Programming with C

Strings

Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

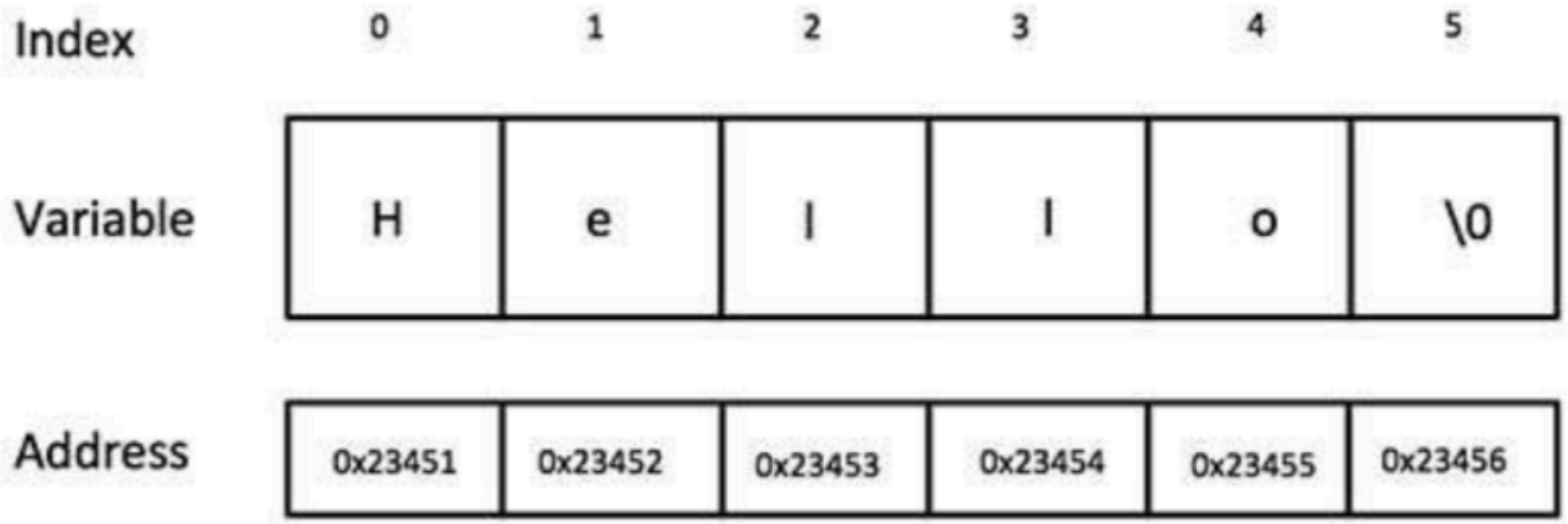
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows:

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C :



Actually, you do not place the null character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array.

Let us try to print the above mentioned string:

```
#include <stdio.h>

int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   printf("Greeting message: %s\n", greeting );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Greeting message: Hello
```

Read String from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters a whitespace(space, newline, tab etc.).

```c
#include <stdio.h>
int main()
{

        char name[20];
         printf("Enter name: ");
         scanf("%s", name);

         printf("Your name is %s.", name);

return 0;

 }
```

C supports a wide range of functions that manipulate null-terminated strings:

S.N.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

The following example uses some of the above-mentioned functions:

```
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];
int  len ;

/* copy str1 into str3 */
strcpy(str3, str1);
printf("strcpy( str3, str1) :  %s\n", str3 );

/* concatenates str1 and str2 */
strcat( str1, str2);
printf("strcat( str1, str2):   %s\n", str1 );

/* total lenghth of str1 after concatenation */
len = strlen(str1);
printf("strlen(str1) :  %d\n", len );
```

When the above code is compiled and executed, it produces the following result:

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

Passing Strings to Function

Strings can be passed to a function in a similar way as arrays. Learn more about passing array to a function.

```c
#include <stdio.h>
void displayString(char str[]);
int main()
{
    char str[50];
    printf("Enter string: ");
    scanf("%s",str);

displayString(str); // Passing string to a function.

return 0;
}
```

```c
void displayString(char str[])
{

    printf("String Output: %s",str);


}
```

Example: Program to count vowels

```
#include <stdio.h>
int main()
{
    char line[150];
    int i, vowels;

     vowels = consonants = digits =  0;

     printf("Enter a line of string: ");
     scanf("%s", line);

       for(i=0; line[i]!='\0'; ++i)
         {
                 if(line[i]=='a' || line[i]=='e' || line[i]=='i' || line[i]=='o' || line[i]=='u' || line[i]=='A' || line[i]=='E' ||  line[i]=='I'
|| line[i]=='O' || line[i]=='U')
{
     ++vowels;
}
}
       printf("Vowels: %d",vowels);
return 0 ;
}
```