Arrays and Strings

Accident Problem

Each year the Department of Traffic Accidents receives accident count reports from a number of cities and towns across the country. Given details of ‘n’ days, develop an algorithm and write a program to determine the average number of accidents and for each day, print the difference between the number of accidents on that day and average. For example, if the number of accidents is 5 and the values are 10, 12, 15, 13, 5 then average is 11 and the difference of values are 1, 1, 4, 2, 6

Accident Problem

Input	Output	Logic Involved
Value of ‘n’, the number of accidents	Average and ‘n’ values that is the difference between average and value	Find average and difference

Algorithm

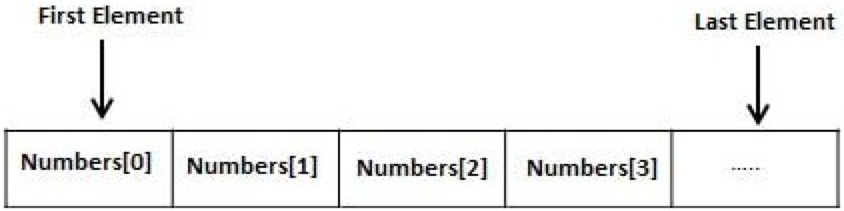
1. Read the value of ‘n’
2. Read the number of accidents happened in ‘n’ days
3. Find average
4. For each value print the difference between average and the value

New Stuff…

- We can find the difference between average and the number of accidents on a particular day only after reading all numbers from the user
- So data has to be stored
- Same type of data is to be stored
- Number of items not known prior
- Best choice would be using arrays in C
- Array - Can store a fixed-size sequential collection of elements of the same type

Arrays in C

- Consist of contiguous memory locations
- lowest address corresponds to the first element
- highest address to the last element
- Array indices start with zero
- The elements have indices from 0 to ‘n-1’
- Similar to list in Python but homogenous



First Element

Last Element

Numbers[0] Numbers[1] Numbers[2] Numbers[3] .....

Array Declaration

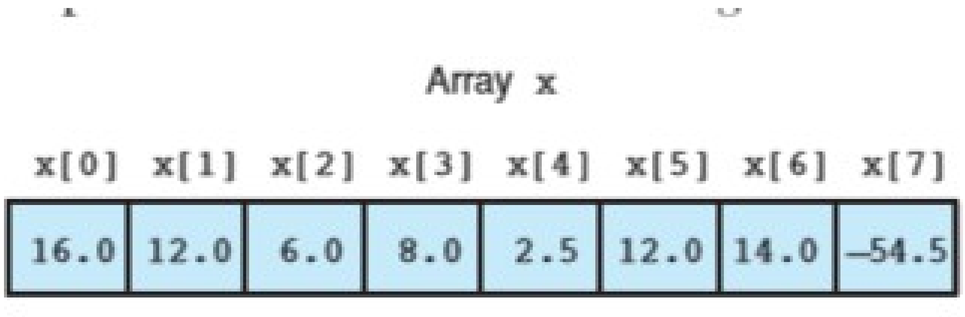
- type arrayName [arraySize];
- double balance[10];

Initializing Arrays

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

(or)

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};



Array x

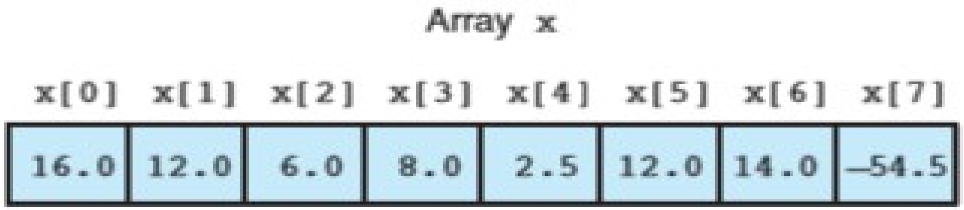
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

Difference between Assignment and Initialization

- Assignment
- int a;
- a = 5;

