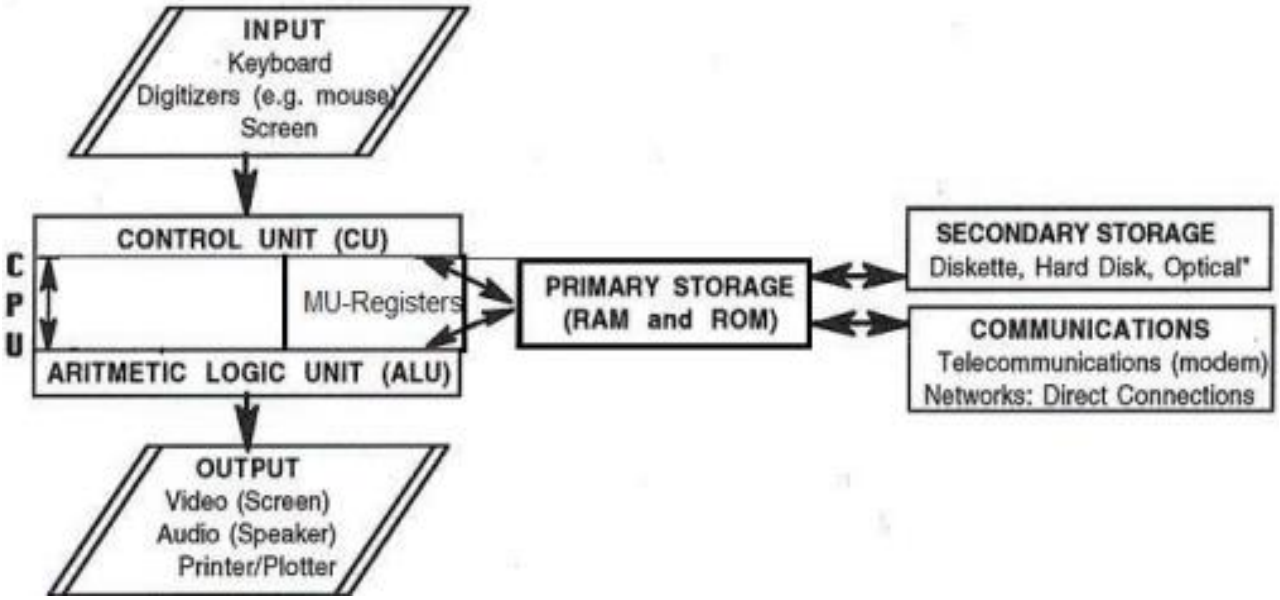
Solution with scheme-Model Answer

Prof.Lynsha Helena Pratheeba HP/Prof.Rajeshwari R/ Prof.Kavyashree/Prof.Reshma									
Internal Assessment Test 1 – November 2024									
Sub	**Principles of Programming Using C**					Sub code	**BPOPS103**	Branch	**ISE, AIML, CSE(AIML), AIDS**
Date	**22.11.2024**	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	**I Sem P-Cycle (A- H)**		**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1. a)	With a neat block diagram of computer explain its components.	[6]	CO1	L1
b)	Compare the input and output devices with examples.	[4]	CO1	L2
2. a)	Illustrate the basic structure of a C Program. Explain each section briefly with suitable examples including flowchart and algorithm.	[10]	CO1	L3
3. a)	Write a C program with flowchart:			
b)	a) To find Mechanical energy of a particle using $E=mgh+1/2mv^2$	[5]	CO1	L3
	b) Convert kilometre into metre, millimetre and centimetre (note: print only 4 values after decimal point)	[5]	CO1	L3
4. a)	Define Operator. Explain the unary and ternary operators with examples.	[5]	CO2	L2
b)	Write the correct output for the below code snippet: (each carries marks) 1) int x=10, y=20, z=5, i; i=x<y<z; printf ("%d", i); 2) int a = 15, b = 10, c = 5, d = 2; a += b-- * ++c? d++: --b; a *= c-- + b++ - d; 3) int x=8,y =3;float result; result =(float)(x + y)/y; printf("Result:%.2f\n", result); 4) int a=500, b=100, c; if (! a>=400) b=300; c=200; printf ("b=%d c=%d", b,c); 5) Write the output statement for below codes: 31.240000 6.360000 5.460000 float a=31.24; double b=6.36; long double c=5.46;	[5]	CO2 CO2	L3
5. a)	Compute the roots of a quadratic equation by accepting the coefficients. Print appropriate messages.	[5]	CO2	L3
b)	Write a C Program to display the following by reading the number of rows as input: 12345 1234 123 12 1 ---------- n^{th} row	[5]	CO2	L3
6. a)	Differentiate while () and do-while ().	[3]	CO2	L2
b)	State the drawbacks of ladder if-else. Explain how do you resolve with suitable example.	[7]	CO2	L2
7. a)	What is user defined function? Give the advantages of using functions. Explain any two categories of function prototype with examples.	[10]	CO3	L2
8. a)	Explain how actual parameters are different from formal parameters.	[4]	CO3	L2
b)	Illustrate the types of parameters passing methods.	[6]	CO3	L2

