QID : 4002

Revision: 2015 Course code: 4044

Course Title: Programming in C

Qst No.		Scoring indicators	Split up score	Sub total	Total
I	1.	Single line comment can be included by starting the comment line by '//' Multi line comment must be enclosed within '/*' and '*/'	1 1	2	10
	2.	Array, pointer,	2	2	
	3.	type ***variable_name[size];***	2	2	
	4	pow() – to find the power of any number sqrt() – to find the square root	1 1	2	
	5	Form 1: return; Form 2: return(expression);	1 1	2	
II	1.	Precedence is used to determine how an expression involving more than one operator is to be evaluated.. there are distinct levels of precedence, and an operator will belong to one of these levels. Operator at higher level of precedence will be evaluated first. Operators of same precedence will be operated either from left to right or from right to left. Following are operators mentioned in the order of decreasing precedence. • Highest precedence parentheses-() • Arithmetic unary operators +,-, increment/decrement (++/--) logical negation,1's complement, pointer reference(*), address(&), sizeof • Arithmetic operators- *, /, % • Addition +, subtraction – • Left, right shifts <<,>> • Relational operators <,<=,>,>= • Equality/inequality ==,!= • Bitwise AND & • Bitwise XOR ^ • Bitwise OR | • Logical AND && • Logical OR || • Conditional expression ?: • Assignment operators • Comma operators	2 4		

	2	Nested if-else is if-else structure within an if-else
		
		If(test-condition1)
		{
		If(test condition2)
		{
		Statements1;
		}
		
		Else
		{
		Statements2;
		}
		}
		Else
		{
		Statements 3;
		}
		Statement x;
		
		Else if ladder- for selecting from one of the many possibilities.
		Multipath decision making chain
		If(test condition1)
		Statements1;
		Else if(test condition2)
		Statements2;
		
		Else if(test condition3)
		Statements3;
		....
		Else if(test condition n)
		Statements n;
		
		Else
		Default statement;
			3
			
			3
				6	30

	3	Nested for loop. The outer being run for number of rows and inner loop run for the number of times as the number of columns. To read a matrix with m rows and n columns
		
		Int My_array[m][n];
		Int i,j;
		for(i=0; i<m; i++)
		{
		For(j=0; j<n; j++)
		{
		Scanf("%d",My_array[i][j];
		}
		printf("\n");
		}
			2
			
			4
				6	

	4.	While(test-condition)
		{
		Body of the loop
		}
		
		An entry controlled looping structure. Loop is executed only when test condition is TRUE.
		--- 	
		--- 	
		value=0;
		n,j=1;
		while(n<=10)
		{
		value=n+2*j;
		n++;
		j++;
		}
		printf("%d",value);
		--- 	
		--- 	
			2
			
			4
				6	

	5.	• Pointer variable declaration
		data_type *pt_name;
		declares a variable named pt_name to point to another variable of type *data_type.*
		
		• Pointer variable initialization can be done through & operator.
		• A variable can be accessed through its pointer through indirection operator
		Example:
		int a,b,*ptr; //declaring variable 'a' and pointer 'ptr'
		ptr=&a; //pointing 'ptr' to 'a'
		b=*ptr; //accessing variable through pointer
			3
			
			3		

	6.	User defined function is a provision through which one can subdivide a large program into smaller functional units. This reduces the length and complexity of a huge program and makes it easier to debug. To include a function in the main program it is to be declared. Function declaration specifies the types and number of arguments to be passed and the return type. A function that is declared to be defined through function definition. Function definition contains the body of the function. Function definition can be done outside the main function. A function call is an instance that invokes the function from the main function, or any other function. Function calling contains the actual parameters to be passed to the function.	3	6	

		The following program shows an example of declaring, defining and invoking a function that calculates the product of two integers and returns the product.
		
