Q1). Define algorithm. Write an algorithm to perform addition of two natural numbers.

Answer

Algorithms

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Algorithm to perform addition of two natural

Step 1. Start
Step 2. Take a variable A.
Step 3. Take a variable B.
Step 4. Take a variable C.
Step 5. Put a value in A.
Step 6. Put a value in B.
Step 7. Add A and B.
Step 8. Store the result in C.
Step 9. Display C
Step 10. Stop.

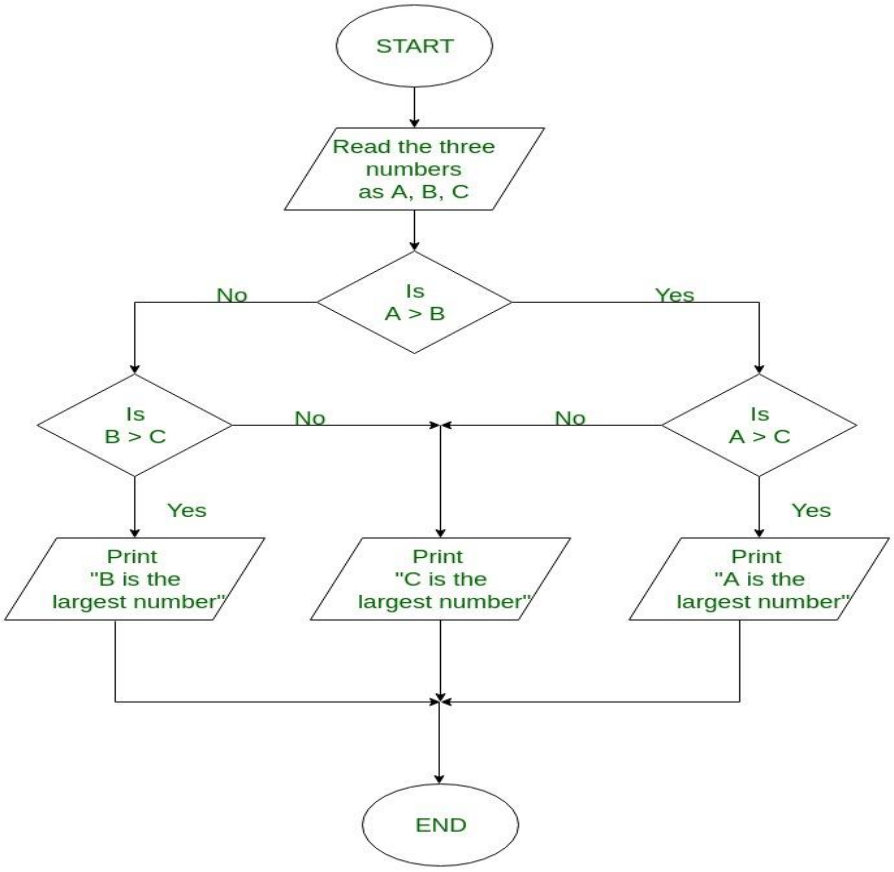
Q2). Define flow chart. Draw a flow chart to find the greatest among three numbers.

Answer

Flowchart

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

Flowchart to find the greatest among three numbers



Q3). Different between Flowchart and Algorithm with example.

Answer

SNo	Flowchart	Algorithm
1.	Block by block information diagram representing the data flow.	Step by step instruction representing the process of any solution.
2.	It is a pictorial representation of a process.	It is step wise analysis of the work to be done.
3.	Solution is shown in graphical format.	Solution is shown in non-computer language like English.
4.	Easy to understand as compared to algorithm.	It is somewhat difficult to understand.
5.	Easy to show branching and looping.	Difficult to show branching and looping
6.	Flowchart for big problem is impractical	Algorithm can be written for any problem
7.	Difficult to debug errors.	Easy to debug errors.
8.	It is easy to make flowchart.	It is difficult to write algorithm as compared to flowchart.

Q4). Differentiate between compiler and interpreter. Explain.

Answer

S.No.	Interpreter	Compiler
1)	Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
2)	Interpreters usually take less amount of time to analyze the source code. However, the overall execution time is comparatively slower than compilers.	Compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters.
3)	No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
4)	Programming languages like JavaScript, Python, and Ruby use interpreters.	Programming languages like C, C++, Java use compilers.

Q5). Discuss various data types available in 'C' language.

