Solution with scheme-Model Answer

Prof.Lynsha Helena Pratheeba HP/ Prof.Kavyashree/ Prof.Reshma									
Internal Assessment Test 1 – May 2025									
Sub	**Principles of Programming Using C**					Sub code	**BPOPS103**	Branch	**CSE, CSDS**
Date	**07.05.2025**	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	**I Sem P-Cycle (A- H)**		**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Define a computer. Explain the characteristics of a digital Computer.	[10]	CO1	L1
2.	Explain the basic structure of a C program in detail. Write a sample program to demonstrate the components in it.	[10]	CO1	L2
3.	Illustrate Ternary operator with syntax and write the control statement equivalent to to this operator with syntax and example program.	[10]	CO2	L2
4.	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. Draw the flowchart and Algorithm.	[10]	CO2	L3
5.	Write a C program to print the following patterns: (i) 1 1 2 1 2 3 1 2 3 4 (ii) 1 1 2 1 1 2 3 2 1 1 2 3 4 3 2 1	[10]	CO2	L3
6.	Define Function and explain the aspects of function with a proper syntax and example.	[10]	CO3	L1
7.	Categorize the various declaration types of a function with proper example for each.	[10]	CO3	L2
8.	Explain the types of functions based on parameters. Write a program to swap two numbers using third variable for the same.	[10]	CO3	L2

1. a) **Define a computer. Explain the characteristics of a digital Computer.**
Definition with Explanation [3M+7M] [10]

A computer is an electronic device that can store, manipulate, and process data according to a set of instructions. It is a basic and functional computer that includes all the hardware and software that are required to make it functional for the user.

Characteristics of Computer are as follows:

1. Speed
Executing mathematical calculation, a computer works faster and more accurately than human. Computers have the ability to process so many millions (1,000,000) of instructions per second. Computer operations are performed in micro and nano seconds. A computer is a time saving device. It performs several calculations and tasks in few seconds that we take hours to solve. The speed of a computer is measure in terms of GigaHertz and MegaHertz.

2. Diligence
A human cannot work for several hours without resting, yet a computer never tires. A computer can conduct millions of calculations per second with complete precision without stopping. A computer can consistently and accurately do millions of jobs or calculations. There is no weariness or lack of concentration. Its memory ability also places it ahead of humans.

3. Reliability
A computer is reliable. The output results never differ unless the input varies. the output is totally depends on the input. when an input is the same the output will also be the same. A computer produces consistent results for similar sets of data, if we provide the same set of input at any time we will get the same result.

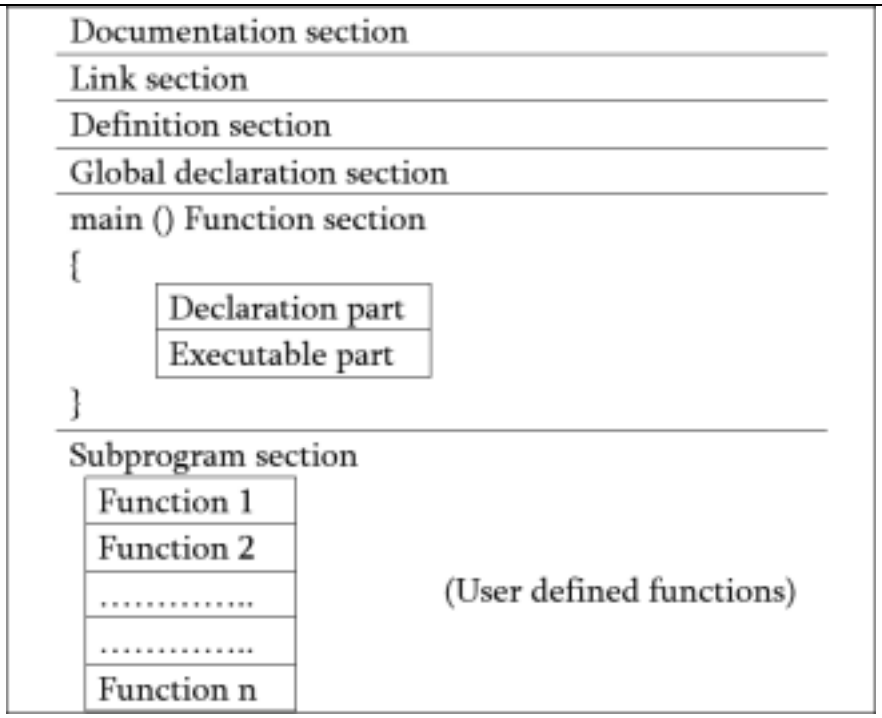
4. Automation
The world is quickly moving toward AI (Artificial Intelligence)-based technology. A computer may conduct tasks automatically after instructions are programmed. By executing jobs automatically, this computer feature replaces thousands of workers. Automation in computing is often achieved by the use of a program, a script, or batch processing.

