"C"

Rs 2100/2

Definition of Software :-

A s/w is a collection of prgms (set of instructions) which uses the resources of the H/w components.

As per the Experts s/w are classified in 3 categeories.

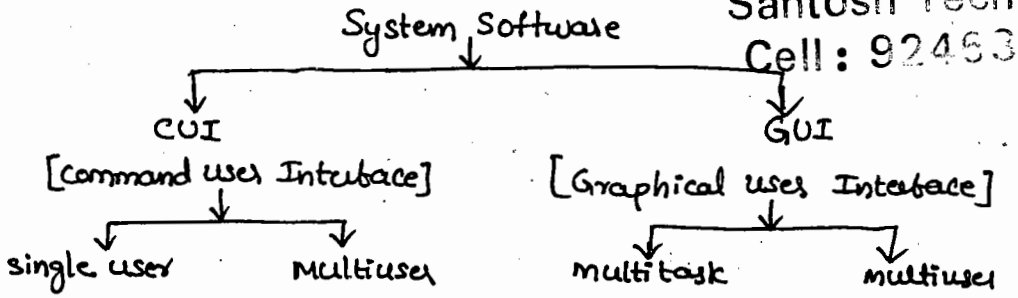
1. System s/w. 2. Application s/w. 3. Internet s/w.

1. System s/w :-

The system s/w is a s/w which does the functionality for the H/w devices, like printers, mobile, processors ... etc.

To develope this system s/w we need of Three languages.

1. Assembly level language [microprocessor Instruction]
2. C-language [procedure Oriented programming]
3. C++ [Object oriented programming]



Single User :-

The processor can does only one job at a time is called as Single user.

Eg:- MS-DOS

Multi User :-

More than one user can use the machine at a time is called as Multiuser

Eg:- UNIX

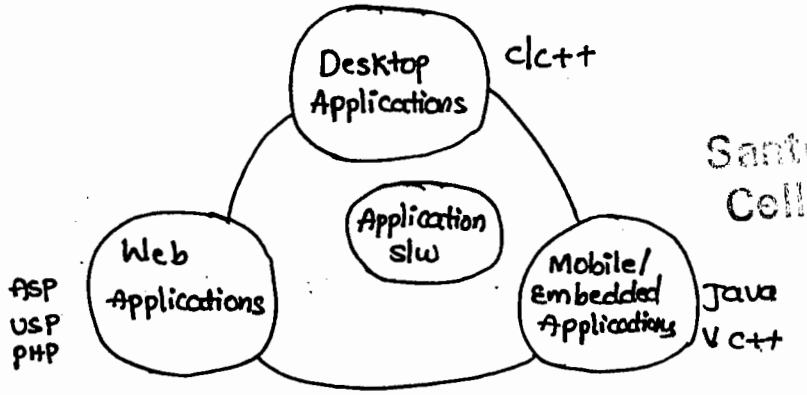
Multi Task :-

more than one (task) job is taking place at a time is called as multitask.

Eg:- Win 95/97/98/NT/2000/me/2003/xp/vista/7/8, Andriod, IOS.

2. Application s/w :-

An App^n s/w is a s/w which does the functionalities for the business oriented applications.



Database:

Every application slw requires database like dbase, foxpro, oracle, SQL--- etc.

programming languages:-

A lang. can be used for the communication purpose.

A computer lang's will be used to communicate with the user & the System.

This Computer languages are classified in Two categories.

1. Low level languages. 2. High level languages.

1. Low Level Languages (LLL's):

LLL's are the lang's which can be easily understandable to the System. These are System dependent languages. In this two lang's are,

1. Machine language. 2. Assembly language.

Machine languages:-

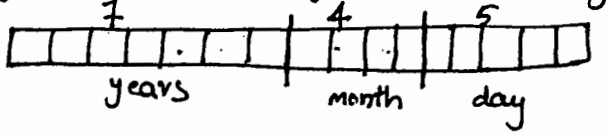
It is the fundamental language for the System it can directly can be understandable without any translation. These are machine-oriented lang's that use collection of binary of 1's and 0's. As w.k.t, computer can understand many language without translation there is only 1 lang. that is binary language.

As the users prefer decimal no's we define to write the decimal no's but System will convert into binary no's.

The Computer measuring units bits are bytes.

Advantages:-

1. As the computer lang. is the machine-oriented it can be understandable very easily & very fast Execution process.
2. writing in machine lang. lot of memory can save.



31 – 10001

Disadvantages:-

1. Remembering dozens of binary code is not an easy job
2. Rectifying the Errors & debugging process is a time taking.
3. modifying the prgm is not easy.

Assembly language:-

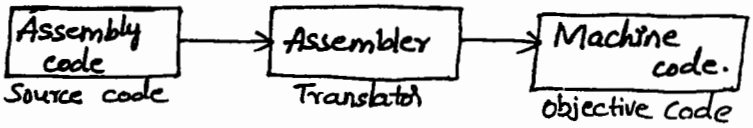
The 2nd generation of lang. is implemented by the assembly code. This lang. can be called as Symbolic language. In order to remember easily the prgmng coding be implementing this lang. In this lang. diff. types of symbols will be used to design the prgmng. But this assembly code can't understandable to the System.

The assembly code directly not understandable to the system so we require Translators. They are 3types.

1. Assembler 2. Interpreter 3. Compiler. (for high level coding)

Assembler: It is a translator which converts the assembly code into the Machine code.

Eg:- 8086



Advantages:

* it is Easy to write the prgm compare to machine lang.
* it is Easy to locate & correct the Errors.

Disadvantages:-

* To learn the assembly lang. we need the knowledge of HLL.

High level languages:-

HLL's are the lang's which are Easy to understandable for the user. They are user dependent lang's.

The HLL's is a combination of alphabets, digits & Symbols. It is called as Macro statements. It is very Easy lang. Since it is general English lang. But the System will not understandable this lang's, for this reason we have Translators.

Interpreter:-

An Interpreter is a translator which converts from HLL's to Machine code by checking the prgm line by line.

Eg:- Scripting lang's
[Javascript, Vbscript, PHP, ... etc.]

Compiles :-

It is a translator which converts the HLL into the machine code [objective code] by checking the whole prgm at a time.

Eg:- C & C++.

This Translators are not a physical components. They are also software [system s/w].
[Very fastest translator is compiler].

Advantages :-

* it is very easy to understandable to write the prgmng in HLL's.
* it is easy to debug the prgmng code.

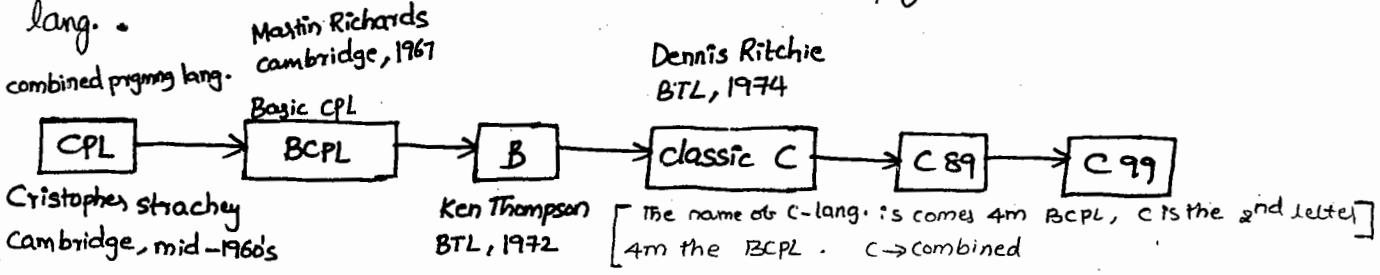
Disadvantages :-

* It takes more memory to implement the applications.
* Machine & assembly lang's are more efficient than HLL's.

History of C-language :-

Around in 1960's the 2 most imp. prgmng lang's COBOL [common business Oriented lang.] for Commerical app's and FORTRAN [formula Translation] for scientific & Engg. App's. [design launchings, development of Engg. calculators ---- Etc].

These many apps can be developed by using this lang. but this lang's can be used only to develope specific app's. for this reason the prgmrs desired to develope a new lang.



we are following ANSI-C [American National standard Institute of C].
we have One more 'C' i.e; K&R-C [kernighan & Ritchie].

Characterstics of 'C' :-

* C is an mid-level prgmng lang.
* As WKT, we have low level prgmng lang's in which the System can understand Easily. Eg:- Micro processors.
And we have HLL's in which users can understand Easily.
Eg:- COBOL, FORTRAN.
And 'C' is a combination of low level & high level prgmng. That's y it is called as Mid level prgmng.
Eg:- C & C++.

* C supports 44 operators, 32 keywords & 14 separators.
* C is an Case-Sensitive lang.

Case-Sensitive means the lower case char & the upper case char. has the diff. meaning.

* Every C statement Ends with an Semicolon (;)
* The C lang supports large set of library functions.
* C is an fun^n oriented, procedure oriented & structure Oriented prgmng lang.

The Source code what we implemented in c-lang. it is totally depends on fun'al structures for this reason it is called as fun'al oriented prgmng.

Deviding the prgm into small modules according to its operations is called as procedure Oriented prgmng.

A structure Oriented prgmng has to Satisfy 3 Criteria's.

1. Sequence of Steps. 2. Decision making 3. Repetetion.

Sequence of steps :-

Executing Every line of the statement without ignoring any of the statements such type of prgmng is Sequence of steps.

Decision Making :-

Every time sequence of prgmng is not suitable, that time we use the Conditional statements in this prgmng based on the condition some statements will be Executed. Some statements will be ignored.

Repetition :-

Executing the statements more than one time is called as Repetition. Inorder to implement the Repetitions we have Iteations Concept(Loopings).

Applications of C-language :-

* C is used to develope a System slw app's like unix, windows, linux, Compiler designings.... etc.
* It also used in development of many app's slw's like Commercial products
* it is used in development of Embedded & mobile apps.

CDMA phones depends on the BREW Technology.

BREW → Binary Runtime Environment Wireless.
developed using C with Java.

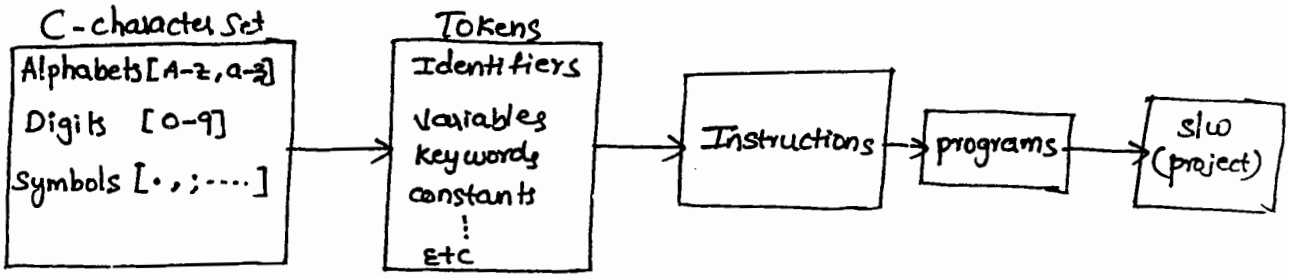
* Gaming framework are also written using c lang.

DIRECT X
↕
COM/DCOM [component/Distributed component object model]
↕
VC++
↕
C++
↑
C

* C lang. much Supports good interaction with H/w.

By using c-lang. we can develope another prgmng lang's also like PHP, micro Soft. foundation classes, database prgmng ---- Etc.

How to write a C-program :-



C-character Set :- A C-char set consist of alphabets [lower & upper], digits [0-9] & Special Symbols.

Tokens :- A Token is an small unit of a prgm. Token consist of identifiers, keywords, constants, variables, datatypes, operators, & special Symbols.

Identifiers :- A name which is given for variables, constants, macros ---- etc is called as Identifiers [user defined]. [case-in sensitive].

Keywords :- keywords are the reserved words in which the meaning already defined by the compiler. All the keywords has to write in lower case. According to the ANSI c we have 32 keywords. according to K&R 'c' we have 27 keywords.

