QUESTIONS

Sr. no.	Question	Options	Answer & Explanation
1	For the following statements, would arr[3] and ptr[3] fetch the same character? char arr[]=”Surprised” char *ptr=”Surprised”	A. Yes B. No	**Answer: A** **Explanation:** Both pointers and array can be used to store a string.
2	What will be the result of assigning a value to an array element whose index exceeds the size of the array?	A. Compilation error B. The element is initialized to 0. C. The compiler will automatically increase the size of the array. D. The program may crash if some important data gets overwritten.	**Answer: D** **Explanation:** The program may crash is the correct answer.But modern compiler will take care of this kind of situations.
3	Are ++*ptr and *ptr++ same ?	A. Yes B. No	**Answer:** B Explanation: ++*ptr increments the value being pointed to by ptr. *ptr++ means grab the Value of (*ptr) and then increment it.
4	What does the array ‘arr’ refer to? int* arr[8];	A. Array of integers B. Array of integer pointers	**Answer: B** **Explanation:** It is an array which can store integer pointers

		C. A pointer to an array of integers D. Any of above.	
5	What is the output of the following code snippet? 1. int x = 5, y = 15; 2. int * p1, * p2; 3. p1 = &x; 4. p2 = &y; 5. *p1 = 10; 6. *p2 = *p1; 7. p1 = p2; 8. *p1 = 20; 9. printf("%d %d",x,y);	A. 20 20 B. 10 10 C. 10 20 D. 20 10	**Answer:C** **Explanation:** In line 5, *p1 = 10; so the value of variable x is changed to 10. In line 6, *p2 = *p1 → value of variable y is changed to 10 In line 7, p1 = p2 → pointer p1 points to variable y now In line 8, *p1 = 20 → value of variable y is now changed to 20
6	What is a "void" pointer?	A. Variable with data type as "void" B. Pointer returning variable of type "void" C. It has no associated data type D. It can hold address of any datatype	**Answer: C,D** **Explanation:** A void pointer is a pointer that has no associated datatype with it.It can hold address of any datatype and can be typcasted to any datatype.

7	What will be the output of the following program?
	#include<stdio.h>
	int main(){
	
	char *name="INDIA";
	int x;
	char *cptr = name;
	while(*cptr != '\0')
	{
	cptr++;
	}
	x = cptr - name;
	printf("%d", x);
	return 0;
	}
		A. 3
		B. 4
		C. 5
		D. 6
		E. compilation error
			Answer: C
			
			Explanation:
			Program is calculating string length using pointer.

8	What will be the output of the program?
	#include<stdio.h>
	
	main()
	{
	int a[3]
	[4]={1,2,3,4,4,3,2,1,1,3,4,1};
	printf("%d",*(*(a+1)+2));
	
	}
		A. 1
		B. 2
		C. 3
		D. 4
			Answer:B
			
			Explanation:
			a:- base address of multidimensional array
			(a+1) :- increments the value of array pointer by 1 that in turn points to row 2 of array(property of multidimensional array pointer as it points to array of pointers(which are pointing to 1D arrays)).
			(*(a+1)+2) now points to exact same location as a[1][2].

9	What will happen when the following program is compiled and run?
	
	#include<stdio.h>
	int main(){
	
	int a[]={1,2,3,4,5};
	int b[]={1,2,3,4,5};
	
	if(a==b){
	printf("yes");
	}else{
	printf("no");
	}
	return 0;
	}
		A. yes
		B. no
		C. The program will encounter a compilation error.
		D. There will be a runtime error in the program.
			Answer: B
			
			Explanation:
			If statement will compare the base address of two arrays ‘a’ and ‘b’,and they are not same.
			So condition becomes false and program prints “no”

10	Given the following program, which statement will produce a compilation error?
	
	#include<stdio.h>
	main()
	{
	
	int k1=1;
	int k2=5;
	int *ptr1=&k1;
	int *ptr2=&k2;
		A. Statement 1
		B. Statement 2
		C. Both
		D. None
			Answer:A
			
			Explanation:
			Addition of pointers are not allowed in C,whereas subtraction is allowed.

	printf("%d\n",((ptr1-ptr2))); //Statement 1
	
	printf("%d\n",*(ptr1+ptr2)); // Statement 2
	
	}		

11	Find the output:
	main() {
	char * A[] = {"C", "C++", "JAVA", "PHP"};
	char **B[] = {A+2, A+1, A+3, A}, ***C;
	C = B;
	++C;
	printf("%s", *(*C+1)+1);
	}
		A. C++
		B. ++
		C. AVA
		D. JAVA
			Answer: C
			
			Assume the following memory locations for different strings and the pointers.
			
