Multiple-choice questions

1. What is the primary purpose of data types in C programming?
   A) To define the color of variables
   B) To specify the type of data a variable can hold and manage memory efficiently
   C) To control the flow of the program
   D) To handle user input only
2. Which of the following is NOT a main category of data types in C?
   A) Basic Data Types
   B) Derived Data Types
   C) User-Defined Data Types
   D) Abstract Data Types
3. In C, what does the 'int' data type store?
   A) Single characters
   B) Whole numbers without decimal points
   C) Floating-point numbers
   D) Memory addresses
4. Explain the memory size of 'int' in most systems and provide an example of its usage. Which option correctly describes it?
   A) 1 byte; int age = 'A';
   B) 2 or 4 bytes; int age = 25;
   C) 8 bytes; int age = 3.14;
   D) 4 bytes; int age = "Hello";
5. What is the output of the following code: char grade = 'A'; printf("%c", grade);?
   A) A
   B) 65 (ASCII value)
   C) grade
   D) Error
6. How much memory does a 'char' typically occupy?
   A) 2 bytes
   B) 4 bytes
   C) 1 byte
   D) 8 bytes
7. Which data type is used for numbers with decimal points and typically occupies 4 bytes?
   A) int
   B) char

Dr.Kavitha N
Dept. of IT

C) float
D) double

8. What is the key difference between 'float' and 'double' in terms of precision and memory?
   A) Float has more precision and uses 8 bytes; double uses 4 bytes
   B) Float uses 4 bytes with less precision; double uses 8 bytes with more precision
   C) Both use 4 bytes but double is for integers
   D) Float is for characters; double is for floats
9. What is the output of: double pi = 3.14159265359; printf("%.10lf", pi);?
   A) 3.1415926536
   B) 3.14
   C) 3.14159265359
   D) Error due to precision
10. Which of the following is a derived data type in C?
   A) int
   B) Arrays
   C) struct
   D) enum
11. In an array, how are elements stored?
   A) In random memory locations
   B) In contiguous memory locations
   C) In separate files
   D) Only in heap memory
12. What does a pointer store, and what is the output of: int x = 10; int *ptr = &x; printf("%d", *ptr);?
   A) Value of x; Output: 10
   B) Address of x; Output: Memory address
   C) Name of variable; Output: x
   D) Type of x; Output: int
13. What is a function in C?
   A) A variable that stores data
   B) A block of code that performs a specific task and can return a value
   C) A type of loop
   D) A memory allocator
14. Which user-defined data type groups variables of different types under a single name?
   A) union
   B) enum

Dr.Kavitha N
Dept. of IT

C) struct
D) array

15. How does a union differ from a struct in memory usage?
A) Union members share the same memory location; struct allocates separate memory for each
B) Union allocates separate memory; struct shares memory
C) Union is for integers only; struct for floats
D) No difference in memory

16. What does an enum consist of?
A) Floating-point constants
B) Named integer constants
C) Character strings
D) Memory addresses

17. In the example: enum Days { Sunday, Monday, Tuesday }; enum Days today = Monday; printf("%d\n", today);, what is the output?
A) Monday
B) 0
C) 1
D) 2

18. Which modifier allows a variable to hold only non-negative values?
A) signed
B) unsigned
C) short
D) long

19. What is the effect of the 'long' modifier on an integer's size?
A) Reduces it to 2 bytes
B) Increases it to 4 or 8 bytes
C) Makes it unsigned
D) Converts it to float

20. What is the signed range for a 'char' data type?
A) 0 to 255
B) -128 to 127
C) -32,768 to 32,767
D) 0 to 65,535

21. Which data type has a memory size of 8 bytes and is used for high-precision floating points?
A) float

Dr.Kavitha N
Dept. of IT

B) int
C) double
D) short

22. Compare the ranges: What is the unsigned range for 'int' (assuming 4 bytes)?
A) -2,147,483,648 to 2,147,483,647
B) 0 to 4,294,967,295
C) -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
D) 0 to 255

23. What does the sizeof operator return?
A) The value of a variable
B) The size in bytes of a data type or variable
C) The address of a variable
D) The type of a variable

24. Why are data types essential for memory management in C? Provide the best reason.
A) They define how much memory to allocate and how to interpret the data
B) They only control program output
C) They are used for loops only
D) They handle file operations

25. Which data type is NOT mentioned as a basic data type in the content?
A) void
B) int
C) float
D) array

26. In the union example, why is the value of 'i' overwritten when assigning to 'f'?
A) Because unions allocate separate memory
B) Because members share the same memory location
C) Because of type conversion
D) Because of compiler error

27. How can type modifiers like 'short' and 'long' be applied, and what is their impact?
A) They change the data type to float; no impact on size
B) Short reduces size (typically 2 bytes); long increases size (4 or 8 bytes)
C) They make variables signed only
D) They are used for arrays only

28. What is the typical size of 'long long' in bytes?
A) 4 bytes
B) 8 bytes

Dr.Kavitha N
Dept. of IT

C) 2 bytes
D) 1 byte

29. Which of the following is an example of a derived data type that includes functions?
    A) struct
    B) enum
    C) Pointers
    D) Functions

30. In practical applications, why might a programmer choose 'double' over 'float'?
    A) For less precision in calculations
    B) For more precision and larger range in floating-point operations
    C) To save memory
    D) For integer storage

True or False Questions:

1. The unsigned keyword can be used with float and double data types. (True/False)
2. An int variable can store both positive and negative whole numbers. (True/False)
3. long double provides less precision than double. (True/False)
4. void is a data type that signifies the absence of a value. (True/False)
5. Arrays are considered derived data types in C. (True/False)

Predict-the-Output Questions

Q1.

```
#include <stdio.h>
int main() {
   int x = 10;
   printf("%d", x);
   return 0;
}
```

D) Error C) 0 B) Garbage A) 10

Q2.

```
#include <stdio.h>
int main() {
   char ch = 'Z';
   printf("%c", ch);
   return 0;
```


Dr.Kavitha N
Dept. of IT

}

D) Error C) ASCII value of Z B) z A) Z

Q3.

```
#include <stdio.h>
int main() {
   float f = 12.3456;
   printf("%.2f", f);
   return 0;
}
```

D) Error C) 12.34 B) 12.35 A) 12.3456

Q4.

```
#include <stdio.h>
int main() {
   double d = 3.1415926535;
   printf("%.4lf", d);
   return 0;
}
```

D) Error C) 3.14 B) 3.1416 A) 3.1415

Q5.

```
#include <stdio.h>
int main() {
   int a[3] = {5,10,15};
   printf("%d", a[1]);
   return 0;
}
```

D) Error C) 15 B) 10 A) 5

Q6.

```
#include <stdio.h>
int main() {
   int num = 50;
   int *ptr = &num;
   printf("%d", *ptr);
   return 0;
}
```

D) Error C) Garbage B) Address of num A) 50

Dr.Kavitha N
Dept. of IT

Q7.

```
#include <stdio.h>
enum Colors {RED, GREEN, BLUE};
int main() {
    enum Colors c = BLUE;
    printf("%d", c);
    return 0;
}
```

D) Error C) 2 B) 1 A) 0

Q8.

```
#include <stdio.h>
struct Student {
  int id;
  float marks;
};
int main() {
  struct Student s = {101, 88.5};
  printf("%d %.1f", s.id, s.marks);
  return 0;
}
```

D) Error C) 0 0.0 B) 101 88 A) 101 88.5

Q9.

```
#include <stdio.h>
union Data {
    int i;
    float f;
};
int main() {
    union Data d;
    d.i = 100;
    d.f = 55.5;
    printf("%d", d.i);
    return 0;
}
```

D) Error C) Garbage B) 55 A) 100

Dr.Kavitha N
Dept. of IT

Q10.

```
#include <stdio.h>
int main() {
   printf("%lu", sizeof(int));
   return 0;
}
```

Assume 4 bytes for int.

