**Unit8 Strings**

**8.1 Define String:**

A string is a **sequence of characters that is treated as a single data item**. Any group of characters defined between double quotation marks is a string constant.

Ex: “Well Done”

 Character strings are used to build meaningful and readable programs. The common operations performed on character strings include:

* Reading and Writing strings.
* Combining strings together.
* Copying one string to another.
* Comparing strings for equality.
* Extracting a portion of a string.

**8.2 Know about declaration and initialization of String variables:**

**Declaration of string variables:**

 C does not support strings as a datatype. It allows us to represent strings as character arrays. The general form of declaration of string variable is:

 Char string\_name[size];

 The **size** determines the number of characters in the string name.

Ex: char name[10];

 When the compiler assigns a character string to a character array, it automatically supplies a *null* character (‘\0’) at the end of the string. The *size* should be equal to the maximum number of characters in the string *plus* one.

**Initialization of string variables:**

 Character arrays may be initialized when they are declared. C permits a character array to be initialized in either of following two forms:

 char name[15]=”ANDHRA PRADESH”;

 charname[15]={‘A’,’N’,’D’,’H’,’R’,’A’,’ ’,’P’,’R’,’A’,’D’,’E’,’S’,’H’,’\0’};

**Note:** When we initialize a character array we must supply explicitly the **NULL** terminator.

* C permits us to initialize a character array without specifying the number of elements.

Ex: char name[ ]={‘A’,’N’,’D’,’H’,’R’,’A’,’\0’};

 The statement defines the array **name** as a **seven** element array.

* We can also declare the size much larger than the string size in the initialize.

Ex: char name[10]=”ANDHRA”;

In the computer creates a character array of size 10, places the value “ANDHRA” in it. Terminated with the null character, and initializes all other elements to NULL.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | N | D | H | R | A | \0 | \0 | \0 | \0 |

* The following declaration is illegal.

Ex: char name[6]=”ANDHRA”;

* We cannot separate the initialization form declaration.

Ex: char name[10];

 name=”ANDHRA”;

* An array name cannot be used as the left operand of an assignment operator.

Ex: char name[10]=”ANDHRA”;

char name1; name1=name;**/\*Error\*/**

**8.3 Know about reading of strings from terminal with sample program:**

The reading of strings from terminal is done by using **four** ways. They are:

1. Using **scanf** function.
2. Using **edit conversion character code.**
3. Using **getchar**function.
4. Using **gets** function.
* Using **scanf** function:

 The input function **scanf** can be used with **%s** format specification to read in a string of characters.

Ex: char name[10];

scanf(“%s”, name);

 The disadvantage with the **scanf** function is that **it terminates its input on the first white space it finds.** A white space includes blanks, tabs, carriage returns, form feeds and new lines.

Ex: NEW YORK

 The only string “NEW” will be read into the array **name**.

Program:

#include<stdio.h>

main()

{

char word1[40],word2[40],word3[40],word4[40];

printf("ENTER THE TEXT.\n");

scanf("%s %s %s %s",word1,word2,word3,word4);

printf("Word1= %s \n Word2= %s \nWord3= %s \n Word4= %s\n",word1,word2,word3,word4 );

}

* Using **edit set conversion code:**

C supports a format specification known as the **edit set conversion code %[..]** that can be **used to read a line containing a variety of characters, including white spaces**.

Ex: #include <stdio.h>

main() O/P

{

char line[80];

printf("enter a string\n");

scanf("%[^\n]",line);

printf("%s",line);

}

Will read a line of input from the keyboard and display the same on the screen.

* Using **getchar**function:

**Using this function repeatedly we can read successive single characters** from the input and place them into a character array. Thus, an entire line of text can be read and stored in an array. The reading is terminated when the newline character (‘\n’) is entered and the null character is then inserted at the end of the string.

Ex: char ch;

ch=getchar();

Note: The getchar() function has no parameters.

Program:

#include<stdio.h>

main()

{

 char line[81],character;

 int i=0;

 printf("Enter the text. press<RETURN> at end\n");

 do

 {

 character=getchar();

 line[i]=character;

 i++;

 }while(character!='\n');

 i=i-1;

 line[i]='\0';

 printf("%s",line);

}

* Using **gets** function:

 Another more convenient method of reading a string of text containing whitespaces is to use the library function gets available in the <stdio.h> header file. The form of gets function is:

 **gets(str);**

str is a string variable declared properly. It reads characters into str from the keyboard until a new-line character is encountered and then appends a null character to the string. Unlike scanf, it does not skip white spaces.

e.g.: #include<stdio.h>

 main()

 {

 char line[80];

 printf("enter a sting\n");

 gets(line);

 printf("%s",line);

 }

reads a li ne of text from the keyboard and displays it on the screens.

**8.4 Know about writing strings to screen with sample program:**

The writing strings to screen are done by using **three** ways. They are:

1. Using **printf**function.
2. Using **putchar**function.
3. Using **puts** function.
* Using **printf** function:

The **printf** function with **%s** format is used to print strings to the screen. The format **%s** is can be used to display an array of characters that is terminated by the null character.