1. a) **With a neat block diagram of computer explain its components.** [6]

Diagram with Explanation [2M+4M]



Input unit: The input unit that links the external environment to input data & tasks with the computer system to execute. Data are entered in different forms through different input devices. Keyboard is used for characters input. Mouse is used in GUI (Graphic User Interface). Internally data is processed in machine readable form.

Output Unit: Output/result is displayed, printed & transmitted to outside world. There are many output devices: monitor, printer/plotters, display boards, speaker etc.
Storage unit: The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. The storage unit is Primary Memory (RAM) & Secondary (permanent storage devices: disks, tapes)

CPU (Central processing Unit): It is the main unit which controls all events within computer. The CPU has 3 internal units below:

CU (Control unit): By selecting, interpreting, and seeing to the execution of the program instructions, the control unit is able to maintain order and directs the operation of the entire system. the control unit acts as a central nervous system for the other components of the computer. It manages and coordinates the entire computer system. It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.

ALU (Arithmetic & Logic Unit): The arithmetic and logic unit (ALU) is the part where actual computations take place. It consists of circuits that perform arithmetic operations (e.g. addition, subtraction, multiplication, division over data received from memory and capable to compare numbers (less than, equal to, or greater than).

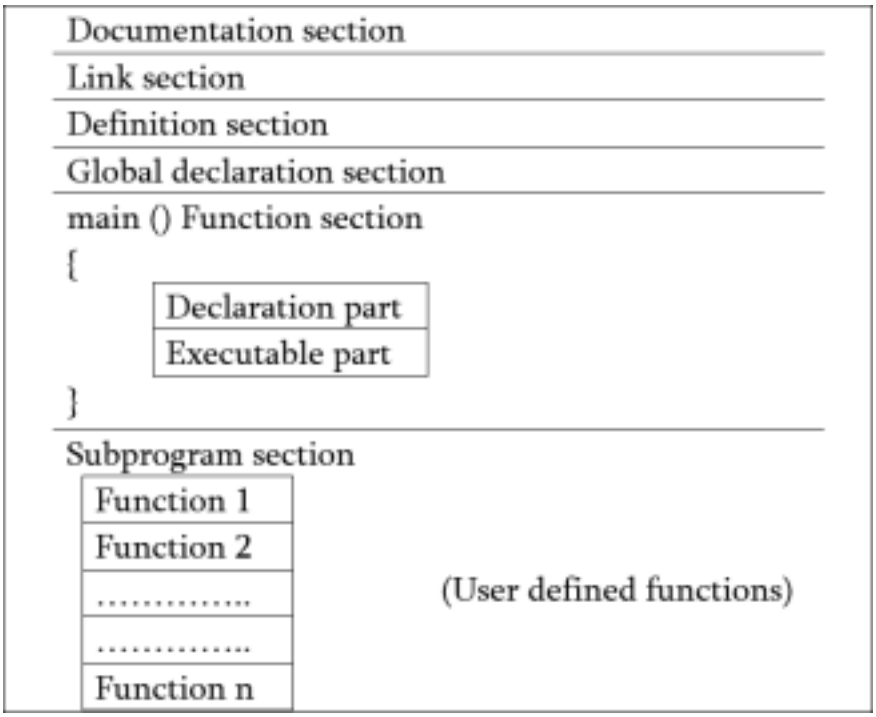
MU (Memory Unit/Registers): Registers are built-in memory with CPU having less storage space in bits. Registers are a group of cells used for memory addressing, data manipulation and processing. Instruction Registers, Address registers, Program Counters, Accumulators are example of registers. ALU takes data from here inside the CPU.
RAM (Random Access Memory): RAM is the memory - primary storage where our data & programs are stored temporarily. It is volatile in nature. After switching off the system everything will be vanished from RAM.
ROM (Read Only Memory): ROM is storage medium/"firmware" where some code of manufacturer is permanently hardwired in chip which always executes automatically when we start the system. The process is known as POST (Power on Self-test). Booting preceeds POST.

