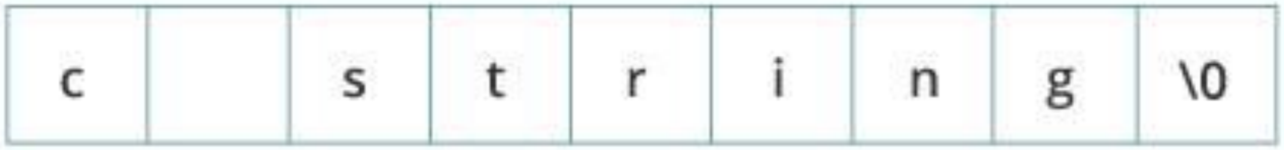
STRINGS

1.Introduction to Strings

- Sequence of character is known as Strings.
- A string is a null-terminated character array.
- This means that after the last character, a null character ('\o') is stored to signify the end of the character array.
- **For example**:   char c[] = "c string";



c		s	t	r	i	n	g	\0

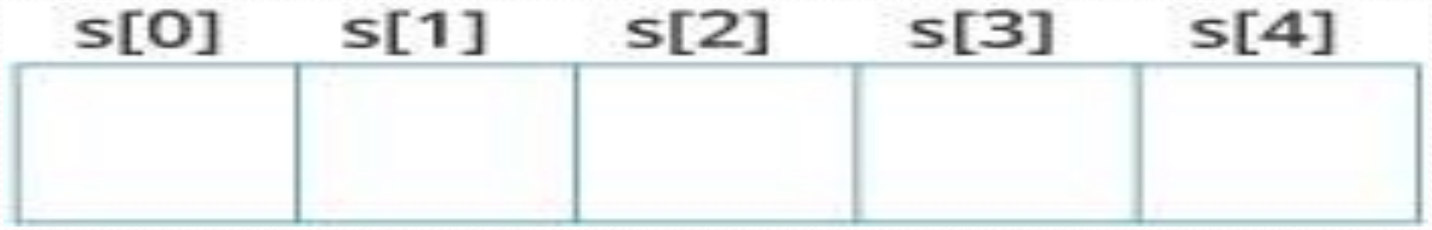
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \o at the end by default.

How to declare a string?

- Here's how you can declare strings:

  char str[size];

  char s[5];

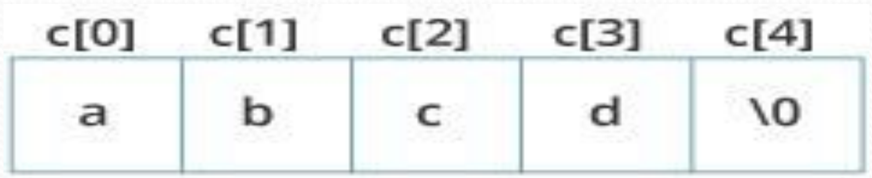


- Here, we have declared a string of 5 characters.
- A string can be declared as a **character array or with a string pointer.**

How to initialize strings?

- You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[5] = {'a','b','c','d','\o'};
char *c="abcd";
```



c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

- Let's take another example:
- char c[**5**] = "**abcde**";
- Here, we are trying to **assign 6 characters** (the last character is'\o') to a char array having 5 characters.
- **This is bad and you should never do this**.

Assigning Values to Strings

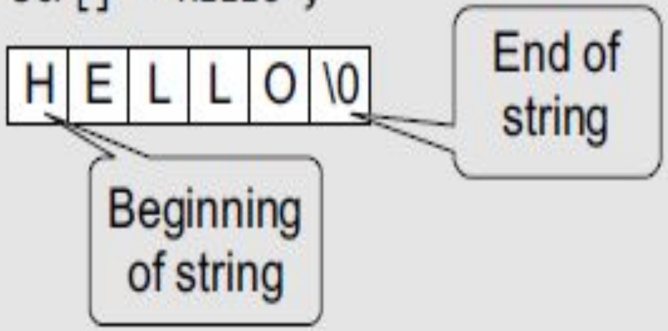
- Arrays and strings are **second-class** citizens in C; they do not support the assignment operator once it is declared.
- For example,

```
        char c[100];
c = "C programming"; // Error! array type is not
                    assignable.
```

- **Note: Use the strcpy() function to copy the string instead**

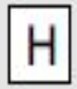
Difference between character storage and string storage

```
char str[] = "HELLO";
```

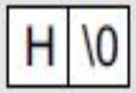


```
char ch = 'H';
```

Here H is a character not a string. The character H requires only one memory location.

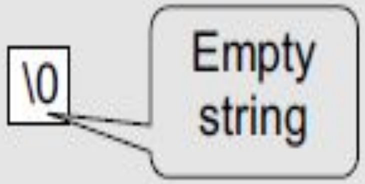


```
char str[] = "H";
```



Here H is a string not a character. The string H requires two memory locations. One to store the character H and another to store the null character.

```
char str[] = "";
```



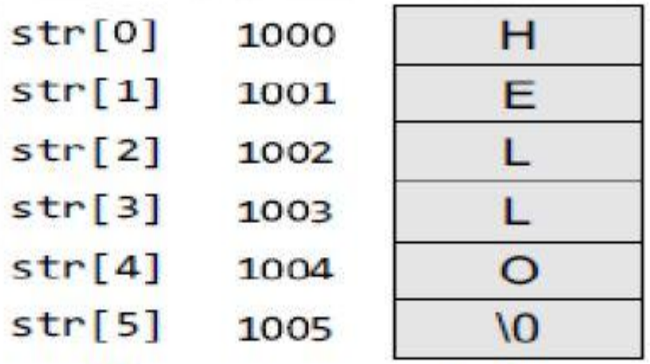
Although C permits empty string, it does not allow an empty character.

For example if we write,

char **str[]** = "HELLO";

We are declaring a character array with 5 characters namely, H, E, L, L and O. Besides, a null character ('\0') is stored at the end of the string. So, the internal representation of the string becomes- HELLO'\0'. Note that to store a string of length 5, we need 5 + 1 locations (1 extra for the null character).

The name of the character array (or the string) is a pointer to the beginning of the string.



str[0]	1000	H
str[1]	1001	E
str[2]	1002	L
str[3]	1003	L
str[4]	1004	O
str[5]	1005	\0

Memory representation of a character array

We can also declare a string with size much larger than the number of elements that are initialized.

For example, consider the statement below.

char **str [10]** = "HELLO";

In such cases, the compiler creates an array of size 10; stores "HELLO" in it and finally terminates the string with a null character. Rest of the elements in the array are automatically initialized to NULL.

Now consider the following statements:

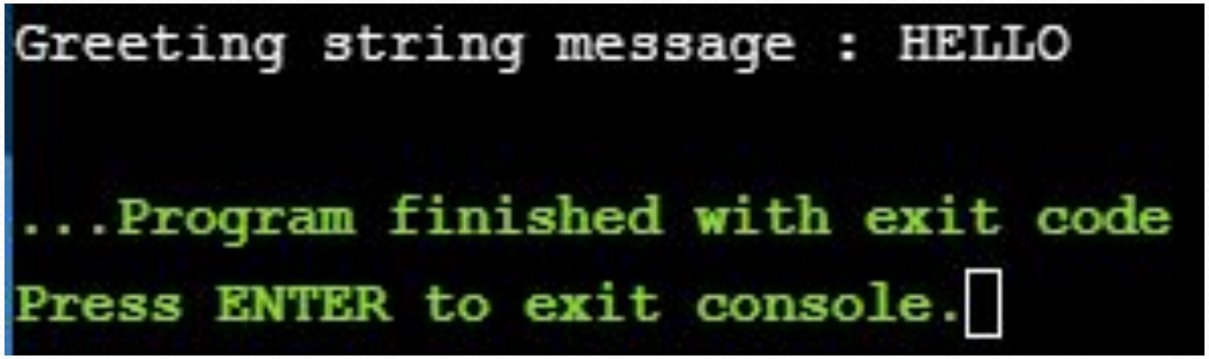
char str[3]; str = "HELLO";

The above initialization statement is illegal in C and would generate a compile-time error because of two reasons. First, the array is initialized with more elements than it can store. Second, initialization cannot be separated from declaration.

Note: When allocating memory space for a string, reserve space to hold the null character also.

Let us try to print above mentioned string:

```
#include <stdio.h>
#include <conio.h>
int main()
{
char str[10]={'H','E','L','L','O','\0'};
printf("Greeting string message : %s", str);
return 0;
}
```



```
Greeting string message : HELLO

...Program finished with exit code
Press ENTER to exit console.
```

Note: %s is used to print the string in C

2.Reading and Writing a string

- **READING STRINGS**

If we declare a string by writing

char str[100];

Then **str** can be read from the user by using three ways

- use **scanf** function
- using **gets**() function
- using **getchar**(), **getch**()**or getche**() function repeatedly

- The string can be read using **scanf()** by writing

  scanf("%s", **str);**

Although the syntax of using **scanf()** function is well known and easy to use, the main pitfall of using this function is that the function terminates as soon as it finds a blank space (white space, newline, tab, etc.)..

For example, if the user enters Hello World, then the **str** will contain only **Hello**. This is because the moment a blank space is encountered, the string is terminated by the scanf() function.

You may also specify a field width to indicate the maximum number of characters that can be read. Remember that extra characters are left unconsumed in the input buffer.

- Unlike int, float, and char values,

%s format **does not** require the ampersand(**&**) before the variable **str**.

- Note: Using & operand with a string variable in the scanf statement generates an **error**.

