Storage for Strings in C

In C, a string can be referred either using a character pointer or as a character array.

Strings as character arrays

```
char str[4] = "GfG"; /*One extra for string terminator*/
/*    OR    */
char str[4] = {'G', 'f', 'G', '\0'}; /* '\0' is string terminator */
```

When strings are declared as character arrays, they are stored like other types of arrays in C. For example, if str[] is an auto variable then string is stored in stack segment, if it's a global or static variable then stored in data segment, etc.

Strings using character pointers
Using character pointer strings can be stored in two ways:

1) Read only string in a shared segment.
When string value is directly assigned to a pointer, in most of the compilers, it's stored in a read only block (generally in data segment) that is shared among functions.

```
char *str  =  "GfG";
```

In the above line "GfG" is stored in a shared read only location, but pointer str is stored in a read-write memory. You can change str to point something else but cannot change value at present str. So this kind of string should only be used when we don't want to modify string at a later stage in program.

2) Dynamically allocated in heap segment.
Strings are stored like other dynamically allocated things in C and can be shared among functions.

```
char *str;
int size = 4; /*one extra for '\0'*/
str = (char *)malloc(sizeof(char)*size);
*(str+0) = 'G';
*(str+1) = 'f';
*(str+2) = 'G';
*(str+3) = '\0';
```

Examples to better understand above ways to store strings.

Example 1 (Try to modify string)
The below program may crash (gives segmentation fault error) because the line *(str+1) = 'n' tries to write a read only memory.

```
int main()
{
 char *str;
 str = "GfG";      /* Stored in read only part of data segment */
 *(str+1) = 'n'; /* Problem:  trying to modify read only memory */
 getchar();
```

```
 return 0;
}
```

Below program works perfectly fine as str[] is stored in writable stack segment.

```
int main()
{
 char str[] = "GfG";  /* Stored in stack segment like other auto variables
*/
 *(str+1) = 'n';   /* No problem: String is now GnG */
 getchar();
 return 0;
}
```

Below program also works perfectly fine as data at str is stored in writable heap segment.

```
int main()
{
  int size = 4;

  /* Stored in heap segment like other dynamically allocated things */
  char *str = (char *)malloc(sizeof(char)*size);
  *(str+0) = 'G';
  *(str+1) = 'f';
  *(str+2) = 'G';
  *(str+3) = '\0';
  *(str+1) = 'n';  /* No problem: String is now GnG */
   getchar();
   return 0;
}
```

Example 2 (Try to return string from a function)
The below program works perfectly fine as the string is stored in a shared segment and data stored remains there even after return of getString()

```
char *getString()
{
  char *str = "GfG"; /* Stored in read only part of shared segment */

  /* No problem: remains at address str after getString() returns*/
  return str;
}

int main()
{
  printf("%s", getString());
  getchar();
  return 0;
}
```

The below program also works perfectly fine as the string is stored in heap segment and data stored in heap segment persists even after return of getString()

```
char *getString()
{
  int size = 4;
  char *str = (char *)malloc(sizeof(char)*size); /*Stored in heap segment*/
```

```
  *(str+0) = 'G';
  *(str+1) = 'f';
  *(str+2) = 'G';
  *(str+3) = '\0';

  /* No problem: string remains at str after getString() returns */
  return str;
}
int main()
{
  printf("%s", getString());
  getchar();
  return 0;
}
```

But, the below program may print some garbage data as string is stored in stack frame of function getString() and data may not be there after getString() returns.

```
char *getString()
{
  char str[] = "GfG"; /* Stored in stack segment */

  /* Problem: string may not be present after getSting() returns */
  return s
int main()
{
  printf("%s", getString());
  getchar();
  return 0;
}
```