b) **Compare the input and output devices with examples.** [4]

Input Devices	Output devices
Data is accepted by the user of the device	It shows the data after processing to the user
It accepts the user's data and transmits it to the processor for saving in the secondary memory or processing.	It receives the data from the processor and returns it to the user
More complex designing	Less complex designing
These devices are used to accept the data	These devices are used to display or show the data
Example: Keyboard, mouse, etc.	Example: Monitor, Printer, etc.

2. a) **Illustrate the basic structure of a C Program. Explain each section briefly with suitable examples including flowchart and algorithm.** [10]

Explanation [4] +Program [2] +Alg[2] +Flow[2]



Comment line:
It indicates the purpose of the program.
It is represented as:
Single line - //content
Multiple line - / * ... ... * /
Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters.
Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Preprocessor Directive:
Preprocessor directives in C are lines included in the code that begin with the # symbol and are processed before the actual compilation of code begins. These directives instruct the preprocessor to perform specific actions, such as including header files.
Syntax: #include<headerfile.h> #include<stdio.h> tells the compiler to include information about the standard input/output library.

Definition section:
Syntax: #define symbolicconstantvariable symbolicconstat
It is also used in symbolic constant such as#define PI 3.14(value).

Global Declaration: This is the section where variables are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function.
Syntax: Datatype variablename;

Main (): It is the user defined function and every function has one main () function from where actually program is started and it is enclosing within the pair of curly braces. The main () function can be anywhere in the program, but in general practice it is placed in the first position.
Syntax: int main ()

Sub program section: There may be other user defined functions to perform specific task when called.
Sample Example:

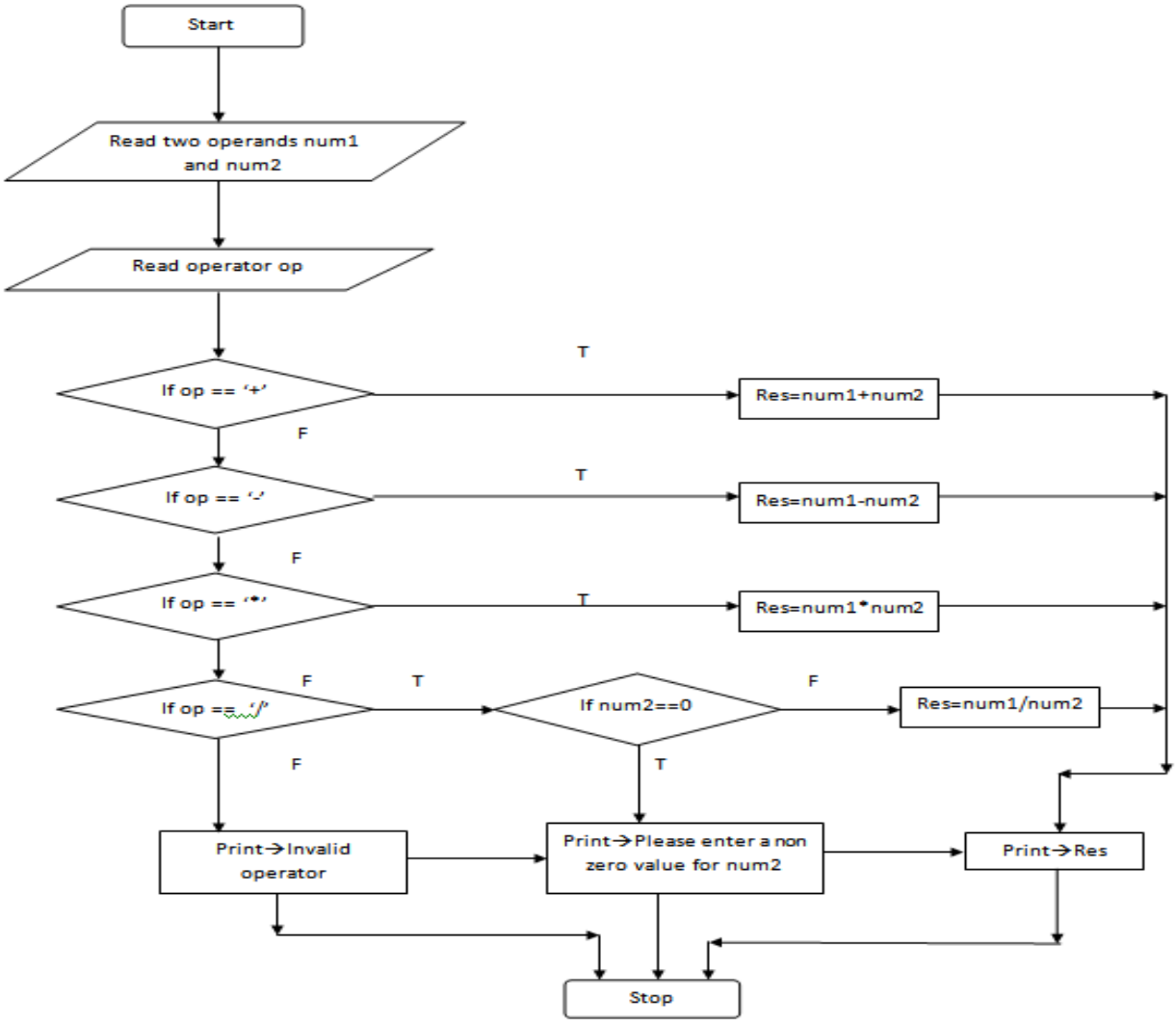
```
//Simple Calculator //Documentation section
# include<stdio.h> // header section
 int main() //main function
{
int num1, num2;
int result;                 //local variable declaration
char op;
printf("Enter the operator \n");
scanf("%c",&op);
printf("Enter two integers :");          //Executable statements
scanf("%d%d", &num1,&num2);
if (op == '+')
{
        result=num1+num2;
}
else if (op == '-')
{
        result=num1-num2;
}
else if (op == '*')
{
        result=num1*num2;
}
else if (op == '/')
{
        if (num2 == 0)
        {
                printf("Divide by zero error \n");
                return (1);
        }
else
{
        result=num1/num2;
}
}
else if (op == '%')
{
        if (num2 == 0)
        {
                printf("Divide by zero error \n");
                return (2);
        }
```

```
else
{
result=num1%num2;
 }
}
 else
 {
        printf("Invalid operator...\n");
        return (3);
 }
        printf("%d %c %d  = %d\n", num1, op, num2, result);
        return 0;
 }
```

Sample Output:

Enter the operator +
Enter two integers: 2 3
2+3 =5

Flowchart:



Algorithm:

Input: Two integers(operands) and operator
Output: Result of the operation
Step 1: Start
Step 2: Read two operands and an arithmetic operator
Step 3: Check if operator equals '+', if yes, then goto step 4 else goto step 5
Step 4: Compute addition operation - res = num1 + num2 and goto step 18
Step 5: Check if operator equals '-', if yes, then goto step 6 else goto step 7
Step 6: Compute subtraction operation - res = num1 – num2 and goto step 18
Step 7: Check if operator equals '*', if yes, then goto step 8 else goto step 9

