Assignment - 1 Date: 16.09.2025

Ques-1 What is the output of with explanation of the program?

Code-
```
#include <stdio.h>
#define scanf "%s India is a great Nation"
int main() {
    printf(scanf, scanf);
    return 0;
}
```

Solution- Output - %s India is a great Nation India is a great Nation

Explanation- This C code uses a preprocessor macro to redefine 'scanf' as the string "%s India is a great Nation".

When code runs:

→ This line has to be executed: printf(scanf, scanf); It shows 'printf' function divided into two parts, the first part goes with its '%s' format specifier that holds second part as argument, that '%s' gets replaced by second argument. And remain statement "India is a great Nation" prints as it is.

→ And so the second arugement taken as a string so that even '%' doesnot create any conflict or barrier in execution of printf statement.

Que-2 What is the output of with explanation of the program?

Code-
```
#include<stdio.h>
int main() {
int x, y;
printf("Enter the value of x\n");
y = scanf("%d", &x);
printf("%d", &y);
return 0;
}
```

Solution- Explanation- The code demonstrates what 'scanf()' returns as its value.
What it does: Asks user to enter a value for x.
'scanf("%d", &x);' reads input AND returns how many values it successfully read.
That return gets stored by in 'y'. Prints the value of 'y'.

Output: If you enter a valid integer (like 7): prints 1 (because scanf successfully read 1 values).
If we enter invalid input (like any string): prints 0 (because scanf failed to read any integer).

Key point: scanf() returns the number of successful conversions, not the actual value that was read. The actual values goes into the variable specified (&x in this case).

So, if you run it type 123, you'll see:

Output: Enter the value of x
123
1

The '1' means scanf successfully read one integer value.

Ques-3 What is the output with explanation of the program?

Code-
```
#include <stdio.h>
int main() {
    int i;
    i = (1, 2, 3);
    printf("i=%d\n", i);
    return 0;
}
```

Explanation- This code demostrates the comma operator in C:
What happens:
→ i = (1, 2, 3); uses the comma operator, evaluates all expressions from left to right but returns only the last one value.
→ do (1, 2, 3) evaluates 1, then 2, then 3, and return 3.
→ i gets assigned the value 3.

Output: i = 3

Key concept: The comma operator ',' in C:
→ Evaluates expressions from left to right
→ Discards all results expects the last one.
→ Returns the value of the right most expression.

This is different from function arguments or array initialization where commas are just separators- here it's actually an operator.

Ques-4 What is the output with explanation of the program?

Code:
```
#include <stdio.h>
int main() {
int i;
i = 1, 2, 3;
printf("i=%d\n", i);
return 0;
}
```

Explation- This code shows operator precedence with the comma.

What happens: → i = 1, 2, 3; is parsed as (i=1), 2, 3; due to operator precedence

→ The assignment operator (=) has higher precedence than the comma operator.

→ So first: i=1 (i gets value 1)

→ Then the comma operator evaluates 2, then, 3 but discards them.

Output: i = 1

⇒ Key differences from previous code:

→ Previous i = (1, 2, 3); → i gets 3 (parantheis force comma operator first).

→ This i = 1, 2, 3; → i gets 1 (assignment happens first due to precendence).

The comma operator still evaluates all expressions (1, 2, 3) but since assignment happened first, i only gets the value 1.

Ques-5 What is the output with explanation of the program?

Code:
```
#include <stdlib.h>
int main() {
    int x, y = 2, z, a;
    if (x = y % 2)
        z = 2;
    a = 2;
    printf("%d %d", z, x);
    return 0;
}
```

Solution- Explanation- Initial values:

→ x is uninitialized
→ y = 2
→ z is uninitialized
→ a is uninitialized (but gets assigned a later)

Key line: if (x = y % 2), This is assignment, not comparison-

→ y % 2 = 2 % 2 = 0
→ x = 0 (assignment)
→ The if condition checks the value of x, which is 0.
→ Since 0 is false in C, the if block doesn't execute.
→ z = 2; is not executed.

