Exercise 9: Recursion

R Ram Kaushik

16-29 March 2018 (Mon-Thurs)

1 Text processing

Read a text from stdin. Define functions for doing each of the following operations. Test the functions individually.

1. Store the lines in 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.

1.1 Specification

5 functions `search_replace()`, which takes the original array, word to be searched and replaceing word as input and modifies the array, `capitalize()`, which takes the array as input and capitalizes the first letter of each line, `count()`, which takes the array as input and returns the number of elements to the calling function, `store_words()`, which takes an array and an empty array as input and stores each word to the empty array, and `print_strings()`, which takes an array as input and prints the array.

1.2 Prototype

```
int search_replace(char* k[],char s[],char r[]);

void capitalise(char* p[]);

int count(char* k[]);

void store_words(char* k[],char* c[]);

void print_strings(char* c[]);
```

1.3 Program Design

The program consists of 5 functions `search_replace(char* k[],char s[],char r[])`,`capitalise(char* p[])`,`count(char* k[])`,`store_words(char* k[],char* c[])`,`print_strings(char* c[])`, which do the necessary task, and `main()`, which gets the input from `stdin`, and calls the function.

1.4 Algorithm

```
def search_replace(k,s,r):
  i,j,p,h=0
  newline=""
  while k[h]!=NULL:
    i=0
    l=k[h]
    while i<len(l):
      while l[i]!=’ ’ and l[i]!=’\0’:
        word[j]=l[i]
j+=1
i+=1
      word[j]=’\0’
      if word==s:
        word=r
      word+=" "
      newline+=word
      j=0
      i++
    print(newline)
    //allocate memory for k[h]
    k[h]=newline
    newline=""
    h++
def capitalise(p):
  i=0
  while p[i]:
    p[i][0]=toupper(p[i][0])
    i+=1
def count(k):
  c=0
  while k[c]:
    c+=1
  return c
def store_words(k,c):
  i,j,h,p=0
  while k[h]!=NULL:
    i=0;
    l=k[h]
    while i<len(l):
      while l[i]!=’ ’ and l[i]!=’\n’ and l[i]!=’\0’:
word[j]=l[i]
j+=1
i+=1
      word[j]=’\0’;
      //allocate size for c[p]
      c[p]=word
      p+=1
      j=0;
      i+=1;
    h++;
  c[p]=NULL;
def print_strings(c):
  i=0
  while c[i]:
    print(c[i])
    i+=1
```

1.5 Source Code

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int search_replace(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(char* k[]){
  int c=0;
  for(;k[c];c++){
    ;
  }
  return c;
}
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;

}
void print_strings(char* c[]){
  for(int i=0;c[i];i++){
    printf("%s \n",c[i]);
  }
}

int main(){
  char *p[100], *c[30];
  int x=0;
  char inp[300],find[50],replace[50];
  while(fgets(inp,300,stdin)!=NULL){
    p[x]=(char*) malloc(sizeof(inp));
    strcpy(p[x],inp);
    x++;
  }
  p[x]=NULL;
  store_words(p,c);
  printf("%d\n", count(p));
  printf("%d\n", count(c));
  printf("\n");
  strcpy(find,"is");
  strcpy(replace,"to");
  scanf("%s%s",find,replace);
  int j=search_replace(p,find,replace);
  print_strings(p);
  printf("\n");
  capitalise(p);
  print_strings(p);
}
```

```
5
24
My   name      to    Ram      Kaushik.
I    am        18    years    old       and
my   ambition  to    to       study
at   MIT.      My    hobby
to   to        play  sports.
My   name      to    Ram      Kaushik.
I    am        18    years    old       and
my   ambition  to    to       study
at   MIT.      My    hobby
to   to        play  sports.

My   name      to    Ram      Kaushik.
I    am        18    years    old       and
My   ambition  to    to       study
At   MIT.      My    hobby
To   to        play  sports.
```

1.6 Test Input

```
My name is Ram Kaushik.
I am 18 years old and
```

```
my ambition is to study
at MIT. My hobby
is to play sports.
```

1.7 Output

```
5
24

My  name      to    Ram     Kaushik.