Answer

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These data types have different storage capacities.

C language supports 2 different types of data types:

1. **Primary data types**:

   These are fundamental data types in C namely integer(int), floating point(float), character(char) and void.

2. **Derived data types**:

   Derived data types are nothing but primary datatypes but a little twisted or grouped together

   like **array**, **stucture**, **union** and **pointer**. These are discussed in details later.

Data type determines the type of data a variable will hold. If a variable x is declared as int. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

Integer type

Integers are used to store whole numbers.

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating point type

Floating types are used to store real numbers.

Type	Size(bytes)	Range
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character type

Character types are used to store characters value.

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

void type

void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

Q6). Write a program to swap two integers

Answer

```
#include<stdio.h>
 int main()
{
    int a=10, b=20;
    printf("Before swap a=%d b=%d",a,b);
    a=a+b;
    b=a-b;
    a=a-b;
    printf("\nAfter swap a=%d b=%d",a,b);
    return 0;
}
```

Q7). Define storage classes. Also explain with suitable examples.

OR

Define storage class. Write name of storage classes used in C. Explain in brief register storage class.

Answer

Storage class

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program –

- auto
- register
- static
- extern

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
   int mount;
   auto int month;
}
```

The example above defines two variables within the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. . It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

```
{
   register int  miles;
```

RSRANA

```
}
```

The static Storage Class

The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.

The extern Storage Class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

Q8). Define local, global and static variable.

Answer

Local Variables

A **local variable** is one that occurs within a specific scope. They exist only in the function where they are created.

They are sometimes called ***automatic variables*** because they are automatically created when the function starts execution, and automatically go away when the function is finished executing.

The keyword **auto** can be used to explicitly create these variables, but isn't necessary since auto is the default.

Global Variables

A **global variable** is a variable that is defined outside all functions and available to all functions.

These variables are unaffected by scopes and are always available, which means that a global variable exists until the program ends.

Static Variables

A **static variable** can be either a global or local variable. Both are created by preceding the variable declaration with the keyword **static**.

A **local static variable** is a variable that can maintain its value from one function call to another and it will exist until the program ends.

When a local static variable is created, it should be assigned an initial value. If it's not, the value will default to 0.

A **global static variable** is one that can only be accessed in the file where it is created. This variable is said to have **file scope**.

Q9). Define operator in 'C' language with example.

Answer

Operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

PRANA

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Show Examples

RSANA

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Assignment Operators

The following table lists the assignment operators supported by the C language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators - sizeof & ternary

Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :** supported by the C Language.

Show Examples

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 2.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true? then value X : otherwise value Y

Q10). Define operator? Explain unary operators ++ and – with example.

Answer

Increment (++) - It is used to increment the value of the variable by 1. The increment can be done in two ways:

1. **prefix increment**
   In this method, the operator precedes the operand (e.g., ++a). The value of operand will be altered *before* it is used.

```
int a = 1;
int b = ++a;  // b = 2
```

2. **postfix increment**
   In this method, the operator follows the operand (e.g., a++). The value operand will be altered *after* it is used.

```
int a = 1;
int b = a++;   // b = 1
int c = a;     // c = 2
```

decrement (--) - It is used to decrement the value of the variable by 1. The increment can be done in two ways:

1. **prefix decrement**
   In this method, the operator preceeds the operand (e.g., – -a). The value of operand will be altered before it is used.

```
int a = 1;
int b = --a;  // b = 0
```

2. **posfix decrement**

   In this method, the operator follows the operand (e.g., a- -). The value of operand will be altered after it is used.

```
int a = 1;
int b = a--;   // b = 1
int c = a;     // c = 0
```

RSRANA

Q11). Define Decision making statements. Write advantages and syntax of IF-Else.

Answer

Decision making in C

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C language handles decision-making by supporting the following statements,

- if statement
- switch statement
- conditional operator statement (? : operator)
- goto statement

Decision making with if statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are,

1. Simple if statement
2. if....else statement
3. Nested if....else statement
4. Using else if statement

Simple if statement

The general form of a simple if statement is,

```
if(expression)
{
    statement inside;
}
    statement outside;
```

If the expression returns true, then the statement-inside will be executed, otherwise statement-inside is skipped and only the statement-outside is executed.

if...else statement

The general form of a simple if...else statement is,