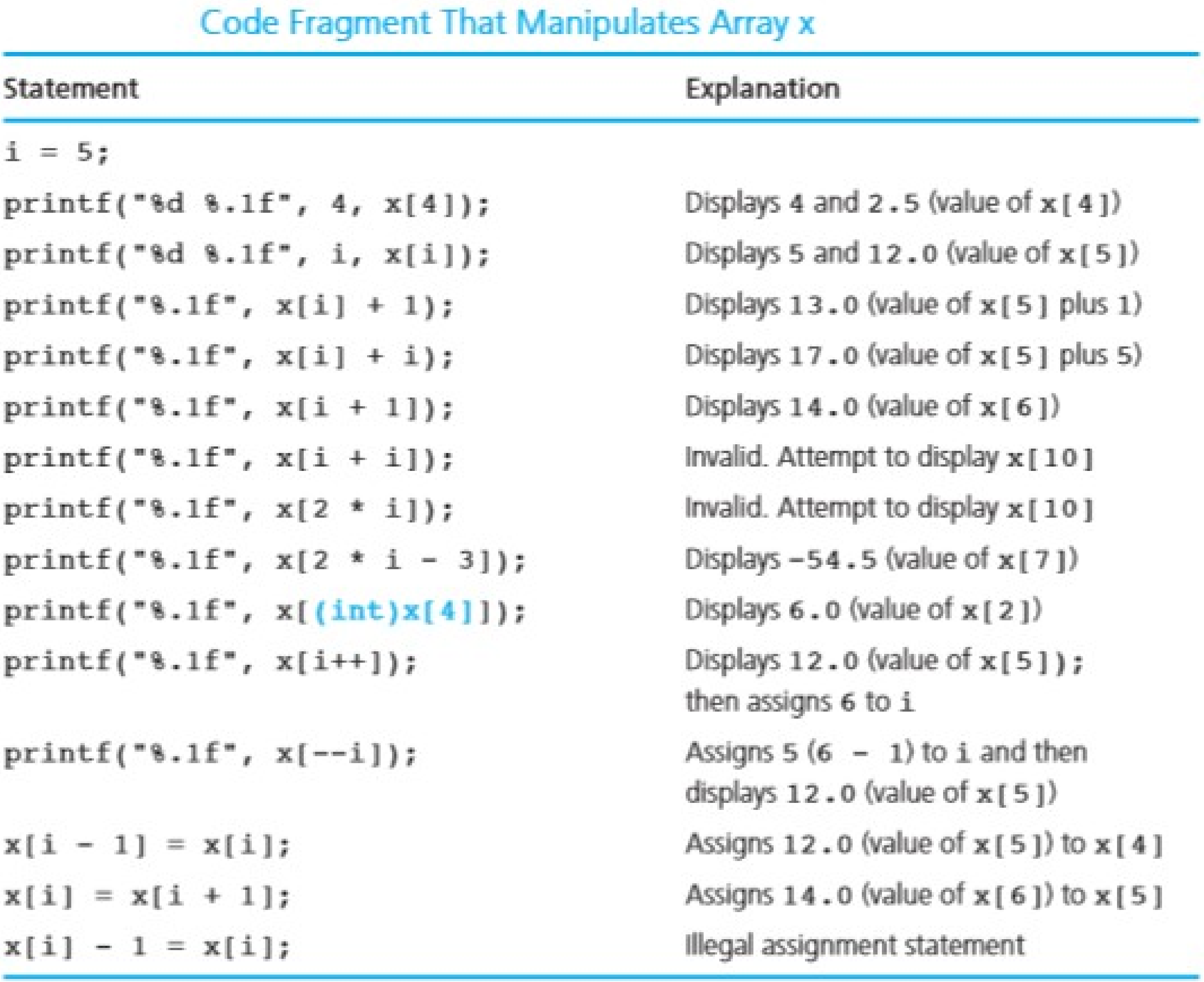
Initialization

- int a = 5;



Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	–54.5

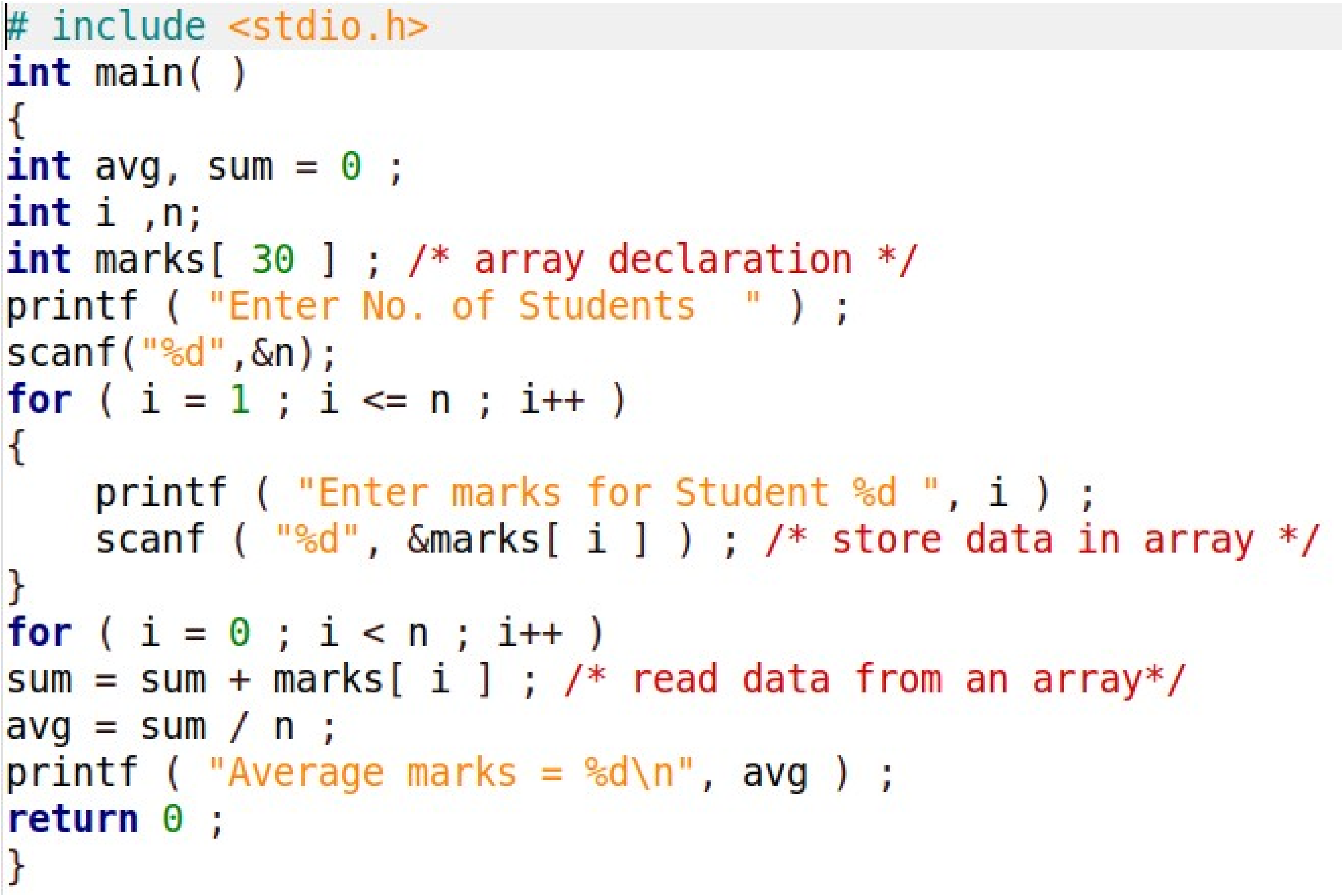


Code Fragment That Manipulates Array x

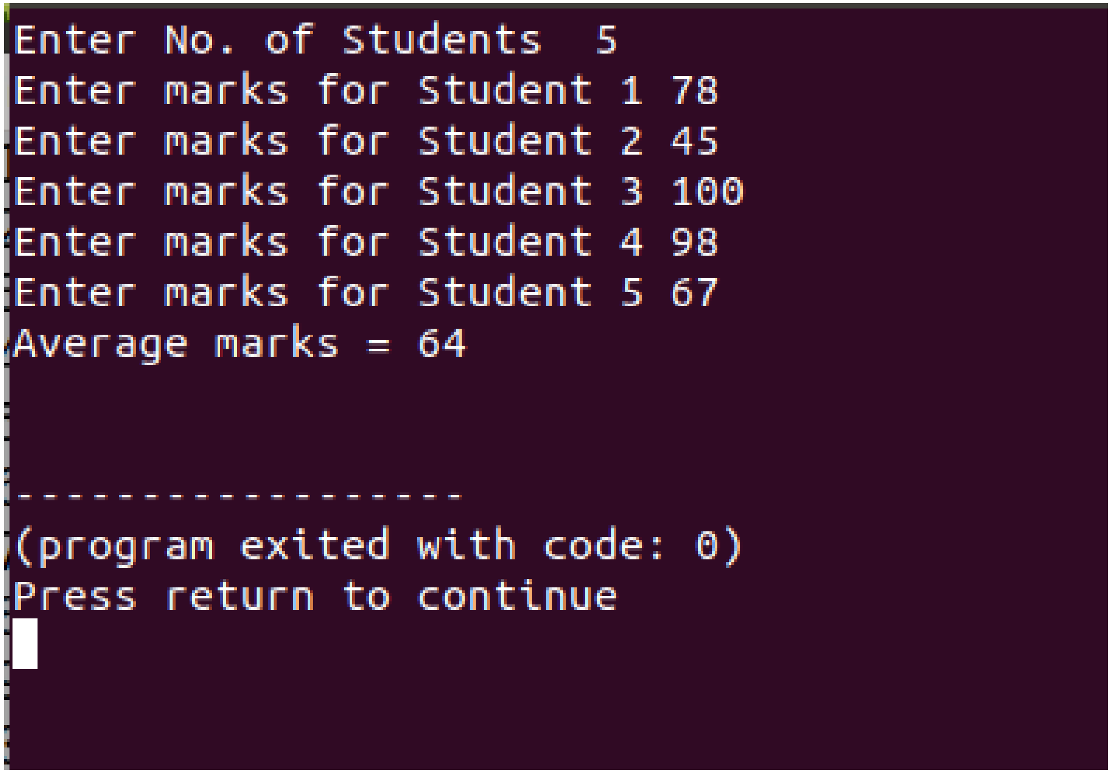
Statement	Explanation
`i = 5;`	
`printf("%d %.1f", 4, x[4]);`	Displays 4 and 2.5 (value of x[4])
`printf("%d %.1f", i, x[i]);`	Displays 5 and 12.0 (value of x[5])
`printf("%.1f", x[i] + 1);`	Displays 13.0 (value of x[5] plus 1)
`printf("%.1f", x[i] + i);`	Displays 17.0 (value of x[5] plus 5)
`printf("%.1f", x[i + 1]);`	Displays 14.0 (value of x[6])
`printf("%.1f", x[i + i]);`	Invalid. Attempt to display x[10]
`printf("%.1f", x[2 * i]);`	Invalid. Attempt to display x[10]
`printf("%.1f", x[2 * i - 3]);`	Displays −54.5 (value of x[7])
`printf("%.1f", x[(int)x[4]]);`	Displays 6.0 (value of x[2])
`printf("%.1f", x[i++]);`	Displays 12.0 (value of x[5]); then assigns 6 to i
`printf("%.1f", x[--i]);`	Assigns 5 (6 − 1) to i and then displays 12.0 (value of x[5])
`x[i - 1] = x[i];`	Assigns 12.0 (value of x[5]) to x[4]
`x[i] = x[i + 1];`	Assigns 14.0 (value of x[6]) to x[5]
`x[i] - 1 = x[i];`	Illegal assignment statement

Cannot assign one array to another

```
int ia[] = {0, 1, 2}; // ok: array of ints
int ia2[] = ia; // error: cannot initialize one array with another
int main()
{
const unsigned array_size = 3;
int ia3[array_size];
// ok: but elements are uninitialized!
ia3 = ia; // error: cannot assign one array to another
return 0;
}
```



```
# include <stdio.h>
int main( )
{
int avg, sum = 0 ;
int i ,n;
int marks[ 30 ] ; /* array declaration */
printf ( "Enter No. of Students  " ) ;
scanf("%d",&n);
for ( i = 1 ; i <= n ; i++ )
{
    printf ( "Enter marks for Student %d ", i ) ;
    scanf ( "%d", &marks[ i ] ) ; /* store data in array */
}
for ( i = 0 ; i < n ; i++ )
sum = sum + marks[ i ] ; /* read data from an array*/
avg = sum / n ;
printf ( "Average marks = %d\n", avg ) ;
return 0 ;
}
```



```
Enter No. of Students  5
Enter marks for Student 1 78
Enter marks for Student 2 45
Enter marks for Student 3 100
Enter marks for Student 4 98
Enter marks for Student 5 67
Average marks = 64



------------------
(program exited with code: 0)
Press return to continue
```

List and Array Comparison

List	Arrays
Can have mixed type of elements	Can have only one type of element
Number of elements in list need not be specified L = []	Size has to be specified during declaration int a[10];
Elements are accessed by subscript operator L[0], L[1]…	Same way a[0], a[1], a[2],…
Size is dynamic, increases when elements are added and decreases when removed	Size is static
Have predefined functions such as len, count, index etc	No such functions

Huffman Coding Problem

Huffman code is a particular type of optimal prefix code for characters. It is commonly used for lossless data compression. It is a variable-length code derived from frequency of occurrence. Given a string develop an algorithm and write a C program to determine frequency of occurrence of each character in the string.

Huffman Coding Problem

Input	Output	Logic Involved
A string S	Frequency count of each letter in S	Convert all letters to uniform case and check if it is a particular letter and increment corresponding count

Algorithm

1. Read a string
2. Make all letters in the string to be in lowercase
3. Process character by character
4. If the character is an alphabet then increment count of it
5. Print count of all alphabets

Strings

What is a pointer?

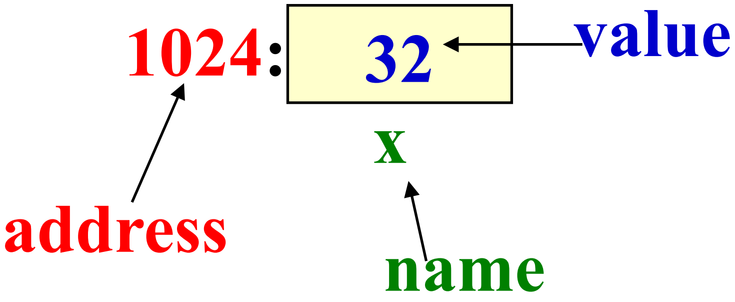
- First of all, it is a variable, just like other variables you studied

So it has type, storage etc.

- Difference: it can only store the address (rather than the value) of a data item
- Type of a pointer variable – pointer to the type of the data whose address it will store
  - Example: int pointer, float pointer,…
  - Can be pointer to any user-defined types also like structure types

Values vs Locations

Variables name memory locations, which hold values



Contd.

Consider the statement

```
int  xyz = 50;
```

- This statement instructs the compiler to allocate a location for the integer variable xyz, and put the value 50 in that location
- Suppose that the address location chosen is 1380

xyz	ㅎ	**variable**
50	ㅎ	**value**
1380	ㅎ	**address**

Contd.

