BASIC OF C LANGUAGE FROM SELFLEARNING.

Basics with exampled repository file.

C LANGUAGE

BASIC

Book Written By
Kaushal Bhatol

About

This book is refrence created by author for owen learning time shaduled, all c files which are created by auther is avalible in this following link.
This book is started written at April, 2021. And by week by week learning and taking refrence are written heare.
You can find both of things heare, my personal refrence as this book and some material which I used is also included in my git directory.
I only want to help some beggener for c language. This is my way to learn something new.
All things are well documented.

Links

All documents, pictures and articles root links for preventing **copyright** issue.

YouTube playlist:
Site: www.geeksforgeeks.org

License

Operators [Week 1, 08]:

An Operators is a symbol used to perform operations in given programming language.

==> **Types of operators**
1. Arithmetic operators
2. Relation Operators
3. Logical Operators
4. Bit-wise operators
5. Assignment operators
6, Miscellaneous

1. Arithmetic operator [week 1 : 08.1]

This operators are basic operators for day to day life.

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	modules

2. Relation operator [week 1 : 08.1]

Operators	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than
<=	Less than or equal to

Note: mostly this operators are used in if, else statement.

3. Logical operator [week 2 : 08.2]

Operators	Description	Example
&&	Logical “AND” operator, both the operands are non-zero then condition is true.	a&&b
||	Logical “OR” operator, if any of these two operands is none-zero, Then condition become true.	a||b
!	Logical “NOT” operator, It is used to reverse the logical state of its operand, if condition is true..	!a

4. Bit-wise operator [week 1, 08.2]

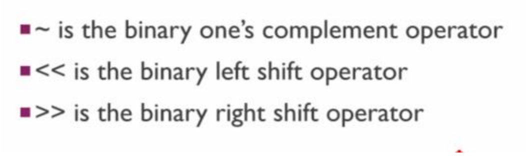
- *Bit-wise operator are functioning on bit.*

Value A	Value B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Symbols	Names
&	AND
|	OR
^	Exclusive (XOR)

XOR Operator:
If one value is true and second one is false, Then XOR give true value.

Other Bit-wise operators



- ~ is the binary one’s complement operator
- << is the binary left shift operator
- >> is the binary right shift operator

