

EXERCISE 9

Content:

1. String in C
2. C gets() & puts()
3. C String Functions

I. String in C

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as `char s[10]`, the character `s[10]` is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language:

- By char array
- By string literal

Let's see the example of declaring string by char array:

```
char ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

As we know, array index starts from 0, so it will be represented as in the figure given below:

0	1	2	3	4	5	6	7	8	9	10
j	a	v	a	t	p	o	i	n	t	\0

While declaring string, size is not mandatory. So we can write the above code as given below:

```
char ch[]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

We can also define the string by the string literal. For example:

```
char ch[]="CLanguage";
```

In such case, '\0' will be appended at the end of the string by the compiler.

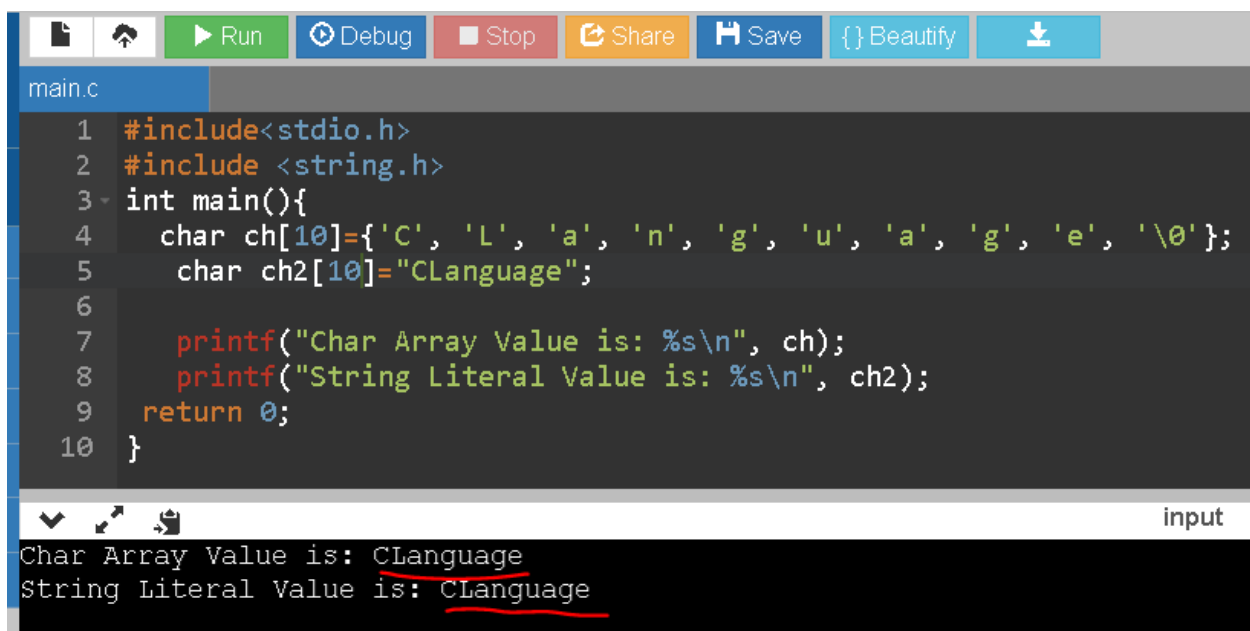
Difference between char array and string literal

There are two main differences between char array and literal:

- We need to add the null character '\0' at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

String Example

Let's see a simple example where a string is declared and being printed. The '%s' is used as a format specifier for the string in c language.