D) Depends on compiler C) 8 B) 4 A) 2

Q11.

```
#include <stdio.h>
int main() {
   int arr[10];
   printf("%lu", sizeof(arr));
   return 0;
}
```

Assume int = 4 bytes.

D) Error C) 4 B) 40 A) 10

Q12.

```
#include <stdio.h>
int main() {
   short int s = 32767;
   s = s + 1;
   printf("%d", s);
   return 0;
}
```

D) Error C) 0 B) -32768 A) 32768

Q13.

```
#include <stdio.h>
int main() {
   unsigned int u = 0;
   u = u - 1;
   printf("%u", u);
   return 0;
}
```

Assume 32-bit unsigned int.

D) Error C) -1 B) 4294967295 A) 0

Dr.Kavitha N
Dept. of IT

Q14.

```
#include <stdio.h>
int main() {
   char c = 120;
   int i = c + 10;
   printf("%d", i);
   return 0;
}
```

D) Error C) Garbage B) 120 A) 130

Q15.

```
#include <stdio.h>
typedef unsigned int UI;
int main() {
   UI num = 500;
   printf("%u", num);
   return 0;
}
```

D) unsigned int C) Error B) 0 A) 500

Q16.

```
#include <stdio.h>
int main() {
   int arr[3] = {1,2,3};
   int *p = arr;
   printf("%d", *(p+2));
   return 0;
}
```

D) Error C) 3 B) 2 A) 1

Q17.

```
#include <stdio.h>
int main() {
   int a = 5, b = 2;
   float res = a / b;
   printf("%.1f", res);
   return 0;
}
```

D) Error C) 2.50 B) 2.0 A) 2.5

Dr.Kavitha N
Dept. of IT

Q18.

```
#include <stdio.h>
int main() {
   int a = 5, b = 2;
   float res = (float)a / b;
   printf("%.1f", res);
   return 0;
}
```

D) Error C) 2 B) 2.0 A) 2.5

Q19.

```
#include <stdio.h>
int main() {
   int x = 10;
   int *p = &x;
   printf("%p", p);
   return 0;
}
```

D) Error C) 0x0 B) 10 A) Address of x in hex

Q20.

```
#include <stdio.h>
int main() {
   char c = 'A';
   printf("%lu", sizeof(c + 1));
   return 0;
}
```

Assume int = 4 bytes.

D) Error C) 8 B) 4 A) 1

Code Snippets

Q1. Predict the output (int + float)

```
#include <stdio.h>
int main() {
   int x = 10;
   float y = 3.5;
   printf("%d + %f = %f\n", x, y, x + y);
   return 0;
}
```


Dr.Kavitha N
Dept. of IT

What will be the output?

Q2. Identify the correct format specifier (long long int)

```
#include <stdio.h>
int main() {
   long long int bigNumber = 123456789012345LL;
   // Your printf statement here
   return 0;
}
```

Fill in the correct format specifier to print bigNumber.

Q3. Size of data types

```
#include <stdio.h>
int main() {
   printf("%zu\n", sizeof(short));
   printf("%zu\n", sizeof(long));
   return 0;
}
```

What two values will be printed on a typical 64-bit system?

Q4. Float vs Double

```
#include <stdio.h>
int main() {
   float f = 3.14159f;
   double d = 3.14159;
   printf("f=%f d=%lf\n", f, d);
   return 0;
}
```

What will be displayed?

Q5. Character arithmetic

```
#include <stdio.h>
int main() {
   char a = 'A';
   char b = 2;
   printf("%c\n", a + b);
   return 0;
}
```

What character will be printed?

Dr.Kavitha N
Dept. of IT

Q6. Unsigned int wraparound

```
#include <stdio.h>
int main() {
   unsigned int x = 0;
   x = x - 1;
   printf("%u\n", x);
   return 0;
}
```

Predict the output.

Q7. Long double printing

```
#include <stdio.h>
int main() {
   long double pi = 3.141592653589793238L;
   // Complete the printf statement to print pi
   return 0;
}
```

Fill in the correct format specifier for long double.

Q8. Casting float to int

```
#include <stdio.h>
int main() {
   float a = 9.99;
   int b = (int)a;
   printf("%d\n", b);
   return 0;
}
```

What is printed?

Q9. Struct size

```
#include <stdio.h>
struct Test {
   char c;
   int x;
};
int main() {
   printf("%zu\n", sizeof(struct Test));
   return 0;
}
```

On a 64-bit system with typical alignment, what will sizeof(struct Test) likely print?

Q10. Enum values

```
#include <stdio.h>
enum Colors {RED, GREEN=5, BLUE};
int main() {
   enum Colors c = BLUE;
   printf("%d\n", c);
   return 0;
}
```

What value will be printed?

Multiple-choice questions

1. Which of the following is a valid C variable name?
   a) 1num
   b) _num1
   c) int
   d) num-1

2. The default value of a local (auto) variable in C is:
   a) 0
   b) NULL
   c) Indeterminate (garbage value)
   d) -1

3. Which keyword declares a variable without allocating memory?
   a) extern
   b) static
   c) auto
   d) register

4. Which of these is not a fundamental data type in C?
   a) int
   b) float
   c) char
   d) string

5. Global and static variables are stored in:
   a) Stack
   b) Heap
   c) Data segment
   d) CPU register

6. Which declaration makes a variable constant in C?
   a) const int x = 5;
   b) int x = const 5;
   c) #define const x = 5
   d) readonly int x = 5;

7. A variable that retains its value between function calls must be declared:
   a) auto
   b) static

Dr.Kavitha N
Dept. of IT

c) register
d) extern

8. To restrict a global variable's visibility to one file, we use:
   a) auto
   b) static
   c) extern
   d) register

9. Which is true about register variables?
   a) They must be global
   b) Their address cannot be taken
   c) They always store in CPU registers
   d) They initialize automatically to 0

10. volatile keyword indicates:
    a) Variable never changes
    b) Variable may change unexpectedly
    c) Variable optimized aggressively
    d) Allocated on CPU registers

11. Variable names in C are:
    a) Not case-sensitive
    b) Case-sensitive
    c) Must be uppercase only
    d) Automatically uppercase

12. Identifiers can begin with:
    a) A digit
    b) A symbol
    c) A letter or underscore
    d) Any ASCII character

13. A global variable not explicitly initialized defaults to:
    a) 0
    b) -1
    c) Garbage value
    d) Random integer

14. The keyword auto is:
    a) Required for every local variable

Dr.Kavitha N
Dept. of IT

b) Optional because it is the default
c) Used for static memory
d) Deprecated in C

15. restrict qualifier applies only to:
a) Arrays
b) Pointers
c) Integers
d) Structures