Step 8: Compute multiplication operation - res = num1 * num2 and goto step 18
Step 9: Check if operator equals '/', if yes, goto step 10 else goto step 13
Step 10: Check if num2 equals Zero, if yes, then goto step 11 else goto step 12
Step 11: Display – "Divide by zero error" and goto step 19
Step 12: Compute division operation - res = num1 / num2 and goto step 18
Step 13: Check if the operator equals '%', if yes, then goto step 14 else goto step 17
Step 14: Check if num2 equals zero, if yes, then goto step 15 else goto step 16
Step 15: Display - "Divide by zero error." and goto step 19.
Step 16: Compute modulus operation – res = num1 % num2 and goto step 18
Step 17: Display "Invalid operator" and goto step 19.
Step 18: Display the result
Step 19: Stop

3.a) **Write a C program with flowchart:** **Program [3] +Flow [2]** [5]

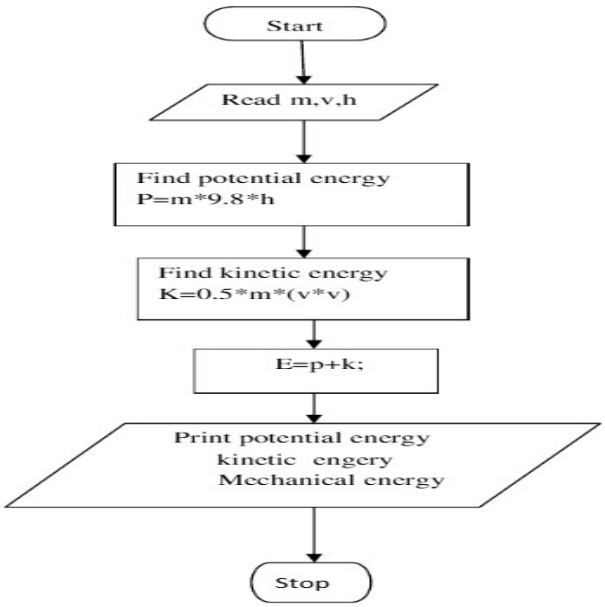
To find Mechanical energy of a particle using $E=mgh+1/2mv^2$.

Code:

```
#include <stdio.h>
int main(void)
{
float m,h,v,p,k,e;
printf("Enter Mass of the body\n");
scanf("%f",&m );
printf("Enter displacement of the body\n");
scanf("%f",&h );
printf("Enter velocity of the body\n");
scanf("%f",&v );
p=m*9.8*h; //To calculate Potential energy
k=0.5*m*(v*v); //To calculate Kinetic energy
e=p+k;
printf("Potential energy of the body = %f\n",p );
printf("Kinetic energy of the body = %f\n",k );
printf("Mechanical energy of a body = %f\n" , e);
}
```

Sample Output:
Enter Mass of the body 1000.00
Enter displacement of the body 10.00
Enter velocity of the body 120.00
Potential energy of the body = 98000.00
Kinetic energy of the body = 7200000.00
Mechanical energy of a body = 7298000.00

Flowchart:



b) Convert kilometre into metre, millimetre and centimetre (note: print only 4 values after decimal point) [5]

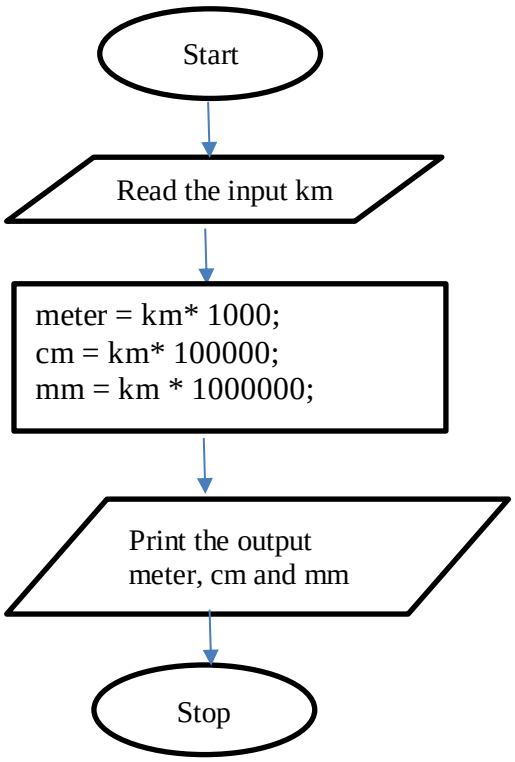
Code:

```
#include <stdio.h>
int main () {
float meter, centimeter, millimeter, kilometer;
printf("Enter Length in Kilometer(km)\n");
scanf("%f", &kilometer);
meter = kilometer * 1000;
centimeter = kilometer * 100000;
millimeter = kilometer * 1000000;
printf("%.4f Kilometer = %.4f Meter\n", kilometer, meter);
printf("%.4f Kilometer = %.4f Centimeter\n",kilometer,centimeter);
printf("%.4f Kilometer = %.4f Millimeter\n",kilometer,millimeter);
return 0;
}
```