A screenshot of a code editor showing a C program. The editor has a toolbar at the top with buttons for Run, Debug, Stop, Share, Save, Beautify, and Download. The code is in a file named 'main.c'. It includes <stdio.h> and <string.h>. The main function declares two character arrays: 'ch' and 'ch2'. 'ch' is initialized with the characters 'C', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', and a null terminator '\0'. 'ch2' is initialized with the string "CLanguage". Both arrays are printed using printf with the format specifier "%s\n". The output shows "Char Array Value is: CLanguage" and "String Literal Value is: CLanguage". The word "CLanguage" in the output is underlined in red.

```
1 #include<stdio.h>
2 #include <string.h>
3 int main(){
4     char ch[10]={'C', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', '\0'};
5     char ch2[10]="CLanguage";
6
7     printf("Char Array Value is: %s\n", ch);
8     printf("String Literal Value is: %s\n", ch2);
9     return 0;
10 }
```

Char Array Value is: CLanguage
String Literal Value is: CLanguage

Traversing String

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which

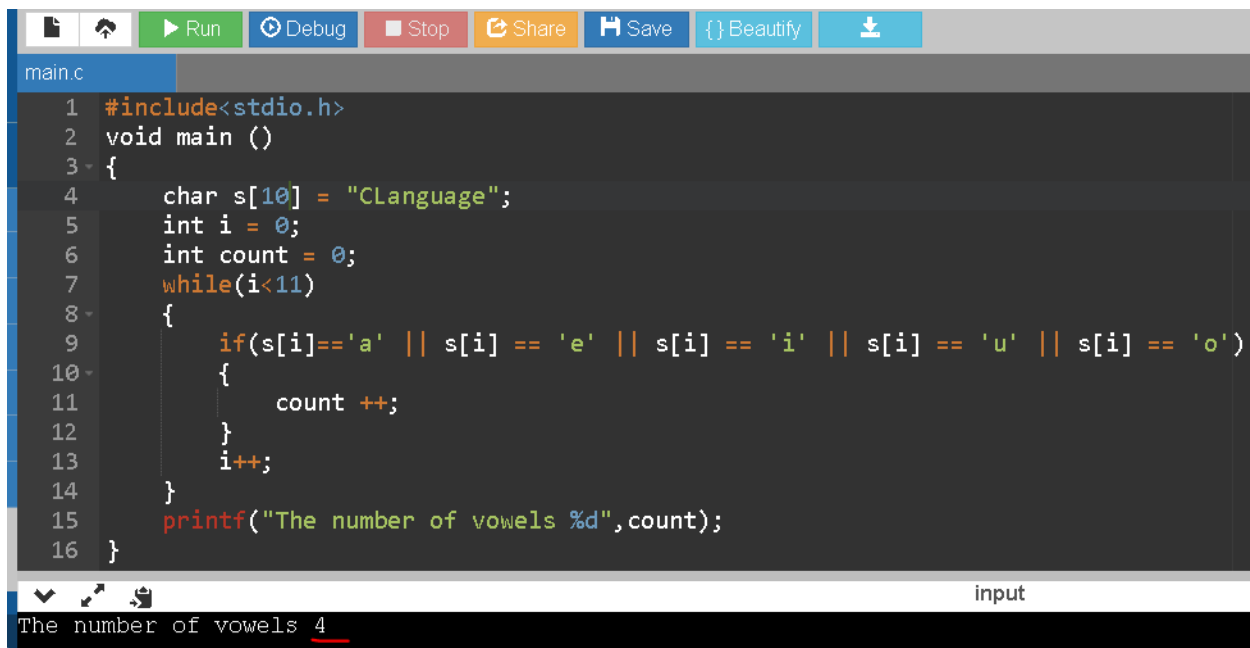
can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.

Hence, there are two ways to traverse a string:

- By using the length of string
- By using the null character.

Using the length of string

Let's see an example of counting the number of vowels in a string:

A screenshot of a code editor showing a C program. The editor has a toolbar at the top with icons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name is 'main.c'. The code is as follows:

```
1 #include<stdio.h>
2 void main ()
3 {
4     char s[10] = "CLanguage";
5     int i = 0;
6     int count = 0;
7     while(i<11)
8     {
9         if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
10        {
11            count ++;
12        }
13        i++;
14    }
15    printf("The number of vowels %d",count);
16 }
```

At the bottom, there is a terminal window with the output 'The number of vowels 4' and an 'input' label on the right.