16. Assigning a value during variable creation is called:
a) Declaration
b) Initialization
c) Instantiation
d) Aliasing

17. A static variable inside a function is destroyed when:
a) Function ends
b) Program ends
c) Next function call
d) Block ends

18. Which keyword lets you declare a variable in one file and define it in another?
a) extern
b) static
c) volatile
d) auto

19. Variables declared in a block have:
a) Global scope
b) Block scope
c) File scope
d) Function scope only

20. Variables declared outside all functions have:
a) File scope
b) Block scope
c) Function scope
d) Local scope

Dr.Kavitha N
Dept. of IT

21. Declaration vs Definition: which allocates memory?
    a) Declaration
    b) Definition
    c) extern
    d) Typedef

22. If a local variable is declared but not initialized, it contains:
    a) Zero
    b) Garbage value
    c) NULL
    d) -1

23. Using extern is useful when:
    a) You need thread safety
    b) Variable is shared between files
    c) Variable is local to a function
    d) Memory needs to be freed

24. Why explicit initialization is better than relying on defaults?
    a) Code readability
    b) Avoids bugs
    c) Portability
    d) All of the above

25. In which case does a variable persist between function calls?
    a) auto variable
    b) register variable
    c) static variable
    d) volatile variable

26. What will be printed?

```
static int x;

printf("%d", x);
```

    a) 0
    b) Garbage
    c) Compilation error
    d) Undefined

27. Output of code:

Dr.Kavitha N
Dept. of IT

auto int num;

printf("%d", num);

a) 0
b) Garbage value
c) Compile error
d) NULL

28. To declare a global variable private to a file:
a) static int marks;
b) extern int marks;
c) auto int marks;
d) register int marks;

29. Which code fragment shows extern usage across files?
a) extern int x; in one file, int x; in another
b) int x; everywhere
c) static int x; everywhere
d) const int x; everywhere

30. Can a const variable be modified through a pointer?
a) Yes, with pointer tricks
b) No, strictly prevented
c) Only at global scope
d) Only with register variables

31. An L-value is:
a) Memory location
b) Constant
c) Temporary expression
d) Literal

32. Which of the following is an invalid L-value?
a) x
b) *(ptr)
c) 10
d) arr

33. Where would you use volatile?
a) Hardware-mapped variables
b) Compiler macros

Dr.Kavitha N
Dept. of IT

c) Local arrays
d) Structs

34. restrict helps C compiler by:
a) Providing thread safety
b) Allowing memory aliasing
c) Preventing simultaneous access
d) Assuming exclusive access to pointer data

35. Difference between static inside vs outside function:
a) Inside retains value; outside restricts file-scope visibility
b) Both same
c) Inside creates globals
d) Outside persists function calls

36. Why register keyword is rarely used now?
a) Compilers optimize register allocation
b) Registers cannot store ints
c) Modern CPUs don't allow register storage
d) Undefined

37. A static local variable is initialized:
a) Each time function is called
b) Only the first time function is called
c) Never
d) Every global initialization

38. When is memory for static local array allocated?
a) At function entry
b) At program start
c) At block end
d) At first recursive call

39. Uninitialized static variables are stored as:
a) Random bytes
b) Zero in data segment/BSS
c) NULL pointers
d) Undefined values

40. Why initialize explicitly?
a) Default initialization may vary across compilers

Dr.Kavitha N
Dept. of IT

b) Garbage may cause bugs
c) Improves clarity
d) All of the above

41. What distinguishes const int *p from int * const p?
a) First: pointer constant, Second: value constant
b) First: constant pointer, Second: pointer to const
c) First: pointer to const, Second: const pointer
d) Both mean same thing

42. Can you declare register static int x;?
a) Yes
b) No, because they conflict
c) Only in global scope
d) Only in local scope

43. Automatic variables are allocated in:
a) Heap
b) Data segment
c) Stack
d) CPU registers

44. Which of these identifiers is invalid?
a) my_var
b) _temp2
c) class
d) var123

45. Using extern int x; without definition in another file causes:
a) Runtime error
b) Linker error
c) Garbage initialization
d) Automatic 0 initialization

46. For a variable shared across files but should be read-only, you declare:
a) extern const int x;
b) static const int x;
c) register const int x;
d) auto const int x;

Dr.Kavitha N
Dept. of IT

47. If global and local variable share same name, which takes precedence inside function?
    a) Global
    b) Local
    c) Random
    d) Compiler dependent

48. Variable lifetime in recursion:
    a) Auto locals persist each call separately
    b) Static locals retain value across recursion
    c) Both a & b
    d) None

49. Why is accessing uninitialized auto variables dangerous?
    a) They default to zero
    b) They may cause segmentation fault directly
    c) They contain garbage values
    d) Compiler sets random defaults

50. In embedded systems, hardware register variables should be declared as:
    a) volatile
    b) register
    c) const
    d) static

True or False Questions:

1. Variable names in C are case-insensitive.
2. The keyword auto is optional since all local variables are auto by default.
3. Static local variables are destroyed when the function ends.
4. A variable declared with extern allocates memory immediately.
5. Uninitialized global variables are automatically initialized to zero.
6. It is possible to take the address of a register variable using &.
7. The keyword restrict is used to tell the compiler that a pointer is the only reference to the object it points to.
8. volatile ensures that the compiler always reads a variable from memory instead of optimizing it into a register.

Dr.Kavitha N
Dept. of IT

9. Global variables have file scope from the point of declaration to the end of the file.

10. A constant variable declared with const can never be modified directly or indirectly.

Predict-the-Output Questions

Q1

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

a) 0
b) Garbage value
c) Compilation error
d) Undefined behavior

Q2

```
#include <stdio.h>
static int x;
int main() {
   printf("%d", x);
   return 0;
}
```

a) 0
b) Garbage value
c) Compilation error
d) Undefined

Dr.Kavitha N
Dept. of IT

Q3

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

a) 5 5
b) 5 6
c) 6 6
d) Garbage output

Q4

```
#include <stdio.h>
void fun() {
   static int count = 0;
   count++;
   printf("%d ", count);
}
int main() {
   fun(); fun(); fun();
   return 0;
}
```

a) 1 1 1
b) 0 1 2
c) 1 2 3
d) Compilation error

Dr.Kavitha N
Dept. of IT

Q5

```
#include <stdio.h>
int main() {
   register int x = 10;
   printf("%d", x);
   return 0;
}
```

a) 10
b) Garbage
c) Error
d) Address of x

Q6

```
#include <stdio.h>
int main() {
   const int x = 10;
   int *p = (int*)&x;
   *p = 20;
   printf("%d", x);
   return 0;
}
```

a) 10
b) 20
c) Error
d) Implementation-dependent

Q7

```
#include <stdio.h>
int main() {
   int a = 5;
   {
      int a = 10;
```


Dr.Kavitha N
Dept. of IT

```
        printf("%d ", a);
    }
    printf("%d", a);
    return 0;
}
```

a) 10 10
b) 10 5
c) 5 10
d) Garbage values

Q8

```
#include <stdio.h>
int x = 5;
int main() {
    int x = 10;
    printf("%d", x);
    return 0;
}
```