Sample Output:

Enter Length in Kilometer(km) 4.0
4.0000 Kilometer = 4000.0000 Meter
4.0000 Kilometer = 400000.0000 Centimeter
4.0000 Kilometer = 4000000.0000 Millimeter

Flowchart:



4. a) **Define Operator. Explain the unary and ternary operators with examples. [1] + [2] + [2]** [5]

In programming, an operator is a symbol that tells the compiler or interpreter to perform specific mathematical, relational, or logical operations and produce a final result. Operators act on operands, which can be values or variables

Unary Operator

A unary operator is an operator that operates on a single operand. Here are a few common unary operators in C:

Increment Operator (++), Decrement Operator (--), Unary Minus Operator (-), Unary plus Operator (+), Logical NOT Operator (!)

Eg: a++, +a

Ternary Operator

The ternary operator is a shorthand for the if-else statement and is the only operator in C that takes three operands. It is also known as the conditional operator and is represented by the ?:. It evaluates a condition and

chooses one of two expressions to return based on the result of the condition.

Syntax:

condition? expression_if_true : expression_if_false;

Eg:

int x = 10; int y = (x > 5) ? 20 : 30

y will be 20 because x > 5 is true.

b) **Write the correct output for the below code snippet: [each carries 1M]** [5]

1. int x=10, y=20, z=5, i; i=x<y<z; printf ("%d", i);

Explanation:

First Comparison: x < y

- With x = 10 and y = 20, x < y is true, so it returns 1

Second Comparison: 1 < z

- Now, z = 5, so 1 < 5 is true, which also returns 1.

So, the statement i = x < y < z will evaluate i as 1.

Output:1

2. int a = 15, b = 10, c = 5, d = 2; a += b-- * ++c? d++: --b; a *= c-- + b++ - d;

Explanation:

a += b-- * ++c ? d++ : --b:

- ++c changes c from 5 to 6
- b-- will use the current value of b (which is 10) and then decrement b to 9
- b-- * ++c evaluates to 10 * 6 = 60
- 60 ? d++ : --b (Since 60 is non-zero, it is true, so the expression evaluates to d++.
- d++ returns 2 (the current value of d before incrementing), and then d becomes 3.
- a += 2 changes a from 15 to 17.

Output:

a = 17, b = 9, c = 6, d = 3

a *= c-- + b++ - d:

Explanation:

- c-- will use the current value of c (which is 6) and then decrement c to 5.
- b++ will use the current value of b (which is 9) and then increment b to 10
- c-- + b++ - d evaluates to 6 + 9 - 3 = 12
- a *= 12 changes a from 17 to 17 * 12 = 204

Output:

a = 204, b = 10, c = 5, d = 3

3. int x=8,y =3;float result; result =(float)(x + y)/y; printf("Result:%.2f\n", result);

- x + y is evaluated: 8 + 3 = 11
- (float)(x + y) casts the result 11 to a floating-point number 11.0
- 11.0 / y then performs the division: 11.0 / 3 = 3.66667

Output:

Result: 3.67

4. int a=500, b=100, c; if (! a>=400) b=300; c=200; printf ("b=%d c=%d", b,c);

