CS19001: Programming and Data Structures Laboratory

Soumyajit Dey, Aritra Hazra;
CSE, IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2018/CS19101_PDS-Lab_Autumn2018.html

08-Oct-2018

Table of Contents

Characters

Declaration and Initialization

```
char ch = ‘a’;   OR   char ch; ch = ‘a’;
```

ASCII Values of Characters

Every character has an integer ASCII value and you can get that by printing it in integer format.

```
char ch = ‘a’;
printf(‘‘%d’’,ch);
   // prints ASCII value (97) of ‘a’
```

Let us not memorize the ASCII values (of a-z, A-Z and 0-9). It can easily be assigned to any integer and can be found/operated. Moreover, integers and characters are inter-operable.

```
int x = ‘A’;
printf(‘‘%c’’,x+3);
   // prints the character ‘D’
```

Character manipulations

Example: Simple text encryption

Caesar cipher is a simple technique of encryption of plain text byreplacing every character in the plain text by a character fixed number of positions down the list of the alphabet. The last characters are folded back to the beginning. The numerical digits and all other characters will remain unchanged.

Shift: 5		Shift: 2	
Original	Encrypted	Original	Encrypted
'A'	'F'	'a'	'c'
'B'	'G'	'b'	'd'
⋮	⋮	⋮	⋮
'Y'	'D'	'y'	'a'
'Z'	'E'	'z'	'b'

Let us program to read a text stream and will encrypt the English alphabets, [A - Z] and [a - z], using Caesar cipher. The value of shift should be within 1 – 10 and will be decided by the `rand()` function.

C-Program: Simple text encryption

```
#include <stdio.h>
#include <stdlib.h> // for rand()
#include <ctype.h> // for isalpha()
int main()
{
  char c, shift;
  // generating random shift
  shift = (char)(rand()%10 + 1);
  while((c = getchar()) != EOF) {
    if(isalpha(c)) { // checking for alphabets
      if(isupper(c)) // upper-case alphabet
        putchar((c-‘A’+shift)%26+‘A’);
      else // lower-case alphabet
        putchar((c-‘a’+shift)%26+‘a’);
    }
    else putchar(c); // other characters unchanged
  }
  putchar(‘\n’);
  return 0;
}
```

Table of Contents

1 Tutorial: Characters

2 Tutorial: Strings

3 Assignments

Strings

In C, a string is defined to be a null-terminated character array. The null character '\0' is used to indicate the end of the string.

```
int main ()
{
  char greet[3]={’H’,’i’,’\0’};
  printf("Greeting message: %s\n",greet);
  return 0;
}
```

Variation in initialization

```
char c[]="abcd";
char c[5]="abcd";
char c[]={’a’,’b’,’c’,’d’,’\0’};
char c[5]={’a’,’b’,’c’,’d’,’\0’};
```

Reading string from terminal

```c
#include <stdio.h>
int main(){
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.",name);
    return 0;
}
```

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

- scanf() function takes only string before the white space.

Reading a line of text

```
int main(){
    char name[30],ch;
    int i=0;
    printf("Enter name: ");
    while(ch!='\n')
    {// terminates if user hit enter
        ch=getchar();
        name[i]=ch;
        i++;
    }// inserting null character at end
    name[i]='\0';
    printf("Name: %s",name);
    return 0;
}
```

Better method

```c
int main(){
    char name[30];
    printf("Enter name: ");
    gets(name);
    //Function to read string from user.
    printf("Name: ");
    puts(name);
    //Function to display string.
    return 0;
}
```

```
Enter name: Dennis Ritchie
Name: Dennis Ritchie
```

Passing Strings to Functions

```c
void Display(char ch[]);
int main(){
    char c[50];
    printf("Enter string: ");
    gets(c);
    Display(c);
    // Passing string c to function.
    return 0;
}
void Display(char ch[]){
    printf("String Output: ");
    puts(ch);
}
```

