Scoring indicators

Code:2131

Qn:No	Scoring indicators	Split score	Total score
11	**Keywords** are predefined, reserved words used in programming that have special meanings to the compiler. **Keywords** are part of the syntax and they cannot be used as an identifier. Examples: auto,break,continue etc	1+1	2

12.	switch(expression) {
	case constant-expression :
	statement(s);
	break;
	case constant-expression :
	statement(s);
	break;
	default :
	statement(s);
	}
		1X2	2

13	A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.	1X2	2
14	Syntax of array declaration is: datatype arrayname[array size]; int a[10];	1+1	2
15	Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.	1X2	2

11 1	Syntax of for loop
	for (init; condition; increment) {
	statement(s);
	}
	The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
	Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
	After the body of the 'for' loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
	The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.
	#include <stdio.h>
	int main () {
	int a;
	/* for loop execution */
	for(a = 10; a < 20; a = a + 1){
	printf("value of a: %d\n", a);
	}
	return 0;
	}
	output
	value of a: 10
	value of a: 11
		2+2+2	6

	value of a: 12 value of a: 13 value of a: 14 value of a: 15 value of a: 16 value of a: 17 value of a: 18 value of a: 19		
11 2	The **C Preprocessor** is not a part of the compiler, but is a separate step in the compilation process. In simple terms, a C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation. We'll refer to the C Preprocessor as CPP. All preprocessor commands begin with a hash symbol (#). It must be the first nonblank character, and for readability, a preprocessor directive should begin in the first column. The following section lists down all the important preprocessor directives 1. **#define** Substitutes a preprocessor macro. 2. **#include** Inserts a particular header from another file. 3. **#undef** Undefines a preprocessor macro. 4. **#ifdef** Returns true if this macro is defined. 5. **#ifndef** Returns true if this macro is not defined. 6. **#if** Tests if a compile time condition is true.	2+4	6

11 3	Recursion is a common method of simplifying a problem into subproblems of same type. This is called divide and conquer technique. A basic example of recursion is factorial function.
	Example:
	int factorial(int n){
	if(n==0) return 1;
	else return (n* factorial(n-1));
	}
		2 fig
		
		4 Desc
			6

11 4	#include<stdio.h>
	Void main()
	{
	int N,sum=0,a[15],I;
	printf("Enter number of elements");
	scanf("%d",&N);
	printf("Enter array");
	for(i=0;i<N;i++)
	scanf("%d",&a[i]);
	for(i=0;i<N;i++)
	sum=sum+a[i];
	printf("sum=%d",sum);
	}
		Declare, read 2 Marks
		
		+ logic 3 marks
		
		+output writing 1
			6

11 5	#include <stdio.h>
	int main()
	{
	int n, i, sum = 0;
	printf("Enter a positive integer: ");
	scanf("%d",&n);
	for(i=1; i <= n; ++i)
		Declare, read 2 Marks
		
		+ logic 3
			6

	{
	sum += i;
	}
	printf("Sum = %d",sum);
	return 0;
	}
		marks
		
		+output
		writing 1	

11 6	The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello." char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; If you follow the rule of array initialization then you can write the above statement as follows char greeting[] = "Hello"; The scanf() function reads the sequence of characters until it encounters a whitespace (space, newline, tab etc.). You can use gets() function to read a line of string.	2+2+2 3	6

II 7	struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived datatypes
	syntax
	struct [structure_tag]
	{
	//member variable 1
	//member variable 2
	//member variable 3
	...
	}[structure_variables];
	Example of structure
	struct Student
	{
	char name[25];
	int age;
	char branch[10];
	// F for female and M for male
	char gender;
	};
	
	Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.
	
	Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag.**
	
	Accessing Structure Members
	
	Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot . operator also called **period** or **member access** operator.
		2+2+2	6

111 a	C operators can be classified into following types:	1+ Any (4*2)	9

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators

Arithmetic operators

C supports all the basic arithmetic operators. The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Relational operators

The following table shows all relation operators supported by C.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Logical operators

C language supports following 3 logical operators. Suppose a = 1 and b = 0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise operators

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left. Bitwise operators are not applied to float or double

Operator	Description

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

The truth table for bitwise &, | and ^

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence.

Example :

a = 0001000
b = 2
a << b = 0100000
a >> b = 0000010
Assignment Operators

Assignment operators supported by C language are as follows.

