PBFU1

Unit-1

Introduction to Programming Language

Hierarchy of Computer Language -

High level language
↓
Assembley Language
↓
Machine Language
↓
computer Hardware

There have been many programming Language some of them are listed below.

(1) C
(2) C#
(3) COBOL
(4) Python
(5) C++ ----

Most popular programming Language :-

- C
- python
- C++
- Java
- SCALA
- C#
- R
- Ruby
- Go
- swift
- JavaScript

PAGE NO.
DATE

- Characteristics of a program Language-

-> A programming language must be simple, easy to learn and use have good readability and human recognizable.

-> A Portable programming Language is always preferred.

-> Programming language's effciency must be high so that is can be easity converted into a machine code and executed consumes little space in memory.

-> A Programming Language shold be well structured and documented so that it is suitable for application development

-> Necessary tools for development, debugging, testing, maintence of a program must be provided by a programing language

-> A programming language shold provided single environment know as Integrated Development Environment (IDE)

-> A Programming Language must be consistent in terms of syntax

PAGE NO.
DATE

Type of computer Languages:

=> There are mainly two types of computer languages:-
- Low Level language
- High Level language

High Level languages:-

-> When we thik about computer programmers we are probaly think about people who write in High-level programming language

-> High level language are written in a form that is close to our human language. enabling to programming to just fouse on the problem being solved.

-> Examples include: C++, Java, Pascal, python, Visual Basic.

Advantages

(1) Easier to modify as it uses English like statemets

(2) Easier/faster to write code as it uses English like statements

(3) Easier to debug during development due to English like statements

(4) Portable code - not designed to run on just one type of machine.

Low level Language:-

Low level language are used to write programs that relate to the specific architecture and hardware of a particular type of computer.

→ Examples of low level language.
- Assembly Language
- Machine code

(1) Assembly Language

→ Few programmers write programs in low level assemly language, but it still used for developing code for specialist hardware, such as device drivers.

→ Advantages:-

- Can make use of special hardware or special machine dependent instructions (e.g. on the spcific chip)
- Translated program requiros less memory
- Write code that can be executed faster
- Total control over the code.
- Can work directly on memory locations.

(2) Machine code

Programmers rarely write in machine code (binary) as it is difficult to understand.

High-level Languages	Low-Level Languages
(1) High-level Languages are easy to learn and understand.	(1) challenging to learn and understand.
(2) They are excuted slower than lower level language because the they require a translator program.	(2) They execute with high speed.
(3) ----	(3) ----
(4) They do not provide many facilites at the hardware level.	(4) They are very close to the hardware and help to write a program at the hardware level.
(5) For writing programs hardware knowledge is not required.	(5) For writing programs hardware knowledge is a must

(6) The programs are easy to modify.	(6) Modifying programs is difficult.
(7) C++, Java, C#, python etc are examples of High-level Language.	(7) Machine Language and Assembly language are low-level Language.

C Language Introduction

C is a procedural programming language. It was initially developed by ~~Dennis Ritch~~ Dennis Ritchie between 1969 and 1973.

Beginning with C programming

1. Structure of a C program

The structure of a C programs is as follows:-

Basic structure of C Programs

Documentation section
Link section
Definition section
Global Declaration section
main() Function section
{
  Declaration Part
  Executable Part
}
subprogram section
  Function 1
  Function 2
  function 3
  -
  -
  -
  Function n

The components of the above structure are:

1. Header Files Inclusion:
⇒ The first and foremost component is the inclusion of the Header files in a C program.
A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.

- Some of C Header files.

(1) stddef.h
⇒ Defines several useful types and macros.

(2) stdint.h
⇒ Defines exact width integer types.

(3) stdio.h
⇒ Defines core input and output functions

(4) stdlib.h
⇒ Defines numeric conversion functions, pseudo-random network generator, memory allocation.

(5) string.h
⇒ Defines string handling functions

(6) math.h
⇒ Defines common mathematical functions

(7) conio.h
⇒ Defines console input output

Syntax to include a header file in C:

⇒ #include<(header_file_name).h>

2. main Method Declaration:
The next part of a C program is to declare the main() function. The syntax to declare the main function is:

Syntax to declare main method:
```
void main()
{
}
```

3. Variable Declaration:
The next part of any C program is the variable declaration. It refers to the variables that are to be used in the function. Please note that in C program, no variable can be used without being declared. Also in a C program, the variables are to be declared before any operation in the function.
Example:
```
void main()
{ int a;
}
```

4 Body:-
Body of a function C program, refers to the operations that are performed in the functions. It can be anything like manipulations, searching, sorting, printing, etc.

Example:
```
Void main()
{
  int a=10;
  printf("%d",a);
}
```

Writing first program!
Following is first program in C

```
#include<stdio.h>
Void main()
{
  Printf("The BCA");
}
```

C character set
As every language contains a set of characters used to construck words, statements etc, C language also has a set of character which include alphabets, digits and special symbols. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C Language character set contains the following set of characters.....

1. Alphabets
2. Digits
3. special symbols

Alphabets
C Language supports all the alphabets from english language. Lower and upper case letters together supports 52 alphabets.

lower case letters - a to z

UPPER CASE LETTERS - A to Z

Digits
C language supports 10 digits which are used to construct numerical values in C Language.

