CBCS SCHEME

USN [redacted] BPOPS103/203

First/Second Semester B.E./B.Tech. Degree Examination, June/July 2024
Principles of Programming using C

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Define Computer. Explain the various types of computer.	10	L2	CO1
	b.	Explain the basic structures of C program in detail. Write a sample program to demonstrate the components in the structure of C program.	10	L2	CO2
		OR			
Q.2	a.	Explain scanf( ) and printf( ) functions in C language with syntax and example.	08	L2	CO2
	b.	What is variable? Explain rules for constructing variable in C. Give example for valid and invalid variable.	06	L2	CO2
	c.	Illustrate the flowchart and write a C program which takes as input p, t, r compute the simple interest and display result.	06	L2	CO2
		Module – 2			
Q.3	a.	Explain the following operators in 'C': i) Relational ii) Logical iii) Conditional iv) Bitwise.	08	L2	CO2
	b.	Explain for loop statement with syntax and example program.	06	L2	CO2
	c.	Write a C program to simulate simple calculator that performs arithmetic operations using switch statement. Error message should be displayed if any attempt is made to divide by zero.	06	L2	CO3
		OR			
Q.4	a.	Explain if, if-else, nested if and cascaded if-else statements with syntax and example.	08	L2	CO2
	b.	Write a C program that takes three coefficient (a, b, c) to calculate roots of quadratic equation, print all possible roots with appropriate messages for a set of coefficients.	06	L2	CO5
	c.	Explain break and continue statements with respect while, do-while and for loops.	06	L2	CO2
		Module – 3			
Q.5	a.	Define function. Explain categories of user defined functions.	10	L2	CO4
	b.	Define two-dimension array. Write a C program to multiply 2 matrix by ensuring their multiplication compatibility.	10	L2	CO3
		OR			
Q.6	a.	Explain function call, function definition and function prototype with syntax and example for each.	10	L2	CO4
	b.	Write a C program to implement Binary search for integers.	05	L2	CO3
	c.	What is Recursion? Write a C program to compute factorial of number using recursion.	05	L2	CO3
		Module – 4			
Q.7	a.	Define string. Explain any four string manipulating functions with example.	10	L2	CO3
	b.	Write a C program to concatenate two strings without using built-in function strcat( ).	05	L2	CO3
	c.	Explain string unformatted input/output functions with example.	05	L2	CO3

		OR			
Q.8	a.	Define pointer. Explain pointer variable declaration and initialization with suitable example.	**08**	**L2**	**CO3**
	b.	Explain pass by value and pass by address with example.	**04**	**L2**	**CO3**
	c.	Write a C program using pointers to compute sum, mean, standard deviation of all elements stored in an array of n real numbers.	**08**	**L2**	**CO3**
		Module – 5			
Q.9	a.	Explain structure declaration and how structure member are accessed with example.	**10**	**L2**	**CO3**
	b.	Implement a structure to read, write and compute average marks and the students scoring above and below average of class N students.	**10**	**L3**	**CO5**
		OR			
Q.10	a.	Compare between structure and union with syntax and example.	**06**	**L2**	**CO3**
	b.	Explain fopen( ), fclose( ), fscanf( ) and fprintf( ) with syntax and example program considering all above functions.	**10**	**L2**	**CO4**
	c.	What are enumeration variable? How are they declared?	**04**	**L2**	**CO3**

* * * * *

Q.No.	Solution and Scheme	Marks
Q1a	A computer in simple terms, can be defined as an electronic device that is designed to accept data, perform the required mathematical and logical operations at high speed, and output the result.	1M
	Types of computers	
	@ Supercomputer : It is fastest, most powerful and most expensive computer. Supercomputers were first developed in the 1980's to process large amount of data and to solve complex scientific problems. A single supercomputer can support thousands of users at same time. Example of supercomputers CRAY-1, CRAY-2 control Data CYBER 205, and ETA A-10.	2M
	* Mainframe computers: These are large scale computers. These are very expensive and need large clean room with air conditioning, thereby making them very costly to deploy. Two types of terminals that can be used with mainframe system are as follows	2M
	Dumb terminal : It consist of only a monitor and keyboard	
	Intelligent terminal: It have their own processor and thus can perform some processing operations	

Q.No.	Solution and Scheme	Marks
	* Minicomputers: These are smaller, cheapes, and slowes than mainframes. They are called minicomputers because they were the smallest computer of thir times. Minicomputers are widly used in business, eduction, hospitals, government organisations. etc. The first minicomputer was introduced by Digital Equipment Corporation (DEC) in the mid 1960s. * Microcomputers: Microcomputers, commonly known as PCs are very small and cheap. The first microcomputers was designed by IBM in 1981 and was named IBM-PC. Later on many computers hardware companies copied this design and termed thir microcomputers as PC-compatible.	8M

1b	Preprocessor Directives
	Global Declarations
	main()
	{
	Local declarations
	Statements
	}
	function 1()
	{
	local Declaration
	Statements
	}
	- - - -
	function N()
	{ local declaration
	{ statements
		2M

Q.No.	Solution and Scheme	Marks

A c program is composed of preprocessor command, a global declaration section, and one or more functions (figure)

* The preprocessor directives contains special instructions that indicate how to prepare the program for compilation. All preprocessor commands start with symbol hash (#).
  Example. #include <stdio.h>

* Global declarations: This is where global variable and function prototypes are declared. These are accessible from any function within program.

* main function: This is an entry point of a c program. Execution start here. It is where the main logic of the program is implemented

* Functions: Functions contain blocks of code that perform specific task. They are called from the main functions or other functions.

8M

Example:

```
#include <stdio.h>          ⟹ preprocessor directive
int globalvar = 10;         ⟹ global declaration
void print();
int main()                  ⟹ main function
{
   print();
}
void print()                ⟹ function
{
   printf("The value of globalval = %d", globalvar);
}
```


2

Q.No.	Solution and Scheme	Marks

2a	scanf() function stands for scan formatting and is used to read formatted data from the keyboard.
	Syntax:
	scanf(" control string", arg1, arg2, arg3---agrn);
	control string specifies the types and format of the data that has to be obtained from the keyboard and stored in the memory location by arguments
	Prototype of control string
	% [*] [width] [modifies] type
	width: is an optional argument that specifies the maximum number of characters to be read
	Modifies: It is an optional argument that can be h, l or L for data pointed by corresponding additional argument.
	Type: Specifies the type of data that has to be read. int n; float f; char a[100];
	Ex: scanf("%d",&n);
	scanf("%f", &d);
	scanf("%s", a);
	scanf("%9s", a) // Read up to 9 characters
		2M
		2M

Q.No.	Solution and Scheme	Marks

printf() is used to display information required by the user and also prints value of variable