During execution of the program, the system always associates the name xyz with the address 1380

- The value 50 can be accessed by using either the name xyz or the address 1380

Since memory addresses are simply numbers, they can be assigned to some variables which can be stored in memory

- Such variables that hold memory addresses are called pointers
- Since a pointer is a variable, its value is also stored in some memory location

Contd.

Suppose we assign the address of xyz to a variable p

- p is said to point to the variable xyz

Variable	Value	Address
xyz	50	1380
p	1380	2545

p = &xyz;

*p=xyz (50)

Pointers

- A pointer is just a C variable whose value can contain the address of another variable
- Needs to be declared before use just like any other variable
- General form:

```
data_type *pointer_name;
```

- Three things are specified in the above declaration:
  - The asterisk (*) tells that the variable pointer_name is a pointer variable
  - pointer_name needs a memory location
  - pointer_name points to a variable of type data_type

Example

```
int     *count;
float   *speed;
char *c;
```

Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement like

```
int *p, xyz;
    :
p = &xyz;
```

– This is called pointer initialization

Strings

- 1-d arrays of type char
- By convention, a string in C is terminated by the end-of-string sentinel ‘\0’ (null character)
- char s[21] - can have variable length string delimited with \0
  - Max length of the string that can be stored is 20 as the size must include storage needed for the ‘\0’
- String constants : “hello”, “abc”
- “abc” is a character array of size 4

Character Arrays and Strings

```
char C[8] = { 'B', 'C', 'E', '1', '0', '2', 'L', '\0' };
```

- C[0] gets the value 'B', C[1] the value 'C', and so on. The last (7th) location receives the null character ‘\0’
- Null-terminated (last character is ‘\0’) character arrays are also called strings
- Strings can be initialized in an alternative way. The last declaration is equivalent to:

```
char C[8] = "BCSE102L";
```

- The trailing null character is missing here. C automatically puts it at the end if you define it like this
- Note also that for individual characters, C uses single quotes, whereas for strings, it uses double quotes

Strings Initialization

```
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

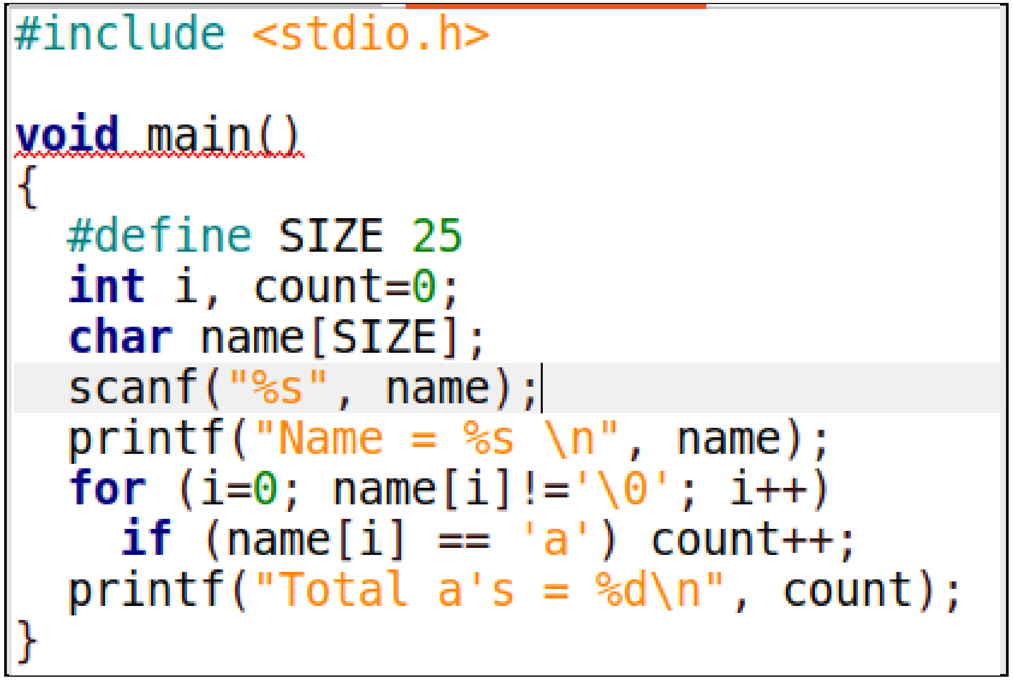
Reading strings: %s format

```
void main()
{
    char name[25];
    scanf("%s", name);
    printf("Name = %s \n", name);
}
```

%s reads a string into a character array given the array name or start address.

It ends the string with ‘\0’

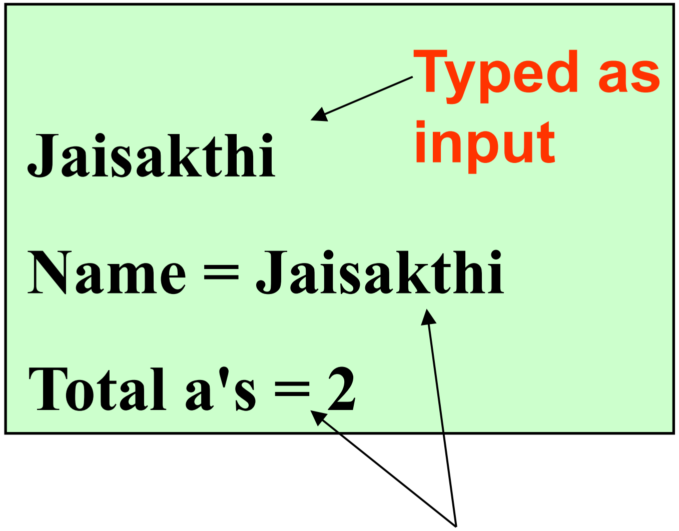
An example



```
#include <stdio.h>

void main()
{
  #define SIZE 25
  int i, count=0;
  char name[SIZE];
  scanf("%s", name);
  printf("Name = %s \n", name);
  for (i=0; name[i]!='\0'; i++)
    if (name[i] == 'a') count++;
  printf("Total a's = %d\n", count);
}
```

Seen on screen