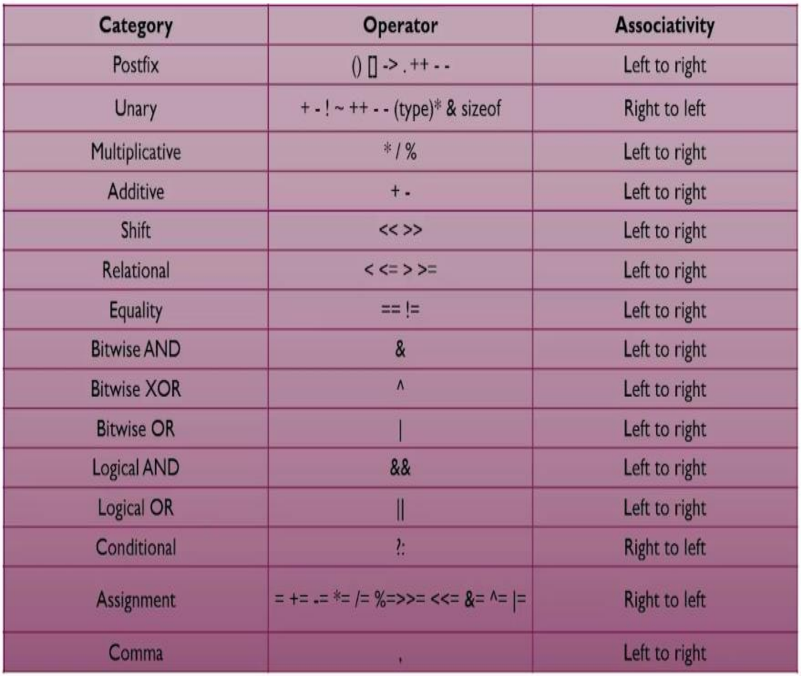
5. Assignment operator

Operators	Description
=	Simple assignment operator assign value from right side operands
+=	Add AND assignment operator. It’s adds the right operand to the left operands and assign the result to the left operands
-=	Subtract AND assignment operator. It subtracts the right operands from the left to the result is assignment to the left operand.
*=	Multiply AND assignment operator. It multiples the right operands from the left to the result is assignment to the left operand.
/=	Divide AND assignment operator. It divide the right operands from the left to the result is assignment to the left operand.

6. Miscellaneous

Operators	Description	Example
sizeof()	Returns the size of variable.	sizeof(a), where a is an integer, will return int's size on that architecture.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to variable	*a;
?:	Conditional Expression	If condition is true? The value X: otherwise value Y

Operators PRECEDENCE IN C



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Format specifier [week 1, 09]

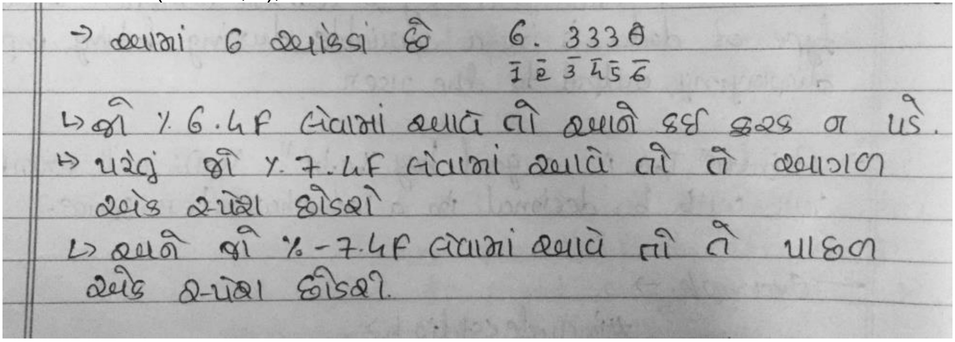
Format	Work
%c	Character print
%d	Integer print
%f	Float print
%l	Long print
%lf	Double print
%LF	Long Double print

-> Format specifier is a way to tell the compiler what type of data I sin a variable during taking input and displaying output to the user.

-> Pritf("This is a good boy %a.bf", var); will print var with b decimale in a "a" character space

-Exp->
Week 1, 09 .c file

```
 float c= 6.333
           Printf("%7.4f", c);
```



-> આમાં 6 આંકડા છે 6.3330
1 2 3 4 5 6

-> જો %6.4f લેવામાં આવે તો આમાં કઈ ફરક ન પડે.
-> પરંતુ જો %7.4f લેવામાં આવે તો તે આગળ એક સ્પેસ છોડશે
-> અને જો %-7.4f લેવામાં આવે તો તે પાછળ એક સ્પેસ છોડશે.

(Avoid above picture if you don't know Guajarati language.)

-> Note: 0 is always included in format
-> It work with - also.. exp: %-2.2f
->It only work if value is less then applied specifier. Else it show real values.

Constant [Week 1, 10]

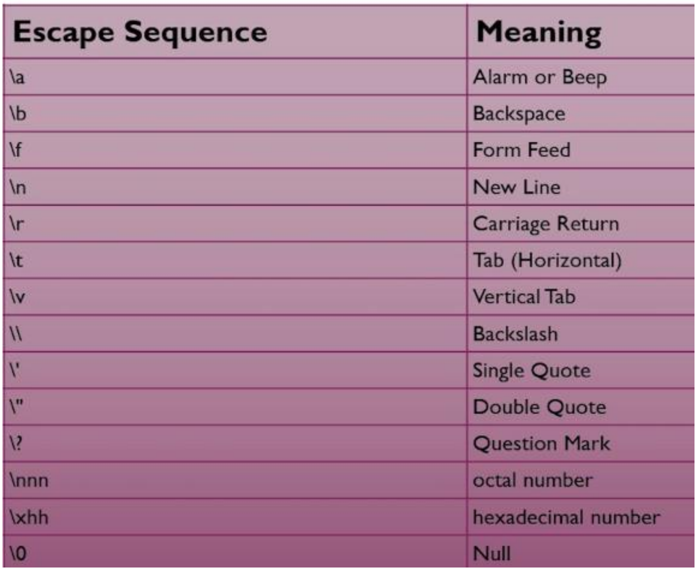
-> *A constant is a value or variable that can't be changed in the program.*