			C= B will initialize it to 200.
			++C => C has address 208
			*C+1 => its pointing to next location of 108 (116)
			*(*C+1)+1 => pointing to 2nd character at 116 printing *(*C+1)+1 will print all characters from 2nd character of JAVA

12	a[x][y][z] is same as	A. *(*(*(a+x)+y)+z) B. *(*(*(a+z)+y)+x) C. *(*(*(a+x+y))+z) D. None of the above	**Answer: A** Explanation: Multidimensional arrays are indexed in the order of highest to lowest. Here, a[x] and *(a+x) refer to the same “plane”. Pointer arithmetic is done internally by the compiler the way it is

			suggested in the answers.
13	char abc[14] = "C Programming"; printf("%s", abc + abc[3] - abc[4]); What would this print?	A. C Programming B. rogamming C. Runtime Error D. Compilation Error	**Answer : B** abc[3] = r = 114(ASCII) abc[4] = o = 111(ASCII) = (abc + 114 - 111) = (abc + 3)
14	What does this mean int ** fun(int **)	A. A function accepting pointer to pointer to an int and returns pointer to pointer to an int B. A function accepting pointer to pointer to an int and returns pointer to an int C. A function accepting pointer to an int and returns pointer to pointer to an int D. None of the above	**Answer : A** Explanation : int indicates an integer variable int * indicates a pointer to an integer variable int ** indicate a pointer to pointer to an integer variable
15	Assume that the size of an integer is 4 bytes. What will be the output of the following code: int a[2][2]={{2,3},{1,6}}; printf("%d",&a[0][1] - &a[0][0]);	A. 1 B. 2 C. 4 D. 8 E. Garbage value	**Answer:** A Explanation: Subtracting pointers gives total number of objects between them
16	Assume the following C variable declaration	A. I, II, and IV only B. II, III, and IV only	Answer: A Explanation:

	int *A[10],B[10][10] Among the following expressions I A[2] II A[2][1] III B[1] IV B[2][3] Which will not give compile-time errors if used as left hand sides of assignment statements in a C program?	C. II and IV only D. IV only	I is valid, assigning value to pointer A[2], II is valid, possible due to array styled indexing of pointers IV is valid, simple assignment to 2-dimensional array Example: int *A[10], B[10][10]; int C[2]={1,6}; A[2]=C; A[2][1]=5; B[2][3]=4
17	I strlen II strchr III strcat IV strcmp Among the above list which of the following are string functions defined?	(A) I only (B) I, III only (C) I, III, IV only (D) All of the Above	Answer: D Explanation: strlen: Computes string length strchr: Search string for a character strcat: Concatenating two strings strcmp: Compare two strings

Programming Questions

<u>Program 0:</u> Largest Sum Contiguous Subarray

Write an efficient C program to find the sum of contiguous subarray within a one-dimensional array of integer which returns the largest sum.

Explanation:

Lets take the example of array {5,-3, 4}

Possible contiguous subarray combinations are

{5}, {-3}, {4}, {5,-3}, {-3,4}, {5,-3,4}

The contiguous subarray {5,-3,4} has got the largest sum 6

Input Constraints:

First line : array size (N), where 1<= N<=100
Second line : N integers of separated by spaces
where each number Ni satisfies
-10000 <= Ni <=10000

Output Constraints:

Single integer SUM which is the largest of sum of all possible contiguous subarray

Public Test Cases:

id	Input	Output
1	3 5 -3 4	6
2	4 1 1 1 1	4
3	8 -2 -3 4 -1 -2 1 5 -3	7
4	5 -5 -2 4 5 1	10

Private Test Cases:

id	Input	Output
1	7 -1 -2 -3 -4 -5 -6 -7	-1
2	2 2000 2000	4000
3	10 -3 -4 1 2 3 -1 2 5 -8 8	12
4	5 -10000 -10000 -10000 -10000 -10000	-10000
5	6 -1 2 -2 4 -3 6	7
6	6 1 -2 2 -4 3 -6	3
7	9 0 -1 0 0 0 0 -1 0 0	0
8	20 1 1 1 1 1 1 1 1 1 1 -2 -3 1 1 1 1 -1 1 1 1	11

Code:

In general one would approach this problem by iterating for all possible start and end combinations which would make N*(N-1)/2 combinations.

