

Strings

String Declarations

- A string, as it is simply an array of characters, is declared as:

`char string1[20];`

- This declares a string named `string1` which can hold strings from length 0 to length 19
- Notice that this fact means we “lose,” or cannot use one of the 20 characters declared. **Why?**

Strings in Memory

- All strings must end with a null character. When a string is enclosed in double quotes, i.e. “Hello there”, a null character is automatically inserted.
- The double quotes mean “the string is whatever is inside the quotes followed by a null character.”

String Initialization

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' } ;
```

```
char pet[5] ;  
pet[0] = 'l' ; pet[1] = 'a' ; pet[2] = 'm' ;  
pet[3] = 'b' ; pet[4] = '\0' ;
```

```
char pet[5] = "lamb" ;
```

All equivalent

String Initialization

- Strings can be initialized using a string constant, in this form:

`char string1[20] = "Hello there";`

- So, what does this string look like in memory?

| | | | | | | | | | |
|---|---|---|---|---|--|---|---|---|---|
| H | e | l | l | o | | t | h | e | r |
|---|---|---|---|---|--|---|---|---|---|

| | | | | | | | | | |
|---|----|---|---|---|---|---|---|---|---|
| e | \0 | ? | ? | ? | ? | ? | ? | ? | ? |
|---|----|---|---|---|---|---|---|---|---|

2D character arrays

- You are required to store names of 5 students, you need to declare a 2D array of character type that is array of strings

```
char stud_names[ 5 ][ 10 ] ;
```

5 rows to store names of 5 students.

Each row will hold a name, 10 columns that is max length of a name could be 10 characters.

Initialize 2-D array

- Initialize an array to hold the names of all the months of a year

```
char month[ ][10] = { "January", "February",  
                      "September", ....., "December" } ;
```

```
puts(month[7]);
```

char stud_names [5] [10];

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
|------------|-----|-----|-----|-----|-----|-----|------|-----|-----|-----|
| Student #1 | J | O | H | N | \0 | | | | | |
| Student #2 | R | I | C | H | A | R | D | \0 | | |
| Student #3 | M | A | R | Y | | J | O | H | N | \0 |
| Student #4 | M | A | R | T | I | N | A | \0 | | |
| Student #5 | J | I | M | M | Y | \0 | | | | |

Reading values in 2D character array

```
int row;  
char stud_names [ 5 ] [ 10 ];
```

```
/* reading values in character array*/
```

```
for(row=0; row<5; row++)
```

```
    gets(stud_names[row]);
```

```
/* displaying values of array */
```

```
for(row=0;row<5; row++)
```

```
    puts(stud_names[row]);
```

stud_name [row]

stud_name[5] [10]

| | | | | | | | | | | | |
|-------|---|---|---|---|---|---|----|----|----|---|----|
| [0] | → | B | H | A | R | A | T | \0 | | | |
| [1] | → | S | A | N | D | E | E | P | \0 | | |
| [2] | → | M | A | R | Y | | J | O | H | N | \0 |
| [3] | → | M | A | R | T | I | N | A | \0 | | |
| [4] | → | J | I | M | M | Y | \0 | | | | |

String Operations

- Common needed operations:
 - Copy (assignment)
 - Compare
 - Find length
 - Concatenate (combine strings)
 - Find substring
 - I/O

What You Can't Do

- Can't assign one string to another with =
- Can't compare strings with ==, <=
- But there are library functions to help do such things

String Library: <string.h>

- C provides a number of routines to manipulate strings, through library functions.
- Standard C includes a library of string functions
 - use *#include <string.h>*

String Input/Output

```
char string1[20];  
char string2[20];  
gets(string1);  
puts(string1);
```

- Here, if we were to type in “Hello there”
`string1` would be set to **“Hello there”**.

Copy String