Final values:

→ $x = 0$

→ z, remains uninitialized (undefined behaviour)

Output: [garbage_value] 0

The exact output is unpredictable because 'z' is uninitialized. You might see something like 'random-number', or any number for z.

Key points:

→ $x = y \% 2$ is assignment, not $x == y \% 2$ (comparison)

→ Since x becomes 0 (false), the if block is skipped.

→ Printing uninitialized variables leads to undefined behaviour.

Ques-6 What is the output with explanation of the program?

Code:
```
#include<stdio.h>
#define prod(a,b) a*b;
int main(){
  int x=3, y=1;
  printf("%d", prod(x+2, y-1));
  return 0;
}
```

Solution- Explanation- This codes demonstrate a common macro pitfall with operator precedence.

What happens:

→ The macro #define prod(a, b) a*b does simple text substitution.

→ When you call prod(x+2, y-1):
  a becomes x+2
  b becomes y-1
  The macro expands to: x+2*y-1

→ Due to operator precedence:

→ Multiplication (*) has higher precedence than addition/subtraction.

→ So x+2*y-1 is evaluated as x+(2*y)-1

→ With x=3, y=4 : 3+(2*4)-1 = 3+8-1 = 10

Output: 10

Ques-7 What is the output with explanation of the program?

```
Code: #define a 10
      #include <stdio.h>
      int main() {
        #define a 50
        printf(" %d ", a);
        return 0;
      }
```

Solution- This code demonstrate macro definition in C.

What happens-

→ first definition: #define a 10 (before main)
→ Second definition: #define a 50 (inside main function)
→ The second #define redefines the macro 'a'.

Preprocessor behaviour-

→ The preprocessor processes '#define' directives regardless of where they appear.
→ When 'a' is redifined, the new definition (50) overwrites the old one (10).
→ By the time printf(" %d ", a); is processed, a is replaced with 50.

Output: 50

Key points -

→ Macros can be redefined anywhere in the program.
→ The most recent definition is used.
→ The preprocessor doesn't care about function boundaries - #define works globally.
→ Most compilers will show a warning about macro redefinition.

Ques-8 Which of the following is true for variable names in C?

a) They can contain alphanumeric characters as well as special character.
b) It is not an error to declare a variable to be one of the keywords (like- goto, static)
c) Variable name cannot start with a digit
d) Variable can be of any length.

Solution- Option c) is TRUE. "Variable name cannot start with a digit".
→ The other options are not following the rules to use variables in C programming.

→ They doesn't properly follow the rules of setting variable in C program.

Ques-9 Which keyword is used to prevent any changes in the variable within a C program?

a) immutable
b) mutable
c) const
d) volatile

Solution- Option- c) Correct const.

Que:- Which of the following is NOT possible with any 2 operators in C?

a) Different precedence, same associativity
b) Different precedence, different associativity
c) Same precedence, different associativity
d) All of the mentioned.

Solution- The correct answer is c) Same precedence, different associativity.

Explanation- In C, operators that have the same precedence level must have the same associativity.

c) Same precedence, different associativity -

→ All operators at the same precedence level must have identical associativity.

Example: *, /, % all have precedence 3 and All are left-to-right associative.

Key Concept: Precedence determines the order of evaluation between different operators, while associativity determines the order of when the same precedence operators appear together. For parsing to be unambiguous, operators at the same precedence level must follow the same associativity rule.

Ques-11 Lila What is the sizeof(char) in a 32-bit C compiler?

Solution- Well sizeof(Char) in any compiler is 1-Byte always. Other data types sizes may vary.

Option - c) is correct.

Ques-12 What is the Output with the explanation of the program?

Code:
```
#include<stdio.h>
int main(){
    int main = 3;
    printf("%d", main);
    return 0;
}
```

Solution- Output: 3

This demonstrated variable shadowing in C:

→ int main = 3; declared a local variable named 'main' with value 3.
→ This local variable shadow (hides) the function name 'main'
→ Inside the function scope, 'main' now refers to the integer variable, not the function.