But instead of this one could solve this using pre-computation i.e, let us consider cur_max(i) is sum of maximum sum contiguous subarray ending at index i, then cur_max(i) is given by,

cur_max(i) = max(cur_max(i-1)+val(i),val(i)) where val(i) is value at index i.

Maximum sum subarray can be found by finding the maximum of all cur_max.

By using the precomputed cur_max of i-1 we can compute cur_max of i. Hence the below logic becomes a optimal computation of largest sum contiguous subarray.

```
#include<stdio.h>
#define MAX 100

int main()
{
	int size,input[MAX],i;
	scanf("%d",&size);
	for(i=0;i<size;i++){
		scanf("%d",&input[i]);
	}
	//curr_max computes the largest contiguous maximum sum ending at cur_idx
	//max_so_far (global maxima)computes the largest contiguous maximum sum till the cur_idx
	long max_so_far = input[0];
	long curr_max = input[0];
	for (i = 1; i < size; i++){
		curr_max = (input[i]>curr_max+input[i]) ? input[i] : curr_max+input[i];
		max_so_far =(max_so_far>curr_max) ? max_so_far : curr_max;
	}
	printf("%ld",max_so_far);
	return 0;
}
```

Program 1: Find whether two given strings are permutations of each other

Write a program to find whether two given strings are permutations of each other. A string str1 is a permutation of str2 if all the characters in str1 appear the same number of times in str2 and str2 is of the same length as str1.

Input: Two strings S1 and S2

Output:

yes - if they satisfy given criteria

no - otherwise

Constraints:
1 <= len(S1), len(S2) <= 100.
Characters from ASCII range 0 to 127.
White space will not be given in the string.

Public Test cases:

Number	Input	Output
1.	india daini	yes
2.	hellobye hellobye!	no
3.	iloveindia loveindiai	yes
4.	12434 43214	yes
5.	aaa aa	no

Private Test cases:

Number	Input	Output
1.	iitmadras. madras.iit	yes
2.	nptelisbest ptenlisestb	yes
3.	abcdefg aabbccddeeffgg	no
4.	12345 1122334455	no
5.	#$%& &%$#	yes
6.	(abc) (xyz)	no
7.	"hellobye" "byehello"	yes

Solution program 1:

```
#include<stdio.h>
int main(){
        int acount[128] = {0}, bcount[128] = {0}, c = 0;
```

```
	char a[100];
	char b[100];

	scanf("%s",a);
	scanf("%s",b);


	//now take every character from string 'a' and using ASCII value increment
corresponding index in array 'acount'
	while (a[c] != '\0')
	{
	acount[(int)a[c]]++;
	c++;
	}

	c = 0;

	//now take every character from string 'b' and using ASCII value increment
corresponding index in array 'bcount'
	while (b[c] != '\0')
	{
	bcount[(int)b[c]]++;
	c++;
	}

	for (c = 0; c < 128; c++)
	{
		//if any single character also mismatch then return no
		if (acount[c] != bcount[c])
		{
			printf("no");
			return 0;
		}
	}
	//satisfy criteria so return true
	printf("yes");
	return 0;
}
```

Program 2: Find balancing index in array

Write a program that given a number n and a sequence of n integers , it outputs the first(lowest) index i where the following condition is satisfied:

- Sum of elements at lower indices(<i) = sum of elements at higher indices (>i)
- if the above condition does not hold for any index then output -1
- The Sum of lower indices(<i) when i=0, should be initialized to 0 and the higher indices(>i) should be initialized to A[1] + A[2] +A[3] ……...A[N-1], where N is the size of the array.

Explanation:

Output the index of an array such that the sum of elements at lower indices is equal to the sum of elements at higher indices.

For example, in an array A let:
A[0] = -7, A[1] = 1, A[2] = 5, A[3] = 2, A[4] = -4, A[5] = 3, A[6]=0

3 is a valid answer, because:
A[0] + A[1] + A[2] = A[4] + A[5] + A[6]

Input: The size of array N, followed by N numbers.