```
int main()
{
    char s1[100], s2[100], i;
    printf("Enter string s1: ");
    scanf("%s",s1); //or gets(s1);

    for(i = 0; s1[i] != '\0'; ++i)
    {
        s2[i] = s1[i];
    }
    s2[i] = '\0';
    printf("String s2: %s", s2); // puts(s2);
    return 0;
}
```

Output:

Enter String s1: programiz
String s2: programiz

String copy function

- The first, and most basic, string manipulation library function is the string-copy function, **strcpy()**.
- This function takes two parameters, **the destination string and the source string**.
- Its prototype looks like:
strcpy(dest,source);

String: strcpy

- `strcpy(dest, source);`
- Copies characters from `source` to `dest`
 - Copies up to, and including the first `'\0'` found
 - Be sure that `dest` is large enough to hold the result!

The **strcpy** Function

- Example:

```
char string1[20],string2[20];  
strcpy(string1,"Hello there");  
strcpy (string2,string1);
```

The **strcpy** Function

- Most of the time, the array will overflow into unallocated memory.
- However, if the memory cells immediately following the array are allocated and then the **strcpy** function oversteps the array bounds, it can overwrite the variables which are allocated to these memory cells.

String Assignment: Examples

```
#include <string.h>
```

```
...
```

```
char medium[21] ;
```

```
char big[1000] ;
```

```
char small[5] ;
```

```
strcpy(medium, "Four score and seven" ) ;
```

medium: Four score and seven\0

String Assignment: Examples

```
char medium[21 ];  
char big[1000] ;  
char small[5] ;  
  
strcpy(big, medium) ;  
strcpy(big, "Bob") ;
```

big: Four score and seven\0?????...

big: Bob\0 score and seven\0?????...

String Assignment Dangers

```
char medium[ 21];
```

```
char big[1000] ;
```

```
char small[5] ;
```

```
strcpy(small, big) ;
```

```
strcpy(small, medium) ;    /* looks like trouble... */
```


small: Bob\0?

small: Four score and seven\0

The **strncpy** Function

- This problem can be resolved using the **strncpy** function.
- This function is a “counted” version of the **strcpy** function. It takes **three arguments**, the **destination string**, the **source destination**, and **then the number of characters to copy**.
- Its prototype looks like:

strncpy(dest, source, size_t size);



size_t is an unsigned integer.
Thus, it cannot be negative

The **strncpy** Function

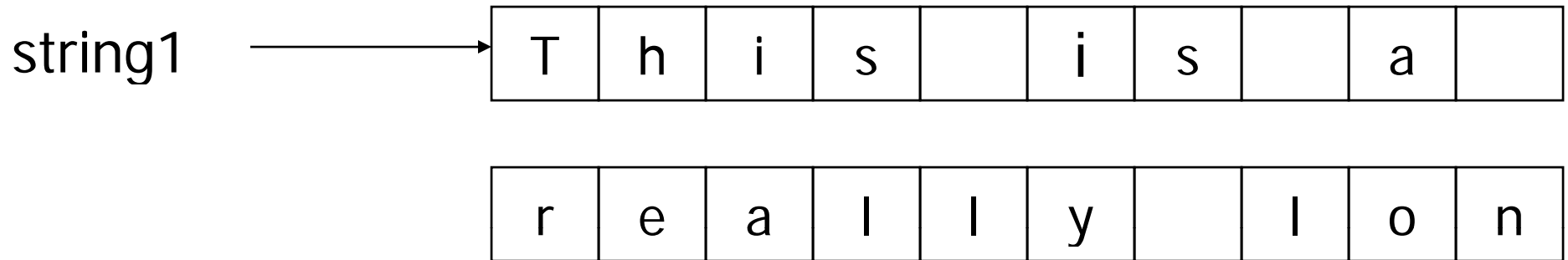
- Thus, in order to prevent copying past the bounds of the array in the previous example, we could write:

```
char string1[20];
```

```
strncpy(string1,"This is a really long string",20);
```

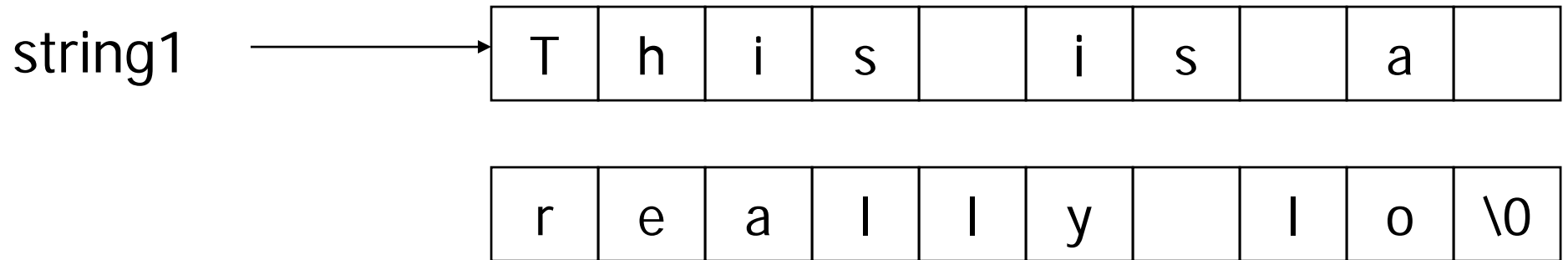
- This would only copy the first twenty characters (into index 0 to 19) and thus would not overwrite any memory past that allocated for the array.

The **strncpy** Function



- Notice that even though we no longer overwrite memory past that allocated for the string, the string is still not valid, as it does not end with a null character.

The **strncpy** Function

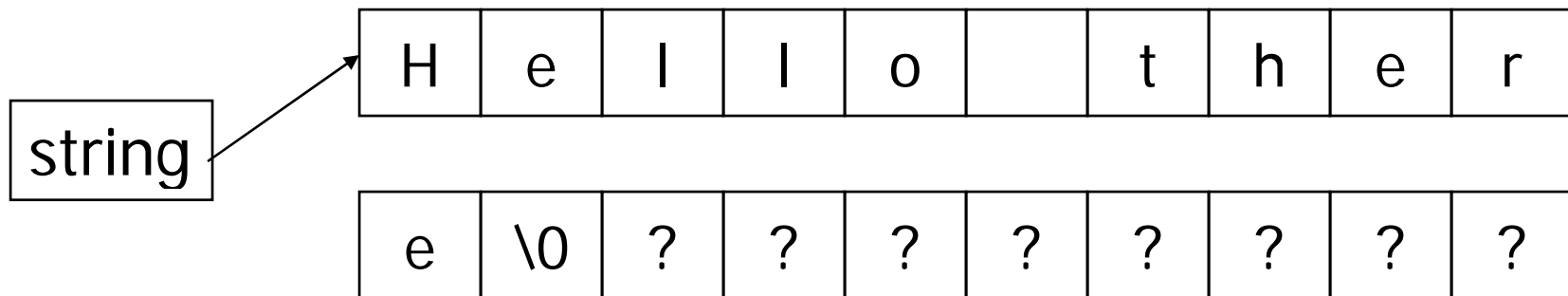


