Question Bank

for

Subject Title: Computer Programming

Subject Code:
CS-214

Prepared by:
Dr. Preetpal Kaur Buttar
Assistant Professor (CSE)

Department of Computer Science & Engineering
Sant Longowal Institute of Engineering & Technology,
Longowal

Table of Contents

Sr. No.	Description	Page No.
1	Multiple Choice Questions (MCQs)	1-7
2	Short Answer Questions	8-10
3	Descriptive Type Questions	11-27

Multiple Choice Questions (MCQs)

Q. No.		Answer
1	Consider the following variable declarations and definitions in C. i) `int var_9 = 1;` ii) `int 9_var = 2;` iii) `int _ = 3;` Choose the correct statement w.r.t. above variables. a) Both i) and iii) are valid. b) Only i) is valid. c) Both i) and iii) are valid. d) All are valid.	a)
2	Let `x` be an integer which can take a value of 0 or 1. The statement `if(x == 0) x = 1; else x = 0;` is equivalent to which one of the following? a) `x = 1 + x;` b) `x = 1 – x;` c) `x = x – 1;` d) `x = 1 % x;`	b)
3	For a given integer, which of the following operators can be used to “set” and “reset” a particular bit respectively? a) `|` and `&` b) `&&` and `||` c) `&` and `|` d) `||` and `&&`	a)

4	Assume `int` is 4 bytes, `char` is 1 byte and `float` is 4 bytes. Also, assume that pointer size is 4 bytes.
	`char *pChar;`
	`int *pInt;`
	`float *pFloat;`
	
	`sizeof(pChar);`
	`sizeof(pInt);`
	`sizeof(pFloat);`
	What’s the size returned for each of `sizeof()` operator?
	a) `4 4 4` b) `1 4 4` c) `1 4 8` d) `1 1 1`
		a)

5	What’s going to happen when we compile and run the following C program snippet?
	`#include "stdio.h"`
	`int main(){`
	` int a = 10;`
	` int b = 15;`
	
	` printf("=%d",(a+1),(b=a+2));`
	` printf(" %d=",b);`
	
	` return 0;`
	`}`
	a) `=11 15=` b) `=11 12=`
	c) Compiler error due to `(b=a+2)` in the first `printf()`.
	d) No compiler error but output would be `=11 X=` where `X` would depend on compiler.
		b)

6	Which of the following statement is correct for `switch` controlling expression? a) Only `int` can be used in `switch` control expression. b) Both `int` and `char` can be used in `switch` control expression. c) All types, i.e., `int`, `char` and `float` can be used in `switch` control expression. d) `switch` control expression can be empty as well.	b)
7	Which of the following is correct with respect to “Jump Statements” in C? a) `goto` b) `continue` c) `break` d) `return` e) All of these.	e)
8	In the context of `break` and `continue` statements in C, pick the best statement? a) `break` can be used in `for`, `while` and `do-while` loop body. b) `continue` can be used in `for`, `while` and `do-while` loop body. c) `break` and `continue` can be used in `for`, `while`, `do-while` loop body and `switch` body. d) `break` and `continue` can be used in `for`, `while` and `do-while` loop body. But only `break` can be used in `switch` body.	d)

9	What’s the meaning of following declaration in C language? `int (*p)[5];` a) `p` is a pointer to 5 integers. b) `p` is a pointer to integer array. c) `p` is an array of 5 pointers to integers. d) `p` is a pointer to an array of 5 integers	d)
10	Suppose `a`, `b`, `c` and `d` are `int` variables. For ternary operator in C (`?:`), pick the best statement. a) `a>b ? : ;` is valid statement i.e. 2nd and 3rd operands can be empty and they are implicitly replaced with non-zero value at run-time. b) `a>b ? c=10 : d=10;` is valid statement. Based on the value of `a` and `b`, either `c` or `d` gets assigned the value of 10. c) `a>b ? (c=10, d=20) : (c=20, d=10);` is valid statement. Based on the value of `a` and `b`, either `c=10, d=20` gets executed or `c=20, d=10` gets executed. d) All of the above are valid statements for ternary operator.	c)

11	Pick the best statement for the following program.
	`#include "stdio.h"`
	`int foo(int a){`
	` printf("%d",a);`
	` return 0;`
	`}`
	`int main(){`
	` foo;`
	` return 0;`
	`}`
	a) It’ll result in a compiler error because `foo` is used without parentheses.
	b) No compile error and some garbage value would be passed to `foo` function. This would make `foo` to be executed with output “garbage integer”.
	c) No compile error, but `foo` function will not be executed. The program will not print anything.
	d) No compile error and ZERO (i.e., 0) would be passed to `foo` function. This would make `foo` to be executed with output `0`.
		c)

12	The below program would give a compiler error because comma has been used after `foo()`. Instead, a semi-colon should be used, i.e., the way it has been used after `bar()`. That's why, if we use semi-colon after `foo()`, the program would compile and run successfully while printing `SLIETLongowal`.
	`#include <stdio.h>`
	`void foo(void){`
	` printf("SLIET");`
	`}`
	`void bar(void){`
	` printf("Longowal");`
	`}`
	`int main(){`
	` foo(), bar();`
	` return 0;`
	`}`
	a) True b) False
		b)

13	In C, 1D array of int can be defined as follows and both are correct.
	`int array1D[4] = {1,2,3,4};`
	`int array1D[] = {1,2,3,4};`
	But given the following definitions (along-with initialization) of 2D arrays
	`int array2D[2][4] = {1,2,3,4,5,6,7,8}; /* (i) */`
	`int array2D[][4] = {1,2,3,4,5,6,7,8}; /* (ii) */`
	`int array2D[2][] = {1,2,3,4,5,6,7,8}; /* (iii) */`
	`int array2D[][] = {1,2,3,4,5,6,7,8}; /* (iv) */`
	Pick the correct statements.
		b)

	a) Only (i) is correct. b) Only (i) and (ii) are correct. c) Only (i), (ii) and (iii) are correct. d) All (i), (ii), (iii) and (iv) are correct.	

