

ASSINGMENT OF  
C PROGRAM OR INTRODUCTION TO C  
PROGRAMMING



**Submitted by –**

Siddhant Kala

2316912025

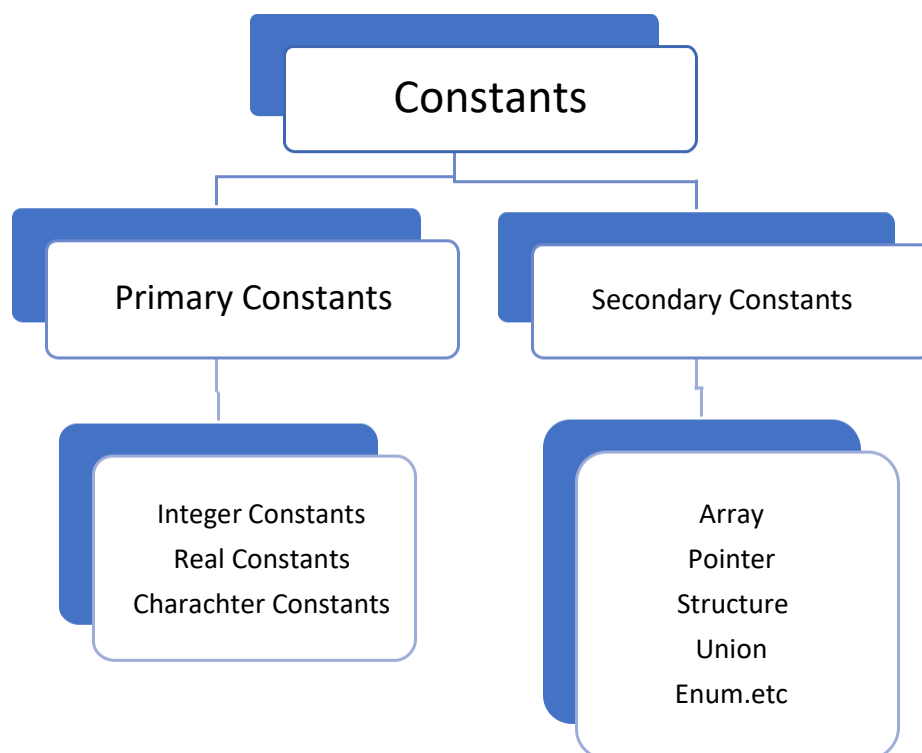
**Submitted to –**

Mr. Rishi  
Kumar

Asst. professor  
GEU

## Q1 What are Constants and variables, Types of Constants, Keywords, Rules for identifiers, int, float, char, double, long, void.

- **Constant** – A constant is an entity that doesn't change, whereas, a variable is an entity that may change. A keyword is a word that carries special meaning.
- **Types of C Constants**  
C constants can be divided into two major categories:
  - (a) Primary Constants
  - (b) Secondary Constants



### **Rules for Constructing Integer Constants**

- (a) An integer constant must have at least one digit.
- (b) It must not have a decimal point.
- (c) It can be either positive or negative.

(d) If no sign precedes an integer constant, it is assumed to be positive.

(e) No commas or blanks are allowed within an integer constant.

(f) The allowable range for integer constants is -2147483648 to +2147483647.

### **Rules for Constructing Real Constants**

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Following rules must be observed while constructing real constants expressed in fractional form:

(a) A real constant must have at least one digit.

(b) It must have a decimal point.

(c) It could be either positive or negative.

(d) Default sign is positive.

(e) No commas or blanks are allowed within a real constant

### **Rules for Constructing Character Constants**

(a) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

(b) Both the inverted commas should point to the left. For example, 'A' is a valid character constant whereas 'A' is not.

## Keywords

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). There are only 32 keywords available in C. A detailed discussion of each of these keywords would be taken up in later chapters wherever their use is relevant.

**Q2. Explain with examples Arithmetic Operators, Increment and Decrement Operators, Relational Operators, Logical Operators, Bitwise Operators, Conditional Operators, Type Conversions, and expressions, Precedence, and associativity of operators.**

### **Arithmetic Operators**

These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus:

<b>Arithmetic Operators/Operation</b>	<b>Example</b>
+ (Addition)	A+B
– (Subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

Example-

```
#include <stdio.h>
#include <conio.h>
int main ()
{
int num1, num2, res;
float f1, f2, res1;

double d1, d2, res2;
```

```
printf (" Enter two integer numbers: ");
```

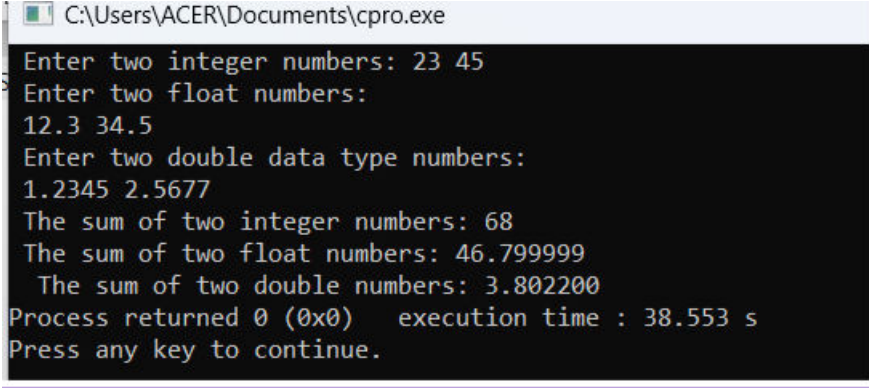
```
scanf ("%d %d", &num1, &num2); res  
= num1 + num2; // use + operator
```

```
printf (" Enter two float numbers: \n ");
```

```
scanf ("%f %f", &f1, &f2);  
res1 = f1 + f2; // use + operator
```

```
printf (" Enter two double data type numbers: \n  
"); scanf ("%lf %lf", &d1, &d2); res2 = d1 + d2; //  
use + operator
```

```
    printf (" The sum of two integer numbers: %d \n", res);  
printf (" The sum of two float numbers: %f \n ", res1);  
printf (" The sum of two double numbers: %lf", res2);  
return 0;  
}
```



```
C:\Users\ACER\Documents\cpro.exe  
Enter two integer numbers: 23 45  
Enter two float numbers:  
12.3 34.5  
Enter two double data type numbers:  
1.2345 2.5677  
The sum of two integer numbers: 68  
The sum of two float numbers: 46.799999  
The sum of two double numbers: 3.802200  
Process returned 0 (0x0)   execution time : 38.553 s  
Press any key to continue.
```

