***************** C- programming notes******************************

24th jan-

A program is a set of instructions that, when executed, generate an output.
The data that is processed by a program consists of various characters and symbols

what exactly the token is?
the smallest unit in a 'C' program. It is each and every word and punctuation that you come across in your C program.
The compiler breaks a program into the smallest possible units (Tokens) and proceeds to the various stages of the compilation.
C Token is divided into six different types, Keywords, Operators, Strings, Constants, Special Characters, and Identifiers.

An identifier is nothing but a name assigned to an element in a program. Example, name of a variable, function, etc. Identifiers in C language are the user-defined names consisting of 'C' standard character set.
As the name says, identifiers are used to identify a particular element in a program. Each identifier must have a unique name.
Following rules must be followed for identifiers:

>The first character must always be an alphabet or an underscore.
>It should be formed using only letters, numbers, or underscore.
>A keyword cannot be used as an identifier.
>It should not contain any whitespace character.
>The name must be meaningful

****

25th jan-

Operator is a symbol that tells the compiler to perform mathematical or logical
functions. It is possible to use operators as tools to update or alter the values of
variables or data

The data items on which an operator acts is called its operands. The operators can be unary, binary or ternary depending upon whether it operates on one,
two or three operands.

The types of operators are
1. Arithmetic operators
2. Logical operators
3. Relational operators

The type of operations that can be performed on the data objects are specified by the
operators. The data items on which an operator acts are called operands. Combine
different types of arithmetic operators forms arithmetic expressions

As C language has a large number of operators, the precedence of the operators is of
greater importance to the programmer. The evaluation order of the operators It will be
discussed after studying different types of operators. Let us now begin with the basics of
the operators
These operators are either binary operators,
which takes two operands or unary operators, which take only one operand. Operators
are classified into different categories depending on their nature of operation like
arithmetic operators, logical and relational operators, assignment operators, bit
operators etc.

so basically these are the examples -

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

****

27th jan-

>All valid C programs must contain the main() function. The code execution begins from the start of the main() function.
>The printf() is a library function to send formatted output to the screen.
> The function prints the string inside quotations.
>To use printf() in our program, we need to include stdio.h header file using the #include <stdio.h> statement.
>The return 0; statement inside the main() function is the "Exit status" of the program. It's optional.

for example

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

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.
The C language comes with standard functions printf() and scanf() so that a programmer can perform formatted output and input in a program.
The formatted functions accept various format specification strings along with a list of variables in the form of parameters.

The C program performs input and output operations using different input and output functions. In this article, we will take a look at formatted input and output

Need of formatted input and output in C

We use the formatted input and output functions in the C language for taking single or multiple inputs from the programmer/user at the console.
These functions also allow a programmer to display single or multiple values,
in the form of output, to the users present in the console.

*******************Decision making statements in C***************************

There come situations in real life when we need to make some decisions and based on these decisions,
we decide what should we do next. Similar situations arise in programming also where
we need to make some decisions and based on these decisions we will execute the next block of code.
For example, in C if x occurs then execute y else execute z

if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements
will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not

```
if(condition)
{
   // Statements to execute if
   // condition is true
}
```

if-else in C-

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false
it won’t. But what if we want to do something else if the condition is false. Here comes the C else statement.
We can use the else statement with if statement to execute a block of code when the condition
is false.

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

nested-if in C

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if
statement inside another if statement. Yes, both C and C++ allow us to nested if statements within if statements, i.e,
we can place an if statement inside another if statement.

following is the syntax

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

if-else-if ladder in C

Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the
if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed.
If none of the conditions are true, then the final else statement will be executed.

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

****

28th Jan -

---Looping statements in C---

>Looping Statements in C execute the sequence of statements many times until the stated condition becomes false.
> A loop in C consists of two parts, a body of a loop and a control statement.
>The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.
>The purpose of the C loop is to repeat the same code a number of times.