Syntax:

printf(" control string", variable_list);

prototype of control string

%[flag][width][.precision][lengthmodifier]type

Flag: - left-justify within given field width

+ Display data with its numeric sign

used to provide additional specifier like 0,x,X,0,0x

0 The number is left-padded with zero

4M

width: it is optional, which specifies the minimum number of positions in output

precision: Maximum number character to print

Length modifier: h, l or L

Type specifier: define the type and the interpretation of value of corresponding argument

c, d, f, E, e, o, u, X, x

Example: printf(" %d %c %f", 12, 'a', 2.3);

printf(" %.6.2f", 245.37154); o/p 245.37

printf(" num is %6d", 12);

o/p num is ____12

str[] = Good Morning

printf("%.20s", str);

printf("%20.10s", str);

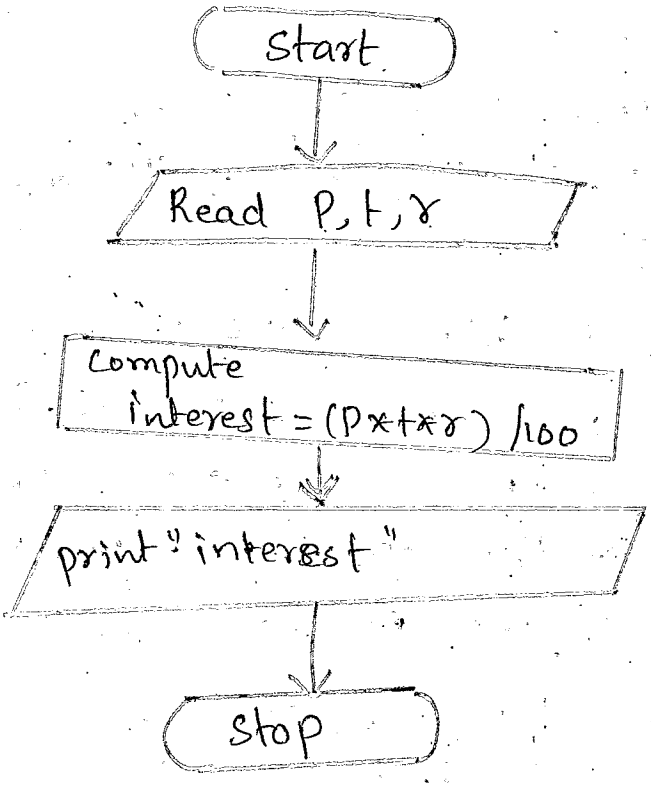
6

Q.No.	Solution and Scheme	Marks
2b	variable is an identifies whose value can be changed during execution of program.	1M
	Rules for defining a variable *) The first character in variable should be a letter or an underscore *) The first character can be followed by letter or digit or underscore *) No extra symbol are allowed * length of variable can be up to maximum of 31 characters * Keywords should not be used as variable names. * C is case sensitive so the case of alphabetic that form variable name is significant. * space should not use to frame variable.	5M

	valid example
	a, principle_amount, sum_of_digits.
	
	invalid example
	3fack // voilate Rule 1
	Sum = sum 62$ // voilalate Rule 2
	for int if // voilate rule 5	

Q.No.	Solution and Scheme	Marks
28 2C		6M

```
#include <stdio.h>
int main()
{
    float P, t, r; interest;
printf(" Enter P, t, r values ");
scanf(" %f %f %f ", &P, &t, &r);
    interest = (P * t * r) /100;
printf(" Interest = %f ", interest);
return 0;
}
```



Q.No.	Solution and Scheme	Marks
3a		27

i) Relational : These are known as comparision operators, is an operator that compare two values Expression that contains relational operators called as relational expression

operator	Meaning	Example
<	Less than	3<5 gives 1
>	greater than	7>9 gives 0
<=	less than or equal	100<=100 gives 1
>=	greater than or equal	50>=100 gives 0

2M

ii) Logical operators :

c language suppor three logical operators logical AND (&&), logical OR (||) and Logical NOT (!)

Logical AND : It is binary operator which simultaneously evaluates two values, If both operand are true then whole expression true if both or one of the operand expression evalution false then evaluates false

Logical OR : It returns false if both operand or false otherwise it return true

Logical NOT : It takes single operand . It produce a '0' (zero) if expression evaluates to non-zero value and produce 1 if expression produces a zero

Q.No.	Solution and Scheme	Marks

Truth table for AND, OR and NOT.

A	B	A&&B	A!!B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A	A!
1	0
0	1

2M

Example

(a<b) && (a<c)

(a>b) !! (b>c)

int a=10, b;

b = !a;

iii) Conditional: The conditional operator or the ternary (?:) is just like an if-else statement that can be used within expressions

Syntax: exp1 ? exp2 : exp3

Exp1 is evaluted first. If it is true then exp2 is evaluated and becomes the result of expression, otherwise exp3 is evaluated and becomes the result of the expression.

Ex: large = (a>b) ? a : b

```
int a=5, b=3, c=7, small;
small = (a<b ? (a<c ? a:c) : (b<c ? b:c));
```

2M

iv) Bitwise: Those operators that perform operations at bit level. These operators bitwise AND, bitwise OR, bitwise XOR, and shift operators.

* Bitwise AND: In this bit in the first operand is ANDed with corresponding bit in second operator: Ex: int a=10, b=20, c=a&b

* Bitwise OR: In this bit in the first operand

5

Q.No.	Solution and Scheme	Marks

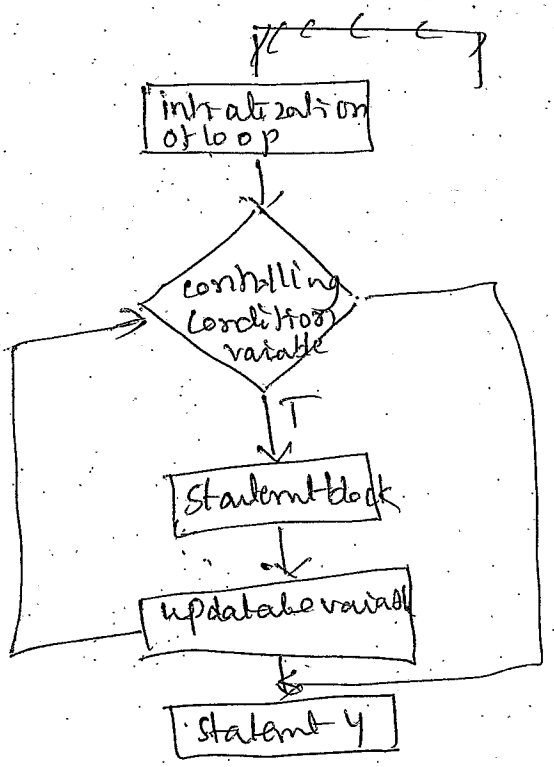
	is ORed with corresponding bit in the second operand
	Ex: int a=10, b=20, c=0;
	c=a|b;
	Bitwise XOR: bitwise XOR operator (^) perform operation on individual bit of the operands.
	10101010 ^ 01010101 = 11111111
	int a=10, b=20, c=0;
	c=a^b;
	Bitwise NOT: It is unary operator performs logical negation on each bit of the operand. by performing negation of given binary numbers.
	~10101011 = 01010100
	Shift operator: shift left <<, shift right >>
	operand op num
	example: x<<1 produce 01110100
	x=00011101
	x<<4 produce 11010000
	x>>1 produce 00001101
	x>>4 produce 00000001
		2M

3b	for loop provides a mechanism to repeat a task until particular condition is true. for loop is usally known as determinate or definite loop because the programmes knows exactly how many times the lopp will repeat. * when for loop is used loop variable initialized only once. with every iteration of the loop the value of loop variable is updated and condition checked.	