## Increment/Decrement Operators

Increment/decrement Operators in C: Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Increment operator: ++var\_name: pre-increment operator

var\_name++ : post-increment operator

Decrement operator: --var\_name: pre-decrement operator

var\_name--: post-decrement operator

Example of increment operator-

```
#include <stdio.h> intmain()
```

```
{  
inti=1;  
while(i<10)  
{  
printf("%d ",i);  
i++;  
}  
}
```

Example of Decrement

Operator #include <stdio.h>

```
intmain()
```

```
{  
inti=20;  
while(i>10)  
{  
printf("%d ",i);  
i--;  
}  
}
```

**Relational operator**



Relational operators are used to find the relation between two variables. *i.e.*

to compare the values of two variables in a C program.

<b>Operators</b>	<b>Example/Description</b>
>	$x > y$ (x is greater than y)
<	$x < y$ (x is less than y)
>=	$x \geq y$ (x is greater than or equal to y)
<=	$x \leq y$ (x is less than or equal to y)
= =	$x == y$ (x is equal to y)
!=	$x != y$ (x is not equal to y)

Example –

**Q3. Explain with Example conditional statements if, if-else, elseif, nested if else.**

### The if Statement

C uses the keyword if to implement the decision control instruction. The general form of if statement looks like this: if ( this condition is true ) execute this statement ;

The keyword if tells the compiler that what follows is a decision control instruction. The condition following the keyword if is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true, then the statement is not executed; instead the program skips past it. Example-

```
# include <stdio.h>
int main( )
{
int bonus, cy, yoj, yos ;
printf ( "Enter current year and year of joining " )
; scanf ( "%d %d", &cy, &yoy ) ;
yos = cy - yoy
; if ( yos > 3 )
{

bonus = 2500 ;
printf ( "Bonus = Rs. %d\n", bonus ) ;
}
return 0 ;
}
```

C:\Users\ACER\Documents\cgfh.exe

Enter current year and year of joining 2023 2020

Process returned 0 (0x0) execution time : 10.336 s  
Press any key to continue.

## The if-else Statement –

The if statement by itself will execute a single statement, or a group of statements, when the expression following if evaluates to true. It does nothing when the expression evaluates to false.

Example-

```
# include <stdio.h>
int main( )
{
float bs, gs, da, hra ;
printf ( "Enter basic salary " ) ;
scanf ( "%f", &bs ) ;
if ( bs < 1500 )
{
hra = bs * 10 / 100 ;
da = bs * 90 / 100 ;
}
else
{
hra = 500 ;
da = bs * 98 / 100 ;
}
gs = bs + hra + da ;
printf ( "gross salary = Rs. %f\n", gs ) ;
return 0 ;
}
```

```
C:\Users\ACER\Documents\cgfh.exe
Enter basic salary 78900
gross salary = Rs. 156722.000000

Process returned 0 (0x0)   execution time : 5.800 s
Press any key to continue.
```

## Elseif-

### The else if Statement

Use the else if statement to specify a new condition if the first condition

### Syntax

```
if (condition1) {
```

```
    // block of code to be executed if condition1 is
```

```
true } else if (condition2) {
```

```
    // block of code to be executed if the condition1 is false and
    condition2 is true
```

```
} else {
```

```
    // block of code to be executed if the condition1 is false and
    condition2 is false
```

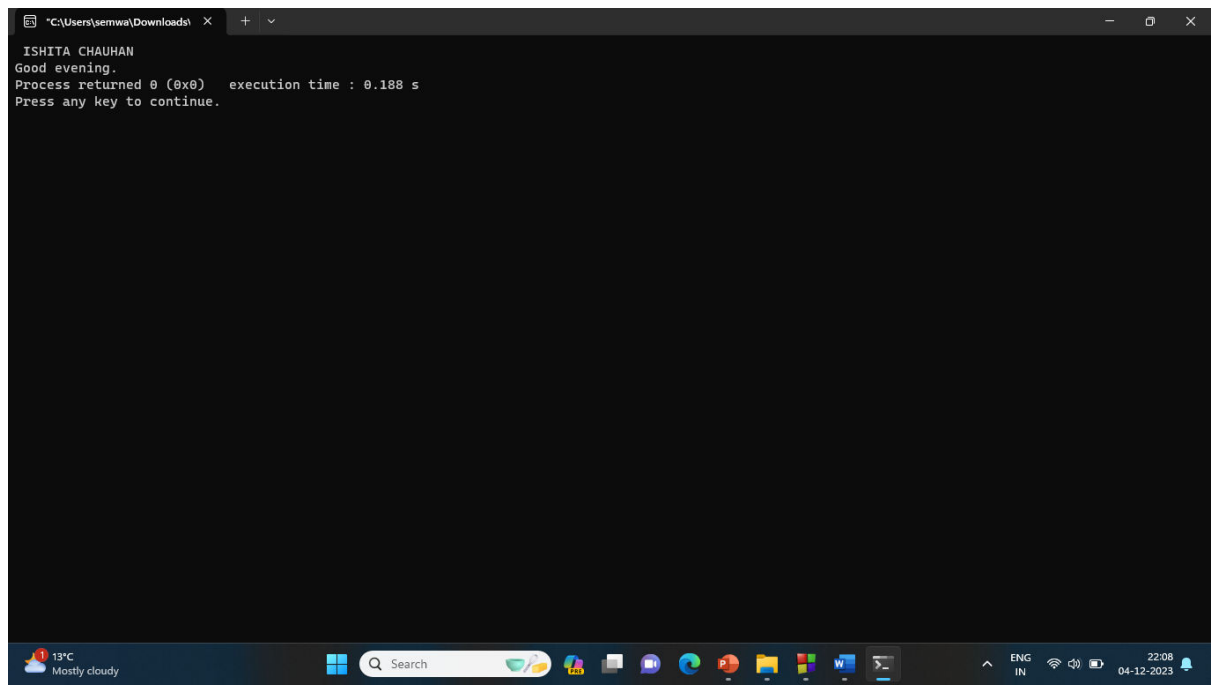
```
}
```

### Example –

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

```
int main ()
```

```
{  
    int time = 22;  
    printf(" ISHITA CHAUHAN \n");  
    if (time < 10) {  
        printf("Good morning.");  
    } else if (time < 20) {  
        printf("Good day.");  
    } else {  
        printf("Good evening.");  
    }  
}
```