Eg:- for, do, int, goto, enum ---- etc.

[exactly opp. to the Identifiers].

Constants :- A constant is a value that doesn't change during the Execution of the prgm.

constants are of 2 types.

1. Numerical constants.
2. character constants.

Numerical Constants:

The Numerical constants are the number type. These are 2 types.

1. Integer constants 2. Real constants.

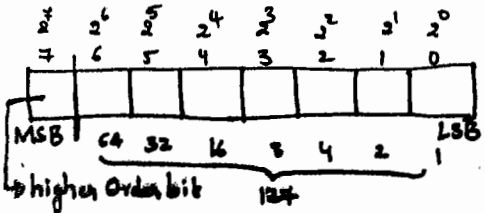
Integer constants:-

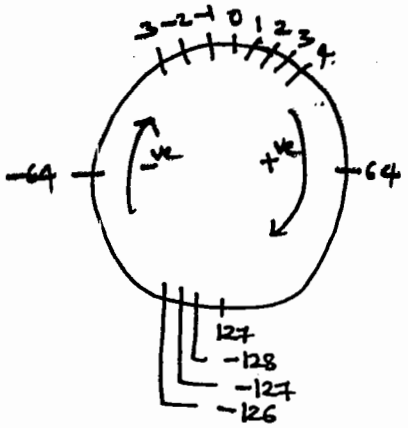
This constants are the decimal type numbers. It can be $\oplus^{ve}$ or $\ominus^{ve}$. by default it is $\oplus^{ve}$ no's. In blw the decimal no's we do not have any separators including comma (,) also.

Eg:- 45 99 -33 3,967 --- etc.

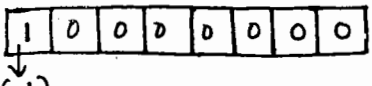
3967 is not valid, bcoz it is comma is present in that value.

The min. range of the integer constants −32,768 to 32,767.





for least:

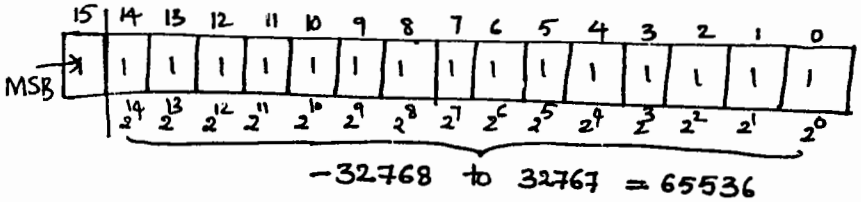


$2^7 = -128$ → least.

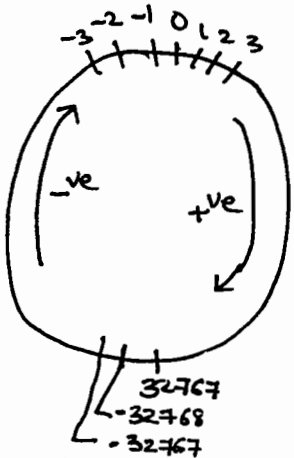
proof:-

$$\begin{array}{r} 01111111 \\ 11111111 \\ \hline 10000000 \end{array} = -128$$

There is NO $\ominus^{ve}$ value binary code in the System.



bytes ↑seq ↔ Range ↑seq



In Integer Categeory, we can also have octal numbers and Hexadecimal no's. The Octal no's will be indicated by prefix with "0".

Eg:- $0^2 8^1 8^0$

0456 → Octal number [0 to 7 = 8 (base)].

456 → Decimal number.

0386 → Invalid no. bcoz '8' is pr. in the value [not octal] & zero is pr. [not decimal].

The Hexadecimal no's will be prefix with "0X"

0 to 15 = 16 ⇒ 0 to 9 & A-10, 11-B, 12-C, 13-D, 14-E, 15-F

Eg:- 0X452, 0xed1, 0X89a.

Here, Alphabets are not Case-Sensitive.

Real Constants:

This real constants are also fractional decimal point no's. it can be $\oplus^{ve}$ or $\ominus^{ve}$. by default it is $\oplus^{ve}$.

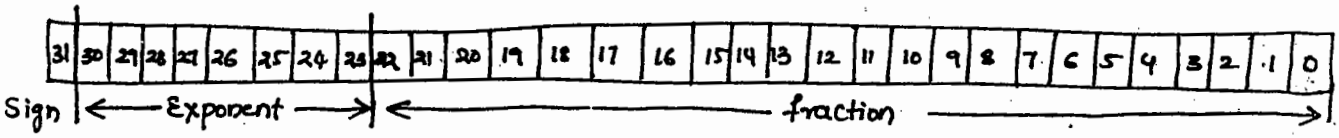
Eg:- 45.924, −100.09., 2486.76795 (2486: Mantissa; 76795: Exponent part)

The min. range of real constants is $-3.4 * 10^{38}$ to $3.4 * 10^{38}$.

IEEE 754 floating Point Standard:-

$(-1)^S \times F \times 2^E$

Single precision (bias = 127).



Character Constants:-

A char. cnstnt is a 1 byte of char. In a keyboard Everything is treated as a char. The char's has to Enclose in Single Quotations (' ').

The Range of char. constant is −128 to 127.

C lang. Supports only the ASCII code.

Eg:- 'A', 'a', '+', '=', '4', '6' --- etc.

If It is a combination of char's then it is called as String constant.

String constants are always enclosed in double Quotations (" ").

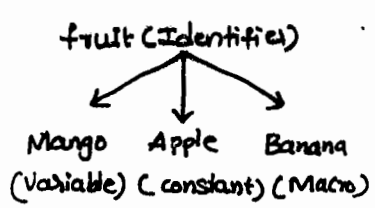
Eg:- "Anuyha", "Nani9871"---- etc

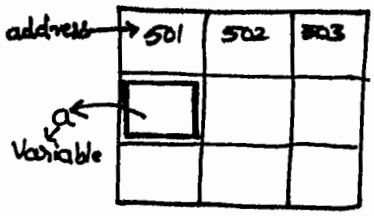
Variables:- A name which is given for any computer memory location is called as Variable Name. The purpose of the variable is, to store some data. The user will access by the Variable name & the compiler will access by the address.

Rules for declaring a Variables:- [To store the data]

* The Variable name can be lower case, upper case and mixed case.
* A variable name can't be a Keyword.
* A variable name can be Alpha numeric char's.
* A variable name can't contain any special char's Except underscore (_).
* A variable name can start with char. or underscore but not digit.

Note:- The length of the variable name can accept 255 char's but the Compiler will read only first 32 char's.





Datatypes :- A datatype describe what type of data we can store in a variable & it also allocates some bytes of memory for the variable.

Datatypes are classified in 3 categories.

1. primary Datatypes (primitive) (predefined) (Basic).
2. Derived Datatypes.
3. user defined Datatypes (Secondary).

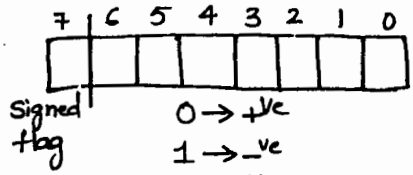
Primary Datatypes :-

This primary datatypes are the datatypes directly interacted with the machine instructions. Every primary datatypes will supports "Type modifiers".

The Type modifiers will alter the meaning of datatypes. There are 4 types of Typemodifiers. (These are nothing but a Keywords)

1. Signed (default.) 2. short 3. unsigned (+ve only) 4. long

when we defined Signed type modifier you are telling to the compiler the higher order bit has to interrupt by the sign. whenever Signed is specified a signed flag is generated.



if the Signed bit is zero, it is $\oplus^{ve}$ no., if it is '1', it is $\ominus^{ve}$ no., The short & long indicates the bytes of memory the short is less bytes & the long is more bytes.

I. int (Integer) (Signed int) (Signed) (short)

format Specifications	:	%i (or) %d (i = integer, d = decimal)
Size	:	2 bytes (16 bits)
Range	:	-32,768 to 32,767
Description	:	It accepts the no's of decimal type. It can be $\oplus^{ve}$ or $\ominus^{ve}$.
Eg:		56, 999, -27, 96379 ---- etc.
Macro's :		for every datatype we have macro's to find out max. int. value & min. int. value. To display max. int. we have a macro as, INT_MAX. To display min. int. we have INT_MIN.

Unsigned int:

format Specifications : %u

Size : 2 bytes (16 bits)

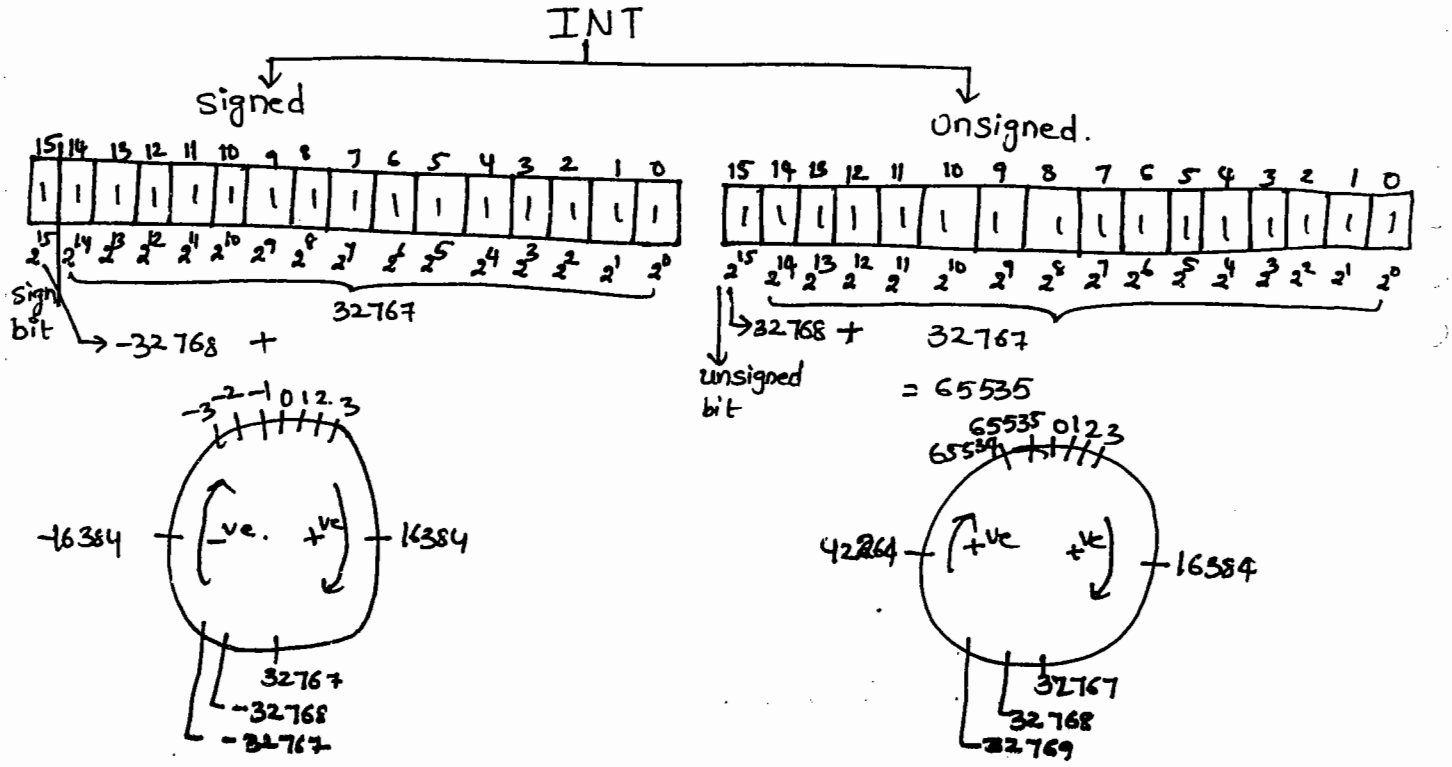
Range : 0 to 65,535

Description : It will accept only the integer of +ve nos.

