22 String processing

When `char *` is used as a type, character strings are almost always intended, that is, arrays in which `'\0'` indicates end of string.

Here are some useful functions.

- **int strlen (char * s)** in **string.h** returns the length of s
- **int strcomp (char * x, char * y)** compares two strings. It is used to sort lines of text. It behaves in the following peculiar way.

  - If x comes before y in lexicographical (dictionary) order, then `strcomp()` returns a negative number. Almost any negative number is possible.
  - If x and y are equal as strings (they have the same length), then `strcmp()` returns 0.
  - If x comes after y in lexicographical (dictionary) order, then `strcomp()` returns a positive number. Almost any positive number is possible.

  Mnemonic: it is as if `strcmp()` returns $x - y$.

  This is also in `string.h`

- **char * fgets (char * buffer, int len, FILE * file)** Reads up to end-of-line (including `'\n'`), or end-of-data, or up to `len` characters ended with `'\0'`, whichever comes first.

  This prevents characters being stored beyond the end of the buffer ('buffer overflow').

  (The word 'buffer' used to mean, among other things, some kind of bin or container for holding coal, etcetera, temporarily.)

  The word FILE is new, and we shall not pay much attention to it. For most purposes,

  ```
    char buffer[200];
  ...
    fgets ( buffer, 200, stdin );
  ```

  will do. There is a 'standard file' `stdin`, and also `stdout` and `stderr`.

  These are in `stdio.h`.

- **snprintf (char * buffer, int len, char * format, item ...)** formats the items for printing like `printf()`, but formats them into the string `buffer` rather than printing to terminal.

  Again, `len` is there to prevent buffer overflow.

  This is in `stdio.h`.

When processing text, the following routine is helpful without being absolutely necessary. It removes the newline from a string.

This is useful because `fgets()` usually includes newlines, but not always. It's best to make sure all newlines are deleted.

```
#include <string.h>
void delete_newline ( char * s )
{
  int last = strlen ( s ) - 1;
        /* last character before '\0' */
  if ( last >= 0 && s[last] == '\n' )
    s[last] = '\0';
}
```

Here is another method, which produces the same result so long as the newline can only be at the end of the string.

```
void delete_newline ( char * s )
{
  int i;
  for ( i=0; s[i] != '\0'; ++i )
  {
    if ( s[i] == '\n' )
    {
      s[i] = '\0';
      return;
    }
  }
}

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

void decr ( char s[] ) // removes newline, if any
{
  int i;
  for ( i=0; s[i] != '\0' && s[i] != '\n'; ++i )
  {}

  s[i] = '\0';
}

int next_word (int i, char s[]) // where next word begins
{
  int found;
  int len = strlen ( s );

  found = 0;
  while ( i < len && !found )
  {
    if ( !isspace ( s[i] ) )
      found = 1;
    else
      ++ i;
  }

  if ( found )
    return i;
  else
    return -1;
}

int word_end ( int i, char s[] )
{
  while ( s[i+1] != ' ' && s[i+1] != '\0' )
  ++i;

  return i;
}

main ()
{
  int i,j,k;
  char buffer[200];

  while ( fgets ( buffer, 200, stdin ) != NULL )
  {
    decr ( buffer );
    i = 0;
    while ( i >= 0 )
    {
      i = next_word ( i, buffer );
      if ( i >= 0 )                // if another word in
                                // buffer, print it.
      {
        k = word_end ( i, buffer );
        for (j=i; j<=k; ++j)
        {
          printf("%c", buffer[j]);
        }
        printf("\n");                // plus newline
        i = k+1;
      }
    }
  }
}
```

For example:

```
File t:
We know that you highly esteem the kind of Learning taught in those
Colleges, and that the Maintenance of our young Men, while with

a.out < t:
We
know
that
you
highly
esteem etcetera
```