Digits - 0,1,2,3,4,5,6,7,8,9

Special symbols

C Language supports rich set of special symbols that include symbols to perform mathematical operations, to chek conditions, white spaces, beck spaces and other special symbols

SPECIAL CHARACTERS

~	tilde	=	equal to
%	percent sign	&	ampersand
|	vertical bar	$	dollar sign
@	at symbol	/	slash
+	plus sign	(	left parenthesis
<	less than	*	asterisk
_	underscore	\	~~right~~ back slash
-	minus sign	)	right parenthesis
>	greater than	'	apostrophe
^	caret	:	colon
#	number sign	[	left bracket
"	quotation mark	{	left flower brace
;	semicolon	?	question mark
]	right bracket	.	dot operator
!	exclamation mark	}	right flower brace
,	comma		

WHITESPACE CHARACTERS

\b	blank space	\\	Back slash
\t	horizontal tab	\'	Single quote
\v	vertical tab	\"	Double quote
\r	carriage return	\?	question mark
\f	form feed	\0	Null
\n	new line	\a	Alarm (bell)

Backslash charcter constants

character	ASCII value	Escape sequence	Result
Null	0000	\0	Null
Alarm (bell	007	\a	Beep sound
Back space	008	\b	Moves previous position
Horizontal tab	009	\t	Moves next horizontal tab

New line	010	\n	Moves next Line
Vertical tab	011	\v	Moves next vertical tab
Form feed	012	\f	Moves initial position of next Page
Carriage return	013	\r	Moves beginning of the line
Double quote	034	\"	Present Doubles quotes
Single quote	039	\'	Present Apostrophe
Question mark	063	\?	Present question Mark
Back slash	092	\\	Present back slash
Octal number	\0000		
Hexadecimal number	\x		

C tokens:

- Tokens are the smallest elements of a program, which are meaningful to the compiler.

- The following are the types of tokens; keywords, Identifiers, constant, Strings, operators etc.

- C tokens are the basic buildings block in C Language which are constructed together to write a C program.

- Each and every smallest individual units in a C program are know as C tokens.

C tokens are of six types, They are;

1. keywords (eg: int, while).
2. Identifiers (eg: main, total).
3. Constants (eg; 10, 20);
4. Strings (eg; "total", "hello").
5. special symbols (eg; (), {}),
6. operators (eg; +, /, -, *)

(1) keywords

There are total 32 keywords in C

Auto	double	int	Struct
Break	Else	Long	switch
case	Enum	register	Typedef
char	Extern	Return	Union
continue	For	Signed	Void
Do	If	static	while
default	Goto	sizeof	Volatile
const	Float	short	Unsigned

(2) Identifiers

Each program element in C programming is known as an identifier. They are used for naming of variables, functions, array etc.

Rules for naming C identifiers—

-> It must begin with alphabets or under-score.

-> Only alphabets, numbers, underscore can be used, no other special character, punctuations are allowed.

-> It must not contain white-space

-> It should not be a keyword.

-> It should be up to 31 characters long

Strings
A string is an array of characters ended with a null character (\0). This null character indicates that string has ended. Strings are always enclosed with double quote(" ").

Let us see how to declare String in C language:-

- char string[20] = {'S', 't', 'u', 'd', 'y', '\0'}
- char string[20] = "demo";
- char string[] = "demo";

Hierarchy of operator, operators precedene in C.

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

C-type casting

Type casting is a way to convert a variable from one data type to another data type, For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly using the cast operator as follows

```
#include <stdio.h>
void main () {
int sum =17, count=5;
double mean;

mean =(double) sum/count;
printf ("value of mean: %f\n",mean);
```

out put:-
Value of mean: 3.400000

Integer Promotion

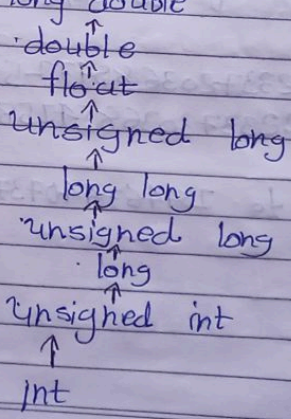
Integer promotion is the process by which values of integer type "smaller" than int or unsigned int converted either to int or unsigned int. Consider an example of adding a character with an integer-

```
#include <stdio.h>
void main ()
{
int i=17;
char c='c'; /* ascii value is 99 */
int sum;
sum = i+c
printf ("value of sum: %d\n", sum);
}
```

Output:
Value of sum = 116

Usual Arithmetic Conversion

The usual arithmetic conversions are implicitly performed to cast their values to a common type. The compiler first performs integer promotion; if the operands still different types then they are converted to the type that appears highest in the following hierarchy.



C- Data Types

Data types in c refer to an extensive system used for variables or functions of different type.

Integer Types

Type	storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 byte	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Floating-Point Types

Type	storage size	Value range	precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

An introduction to Flow charts

What is a Flow chart

Flowchart is a graphical representation of an algorithm. programmers often use it as a program planning tool to solve a problm.

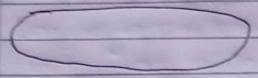
The process of drawing a flowchart for an algorithm is known as "flowcharting"

Basic symbols used in Flow chart Designs.

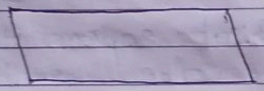
1. Terminal:

The oval symbol indicates start, stop and Halt in a program's logic flow. A pause/halt is generally used in a program logic under

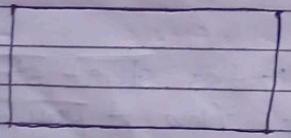
name symbol ..
some error conditions. Terminal is the first and last symbols in the flowchart.



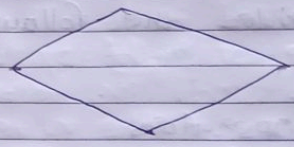
2. Input/output :-
A parallelogram denotes any function of input/output type. program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.



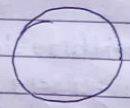
3. Processing :- arithmetic
A box represents instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.



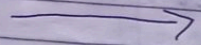
4. Decision :
Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.



5. Connectors :-
Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusios. It is represented by a circle.



6. Flow lines :-
Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.



C-OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C Language is rich in built-in operators and provides the following type of operators.