→ printf ("%d", main); prints the value of local variable: 3.

Key points: → It's legal to declare a variable with the same name as the function you're inside.

→ The local variable keeps takes precedence over the function name within that scope.

→ This is generally a bad practice as it makes code confusing.

→ The function still works normally - you can still call it from outside.

Why it works: → C allows identifier reuse in different scopes.

→ local scope has higher precedence than global / function scope.

→ The function name 'main' is still accessible from outside this scope.

Note: While this compiles, it's considered poor coding style because it's confusing and makes the code harder to understand.

Ques-13 What is the output with explanation of the program?

Code:
```
#include <stdio.h>
int main() {
    signed char chr;
    chr = 128;
    printf("%d\n", chr);
    return 0;
}
```

Solution- Output- -128

Explanation- Signed char range → -128 to +127 (8 bits, with 1 bit for sign).

What happens: → chr = 128; tries to assign 128 to a signed char

→ 128 is outside the valid range for signed char (+127 is the maximum).

→ This cause integer overflow.

Binary representation: 128 in binary : 10000000

→ In signed char (two's complement), the left most bit is the sign of bit.

→ 10000000 represents -128 in two's complement notation.

Two's complement overflow:

→ Values beyond +127 "wrap around" to negative values.

→ 128 becomes -128.

→ 129 would become -127, etc.

Key concept:

→ Signed char uses two's complement representation.

→ Overflow behaviour is implementation-defined but typically wraps around.

→ The bit pattern 10000000 = 128 (unsigned) = -128 (signed)

Note: Some compilers might warn about this overflow, but the code will still compile and run with this wraparound behavior.

Ques-14 What is the output with explanation of this C Code?

```
Code: #include <stdio.h>
      int main() {
          float f = 0.1;
          if (f == 0.1)
              printf("True");
          else
              printf("False");
      }
```

Solution- Output: False

Explanation: This demonstrates floating-point precision issues in C.

What happens:

→ float f = 0.1; The value 0.1 is stored as a float (32-bit).

→ if (f == 0.1) - The literal 0.1 is treated as a double (64-bit) by default.

→ The Comparison between a float and a double.

Ques-15 What is the output with explanation of this C code?

Code:
```
#include<stdio.h>
main(){
  int n=0, m=0;
  if(n>0)
    if(m>0)
      printf("True");
  else
    printf("False");
}
```

→ This else depicts that it is part of inner if block not outer one.

Solution- Output - False ✗

⇒ No output printed. Run the code & check.

Explanation: → here n=0, but in if condition it checks n>0, which is 0>0, that is return False statement.

→ so, the else block gonna execute.

Ques-16 Write a program in C that reads a forename, Surname and year of birth and displays the names and the year one after another sequentially.

Code:

```
#include <stdio.h>
#include <string.h>
int main(){
    char forename[50];
    char Surname[50];
    int year;

    printf("Enter forename: "); // input from user...
    scanf("%s", forename);

    printf("Enter Surname: ");
    scanf("%s", Surname);

    printf("Enter year of birth: ");
    scanf("%d", &year);

    // Display the year of Surname & forname...

    printf("\n Displaying information sequentially:\n");
    printf("%s\n", forename);
    printf("%s\n", Surname);
    printf("%d\n", year);

} return 0;
```

→ Output: Enter forename: Adarsh
Enter surname: Pandey
Enter year of birth: 2004

Displaying information sequentially:
Adarsh
Pandey
2004

Ques-17 Write a program in C that takes minutes as input, and display the total number of hours and minutes.

Code-

```
#include <stdio.h>
int main() {
    int minutes;
    scanf("%d", &minutes);
    printf("hours: %d", minutes/60);
    printf("minutes: %d", minutes%60);
    return 0;
}
```

Output: 256

hours: 4 minutes: 16

Ques-18 Write a C program to read the roll no., name and marks of three subjects and calculate the total, percentage and division.