		main()
		{
		int n1,n2,pdt;
		int product(int,int);
		n1=20;
		n2=15;
		pdt=product(n1,n2);
		}
		
		int product(int a,int b)
		{
		int w;
		w=a*b;
		return(w);
		}
			3		

	7	Both are methods to pass data to a called function, or *parameter passing* Call by value: The values of actual parameters are copied to the variables in the parameter list of the called function. The called function works on the copy and not on the original values of the actual parameter. Call by reference: Also called pass by pointers, the memory address of the variables rather than the copies of the values are sent to the called function. The called function works directly on the data in calling function.	3 3	6	
III	(a)	(i) Integer Constant ○ Sequence of digits ○ Can be decimal, octal or hexadecimal ○ Decimal-combination of digits- 0 through 9 ○ Octal- combination of digits 0 through 7 with a leading 0, Ex: 037, 0, 0435 ○ Hexadecimal- from 0 through 9 and A through F Ex: 0X2, 0x9F ○ Largest that can be stored is machine dependent ○ Number followed by 0x or 0X considered to be hexadecimal ○ Larger integer values can be stored using qualifiers Ex: 56789U, 27864372UL (ii) Real Constants ○ Numbers containing fractional parts ○ Ex: 0..34, -23.75, +345.0 ○ Can be expressed in exponent notation as mantissa e exponent Ex: 0.65e4 equals 6500 (iii) Single character constant	2 2 2	8	15

		○ Single character enclosed in single quote (' ') mark. ○ Ex: '5' 'X' ';' ○ Character constants have integer values associated with them known as ASCII values. (iv) String constants ○ Sequence of characters enclosed in double quotes ○ Ex: "Hai" "1985" "5+3" "x" ○ Does not have equivalent integer value. (iv) Backslash character constants ○ Special characters used in output functions ○ Ex: '\n' –Newline, '\a'- audible bell, '\t'- Horizontal tab ○ Also called escape sequences.	2		

	(b)	#include<stdio.h>
		main()
		{
		Int sec,min,hr; //variable declaration
		Printf("ENTER THE TIME IN SECONDS-");
		scanf("%d",&sec); //reading the input
		hr=sec/3600; //calculating hours
		sec=sec%3600; //seconds left out after no. of hrs removed
		min=sec/60; //calculating the minutes
		sec=sec%60; // calculating the seconds
		printf("THE TIME IN SECONDS IS\n");
		printf("%d hours:%d minutes:%d seconds");
		}
			7	7	

IV	(a)	Arithmetic operators ○ Addition or unary plus→ a + b ○ Subtraction or unary minus → a-b ○ Multiplication → a*b ○ Division → a/b ○ Modulo division → a%b (remainder after dividing a by b)	3	8	15
		Relational Operator ○ Is less than → < → a<b ○ Is less than or equal to → <= → a<=b ○ Is greater than → > → a>b ○ Is greater than or equal to → >=→ a>=b ○ Is equal to → ==→ a==b ○ Is not equal to → != → a!=b	3		
		Logical Operator ○ Logical AND → && → a>b && b<25 Returns a TRUE value only when both a is greater than b AND b is less than 25 ○ Logical OR → || → a>b || b<25 Returns a TRUE value when either a is greater than b OR b is less than 25 ○ Logical NOT → !	2		

	(b)	main()
		{
		int a,b,c;
		printf("ENTER NUMBER1-");
		scanf("%d",&a);//reading number1
		printf("ENTER NUMBER2-");
		scanf("%d",&b);//reading number2
		printf("ENTER NUMBER3-");
		scanf("%d",&c);//reading number3
		if(a>b)//check whether num1 is largest
		{
		if(a>c)
		printf("NUMBER1 IS LARGEST-");
		else
		printf("NUMBER3 IS LARGEST-");
		}
		else
		{
		if(b>c)
		printf("NUMBER2 IS LARGEST-");
		else
		printf("NUMBER3 IS LARGEST-");
		}
		}
			7	7	

V	(a)	Deletion: Deleting specified element from an array
		
		void main()
		{
		Int num[20]={0}, j, k, n, p, t;
		clrscr();
		printf("\n ENTER THE NUMBER OF ELEMENTS-");
		scanf("%d",&n);
		printf("\n ENTER THE ELEMENTS-");
		for(j=0; j<n; j++)
		scanf("%d",&num[j]);
		printf("\n THE ELEMENTS ARE-");
		for(j=0; j<n; j++)
		printf("\t %d",num[j]);
		printf("\n ENTER THE ELEMENT TO DELETE-");
		scanf("%d",&p);
		p--;
		for(j=0; j<n; j++)
		{
		If(j>=p)
		Num[j]=num[j+1];
		}
		printf("\n THE MODIFIED ARRAY IS-");
		for(j=0; j<n; j++)
			4	8	15

```
        {
                If(num[j]!=0
                        Printf("\n %d", num[j]
        }
}
```