Q.No.	Solution and Scheme	Marks

Syntax of for loop

```
for (initialization; condition; inc/dec/update)
{
    statement block;
}
statement-y;
```



Example

```
#include <stdio.h>
int main()
{
    int i, n=5;
    for (i=1; i<=n; i++)
    {
        printf("%d", i)
    }
    return 0;
}
```

6M

3c

```
#include <stdio.h>
int main()
{
    int num1, num2;
    char operator;
    printf("Enter two number \n");
    scanf("%d %d", &num1, &num2)
    printf("Enter operator for which operation
                should perform");
    scanf("%c", &operator);
switch(operator)
{
case '+' : printf("Result = %d ", num1+num2);
           break;
```


6

Q.No.	Solution and Scheme	Marks

```
case '-' : printf("Result = %d", num1 - num2);
           break;
case '*' : printf("Result = %d", num1 * num2);
           break;
case '/' : if (num2 == 0)
             printf("Division by zero not allowed");
           else
             printf("Result = %d", num1 / num2);
           break;
default: printf("Error unsupported operator\n");
           break;
}
return 0;
}
```

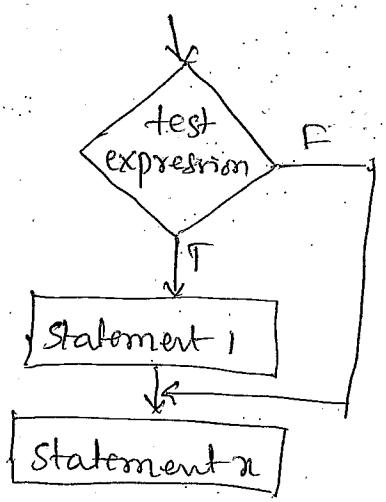
6M

4a

if statement

Syntax of If statement

```
if (test expression)
{
  statement 1;
  .
  .
  statement n;
}
statement x
```



Example:

```
#include<stdio.h>
int main()
{
  int x=10
  if(x>0)
```

Q.No.	Solution and Scheme	Marks

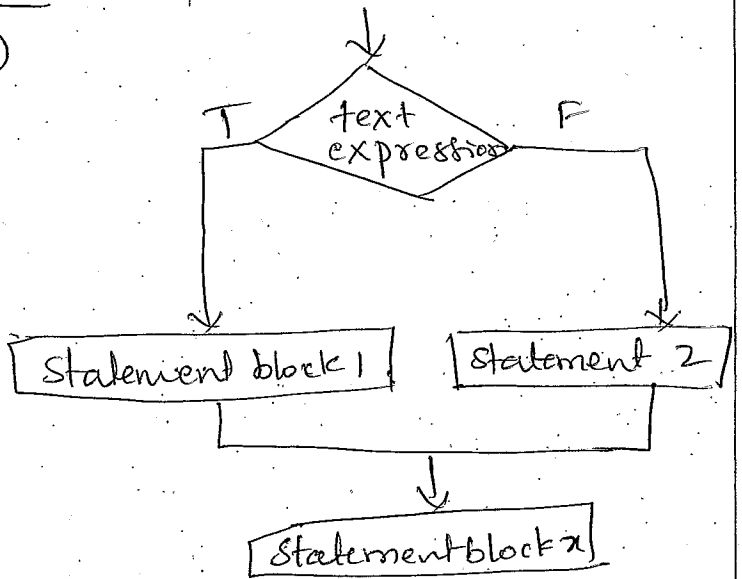
```
    x++;
    printf("x=%d",x);
    return 0;
}
```

If statement is simplest form of decision control statement that is frequently used in decision making. — 2M

if-else statement; Its usage is very simple the test expression is evaluated. If the result is true the statements followed by the expression is executed else if expression is false, the statements skipped by compiles.

Syntax of if-else

```
if (test expression)
{
  statement block 1;
}
else
{
  statement block 2;
}
Statement blockx;
```



2M

example:

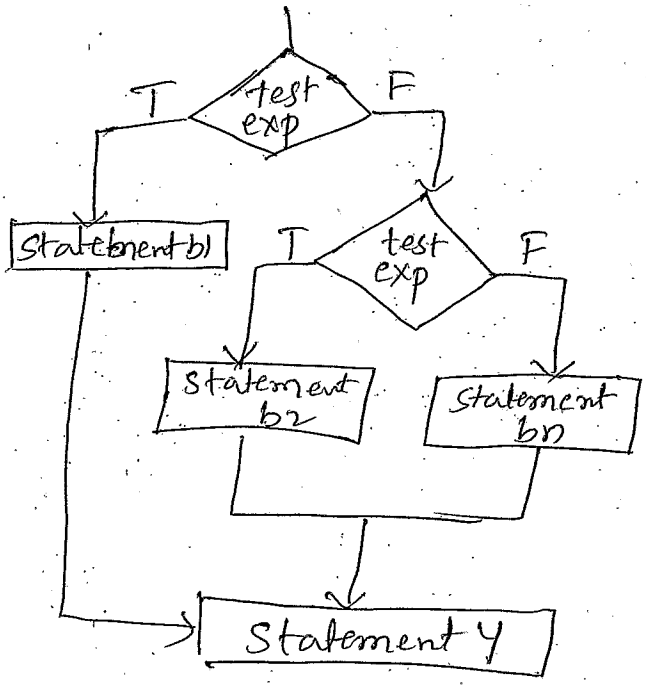
```
#include<stdio.h>
int main()
{
  int num=2;
  if(num%2==0)
  printf("%d is even", num);
  else
   printf("%d is odd", num);
 } return 0;
```

Q.No.	Solution and Scheme	Marks

if else if

Syntax :

```
if (test expression)
{
  statement b1;
}
else if (exp)
{
 statement b2;
}
- - - - -
- - - - - -
else
{
 statement bn;
}
statement y;
```



2M

Example

```
#include<stdio.h>
int main()
{
  int x=10, y=6;
   if (x==y)
    printf("Equal");
  else if (x>y)
    printf("x is greater");
  else
    printf("y is greater");
  return 0;
}
```

Q.No.	Solution and Scheme	Marks

nested if statements

which refer to placing one if statement inside another. This is useful when you need to make multiple, dependent decisions.