Output:
index i - if i is the lowest index of the array satisfying the required condition
-1 - if there does not exist any such index

Constraints:
2 <= sizeof array <= 100
All entries of array,Arr[i] will follow the following property
-1000 <= Arr[i] <= 1000

Public Test cases:

Number	Input	Output
1.	7 -7 1 5 2 -4 3 0	3
2.	5 -1 5 4 2 -7	1
3.	4 3 2 1 -3	0
4.	3 -3 1 3	-1
5.	2 0 5	1

Private Test cases:

Number	Input	Output
1.	2 1 0	0
2.	11 1 4 6 -3 2 4 5 4 10 5 -5	6
3.	20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0	9
4.	5 1 2 3 4 5	-1
5.	20 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	-1
6.	4 -3 -2 -1 -2	1
7.	7 30 1 2 3 -1 -2 -3	0

Solution program 2:

```
#include <stdio.h>


int main()
{

  int arr_size,i,ans=-1,sum=0,leftsum=0 ;
  scanf("%d",&arr_size);
  int arr[arr_size];// = {-7, 1, 5, 2, -4, 3, 0};
  for(i=0;i<arr_size;i++)
    scanf("%d",&arr[i]);


  /* Find sum of the whole array */
  for (i = 0; i < arr_size; i++)
    sum += arr[i];

  for( i = 0; i < arr_size; i++)
    {
      sum -= arr[i]; // sum is now right sum for index i

      if(leftsum == sum)
        {
            ans=i;
            break;
          }
```

```
      leftsum += arr[i];

    }

  printf("%d",ans);
  return 0;
}
```

Programming 3

The depth of a alphabet is the number of parentheses it is surrounded by. So write a C program to find the depth of each alphabet in the input.

Explanation:
(a (b) ((c d) e) f) g
g is at depth 0
a and f are at depth 1
b and e are at depth 2
c and d are at depth 3

Input Constraints:

- Number of characters in a input ranges from 1 - 100
- The input will have only ‘(‘ , ‘)’ and letters from English alphabet
- There will be no repetition of letters.
- Only lowercase letters are used.
- The letters can be in any sequence.

Input: An array of characters
Output:

- The depth of each letter separated by a space.
- The order of the depth of the letters should be the same order that the letters appear in the input.
- To mark the end of the output it should end with a space and a ‘#’ character.

Example 1:
Input: (a(b)((cd)e)f)g
Output: 1 2 3 3 2 1 0 #

Example 2:
Input: p(r(q))(s)
Output: 0 1 2 1 #

In this example, letters are appearing in the order p followed by r followed by q and s. They have depth of 0, 1, 2 and 1 respectively. Note that the depth is not printed in the order p,q,r,s (the alphabetical order) but p,r,q,s (the order in which they appear in the input string).

Public Test cases:

Number	Input	Output
1.	(a(b)((cd)e)f)g	1 2 3 3 2 1 0 #
2.	a(b(c))(d)	0 1 2 1 #
3.	a(b(c))(d(fe))	0 1 2 1 2 2 #

Private Test cases:

Number	Input	Output
1.	a(b(c))(d(f()e))	0 1 2 1 2 2 #
2.	()	#
3.	ab()(c(d(e(f)()(g)h)))	0 0 1 2 3 4 4 3 #
4.	((((a))b))cdegfhi(jklmnop)	4 2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 #
5.	((a))((b(c)((d(e(f(g(h(i(j(k(l))))))))))))	2 2 3 4 5 6 7 8 9 10 11 12 #
6.	((a))((b(c)((d()(e(f(g(h(i(j(k(l)))))()))))))(m(n() (o(p(q(r(s(t()(u(v(w(x(y(z)))))(()))))))()))())))	2 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12 13 14 #
7.	()(()((())))	#
8.	(a)b(c(def)g(h(ijkl(mn)o)p))	1 0 1 2 2 2 1 2 3 3 3 3 4 4 3 2 #
9.	(z)	1 #
10.	((x))	2 #
11.	a	0 #
12.	abc(d)	0 0 0 1 #

Solution:

```
#include<stdio.h>
```

```
#include<string.h>
int main()
 {
  int a=0,i,set=0;
  char input[100];
  scanf("%s",input);
  for(i=0;input[i]!='\0';i++)
   {
        switch(input[i])
        {
        case '(':
                a++;
                break;
        case ')':
                a--;
                break;
        default :
                printf("%d ",a);
                set = 1;
        }

   }
  if(set==0)
        printf(" #");
  else
        printf("#");
  return 0;
}
```