Working with Strings

Topics

- String Data In C
- Various Library Functions
- Strings As Function Arguments

Text Data

- Textual data in C is based on the ASCII table
  - Http://www.Asciitable.Com
- Characters are stored in one byte (8-bits)
- As A result, only 256 possible characters exist in C limiting us to just the north American keyboard letters
- New languages use A better text standard called UNICODE which enables lots of different kinds of character sets and alphabets

Strings In The C Language

- C strings are arrays of characters
- C strings are, by convention, always null terminated
  - NULL is a special symbol: '\0'
- C string arrays are partially filled arrays
  - You MUST leave enough space for the NULL!

Book Reading

- Read section 24.2 before moving on. It's a short section.

C Strings

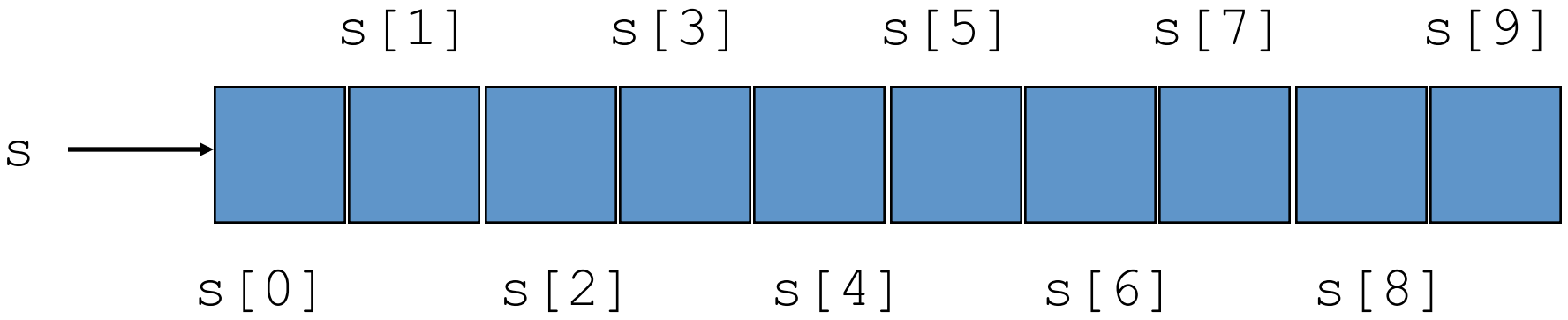
- Example:

```
char s[ 10 ];
```

C Strings

- Example:

```
char s[ 10 ];
```



C Strings

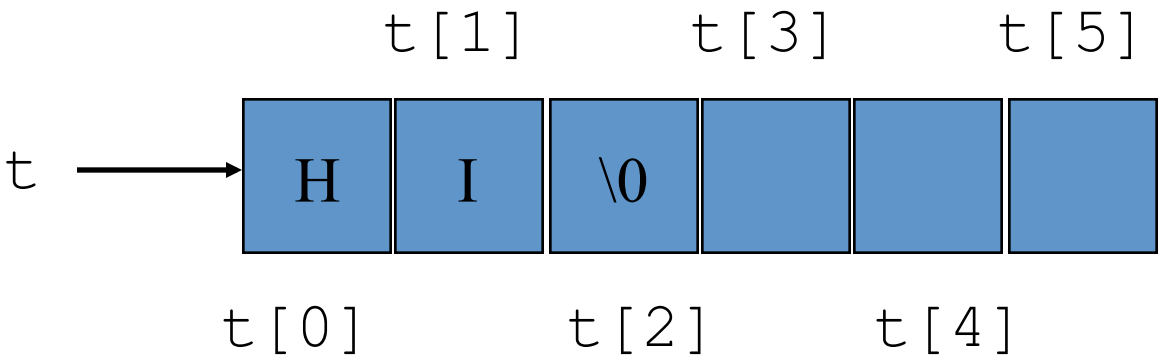
- Example:

```
char t[ 5 ] = “HI”;
```

C Strings

- Example:

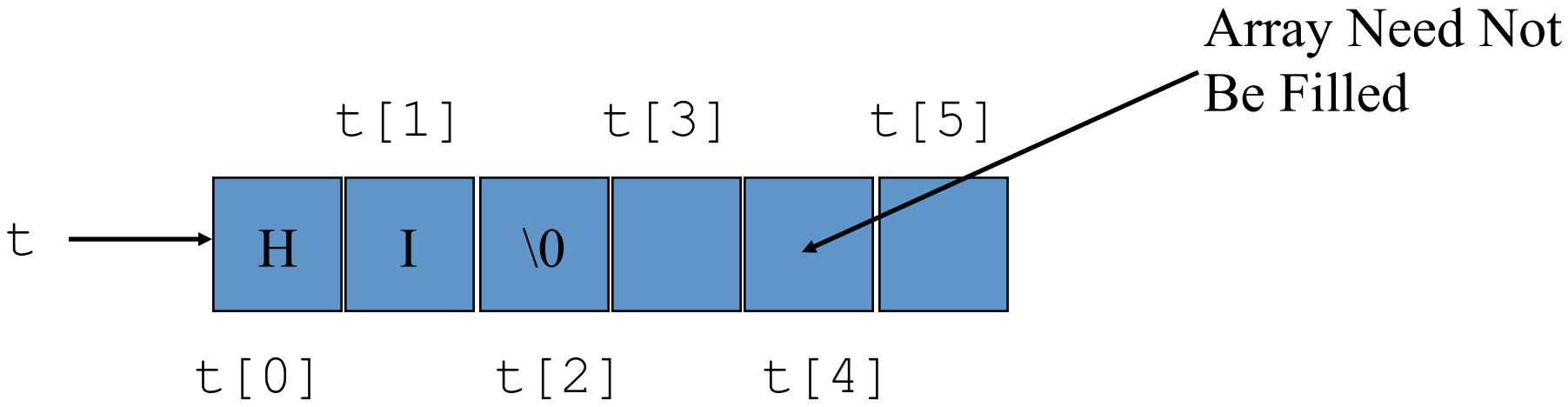
```
char t[ 6 ] = "HI";
```



C Strings

- Example:

```
char t[ 6 ] = "HI";
```



C Strings

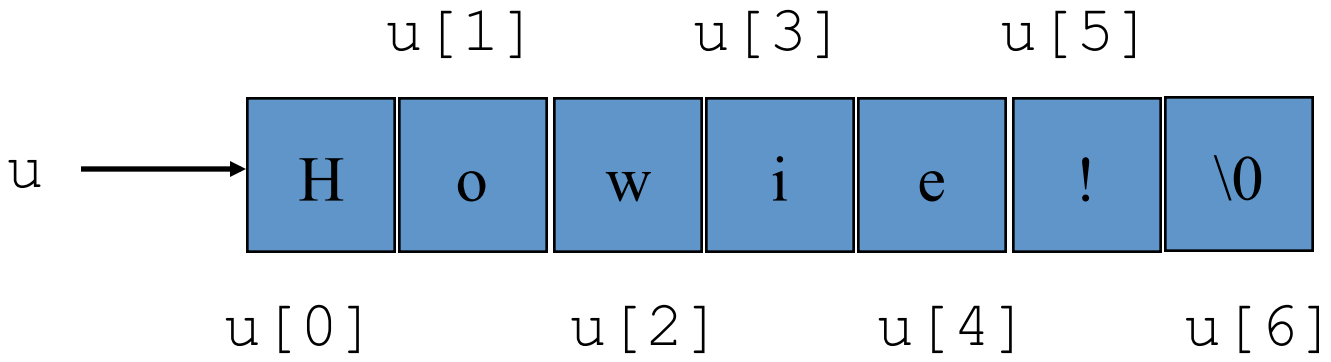
- Example:

```
char u[] = “Howie!”;
```

