CS1100 - Lecture 18

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

In the last lecture we introduced strings. Strings are character arrays which terminate with a null character (\0). In this lecture we will develop some string handling programs using ideas developed in the previous lecture.

Upper case conversion

First we will develop a program that takes a sequence of alphabets from user and computes a new string in which all letters in the input string are replaced by their corresponding upper case letters.

As explained in the previous class, the integer value of `'A'-'a'` is same as the difference between the integer value of any upper case letter and the integer value of its corresponding lower case letter. Suppose we define an integer variable `shift` whose value is `'A'-'a'`. If we add `shift` to a lower case letter, we get the corresponding upper case letter. For example, `'b'+shift` has the same value as the expression `'b'+'A'-'a'` which has the same value as `'b'-'a'+'A'` which has the same value as `1+'A'` which is the same value as `'B'`. This is the basic property that we use in this program.

The following program is a first attempt for upper case conversion.

```
#include<stdio.h>
int main()
{
   char s1[20], s2[20];
   int i, shift, errorFlag=0;

   printf("Give a string to convert\n");
   scanf("%s",s1);
   i=0;
   shift='A'-'a';
   while(i<19&&s1[i]!='\0')
   {
       if(s1[i] >= 'a' && s1[i] <= 'z')
       {
           s2[i]=s1[i]+shift;
       }
       else if(s1[i] >= 'A' && s1[i] <= 'Z')
       {
           s2[i]=s1[i];
       }
       else
```

```
        {
            errorFlag=1;
            break;
        }
        i++;
    }
    s2[i]='\0';

    if(errorFlag)
       printf("Error! The string has non-alphabets \n");
    else
       printf("new string is %s \n", s2);
    return 0;
}
```

The above program takes a string `s1` from the user as input. The length of the character array `s1` is 20. So it can store strings with length at most 19 non-null characters. If we input a string of length smaller than 20, a `\0` character is automatically after all the other characters. The condition `if(s1[i] >= 'a' && s1[i] <= 'z')` checks if the integer value corresponding to the each character of the string `s1` is either equal to that of `'a', 'z'` or lies between these two (i.e., if the character `s1[i]` is lower case character). If the condition becomes true, `s1[i]+shift` is assigned to `s2[i]`. As we explained already, this makes `s2[i]` to get assigned with the upper case character corresponding to the lower case character `s1[i]`. If the condition `if(s1[i] >= 'a' && s1[i] <= 'z')` is false, the program checks `if(s1[i] >= 'A' && s1[i] <= 'Z')`, i.e., if `s1[i]` is an upper case character. If this condition is true, `s1[i]` is copied to `s2[i]`. If both these conditions are false, an `errorFlag` is set and the loop is exited. This process repeats from `i=0`, till `i` becomes 19 or a `'\0'` (null character) is reached. After exiting the `while` loop, the string `s2` will contain the upper characters of corresponding lower case letters in `s1`. A `'\0'` character is appended to `s2` to mark the end of string. This is to prevent a buffer overflow when we later try to print the string `s2`. If `errorFlag` is not set, the string `s2` is displayed.

Sample output of the program corresponding to the input "Hello" is as follows.

```
Give a string to convert
Hello
new string is HELLO
```

The above program works if we enter strings of at most 19 characters length. If we try to input a string which has length more than 19 characters, this might cause a buffer overflow or a segmentation fault.

If we enter a string with spaces in it, the above program will not read the entire string. This is because here the string is read using `%s` which reads characters until the first white space character is encountered.

Suppose we give an input like "Hello how are you", the following output will be displayed.

```
Give a string to convert
Hello how are you?
new string is HELLO
```

There are two modifications that can be used for rectifying the above program. We can restrict the number of characters input to `s1` to prevent buffer overflows. Moreover, we can modify the format string used in `scanf` to allow input strings containing spaces. If we use the instruction `scanf(``%19[^\n]s'',s1);` to input `s1`, then only a maximum of 19 characters are stored into the array `s1`. Input to the array stops when either 19 characters have already been stored or a newline character `\n` is entered, whichever happens earlier. A null character `'\0'` is automatically appended to `s1` after other input characters.