Eg : 56u, 796u, 54396u ---- etc.

Macros : UINT_MAX.

UINT_MIN.



Note:- Externally the nos are diff. but internally the binary codes are same.

Long int :- (long)

format Specifications : %li or %ld.

Size : 4 bytes (32 bits)

Range : -2147 483 648 to 2 147 483 647

Description : It also accept integer type of data with more range.

Eg : 56l, 7961l, -69 432 l --- etc.

Macros : LONG_MAX

LONG_MIN.

Unsigned long :-

format Specifications : %lu

Size : 4 bytes (32 bits)

Range : 0 to 4 294 967 295

Description : It accept more range of +ve nos.

Eg : 56 lu, 967 lu, 79634 lu ---- etc

Macros : ULONG_MAX.
ULONG_MIN.

II Char (signed char) :-

format Specifications : %c

Size : 1 byte (8 bits)

Range : −128 to 127

description : Every key in the keyboard act as a char. Every char must be enclosed in single quotations (' ').

Eg : 'A' 'N' 'u' '9' '8' '@' ---- etc.

macros : CHAR_MAX.
CHAR_MIN.

unsigned char :

format specifications : %c

Size : 1 byte (8 bits)

Range : 0 to 255

Description : By this unsigned char we can accept some extra range of char's.

Eg : 'A', 'n' '@', '9' --- etc.

macros : UCHAR_MAX
UCHAR_MIN.

Note : All The above macros will be defined from the Header file <limits.h>

III Float :

format Specification : %f (or) %e (or) %g

size : 4 bytes (32 bits)

Range : $-3.4 * 10^{38}$ to $3.4 * 10^{38}$

Description : It accepts fractional point of no's. we can define any no. of decimal pts at the time of i/p but the compiler will display max. of 6 points.

Eg : 963.34567 8893636 9f, 634 57936.792134 5f, −479.497 f --- etc

macros : FLT_MAX.
FLT_MIN.

double :-

format Specification : %lf

Size : 8 bytes (64 bits)

Range : $-1.7 * 10^{308}$ to $1.7 * 10^{308}$

Description : It also store the fractional point of no's with 12 decimal points.

Eg : 396.784532129768 9 , ----- etc.

Macros : DBL_MAX
DBL_MIN

Long double :-

format Specification : %LF (or) %Lf

Size : 10 bytes (80 bits)

Range : $-3.4 * 10^{4982}$ to $3.4 * 10^{4982}$

Description : with fractional point accepts upto 19 decimal points.

Eg : 378.3445677912389345 Lf, ---- etc.

Macros : LDBL_MAX
LDBL_MIN.

Note: The above macros are debined the under the header file <float.h>

Finally in C-lang only 2 categories of datatypes → Integers & Non-Integers (float)

Since, char's internally ~~pass~~ the ASCII code which is integer type.

Note:- In all the data types the only integer will change the size depends on the bit Np. If it is 16 bit it is 2byte, if it is 32 bit it is 4byte.

Declaration and Initialization of Variables :

Syntax :-

<type> <size> <sign> datatype var-name = <initialization> ;

(default) auto short (d) +ve (d)

We have 4 storage classes of datasequence from the RAM, in that 4 "Type" is used to select 1 storage class. Defaultly we take "auto". Size indicates about short & long but defaultly short. Sign indicates +ve & -ve but defaultly +ve.

Eg:- ① int a ;
or
Signed a ;
or
short a ;

} Same

// 3 tokens are pri.e. datatype, variable, separators.

② float k; // k is a variable of type float

③ char s;

④ int a;
int b;
int c;
} ⇒ int a,b,c; // ≠ tokens

⑤ int x = 45;
declaration with initialization ⇒ int x;
x=45;

⑥ char india = 'A';

⑦ float s = 4.789;

Here if we didn't wrote the float before 's' then for reading as float we have wrote the value as s = 4.789f; otherwise it treated as double.

* int a = 300 * 300/300; //valid

* int _;
* int _1;
* int a_1;
* int _at;
} Here '_' is also treated as variable.

* int for; // for is a keyword // invalid
* int 1a; // invalid // variable can't start with Numeric.
* int 1,2,3; // invalid // no's not allowed.
* int e no; // Spaces not allowed.
* float e.no; // Special char's not allowed.

Flavors of c/c++ Editors :-

we have diff. types of Editors & Every Editor has its own compilers. Some of the C- compilers are,

TURBO C
QUICK C
MICROSOFT C
AZTECH C
ZORTECH C
LATTICE C
WATCOM C
GREEN LEAF C
VITAMIN C --- Etc.

Every compiler has its own editors called as IDE [Integrated Development Environment]

IDE contains : i.e it performs 5 Jobs.

1. Editor [Typing / Editing].
2. Debugger [Removal of Error].
3. preprocessor.
4. Compiler [c to machine].
5. Linker.

Installation of Windows-7 and Higher Versions of TURBO-C :-

Turbo-C Editor is developed under µp 8086 (16bit).

windows-7 64 bit :-

Select TurboC 3.0.5 ---> Turboc.exe & Install.

windows-7 32 bit :-

* first copy the slw in any like C or D or E drive. [for source path].
* Open the folder & select install.exe.
* press Enter.
* Enter the source drive to use : E
* press Enter
* E:\ tc\bin\tc.exe.

Windows XP :-

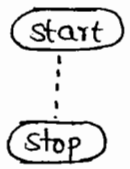
Turbo C++ V.3.0 → Setup → Run. (Then automatically Installed).

How to write a C-prgm :-

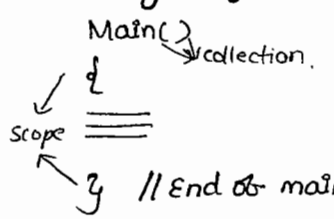
Algorithms

begin
≡≡≡
End

flow charts



prgmng lang's



The prgmng lang. will be Executed from the stmt of main(). The main() scope contains set of stmts. The main() will Execute the prgm.

In the Main() scope 2 blocks can be takes place.

1. Declaration block 2. Execution block

The Declaration block we declare the vasiables once the declaration is finish we have the Executable stmts.

Ilo statements :-

Every prgmng lang. contains Ilo stmts. This stmts is used to capture the data from the ilp device called as keyboard and display the olp on the screen by using olp stmts.

In C-lang the Ilo stmts are classified in 2 categeries formatted Ilo & unformatted Ilo.

The formatted Ilo can capture & display any type of data. The Unformatted Ilo can captue only specific type of data.

In C-lang. The Ilo stmts are called as functions.

Input output

formatted I/O

Datatype	Input	output
int	scanf()	printf()
float	scanf()	printf()
char	scanf()	printf()
string	scanf()	printf()

unformatted I/O

Datatype	Input	output
char	getchar() getch() getche()	putchar() putchar()
int	—	—
float	—	—
string	gets()	puts()

printf() :-

Syntax 1 : printf("user-defined string");

The first Syntax of the printf() is used to display the user defined stmts on the screen.

The first syntax of printf() contains only one argument within the double quotes(" "). whatever the stmts we are defining it is collection of characters.

```
Main()
{
printf("welcome TO C");
}
```

after writing the prgm, the prgm has to compile & Execute. for Compilation press "ALT+F9".

To run the prgm "CTRL+F9", F9 only for linking.

clrscr(); it makes to clear the screen before Execution of the prgm we use.

After clearing the screen it makes the cursor to blink in first row & first colomn.

To see the o/p on the screen press "ALT+F5" (it works only in Turboc).

In the above prgm the o/p not in the clear order to make the o/p in better way of Rep's, 'c' supports Escape Seauences.

Escape Sequences:-

In order to design the o/p in a preformatted Text, we use Escape Sequences in c-lang. The Escape sequences mostly will be used in the o/p stmts of printf().

ALL the Escape seauences will be defined with Backslash(\) followed by the char. All the Escape seauences ~~will~~ can be ~~defined~~ used any where in the printf(), within the " ". But mostly they used either begining & Ending of the stmts.

So they are called as "Boundaries" or "delimeters".

"\n" [newline character] :- It makes the cursor to blink in the next line.

```
Main()
{
printf ("welcome to c\n");
printf(" It is pop's \n");
printf(" welcome to c\n It is pop's");
printf(" \n\n\n *** welcome To c *** \n");
}
```

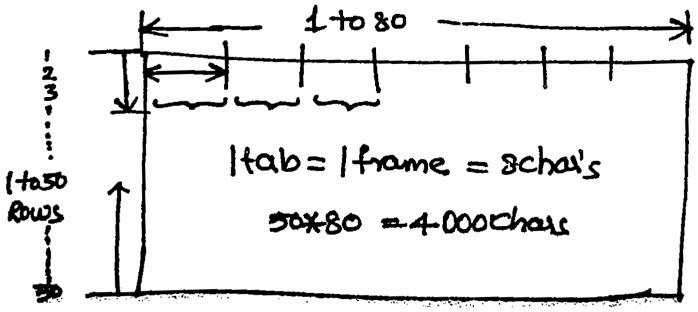
O/P:-
welcome to c
It is pop's
welcome To c
It is pop's

*** welcome To c ***.

"\t" [Tab]:- The tab will makes to move frame by frame not by chars.

```
Main()
{
printf ("1234");
printf ("1234\t 5");
printf ("1234\t5\t6");
}
```

O/P:-
1234
1234 - - - - - - - - 5 → cursor at next frame
1234 - - - - 5 - - - - - - - - 6
8 chars   8 chars



"\b" [Backspace]:- The Backspace will be used in development of password logics.

```
Main()
{
printf ("1234\b5678");
printf ("12345678\b");
printf ("1234\b \b5678");
}
```

O/P:
1235678
12345678
12 35678

printf(" 1234\b5678")
O/P: 1235678.

The Backspace moves back one position.

"\r" [Carriage Return] :- The \r makes to move the cursor on the same row of the 1st position [1st column].

```
Main()
{
printf("Anu\r");
printf("Anu\rc");
printf("Anu\rNag");
}
```

O/P:
ANU
cnu
NAG

Program for Escape Seauences:-

```
Main()
{
printf("\nab");
printf("\bsi");
printf("\rha");
}
```

O/P:

ab-
asi
hai

By using pf() it is possible to display any char on the screen. If any char not displaying use the Backslash & followed the char it display that char's.

*
```
Main()
{
clrscr();
pf("welcome to \"c\" ");
}
```
O/P: welcome to "c";

* pf("\\\\\\\"); O/P: \\\ [first we are getting Error bcoz compiler doesn't understand while there is no space at the end of the \ & double quote] so we maintain space, i.e; pf("\\\\\\ ");

* pf("\"); O/P: _

* pf("\\"); O/P: \

* pf("c is a procedure Oriented\
prgmng & function Oriented"); O/P: C is a Procedure oriented prgmng & fn Oriented

To whenever we break the stmt use the "\" for indication of the stmt is continuing

\n → Newline char
\t → tab char
\b → backspace
\r → Carriage Return
\" → "
\\ → \
\0 → Null char
\a → alert Sound
\f → form feed
\h → Horizontal
\v → Vertical

Note:- Every escape seauence is a char it is 1 byte of memory space.

* Eg: prgm :-

```
Main()
{
pf("Hello\n");
pf("welcome\\\\n Santhosh");
pf("welcome\\\\\n Santhosh");
pf("welcome\b\b Hello");
pf("welcome\b\b Hello");
pf("Hello\r");
pf("welcome\r Hello");
pf("Hello\t welcome");
}
```


*
```
main()
{
clrscr();
pf("Escape\\n\n Sequences");
getch();
}
```

O/P: Escape\n
Sequences.

Getch();- whenever the compiler read the getch() stmt it makes the prgm to suspend for a while until the user press any key on the keyboard.
The Actual purpose of getch() fun^n to capture Nondata keys.

getch() is waiting for any char type i/p. by using getch() in the source code then we need not to use ALT+F9 to get the o/p after compilation.

by using getch() we will access data keys (A-Z, 0-9, Enter, ---) & also Non data (F1 F2 ... → ← ↑↓, page up/down) keys.

by using getche() we only read data keys. getche() read both data & Non data keys. but getch() is better.