a) 5
b) 10
c) Garbage
d) Error

Q9

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

a) 0
b) Garbage

Dr.Kavitha N
Dept. of IT

c) Error
d) Undefined

Q10

```
#include <stdio.h>
int x = 100;
void fun() {
   extern int x;
   printf("%d", x);
}
int main() {
   fun();
   return 0;
}
```

a) 0
b) Garbage
c) 100
d) Error

Q11

```
#include <stdio.h>
int x = 10;
int main() {
   {
      extern int x;
      printf("%d", x);
   }
   return 0;
}
```

a) 0
b) 10
c) Garbage
d) Error

Dr.Kavitha N
Dept. of IT

Q12

```
#include <stdio.h>
void test() {
   static int x;
   x++;
   printf("%d ", x);
}
int main() {
   test();
   test();
   test();
   return 0;
}
```

a) 1 1 1
b) 1 2 3
c) 0 1 2
d) Garbage

Q13

```
#include <stdio.h>
int main() {
   volatile int x = 5;
   printf("%d", x);
   x = 10;
   printf(" %d", x);
   return 0;
}
```

a) 5 5
b) 5 10
c) Garbage values
d) Undefined

Dr.Kavitha N
Dept. of IT

Q14

```
#include <stdio.h>
int main() {
   int x = 5;
   int *p = &x;
   *p = *p + 5;
   printf("%d", x);
   return 0;
}
```

a) 5
b) 10
c) Garbage
d) Error

Q15

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

a) 5
b) 6
c) Error
d) Garbage

Q16

```
#include <stdio.h>
int main() {
   int a = 5, b = 10;
   {
      int b = a;
```


Dr.Kavitha N
Dept. of IT

```
        printf("%d %d", a, b);
    }
    return 0;
}
```

a) 5 10
b) 5 5
c) 10 5
d) Garbage

Q17

```
#include <stdio.h>
int a = 1;
void foo() {
    static int a = 5;
    a++;
    printf("%d ", a);
}
int main() {
    foo();
    foo();
    printf("%d", a);
    return 0;
}
```

a) 6 7 1
b) 6 7 7
c) 1 2 1
d) Error

Q18

```
#include <stdio.h>
int main() {
    static int x = 10;
    if (x-- > 7) {
```


Dr.Kavitha N
Dept. of IT

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

a) 9 8 7
b) 10 9 8
c) 7 6 5
d) Infinite recursion

Q19

```
#include <stdio.h>
int main() {
    int arr[3] = {0};
    for (int i = 0; i < 3; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

a) 0 0 0
b) Garbage values
c) Error
d) Undefined behavior

Q20

```
#include <stdio.h>
int main() {
    static int arr[3];
    for (int i = 0; i < 3; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

a) 0 0 0
b) Garbage values
c) Error
d) Undefined behavior

Dr.Kavitha N
Dept. of IT

Multiple Choice Questions

1. Which section of a C program usually contains information about the program's author and purpose?
   a) Definition Section
   b) Documentation Section
   c) Main Function
   d) Preprocessor Section
2. What is the primary purpose of comments in C?
   a) Increase execution speed
   b) Provide documentation and readability
   c) Store variable values
   d) Define constants
3. Which of the following is a valid single-line comment in C?
   a) /* This is a comment */
   b) # This is a comment
   c) // This is a comment
   d) -- This is a comment
4. What happens if you forget to close a multi-line comment (*/)?
   a) The compiler will automatically close it
   b) The compiler will ignore it
   c) It causes a compilation error
   d) It converts to single-line comment
5. Which type of comment can span multiple lines?
   a) // comment
   b) /* comment */
   c) ### comment
   d) None of these
6. Consider the code:

   // int x = 10;

Dr.Kavitha N
Dept. of IT

/* printf("%d",x); */

What will be the output?

a) 10

b) Compilation error

c) No output

d) Garbage value

7. Which statement is true about comments?
   a) They affect the executable size
   b) They are executed before main()
   c) They are ignored by the compiler
   d) They must be declared
8. Can comments be nested in C?
   a) Yes, always
   b) Only single-line comments
   c) No, nested comments cause errors
   d) Only with #define

Preprocessor Section

9. Which section of a C program contains #include directives?
   a) Documentation Section
   b) Definition Section
   c) Preprocessor Section
   d) Main Function
10. Which of the following includes the standard input-output header in C?
    a) #include <stdio.h>
    b) include <stdio.h>

Dr.Kavitha N
Dept. of IT

c) @include stdio.h

d) import stdio.h

11. The #include directive is processed by:

a) Compiler

b) Linker

c) Preprocessor

d) Assembler

12. What happens if you include a header file twice without guards?

a) Program runs faster

b) Program will not compile due to multiple definitions

c) Program runs slower

d) No effect

13. Choose the correct statement about header files:

a) They must always end with .h

b) They can be user-defined or standard

c) They are optional for printf

d) They execute before main

Definition Section

14. In which section would you define a constant using #define?

a) Global Declaration Section

b) Definition Section

c) Subprogram Section

d) Main Function

15. Which is the correct way to define a constant PI as 3.14?

a) define PI = 3.14;

b) #define PI 3.14

Dr.Kavitha N
Dept. of IT

c) const define PI = 3.14;

d) int PI = 3.14;

16. What is the scope of constants defined using #define?

a) Only in main()

b) Throughout the file after definition

c) Limited to a block

d) Only in functions

17. Which keyword defines a constant variable whose type is checked by the compiler?

a) #define

b) const

c) static

d) extern

18. What is the difference between #define and const in C?

a) const is a preprocessor directive

b) #define allows type checking

c) const creates a typed variable

d) They are identical

19. What will happen?

```
#define VALUE 10
int main() {
  printf("%d", VALUE*VALUE);
  return 0;
}
```

a) 10

b) 100

c) Compilation error

d) Undefined

Global Declaration Section

20. Variables declared in the global declaration section are accessible:
    a) Only in main()
    b) Only in subprograms
    c) Throughout the program
    d) Only in header files
21. Which of the following can be declared globally?
    a) Variables
    b) Function prototypes
    c) Both a and b
    d) Only #defines
22. A variable declared outside any function is called:
    a) Local variable
    b) Global variable
    c) Static variable
    d) Register variable
23. The default initial value of a global int variable in C is:
    a) 0
    b) Undefined
    c) Garbage
    d) -1
24. Which statement is true about global variables?
    a) They are stored in the stack
    b) They are automatically extern
    c) They are accessible to all functions
    d) They can't be modified

Dr.Kavitha N
Dept. of IT

Main() Function

25. What is the starting point of execution in every C program?
    a) Documentation Section
    b) main() function
    c) Preprocessor
    d) Global Declaration
26. How many main() functions can a C program have?
    a) Unlimited
    b) Two
    c) Only one
    d) Depends on compiler
27. What is the correct return type of main() in standard C?
    a) void
    b) int
    c) char
    d) float
28. If main() does not return a value explicitly, what happens?
    a) Compilation error
    b) Returns 0 automatically (in most modern compilers)
    c) Returns garbage
    d) Undefined behavior
29. Which statement is true about main()?
    a) It must be the first function
    b) It must be the last function
    c) It can call other functions
    d) It cannot call user-defined functions

Subprograms (User-defined Functions)

Dr.Kavitha N
Dept. of IT

30. Subprograms in C are also called:
    a) Modules
    b) User-defined functions
    c) Blocks
    d) Loops
31. Functions are declared outside main() but:
    a) Cannot be called
    b) Must be called from main() or other functions
    c) Execute automatically
    d) Need #define
32. Advantages of functions include:
    a) Modularity
    b) Reusability
    c) Better organization
    d) All of these
33. Which keyword is used to declare a function prototype?
    a) function
    b) prototype
    c) extern
    d) None of these
34. If a function is declared but not defined, the compiler:
    a) Ignores it
    b) Throws an error at compile-time
    c) Throws an error at link-time
    d) Executes without it
35. Where should user-defined functions be written?
    a) Inside main()
    b) Outside main()
    c) In preprocessor section
    d) Inside #define

Dr.Kavitha N
Dept. of IT

Arithmetic Expressions

36. Which of the following is NOT an arithmetic operator in C?
    a) +
    b) -
    c) *
    d) &&
37. What will be the output?