C Strings

- Example:

```
char u[] = “Howie!”;
```

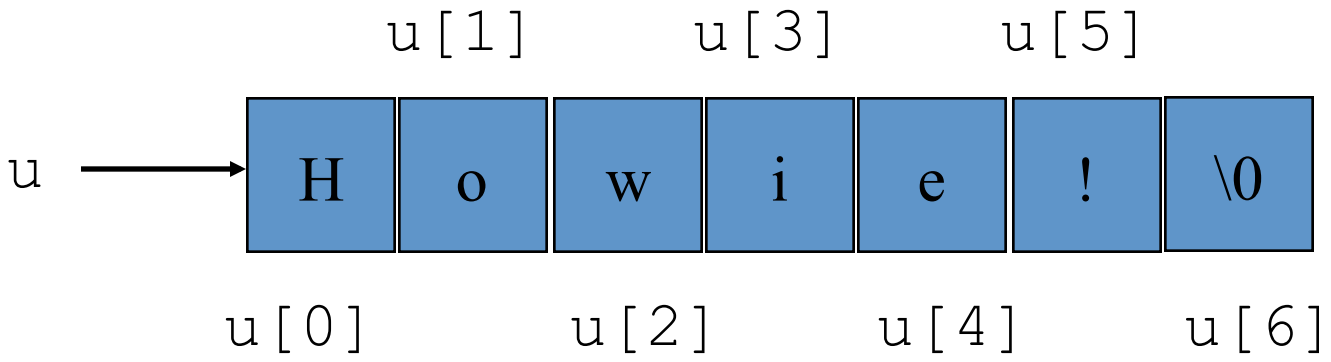


C Strings

- Example:

```
char u[] = "Howie!";
```

Size is not required when you initialize



C Strings Observations

- C strings are implemented as arrays of char
- The sentinel \0 is vital to working with strings in C.
- Why? The standard compiler has no functions to measure the length of a string.
- <string.h> must be added
- The above library contains functions that allow us to “manipulate” strings

But why do we need '\0'

- Imagine x to be a string variable but not required to be null terminated.
- If you are to count how many characters in x, how do you know when to stop?
  - It's a pointer, so you can't see where it ends
- Using null, all c string processing functions can then "find" the end of the string. This is vital for any language to process strings.

C Strings Observation #1

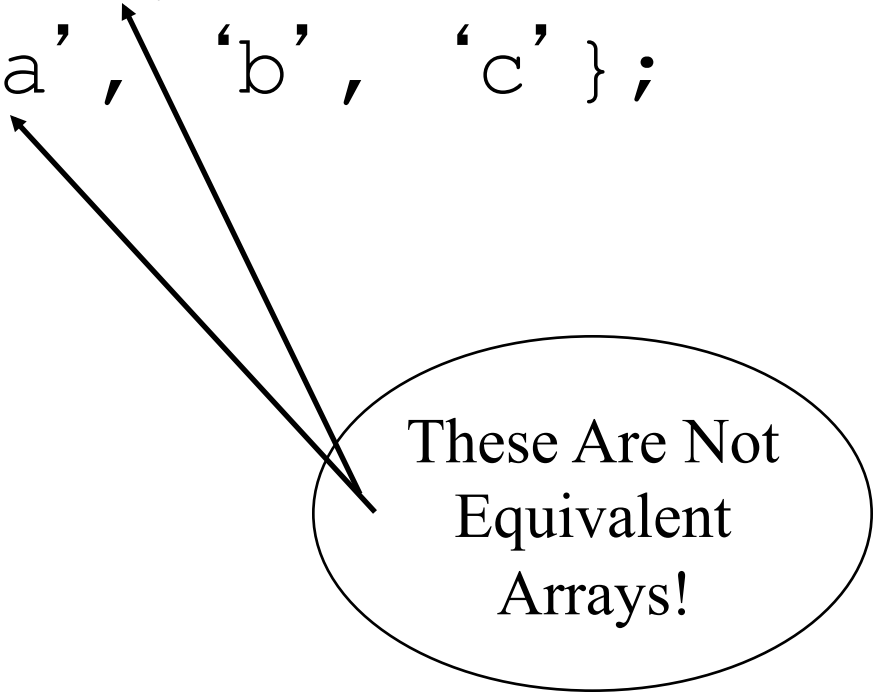
- Example:

```
char v[] = “abc”;
char w[] = {‘a’, ‘b’, ‘c’};
```

C Strings Observation #1

- Example:

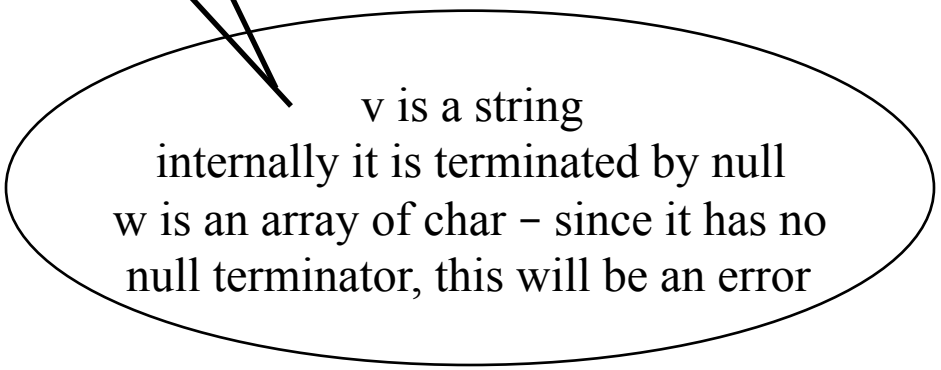
```
char v[] = “abc”;
char w[] = {‘a’, ‘b’, ‘c’};
```



C Strings Observation #1

- Example:

```
char v[] = “abc”;
char w[] = {‘a’, ‘b’, ‘c’};
```



C String Observation #2

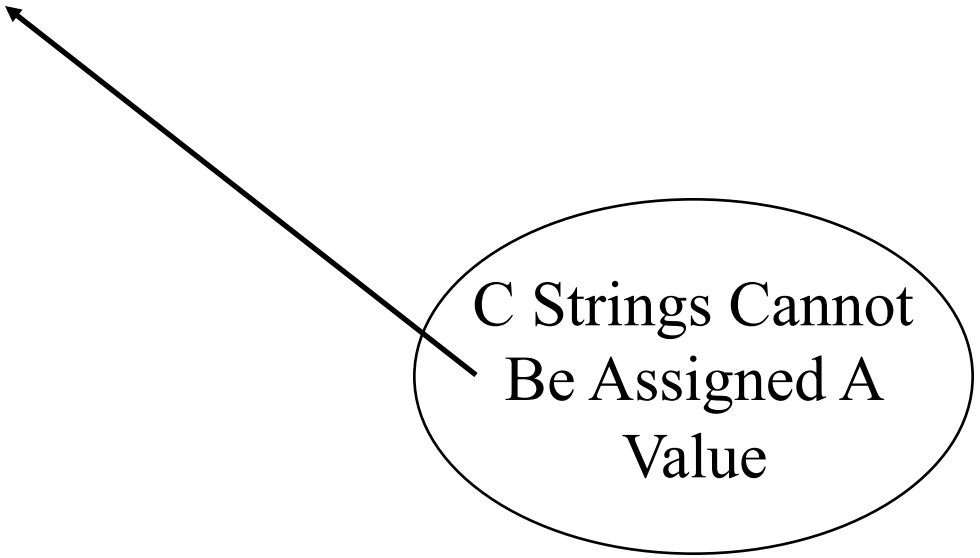
- Example:

```
char x[ 5 ];
x = “Foo”;
```

C String Observation #2

- Example:

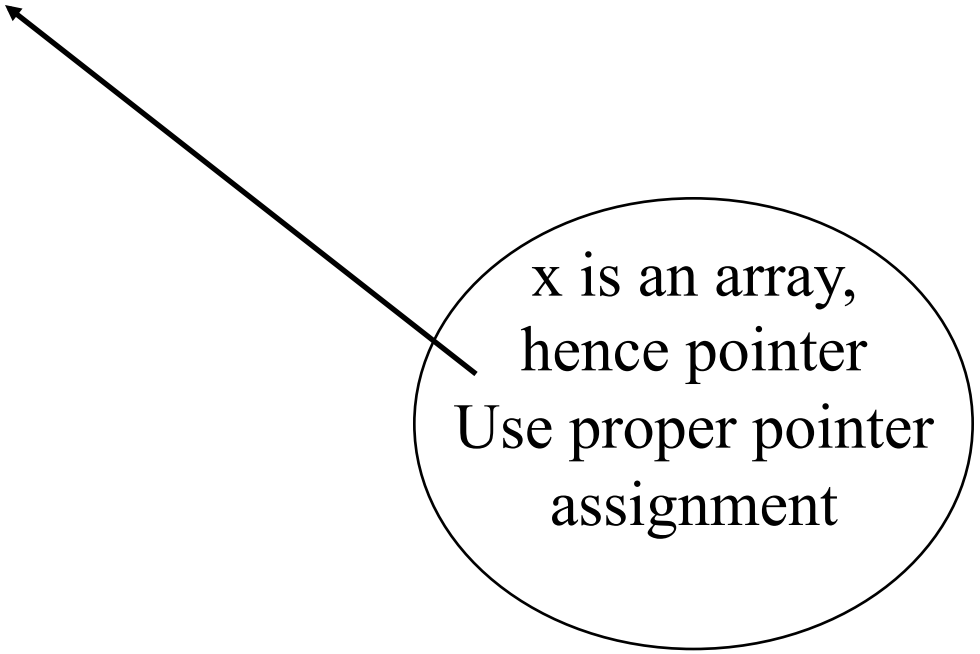
```
char x[ 5 ];
x = “Foo”;
```



C String Observation #2

- Example:

```
char x[ 5 ];
x = “Foo”;
```



A String Is An Array

- You can use an index to walk the array
- Example:

```
char lots_of_x[] = “Hello World”;

int index = 0;
while (lots_of_x[index] != ‘\0’) {
   lots_of_x[ index ] = ‘x’;
   index++;
}
//what will x be after the loop?
```

What can we do with strings

```
#include <stdio.h>          ←note we need this for input and output
#include <string.h>         ← we need this for working with strings
int main()
{
   /* String Declaration*/
   char yourname[20];

   printf("Enter your name:");

   /*getting input string and storing it in yourname*/
   scanf("%s", &yourname);  //.net likes scanf_s to avoid running into runtime error if input is too long

   /*Displaying String*/
   printf("%s",yourname);   ← note %s is used to output a string

   return 0;
}
```

Various Library Functions

- The standard system library provides various string manipulation routines

FUNCTION	MEANING	ARGUMENTS	RETURNS
strcpy(dest, src)	dest = src	cstring, cstring	void
strcat(dest, src)	dest = dest + src	cstring, cstring	void
strlen(src)	length of cstring src	cstring	int
strcmp(s1, s2)	compares s1 and s2	cstring, cstring	int

Various Library Functions

- `strlen( char s[] ) :: int`
  - returns length of s NOT including \0
- `strcmp(char s[],char t[])::int`
  - returns 0 if s equals t lexicographically
  - returns <0 if s is lexicographically less than t
  - returns >0 if s is lexicographically greater than t
  - NOTE: `0` equates to `false` in expressions

FYI - Various Library Functions

- The Standard System Library Provides Various Conversion Routines

FUNCTION	MEANING	ARGUMENTS	RETURNS
atoi(src)	ascii to integer	cstring	int
atof(src)	ascii to floating-point	cstring	double
atol(src)	ascii to long	cstring	long

C Strings – Watch Outs

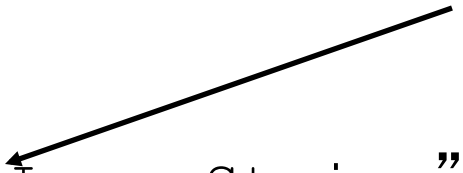
- Example:

```
char var[3];
strcpy( var, “A Nice Long String” );
```