14	In a C program, the following variables are defined:
	`float       x = 2.17;`
	`double    y = 2.17;`
	`long double z = 2.17;`
	Which of the following is correct way for printing these variables via `printf`.
	a) `printf("%f %lf %Lf",x,y,z);`
	b) `printf(“%f %f %f”,x,y,z);`
	c) `printf("%f %ff %fff",x,y,z);`
	d) `printf("%f %lf %llf",x,y,z);`
		a)

15	In a C program snippet, the following are used for the definition of integer variables.
	`signed s;`
	`unsigned u;`
	`long l;`
	`long long ll;`
	Pick the best statement for these.
	a) All of the above variable definitions are incorrect because the basic type `int` is missing.
	b) All of the above variable definitions are correct because `int` is implicitly assumed in all of these.
	c) Only `long l;` and `long long ll;` are valid definitions of variables.
	d) Only `unsigned u;` is valid definition of variable.
		b)

16	What is the return type of getchar()? a) `int` b) `char` c) `unsigned char` d) `float`	a)

17	Predict the output of the following program:
	`#include <stdio.h>`
	`int main(){`
	`    int i = (1, 2, 3);`
	
	`    printf("%d", i);`
	
	`    return 0;`
	`}`
	a) `1` b) `3`
	c) Garbage value d) Compiler-time error
		b)

18	Which of the following can have different meaning in different contexts? a) `&` b) `*` c) Both of the above. d) There are no such operators in C.	c)
19	In C, two integers can be swapped using minimum ………… extra variable(s). a) 0 b) 1 c) 2 d) 4	a)

20	Assume that the size of an integer is 4 bytes, predict the output of following program.
	`#include <stdio.h>`
	`int main(){`
	`    int i = 12;`
	`    int j = sizeof(i++);`
	`    printf("%d  %d", i, j);`
	`    return 0;`
	`}`
	a) `12 4` b) `13 4` c) Compiler error d) `0 4`
		a)

21	Which of the following is not a logical operator? a) `&&` b) `!` c) `||` d) `|`	d)

22	What would be the output of the following program?
	`#include <stdio.h>`
	`int main(){`
		b)

	`  int i;`
	`  for (i = 1; i != 10; i += 2)`
	`    printf(" SLIET ");`
	`  return 0;`
	`}`
	a) `SLIET SLIET SLIET SLIET SLIET`
	b) `SLIET SLIET SLIET` ... infinite times
	c) `SLIET SLIET SLIET SLIET`
	d) `SLIET SLIET SLIET SLIET SLIET SLIET`	

23	How many times will `SLIET` be printed in the following program?
	`#include <stdio.h>`
	`int main(){`
	`    int i = 1024;`
	`    for (; i; i >>= 1)`
	`        printf("SLIET");`
	`    return 0;`
	`}`
	a) 10 b) 11
	c) Infinite d) The program will show compile-time error
		b)

24	What would be the output of the following program?
	`#include<stdio.h>`
	`int main(){`
	`  int a = 2,b = 5;`
	`  a = a^b;`
	`  b = b^a;`
	`  printf("%d %d",a,b);`
	`  return 0;`
	`}`
	a) `5 2` b) `2 5` c) `7 7` d) `7 2`
		d)

25	What is the output of the following?
	`#include <stdio.h>`
	`int main(){`
	`    int x = 10;`
	`    int y = 20;`
	`    x += (y += 10);`
	`    printf("%d %d", x, y);`
	`    return 0;`
	`}`
	a) `40 20` b) `40 30` c) `30 30` d) `30 40`
		b)

26	What is the output of given program?
	`#include <stdio.h>`
	`int main(){`
	`    int x = 10;`
	`    int y = (x++, x++, x++);`
	`    printf("%d %d\n", x, y);`
	`    return 0;`
	`}`
	a) `13 12` b) `13 13` c) `10 10` d) Compiler dependent
		a)

27	What is the output of the following program?
	`#include <stdio.h>`
	`int main() {`
	`    int i = 2;`
	`    switch (i) {`
	`        case 0:`
	`            printf("SLIET");`
	`            break;`
		c)

	`        case 1:`
	`            printf("Longowal");`
	`            break;`
	`        default:`
	`            printf("SLIETLongowal");`
	`    }`
	`    return 0;`
	`}`
	a) `SLIET` b) `Longowal` c) `SLIETLongowal` d) Compiler error	

28	Consider the following program fragment:
	`if(a > b)`
	`if(b > c)`
	`s1;`
	`else s2;`
	`s2` will be executed if
	a) `a <= b` b) `b > c` c) `b >= c and a <= b` d) `a > b and b <= c`
		d)

29	What will be the output of the following?
	`#include <stdio.h>`
	`int i;`
	`int main() {`
	`    if (i) {`
	`        // Do nothing`
	`    } else {`
	`        printf("Else");`
	`    }`
	`    return 0;`
	`}`
	a) `if` block is executed. b) `else` block is executed.
	c) It is unpredictable as `i` is not initialized. d) Error: misplaced else
		b)

30	In C, the sizes of an integer and a pointer must be same. a) True b) False	b)

31	Predict the output of the following program:
	`#include <stdio.h>`
	`int main(){`
	`    char a = 012;`
	
	`    printf("%d", a);`
	
	`    return 0;`
	`}`
	a) Compiler error b) `12` c) `10` d) Empty
		c)

32	Consider the following C function:
	`void swap (int a, int b){`
	`   int temp;`
	`   temp = a;`
	`   a = b;`
	`   b = temp;`
	`}`
	In order to exchange the values of two variables `x` and `y`
	a) Call `swap (x, y)`.
	b) Call `swap (&x, &y)`.
	c) `swap(x,y)` cannot be used as it does not return any value.
	d) `swap(x,y)` cannot be used as the parameters are passed by value.
		d)