Syntax

```
if (condition)
{ if (condition)
   { statement 1;
   }
   else
   { statement 2;
   }
}
else
{
   if (cond)
   { statement 3; // optional.
   }
}
```

2M

Example.

```
#include<stdio.h>
int main()
{ x=10;
  y=20;
  if(x>5)
  { if (y>15)
      { printf("x is greater than 5 and
                y is greater than 15");
      }
    else printf("x is greater than 5 but y
                is not greater than 15");
  }
  else printf("x is not greater than 5");
}
```


8

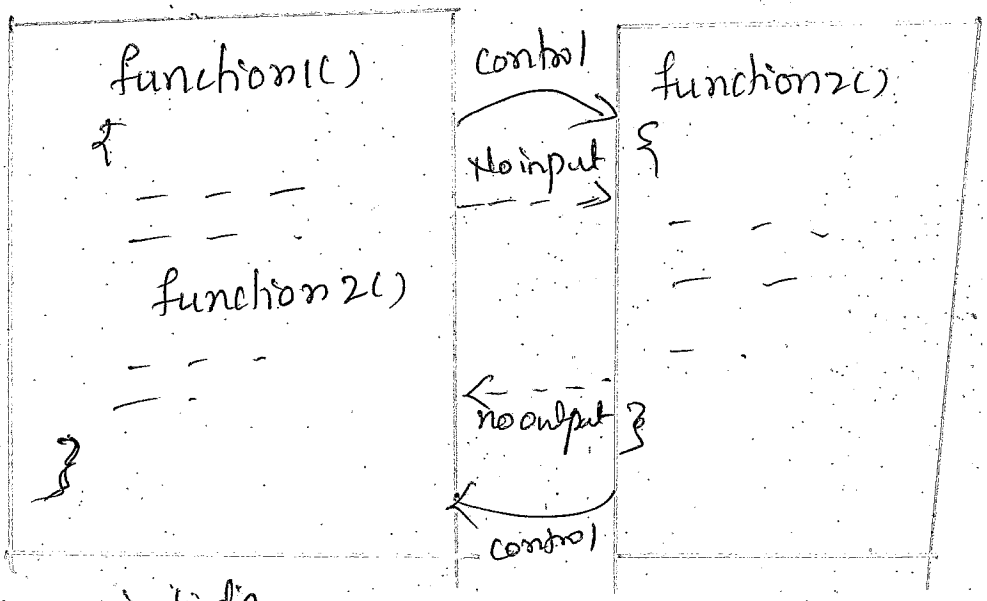
Q.No.	Solution and Scheme	Marks
5a	function: C enable programmer to break up a program into segments commonly known as functions. Each of which can be written more or less independently of the others.	1M

Categories of functions

* Function with no arguments and no return value
* Function with arguments and no return values
* Functions with arguments and one return values
* Functions with no argument but return values.
* Function that return multiple values.

* Function with no argument and no return value

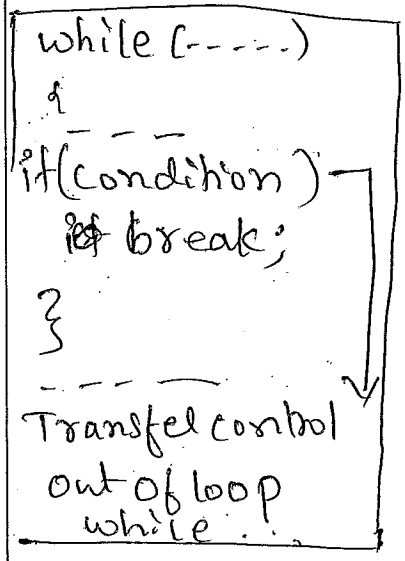
when function has no argument, it does not receive any data from the calling function. Similarly when it does not return value, the calling function does not receive any data from the called function. 2M

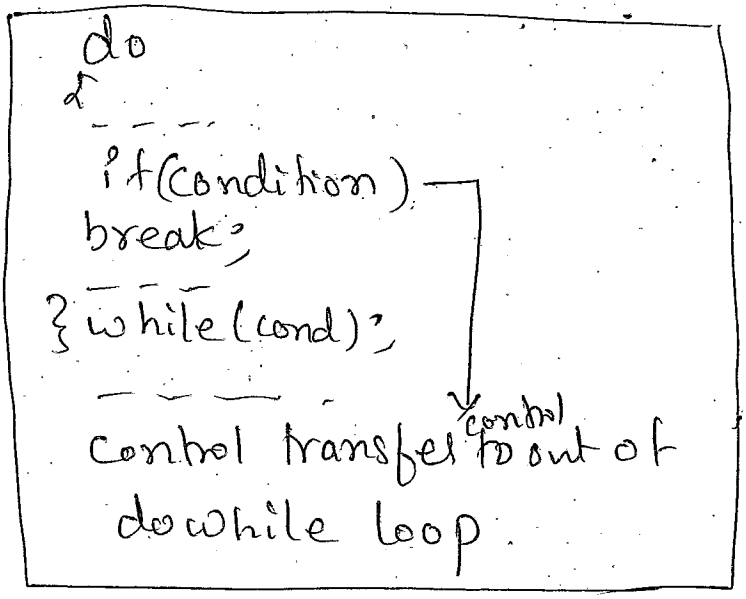


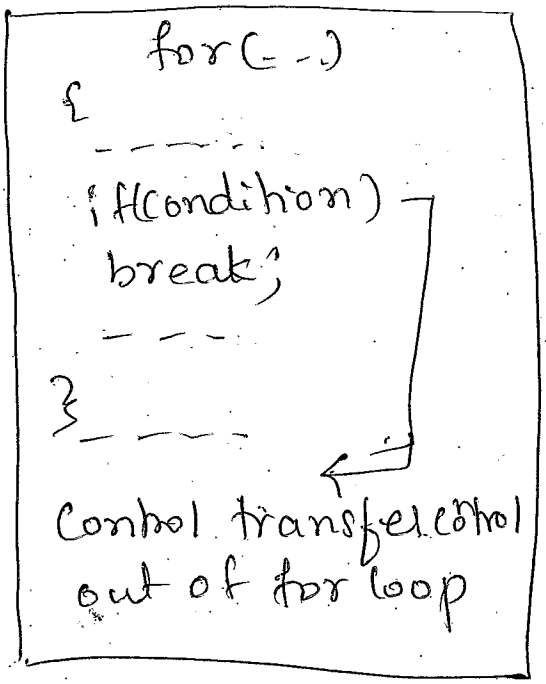

```
#include
void print();
int main()
{
  print()
}
```

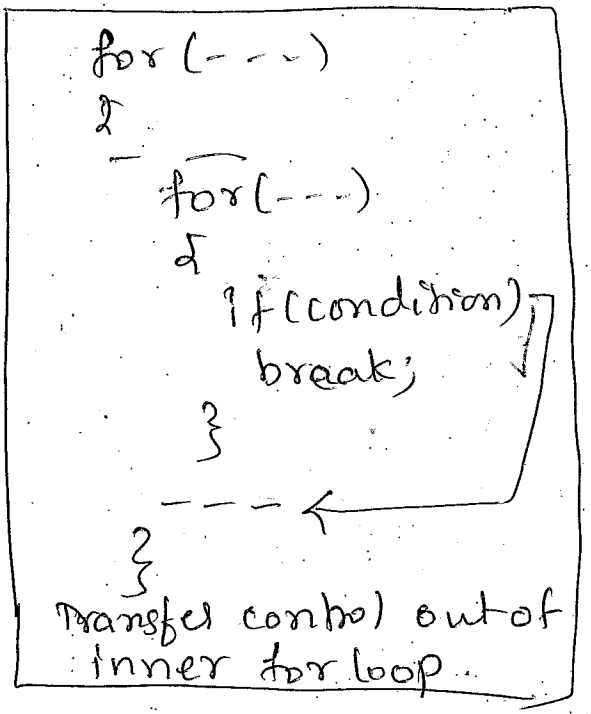
```
void print()
{
  printf("Hello");
}
```