    ```
    int a=5;
    printf("%d", ++a);
    ```

    a) 5
    b) 6
    c) Error
    d) Undefined
38. Post-increment means:
    a) Increment occurs after the value is used
    b) Increment occurs before the value is used
    c) No increment
    d) None
39. The ternary operator in C is:
    a) ? :
    b) if-else
    c) switch
    d) ::
40. Which expression correctly uses the ternary operator?
    a) x > y ? x : y;
    b) x > y then x else y;

Dr.Kavitha N
Dept. of IT

c) x > y :: x : y;

d) (x>y)?

41. Which of the following will compute remainder?

a) /

b) %

c) *

d) rem()

42. What is the output?

```
int a=5,b=10;
int result=(a>b)?a:b;
printf("%d",result);
```

a) 5

b) 10

c) 0

d) Error

43. The operator ++a is called:

a) Pre-increment

b) Post-increment

c) Unary minus

d) None

44. Which operator has the highest precedence in C?

a) *

b) /

c) ++

d) %

45. Which is true about operator associativity?

a) Decides order when precedence is same

b) Irrelevant in C

Dr.Kavitha N
Dept. of IT

c) Decides data types

d) None

46. Which expression correctly calculates area of circle using #define PI?

a) area=PI*r*r;

b) area=PI*r*^2*;*

*c) area=PI(*r*r*);

d) area=PI*r*r();

47. What will be printed?

```
int a=5;
printf("%d\n",a++);
printf("%d\n",a);
```

a) 6 then 7

b) 5 then 6

c) 6 then 6

d) 5 then 5

48. The evaluation of arithmetic expressions depends on:

a) Precedence

b) Associativity

c) Data types

d) All of these

49. Which operator can work as both unary and binary in C?

a) -

b) %

c) ++

d) /

50. Which section of a C program would you place function prototypes to make them accessible to main()?

a) Documentation Section

b) Definition Section

c) Global Declaration Section

d) Preprocessor Section

True /false Questions

1. The documentation section of a C program is compulsory for execution.
(True / False)

2. Comments in C improve readability but do not affect program execution.
(True / False)

3. In C, single-line comments start with // and multi-line comments start with /* and end with */.
(True / False)

4. Nested multi-line comments (/* /* ... */ */) are allowed in standard C.
(True / False)

5. The preprocessor section of a C program is executed after the main() function.
(True / False)

6. Header files included using #include are processed by the preprocessor before compilation.
(True / False)

7. Constants defined with #define are replaced by the preprocessor before the code is compiled.
(True / False)

8. Global variables declared outside any function are automatically initialized to zero if not explicitly initialized.
(True / False)

Dr.Kavitha N
Dept. of IT

9. A C program can have multiple main() functions.
(True / False)

10. The execution of a C program always starts from the main() function.
(True / False)

11. User-defined functions must always be declared before the main() function.
(True / False)

12. The ternary operator ?: is the only operator in C that takes three operands.
(True / False)

13. The operator % can be used with floating-point numbers in C.
(True / False)

14. The const keyword defines a typed constant variable in C.
(True / False)

15. The minus sign - can act as both a unary and binary operator in C.
(True / False)

Predict the output questions:

Q1

```
#include <stdio.h>
int main() {
   // Printing Hello
   printf("Hello ");
   /* This part is commented
   printf("World");
   */
   return 0;
```


Dr.Kavitha N
Dept. of IT

```
}
```

a) Hello World

b) Hello

c) World

d) No output

Q2

```
#include <stdio.h>
#define VALUE 5+5
int main() {
   printf("%d", VALUE*VALUE);
   return 0;
}
```

a) 25

b) 100

c) 55

d) Compilation error

Q3

```
#include <stdio.h>
const int x = 10;
int main() {
   printf("%d", x);
   return 0;
}
```


Dr.Kavitha N
Dept. of IT

a) 10

b) Garbage value

c) Compilation error

d) 0

Q4

```
#include <stdio.h>
int x = 5;
int main() {
    int x = 10;
    printf("%d", x);
    return 0;
}
```

a) 5

b) 10

c) Garbage

d) Compilation error

Q5

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


Dr.Kavitha N
Dept. of IT

a) 0

b) Garbage

c) Error

d) Undefined

Q6

```
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    int result = (a > b) ? a : b;
    printf("%d", result);
    return 0;
}
```

a) 5

b) 10

c) 0

d) Error

Q7

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


Dr.Kavitha N
Dept. of IT

a) 5

b) 6

c) Error

d) Undefined

Q8

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

a) 5 6

b) 6 6

c) 6 7

d) 5 5

Q9

```
#include <stdio.h>
#define PI 3.14
int main() {
    float r = 2;
    printf("%.2f", PI*r*r);
    return 0;
}
```


Dr.Kavitha N
Dept. of IT

a) 12.56

b) 6.28

c) 3.14

d) Error

Q10

```
#include <stdio.h>
int main() {
    int a = -5;
    printf("%d", -a);
    return 0;
}
```

a) -5

b) 5

c) Error

d) Garbage

Q11

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

```
    printf("%d", a);
    return 0;
}
```

a) 10 5
b) 5 10
c) 10 10
d) 5 5

Q12

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

a) 5 6
b) 5 7
c) 6 7
d) 7 7

Q13

```
#include <stdio.h>
int main() {
    printf("%d", sizeof(3.14));
```


Dr.Kavitha N
Dept. of IT

```
    return 0;
}
```

a) 4

b) 8

c) Compiler dependent

d) Error

Q14

```
#include <stdio.h>
int main() {
    int a=5,b=2;
    printf("%d", a/b);
    return 0;
}
```

a) 2.5

b) 2

c) 3

d) Error

Q15

```
#include <stdio.h>
int main() {
    int a = 5;
    printf("%d", (a>3)?(a<10)?1:2:3);
    return 0;
```

```
}
```

a) 1

b) 2

c) 3

d) Error

Q16

```
#include <stdio.h>
int main() {
   int a=3;
   a+=a+=a+=2;
   printf("%d",a);
   return 0;
}
```

a) 3

b) 7

c) 11

d) Error

Q17

```
#include <stdio.h>
int main() {
   int a=5,b=0;
   if (b=a)
      printf("True");
```

```
    else
        printf("False");
    return 0;
}
```

a) True
b) False
c) Error
d) Garbage

Q18

```
#include <stdio.h>
int main() {
    int x=2;
    printf("%d %d", x<<1, x>>1);
    return 0;
}
```

a) 2 2
b) 4 1
c) 1 4
d) Error

Q19

```
#include <stdio.h>
int main() {
    int x=5;
```


Dr.Kavitha N
Dept. of IT

```
    printf("%d", x==5?x+1:x-1);
    return 0;
}
```

a) 4
b) 5
c) 6
d) Error

Q20

```
#include <stdio.h>
void fun() {
    printf("Hello");
}
int main() {
    fun();
    return 0;
}
```

a) Hello
b) Error
c) No output
d) Garbage

Dr.Kavitha N
Dept. of IT

Multiple Choice Questions

1–10: Arithmetic Expressions & Unary/Binary/Ternary Operators

1. Which of the following is a valid arithmetic expression in C?
   a) 5 +
   b) a * b
   c) + -
   d) * /
2. Unary operators in C work on:
   a) One operand
   b) Two operands
   c) Three operands
   d) Four operands
3. The ternary operator ?: in C requires how many operands?
   a) 1
   b) 2
   c) 3
   d) 4
4. What is the output of:

   ```
   int a = 5;
   printf("%d", -a);
   ```

   a) 5
   b) -5
   c) 0
   d) Error

5. What does the following expression do?

   ```
   result = (a > b) ? a : b;
   ```

   a) Assigns a to result
   b) Assigns b to result
   c) Assigns the greater of a and b to result
   d) Error

6. Which of these is not a unary operator?
   a) ++
   b) --

c) +
d) * (binary multiplication)

7. Pre-increment ++a vs post-increment a++ differs in:
   a) When the increment happens
   b) Variable type
   c) Operator precedence
   d) Nothing
8. If a = 5; b = 10; what is the output of:

   ```
   printf("%d", a + b);
   ```

   a) 5
   b) 10
   c) 15
   d) 50

9. Which operator can change the sign of a number?
   a) ++
   b) - (unary)
   c) %
   d) *
10. Consider:

    ```
    int a = 7, b = 3;
    int res = (a < b) ? a : b;
    ```

    res will be:
    a) 3
    b) 7
    c) 10
    d) Error

11–20: Arithmetic & Assignment Operators

11. Which operator gives the remainder after division?
    a) /
    b) %
    c) *
    d) +

12. Which of the following assigns and adds at the same time?
    a) a = a + 3
    b) a += 3
    c) a =+ 3
    d) Both a and b
13. If x = 10; x -= 4; the value of x is:
    a) 6
    b) 14
    c) 4
    d) 10
14. Which of the following will increment a by 1?
    a) a++
    b) ++a
    c) Both
    d) None
15. Which is true about / operator in C?
    a) Returns remainder
    b) Returns quotient
    c) Returns both quotient and remainder
    d) Error
16. % operator is undefined for:
    a) int / int
    b) float / int
    c) int / int
    d) char / int
17. a *= 5; is equivalent to:
    a) a = 5
    b) a = a + 5
    c) a = a * 5
    d) a = 5 * 5
18. If a = 7; b = 2; printf("%d", a % b); output is:
    a) 3
    b) 1
    c) 2
    d) 0
19. Which operator cannot be used with floating-point numbers?
    a) +
    b) *
    c) %
    d) -

20. Which of these is a binary arithmetic operator?
    a) ++
    b) - (unary)
    c) + (binary)
    d) !

21–30: Relational & Logical Operators

21. a == b checks:
    a) a not equal to b
    b) a equal to b
    c) a greater than b
    d) a less than b
22. Which returns true if both conditions are true?
    a) ||
    b) &&
    c) !
    d) %
23. Which operator negates a condition?
    a) &&
    b) ||
    c) !
    d) ~
24. Output of:

    ```
    int a = 5, b = 10;
    printf("%d", a >= b);
    ```

    a) 0
    b) 1
    c) 5
    d) 10

25. Which operator is used in if a number is divisible by 3 or 5?
    a) &&
    b) ||
    c) %
    d) Both b and c

26. if (a != b) executes when:
    a) a equals b
    b) a not equal to b
    c) a > b
    d) a < b
27. Logical AND && has higher precedence than:
    a) +
    b) *
    c) ||
    d) ==
28. Which of these evaluates first?
    a) a && b
    b) a || b
    c) a + b
    d) a == b
29. What will !(5 > 3) evaluate to?
    a) 0
    b) 1
    c) 5
    d) 3
30. Which operator is short-circuiting in C?
    a) &&
    b) ||
    c) Both a and b
    d) %

31–40: Bitwise & Miscellaneous Operators

31. num & 1 checks:
    a) Even number
    b) Odd number
    c) Greater than 1
    d) Less than 1
32. ~a performs:
    a) Addition
    b) Bitwise NOT
    c) Subtraction
    d) XOR

33. a << 2 means:
    a) Divide a by 2
    b) Multiply a by 2
    c) Left shift a by 2 bits
    d) Right shift a by 2 bits
34. a | b is:
    a) Bitwise AND
    b) Bitwise OR
    c) Bitwise XOR
    d) Logical OR
35. *ptr gives:
    a) Address of ptr
    b) Value at ptr
    c) Bitwise NOT
    d) None
36. sizeof(char) returns:
    a) 1
    b) 2
    c) 4
    d) Depends on system
37. Comma operator , in:

    int result = (a=5, b=10, a+b);

    gives:
    a) 5
    b) 10
    c) 15
    d) Error

38. &a gives:
    a) Value of a
    b) Address of a
    c) Pointer value
    d) NULL
39. Output of:

    int a = 5;
    printf("%p", &a);

a) 5
b) Address of a
c) Error
d) Garbage

40. Which operator is right-to-left associative?
    a) =
    b) +
    c) *
    d) /

41–50: Format Specifiers & Input/Output

41. Correct specifier for unsigned int x; is:
    a) %d
    b) %u
    c) %f
    d) %ld
42. Correct format specifier for long long int is:
    a) %lld
    b) %ld
    c) %ll
    d) %l
43. printf("%.2f", 12.3456); outputs:
    a) 12.34
    b) 12.35
    c) 12.3456
    d) 12
44. scanf("%d", &x); requires:
    a) Variable name
    b) Address of variable
    c) Value of variable
    d) Pointer variable
45. %c in printf is used to print:
    a) Integer
    b) Character
    c) Float
    d) String

46. Which specifier prints hexadecimal?
    a) %x
    b) %o
    c) %d
    d) %f
47. Which prints number of characters printed so far?
    a) %n
    b) %d
    c) %s
    d) %c
48. Inputting string with spaces requires:
    a) %s
    b) gets() or scanf("%[^\n]", str)
    c) %c
    d) %f
49. Output of:

    ```
    int age = 20;
    printf("Age = %d", age);
    ```

    a) Age = 20
    b) Age = %d
    c) 20
    d) Error

50. Which of the following is correct to print pointer address?
    a) %p
    b) %d
    c) %x
    d) %f

51–60: Operator Precedence & Associativity

51. Which has the highest precedence?
    a) *
    b) ()
    c) +
    d) &&
52. Which operator has right-to-left associativity?
    a) *

b) =
c) +
d) &&

53. In a + b * c, which operation occurs first?
    a) a + b
    b) b * c
    c) a + c
    d) Depends on compiler
54. Which has lowest precedence?
    a) ,
    b) =
    c) ?:
    d) &&
55. Precedence of == is higher than:
    a) >
    b) <
    c) +
    d) &&
56. Associativity of ternary operator ?: is:
    a) Left to right
    b) Right to left
    c) Depends on compiler
    d) Both
57. Which of these evaluates first?
    a) a = b + c
    b) b + c
    c) a
    d) =
58. In x = y = z = 5; value of x is:
    a) 5
    b) 0
    c) Depends on system
    d) Error
59. Which is evaluated first in a + b << c?
    a) a + b
    b) b << c
    c) a + b << c together
    d) << has higher precedence
60. Operator with higher precedence than * is:
    a) /

b) %
c) Unary ++
d) +

61–70: Tricky Unary/Binary Questions

61. If a = 5; printf("%d", a++ + ++a); output is:
    a) 11
    b) 12
    c) 10
    d) Undefined
62. --a + a++ with a = 3 → result is:
    a) 5
    b) 6
    c) 4
    d) 3
63. Unary * is used for:
    a) Multiplication
    b) Pointer dereference
    c) Both
    d) Bitwise AND
64. Unary & is used to:
    a) Bitwise AND
    b) Get address
    c) Logical AND
    d) Dereference
65. Which operator cannot be chained?
    a) +
    b) =
    c) ? :
    d) *
66. Output of:

```
int a = 5;
printf("%d", a-- - --a);
```

    a) 1
    b) 0

c) -1
d) 2

67. In result = - -a; value of result when a = 5:
    a) -5
    b) 5
    c) 0
    d) Undefined
68. Postfix increment has higher precedence than:
    a) Unary +
    b) Multiplication
    c) Binary +
    d) Ternary
69. Unary ! applied to 0 gives:
    a) 0
    b) 1
    c) -1
    d) Undefined
70. Which of these is a common interview pitfall?
    a) a+++b
    b) a + ++b
    c) a + b
    d) a += b

71–80: Bitwise & Conditional Operators

71. Output of 5 & 3 in binary:
    a) 7
    b) 1
    c) 0
    d) 8
72. Output of 5 | 2 in decimal:
    a) 7
    b) 3
    c) 5
    d) 2
73. 5 ^ 3 outputs:
    a) 7

b) 6
c) 1
d) 0

74. 5 << 1 gives:
    a) 2
    b) 10
    c) 5
    d) 0
75. 10 >> 2 gives:
    a) 2
    b) 5
    c) 0
    d) 3
76. Conditional operator ?: can replace:
    a) if-else
    b) switch-case
    c) for loop
    d) while loop
77. Output of:

```
int a = 7;
(a % 2 == 0) ? printf("Even") : printf("Odd");
```

    a) Even
    b) Odd
    c) 7
    d) Error

78. Ternary operator associativity is:
    a) Left to right
    b) Right to left
    c) Depends on compiler
    d) None
79. Can bitwise operators be used on float?
    a) Yes
    b) No
    c) Only OR
    d) Only AND
80. Which operator is used to check even/odd using bits?
    a) & 1

b) %2
c) |1
d) <<1

81–90: Miscellaneous Operators & Pointers

81. sizeof(int) typically gives:
    a) 1
    b) 2
    c) 4
    d) 8
82. int *ptr = &a; then *ptr = 10; changes:
    a) ptr value
    b) a value
    c) both
    d) none
83. Comma operator (a=5, b=10, a+b) evaluates to:
    a) 5
    b) 10
    c) 15
    d) 0
84. printf("%p", &a); prints:
    a) 10
    b) Value of a
    c) Address of a
    d) Error
85. Dereference operator * is used with:
    a) Arrays
    b) Pointers
    c) Strings
    d) None
86. Which is not a valid C operator?
    a) **
    b) ->
    c) ~
    d) ,
87. What is the result of:

```
int a = 5;
int b = 10;
int c = (a=1, b=2, a+b);
```

a) 1
b) 2
c) 3
d) 12

88. & operator is unary when used as:
    a) Bitwise AND
    b) Address-of
    c) Logical AND
    d) Dereference
89. * operator is unary when used as:
    a) Pointer dereference
    b) Multiplication
    c) Both
    d) None
90. Which of these is useful for memory-efficient programs?
    a) Comma operator
    b) Bitwise operators
    c) Ternary operator
    d) All of above

91–100: Input/Output & Format Specifiers

91. Correct specifier for float x is:
    a) %d
    b) %f
    c) %lf
    d) %c
92. To print 3 decimal places of a float:
    a) %.2f
    b) %.3f
    c) %f
    d) %d

93. %s is used for:
    a) Single character
    b) String
    c) Integer
    d) Float
94. scanf("%d", &x); requires:
    a) Variable
    b) Address of variable
    c) Pointer variable
    d) Both b and c
95. Correct printf to show number of characters printed:
    a) %n
    b) %d
    c) %s
    d) %c
96. Specifier for long double:
    a) %Lf
    b) %lf
    c) %f
    d) %d
97. %u is used for:
    a) int
    b) unsigned int
    c) float
    d) char
98. Output of:

```
float num = 12.34567;
printf("%.2f", num);
```

    a) 12.34
    b) 12.35
    c) 12.34567
    d) 12

99. To input a string with spaces:
    a) %s
    b) scanf("%[^\n]", str)
    c) %c
    d) %d

100. %p is used to print:
   a) Pointer address
   b) Integer
   c) Float
   d) Character

True / False Questions:

1. Unary operators work with only one operand.
2. The ternary operator ?: can take only two operands.
3. ++a is called a pre-increment operator.
4. a++ increments the value before using it in an expression.
5. Binary operators always work with two operands.
6. The modulus operator % can be used with floating-point numbers.
7. The precedence of * is higher than +.
8. The comma operator , has the lowest precedence in C.
9. The ternary operator is right-to-left associative.
10. Logical AND && has higher precedence than relational operators like < or >.
11. The bitwise AND & can be used to check if a number is odd or even.
12. The sizeof operator returns the memory size of a data type or variable in bytes.
13. The dereference operator * is unary when used with pointers.
14. The address-of operator & can only be used with variables, not constants.
15. The assignment operator = has left-to-right associativity.
16. printf("%d", 10/3); prints 3.
17. printf("%.2f", 12.3456); prints 12.35.
18. scanf("%d", &x); requires the address of the variable.
19. The logical NOT operator ! applied to a non-zero value returns 1.
20. int *ptr = &a; *ptr = 10; changes the value of a to 10.
21. Bitwise operators can be applied to floating-point numbers.
22. The relational operator >= returns 1 for true and 0 for false.
23. a+++b is valid and equivalent to (a++) + b.
24. Unary minus -a can be used to negate a number.
25. Postfix increment a++ has higher precedence than unary -.
26. The ternary operator can be used as a shorthand for if-else statements.
27. %s format specifier is used to print a single character.
28. %lld is used for printing long long int values.
29. int a = (b=5, b+2); sets a to 7.
30. printf("%p", &a); prints the value of variable a.