Searching: The process of seeking specific elements in an array is called searching | 4 | 7

```
void main()
{
        Int num[20]={0}, n, j,count=0;
        clrscr();
        printf("\n ENTER THE NUMBER OF ELEMENTS-");
        scanf("%d",&n);
        printf("\n ENTER THE ELEMENTS-");
        for(j=0; j<n; j++)
                scanf("%d",&num[j]);
        printf("\n THE ELEMENTS ARE-");
        for(j=0; j<n; j++)
                printf("\t %d",num[j]);
        printf("\n ENTER THE ELEMENT TO SEARCH FOR-");
        scanf("%d",&n);

        for(j=0; j<n; j++)
        {
                If(num[j]==n)
                {
                Printf("ELEMENT FOUND AT POSITION %d", j+1);
                count++;
                }

        }
If(count==0)
Printf("THE ELEMENT %d IS NOT FOUND IN THE ARRAY",n);
}
```

(b) WHILE LOOP | 3

```
While(test-condition)
{
        Body of the loop
}
```

While loop is an entry controlled loop. Test condition is evaluated, loop is executed only when the condition is TRUE. After execution of the loop the condition is tested again. If TRUE, the loop is again executed. This repeats till the condition finally results FALSE and loop exits.

Example: A segment of the program to find out $1^2+2^2+3^2+...10^2$

		---------- 	
		---------- 	
		sum=0;
		n=1;
		while(n<=10)
		{
		sum=sum+n*n;
		n=n+1;
		}
		Printf("sum=%d\n",sum);
		---------- 	
		---------- 	
		
		<u>DO-WHILE LOOP</u>
		
		Do
		{
		Body of the loop
		}
		While(*test-condition*)
		
		Do-while loop is an example of exit –controlled loop. The body of the loop is executed once before the condition in the while statement is tested. If the condition evaluates to TRUE the loop body is executed again. This continues till the condition evaluates to FALSE.
		Consider the following code snippet.
		
		---------- 	
		---------- 	
		i=1;
		Sum=0;
		{
		sum=sum+i;
		i=i+2;
		}
		While(sum<40||i<10)
		Printf("%d %d\n",I,sum);
		---------- 	
		---------- 	
		Here, the loop loop body executes once to calculate the value of sum and increment i by 2. Then it checks whether sum is less than 40 or i is less than 10. If TRUE the loop is again run else it exits the loop. Irrespective of the condition being TRUE or FALSE, the loop is executed once.
			4		

VI	(a)	`main()`
		{
		`int p,q,r,s,i,j,k,term=0;`
			8	8	15

```
    printf("ENTER THE DIMENSIONS OF FIRST
MATRIX-\n");
    scanf("%d %d",&p,&q);
    printf("ENTER THE DIMENSIONS OF SECOND
MATRIX-\n");
    scanf("%d %d",&r,&s);
    if(q!=r)
    {
        printf("THE MATRICES CANNOT BE
MULTIPLIED...!");
    }


    int mat1[p][q],mat2[r][s],pdt[p][s];

    //----Reading the matrices----------

    printf("ENTER THE ELEMENTS OF FIRST
MATRIX-\n");
    for(i=0;i<p;i++)
        {
          for(j=0;j<q;j++)
            {
                scanf("%d",&mat1[i][j]);
            }
        }

    printf("ENTER THE ELEMENTS OF SECOND
MATRIX-\n");
    for(i=0;i<r;i++)
        {
          for(j=0;j<s;j++)
            {
                scanf("%d",&mat2[i][j]);
            }
        }
    //-----displaying the matrices-----

    printf("THE FIRST MATRIX YOU ENTERED IS-
\n");
    for(i=0;i<p;i++)
        {
          for(j=0;j<q;j++)
            {
                printf("%3d",mat1[i][j]);
            }
            printf("\n");
        }
    printf("\n");

    printf("THE SECOND MATRIX YOU ENTERED
```

```
IS-\n");
    for(i=0;i<p;i++)
        {
          for(j=0;j<q;j++)
            {
                printf("%3d",mat2[i][j]);
            }
            printf("\n");
        }
    //-----matrix multiplication---

    for(i=0;i<p;i++)
        {
          for(j=0;j<s;j++)
          {
              for(k=0;k<q;k++)
              {

term=term+(mat1[i][k]*mat2[k][j]);
              }
              pdt[i][j]=term;
              term=0;
          }
        }

    //------printing the product--------

    printf("THE PRODUCT MATRIX IS-\n");
    for(i=0;i<p;i++)
        {
          for(j=0;j<s;j++)
            {
                printf("%5d",pdt[i][j]);
            }
            printf("\n\n");
        }
    printf("\n");
}
```