```
scanf("%s", &str);
```

The string can be read by writing

gets(str);

- gets() is a simple function that overcomes the **drawbacks** of the **scanf()** function.
- gets() takes the starting address of the string which will hold the input.
- The string inputted using gets() is automatically terminated with a null character.

- **Note: that in this method, you have to deliberately append the string with a null character. The other two functions automatically do this**

The string can also be read by calling **the getchar**() repeatedly to read a sequence of single characters (unless a terminating character is entered) and simultaneously storing it in a character array.

```
i=0;
getchar(ch);
while(ch!='*’)
{           str[i] = ch;
             i++;
             getchar(ch);
} str[i] ='\0
```

WRITING STRINGS

The string can be displayed on screen using three ways

- use **printf()** function
- using **puts()** function
- using **putchar()**function repeatedly

The string can be displayed using printf() by writing

```
printf("%s", str);
```

We use the format specifier %s to output a string. Observe carefully that there is no'&' character used with the string variable. We may also use width and precision specifications along with %s. The width specifies the minimum output field width. If the string is short, the extra space is either left padded or right padded. A negative width left pads short string rather than the default right justification. The precision specifies the maximum number of characters to be displayed, after which the string is truncated. For example,

```
printf ("%5.3s", str);
```

The above statement would print only the first three characters in a total field of five characters. Also these characters would be right justified in the allocated width. To make the string left justified, we must use a minus sign. For example,

```
printf ("%–5.3s", str);
```

- **The string can be displayed by writing**

  puts(str);
- puts() is a simple function that overcomes the drawbacks of the printf() function.

- Note: When the field width is less than the length of the string, the entire string will be printed, if the number of characters to be printed is specified as zero, then nothing is printed on the screen.

The string can also be written by calling the **putchar**() repeatedly to print a sequence of single characters

```
i=0;
while(str[i] !='\0*)
{
putchar(str[i]);
i++;
}
```

Reading A Line Of Text

gets() and **puts()** are two string functions to take string input from user and display string respectively

```
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;
}
```

Note: Though, gets() and puts() function handle strings, both these functions are defined in "stdio.h" header file.

SUPPRESSING INPUT

- scanf() can be used to read a field without assigning it to any variable. This is done by preceding that field's format code with a

For example, given:

scanf("%d*c%d", &hr, &min);

- The time can be read as 9:05 as a pair.
- Here the colon would be read but not assigned to anything.

Using a Scanset

- The ANSI standard added the new scanset feature to the C language.
- A scanset is used to define a **set of characters** which may be **read and assigned** to the corresponding string.
- A scanset is defined by placing the characters inside square brackets prefixed with a **%**

```
intmain()
{
charstr[10];
printf("\nEnter string:");
scanf("%[aeiou]",str);
printf("The string is : %s",str);
Return 0;
}
```

- The code will stop accepting character as soon as the user will enter a character that is not a vowel.
- However, if the first character in the set is a ^ (caret symbol), then scanf() will accept any character that is not defined by the scanset.
- For example, if you write

```
scanf("%[^aeiou]",str);
```

3.String operations

(without using built–in string functions)

- In this section, we will learn about different operations that can be performed on strings without using built in functions which includes :

i) Length
ii) Compare
iii) Concatenate
iv) Copy
v) Reverse
vi) Substring
vii) Insertion
viii) Indexing
ix) Deletion
x) Replacement

i) Length - Finding Length of a String

- The number of characters in the string constitutes the length of the string.
- For example, LENGTH("C PROGRAMMING IS FUN") will return 20.
- LENGTH('o') = o and LENGTH("") = o because both the strings does not contain any character.

Algorithm to calculate the length of a string

```
Step 1: [INITIALIZE] SET I = 0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:        SET I = I + 1
        [END OF LOOP]
Step 4: SET LENGTH = I
Step 5: END
```

I is used as an index for traversing string STR.

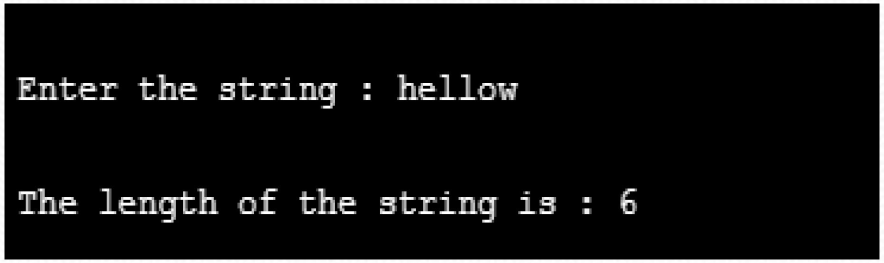
To traverse each and every character of STR, we increment the value of I.

Once we encounter the null character, the control jumps out of the while loop and the length is initialized with the value of I.

Write a program to find the length of a string.

```
#include <stdio.h>
#include <conio.h>
int main()
{
char str[100], i= 0, length=0;
printf("\n Enter the string : ");
gets(str);
/*while(str[i] != '\0') // using while loop
i++;
length = i;*/
for (i=0;str[i]!='\0';i++)
length = length+1;
printf("\n The length of the string is : %d", length);
return 0;
}
```

OUTPUT



```
Enter the string : hellow

The length of the string is : 6
```

ii) Compare

- If S1 and S2 are two strings, then comparing the two strings will give either of the following results:

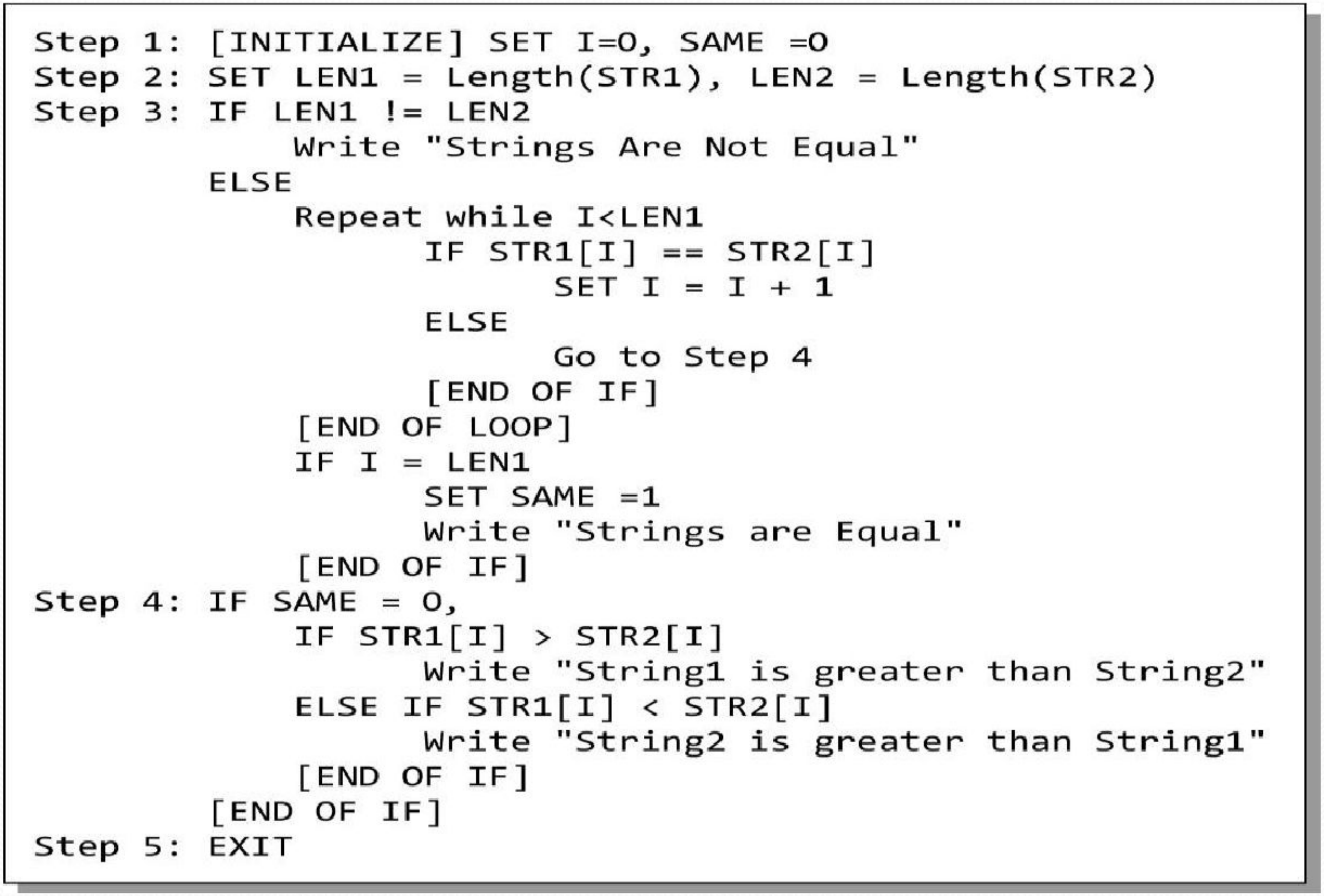
  S1 and S2 are equal

  S1>S2, when in dictionary order, S1 will come after S2

  S1<S2, when in dictionary order, S1 precedes S2
- To compare the two strings, each and every character is compared from both the strings.
- If all the characters are the **same,** then the two strings are said to be **equal**.

- In this algorithm, we first check whether the two strings are of the same length. If not, then there is no point in moving ahead, as it straight away means that the two strings are **not** the same.
- However, if the two strings are of the same length, then we compare character by character to check if all the characters are same.
- If yes, then the variable SAME is set to 1. Else, if SAME = 0, then we check which string precedes the other in the dictionary order and print the corresponding message.

Algorithm