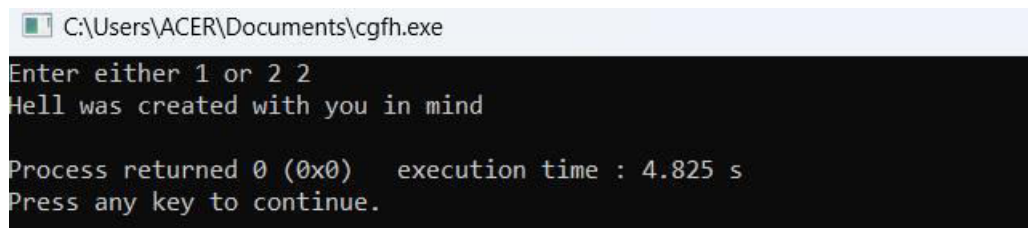


### **Nested if-elses –**

It is perfectly alright if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called 'nesting' of ifs.

Example-

```
#include <stdio.h>
int main( )
{
int i ;
printf ( "Enter either 1 or 2 " ) ;
scanf ( "%d", &i ) ;
if ( i == 1 )
printf ( "You would go to heaven !\n" )
; else
{
if ( i == 2 )
printf ( "Hell was created with you in mind\n" ) ;
else
printf ( "How about mother earth !\n" ) ;
}
return 0 ;
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\ACER\Documents\cgfh.exe". The prompt displays the output of the C program: "Enter either 1 or 2 2" followed by "Hell was created with you in mind". Below this, it shows "Process returned 0 (0x0) execution time : 4.825 s" and "Press any key to continue.".



#### **Q4. Explain Switch Case statement with example.**

The control statement that allows us to make a decision from the number of choices is called a switch, or more correctly a switch-case default, since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch ( integer expression )
{
case constant 1 :
do this ;
case constant 2 :
do this ;
case constant 3 :
do this ;
default :
do this ;
}
```

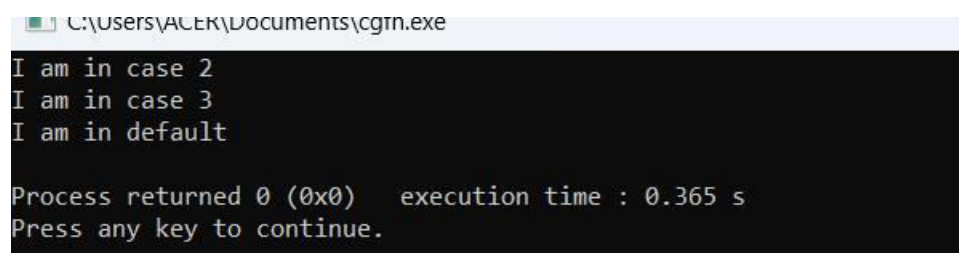
The integer expression following the keyword switch is any C expression that will yield an integer value. It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer. The keyword case is followed by an integer or a character constant. Each constant in each case must be different from all the others. The “do this” lines in the above form of switch represent any valid C statement.

What happens when we run a program containing a switch? First, the integer expression following the keyword switch is evaluated. The value it gives is then matched, one-by-one, against the constant values that follow the case statements. When a match is found, the program executes the statements following that case, and all subsequent case and default

statements as well. If no match is found with any of the case statements, only the statements following the default case are executed.

Example-

```
# include <stdio.h>
int main( )
{
int i = 2 ;
switch ( i )
{
case 1 :
printf ( "I am in case 1 \n" ) ;
case 2 :
printf ( "I am in case 2 \n" ) ;
case 3 :
printf ( "I am in case 3 \n" ) ;
default :
printf ( "I am in default \n" ) ;
}
return 0 ;
}
```



```
C:\Users\ACER\Documents\cgrn.exe
I am in case 2
I am in case 3
I am in default

Process returned 0 (0x0)   execution time : 0.365 s
Press any key to continue.
```

**Q5. Explain Loops, for loop, while loop, do while loop with examples.**

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control instruction. There are three methods by way of which we can repeat a part of a program.

They are:

- (a) Using a for statement
- (b) Using a while statement
- (c) Using a do-while statement

**The for Loop -**

Perhaps one reason why few programmers use while is that they are too busy using the for, which is probably the most popular looping instruction. The for allows us to specify three things about a loop in a single line:

- (a) Setting a loop counter to an initial value.
- (b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.
- (c) Increasing the value of loop counter each time the body of the loop has been executed.

The general form of for statement is as under:

```
for ( initialize counter ; test counter ; increment counter )  
{  
    do this ;  
    and this ;  
    and this ;  
}
```

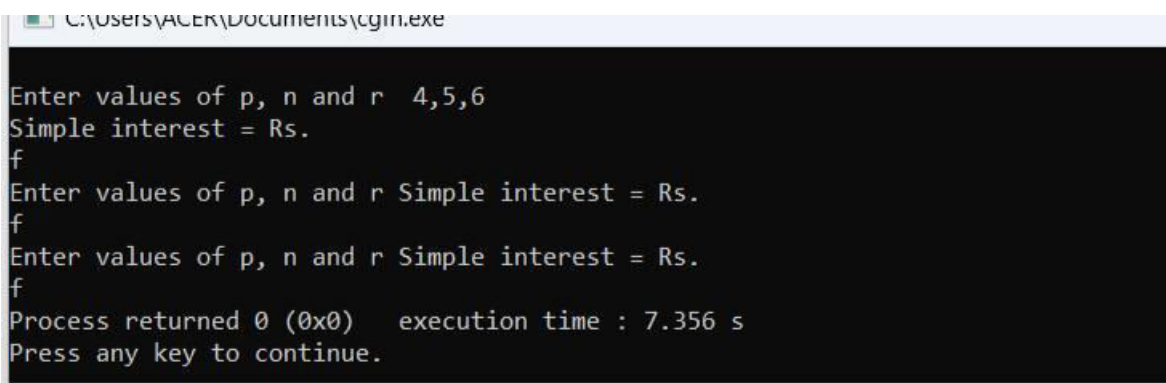
## The while Loop-

It is often the case in programming that you want to repeat something a fixed number of times. Perhaps you want to calculate gross salaries of ten different persons, or you want to convert temperatures from Centigrade to Fahrenheit for 15 different cities. The while loop is ideally suited for this.