Predict the Output Questions

Basics of Arithmetic Expressions

1.

```
int a = 5;

printf("%d\n", -a);
```

Options:
A) 5
B) -5
C) Garbage value
D) 0

2.

```
int a = 10, b = 20;

printf("%d\n", a + b);
```

Options:
A) 30
B) 10
C) 20
D) Garbage

3.

```
int a = 7, b = 2;

printf("%d\n", a / b);
```

Options:
A) 3.5
B) 3
C) 3.0
D) Error

4.

```
int a = 7, b = 2;

printf("%d\n", a % b);
```

Options:
A) 1
B) 2
C) 3
D) Error

5.

```
int a = 5;

printf("%d\n", ++a);
```

Options:
A) 6
B) 5
C) Garbage
D) Compile Error

6.

```
int a = 5;

printf("%d\n", a++);
```

Options:
A) 5
B) 6
C) Garbage
D) Compile Error

7.

```
int a = 5, b = 10;

int result = (a > b) ? a : b;

printf("%d\n", result);
```

Options:
A) 5
B) 10
C) 15
D) Garbage

8.

int a = -5;

printf("%d\n", +a);

Options:
A) 5
B) -5
C) +5
D) Error

9.

int a = 12, b = 5;

printf("%d\n", a - b * 2);

Options:
A) 14
B) 2
C) 22
D) 7

10.

int a = 2;

printf("%d\n", a * 3 + 4 / 2);

Options:
A) 8
B) 7
C) 6
D) 10

Relational and Logical Operators

11.

int a = 10, b = 20;

printf("%d\n", a < b);

Options:
A) 1

B) 0
C) 10
D) 20

12.

```
int a = 10, b = 10;

printf("%d\n", a == b);
```

Options:
A) 1
B) 0
C) Error
D) Garbage

13.

```
int a = 10;

printf("%d\n", a != 5);
```

Options:
A) 1
B) 0
C) Garbage
D) Error

14.

```
int x = 5;

printf("%d\n", (x > 2) && (x < 10));
```

Options:
A) 1
B) 0
C) 5
D) Compile Error

15.

```
int x = 7;

printf("%d\n", (x % 2 == 0) || (x > 5));
```

Options:
A) 1
B) 0
C) Garbage
D) Error

16.

```
int x = 0;

printf("%d\n", !x);
```

Options:
A) 0
B) 1
C) Garbage
D) Error

17.

```
int a = 1, b = 0;

printf("%d\n", a && b);
```

Options:
A) 1
B) 0
C) Error
D) Garbage

18.

```
int a = 1, b = 0;

printf("%d\n", a || b);
```

Options:
A) 1
B) 0
C) Error
D) Garbage

19.

```
int a = 5, b = 2;
```

```
printf("%d\n", (a > b) ? (a - b) : (b - a));
```

Options:
A) 3
B) -3
C) 2
D) 0

20.

```
int x = 10, y = 5;

printf("%d\n", (x < y) && (x / y));
```

Options:
A) 1
B) 0
C) 2
D) 10

Assignment & Compound Assignment

21.

```
int a = 5;

a += 3;

printf("%d\n", a);
```

Options:
A) 5
B) 8
C) 3
D) Garbage

22.

```
int a = 10;

a -= 4;

printf("%d\n", a);
```

Options:
A) 6
B) 4
C) 14
D) Garbage

23.

```
int a = 3;

a *= 2 + 3;

printf("%d\n", a);
```

Options:
A) 15
B) 12
C) 9
D) 6

24.

```
int a = 20;

a /= 4;

printf("%d\n", a);
```

Options:
A) 5
B) 4
C) 20
D) Garbage

25.

```
int a = 13;

a %= 5;

printf("%d\n", a);
```

Options:
A) 3
B) 2

C) 5
D) Garbage

Bitwise Operators

26.

int a = 5, b = 3;

printf("%d\n", a & b);

Options:
A) 1
B) 7
C) 5
D) 3

27.

int a = 5, b = 3;

printf("%d\n", a | b);

Options:
A) 1
B) 7
C) 5
D) 3

28.

int a = 5, b = 3;

printf("%d\n", a ^ b);

Options:
A) 6
B) 7
C) 2
D) 1

29.

int a = 5;

printf("%d\n", ~a);

Options:
A) -6
B) 5
C) 6
D) Error

30.

int a = 1;

printf("%d\n", a << 3);

Options:
A) 8
B) 3
C) 2
D) 1

Precedence and Associativity

31.

int a = 2, b = 3, c = 4;

printf("%d\n", a + b * c);

Options:
A) 14
B) 20
C) 24
D) 10

32.

int a = 2, b = 3, c = 4;

printf("%d\n", (a + b) * c);

Options:
A) 14
B) 20

C) 24
D) 10

33.

int x = 4, y = 2;

printf("%d\n", x / y * y);

Options:
A) 4
B) 2
C) 1
D) 0

34.

int x = 8, y = 3;

printf("%d\n", x % y * y);

Options:
A) 2
B) 6
C) 3
D) 5

35.

int x = 3;

printf("%d\n", x > 2 ? x++ : ++x);

Options:
A) 3
B) 4
C) 5
D) Error

Formatting and I/O

36.

float num = 12.34567;

printf("%.2f\n", num);

Options:
A) 12.34567
B) 12.35
C) 12.34
D) 12

37.

printf("%d %d %d\n", sizeof(char), sizeof(int), sizeof(long));

Options:
A) 1 4 8
B) 1 2 4
C) 1 4 4
D) Depends on compiler

38.

int x = 65;

printf("%c\n", x);

Options:
A) A
B) 65
C) Garbage
D) Error

39.

char str[] = "Hello";

printf("%s\n", str);

Options:
A) Hello
B) H
C) Garbage
D) Error

40.

int age;

```
scanf("%d", &age);

printf("Age = %d\n", age);
```

Input: 21
Options:
A) Age = 20
B) Age = 21
C) Garbage
D) Error

Tricky Questions

41.

```
int a = 5;

printf("%d %d\n", a++, ++a);
```

Options:
A) 5 7
B) 6 6
C) 6 7
D) Undefined behavior

42.

```
int a = -3 % 2;

printf("%d\n", a);
```

Options:
A) -1
B) 1
C) 0
D) Error

43.

```
int x;

printf("%d\n", x);
```

Options:
A) Garbage
B) 0
C) Error
D) Undefined output

44.

```
int a = 0;

if (a = 5) printf("YES\n");

else printf("NO\n");
```

Options:
A) YES
B) NO
C) Error
D) Garbage

45.

```
int a = 10, b = 0;

printf("%d\n", b || a++);

printf("%d\n", a);
```

Options:
A) 1 10
B) 1 11
C) 0 10
D) 0 11

46.

```
int a = 10, b = 20;

int result = a, b;

printf("%d\n", result);
```

Options:
A) 10
B) 20

C) Syntax Error
D) Garbage

47.

```
int a = 1, b = 2, c = 3;

printf("%d\n", a = b = c);
```

Options:
A) 1
B) 2
C) 3
D) Error

48.

```
printf("%d\n", printf("Hello"));
```

Options:
A) Hello5
B) Hello4
C) Hello6
D) Error

49.

```
int a = 5;

printf("%d\n", a == 5 ? a = 10 : a = 20);
```

Options:
A) 5
B) 10
C) 20
D) Error

50.

```
int i = 3;

printf("%d %d %d\n", i, i<<1, i>>1);
```

Options:
A) 3 6 1

B) 3 7 1
C) 3 6 2
D) 3 7 2