(b) 7

```
main()
{
int i,n,c=0,temp;//declarations

printf("\nENTER THE NUMBER OF ELEMENTS-");
scanf("%d",&n);//reading the number of
               //elements

int my_array[n];

printf("\nENTER THE ELEMENTS-\n");
```

```
for(i=0;i<n;i++)
     scanf("%d",&my_array[i]);//reading
                              //the elements

printf("\nTHE ARRAY YOU ENTERED IS-\n");
     for(i=0;i<n;i++)
     {
          printf("%6d",my_array[i]);
     }
while(c<n)//n-1 number of passes
     {
     for(i=0;i<n-1;i++)
          {
          if(my_array[i+1]<my_array[i])
               {
               temp=my_array[i];
               my_array[i]=my_array[i+1];
                              //swapping
               my_array[i+1]=temp;
               }
          }
     c++;
     }
printf("\n\nTHE SORTED ARRAY IS-\n");
```

VII	(a)	(a) scanf()
		scanf() function with %s format specification. But it terminates its input on the first white space it finds. Hence when a string ."NEW DELHI" only "NEW" will be read in.
		
		char address[10];
		scanf("%s",address);
		
		(b) getchar()
		getchar() function can read in a single character . We use this function repeatedly to read a string. To read a line of text we can use the function as follows.
		
		Char line[100];
		Int c;
		---------- 	
		Printf("Enter the text and press 'Enter'-");
		Do
		{
		character=getchar();
		line[c]=character;
		c++;
		}
		While(character!='\n');
		---------- 	
			3
			
			3
				8	15

		---------- (c) gets() Can read strings with white spaces. Available in the <stdio.h> header file. Syntax: gets(str); Where str is a string variable declared properly. It reads the characters into str from the keyboard until a newline character is encountered and then appends a null character at the end of the string. Ex: char line[80]; gets(line);	2		

	(b)	`int main()`
		`{`
		`int arr[10]; //declare integer array`
		`int *pa; //declare an integer pointer`
		`int i;`
		
		`pa=&arr[0]; //assign base address of array`
		
		`printf("Enter array elements:\n");`
		`for(i=0;i < 10; i++){`
		`printf("Enter element %02d: ",i+1);`
		`scanf("%d",pa+i); //reading through pointer`
		`}`
		
		`printf("\nEntered array elements are:");`
		`printf("\nAddress\t\tValue\n");`
		`for(i=0;i<10;i++){`
		`printf("%08X\t%03d\n",(pa+i),*(pa+i));`
		`}`
		
		`return 0;`
		`}`
			7	7	

VIII	(a)	`#include <stdio.h>`
		`#include <string.h>`
		`int main()`
		`{`
		`char text[20], reverse_text[20];`
		`int i,n, length = 0;`
		`printf("Enter text: ");`
		
		`gets(text);`
		`for (i = 0; text[i] != '\0'; i++)`
			8	8	15

```
  {
    length++; //this will calculate the length of given text
  }
  //Reverse the original text and store into reverse_text
  for (i = length - 1; i >= 0; i--)
  {
    reverse_text[length - i - 1] = text[i];
  }

  //Check whether reverse_text is same to original text
  for (n = 1, i = 0; i < length; i++)
  {
    if (reverse_text[i] != text[i])
    n = 0;
  }

  if (n == 1)
    printf("%s is a palindrome.", text);
  else
    printf("%s is not a palindrome", text);

  return 0;
}
```

(b)

```
main()
{
int n1,n2,temp,*ptrn1,*ptrn2;

printf("ENTER THE FIRST NUMBER-");
scanf("%d",&n1);
printf("ENTER THE SECOND NUMBER-");
scanf("%d",&n2);

ptrn1=&n1;
ptrn2=&n2;

//---------swapping the numbers---------
temp=*ptrn1;
*ptrn1=*ptrn2;
*ptrn2=temp;

//----------printing the results---------
printf("\n\nFIRST NUMBER AFTER SWAPPING IS-
%d\n",n1);
printf("SECOND NUMBER AFTER SWAPPING IS-
%d\n\n",n2);
}
```

OUTPUT:
ENTER THE FIRST NUMBER-100
ENTER THE SECOND NUMBER-500

7 | 7

		FIRST NUMBER AFTER SWAPPING IS-500 SECOND NUMBER AFTER SWAPPING IS-100			

IX	(a)	Depending on arguments present, the return value sends the result back to the calling function. Based on this the functions are divided into four types.
		1. Without arguments and return values:
		• Data is neither passed through the calling function nor sent back from the called function
		• Such functions may be useful to print some message, draw a line or split the line
		• Example, a program to display a simple message
		
		void main()
		{
		void message();
		Message();
		}
		void message()
		{
		puts("Have a nice day");
		}
		2. With arguments but with no return values:
		• Arguments are passed through the calling function. The called function operates on the values. But no result is sent back
		• Example: A program to display the date in proper format when the day month and year are passed as inputs.
		
