Exercise 8: Arrays of Strings

Nanda H Krishna

22 March 2018

1 Count the number of strings

Problem description: Define a function to count the number of strings in the array of strings.

Specification: Function `strings_length()` takes an array of pointers as input and returns the length of the array.

Prototype:

```
int strings_length(char* names[])
```

Program design: The program consists of `strings_length(char* names[])` which counts the number of strings, and `main()` which is used for testing.

Program:

```
#include<stdio.h>
int strings_length(char* names[])
{
  int i;
  for(i = 0; names[i] != NULL; i++);
  return i;
}
int main()
{
  char* a[13] = {"January","February","March",
 "April","May","June","July",
 "August","September","October",
 "November","December", NULL};
  printf("%d\n", strings_length(a));
  return 0;
}
```

Output:

```
12
```

2 Print an array of strings

Problem description: Define a function to print the array of strings.

Specification: The function takes the array of pointers as input and the output is the strings printed on `stdout`.

Prototype:

```
void strings_print(char* a[])
```

Program design: The function `strings_print(char* names[])` is used to print the array of strings and `main()` is used for testing.

Program:

```
#include<stdio.h>
  int strings_print(char* a[])
  {
    int i;
    for(i = 0; a[i] != NULL; i++) printf("%s\n", a[i]);
  }
  int main()
  {
    char* a[13] = {"January","February","March",
   "April","May","June","July",
   "August","September","October",
   "November","December", NULL};
    strings_print(a);
    return 0;
  }
```

Output:

```
January
February
March
April
May
June
```

```
July
August
September
October
November
December
```

3 Cloning a string

Problem description: Write a function to create a clone for a C-string.

Specification: The function `string_clone()` takes a string and returns the address of the clone.

Prototype:

```
char* string_clone(char s[])
```

Program design: The program consists of a function `string_clone(char s[])`, which clones a string, and `main()`, which calls the function and prints the result on `stdout`.

Program:

```
#include<stdio.h>
char* string_clone(char s[])
{
  char *t = (char*)malloc(strlen(s));
  strcpy(t, s);
  return t;
}
int main()
{
  char *s = "In the beginning was the word.";
  char *t = string_clone(s);
  printf("%p %s\n", &s, s);
  printf("%p %s", &t, t);
}
```

Output:

```
0x7ffe2cfff798 In the beginning was the word.
0x7ffe2cfff7a0 In the beginning was the word.
```

4 Read a sequence of lines from `stdin`

Problem description: Write a function `strings_read(lines)` to read a sequence of lines from `stdin`. It stores the lines in an array of strings `char* lines[]`, and returns the count of lines as the result. After reading each line from `stdin`, allocate memory using `string_clone()` and store it as a string in `char* lines[]`. Read the class name-list from stdin. Sort it and print it.

Specification: Function `strings_read()` gets an array of pointers as input, reads the array and returns the length.

Prototype:

```
char* string_clone(char s[])
int strings_read(char* names[])
void print_string(char* names[], int n)
```

Program design: The program consists of `string_clone(char s[])`,`strings_read(char* names[])`, and `print_string(char* names[], int n)`, all of which help to read the lines from `stdin` and print it on `stdout`. The `main()`, which calls the function.

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define N 100
#define MAXLINE 1000
char* string_clone(char s[])
{
  char* t = (char*)malloc(strlen(s)+1);
  strcpy(t,s);
  return t;
}
int strings_read(char* names[])
{
  char line[MAXLINE];
  int i;
  for(i = 0; fgets(line, MAXLINE, stdin) != NULL; i++) {
    int n = strlen(line);
    line[n-1] = '\0';
    names[i] = string_clone(line);
  }
  return i;
}
void print_string(char* names[], int n)
{
  for(int i = 0; i < n; i++) {
    printf("%s\n",names[i]);
  }
}
int main()
{
  char* names[N];
  int n = strings_read(names);
  print_string(names, n);
}
```

Test Input:

```
Alpha 001
Bravo 002
Charlie 003
Delta 004
Ergo 005
Fuhrer 006
Bond 007
```

Output:

```
Alpha 001
Bravo 002
Charlie 003
Delta 004
Ergo 005
Fuhrer 006
Bond 007
```

5 Sort an array of strings

5.1 Alphabetical order

Problem description: Sort in alphabetical order an array of strings, using selection sort.

Specification: Function `swap()` gets 2 strings as inputs and swaps them, `strings_print()`

gets an array of pointers as input and prints the strings, `min()` gets an array of pointers and 2 indices as inputs, and returns the index of the lowest string and `sel_sort()` gets an array of pointers and its length as input, and sorts the array in alphabetical order.

Prototype:

```
void swap(char* a[], int i, int j)
void strings_print(char* names[], int n)
int min(char* a[], int low, int high)
void sel_sort(char* a[], int n)
```

Program design: The program consists of `swap(char a[], char b[])`,`strings_print (char* names[])`,`min(char* a[], int low, int high)` and `sel_sort(char* a[], int n)`, which are used to sort the strings in alphabetical order, and `main()`, which calls the functions and prints the result on `stdout`.

Algorithm: The algorithm to sort is as follows:

```
min(a[], l, h):
  p = low
  for i in range(l+1, h):
    if a[i] < a[p]:
      p = i
  return p
sel_sort(a[], n):
  for i in range(n):
    m = min(a, i, n)
    swap(a[i], a[m])
```

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void swap(char* a[], int i, int j)
{
  char* t = a[i];
  a[i] = a[j];
  a[j] = t;
}
void strings_print(char* names[], int n)
{
  for (int i = 0; i < n; i++)
    printf("%s\n", names[i]);
}
int min(char* a[], int low, int high)
{
  int i, p = low;
  for(i = low + 1; i < high; i++) {
    if(strcmp(a[i], a[p]) < 0) p = i;
  }
  return p;
}
void sel_sort(char* a[], int n)
{
  for(int i = 0; i < n - 1; i++) {
    int m = min(a, i, n);
    swap(a, i, m);
  }
}
int main()
{
  char* a[13] = {"January","February","March",
 "April","May","June","July",
 "August","September","October",
 "November","December"};
  sel_sort(a, 12);
  strings_print(a, 12);
  return 0;
}
```

Output:

```
April
August
December
February
January
July
June
March
May
November
October
```

```
September
```

5.2 Based on string length

Problem description: Sort an array of strings in ascending order of the string lengths.

Specification: Function `string_clone()` gets a string as input, clones the string and returns it to the calling function, `strings_read()` gets an array of pointers as input, reads the array and returns the length, `minimum()` gets an array of pointers and 2 indices as input and returns the index of the string with minimum length to the calling function, `swap()` gets an array of pointers and 2 indices as input and swaps the two strings at those indices, `sel_sort_len()` gets an array of pointers and length as input and sorts the array based on length, and `print_string()` gets an array of pointers and its length as input and prints the output.

Prototype:

```
char* string_clone(char s[])
int strings_read(char* names[])
int minimum(char* a[], int l, int h)
void swap(char* m[], int a, int b)
void sel_sort_len(char* m[], int l, int h)
void print_string(char* names[], int n)
```

Program design: The program consists of `string_clone(char s[])`,`strings_read(char* names[])`,`minimum(char* a[], int l, int h)`,`swap(char* m[], int a, int b)`,`sel_sort_len(char* m[], int l, int h)` and `print_string(char* names[], int n)`, all of which help to sort the strings from `stdin` and print it on `stdout`, and `main()`, which is used for testing.

Algorithm: The algorithm to sort is as follows:

```
minimum(a[], l, h):
  m = l
  for i in range(l+1, h):
    if len(a[m]) > len(a[i]):
      m = i
  return m
sel_sort_len(a, l, h):
  for i in range(l, h-1):
    m = minimum(a, i, h)
    swap(a[i], a[m])
```

Program:

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

```
#include<string.h>
#include<stdlib.h>
#define N 100
#define MAXLINE 1000
char* string_clone(char s[])
{
  char* t = (char*)malloc(strlen(s) + 1);
  strcpy(t, s);
  return t;
}
int strings_read(char* names[])
{
  char line[MAXLINE];
  int i;
  for(i = 0; fgets(line, MAXLINE, stdin) != NULL; i++) {
    int n = strlen(line);
    line[n-1] = ’\0’;
    names[i] = string_clone(line);
  }
  return i;
}
int minimum(char* a[], int l, int h)
{
  int i, m = l;
  for(i = l+1; i < h; i++) {
    if(strlen(a[m]) > strlen(a[i]))
      m = i;
  }
  return m;
}
void swap(char* m[],int a,int b)
{
  char* t = m[a];
  m[a] = m[b];
  m[b] = t;
}
void sel_sort_len(char* m[], int l, int h)
{
  int min;
  for(int i = l; i < h - 1; i++) {
    min = minimum(m,i,h);
    swap(m, i, min);
  }
}
void strings_print(char* names[], int n)
{
  for(int i = 0; i < n; i++)
    printf("%s\n",names[i]);
}
int main()
{
  char* names[100];
  int n = strings_read(names);
  sel_sort_len(names, 0, n);
  strings_print(names, n);
}
```

Test Input:

```
Nayeon
Momo
Jeongyeon
Mina
Dahyun
Chaeyoung
Jihyo
Sana
Tzuyu
```

Output:

```
Momo
Mina
Sana
Jihyo
Tzuyu
Nayeon
Dahyun
Jeongyeon
Chaeyoung
```

6 Search for a string

Problem description: Search for a string in a sorted array of strings.

Specification: Function `string_clone()` gets a string as input, clones the string and returns it to the calling function, `read_line()`, gets an array of pointers as input, reads the array and returns the length, and `binary_partition()`, which gets an array of pointers, a string, and 2 indices as inputs and returns the index where the string is found.

