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.)

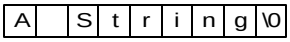
Outline

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

- 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'

"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 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

String Input (cont)

- 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
- Strings shorter than the field specification are read normally, but C always stops after reading 10 characters

String Input (cont)

- 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] = “Rich”;
  printf(“|%s|”,Name); /* outputs |Rich| */
- Can use width value to print string in space:
  printf(“|%10s|”,Name); /* outputs | Rich| */
- Use - flag to left justify:
  printf(“|%-10s|”,Name); /* outputs |Rich | */

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);
}
```

Reading a Whole Line

Commands:

char *gets(char *str)
- reads the next line (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

Commands:

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 (for strings read with gets or fgets)

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

fgets/fputs Example

```
#include <stdio.h>

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

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

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

fgets/fputs Example (cont)

```
  printf("\n%s:\n",fname);

  while (fgets(buffer,sizeof(buffer)-1,instream)
         != NULL)
    fputs(buffer,stdout);

  fclose(instream);
}
```

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 diff") -- returns value < 0

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

!strcmp(s1,s2)

String Comparison (cont)

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

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);
  }
```

strcpy/strcmp Example

```
  /* read a line of characters */
  while (fgets(buffer,sizeof(buffer)-1,instream) != NULL) {
    /* if current line same as previous */
    if (!strcmp(buffer,prevline))
      printf("Duplicate line: %s",buffer);
    /* otherwise if the first 10 characters of the current
       and previous line are the same */
    else if (!strncmp(buffer,prevline,10))
      printf("Start the same:\n  %s  %s",prevline,buffer);
    /* Copy the current line (in buffer) to the previous
       line (in prevline) */
    strcpy(prevline,buffer);
  }
  fclose(instream);
}
```

String Comparison (ignoring case)

Syntax:

- 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

Syntax:

- char *strcat(char *dstS, char *addS)
  - appends the string at addS to the string dstS (after dstS's delimiter)
  - 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

strcat Example

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

void main() {
  char fname[81];
  char buffer[101];
  char curraddress[201] = "";
  FILE *instream;
  int first = 1;

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

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

strcat Example

```
/* Read a line */
while (fgets(buffer,sizeof(buffer)-1,instream) != NULL) {
    if (buffer[0] == '*') { /* End of address */
      printf("%s\n",curraddress); /* Print address */
      strcpy(curraddress,""); /* Reset address to "" */
      first = 1;
    }
    else {
      /* Add comma (if not first entry in address) */
      if (first) first = 0; else strcat(curraddress,", ");
      /* Add line (minus newline) to address */
      strncat(curraddress,buffer,strlen(buffer)-1);
    }
  }

  fclose(instream);
}
```

Searching for a Character/String

Syntax:

- 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

strchr Example

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

void main() {
  char fname[81];
  char buffer[101];
  char noncom[101];
  FILE *instream;
  char *loc;

  printf("File: ");
  scanf("%80s",fname);

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

strchr Example

```
/* read line */
while (fgets(buffer,sizeof(buffer)-1,instream) != NULL) {
    /* Look for the character % in the line */
    if ((loc = strchr(buffer,'%')) != NULL) {
      /* Copy the characters before the % to noncom */
      strncpy(noncom,buffer,(loc - buffer));
      /* Add a delimiter to noncom */
      noncom[ loc - buffer] = '\0';
      printf("%s\n",noncom);
    }
    else
      printf("%s",buffer);
  }
  flose(instream);
}
```

strstr Example