Using the null character

Let's see the same example of counting the number of vowels by using the null character.

```
main.c
1 #include<stdio.h>
2 void main ()
3 {
4     char s[10] = "CLanguage";
5     int i = 0;
6     int count = 0;
7     while(s[i] != NULL)
8     {
9         if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
10        {
11            count ++;
12        }
13        i++;
14    }
15    printf("The number of vowels %d",count);
16 }
```

input

main.c:7:16: warning: comparison between pointer and integer
7 | while(s[i] != NULL)
| ^~
The number of vowels 4

Accepting string as the input

Till now, we have used scanf to accept the input from the user. However, it can also be used in the case of strings but with a different scenario. Consider the below code which stores the string while space is encountered.

```
main.c
1 #include<stdio.h>
2 void main ()
3 {
4     char s[20];
5     printf("Enter the string?");
6     scanf("%s",s);
7     printf("You entered %s",s);
8 }
```

Enter the string?CLanguage
You entered CLanguage

It is clear from the output that, the above code will not work for space separated strings. To make this code working for the space separated strings, the minor changed required in the scanf function, i.e., instead of writing

scanf("%s",s), we must write: scanf("%[^\n]s",s) which instructs the compiler to store the string s while the new line (\n) is encountered. Let's consider the following example to store the space-separated strings.

```
main.c
1  #include<stdio.h>
2  void main ()
3  {
4      char s[20];
5      printf("Enter the string?");
6      scanf("%[^\n]s",s);
7      printf("You entered %s",s);
8  }
```

Enter the string?My Name is Zeki
You entered My Name is Zeki

Here we must also notice that we do not need to use address of (&) operator in scanf to store a string since string s is an array of characters and the name of the array, i.e., s indicates the base address of the string (character array) therefore we need not use & with it.

Some important points

However, there are the following points which must be noticed while entering the strings by using scanf.

- The compiler doesn't perform bounds checking on the character array. Hence, there can be a case where the length of the string can exceed the dimension of the character array which may always overwrite some important data.
- Instead of using scanf, we may use gets() which is an inbuilt function defined in a header file string.h. The gets() is capable of receiving only one string at a time.