```
char string1[20];
```

```
strncpy(string1, "This is a really long string", 20);
```

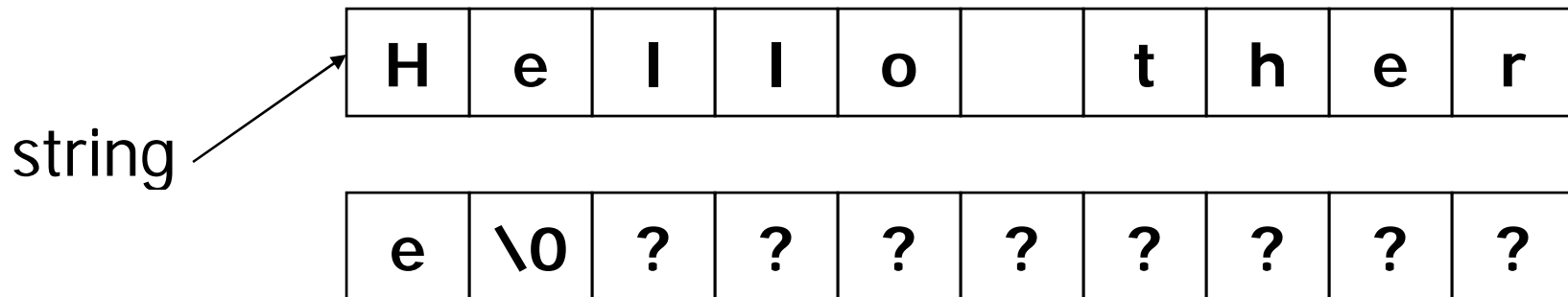
```
string1[19] = '\0';
```

Substrings



- Keep in mind that `string` evaluates to the address of the first character, or
`string = &string[0]`.
- Using this, we can see how to use a substring. For example, to reference the substring “**there**”, we could simply use `&string[6]` instead of `string`.

Substrings

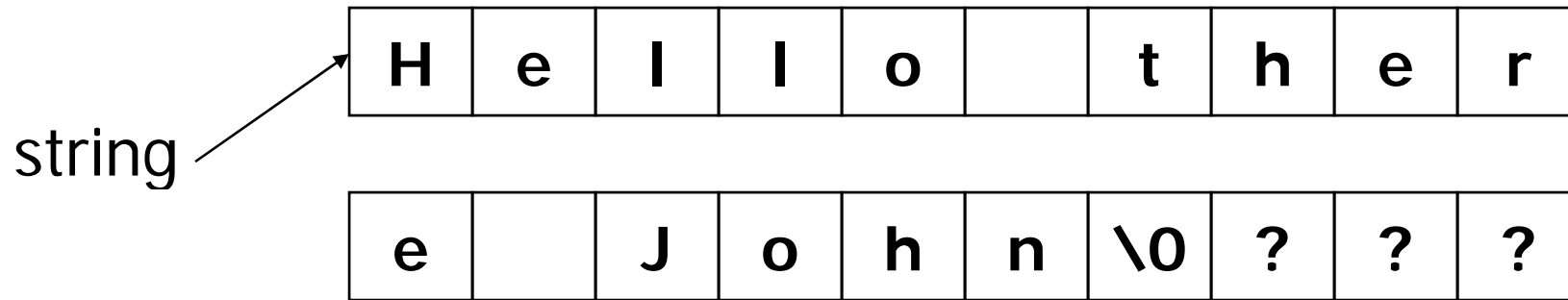


- So, what would be the output of the following code fragment, assuming string as shown above?

```
printf("%s\n",string);  
printf("%s\n",&string[0]);  
printf("%s\n",&string[6]);  
printf("%s\n",&string[2]);
```

```
Hello there  
Hello there  
there  
llo there
```

Substrings



- So, we can combine **this** and the **strncpy** function to extract a substring.
- Therefore, to extract “**there**” from string and put it in string2, we could write:

```
strncpy(string2,&string[6],5);
```

```
string2[5] = '\0';
```

- Note that we need to manually insert the null character into string2, as the data **strncpy** copies into string2 does not end with one.

String Lengths

- We need some way to find the size of the string contained in a character array (the actual number of characters present **before the null character**).

String Length: strlen

- **strlen** returns the length of its string argument
 - Does not count the null '\0' at the end
 - Examples:
 - The length of "A" is 1
 - The length of "" is 0
- `k = strlen("null-terminated string");`
stores 22 in k

A strlen implementation