Operator	Description
=	assigns values from right side operands to left side operand
+=	adds right operand to the left operand and assign the result to left
-=	subtracts right operand from the left operand and assign the result to left operand
*=	mutiply left operand with the right operand and assign the result to left operand
/=	divides left operand with the right operand and assign the result to left operand
%=	calculate modulus using two operands and assign the result to left operand

Conditional operator

The conditional operators in C language are known by two more names

1. **Ternary Operator**
2. **? : Operator**

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

expression 1 ? expression 2: expression 3

Explanation:

- The question mark "?" in the syntax represents the **if** part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the expression on the left side of " : " i.e (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of " : " i.e (expression 3) is executed.

Special operator

Operator	Description	Example
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x** ; return address of the variable **x**
*	Pointer to a variable	***x** ; will be pointer to a variable **x**

III b

3+3 | 6

Difference Between break and continue in C

S.No.	break	continue
1.	break statement is used in switch and loops.	continue statement is used in loops only.
2.	When break is encountered the switch or loop execution is immediately stopped.	When continue is encountered, the statements after it are skipped and the loop control jump to next iteration.
	Example:	**Example:**
	#include<stdio.h>	#include<stdio.h>
	int main(){	int main(){
	int i;	int i;
	for(i=0;i<5;++i){	for(i=0;i<5;++i){

```
if(i==3)
    break;
    printf("%d ",i);
}
return 0;
}
```

Output:

0 1 2

```
if(i==3)
    continue;
    printf("%d ",i);
}
return 0;
}
```

Output:

0 1 2 4

IV a

1. If statement

Syntax of if statement:
The statements inside the body of "if" only execute if the given condition returns true. If the condition returns false then the statements inside "if" are skipped.

```
if (condition)

{
   //Block of C statements here
   //These statements will only execute if the condition is true
}
#include <stdio.h>
int main()
{
   int x = 20;
   int y = 22;
   if (x<y)
   {
      printf("Variable x is less than y");
   }
   return 0;
}
```

2. If else statement

Syntax of if else statement:
If condition returns true then the statements inside the body of "if" are executed and the statements inside body of "else" are skipped.
If condition returns false then the statements inside the body of "if" are skipped and the statements in "else" are executed.

```
if(condition) {
  // Statements inside body of if
}
else {
  //Statements inside body of else
}
```

2+2+2+2 | 8

```
include <stdio.h>
int main()
{
  int age;
  printf("Enter your age:");
  scanf("%d",&age);
  if(age >=18)
  {
            /* This statement will only execute if the
             * above condition (age>=18) returns true
             */
            printf("You are eligible for voting");
  }
  else
  {
            /* This statement will only execute if the
             * condition specified in the "if" returns false.
             */
            printf("You are not eligible for voting");
  }
  return 0;
}
```

3. if else ladder

The if else ladder statement in C programming language is used to test set of conditions in sequence. An if condition is tested only when all previous if conditions in if-else ladder is false. If any of the conditional expression evaluates to true, then it will execute the corresponding code block and exits whole if-else ladder.

```
#include<stdio.h>
#include<conio.h>
 int main(){
   int marks;

   printf("Enter your marks between 0-100\n");
   scanf("%d", &marks);
   /* Using if else ladder statement to print
     Grade of a Student */
   if(marks >= 90){
      /* Marks between 90-100 */
      printf("YOUR GRADE : A\n");
   } else if (marks >= 70 && marks < 90){
      /* Marks between 70-89 */
      printf("YOUR GRADE : B\n");
   } else if (marks >= 50 && marks < 70){
      /* Marks between 50-69 */
      printf("YOUR GRADE : C\n");
   } else {
      /* Marks less than 50 */
      printf("YOUR GRADE : Failed\n");
   }

   return(0);
}
```

4. Nested if...else

It is possible to include if...else statement(s) inside the body of another if...else statement.

```
#include <stdio.h>
int main()
```

```
{
  int number1, number2;
  printf("Enter two integers: ");
  scanf("%d %d", &number1, &number2);

  if (number1 >= number2)
  {
   if (number1 == number2)
   {
    printf("Result: %d = %d",number1,number2);
   }
   else
   {
    printf("Result: %d > %d", number1, number2);
   }
  }
  else
  {
    printf("Result: %d < %d",number1, number2);
  }

  return 0;
}
```

V a

No	Macro	Function
1	Macro is **Preprocessed**	Function is **Compiled**
2	**No Type Checking**	**Type Checking** is Done
3	**Code** Length **Increases**	**Code** Length remains **Same**
4	Use of macro can lead to **side effect**	No **side Effect**
5	Speed of Execution is **Faster**	Speed of Execution is **Slower**
6	Before Compilation macro name is replaced by macro value	During function call , Transfer of Control takes place
7	Useful where small code appears many time	Useful where large code appears many time
8	Generally Macros do not extend beyond one line	Function can be of any number of lines
9	Macro does not Check **Compile Errors**	Function Checks **Compile Errors**