; → End of the line.

Comments:- whenever if you want to provide a documentation for the prgm we can write the documentation by placing in cmnts. whatever the stmts we place in cmnts. They will be ignored by the compiler. They are 2 types of cmnts.

1. Single line cmnt (//)
2. multiline cmnt (/* */)
   ↓ starting ↓ ending

Note:- C-lang doesn't Support Nested cmnts.

Eg:
```
/* printf("Hello");
   /* pf("Hello");
      pf("Anu");*/
      pf("Hai");*/
```

O/P: we can get errors bcoz of Nested cmnts.

printf()-Syntax-2:-

printf("userdefined stmts with format specification", arg1, arg2, --- argn);

The 2nd syntax of the pf() used to display any type of o/p on the console screen The arg's can be accepted any no. of, with format specifications.

The arg's can be variable type, constant no's & Expressions.

* printf("A:%d B:%d", 10,20);

  o/p: A:10 B:20 "A:10 B:20" internally takes as collection of chars.

* pf("%d%d", 10,20);

  "1020"

  o/p: 1020

* pf("%d,%d", 100,-5);

  "100,-5"

  o/p: - 100,-5.

* pf("%d %d", 100,200,300);

  o/p: 100 200

* pf("%d %d %d", 10,20);

  o/p: 10 20 GV.


```
* Main()
  {
  clrscr();
  pf("2+3");
  pf("2 + 3=");
  pf("\n%d", 2+3);
  pf("\n2 +3 = %d", 2+3);
  getch();
  }
```

o/p:

2+3
2 + 3 =
5
2 + 3 = 5.

Scanf():-

Scanf() is an i/p fun. which can capture any type of data from the key-board. we can accept i/p's for the variables in 2 ways:

1. at the time of variable declaration & initialization.

```
int a=5;
a=5;
```

2. By using scanf() at the time of Execution. The advantage by scanf() fun

Every time the values of the variables can be change.

Syntax:-

scanf("format Specifications", &arg1, &arg2, ----- &argn);

. The scanf() of the 1st arg is within the " " only containg. format Specifications. Remaining args of any datatype must be define with '&' indicates adrs [of variable in RAM]. The args must be Variable type only.

* Write a prgm to accept 2 no's find the sum of 2 no's?

To write a prgm basic things we have to analyse is,
1. i/ps of a prgm
2. o/ps of a prgm
3. Type of prgm (int, float --)
4. logic.

```
Main()
{ // Declaration.
int l,m, sum;
clrscr();
// Inputs
pf("Enter the values :");
sf("%d %d",&m &n);
// logic
sum = l+m;
//output
pf("sum of 2 no's : %d", sum);
getch();
}.
```


o/p: Enter the values : 9 ↵
6 ↵
Sum of 2 no's : 15.

* . int = int + int
float = float + float
float = int + float → Higher order bytes will dominate.
invalid = char + char

} works for only no's.

[assignment always done with Right to left.

Tracing of the prgm:.

A prgm can be Trace line by line to understand more clearly by pressing "F7". when we tracing the prgm we can watch the variables values by "Add watch" (ctrl + F7). if the Add watch not visible then press "F6".

%0.2f:. O/p is Restricted for 2 decimal points.

Help :- [Self learning from Turbo-C] "Ctrl + F1" (place the cursor at Required word).

```
Main()
{
int a;
float f;
long s;
a = 32767+1;
f = 32767+1;
s = 32767+1;
pfc("o/p d o/p f o/p Ld", a, f, s);
getch();
}
```

```
f = 32767+1.0; → double
f = 32767+1.0f; → float.
f = 32767.0+1; → 32768.000000
s = 32767S+1; → 32768
```

o/p:-

-32768
-32768.000000
-32768.

int = int + int
-32768 = 32767 + 1

pre-defined Header files :-

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

↓ header.

file inclusion macro.

→ folder name.

→ preprocessor directive (to substitute prgm code).

whenever if we are using predefined fun's in the prgm we have to define that related header file in the prgm by using "#include" preprocessor directive. This #include makes the file contents to be added in the prgm. what ever the stmts starts with '#' They are called as preprocessor directives. The job of the preprocessor directive is to Substitute the prgm code.

placing the header files is optional in the ".c", but it is mandatory in ".CPP". [Better to place in any language.]. [in c it is optional bcoz c is loosely checked but c++ is strong checked].

Debugging the program :-

Whenever a prgm is take in place, a prgm has to debug when debugging is taken place there are diff. types of errors will occurs.

Differences blw Errors, Bugs & warnings :-

Error :- It is nothing but mistakes in the prgmng language. These Errors are 3 types.

1- Syntax Errors 2. logical Errors 3. Runtime & Execution Errors.

1. Syntax Errors :-

Syntax means the correct way of "grammar" of writing a cmd & series of cmds, including all the proper options & command line stmts.

Whenever if we not follow the rules of the prgmng lang's this syntax errors will be raised. This errors can Easily can identify by the prgmr. Bcoz, it will clearly show that help with line no.

Eg: ; missing, writing the key words in upper case --- etc.

F5 → to Help what is the Error F1 → total details of Error.

2. logical Errors:

Semantics meant the logical meaning of a stmnt, separate from the grammatical structure. The logical mistakes will not identified by the System, its purely mistake by the user.

Eg:- for calculating the Net Salary of an Employee the formula is,

Net Salary = Basic Salary + Allowances ⊖ Deductions,

But through oversight,

Net Salary = Basic Salary + Allowances ⊕ Deductions
↳ Mistake done here (logical Error).

3. Runtime Errors [Execution Errors]:-

Whenever the user pass wrong I/P's as values, this run time errors will occurs. when runtime Errors is occur the compiler will take the control & raises a predefined Error msgs. This type of concept is called as Error handling. [C doesn't support Exception handling]

Eg: divided no. with zero, finding the sq.root for ⊖ve nos ---- etc.

There is a chances of "Linker Errors" can be takesplace.

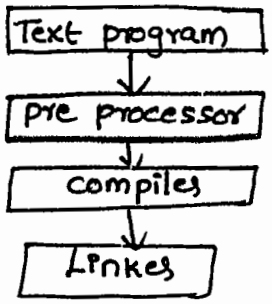
Bugs:- Bugs will be takes place after the Errors is completed. This Bugs will be raised by the Testing team. So, Bug is diff. when we compile with Errors.

Warnings:- whenever if you frame the stmts, the stmt doesn't having any meaning at that times warnings will be raise.

fd eg. in place of Sum=a+b; if we written a+b; it raises a warning as Code has No Effect.

Internal Execution of C-program:-

The Internal Execution of C-prgm will takesplace on 4 stages.



Text Program: The 1st stage of the C-prgm is the text prgm it contains not only '.c' prgm. It contains predefined '.h' files, user defined & supporting files. So, the text prgm is a combination of high level lang & macro lang. is submitted to the pre processor stage.

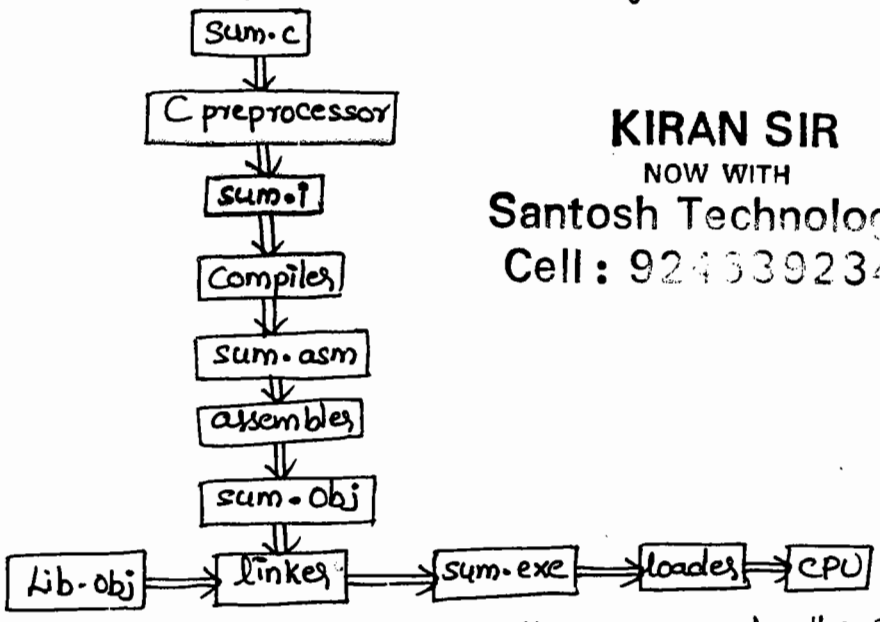
Preprocessor: The preprocessor is a 2nd stage of our 'c' prgm the Job of the preprocessor replace all the macro's & generate a file with an Extension of '.i' [Intermediate code]. which contains purely only the HLL. Since the compiler can understand only HLL's, This ".i" file will submit to the compiler.

Compiler :- The compiler will recieve IM code with HLL Every ins^{n} of the HLL is replaced with the assembly code of the C-compiler & generates a file with an Extension of ".asm". This assembly prgm will be converted into the objective code by the "assembler" translator

from the compiler we get the file of ".obj".

when we press ALT+F9, the objective file is generated. This file can't been takes place when there is a Syntax Errors.

Linker :- The JOb of the Linker accepting the your object code and the System library fun's obj. code. It links both the files if there is no linker errors then Executive file will be generated with an Extension of ".txt" file. This file will be generated when you press F9 (linking) Option.

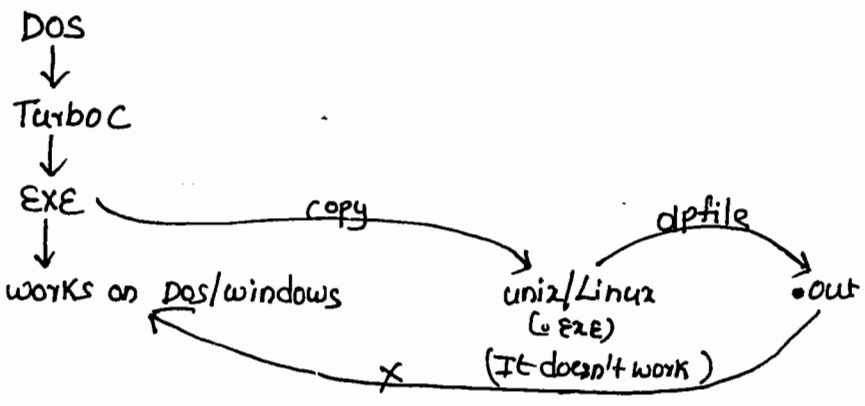


The advantage of ".exe" file without having the source code the prgm can Execute.

To get the Backup file press F_2 more than ones.

[if press F_2 1 time file will be saved, then press again backup will be created, Backupfiles Equal to the source file. if we lose sum.c then by using sum.bak we will manage]

C - olp file is a platform dependent. C - prgm is portable.



Operators:-

A operator is a special symbol which will be manipulated on operands.

Eg:. a ⊕ b → operand
↳ operator

In c-lang these are 2 categeories of operators are existed.

1. Binary Categeory 2. Unary Categeory.

The Binary operators are the operators which will be manipulated on 2 operands.

Eg:. a+b;

The unary operators are the operators which will be manipulated on single operand.

Eg:. -a;

whenever operators are considering with an expression we have to give the priorities to the operators & solve the Expressions.

What is an Expression:.

The Sequence of operators & operands that reduces to a single value after evaluation is called an Expression.

Eg:- 6*(2+3)-5 ⇒ 6*(5)-5 ⇒ 30-5 ⇒25.

when we solving this expressions based on the priorities it has to takesplace Sometimes Some operators will have the same priority. then we follow associativity.

