# Exercise 9: Recursion

Nanda H Krishna

29 March 2018

## 1 Text processing

**Problem description:** Read a text from `stdin`. Define functions for doing each of the following operations. Test the functions individually.

1. Store the lines as an array of lines, each line a C-string.
2. Store each line as an array of words, each word a C-string.
3. Count the number of lines.
4. Count the number of words.
5. Define a function to search and replace a word by another word.
6. Capitalize the first letter of each line.

**Algorithm development:**

- **Storing lines as array of lines:** Since the delimiter for `scanf()` is a white space lines cannot be read from `stdin` by `scanf()`. Therefore `fgets()` is used read the input.
- **Store each line as an array of words, each word a C-string:** Considering the fact that each word is separated by a white space each line is analysed and broken down in to words.
- **Count the number of lines and Count the number of words:** Since the lines and words are already separated and stored in an array the number of elements in the array will be the number of lines and words in given text.
- **Search and replace:** Each word in the text is compared with the word that is to be replaced and when the word to be replaced is found the word is replaced with new word.
- **Capitalise:** A loop is used to traverse through all elements in the array and first character of each line is capitalised.

**Program:**

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

```
#include<string.h>
int search(char* k[],char s[],char r[])
{
  int i = 0,j = 0, p = 0, h = 0;
  char l[300], word[20], newline[300];
  strcpy(newline,"");
  while(k[h]!=NULL) {
    i = 0;
    strcpy(l,k[h]);
    while(i<strlen(l)) {
 while(l[i]!=' '&&l[i]!='\0') word[j++]=l[i++];
 word[j]='\0';
 if(strcmp(word,s)==0) strcpy(word,r);
 strcat(word," ");
 strcat(newline,word);
 j = 0;
 i++;
    }
    printf("%s \n",newline);
    k[h] = (char*) malloc(sizeof(newline));
    strcpy(k[h],newline);
    strcpy(newline,"");
    h++;
  }
  return 0;
}
void capitalise(char* p[])
{
  int i = 0;
  while(p[i]) {
      p[i][0] = toupper(p[i][0]);
      i++;
  }
}
int count_lines(char* k[])
{
  int count = 0;
  for(;k[count];count++);
  return count;
```

```
}
void store_words(char* k[],char* c[])
{
  int i = 0,j = 0,p = 0,h = 0;
  char l[300],word[20];
  while(k[h]!=NULL) {
    i = 0;
    strcpy(l,k[h]);
    while(i<strlen(l)) {
      while(l[i]!=' '&&l[i]!='\n'&&l[i]!='\0')
word[j++] = l[i++];
     word[j] = '\0';
     c[p] = (char*) malloc(sizeof(word));
     strcpy(c[p],word);
     p++;
     j = 0;
     i++;
    }
    h++;
  }
  c[p]=NULL;
}
int count_words(char *c[])
{
   char *k[200];
   store_words(c,k);
   return count_lines(k);
}
void print_strings(char* c[])
{
  for(int i=0;c[i];i++)
   printf("%s \n",c[i]);
}
int main()
{
  char *p[100];
  int x = 0;
  char inp[300];
  char *c[30];
```

```
  while(fgets(inp,300,stdin)!=NULL) {
      p[x] = (char*) malloc(sizeof(inp));
      strcpy(p[x],inp);
      x++;
  }
  p[x]=NULL;
  printf("\n\n");
  store_words(p,c);
  printf("%d\n", count_lines(p));
  printf("\n%d\n", count_words(p));
  char find[5]="C";
  char replace[5]="D";
  int j = search(p,find,replace);
  print_strings(p);
  printf("\n");
  capitalise(p);
  print_strings(p);
}
```

**Test Input:**

```
Hoorah! Certified programmer!
This
is a C program
written for the record.
```

**Output:**

```
4

12

Hoorah! Certified programmer!
This
is a D program
written for the record.

Hoorah! Certified programmer!
This
Is a D program
Written for the record.
```

## 2 Towers of Hanoi

**Problem description:** Solve the Towers of Hanoi problem using recursion.

**Program design:** The functions are `main()` for testing, `move_tower()` for the processes involved and `move_disk()` to display the process done.

**Algorithm:** The algorithm is as follows:

```
move_tower(n, pole, cw pole, acw pole)
  pre:  tower of size n on pole,
  towers in cw and acw poles are broader than the tower on pole
  post: tower of size n on cw pole
   if n > 0:
      move_tower(n-1, pole, acw pole, cw pole)
      move_disk(pole, cw pole)
      move_tower(n-1, acw pole, cw pole, pole)
```

**Program:**

```
#include<stdio.h>
void move_disk(char from, char to)
{
   printf("Move the topmost disk from %c to %c\n",from,to);
}
void move_tower(int n,char pole, char cw_pole,char acw_pole)
{
   if(n > 0) {
move_tower(n-1,pole,acw_pole,cw_pole);
move_disk( pole,cw_pole);
move_tower(n-1,acw_pole,cw_pole,pole);
    }
}
int main()
{
   int n;
   scanf("%d",&n);
   move_tower(n,'A','C','B');
}
```

**Test Input:**

```
3
```

**Output:**

```
Move the topmost disk from A to C
Move the topmost disk from A to B
Move the topmost disk from C to B
Move the topmost disk from A to C
Move the topmost disk from B to A
Move the topmost disk from B to C
Move the topmost disk from A to C
```