```
Step 1: [INITIALIZE] SET I=0, SAME =0
Step 2: SET LEN1 = Length(STR1), LEN2 = Length(STR2)
Step 3: IF LEN1 != LEN2
            Write "Strings Are Not Equal"
        ELSE
            Repeat while I<LEN1
                  IF STR1[I] == STR2[I]
                        SET I = I + 1
                  ELSE
                        Go to Step 4
                  [END OF IF]
            [END OF LOOP]
            IF I = LEN1
                  SET SAME =1
                  Write "Strings are Equal"
            [END OF IF]
Step 4: IF SAME = 0,
            IF STR1[I] > STR2[I]
                  Write "String1 is greater than String2"
            ELSE IF STR1[I] < STR2[I]
                  Write "String2 is greater than String1"
            [END OF IF]
        [END OF IF]
Step 5: EXIT
```

Write a program to compare two strings

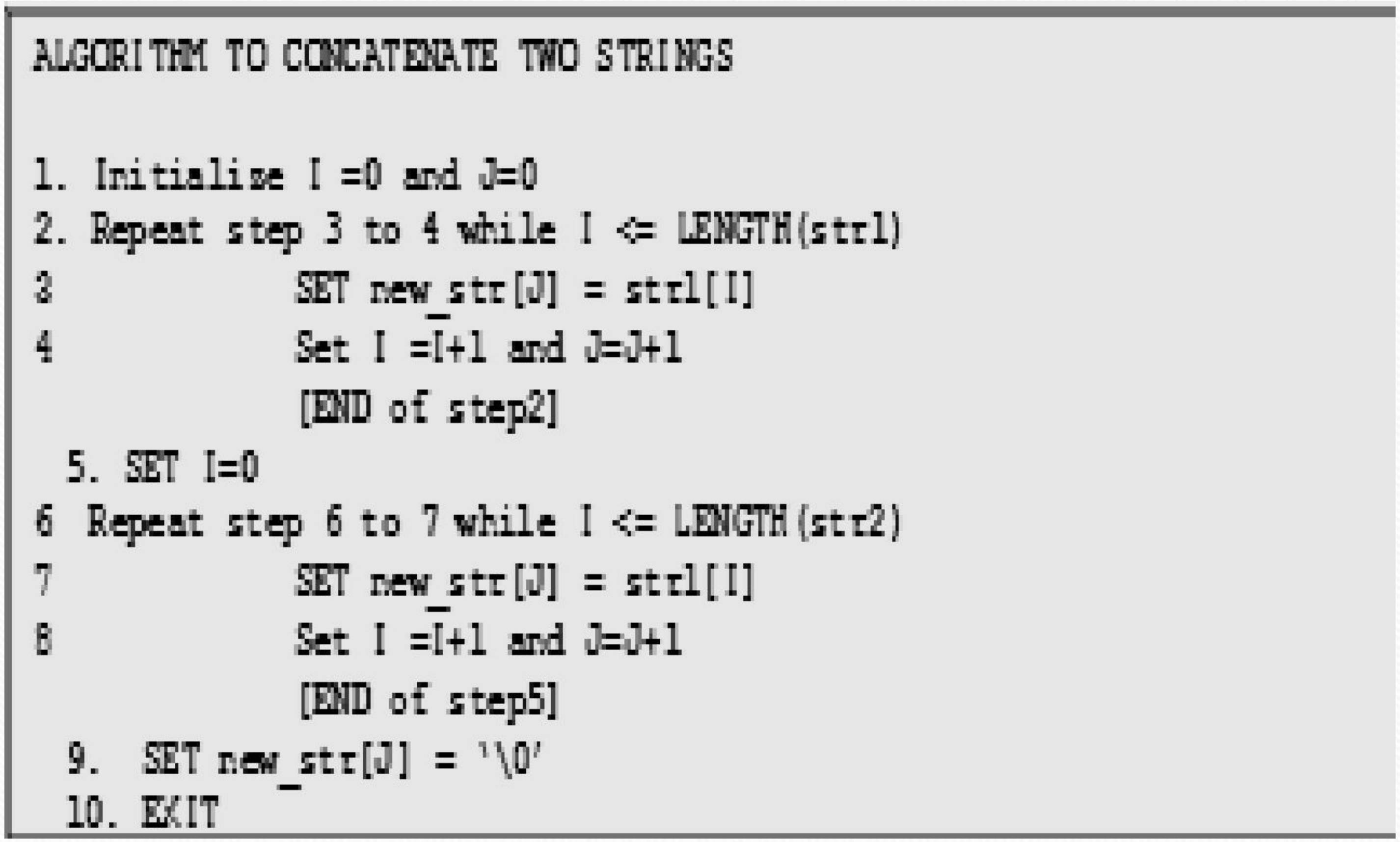
```
#include <stdio.h>
 #include <conio.h>
#include <string.h>
 intmain()
{
char str1[50], str2[50];
int i=0, len1=0, len2=0, same=0;
 clrscr();
printf("\n Enter the first string : ");
 gets(str1);
printf("\n Enter the second string :);
gets(str2);
len1 = strlen(str1);
len2 = strlen(str2);
if(len1 == len2)
{
while(i<len1)
{
        if(str1[i] == str2[i])
      i++;
else
break;
}
if(i==len1)
{
same=1;
printf("\n The two strings are  equal");
}}
 if(len1!=len2)
printf("\n The two strings are not equal");
 if(same == 0){
if(str1[i]>str2[i])
printf("\n String 1 is greater than string 2");
else if(str1[i]<str2[i])
printf("\n String 2 is greater than string 1");
}
getch();
return 0;
}
```

OUTPUT:

```
Enter the first string : Hello
Enter the second string : Hello
The two strings are equal
```

iii) Concatenate

- CONCATENATING TWO STRINGS TO FORM A NEW STRING
- IF S1 and S2 are two strings, then concatenation operation produces a string which contains characters of S1 followed by the characters of S2.

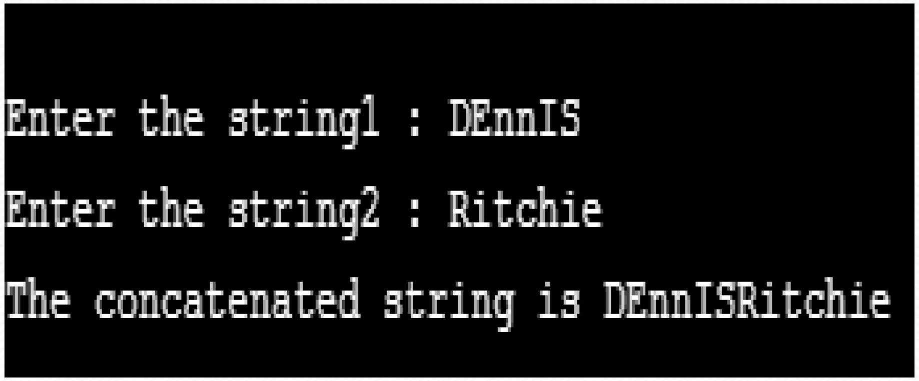


```
ALGORITHM TO CONCATENATE TWO STRINGS

1. Initialize I =0 and J=0
2. Repeat step 3 to 4 while I <= LENGTH(str1)
3            SET new_str[J] = str1[I]
4            Set I =I+1 and J=J+1
             [END of step2]
 5. SET I=0
6  Repeat step 6 to 7 while I <= LENGTH(str2)
7            SET new_str[J] = str1[I]
8            Set I =I+1 and J=J+1
             [END of step5]
 9.  SET new_str[J] = '\0'
 10. EXIT
```

PROGRAM:

```
#include <stdio.h>
intmain()
{
inti,j=0,len=0;
char str1[100], str2[100];
printf("\nEnterthe string1 : ");
gets(str1);
printf("Enter the string2 : ");
gets(str2);
for (i=0;str1[i]!='\0';i++)
len=len+1; // length of the first string
j=len;
for (i=0;str2[i]!='\0';i++){
str1[j]=str2[i]; // 2nd string copied to 1st string //from jthposition
j=j+1;}
str1[j]='\0';
printf("The concatenated string is %s",str1);
}
```

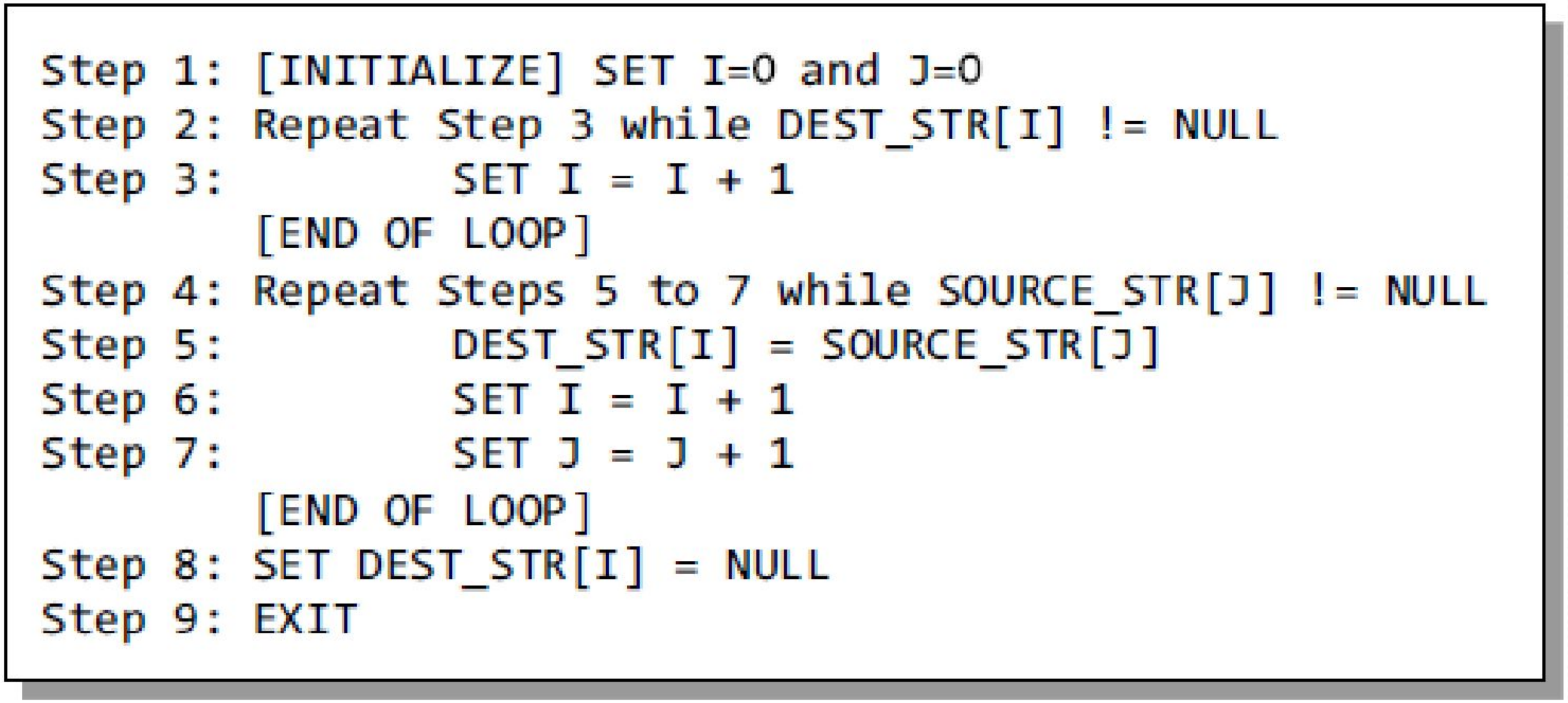
OUTPUT



```
Enter the string1 : DEnnIS
Enter the string2 : Ritchie
The concatenated string is DEnnISRitchie
```

- concatenate or **append** have same implementation in C
- Appending one string to another string involves copying the contents of the source string at the end of the destination string.

- **algorithm** that appends two strings



```
Step 1: [INITIALIZE] SET I=0 and J=0
Step 2: Repeat Step 3 while DEST_STR[I] != NULL
Step 3:         SET I = I + 1
        [END OF LOOP]
Step 4: Repeat Steps 5 to 7 while SOURCE_STR[J] != NULL
Step 5:         DEST_STR[I] = SOURCE_STR[J]
Step 6:         SET I = I + 1
Step 7:         SET J = J + 1
        [END OF LOOP]
Step 8: SET DEST_STR[I] = NULL
Step 9: EXIT
```

Write a program to append a string to another string.

```
#include <stdio.h>
#include <conio.h>
int main()
{
char Dest_Str[100], Source_Str[50];
inti=0, j=0;
printf("\n Enter the source string : ");
gets(Source_Str);
printf("\n Enter the destination string : ");
gets(Dest_Str);
while(Dest_Str[i] != '\0')
i++;
while(Source_Str[j] != '\0')
{       Dest_Str[i] = Source_Str[j];
                i++;
                 j++;
}
Dest_Str[i] = '\0';
printf("\n After appending, the destination string is : "); puts(Dest_Str);
return 0;
}
```

OUTPUT

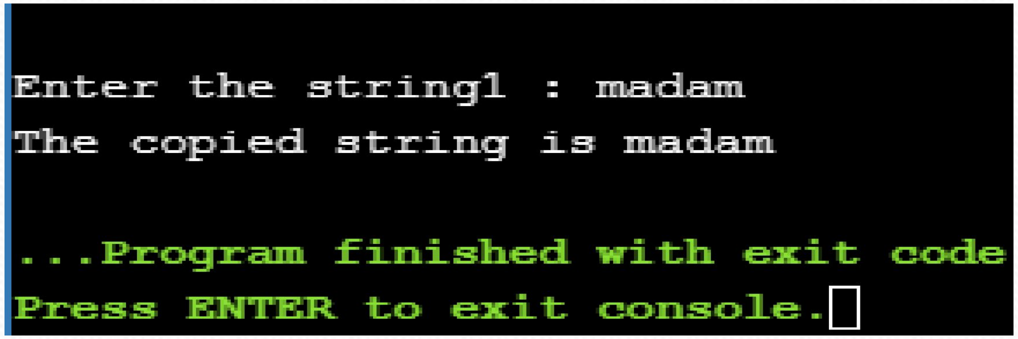
```
Enter the source string : How are you?
Enter the destination string : Hello,
After appending, the destination string is :
Hello,How are you?
```

iv) Copy

- Copying the contents of one string to another string.

```
#include <stdio.h>
int main()
{
int i;
char str1[100], str2[100];
printf("\n Enter the string1 : ");
gets(str1);
for (i=0;str1[i]!='\0';i++)
str2[i]=str1[i];
str2[i]='\0';
printf("The copied string is %s",str2);
return 0;
}
```

OUTPUT

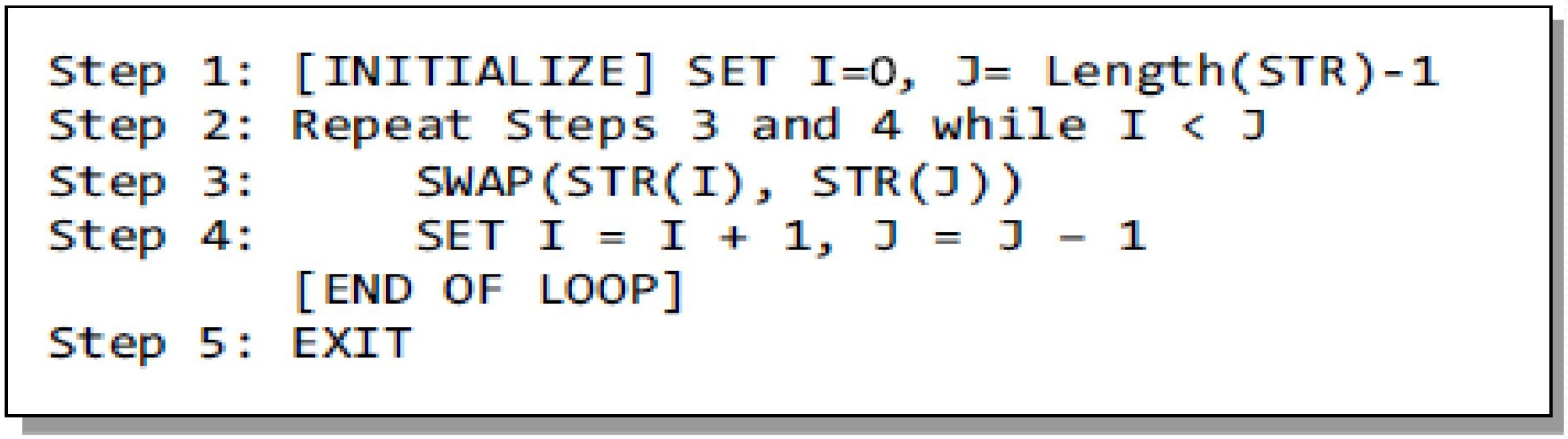


```

Enter the string1 : madam
The copied string is madam

...Program finished with exit code
Press ENTER to exit console.
```

v) Reverse

- If S1 = "HELLO", then reverse of S1 = "OLLEH".
- To reverse a string, we just need **to swap** the first character with the last, second character with the second last character, and so on.

Algorithm

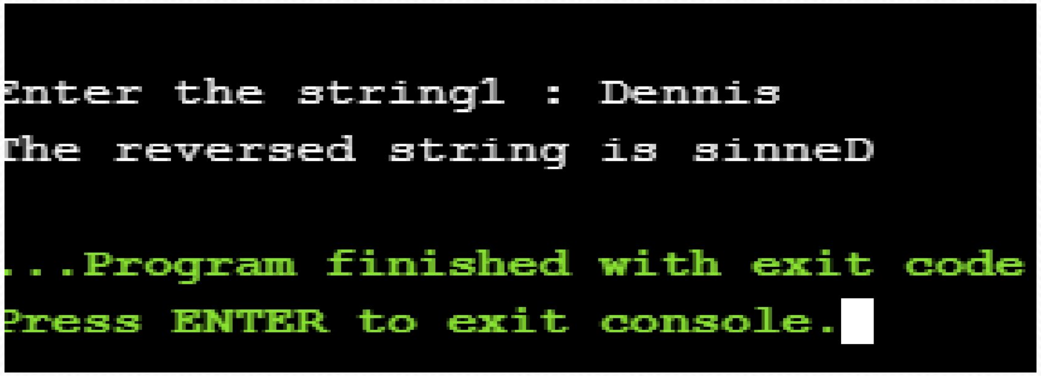


```
Step 1: [INITIALIZE] SET I=0, J= Length(STR)-1
Step 2: Repeat Steps 3 and 4 while I < J
Step 3:         SWAP(STR(I), STR(J))
Step 4:         SET I = I + 1, J = J - 1
        [END OF LOOP]
Step 5: EXIT
```

Write a program to reverse a given string.