They are 2 types:

1. Left to Right [Left associative]. 2. Right to Left [Right Associative].

Categeory	Operators	Associativity
Highest	( ) fun.call [ ] Subscript → pointer to membr • Member access.	Left to Right.
unary	! Not ~ 1's complement − unary minus + unary plus ++/-- pre ↑/↓ mnt & adrs of * -Value at adrs Sizeof typecast	Right to Left
Arithmetic	* mul / div % (Remainder) modulus	Left to Right
	+ plus − minus.	L to R
Bitwise	<< (left shift)	L to R
	>> (Right shift)	L to R
Relational	< lessthan <= lessthan or Equal to > greater than >= greater than Equal to	L to R
	== Equality != Not Equal to	L to R
Bitwise	& Bitwise and	L to R
	^ Bitwise Exclusive OR	L to R
	| Bitwise OR	L to R

logical	&& Clogical And)	L to R
	|| Clogical OR)	L to R
conditional & Ternary operator	? :	Right to Left
Assignment	= Equal to *= /= %= += -= <<= >>= &= ^= |= Assignment	R to L
comma	,	L to R
	++/-- post increment/decrement.	Right to Left.

Arithmetic :-

Expression	o/p d	o/p f
5/2	2	2.000000
2/5	0	0.000000
5.0/2.0	2	2.500000
2.0/5.0	0	0.400000
5.0/2	2	2.500000

Expression	output
47 %5	2
47 % -5	2
-47 % -5	-2 [consider Numerator Sign]
-47 % 5	-2
7 % 7	0
5 % 7	5
5.0 % 200	invalid [denominator didn't contain floating values]

Note: Modulus operator doesn't accept floating point no's. If we want to implement floating pt no's for modulus operator we can use a predefined function "fmod()"

* WAP accept a 4 digit no. display the no. in reverse order?

```
#include <stdio.h>
#include <conio.h>
main()
{
int a,x;
clrscr();
pfc"Enter the value:");
sfc"%d",&a); //7826
//logic
x=a%10; //6
pfc"%d",x); //6
a=a/10; //7826/10=782
x=a%10; //782%10=2
pfc"%d",x); //2
a=a/10; //78
x=a%10; //78%10=8
pfc"%d",x); //8
a=a/10; //7
pfc"%d",a);
getch();
}
```

o/p:- 6287.

* ~~main()~~
~~{~~
~~int a,b;~~
~~clrscr();~~
~~a=b=1;~~
~~while(a)~~
~~{~~
~~a=b<=3;~~
~~b=b+1;~~
~~printf("\n%d%d",a,b);~~
~~}~~
~~printf("\n%d%d",a+10,b+10);~~
~~}~~

Main()
{
int a,b;
a=b=1;
while(a)
{
a=b++<=3;
printf("\n%d%d",a,b);
}
printf("\n%d%d",a+10,b+10);
}
Main

* WAP accept a 4 digit no. find the sum of the 4 digits?

```
Main()
{
int num,x,sum=0;
printf("Enter fourdigit no.");
scanf("%d",&num);
x=num%10;
sum=sum+x;
num=num/10;
x=num%10;
sum=sum+x;
num=num/10;
x=num%10;
sum=sum+x;
num=num/10;
sum=sum+num;
printf("sum of %d numbers is ",sum);
getche(); }
```

* write a prgm accept a 5 digit No. find the sum of 2nd & last digit.

* find out Expression:

```
float c, f = 2/2.0;
        C = 5/9 *CF-32);
        c = 5/9.0 * CF-32);
        C = CF-32) * 5/9;
```

* write a prgm accept an amount in rupees find out the denominators in 100s, 50s & 10s and display the denominators.

```
#include <stdio.h>
#include <conio.h>
main()
{
int num, sum;
pfc" Enter the value");
sf c"%d", num)
num = num %1000.
num = num/100;
num = sum +num/10;
num = num%100;
num = num/100;
num = num%10;
num = num/10;
sum = sum+num;
pfc"value is", sum);
}.
```


*. WAP Accept 2 var. values & swap the var. values.

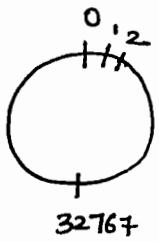
```
Main()
{
int a=20, b=30, temp;
clrscr();
pfc"%d %d", &a,&b);
temp= a;
a=b;
b=temp;
pf c"%d %d", &a,&b);
pfc"After swaping a: %d b: %d",a,b);
}
```

a=5 b=7	a=9 b=1	a=2 b=5
a=a+b;	a=a*b;	b=(a+b)-(a=b)
b=a-b;	b=a/b;	
a=a-b;	a=a/b;	

* WAP accept 3 nos swap the 3 Var. Values?

```
Main()
{
int a=30, b=20, c=10;
pfc"%d %d %d", &a, &b &c);
a=a+b+c;
a=a-b-c;
c= a-b-c;
b= a-b-c;
pfc"After swaping a: %d, b: %d, c: %d "a, b, c);
}
```


*.
```
Main()
{
int a = 300 * 300 / 300;
pf("%d", a);
}
```



Turbo C → MP. 8086 (16 bit)

[0-65535] → $\frac{300 \times 300}{300} = \frac{90000}{300}$

$= 90000$

65536

$24464/300 = 81$

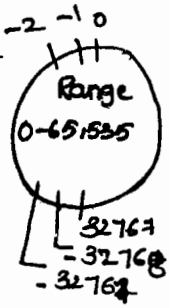
we have 2 types of memory will be takes place, compile time memory & Run time memories. This Run time memories will be changes depends on the bit MP. If it is an 16 bit editors like Turbo-C, we have max. run space 65,536 [0 to 65535]. If it is 32-bit editors like Borland ... etc, we have millions of runspace. The above prgm we are executing on 16-bit Turbo-C the expression 300 to 300 is 90,000. it is out of Run space. So, the compiler makes you to reduce within the range by substracting "from the Run Space". Then devide by 300 leads to "81".

*.
```
Main()
{
int a = 200 * 200 / 200;
int b;
b = 200/200 * 200;
pf ("%d %d", a, b); // -127, 200
}
```

$-65536 \quad 40000 \quad -25536$ ⇒ $\frac{-25536}{200} = -127$

$40000/200 = 200$

$\frac{200}{200} = 1 \quad 1 * 200 = 200$



Here, 40,000 is within runspace but not in range of +ve range of signed int.

```
* Main()
  {
  pf("%d", 300*200/100); //-55
  }
```

-65536
60000
-5536/100
= -55

```
* Main()
  {
  int a = 32000+1536;
  pf("%d", a); //-32000
  }
```

32000
1536
33536

-65536
33536
-32000

```
* main()
  {
  long a = 32767+1;
  pf("%ld", a); //-32768
  }
```

-65536
32768
-32768.


```
* main()
  {
  long a = 65536+300;
  pf("%ld", a); //300
  }
```

-65536
65836
300

Relational Operators :-

The Relational expression is takes place by a valid combination of numeric val's, numeric cnstnts, & relational & equality operators.

The Relational operators always returns either 1 or 0.

If the condition is satisfied. i.e; non-zero, it returns 1. if the condition is zero, it returns "Zero".

```
* Main()
  {
  int a;
  a = 3>2; //1(T)
  pf("%d", a);
  pf("%d", 5<4); //0(F)
  }
```

```
a = 4>5; //0(F)
a = 10==10; //1(T)
a = 10>9>8; //0(F)
      1>8
      0
a = 10>; //invalid
a = 6!>8; //1(T)
a = 6!=6; //0(F)
```

True	False
10, 25, ----	Zero.
-5, -9, ----	
0.01, 05, ----	
-0.2, -0.9, ---	
'A', 'N', 'U' ---	

```
a = (5>4) + (5>4); //2
a = 5>4>3>2>1>0>-1; //1(T)
a = 5>4 + 5>4; //0(F)
      5>9>4
      0>4=0
```

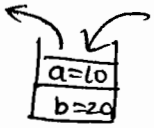
```
* Main()
  {
  int K=30;
  pf("%d %d %d", K<=30, K=40, K==30);
  }
```

0 40 1

← R to L

*. Sf("%d %d", &a, &b); (R to L)

a=10 b=20



pf("%d %d", a, b);

* display is from L to R & data flow into stack is Right to left.

In fun's like pf & sf we have only one associativity i.e; R to L, not depends on the operators associativity]

* Main()
{
int a=1, b=2;
pf("a: %d b: %d", a, b); //1 2 (R to L)
pf("\n a: %d b: %d", a, a+b); //1 3 (R to L)
pf("a: %d b: %d", a+b, a=a+b); //5 3 (R to L)
}

Logical Operators:

The logical Expression is a valid combination of logical values, variables & operators The logical operators can be combined with relational Expressions.

Not(!)

A	B
0	1
1	0

logical AND (&&)

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

logical OR (||)

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

a = 5>4 && 4>3; //1

1 && 1 = 1

According to the AND operator before condition is dissatisfied it doesn't check the Remaining Expressions.

According to the OR operator bcoz condition is Satisfied then Remaining conditions will not checks.

a = 7 && 8; //1, the both values are non zero, so, o/p is 1.

1 && 1 = 1

a = 7 || 0; //1

a = 5>4 && 5>10; //0

a = 5>4 || 5>10; //1

a = 1+2 && 2+3; //1

a = 1+(2&&2)+3; //5

a = 6||0; //1

a = 6>6 || 5>5; //0

```
a=!1; //0
a=!25; //0
a=!-10; //0
a=!-3.5; //0
a=!-2.3; //0
a=!-0.01; //0
a=!7; //0
a=!0 //1
a=!-3>>3; //0
a=!(-3>>3); //1
```


Sizeof Operator :- The sizeof operator is used to estimate the size of the data-type or variables, fun's --- etc. The sizeof operator always returns an integer value

*.
```
#include<limits.h>
#include<float.h>
main()
{
double s;
pf("sizeof integer: %d \n", sizeof(int));
pf("size of integer/float: %d \n", sizeof(float));
pf("size of s : %d \n", sizeof(s));
pf("%d \n", sizeof(char));
pf("max. Integer : %d \n", INT_MAX);
pf("max. float : %g \n", FLT_MAX);
getch();
}
```

O/P:-
```
sizeof integer : 2
sizeof float : 4
sizeof s : 8
1
MAX. Integer : 32767
MAX. float : 3.40282e+38.
```

*.
```
Main()
{
char s;
pf("%d %d %d %d %d %d", sizeof 45, sizeof 'A', sizeof s, sizeof 1.0, sizeof(32768),
                                                    sizeof (0x234) );
}
```
'A' is taken as ASCII value i.e. A=65, that is 2 bytes
32768 is the integer out of range value so, 4 bytes
R to L

O/P:- 2 2 1 8 4 2

Except datatype the sizeof operator have bracket is not mandatory but for datatype it is must & should.

*
```
Main()
{
pf("%d, %d", sizeof(3.14+67), sizeof 3.14+67 );
}
```
R to L

O/P:- 8 75

sizeof(3.14)+67 = 8+67 = 75

```
*. Main()
   {
   char a,b;
   float c,d;
   pfc"%u %u %u", sizeof(a+b), sizeof(b+c), sizeof(c+d));
   }
```

O/P:. 1 4 4

Type Casting:-

Converting 4m one datatype to anothes datatype is called as Typecasting

They are 2 types.

1. Implicit T.C 2. Explicit T.C.

Implicit T.C:- The compiler itself will convert 4m onetype to another type such type of casting is Implicit T.C.

```
* main()
  {
  char s='z';
  pf C"%d",s);//90 (Ascii code of z)
  getch();
  }
```

Explicit T.C:- If the compiler unable to convert then the user explicitly has to convert 4m one type to another type is called as Explicit T.C.

```
* Eg:- main()
      {
      int a=5, b=2;
      float quo;
      quo=a/b;
      pfc"%f", quo);//2.000000
      getch();
      }
```

```
Main()
{
int a=5, b=2;
float quo;
quo = (float)a/b;
pf C"%f", quo); //2.500000
getch();
}
```

Syntax:- E.T.C:-

Var-name = Cdatatype) Expression

Eg:- * 32767+1; // Explicit T.C

```
* int m1,m2,m3,tot;
  float avg;
  tot = m1+m2+m3;
  avg = tot/3.0;
  (or)
  avg = tot/3f;
```

// E.T.C.