33	What will be the value of `j` at the end of the execution of the following C program?
	`int incr(int i){`
	`   static int count = 0;`
	`   count = count + i;`
		a)

	`   return (count);`
	`}`
	`main(){`
	`   int i,j;`
	`   for (i = 0; i <=4; i++)`
	`      j = incr(i);`
	`}`
	a) `10` b) `4` c) `6` d) `7`	

34	Which of the following is true about return type of functions in C? a) Functions can return any type. b) Functions can return any type except array and functions. c) Functions can return any type except array, functions and union. d) Functions can return any type except array, functions, function pointer and union.	b)
35	Which of the following storage classes have global visibility in C/C++ ? a) `auto` b) `extern` c) `static` d) `register`	b)
36	Which of the following is not a storage class specifier in C? a) `auto` b) `volatile` c) `static` d) `register`	b)
37	In C, `static` storage class cannot be used with: a) global variable. b) function parameter. c) function name. d) local variable.	b)
38	Which of the following is true about arrays in C? a) For every type T, there can be an array of T. b) When an array is passed to a function, C compiler creates a copy of array. c) 2D arrays are stored in column major form. d) For every type T except void and function type, there can be an array of T.	d)

39	Consider the following C program, which variable has the longest scope?
	`int a;`
	`int main(){`
	`   int b;`
	`   // ..`
	`   // ..`
	`}`
	`int c;`
	a) `a` b) `b`
	c) `c` d) All have same scope.
		a)

40	When an array is passed as parameter to a function, which of the following statements is correct? a) The function can change values in the original array. b) In C, parameters are passed by value, the function cannot change the original value in the array. c) It results in a compilation error when the function tries to access the elements in the array. d) Results in a run time error when the function tries to access the elements in the array.	a)
41	Consider the following statements S1, S2 and S3: S1: In call-by-value, anything that is passed into a function call is unchanged in the caller's scope when the function returns. S2: In call-by-reference, a function receives an implicit reference to a variable used as argument. S3: In call-by-reference, caller is unable to see the modified variable used as argument. a) S3 and S2 are true. b) S3 and S1 are true. c) S2 and S1 are true. d) S1, S2, and S3 are true.	c)

42	The following C declarations
	`struct node`
	`{`
	`   int i;`
	`   float j;`
	`};`
	`struct node *s[10] ;`
		a)

	define `s` to be a) an array, each element of which is a pointer to a structure of type node. b) a structure of 2 fields, each field being a pointer to an array of 10 elements. c) a structure of 3 fields: an integer, a float, and an array of 10 elements. d) an array, each element of which is a structure of type node	
43	The number of tokens in the following C statement is `printf("HELLO WORLD");` a) 3 b) 5 c) 9 d) 8	b)
44	Which of the following best describes C language? a) C is a low level language. b) C is a high level language with features that support low level programming. c) C is a high level language. d) C is a very high level language.	b)
45	As per C language standard, which of the followings is/are not keyword(s)? Pick the best statement. `auto make main sizeof elseif` a) `sizeof elseif make` b) `make main elseif` c) `make main` d) `auto make`	b)
46	Which of the following has the compilation error in C? a) `int n = 17;` b) `char c = 99;` c) `float f = (float)99.32;` d) `<include>`	d)
47	Which of the following true about `FILE *fp`? a) `FILE` is a keyword in C for representing files and `fp` is a variable of `FILE` type. b) `FILE` is a structure and `fp` is a pointer to the structure of `FILE` type. c) `FILE` is a stream. d) `FILE` is a buffered stream.	b)
48	When `fopen()` is not able to open a file, it returns a) `EOF` b) `NULL` c) runtime error d) compiler dependent	b)
49	`getc()` returns `EOF` when a) end of files is reached. b) when `getc()` fails to read a character. c) Both of the above d) None of the above	c)
50	In `fopen()`, the open mode `wx` is sometimes preferred to `w` because: 1) Use of `wx` is more efficient. 2) If `w` is used, old contents of file are erased and a new empty file is created. When `wx` is used, `fopen()` returns `NULL`,if file already exists. a) Only 1 b) Only 2 c) Both 1 and 2 d) Neither 1 nor 2	b)

Short Answer Questions

Q. No.		
1	Q.	Why is the `main()` function special?
	A.	It’s the first function executed when the program starts.
2	Q.	What is meant by the structure of a program?
	A.	The structure of a program is defined by its control flow, as structures are built up of blocks of codes. These blocks have a single entry and exit in the control flow.
3	Q.	Name and describe the usual purpose of three expressions in a `for` statement.
	A.	The initialize expression initializes the loop variable, the test expression tests the loop variable, and the increment expression changes the loop variable.
4	Q.	Write a `for` loop that displays the numbers from 100 to 110.
	A.	`for(int j=100; j<=110; j++)` `printf(“%d\n”, j);`
5	Q.	Write a structure specification that includes three variables—all of type `int`—called `hrs`, `mins`, and `secs`. Call this structure `time`.
	A.	`struct time {` `int hrs;` `int mins;` `int secs;` `};`
6	Q.	What is the purpose of using argument names in a function declaration?
	A.	To clarify the purpose of the arguments.
7	Q.	What is the significance of empty parentheses in a function declaration?
	A.	Empty parentheses mean the function takes no arguments.
8	Q.	What is a principal reason for passing arguments by reference?
	A.	To modify the original argument or to avoid copying a large argument.
9	Q.	What is the structure of C program syntax?
	A.	Any C program can be divided into header, `main()` function, variable declaration, body, and return type of the program.
10	Q.	Why C is a structured programming language?
	A.	C is a structured programming language because it divides the programs into small modules called functions which makes the execution easier.
11	Q.	What is the difference between variable declaration and definition in C?
	A.	In variable declaration, only the name and type of the variable is specified but no memory is allocated to the variable. In variable definition, the memory is also allocated to the declared variable.
12	Q.	What is meant by a variable’s scope?
	A.	The scope of a variable is the region in which the variable exists, and it is valid to perform operations on it. Beyond the scope of the variable, we cannot access it, and it is said to be out of scope.
13	Q.	What is the difference between `intarr[3]` and `*(intarr+3)`?
	A.	They both do the same thing.
14	Q.	What is the difference between the ‘`=`’ and ‘`==`’ operators?
	A.	‘`=`’ is a type of assignment operator that places the value in right to the variable on left, whereas ‘`==`’ is a type of relational operator that is used to compare two elements if the elements are equal or not.
15	Q.	What is the difference between prefix and postfix operators in C?
	A.	In prefix operations, the value of a variable is incremented/decremented first and then the new value is used in the operation, whereas, in postfix operations first the value of the variable is used in the operation and then the value is incremented/decremented.

