Qn. No.	**Code :(15)2131 Programming in C** scoring Indicators	Split score	Total score	Total
I 1.	Any 4 keywords auto double int struct break else long switch case enum register typedef char extern return union const float short unsigned continue for signed void default goto sizeof volatile do if static while	½ * 4	2	

2.	We write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement.
	if(condition)
	{……..
	  if(condition)
	  {……}
	  else
	  {……}
	}
	else
	{
	…..
	}
		2	2	10

3.	A[0],………,A[8], so index of last element is 8	2	2	
4.	#include, #define, #if , #elif, #undef (Any 2)	2	2	
5.	Strcat(str1,str2)	2	2	

II 1.	The control statement that allows us to make a decision from the number of choices is called a **switch.**
	
	switch (integer expression)
	{
	    case constant 1 :
	    do this ;
	    case constant 2 :
	    do this ;
	    case constant 3 :
	    do this ;
	    default :
	    do this ;
	}
	Example code
		4
		2
			6	

2.	Pointer is a variable that stores the address of another variable. Pointer declaration and initialization with Any example to show declaration and initialization main( ) { int i = 3 ; int *j ; j = &i ; printf ("\nAddress of i = %u", &i) ;	2 4	6	

	printf ("\nAddress of i = %u", j) ; }			
3	**Register Storage Class** The features of a variable defined to be of **register** storage class are as under: Storage - CPU registers. Default initial value - Garbage value. Scope - Local to the block in which the variable is defined. Life - Till the control remains within the block in which the variable is defined. if a variable is used at many places in a program it is better to declare its storage class as **register**.Example of frequently used variables is loop counters. **External Storage Class** The features of a variable whose storage class has been defined as external are as follows: Storage – Memory. Default initial value – Zero. Scope – Global. Life – As long as the program's execution doesn't come to an end.	3 3	6	30

4	There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer –
	
	`int *ptr[MAX];`
	
	It declares **ptr** as an array of MAX integer pointers. Thus, each element in ptr, holds a pointer to an int value. The following example uses three integers, which are stored in an array of pointers, as follows –
	
	`#include <stdio.h>`
	`Void main () {`
	
	`int  var[] = {10, 100, 200};`
	`int i, *ptr[10];`
	
	`for ( i = 0; i < 10; i++) {`
	`ptr[i] = &var[i]; /* assign the address of integer. */`
	`}`
	
	`for ( i = 0; i < 10; i++) {`
	`printf("Value of var[%d] = %d\n", i, *ptr[i] );`
	`}`
	
	`}`
		2
		4		

5	Arrays can hold a number of pieces of information of the same data type. To deal with entities that are collection of dissimilar data types, structure is used. Array elements are stored in contiguous memory location. Array elements are accessed by their index number Array declaration and element accessing operator is "[ ]" (square bracket). Every element in array is of same size. There is no keyword to declare an array Structure elements may not be stored in a contiguous memory location. Structure elements are accessed by their names. Structure element accessing operator is "." (Dot operator). Every element in a structure is of different data type. "struct" is a keyword used to declare the structure.	3	6	
	Example for array and structure declaration, initialization	3		
6	Any three **strlen( )** This function counts the number of characters present in a string. While calculating the length it doesn't count '\0'. **strcpy( )** This function copies the contents of one string into another. **strcpy( )** goes on copying the characters in source string into the target string till it doesn't encounter the end of source string ('\0') **strcat( )** This function concatenates the source string at the end of the target string. **strcmp( )** This is a function which compares two strings to find out whether they are same or different. If the two strings are identical, **strcmp( )** returns a value zero. If they're not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters.	2* 3	6	
7	Arrays are the collection of homogenous datatypes (similar elements). Similar elements could be all **ints**, or all **floats**, or all **chars**, etc. The array of characters is called a 'string'. The first element in the array is numbered 0, so the last element is 1 less than the size of the array. An array is also known as a subscripted variable. **1-D Array Declaration** int a[8] ; **int** specifies the type of the variable **marks** specifies the name of the variable. The number 8 tells how many elements of the type **int** will be in array. **Array Initialisation**	3		

	int num[6] = { 2, 4, 12, 5, 45, 5 } ; int n[ ] = { 2, 4, 12, 5, 45, 5 } ; float press[ ] = { 12.3, 34.2 -23.4, -11.3 } ; **Two Dimensional Arrays** It is also possible for arrays to have two or more dimensions. The two dimensional array is also called a matrix. Declaration int a[50][50]; **Initialising a 2-Dimensional Array** int stud[4][2] = { { 1234, 56 }, { 1212, 33 }, { 1434, 80 }, { 1312, 78 } } ;	3	6	