Example: 15, 'a', 3.4, "Name, etc..

-> There are Two ways to define constant in C Program.
1) Constant Keyword
    Exp: const float b = 7.499;
2) #define preprocessor
    Exp: #define PI 3.14

Escape sequence [Week 1]

***-> An** escape sequence inn C programming, language is a sequence of characters.*
-> It doesn't represent itself when used inside string literal or character.
-> It is composed of two or more characters string with backslash \.

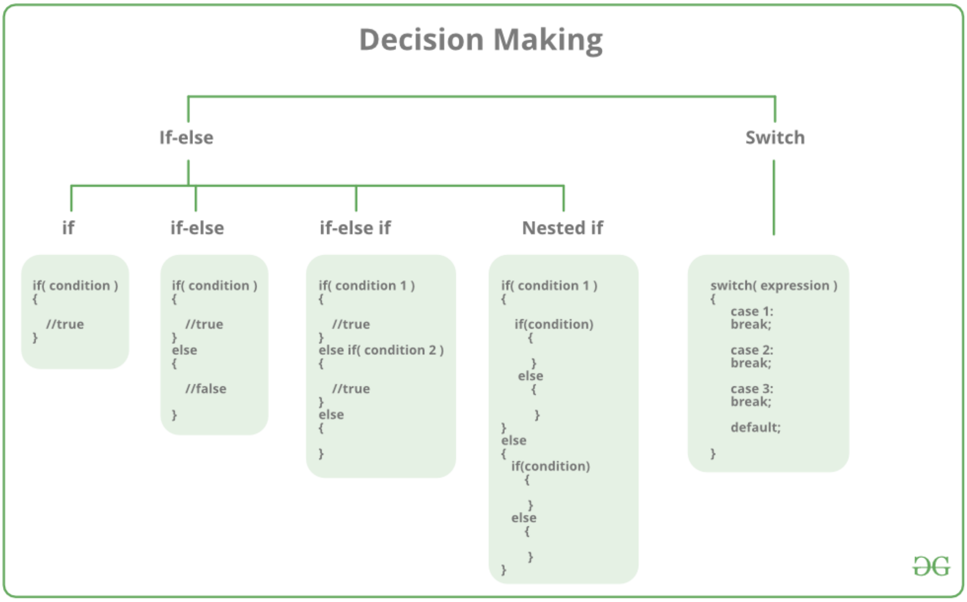


Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

if else [week 1, 11]

if else [11]

- ➔ *It is used to perform operation based on some condition*
- ➔ *Types of if statement:*
  - If statement
  - If else statement
  - if-else-if ladder
  - Nested if statement


Decision Making
If-else
Switch
if
if-else
if-else if
Nested if
if(condition) { //true }
if(condition) { //true } else { //false }
if(condition 1) { //true } else if(condition 2) { //true } else { }
if(condition 1) { if(condition) { } else { } } else { if(condition) { } else { } }
switch(expression) { case 1: break; case 2: break; case 3: break; default; }

