Strings

- **Problem:**

A direct-mail advertising agency has decided to personalize its sweepstakes offers. It has prepared a basic letter with a particular customer's name. address, spouse's name, and other personal information. The company would like a computer program to make the appropriate changes to address each customer individually. As a first test, the company would like the program to make one set of changes: to replace all occurrences of

1. Smith	by	Johnson
2. Mr.	by	Ms.
3. 421 Main St.	by	18 Windy Lane
4. wife	by	husband
5. Susan	by	Robert
6. her	by	his

Here is the basic letter:

Congratulations, Mr. Smith! The Smith family of 421 Main St. may have already won a new One-million-dollar house!! Your neighbors at 421 Main St. will be so surprised when you, Mr. Smith, move into your new house with your wife, Susan! And her eyes will light up with joy at her fabulous new home! Enter the sweepstakes now, and you and the Smith family may win it all!.

Write a C program that reads in the text line by line, displays each line as it is read in, makes the designated changes, and displays the revised text.

- A **string** is a sequence of elements of the **char** data type.

- a string must end (or terminate) in the **null character** ('\0').

- **A string literal** is a sequence of characters enclosed by **double** quotation marks. (Note: a string literal automatically inserts the null character.) A string literal is a **constant**.

- **Declaring String Variables:**
  A string is declared like an array of characters. You must leave room for the null character.

  ```
  char name[21];          // this can hold a string up to length 20
  ```

- **Initializing a String:**
  ```
  char first[10] =  {'t','a','b','l','e','\0'};
  char second[10] =  "table";
  ```

- The **length** of a string is the number of characters stored in the string up to, but not including, the null character.

- The name of a string can be viewed as a **constant pointer** to a string.

- **Variable Pointer to a String:**
  ```
  char * sptr;
  ```

- Legal Characters:
  Each occurrence of double quotation marks or a backslash (or any other special character) must be preceded by the escape character (\) to tell the compiler that this is a character and not a control character.

  ```
  char quotes[20] =  "the \"king\" of rock";
  char filename[20] =  "c:\\hwork\\prob9.c";
  ```

- **Initializing a String within the Code:**

```
char city[15];

city[0] = 'L';
city[1] = '.';
city[2] = 'A';
city[3] = '.';
city[4] = '\0';

city = "L.A.";              // illegal
if (city = = "L.A.")    // illegal
   ...
```

- **A String as an Array of char:**

```
char c,d;
char str[5] = "wing";
char item[10] = "compater";

c = str[3];                 // c = = 'g'
d = str[0];                 // d = = 'w'

item[4] = 'u';              // item = = "computer"
```

- **Printing a String (the %s conversion specification):**

```
char first[11] = "Andy";

printf("The name is %s\n",first);
```

- **Reading Strings Using scanf:**

```
char name[16];

printf("Enter a name of up to 15 characters: ");
scanf("%s",name);
```

Note: DO NOT TYPE IN THE QUOTATION MARKS!

- **Problems with Using scanf:**
  - scanf stops reading at the first whitespace character (e.g., space, tab, newline)!
  - This is true even if the string entered is larger than the maximum string length. (It will corrupt subsequent memory locations.)

- **Reading Multiple Strings Using scanf:**
  - It is recommended that each variable value be entered using a separate call to scanf.

```
char  first[16];
char  last[16];
int   number;

printf("Enter a first name of up to 15 characters: ");
scanf("%s",first);
printf("Enter a last name of up to 15 characters: ");
scanf("%s",last);

scanf("%s %s",first,last);          // considered bad style
scanf("%d %s",&number,last);        // also considered bad style
```

The string.h Standard Library Functions

- **Using string.h:**
  #include < string.h>

- **Assigning a Value to a Sting Variable - strcpy():**
  strcpy(*dest_str,source_str*);

  - copies the source_str to the dest_str.

- **Examples:**
  char first[16];
  char last[16];

  strcpy(first,"Wayne Smith");
  strcpy(last,first);

  strcpy(last,&first[6]);
  strcpy(last,first+ 6);

  Note: strcpy continues to copy until the first null character.

- **Determining the length of a String - strlen():**
  length = strlen(*str*);

  - returns the current length of str as an integer.