III (a)

+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	
%	Modulus Eg : 23%2 = 1(remainder)	

Any 3 operators (3*3=9) from each group

==	Checks if value of two operands is equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands is equal or not, if values are not equal then conditior becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes ther condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then conditior becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true

&&	Called Logical AND operator. If both the operands are non zero then the condition becomes true.	A=5 and B =6 ((A ==5) && (B>5)) returns false.
||	Called Logical OR Operator. If any of the two operands is non zero then the condition becomes true.	((A ==5) && (B>5)) returns true
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A==5) returns false.

III (b)	#include<stdio.h>
	void main()
	{
	int n,num,digit,rev=0;
	printf("type in an integer");
		6	6	

	scanf("%d",&num);
	n=num;
	do
	{
	digit=num%10;
	rev=rev*10+digit;
	num/=10;
	}
	while(num!=0);
	{
	printf("reverse of the no is%d\n",rev);
	if(n==rev)
	{
	printf("the no is a palindrome\n");
	}
	else
	{
	printf("the no is not a palindrome");
	}
	}
	}
		Header file -1
		Declaration -1
		Input -1
		output -1
		Logic - 2
			(6)	15

IV(a)	While loop, do while loop ,for loop
	Syntax
	The syntax of a **while** loop in C programming language is while(condition) {
	`statement(s);`
	`}`
	The syntax of a **do...while** loop in C programming language is –
	`do {`
	`statement(s);`
	`} while( condition );`
	The syntax of a **for** loop in C programming language is –
	`for ( init; condition; increment ) {`
	`statement(s);`
	`}`
	working
	example
		3*3
		(Syntax -1
		working -1
		Ex: -1)
			9	

IV (b)	**The *break* Statement** If we want to jump out of a loop instantly, without waiting to get back to the conditional test. The keyword **break** allows us to do this. When **break** is encountered inside any loop, control automatically passes to the			

	first statement after the loop. A **break** is usually associated with an **if.**
	#include<stdio.h>
	void main( )
	{
	int num, i ;
	printf ("Enter a number ") ;
	scanf ("%d", &num) ;
	i = 2 ;
	while (i <= num - 1)
	{
	if (num % i == 0)
	{
	printf ("Not a prime number") ;
	break ;
	}
	i++ ;
	}
	
	The continue statement
	
	The keyword **continue** allows to bypass the statements inside the loop. A **continue** is usually associated with an **if.**
	
	#include<stdio.h>
	void main()
	{
	int i,j;
	for(i=0;i<7;i++)
	{
	for(j=0;j<3;j++)
	{
	if(i==j)
	{
	continue;
	}
	printf("%d \t %d \n ",i,j);
	}
	}
	
	Explain an example
		3*2	6	15

V (a)	A function is a self-contained block of statements that perform a coherent task of some kind. Every C program can be thought of as a collection of these functions.
	main( )
	{
	message( ) ;
	printf ("\nCry, and you stop the monotony!") ;
	}
	message( )
	{
	printf ("\nSmile, and the world smiles with you...") ;
	}			

	Here, **main( )** itself is a function and through it we are calling the function **message( )**. **main( )** becomes the 'calling' function, whereas **message( )** becomes the 'called' function.
	A function has function call , function Definition and function declaration (function Prototype)
	• Function Call: A function gets called when the function name is followed by a semicolon.
	For example,
	main( )
	{
	argentina( ) ;
	}
	• Function Definition: A function is defined when function name is followed by a pair of braces in which one or more statements may be present. For example,
	void argentina( )
	{
	statement 1 ;
	statement 2 ;
	statement 3 ;
	}
	Function definition can be placed before or after the main. If given after main.
	• Function Prototype: Like variables functions must be declared. It consists of return type, function name, parameter list, and semicolon. Function declarations are not essential. If function is used before declaration linker will fail.
	void argentina();
	➢ Any function can be called from any other function. Even **main( )** can be called from other functions. For example,
	main( )
	{
	message( ) ;
	}
	void message( )
	{
	printf ("\nCan't imagine life without C") ;
	main( ) ;
	}
		3*3	9	