```
#include <stdio.h>
intmain()
{
int i, j=0,len=0;
char str1[100], rev[100];
printf("\nEnterthe string1 : ");
gets(str1);
for (i=0;str1[i]!='\0';i++)
len=len+1;
for (i=len-1;i>=0;i--)
{
rev[j]=str1[i];
j=j+1;
}
rev[j]='\0';
printf("The reversed string is %s",rev);
return 0;
}
```

OUTPUT



```
Enter the string1 : Dennis
The reversed string is sinneD

...Program finished with exit code
Press ENTER to exit console.
```

vi) Substring

To extract a substring from a given string, we need the following three parameters:

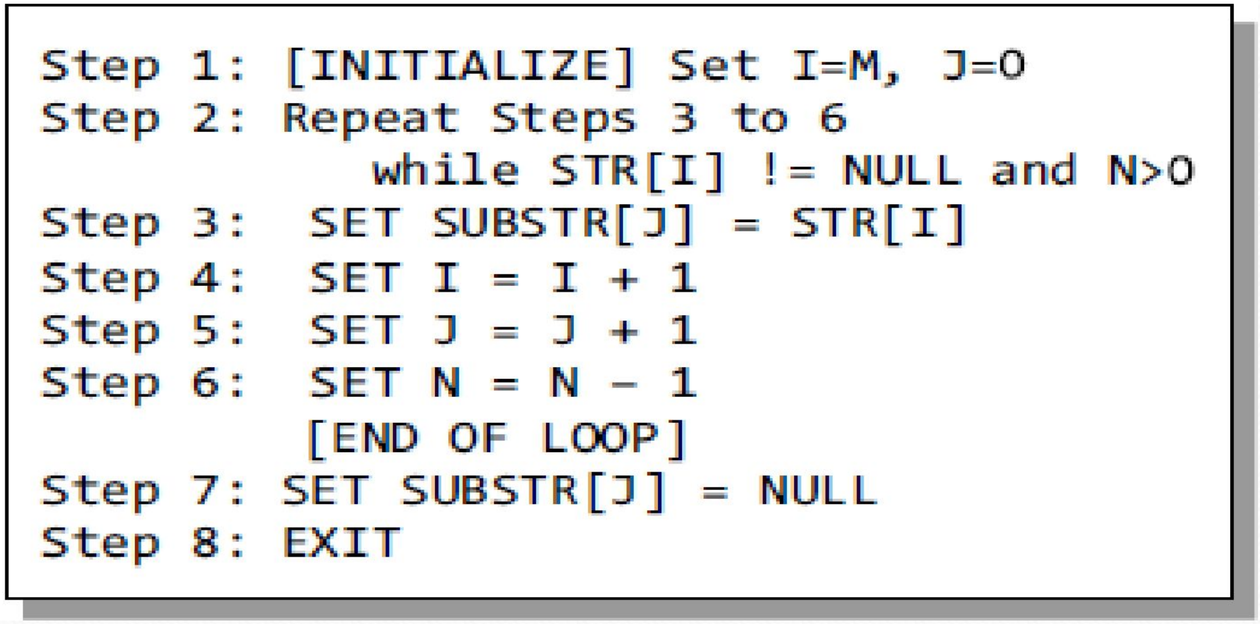
- the main string,
- the position of the first character of the substring in the given string, and
- the maximum number of characters/length of the substring.

For example, if we have a string

- str[] = "Welcome to the world of programming";
- Then, SUBSTRING(str, 15, 5) = world

Algorithm

- we initialize a loop counter I to M, that is, the position from which the characters have to be copied.
- Steps 3 to 6 are repeated until N characters have been copied.
- With every character copied, we decrement the value of N.
- The characters of the string are copied into another string called the SUBSTR.
- At the end, a null character is appended to SUBSTR to terminate the string.



```
Step 1: [INITIALIZE] Set I=M, J=0
Step 2: Repeat Steps 3 to 6
           while STR[I] != NULL and N>0
Step 3:  SET SUBSTR[J] = STR[I]
Step 4:  SET I = I + 1
Step 5:  SET J = J + 1
Step 6:  SET N = N - 1
         [END OF LOOP]
Step 7: SET SUBSTR[J] = NULL
Step 8: EXIT
```

Write a program to extract a substring from the middle of a given string.

```
#include <stdio.h>
#include <conio.h>
intmain()
{
char str[100], substr[100];
inti=0, j=0, n, m;
printf("\n Enter the main string : ");
gets(str);
printf("\n Enter the position from which to start the substring: ");
scanf("%d", &m);
printf("\n Enter the length of the substring: ");
scanf("%d", &n);
i=m;
while(str[i] != '\0' && n>0)
{
substr[j] = str[i];
i++; j++; n--;
}
substr[j] = '\0';
printf("\n The substring is : "); puts(substr);
return 0;
}
```

OUTPUT