- **The sizeof Operator:**
  size = sizeof *item*; // returns size in bytes
  or
  size = sizeof(*item*); // returns size in bytes
  or
  size = sizeof(*typename*); // returns number of bytes
  // allocated to that type

- **Examples:**

```
int length,size;
char dest[25];
char source[30];

scanf("%s",source);
length = strlen(source)
size = sizeof dest;
if (length < size)
   strcpy(dest,source);
else
   printf("won't fit\n");
```

- **Comparing Strings - strcmp():**
  result = strcmp(*first_str,second_str*);

  - strings are compared according to their ASCII values.
  - strings are compared character by character until either a null character or a difference in characters is found.
  - returns an integer result of the comparison:
    result > 0 if first_str > second_str
    result = 0 if first_str = = second_str
    result < 0 if first_str < second_str
  - A < Z < a < z
  - " " < "Car" < "Cat" < "car" < "cat" < "cats" < "cub"

- **Examples:**

```
int result;
char first[15] = "cat";
char second[15] = "car";

result = strcmp("cat","car");       // result > 0
result = strcmp("big","little"); // result < 0
result = strcmp("ABC","abc");     // result < 0
result = strcmp(" ab","ab");       // result < 0
result = strcmp("pre","prefix");   // result < 0
result = strcmp("potato","pot");  // result > 0
result = strcmp("cat","cat");       // result = = 0

result = strcmp(first,second);      // result > 0
result = strcmp(first,"catalog");   // result < 0

scanf("%s",first);
scanf("%s",second);
if (strcmp(first,second) = = 0)
   printf("they are equal\n");
```

- **Concatenating Two Strings - strcat():**
  strcat(*first_str,second_str*);

  - This functions concatenates the second string onto the end of the first string.

- **Example:**

```
char bigstr[1024];
char dest[30] =  "computer";
char second[15] =  " programming";

strcat(dest,second);        // dest =  "computer programming"
strcpy(bigstr,dest);        // bigstr =  "computer programming"
strcat(bigstr," is fun and very demanding");

// bigstr =  "computer programming is fun and very demanding"
```

- **Example:**

```
char dest[30] =  "computer";
char second[15] =  " programming";

if (strlen(dest) +  strlen(second) <  sizeof(dest))
    strcat(dest,second);
else
    printf("error: can't concatenate - dest too small\n");
```

Substring Functions

- **Comparing Substrings - strncmp():**
  result = strncmp(*address_1,address_2,numchars*);

  - Compares up to **numchars** from two strings, starting at the addresses specified (**address_1 and address_2**).

  - Returns an integer representing the relationship between the strings. (As per strcmp().)

  - Strings are compared character by character until numchars are compared or either a null character or a difference in characters is found.

- **Example:**

```
char first[30] = "strong";
char second[10] = "stopper";
int numchars;

if (strncmp(first,second,4) == 0)
    printf("first four characters are alike\n");
else if (strncmp(first,second,4) < 0)
    printf("first four characters of first string are less\n");
else
    printf("first four characters of first string are more\n");

if (strncmp(&first[2],"ron",3) == 0)
    printf("ron is found in position 2\n");
else
    printf("ron is not found in position 2\n");
```

- **Copying a Substring - strncpy():**
  strncpy(*dest_str,source_str,numchars*);

  - copies numchars from the source_str to the dest_str.

- **Example:**

```
char dest[10];
char source[20] =  "computers are fun";
char result[18] =  "I have a cat";
char insert[10] =  "big ";
int len;

strncpy(dest,source+ 3,3);
dest[3] =  '\0';                        // dest =  "put"
printf("%s\n",dest);

len =  strlen(insert);
strcpy(result+ 9+ len,result+ 9);   // make room for insertion
strncpy(result+ 9,insert,len);    // result =  "I have a big cat"
```

String Input/Output Functions

- **Printing a String - puts()**

  puts(*str*);

  - sends str to **stdout**, the standard output stream.
  - puts() automatically prints a newline character after str.

