DATA STRUCTURES

Unit – II Strings

Outline

- String Representation
- String operations :
  - Finding length of a string
  - Copying a string
  - Converting Characters of a string into upper case and lower case,
  - Concatenation of two strings to form a new string,
  - String Append,
  - Reversing a string
  - Comparing strings
  - Insertion, Substring, Deletion

String representation

- String is defined as a sequence of characters.
- In terms of c language string is an array of characters.
- The string terminates with NULL or '\0', which is known as null terminated string.
- While storing the string in character array the size of the array must be one more then the size of the string. Because the last character in the string is '\0' or NULL.
- For example: To store "HELLO" in array the array must be declared as char a[6].

'H'	'E'	'L'	'L'	'O'	\0
0	1	2	3	4	5

String representation

String Character set:

- Lower case: a to z
- Upper case: A to Z
- Number: 0 to 9
- Special Characters: + - * % / ( ) [ ] { } $ # &, . ? ‘’ “” @ Etc.

Reading and Writing strings

Declaration of a string:

- char name[10];
- It will create character array of size 10 (excluding NULL).

Assigning a value to the string:

- char name[10]="avpti"

OR

- char name[10]={'a','v','p','t','i','\0'};
- This is static assignment of string.

Memory Representation

'a'	'v'	'p'	't'	'i'	'\0'	-	-	-	-	-
0	1	2	3	4	5	6	7	8	9	10

Reading and Writing strings

Reading a string:

Two functions can be used: gets(), scanf();

- scanf() can read only words whereas gets() can read full string with space.

For example

- scanf("%s", "hello world")→ **will display "hello"**
  - Only first word will be printed, next word can be displayed by subsequent scanf() if any.
- gets("hello world")→**will display "hello world"**

Writing a string:

- printf() with %s specifier or puts() can be used
- Example: printf("%s","hello world") or puts("hello world")

String Operation : Finding length of a string

- The number of characters in string defines string length.
- What will be the length of “hello world” string?
  - 11(including space, \0 is not included)
- C library function: **size_t strlen(const char * str)**
  - strlen() returns number of charactes
  - Example: length = strlen(“hello world”)
  - Which returns **11** and will be stored in **length** variable

Algorithm: Finding length of a string

```
SLEN(str)        ← user defined function
This function finds the length of the string.
Step 1: [initialization]
        count=0
Step 2: [Process until end of the string]
1.      Repeat step 3 until str[count] !='\0'
Step 3: [increment counter]
        count=count + 1
Step 4: [finish]
        Return count
```

String Operation :copying a string

- The content of one string can be copied to another string.

C library function strcpy()

char* strcpy(char* destination, const char* source);

- The string will be copied from source to destination
  - Example: str1="hello", str2="world"
  - what will be the output of strcpy(str1,str2) ?
    - **"world"**

Algorithm: copying a string

- **SCOPY(destination, source)**

```
step 1: [initialize]
        count←0
step 2: [Process until end of the loop]
        Repeat step 3 until source [count] !=NULL
Step 3: [copy character by character and increment counter]
         Destination [count] ←source [count]
         Count←count+1
Step 4: [finish]
         Destination [count]←NULL
```

Algorithm: Concatenation of two strings

- This operation concate two strings into one

C library function : strcat()

char * strcat(char * destination, const char * source)

- The strcat() function concatenates the *destination* string and the *source* string, and the result is stored in the **destination** string.
- For example *str1="hello"*, *str2="world"* then concatenation of str1 and str2 will give "*hello world*" as an output

Algorithm: Concatenation of two strings

SCAT(s1,s2,s3)

- Step 1: [initialize]

```
        count1 ← 0
        count2 ← 0
```

- Step 2: [process until end of first string]

```
        Repeat step 3 until s1 [count1] !=NULL
```

- Step 3: [copy character by character and increment counter]

```
        s3 [count1] ← s1 [count1]
        count1 ← count1+1
```

- Step 4: [process until end of second string]

```
        repeat step 5 until s2 [count2] !=NULL
```

Algorithm: Concatenation of two strings

```
Step 5: [copy character by character and increment counter]
        s3 [count1]←s2[count2]
        count1←count1+1
        count2←count2+1
Step 6: [finish]
        s3 [count1]← NULL
```

Algorithm: Comparison of two strings

- It compares two strings character by character until the corresponding characters of two strings are different or a null character '\0' is reached.

C library function: strcmp()

int strcmp (const char* str1, const char* str2);

- The strcmp() function takes two strings and returns an integer.

Return Value	Remarks
0	if both strings are identical (equal)
Negative	if the ASCII value of the first unmatched character is less than second.
Positive Integer	if the ASCII value of the first unmatched character is greater than second.