```
if(expression)
{
    statement block1;
}
else
{
    statement block2;
}
```

If the expression is true, the statement-block1 is executed, else statement-block1 is skipped and statement-block2 is executed.

Nested if....else statement

The general form of a nested if...else statement is,

```
if( expression )
{
    if( expression1 )
    {
        statement block1;
    }
    else
    {
        statement block2;
    }
}
else
```

```
{
    statement block3;
}
```

if expression is false then statement-block3 will be executed, otherwise the execution continues and enters inside the first if to perform the check for the next if block, where if expression 1 is true the statement-block1 is executed otherwise statement-block2 is executed.

else if ladder

The general form of else-if ladder is,

```
if(expression1)
{
    statement block1;
}
else if(expression2)
{
    statement block2;
}
else if(expression3 )
{
    statement block3;
}
else
    default statement;
```

The expression is tested from the top(of the ladder) downwards. As soon as a true condition is found, the statement associated with it is executed.

RSRAM

Q12). Difference between break statement and continue statement

Answer

Key	Break	Continue
Functionality	Break statement mainly used to terminate the enclosing loop such as while, do-while, for or switch statement wherever break is declared.	Continue statement mainly skip the rest of loop wherever continue is declared and execute the next iteration.
Executional flow	Break statement resumes the control of the program to the end of loop and made executional flow outside that loop.	Continue statement resumes the control of the program to the next iteration of that loop enclosing 'continue' and made executional flow inside the loop again.
Usage	As mentioned break is used for the termination of enclosing loop.	On other hand continue causes early execution of the next iteration of the enclosing loop.
Compatibility	Break statement can be used and compatible with 'switch', 'label'.	We can't use continue statement with 'switch','lablel' as

Q13). Write a program to find sum of a four digit positive integer number

Answer

```
#include<stdio.h>
void main()
{
        int a, b, c, d,sum;
        printf("enter four integer values : ");
        scanf("%d%d%d%d",&a, &b, &c, &d);

        if(a<0 || b<0 || c<0 || d<0)
        {
                printf("Atleast one number is negative.");
        }else
        {
                sum=a+b+c+d;
                printf("Sum is : %d",sum);
        }
}
```

PRANA

Q14). Explain and differentiate between 'while' and 'do while' statement in 'C' language with example.

Answer

WHILE	DO-WHILE
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);

Q15). Define control statements? Explain in brief 'for loop' with suitable example.

Answer

A **control statement** is a statement that determines whether other statements will be executed.

- An if statement decides whether to execute another statement, or decides which of two statements to execute.
- A loop decides how many times to execute another statement. There are three kinds of loops:
  - while loops test whether a condition is true before executing the controlled statement.
  - do-while loops test whether a condition is true after executing the controlled statement.
  - for loops are (typically) used to execute the controlled statement a given number of times.
- A switch statement decides which of several statements to execute.

C For loop

This is one of the most frequently used loop in C programming.

Syntax of for loop:

```
for (initialization; condition test; increment or decrement)
{
      //Statements to be executed repeatedly
}
```

Example of for loop

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

Q16). Write a program to print tables from 1 to 10 using do while loop?

Answer

```
#include<stdio.h>
Void main()
{
        int loop1, loop2, mul;
        loop1=1;
        do
        {
                loop2=1;
                do
                {
                        mul=loop1*loop2;
                        Printf("%d",mul);
                }while(loop2<=10);
        }while(loop1<=10);
}
```

Q17). Write a program to print factorial of a given number using for loop.

OR

Write a program in C to find value of following:

X*(X-1)*(X-2)*…..*2*1

Answer

```
#include<stdio.h>
Void main()
{
        int fac, loop, num;
        fac=1;
        printf("enter a number : ");
        scanf("%d",&num);
        for(loop=1;loop<=num; loop++)
        {
                fac=fac*loop;
        }
        Printf("Factorial is : %d",fac);
}
```

RSRANA

Q18). Define Array? Write its type with syntax.

OR

Define Array? Write its types? Write syntax for multidimensional Array.

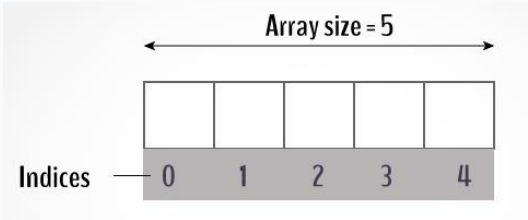
OR

Define array. Write its type with an example. Write declaration statement for 2D and 3D array.

Answer

C Arrays

An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.




