# STRINGS

- A string is a sequence of characters treated as a group
- We have already used some string literals:
  - “filename”
  - “output string”
- Strings are important in many programming contexts:
  - names
  - other objects (numbers, identifiers, etc.)

# STRINGS IN C

- No explicit type, instead strings are maintained as arrays of characters
- Representing strings in C
  - stored in arrays of characters
  - array can be of any length
  - end of string is indicated by a *delimiter*, the zero character '\0'

# STRING LITERALS

- String literal values are represented by sequences of characters between double quotes (“)
- Examples
  - “” - empty string
  - “hello”
- “a” versus ‘a’
  - ‘a’ is a single character value (stored in 1 byte) as the ASCII value for a
  - “a” is an array with two characters, the first is a, the second is the character value \0

# REFERRING TO STRING LITERALS

- String literal is an array, can refer to a single character from the literal as a character
- Example:

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

  outputs the character 'e'
- During compilation, C creates space for each string literal (# of characters in the literal + 1)
  - referring to the literal refers to that space (as if it is an array)

# DUPLICATE STRING LITERALS

- Each string literal in a C program is stored at a different location
- So even if the string literals contain the same string, they are not equal (in the == sense)
- Example:
  - char string1[6] = “hello”;
  - char string2[6] = “hello”;
  - but string1 does not equal string2 (they are stored at different locations)

# STRING VARIABLES

- Allocate an array of a size large enough to hold the string (plus 1 extra value for the delimiter)
- Examples (with initialization):

```
char str1[6] = "Hello";
char str2[] = "Hello";
char *str3 = "Hello";
char str4[6] = {'H','e','l','l','o','\0'};
```

- Note, each variable is considered a constant in that the space it is connected to cannot be changed

```
str1 = str2; /* not allowable, but we can copy the contents
                of str2 to str1 (more later) */
```

# CHANGING STRING VARIABLES

- Cannot change space string variables connected to, but can use pointer variables that can be changed
- Example:

```
char *str1 = “hello”;   /* str1 unchangeable */
char *str2 = “goodbye”; /* str2 unchangeable */
char *str3; /* Not tied to space */
str3 = str1; /* str3 points to same space s1 connected to */
str3 = str2;
```

# CHANGING STRING VARIABLES (CONT)

- Can change parts of a string variable

```
char str1[6] = "hello";
str1[0] = 'y';
/* str1 is now "yello" */
str1[4] = '\0';
/* str1 is now "yell" */
```

- Important to retain delimiter (replacing str1[5] in the original string with something other than '\0' makes a string that does not end)
- Have to stay within limits of array

# STRING INPUT

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

```
char Name[11];
scanf("%s",Name);
```

- Problem: no limit on number of characters read (need one for delimiter), if too many characters for array, problems may occur

# INPUT/OUTPUT EXAMPLE

```
#include <stdio.h>


void main() {
  char LastName[11];
  char FirstName[11];


  printf("Enter your name (last , first): ");
  scanf("%10s%*[^,],%10s",LastName,FirstName);


  printf("Nice to meet you %s %s\n",
         FirstName,LastName);
}
```

# ARRAY OF STRINGS

- Sometimes useful to have an array of string values
- Each string could be of different length (producing a ragged string array)
- Example:

```
char *MonthNames[13]; /* an array of 13 strings */
MonthNames[1] = "January"; /* String with 8 chars */
MonthNames[2] = "February"; /* String with 9 chars */
MonthNames[3] = "March"; /* String with 6 chars */
etc.
```

# ARRAY OF STRINGS EXAMPLE

```
#include <stdio.h>


void main() {

  char *days[7];

  char TheDay[10];

  int day;


  days[0] = "Sunday";

  days[1] = "Monday";

  days[2] = "Tuesday";

  days[3] = "Wednesday";

  days[4] = "Thursday";

  days[5] = "Friday";

  days[6] = "Saturday";
```

# ARRAY OF STRINGS EXAMPLE

```
  printf("Please enter a day: ");
  scanf("%9s",TheDay);

  day = 0;
  while ((day < 7) && (!samestring(TheDay,days[day])))
    day++;

  if (day < 7)
    printf("%s is day %d.\n",TheDay,day);
  else
    printf("No day %s!\n",TheDay);
}
```

# ARRAY OF STRINGS EXAMPLE

```
int samestring(char *s1, char *s2) {
  int i;

  /* Not same if not of same length */
  if (strlen(s1) != strlen(s2))
    return 0;
  /* look at each character in turn */
  for (i = 0; i < strlen(s1); i++)
    /* if a character differs, string not same */
    if (s1[i] != s2[i]) return 0;
  return 1;
}
```

# STRING FUNCTIONS

- C provides a wide range of string functions for performing different string tasks
- Examples

  strlen(str) - calculate string length

  strcpy(dst,src) - copy string at src to dst

  strcmp(str1,str2) - compare str1 to str2
- Functions come from the utility library string.h
  - #include <string.h> to use

# STRING LENGTH

Syntax: int strlen(char *str)

returns the length (integer) of the string argument

counts the number of characters until an \0 encountered

does not count \0 char

Example:

char str1 = “hello”;

strlen(str1) would return 5

# COPYING A STRING

Syntax:

- char *strcpy(char *dst, char *src)
  - copies the characters (including the \0) from the source string (src) to the destination string (dst)
  - dst should have enough space to receive entire string (if not, other data may get written over)
  - if the two strings overlap (e.g., copying a string onto itself) the results are unpredictable
  - return value is the destination string (dst)
- char *strncpy(char *dst, char *src, int n)
  - similar to strcpy, but the copy stops after n characters
  - if n non-null (not \0) characters are copied, then no \0 is copied

# STRING COMPARISON

Syntax:

int strcmp(char *str1, char *str2)

compares str1 to str2, returns a value based on the first character they differ at:

*less than 0*

if ASCII value of the character they differ at is smaller for str1

or if str1 starts the same as str2 (and str2 is longer)

*greater than 0*

if ASCII value of the character they differ at is larger for str1

or if str2 starts the same as str1 (and str1 is longer)

0 if the two strings do not differ

# STRING COMPARISON (CONT)

strcmp examples:

strcmp(“hello”,”hello”) -- returns 0

strcmp(“yello”,”hello”) -- returns value > 0

strcmp(“Hello”,”hello”) -- returns value < 0

strcmp(“hello”,”hello there”) -- returns value < 0

strcmp(“some diff”,”some dift”) -- returns value < 0

expression for determining if two strings s1,s2 hold the same string value:

!strcmp(s1,s2)

# STRCPY/STRCMP EXAMPLE

```
#include <stdio.h>
#include <string.h>

void main() {
  char fname[81];
  char prevline[101] = "";
  char buffer[101];
  FILE *instream;

  printf("Check which file: ");
  scanf("%80s",fname);

  if ((instream = fopen(fname,"r")) == NULL) {
    printf("Unable to open file %s\n",fname);
    exit(-1);
  }
```

- Other functions: STRCAT