```
/* read line */
while (fgets(buffer,sizeof(buffer)-1,instream) != NULL) {
    /* Look for the character % in the line */
    if ((loc = strstr(buffer,"\\")) != NULL) {
      /* Copy the characters before the % to noncom */
      strncpy(noncom,buffer,(loc - buffer));
      /* Add a delimiter to noncom */
      noncom[ loc - buffer] = '\0';
      printf("%s\n",noncom);
    }
    else
      printf("%s",buffer);
  }
  flose(instream);
}
```

String Spans (Searching)

Syntax:

int strspn(char * str, char * cset)

- specify a set of characters as a string cset
- strspn searches for the first character in str that is not part of cset
- returns the number of characters in set found before first non-set character found

int strcspn(char * str, char *cset)

- similar to strspn except that it stops when a character that is part of the set is found

Examples:

strspn("a vowel","bvcwl") returns 2

strcspn("a vowel","@,*e") returns 5

Parsing Strings

The strtok routine can be used to break a string of characters into a set of tokens

- we specify the characters (e.g., whitespace) that separate tokens
- strtok returns a pointer to the first string corresponding to a token, the next call returns the next pointer, etc.
- example:
  - call strtok repeatedly on "A short string\n" with whitespace (space, \t, \n) as delimiters, should return
    - first call: pointer to string consisting of "A"
    - second call: pointer to string consisting of "short"
    - third call: pointer to string consisting of "string"
    - fourth call: NULL pointer

Parsing Strings

Syntax:

char * strtok(char * str, char *delimiters)

- delimiters is a string consisting of the delimiter characters (are ignored, end tokens)
- if we call strtok with a string argument as the first argument it finds the first string separated by the delimiters in str
- if we call strtok with NULL as the first argument it finds the next string separated by the delimiters in the string it is currently processing
- strtok makes alterations to the string str (best to make a copy of it first if you want a clean copy)

strtok Example

```
#include <stdio.h>
#include <string.h>
void main() {
  FILE *instream;
  char fname[81];
  char buffer[101];   /* current line */
  char *token;        /* pointer to current token */
  char *tokens[100];  /* array of unique tokens in file */
  int tokenline[100]; /* line number where token found */
  int ntokens = 0;    /* number of tokens found so far */
  int linenum = 0;    /* current line number */
  int tnum;
  printf("File: ");
  scanf("%80s",fname);

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

strtok Example (cont)

```
while (fgets(buffer,sizeof(buffer)-1,instream) != NULL) {
  linenum++;
  /* Get first token from string buffer */
  token = strtok(buffer," \t\n");
  while (token != NULL) {
    tnum = 0;
    /* Search for current token in tokens */
    while ((tnum < ntokens) && (strcmp(token,tokens[tnum])))
      tnum++;
    /* If the token isn't found, add it to the tokens */
    if (tnum == ntokens) {
      tokens[tnum] =
        (char *) calloc(strlen(token) + 1,sizeof(char));
      strcpy(tokens[tnum],token);
      tokenline[tnum] = linenum;
      ntokens++;
    }
    /* Get next token from string buffer */
    token = strtok(NULL," \t\n");
  }
}
```

strtok Example (cont)

```
  /* print out the set of tokens appearing in the file */
  for (tnum = 0; tnum < ntokens; tnum++)
    printf("%s first appears on line %d\n",
           tokens[tnum],tokenline[tnum]);

  fclose(instream);
}
```

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);
```

sprintf Example

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

void main() {
  FILE *instream;
  FILE *outstream;
  char basefname[81];
  char readfname[101];
  char savefname[81];
  char buffer[101];
  int fnum;

  printf("File Prefix: ");
  scanf("%80s",basefname);
  printf("Save to File: ");
  scanf("%80s",savefname);

  if ((outstream =
       fopen(savefname,"w")) ==
       NULL) {
    printf("Unable to open %s\n",
           savefname);
    exit(-1);
  }
```

sprintf Example

```
for (fnum = 0; fnum < 5; fnum++) {
   /* file name with basefname as prefix, fnum as suffix */
   sprintf(readfname,"%s.%d",basefname,fnum);
   if ((instream = fopen(readfname,"r")) == NULL) {
     printf("Unable to open input file %s\n",readfname);
     exit(-1);
   }
   while (fgets(buffer,sizeof(buffer)-1,instream) != NULL)
     fputs(buffer,outstream);
   fclose(instream);
  }
  fclose(outstream);
}
```

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 */
```