```
int data[100];
```

How to declare an array?

```
dataType arrayName[arraySize];
```

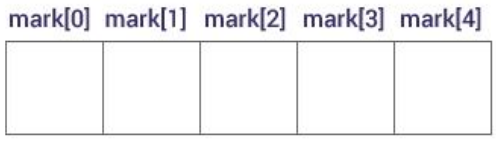
For example,

```
float mark[5];
```

- Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.
- It's important to note that the size and type of an array cannot be changed once it is declared.

Access Array Elements

You can access elements of an array by indices.
Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



How to initialize an array?

It is possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

You can also initialize an array like this.

```
int mark[] = {19, 10, 8, 17, 9};
```

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

Input and Output Array Elements

Here's how you can take input from the user and store it in an array element.

```
// take input and store it in the 3rd element
scanf("%d", &mark[2]);

// take input and store it in the ith element
scanf("%d", &mark[i-1]);
```

C Multidimensional Arrays

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example,

```
float x[3][4];
```

Here, **x** is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example,

```
float y[2][4][3];
```

Here, the array **y** can hold 24 elements.

Initializing a multidimensional array

Here is how you can initialize two-dimensional and three-dimensional arrays:

Initialization of a 2d array

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

Initialization of a 3d array

You can initialize a three-dimensional array in a similar way like a two-dimensional array. Here's an example,

```
int test[2][3][4] = {
    {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
    {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Q19). Define function. Write its types and calling methods with a suitable example.

Answer

Function

A function is a group of statements that together perform a task. A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Defining a Function

The general form of a function definition in C programming language is as follows –

```
return_type function_name( parameter list ) {
   body of the function
}
```

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function –

- **Return Type** – A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as **actual parameter or argument**. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** – The function body contains a collection of statements that define what the function does.

Types of function

There are two types of function in C programming:

- Standard library functions
- User-defined functions

Standard library functions

The standard library functions are built-in functions in C programming. These functions are defined in header files. For example,
The `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the `stdio.h` header file.

Hence, to use the `printf()` function, we need to include the `stdio.h` header file using `#include <stdio.h>`.

User-defined function

You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

How user-defined function works?

```
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

The execution of a C program begins from the main() function.
When the compiler encounters functionName();, control of the program jumps to

```
void functionName()
```

And, the compiler starts executing the codes inside `functionName()`.

Q20). What do you mean by call by value and call by reference. Explain with example.

Answer

Difference between Call by Value and Call by Reference

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called *actual parameters* whereas the parameters received by function are called *formal parameters*.

CALL BY VALUE	CALL BY REFERENCE
While calling a function, we pass values of variables to it. Such functions are known as "Call By Values".	While calling a function, instead of passing the values of variables, we pass address of variables (location of variables) to the function known as "Call By References.
In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.

// C program to illustrate call by value
 	
#include<stdio.h>
 	
void swapx(int x, int y);
 	
// Main function
int main()
{
int a = 10, b = 20;
 	
// Pass by Values
swapx(a, b);
 	
printf("a=%d b=%d\n", a, b);
 	
return 0;
}
 	
// Swap functions that swaps
// two values
void swapx(int x, int y)
{
int t;
 	
t = x;
x = y;
y = t;
 	
printf("x=%d y=%d\n", x, y);
}
	// C program to illustrate Call by Reference
	
	#include<stdio.h>
	
	void swapx(int*, int*);
	
	// Main function
	int main()
	{
	int a = 10, b = 20;
	
	// Pass reference
	swapx(&a, &b);
	
	printf("a=%d b=%d\n", a, b);
	
	return 0;
	}
	
	// Function to swap two variables
	// by references
	void swapx(int* x, int* y)
	{
	int t;
	
	t = *x;
	*x = *y;
	*y = t;
	
	printf("x=%d y=%d\n", *x,
	*y);
	}

Q21). Write advantages of call by reference over call by value.

Answer

- The function can change the value of the argument, which is quite useful.
- It does not create duplicate data for holding only one value which helps you to save memory space.
- In this method, there is no copy of the argument made. Therefore it is processed very fast.
- Helps you to avoid changes done by mistake
- A person reading the code never knows that the value can be modified in the function.

Q22). Define recursive function. Write a program in 'C' to sum the given number using recursive function.

Answer

Recursion

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
   recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

program in 'C' to sum the given number using recursive

