programming for Advanced Earth Sciences Applications
Lesson 3 – Strings

Reading: None

STRING VARIABLES AND CONSTANTS

- String variables and constants consist of text and special characters.
- Values are assigned to a string variable or constant by enclosing the text in either single or double quotes.
  - With ONE IMPORTANT EXCEPTION, it doesn’t matter which ones you use, as long as you don’t use one at the beginning and a different one at the end of the string.
  - IMPORTANT EXCEPTION – A double quote should never be used before a digit, unless you are representing octal numbers.
    - In IDL, “056” is the octal number 56, while “056” is not a valid IDL expression.
    - Examples
      - `help, “056” => <Expression> INT = 46`
      - `help, “056” => “Syntax Error”`
  - To avoid this ever becoming an issue, it is probably best to use single quotes as much as possible (plus, they are easier to type since you don’t have to hit the shift key.)
- If your text needs to include quotes, then either repeat the quote within the string, or use the other type of quote to define your string.
  - Examples:
    - “Alex’s Text”
    - ‘Alex’s Text”
    - “I said, “”NO!”””

- Strings can be up to 32,767 characters in length.

ASCII VALUES

- ASCII stands for American Standard Code for Information Exchange
- ASCII is a way of representing text characters using numbers.
- You can assign ASCII characters to an IDL string by using the `string()` function and using the ASCII code in byte format as the argument.
• Not all available ASCII characters may actually print on your screen. It depends on which fonts are used by your system.

• You can convert a string character to its ASCII value by using the character as an argument to the BYTE function (oddly enough, this actually converts it to a BYTE array with a single element).
  o Example:
    ▪ print, string(102b) => “f”
    ▪ print, string(102) => “102”
    ▪ print, byte(“f”) => 102
    ▪ help, byte(“f”) => <Expression>    BYTE      = Array[1]

USEFUL STRING FUNCTIONS AND OPERATIONS
• IDL has many useful string functions and procedures.
  o You’re probably not going to use IDL to do heavy manipulation of strings, as this is easier done in modern scripting languages.
  o But you will want to know how to do basic string operations, particularly for creating dynamic captions and titles for figures.

• Concatenation of strings refers to adding strings together.
  o Concatenation is done simply by using the plus sign.
  o Example: “Alex ” + “DeCaria” => “Alex DeCaria”

• The table below shows some of the string functions available in IDL.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| strcompress | • Eliminates or compresses whitespace.  
  • The /remove_all keyword removes all whitespace.  
  • Without /remove_all keyword, all whitespace is compressed into a single space.  
  • Example:  
    a = “ String with whitespace ”  
    strcompress(a) => “ String with whitespace ”  
    strcompress(a, /remove_all) => “Stringwithwhitespace” |
| strtrim   | • Eliminates either leading whitespaces, trailing whitespaces, or both, depending on the optional flag character.  
  • Unlike strcompress, which removes or compresses all |
whitespace.

**strcmp**

- Compares two strings to see if they are equivalent.
- It returns a BYTE value of 1 if true, and a BYTE value of 0 if false.
- It can be used to compare the entire strings, or only the first *n* characters of the strings.
- The optional `/FOLD_CASE` keyword can be used for case-insensitive comparisons.
- **Example:**
  
  ```
  a = "hi"; b = "Hi"; c = "Hilo"
  
  strcmp(a, b) => 0
  strcmp(a, b, /FOLD_CASE) => 1
  strcmp(a, c) => 0
  strcmp(b, c, 2) => 1
  ```

  Strings can also be compared using the `EQ` comparison operator, but this compares the entire string, and is case-sensitive.

**strlowcase/strucase**

- Convert the contents of a string to all lower case or all upper case respectively.
- **Example:**
  
  ```
  a = "hi"; b = "Hi"; c = "Hilo"
  
  strlowcase(b) => "hi"
  strucase(c) => "HIL0"
  ```

**strmatch**

- This function can compare two strings using wildcards.

**strmid**

- Returns a substring of a string, by specifying the first character and length of the substring to be extracted.

**strpos**

- Returns the position of the first occurrence of a search string within a target string.

**strput**

- Inserts characters into an existing string.

**strlen**

- Returns the string length.

**strsplit**

- Splits a string based on a given delimiter.

**strjoin**

- Collapses a one-dimensional string array into a single string, or an *n*-dimensional string array into an (*n*-1)-dimensional string array.

**INSERTING TEXT INTO A STRING**

- Inserting text into a string in IDL is a pain, because it overwrites it the existing text.
  
  ```
  a = "Alex DeCara is a teacher."
  strput, a, "i", 10 => "Alex DeCari is a teacher."
  ```

- One way to overcome this is to break the string into two substrings around where you want to insert a character
  
  ```
  b = strmid(a, 0, 10)
  c = strmid(a, 10, 15)
  ```
print, b => "Alex DeCar"
print, c => "a is a teacher."

and then concatenate the two substrings sandwiching the text you want to insert.
d = b + "a" + c

print, d => "Alex DeCaria is a teacher."