Library functions

```
#include <stdio.h>
#include <string.h>
int main ()
{
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;
   strcpy(str3, str1);
   printf("strcpy(str3,str1): %s\n",str3);
   strcat( str1, str2);
   printf("strcat(str1,str2): %s\n",str1);
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );
   return 0;
}
```

Result

strcpy(str3, str1) : Hello
strcat(str1, str2): HelloWorld
strlen(str1) : 10

- do not forget to include string.h

A bit more about string manipulation

- `int strcmp (char s[ ], char t[ ])`:
  Returns 0 if the two strings are identical, a negative value if s is lexicographically smaller than t (i.e., if s comes before t in the standard dictionary order), and a positive value if s is lexicographically larger than t. Comparison is done with respect to ASCII values (A - 65, a - 95)
- `int strlen (char s[ ])`:
  Returns the length (the number of characters before the first null character) of the string s.

Table of Contents

1 Tutorial: Characters

2 Tutorial: Strings

3 Assignments

Programming Assignments

Complete and submit during lab

Assignment 1 [Text-Stats]

Write a C-program to perform the following:

- Ask the user to input some text/string (may contain anything that can be entered via keyboard including spaces, tabs, new-lines etc.). The end of entry will be determined by pressing $\langle Ctrl + D \rangle$ keys (together).
- Computes and Displays the following statistics:
  1. the number of lines present in the text;
  2. the number of words presnt in the text;
  3. the number of total characters present in the text;
  4. the number of lower-case alphabets, upper-case alphabets and numeric digits present in the text (report these three statistics additionally);
  5. the number of spaces entered in the text; and
  6. the number of tabs entered in the text.

Assignment 2 [Code-Word]

Assuming that the fixed codes for the alphabets [a - z] are given as, ‘a’=1, ‘b’=2, . . ., ‘y’=25 and ‘z’=26, write a recursive function to generate all possible alphabatic words from a given code string.

Example:

Let the given code string be, ``1123''.
Then, all the possible alphabetic words are:

```
aabc // a =  1, a =  1, b =  2, c =  3
aaw  // a =  1, a =  1, w = 23
alc  // a =  1, l = 12, c =  3
kbc  // k = 11, b =  2, c =  3
kw   // k = 11, w = 23
```

Write a (C-program) main function that takes a code string from the user and displays all the possible alphabetic words.

Assignment 3 [Rotation-Equivalence]

Definitions

***k*-rotation:** For any string *str* of length n, the k-rotation of *str* from index i $(0 < i + k \leq n)$ creates a new string, where *only* the k-character substring of *str* (from index i), i.e. $str[i..(i + k - 1)]$, is reversed/rotated and all other characters remain intact.

***k*-rotation equivalence:** A string $str1$ is said to be k-rotation equivalent with another string $str2$, if the k-rotation from any index i $(0 < i + k \leq n)$ of $str1$ can produce identical $str2$ (both $str1$ and $str2$ are of equal length n).

Example

Let, $str1 =$ ``abacus'', $str2 =$ ``abucas'' and $str3 =$ ``baacsu''. Suppose, $k = 3$.
Then, $str1$ is 3-rotation equivalent with $str2$, because $str1$ can be 3-rotated from index 2 to get $str2$.
However, $str1$ is NOT 3-rotation equivalent with $str3$.

Assignment 3 [Rotation-Equivalence]

Recursive *k*-rotation function: `rotateStr(char str[], int idx, int k);`

Write a **recursive** function `rotateStr` which produces rotations in the *k*-length substring of *str* starting from index, *idx*.

Write a C-Program (main) that,

- prompts the user to enter two strings, `String1` and `String2`[a];
- asks the user to enter a rotation length[b], say *k*;
- uses the `rotateStr` function and finds out whether `String2` is a *k-rotation equivalent* of `String1`,
- if yes, reports the index of `String1` from which *k*-rotation creates `String2`. Otherwise, reports the *non-equivalence* as a result.

[a] Remember, the first criteria of equivalence between any two strings is that they must be of equal size – *so make appropriate checks for that*!

[b] If the rotation length is more than the string length, automatically *it will be set to string length*, which is the maximum applicable part!

Thank You