Code-

```
#include <stdio.h>
#include <string.h>
int main() {
    int rollNO;
    char name[50];
    float subject1, subject2, subject3;
    float total, percentage;
    char division[20];

// read student information.....
printf("Enter roll number: ");
scanf("%d", &rollNo);

printf("Enter student name: ");
scanf("%s", name);

printf("Enter marks for subject1: ");
scanf("%f", &subject1);

printf("Enter marks for subject2: ");
scanf("%f", &subject2);
```

```
printf("Enter marks for Subject 3: ");
scanf("%f", &subject3);

// Calculate total and percentage

total = subject1 + subject2 + subject3;
percentage = (total / 300.0) * 100; // assuming
                // each subject is out of 100 marks

// Determine division based on percentage

if (percentage >=60) {
    strcpy(division, "First division"); }
else if (percentage >= 50) {
    strcpy(division, "Second division"); }
else if (percentage >= 35) {
    strcpy(division, "Third division"); }
else {
    strcpy(division, "Fail"); }

// Display results...

printf("\n=== STUDENT RESULTS ===\n");
printf("Roll number: %d\n", rollNo);
printf("Name: %s\n", name);
printf("Subject 1 Marks: %.2f\n", subject1);
```

```
printf("Subject 2 Marks: %.2f\n", subject2);
printf("Subject 3 Marks: %.2f\n", subject3);

printf("Total Marks: %.2f/300\n", total);
printf("Percentage: %.2f%%\n", percentage);
printf("Division: %s\n", division);

return 0;
}
```

→ Output:

```
Enter roll number : 8
Enter student name: adarsh
Enter marks for subject 1: 42
Enter marks for subject 2: 56
Enter marks for Subject 3: 67

=== STUDENT RESULT ===
Roll number: 8
Name: adarsh
Subject 1 Marks: 42.00
Subject 2 Marks: 56.00
Subject 3 Marks: 67.00
Total Marks: 165.00/300
Percentage: 55.00%
Division : Second Division
```

Ques 19 Given a 4 digit number, write program that displays the number as follows:

First line: all digits

Second line: all except first digits

.

.

.

Last line: the last digit

Solution:

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter a 4-digit number: ");
    scanf("%d", &num);

    // Print all digits
    printf("%d\n", num);

    // print last 3 digits
    printf("%d\n", num % 1000);

    // print last 2 digits
    printf("%d\n", num % 100);

    // print last digit
    printf("%d\n", num % 10);

    return 0; }
```

Output:

Enter a 4-digit number: 1234
1234
234
34
4

Ques-20 You are given two integers, say M and N. You must check whether M is an exact multiple of N, without using loops.

Solution-

```
#include<stdio.h>
int main(){
    int M, N;
    printf("Enter two integers (M and N): ");
    scanf("%d %d", &M, &N);

    if(N==0){
        printf("Division by zero is not allowed.\n");}
    else if(M%N==0){
        printf("%d is an exact multiple of %d\n", M, N);
    }else{
        printf("%d is NOT an exact multiple of %d\n", M, N);
    }
    return 0;
}
```

Output: Enter two integers (M and N): 20 5
20 is an exact multiple of 5.

Ques-21 Write a C program to enter a 4 digit number and check if it is a palindrome number or not.

Solution:
```
#include <stdio.h>
int main() {
    int num, original, reversed = 0, digit;

    printf("Enter a 4-digit number: ");
    scanf("%d", &num);

    original = num;

// Reverse the number
    digit = num % 10;
    reversed = reversed * 10 + digit;
    num = num / 10;

    digit = num % 10;
    reversed = reversed * 10 + digit;
    num = num / 10;

    digit = num % 10;
    reversed = reversed * 10 + digit;
    num /= 10

    reversed = reversed * 10 + (num % 10);
    num = num / 10;
```

```
// Check Palindrome
    if (original == reversed) {
        printf("%d is a Palindrome number\n",
                                    original);
    }

    else {
        printf("%d is NOT a Palindrome number.\n",
                                    original);
    }
    return 0;