```
Enter the main string : Hi there
Enter the position from which to start the
substring: 1
Enter the length of the substring: 4
The substring is : i th
```

vii) Indexing –program

Pattern Matching

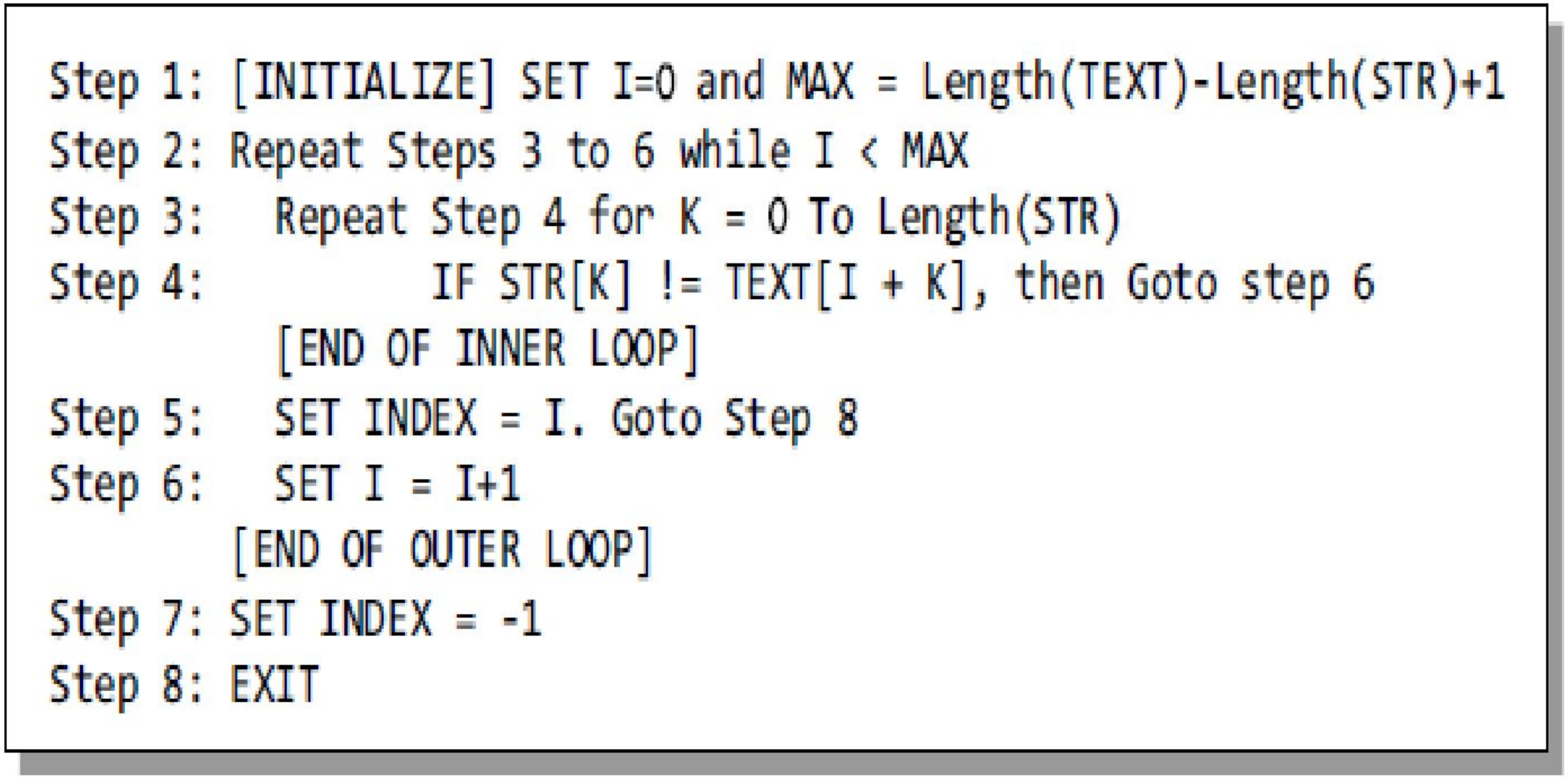
- This operation returns **the position** in the string where the string pattern first occurs.

For example,

- INDEX("Welcome to the world of programming", "world") = 15
- However, if the pattern does not exist in the string,

  the INDEX function returns 0.

Algorithm to find the index of the first occurrence of a string within a given text



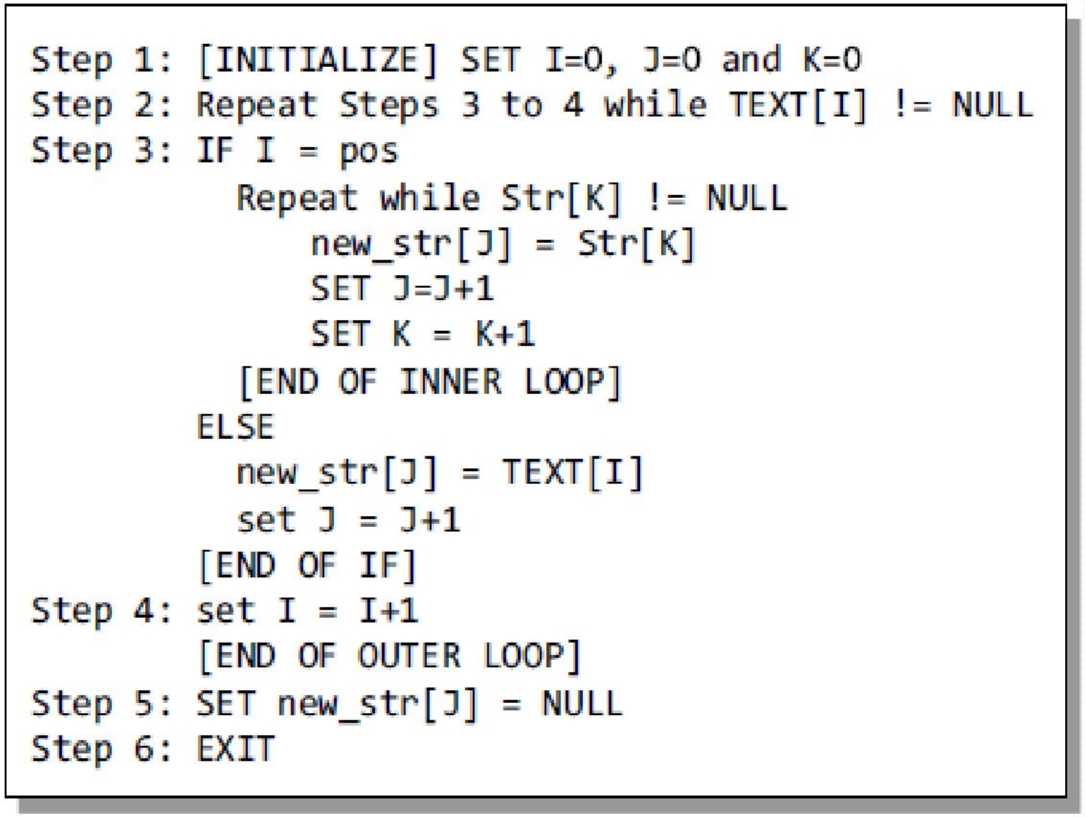
```
Step 1: [INITIALIZE] SET I=0 and MAX = Length(TEXT)-Length(STR)+1
Step 2: Repeat Steps 3 to 6 while I < MAX
Step 3:   Repeat Step 4 for K = 0 To Length(STR)
Step 4:            IF STR[K] != TEXT[I + K], then Goto step 6
          [END OF INNER LOOP]
Step 5:   SET INDEX = I. Goto Step 8
Step 6:   SET I = I+1
        [END OF OUTER LOOP]
Step 7: SET INDEX = -1
Step 8: EXIT
```

- MAX is initialized to **length(TEXT) –Length(STR) + 1**.
- For example, if a text contains 'Welcome To Programming' and the string contains 'World', in the main text, we will look for at the most 22 –5 + 1 = 18 characters because after that there is no scope left for the string to be present in the text.
- Steps 3 to 6 are repeated until each and every character of the text has been checked for the occurrence of the string within it. In the inner loop in Step 3, we check the n characters of string with the n characters of text to find if the characters are same.
- If it is not the case, then we move to Step 6, where I is incremented. If the string is found, then the index is initialized with I, else it is set to –1.
- For example, if TEXT = WELCOME TO THE WORLD
- STRING = COME
- In the first pass of the inner loop, we will compare COME with WELC character by character. As W and C do not match, the control will move to Step 6 and then ELCO will be compared with COME. In the fourth pass, COME will be compared with COME.
- We will be using the programming code of pattern matching operation in the operations that follow.

viii) Insertion

- The insertion operation inserts a string S in the main text T at the **kth** position.
- The general syntax of this operation is **INSERT(text, position, string)**.
- For example, INSERT("XYZXYZ", 3, "AAA") = "XYZAAAXYZ“
- **Algorithm**

- first initializes the indices into the string to zero.
- From Steps 3 to 5, the contents of NEW_STR are built.
- If I is exactly equal to the position at which the substring has to be inserted, then the inner loop copies the contents of the substring into NEW_STR.
- Otherwise, the contents of the text are copied into it.



```
Step 1: [INITIALIZE] SET I=0, J=0 and K=0
Step 2: Repeat Steps 3 to 4 while TEXT[I] != NULL
Step 3: IF I = pos
          Repeat while Str[K] != NULL
              new_str[J] = Str[K]
              SET J=J+1
              SET K = K+1
          [END OF INNER LOOP]
        ELSE
          new_str[J] = TEXT[I]
          set J = J+1
        [END OF IF]
Step 4: set I = I+1
        [END OF OUTER LOOP]
Step 5: SET new_str[J] = NULL
Step 6: EXIT
```

Write a program to insert a string in the main text

```
#include <stdio.h>
intmain() {
char text[100], str[20], ins_text[100];
inti=0, j=0, k=0,pos;
printf("\n Enter the main text : ");
gets(text);
printf("\n Enter the string to be inserted : ");
gets(str);
printf("\n Enter the place at which the string has to
be inserted: ");
scanf("%d", &pos);
while(text[i] != '\0‘)
{
if(i==pos){
while(str[k] != '\0‘){
ins_text[j] = str[k];
j++;
k++;
}}
else
{
ins_text[j] = text[i];
j++;
}
i++;
}
ins_text[j] = '\0';
printf("\n The new string is : ");
puts(ins_text);
return 0;
}
```

OUTPUT

```
Enter the main text : newsman
Enter the string to be inserted : paper
Enter the place at which the string has to be
inserted: 4
The new string is: newspaperman
```

ix) Deletion

Deleting a Substring from the Main String

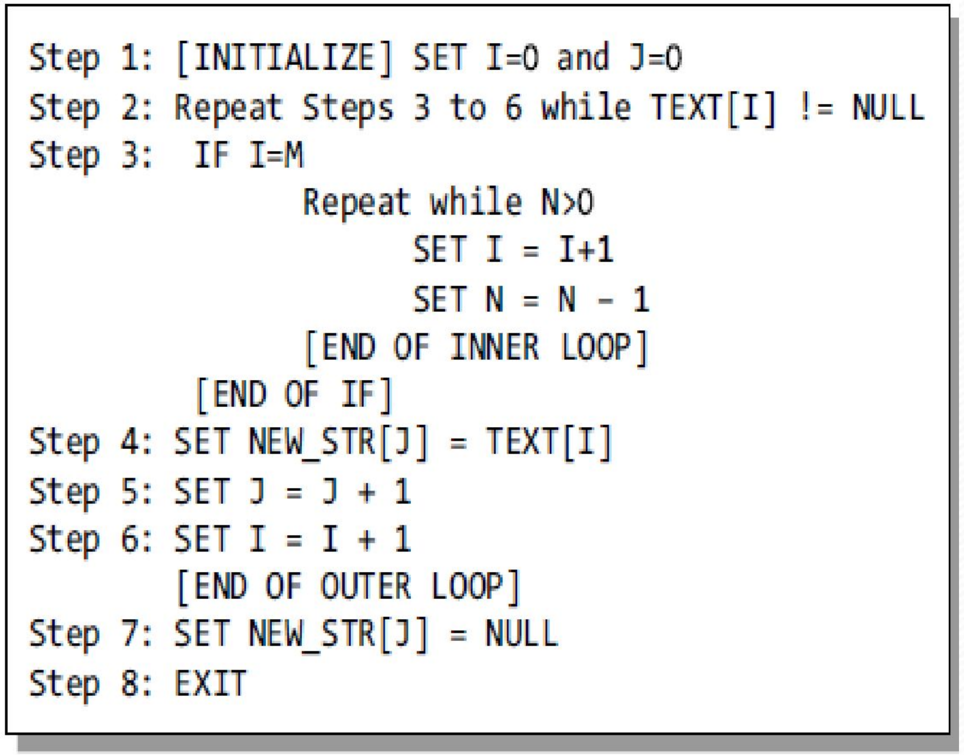
The deletion operation deletes a substring from a given text.

We can write it as **DELETE(text, position, length)**

For example, **DELETE("ABCDXXXABCD", 4, 3)** = "ABCDABCD”

Algorithm:

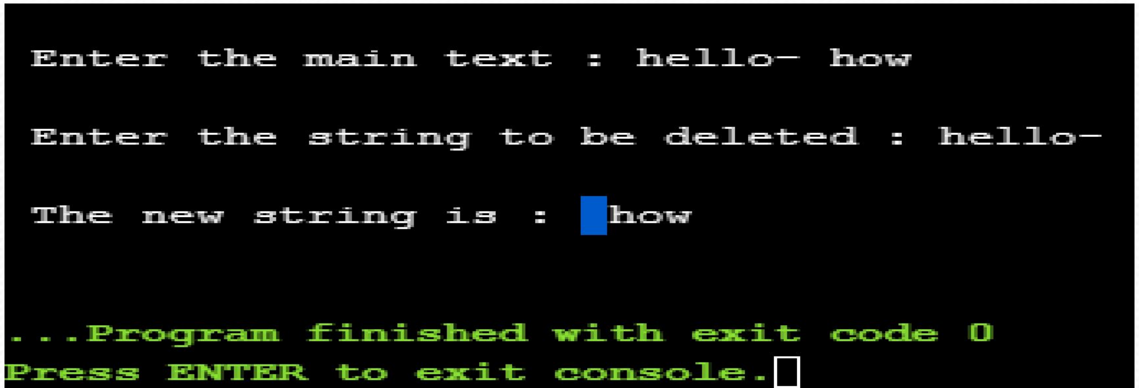
- we first initialize the indices to zero.
- Steps 3 to 6 are repeated until all the characters of the text are scanned.
- If I is exactly equal to M (the position from which deletion has to be done), then the index of the text is incremented and N is decremented.
- N is the number of characters that have to be deleted starting from position M.
- However, if I is not equal to M, then the characters of the text are simply copied into the NEW_STR.