1. Arithmetic Operators
2. Relational operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Misc operators

[1] Arithmetic operators

Following table shows all the arithmetic operators supported by c language. Assume variable A holds 10 and variable B holds 20 then.

operator	Description	Example
+	Adds two operands	A+B will give 30
-	subtracts second operand from the first	A-B will give -10
*	Multiplies both operands	A*B will give 200
/	Divides numerator by de-numerator	B/A will give 2
%	Modulus operator and remainder of after an integer division	B%A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

[2] Relational operators

Following table shows all the relational operators supported by c Language. Assume variable A holds 10 and variable B holds 20, then.

operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true	A==B is not true
!=	Checks if the values of two operands are equal or not; if value are not equal then condition becomes true.	A!= is true

>	checks if the value of left operand is greater than the value of right operand if yes then 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 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 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 condition becomes true	A<=B is true

(3) Logical operators
following table shows all the logical operators supported by c language. Assume variable A holds 1 and variable B holds 0, then.

operator	Description	Example
&&	Called logical AND operator. If both the operands are non-zero, then condition becomes true	A&&B is false
||	Called Logical OR operator. If any of the two operands is non-zero, then condition becomes true	A||B is true
!	Called Logical NOT operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will Make false	!A&&B is true

Dry-ran = c trubo F7 ઉપયોગ થાય લાઈન-બાય લાઈન dibginary નો Dry-run કહેવાય છે.

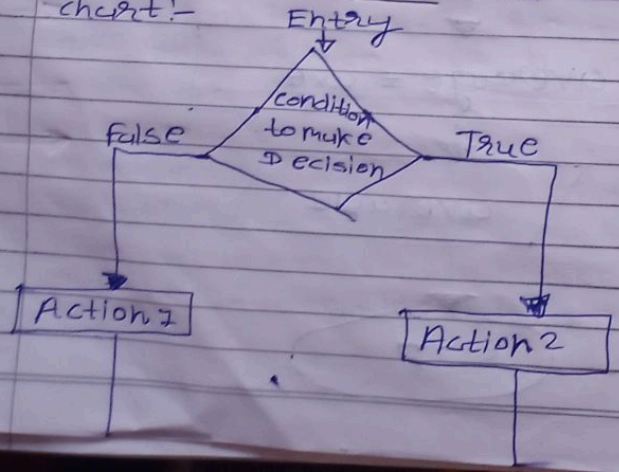
Selective control structure
- if statements
- Switch statement

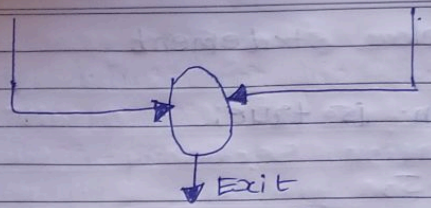
Syntax
The syntax an if statement in c programming language.

If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed. If the Boolean expression evaluates of false then the first set of code after the end of 'if' statement (after the closing curly brace) will be executed.

C programming language assumes any non-zero and non-null values as true and if it is either zero or null, then it is assumed as false value.

chart:-





selection Structure

Implemented using:- if and if... else control statements
Switch is used for Multi branching

* C If statement

1. Expression is true,

```
int test =5;
if (test <10)
{
  //codes
}
  // codes after if
```

2. Expression is false;

```
int test =5;
if (test >10)
{
  //codes
}
//codes after if
```

PAGE NO.
DATE

C if.... else statement

Expression is true;

```
int test =5;
if (test <10)
{
  // body of if
}
else
{
  // body of else
}
```

Expression is false;

```
int test =5;
if (test>10)
{
  // body of if
}
else
{
  // body of else
}
```

if... else Ladder (if... else if... else statement)

- Syntax of nested if... else statement.

```
if (test Expression1)
{
  //statement (s)
}
else if (test Expression2)
{
  //statement(s)
}
else if (test Expression 3)
{
  // statement(s)
}
.
.
else
{
  //statement(s)
}
```

Example 3: C if... else Ladder

```
//Program to relate two intergers using
// =, >or<

#include <stdio.h>
int main()
{
  int number1, number2
  printf ("Enter two integers:");
  Scanf("%d %d", &number1, &number2);
  // checks if two integers are equal.
  if (number 1 == number 2)
  {
  printf ("Result: %d = %d", number1, number2);
  }
  // checks if number 1 is greater than
    number 2
    else if (number1 > number 2)
  {
   Printf ("Result: %d > %d", number 1, number 2);
  }

  // if both test expression in false
  else
  {
   Printf ("Result: %d < %d;" number1, number2);
  }
   return 0;
}
```

Output

```
Enter two integers: 12
23
Result: 12<23
```

* The if... else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilites.

The if... else ladder allows you to check for multiple test expressions and execute different statement(s).

Syntax of nested if... else statement.

```
if (testExpression1)
{
  //statement(s)
}
else if (test Expression2)
{
  //statement(s)
}
else if (test Expression3)
{
  //statement(s)
}
.
.
.
else
{
  // statement(s)
}
```

Nested if... else

It is possible to inclue if... else statement(s) inside the body of another if... else statement.

This program below relates two integers using either <, > and = similar like in if... else ladder example. However, we will use nested if... else statement to solve this problem.

C. Switch... case statement

The if... else ladder allows you to execute a block code among many alternatives. If you are cheking on the value of a single variable in if... else ... if, it is better to use switch statement.

The switch statement is often faster than nested if... else (not always). Also the syntax of switch statement is cleaner and easy to understand.