Assignment Operators:-

Assignment	updation operator
int a=5;	int a=5;
a=a+1; (2 operators)	a+=1; (1 operator)
a=6;	a=6;
num=num/10;	num/=10;
sum=sum+num%10; (3 operators)	sum+=num%10; (2 operators)

Modulus Operator doesn't work with floating point values.

```
* main()
  {
  int a=5;
  pf("old %d\n",a);
  a=a+1; //updated
  pf("old %d\n",a);
  a+=1; //updated
  pf("old %d\n",a);
  a+1; // not updated.
  pf("old %d\n",a);
  pf("old %d\n",a+1);
  pf("old %d",a);
  getch();
  }
```

o/p:- 5 6 7 7 8 7

Note:. The updatable stmts in the memory locations.

a=a+1; a+=1; a++; ++a; a--; --a;

Increment / decrement Operators:-

Things to note:-

1. a++, The Result of the Expression is the original value of 'a'.
   Eg:- a=5
   a++ ⇒ 5.
2. ++a, The Result of the Expression is the ped value of 'a'
   Eg:- a=5
   ++a = 6.
3. a--, The Result of the Expression is the original value of a.
   Eg:. a=5; a-- = 5;

A. --a, The Result of the Expression is the dec value of a.

Eg:- a=5; --a=4;

Type-1:- The Recognization of type 1, it is always within the printf() with single argument.

*.
```
Main()
{ int x=5;
clrscr();                                   o/p.   Memory location
pf("value of x: %d \n", x++); // 5          6
pf("value of x: %d \n", --x); // 5          5
pf(" value of x: %d \n", x--); // 5         4
pf(" value of x: %d \n", ++x); // 5         5
pf(" value of x: %d \n", x++); // 5         6
getch();
}
```

Type-1 with Expressions :-

```
Main()
{
int a;
a=1;
clrscr();
pf("\n %d", ++a * ++a * ++a); //24
pf("\n %d", ++a * a++ * ++a); //16
pf("\n %d", a++ * a++ * a++); //6
pf("\n %d", ++a * ++a * a++); //18
pf("\n %d", a++ * a++ * ++a); //8
pf("\n %d", a++ * ++a * ++a); //12
getch();
}.
```

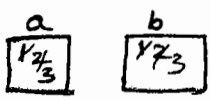

Type-2:- Type-2 also within the pf() but it has more than 1 arg.

*.
```
#include <stdio.h>
Main()
{
int x=6;
pf("%d %d %d", x++, ++x, x--);
}.
```
R to L

o/p:.. 6 6 6

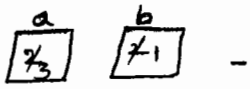
Type-2 with Expressions :-

```
void main()
{
int a,b;
a=b=1;
pfc("\n%d %d", ++a *b++ , a++ + ++b); // 6,3
a=b=2;
pfc("\n%d %d", --a *++b , a++ - --b); // 4,1
}
a=b=3;
pfc("\n%d %d", a-- +b++ , --a *b--);//4,6
```



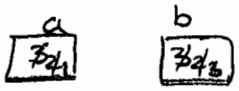
++b = 2 a++ = 1

b++ = 2 ++a = 3



--b = 1
a++ = 2 = 1

--a = 2
++b = 2 * = 4



--a = 2
b-- = 3 * => 6

a-- = 2
b++ = 2 = 4

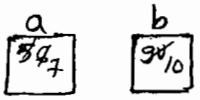
Type-3:-

```
Main()
{
int a;
a=5;
++a;//a=a+1
pfc("%d",a);//6
}
```

```
Main()
{
int a;
a=5;
a++;//a=a+1
pfc("%d",a);//6
}
```

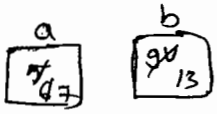
```
*. Main()
{
int x,y,z;
x=5;
y=x++;
pfc("x: %d y: %d",x,y);//5 5
z=++y;
pfc("y: %d z: %d",x,y);//6 6
}.
```

```
*. Main()
{
int a=5,b;
b= a++ + a++;
pfc("a: %d b: %d",a,b);//7,10
}.
```



b = a++ + a++
b = a + a.

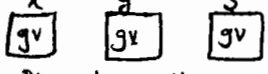
```
*. Main()
{ int a=5,b;
b=++a + ++a;
pfc("a: %d b: %d",a,b);//7,14
}
```



b = ++a + ++b
(3) (1) (2) ← R to L

Steps to follow :-

1. Take the memory blocks.



↕ Every thing update in the memory blocks only

2. write down the expression.

x = y++; / z = ++y; z = y;
(2) (1) (3)

3. If Expression contain post operator remove the post & write down the expression again.

y = x;

4. And then Assign priority

y = x;

5. finally update the post values into variables (memory blocks)

* b = ++a + a++; //7,12

b = ++a + a
1 1 1
③ ① ②

a 7 6 7 7 b 9 9 2

* b = a++ + ++a; //7,12

b = a+ ++a
↓ ↓ ↓
③ ② ①

* b = a++ + --a + a-- + ++a; //5,20

b = a+ --a + a + ++a;
↓ ↓ ↓ ↓
④ ② ③ ①

a 7 6 9 5 b 9 9 20

* Main()
{ int a=10, b;
clrscr();
b = a++ + ++a;
pfc"olod olod olod olod", b, a++, a, ++a);
getch(); ← R to L
}

O/P:- 22 13 13 13

a 10 4 13 b 9 9 22

b = a++ + ++a;
b = a+ ++a;
↓ ↓ ↓
③ ② ①

* Main()
{ int x=5, y=4, z=3;
z = y++ + x--;
pfc"x= olod y= olod z= olod \n", x, y, z);
x = -4;
y += 5;
z = y-- + x++;
pfc"x = olod y = olod z = olod \n", x, y, z);
}

x 5 4 -4 -3 y 4 5 9 z 3 9 6

z = y++ + x--
z = y + x
x ⇒ 4 y ⇒ 5 z ⇒ 9.
x = -4
y = y + 5
z = y-- + x++
= y + x

* Main()
{
int z, x=5, y=-10, a=4, b=2;
clrscr();
z = x++ - --y * b/a;
pfc"olod", z); //10
}

x 5 6 y -10 -9 z 9 9 10

z = x++ - --y * b/a
z = x - --y * b/a
z = x - --y * b/a
3 0 2 1
5 - -10 * 2/4 ⇒ 5 - -11 * 2/4 = 5 - 22/4
= 5 - -5 = 10

Increment/decrement operators with logic AND & OR operators:-

According to the OR operator, before condition is satisfied there is no need to check the next condition.

According to the AND operator, before condition is dissatisfied there is no need to check

```
*. Main()
   {
   int a,b,c;
   clrscr();
   b=c=1;
   a= ++b>1||++c>1;
   pf("%d %d %d", a,b,c);
   }
   b=c=1;
   a= ++b>2 ||++c>1; //122
   b=c=1;
   a = ++b>1 && ++c>1; //122
   a= ++b>2&& ++c>1; //021
   b=c=d=1;
   a= ++b>1 || ++c>1&& ++d>1; //1211
   a = (++b>1)|| (++c>1&&++d>1); //1211
   a = (++b>1||++c>1) && ++d>1; //1212
```


Comma Operator:-

```
* Main()
  { int a;
  a= (3+2, -5, 12);
  pf("%d", a); //12
  getch();
  }
```

```
*. Main()
   {
   int a;
   a=3+2, -5, 12;
   pf("%d", a); //5 and gives a warning as Code has no Effect
   getch();      to effect comma we have to take values in ()
   }.
```

```
* Main()
  {
  int a;          a*=10+2
  clrscr();       a=a*12
                    =10*12=120
  a=10;
  a*=10+2;
  pf("%d", a); //120
  }
```

Misallineous Concepts:-

As w.k.t operators returns a value similarly a fun. also returns a value, but recieving that value is optional. for suppose a pf() not only display the o/p. it also returns a value. As w.k.t, pf() accepts any no. of arg's but it returns the 1st arg. values. i.e; with in the " " (after substituting). Sf() also returns no. of arg's with in the " " but only the format Specifications.

```
*. Main()
   {
   int x;
   x = pf(" Hello");//Hello
   pf(" old",x);//5
   getch();
   }
```

```
x = pf (" Hello old",100); // Hello100
pf("old",x); //9
```



```
* Main()
   {
   pf("old", pf ("Hello"));// Hello5
   }
   x= pf("old", pf("Hello"));//1
   pf ("old", pf ("old ,old hi ",10,20));// 10 20hi7
```

```
*. Main()
   { int a;
   clrscr();
   a=pf (" Hai old Kiran", pf ("SQL"));// Hai3Kiran
                                       9
   pf("old",a);   -> O/P:- SQLHai3Kirang
   }
```

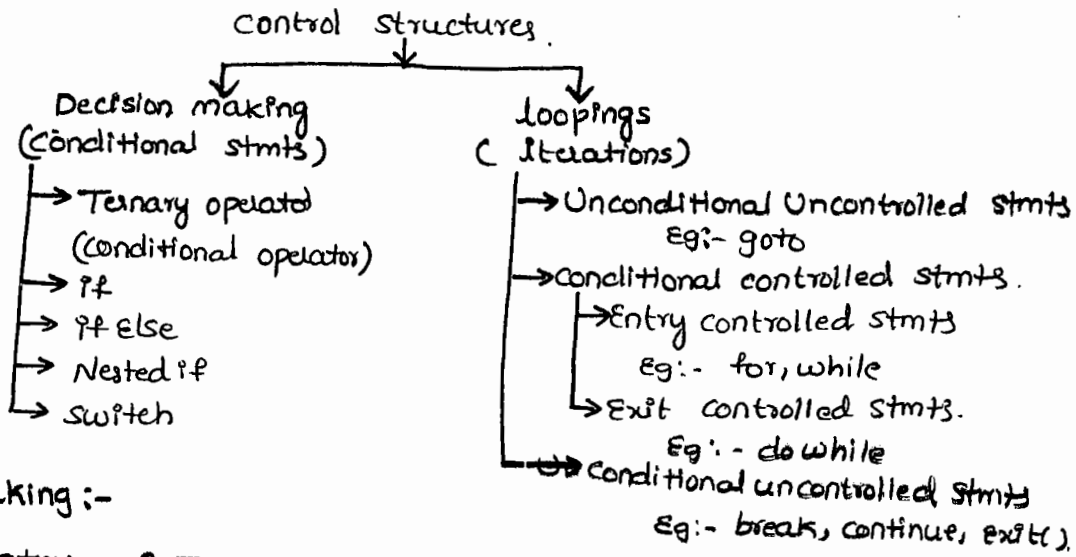
```
*. #include<stdio.h>
  Main()
  { int a,b,c;
a= pf ("welcome Santhosh");// welcome Santosh
b=sf("old old", &c);// 2 values returns & c takes 20 as i/p.
                    20
// Entered values are 20 30
pf("\n old old old ",a,b(),);// 15,2,20
  }
```

```
*. #include <stdio.h>
 Main() {
 int a,b,c;
 b=c=100;
pf("\n old old old",a,b,c);// gv 100 100
a=pf("welcome Santosh old", pf("Hello Kiran"));// Hello Kiran welcome Santosh11
a=pf("\n old old old old old", sf("old old", b, &c), a,b,c);//\n217 100 100
  // Entered values are 20 30
                        30.
```

```
pf(" \n old old old ",a,b(,);// 13 100 30
 }
```

Control Structures:-

Upto right now we are implementing sequence of prgmng in this every line of the stmt, in this every stmt is executed step by step, any stmt will not be ignore. In order to ignore some of the stmts at the time of execution we need to implement control structure prgmng. In this prgmng, based on the conclitions some stmts will executed & some stmts will not executed.

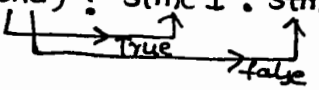


Decision Making :-

Ternary Operator :- A Ternary operator it uses 2 special symbols "?" and ":"

first, it check the condition, if the condition is satisfied then abtd the "?" what ever the stmt is pr. i.e. executed. If the condition is not satisfied, abtd the ":" mark what ever the stmt is pr. i.e. executed.