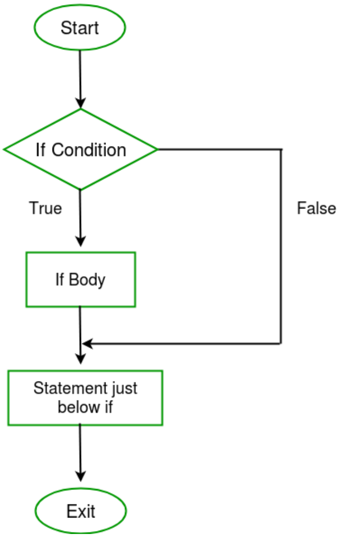
Source: https://www.geeksforgeeks.org/

1. If statement [11.1]:

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. If a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)

{

   // Statements to execute if

   // condition is true

}
```



Here, **condition** after evaluation will be either true or false. C if statement accepts Boolean values – if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

Example:

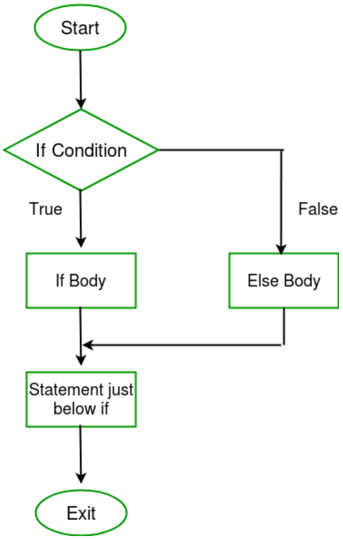
```
if(condition)

   statement1;

   statement2;
```

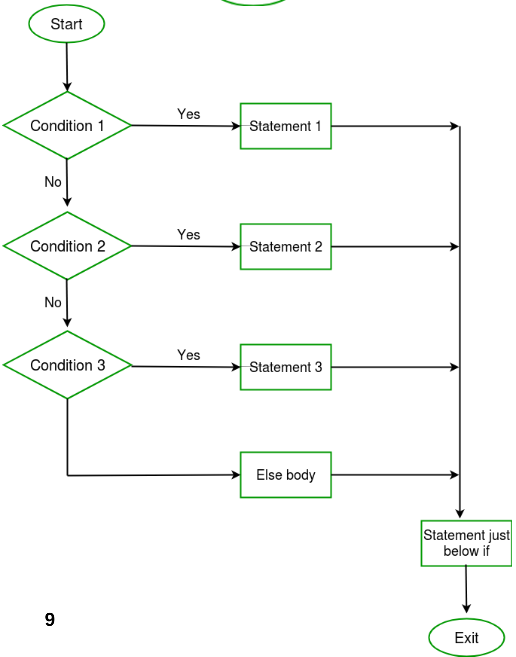
```
// Here if the condition is true, if block

// will consider only statement1 to be inside

// its block.
```

2. If else [11.1]:

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.
Syntax:

```
if (condition)

{

    // executes this block if

    // condition is true

}

else

{

    // executes this block if

    // condition is false

}
```





3. if-else-if ladder [11.1]:

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.

Syntax:

```
if (condition)

    statement;

else if (condition)
```

statement;

.

.

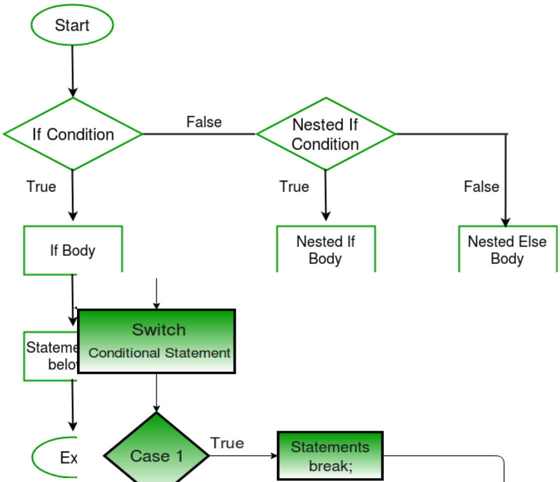
else

4. Nested if statement [11.2]:

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

Syntax:

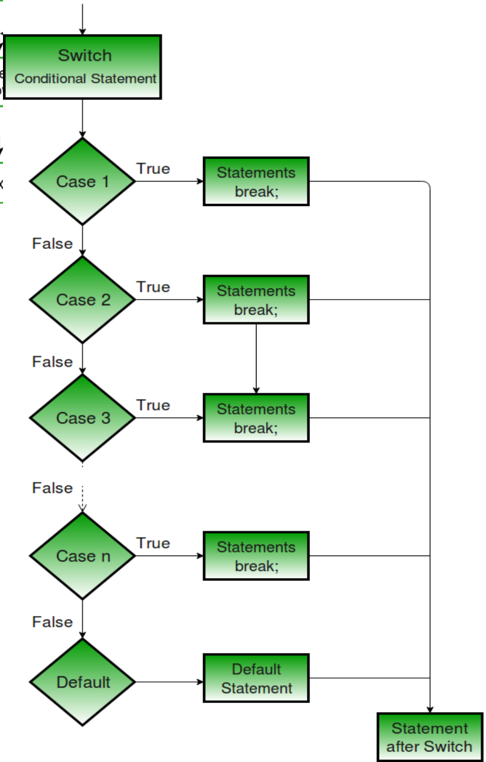
```
if (condition1)