- **Example:**

```
char str[28] =  "This is a string to display";

puts(str);
```

- **Reading a value into a Character Array - gets():**

```
result =  gets(str);
   or
gets(str);
```

  - **str** is pointer to a character array.
  - fills the array **str** from **stdin**, the standard input stream.
  - gets() continues reading (including whitespace characters) until a newline character is encountered (ENTER).
  - gets() returns a value of type **char *** (or **NULL** if it fails to read).

- **Example:**

```
char str[128];
char *  instring;

gets(str);
puts(str);

instring =  gets(str);
puts(instring);
```

The NULL Pointer

- In C, there is a special value for a pointer to indicate that it is currently not pointing at anything, This value is **NULL**.

- The gets() function returns NULL when it fails to read in a value.

- A user can interactively signal NULL by entering CTRL-Z ENTER in Windows (or CTRL-D in UNIX).

- **Example:**

```
char str[128];
char * instring;

instring = gets(str);
while(instring != NULL) {
    puts(str);                              // or puts(instring);
    instring = gets(str);
}

while ((instring = gets(str)) != NULL)      // equivalent code
    puts(str);                              // or puts(instring);


while (gets(str) != NULL)                   // equivalent code
    puts(str);
```

- **Problems with gets():**
  - gets() **does not check** to see if the destination has room for the string being read in.

Writing String Functions

- **The Function length():**
  - Write a function that returns the length of a string.

```
/* function to find and return length of a string */
int length(char *str)
{
    int i = 0;

    while (str[i] != '\0')
        i++;
    return(i);
}
```

- **The Function countchar():**
  - Write a function that counts how many times a particular character appears within a string.

```
/* function to find number of occurrences of a particular
 * character within a string
 */
int countchar(char str[], char let)
{
    int i= 0, count= 0;

    while (str[i] != '\0') {
        if (str[i] == let)
            count++;
        i++;
    }
    return(count);
}
```

- **The Function findchar():**
  - Write a function to determine the position of a particular character within a string.

```
/* function to return the position of a particular character
 * within a string. Returns -1 if not found.
 */
int findchar(char * str, char let)
{
    int i= 0;

    while (str[i] != '\0') {
        if (str[i] == let)
            return(i);
        i++;
    }
    return(-1);
}
```

- **Alternate Code:**

```
int findchar(char * str, char let)
{
    int i= 0, found= 0;

    while (*(str+ i) != '\0' && !found)
        if (*(str+ i) == let)
            found = 1;
        else
            i++;
    if (!found)
        i = -1;
    return(i);
}
```

Functions that Return a Value of char *

- **The Function classify():**

```
/*  classifies monthname into one of four seasons */
char * classify(char * monthname)
{
    if (strcmp(monthname, "December") = =  0 ||
        strcmp(monthname, "January") = =  0 ||
        strcmp(monthname, "February") = =  0)
        return("winter");
    else if (strcmp(monthname, "March") = =  0 ||
        strcmp(monthname, "April") = =  0 ||
        strcmp(monthname, "May") = =  0)
        return("spring");
    else if (strcmp(monthname, "June") = =  0 ||
        strcmp(monthname, "July") = =  0 ||
        strcmp(monthname, "August") = =  0)
        return("summer");
    else if (strcmp(monthname, "September") = =  0 ||
        strcmp(monthname, "October") = =  0 ||
        strcmp(monthname, "November") = =  0)
        return("fall");
    else
        return("error");
}
```

- **Calling Program:**

```
char * season;
char month[10];

gets(month);
season =  classify(month);
if (strcmp(season,"error") !=  0)
    printf("%s is in the %s\n",month,season);
else
    printf("%s is not a valid month\n",month);
```

- **The Function split():**
  - Write a function to split a string into two parts at its first blank.

```
/* function to split a string into two parts at the first
 * occurrence of a blank character. Both parts are to be
 * returned to the calling program.
 */
int split(char *stringtosplit, char *first, char *second)
{
    int pos;

    pos = findchar(stringtosplit,' ');
    if (pos != -1) {
        strncpy(first,stringtosplit,pos);
        *(first+pos) = '\0';                    //first[pos] = '\0';
        strcpy(second,stringtosplit+pos+1);
    }
    return pos;
}
```