Syntax:- (cond) ? stmt 1 : stmt 2 ;



In either of the case only one stmt is executed.

* WAP accept a no. find out the given no. is Even or odd?

```
#include<stdio.h>
#include<conio.h>
main()
{
int num;
Pf("Enter the num:");
Sf("%d", &num);
(num %2==0) ? pf("Even") : pf("odd");
getch();
}
```


* WAP accept age of the person, find out the person is Eligible to Vote or not?

```
Main()
{
int age;
```

```
pfc"Enter the age:");
Sfc"%d", &age);
(age>=18)? pfc"Eligible"): pfc("not Eligible");
getch();
}.
*. Main()
{
int a;
a=3>2 ? 10:20;
pfc"%d",a);//10
getch();
}
```

* a=3>4 ? 10:20; //20

a=1?2:3; //2

a=0? 10:20; //20

a= 3>2>1 ? 10:20; //20

a = 1+2 ? 2+3 : 3+4; //5

a = 7?8; // invalid, bcoz there is no combination of ? &:

a=6:9; //

a = 1:2?3; //invalid

a = 3>2 && 2>1 ? 10:20; //10

a = !-4>4 ? 1:2; //2

a = 3>4 ? 5<6 ? 1:2: 10>8?3:4 //3

= 3>4? 5<6? 1:2:3 = 3>4?1:3 =3

a= 5>6? 10:6>4?20:30; //20

=5>6 ? 10:20 =20

a = 5>4 ? 6>7 ? 10:20:30; //20

= 5>4 ! 20:30; =20

a = 3>4 ? 2>1 ? 10:20: 5>4 ? 30:40; //30

= 3>4 ? 2>1 ? 10:20:30 = 3>4 ? 10:30 =30

a = 3>4 ? 2>1 ? 10:20; 5>6 ? 30:8>7? 3>4? 40:50:60; //50

= 3>4 ? 2>1 ? 10:20; 5>6? 30:8>7? 50:60;

= 3>4 ? 2>1 ? 10:20 : 5>6? 30:50; = 3>4 ? 2>1? 10:20:50 = 3>4? 10:50 =50

a = 2?1? 0? 10:20:30:40; //20

= 2? 1?20:30:40; = 2?20:40; =20

* WAP accept a no. find out the given no. is +ve, -ve or zero!

```
Main()
{
int a;
pf("Enter the value:");
sf("%d", &a);
(a>0)? pf("+ve"): (num<0)? pf("-ve"): pf("zero");
getch();
}
```

*
```
Main()
{
int K, num = 30;
K = (num>5 ? (num<10? 100:200):500);
pf(" In %d", num); //30
}
```

```
Main()
{
int a=100;
100++;
pf("%d", a); //Error, Lvalue required.
}
```

* WAP accept 4 nd's find the min. value!

*
```
Main()
{ int K=12, n=30;
K = (K>5 && n=4? 100:200);
pf("K = %d", K); // Error, Lvalue Required
}
```

12>5 && 30=4? 100:200
1 && 30=4? 100:200
1=4? 100:200
1=100

*. it raises an error, Lvalue required The lefthside must be a variable not a constant no.

* write a prgm accept 3 nds, sort the 3 nds in ascending or descending?

Advantage of Ternary Operator:-

*. Every stmt is possible to finish in single stmt, it is very flexible

Drawback:

* it will execute only single stmt it can't possible to execute multiple stmts by using ternary operators.

If :-

Syntax:- if (cond) → Replacement by compiler.

```
Stmt 1;          if (cond)
Stmt 2;          {
                 Stmt 1;
                 }
                 stmt 2;
```

for single stmts it is optional.

Cond	Stmt 1	Stmt 2
T	✓	✓
F	X	✓

*. Main()

case-1

```
{
if(3<2)
Pfc("Right")
Pfc("wrong")
getch();
}
```

o/p:- Wrong

case-2

```
Main()
{
if(3>2)
Pfc("Right");
Pfc("wrong");
getch();
}.
```

o/p:. Right wrong.

In the 1st case, the o/p properly displayed
In the 2nd case it is not a proper way of o/p.
for this reason, we use the "else" keyword.

*. if(5>2) → Replacement.

```
Pfc("A");
Pfc("B");
Pfc("C");
Pfc("D");
```

o/p:- OBCD

```
if(5>2)
{
Pfc("A");
}
Pfc("B");
Pfc("C");
Pfc("D");
```

*. if(5>2)

```
{
Pfc("A");
Pfc("B");
}
Pfc("C");
Pfc("D");
```

o/p:. ABCD & no need of Replacement.

If else :-

Syntax:1

```
if(cond)
Stmt 1;
else
Stmt 2;
```

Replacement

```
if(cond)
{
 Stmt 1;
}
else
{
Stmt 2;
}
```

Condition	Stmt 1	Stmt 2
T	✓	x
F	x	✓

Syntax 2 :-

```
if (cond)
Stmt 1;
else
Stmt 2;
Stmt 3;
----
Stmt n;
```

Replacement

```
if(cond)
{
Stmt 1;
}
else
{
Stmt 2;
}
Stmt 3;
----
Stmt n;
```

cond	S_1	S_2	S_3	S_n
T	✓	x	✓	✓
F	x	✓	✓	✓

Syntax 3 :-

```
if (cond)
Stmt 1;
Else
{
Stmt 2;
Stmt 3;
----
}
Stmt n;
```

Replacement

```
if (cond)
{
Stmt 1;
}
Else
{
Stmt 2;
Stmt 3;
------
---
}
Stmt n;
```

cond	S_1	S_2	S_3	S_n
T	✓	x	x	✓
F	x	✓	✓	✓

* WAP accept basic Sal. of Emp. if the Sal>20000 deduct the tax 20% or else 15%.

```
#include <stdio.h>
#include <conio.h>
void main()
{
float salary;
clrscr();
pfc" Enter the sal:");
sfc"%of", &salary);
If (salary >20000)
{
Tax = 20* bs/100;
Else
Tax= 15* bs/100;
pfc" Tax calculated %0.2f on basic Sal: %0.2f ", tax, bs);
getch(); }.
```

Multiple if :-

Syntax:

```
if(cond)
Stmt 1;
Stmt 2;
Else
Stmt 3;
Stmt 4;
----
Stmt n;
```

Replacement

```
if (cond)
{
Stmt 1; } multiple stmts.
Stmt 2;
}
Else
{
Stmt 3;
Stmt 4;
--
Stmt n;
}.
```

This is not a proper way, whenever multiple stmts are Executing after if cond^n before Else, after the cond. compulsary to place the stmts within the body.

Cond	S_1	S_2	S_3	S_4	S_n
T	✓	✓	x	x	✓
F	x	x	✓	✓	✓

Syntax:

```
if(cond)
{
S1;
S2;
}
Else
S3;
S4;
---
Sn;
```

Replacement.

```
if (cond)
{
S1;
S2;
}
Else
{
S3;
}
S4;
--
Sn;
```

Cond	S_1	S_2	S_3	S_4	S_n
T	✓	✓	x	✓	✓
F	x	x	✓	✓	✓

* WAP accept basic sal. of an Emp. if sal > 10,000. HRA is 25%, DA 20% TA 15% Else HRA = 20% DA = 15% TA = 10% on basic sal. Calculate the net sal. & display clearly the basic sal. HRA, DA, TA.

```
Main()
{
float bsal, hra, da, ta, net;
clrscr();
pfc" Enter the bsal");
sfc"%of", &bsal);
if (bsal>10000)
{
hra = 25*bsal/100;
Da = 20* bsal /100;
Ta = 15* bsal /100;
}
```

```
else
{
hra = 20*bsal/100;
Da = 15*bsal/100;
Ta = 10*bsal/100;
}
net= bsal+Hra+Da+Ta;
pfc" The net sal %f, Hra, Da, Ta for bsal is : %f %f %f %f",netsal, Hra, Da, Ta);
getch();
}
```

*
```
Main()
{
clrscr();
if( 'A'<'a')
pf ("A");
else
pf ("a");
getch();
}
```
o/p:- A

*
```
Main()
{
int a=10;
if(a=0)
pfc("Hello %d", a);
else
pfc" %d Hai", a);
getch();
}
```
o/p:- 0 Hai

*
```
Main()
{
if(pfc("Hello"))
}
```
o/p:- Error.

As the cond. is satisfied but there is no stmt to Execute, so, it raises an Error.

To make to Execute,

```
Main()
{
if(pfc("Hello"))//if 5
{
}
}
```

(or)

```
Main()
{
if(pfc"Hello"));
}
```

o/p:- Hello

It is valid but it is not Treating as cond. stmt.

```
* Main()
  {
  if(?)  →  if(!pf("Hello"))
  pf("Hello");
  Else
  pf("world");
  }.
```

```
* Main()
  {
  clrscr();
  if(1);
  pf("yes");
  Else
  pf("No");
  }
```

```
* Main()
  {
  if(1)
  ; //null Stmt.
  Else
  pf("No");
  }.  O/P:- Error.
```

```
* Main()
  {
  pf(0);
  pf("yes");
  pf("No");
  }
```

O/P:- yes No

It is not a cond. stmt. bcoz pr. of semicolon it means of EndOfLine

```
* Main()
  {
  if(0);
  pf("yes");
  Else
  pf("No");
  }
```

O/P:- Error

bcoz of there is no if cond., without if there is no possibility of else cond.

```
* .Main()
  {
  if(0)
  ;
  Else
  pf("No");
  }
```

O/P:- NO

The Semicolon indicates Null stmt blw if & Else

without writing a stmt we are placing Empty stmt.

```
* .  Main()
  {
  if(0)
  ;
  else;
  }
```

O/P:- Null

```
*
  Main()
  {
  int a;
  a=10;
  if(a=0)
  pf("welcome to od",a);
  Else
  pf("\n od Hello",a);
  }
```

O/P:- 0Hello

```
* Main()
  {
  int a=0;
  int b=20;
  char x=1;
  char y=10;
  if(y,b,x,a)
  pf("Hello");
  else
  pf("hi");
  }
```

O/P:- Hi

```
* Main()
  {
  float me=1.1;
  double you=1.1;
  if (me==you)
  pf("OK");
  else
  pf("yes");
  }
```

O/P:- yes

```
* Main()
  {
  int x,y=2,z,a;
  if(x=y*1.2) z=2;
  a=2;
  pf("\n od %d",z);
  }
```

O/P:- garbage

Here, in this prgm it is not only checking the no. of decimal points of float & double.

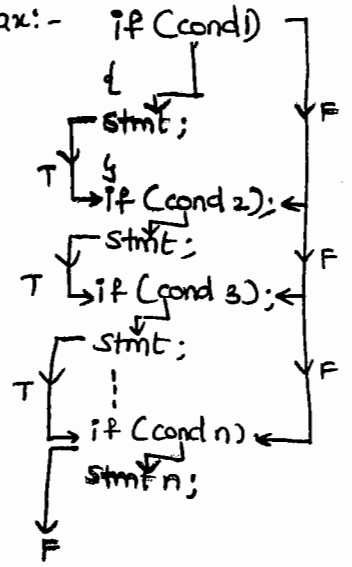
It is also checking the given no. is recurring or Non-recurring no.

The Recurring no's are the no's which never ends. whenever recurring no. is occuring, the float & double never equal.

The double recurring no. is always greater than the float recurring no. if the no's are non-recurring no's then the cond. is satisfied

Multiple if Condition :-

Syntax:-



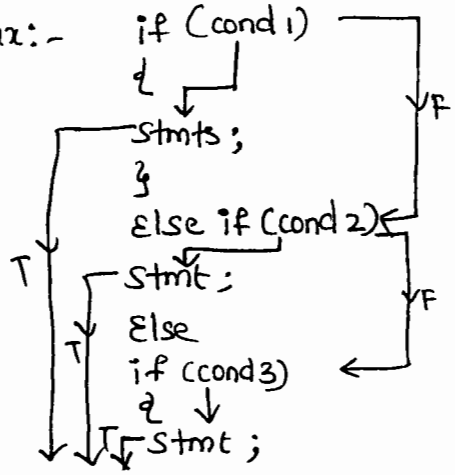
In this multiple if conditions every cond. will check, either the cond. is satisfied or not satisfied. There is no choice of leaving any condition.