		Int main()
		{
		Int dat(int,int,int);
		Int d,m,y;
		Printf("ENTER THE DATE date-month-year");
		Scanf("%d %d %d",&d,&m,&y);
		Dat(d,m,y);
		return 0;
		}
		
		dat(int x,int y,int z)
		{
		printf("Date - - > %d/%d/%d ",x,y,z);
		}
		Output
		ENTER THE DATE date-month-year 23 7 1985
		Date - - > 23/7/1985
			2
			
			2
				8	15

		3. With arguments and return values:
		Data is transferred between calling function and called function
		Example: Program to send values to user defined function and display the return value.
		Int main()
		{
		Int sum(int,int,int),a,b,c,s;
		Printf("Enter three numbers:");
		Scanf("%d%d%d",&a,&b,&c);
		S=sum(a,b,c);
		Printf("Sum=%d",s);
		Return 0;
		}
		
		Int sum(int x,int y,int z)
		{
		Return(x+y+z);
		}
			2		

		4. Without arguments but with return values:
		
		The called function is independent. It reads value from keyboard or generates the values from initialization and returns the value.
		Example:
		Int main()
		{
		Int sum(),a,s;
		S=sum();
		Printf("sum=%d",s);
		Return 0;
		}
		
		Int sum()
		{
		Int x,y,z;
		Printf("Enter three values ");
		Scanf("%d %d %d",&x,&y,&z);
		Return(x,y,z);
		}
			2		

	(b)	#include<stdio.h>
		
		void swap(int *a, int *b);
		
		int main()
		{
		int m = 22, n = 44;
		// calling swap function by reference
		printf("values before swap m = %d \n and n = %d",m,n);
		swap(&m, &n);
		}
			7	7	

		void swap(int *a, int *b)
		{
		int tmp;
		tmp = *a;
		*a = *b;
		*b = tmp;
		printf("\n values after swap a = %d \nand b = %d", *a, *b);
		}			

X	(a)	Recursion is a programming technique that allows the programmer to express operations in terms of themselves. In C, this takes the form of a function that calls itself.
		
		A simple example of recursion would be:
		
		`void recurse()`
		`{`
		`recurse(); /* Function calls itself */`
		`}`
		
		`int main()`
		`{`
		`recurse(); /* Sets off the recursion */`
		`return 0;`
		`}`
		
		Program for finding the sum of N natural numbers using function recursion
		
		`#include <stdio.h>`
		`int sum(int n);`
		
		`int main()`
		`{`
		`int number, result;`
		
		`printf("Enter a positive integer: ");`
		`scanf("%d", &number);`
		
		`result = sum(number);`
		
		`printf("sum = %d", result);`
		`return 0;`
		`}`
		
		`int sum(int num)`
		`{`
		`if (num!=0)`
		`return num + sum(num-1); // sum() function calls itself`
		`else`
		`return num;`
		`}`
			2
			
			2
			
			4
				8	15

Output

```
Enter a positive integer:4
sum = 10
```

(b)

```
void sort(int m, int x[ ]);
   main()
   {
       int i;
       int array1[5] = {40, 90, 73, 81, 35};

       printf("Array before sorting\n");
       for(i = 0; i < 5; i++)
          printf("%d ", array1[i]);
       printf("\n\n");

       sort (5, elements);
       printf("Array after sorting\n");
       for(i = 0; i < 5; i++)
          printf("%4d", array1[i]);
       printf("\n");
   }
   void sort(int m, int x[ ])
   {
       int i, j, t;

       for(i = 1; i <= m-1; i++)
          for(j = 1; j <= m-i; j++)
             if(x[j-1] >= x[j])
             {
                t = x[j-1];
                x[j-1] = x[j];
                x[j] = t;
             }
   }
```

Output

```
Array before sorting
40 90 73 81 35

Array after sorting
  35  40  73  81  90
```

7 | 7