Depending upon the position of a control statement in a program, looping statement in C is classified into two types:

1. Entry controlled loop

2. Exit controlled loop

In an entry control loop in C, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an exit controlled loop, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

>In an entry control loop in C, a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

>In an exit controlled loop, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

C' programming language provides us with three types of loop constructs:

1. The while loop

2. The do-while loop

3. The for loop

1) while loop-

A while loop is the most straightforward looping structure. While loop syntax in C programming language is as follows:

```
while (condition) {
             statements;
}
```

2) Do-While loop in C

A do…while loop in C is similar to the while loop except that the condition is always executed after the
body of a loop. It is also called an exit-controlled loop.

```
 do {
  statements
} while (expression);
```

3) For loop in C-

>The initial value of the for loop is performed only once.
>The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
>The incrementation/decrementation increases (or decreases) the counter by a set value.

--------------ARRAYS----------------------------

An array is defined as an ordered set of similar data items. All the data items of an array are
stored in consecutive memory locations in RAM

The elements of an array are of same data type
and each item can be accessed using the same name

Declaration of an array:-
We know that all the variables are declared before they are used in
the program. Similarly, an array must be declared before it is used.
During declaration, the size
of the array has to be specified. The size used during declaration of the
array informs the
compiler to allocate and reserve the specified memory locations.

Syntax:-

```
data_type array_name[n];
```
where, n is the number of data items (or) index(or) dimension.
0 to (n-1) is the range of array.
Ex: int a[5];
float x[10];

(i) Initializing all specified memory locations:-

Arrays can be initialized at the time of
declaration when their initial values are known in advance. Array
elements can be initialized
with data items of type int, char etc.
Ex:-
```
int a[5]={10,15,1,3,20};
```

(ii) Partial array initialization:-

Partial array initialization is possible in c language. If the
number of values to be initialized is less than the size of the array,
then the elements will be
initialized to zero automatically.
Ex:-

```
int a[5]={10,15};
```

(iii) Initialization without size:-

Consider the declaration along with the initialization.
Ex:-
```
char b[]={'C','O','M','P','U','T','E','R'};
```
In this declaration, eventhough we have not specified exact number of
elements to be used in
array b, the array size will be set of the total number of initial values
specified. So, the array size
will be set to 8 automatically. The array b is initialized as shown in
figure

-----Two-dimensional arrays of variable length-------

An array consisting of two subscripts is known as two-dimensional array.
These are often known
as array of the array. In two dimensional arrays the array is divided
into rows and columns,.
These are well suited to handle the table of data. In 2-D array we can
declare an array as :
Declaration:-
Syntax: data_type array_name[row_size][column_size];

Ex: int arr[3][3] ;

---------Multidimensional arrays ------------------------------

these are often known as array of the arrays. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional
arrays we will be discussing mainly about two dimensional arrays and a bit about three
dimensional arrays.
Syntax: data_type array_name[size1][size2][size3]------[sizeN];
In 2-D array we can declare an array as :

```
int arr[3][3] = { 1, 2, 3,
 4, 5, 6,
 7, 8, 9
 };
```

****
****
****

31st Jan-

----------------String in C-------------

A String in C is nothing but a collection of characters in a linear sequence.
'C' always treats a string a single data even though it contains whitespaces. A single character is defined using
 single quote representation. A string is represented using double quote marks.

```
char string_variable_name [array_size];
```

more prescize way

he classic Declaration of strings can be done as follow:

```
 char string_name[string_length] = "string";
```

The size of an array must be defined while declaring a C String variable because it is used to calculate how many characters are going to be stored inside the string variable in C.
 Some valid examples of string declaration are as follows,

```
char first_name[15];    //declaration of a string variable
char last_name[15];
```

The above example represents string variables with an array size of 15. This means that the given C string array is capable of holding 15 characters at most.
The indexing of array begins from 0 hence it will store characters from a 0-14 position. The C compiler automatically adds a NULL
character '\0' to the character array created.

 strcpy function
(String Copy)

In the C Programming Language, the strcpy function copies the string pointed to by s2 into the object pointed to by s1.
It returns a pointer to the destination.

The syntax for the strcpy function in the C Language is:

```
char *strcpy(char *s1, const char *s2)
```

In the above syntax, two parameters are passed as strings, i.e., str1 and str2, and the return type is int means that the strcmp() returns an integer value.

>The strcmp() function compares the character of both the strings.
 >If the first character of both the strings are same, then this process of comparison will continue until all the characters are compared or the pointer points to the null character '\0'.

As we know, the best way to concatenate two strings in C programming is by using the strcat() function.
However, in this example, we will concatenate two strings manually.

