Strings

- data[0] is first character
- can be accessed char by char
- printf(\"%s\", data); 
- use printf to print strings
- char data[10] = \"HELLO\";
- A string is an array of characters

End of String Symbol

End of String Symbol

Only saves data up until
Each character has an integer representation.

Strings
Characters can be interpreted as integers

Strings
String literals are a sequence of characters between double quotes ("""). Examples:
- "hello"
- "a versus a"
- "a" is a single character value (stored in 1 byte) as the ASCII value of character 'a'
- "a" is an array with two characters, the first is 'a', the second is the \0 character value for a

String literal values are represented by sequences of characters between double quotes (""").
Referring to the literal refers to that space (as if it is an array)

- String literal (# of characters in the literal + 1)

During compilation, C creates space for each output the character 'e',

```
printf("%c", 'hello')[1];
```

Example:

character from the literal as a character

String literal is an array, can refer to a single

Referring to String Literals
Duplicate String Literals

different locations

but string1 does not equal string2 (they are stored at

char string2[6] = "hello"

char string1[6] = "hello"

Example:

(string1, they are not equal (in the == sense)

so even if the string literals contain the same
different location

Each string literal in a C program is stored at a
of str2 to str1 (more later) */

str1 = str2; /* not allowable, but we can copy the contents

Note, each variable is considered a constant in that the

char str4[6] = [o, e, t, l, l, o] = "Hello"

char str3 = "Hello"

char str2[6] = [h, e, l, l, o] = "Hello"

Examples (with initialization):

Allocate an array of a size large enough to hold the string

String Variables
str3 = str2
    /* connected to */
str3 = str1; /* str3 points to same space str1*/
    /* Not tied to space*/
char *str3; /* Not tied to space*/
    /* str2 unchangeable*/
char *str2 = "goodbye"; /* str2 unchangeable*/
    /* str1 unchangeable*/
char *str1 = "hello"; /* str1 unchangeable*/

Example:

- Cannot change space
- String variables
- Connected to

Changing String Variables
Have to stay within limits of array

String that does not end makes a

original string with something other than \0 makes a

Important to retain delimiter (replacing str1[5] in the

/* "\Vell" now str1 is */  
\0 \ = [4]str1  
/* "\Vell\Vell\" now str1 is */  
\"A\" \ = [0]str1  
char str1[6] = "\Vell\Vell\"  

Can change parts of a string variable

Changing String Variables
may occur

for delimiters, if too many characters for array, problems
Problem: no limit on number of characters read (need one

scanf("%s", &name);
char name[11];

Example:

Reads into array (no & before name, array is a pointer)
C stores null (\0) char after last non-white space char
reads characters until next white space encountered
ignores leading white space
use %s field specification in scanf to read string

String Input
characters

normally, but C always stops after reading 10
Strings shorter than the field specification are read
Width should be one less than size of array

Remember, you need one space for the \0
scanf("%10s", Name); Only reads 10 characters
char Name[11];

Can use the width value in the field specification

String Input
string (continued)
String Output

/* */ printf("\%10s\" Name\") ; /* outputs Rich */

Use - flag to left justify:

/* */ printf("\%10s\" Name\") ; /* outputs Rich */

Can use width value to print string in space:

/* */ printf("\%s\" Name\") ; /* outputs Rich */

char Name[10] = \"Rich\";

Characters in string printed until \0 encountered

Use \%s field specification in printf:
And stores in last name.

And stores the next 10 characters

Last name then reads

Reads up to

until the comma.

<table>
<thead>
<tr>
<th>Firstname, Lastname</th>
</tr>
</thead>
</table>

printf("Nice to meet you %s\n",
    scanf("%10s", &firstname, lastname, firstname));

printf("Enter your name (last, first): ");

char firstname[11];
char lastname[11];

#include <stdio.h>

Input/Output Example
newline included in string read

to read from keyboard: fgets(mystring,100,stdin)

returns string Read or NULL if problem/END-OF-FILE

reads at most size characters (plus one for \0)

fp must be an input connection

reads next line from file connected to fp, stores string in str

char *fgets(char *, int size, FILE *fp)

newline included in string read

not limited in how many chars read (may read too many for array)

returns string Read or NULL if problem/END-OF-FILE

stores it in the array of chars pointed to by str

reads the next line (up to the next newline) from keyboard and

 Command:

Reading a Whole Line
Printing a String

Commands:

```c
int puts(char *str, FILE *fp)

fgets
outputs newline if \n encountered
returns EOF if the fgets fails
fp must be an output connection
prints the string pointed to by str to the file connected to fp

int puts(char *str, FILE *fp)

fgets
outputs newline if \n encountered (for strings read with gets or
returns EOF if the puts fails
prints until delimiter reached (string better have a \0)
prints the string pointed to by str to the screen
```
```c
#include <stdio.h>

int main() {
    FILE *inputstream;
    char buffer[101];
    char frame[81];

    exit(-1);  
    printf("unable to open title " frame);  
}

if (inputstream = fopen("frame","r")) == NULL)  
    scanf("%80s", frame);  
    printf("show title: " frame);  
    printf("one line at a time
Reads file
```

Targets/Traps Example
```c
{
    Eclose(istream);
    fprintf(buffer, stdout);
    while (fgets(buffer, sizeof(buffer) - 1, istream) != NULL)
        printf("\""\"\name\"\");
```
# include <string.h>

Functions come from the utility library string.h

- strcmp (str1, str2) - compare str1 to str2
- strcpy (dst, src) - copy string at src to dst
- strlen (str) - calculate string length

Examples

Performing different string tasks

C provides a wide range of string functions

String Functions
Example:

char str[] = "hello";

String (str) would return 5

count the number of characters until an \0 encountered

returns the length (integer) of the string argument

Syntax: int strlen(char * str)
char *strcpy(char *dst, char *src, int n)

Only copies N characters

Similar to strncpy, but the copy stops after N characters

Return value is the destination string (dst)

If the two strings overlap (e.g., copying a string onto itself), the results are unpredictable

If the string to receive entire string (if not, other data may get written over)

dst should have enough space to receive entire string (not, other data)

copies the characters (including the \0) from the source string (src) to

dst

Syntax:

Copying a String
0 if the two strings do not differ
or if str2 starts the same as str1 (and str1 is longer)

If ASCII value of the character they differ at is larger for str1
    greater than 0

or if str1 starts the same as str2 (and str2 is longer)
If ASCII value of the character they differ at is smaller for str1
    less than 0

They differ at:

compares str1 to str2, returns a value based on the first character

int strcmp(char *str1, char *str2)

Syntax:

String Comparison
String Comparison

Expression for determining if two strings $s_1$, $s_2$ hold the same string value:

```c
strcmp(s1, s2)
```

Examples:

```c
strcmp(some diff, some diff)  -- Returns value > 0
strcmp(heello, hello there)  -- Returns value > 0
strcmp(heello, heello)       -- Returns value > 0
strcmp(heello, heello)       -- Returns value < 0
```

$\neq 0$
Strings Comparison

strcmp() "some diff", "someDIFF", 4 \returns 0
\returns value \gt 0

shorter than n
looks at first n characters of either string is
character works the same as strcmp except that it stops at the nth
\int strcmp(char* st, char* s2, int n)

sometimes we only want to compare first n chars:
Checks file for duplicate lines

strcpy/strncpy Example
{ 
    if (close(istream)) {
        strcpy(buffer, previous); /* Copy the current line (in buffer) to the previous */

        printf("Start the same:
% % % % % previous\nbuffer\n");
        else if (strcmp(buffer, previous) == 0) /* and previous line are the same */
            printf("Duplicate line: %s\nbuffer\n");
        else /* otherwise */
            printf("Different line as previous\n");
    } (fgets(buffer, sizeof(buffer), stdin) == NULL) /* Read a line of characters */
}
Version of strcmp that ignores case

```c
int strcasecmp(char *str1, char *str2, int n);
```

Similar to strcmp except that upper and lower case characters (e.g., 'a', 'A') are considered to be equal.

```c
int strcasecmp(char *str1, char *str2);
```

**Syntax:**

```
strcmp Comparsion (Ignoring Case)
```