```
#include <stdio.h>
/* Function declaration */

int sumOfDigits(int num);

int main()
{
    int num, sum;

    printf("Enter any number to find sum of digits: ");
    scanf("%d", &num);

    sum = sumOfDigits(num);

    printf("Sum of digits of %d = %d", num, sum);

    return 0;
}

int sumOfDigits(int num)
{
    // Base condition
    if(num == 0)
        return 0;

    return ((num % 10) + sumOfDigits(num / 10));
 }
```

PSRANA

Q23). Write a program in C to add elements of an array using recursive function.

Answer

```
#include <stdio.h>
#define MAX_SIZE 100

int sum(int arr[], int start, int len);


int main()
{
    int arr[MAX_SIZE];
    int N, i, sumofarray;


    printf("Enter size of the array: ");
    scanf("%d", &N);
    printf("Enter elements in the array: ");
    for(i=0; i<N; i++)
    {
        scanf("%d", &arr[i]);
    }


    sumofarray = sum(arr, 0, N);
    printf("Sum of array elements: %d", sumofarray);

    return 0;
}

int sum(int arr[], int start, int len)
{
    /* Recursion base condition */
    if(start >= len)
        return 0;

    return (arr[start] + sum(arr, start + 1, len));
}
```

Q24). Write a program to find sum of first 'n' natural numbers using recursive function?

Answer

```
#include <stdio.h>
int addNumbers(int n);
int main() {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    printf("Sum = %d", addNumbers(num));
    return 0;
}

int addNumbers(int n) {
    if (n != 0)
        return n + addNumbers(n - 1);
```

BSANA

```
    else
        return n;
}
```

Q25). A function is defined as :

```
G(n)=1+G(n-1)+G(n-2)
G(0)=0 and G(1)=1
Write a function in C to find the value of G(n).
```

Answer

```
int GFun(int a)
{
      If(a==0)
      {
            return(0);
      }else if(a==1)
      {
            return(1);
      }else
      {
            return(1+GFun(a-1)+GFun(a-2));
      }
}
```

Q26). Define library function. Write prototype of the following functions.

Answer

a. **printf()** - The printf function is not part of the C language, because there is no input or output defined in C language itself. The printf function is just a useful function from the standard library of functions that are accessible by C programs. The behavior of printf is defined in the ANSI standard. If the compiler that you're using conforms to this standard then all the features and properties should be available to you.

Syntax of printf()

```
int printf (const char* c-string, ...);
```

b. **getch()** - getch() is a standard input library function in C language.getch() -known as get character- is a function get an input from the keyboard. Upon executing the getch() function the program control will wait until a character is inputted by the user from keyboard.

Syntax of getch()

```
int getch(void);
```

c. **malloc()** - The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be casted into pointers of any form.

Syntax of malloc()

```
ptr = (castType*) malloc(size);
```

d. **clrscr() -** It is a predefined function in "conio.h" (console input output header file) used to clear the console screen. It is a predefined function, by using this function we can clear the data from console (Monitor). Using of **clrscr() function in C** is always optional but it should be place after variable or function declaration only.

Syntax of clrscr()

```
clrscr();
```

e. **gets() -** The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Syntax of gets()

```
gets(<variable name>);
```

f. **getche() -** getche() function is a function in C programming language which waits for any character input from keyboard and it will also echo the input character on to the output screen.

Syntax of getche()

```
character_variable = getche();
```

Q27). Write a program to find sum of elements of an array using function.

Answer

```
#include <conio.h>

 int sumofarray(int a[],int n)
 {
     int i,sum=0;
    for(i=0; i<n; i++)
    {
        sum+=a[i];
    }
     return sum;
 }
int main()
{
    int a[1000],i,n,sum;

    printf("Enter size of the array : ");
    scanf("%d", &n);

    printf("Enter elements in array : ");
    for(i=0; i<n; i++)
    {
        scanf("%d",&a[i]);
    }
    sum=sumofarray(a,n);
    printf("sum of array is :%d",sum);

}
```

Q28). Explain structure with suitable example.

OR

Q29). Define structure. Write a program to access elements of structure.

OR

Q30). Define structure. Write syntax for structure.

Answer

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure –

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program –

```
#include <stdio.h>
#include <string.h>

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {
   struct Books Book1;

   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   printf( "Book 1 title : %s\n", Book1.title);
```

RISHANA