Algorithm: Comparison of two strings

```
SCOMP(s1,s2)
Step 1: [initialize]
          I←0
          FLAG←0
Step 2: [Find Length of two strings]
          L1←strlen (s1)
          L2←strlen (s2)
Step 3: [Compare Length of two string]
          If (L1 != L2) then
              Write “Strings are not equal”
Step 4: [Process string]
          Repeat step 5 while I < L1
```

Algorithm: Comparison of two strings

```
Step 5: [compare character by character]
        If s1[I] !=s2[I] then
            FLAG←1
        Else
            I←I + 1
Step 6: [Are string equal?]
        If FLAG = 0 then
            Write "Strings are equal"
        Else
            Write "Strings are not equal"
Step 7: [Finished]
```

Algorithm: Finding Substring

- This algorithm is used to find specific string or characters from a given string
- To find substring, Starting position and number of characters should be specified.

User defined function: substr(str, start, nchar)

- **str** : Input string , **start** : Starting position, **nchar**: number of characters
- Example: str = "Hello World"
- start=6 and nchar=5
- Then the output will be "World"

Algorithm: Finding Substring

Algorithm :substr(str, start, nchar)

```
Step1: [initialize]
        I←0
        start←start-1
Step 2: [Process the string]
        Repeat step 3 until nchar > 0
Step 3: [copy substring characters to output string]
        subs[I] ←str [start]
        I←I+1
        start←start+1
        nchar←nchar-1
Step 4: [Finished]
        subs [I] ←NULL
```

H	E	L	L	O		W	O	R	L	D
0	1	2	3	4	5	6	7	8	9	10

Algorithm : Reversing a string

- String reverse operation prints original input string into reverse order i.e. from last character to first.

C library function: strrev()

char *strrev(char *str)

- **str:** The input string which is needed to be reversed.
- This function returns the reversed string

- For example : str =“hello”

  rev =strrev(str)

  If we print **rev** then the output will be **“olleh”**

Algorithm : Reversing a string

```
REVERSE(org, rev)
Step 1: [Find Length of given string]
        L←strlen (org)
Step 2: [Initialize]
        COUNT1←L-1
        COUNT2←0
Step 3: [Process until end of the string]
        Repeat step 4 until org [COUNT2] ≠ NULL
Step 4: [Copy Character by character and increment counter]
        Rev [COUNT1] ←org [COUNT2]
        COUNT1←COUNT1-1
        COUNT2←COUNT2+1
Step 5: [Finished]
        Rev [L] ←NULL
```

H	E	L	L	O		W	O	R	L	D
0	1	2	3	4	5	6	7	8	9	10

Algorithm : Case conversion

- String can be converted from uppercase to lowercase as well as lowercase to uppercase
- For lowercase to uppercase conversion 32 will be subtracted from the each character of the string
- For uppercase to lowercase conversion 32 will be added to the each character of the string

(ASCII value of 'A' is 65 and 'a' is 97, so the difference is 32)

C library function: toupper() and tolower()

- int toupper(int character) and
- Int tolower(int character)
- Headerfile : ctype.h
- These functions take single character as an input and returns converted characters
- These functions deal with ASCII values of characters that is why return value and argument type are integer.

Algorithm : Lowercase to uppercase

```
UPPER_CASE(string, output)
    Step 1: [initialization]
              i← 0
    Step 2: [Process until the end of the string]
             Repeat step 3 until string[i] != NULL
    Step3 : [convert lower to upper case]
             if string [i] >='a' AND string[i]<='z'  then
                  output[i] = string[i]-32
                  i=i+1
             else
                 output[i]=string[i]
                 i=i+1
    Step 4: [finished]
```

Algorithm : uppercase to lowercase

```
LOWER_CASE(string, output)
    Step 1: [initialization]
             i← 0
    Step 2: [Process until the end of the string]
             Repeat step 3 until string[i] != NULL
    Step3 : [convert upper to lower case]
             if string[i] >='A' AND string[i]<='Z'  then
                    output[i] = string[i]+32
                    i=i+1
             else
                   output[i]=string[i]
                   i=i+1
    Step 4: [finished]
```

Other string conversion functions

String conversion : library stdlib.h

- The 'c' support the function that convert a string of digits in to their integer values.
- **int atoi(const char *str)**
- Where **str** is a character array and it will return integer value
- **Example:** consider string variable **number** = "1972"

  Year= atoi(number)
- This function convert string "1972" into integer 1972 and store into **integer variable year.**
- There are a few functions that exist to convert strings to integer, long integer and float values.
- **double atof(char *string)** -- Convert string to floating point value.

  int atoi(char *string) -- Convert string to an integer value

  int atol(char *string) -- Convert string to a long integer value.