16	Q.	How many types of decision-making statements are there in the C language?
	A.	There are 5 types of conditional statements or decision-making statements in C language: 1. `if` Statement 2. `if`-`else` Statement 3. `if`-`else`-`if` Ladder 4. `switch` Statement 5. Conditional Operator
17	Q.	Can we skip braces around the body of the if-else block in C?
	A.	We can skip the braces of the body of the if or else block as long as there is only a single statement inside their body. We will get an error if there is more than one statement in the body without braces.
18	Q.	Why do we need arrays?
	A.	We can use normal variables when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.
19	Q.	How can we determine the size of the C array?
	A.	We can determine the size of the Array using sizeof operator in C. We first get the size of the whole array and divide it by the size of each element type.
20	Q.	What is a forward declaration?
	A.	Sometimes we define the function after its call to provide better readability. In such cases, we declare function before their definition and call. Such declarations are called forward declarations.
21	Q.	What is the difference between function declaration and definition?
	A.	The data like function name, return type, and parameter are included in the function declaration whereas the definition is the body of the function. All these data are shared with the compiler according to their corresponding steps.
22	Q.	Can we return multiple values from a C function?
	A.	No, it is generally not possible to return multiple values from a function. But we can either use pointers to static or heap memory locations to return multiple values or we can put data in the structure and then return the structure.
23	Q.	What are actual and formal parameters?
	A.	Formal parameter: The variables declared in the function prototype are known as formal arguments or parameters. Actual parameter: The values that are passed in the function are known as actual arguments or parameters.

24	Q.	What is the size of the given union?
		`union un {`
		`int a;`
		`int arr[20];`
		`}`

	A.	The size of the given union is 20 x 4 bytes = 80 bytes. Even if the array is a collection of similar data elements, it is considered to be a single entity by the C compiler.
25	Q.	Can we store data in multiple union members at the same time?
	A.	No. We can only store data in a single member at the same time.
26	Q.	What is the difference between a constant pointer and a pointer to a constant?
	A.	A constant pointer points to the fixed memory location, i.e. we cannot change the memory address stored inside the constant pointer. On the other hand, the pointer to a constant point to the memory with a constant value.
27	Q.	Why do we need to specify the type in the pointer declaration?
	A.	Type specification in pointer declaration helps the compiler in dereferencing and pointer arithmetic operations.
28	Q.	What is the function of `sizeof` operator?
	A.	Returns the size, in bytes, to store an object of a given type name or of the type of a given expression.
29	Q.	What is a null pointer?

	A.	Pointer whose value is 0. A null pointer is valid but does not point to any object.
30	Q.	What do you mean by the term ‘declaration’?
	A.	It asserts the existence of a variable, function, or type defined elsewhere. Names may not be used until they are defined or declared.

Descriptive Type Questions

Q. No.

1 Q. What are the differences between constant defined using `const` qualifier and `#define`?

A. The following table lists the differences between the constants defined using `const` qualifier and `#define` in C:

Constants using `const`	**Constants using `#define`**
They are the variables that are immutable, i. e., they cannot be changed during the course of the program.	They are the macros that are replaced by their value.
They are handled by the compiler.	They are handled by the preprocessor.
Syntax: `const type name = value;`	Syntax: `#define name value`

Example:
```
#include <stdio.h>

int main()
{

    const int int_const = 25;

    const char char_const = 'A';

    const  float  float_const  =
15.66;

    printf("Printing   value   of
Integer Constant: %d\n",
           int_const);
    printf("Printing   value   of
Character Constant: %c\n",
           char_const);
    printf("Printing   value   of
Float Constant: %f",
           float_const);

    return 0;
}
```

Example:
```
#include <stdio.h>

// Defining macros with constant
value
#define PI 3.14159265359

int main()
{

     int radius = 21;
     int area;

     area = PI * radius * radius;

     printf("Area  of  Circle  of
radius %d: %d", radius, area);

     return 0;
}
```

2 Q. Explain a way to change the value of a constant variable in C?

A. We can take advantage of the loophole created by pointers to change the value of a variable declared as a constant in C. The below program demonstrates how to do it.

```
// C Program to change the value of a constant variable
#include <stdio.h>
int main()
{
    // defining an integer constant
    const int var = 10;
    printf("Initial Value of Constant: %d\n", var);
    // defining a pointer to that const variable
    int* ptr = (int*)&var; // explicit cast to remove constness
    // changing value
    *ptr = 500;
    printf("Final Value of Constant: %d\n", var);
    printf("Accessing through pointer: %d\n", *ptr);
    return 0;
}
```

Output

```
Initial Value of Constant: 10
Final Value of Constant: 500
Accessing through pointer: 500
```

3 Q. How can we specify multiple conditions in `if` statement?

A. We can specify multiple conditions in the `if` statement but not separately. We have to join these multiple conditions using logical operators making them into a single expression. We can then use this expression in the `if` statement.

Valid Expressions:

```
if (a < b && a < c);
if (a == 25 || a < 25);
```

Invalid Expressions:

```
if (a < b, a < c);
```

In the above expression, the rightmost expression in the parenthesis will be considered.