Prototype:

```
char* string_clone(char s[])
int strings_read(char* names[])
int binary_partition(char* m[],char n[],int low,int high)
```

Program design: The program consists of `string_clone(char s[])`,`strings_read(char* names[])`, and `binary_partition(char* m[],char n[],int low,int high)`, all of which help to get the input from `stdin` and find the index, and `main()`, which tests the functions.

Algorithm: The algorithm to search using binary partition is as follows:

```
binary_partition(m,n,l,h):
  while l != h:
    mid = (l+h)//2
    if n == m[mid]:
      return mid
    elif n < m[mid]:
      h = mid
    else:
      l = mid + 1
  return mid
```

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100
#define MAXLINE 1000
char* string_clone(char s[])
{
  char* t = (char*)malloc(strlen(s));
  strcpy(t, s);
  return t;
}
int strings_read(char* names[])
{
  char line[MAXLINE];
  int i;
  for(i = 0; fgets(line,MAXLINE,stdin) != NULL; i++) {
    int n = strlen(line);
    line[n - 1] = ’\0’;
    names[i] = string_clone(line);
  }
  return i;
}
int binary_partition(char* m[],char n[],int low,int high)
{
  int mid;
  while(low != high) {
    mid = (low+high)/2;
    if(strcmp(n, m[mid]) == 0)
      return mid;
    else if(strcmp(n, m[mid]) < 0){
      high = mid;
    }
    else {
      low = mid + 1;
    }
  }
  return high;
}
int main()
{
  char* names[N];
  int n = strings_read(names);
  int r = binary_partition(names,"God", 0, n);
  printf("%d\n",r);
}
```

Test Input:

```
Ant
Boy
```

```
Cat
Dog
Elf
God
Hat
Ink
Jet
```

Output:

```
5
```

7 Insert a string in a sorted array of strings

Problem description: Insert a string in the sorted array of strings using the `binary_partition()` function to obtain the right position.

Specification: Function `string_clone()` gets a string as input, clones the string and returns it to the calling function, `strings_read()` gets an array of pointers as input, reads the array and returns the length, `binary_partition()` gets an array of pointers, a string, and 2 indices as inputs and returns the index where the string can be inserted, `insert()` gets an array of pointers, a string and 2 indices as inputs and adds the string to the array, and `print_string()` gets an array of pointers and its length as input and prints the output.

Prototype:

```
char* string_clone(char s[])
int strings_read(char* names[])
int binary_partition(char* m[],char n[],int low,int high)
void insert(char* a[], char k[],int r,int* n)
void print_string(char* names[],int n)
```

Program design: The program consists of `string_clone(char s[]),read_line(char* names[]),binary_partition(char* m[],char n[],int low,int high),insert(char* a[], char k[],int r,int* n)` and `print_string(char* names[],int n)`,all of which help to get the input from `stdin`, find the index, insert, and print it on `stdout`, and `main()`, which tests the function.

Algorithm: The algorithm to insert is as follows:

```
insert(a, k, r, n):
  int i = n - 1
  while i >= r:
    a[i+1] = a[i]
    i--
  a[r] = k
  n += 1
  return n
```

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define N 100
#define MAXLINE 1000
char* string_clone(char s[])
{
  char* t = (char*)malloc(strlen(s));
  strcpy(t,s);
  return t;
}
int strings_read(char* names[])
{
  char line[MAXLINE];
  int i;
  for(i = 0; fgets(line,MAXLINE,stdin) != NULL; i++) {
    int n = strlen(line);
    line[n - 1] = '\0';
    names[i] = string_clone(line);
  }
  return i;
}
void string_print(char* names[],int low ,int high)
{
  for(int i = low; i < high; i++) {
    printf("%s,\n",names[i]);
  }
  printf("\n");
}
int binary_partition(char* m[],char n[],int low,int high)
{
  int mid;
  while(low != high) {
    mid = (low+high)/2;
    if(strcmp(n,m[mid]) == 0)
      return mid + 1;
    else if(strcmp(n,m[mid]) < 0) {
      high = mid;
    }
    else {
      low = mid + 1;
    }
  }
  return high + 1;
}
void insert(char* a[], char k[],int r,int* n){
  int i = *n - 1;
  while(i >= r) {
    a[i+1] = a[i];
    i--;
  }
  a[r] = (char*)malloc(strlen(k)+1);
  strcpy(a[r], k);
  (*n)++;
}
int main()
{
  char* names[N];
  int n = strings_read(names);
  int r = binary_partition(names,"God",0,n);
  insert(names,"Goddess",r,&n);
  string_print(names,0,n);
}
```

Test Input:

```
Ant
Boy
Cat
Dog
Elf
God
Hat
```

```
Ink
Jet
```

Output:

```
Ant
Boy
Cat
Dog
Elf
God
Goddess
Hat
Ink
Jet
```