Example-

```
# include <stdio.h>
int main( )
{
int p, n, count ;
float r, si ;
count = 1 ;
while ( count <= 3 )
{
printf ( "\nEnter values of p, n and r " )
; scanf ( "%d %d %f", &p, &n, &r ) ;
si = p * n * r / 100 ;
printf ( "Simple interest = Rs. %\nf", si ) ;
count = count + 1 ;

}
return 0 ;
}
```



```
C:\users\ACER\Documents\cgim.exe
Enter values of p, n and r 4,5,6
Simple interest = Rs.
f
Enter values of p, n and r Simple interest = Rs.
f
Enter values of p, n and r Simple interest = Rs.
f
Process returned 0 (0x0)   execution time : 7.356 s
Press any key to continue.
```

## The do-while Loop-

The do-while loop looks like this:

```
do
{
    this ;
    and this ;
    and this ;
    and this ;
}
while ( this condition is true ) ;
```

There is a minor difference between the working of while and do-while loops. This difference is the place where the condition is tested. The while tests the condition before executing any of the statements within the while loop. This means that do-while would execute its statements at least once, even if the condition fails for the first time. The while, on the other hand will not execute its statements if the condition fails for the first time. Example-

```
# include <stdio.h>
int main( )
{

do
{
printf ( "Hello there \n" ) ;
} while ( 4 < 1 ) ;
return 0 ;
}
```

C:\Users\ACER\Documents\cgfh.exe

Hello there

Process returned 0 (0x0) execution time : 0.169 s

Press any key to continue.

**Q6. Explain with examples debugging importance, tools common errors: syntax, logic, and runtime errors, debugging, and Testing C Programs.**

**Debugging-**

To launch an application into the market, it is very necessary to cross-check it multiple times so as to deliver an error-free product.

When we talk about delivering a bug-free product, then our main concern is all about customer satisfaction because if your application is not up to the mark, then eventually it will demolish the company's reputation in the market.

In this article, we are going to see what makes debugging stand out of the queue and how it is different from software testing.

**1. Syntax Error-**

Syntax errors occur when a programmer makes mistakes in typing the code's syntax correctly or makes typos. In other words, syntax errors occur when a programmer does not follow the set of rules defined for the syntax of C language.

Syntax errors are sometimes also called compilation errors because they are always detected by the compiler. Generally, these errors can be easily identified and rectified by programmers.

The most commonly occurring syntax errors in C language are:

- Missing semi-colon (;)
- Missing parenthesis ({} )
- Assigning value to a variable without declaring

it Example of syntax errors

```
#include <stdio.h>

void main() {
    var = 5;    // we did not declare the data type of variable
    printf("The variable is: %d", var);
}
```

## 2. Runtime Error

Errors that occur during the execution (or running) of a program are called RunTime Errors. These errors occur after the program has been compiled successfully. When a program is running, and it is not able to perform any particular operation, it means that we have encountered a run time error. For example, while a certain program is running, if it encounters the square root of -1 in the code, the program will not be able to generate an output because calculating the square root of -1 is not possible. Hence, the program will produce an error.

Runtime errors can be a little tricky to identify because the compiler can not detect these errors. They can only be identified once the program is running. Some of the most common run time errors are: number not divisible by zero, array index out of bounds, string index out of bounds, etc.



Runtime errors can occur because of various reasons. Some of the reasons are:

1. Mistakes in the Code: Let us say during the execution of a while loop, the programmer forgets to enter a break statement. This will lead the program to run infinite times, hence resulting in a run time error.
2. Memory Leaks: If a programmer creates an array in the heap but forgets to delete the array's data, the program might start leaking memory, resulting in a run time error.
3. Mathematically Incorrect Operations: Dividing a number by zero, or calculating the square root of -1 will also result in a run time error.
4. Undefined Variables: If a programmer forgets to define a variable in the code, the program will generate a run time error.

Example of runtime error-

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

```
#include <math.h>
```

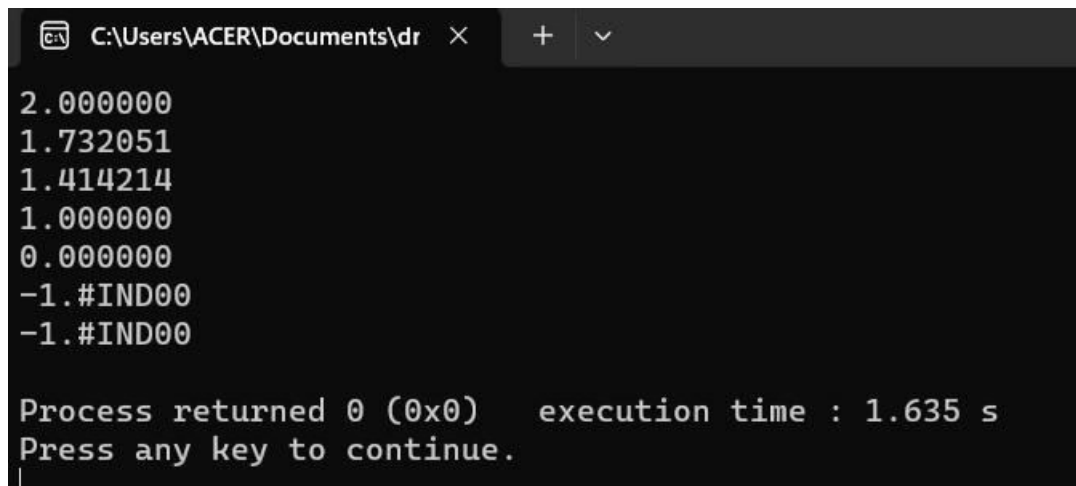
```
int main() {
```

```
    for (int i = 4; i >= -2; i--)    {
```

```
        printf("%f", sqrt(i));
```

```
        printf("\n");
```

```
}  
  
return 0;  
  
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\ACER\Documents\dr' and standard window controls. The command prompt displays the following output:

```
2.000000  
1.732051  
1.414214  
1.000000  
0.000000  
-1.#IND00  
-1.#IND00  
  
Process returned 0 (0x0)   execution time : 1.635 s  
Press any key to continue.
```

### 3. Logical Error

Sometimes, we do not get the output we expected after the compilation and execution of a program. Even though the code seems error free, the output generated is different from the expected one. These types of errors are called Logical Errors. Logical errors are those errors in which we think that our code is correct, the code compiles without any error and gives no error while it is running, but the output we get is different from the output we expected.

In 1999, NASA lost a spacecraft due to a logical error. This happened because of some miscalculations between the English and the American Units. The software was coded to work for one system but was used with the other.



C:\Users\ACER\Documents\dr



The output is: 1.#INF00

Process returned 23 (0x17)      execution time : 1.437 s

Press any key to continue.

**Q7. What is the user defined and pre-defined functions. Explain with example call by value and call by reference.**

A function is defined as a block of codes that are used to perform a specific task. These codes are defined a single time and can be used multiple times as per the need of the programmer. Types of functions in C language are mainly divided into two major types:

- 

1. **Predefined Functions** (also known as library functions)

- 

1. **User-defined Functions** (defined by the user)

This classification is done based on defining the task or function in a program. Each of them is discussed below using suitable examples wherever necessary.

### **Predefined Functions**

Predefined functions are those types of functions in C that are already included and defined in the system library. The code of these functions is stored in the system library and users can use them to write error-free code. The user must have knowledge of the working of these functions. For using this type of function, it is a must to include a header file where these functions are located.

Suppose we need to write a program where we will print a sentence. We need to use here `printf()` function which exists in `<stdio.h>` header file. If not done this way, we will get an error while executing the code. Because prototype data

definitions of these kinds of functions exist in their corresponding header file. Let us see an example for a better understanding.

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

```
#include <math.h>
```

```
int main()
```

```
{
```

```
    double num, root;
```

```
    printf(" Ishita Chauhan ");
```

```
    printf(" \n enter the number you want to find square  
root of: "); // print function is used to take input from user
```

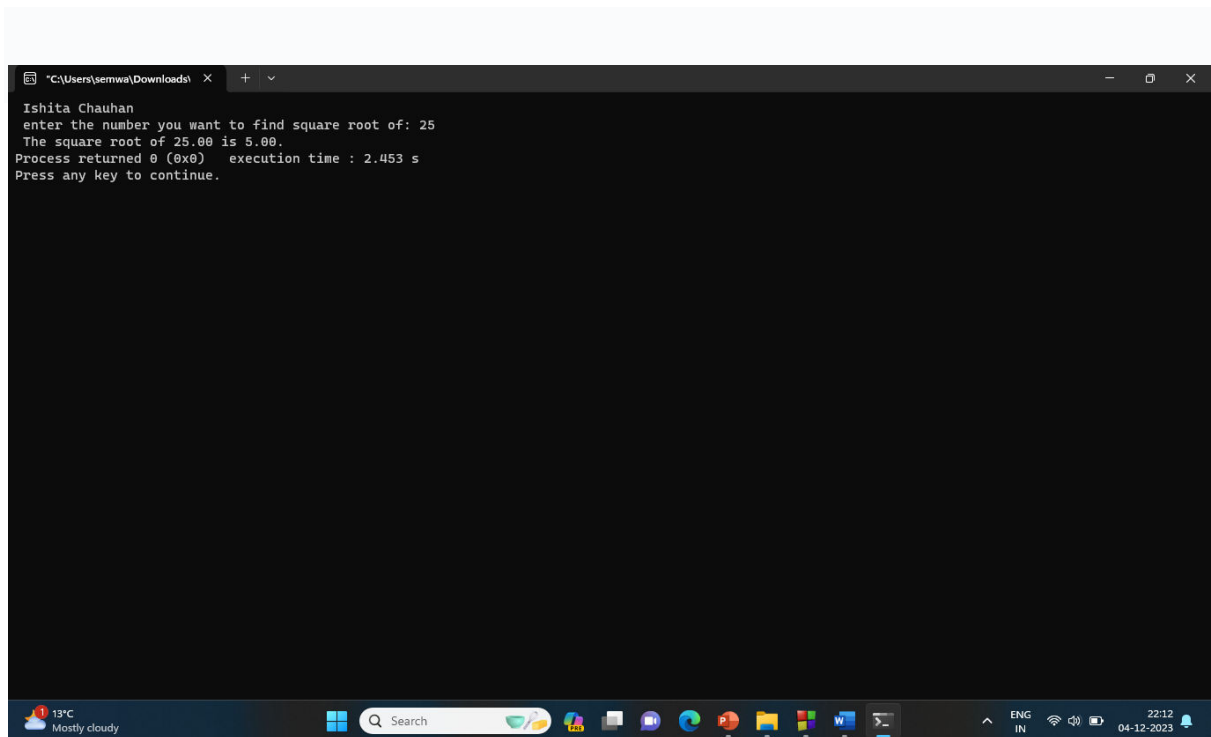
```
    scanf("%lf", &num);
```

```
    root = sqrt(num); // square root is done and stored in root
```

```
    printf(" The square root of %.2lf is %.2lf.", num, root);
```

```
    return 0;
```

```
}
```



## User-defined Functions

It is defined as that type of function in C which is defined by the user. The user writes the functionality of the function according to their need before using them in the program. Since the function is user-defined, there is no need to include a header file for that. Suppose we need to find the area and perimeter of a rectangle. Then, we will create two functions. One for finding area and one function for finding perimeter. Creating two functions will divide the task into two simpler tasks and make the whole task easy. This is called user-defined functions.

Let us see an example of a user-defined function which will add two integers. It will help us to understand 'types of functions in c' in a better way.

