CMSC 104 - Lecture 23
Park , adapted by C Grasso

Strings: Part 1 of 2

Strings, Part 1 of 2

Topics

- Static Strings
- Character Arrays
- The Null Terminator
- Reading in Strings
- string.h
- ctype.h

Static Strings / String Literals

- String literals are characters surrounded by double quotes.

```
"This is a static string"
```

Static Strings / String Literals

- Strings are arrays of chars.

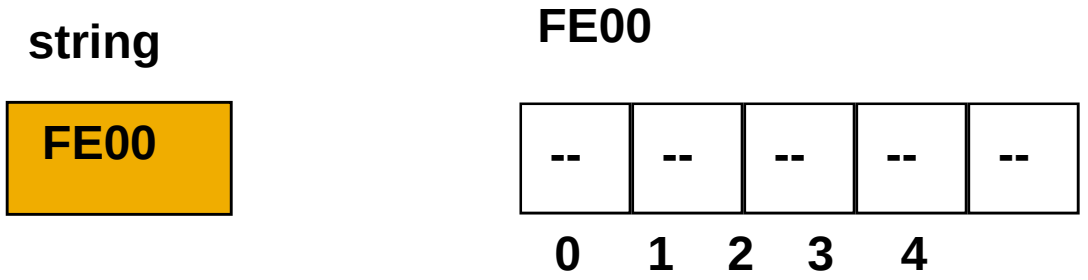
```
char s[] = "This is a static string";

char *s  = "This is a static string";
```

Declaring Strings

```
char   string[5] ;
```

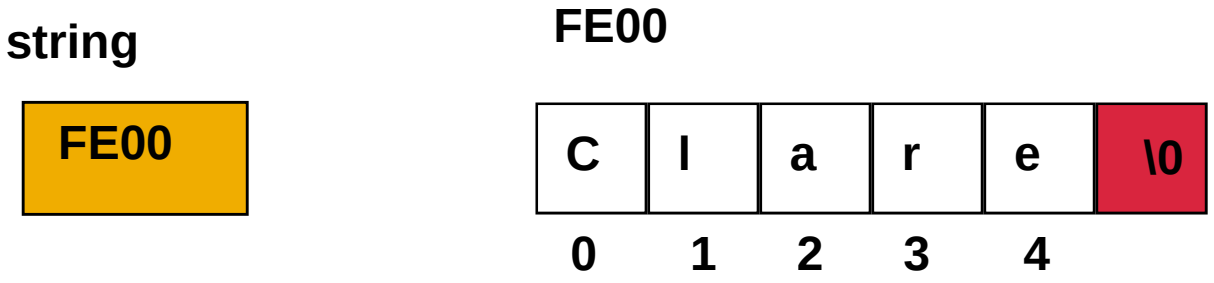
- This declaration sets aside a chunk of memory that is big enough to hold 5 characters.
- Besides the space needed for the array, there is also a variable allocated that has the name of the array. This variable holds the address of the beginning (address of the first element) of the array.



The Null Terminator

```
char   string[6] = { 'C', 'l', 'a', 'r', 'e', '\0' };
```

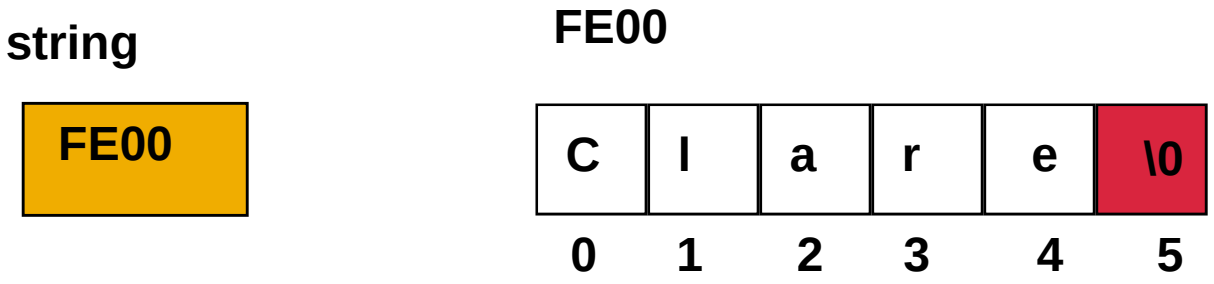
```
string[0] = 'C';
string[1] = 'l';
string[2] = 'a';
string[3] = 'r';
string[4] = 'e';
string[5] = '\0';
```