II. C gets() & puts()

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

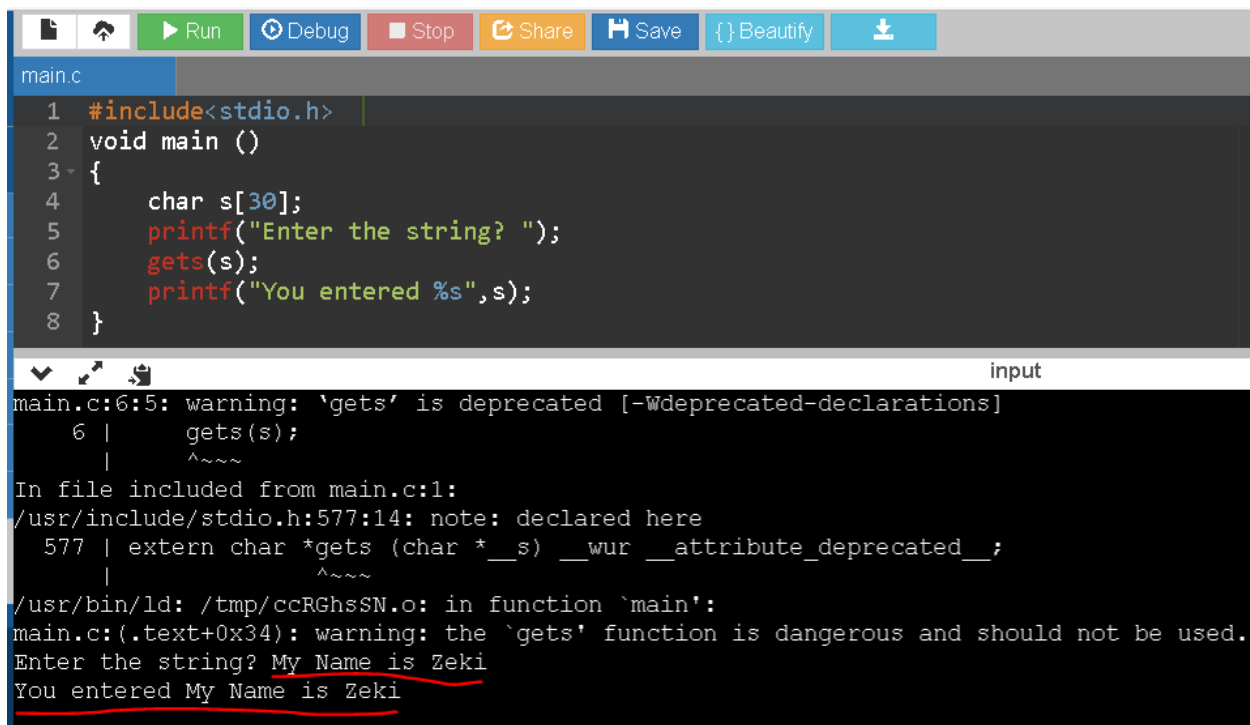
C gets() function

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Declaration

```
char[] gets(char[]);
```

Reading string using gets():

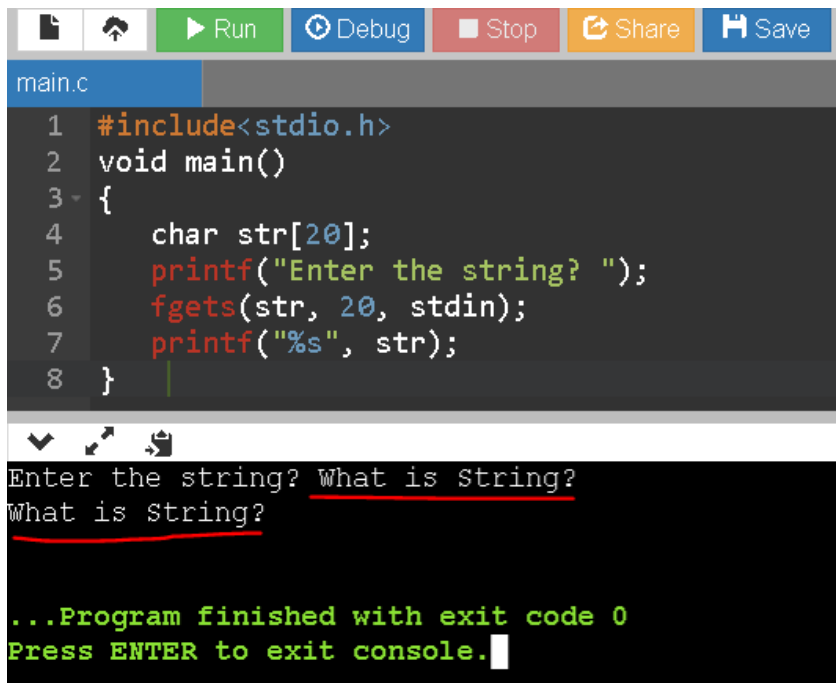


```
main.c
1 #include<stdio.h>
2 void main ()
3 {
4     char s[30];
5     printf("Enter the string? ");
6     gets(s);
7     printf("You entered %s",s);
8 }
```

input

```
main.c:6:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]
6 |     gets(s);
  |     ^~~~
In file included from main.c:1:
/usr/include/stdio.h:577:14: note: declared here
577 | extern char *gets (char *__s) __wur __attribute_deprecated__;
    |             ^~~~
/usr/bin/ld: /tmp/ccRGhsSN.o: in function `main':
main.c:(.text+0x34): warning: the `gets' function is dangerous and should not be used.
Enter the string? My Name is Zeki
You entered My Name is Zeki
```

The gets() function is risky to use since it doesn't perform any array bound checking and keep reading the characters until the new line (enter) is encountered. It suffers from buffer overflow, which can be avoided by using fgets(). The fgets() makes sure that not more than the maximum limit of characters are read. Consider the following example:

The image shows a code editor window with a toolbar at the top containing icons for file operations, a 'Run' button, a 'Debug' button, a 'Stop' button, a 'Share' button, and a 'Save' button. The editor displays a C program in a file named 'main.c'. The code includes the standard input/output header, defines the main function, declares a character array 'str' of size 20, prompts the user to enter a string, reads the input using 'fgets', and prints it back using 'printf'. Below the code editor is a terminal window showing the program's execution. It displays the prompt 'Enter the string? ', the user input 'What is String?', and the program output 'What is String?'. The terminal also shows the message '...Program finished with exit code 0' and 'Press ENTER to exit console.'.

```
1 #include<stdio.h>
2 void main()
3 {
4     char str[20];
5     printf("Enter the string? ");
6     fgets(str, 20, stdin);
7     printf("%s", str);
8 }
```

Enter the string? What is String?
What is String?
...Program finished with exit code 0
Press ENTER to exit console.

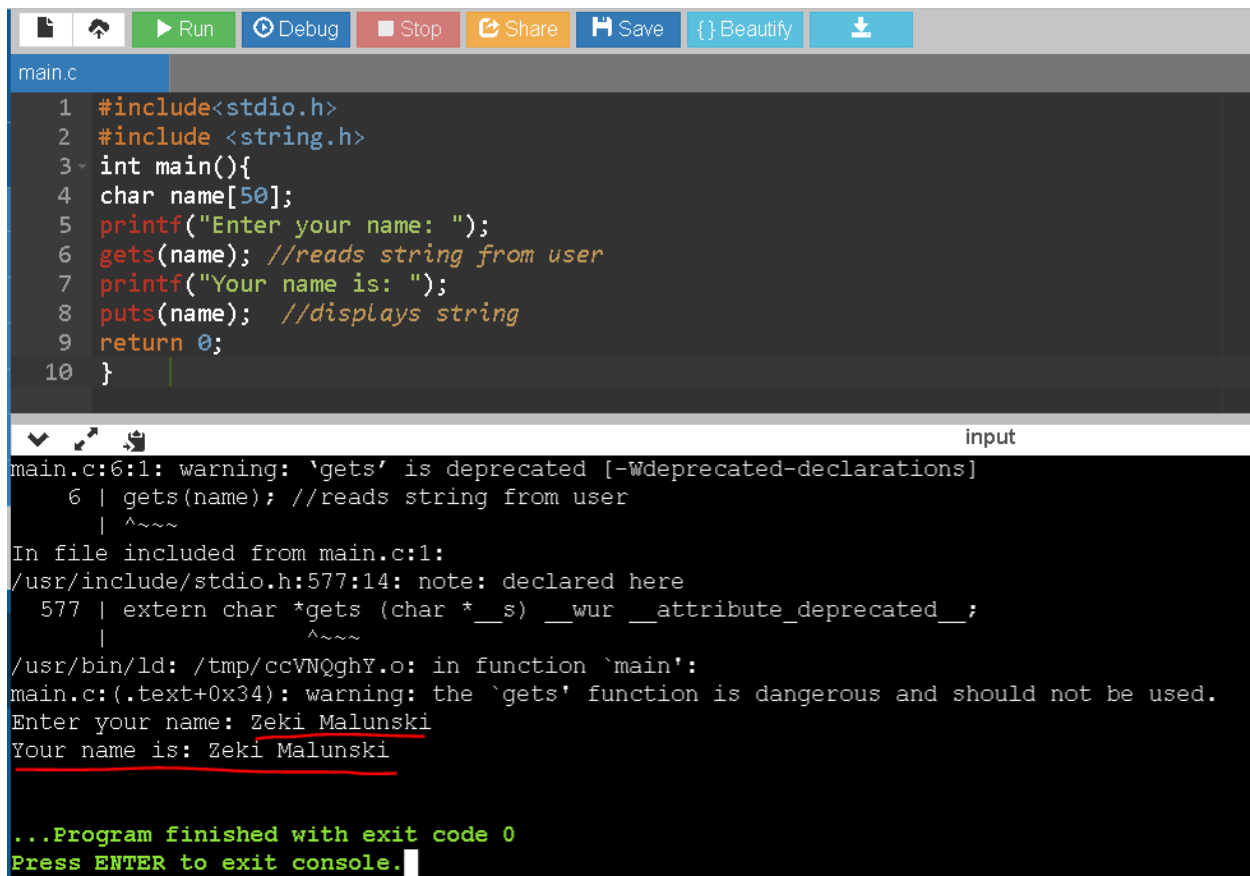
C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