```
Jaisakthi

Name = Jaisakthi

Total a's = 2
```

Typed as input

Printed by program

Note that character strings read in %s format end with ‘\0’

Differences : array & pointers

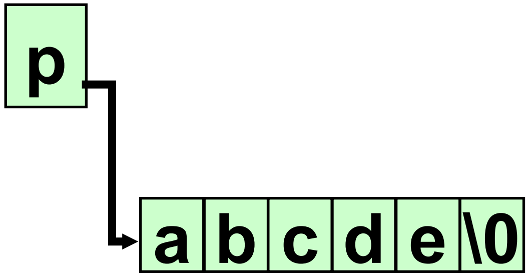
char *p = “abcde”;

The compiler allocates space for p, puts the string constant “abcde” in memory somewhere else, initializes p with the base address of the string constant

char s[ ] = “abcde”;

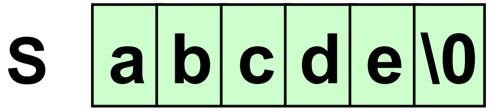
char s[ ] = **{‘a’,’b’,’c’,’d’,’e’,’\o’};**

The compiler allocates 6 bytes of memory for the array s which are initialized with the 6 characters



p

a	b	c	d	e	\0



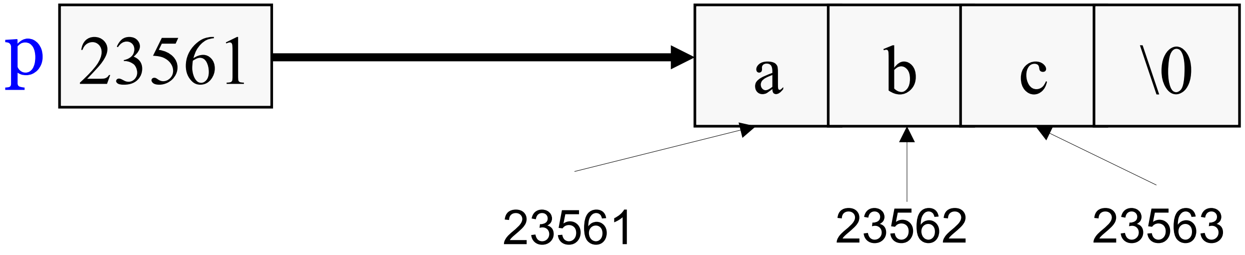
s

a	b	c	d	e	\0

String Constant

- A string constant is treated as a pointer
- Its value is the base address of the string

char *p = “abc”;



printf (“%s %s\n”,p,p+1); /* abc bc is printed */

printf ("%c %c\n",*p,*p+1);

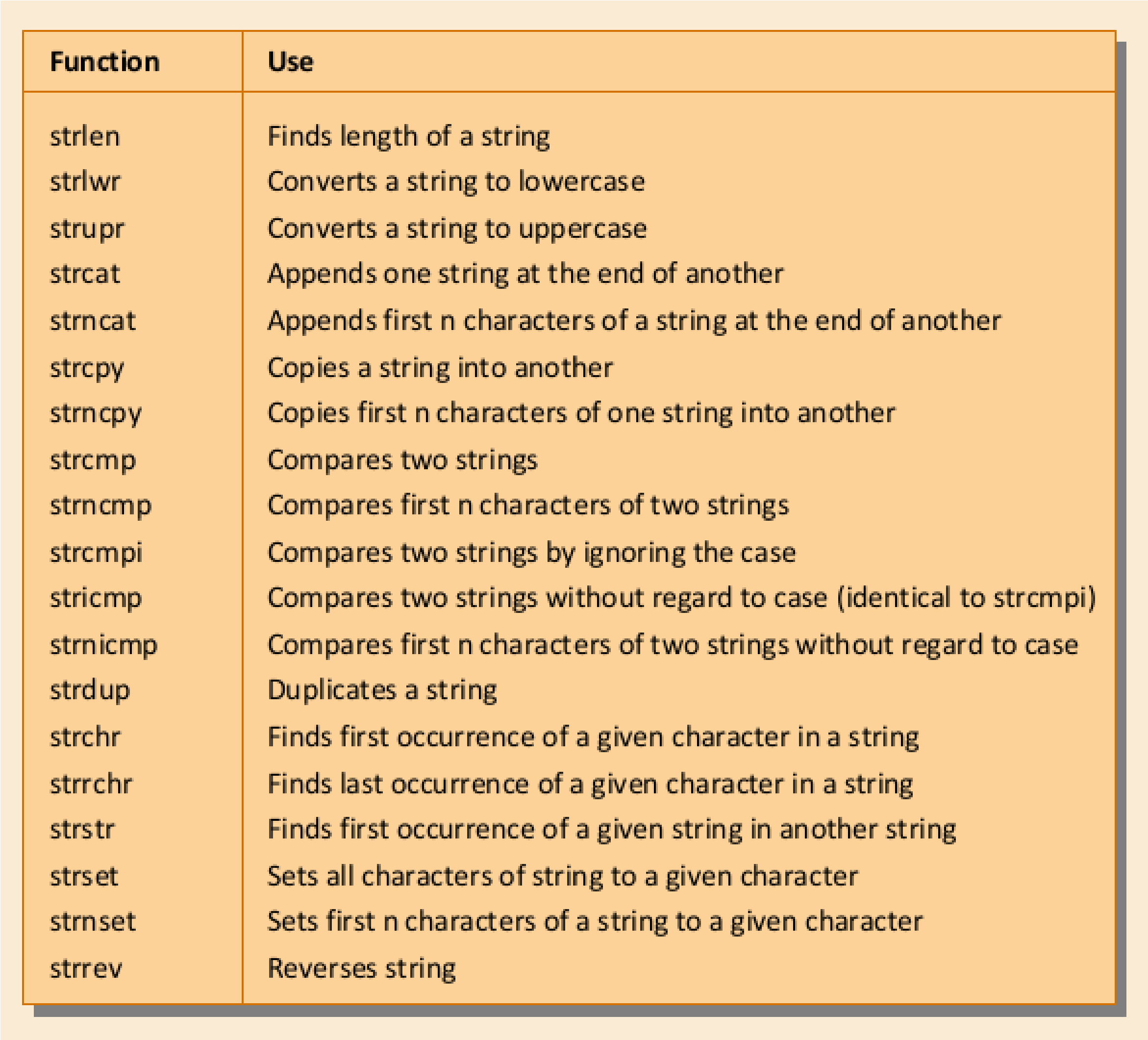
Library Functions for String Handling

- You can write your own C code to do different operations on strings like finding the length of a string, copying one string to another, appending one string to the end of another etc.
- C library provides standard functions for these that you can call, so no need to write your own code
- To use them, you must do

  #include <string.h>

  At the beginning of your program (after #include <stdio.h>)

String functions



Function	Use
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strcmpi	Compares two strings by ignoring the case
stricmp	Compares two strings without regard to case (identical to strcmpi)
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

String functions

strlen : finds the length of a string

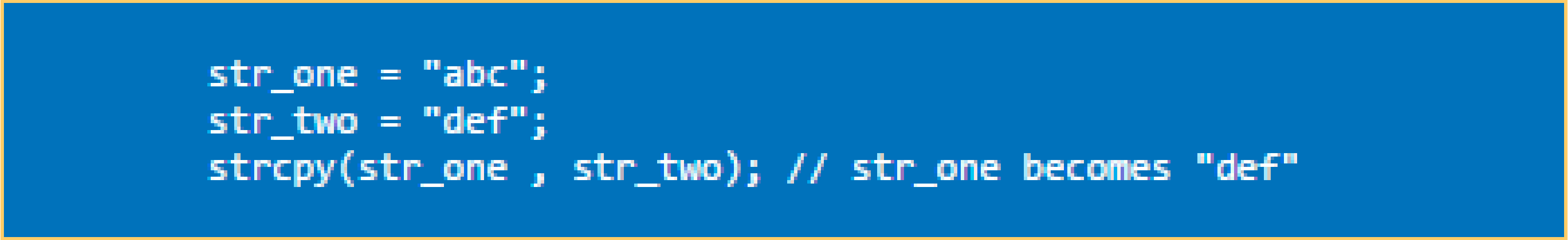
strcat : concatenates one string at the end of another

strcmp : compares two strings lexicographically

strcpy : copies one string to another

strcpy

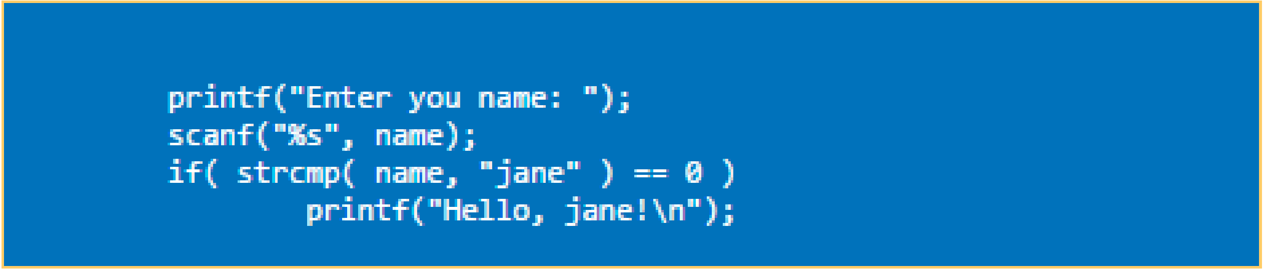
- used to copy a string and can be used as strcpy(destination, source)
- Will not perform any boundary checking, and thus there is a risk of overrunning the strings