C Strings Assignment

- Example:

```
char var[3];
strcpy( var, “A Nice Long String” );
```

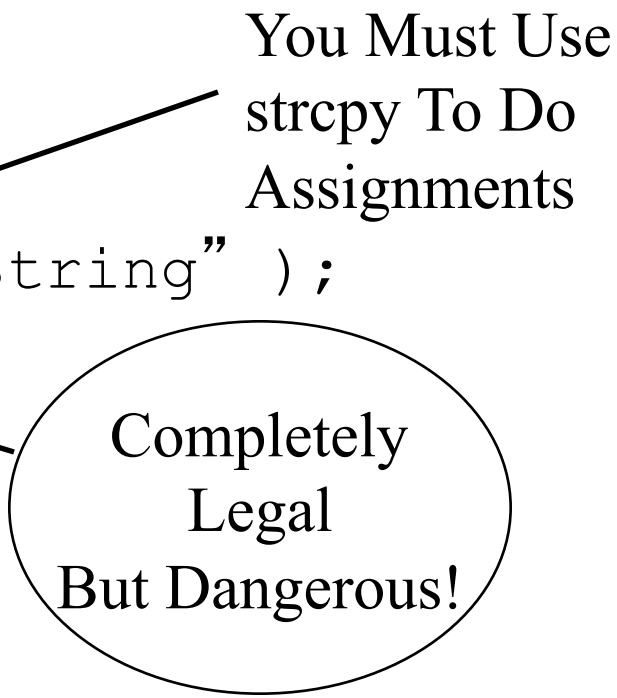


You Must Use strcpy To Do Assignments

C Strings – Watch Outs

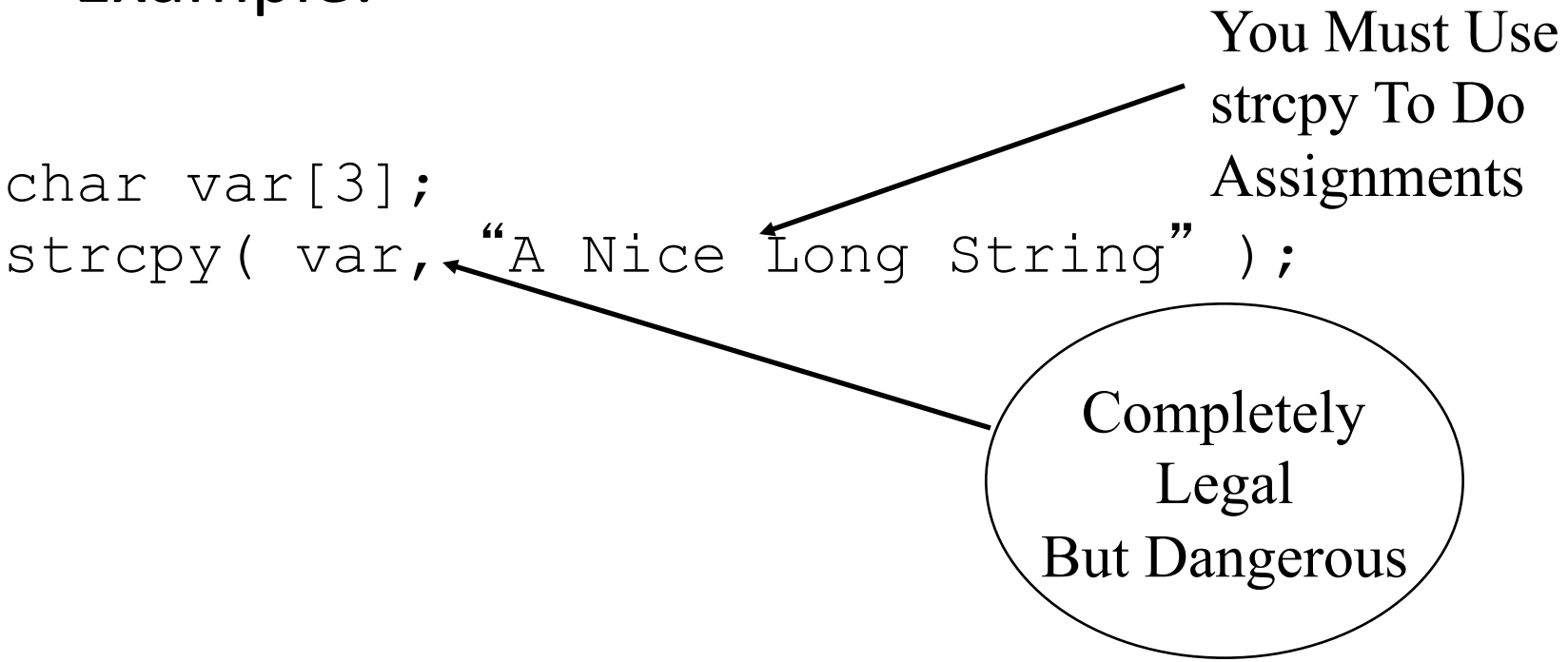
- Example:

```
char var[3];
strcpy( var, “A Nice Long String” );
```



C Strings – Watch Outs

- Example:



It is always the programmer's responsibility to ensure That the destination is large enough for the string Being placed there

Strings As Function Arguments

- Like other array variables, strings can be passed to functions
- Generally, it's a good idea to provide an argument that specifies the maximum string size allowed
- Built-in functions can use the \0 sentinel to determine the end of the string

