**Q-1. Update the following code to swap the contents of two arrays of equal size. *[You can only write code inside the loop (in the section marked “Write Code Here”). Do not modify anything else.]***

**int n = 10;**

**int a[n], b[n];**

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

**{**

**//Write Code Here**

**}**

**Ans:**

int n = 10;

int a[n], b[n];

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

{

//Write Code Here

}

**Q-2. char s[ ] = “banana”;**

**How many bytes does the string s need in memory?**

**Ans:**

Here, the string “banana” has six characters.

We know, when the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at the end of the string. Therefore, the size of the above character string should be at least 7 (6 for characters and 1 for null terminator).

It is also known to us that each character is usually stored in 8 bits or 1 byte of internal memory storage. Therefore, the above string s[7] needs (1x7)= 7 bytes of memory.

**Q-3. Suppose you want to declare an array with the elements 1, 4, 7, 10, 15, 18, 5, 10. Show how you can declare this array.**

**Ans:**

**Here, the number of elements is equal to 8. Therefore, the minimum size of the array should be 8. The general format for declaration of an array is**

**data type array-name[size];**

**If we choose the array-name as “a”, for the above 8 nos integer type data elements the array can be declard as follows:**

int a[8];

The array can be initialized during declaration as

int a[8] = {**1, 4, 7, 10, 15, 18, 5, 10}**;

It can also be declared as

int a[] = {**1, 4, 7, 10, 15, 18, 5, 10}**;

**Q-4. To declare a string, Abul uses the following code. What is wrong with it? Fix it by adding one line only.**

**char a[10];**

**a[0] = 'b';**

**a[1] = 'a';**

**a[2] = 'n';**

**a[3] = 'a';**

**a[4] = 'n';**

**a[5] = 'a';**

**Ans:**

**In the above example, initialization has been done separately after declaration which is not allowed.**

**In character array, if an array is initialized, it must be initialized during declaration. Otherwise, it cannot be initialized later. Though initialization is not mandatory, it is a good practice to initialize an array during declaration. Otherwise, it can take some garbage value in some cases.**

**The above string can be declared as**

char a[10] = “banana”;

It can also be declared as

char a[10] = { ‘b’, ’a’, ’n’, ’a’, ’n’, ’a’, ’\0’ };

**Q-5. Consider the following code.**

**char a[10];**

**gets(a);**

**What happens when a user inputs “1234567890”. How can you fix this problem?**

**Ans:**

We know, in any string array there must be a null (‘\0’) terminator at the end of the string. Therefore, in the given array (char a[10]), a maximum of (10-1)= 9 characters can be taken as input if we reserve a space for null (‘\0’) character. If any user gives an input of a string which is greater than 9 characters long in the above array, it may create some complications. The compiler may run and take the input, but there might be some runtime error. Compiler may report some error like-

\*\*\* stack smashing detected \*\*\*: terminated

Aborted (core dumped)

Process returned 134 (0x86) [*Ideally Process returned should be 0 (0x0).]*

It may create **Buffer Overflow** which may cause severe complications. Due to Buffer Overflow, program may crash. The security code of a program or even of a server may be exploited finding the loop-hole created due to this buffer overflow. A server may be hacked or crashed and anything can be done by taking control of the server. That is why, it is essential to prevent buffer overflow.

We can prevent buffer overflow by using fgets() function instead of gets() or scanf() functions as below:

char a[10];

fgets(a, 10, stdin);

where, the 1st parameter is the name of the string array,

the 2nd parameter is the size of the string array, and

the 3rd argument is the input media, from where the input will be taken, which may be a file or standard input-output etc.

Here, fgets(a, 10, stdin) will take input of (10-1)= 9 characters from the standard input-output console.

**Q-6. How much bytes does an int type array with 100 elements take in memory?**

**Ans:**

**In general, an integer type data occupy 2 bytes of memory. Therefore, an int type array with 100 elements should take a total memory of (2x100)= 200 bytes.**

**Ideally, integers occupy one word of storage, and since the word sizes of machines vary, the size of an integer that can be stored depends on the computer. In a 32-bit machine, each integer type data element can occupy 4 bytes of memory. Therefore, an int type array with 100 elements should take a total memory of (4x100)= 400 bytes. in a 32-bit machine.**

**Q-7. What does lexicographical comparison mean? Explain with examples. How can we order strings in lexicographic order in C?**

**Ans:**

Lexicographical order is the way of ordering words based on the alphabetical order of their component letters. It is also known as lexical order, dictionary order, and alphabetical order. It is similar to the way we search for any word in the dictionary. We start our search by simply searching for the first letter of the word. Then we try to find the second letter and so on. The words in the dictionary are arranged in lexicographic order.

For example: The word “apple” will come before “banana” as the first letter a comes before b. But in case of “apple” and “angry” the first letter is the same. So, we should compare the second letter. As n comes before p, “angry” will be found before “apple”. In case of “apple” and “apply”, the first 4 letters are same. So, we should compare the fifth or last letter. Here, “apple” must be ordered before “apply” as e comes before y.

Two string can lexicographically ordered in the following way

#include <stdio.h>

#include <string.h>

int main()

{

char str1[10]= "apple";

char str2[10]= "apply";

int cmp= strcmp(str1, str2);

if(cmp==0)

{

printf("The strings are equal");

}

else if (cmp > 0)

{

printf("%s is lexicographically greater than %s", str1, str2);

}

else

{

printf("%s is lexicographically smaller than %s", str1, str2);

}

return 0;

}

**Q-8. What is a null string in C? How to declare it?**

**Ans:**

Null string is a string which has no string value. During declaring a character array, if we do not initialize it, it can act as a null string. A null string can be declared by the following ways:

char str[size];

or char str[size]=“”;

or char str[size]={};

or char str[size]={0};

or char str[size]={’\0’};