```
str_one = "abc";
str_two = "def";
strcpy(str_one , str_two); // str_one becomes "def"
```

strcmp

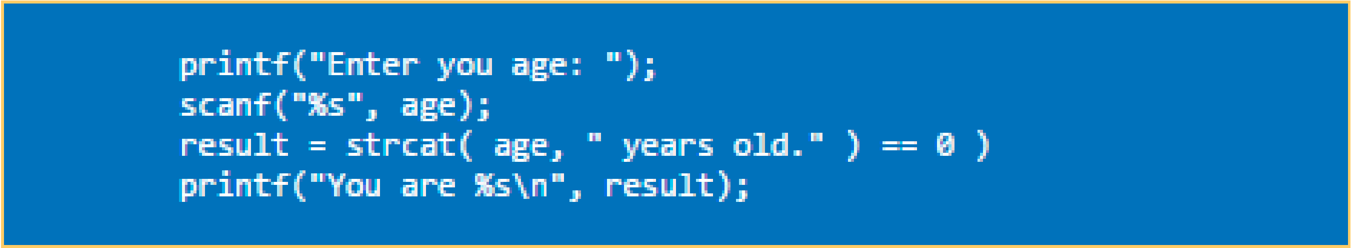
- used to compare two strings and can be used as strcmp(str1, str2)
- If the first string is greater than the second string a number greater than 0 is returned.
- If the first string is less than the second string a number less than 0 is returned.
- If the first and the second string are equal 0 is returned.



```
printf("Enter you name: ");
scanf("%s", name);
if( strcmp( name, "jane" ) == 0 )
        printf("Hello, jane!\n");
```

strcat

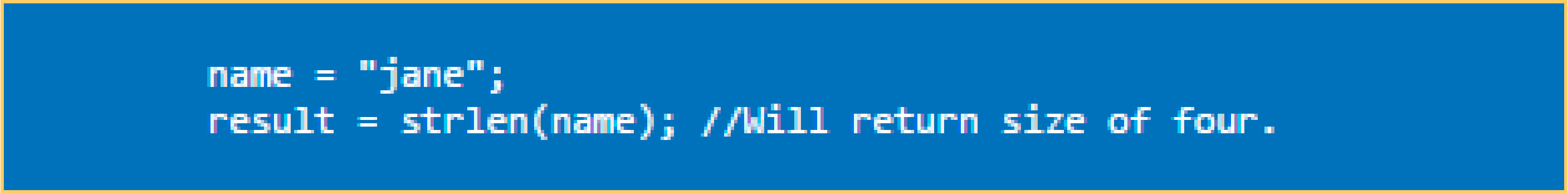
- concatenates a string onto the end of the other string and the resultant string is returned
- strcat() will not perform any boundary checking, and thus there is a risk of overrunning the strings.