Q.No.	Solution and Scheme	Marks
46	Break is used to terminate the execution of the nearest enclosing loop in which it appears.	3M









Example ⇒ Break

```
#include<stdio.h>
int main()
{
  int i=1;
  while (i<=10)
  { if(i==5)
      break;
    printf("%d", i);
    i++;
  }
}
```

o/p = 1 2 3 4

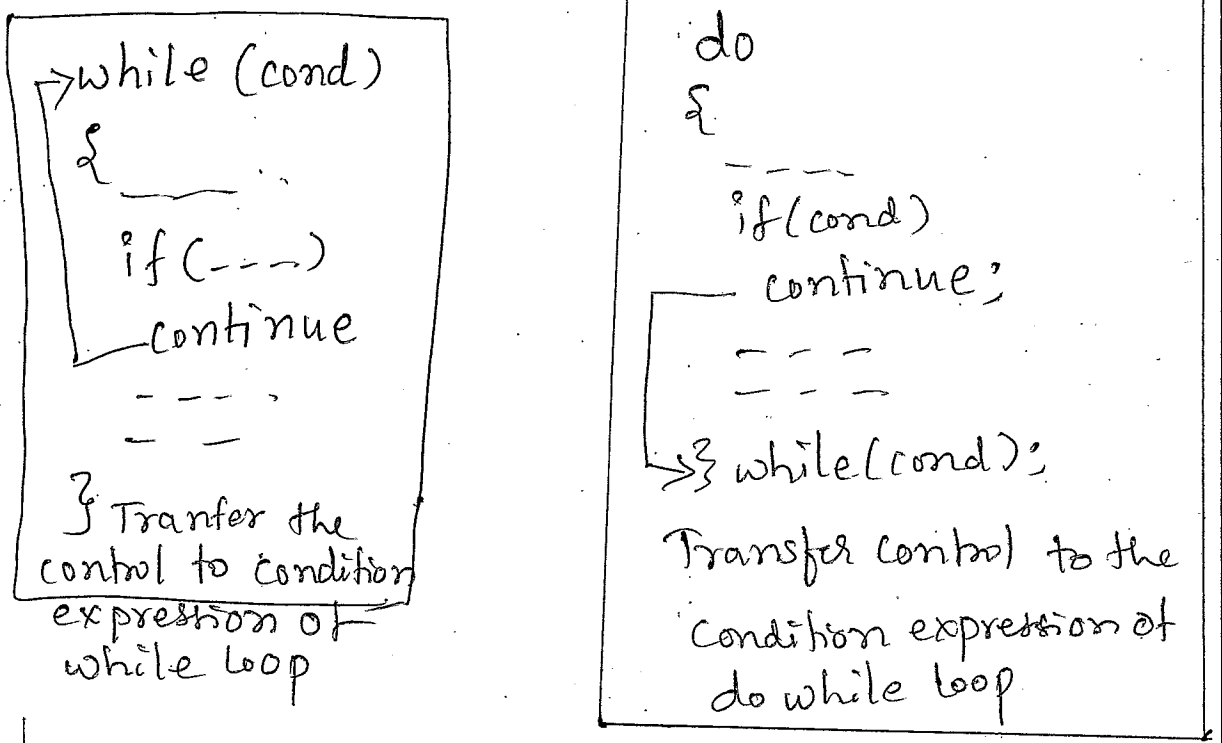
continue

```
#include<stdio.h>
void main()
{ int i;
  for(i=1; i<=10; i++)
  { if (i==5)
      continue;
    printf("%d", i);
  }
}
```

o/p 1 2 3 4 6 7 8 9 10

Q.No.	Solution and Scheme	Marks

4b	#include<stdio.h>
	void main()
	{ float a,b,c,d,r1,r2;
	printf("Enter co-efficent of Quadratic eqn");
	scanf("%d%d%d",&a,&b,&c);
	if(a==0)
	{ printf("Invalid Inputs\n");
	exit(0);
	}
	d=b*b-4*a*c;
	if(d==0)
	{ printf("Root are equal & real\n");
	r1=-b/(2*a); r2=-b/(2*a);
	printf("R1=%f, r2=%f",r1,r2);
	}
	else if(d>0)
	{ printf("Roots are real & Distinct\n");
	r1=(-b+(sqrt(fabs(d))))/(2*a);
	r2=(-b-(sqrt(fabs(d))))/(2*a);
	printf("r1=%f, r2=%f",r1,r2);
	}
	else
	{ printf("Roots are Imaginary\n");
	r1=-b/(2*a);
	r2=sqrt(fabs(d))/(2*a);
	printf("R1=%f+i%f",r1,r2);
	printf("R2=%f-i%f",r1,r2);
	}
	}
		6M

Q.No:	Solution and Scheme	Marks
26	~~goto~~ continue; The continue statement tells the compiler "skip the following statements and continue with next iterations The use of the continue statement in loop is in following - 	3M

Q.No.	Solution and Scheme	Marks

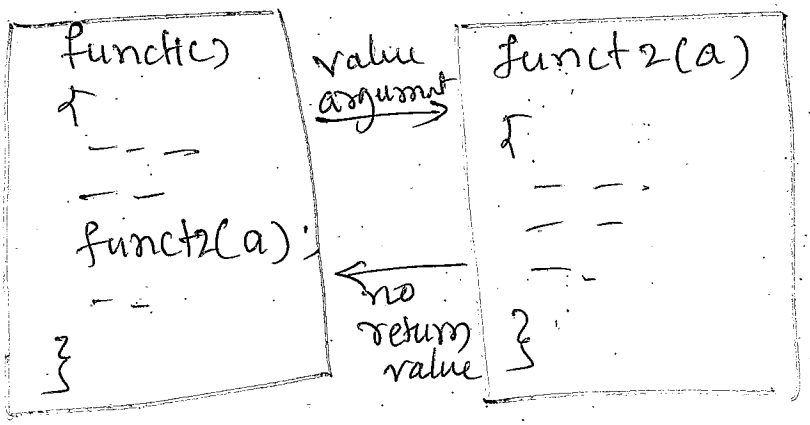
* No argument but returns a value

There could occasion where we may need to design functions that may not take any arguments but returns a value to the function

```
int get_number(void);
main()
{
  int m = get_number();
  printf("%d", m);
}
int get_number(void)
{
   int number;
   scanf("%d", &number);
   return(number);
}
```

2M

* Function that ~~Returns~~ with argument but no return values.