4 Q. Explain the difference between function arguments and parameters with the help of (a) suitable example(s)?

A. **Argument:** An argument is referred to the values that are passed within a function when the function is called. These values are generally the source of the function that require the arguments during the process of execution. These values are assigned to the variables in the definition of the function that is called. The type of the values passed in the function is the same as that of the variables defined in the function definition. These are also called Actual arguments or Actual Parameters.

Example: Suppose a `sum()` function is needed to be called with two numbers to add. These two numbers are referred to as the arguments and are passed to the `sum()` when it called from somewhere else.

```
// C code to illustrate Arguments

#include <stdio.h>

// sum: Function definition
int sum(int a, int b)
{
    // returning the addition
    return a + b;
}

// Driver code
int main()
{
    int num1 = 10, num2 = 20, res;
    // sum() is called with
    // num1 & num2 as ARGUMENTS.
    res = sum(num1, num2);

    // Displaying the result
    printf("The summation is %d", res);
    return 0;
}
```

Output:

```
The summation is 30
```

Parameters: The parameter is referred to as the variables that are defined during a function declaration or definition. These variables are used to receive the arguments that are passed during a function call. These parameters within the function prototype are used during the execution of the function for which it is defined. These are also called Formal arguments or Formal Parameters.

Example: Suppose a `Mult()` function is needed to be defined to multiply two numbers. These two numbers are referred to as the parameters and are defined while defining the function `Mult()`.

```
// C code to illustrate Parameters

#include <stdio.h>

// Mult: Function definition
// a and b are the PARAMETERS
int Mult(int a, int b)
{
    // returning the multiplication
    return a * b;
}

// Driver code
int main()
{
    int num1 = 10, num2 = 20, res;

    // Mult() is called with
    // num1 & num2 as ARGUMENTS.
    res = Mult(num1, num2);

    // Displaying the result
    printf("The multiplication is %d", res);
    return 0;
}
```

Output:

```
The multiplication is 200
```

Difference between Argument and Parameter

Argument	**Parameter**
When a function is called, the values that are passed during the call are called as arguments.	The values which are defined at the time of the function prototype or definition of the function are called as parameters.

These are used in function call statement to send value from the calling function to the receiving function.	These are used in function header of the called function to receive the value from the arguments.
During the time of call each argument is always assigned to the parameter in the function definition.	Parameters are local variables which are assigned value of the arguments when the function is called.
They are also called Actual Parameters	They are also called Formal Parameters

5 Q. Discuss the applications of unions with the help of C code.

A. Unions can be useful in many situations where we want to use the same memory for two or more members. For example, suppose we want to implement a binary tree data structure where each leaf node has a double data value, while each internal node has pointers to two children, but no data. If we declare this as:

```
struct NODE {
    struct NODE* left;
    struct NODE* right;
    double data;
};
```

then every node requires 16 bytes, with half the bytes wasted for each type of node. On the other hand, if we declare a node as the following, then we can save space.

```
struct NODE {
    bool is_leaf;
    union {
        struct {
            struct NODE* left;
            struct NODE* right;
        } internal;
        double data;
    } info;
};
```

6 Q. Explain the difference between arrays and pointers with the help of suitable examples.

A. Difference between Arrays and Pointers
The following table lists the major differences between an array and a pointer:

Pointer	**Array**
It is declared as: `*var_name;`	It is declared as: `data_type var_name[size];`
It is a variable that stores the address of another variable.	It is the collection of elements of the same data type.
We can create a pointer to an array.	We can create an array of pointers.

A pointer variable can store the address of only one variable at a time.	An array can store a number of elements the same size as the size of the array variable.
Pointer allocation is done during runtime.	Array allocation is done during compile runtime.
The nature of pointers is dynamic. The size of a pointer in C can be resized according to user requirements which means the memory can be allocated or freed at any point in time.	The nature of arrays is static. During runtime, the size of an array in C cannot be resized according to user requirements.

Distinguishing Features of Arrays and Pointers
1. Behavior of `sizeof` operator

When used with arrays, `sizeof` operator returns the size in bytes occupied by the entire array whereas when used with pointers, it returns the size in bytes of the pointer itself regardless of the data types it points to.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    int arr[] = { 10, 20, 30, 40, 50, 60 };
    int* ptr = arr;

    // sizof(int) * (number of element in arr[]) is printed
    cout << "Size of arr[] " << sizeof(arr) << "\n";

    // sizeof a pointer is printed which is same for all
    // type of pointers (char *, void *, etc)
    cout << "Size of ptr " << sizeof(ptr);
    return 0;
}
```

Output:

```
Size of arr[] 24
Size of ptr 8
```

Properties of Array that Make it Resemble Pointers

Although array and pointer are different concepts, the following properties of array make them look similar.

1. Array name gives the address of the first element of the array

When we use the array name in the program, it implicitly represents the memory address of the first element in the array.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    int arr[] = { 10, 20, 30, 40, 50, 60 };
    // Assigns address of array to ptr
    int* ptr = arr;
    cout << "Value of first element is " << *ptr;
    return 0;
}
```

Output:

```
Value of first element is 10
```

2. Array members are accessed using pointer arithmetic

The compiler uses pointer arithmetic to access the array elements. For example, an expression like "`arr[i]`" is treated as `*(arr + i)` by the compiler. That is why the expressions like `*(arr + i)` work for array `arr`, and expressions like `ptr[i]` also work for pointer `ptr`.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    int arr[] = { 10, 20, 30, 40, 50, 60 };
    int* ptr = arr;
    cout << "arr[2] = " << arr[2] << "\n";
    cout << "*(arr + 2) = " << *(arr + 2) << "\n";
    cout << "ptr[2] = " << ptr[2] << "\n";
    cout << "*(ptr + 2) = " << *(ptr + 2) << "\n";
    return 0;
}
```

Output:

```
arr[2] = 30
*(arr + 2) = 30
ptr[2] = 30
*(ptr + 2) = 30
```

3. Array parameters are always passed as pointers, even when we use square brackets

When an array is passed as a parameter to a function, the array name is converted to a pointer to its first element and the function receives the pointer that points to the first element of the array instead of the entire array.

Example:

```
#include <bits/stdc++.h>
using namespace std;
int fun(int ptr[])
{
    int x = 10;
    // Size of a pointer is printed
    cout << "sizeof(ptr) = "
         << (int)sizeof(*ptr)
         << endl;
    // This allowed because ptr is a
    // pointer, not array
    ptr = &x;
    cout <<"*ptr =  " << *ptr;
    return 0;
}
int main()
{
    int arr[] = { 10, 20, 30, 40, 50, 60 };

    // Size of a array is printed
    cout << "sizeof(arr) = "
         << (int)sizeof(arr)
         << endl;
    fun(arr);
    return 0;
}
```

Output:

```
sizeof(arr) = 24
sizeof(ptr) = 4
*ptr = 10
```

7 Q. What are tokens in C?

A. Tokens are identifiers or the smallest single unit in a program that is meaningful to the compiler. In C we have the following tokens:

- **Keywords:** Predefined or reserved words in the C programming language. Every keyword is meant to perform a specific task in a program. C programming language supports 32 keywords.
- **Identifiers:** Identifiers are user-defined names that consist of an arbitrarily long sequence of digits or letters with either a letter or the underscore (`_`) as a first Character. Identifier names can't be equal to any reserved keywords in the C programming language. There are a set of rules which a programmer must follow in order to name an identifier in C.
- **Constants:** Constants are normal variables that cannot be modified in the program once they are defined. Constants refer to a fixed value. They are also referred to as literals.
- **Strings:** Strings in C are an array of characters that end with a null character `'\0`. Null character indicates the end of the string.
- **Special Symbols:** Some special symbols in C have some special meaning and thus, they cannot be used for any other purpose in the program. `#`, `=`, `{}`, `()`, `,`, `*`, `;`, `[]` are the special symbols in C programming language.
- **Operators:** Symbols that trigger an action when they are applied to any variable or any other object. Unary, Binary, and ternary operators are used in the C Programming language.

8 Q. What are header files and their uses?

A. C language has numerous libraries which contain predefined functions to make programming easier. Header files contain predefined standard library functions. All header files must have a '.h' extension.
Header files contain function definitions, data type definitions, and macros which can be imported with the help of the preprocessor directive '#include'.
Preprocessor directives instruct the compiler that these files are needed to be processed before the compilation.
There are two types of header files, i. e., user-defined header files and pre-existing header files. For example, if our code needs to take input from the user and print desired output to the screen then '`stdio.h`' header file must be included in the program as `#include<stdio.h>`. This header file contains functions like `scanf()` and `printf()` which are used to take input from the user and print the content.

9 Q. Explain the use of break and continue statements in switch case statements.

A. **The `switch` Statement**

The **switch** and **case** statements help control complex conditional and branching operations. The **switch** statement transfers control to a statement within its body.

Syntax:
selection-statement :
switch (*expression***)** *statement*
labeled-statement:
case *constant-expression***:** *statement*
default: *statement*

Control passes to the statement whose **case** *constant-expression* matches the value of **switch (***expression***)**. The **switch** statement can include any number of **case** instances, but no two case constants within the same **switch** statement can have the same value. Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a **break** statement transfers control out of the body.

Use of the **switch** statement usually looks something like this:

```
switch (expression) {
    declarations
    . .
    . .
    case constant-expression:
        statements executed if the expression equals the value of this constant-expression
        .
        .
        .
        break;
    default:
        statements executed if expression does not equal any case constant-expression
}
```

We can use the **break** statement to end processing of a particular case within the **switch** statement and to branch to the end of the **switch** statement. Without **break**, the program continues to the next **case**, executing the statements until a **break** or the end of the statement is reached. In some situations, this continuation may be desirable.

The **default** statement is executed if no **case** *constant-expression* is equal to the value of **switch (***expression***)**. If the **default** statement is omitted, and no **case** match is found, none of the statements in the **switch** body are executed. There can be at most one **default** statement. The **default** statement need not come at the end; it can appear anywhere in the body of the **switch** statement. In fact, it is often more efficient if it appears at the beginning of the **switch** statement.

A **case** or **default** label can only appear inside a **switch** statement. The type of **switch** *expression* and **case** *constant-expression* must be integral. The value of each **case** *constant-expression* must be unique within the statement body.

The **case** and **default** labels of the **switch** statement body are significant only in the initial test that determines where execution starts in the statement body. Switch statements can be nested. Any static variables are initialized before executing into any **switch** statements.
The following examples illustrates **switch** statements:

```
switch(c) {
      case 'A':
            capa++;
      case 'a':
            lettera++;
      default :
            total++;
}
```

All three statements of the `switch` body in this example are executed if `c` is equal to '`A`' since a `break` statement does not appear before the following case. Execution control is transferred to the first statement: `capa++;` and continues in order through the rest of the body. If `c` is equal to '`a`', `lettera` and `total` are incremented. Only `total` is incremented if `c` is not equal to '`A`' or '`a`'.

10 Q. Differentiate between recursion and iteration.

A. **Difference between recursion and iteration**

Recursion and iteration are two very commonly used, powerful methods of solving complex problems, directly harnessing the power of the computer to calculate things very quickly. Both methods rely on breaking up the complex problems into smaller, simpler steps that can be solved easily, but the two methods are subtlety different. Iteration, perhaps, is the simpler of the two.

In iteration, a problem is converted into a train of steps that are finished one at a time, one after another. For instance, if you want to add up all the whole numbers less than 5, you would start with 1 (in the 1st step), then (in step 2) add 2, then (step 3) add 3, and so on. In each step, you add another number (which is the same number as the number of the step you are on). This is called "iterating through the problem." The only part that really changes from step to step is the number of the step, since you can figure out all the other information (like the number you need to add) from that step number. This is the key to iteration: using the step number to find all of your other information. The classic example of iteration in languages like BASIC or C++, of course, is the for loop.

If iteration is a bunch of steps leading to a solution, **recursion is like piling all of those steps on top of each other and then quashing them all into the solution.** Recursion is like holding a mirror up to another mirror: in each image, there is another, smaller image that's basically the same.

Example:

Recursion