{

   // executes when condition1 is true

   if (condition2)

   {

      // executes when condition2 is
true

   }

}
```



Note: when you want a is equal to some value use "==" instead of "=", else it don't go on next steps and print the"=" condition in all situation.

Switch Case [week 1, 12]

Rules:
1. Switch expression must be int or char.
2. Case value must be int or char.
3. Case must come inside switch.
4. Break is not a must.



Note: If break is not added, code will execute below cases also. And Stop when it get break in any case.

Types:
1. Switch case without break.
2. Switch case.
3. Nested switch case.

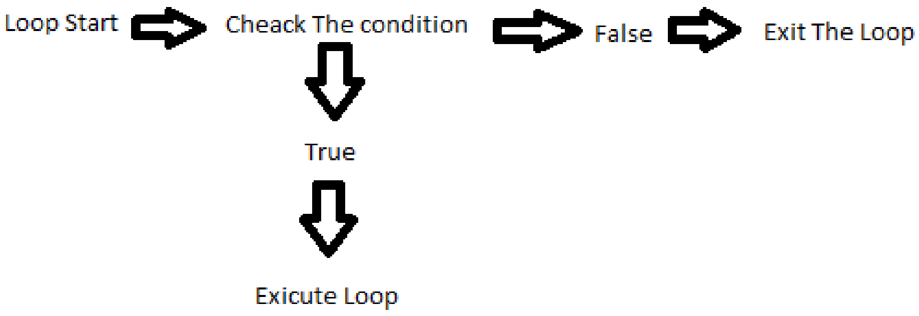
Syntax:
```
switch (n)
{
    case 1: // code to be executed if n = 1;
        break;
    case 2: // code to be executed if n = 2;
        break;
    default: // code to be executed if n doesn't match any cases
}
```

Loops [week 1, 13]

Advantage of Loops
- Code reusability
- Saves time
- Traversing

Types of Loops
- Do while loop
- While loop
- For loop

Basic syntax:



1. Do While Loop [13.1]

Syntax:

```
do{
        // Code to be Executed
} while(condition);
```

Note:

- Do while loop execute once time without checking condition.

Example:

- Week 1, 13.1 do while loop.c

Exercise:

- Write Multiplication table for users input number.

  a. Exercise / 05 multiplication do while loop.c

2. While Loop [13.2]

Syntax:

```
while(condition){
        // code to be executed
}
```

Note:

While loop check condition first then execute.

Example & Exercise:

- Week 1, 13.2 file
- Print numbers 0 to user input. (A. Exercise /06 file)
  - Maximum 50 allowed
  - Validate for positive values only.

3. for Loop [13.3]

- ➔ The for loop is used to itreate the statement or put the program sraval times.
- ➔ It used to traverse the data structure like the arrays and linked lists.
- ➔ It has a little different syntax then while and do while loops.

Syntax:

```
for(expression 1; expression 2; expression 3;) {
        // Co de to be executed
        }
```

Example:

```
For( i=0; i < 5; i++) {
        Printf("%d", i);
}
```

Note: first i=0, then condition two i<5, then increment i++, Now loop again to condition i<5 then increment.

Properties of expression 1:

- The expression represent the initialization of the loop variable.
- We can initialize more than one variable in expression 1.

Properties of expression 2:

- If is a condition expression. It creaks for a specifies condition to be satisfied. If it is not, the loop is terminated.
- It can have more than one condition. However the loop will iterate until the last condition become false other condition will be treated as statements.
- Expression 2 can perform the task of expression and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2, however in c, any non-zero value is true, and zero is false.

Properties of expression 3:

- Expression 3 is used to update the loop variable
- We can update more than one variable at the same time.
- Expression 3 is optional.

Break & Continue [week 2, 01 & 02]

Break [01]:

- ➔ Used to bring the program control out of the loop.
- ➔ The break statement is used inside loops or switch statement.
- ➔ Break statement can be used with
  - ○ Loops
  - ○ Switch case expression
- ➔ Break exit the loop.

Ex:

```
If (name==”harry”) {
        Break;
}
```

Continue [02]:

- ➔ Used to bring the program control to the next iteration of the loop.
- ➔ The continue statement skips some code inside the loop and continues with the next iteration.
- ➔ It is mainly used for condition so that we can skip some lines of code for a particular condition.

Ex. Look week 2, 02 file for example.

Go to statement [week 2, 03]

- Also called jump statement.
- Used to transfer program to a predefined label
- Its use is avoided since it cause confusion for the fallow programmers in understanding the code.
- Goto statement is preferable when we need to break multiple loops using a single statement at the same time.

Type casting [week 2, 04]

Use1:

- Converting one data type to other, ex. Float to int…
- Watch week2, 04 file's first example.
- Example: *printf*("the value of float b in integer is %d\n", (int) b);

Use2:

- Int and int arthamatic opration gives int ans also in float.
- For gatting float value must select one values as float.
- Watch out week2, 04 file's second example.
- Example: *printf*("the value of (float) b/5 is %f\n", (float) b/5);

Function [week 2, 05]

- Function are used to divide a large c program into smaller pieces.
- A function can be called multiple times to provide reusability and modularity to the c program.
- Also called procedures or subroutine

Syntax:

```
Return type function name (data type parameter 1, parameter 2…)
{
        // code to be executable
}
```

Advantage of Functions:

- We can avoid rewriting same logic through function.
- We can divide work among programmers using function.
- We can easily debug a program using function.

Declaration, Definition and Call:

- Library function: function which include in c header file.
- User defined function: Function created by c programmer to reduce complexity of a program.
-

Function code examples:

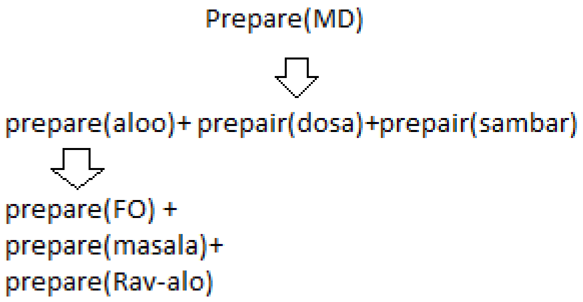
- With argument, with return value (ex. Week 2 05.1)
  - Normal code (ex. Week 2 05.1 > 1)
  - Declare function (ex. Week 2 05.1 > 2)
- With argument, without return value (ex. Week 2 05.2)
- Without argument, with return value (ex. Week 2 05.3)

- Without argument, without return value (ex. Week 2 05.2)

Recursion Function [week 2, 06]

- Recursion function or recursion is a process when a function calls a copy of itself to work on a small problem.
- Any function which call itself is called recursive function.
- This makes the life of programmer easy by dividing a given problem into easier problems.
- A termination condition is imposed on such functions to stop the executing copies of themselves forever.
- Any problem that can be solved recursively, can also be solved iteratively.

Intuitive understanding: preparing of masala dose



Why recursion?
- However, some problems are best suitable to be solved using recursion.
- For example, tower Hanoi, Fibonacci series, Factorial finding…

Example of Factorial calculation:

Rules:
- n! = nx(n-1)!
- 0! = 1
- 1! = 1

How:
- The case at which the function doesn’t recure is called the base case.
- Instance where the function keeps calling itself to perform a subtask, is called the recursive case.
- For factorial calculation the base case occurs at the parameter value of 0 and 1.

Rhythm:

factorial(5)
5 x factorial(4)
5 x 4 x factorial(3)
5 x 4 x 3 x factorial(2)

5 x 4 x 3 x 2 x factorial(1)
5 x 4 x 3 x 2 x 1 = 120

Code:

```
Int factorial(int n)
{
        If(n == 1 || n == 0){
        Return 1;
        }
        else{
        Return (n * factorial(n-1));
}

Int main()
{
Int n = // get value from user
Printg("the factorial of %d is %d", n, factorial(n));
Return 0;
}
```

Arrays [week 2, 07]

What is arrays:

- Arrays is a collection of data items of the same type.
- Itms are stored at contiguos memory locations.
- It can also store the collection of derived data type, such as pointer, structure etc.
- A one-dimensional array is like a list.
- A two-dimemsional array is llike table.
- The c language places no limts on the number of dimansions in an array.
- Some texts refer to one-dimensional arrays as **vetors**, two dimensional arrays as **matric**, and use the genral term arrays when the number of dimensions is unspacified or unimportant.

Why we do need arrays?

- Code that use arrays is sometime more organized and readable.
- If you were to store the marks in a test of 56 students, creating 56 variables will make program look clutter and messy.
- We can create arrays of integers and store the consecutive marks corresponding to the roll number in the arrays.

Advantage of arratys:

- It is used to represent multiple data items of same type by using only single name
- Accessing an items in a given arrays is a **verry fast**.
- Two dimensions arrays make it easy in mathematical applications as it used to ropresnt a matrix.

Properties of arrays:

- Data in arrays is stored in continguos memory locations.
- Each elements of an array is of same size.

- Any element of the arrays with give index can accessed very quiqly by using its address which can be calculated using the base address and thi index.

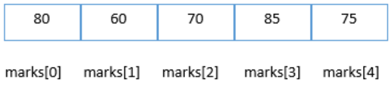
Syntax of arrays:

- Data_type name[size];
- Data_type name[size] = {x, y, z….}; // size not requierd in one-dimension.
- Data_type name[row] [column];
- Data_type name[row] [column] = {{1, 2, 3.. }, {44, 55, 554….}};

Examples:

Ex 1. One dimensional –

```
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



Ex 2. Two dimensional-

```
        int arr[4][3] = {{1, 2, 3}, {2, 3, 4}, {3, 4, 5}, {4, 5, 6}};
// marks[line][row]
  // i, j = 0 1 2
  // 0     1 2 3
  // 1     2 3 4
  // 2     3 4 5
  // 3     4 5 6
```

- What is value of arr[2][1]?
    - ans 4.

identify arrays:

- for idetifying arrays. My own created method is below:

```
int id[2][3] = {{11,22,33}, {4,5,77}};
          x   y
```

Box:

x->		0	1
y	0	11	4
	1	22	5
	2	33	7

output:

[0][0] = x y = 11

[0][1] = x y = 22

[0][2] = x y = 33

Disadvantage of arrays:

- Poor time complexity of insertation and deletation operation.
- Wastage of memory since arrays are fixed size.
- If there is enough space present in the memory but not in the memory but not in contiguos form, you will not be initialize your arrays.
- It is not possible to inarrase the size of the arrays, once you have declared the arrays.