Here, two strings s1 and s2 and concatenated and the result is stored in s1.

It's important to note that the length of s1 should be sufficient to hold the string after concatenation.
If not, you may get unexpected output.

------------User defined function in C------------------------------

C function can be classified into two categories, namely

 (i) library functions
 (ii) user-defined function.

• Main() is an example of user-defined functions.
• While printf() and scanf() belong to the category of library functions.
• The main distinction between these two categories is that library functions are
not required to be written by us whereas a user- defined function has to be
developed by the user at the time of writing a program.

• A function is a self-contained code that
performs a particular task.
• Once a function has been designed , it can
be treated as a block-box that takes some
data from the main program and return a
value.
• Every c program can be designed using the
collection of these black boxes known as
function.
• Any function can call any other function.
• It can call itself.
• A called function can also can call another
function.

---------------------------calling a function in C---

>Depending on whether arguments are present or not and whether a value is returned or not, functions are categorized into –

>Functions without arguments and without return values

>Functions without arguments and with return values

>Functions with arguments and without return values

>Functions with arguments and with return values

------------------------------Nested & recursion in C-----------------------------

Nested Recursion in C Language with Examples. P where we discussed Indirect Recursion in C with Examples.
In Nested recursion, the recursive function will pass the parameter as a recursive call.

Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.

That being said, recursion is an important concept. It is frequently used in data structure and algorithms.
For example, it is common to use recursion in problems such as tree traversal.

----------------------Scope and Lifetime of Variables in C--------------------

In a previous posting we looked at the principles (and peculiarities) of declarations and definitions. Here I would like to address the concepts of scope and lifetime of variables (program objects to be precise).

In the general case:

The placement of the declaration affects scope
The placement of the definition affects lifetime

```
int global_a;       /* tentative defn; become actual defn init to 0 */
int global_b = 20;      /* defn and implicit-decl */


int f(int* param_c)
{
   int local_d = 10;
   . . .
   return local_d;
}
```

****

1st feb-

-----------------------------------structure & union---

A structure is a data type in C that allows a group of related variables to be treated as a single unit instead of separate entities.
A structure may contain elements of different data types – int, char, float, double, etc.
 It may also contain an array as its member. Such an array is called an array within a structure. An array within a structure is a member of the structure
and can be accessed just as we access other elements of the structure.

What Is Structures In C and How to Create It?

Structure in C
Table of Contents

At times, during programming, there is a need to store multiple logically related elements under one roof. For instance, an employee's details like name, employee number, and designation need to be stored together. In such cases, the C language provides structures to do the job for us.

Structure_in_C_1

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.

Now, lets get started with Structure in C programming.

Why Do We Use Structures in C?
Structure in C programming is very helpful in cases where we need to store similar data of multiple entities. Let us understand the need for structures with a real-life example. Suppose you need to manage the record of books in a library. Now a book can have properties like book_name, author_name, and genre. So, for any book you need three variables to store its records. Now, there are two ways to achieve this goal.

The first and the naive one is to create separate variables for each book. But creating so many variables and assigning values to each of them is impractical. So what would be the optimized and ideal approach? Here, comes the structure in the picture.

We can define a structure where we can declare the data members of different data types according to our needs. In this case, a structure named BOOK can be created having three members book_name, author_name, and genre. Multiple variables of the type BOOK can be created such as book1, book2, and so on (each will have its own copy of the three members book_name, author_name, and genre).

Moving forward, let’s he a look at the syntax of Structure in C programming

basic syntax-

```
{

   // structure definition

   Data_type1 member_name1;

   Data_type2 member_name2;

   Data_type2 member_name2;

};
```

Keyword struct:

The keyword struct is used at the beginning while defining a structure in C. Similar to a union, a structure also starts with a keyword.
structName:
This is the name of the structure which is specified after the keyword struct.
data_Type: The data type indicates the type of the data members of the structure. A structure can have data members of different data types.
member_name:
This is the name of the data member of the structure. Any number of data members can be defined inside a structure.
 Each data member is allocated a separate space in the memory.

---------------Pointers in C-----------------------

pointers are powerful features of C and C++ programming. Before we learn pointers, let's learn about addresses in C programming.
```
scanf("%d", &var);
```

the basic example
```
int* pc, c;
c = 5;
pc = &c;
```

Note: In the above example, pc is a pointer, not *pc.
You cannot and should not do something like *pc = &c;

By the way, * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

We have assigned the address of c to the pc pointer.