5. Versatility
Versatility refers to a capacity of computer. Computer perform different types of tasks with the same accuracy and efficiency. A computer can perform multiple tasks at the same time this is known as versatility. For example, while listening to music, we may develop our project using PowerPoint and Wordpad, or we can design a website.

6. Memory
A computer can store millions of records. these records may be accessed with complete precision. Computer memory storage capacity is measured in Bytes, Kilobytes(KB), Megabytes(MB), Gigabytes(GB), and Terabytes(TB). A computer has built-in memory known as primary memory.

7. Accuracy
When a computer performs a computation or operation, the chances of errors occurring are low. Errors in a computer are caused by human's submitting incorrect data. A computer can do a variety of operations and calculations fast and accurately.

2. **Explain the basic structure of a C program in detail. Write a sample program to demonstrate the components in it.** **Explanation [8] +Program [2]** [10]



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
```

```
 printf("Enter two integers :");                    //Executable statements
 scanf("%d%d", &num1,&num2);
 result=num1+num2;
printf("The sum of %d and %d is %d", num1,num2,result);
return 0;
}
```

Sample Output:
Enter two integers: 5 4
The sum of 5 and 4 is 9.

3. **Illustrate Ternary operator with syntax and write the control statement equivalent to to this operator with syntax and example program. Ternary [5] + If-else[5]** [10]

Ternary Operator:
Syntax:
condition ? expression_if_true : expression_if_false;

- condition: An expression that evaluates to either true or false.
- expression_if_true: This expression is executed if the condition is true.
- expression_if_false: This expression is executed if the condition is false.

Example Program:

```
#include <stdio.h>
int main() {
    int a = 10, b = 20, max;
    max = (a > b) ? a : b;
    printf("Maximum is %d\n", max);
    return 0;
}
```

Sample Output: Maximum is 20

The Equivalent Control Statement is **if-else.**

Syntax:

```
if (condition) {
    expression_if_true;
} else {
    expression_if_false;
}
```

Example Program:

```
#include <stdio.h>
int main() {
   int a = 10, b = 20, max;
   if (a > b) {
   max = a;
   } else {
   max = b;
   }
    printf("Maximum is %d\n", max);
    return 0;
}
```

Sample Output: Maximum is 20

4. **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. Draw the flowchart and Algorithm. Program [6] + Flowchart [2] + Algorithm [2]** [10]

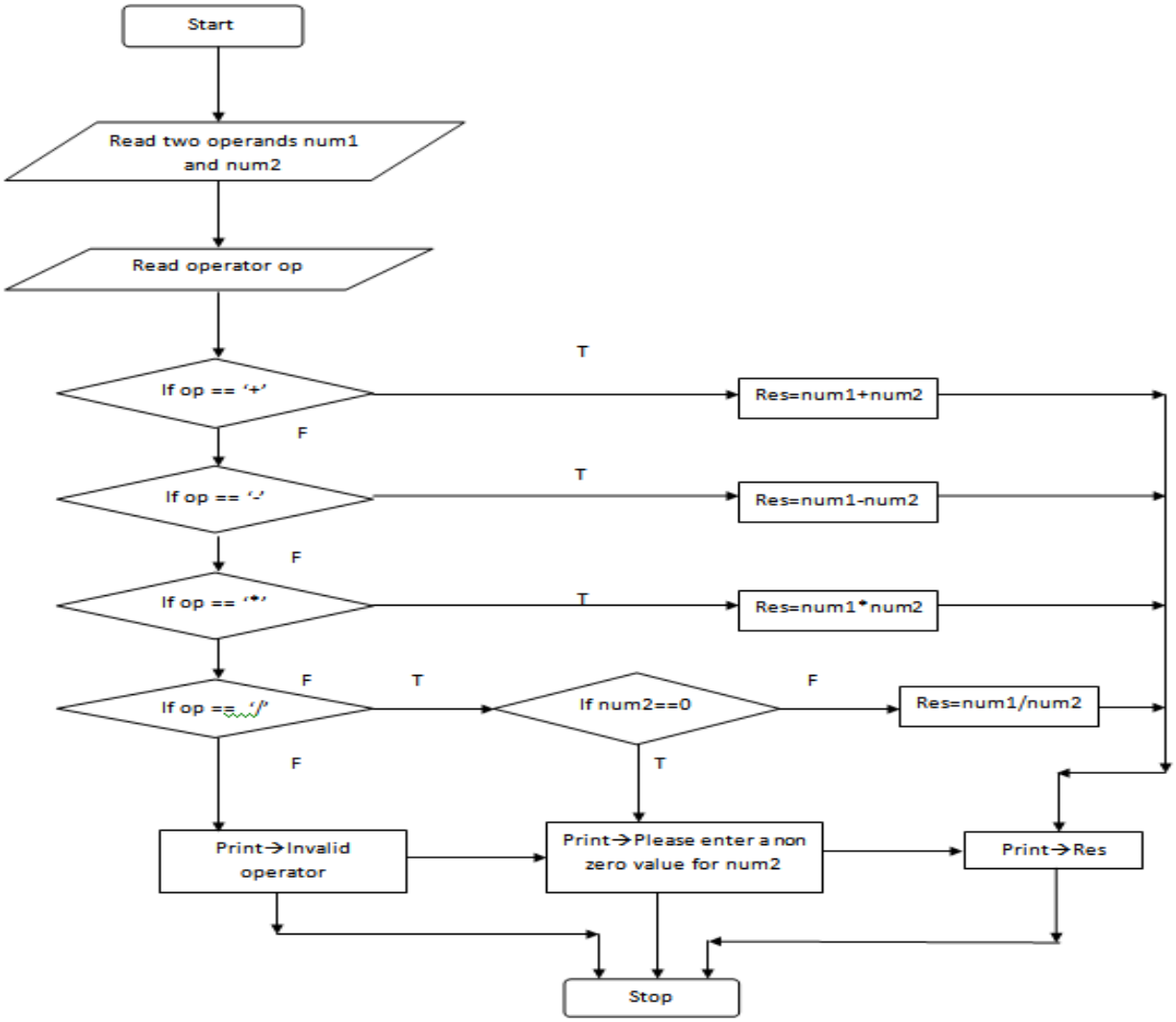
```
//Simple Calculator
# include<stdio.h>
 int main()
{
int num1, num2;
int result;              //local variable declaration
char op;
printf("Enter the operator \n");
scanf("%c",&op);
printf("Enter two integers :");          //Executable statements
scanf("%d%d", &num1,&num2);
if (op == '+')
{
        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