```
#include<stdio.h>

int main()
{
    int number_1,number_2,sum;
    printf(" Ishita Chauhan");
```

```
printf("\nEnter two numbers to add: ");  
scanf("%d %d",&number_1,&number_2);
```

```
sum = addNumbers(number_1, number_2);  
printf("sum of the given two numbers is = %d",sum);
```

```
return 0;
```

```
}
```

```
int addNumbers(int a, int b)
```

```
{
```

```
    int result;
```

```
    result = a+b;
```

```
    return result;
```

```
}
```



```
*C:\Users\semwa\Downloads\ x + v
Ishita Chauhan
Enter two numbers to add: 25
45
sum of the given two numbers is = 70
Process returned 0 (0x0) execution time : 2.930 s
Press any key to continue.
```

13°C Mostly cloudy Search 22:14 04-12-2023

**Q8. 1) Explain with Passing and returning arguments to and from Function. 2) Explain Storage classes, automatic, static, register, external. 3) Write a program for two strings S1 and S2. Develop a C Program for the following operations. a) Display a concatenated output of S1 and S2 b) Count the number of characters and empty spaces in S1 and S2.**

### **Types of Function According to Arguments and Return Value**

Functions can be differentiated into 4 types according to the arguments passed and value returns these are:

1. Function with arguments and return value
  2. Function with arguments and no return value
  3. Function with no arguments and with return value
  4. Function with no arguments and no return value
1. Function with arguments and return value

#### **Syntax:**

**Function declaration :** `int function ( int );`

**Function call :** `function( x );`

**Function definition:**

`int function( int x )`

```
{  
    statements;  
    return x;  
}
```

. Function with arguments but no return value

When a function has arguments, it receives any data from the calling function but it returns no values. These are void functions with no return values.

**Syntax:**

**Function declaration :** void function ( int );

**Function call :** function( x );

**Function definition:**

```
void function( int x )
{
    statements;
}
```

. Function with no argument and no return value

When a function has no arguments, it does not receive any data from the calling function. Similarly, when it does not return a value, the calling function does not receive any data from the called function.

**Syntax:**

**Function declaration :** void function();

**Function call :** function();

**Function definition :**

```
void function()
{
    statements;
}
```

4. Function with no arguments but returns a value There could be occasions where we may need to design functions that may not take any arguments but returns a value to the calling function. An example of this is getchar function which has no parameters but it returns an integer and integer-type data that represents a character.

**Syntax:**

**Function declaration :** int function();

**Function call :** function();

### **Function definition :**

```
int function()  
{  
    statements;  
    return x;  
}
```

**Storage classes in C** We use the storage class in the C language for determining the visibility, lifetime, initial value, and memory location of any given variable. The storage classes define the visibility (scope) and the lifetime of any function/ variable within a C program. These classes precede the type that they are going to modify.

### **Types of Storage Classes in C**

There are four different types of storage classes that we use in the C language:

- Automatic Storage Class
- External Storage Class
- Static Storage Class
- Register Storage Class

### **Automatic Storage Class**

It is also known as the auto storage class, and it acts as the default storage class for all the variables that are local in nature.

For example,

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

Look at the example that we used above- it defines two of the variables in the very same storage class. One can use 'auto' only within the functions- or the local variables.

Features of automatic variables:

- The allocation of memory for these variables occurs automatically during the runtime.
- The scope of an automatic variable is limited to that block in which we are defining them.
- The visibility of these variables is also limited to that block in which we are defining them.
- The initialization of these variables is, by default, a garbage value.
- The memory that is assigned to an automatic variable gets free when it exits from a block.
- We use the keyword auto to define the automatic variables.
- Any local variable that exists in the C language is, by default, automatic in nature.

## **External Storage Class**

It is also known as the extern storage class, and we use it for giving a reference of any global variable which is visible to all the files present in a program. When using the extern storage class, we cannot initialize the variable. However, note that it points to the name of the variable at any storage location that we have already defined (previously).

Whenever we have multiple numbers of files while we are trying to define any global variable or a function (that will be used in various other files too), then we will use the extern in another file for providing the reference of the defined function or variable. In simpler words, we use the extern for declaring a function or a global variable in another file.

The most common use of the extern modifier is when we have two or more than two files that share a similar global variable or function.

### **Static Storage Class**

This type of storage class gives an instruction to a compiler to keep the given local variable around during the program's lifetime- instead of creating it and then destroying it every time it comes into a scope and goes out of it. Thus, when we make a local variable static, it allows the variable to maintain the values that are available between various function calls.

We may also apply a static modifier to a global variable. When we do so, it will cause the scope of the variable to be restricted to that file in which its declaration happened.

In a C programming language, when we use the static on a global variable, it will cause all the objects present in a class to share a single copy of that given member.

## Register Storage Class

We use the register storage class for defining the local variables that must be stored in any register, and not in a RAM. It means that the maximum size of this variable is equal to that of the register size (it is usually one word). Also, we cannot apply the ‘&’ unary operator to it because it has no memory location.

```
#include <stdio.h>
int main() {
    char s1[100] = "programming ", s2[] = "is
    awesome"; int length, j;

    // store length of s1 in the length variable
    length = 0;
    while (s1[length] != '\0') {
        ++length;
    }

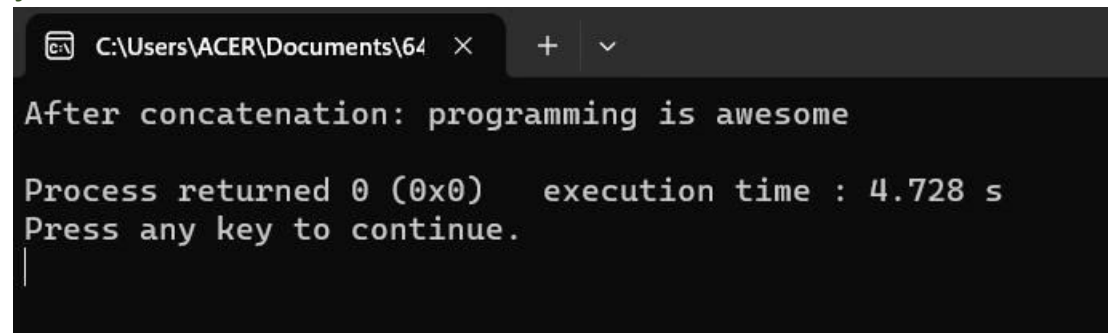
    // concatenate s2 to s1
    for (j = 0; s2[j] != '\0'; ++j, ++length) {
        s1[length] = s2[j];
    }

    // terminating the s1
    s1[length] = '\0';

    printf("After concatenation:
    "); puts(s1);
}
```



```
return 0;  
}
```



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\ACER\Documents\64' with a close button. The window content displays the output of a program: 'After concatenation: programming is awesome', followed by 'Process returned 0 (0x0) execution time : 4.728 s', and 'Press any key to continue.' with a cursor on the next line.