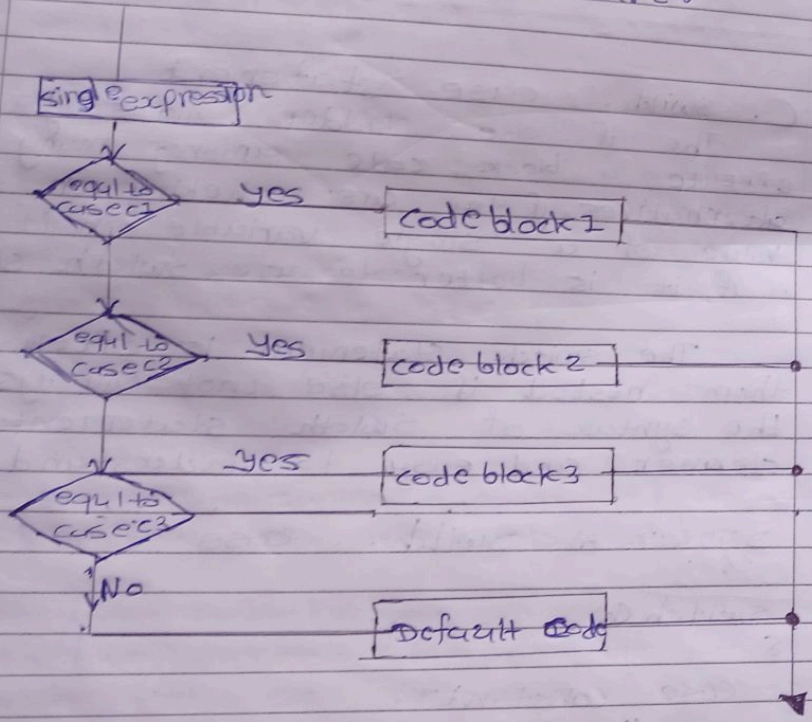
Syntax of switch... case

=>

```
switch (n)
{
  case constant1:
    // code to be executed if n is equal
    to constant 1;
    break;

  case constant 2:
    // code to be executed if n is equal
    to constant 2;
    break;
```

default:
// code to be executed if n doesn't match any constant
}

Switch Statement Flowchart :-



The ? : Operator

name :- Makwana Ritam P.
GR No :- 2020BCA67
Roll No :- 1072

⇒ We have covered conditional operator ? : in the previous chapter which can be used to replace if....else statements. It has the following general form -

Exp 1 ? Exp 2 : Exp 3;

```
x = 50;
y = 60;
x > y ? x : y;
```

output
60

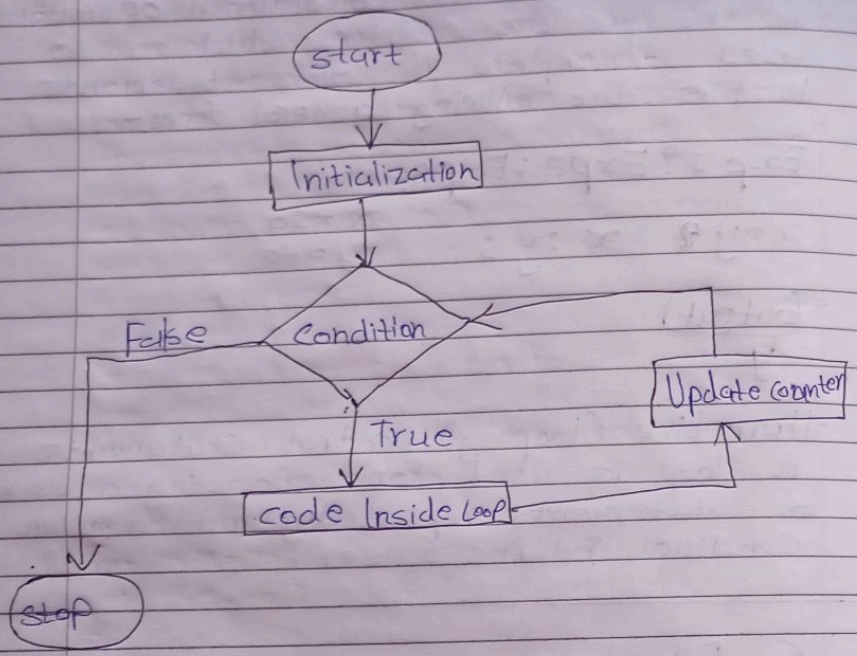
Iterative (looping) control statements
A loop is used for executing a block of statements repeatedly until a given condition returns false.

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

Syntax of for loop :

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

name :- Makwana Pritam P.
GTRNo :- 2020BCA67
Roll No :- 1072



Step 1: First initialization happens and the counter variable gets initialized.

Step 2: In the second step the condition is checked, where the counter variable is tested for the given condition if the condition returns true then C statements inside the body of for loop gets executed. if the condition returns false then the for loop gets terminated and the control comes out of the loop.

name :- Makwana Pritam P.
GTRNo :- 2020BCA67
Roll No :- 1072

step:-3
After successful execution of statements inside the body of loop, the counter variable is incremented or decremented, depending on the operation (++ or --).

✱ Various forms of for loop in C

I am using variable num as the counter in all the following examples.

(1) Here instead of num++, I'm using num = num+1 which is same as num++

```
for (num=10; num<20; num=num+1)
```

(2) Initialization part can be skipped be skip. from as shown below, the counter variable is declared before the loop.

```
int num=10;
for (;num<20;num++)
```

Note: Even though we can skip initialization part but semicolon (;) before condition is must, with which you will get compilation error.

name:- Makwana Pritam P.
GR No:- 2020BCA67
Roll No:- 1072

3) Like initialization, you can also skip the increment part as we did below, In this case semicolon (;) is must after condition logic. In this case the increment or decrement part is done inside the loop.

```
for (num=10; num<20;)
{
  //statements
  num++;
}
```

(4) This is also possible. The counter variable is initalized before the loop and incremented inside the loop.

```
int num=10;
for(;num<20;)
{
  //statement
  num++;
}
```

(5) As mentiond above, the counter variable can be decremented as well. In the below example the variable gets decremented each time the loop runs until the condition num>10 returns false.