About the field width specification in the format string, the following description is given in `man` pages for `scanf`:

> *An optional decimal integer which specifies the maximum field width. Reading of characters stops either when this maximum is reached or when a non-matching character is found, whichever happens first. Most conversions discard initial white space characters (the exceptions are noted below), and these discarded characters don't count toward the maximum field width. String input conversions store a terminating null byte ('\0') to mark the end of the input; the maximum field width does not include this terminator.*

The modified program is given below.

```
#include<stdio.h>
int main()
{
   char s1[20], s2[20], s3[20];
   int i, shift, errorFlag=0;

   printf("Give a string to convert\n");
   scanf("%19[^\n]s",s1);

   i=0;
   shift='A'-'a';
   while(i<19&&s1[i]!='\0')
   {
       if(s1[i] >= 'a' && s1[i] <= 'z')
       {
           s2[i]=s1[i]+shift;
       }
       else if(s1[i]==' ' || s1[i] >= 'A' && s1[i] <= 'Z')
       {
           s2[i]=s1[i];
       }
       else
       {
           errorFlag=1;
           break;
```

```
        }
        i++;
    }

    s2[i]='\0';
    if(errorFlag)
       printf("Error! The string has non-alphabets \n");
    else
       printf("new string is %s \n", s2);
    return 0;
}
```

If we give the input as "hEllO How Are You", after executing the program the following output will be displayed.

```
Give a string to convert
hEllO How Are You
new string is HELLO HOW ARE YOU
```

If we give the input as "This is a string for checking buffer overflow", after executing the program the following output will be displayed.

```
Give a string to convert
This is a string for checking buffer overflow
new string is THIS IS A STRING FO
```

Note that, big strings do not cause a buffer overflow in this modified program.

Substring search

A sequence of characters occurring in consecutive positions of a given string is called a substring of the given string.
Consider a string ``hellow how are you''. Any character occurring in the string is a substring of length 1. ``el'', ``ow'', ``are you'' etc. are also substrings of the given string. But, ``low!'', ``low d'' etc. are not substrings of the given string.

We now consider the problem of taking two strings `s1` and `s2` as inputs and checking whether `s2` is a substring of `s1`. If `s2` is a substring of `s1` we need to find the starting position of the first occurrence of the substring `s2` in the string `s1`. If `s2` is not a substring of `s1` we need to output a message to this effect.

An overview of the method we are planning to use is given below.

```
1. found=0
2. Input s1, compute the length of s1 and store it in l1
3. Input s2, compute the length of s2 and store it in l2
3. for(pos1=0;pos<l1-l2;pos++)
       check if l2 charactes from pos1 in s1 match with s2.
       if yes, found=1; break;
4. if found is 1
     the substring is found, output pos1.
   else
     the substring is not found.
```

In the above method, `s1` is the first string and $l1$ is the length of this string. The substring to be searched is `s2` and $l2$ is its length. The variable `pos1` stores the current starting position of our search in `s1`. Initially, `pos1` is 0 and the first $l2$ characters of the string `s1` is checked with the substring `s2`. If there is a match, the searching is stopped. Otherwise the `pos1` is incremented to 1 and $l2$ characters of `s1` from position `pos1` are compared with the string `s2` and so on. This repeats until either the substring is found or `pos1` becomes $l1 - l2$. After this, there is no possibility of finding `s2`, because the number of remaining characters in $l1$ is less than the length of the string `s2`.

To check if $l2$ characters from `pos1` in `s1` match with `s2`, we can do the following:

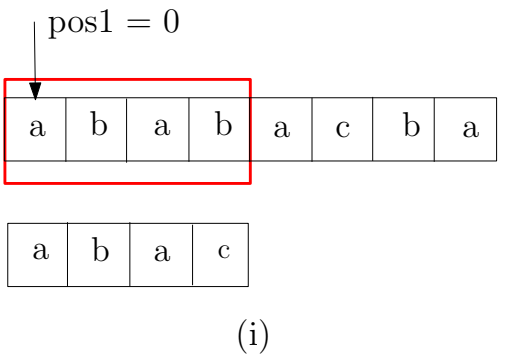
```
pos2=0;
while(pos2 < l2 && s1[pos1 + pos2]==s2[pos2])
      pos2++;
if(pos2 == l2)
      found=1;
```

If the variable `found` was set to 0 before entering the above block of code, this variable will become 1 after executing the block of code only if `s2` occurs as a substring of `s1` starting from `pos1`. Note that, the position indicator `pos1` of the string `s1` is not varied when this code is executed.