- **Calling Program:**

```
char *stringtosplit;
int result;
char buffer[50], first[50], second[50];

stringtosplit = gets(buffer);
result = split(stringtosplit,first,second);
if (result != -1)
    printf("%s %s\n",second,first);
else
    printf("no blank in string\n");
```

Returning to Our Problem

- **Pseudocode:**

```
while there is a line of the letter to read
    read in a line of the original letter
    print the original line
    replace the old strings in the line by the new ones
    print the new line
```

- **Main Program:**

```
/*  program to read a letter and replace all occurrences of old
 *  strings with new strings.
 */
#include < stdio.h>
#include < string.h>
#define LINESIZE    120
#define REPSIZE     15

/*  Function Prototypes Go Here*/

void main()
{
    char text[LINESIZE];

    while (gets(text) !=  NULL) {
        puts(text);
        replace(text);              // MUST STILL BE WRITTEN
        puts(text);
    }
}
```

- **Pseudocode for replace():**

```
while there are data values
    read in a set of replacements (oldstr & newstr)
    while oldstr occurs in text
        search for next occurrence of oldstr in text
        replace oldstr by newstr
```

- **Revised Pseudocode for replace():**

```
while there are data values
    read in a set of replacements (oldstr & newstr)
    while oldstr occurs in text
        call pos() to search for next occurrence of oldstr in text
        call splitup() to break up text and remove oldstr
        call reassemble() to reconstruct text and insert newstr
```

- Function replace():

```
/* ... */
void replace(char *text)
{
    int p,lenold;
    char part1[LINESIZE], part2[LINESIZE];
    char *oldstr, *newstr;
    char oldin[REPSIZE], newin[REPSIZE];

    while ((oldstr = gets(oldin)) != NULL) {
        newstr = gets(newin);
        lenold = strlen(oldstr);
        while ((p = pos(text,oldstr)) != -1) {
            splitup(text,lenold,part1,part2,p);
            reassemble(text,newstr,part1,part2);
        }
    }
    return;
}
```

- **Finding one String Within Another String - pos():**

```
/* Function pos:
 * Input:
 *     oldstr - string to search for
 *     text - string in which to search
 * Process:
 *     finds position of first occurrence of oldstr in text
 * Output:
 *     if found, returns position; if not found, returns -1
 */
int pos(char *text, char *oldstr)
{
    int lenold,result,i= 0;

    lenold =  strlen(oldstr);
    while (text[i] !=  '\0') {
        result =  strncmp(&text[i],oldstr,lenold);
        if (result = =  0)
            return(i);
        i+ + ;
    }
    return(-1);
}
```

- **Splitting the String - splitup():**

```
/*  Function splitup:
 *  Input:
 *      text - string to split
 *      lenold - length of old string
 *      p - position of old string
 *      part1, part2 - strings to fill
 *  Process:
 *      splits text at positions p and p+ lenold
 *      part1 gets text prior to oldstr
 *      part2 gets text after oldstr
 *  Output:
 *      part1 and part2 get new values
 */
void splitup(char *text, int lenold, char *part1, char *part2,
    int p)
{
    strncpy(part1,text,p);
    part1[p] =  '\0';
    strcpy(part2,&text[p+ lenold]);
    return;
}
```

- **Putting the String back Together - reassemble():**

```
/*  Function reassemble:
 *  Input:
 *      newstr - the replacement string
 *      part1, part2 - first and last parts of the original string
 *  Process:
 *      reassembles text using concatenation of part1, newstr,
 *      & part2
 *  Output:
 *      text has new value of part1+ newstr+ part2
 */
void reassemble(char *text, char *newstr, char *part1,
    char *part2)
{
    strcpy(text,part1);
    strcat(text,newstr);
    strcat(text,part2);
    return;
}
```

- **Revised Main Program:**

```
/*  program to read a letter and replace all occurrences of old
 *  strings with new strings.
 */
#include < stdio.h>
#include < string.h>
#define LINESIZE    120
#define REPSIZE     15

/*  Function Prototypes */
void replace(char * );
int pos(char * ,char * );
void splitup(char * ,int,char * ,char * ,int);
void reassemble(char * ,char* ,char * ,char * );

void main()
{
    char text[LINESIZE];

    while (gets(text) !=  NULL) {
        puts(text);
        replace(text);
        puts(text);
    }
}
```

Additional Character Related Functions

- **The Function getchar():**

  c = getchar();

  - This function returns a single **integer** value indicating the character just read from stdin. On error, EOF is returned.