```
printf("Enter you age: ");
scanf("%s", age);
result = strcat( age, " years old." ) == 0 )
printf("You are %s\n", result);
```

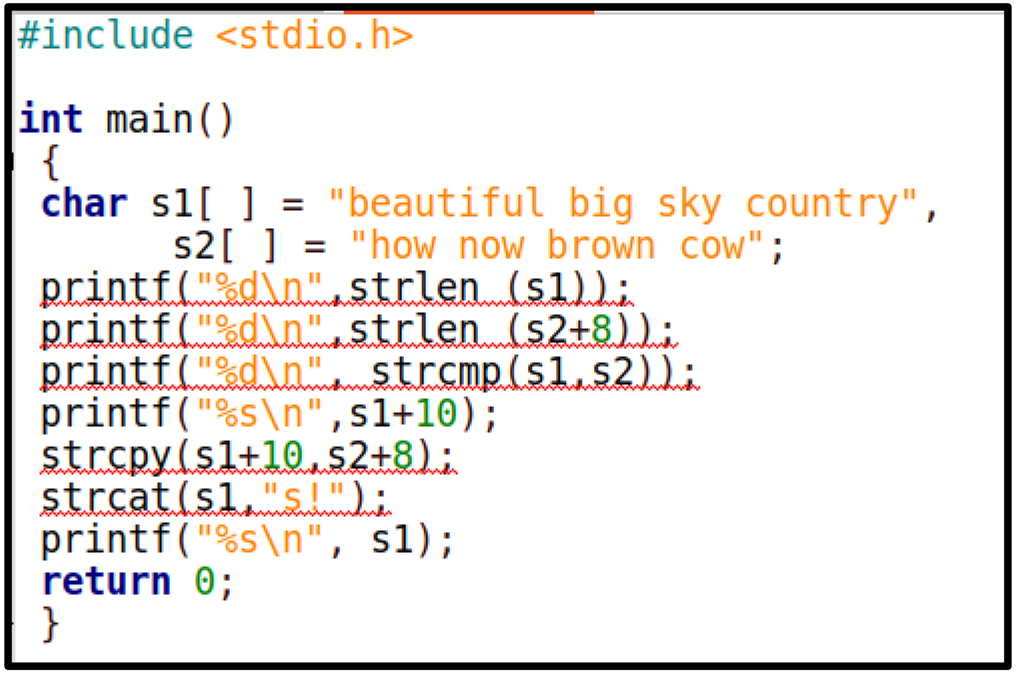
strlen

- returns the length of a string
- All characters before the null termination



```
name = "jane";
result = strlen(name); //Will return size of four.
```

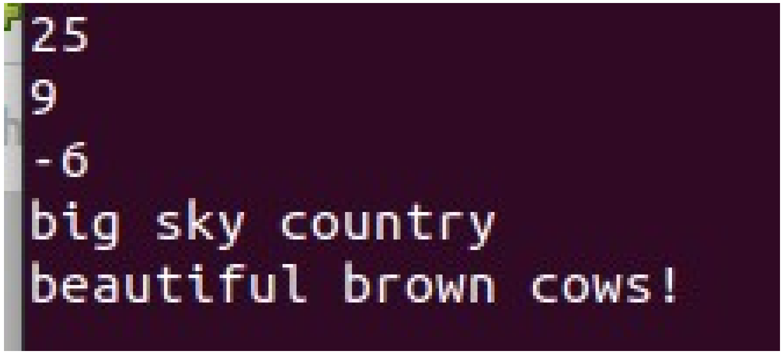
Example using String Functions



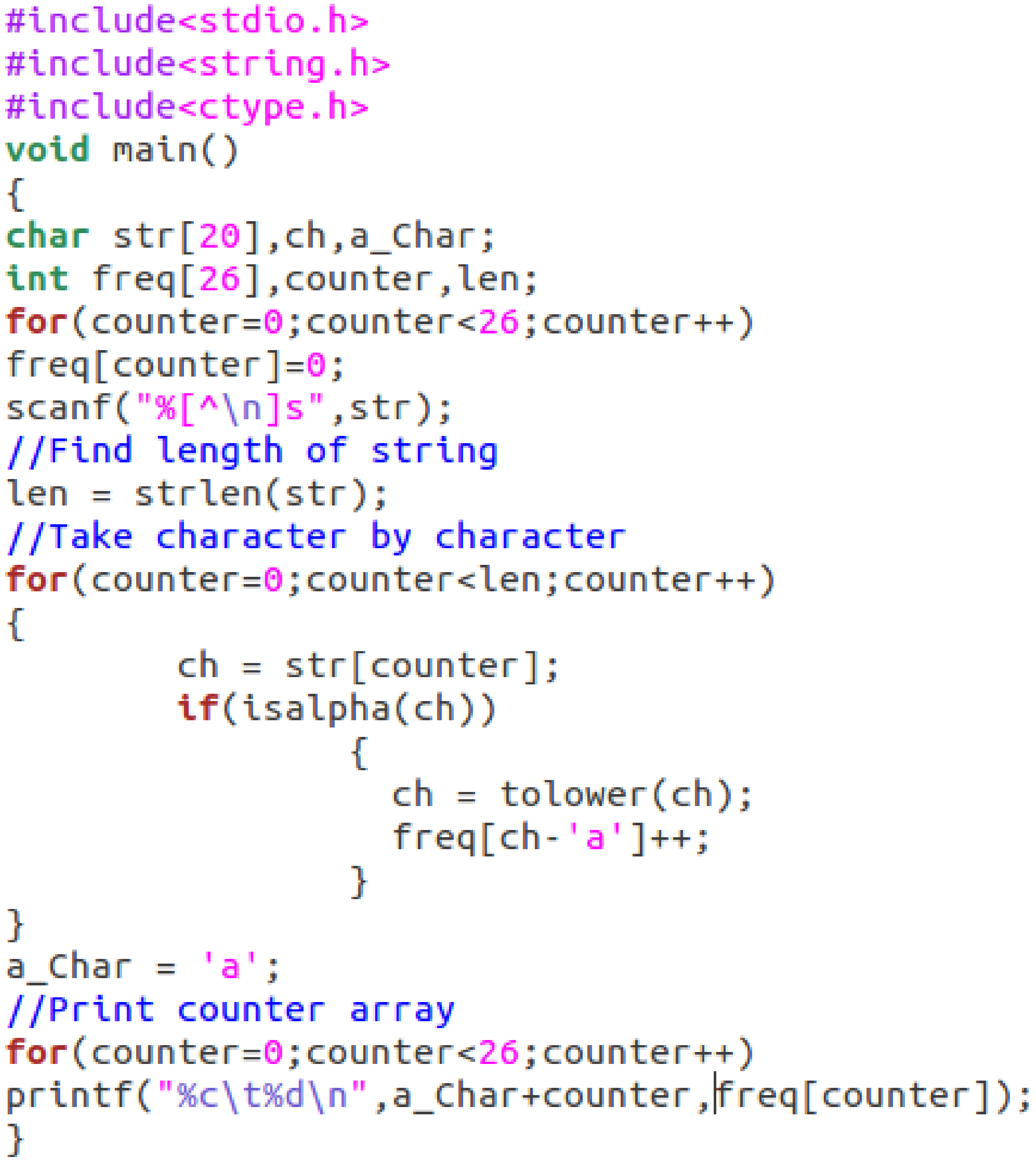
```
#include <stdio.h>