```
C:\Users\ACER\Documents\64 × + ▾  
After concatenation: programming is awesome  
Process returned 0 (0x0) execution time : 4.728 s  
Press any key to continue.  
|
```

**Q9. Explain with example 1D array and multidimensional array. Consider two matrices of the size m and n. Implement matrix multiplication operation and display results using functions. Write three functions 1) Read matrix elements 2) Matrix Multiplication 3) Print matrix elements.**

### **One dimensional array in C**

Arrays are a fundamental concept in programming, and they come in different dimensions. One-dimensional arrays, also known as single arrays, are arrays with only one dimension or a single row. In this article, we'll dive deep into one-dimensional arrays in C programming language, including their syntax, examples, and output.

#### **Syntax of One-Dimensional Array in C**

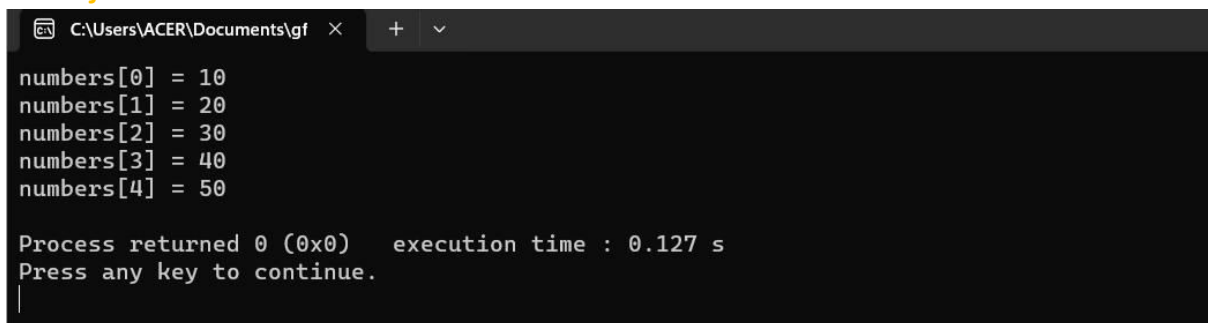
The syntax of a one-dimensional array in C programming language is as follows:

1. **dataType** **arrayName**[**arraySize**];
  - **dataType** specifies the data type of the array. It can be any valid data type in C programming language, such as int, float, char, double, etc.
  - **arrayName** is the name of the array, which is used to refer to the array in the program.
  - **arraySize** specifies the number of elements in the array. It must be a positive integer value.

#### **Example of One-Dimensional Array in C**

Let's take a simple example of a one-dimensional array in C programming language to understand its syntax and usage.

```
#include <stdio.h>
int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    for(int i=0; i<5; i++) {
        printf("numbers[%d] = %d\n", i, numbers[i]);
    }
    return 0;
}
```



```
C:\Users\ACER\Documents\gf x + v
numbers[0] = 10
numbers[1] = 20
numbers[2] = 30
numbers[3] = 40
numbers[4] = 50

Process returned 0 (0x0)   execution time : 0.127 s
Press any key to continue.
|
```

## Multidimensional array in C

Multidimensional arrays are one of the most powerful features of the C programming language. They allow you to store data in a table-like format, where each row and column can be accessed using an index. In this blog post, we'll look at multidimensional arrays in C, including their syntax, example usage, and output.

### Syntax of Multidimensional Arrays in C

To create a multidimensional array in C, you need to specify the number of dimensions and the size of each dimension. The general syntax for declaring a multidimensional array is as follows:

1. `type array_name[size1][size2]...[sizeN];`

Here, `type` is the data type of the elements that will be stored in the array, `array_name` is the name of the array,

and *size1*, *size2*, ..., *sizeN* are the sizes of each dimension of the array.

## Example Usage of Multidimensional Arrays in C

Let's look at a practical example of using multidimensional arrays in C. Suppose we want to create a program that stores the grades of 5 students in 4 different subjects. We can use a 2-dimensional array to store this data, where each row represents a student, and each column represents a subject.

### Example:

Here's an example program that prompts the user to enter the grades for each student and subject, and then calculates the average grade for each student and subject:

```
#include <stdio.h>

int main() {
    int grades[5][4];

    // Prompt user to enter
    grades for (int i = 0; i < 5; i++) {
        printf("Enter grades for student %d:\n", i+1);
        for (int j = 0; j < 4; j++) {
            printf("Subject %d: ", j+1);
            scanf("%d", &grades[i][j]);
        }
    }

    // Calculate average grade for each student
    printf("\nAverage grade for each student:\n");
    for (int i = 0; i < 5; i++) {
```

```
float sum = 0;
for (int j = 0; j < 4; j++) {
    sum += grades[i][j];
}
float avg = sum / 4;
printf("Student %d: %.2f\n", i+1, avg);
}
```

```
// Calculate average grade for each subject
printf("\nAverage grade for each subject:\n");
for (int j = 0; j < 4; j++) {
    float sum = 0;
    for (int i = 0; i < 5; i++)
    { sum += grades[i][j];
    }
    float avg = sum / 5;
    printf("Subject %d: %.2f\n", j+1, avg);
}
```

```
return 0;
}
```

```
C:\Users\ACER\Documents\gf  X  +  v

Subject 2: 84
Subject 3: 80
Subject 4: 72
Enter grades for student 4:
Subject 1: 64
Subject 2: 45
Subject 3: 76
Subject 4: 46
Enter grades for student 5:
Subject 1: 81
Subject 2: 56
Subject 3: 76
Subject 4: 81

Average grade for each student:
Student 1: 96.00
Student 2: 86.00
Student 3: 78.50
Student 4: 57.75
Student 5: 73.50

Average grade for each subject:
Subject 1: 82.00
Subject 2: 72.80
Subject 3: 83.20
Subject 4: 75.40

Process returned 0 (0x0)    execution time : 73.311 s
Press any key to continue.
|
```