Other string conversion functions

String conversion : library stdlib.h

- itoa () function converts integer data type to string data type.

  char * itoa (int value, char * str, int base);

- Where **value** is an integer variable, **str** is string variable which stores output and **base** is for specifying base of value i.e decimal, binary etc
- Example: consider a=54325

```
itoa(a,buffer,2);   // here 2 means binary
printf("Binary value = %s\n", buffer);  // prints  1101010000110101 (As a string)
itoa(a,buffer,10);   // here 10 means decimal
printf("Decimal value = %s\n", buffer);   // prints 54325
itoa(a,buffer,16);   // here 16 means Hexadecimal
printf("Hexadecimal value = %s\n", buffer);    // prints D435 as a string
```

- Similarly **char *ltoa(long N, char *str, int base)** converts long data type into string data type

String input output functions

Input functions: used to get input from user

- **scanf()** : scanf("%s", str) → used to read a single word
  scanf("%[^\n]", str)→ used to read full string with space
- **gets() : gets(str)** → can read full string including space
- **gectchar() : ch = getchar()**→ used to read single character at a time

Output functions : used to display output to the screen

- **printf()** : printf("hello World")
- **puts():** puts("hello world")
- **putchar()** : putchar('c') → used to display single character at a time

Algorithm: String Insertion

- String insertion operation will insert sub-string at specified position into the input string.
- Example: Input string = "hello, How you?"

  String to be inserted = "are "

  After insertion starting from position 11th, new string will be

  "hello, How are you?"

Function : Insertion (text, position, string)

- **Text :** input string
- **Position:** starting position for insertion of new string
- **String :** String to be inserted

Algorithm: String Insertion

Algorithm: Insertion (text, position, string)

String : “are ”, position: 4

text:	**h**	**o**	**w**		**y**	**o**	**u**				
	0	1	2	3	4	5	6				
temp	**h**	**o**	**w**								
	0	1	2	3	4	5	6	7	8	9	10

i	**0 1 2 3 4**
k	**0**

```
Step 1: [Initialization]
        i← 0
        k←0
        Temp← NULL
Step 2: [ Reach at the insert position]
        Repeat step 3 until i != position
Step 3: [ Copy data up to specified position into temp]
        temp[i]←text[i]
        i←i+1
```

Algorithm: String Insertion

```
Step 4: [Insert string at specified position]
        Repeat  following steps until  k != strlen(string)
          temp[i]←string[k]
          i←i+1
          k←k+1
Step 5: [Insert rest of the characters after inserting new string]
        Repeat  following steps until  text [position] != NULL
          temp[i]←text[position]
          i←i+1
          position ← position+1
Step 6: [Finished]
        Temp[position]=NULL
```

i	**0 1 2 3 4 5 6 7 8 9 10 11**
k	**0 1 2 3 4**
Position	**4 5 6 7**

String : "are ", position: 4 , strlen=4

text:	**h**	**o**	**w**		**y**	**o**	**u**				
	0	1	2	3	4	5	6				
temp	**h**	**o**	**w**		**a**	**r**	**e**		**y**	**o**	**u**
	0	1	2	3	4	5	6	7	8	9	10

Algorithm: String Deletion

- String deletion operation will delete sub-string starting from specified position of input string. (Number of characters to be deleted should be specified)
- Example: Input string = “hello, How are you?”

  starting position = 11, number of characters=4

  After deletion new string will be

  “hello, How you?”

Function : Deletion (text, position, length)

- **Text** : input string
- **Position:** starting position for deletion
- **Length** : Number of characters to be deleted

Algorithm: String Deletion

- **Algorithm: Deletion (text, position, length)**

position: 4 , length : 4

text:	h	o	w		a	r	e		y	o	u
	0	1	2	3	4	5	6	7	8	9	10
temp	h	o	w								
	0	1	2	3	4	5	6	7	8	9	10

i	0 1 2 3 4
k	0 1 2 3 4

```
Step 1: [Initialization]
        i← 0
        k←0
        Temp← NULL
Step 2: [ Reach at the deletion position]
        Repeat step 3 until i != position
Step 3: [ Copy data upto specified position into temp]
        temp[k]←text[i]
        i←i+1
        k←k+1
```

Algorithm: String Deletion

```
Step 4: [Find new position after deletion]
        i ← position + length
Step 5: [Insert rest of character after deletion]
        Repeat  following steps until  text [i] != NULL
        temp[k]←text[i]
        i←i+1
        k← k+1
Step 6: [Finished]
        Temp[k]=NULL
```

position: 4 , length : 4

text:	**h**	**o**	**w**		**a**	**r**	**e**		**y**	**o**	**u**
	0	1	2	3	4	5	6	7	8	9	10
temp	**h**	**o**	**w**		**y**	**o**	**u**				
	0	1	2	3	4	5	6	7	8	9	10

i	**0 1 2 3 4 8 9 10 11**
k	**0 1 2 3 4 5 6 7**

Algorithm: String Append

- Append operation adds new string at the end of existing string
- For example : input string =“hello,”

  New string : “ How are you?”

  After append operation Input string will be:

  “Hello, How are you?”

Function : Append (text, string)

- **Text :** input string
- **String :** String to append

Algorithm: String Append

- **Algorithm: Append(text, string)**

```
Step 1: [Initialization]
            i← 0
            k←0
Step 2: [ Find length of the string]
            i← strlen(text)
Step 3: [ Append the string]
            Repeat until  k != strlen(string)
               text[i]←string[k]
               i←i+1
               k←k+1
Step 4 : [Finish]
            text[i]←NULL
```