Explanation:

if (!(a >= 400)) evaluates the expression a >= 400, which is true

! operator negates this, making the condition false

Since the condition is false, the statement b = 300; does not execute

c is then assigned the value 200

Output:

b=100 c=200

5. Write the output statement for below codes: 31.240000 6.360000 5.460000

float a=31.24; double b=6.36; long double c=5.46;

Output:

printf("%f %lf %Lf", a, b, c);

5. a) **Compute the roots of a quadratic equation by accepting the coefficients. Print appropriate messages. [4] + [1]** [5]

Code:

```
#include<stdio.h>
#include<math.h>
int main()
{
float a,b,c,desc,r1,r2,realpart,imgpart;
printf("Enter the coefficients of a, b and c :");
scanf("%f%f%f",&a,&b,&c);
if(a == 0)
{
 printf("Coefficient of a cannot be zero....\n");
 printf("Please try again....\n");
 return 1;
 }
 desc=(b*b)-(4.0*a*c);
 if(desc==0)
 {
 printf("The roots are real and equal\n");
 r1=r2=(-b)/(2.0*a);
 printf("The two roots are r1=r2=%f\n",r1);
 }
 else if(desc>0)
 {
 printf("The roots are real and distinct\n");
 r1=(-b+sqrt(desc))/(2.0*a);
 r2=(-b-sqrt(desc))/(2.0*a);
 printf("The roots are r1=%f and r2=%f\n",r1,r2);
 }
 else
 {
 printf("The roots are imaginary\n");
 realpart=(-b)/(2.0*a);
 imgpart=sqrt(-desc)/(2.0*a);
 printf("The roots are r1=%f + i %f\n",realpart,imgpart);
 printf("r2=%f - i %f\n",realpart,imgpart);
 }
return 0;
 }
```

Sample Output:

```
Enter the coefficients of a, b and c :1 2 1
The roots are real and equal
The two roots are r1=r2=-1.000000
```

b) **Write a C Program to display the following by reading the number of rows as input:** [5]

12345
1234
123
12
1
----

n^{th} row **Code [4] + Output [1]**

Code:

```
#include <stdio.h>
int main()
{
int n;
printf("Enter the number of rows: ");
scanf("%d", &n);
for (int i = n; i >= 1; i--)
{
for (int j = 1; j <= i; j++)
{ printf("%d", j);
}
printf("\n");
} return 0;
}
```

Output:

```
Enter the number of rows: 5
12345
1234
123
12
1
```

6. a) **Differentiate while () and do-while (). [3]** [3]

Sl.no	while Loop	do-while Loop
1	while (condition) { }	do { } while (condition);
2	Condition is checked before the loop block is executed.	Loop block is executed at least once before checking the condition.
3	Suitable when the loop block should be executed only if the condition is initially true.	Useful when the loop block must be executed at least once, regardless of the initial condition.

b) **State the drawbacks of ladder if-else. Explain how do you resolve with suitable example.** [7]

Ladder if-else statement is multi-way decision statement [2] + [2] + [3]

Syntax of ladder if-else statement:

```
if(condition1)
statement(s);
else if(condition2)
statement(s);
else if(condition3)
statement(s);
…….
else if(conditionn)
statement(s);
else
statement(S);
```

if any of the condition1, condition2 & condition3 is evaluated true then corresponding statements are executed and the control comes out entire of the ladder if-else statement. If all conditions are false then last statement will be executed.

Drawback of ladder if-else statement:

a. Multiple if-else conditions are little tough to understand & check to modify for correct output.
b. As depth of ladder increases, readability of program decreases.
c. Each condition and decision may evolve expressions

Instead of else-if ladder, we can go for switch statement.

Syntax of switch statement:

```
switch (expression)
{ case condition1
statement1;
statement2;
.....
break;
case condition2
statement1;
statement2;
.......
break;
.....
default:
statement1;
statement2;
.......
}
```

Sample Example:

Problem definition:

Bonus of employee in a company is based on grade as following:
Grade ‘a’ employee gets bonus equal to their salary
Grade ‘b’ and ‘c’ employees get bonus salary + 5000
Grade ‘d’ and ‘D’ employees get bonus salary + 10000
Other than these they get bonus salary + 15000.

