Strings

Arrays of characters

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Basics

Strings

A string is a sequence of characters treated as a group

We have already used some string literals:

- As in: printf(“Hello World”);

Strings are important in many programming contexts:

- names
- other objects (subject codes, telephone numbers, etc.)

What we should learn about strings

- Representation in C
- String Literals
- String Variables
- String Input/Output
  - printf, scanf, gets, fgets, puts, fputs
- String Functions
  - strlen, strcpy, strncpy, strcmp, strncmp, strcat, strncat, strchr, strrchr, strstr, strspn, strcspn, strtok
- Reading from/Printing to Strings
  - sprintf, sscanf

Strings in C

Strings are maintained as arrays of characters

Representing strings in C

- **stored in arrays of characters**
- **array can be of any length – *the string may be shorter, may not fill the array***
- **end of string is indicated by a *delimiter*, the null character '\0'**

"A String"

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 the letter, 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)

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 allowed, but we can copy the contents of str2 to str1 */
```

Changing String Variables

Cannot change string variables connected to string constants, but can change pointer variables that are not tied to space.

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

Example:

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

Problem: ***How to limit the input to within array bounds?***

String Input

Can use the width value in the field specification to limit the number of characters read:

```
char Name[11];
scanf("%10s", Name);
```

Remember, you need one space for the \0

- width should be one less than size of array

In this example, strings shorter than the field specification are read normally, but C always stops after reading 10 characters

More on String Input

Edit set input %[*ListofChars*]

- ListofChars specifies set of characters (called scan set)
- Characters read as long as character falls in scan set
- Stops when first non scan set character encountered
- Note, does not ignored leading white space
- Any character may be specified except]
- Putting ^ at the start to negate the set (any character BUT list is allowed)

Examples:

```
scanf("%[–+0123456789]", Number);
scanf("%[^\n]",Line);  /* read until newline char */
```

String Output

- Use %s field specification in printf:

  characters in string printed until \0 encountered

```
char Name[10] = “Joy”;
printf(“|%s|”,Name);                    /* outputs |Joy| */
```

- Can use width value to print string in space:

```
printf(“|%10s|”,Name);                  /* outputs |      Joy| */
```

- Use - flag to left justify:

```
printf(“|%-10s|”,Name);                 /* outputs |Joy      | */
```

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);
}
```

String Functions

Reading a Whole Line

char *gets(char *str)

- reads up to the next newline from keyboard and stores it in the array of chars pointed to by str
- returns str if string read or NULL if problem/end-of-file
- not limited in how many chars read (may read too many for array)
- newline included in string read

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

- reads next line from file connected to fp, stores string in str
- fp must be an input connection
- reads at most size characters (plus one for \0)
- returns str if string read or NULL if problem/end-of-file
- to read from keyboard: fgets(mystring,100,stdin)
- newline included in string read

Printing a String

int puts(char *str)

- prints the string pointed to by str to the screen
- prints until delimiter reached (string better have a \0)
- returns EOF if the puts fails
- outputs newline if \n encountered

int fputs(char *str, FILE *fp)

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

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>
```

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

char *strcpy(char *dst, char *src)

- copies the characters (including the \0) from the string src to the destination string dst
- dst should have enough space to receive entire string
- 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

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

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)
```

String Comparison

Sometimes we only want to compare first n chars:

```
int strncmp(char *s1, char *s2, int n)
```

Works the same as strcmp except that it stops at the nth character

looks at less than n characters if either string is shorter than n

```
strcmp("some diff","some DIFF")        -- returns value > 0
strncmp("some diff","some DIFF",4)     -- returns 0
```

String Comparison (ignoring case)

int strcasecmp(char *str1, char *str2)

- similar to strcmp except that upper and lower case characters (e.g., ‘a’ and ‘A’) are considered to be equal

int strncasecmp(char *str1, char *str2, int n)

- version of strncmp that ignores case

String Concatenation

char *strcat(char *dstS, char *addS)

- appends the string at addS to the string dstS
- returns the string dstS
- can cause problems if the resulting string is too long to fit in dstS

char *strncat(char *dstS, char *addS, int n)

- appends the first n characters of addS to dstS
- if less than n characters in addS only the characters in addS appended
- always appends a \0 character

Searching for a Character/String

char *strchr(char *str, int ch)

- returns a pointer (a char *) to the first occurrence of ch in str
- returns NULL if ch does not occur in str
- can subtract original pointer from result pointer to determine which character in array

char *strstr(char *str, char *searchstr)

- similar to strchr, but looks for the first occurrence of the string searchstr in str

char *strrchr(char *str, int ch)

- similar to strchr except that the search starts from the end of string str and works backward

Printing to a String

The sprintf function allows us to print to a string argument using printf formatting rules

First argument of sprintf is string to print to, remaining arguments are as in printf

Example:

```
char buffer[100];
sprintf(buffer, ”%s, %s”, LastName, FirstName);
if (strlen(buffer) > 15)
        printf(“Long name %s %s\n”, FirstName, LastName);
```

Reading from a String

The sscanf function allows us to read from a string argument using scanf rules

First argument of sscanf is string to read from, remaining arguments are as in scanf

Example:

```
char buffer[100] = “A10 50.0”;
sscanf(buffer,”%c%d%f”,&ch,&inum,&fnum);
                    /* puts ‘A’ in ch, 10 in inum and 50.0 in fnum */
```

Example: PigLatin

Write a C function that takes a string as an argument, and prints the string in PigLatin.

In PigLatin, you move the first letter of the word to the end, and then add "ay".

E.g., "this was fun" becomes "histay asway unfay".

```
void printPigLatinWord( char word[ ] ) {
        printf("%s", &word[1] );    /* print word except first character */
        printf("%c", word[0]);      /* print first character of word */
        printf("ay\n");             /* print "ay" */
}
```

Example: Duplicate Removal

Write a C function that takes a string as an argument and modifies the string so as to remove all consecutive duplicate characters, e.g., mississippi -> misisipi.

```
void remove_duplicates( char word[ ] ) {
        int k, j;
        char prev = '\0';
        for (k = j = 0; word[ k ] != '\0'; k++)  {
                if (prev != word[ k ]) word[ j++ ] = word[ k ];
                prev = word[ k ];
        }
        word[ j ] = '\0';
}
```