Declaration

```
int puts(char[])
```

Let's see an example to read a string using gets() and print it on the console using puts().



```
main.c
1 #include<stdio.h>
2 #include <string.h>
3 int main(){
4     char name[50];
5     printf("Enter your name: ");
6     gets(name); //reads string from user
7     printf("Your name is: ");
8     puts(name); //displays string
9     return 0;
10 }
```

input

```
main.c:6:1: warning: 'gets' is deprecated [-Wdeprecated-declarations]
6 | gets(name); //reads string from user
  | ^~~~
In file included from main.c:1:
/usr/include/stdio.h:577:14: note: declared here
577 | extern char *gets (char *__s) __wur __attribute_deprecated__;
    |              ^~~~
/usr/bin/ld: /tmp/ccVNQghY.o: in function `main':
main.c:(.text+0x34): warning: the `gets' function is dangerous and should not be used.
Enter your name: Zeki Malunski
Your name is: Zeki Malunski

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

III. C String Functions

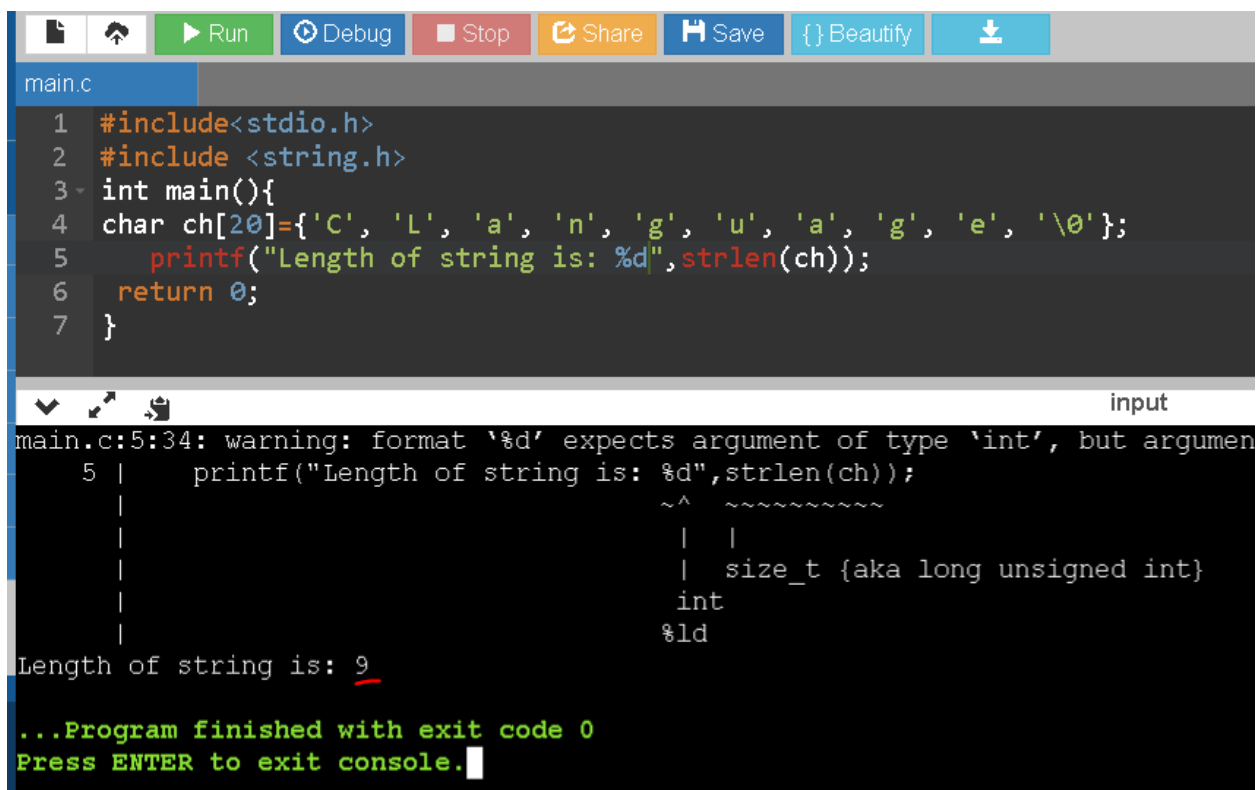
There are many important string functions defined in "string.h" library.

No.	Function	Description
1)	<u>strlen(string_name)</u>	returns the length of string name.
2)	<u>strcpy(destination, source)</u>	copies the contents of source string to destination string.
3)	<u>strcat(first_string, second_string)</u>	concat or joins first string with second string. The result of the string is stored in first string.
4)	<u>strcmp(first_string, second_string)</u>	compares the first string with second string. If both strings are same, it returns 0.

5)	<u>strrev(string)</u>	returns reverse string.
6)	<u>strlwr(string)</u>	returns string characters in lowercase.
7)	<u>strupr(string)</u>	returns string characters in uppercase.

C String Length: strlen() function

The strlen() function returns the length of the given string. It doesn't count null character '\0'.