Initializing Strings

```
char   string[ 5+1 ] = "Clare" ;

char  *string        = "Clare";
```



Array Name Holds an Address

```
#include <stdio.h>

int main( )
{
   char  string[5+1] = "Clare" ;
   printf ("string[0]   = %c  \n",     string[0]  ) ;
   printf ("string[0]   = %d  \n",     string[0]  ) ;
   printf ("string      = %x  \n",     string     ) ;
   printf ("&string[0] = %x  \n",   &string[0]  ) ;
   return 0 ;
}
```

output

```
string[0]      = C
string[0]      = 67
string         = FE00
&string[0]     = FE00
```

How Indexing Works

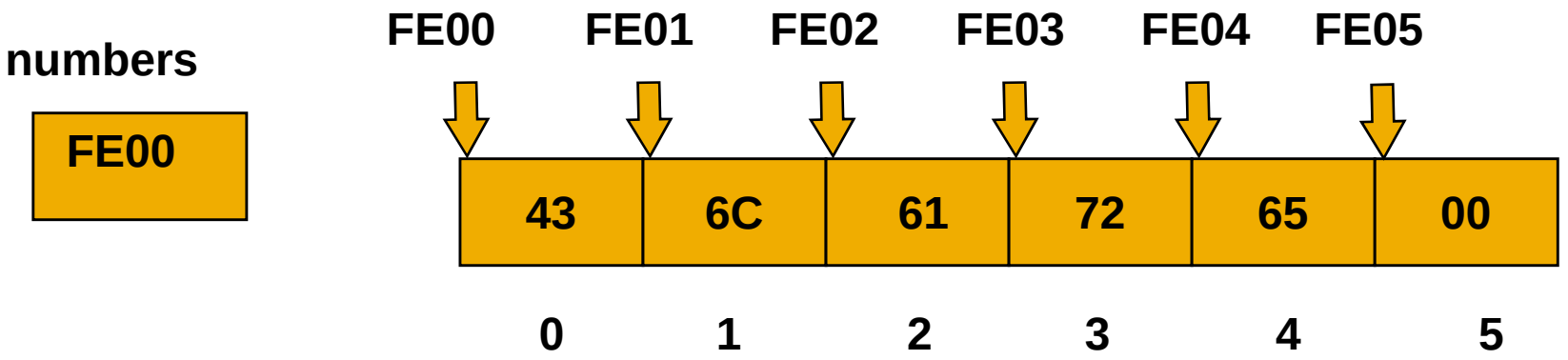
string[2] = 'a' ;

- The element assigned the value 'a' is stored in a memory location that is calculated using the following formula:

Location = (beginning address) + (index * sizeof(array data type))

Assuming a 1-byte char,

Location = FE00 + (2 * 1)



Indexing Arrays

- As long as we know
  - the beginning location of an array,
  - the data type being held in the array, and
  - the size of the array (so that we don't go out of range),

  then we can access or modify any of its elements using indexing.
- The array name alone (without [ ]) is just a variable that contains the starting address of the block of memory where the array is held.

Passing Arrays to Functions

- The array size

  ```
  #define SIZE 20
  ```

- The array definition

  ```
  int   ages[ SIZE] ;
  ```

- The function call:

  ```
  FillArray (ages, SIZE);
  ```

- The function definition :

  ```
  void FillArray (int ages[ ], int numElements) {… }
  ```

Passing String to Functions

- #define the string size

  ```
  #define SIZE 5
  ```

- The string definition

  ```
  char string[ SIZE +1] ;
  ```

- The function call:

  ```
  FillString (string);
  ```

- The function definition :

  ```
  void FillString (char *string ) { … }
  ```

Call (Pass) by Reference

- As demonstrated with arrays, we can pass addresses to functions. This is known as **calling (passing) by reference**.