4+4

8

IV b.	#include <stdio.h>
	
	void main()
	{
	int i, N, oddSum = 0;
	
	printf("Enter the value of N\n");
	scanf ("%d", &N);
	
	for (i=1; i <=N; i++)
	{
	if (i % 2 != 0)
	oddSum = oddSum + i;
	}
	
	printf ("Sum of all odd numbers = %d\n", oddSum);
	
	}
		Declare, read 3 Marks
		
		+ logic 3 marks
		
		+output writing 1
			7

V b	There are four storage classes in C they are as follows: 1. Automatic Storage Class 2. Register Storage Class 3. Static Storage Class 4. External Storage Class 1. Automatic Storage Class A variable defined within a function or block with auto specifier belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block. Register Storage Class The register specifier declares a variable of register storage class. Variables belonging to register storage class are local to the block which they are defined in, and get destroyed on exit from the block. A register declaration is equivalent to an auto declaration, but hints that the declared variable will be accessed frequently; therefore they are placed in CPU registers, not in memory. Only a few variables are actually placed into registers, and only certain types are eligible; the restrictions are implementation-dependent. However, if a variable is declared register, the unary & (address of) operator may not be applied to it, explicitly or implicitly. Register variables are also given no initial value by the compiler. Static Storage Class The static specifier gives the declared variable static storage class. Static variables can be used within function or file.Unlike global variables, static variables are not visible outside their function or file, but they maintain their values between calls. The static specifier has different effects upon local and global variables. See the following flavours of static specifier. External Storage Class The extern specifier gives the declared variable external storage class. The principal use of extern is to specify that a variable is declared with *external linkage* elsewhere in the program. To understand why this is important, it is necessary to understand the difference between a declaration and a definition. A declaration declares the name and type of a variable or function. A definition causes storage to be allocated for the variable or the body of the	*Any 3*3*	9

	function to be defined. The same variable or function may have many declarations, but there can be only one definition for that variable or function. When extern specifier is used with a variable declaration then no storage is allocated to that variable and it is assumed that the variable has already been defined elsewhere in the program. When we use extern specifier the variable cannot be initialized because with extern specifier variable is declared, not defined.		

VI a	**Call By Value**: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.
	
	#include <stdio.h>
	
	// Function Prototype
	void swapx(int x, int y);
	
	// Main function
	int main()
	{
	int a = 10, b = 20;
	
	// Pass by Values
	swapx(a, b);
	
	printf("a=%d b=%d\n", a, b);
	
	return 0;
	}
	
	// Swap functions that swaps two values
	void swapx(int x, int y)
	{
	int t;
	
	t = x;
	x = y;
	y = t;
	
	printf("x=%d y=%d\n", x, y);
	}
	
	Call by Reference: Both the actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.
	
	#include <stdio.h>
	
	// Function Prototype
	void swapx(int*, int*);
	
	// Main function
	int main()
	{
	int a = 10, b = 20;
		*3.5*2*	7

```
    // Pass reference
    swapx(&a, &b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

// Function to swap two variables by references
void swapx(int* x, int* y)
{
    int t;

    t = *x;
    *x = *y;
    *y = t;
     printf("x=%d y=%d\n", *x, *y);
}
```

VI b

Data Types in C

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it.Following are the examples of some very common data types used in C:

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer.
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision.

*Any 4*2*

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c

8

VII a	**Declaring Arrays** To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows – type arrayName [arraySize]; This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement – double balance[10]; Here *balance* is a variable array which is sufficient to hold up to 10 double numbers. **Initializing Arrays** You can initialize an array in C either one by one or using a single statement as follows double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};	*2+4*	6

VII b	#include <stdio.h>
	
	int main()
	
	{
	
	int m, n, c, d, first[10][10], second[10][10], sum[10][10];
	
	printf("Enter the number of rows and columns of matrix\n");
	
	scanf("%d%d", &m, &n);
	
	printf("Enter the elements of first matrix\n");
	
	for (c = 0; c < m; c++)
	
	for (d = 0; d < n; d++)
	
	scanf("%d", &first[c][d]);
	
	printf("Enter the elements of second matrix\n");
	
	for (c = 0; c < m; c++)
	
	for (d = 0 ; d < n; d++)
	
	scanf("%d", &second[c][d]);
	
	printf("Sum of entered matrices:-\n");
	
	for (c = 0; c < m; c++) {
		Declare, read 3 Marks
		
		+ logic 4 marks
		
		+output writing 2
			9

	for (d = 0 ; d < n; d++) {
	sum[c][d] = first[c][d] + second[c][d];
	printf("%d\t", sum[c][d]);
	}
	printf("\n");
	}return(0);}		