```
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

 return 0;
}
```

Q31). Define Union.

Answer

Defining a Union

To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows –

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

The union tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data having three members i, f, and str –

```
union Data {
   int i;
   float f;
   char str[20];
} data;
```

Now, a variable of Data type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

Q32). Define pointer with a program in 'C' language.

Answer

Pointer

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.
Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
int* p = &n;
```

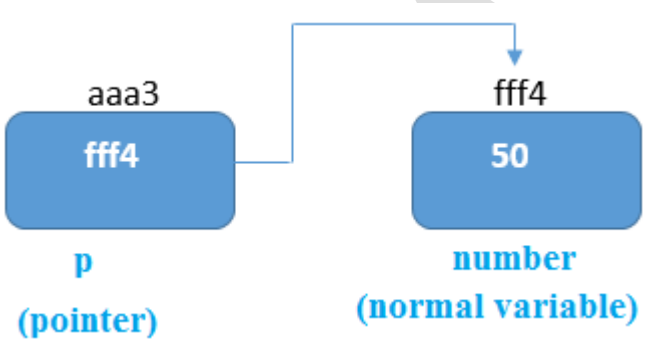
Declaring a pointer

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

int *a;
char *c;

Pointer Example

An example of using pointers to print the address and value is given below.



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

```
#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;
printf("Address of p variable is %x \n",p);
printf("Value of p variable is %d \n",*p);
return 0;
}
```

Q33). Write advantages and disadvantages of pointer?

Answer

Advantage (use) of pointers in c:

- Pointers provide direct access to memory
- Pointers provide a way to return more than one value to the functions
- Reduces the storage space and complexity of the program
- Reduces the execution time of the program
- Provides an alternate way to access array elements

- Pointers can be used to pass information back and forth between the calling function and called function.
- Pointers allows us to perform dynamic memory allocation and deallocation.
- Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
- Pointers allows us to resize the dynamically allocated memory block.
- Addresses of objects can be extracted using pointers

Disadvantage of pointers in c:

- Uninitialized pointers might cause segmentation fault.
- Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
- Pointers are slower than normal variables.
- If pointers are updated with incorrect values, it might lead to memory corruption.

Q34). Write a function in C that performs swap operations of two integers using pointer.

Answer

```
#include <stdio.h>

void swap(int *x,int *y)
{
    int t;
     t   = *x;
    *x   = *y;
    *y   =  t;
}

int main()
{
    int num1,num2;

    printf("Enter value of num1: ");
    scanf("%d",&num1);
    printf("Enter value of num2: ");
    scanf("%d",&num2);

    printf("Before Swapping: num1 is: %d, num2 is: %d\n",num1,num2);

    swap(&num1,&num2);

    printf("After  Swapping: num1 is: %d, num2 is: %d\n",num1,num2);

    return 0;
}
```

Q35). Define header file. Write name of any five header files used in C.

Answer

Header files in C

C language has numerous libraries that include predefined functions to make programming easier. In C language, header files contain the set of predefined standard library functions. Your request to use a header file in your program by including it with the

C preprocessing directive **"#include"**. All the header file have a **'.h'** an extension. By including a header file, we can use its contents in our program.
C++ also offers its users a variety of functions, one of which is included in header files. In C++, all the header files may or may not end with the **".h"** extension but in C, all the header files must necessarily end with the **".h"** extension.

A header file contains:
1. Function definitions
2. Data type definitions
3. Macros

Five Header Files used in C

1. **#include<stdio.h> (Standard input-output header)**

   Used to perform input and output operations in C like scanf() and printf().

2. **#include<string.h> (String header)**

   Perform string manipulation operations like strlen and strcpy.

3. **#include<conio.h> (Console input-output header)**

   Perform console input and console output operations like clrscr() to clear the screen and getch() to get the character from the keyboard.

4. **#include<math.h> (Math header)**

   Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.

5. **#include<time.h>(Time header)**

   Perform functions related to date and time like setdate() and getdate(). To modify the system date and get the CPU time respectively.

Q36). Write a program in C to read contents of text file using fread() function?

Answer

```
#include<stdio.h>

int main() {
  char buffer[20];
  FILE * stream;
  stream = fopen("file.txt", "r");
  int count = fread(&buffer, sizeof(char), 20, stream);
  fclose(stream);
  printf("Data read from file: %s \n", buffer);
  printf("Elements read: %d", count);
  return 0;
}
```