Ex: printf(“%s”, name);

 Can be used to display the entire contents of the array name.

**Using putchar function**:

Like getchar, C supports another character handling function putchar to output the value of character variables. It takes the following form:

 **char ch=’A’;**

 **putchar (ch);**

The putchar function requires only one parameter.

* **Using puts function**:

Another way of printing string values is use the puts function declared in the header file

<stdio.h>. It takes the following form:

**puts(str);**

This prints the value of the string variable str and hen moves the cursor to the beginning of the next line on the screen.

**8.5 Explain about various String handling functions with sample program:**

The C-library supports a large number of string-handling functions that can be **used to carry out many of the string manipulations.** Following are the most commonly used string handling functions.

|  |  |
| --- | --- |
| FUNCTION | ACTION |
| strcat()strcmp()strcpy()strlen()strrev() | concatenates two strings.compares two strings.copies one string over another.finds the length of a string.Reverse the given string. |

* **strcat() function:**

The **strcat** function joins two strings together. It takes the following form:

 **strcat( string1, string2);**

**Note:** string1 and string2 are character arrays.

 The function **strcat** is executed, **string2** is appended to **string1.**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| V | E | R | Y |  | \0 |  |  |  |  |  |

**Ex:**

Str1=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | O | O | D |  | \0 |  |  |  |  |

Str2=

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M | O | R | N | I | N | G | \0 |  |  |

Str3=

 Execution of the statement

 strcat(str1,str2);

 will result in:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| V | E | R | Y |  | G | O | O | D | \0 |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | O | O | D |  | \0 |  |  |  |  |  |

Str1=

Str2=

* **strcat** function may also append a string constant to a string variable.

Ex: strcat(str1,”GOOD”); is valid statement

* C permits nesting of **strcat** functions.

Ex: strcat(strcat(str1, str2), str3);

* **strcmp ()** function: Compares the two strings identified by the arguments and has a value 0, if they are equal. If they are not equal , it has the numeric difference between the first non-matching characters in the strings. It takes the form:

**strcmp(string1,string2);**

string1, string2 may be string variables or string constants. Examples are:

 strcmp(name1,name2);

 strcmp(name1,”john”);

 strcmp(“rom”,”ram”);

* **strcpy()** function: copies one string over another. It takes the following form:

**strcpy(string1,string2);**

 Assigns the contents of string2 to string1. The string2 may be a character array variable or a string constant.

Ex: strcpy(city, “DELHI”);

will assign the string DELHI to the string variable city.

* **strlen()** function: this function counts and returns the number of characters in a string . it takes the form:

 **n=strlen(string);**

 Where, n is an integer variable, which receives the value of the length of the string.

**Other String Functions:**

The header file **<string.h>**contains many more string manipulation functions. They are:

1. strncpy: Copies the left-most n characters of the source string to the target string.
2. strncmp: Compares the left-most n characters from first string with second string.
3. strncat: Concatenate the left-most n characters of second string to first string.
4. strstr: It is a two-parameter function that can be used to locate a sub-string in a string.

**8.6 Explain Arithmetic operations on Characters:**

* C-allows us to manipulate characters the same way we do with numbers.
* Whenever a character constant or character variable is used in an expression, **it is automatically converted into an integer value**.
* To **write a character in its integer representation**, we may write it as an integer.for e.g.:

 x=’a’;

 printf(“%d”,x);

Will display the number 97 on the screen.

* It is also **possible to perform arithmetic operation on the character constants** and variables. For e.g.:

x=’z’-1;

in ASCII , the value of ‘z’ is 122 , therefore the above statement will assign the value 121 to variable x.

* We can also **use character constants in relational expressions**. For example the expression

ch>=’A’ &&ch<= ‘Z’

 Would test whether the character contained in ch is an upper-case letter.

* We can convert a character digit to its equivalent integer value using the following relationship:

 x=ch-‘0’;

x🡪integer variable

ch🡪character constant contains the character digit let say the digit is ‘7’.

Then , x=ASCII value of ‘7’-ASCII value of ‘0’

 =55-48

 =7

* The C library supports a function that converts a string of digits into their integer value. the function takes the form:

 **x=atoi(string);**

 for e.g.: number=”1988”;

 year=atoi(number);

 The number is a string variable which is assigned the string constant “1988” .the function atoi converts the string “1988” to its numeric equivalent 1988 and assign to x (integer variable).

**Example Program:**

 main()

 {

 charch;

 for(ch=65;ch<=122;ch++)

 {

 if(ch>90&&ch<97)

 continue;

 printf("%d-->%c:",ch,ch);

 printf("\n");

 }

getch();

 }

**Write a C-program to check whether given string is palindrome or not without using string handling functions.**

#include<string.h>

main()

{

charstr[10];

int n,i,j;

clrscr();

printf("enter a string\n");

scanf("%s",str);

n=strlen(str);

i=0;

j=n-1;

while(i!=j)

 {

if(str[i]==str[j])

 {

i++;

j--;

 }

else

break;

}

if(i==j)

{

printf("given string is palindrome");

 }

else

printf("not a palindrome\n");

}