```

main.c
1  #include<stdio.h>
2  #include <string.h>
3  int main(){
4  char ch[20]={'C', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', '\0'};
5      printf("Length of string is: %d",strlen(ch));
6      return 0;
7  }

main.c:5:34: warning: format '%d' expects argument of type 'int', but argumen
5 |     printf("Length of string is: %d",strlen(ch));
  |                                ~^  ~~~~~
  |                                |  |
  |                                |  size_t {aka long unsigned int}
  |                                int
  |                                %ld
Length of string is: 9

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

```

C Copy String: strcpy()

The strcpy(destination, source) function copies the source string in destination.

```
main.c
1  #include<stdio.h>
2  #include <string.h>
3  int main(){
4      char ch[20]={'C', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', '\0'};
5      char ch2[20];
6      strcpy(ch2,ch);
7      printf("Value of second string is: %s",ch2);
8      return 0;
9  }
```

input

Value of second string is: CLanguage

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

C String Concatenation: strcat()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

```
main.c
1  #include<stdio.h>
2  #include <string.h>
3  int main(){
4      char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
5      char ch2[10]={'c', '\0'};
6      strcat(ch,ch2);
7      printf("Value of first string is: %s",ch);
8      return 0;
9  }
```

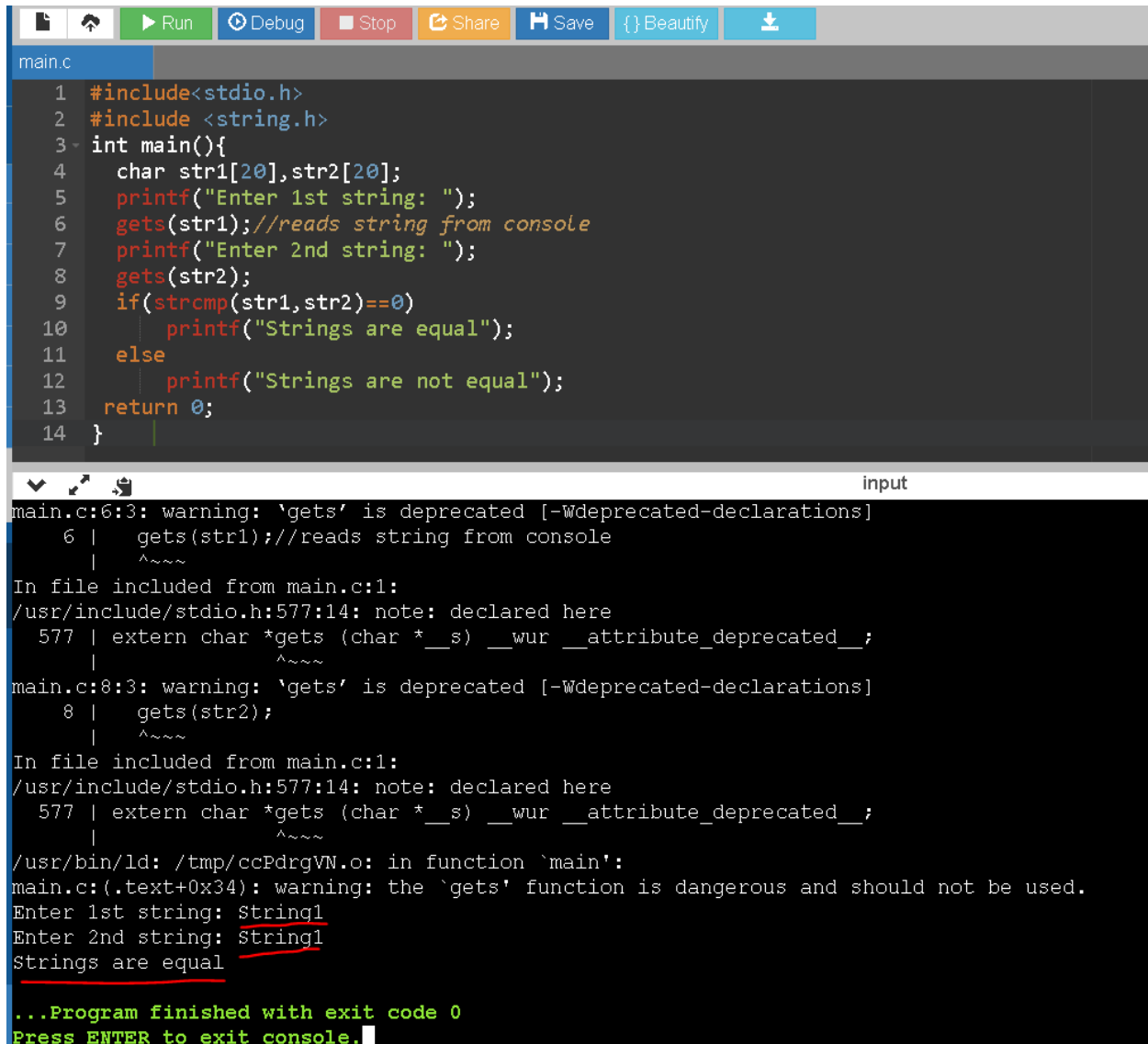