```
Step 1: [INITIALIZE] SET I=0 and J=0
Step 2: Repeat Steps 3 to 6 while TEXT[I] != NULL
Step 3:  IF I=M
              Repeat while N>0
                    SET I = I+1
                    SET N = N - 1
              [END OF INNER LOOP]
         [END OF IF]
Step 4: SET NEW_STR[J] = TEXT[I]
Step 5: SET J = J + 1
Step 6: SET I = I + 1
        [END OF OUTER LOOP]
Step 7: SET NEW_STR[J] = NULL
Step 8: EXIT
```

Write a program to delete a substring from a text.

```
#include <stdio.h>
intmain() {
char text[200], str[20], new_text[200];
int i=0, j=0, found=0, k, n=0, copy_loop=0;
printf("\n Enter the main text : ");
gets(text);
printf("\n Enter the string to be deleted : ");
gets(str);
while(text[i]!='\0'){
j=0, found=0, k=i;
while(text[k]==str[j] && str[j]!='\0')
{
k++;
j++;
}
if(str[j]=='\0')
copy_loop=k;
new_text[n] = text[copy_loop];
i++;
copy_loop++;
n++;
}
new_text[n]='\0';
printf("\n The new string is : ");
puts(new_text);
getch();
return 0;}
}
```



```
Enter the main text : hello- how

Enter the string to be deleted : hello-

The new string is :  how


...Program finished with exit code 0
Press ENTER to exit console.
```

x) Replacement

Replacing a Pattern with Another Pattern in a String

The replacement operation is used to replace the pattern P1 by another pattern P2 .

This is done by writing **REPLACE(text, pattern , pattern**).

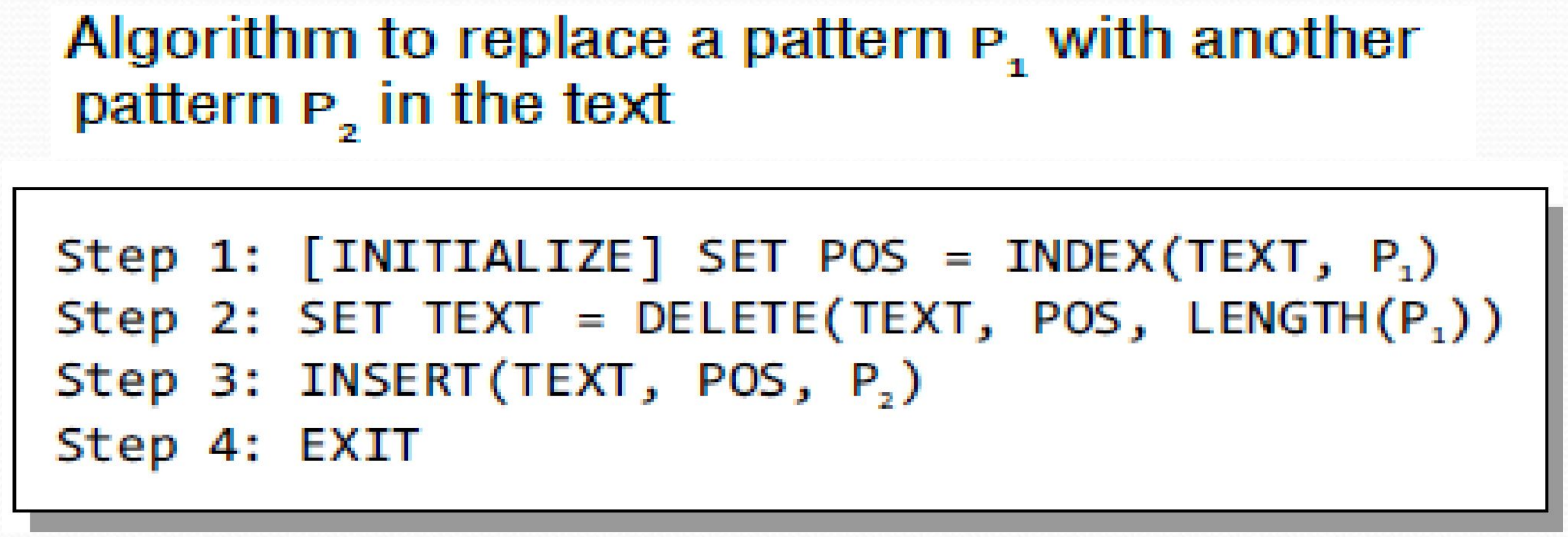
For example,

("AAABBBCCC", "BBB", "X") = AAAXCCC

("AAABBBCCC", "X", "YYY")= AAABBBCC

In the second example, there is no change as X does not appear in the text.

- The algorithm is very simple, where
- we first find the position **POS**, at which the pattern occurs in the text, then delete the existing pattern from that position and
- insert a new pattern there.



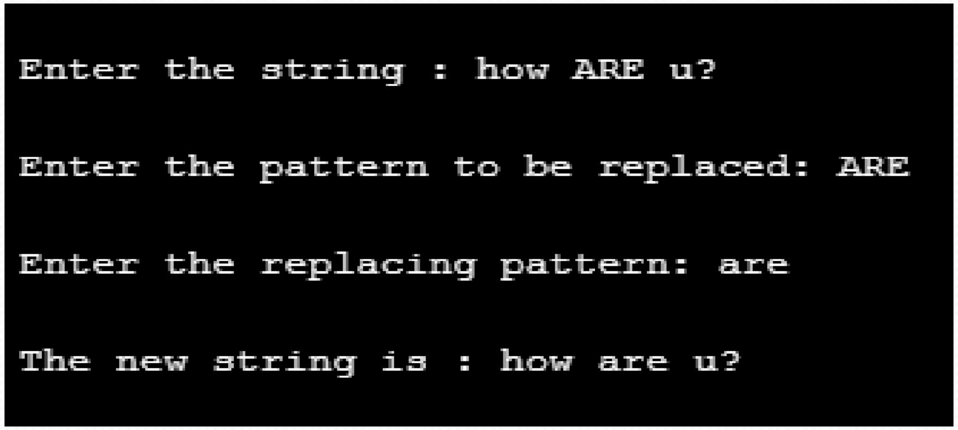
Algorithm to replace a pattern P_1 with another pattern P_2 in the text

```
Step 1: [INITIALIZE] SET POS = INDEX(TEXT, P₁)
Step 2: SET TEXT = DELETE(TEXT, POS, LENGTH(P₁))
Step 3: INSERT(TEXT, POS, P₂)
Step 4: EXIT
```

Write a program to replace a pattern with another pattern in the text.

```
#include <stdio.h>
intmain() {
char str[200], pat[20], new_str[200], rep_pat[100];
inti=0, j=0, k, n=0, copy_loop=0, rep_index=0;
printf("\n Enter the string : ");
gets(str);
printf("\n Enter the pattern to be replaced: ");
gets(pat);
printf("\n Enter the replacing pattern: ");
gets(rep_pat);
while(str[i]!='\0'){
j=0,k=i;
while(str[k]==pat[j] && pat[j]!='\0'){
k++;
j++;}
if(pat[j]=='\0'){
copy_loop=k;
while(rep_pat[rep_index] !='\0'){
new_str[n] = rep_pat[rep_index];
rep_index++;
n++;
}}
new_str[n] = str[copy_loop];
i++;
copy_loop++;
n++;}
new_str[n]='\0';
printf("\n The new string is : ");
puts(new_str);
return 0;
}
```

OUTPUT



```
Enter the string : how ARE u?

