Using pointers in structures

There is an advantage in using a character pointer in a structure as a string container over using a character array for the same purpose.

L-values are objects whereas R-values are data.

An entire string can only be assigned to a **char** array upon its declaration. The only way to subsequently assign a string to a **struct char** array using the = operator is to assign individual characters to one element at a time.

In each assignation the value to the left of the = operator is known as the "L-value", representing the memory Location, whereas the value to the right of the = operator is known as the "R-value", representing the data to Read into that location.

One important rule in C programming is that an R-value cannot appear on the left side of an = operator. An L-value, on the other hand, may appear on either side of an = operator.

Each individual element of a **char** array is an L-value to which an individual character may be assigned, but a **char** pointer is also an L-value to which an entire string may be assigned after that pointer has been declared.

strmbr.c

1. Begin a new program with a preprocessor instruction to include the standard input/output library functions
```
#include <stdio.h>
```

2. Next, declare an un-named structure defining a data type with a single character array member and one tag name
```
typedef struct
{
  char str[5] ;
} ArrType ;
```

3. Now, declare an un-named structure defining a data type with a single character pointer member and one tag name
```
typedef struct
{
  char *str ;
} PtrType ;
```

4. Then, declare one variable of each structure-defined data type, initializing the single member of each variable
```
ArrType arr = { 'B', 'a', 'd', ' ', '\0' } ;
PtrType ptr = { "Good " } ;
```