int main()
 {
 char s1[ ] = "beautiful big sky country",
       s2[ ] = "how now brown cow";
 printf("%d\n",strlen (s1));
 printf("%d\n",strlen (s2+8));
 printf("%d\n", strcmp(s1,s2));
 printf("%s\n",s1+10);
 strcpy(s1+10,s2+8);
 strcat(s1,"s!");
 printf("%s\n", s1);
 return 0;
 }
```

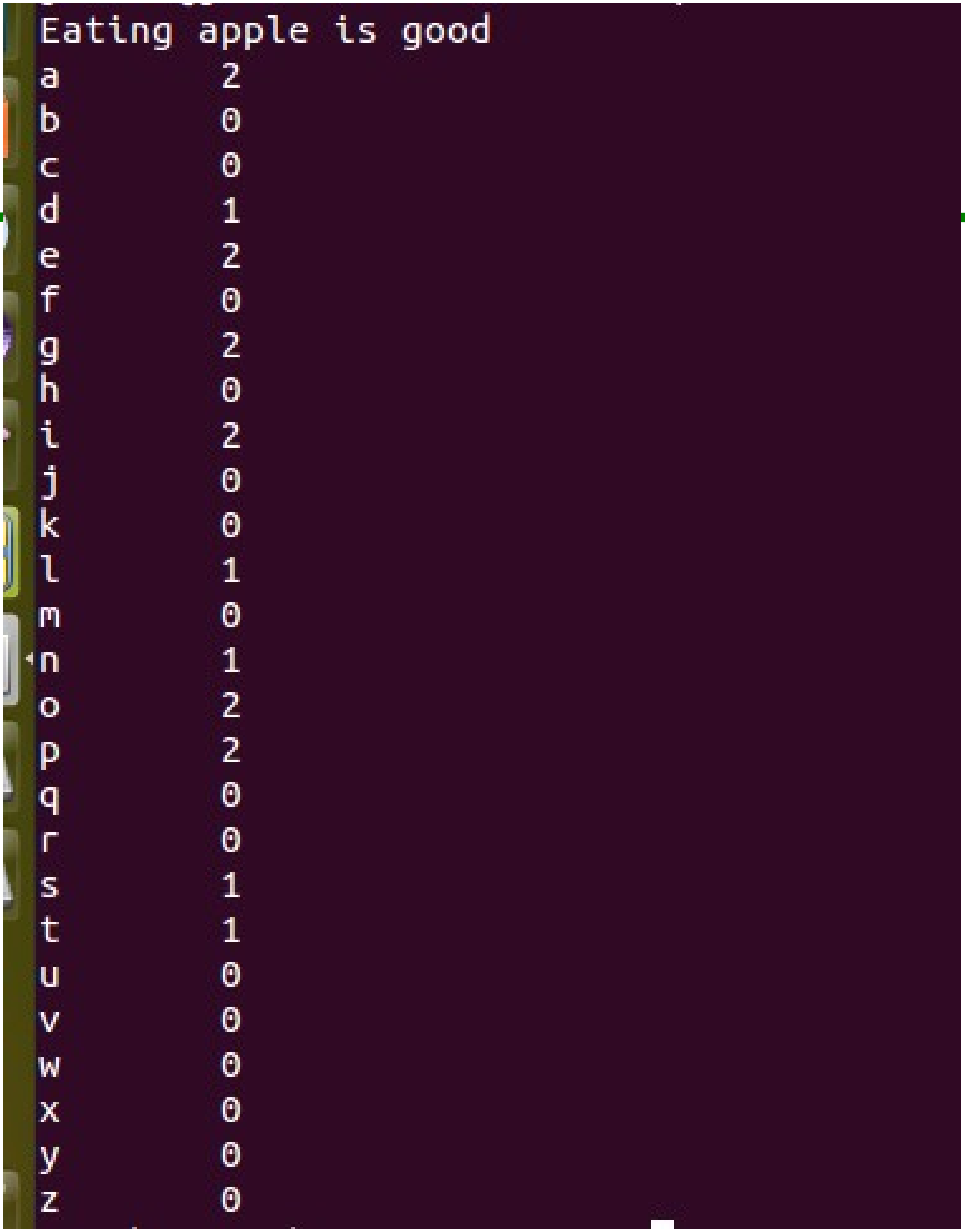
Output



```
25
9
-6
big sky country
beautiful brown cows!
```



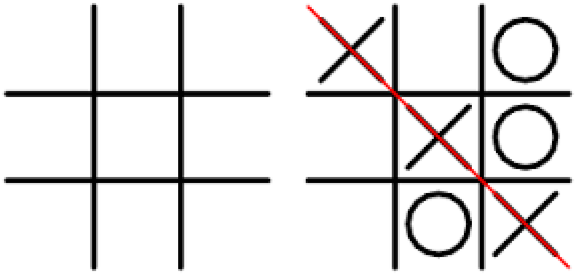
```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void main()
{
char str[20],ch,a_Char;
int freq[26],counter,len;
for(counter=0;counter<26;counter++)
freq[counter]=0;
scanf("%[^\n]s",str);
//Find length of string
len = strlen(str);
//Take character by character
for(counter=0;counter<len;counter++)
{
        ch = str[counter];
        if(isalpha(ch))
                {
                  ch = tolower(ch);
                  freq[ch-'a']++;
                }
}
a_Char = 'a';
//Print counter array
for(counter=0;counter<26;counter++)
printf("%c\t%d\n",a_Char+counter,freq[counter]);
}
```



```
Eating apple is good
a       2
b       0
c       0
d       1
e       2
f       0
g       2
h       0
i       2
j       0
k       0
l       1
m       0
n       1
o       2
p       2
q       0
r       0
s       1
t       1
u       0
v       0
w       0
x       0
y       0
z       0
```

Tic Tac Toe Problem

Tic-tac-toe is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. Player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



Tic Tac Toe Problem contd…

Given the board configuration of the tic tac toe game, determine if the board is in either of the following states: empty, player1 wins, player2 wins, draw or intermediate. The board is said to be in initial state if all the cells contain ‘-1’, player1 uses ‘1’ as his coin and player2 uses ‘2’ as his coin. The game is draw when the board is full and no one has won the game. The game is in intermediate state when no one has won and board is not full

Tic Tac Toe Problem

Input	Output	Logic Involved
Current board configuration	State of the board as win, draw, initial or intermediate	Find average and difference

Algorithm

- Represent the board in memory
- Get the elements in first row, second row and so on
- Process the elements
- If all are -1 then print ‘empty’
- If ‘1’ is placed row wise, column wise or diagonally then print ‘Player 1’ wins
- If ‘2’ is placed row wise, column wise or diagonally then print ‘Player 2’ wins
- If all cells are full and no one has won the game then print ‘Draw’
- Otherwise print intermediate

Yet to learn 2D Arrays