VIII a	#include<stdio.h>
	Void main()
	{
	int N,sum=0,a[15],i;
	float average;
	float avg(int[],int);
	printf("Enter number of elements");
	scanf("%d",&N);
	printf("Enter array");
	for(i=0;i<N;i++)
	scanf("%d",&a[i]);
	average=avg(a,N);
	printf("Average=%f",average);
	}
	float average(int x[][],int p)
	{
	float m,s=0;
	for(i=0;i<p;i++)
	s=s+x[i];
	m=s/p;
	return(m);
	}
		Declare, read 2 Marks
		+ logic 3 marks
		+output writing 2
			7

VIII b	#include<stdio.h>
	Void main()
	{
	int N,l,a[15],I,*p;
	printf("Enter number of elements");
	scanf("%d",&N);
	printf("Enter array");
	for(i=0;i<N;i++)
	scanf("%d",&a[i]);
	p=&a[0];
	for(i=1;i<N;i++)
		Declare, read 3 Marks
		+ logic 5 marks
		+output writing 1
			9

	{
	If(a[i]>*p)
	*p=a[i];
	}
	Printf("largest one=%d",l);
	}		

IX	1. strlen()-return the length of a string
	#include <stdio.h>
	#include <string.h>
	int main()
	{
	char str1[20] = "welcome";
	printf("Length of string str1: %d", strlen(str1));
	return 0;
	}
	2. strcpy(string1,string2)-It copies the string str2 into string str1
	include <stdio.h>
	#include <string.h>
	int main()
	{
	char s1[30] = "string 1";
	char s2[30] = "string 2 :
	/* this function has copied s2 into s1*/
	strcpy(s1,s2);
	printf("String s1 is: %s", s1);
	return 0;
	}
	3. strcat(str1,str2)-It concatenates n characters of str2 to string str1
	#include <stdio.h>
	#include <string.h>
	int main()
	{
	char s1[10] = "Hello";
	char s2[10] = "World";
	strncat(s1,s2);
	printf("Concatenation using strncat: %s", s1);
	return 0;
	}
	4. strcmp()- It compares both the string till n characters or in other words it compares first n characters of both the strings.
	#include <stdio.h>
	#include <string.h>
	
	int main () {
	char str1[20];
	char str2[20];
	int result;
	
	//Assigning the value to the string str1
	strcpy(str1, "hello");
	
	//Assigning the value to the string str2
	strcpy(str2, "hEllo");
	
	result = strcmp(str1, str2);
		*Any 3 with example (3*5)*	15

```
    if(result > 0) {
      printf("ASCII value of first unmatched character of str1 is greater than str2");
    } else if(result < 0) {
      printf("ASCII value of first unmatched character of str1 is less than str2");
    } else {
      printf("Both the strings str1 and str2 are equal");
    }

    return 0;
}
```

X a	```#include <stdio.h>```	Declare, read 2 Marks Logic 6 marks output writing 2	10

```
struct student
{
   char name[50];
   int roll;
   int mark1,mark2,mark3,total;
} s[10];

int main()
{
   int i;

   printf("Enter information of students:\n");

   // storing information
   for(i=0; i<10; ++i)
   {
      s[i].roll = i+1;

      printf("\nFor roll number%d,\n",s[i].roll);

      printf("Enter name: ");
      scanf("%s",s[i].name);

      printf("Enter marks: ");
      scanf("%d%d%d",&s[i].mark1,&s[i].mark2,&s[i].mark3);

      printf("\n");
   }
for(i=0; i<10; ++i)
   {
S[i].total=s[i].mark1+s[i].mark2+s[i].mark3;
}

   printf("Displaying Information:\n\n");
   // displaying information
   for(i=0; i<10; ++i)
   {
      printf("\nRoll number: %d\n",i+1);
      printf("Name: ");        printf("%d\n%d\n%d\n",s[i].mark1,s[i].mark2,
s[i].mark3,s[i].total);
   }
   return 0;
}
```

X b	Structure is collection of different data type. An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an	2+2+1	5

array of structure or object. An array is a collection of similar type, therefore an array can be of structure type.

```
struct struct-name
        {
            datatype var1;
            datatype var2;
            ----------
            ----------
            datatype varN;
        };

        struct struct-name obj [ size ];
struct Employee
    {
        int Id;
        char Name[25];
        int Age;
        long Salary;}
    };
```