```
int main()
{
    char s[1000], i;
    printf("Enter a string: ");
    scanf("%s", s);
    for(i = 0; s[i] != '\0'; ++i);
    printf("Length of string: %d", i);
    return 0;
}
```

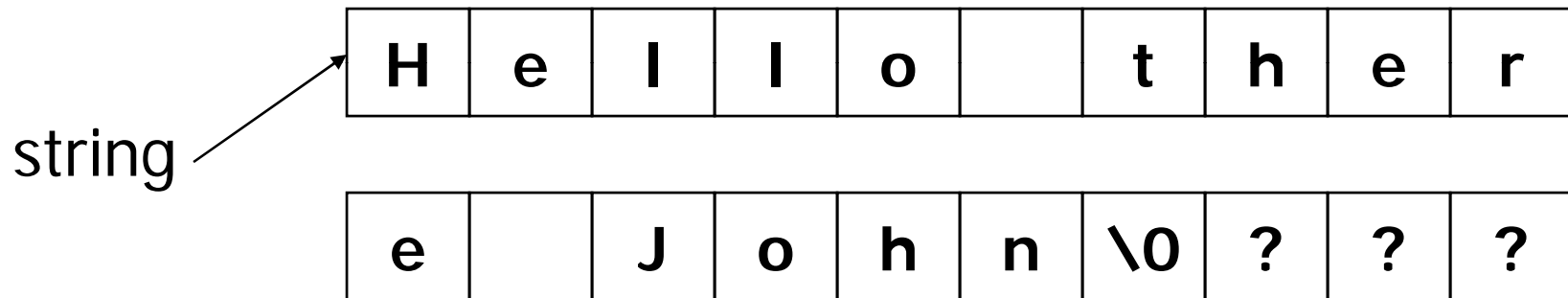
Output:

Enter a string: Program

Length of string: 7

String Lengths

- What would be the output of the following program fragment?



```
int string_size;  
string_size = strlen(string);  
printf("Size = %d\n",string_size);
```

Size = 16

String Lengths

What would be the output of the following program fragment?

```
char string[40];  
int string_size;  
strcpy(string, "I think this is a nice string.");  
string_size = strlen(string);  
printf("Size = %d\n", string_size);  
printf("%s\n", &string[18]);
```

Size = 30
nice string.

String Concatenation

- Sometimes, a program may need to append one string onto another string.
- This is known as *concatenation*, or *concatenating* the arrays.
- This is accomplished using the C library functions **strcat** and **strncat**.
- The **strncat** function is simply a “counted” version of **strcat**, just as **strncpy** is a “counted” version of **strcpy**.

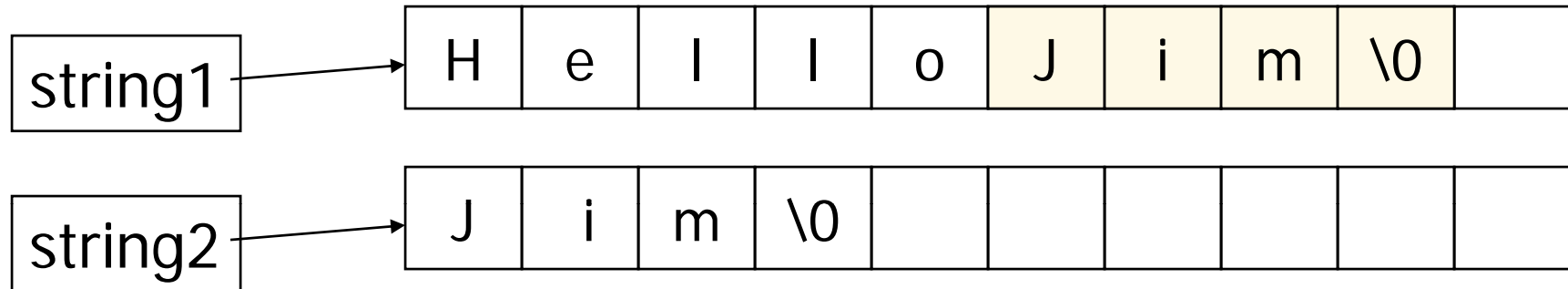
String Concatenation

- This function, **strcat**, takes two parameters, the **destination string and the source string**.
- This function takes the second string and appends it to the first string, overwriting the null character at the end of the first string with the first character of the second string.
- The **last character copied** onto the end of the first string is the **null character** of the second string.
- No bounds checking is present

String Concatenation: strcat

- To append means to place one string directly after another
- “ram” appended to “krishnan” should result in “ramkrishnan”
- *strcat(dest, source);*
- Appends characters from *source* to *dest*
 - Copy is stored starting at first ‘\0’ in *dest*
 - Copies up to, and including the first ‘\0’ in *source*
 - Be sure that *dest* is large enough!

String Concatenation