V (b)	#include<stdio.h>
	int Fibonacci(int);
	int main()
	{
		6	6	15

	int n, i = 0, c;
	scanf("%d",&n);
	printf("Fibonacci series\n");
	for (c = 1 ; c <= n ; c++)
	{ printf("%d\n", Fibonacci(i));
	i++;
	}
	return 0;
	}
	int Fibonacci(int n)
	{ if (n == 0)
	return 0;
	else if (n == 1)
	return 1;
	else
	return (Fibonacci(n-1) + Fibonacci(n-2));}
		Header - 1
		Declaration - 1
		Input - 1
		Output statement - 1
		Logic - 2		

VI (a)	**Types of function calls** 1. **Call by Value (Pass by value)** 2. **Call by Reference (Pass by reference)** **Call by Value** Values of the arguments is passed to the function. In this method the 'value' of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. With this method the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function. void swap (int x, int y) { int t ; t = x ;	3+3 Eg: 1.5*2= 3	9	

```
        x = y ;
        y = t ;
        printf ( "\nx = %d y = %d", x, y ) ;
}
main( )
{
        int  a = 10, b = 20 ;
        swap ( a, b ) ;
        printf ( "\na = %d b = %d", a, b ) ;
}
```

Call by reference

Address of the arguments is passed to the function.

In this method the addresses of actual arguments in the calling function are copied into formal arguments of the called function. This means that using these addresses we would have an access to the actual arguments and hence we would be able to manipulate them.

```
void swap( int *x, int *y )
{
        int  t ;
        t = *x ;
        *x = *y ;
        *y = t ;
}

main( )
{
        int  a = 10, b = 20 ;
        swap ( &a, &b ) ;
        printf ( "\na = %d b = %d", a, b ) ;
}
```

15

VI (b)

```
#include <stdio.h>

// Define macro to find square and cube
#define CUBE(x) (x * x * x)

int main()
{
   int num;

   // Input a number from user
   printf("Enter any number to find square and cube: ");
   scanf("%d", &num);

   // Calculate and print cube
   printf("CUBE(%d) = %d\n", num, CUBE(num));

   return 0;
}
```

6	6

Header -1

Declaration-1

Macro definition-2

Macro call - 2

VII

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

(a)	int main() {
	  int array[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
	  int even[10], odd[10];
	  int loop, e, d;
	
	  e = d = 0;
	
	  for(loop = 0; loop < 10; loop++) {
	    if(array[loop]%2 == 0) {
	      even[e] = array[loop];
	      e++;
	    } else {
	      odd[d] = array[loop];
	      d++;
	    }
	  }
	
	  printf("\n even -> ");
	
	  for(loop = 0; loop < e; loop++)
	    printf(" %d", even[loop]);
	
	  printf("\n odd -> ");
	
	  for(loop = 0; loop < d; loop++)
	    printf(" %d", odd[loop]);
	
	  return 0;
	}
		9
		Header file - 1
		Array Declaration - 2
		Array input - 2
		Logic - 2
		output statement - 2
			9	

VII
(b)
	#include<stdio.h>
	void largest(int x[],int size)
	{
	  int large=-999,i;
	  for(i=0;i<size;i++)
	  {
	    if (x[i]>large)
	      large=x[i];
	  }
	  printf("largest is : %d",large);
	}
	void main()
	{
	
	  int a[100];
	  int n,i;
	  printf("Enter the size of array");
	  scanf("%d",&n);
	  for(i=0;i<n;i++)
	  {
	    scanf("%d",&a[i]);
	  }
	  largest(a,n);
	}
	OR
		6
		Header - 1
		Array declaration - 2
		Function declaration - 1
		function defn - 1
		function call - 1
			6	