```
#include <stdio.h>
int main()
{
    int rows, cols;
    printf(" Ishita Chauhan");
    printf("\nEnter the number of rows:
"); scanf("%d", &rows);
    printf("Enter the number of columns: ");
```

```
scanf("%d", &cols);
int matrix[rows][cols];
int transpose[cols][rows];
printf("Enter the elements of the
matrix:\n"); for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        scanf("%d", &matrix[i][j]);
    }
}
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        transpose[j][i] = matrix[i][j];
    }
}
printf("Transpose of the matrix:\n");
for (int i = 0; i < cols; i++) {
    for (int j = 0; j < rows; j++) {
        printf("%d ", transpose[i][j]);
    }
    printf("\n");
}

return 0;
}
```

```
C:\Users\semwa\Downloads X + v
Ishita Chauhan
Enter the number of rows: 2
Enter the number of columns: 4
Enter the elements of the matrix:
14
45
74
45
78
74
45
45
Transpose of the matrix:
14 78
45 74
74 45
45 45

Process returned 0 (0x0)   execution time : 11.304 s
Press any key to continue.
```

13°C Mostly cloudy

Search

ENG IN 22:16 04-12-2023



**Q10. Explain with example with Structure, Declaration, and Initialization, Structure Variables, Array of Structures, and Use of typedef, Passing Structures to Functions. Define union declaration, and Initialization Passing structures to functions. Explain difference between Structure and Union. Write a program on details of a bank account with the fields account number, account holder's name, and balance. Write a program to read 10 people's details and display the record with the highest bank balance.**

### **What Is Structures In C**

A structure can be defined as a single entity holding variables of different data types that are logically related to each other. All the data members inside a structure are accessible to the functions defined outside the structure. To access the data members in the main function, you need to create a structure variable.

### **Syntax to Define a Structure in C**

```
struct structName  
  
{  
  
    // structure definition  
  
    Data_type1 member_name1;  
  
    Data_type2 member_name2;
```

```
Data_type2 member_name2;  
  
};
```

## How to Declare Structure Variables?

Variables of the structure type can be created in C. These variables are allocated a separate copy of data members of the structure. There are two ways to create a structure variable in C.

- **Declaration of Structure Variables with Structure Definition**

This way of declaring a structure variable is suitable when there are few variables to be declared.

### ***Syntax***

```
struct structName  
{  
  
    // structure definition  
  
    Data_type1 member_name1;  
  
    Data_type2 member_name2;  
  
    Data_type2 member_name2;
```

```
} struct_var1, struct_var2;
```

The structure variables are declared at the end of the structure definition, right before terminating the structure. In the above example, storeA and storeB are the variables of the structure bookStore. These variables will be allocated separate copies of the structure's data members that are storeName, totalBooks, and storeLicense.

- **Declaration of Structure Variables Separately**

This way of creating structure variables is preferred when multiple variables are required to be declared. The structure variables are declared outside the structure.

### ***Example***

```
struct bookStore  
{  
  
    char storeName  
  
    int totalBooks;  
  
    char storeLicense[20];  
  
};  
  
int main()
```

```
{  
  
    struct bookStore storeA, storeB;  
  
}
```

When the structure variables are declared in the main() function, the keyword struct followed by the structure name has to be specified before declaring the variables. In the above example, storeA and storeB are the variables of the structure bookStore.

Next, let us see how to initialize member in Structure in C programming

### How to Initialize Structure Members?

Structure members cannot be initialized like other variables inside the structure definition. This is because when a structure is defined, no memory is allocated to the structure's data members at this point. Memory is only allocated when a structure variable is declared. Consider the following code snippet.

```
struct rectangle  
{  
  
    int length = 10;  
  
    int breadth = 6;
```

```
};
```

A compilation error will be thrown when the data members of the structure are initialized inside the structure.

To initialize a structure's data member, create a structure variable. This variable can access all the members of the structure and modify their values. Consider the following example which initializes a structure's members using the structure variables.

```
struct rectangle  
{  
    // structure definition  
    int length;  
    int breadth;  
};  
  
int main()  
{  
    struct rectangle my_rect; // structure variables;  
    my_rect.length = 10;  
    my_rect.breadth = 6;
```

```
}
```

In the above example, the structure variable `my_rect` is modifying the data members of the structure. The data members are separately available to `my_rect` as a copy. Any other structure variable will get its own copy, and it will be able to modify its version of the length and breadth

## What is Structure and Union in C.

The structure is the container defined in C to store data variables of different types and also supports the user-defined variable's storage. On the other hand, Union is also a similar kind of container in C which can also hold the different types of variables along with the user-defined variables. A structure contains an ordered group of data objects. Unlike the elements of an array, the data objects within a structure can have varied data types. Each data object in a structure is a member or field. **A union is an object similar to a structure except that all of its members start at the same location in memory.**

- **Structure:** Structure is an arrangement and organization of interrelated elements in a material object or system, or the object or system so organized.

The structure is a user-defined data type in C language which allows us to combine data of different types together. It helps to construct complex data.

- **Union:** A union is a class all of whose data members are mapped to the same address within its object. A union is a special data type available in C that allows to the storage of different data types in the same memory location.

## Difference Between Structure and Union in C

Parameter	Structure	Union
<b>Keyword</b>	A user can deploy the keyword <b>struct</b> to define a Structure.	A user can deploy the keyword <b>union</b> to define a Union.
<b>Internal Implementation</b>	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.
<b>Accessing Members</b>	A user can access individual members at a given time.	A user can access only one member at a given time.
<b>Syntax</b>	<p>The Syntax of declaring a Structure in C is:</p> <pre> struct [structure name] { type element_1; type element_2; . . . } variable_1, variable_2, ...; </pre>	<p>The Syntax of declaring a Union in C is:</p> <pre> union [union name] { type element_1; type element_2; . . } variable_1, variable_2, ...; </pre>
<b>Size</b>	A Structure does not have a shared location for all of its members. It makes the size of a Structure to be	A Union does not have a separate location for every member in it. It makes

	greater than or equal to the sum of the size of its data members.	its size equal to the size of the largest member among all the data members.
<b>Value Altering</b>	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.
<b>Storage of Value</b>	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.
<b>Initialization</b>	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.