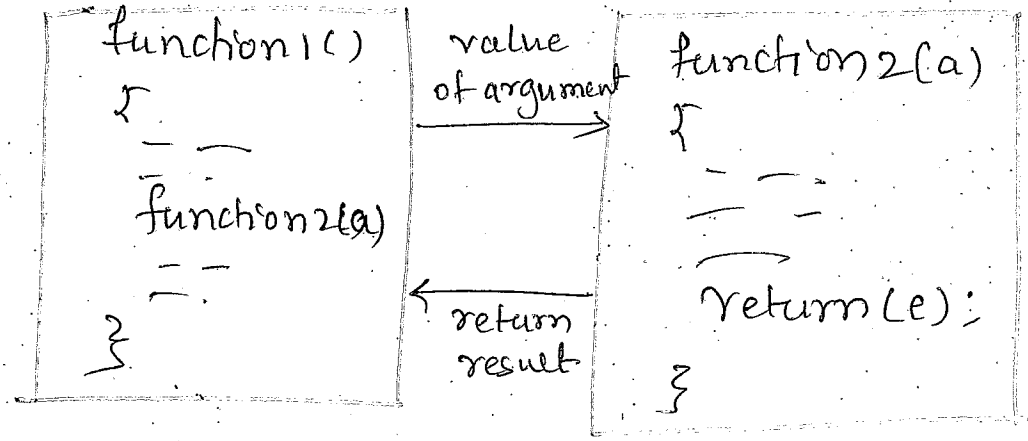


2M

Example
```
#include<stdio.h>
void sum(int a, int b);
int main()
{
  int x=8, y=9;
  sum(x,y);
}
void sum(int a, int b)
{
  int r;
  r=a+b;
  printf("%d", r);
}
```

Q.No.	Solution and Scheme	Marks

* function with argument and return value
* function receives data from calling function through arguments and return value to called function.



2M

```
#include<stdio.h>
int sum(int a, int b)
int main()
{
  int r;
  r = sum(10, 5)
  printf("sum=%d", r);
}
```

```
int sum(int a, int b)
{
  int r;
  r=a+b;
  return(r);
}
```

* Function with Multiple returns.

To get more information from a function. The arguments not only to receive information but also to send back information to the calling function. The mechanism of sending back information through arguments is achieved using what are known as addres operator(&) and indirection operator(*). 1M

```
void mathope(int x, int y, int *s, int *d);
main()
{
  int x=20, y=10, s, d;
  mathope(x, y, &s, &d)
  printf("%d %d", s, d);
}
```

Q.No.	Solution and Scheme	Marks

36	#include<stdio.h>
	void main()
	{ int a[5][5], b[5][5], c[5][5]={0}, m,n,p,q,i,j,k;
	printf("Enter size of 1st matrix");
	scanf("%d %d", &m, &n);
	printf("Enter size of 2nd matrix");
	scanf("%d %d", &p, &q);
	if(n!=p)
	{ printf("Matrix multiplication not possible\n");
	getch(); exit(0);
	}
	else
	{ printf("Enter element of 1st matrix\n");
	for(i=0; i<m; i++)
	{ for(j=0; j<n; j++);
	scanf("%d", &a[i][j]);
	}
	printf("Enter element of 2nd matrix\n");
	for(i=0; i<p; i++)
	{ for(j=0; j<q; j++)
	scanf("%d", &b[i][j]);
	}
	for(i=0; i<m; i++)
	{ for(j=0; j<q; j++)
	{ c[i][j]=0
	for(k=0; k<n; k++)
	{ c[i][j] += a[i][k] * b[k][j];
	} } }
	printf("Product of two Matrix is\n");
	for(i=0; i<n; i++)
	{ for(j=0; j<q; j++)
	{ printf("%d", c[i][j]);
	} } }
		10M

37		

Q.No.	Solution and Scheme	Marks

6a	Function prototype :- Before using, compiler must know about
	⇒ numbes of parameter and tepe of parameter
	⇒ that function expects to receive.
	⇒ data type of value that it will return to calling function
	Syntax!
	`return_data_type function_name(datatype var1, datatype var2 - - - - );`
	Function name ~~valid~~ is valid name
	return data type is data type of value that will returned to calling function as result.
	var1, var2 variable and their data type.
	Example.
	`void swap (int a, int b)`
	`float avg (int a, int b)`
	return type. function name. variable
	* Function definition: when function is defined space is allocated for that function in memory function definition comprises two parts.
	* Function header * Function Body.
	Syntax:
	`return_datatype func_name(data_type var1, data type var2, - - -)` ⇒ function Header
	`{ Local variable statement return(variable) }` ⇒ Function Body
	`}`
		2.5M
		2.5M

Q.No.	Solution and Scheme	Marks

Function Call: function call statement invoke function

* when function is invoked compiler jump to called function execute the statement that are part of function.
* Once function execute it return back to calling function.

2.5M

Syntax: function_name(val1, val2, ------);

Example mul(a, b)
mul(2, 5)

Example:

```
#include <stdio.h>
int sum(int a, int b);          <- Function declaration
int main()
{
  int n1, n2, total=0;
  printf(" Enter two number ");
  scanf("%d %d", &n1, &n2);
  total = sum (n1, n2);         => Function call
  printf(" total = %d", total);
}
int sum (int a, int b)          => called function / function definition
{
  int res;
  res = a+b;
  return res;                   => Function body
}
```

2.5M

Q.No.	Solution and Scheme	Marks

6b	#include <stdio.h>
	void main()
	{
	int a[50], key, i, n, low, high, mid, found=0;
	Printf("Enter the number of elements in array\n");
	scanf("%d", &n);
	Printf("Enter the elements of the array\n");
	for(i=0; i<n; i++);
	scanf("%d", &a[i]);
	Printf("Enter the key to be searched\n");
	scanf("%d", &key);
	low=0; high=n-1;
	while(low<=high)
	{
	mid=(low+high)/2;
	if(a[mid]==key)
	{
	found=1;
	break;
	}
	if(a[mid]>key)
	high=mid-1;
	else
	low=mid+1;
	}
	if(found)
	Printf("%d is present at position %d", mid+1);
	else
	printf("key not found");
	}
		5M

Q.No.	Solution and Scheme	Marks
6C	Recursion is ~~progra~~ function calls itself in order to solve problem. function which is used in recursion is call recursive function.	1M

	`#include <stdio.h>`
	`int fact(int);`
	`int main()`
	`{`
	`int num, factorial;`
	`printf("Enter number: ");`
	`scanf("%d", &num);`
	`factorial = fact(num);`
	`printf("factorial = %d", factorial);`
	`return 0;`
	`}`
	`int fact(int n)`
	`{`
	`if(n==1)`
	`return 1;`
	`else`
	`return (n*fact(n-1));`
	`}`
		4M

7a	string is null terminated character array string manipulation functions * strcat function Syntax: char *strcat(char *str1, const char *str2); strcat function appends the string pointed to by str2 to the end of the string pointed to by str1. The terminating null character of str1 is overwritten. The process stop when the terminating null character of str2 is copied.	1M

30

Q.No.	Solution and Scheme	Marks

The argument str1 is returned.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[10] = "programming";
    char str2[] = "In C";
    strcat (str1, str2);
    printf(" str1 = %s", str1);
    return 0;
}
```

strncat function

Syntax:

```
char *strncat (char *str1, const char *str2,
                                size_t n);
```

function append string pointed by str2 to end of string pointed by str1 up to n characters.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[50] = "Programming";
    char str2[] = "In C";
    strncat (str1, str2, 2)
    printf(" str1 = %s", str1); return 0;
}
```

strcpy function

Syntax:

```
char *strcpy (char *str1, const char *str2)
```

This function copies string pointed to by str2 to str1 including null char of str2.

```
#include <stdio.h>
#include <string.h>
void main()
```

PTO

12

Q.No.	Solution and Scheme	Marks

	{ char str1[10], str2[10] = "Hello"};
	strcpy(str1, str2);
	printf("str1 = %s", str1);
	}
	strncpy function
	Syntax:
	char *strncpy(char *str1, const char *str2, size_t n);
	This function copies upto n characters from string pointed to by str2 to str1.
	#include <stdio.h>
	#include <string.h>
	int main()
	{
	char str1[10], str2[50] = "Hello";
	strncp(str1, str2, 2);
	printf("str1 = %s", str1);
	return 0;
	}
		9m

7b	#include <stdio.h>
	void my_strcat(char str1[], char str2[])
	{
	int i, j;
	i = 0;
	while(str1[i] != '\0') i++;
	j = 0;
	while(str2[j] != '\0')
	{ str1[i++] = str2[j++];
	}
	str1[i++] = '\0';
	} printf("concatinated string %s", str1);	

34.

Q.No.	Solution and Scheme	Marks

	int main()
	{
	char str1[50], str2[50];
	int res1, res2;
	printf("Enter string one \n");
	gets(str1);
	printf("Enter string two \n");
	gets(str2);
	my-strcat(str1, str2);
	}
		5 m

7C	gets(): which read an entire line from the standard input until a newline character is encountered or EOF is reached. It store the line into the provided Buffer.
	char Buffer[100];
	printf("Enter string");
	gets(Buffer);
	getchar(): which read single characters from standard input. It often used in loop for characters by character processing.
	char ch[100];
	getchar(ch);
	puts(): which write string to the standard output and automatically appends newline character at the end.
	syntax: int puts(const char *str);
		2 1/2 m.

Q.No:	Solution and Scheme	Marks

```
char *str[10] = "Hello";
puts(str);
```

2.5M

putchar() which is print single character to standard output. if we use in loop can print character by characters

```
char str = "Hello";
putchar(str);
```

8a pointer is variable which contains address of the another variable. 1M

which contain memory address as their value

Declaring pointer variable

Syntax :

```
datatype *pt_name;
```

- * tell that variable pt_name is pointer variable
- pt_name need memory location 3.5M
- pt_name point the variable of type data type.

Example int *p;

p is pointer variable that point to integer data type.

```
float *x;
int *p
```

p [?] pointer to unknown location. Contains garbage.

pointer declaration styles

```
int *p;
int* p;
int * p;
```

Q.No.	Solution and Scheme	Marks

3.2	**Initialization of pointer variable**
	The process of assigning address of variable to pointer variable is known as initialization.
	`int a;`
	`int *p;` ⇒ declaration of pointer
	`p = &a` ⇒ Initialization
	we can also write this as
	`int *p = &a;`
	--- 	
	`int b;`
	`int *p=&a;`
	`int a;`
	} wrong because assigned before declaration of variable 'a'
	--- 	
	`float a, b;`
	`int x, *p;`
	`p = &a` ⇒ wrong because datatype mismatched.
	we could also define pointer variable with value NULL or 0 (zero)
	`int *p = NULL;`
	`int *p = 0;`
		3.5M

8b	**Call by value**: In this method the called function creates new variables to store the value of arguments passed to it. ⇒ called function uses a copy of the actual argument to performs its intended task. ⇒ If the called function is supported to modify the value of the parameters passed	

Q.No.	Solution and Scheme	Marks

to it, then changes will be reflected only in called function

```
main()
{ int n=2
  add(n);
  printf("n = %d", n);
}
add(int n)
{
  n=n+10
  Printf("n = %d", n);
}
```

2m

Call by Reference & or address

In this method address passed. to when function wants to modify the value of argument it to pass argument using call by reference technique.

⇒ Modify values are reflected both in called function and calling function.

2m

```
main()
{
  int n=2;
  add(&n)
  printf("n = %d", n)
}
void add(int *n)
{
  *n = *n+10;
  printf("n = %d", *n);
}
```

3½

18

Q.No.	Solution and Scheme	Marks
8C		8 M

```
#include<stdio.h>
#include<math.h>
void main()
{
  float a[10],*ptr,mean,std,sum=0;
  float sumstd=0;
  int n,i;
  printf("Enter no.of elements\n");
  scanf("%d",&n);
  printf("Enter the array elements\n");
   for(i=0;i<n;i++)
    scanf("%d",&a[i]);
    ptr=a;
    for(i=0;i<n;i++)
    {sum=sum+*ptr;
     ptr++;
    }
    mean=sum/n;
     ptr=a;
     for(i=0;i<n;i++)
     {sumstd=sumstd+pow((*ptr-mean),2);
      ptr++;
     }
     std=sqrt(sumstd/n);
     printf("sum=%f",sum);
     printf("Mean=%f",mean);
     printf("Standard dev=%f",std);
}
```

22.

Q.No.	Solution and Scheme	Marks
9a	structure is user defined data type that can store related information together.	

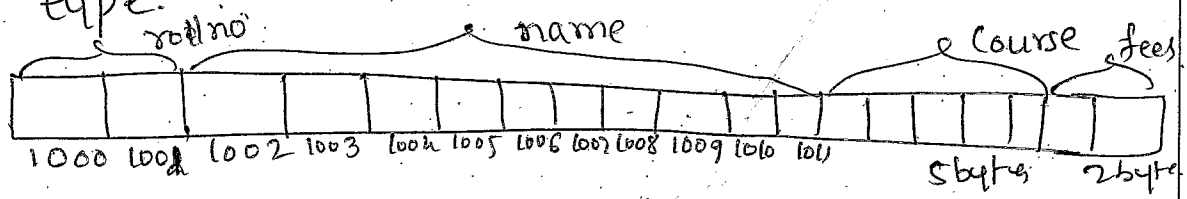
Structure Declaration

A structure is declared using struct followed by structure name.

Syntax:
```
struct struct_name
{
  datatype var_name;
  datatype var_name;
  - - -
  - - -
} var1, var2 - - -;
```

Example:
```
struct student
{
  int rollno;
  char name[10];
  char course[5];
  float fees;
} stud;
```

now this structure becomes user defined data type.



Declare structure variable

After defining structure format. we can declare variable of that type. A structure variable is declare ~~variable~~ same as declaration of variable of any other data type but little difference is data type type is structure struct type

Q.No	Solution and Scheme	Marks

⇒ Keyword struct
⇒ structure tagname.
⇒ list of variable name separated by commas
⇒ A terminating semicolon.