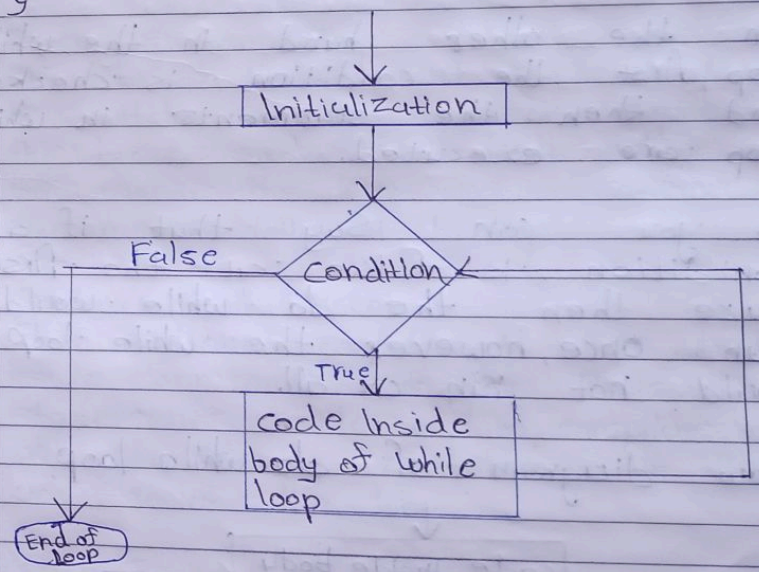
```
for (num=20; num>10; num--)
```

name:- Makwana Pritam P.
GR No:- 2020BCA67
Roll No:- 1072

C - while loop

Syntax of while loop:

```
while (condition test)
{
  //statements to be executed repeatedly
  //Increments (++) or Decrement (--)
    operation
}
```



C - do... while loop

syntax of do-while loop

```
do
{
  //statements
} while (condition test);
```

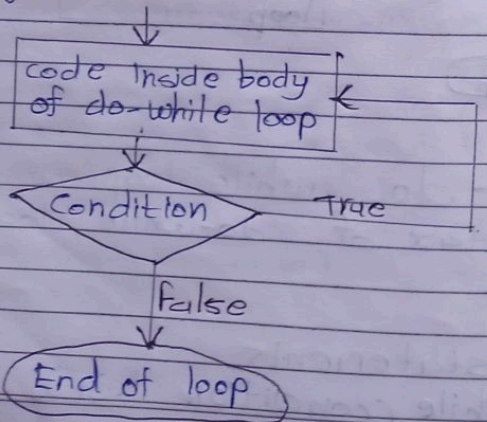
→ In the previous tutorial we learned while loop in C.

→ A do while loop is similar to while loop with one exception. that it execties the statements inside the body of do-while before cheking the condition.

→ On the other hand in the while loop, first the condition is checked and then the statements in while loop are executed.

→ So you can say that if a conidition is false at the first place then the do while would run once, however the while loop would not run at all.

→ Flow diagram of do while loop



✱ Jump statements in C

Jump statements are used to interrupt the normal flow of program.

- Type of jump statements
1. Break
2. continue
3. GoTo

1. Breck statement
The break statement is used inside loop or switch statement. when compiler finds the break statement inside a loop, compilar will abort the loop and continue to execute statements followed by loop.

2. Continue statement
The continue statement is also used inside loop. when compiler finds the break statement inside a loop, compilar will skip all the following statements in the loop and resume the loop.

3. Goto statement
The goto statement is a jump statement which jumps from one point to another point within a function.

Syntax of goto statements

```
goto. label;.
--.-----
--------
label.
-------
-------
```

In the above syntax, lubal is an identifer. when, the control of program reaches to goto statement, the control of program will jump to the label: and executes the code after it.

Unit-3

Unit-3

(1) Strcpy

-> strcpy() is a standard library function in c/c++ and is used to copy one string to another In c it is present in string.h header file and c++ it is present is cstring header file.

Syntax:-

```
char* strcpy (char* dest, const char* src);
```

Paramters :- This method accepts following paramters:

- dest: Pointer to the destination array where the content is to be copied.
- src : string which will be copied.

Return Value: After copying the source string to the destion string, the strcpy() function returns a pointer to the destination string.

(2) strncpy() function

-> The strncpy() function is similar to strcpy() function, except that at most n bytes of src are copied.

-> If there is no Null character among the first n character of scr, the string placed in dest will not be NULL terminated.

→ If the length of scr is less than n, strncpy() writes additional Null character to dest to ensure that total of n character are written.

Syntax:

char * strncpy (char* dest, const char* src, size.

parameters:- This function accepts two parameters as mentioned above and described below.

- Src: This string which will be copied.
- dest: pointer to the destination array where the content is to be copied.
- n: This first n characchar copied form src to dest.
- Return Value: It returns a pointer to the destination string

(3) strcat() function

→ strcat() function will append a copy of the source string to the end of destination string.

→ The strcat() function takes two arguments:

(1) dest

(2) src

→ It will append copy of source string in the destination string. The terminating character at the end of dest is resplaced by the first character of src.

- Retun Value:-
  The strcat() function returns dest, the pointer to the destination string.

[4] Strncat() function

→ strncat is a predefind function used for string handling. string.h is the header file required for string functions.

→ This function appends not more than n character from the string pointed to by src to the end of the string pointed to by dest plus a terminating Null-characchar.

→ The initial character of string (src) overwrites the Null-character present at the end of string (dest).

→ Thus, length of the string (dest) becomes strlen (dest)+n. But, if the length of the string (src) is less than n, only the content up to the terminating null-character is copied and length of the string (dest) becomes strlen (src)+strlen(dest).

The behavior is undefined if
-> the string overlap.
-> the dest array is not large enough to append the contents of src.

Syntax:

char* strncat (char*dest, const char *src, size_t n)

Parameters: This method accepts following paramters.

- dest: the string where we went to append.
- src: the string from which 'n' character are going to append.
- n: represents maximum number of charater to be appended. size_t is an unsigned integral type.

Return Value:- The strncat() function shall return the pointer to the string (dest).

(5) Strchr() functions
The function strchr() searches the occurrence of a specified character in the given string and returns the pointer to it.