Understanding String Parameters

- Built-in functions can work off the \0 sentinel embedded in string
- Functions updating a string variable should be provided A maximum size allowed value as a parameter
- Use `strcmp` **carefully!**
  - Return `0` when the two strings are equal
  - `0` equates to `false`

Reading Text

- By default, `scanf( … )` uses whitespace as a terminator
- But whitespace is meaningful to strings
- To read character data, use the `gets` function

Reading Text

- Example:
  `char * gets( char * buffer )`
  reads data from standard input up to the newline character which is discarded and replaced by '\0'. Returns NULL if there is an error

```
char input[ 80 ];
char * data;

data = gets( input );
```

Writing Text

- Output of strings can be performed by `puts( … )`
  - Just an alternative to using `printf( … )`

```
char input[ 80 ];
char * data;

data = gets( input );
printf( "Here's Your Data: " );
puts( data );
```

String Length

- Use **strlen** to get the number of characters in a string
- Some programmers thing sizeof is the same function but it's not.
- We did not cover sizeof as it applies to dynamic arrays.
- In short, size returns the number of elements in an array.

```
char str1[20] = ”Hello";
printf("Length of string str1: %d", strlen(str1));
printf(”Num of elements in string str1: %d", sizeof(str1));
return 0;
```

  - strlen(str1) returned value 5.
  - sizeof(str1) would return value 20.

String Comparison

- strcmp is used if two strings are equal, or one is larger or smaller than then other.

- It compares the two strings. If both the strings are same (**equal**) then this function would **return 0** otherwise:
- If string1 < string2 OR string1 is a substring of string2 then it would result in a negative value. If string1 > string2 then it would return positive value.
- **If string1 == string2 then you would get 0(zero) when you use this function for compare strings.**

strcmp Function Examples

```
#include <stdio.h>
#include <string.h>
int main( )
{
   char s1[ 20 ] = ”Hello";
   char s2[ 20 ] = ”Hello World";
   if (strcmp(s1, s2) ==0)
   {
     printf("string 1 and string 2 are equal”);
   }else
   {
     printf("string 1 and 2 are different”);   ← this will be the output
   }
return 0;
}
```

strcmp Function Examples

```
#include <stdio.h>
#include <string.h>
int main( )
{
   char s1[ 20 ] = ”Hello";
   char s2[ 20 ] = ”Hello World";
   if (strcmp(s1, s2,4) ==0)   //compare only first four characters
   {
     printf("string 1 and string 2 are equal" ); ← this will be the output
   }else
   {
     printf( "string 1 and 2 are different" );
}
   return 0;
}
```

String Concatenation

- strcat function is used to “glue” two strings together and returns the combined one string.
- A terminator char (‘\0’) will be appended at the end of the concatenated string.
- Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strcat(s1,s2);
    printf("Output string after concatenation: %s", s1);
    return 0;
}
```

Copying Strings

- strcpy function copies the string str2 into string str1, including the end character (terminator char '\0').

Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[30] = "string 1";
    char s2[30] = "string 2 : copied into s1";
    strcpy(s1,s2); // function copies s2 into s1//
    printf("String s1 is: %s", s1); ←Output
    return 0;
}
```

strncpy

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[30] = "string 1";
     char s2[30] = "string 2 : copied into s1";
     strncpy(s1,s2,7);

    printf( "String s1 is: %s", s1 ); // s1 gets “string “ from s2 but remaining chars stay//
//it’s a good idea to clear s1 to get clean results

        getchar();
  return 0;
}
```

Slicing Strings

```
#include <stdio.h>
#include <string.h>
int main()
{
   char mystr[30] = "an example of function strchr";

   printf ( "%s", strchr( mystr, 'f' ) );
// output is from first f from start of string:    f function strchr

      getchar();
  return 0;
}
```

Reverse Slicing Strings

```
#include <stdio.h>
#include <string.h>
int main()
{
   char mystr[30] = "an example of function strchr";

   printf ( "%s", strrchr( mystr, 'f' ) );
// output is from end of string to first f:    function strchr

      getchar();
  return 0;
}
```

Summarizing Strings

- strings can be tricky!
- Exercise care when using them, since they are very unforgiving and prone to causing errors…
- You can use loops to process the character data looking for ending '\0'
- Read section 27 until end of 27.2.1