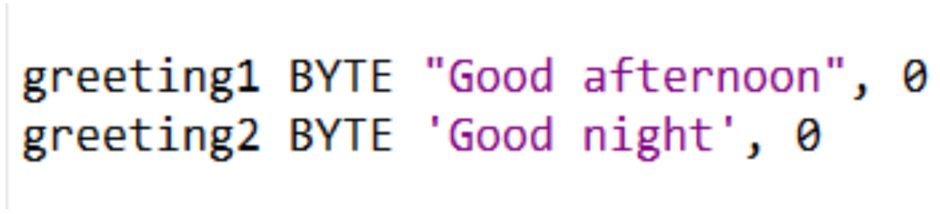
Strings in Assembly

Let's break down and clarify the information about defining strings in assembly language:

• Defining Strings: To define a string of characters in assembly language, you enclose the characters within either single or double quotation marks. The most common type of string is one that ends with a null byte, which has a value of 0. This type of string is known as a null-terminated string and is used in many programming languages.

• Example String Definitions:



```
greeting1 BYTE "Good afternoon", 0
greeting2 BYTE 'Good night', 0
```

greeting1 BYTE "Good afternoon",0: This defines a null-terminated string named greeting1 containing the text "Good afternoon."

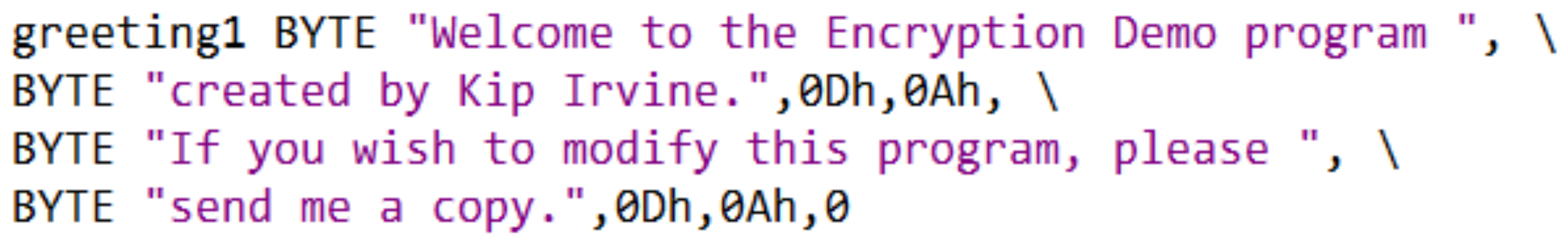
greeting2 BYTE 'Good night',0: Similarly, this defines another null-terminated string named greeting2 containing the text 'Good night.'

• Character Storage: Each character in a string uses a single byte of storage. However, unlike other data types where byte values must be separated by commas, strings are an exception. You can directly define strings with their characters enclosed in quotation marks.

• Multi-line Strings: You can also define multi-line strings without the need for separate labels for each line. Hexadecimal codes like 0Dh and 0Ah are used to represent CR/LF (carriage-return line-feed) or end-of-line characters in the string. These characters are used to move the cursor to the left column of the next line when output to standard output.

• Line Continuation: The line continuation character \ is used to concatenate two source code lines into a single statement. It must be the last character on the line. This allows you to break long strings across multiple lines for better readability without introducing unnecessary labels.

For example, you can define a multi-line string like this:



```
greeting1 BYTE "Welcome to the Encryption Demo program ", \
BYTE "created by Kip Irvine.",0Dh,0Ah, \
BYTE "If you wish to modify this program, please ", \
BYTE "send me a copy.",0Dh,0Ah,0
```

How do such strings fit in a byte?? yet you say each character literal is stored in a byte??

In assembly language, each character in a string is indeed stored in a byte. This means that each character is represented by a single byte of memory, where the byte contains the ASCII code for that character.

The reason null-terminated strings fit in a byte is because they are arrays of bytes, with each byte representing a single character. The null terminator, which marks the end of the string, is also a single byte with a value of 0.

So, when you define a null-terminated string like this:

```
greeting1 BYTE "Good afternoon", 0
```

It's essentially defining an array of bytes where each character ('G', 'o', 'o', 'd', ' ', 'a', 'f', 't', 'e', 'r', 'n', 'o', 'o', 'n') is stored in a separate byte, and the null terminator (0) is placed at the end to indicate the end of the string.

In summary, strings in assembly are arrays of bytes where each byte represents a character, and a null byte (0) is used to mark the end of the string.

So, in short, we write BYTE, coz we are seeing the whole string as a series of bytes.

```
names1 DB "Learning Assembly then WinAPI", 0
names2 BYTE "Learning Reverse Engineering then C#", 0
names3 BYTE "Learning to be good as C programming", 0
```

So, the three will assign the string to names 1,2,3 variables/labels, and each character stored in its own byte in memory?

- names is the label or variable name.

- DB indicates that you are defining a sequence of bytes.
- The string "Learning Assembly then WinAPI" is enclosed in double quotation marks and is assigned to the names variable.

Each character in the string is stored in its own byte in memory. So, the string is effectively broken down into individual bytes, with each byte holding the ASCII code for a character, and it ends with a null byte (0) to mark the end of the string. This allows you to work with and manipulate the string using individual byte-sized elements in memory.