Syntax:

char* strchar (const char* str, int ch)

str - The string in which the character is searched.

ch - The character that is searched in the string str.

(6) strrchr():
syntax:-
const char* strrchr (const char* str, int ch)
char* strrchr (char* str, int ch)

Parameter: The function takes two mandatory parameters which are described below.

- str: specifies the pointer to the null terminated string to be searched for.
- ch: specifies the charater to be search for.

Return Value: The functions returns a pointer to the last location of ch in string. if the ch is found, it not, it returns a null pointer.

(7) strcmp() function

Syntax;

char*strcmp (const char* leftstr, const char* rightstr);

-> strcmp() compares the two strings lexicographically means it stars comparison character by character starting from the first character until the characters in both strings are equal or a NULL character is encounted.

Zero (0);
-> A values equal to zero when both strings are found to be identical. That is, That is all of the characters in both strings.

Greater than zero (>0); A value greater than zero is returned when the first not matching character in leftstr have the greater Ascll value than the corresponding character in rightstr or we can also say If character in leftstr is lexicographically. ~~after the character of rightstr.~~

Less than zero (<0): A value less than zero is returned when the first not matching character in leftstr have lesser Ascll value than the corresponding character in rightstr. If character or rightstr.

[8] strncmp() function

Syntax:

int strncmp (const char* str1, const char* str2, size_t count);

Paramaters:
str1 and str2: C string to be compared.
count; maximum number of characters to compare.
size_t is an unsigral type.

Return value

Value	Meaning:
Less than zero	-> str1 is less than str2
zero	-> str1 is equal than str2
Greater than zero	-> str1 is greater than str2

[9] Strspn()

Syntax;
size_t strspn (const char *str1, const char * str2)

str1: string to be scanned.
str2: string containing the character to match.

Return value:- This function returns the numbers the number of characters in the initial segment of str1 which consist only of characters from str2.

[10] strcspn ()

The c library function strcspn() calculates the numbers of characters before the 1st occurrence of character present in both the string.

Syntax:

strcspn(const char* str1, const char* str2)

[11] strlen ()

function in c gives the length of the given of the string. Syntax for strlen() function is given below.

size_t strlen (const char* str);

12] strpbrk () in c

This function finds the first character in the string s1 that matches any character specified in s2 (it excludes terminating null-characters).

Syntax;

char * strpbrk (const char * s1, const char * s2)

[13] strstr ()

strstr() is a predefined function used for string handing. string.h is header file required for string functions.

This function takes two string s1 and s2 as an argument and finds the first occurrence of the sub-string s2 in the string s1.

[14] strstr strtok:

C provides two functions strtok() and strtok_r(). strtok_r() for splitting a string by some delimiter. splitting a string is a very common task. for example, we have a comma separated list of items from a file and we want individual items in an array.

Syntax;

char * strtok (char str [], const char * delims);

Unit 3

C Pointer

name:-Makwana Pritam P.
UR No:-2020BCA67
Roll No:-1072

PAGE NO.
DATE

Q7] What is pointer? Explain with one example:-

→ 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:-

1. int n=50

2. int *p= &n; // Variable p of type pointer is pointing to the address of variable n of type integer.

Declaring a pointer:-

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

%x = Hexadecimal
%x = p
%d = *p

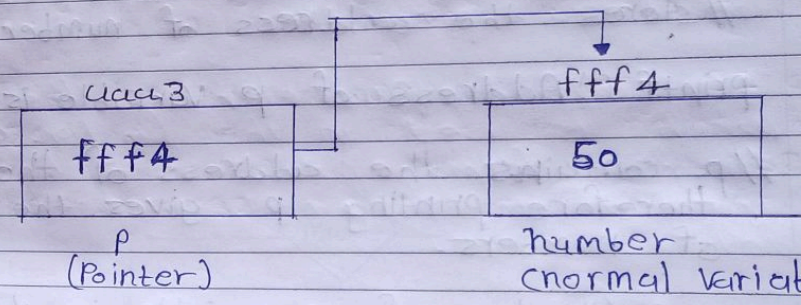
name:-Makwana Pritam P.
UR No:-2020BCA67
Roll No:-1072

PAGE NO.
DATE

1. int * a; // pointer to int

2. char * c; // pointer to char

Pointer a pointer:-
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 varible, i.e., ffff4. 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.

name:- Makwana Pritam P.
GRNo:- 2020BCA67
RollNo:- 1072

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

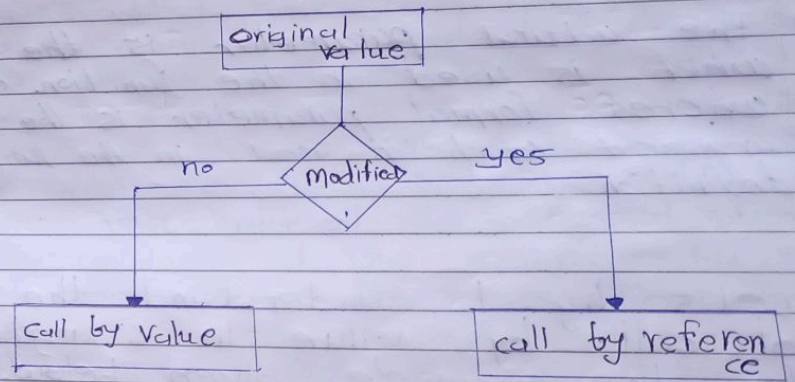
```
#include<stdio.h>
Void main()
{
int number=50;   //fff4
int *p;          //vvv2
p=&number;       //vvv2=fff4
// stores the address of number varible

printf("Address of p variable is %x \n",p);

//p contains the address of the number therefore printing p gives the address of numbers.

printf("Value of p variable is %d \n",*p);

// As we know that * is used to dereference a pointer therefore if we printf* p, we will get the value stored at the address contained by p.
}
```

OUTPUT

```
//Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```


Q-12 ☆ Example of function call by value:- call by value and call by reference

→ There are two methods to pass the data into the function in c language. i,e, call by value and call by refernce.



Call by value in C

→ In call by value method, the value of the actual parameter is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.

name:- Makwan Pritam P.
GRNo:- 2020BCA67
Roll No:- 1072

PAGE NO.
DATE

-> In ~~ct~~ call by value method, we can not modify the value of the actual parameter by the formal parameter.

-> In call by value, different memory is allocated for actual and formal parameters since the value of the ~~act~~ actual parameter is copied into the formal parameter.

-> The actural parameter is the argument which is used in the function call wheres formal parameter is the argument which is used in the function definition.

-> Let's try to understand the concept of call by value in c language by the example given below.