Then, we changed the value of c to 1. Since pc and the address of c is the same, *pc gives us 1.

Let's take another example.

```
int* pc, c;
```

```
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);  // Ouptut: 1
printf("%d", c);    // Output: 1
```

-------------------------------Pointers and arrays, Pointers & character strings--

An array of pointers to strings is an array of character pointers where each pointer points to the first character of the string or the base address
 of the string. Let's see how we can declare and initialize an array of pointers to strings.

It is important to emphasize that in an array of pointers to strings, it is not guaranteed that the all
the strings will be stored in contiguous memory locations. Although the characters of a particular string literal are always stored in contiguous memory location.

****

2nd Feb-

-----file handling in C language-----

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.
FILE *fptr;

opening a file

Opening a file - for creation and edit
Opening a file is performed using the fopen() function defined in the stdio.h header file.

The syntax for opening a file in standard I/O is

ptr = fopen("fileopen","mode");

>Let's suppose the file newprogram.txt doesn't exist in the location E:\cprogram.
>The first function creates a new file named newprogram.txt and opens it for writing as per the mode 'w'.
>The writing mode allows you to create and edit (overwrite) the contents of the file.
>Now let's suppose the second binary file oldprogram.bin exists in the location E:\cprogram. The second function opens the existing file for reading in binary mode 'rb'.

Closing a File
The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using the fclose() function.

```
fclose(fptr);
```

Here, fptr is a file pointer associated with the file to be closed.

How to check whether the file has opened successfully?

If file does not open successfully then the pointer will be assigned a NULL value, so you can write the logic like this:
This code will check whether the file has opened successfully or not. If the file does not open, this will display an error message to the user.

```
FILE fpr;
fpr = fopen("C:\\myfiles\\newfile.txt", "r");
if (fpr == NULL)
{
    puts("Error while opening file");
    exit();
}
```

------------------Dynamic memory allocation in C-----------------------

Since C is a structured language, it has some fixed rules for programming. One of them includes changing the size of an array.
An array is a collection of items stored at contiguous memory locations.

This procedure is referred to as Dynamic Memory Allocation in C.
Therefore, C Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.
C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under <stdlib.h>
header file to facilitate dynamic memory allocation in C programming.
They are:

>malloc()
>calloc()
>free()
>realloc()

1. MALLOC()-

The “malloc” or “memory allocation” method in C is used to dynamically allocate a single large block of memory with the specified size.
It returns a pointer of type void which can be cast into a pointer of any form.
 It doesn’t Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

```
ptr = (cast-type*) malloc(byte-size)
```

2.C calloc() -

“calloc” or “contiguous allocation” method in C is used to dynamically allocate the specified number of blocks of memory of the specified type.
it is very much similar to malloc() but has two different points and these are:
It initializes each block with a default value ‘0’.
It has two parameters or arguments as compare to malloc().

3.C free() -

“free” method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax:

```
free(ptr);
```

4.C realloc() -

“realloc” or “re-allocation” method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

Syntax:

```
ptr = realloc(ptr, newSize);
```

```
************************************************************************
************************************************************************
************************************************************************
************************************************************************
************************************************************************
************************************************************************
****************************************************
```

3rd feb-

GDB stands for GNU Project Debugger and is a powerful debugging tool for C(along with other languages like C++).
It helps you to poke around inside your C programs while they are executing and also allows you to see what exactly happens when your program crashes.
GDB operates on executable files which are binary files produced by compilation process.

For demo purpose, below example is executed on a Linux machine with below specs.

Step 1. Compile the C program with debugging option -g
Compile your C program with -g option. This allows the compiler to collect the debugging information.

```
$ cc -g factorial.c
```

launching in GDb

```
$ gdb a.out
```

Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

For example:

```
gcc -
Wall -
Werror -ansi -pedantic-errors prog1.c -o prog1.x
```

Now you add a -g option to enable built-in debugging support (which gdb needs):

```
gcc [other flags] -g <source files> -o <output file>
```

For example:

```
gcc -Wall -Werror -ansi -pedantic-errors -g prog1.c -o prog1.x
```