  - The variable *c* should be of type integer (not of type char).

  - Note, a value of type char can be interpreted as either a character or an integer. An integer in the range of type char (0...255) can be interpreted as either a character or an integer.

- **The Function putchar():**

  putchar(*ch*);

  - The putchar() function sends a value of type **char** to stdout.

- **Example:**

```
int c,count= 0;

c = getchar();
while (c != EOF) {
    getchar();                  // throw out ENTER from buffer
    putchar(c);
    count++;
    c = getchar();
}
printf("there are %d characters in the input\n",count);
```

Testing the type of a char Value

- **Selected Functions from ctype.h:**

Function	Checks
isalpha(ch)	is the parameter alphabetic (A..Z or a..z)
isdigit(ch)	is the parameter a digit (0..9)
isalnum(ch)	is the parameter alphabetic or a digit
isspace(ch)	is the parameter a space (' ')
ispunct(ch)	is the parameter a punctuation mark
islower(ch)	is the parameter a lowercase alphabetic (a..z)
isupper(ch)	is the parameter an uppercase alphabetic (A..Z)

  - All these functions return 1 if true and 0 if false.

- **Example:**

```
int ch;
int alpha= 0,digit= 0,space= 0,punct= 0;

while ((ch =  getchar()) !=  EOF) {
   getchar();                    // throw out ENTER from buffer
   if (isalpha(ch))
      alpha+ + ;
   else if (isdigit(ch))
      digit+ + ;
   else if (isspace(ch))
      space+ + ;
   else if (ispunct(ch))
      punct+ + ;
}
```

- **The Functions toupper() & tolower():**

```
chout = toupper(ch);

chout = tolower(ch);
```

- These functions force the case of the input character.

- **Example:**

```
int ch;

do {
    ...
    printf("Do you want to continue? (y/n): ");
    ch = getchar();
    getchar();                  // throw out ENTER from buffer
} while (toupper(ch) == 'Y');
```

- **Example:**

```
char str[15];
int i= 0;

strcpy(str,"alPHABetic");
while (str[i] != '\0') {
    str[i] = tolower(str[i]);
    i++;
}
```

Arrays of Strings

- **Declaring an Array of Strings:**
  char *identifier*[*ARRAYSIZE*][*STRINGSIZE*];

- **Example:**

```
char months[12][10] = {"January","February","March",
          "April","May","June","July","August","September",
          "October","November","December"};
int i;

for (i = 0; i < 12; i++)
   printf("%s\n",months[i]);
```

- **Example:**

```
char str[10][20];
int i= 0;

while (scanf("%s",str[i]) != EOF && i < 10) {
   printf("%s\n",str[i]);
   i++;
}
```

- **Example:**

```
char animal[3][12];
int i,result;

strcpy(animal[0],"giraffe");
strcpy(animal[1],"tiger");
strcpy(animal[2],"rhinoceros");
for (i = 0; i <= 2; i++) {
   result = strcmp(animal[i],"tiger");
   if (result == 0) {
      printf("tiger was found in position %d\n",i);
      break;
   }
}
```

- **Example:**

```
/*  Function classify:
 *      finds monthname in array months and classifies its
 *      position into one of four seasons
 */
char * classify(char * monthname)
{
    char months[12][10] =  {"January","February","March",
          "April","May","June","July","August","September",
          "October","November","December"};

    int i,found= 0;

    for (i =  0; i < =  11 && !found; i+ + )
       if (strcmp(monthname,months[i]) = =  0) found =  1;
    if (!found) return ("error");
    switch (i-1) {
       case 11:
       case 0:
       case 1:
          return ("winter");
       case 2:
       case 3:
       case 4:
          return ("spring");
       case 5:
       case 6:
       case 7:
          return ("summer");
       case 8:
       case 9:
       case 10:
          return ("autumn");
    }
}
```