5. **Write a C program to print the following patterns:** [10]

(i)
```
1
1 2
1 2 3
1 2 3 4
```

(ii)
```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
```

[5M Each]

i) Code:

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

Sample Output:

```
Enter the number of rows: 4
      1
    1 2
  1 2 3
1 2 3 4
```

ii) Code:

```
#include <stdio.h>
int main() {
int i, j, n = 4;
for (i = 1; i <= n; i++)
{
for (j = 1; j <= i; j++)
{
printf("%d ", j);
}
for (j = i - 1; j >= 1; j--)
{
printf("%d ", j);
}
printf("\n");
}
return 0;
}
```

Sample Output:

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
```

6. **Define Function and explain the aspects of function with a proper syntax and example.** [10]

Function: [1M]
Function is a building block that contains set of programming statements to perform a particular task.

Aspects of the Function: [2M each]

1. **Function declaration:**
   A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
   Syntax:
   return-type function-name(datatype1 parameter1, datatype parameter2,....., datatype n parameter n);

2. **Function call:**
   Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
   Syntax:
   function-name (parameter1,parameter2, ... ,parameter n);

3. **Function definition:**
   It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.
   Syntax:
   return-type function-name(datatype1 parameter1, datatype parameter2,....., datatype n parameter n)

Example Program [3M]

Source code:

```
#include<stdio.h>
int sum(int a, int b);  //FunctionDeclaration
int main()
{
int a,b,result;
printf("\nEnter two numbers:");
scanf("%d %d", &a,&b);
result = sum(a,b);  //FunctionCall
printf("\nThe sum is : %d",result);
}
int sum(int a, int b)   ////FunctionDefinition
{
int r;
r=a+b;
return r;
}
```

Sample Output:

```
Enter two numbers: 4 5
The sum is 9.
```

7. **Explain any two categories of function prototype with examples. Categories [2] + Expl[2 each]** [10]

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 without parameter and without return value
2) Function without parameter and with return value
3) Function with parameter and without return value
4) Function with parameter and with return

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. **Functions with arguments and No return value**

This type of function takes parameters and does not return a value.

Source code:

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

Sample Output: Sum:30

3. **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)
{
```

[5]

```
int sum = a + b;
printf("Sum: %d\n", sum);
}
```
Sample Output: Sum:30

4. **Function with Arguments and With Return Value**
   This type of function takes parameters and also return a value. It's useful for performing operations that need input values and need to return a result.

Source code:
```
#include <stdio.h>
int displaySum (int a, int b);
int main ()
{
int num1 = 10, num2 = 20;
displaySum(num1, num2);
printf("Sum: %d\n", sum);
return 0;
}
int displaySum(int a, int b)
{
int sum = a + b;
return sum;
}
```
Sample Output: Sum:30

8. **Illustrate the types of parameters passing methods. [3]+[3]** [10]

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 [5M]

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