```
#include<stdio.h>
void change (int num){
printf ("Before adding value inside function
        num=%d\n", num);
num=num+100;
printf ("After adding value inside funcion
        num=%d\n", num);
}
int main (){
int x=100;
printf("Before function call x=%d\n",x);
change(x);// passing value in function
printf ("After function call x=%d\n",x);
return 0;
}
```

PAGE NO.
DATE

OUTPUT

```
Before function call x =100
Before adding value inside function num=100
After adding value inside function num=200
After function call x =100
```

Call by reference in c

-> જો oringan માં chage આવે તો call refercе કહેવાય છે
-> swap ઉપર લખેલુ ના હોય અને void main
નીચે લખેલુ હોય તો ઉપર swap(int, int) value
(Prototype) આપવા હોય તેટલી વાર int મુકવામાં આવે છે.

-> In ~~ct~~ call by reference the address of the variable is passed into the function call as the actual parmeter.

-> The value of the actual parameter can be modified by changing the formal parameters since the address of the actual parmeters is passd.

-> In call by reference, the memory allocation is similar for both formal parameter

and actual parameters. All the operations in the function are performed on the value stored at the address of the actual paramater, and the modified value gets stored at the same address.

Recursion :-

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

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

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

→ The C programming language supports recursion, i.e, a function to call itself. But while using recurison, programmers need to be carful define an exit condition from the function, otherwise it will go into an infinite loop.

→ Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc

Array

PAGE NO.
DATE

STRING array

→ In c programming, a string is a sequence of characters terminated with a null character \0. for example

```
char c[] = "c string";
```

→ When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.

c		s	t	r	i	n	g	\0

How do declare a string?

Here's how you can declare strings:

```
char s[5];
```

s[0]	s[1]	s[2]	s[3]	s[4]

Here, we have declared a string of 5 characters.

How to initialize strings?

→ You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[50]="abcd";
char c[] = {'a','b','c','d','\0'};
char c[5] = {'a','b','c','d','\0'};
```

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

Let's take another example

```
char c[5] = "abcde";
```

→ Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters.

→ This is bad and you should never do this.

Example

```
#include<stdio.h>
int main()
{
  char name[20];
  printf("Enter name:");
  scanf("%s",name);
  Printf("Your name is %s.",name);
  return 0;
}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

- Two dimensional (2D) arrays in C programming with example

→ An array of arrays is known as 2D array. The two dimensional (2D) array is in programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array lets have a look the following C program.

	D1	D2	D3
S1	10	20	45
S2	100	500	600

- Simple Two dimensional (2D) Array Example

→ For now don't worry how to initialize a two dimensional array, we will discuss that part later. This program demonstrates how to store the elements entered by user in a 2d array and how to display the elements of a two dimensional array.

→ multi Dimensional Array Example is PracticalBook

Multi demenational array (three)

Example

name [2] [2] [3]
Row Row Colem

1	2	3
4	5	6

2X3

7	8	9
10	11	12

2X3

Unit 4

Structure

What is a structure?

→ A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

→ Arrays allow to define type of variables that can hold several data items of the same kind. similarly structure is another user defined data type available in C that allows to combine data items of different kinds.

→ Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book -

- Title
- Author
- Subject
- Book ID

```
Int age[50];

Age[0] = 23;

Age[1] = 34;
```

How to create a structure?
'struct' keyword is used to create a structure. following is an example.

```
struct address
{
    char  name [50];
    char  street [100];
    char  city [50];
    char  state [20];
     int  pin;
};
```

How to declare structure variables?

→ A structure variable can either be declared with structure declaration or as a separate declaration like basictypes.

```
// A variable declaration like basic datatypes
struct Point
{
    int x,y;
};

int main()
{
struct Point p1; // The variable p1 is declared
 like a normal variable
}
```

How to initialize structure members?

→ Structure members cannot be initialized with declaration. for example the following C program fails in compilation.

```
struct Point
{
 int x= 0; // COMPILER ERROR: cannot initlize
 int y= 0; // COMPILER ERROR: cannot initlize
};
```

→ structure members can be initialized using curly braces '{}'. for example, following is a valid initialization.

```
struct point
{
  int x,y;
};
int main()
{
// A valid initialzation. member x gets value
  0 and y
// gets value 1. The order of declaration
  is followed.
  struct point p1={0,1};
}
```

How to access sturucture elements?

⇒ Structure members are accessed using dot (.) operator.

PAGE NO.
DATE

What is an array of structures?

→ Like other primitive data types, we can create an array of structures.

What is a stucture pointer?

=> Like primitive type, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator.

→ start Teory:-

- Type of User-defined Functions in c

→ There can be 4 different type of user-defined functions, they are:-

1. Function with on argument and no return value.
2. function with on argument and a return value
3. function with argruments and no return value
4. function with argruments and a return value

1. function with no arguments and no return value:-

→ Such functions can either be used to ~~display~~ display information to they are completely depend on user inputs:-

→ Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

2. function with no arguments and a return value:-

→ We have modified the above example to make the function greatNum() return the number which is greater amongst the 2 input numbers.

3. Function with arguments and no return value:-

→ We are using the same function as example again, to demonstrate that to solve a problem there can be many different ways.

4. Function with arguments and a return value

→ This is the best type, as this makes function complely indepent of inputs and outputs, and only the logic is defined inside the function body.

• Nested structure in c

→ c provides us the feature of nesting one structure within another structure by using which complex data types are created.

→ for example, we any may need to store the address of an ~~one~~ entity employee in a structure.

→ The attribute address may also have the subparts as street number, city, state, and pin code.

→ Hence, to store the address ~~to~~ of the ~~empol~~ employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee. Consider the following program.

☆ Union in c

• Like structures, union is a user defined data type. In union, all members share the same memory location.

How do define a union?

→ We use the union keyword to define unions Here's an example.