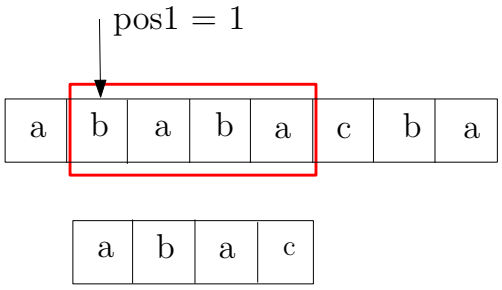
The following code segment extends the above idea by repeating the search by varying `pos1` from 0 to $l1 - l2$ in the string `s1`.

```
found=0;
for(pos1=0; pos1<= l1 -l2; pos1++)
{
        pos2=0;
        while(pos2 < l2 && s1[pos1 + pos2]==s2[pos2])
              pos2++;
        if(pos2 == l2)
        {
            found=1;
            break;
        }
}
```

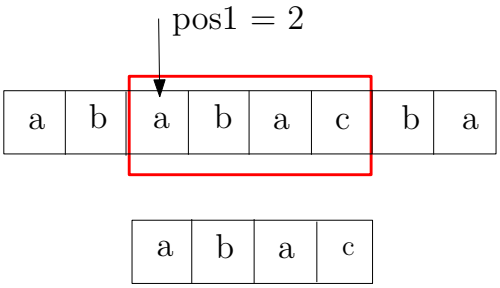
The following figure shows the above method applied to strings `s1=‘‘ababacba’’` and `s2=‘‘abac’’`.



(i)



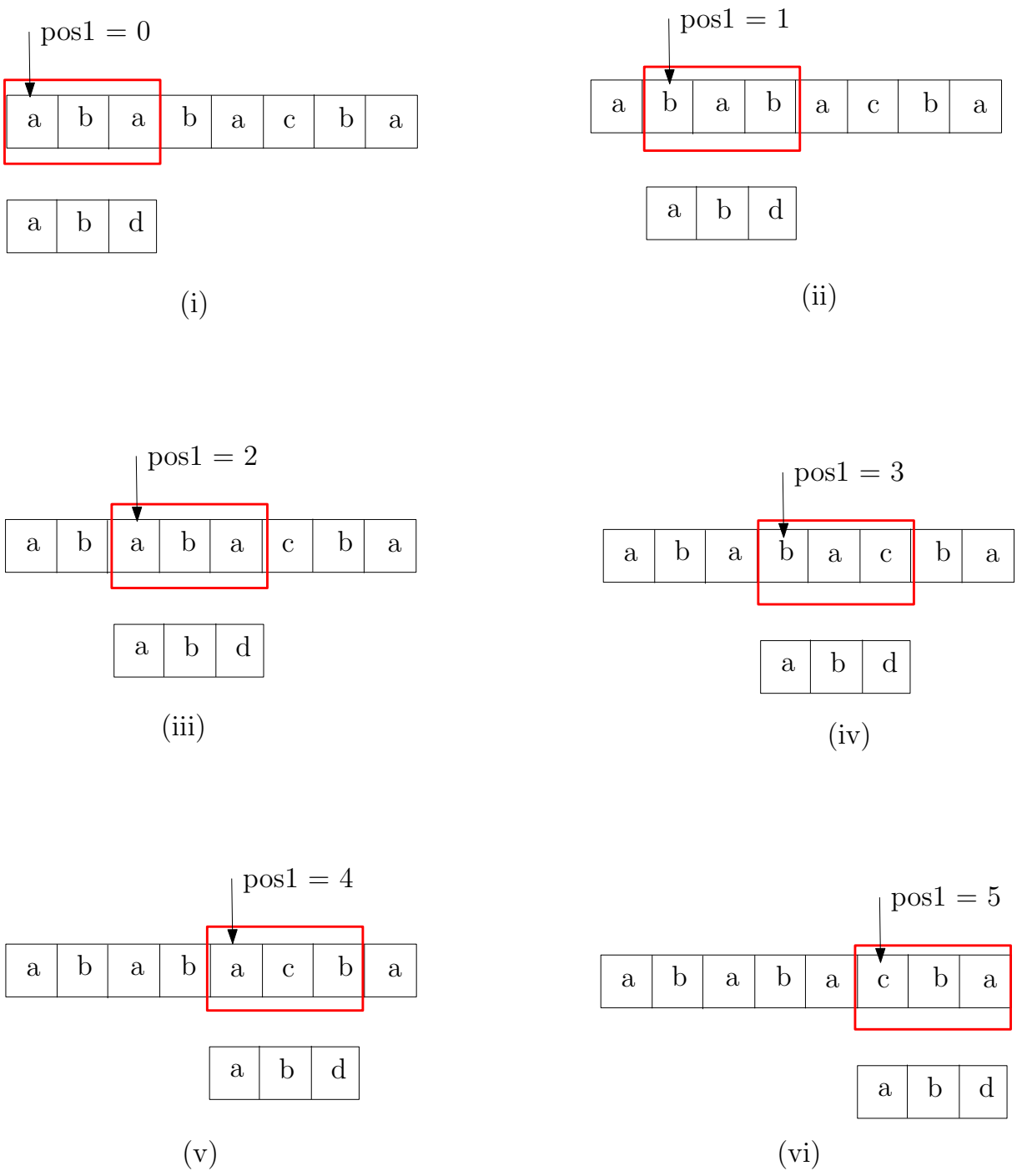
(ii)



