String

Strings are character array.
Character array are stored in contiguous memory location.

String Constant:
- A string constant is a one-Dimensional array of characters terminated by a null character(\0).
- Each character in the array occupies one byte of memory and the last character is always \0
Example:
  char str[]={'C','P','U','\0'};
- The terminating null character(\0) is very important in a string because its the only way for the compiler to know where the string ends.

Null Character(\0):
- \0 is called null character.
- \0 and 0 are not same because both has different ASCII value. ASCII value of \0 is 0 but ASCII value of 0 is 48.

String Initialization:
- A string can be initialized without adding the terminating null character as,

  char str[]="WELCOME";

Here C inserts the NULL character automatically.

Accessing Character array elements:
- Similar to integer array we can access array elements
#include< stdio.h >
int main()
{
  int i=0;
  char str[]="WELCOME";
  while(str[i]!="\0")
  {
    printf("%c",str[i]);
    i++;
  }
  return;
}

This can be done by using pointer also. Mentioning th name of the array we get the base address(zeroth element) of the array.
#include< stdio.h >
int main()
{
  char str[]="WELCOME",*ptr;
  ptr=str;
while(*ptr!='\0')
{
    printf("%c",*ptr);
    ptr++;
}
return;

**Reading a string from a keyboard:**
- For reading and writing a string the format specifier is %s and no & is needed

```c
#include< stdio.h >
int main()
{
    char str[30];
    printf("Enter the string \n");
    scanf("%s",str);
    printf("String=%s",str);
    return;
}
```

The `scanf()` function fills in the character typed at keyboard into the `str[]` array until the blank space or enter key is hit. If blank space or enter key is hit `scanf()` automatically place a '\0' in the array.

- Normally the %s will read a string upto a blank space. To read a multi word string untill the end of line including blank space use %[^\n]s

```c
#include< stdio.h >
int main()
{
    char str[100];
    printf("ENter the string \n");
    scanf("%[^\n]s", str);
    printf("String=%s",str);
    return;
}
```

Output:
Enter the string
welcome honey
String=Welcome honey

- Other way of reading multiword string is `gets()` and `puts()`. But it is dangerous to use `gets()` in programs so try to avoid `gets()`

```c
#include< stdio.h >
int main()
{
    char str[30];
    printf("Enter the string \n");
    gets(str);
    puts(str);
```
\textbf{Pointer and String:}

- A string can be stored in 2 forms
  
  1. \texttt{char str[]}="Welcome"; //here Welcome is stored in a location called str
  2. \texttt{char *str="Welcome";} //here Welcome is stored in some other location in memory and assign the address of the string in a char pointer

- The advantage of pointing a string using a character pointer is we cannot assign a string to another string but we can assign a char pointer to another char pointer

\begin{verbatim}
#include< stdio.h >
int main()
{
    char str1[]="Welcome",str2[30];
    char *ptr1="Welcome",*ptr2;
    //str2=str1; //error
    ptr2=ptr1;
    printf("ptr2=%s",ptr2);
    return;
}
\end{verbatim}

- Once a string has been defined it cannot be initialized to another set of characters but using char pointer we can redefine the string.

\begin{verbatim}
#include< stdio.h >
int main()
{
    char str1[]="Welcome";
    char *ptr1="Welcome";
    // str1="Good";//error
    ptr1="Good";
    printf("ptr1=%s",ptr1);
    return;
}
\end{verbatim}

\textbf{Points to consider in Strings:}

1. The size of the string depends on the size of declaration not the number of characters in the string

\begin{verbatim}
#include <stdio.h>
void main()
{
    char str[30]="Network programming";
    printf("%d",sizeof(str));
}
\end{verbatim}

The output here is 30 not 1
Source:

http://datastructuresprogramming.blogspot.in/2010/06/string.html