```
union car
{
    char name[50];
    int price;
};
```

Create union variables

→ When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

→ Here's how we create union variables.

```
union car
{
    char name[50];
    int price;
};

int main()
{
union car car1, car2, *car3;
return 0;
}
```

→ For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

o Structure vs. Union

Structure	Union
1. struct keyword is used to define a ~~stru~~ structure.	Union keyword is used to define a union.
2. members do not share memory in a structure	members share the memory space in a union.
3. Any member can be retriered at any time in a structure	only one member can be accessed at a time in a union
4. Several members of a structure can be initialized at once.	Only the ~~first~~ first member can be initilized.
5. size of the structure is equal to the sum of size of the each member.	size of the union is equal to the size of the largest member.
6. Altering value of one member will not affect the value of another.	change in value of one member will affect ~~another~~ other member values.
7. Stores different ~~value~~ values for all the members.	stores same value for all the members

Unit-5

Pointer and Arrays in C

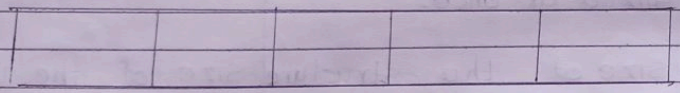
PAGE NO.
DATE

→ When an array is declared, compiler allocates ~~allo~~ sufficient amount of memory to contain all the elements of the array. Base address i.e address of the first elements of the array ~~is~~ is also allocated by the compiler.

Suppose we declare an array arr;

int arr[5] = {1,2,3,4,5};

→ Assuming that the base address of arr is 1000 and each integer requires two bytes, the five elements will be stored as follows:-



element:-	arr [0]	arr [1]	arr [2]	arr [3]	arr [4]
Address	1000	1004	1008	1012	1016

→ Note that there is difference of 2|4 bytes between each element because that's the size of an integer.

→ Which means all the elements are stored in consecutive contiguous memory locations in the memory.

PAGE NO.
DATE

arr is equal to &arr[0] by default

→ We can also declare a pointer of type int to point to the ~~arry~~ array arr.

```
int *p;
p = arr;
 //or,
p = &arr[0]; //both the statements are
                equivalent.
```

→ Now we can access every element of the array arr using p++ to move from one element to another.

→ Note:- You cannot decrement a pointer once incremented. p-- won't work.

• Pointer to Array:-

→ As studied above, we can use a pointer to point to an array, and then we can use that pointer to access the array elements. Let's have an example:-

→
```
//Pointer to array
#include<stdio.h>
#include<conio.h>

Void main()
{
```

```
int i;
int a[5] = {1,2,3,4,5};
int *p = a;

for (i=0; i<5; i++)
{
  Printf("\n %d", *p);
  Printf("\n Add %x", p);
  p++;
}
getch();
}
```

Output

```
1
Add ffec
2
Add ffee
3
Add fff0
4
Add fff2
5
Add fff4
```

- Pointer to Arrays of structures in C

→ Like we have array of integers, array of pointers, we can also have array of ~~struc~~ structure variables.

→ And to use the array of structure variables efficiently, we use Pointers of ~~struct~~ structure type. We can also have pointer to a single structure variable, but it is mostly used when we are dealing with array of structure variables.

→ Accessing structure Members with pointer

→ To access members of structure using the structure variable, we used the dot operator. But when we have a pointers of structure type, we use arrow -> to access structure members.

→ This example is practical book in.

- Pointer to Pointer

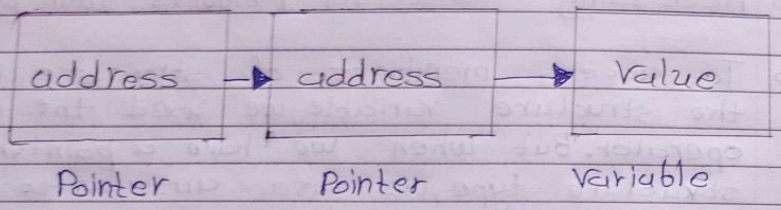
→ As we know that, a pointer is used to store the address of a variable in C.

→ Pointer reduces the access time of a variable.

→ However, In C, we can also define a pointer a pointer to store the address of another pointer.

→ ~~The~~ such pointer is know as a double Pointer (Pointer to pointer).

→ The first pointer is used to store the address of variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.

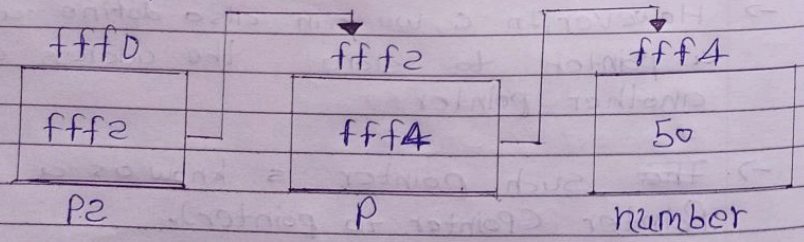


→ The syntax of declaring a double pointer is given below.

1. int **P; // Pointer to a pointer which is
// pointing to an integer.

c double pointer example:-

→ Let's see an example where one pointer points to the address of another pointer.



example:-

```
int i=50;   //i address fff4 value 50
int *p=&i;  //*p address fff2 value fff4
int **p2=&p //**p2 address fff0 value fff2
```