Example for structure student after declaration of structure.

```
struct student stud, stud1, stud2;
or
struct student
{
  ---
  ---
  ---
}stud, stud1, stud2;
```

Initialization of structure

5M

Syntax:

```
struct struct_name
{
 datatype  member name1
 datatype  member name2
 ---
 ---
 datatype  member namen
} strct_var = {const1, const2, ---, costnn};
or
struct struct_name
{
 datatype member1;
 ---
 ---
 datatype meben;
}
struct struct_name = {const1, const2, --- constn};
```

Q.No.	Solution and Scheme	Marks

Example:

```
struct student
{
   int rollno;
   float fees;
} stud1={28, 40000};
```

Accessing member of structures:

Each member of structure can be used just like normal variable, but its name will be a bit longer.

⇒ structure member variable is generally accessed by . (dot) operator.

Syntax: struct_variable.member name;

Example: for student structure.

```
stud1.rollno = 28;
stud1.fees   = 40000;
```

SM

To input value for data members of structure

```
scanf("%d", &stud1.rollno);
scanf("%d", &stud1.fees);
```

To print value of structure variable.

```
printf("%d", stud1.rollno);
printf("%d", stud1.fees);
```

And if are having pointers to structure variable then we are using → to accen the member

Q.No.	Solution and Scheme	Marks
9b		

```
#include<stdio.h>
#include<conio.h>
struct student
{
  char name[30];
  char usn[11];
  int m1, m2, m3;
  int total;
} s[100];
void main()
{
  int n, i;
  double avg=0.0;
  printf("Enter number of student\n");
  scanf("%d", &n);
  printf("Enter student details");
  for(i=0; i<n; i++)
  {
    printf("Name: ");
    scanf("%s", s[i].name);
    printf("USN");
    scanf("%s", s[i].usn);
    printf("Enter m1, m2, m3, marks");
    scanf("%d%d%d", &s[i].m1, &s[i].m2, &s[i].m3);
    scanf("%d%d%d", &s[i].m1, &s[i].m2,
                    &s[i].m3);
    s[i].total = s[i].m1+s[i].m2+s[i].m3
    avg = avg+s[i].total;
  }
```

Q.No.	Solution and Scheme	Marks

16	avg = avg / n;
	printf(" The average mark for class is %.f", avg);
	for(i=0; i<n; i++)
	{
	printf(" Name = %s", s[i].name);
	printf(" usn = %s", s[i].usn);
	printf(" m1 = %d", s[i].m1);
	printf(" m2 = %d", s[i].m2);
	printf(" m3 = %d", s[i].m3);
	printf(" total = %d", s[i].total);
	if (s[i].total < avg)
	printf(" The student scored below avg");
	else
	printf(" The student scored above avg");
	}
	}
		10M

10 a	structure	union
	* structure is declared by using keyword struct	* union is declared by using keyword union
	* syntax struct structure_name { data type member1; --- --- } var1;	* syntax union union_name { data type member1 --- --- } var1;
	* Each member of structure has its own memory location. The total memory used by a structure sum of all its members location	* All members of union share the same memory location. The size is determined by size of its largest member.

Q.No:	Solution and Scheme	Marks

structure	union
* used when you need to group related variable that need to be accessed independently	* used when you need to store different types of data in the same memory location but not simultaneously
* You can access all members of a structure at the same time as they occupy different locations	* only one member of union can be accessed at a time, as they occupy same memory location

Example: struct student
{
int rollno;
char usn[10];
}stud;
	union student
	{
	int rollno;
	char usn[10];
	};
	union student s1;

6M

10b **fopen() function:**

syntax: FILE *fopen(const char *filename, const char *mode);

*filename is any file name, here we need specify correct path of filename existed.

* mode is type of procesing that will be done with the file.

```
FILE *fp;
fp = fopen("a.txt", "r");
if (fp==NULL)
{ printf("file could not be opened");
} exit(1);
```

Q.No.	Solution and Scheme	Marks

Modes	Description
r	open a text file for reading
w	open text file for writing, if does not then file will be created
a	Append text file.

2m

same thing binary file rb, wb, ab

fclose(): syntax

```
int fclose(FILE *fp);
```

2m

fp is file pointer which point to the file that has to be closed. A zero returned if function was succesful and non-zero value is returned if error occured.

fscanf(): syntax

```
int fscanf(FILE *stream, const char *format);
```

fscanf() function is used to read data from stream. Here according to parameter format into the location pointed by arguments. 2m

format specifier is

```
% [*] [width] [modifier] type
```

* is optional argument
*width specifies maximum number of characters
*modifiers can be h, l, or L.
*type can be c, d, e, E, f, g, G, s, u, U.

fprintf() function: syntax

```
int printf(FILE * stream, const char * format);
```

2m

prototype of format

```
% [flag] [width] [.precision] [length] specifier
```

20

Q.No.	Solution and Scheme	Marks

20	Example for fopen(), fclose(), fscanf, fprintf()
	#include <stdio.h>
	int main()
	{
	FILE *fp, *fp1;
	char name[50];
	int rollno;
	fp = fopen ("Student.txt", "r");
	fp1 = fopen ("student1.txt", "w");
	if (fp == NULL && fp1 == NULL)
	{
	printf ("The file could not be opened");
	exit(1);
	}
	printf(" Enter roll no. & name ");
	fscanf(fp, "%d %s %d", name, &rollno);
	fprintf(fp1, "name = %s, rollno = %d", name, rollno);
	fclose (fp); fcelose (fp1);
	}
	In the above example fp is opend to read the file student.txt and student1.txt is opended to write the content of student.txt and after reading and writing ores both files are closed by using fclose() function.
		2M

Q.No.	Solution and Scheme	Marks
24 10C	Enumerated data type is a uses defined type based on the standard integer type. An enumeration consists of a set of named integer constants.	1M
	To define enumerated data type we use keyword enum.	
	Syntax: to declared enumerated datatype	
	enum enumeration_name {identifier1, identifier2, ------- identifiern};	
	enum is keyword used declare and initialize a sequence of integer constants. Enumeration_name is optional.	
	Example: enum COLOR {RED, BLACK, PINK};	4M
	no fundamental data type is used in declaration of COLOR., after this COLOR become datatype COLOR is name given to the set of constant.	
	⇒ If you do not initilize the constant by default each one would have unique value and first would be zero. So above example RED=0, BLACK=1, PINK=2.	
	Example: enum COLOR {RED=1, BLACK=2, PINK=3, WHITE=4};	
	COLOR bg-color, fg-color;	
	bg-color = WHITE; fg-color = PINK;	
	Now COLOR is data type & variables are bgcolor & fg-color	

Q.No.	Solution and Scheme	Marks

Example:

```
#include<stdio.h>
enum COLORS{ RED, BLUE, BLACK, GREEN, YELLOW}
int main()
{
enum COLORS C;
char *color_name[] = {"RED", "BLUE", "BLACK",
                      "GREEN", "YELLOW"}
for (C= RED; C<= YELLOW; C++)
  printf("%s", color_name[C]);
  return 0;
}
```

HOD
Computer Science & Engineering
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal.

39

42