Value of first string is: helloc

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

C Compare String: strcmp()

The `strcmp(first_string, second_string)` function compares two string and returns 0 if both strings are equal.

Here, we are using `gets()` function which reads string from the console.



```
main.c
1 #include<stdio.h>
2 #include <string.h>
3 int main(){
4     char str1[20],str2[20];
5     printf("Enter 1st string: ");
6     gets(str1);//reads string from console
7     printf("Enter 2nd string: ");
8     gets(str2);
9     if(strcmp(str1,str2)==0)
10         printf("Strings are equal");
11     else
12         printf("Strings are not equal");
13     return 0;
14 }
```

input

```
main.c:6:3: warning: 'gets' is deprecated [-Wdeprecated-declarations]
  6 |     gets(str1);//reads string from console
    |     ^~~~
In file included from main.c:1:
/usr/include/stdio.h:577:14: note: declared here
 577 | extern char *gets (char *__s) __wur __attribute_deprecated__;
    |             ^~~~
main.c:8:3: warning: 'gets' is deprecated [-Wdeprecated-declarations]
  8 |     gets(str2);
    |     ^~~~
In file included from main.c:1:
/usr/include/stdio.h:577:14: note: declared here
 577 | extern char *gets (char *__s) __wur __attribute_deprecated__;
    |             ^~~~
/usr/bin/ld: /tmp/ccPdrGVN.o: in function `main':
main.c:(.text+0x34): warning: the `gets' function is dangerous and should not be used.
Enter 1st string: String1
Enter 2nd string: String1
Strings are equal

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