}
```

Output: Enter a 4-digit number: 1221
1221 is a Palindrome number

Enter a 4-digit number: 1234
1234 is NOT a Palindrome number

Ques-22 Write a C program to check if a given year is a leap year or not.

Solution:

```
#include <stdio.h>
int main(){
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);

    if ((year%400==0)||(year%4==0&&year%100!=0)){
    }printf("%d is a leap year.\n", year);

    else{
    }printf("%d is NOT a leap year.\n", year);

    return 0;
}
```

Output: Enter a year: 2024
2024 is a leap year.

Ques-25 Write a C program to read a floating-point number and display the rightmost digit of its integral part.

Solution:

```
#include <stdio.h>
int main() {
    float num;
    int integralPart, rightmostDigit

    printf("Enter a floating-point number:");
    Scanf("%f",&num);

    // Get the integral part
    if (num<0)
        integralPart = -(int)(-num); // handle
                                     //negatives
                                     //properly
    else
        integralPart = (int)num;

    // Find the right most digit

    rightmostDigit = integralPart %10;
```

```
// Ensure digit is positive even for negative numbers
if (rightmostDigit < 0) {
    rightmostDigit = -rightmostDigit;
}
printf("The right most digit of the integral
        part is: %d \n", rightmostDigit);

return 0;
}
```

Output: Enter a floating-point number: 123.456
The right most digit of the integral
part is: 3.

Ques. 24 What do you mean by operator precedence and associativity?

Solution: Operator Precedence -

Operator Precedence in C (or any language) defines the priority order in which different operators are evaluated in an expression.

For example:

```
int x = 10 + 3 * 2;
```

Here:

→ * (multiplication) has higher precedence than + (addition).

→ Do first 3 * 2 = 6, then 10 + 6 = 16.

→ If precedence didn't exist, this could be misread as (10 + 3) * 2 = 26.

⇒ Precedence avoids confusion by deciding which operator to apply first.

Precedence - Which operator to evaluate first.

Associativity –

→ Associativity defines the direction of evaluation when two or more operators of the same precedence appear in an expression.

→ It can be left to right or right to left.

Example:

1. Left to Right associativity (most operators):

```
int y = 100 / 10 * 2;  // both / & * have
                       // same precedence
```

→ Evaluated left to right: (100/10)*2 = 20.

2. Right to left associativity (assignment =, unary operators):

```
int a, b, c;
a = b = c = 5;  // assignment operator is right to
                // left.
```

→ First c=5, then b=c, then a=b.
→ So all become 5.

Associativity – In which direction to evaluate when precedence is same.

Ques. 25 Define the term Implicit type Conversion and Explicit conversion.

Solution- Implicit type conversion (Type Casting)

Also called the Type Promotion or Type Casting by compiler.

→ It happen automatically when you mix different data types in an expression. The compiler converts the smaller type into a larger type (to avoid data ty loss).

Rules:

1. char, short, int → promoted to int (if not larger)
2. If an expression contains different types, Conversion happens in the following order:

char / short → int → unsigned int → long → unsigned long

unsigned long → float → double → long double

Example:

```
#include <stdio.h>
int main() {
    int a = 10;
    float b = 3.5;
    float c = a+b  // int a, is implicitly
                   // converted to float.
    printf ("c = %f\n", c);

    return 0;
}
```

Output: c = 13.500000

Here, a (int) is automatically converted to float before addition.

Explicit Type conversion: (Also called Type casting by User).

→ We manually tell the compiler to convert a value from one type to another using a cast operator.

Syntax - (type) expression

Example:

```
#include <stdio.h>
int main() {
    float a = 5.75;
    int b;
    b = (int)a; //explicit conversion
    printf("a=%f, b=%d\n", a, b);
} return 0;
```

Output: a = 5.750000, b = 5

Here, a was explicitly cast to int, so decimal part is truncated.

Implicit conversion – done automatically by compiler (safe widening conversion).

Explicit conversion – done manually by programmer using cast (type).