```
int factorial(int number) {
      if (number < 0) {
            printf("\nError - negative argument to factorial\n");
            exit(1);
      }
      else if (number == 0)
            return 1;
      else
            return (number * factorial(number - 1));
}
```

Iteration:

```
int factorial(int number) {
      int product = 1;
      if (number < 0) {
            printf("\nError - negative argument to factorial\n");
            exit(1);
      }
      else if (number == 0)
            return 1;
      else {
            for ( ; number > 0 ; number--)
                  product *= number;
            return product;
      }
}
```

So, the difference between iteration and recursion is that **with iteration, each step clearly leads onto the next, like steppingstones across a river**, while **in recursion, each step replicates itself at a smaller scale, so that all of them combined together eventually solve the problem.** These two basic methods are very important to understand fully, since they appear in almost every computer algorithm ever made.

11 Q. What are the different storage classes in C++.

A. C++ provides 4 storage class specifiers:
AUTO, REGISTER, EXTERN, and STATIC

An identifier's storage class specifier helps determine its storage class, scope, and linkage.
Storage class - determines the period during which that identifier exists in memory
Scope - determines if the identifier can be referenced in a program
Linkage - determines for a mulitple-source-file program whether an identifier is known only in the current source file or in any source file with proper declarations.

Automatic storage class
The `auto` and `register` keywords are used to declare variables of the automatic storage class. Such variables are created when the block in which they are declared is entered, they exist while the block is active, and they are destroyed when the block is exited.

Example:
`auto float x, y;`
declares that `float` variables `x` and `y` are local variables of automatic storage class, they exist only in the body of the function in which the definition appears.

`register int counter = 1;`
declares that the integer variable `counter` be placed in one of the computer's register and be initialized to 1.

Either write `auto` or `register` but not both for an identifier.

Static storage class
The keywords `extern` and `static` are used to declare identifiers for variables and functions of the static storage class. Such variables exist from the point at which the program begins execution.

There are two types of identifiers with static storage class: external identifiers (such as global variables and function names) and local variables declared with the storage class specifier `static`. Global variables and function names default to storage class specifier `extern`. Global variables are created by placing variable declarations outside any function definition. They retain their values throughout the execution of the program.

12 Q. Write a C program to check whether a number is prime or not.

A.
```
// C program to check if a
// number is prime
#include <math.h>
#include <stdio.h>

// Driver code
int main()
{
    int num;
    int check = 1;

    printf("Enter a number: \n");
    scanf("%d", &num);

    // Iterating from 2 to sqrt(num)
    for (int i = 2; i <= sqrt(num); i++) {
        // If the given number is divisible by
        // any number between 2 and n/2 then
        // the given number is not a prime number
        if (num % i == 0) {
            check = 0;
            break;
        }
    }

    if (num <= 1) {
        check = 0;
    }

    if (check == 1) {
        printf("%d is a prime number", num);
    }
    else {
        printf("%d is not a prime number", num);
    }

    return 0;
}
```

Output:

```
Enter a number: 8
8 is not a prime number
```

13 Q. Write a program to check whether a string is a palindrome or not.

A.

```
// C program to check whether a
// string is palindrome or not.
#include <stdio.h>
#include <string.h>

// Palindrome function to check
// whether a string is palindrome
// or not
void Palindrome(char s[])
{
    // Start will start from 0th index
    // and end will start from length-1
    int start = 0;
    int end = strlen(s) - 1;

    // Comparing characters until they
    // are same
    while (end > start) {
        if (s[start++] != s[end--]) {
            printf("%s is not a Palindrome \n", s);
            return;
        }
    }
    printf("%s is a Palindrome \n", s);
}

// Driver code
int main()
```

```
{
    Palindrome("abba");
    return 0;
}
```

Output:

```
abba is a Palindrome
```

14 Q. Explain any three drawbacks of procedure-oriented languages. Give an example each in C language to highlight the drawback.

A. Previously, programs were written in a procedural fashion, i.e., the statements were written in the form of a batch. But as the requirements grew, it was seen that the programs were getting larger and larger, and it became difficult to debug. So, functions were introduced to reduce the size of the programs and improve readability in them. Still, that was not enough. One of the major problems with the “Procedural Paradigm” was that data was treated as a stepson and functions were given more priority. Whereas it is the other way. In this procedure the original data could easily get corrupted, as it was accessible to all the functions, even to those which do not have any right to access them. Before OOP, the programmer was restricted to use the predefined data types such as integer, float and character. If any program required handling of the x-y coordinates of some point, then it is quite a headache for the programmer. Whereas, in OOP this can be handled very easily as the programmer can define his own data types and the corresponding functions.

15 Q. What is the use of `printf()` and `scanf()` functions in C programming language? Also, explain format specifiers.

A. `printf()` function is used to print the value which is passed as the parameter to it on the console screen.

Syntax:

```
print(“%X”,variable_of_X_type);
```

`scanf()` method, reads the values from the console as per the data type specified.

Syntax:

```
scanf(“%X”,&variable_of_X_type);
```

In C format specifiers are used to tell the compiler what type of data will be present in the variable during input using `scanf()` or output using `print()`.

`%c`: Character format specifier used to display and scan character.
`%d`, `%i`: Signed Integer format specifier used to print or scan an integer value.
`%f`, `%e`, or `%E`: Floating-point format specifiers are used for printing or scanning float values.
`%s`: This format specifier is used for string printing.
`%p`: This format specifier is used for address printing.

16 Q. What are file operations in C? Explain with an example how to read and write to a file.

A. File operations in C include `fopen()`, `fclose()`, `fprintf()`, `fscanf()`, and `fgets()`.

Example:

```
#include <stdio.h>

int main() {
    FILE *fp = fopen("file.txt", "w");
    fprintf(fp, "Hello, World!");
    fclose(fp);

    fp = fopen("file.txt", "r");
    char buffer[50];
    fgets(buffer, 50, fp);
    printf("%s\n", buffer);
    fclose(fp);
    return 0;
}
```

17 Q. Highlight the differences between structures and unions.

A. Difference Between Structure and Union in C

Parameter	**Structure**	**Union**
Keyword	A user can deploy the keyword `struct` to define a Structure.	A user can deploy the keyword `union` to define a Union.
Internal Implementation	The implementation of Structure in C occurs internally- because it contains separate memory locations allotted to every input member.	In the case of a Union, the memory allocation occurs for only one member with the largest size among all the input variables. It shares the same location among all these members/objects.
Accessing Members	A user can access individual members at a given time.	A user can access only one member at a given time.

Syntax	The Syntax of declaring a Structure in C is:
	`struct [structure name]`
	`{`
	`type element_1;`
	`type element_2;`
	`.`
	`.`
	`} variable_1, variable_2, …;`
		The Syntax of declaring a Union in C is:
		`union [union name]`
		`{`
		`type element_1;`
		`type element_2;`
		`.`
		`.`
		`}  variable_1,  variable_2, …;`

Size	A Structure does not have a shared location for all of its members. It makes the size of a Structure to be greater than or equal to the sum of the size of its data members.	A Union does not have a separate location for every member in it. It makes its size equal to the size of the largest member among all the data members.
Value Altering	Altering the values of a single member does not affect the other members of a Structure.	When you alter the values of a single member, it affects the values of other members.
Storage of Value	In the case of a Structure, there is a specific memory location for every input data member. Thus, it can store multiple values of the various members.	In the case of a Union, there is an allocation of only one shared memory for all the input data members. Thus, it stores one value at a time for all of its members.

Initialization	In the case of a Structure, a user can initialize multiple members at the same time.	In the case of a Union, a user can only initiate the first member at a time.

18 Q. Explain the difference between **break** and **continue** statements with the help (a) of suitable example(s).

A. **break** statement terminates the smallest enclosing loop. Below is the program to illustrate the same:

```
// C program to illustrate the
// break statement
#include <stdio.h>
int main()
{

    int i = 0, j = 0;

    // Iterate a loop over the
    // range [0, 5]
    for (int i = 0; i < 5; i++) {

        printf("i = %d, j = ", i);

        // Iterate a loop over the
        // range [0, 5]
        for (int j = 0; j < 5; j++) {

            // Break Statement
            if (j == 2)
                break;

            printf("%d ", j);
        }

        printf("\n");
    }

    return 0;
}
```

Output:

```
i = 0, j = 0 1
i = 1, j = 0 1
i = 2, j = 0 1
i = 3, j = 0 1
i = 4, j = 0 1
```

In the above program the inner for loop always ends when the value of the variable `j` becomes **2**.

continue statement skips the rest of the loop statement and starts the next iteration of the loop to take place. Below is the program to illustrate the same:

```
// C program to illustrate the
// continue statement
#include <stdio.h>
int main()
{
    int i = 0, j = 0;
    // Iterate a loop over the
    // range [0, 5]
    for (int i = 0; i < 5; i++) {

        printf("i = %d, j = ", i);

        // Iterate a loop over the
        // range [0, 5]
        for (int j = 0; j < 5; j++) {

            // Continue Statement
            if (j == 2)
                continue;

            printf("%d ", j);
        }

        printf("\n");
    }

    return 0;
}
```

Output:

```
i = 0, j = 0 1 3 4
i = 1, j = 0 1 3 4
i = 2, j = 0 1 3 4
i = 3, j = 0 1 3 4
i = 4, j = 0 1 3 4
```

In the above program, the inner `for` loop always skip the iteration when the value of the variable `j` becomes **2**.

Tabular Difference Between the `break` and `continue` statement:

`break` **Statement**	`continue` **Statement**
The `break` statement is used to exit from the loop constructs.	The `continue` statement is not used to exit from the loop constructs.
The `break` statement is usually used with the `switch` statement, and it can also use it within the `while` loop, `do`-`while` loop, or the `for`-loop.	The `continue` statement is not used with the `switch` statement, but it can be used within the `while` loop, `do`-`while` loop, or `for`-loop.
When a `break` statement is encountered, then the control is exited from the loop construct immediately.	When the `continue` statement is encountered, then the control automatically passed from the beginning of the loop statement.
Syntax: `break;`	**Syntax:** `continue;`
`break` statements uses `switch` and label statements.	It does not use `switch` and label statements.

Leftover iterations are not executed after the `break` statement.	Leftover iterations can be executed even if the `continue` keyword appears in a loop.

19 Q. What is a `static` variable in C? How does it differ from a regular variable?

A. A `static` **variable** in C retains its value between function calls. It is initialized only once and its scope is limited to the block in which it is defined.

Example:
```
#include <stdio.h>

void increment() {
    static int count = 0;
    count++;
    printf("Count: %d\n", count);
}

int main() {
    increment();
    increment();
    increment();
    return 0;
}
```

Output:
```
Count: 1
Count: 2
Count: 3
```

Difference:
- A normal variable gets re-initialized every time a function is called.
- A `static` variable retains its value between function calls.

20 Q. Explain the difference between call by value and call by reference in C with examples.

A. In C, there are two ways to pass arguments to a function:

Call by Value: The function receives a copy of the actual value.
```
#include <stdio.h>

void modify(int x) {
    x = 20;
}

int main() {
    int a = 10;
    modify(a);
    printf("a: %d\n", a); // Output: a: 10
    return 0;
}
```

Output:
```
a: 10
```

Call by Reference: The function receives the address of the variable, allowing it to modify the original value.

```
#include <stdio.h>

void modify(int *x) {
```

```c
    *x = 20;
}

int main() {
    int a = 10;
    modify(&a);
    printf("a: %d\n", a); // Output: a: 20
    return 0;
}
```

Output:

```
a: 20
```