(iii)

The substring `s2` is found at position 2.

The following figure shows the method given in page 5 applied to strings `s1=‘‘ababacba’’` and `s2=‘‘abd’’`.



The substring `s2` is not found in the search.

Using the method developed earlier we give a complete program for substring search below.

```
#include<stdio.h>
int main()
{
   char s1[20], s2[20], ch;
   int l1, l2;
   int i=0, pos1, pos2, found;

   printf("Give first string (at most 19 characters)\n");
   while(i<19)
   {
       scanf("%c",&s1[i]);
       if(s1[i]=='\n')
          break;
       i++;
   }
   s1[i]='\0';
   l1=i;
   if(i==19)
     while(getchar()!='\n');

   i=0;
   printf("Give second string (at most 19 characters)\n");
   while(i<19)
   {
       scanf("%c",&s2[i]);
       if(s2[i]=='\n')
          break;
       i++;
   }
   s2[i]='\0';
   l2=i;
   if(i==19)
      while(getchar()!='\n');
   if(l2==0)
   {
      printf("nothing to search \n");
      return 0;
   }
   else
   {
       found=0;
       for(pos1=0; pos1<= l1 -l2; pos1++)
       {
          pos2=0;
          while(pos2 < l2 && s1[pos1 + pos2]==s2[pos2])
              pos2++;
```

```
            if(pos2 == l2)
            {
               found=1;
               break;
            }
         }
         if(found==1)
            printf("found substring at posn %d \n", pos1);
         else
            printf("substring not found \n");

         return 0;
      }
   }
```

This program can accept strings of length at most 19 and the strings may contain spaces. The following code segment reads all characters including spaces from the user, until a newline character is encountered or till 19 characters are read from the terminal, whichever is earlier.

```
while(i<19)
 {
     scanf("%c",&s1[i]);
     if(s1[i]=='\n')
        break;
     i++;
 }
```

After executing this loop, the value of `i` will be equal to the number of characters accepted into `s1` inside the loop (a maximum of 19 characters), except the newline character (if any). The newline character entered as `s1[i]` is replaced by '`\0`' to mark the end of the string `s1`. After this, the value of `i` is stored into *l*1, the variable which represents the length of `s1`. These tasks are done in the following two lines.

```
s1[i]='\0';
l1=i;
```

If we enter a string of length greater than 19, the first 19 characters are placed into the array `s1` and the character \0 is appended as `s1[19]`. The remaining characters entered from the terminal will be present in the input buffer. This content from the buffer has to be removed. Otherwise these characters will be read by the next `scanf` instruction in the program and the user will not get a chance to enter the second string. The cleaning of buffer is done using the instruction `while(getchar()!='\n');`. Here, `getchar()` reads a character from the input (or buffer) and the expression `getchar()!='\n'` will have value 1 only if the character read is a not a newline character. Note that, the `while` instruction is immediately followed by a semicolon (`;`). Hence, the `while` instruction is repeated until a newline character is read from the input (or buffer).

In a similar way, input of the second string `s2` is also handled using the following lines.

```
i=0;
printf("Give second string (at most 19 characters)\n");
while(i<19)
{
    scanf("%c",&s2[i]);
    if(s2[i]=='\n')
       break;
    i++;
}
s2[i]='\0';
l2=i;
if(i==19)
   while(getchar()!='\n');
```

The remaining part of the code is already explained earlier.

```
Give first string (at most 19 characters)
ababacba
Give second string (at most 19 characters)
abac
found substring at posn 2


Give first string (at most 19 characters)
ababacba
Give second string (at most 19 characters)
abd
substring not found
```

For the inputs "hellow how are you" and "low", the above program displays the following output.

```
Give first string (at most 19 characters)
hellow how are you
Give second string (at most 19 characters)
low
found substring at posn 3
```