<table>
<tr><td></td><td>Starting address of array can be passed to a pointer variable and pointer variable can be incremented to access elements one by one</td><td></td><td></td><td></td></tr>
<tr><td>VIII
(a)</td><td>

```
#include<stdio.h>
void main()
{
int a[50][50],b[50][50],ra,rb,ca,cb,i,j,k,c[50][50];
printf("enter the row and column of the matrix A");
scanf("%d%d",&ra,&ca);
printf("enter the elements");
for(i=0;i<ra;i++)
{
for(j=0;j<ca;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter the row and column of the matrix B");
scanf("%d%d",&rb,&cb);
printf("enter the elements");
for(i=0;i<rb;i++)
{
for(j=0;j<cb;j++)
{
scanf("%d",&b[i][j]);
}
}
if(ra==cb)
{
for(i=0;i<ra;i++)
{
for(j=0;j<ca;j++)
{
c[i][j]=0;
for(k=0;k<ca;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[j][k];
}
}
}
}
else
{
printf("multiplication is not possible");
}
printf("the resulting matrix is:\n");
for(i=0;i<ra;i++)
{
for(j=0;j<ca;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

</td><td>Array Declaration - 1
Array i/p - 2
logic - 4
Array o/p Matrix format } 2</td><td>(9)</td><td></td></tr>
</table>

VIII
(b)
	#include<stdio.h>
	void main()
	{
	int a[100],i,n,c;
	int search=0;
	printf("enter the value");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
	scanf("%d",&a[i]);
	}
	printf("enter the search value");
	scanf("%d",&c);
	for(i=0;i<n;i++)
	{
	if(a[i]==c)
	{
	search=1;
	}
	else
	{
	search=0;
	}
	}
	if(search==1)
	{
	printf("%d is present",c);
	}
	else
	{
	printf("%d is absent",c);
	}
	}
		Array
		Declaration
		-1
		Array
		i/p - 2
		Logic-3
			6	15

IX (a)	#include<stdio.h>
	#include<conio.h>
	struct student
	{
	int Roll_No;
	char name[20];
	int phy_marks;
	int chem_marks;
	int math_marks;
	int t;
	float per;
	};
	void main()
	{
	struct student a[5], i;
	clrscr();
	for(i=0;i<5;==i)
	{
	printf(" Enter RollNo, Name amd three sub marks\n");
	scanf("%d%s%d%d%d",&a[i].Roll_No,&a[i].name,&a[i].phy_
		Structure
		Defn-3
		Declaration -1
		i/p - 2
		logic- 2
		o/p - 1
			9	

	marks,&a[i].chem_marks,&a[i].math_marks);
	a[i].t=a[i].phy_marks+a[i].chem_marks+a[i].math_marks ;
	a[i].per=a[i].t/3.0;
	//print the details
	getch();
	}			

IX(b)	#include <stdio.h>
	int main()
	{
	char s1[100], s2[100], i;
	
	printf("Enter string s1: ");
	scanf("%s",s1);
	
	for(i = 0; s1[i] != '\0'; ++i)
	{
	s2[i] = s1[i];
	}
	
	s2[i] = '\0';
	printf("String s2: %s", s2);
	
	return 0;
	}
		Declaration-2
		i/p - 2
		logic-2
			6	

X(a)	**Defining a Structure**
	
	To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –
	
	struct [structure tag] {
	
	member definition;
	member definition;
	...
	member definition;
	} [one or more structure variables];
	
	At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Structure variables can be declared inside main function also.
	
	struct Books {
	char title[50];
	char author[50];
	char subject[100];
	int book_id;
	} book;
	Or
	
	void main( ) {
		3+3+
		3
			9	

```
   struct Books Book1;

}
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. use the keyword **struct** to define variables of structure type.

An example:

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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1
of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming
Tutorial");
   Book1.book_id = 6495407;



   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n",
Book1.author);
   printf( "Book 1 subject : %s\n",
Book1.subject);
   printf( "Book 1 book_id : %d\n",
Book1.book_id);
}
```

X(b)

```
#include <stdio.h>
int main()
{
 int c = 0, count = 0;
 char s[1000];
  printf("Input a string\n");
 gets(s);
```

Declaration-2
i/p - 2
logic-2

6

```
  while (s[c] != '\0') {
   if (s[c] == 'a' || s[c] == 'A' || s[c] == 'e' || s[c] == 'E' ||
s[c] == 'i' || s[c] == 'I' || s[c] =='o' || s[c]=='O' || s[c] ==
'u' || s[c] == 'U')
     count++;
   c++;
  }
  printf("Number of vowels in the string: %d", count);
  return 0;
}
```