- When the function is passed an address, it can make changes to the original (the corresponding actual parameter). There is no copy made**.**

Strings and scanf( )

- **NEVER, NEVER, NEVER** use scanf() with strings.

```
char ca[2];
int   x = 0, y = 0;

scanf("%s",ca);        // input "abcdefg"
```

Reading in Strings – fgets()

```
#define SIZE 100
...
char    str1 [SIZE+1];
...
fgets ( str1, SIZE, stdin );
```

Reading in Strings – fgets()

```
#define SIZE 100+1

char    str1 [SIZE];          // what's the difference?
char  *str2;

fgets ( str1, SIZE, stdin );
fgets ( str2, SIZE, stdin );  // will this work?
```

string.h

- **String handling functions**
- The strings.h header file has some useful functions for working with strings. Here are some of the functions you will use most often:

Length of a String

- **int strlen(char *str)**
  Returns the amount of characters in a string that appear before the first '\0'.

```
char *s = "abcde";
int      i = strlen( s );          // i = 5
```

Comparing Two Strings

- **int strcmp(char *str1, char *str2)**
  Compares the first and second strings.
  - If the str1 > str2, return int > 0
  - If the str2 < str2, return int < 0
  - If the str1 == str2, return 0

```
s1 = "abc";
s2 = "abc";
i  =  strcmp(s1,s2);                    // i = 0
```

Copying Strings

- **char *strcpy(char *dest, char *src)**
- You can't just assign `string1 = string2 ;`
- Use the strcpy( ) to copy the source string to the destination string.
  - Overwrites whatever was there before.

```
char *s1 = "abc";
char *s2 = "xyz";
strcpy(s1,s2);          // s1 = "xyz"
```

Concatenating Strings

- **char *strcat(char *dest, char *src)**
  Joins the destination and source strings and puts the joined string into the destination string.

```
char s1[3+3+1] = "abc";
char s2[3+1]   = "xyz";
strcat(s1,s2);                  // s1 = "abcxyz"
```

Finding a Char in a String

- **char *strchr(char *str, int ch)**
  Returns a pointer to the first occurrence of *ch* in *string*, or NULL if *ch* is not found.

```
char s1[3+3+1] = "abc";
char *s2 = strchr(s1,'b');
printf("%s",s2);                    // what will print?
```

Finding a String in a String

- **char *strstr(char *str1, char *str2)**
  Returns a pointer to the first occurrence of *str2* in *str1*, or NULL if *str2* is not found.

```
char *s1 = "I am here";
char *s2 =  strstr( s1,"am" );
printf("%s",s2);                    // what will print?
```

Controlling # of Chars in String Functions

```
char *s1 = "a";
char *s2 = "xyz";
strcpy(s1,s2);
```

What will happen?

Controlling # of Chars in String Functions

- **char *strncpy(char *str1, char * str2, int n)**
- **char *strncat(char *str1, char * str2, int n)**
- **int strncmp(char *str1, char * str2, int n)**

```
#define SIZE 20
char s1[SIZE+1] = "abc";

strncat(s1, "defghijk", SIZE);
```

ctype.h

- isalnum() Check if character is alphanumeric
- isalpha() - Check if character is alphabetic
- iscntrl() - Check if character is a control character
- isdigit() - Check if character is decimal digit
- isgraph() - Check if character has graphical representation
- islower() - Check if character is lowercase letter
- isprint() - Check if character is printable
- ispunct() - Check if character is a punctuation character
- isspace() - Check if character is a white-space
- isupper() - Check if character is uppercase letter
- isxdigit() - Check if character is hexadecimal digit
- toupper() – Convert a character to uppercase
- tolower() – Convert a character to lowercase

Exercise

- Create an interactive program that gets a person's firstname and displays it, gets the last name and displays it. Next, put the two names together to make the whole name and display it.