```
char string1[10] = "Hello";  
char string2[10] = "Jim";
```

```
strcat(string1, string2);
```

Concatenate Two Strings

```
int main()
{
    char s1[100], s2[100], i, j;
    printf("Enter first string: ");
    gets(s1);
    printf("Enter second string: ");
    gets(s2);
    // calculate the length of string s1
    // and store it in i
    for(i = 0; s1[i] != '\0'; ++i);
    for(j = 0; s2[j] != '\0'; ++j, ++i)
    {
        s1[i] = s2[j];
    }
    s1[i] = '\0';
    puts(s1);
    return 0;
}
```

- Enter first string: lol
- Enter second string: :)
- After concatenation: lol:)

String Comparison

- Frequently, we may wish to compare strings.
- For integers, floating-point numbers, and other such data types, we can use simple comparison operators such as:

```
int x=9, y=27;
```

```
if (x < y) { ... }
```

```
if (y < x) { ... }
```

```
if (y == x) { ... }
```


String Comparison

- If we were to do this with strings...

```
char string1[20] = "Cat.";
char string2[20] = "Dog.";
if (string1 < string2) { ... }
```

- This is a valid comparison; however it does not compare the strings "Cat." and "Dog."
- As the name of an array is the memory address of its first element, this comparison statement tests to see if the memory address string1 begins at is less than the memory address string2 begins at.

String Comparison

- Thus, C provides the **strcmp** function to compare strings.
- This function takes **two strings and returns an integer value.**

strcmp(s1, s2);

- A **negative integer** if str1 is less than str2.
- **Zero** if str1 equals str2.
- A **positive integer** if str1 is greater than str2.

String Comparison

- Every **character** (symbol) is actually a number, as defined by the **ASCII code**.
- As such, **these numbers are what is actually compared** when characters are being compared
- This is fairly transparent as “A” is less than “B” which is less than “C”, etc...
- However, it is important to realize that **“Z” is less than “a”**.

String Comparison

- The uppercase letters, in the ASCII code, have lower codes than the lowercase letters.
- Thus, if we were to call:
`strcmp("Zebra", "tiger");`
- This function would return a negative value as 'Z' is less than 't', thus "Zebra" is less than "tiger".

String Comparison

- So what would be the result of the following comparison function calls?

- `strcmp("hello", "jackson");` -2
- `strcmp("red balloon", "red car");` -1
- `strcmp("blue", "blue");` 0
- `strcmp("AARDVARK", "AARDVARKC");` -67

String Comparison

- So what would be the result of the following comparison function calls?

- `strcmp("first", "first");` 0
- `strcmp("firsta", "first");` 97
- `strcmp("first", "firsta");` -97
- `strcmp("first", "second");` -13

WAP in C to find the Frequency of Characters

```
int main()
{
    char str[1000], ch;
    int i, frequency = 0;
    printf("Enter a string: ");
    gets(str);
    printf("Enter a character to find
    the frequency: ");
    scanf("%c",&ch);

    for(i = 0; str[i] != '\0'; ++i)
    {
        if(ch == str[i])
            ++frequency;
    }
    printf("Frequency of %c = %d",
        ch, frequency);
    return 0;
}
```

▪Output:

- Enter a string: This website is awesome.
- Enter a character to find the frequency: e
- Frequency of e = 4

C program to check given string Is palindrome or not

```
int main()
{ char string1[20];
  int i, length; int flag = 0;
  printf("Enter a string:");
  scanf("%s", string1);
  length = strlen(string1);
  for(i=0;i < length ;i++)
  { if(string1[i] != string1[length-i-1])
    { flag = 1; break;
    }
  }
  if (flag)
  { printf("%s is not a palindrome", string1);
  }
  else { printf("%s is a palindrome", string1);
  }
  return 0; }
```


Reverse a string

```
int main() {  
    char str[100], temp;  
    int i, j = 0;  
    printf("\nEnter the string :");  
    gets(str);  
    i = 0;  
    j = strlen(str) - 1;  
    while (i < j) {  
        temp = str[i];  
        str[i] = str[j];  
        str[j] = temp;  
        i++;  
        j--;  
    }  
    printf("\nReverse string is :%s", str);  
    return (0);  
}
```