Enter the pattern to be replaced: ARE

Enter the replacing pattern: are

The new string is : how are u?
```

String operations

(using built-in string functions)

- In this section, we will learn about different operations that can be performed on strings using built in functions.
- C provides **string manipulating functions** in the "string.h" library.
- String in C – Library Functions

Function	Purpose	Example
strlen	Returns the number of characters in a string	strlen("Hi") returns 2.
strlwr	Converts string to all lowercase	strlwr("Hi") returns hi.
strupr	Converts s to all uppercase	strupr("Hi");
strrev	Reverses all characters in s1 (except for the terminating null)	strrev(s1, "more");
strtok	Breaks a string into tokens by delimiters.	strtok("Hi, Chao", " ,");
strcpy	Makes a copy of a string	strcpy(s1, "Hi");
Strncpy	Copy the specified number of characters	strncpy(s1, "SVN",2);
strcat	Appends a string to the end of another string	strcat(s1, "more");
Strncat	Appends a string to the end of another string up to n characters	strncat(s1, "more",2);

Function	Purpose	Example
strcmp	Compare two strings alphabetically	strcmp(s1, "Hu");
Strncmp	Compare two string upto given n character	strncmp("mo", "more",2);
Stricmp	Compare two strings alphabetically without case sensitivity.	stricmp("hu", "Hu");
strchr()	Find first occurrence of a given character in the string	strchr(str1,c); Where c is the character variable
strrchr()	Find the last occurrence of a given character in the string	strrchr(str1,c)
strstr()	Finds the first occurrence of a given string in another string	strstr(str1,str2); Where str2 is the string to be searched in str1
strset()	sets all characters of a string to a given character	strset(str1,c);
strnset()	Sets first character of a string to a given character	Strnset(str1,c,n)

Length –reverse –upper case & lower case

Length of the string

- The number of characters in the string constitutes the length of the string.

  For example, **LENGTH("C PROGRAMMING IS FUN")** will return **20**.

- Note that even blank spaces are counted as characters in the string. LENGTH('0') = 0 and LENGTH('') = 0 because both the strings does not contain any character.

- **strlen()** function in C gives the **length of the given string**.
- **Syntax**

  size_t strlen(const char * str);
- strlen() function counts the number of characters in a given string and returns the integer value.
- Its tops counting the character when null character is found. Because, null character indicates the end of the string in C.

Reversing a String

- If S1 = "HELLO", then reverse of S1 = "OLLEH".
- To reverse a string, we just need to swap the first character with the last, second character with the second last character, and so on.
- strrev( ) function reverses a given string in C language.

Syntax

```
char *strrev(char*string);
```

- strrev( ) function is non-standard function which may not available in standard library inC.

Upper Case of a String

strupr( ) function converts a given string into uppercase.

Syntax

```
char *strupr(char*string);
```

strupr( ) function is non-standard function which may not available in standard library in C.

Lower Case of a String

strlwr( ) function converts a given string into lowercase.

Syntax

```
char *strlwr(char*string);
```

strlwr( ) function is non-standard function which may not available in standard library in C.

Example: C program to illustrate

strlen(), strupr() , strlwr() , strrev()

```
#include<stdio.h>
#include<string.h> //c header file for string library functions
void main(){
char str1[10]="DeNnis" , str2[10]="RitChiE";
int len;
len=strlen(str1);
//strupr(str1);
//strlwr(str2);
printf("\n Length of string is %d", len);
//printf("\n upper case is %s" , str1);
//printf("\n lower case is %s" ,str2);
strrev(str1);
printf("\n Reverse of string is %s", str1);
}
```

OUTPUT:

```
Length of string is 6
Reverse of string is sinNeD
```

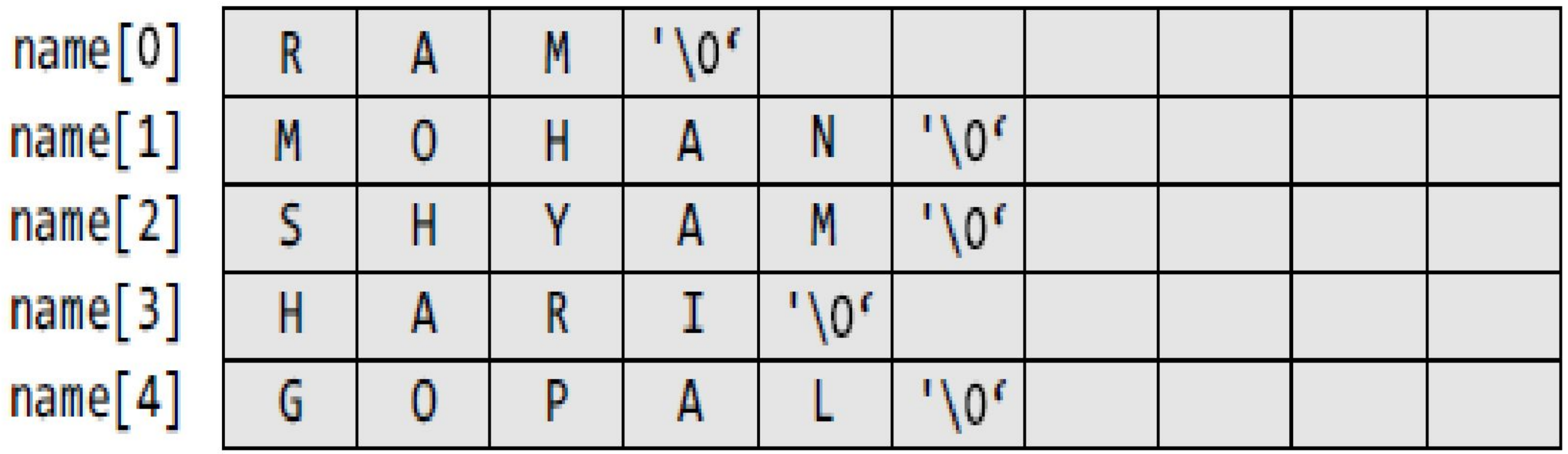
4.ARRAY OF STRINGS

- Till now we have seen that a string is an array of characters. For example, if we say char name[] = "Mohan", then the name is a string (character array) that has five characters.
- Now, suppose that there are 5 students in a class and we need a string that stores the names of all the 5 students. How can this be done? Here, we need a string of strings or an array of strings.
- Such an array of strings would store 5 individual strings. An array of strings is declared as char names[5][10];
- Here, the first index will specify how many strings are needed and the second index will specify the length of every individual string. So here, we will allocate space for 5 names where each name can be a maximum 10 characters long.
- Let us see the memory representation of an array of strings. If we have an array declared as

char name[5][10] = {"Ram", "Mohan", "Shyam", "Hari", "Gopal"};

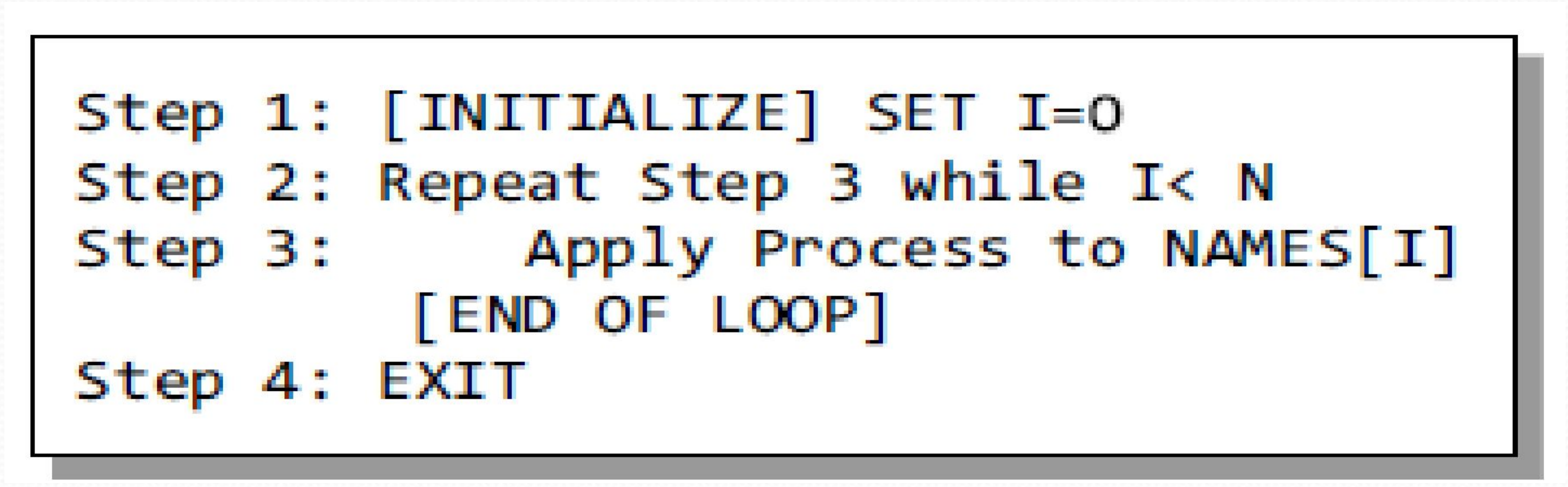
- Then in the memory, the array will be stored as shown in Fig.

Memory representation of a 2D character array



name[0]	R	A	M	'\0‘						
name[1]	M	O	H	A	N	'\0‘				
name[2]	S	H	Y	A	M	'\0‘				
name[3]	H	A	R	I	'\0‘					
name[4]	G	O	P	A	L	'\0‘				

- Algorithm
- In Step 1, we initialize the index variable I to zero.
- In Step 2, a while loop is executed until all the strings in the array are accessed.
- In Step 3, each individual string is processed.



```
Step 1: [INITIALIZE] SET I=0
Step 2: Repeat Step 3 while I< N
Step 3:         Apply Process to NAMES[I]
            [END OF LOOP]
Step 4: EXIT
```

WRITE A PROGRAM TO READ AND PRINT THE NAMES OF N STUDENTS OF A CLASS

```
#include<stdio.h>
#include<conio.h>
main()
{
char names[5][10];
inti, n;
clrscr();

printf(“\n Enter the number of students : “);
scanf(“%d”, &n);
for(i=0;i<n;i++)
{
printf(“\n Enter the name of %dthstudent : “, i+1);
gets(names[i]);
}
printf(“\n Names of the students are : \n”);
for(i=0;i<n;i++)
puts(names[i]);
getch();
return 0;
}
```