int n = 10;

int x = 5;

int index = 0;

int a[] = {12, 7, 3, 71, 2, 43, 38, 23, 45, 22};

int b[n];

for (int i=0; i<n; i++) {

printf("%d ", a[i]+x);

}

The problem of this statement is wrong variable declaration

char s = “america”

The right variable declaration of a string data type is :

char s[] = “america”



#include<stdio.h>

int main()

{

int i, a=1, temp=1, arr[15];

printf("%d ", a);

for(i=1; i<15; i++){

a = temp;

temp = a \* 2;

arr[i] = a \* 2;

printf("%d ", arr[i]);

}

return 0;

}



char a[10];

a[0] = 98;

a[1] = 97;

a[2] = 'n';

a[3] = 'a';

a[4] = 'n';

a[5] = 'a';

a[6] = ‘\0’;

Here, declared string variable named a which size is 10 and data type is char means characteristics data type.

But in 2nd & 3rd statement its value stored with integer data type which will convert in char by the help of ASCII values. So the 97 value will here will work as ASCII value and convert in its character value.

As string format we use “%S” so the 97 or 98 the integer value will convert in its character value automatically. Which main reason is 97 or 98 as a variable value is an integer value and its not a sttirng value. String value only can be character values. So integer value and character value are not same they are not same data type in c language. Thats why it wont be a problem while printing the whole string together. But if we want to print the 2nd n 3rd statement we will see that the value changed in character value following the ASCII value.



The output of the character array a is = ♠♣♦♥☻☺☻

We can see that, there is a for loop which will iterate total 8 times.

In every interation time the index of the array a is pointing by loop’ s “i” value. And the a[i

] = ch + 8 - i the value is addressing in the arry by the index. Here char ch = ‘a’;

The value of ch is setted with a string value but its ascii value is 0. Because its declared by the coder. So in a[i] statement ch + 8 - i will work as 0 + 8 - i, and here i will be the values from 0 to 7 for the loop iteration.

And finaly the a[i] value will be from 1 to 8 and the value of 1 to 8 will convert into this ♠♣♦♥☻☺☻

string from index 0 to 7 of the array a. The last index which is index 8 will be ended with the null value.



An int type array with 12 elements will take 12 \* 4 = 48 bytes memory in c language.

Since integer type data occupy 4 bytes per variable value.. Array with only one element will also cost the same memory. But while array have more than 1 elements it will multiply by 4 in its size.

Such, if an int type array has 12 elements, it will took 12 in 4 times memory in bytes.

we can check this by printing the size of an arry in c language by this :

Int array[12];

printf(“%d”, sizeof(array));

This output will be 48.

12 elements \* bytes 4 per elements = 48 bytes.

1. . A lexicographic order is an arrangement of characters, words, or numbers in alphabetical order, that is, the letters are sorted from A-Z. This is also known as dictionary order because it is similar to searching for a particular word in an actual dictionary. We start by searching for the section containing the words starting with the first letter of the desired word. From that section, we search for the words that contain the second letter of the desired word, and so on.

Lexicographic ordering is putting types that have more than one element in a set order, like a dictionary. If the first item in sequence A is smaller than the first item in sequence B, then sequence A is lexicographically smaller than sequence B. Also, lexicographically, A is bigger than B if the first item in A is bigger than the first item in B. If the first element of A is the same as the first element of B, then the lexicographic order of A and B is determined by the lexicographic order of A's sequence and B's sequence without the first element. A sequence that is not empty is smaller than a sequence that is not empty.

In the above example, we initially had a list of prime numbers: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}. After lexicographically arranging these numbers, the list becomes: {11, 13, 17, 19, 2, 23, 29, 3, 5, 7}.

We can order strings in Lexicographic order in c by using the indexs of the character from the string and by

the help of ASCII values



ASCII (American Standard Code for Information Interchange) is the most common character encoding format for text data in computers and on the internet. In standard ASCII-encoded data, there are unique values for 128 alphabetic, numeric or special additional characters and control codes.

If we want to print the ASCII value of a character in c we need to use %d format specifier instead of %c

Like this printf(“%d”, a/b/A/B). This will print the character’s integer ASCII value.



Bubble sort is a simple way to put a list of numbers or other things in the right order. The method works by looking at each set of elements in the string that are next to each other from left to right and switching their positions if they are not in the right order.

The sequence of these integers 12, 7, 9, 1, 3, 73, 11, 15, 62, 19, 4 if we run Bubble sort for only 5 iterations will be :

12, 7, 9, 1, 3, 73, 11, 15, 62, 19, 4

1st iteration : 7, 9, 1, 3, 12, 11, 15, 62, 19, 4, 73

2nd iteration : 7, 1, 3, 9, 11, 12, 15, 19, 4, 62, 73

3rd iteration : 1, 3, 7, 9, 11, 12, 15, 4, 19, 62, 73

4th iteration : 1, 3, 7, 9, 11, 12, 4, 15, 19, 62, 73

5th iteration : 1, 3, 7, 9, 11, 4, 12, 15, 19, 62, 73



How can you find the sum of digits of a number?

By using mod and extracting the last digit every time and add them.

#include<stdio.h>

int main()

{

int n, t, s = 0, rem;

scanf("%d", &n);

t = n;

while (t != 0){

rem = t % 10;

s += rem;

t = t / 10;

}

printf("%d", s);

return 0;

}



#include <stdio.h>

int main()

{

int i, j, arr[3][3], sum, temp, p\_d=0, s\_d=0, c\_sum;

for(i=0; i<3; i++){

temp = 0;

for(j=0; j<3; j++){

scanf("%d", &arr[i][j]);

temp += arr[i][j];

}

}

for(i=0; i<3; i++){

sum = 0;

c\_sum = 0;

for(j=0; j<3; j++){

sum += arr[i][j];

c\_sum += arr[j][i];

if(i == j){

p\_d += arr[i][j];

}

if(i+j == 2){

s\_d += arr[i][j];

}

}

if(c\_sum != sum){

printf("NO");

break;

}

}

if(p\_d == s\_d && p\_d == sum){

printf("YES");

}

else{

printf("NO");

}

return 0;

}