Source Code:

```
#include <stdio.h>
int main()
{
int salary,bonus;
char grade;
printf(“Enter grade : “);
scanf(“%c”, &grade);
printf(“Enter salary : “);
scanf(“%d”, &salary);
switch (grade)
{
case ‘a’:
case ‘A’: bonus=salary;
break;
case ‘b’:
case ‘B’:
case ‘c’:
case ‘C’: bonus=salary+5000;
break;
case ‘d’:
case ‘D’: bonus=salary+10000;
break;
default :
bonus=salary+15000; /*lower grade-more bonus*/ }
printf(“Bonus = %d\n”, bonus);
return (0);
}
```

Output:
Enter grade: A
Enter salary: 50000
Bonus = 50000

7. **What is user defined function? Give the advantages of using functions. Explain any two categories of function prototype with examples. [2] + [2] + [3] + [3]** [10]

User-Defined Function in C

A user-defined function in C is a function created by the programmer to perform specific tasks. Unlike built-in library functions, user-defined functions are designed to address particular needs within a program. They help in breaking down complex programs into smaller, manageable, and reusable code blocks.

Advantages of Using Functions

1. **Modularity**: Functions break down a large program into smaller, manageable sections. This modular approach makes the program easier to read, understand, and maintain.
2. **Reusability**: Once a function is defined, it can be reused in multiple places within the program without rewriting the same code. This saves time and reduces redundancy.
3. **Abstraction**: Functions allow programmers to hide the implementation details and expose only the necessary interfaces. This abstraction helps in focusing on high-level design and functionality.
4. **Debugging**: Functions isolate different parts of the program, making it easier to test and debug each part independently.

Categories of Function Prototypes

Function prototypes declare the function's name, return type, and parameters without defining the function's body. Two common categories are:

1. **Function with No Arguments and No Return Value**
   This type of function does not take any parameters and does not return a value. It's typically used for performing tasks that do not require input and do not produce a result.

Source code:

```
#include <stdio.h>
void greet();
int main()
{
greet();
}
void greet()
{
printf("Hello, world!\n");
}
```

Output:
Hello, world!

2. **Function with Arguments and No Return Value**
   This type of function takes parameters but does not return a value. It's useful for performing operations that need input values but do not need to return a result.

Source code:

```
#include <stdio.h>
void displaySum (int a, int b);
int main ()
{
int num1 = 10, num2 = 20;
displaySum(num1, num2);
return 0;
}
void displaySum(int a, int b)
{
int sum = a + b;
printf("Sum: %d\n", sum);
}
```

Sample Output: Sum:30

8 a) **Explain how actual parameters are different from formal parameters. [4points]** [4]

	Actual Parameters	Formal Parameters
1	Also called actual arguments list	Also known as dummy parameters
2	These are variables used in function call	These are variables defined in function header
3	They are actual values passed to a function on which the function will perform operations	They are the variables in the function definition that would receive the values when the function is invoked
4	Occurs when we invoke a function	Occurs when we declare and define a function

b) **Illustrate the types of parameters passing methods. [3]+[3]** [6]

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 the caller.

Example:

```
#include<stdio.h>
void swapx(int x, int y);
int main()
{
int a = 10, b = 20;
swapx(a, b);
printf("a=%d b=%d\n", a, b);
return0;
}
void swapx(intx,inty)
{ int t;
t = x;
x = y;
y = t;
printf("x=%d y=%d\n", x, y);
}
```

Sample Output: x=20 y=10
a=10 b=20

Program with Output [3M]

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

Example:

```
#include <stdio.h>
voidswapx(int*,int*);
int main()
{
int a = 10, b = 20;
swapx(&a,&b);
printf("a=%d b=%d\n", a, b);
return0;
 }
void swapx(int*x,int*y)
{ intt;
t =*x;
*x =*y;
*y =t;
printf("x=%d y=%d\n", *x, *y);
}
```

Sample output: x=20 y=10
a=10 b=20