I   am        18    years   old       and

my  ambition  to    to      study

at  MIT.      My    hobby

to  to        play  sports.

My  name      to    Ram     Kaushik.

I   am        18    years   old       and

My  ambition  to    to      study

At  MIT.      My    hobby

To  to        play  sports.
```

2 Tower of Hanoi

There are three poles fixed in the ground. On the first of these poles, 8 discs are placed, each of different size, in decreasing order of size. How will you move the discs from its pole to the clockwise pole (`cw_pole`) according to the rule that no disc may ever be above a smaller disc. Figures 1.

We can solve the problem recursivley.

- Base case: There is no disc in the pole.
- Recursion step: Reduce the size of the tower to $n-1$ discs. Move the tower of top $n-1$ discs to the anti-clockwise pole. Move the exposed disc (n) on the pole to the clockwise pole. Then, move the tower of $n-1$ discs from anti-clockwise pole to the clockwise pole. This idea is illustrated in Figure 2. Define `hanoi()`. Let the function print the sequence of moves on the `stdout`.

```
1: 1 -> 2
```

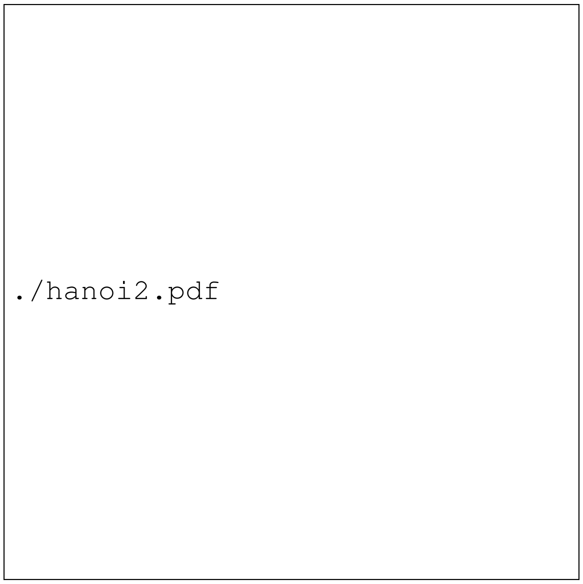

Figure 1: Tower of Hanoi, pole, clockwise pole, anti-clockwise pole

```
2: 1 -> 3

...
```

```
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)
```

2.1 Specification

2 functions `print()`, whcih takes 2 characters as input and prints the result, and `tower_of_hanoi()`, which takes an integer and 3 characters as the input and recursively calls itself and does the required steps.

2.2 Prototype

```
void print(char c, char d);
void tower_of_hanoi(int n, char fpole, char tpole, char apole)
```

2.3 Program Design

The program consists of 2 functions `print(char c, char d)`, which prints the result on `stdout`,`tower_of_hanoi(int n, char fpole, char tpole, char apole)`, which calls itself recursively until the condition is satisfied, and `main()`, which gets the input from `stdin`, and calls the function.

2.4 Algorithm

```
def tower_of_hanoi(n,fpole,tpole,apole):
  if n>0:
    tower_of_hanoi(n-1,fpole,apole,tpole)
    print(fpole,tpole)
    tower_of_hanoi(n-1,apole,tpole,fpole)
```

2.5 Source Code

```
#include<stdio.h>
void print(char c, char d){
  printf("%c->%c\n", c, d);
}
void tower_of_hanoi(int n, char fpole, char tpole, char apole){
  if(n>0){
    tower_of_hanoi(n-1,fpole,apole,tpole);
    print(fpole,tpole);
    tower_of_hanoi(n-1,apole,tpole,fpole);
  }
}
int main(){
  int n;
  scanf("%d",&n);
  tower_of_hanoi(n,’A’,’B’,’C’);
}
```

2.6 Test Input

```
3
```

2.7 Output

```
A->B
A->C
B->C
A->B
C->A
C->B
A->B
```

./hanoi5.pdf

Figure 2: Tower of Hanoi: move tower in two recursive steps