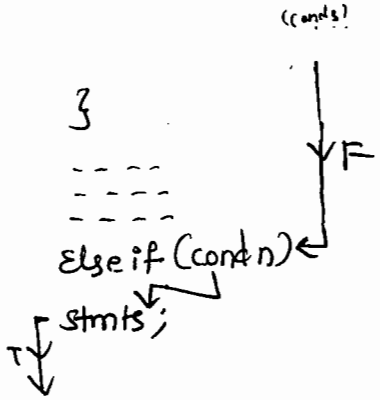
* WAP accept 4 no's find out the big numbers.

```
Main()
{
int a,b,c,d, big;
Pfc" Enter the values").
Sf c"%d%d%d%d", &a,&b,&c,&d);
big=a;
if (b>big)
big=b;
if (c>big)
big=c;
if (d>big)
big=d;
Pfc" Biggest no-%d", big);
getch();
}
```


* if else if :-

Syntax:-





* WAP accept a char. find out the given char is upper, lower, digit or any special char?

```
Main()
{
char ch;
clrscr();
pf("Enter the char.");
sf("%d", &ch);
if(ch==65 && ch<='z')
pf("%c is, upper case or ASCII %d", ch,ch);
Else if(ch>=97 && ch<=122)
pf("%c is lower case or ASCII %d", ch,ch);
Else if(ch>=48 && ch<=57)
pf("%c is digit or ASCII %d", ch,ch);
else
pf("%c is special char or ASCII %d", ch,ch);
getch();
}
```

1 → 26 → A-Z (upper case) (65 to 90)

2 → 26 → a-z (lower case) (97 to 122)

3 → 10 → digits (0-9) (48 to 57)

4 → 32 → Special Symbols (+, -, ;,)

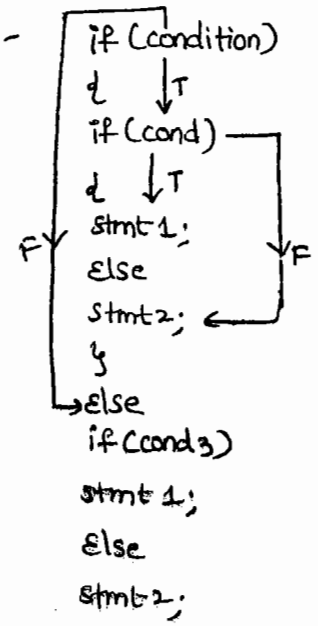
5 → 34 → Non printable char's & Cntrl keys
Eg:- Esc, Space, Enter (white space)

6 → 128 → Graphic char's.

7 → 0-255 → 256 char's.

Nested if Condition:-

Syntax:-



* Write a prgm accept marks in 3 subjects. find out total & avg. display total & display the grade for the students. if avg > 60 → grade "A", avg > 50 → grade "B", else ordinary pass. If any one subject as to fail display as fail category without grade, min. 35 marks for Each subject

[Dont use logical operators]

```
Main()
{
int m1, m2, m3, tot;
float avg;
clrscr();
pfc"enter the marks:");
sfc"%d%d%d", &m1, &m2, &m3);
tot = m1+m2+m3;
avg = (float)tot/3;
pfc"total : %d and Avg %.2f\n", tot, avg);
if( m1>35)
{
if(m2>35)
{
if (m3>35)
{
if (avg>60)
pfc"PASS--- GRADE A");
else
if (avg>50)
PFC"PASS--- GRADE B");
Else
pfc"ordinary pass");
}
Else
Pfc"fail");
}
Else
pfc"fail");
getch();
}
```


* WAP to compute electricity bill for domestic & commercial purpose with the following

No. of units	Rate/unit.
0-50	1.45
51-100	2.85
101-200	3.95
>200	4.50

Extras RS 10/- Sevice charge + RS. 0.06/- per unit power tax. subject to min. of RS 20/- for Single phase amt 50 for 3 phase.

for commercial purpose.

No. of units	Rate/unit
0-100	3.95
>100	7.00

Extras RS 20/- Sevice charge + RS-0.06/- per unit power tax subject to min. of RS 50/- for Single phase & RS-100/- for 3 phase.

```
Main()
{
int nu,ph;
char type;
float amt;
clrscr();
pf("\n\t\t 1. Domestic (D/d)");
pf("\n\t\t 2. commecial (C/c)");
pf("\n Enter option:");
sf("%c", &type);
if(type != 'D' && type != 'd' && type != 'C' && type != 'c')
{
pf("Invalid type");
getch();
Exit(0);
}
clrscr();
pf("\n Enter no. of units");
sf("%d", &nu);
pf("\n Enter phase (1 or 3):");
sf("%d", &ph);
if(ph != 1 && ph != 3)
{
pf("Invalid type");
getch();
Exit(0);
}
if(type == 'o' || type == 'd')
{
if(nu<=50)
amt = nu * 1.45;
else if(nu <=100)
amt = (50 * 1.45) + ((nu-50) * 2.85);
else if(nu <=200)
amt = (50*1.45) + (50 * 2.85) + ((nu-100) * 3.95);
else
amt = (50*1.45) + (50*2.85) + (100 * 3.95) + ((nu-200) * 4.50);
sc = 10.00;
pt = nu * 0.06;
```

```
if (ph==1)
{
if (pt<20)
 pt=20;
}
else
{
if (pt<50)
pt=50;
}
} //domestic End Cif)
Else
if(nu<=100)
amt = nu*3.95;
Else
amt = 100*3.95+(nu-100)*7.00)
sc=20.00;
pt = nu*0.66;
if (ph==1)
{
if (pt<50)
pt=50;
}
else
{
if (pt<100)
pt=100;
}
}//(Else) Commercial.
tamt = amt+sc+pt;
pfc"in purpose (c-commercial, D-Domestic): %c",type)
pfc"in Total no. of units : %d",nu);
pfc"in phase type : %d", ph);
pfc"in Bill amount : %.2f", amt);
pfc"in service charge : %.2f", sc);
pfc"in power Tax : %.2f", pt);
pfc"in Total Bill amount : %.2f", amt);
getch();
}
```

Buffer cleaner:-

In a keyboard every key acts as a char. so, when we accepting the char's. we need to clean the buffers before accepting an i/p.

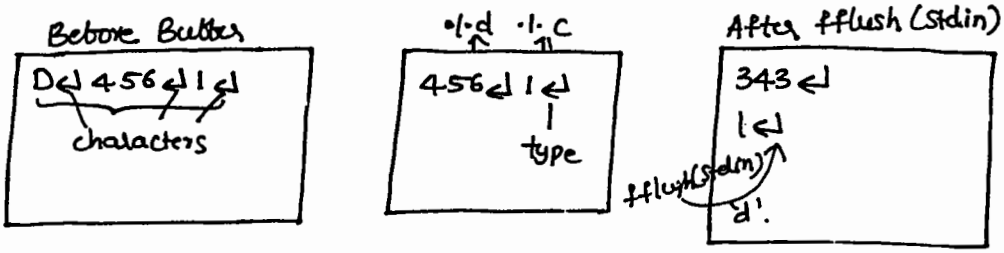
Before accepting i/p of char. type do not use the key Enter, Space bar --- etc. because they accepted as char's.

we can use by cleaning the buffers.

fflush (stdin); → <stdio.h>

(or)

flushall();




```
#include <stdio.h>
Main()
{
int a;
char s;
pf ("Enter the Integer value:");
sf("%d", &a);
pf("Enter the character:");
flushall(); // fflush (stdin);
sf ("%c", &s);
pf ("%d %c", a, s);
getch();
}
```

Output:-

without fflush():-

Enter the Integer value: 343g ← it takes space

enter the character : 343

with fflush():

Enter the Integer value : 74

Enter character : h

74h.

* WAP A shopkeeper allows a commission for the sales person.

Item	Sales amt	Rate of commsn.
CPU	<10,000	nill
	>= 10000 & < 25000	8%
	>= 25000	Rs. 2000 + 10% on sale amt in excess of 25,000.
Monitor	< 10000	5%
	>= 10000	5% upto 10000 + 8% above.

Accept item, amount as an i/p. find out how much commission got to the sales person.

```
Main()
{
int amt;
char item;
float cmsn;
clrscr();
pf("\n\t\t 1. CPU (Cld)");
pf("\n\t\t 2. monitor (M/m)");
pf("Enter the item");
sf("%c", &item);
flushall();
if (Item != 'C' && Item != 'c' && Item != 'M' && Item != 'm')
{
pf("Invalid Item");
getch();
exit(0);
}
if (Item == 'C' || Item == 'c')
{
if(amt < 10000)
cmsn = 0;
Else if (amt >= 10000 || amt < 25000)
cmsn = amt * 8/100;
else
cmsn = (amt - 25000) * 10/100;
}
Else
{
if (Item == 'M' || Item == 'm')
{
if (amt < 10000)
cmsn = amt * 5/100;
else
cmsn = 10000 * 5/100 + (amt - 10000) * 8/100;
}
}
pf("Sales person cmsn is : %.2f", cmsn);
getch();
}
```

Switch :-

Syntax:- Switch (integer variable or integer Expression or character variable)

```
{
case : integer or character constant 1 : stmt(s);
       break;
case : integer or character constant 2 : stmt(s);
       break;
case : integer or character constant 3 : stmt(s);
       break;
.....
.....
case : integer or character constant n : stmt(s);
       break;
default : statement (s);
}
```

The switch() one more decision making stmt inorder to increase the performance of the system we use switch() stmt.

The switch() is more flexible based on the choice some particular case will be Executed.

If the choice is not matched then default stmt will be Executed.

Every case will be Ends with break stmt.

whenever the compiler read the break stmt it will come out of the switch()

"break" & "default" are optional stmts.

The default can be present in any order.

The cases must be "integer type" or "character" it can't be "floating point".

The case & switch() choice must be match, if it is not matched, it will not Execute the case.

Valid Switch() & Cases:-

```
case 100:
case -56:
case 0:
case 26:
case 2+3*4:
case 'A':
case 'a':
```

```
switch(s)
switch(choice)
{
case 121:
}
Switch (2-3*2)
case '+':
```


Invalid:-

```
case 34.56 : // floating pt not allowed
case 'abc' : // more characters not allowed
case "abc" : // not allowed.
case 23 : , case 23 : →//duplicates not allowed
```

case 5,6,4 : // comma not allowed in case, but in switch it is allowed

case 5>6 : // not allowed

switch() // empty not allowed

switch (a,b) // two variables not allowed.

Eg:-
```
Main()
{
int choice ;
clrscr();
pf C"\n\t\t MAIN MENU");
pfC"\n\t\t 100 POLICE");
pfC"\n\t\t 108 EMERGENCY");
pf C"\n\t\t 103 ENQUIRY");
pf C"\n Enter the choice :");
Sf C"%d", &choice);
Switch (choice);
{
case 100 : pf C"POLICE");
           break;
case 108 : pfC"AMBULANCE");
           break;
case 103 : pfC"ENQUIRY");
           break;
debault : pfC"wrong choice");
}
getch();
}
```


WAP Implement the asithmetic operation using switch:-

```
<stdio.h>
Main()
{
int a,b;
char op;
pfC"Enter a&b values :\n");
SfC"%d %d", &a, &b);
PfC" Enter the choice (+,-,*):");
fflush (stdin);
Sf C"%c", &op);
Switch (op)
{
case '/': pf C"division = %d", a/b); break;
Case '+' : pf C"addition = %d", a+b); break;
case '-' : pf C"In Subtraction = %d", a-b); break;
Case '*' : pfC"In Multiplication = %d", a*b); break;
debault : pfC" In wrong operator");
}
getch();
}.
```

* WAP accept day, month, and year. find out & display the week day.

```
Main()
{
int dd,mm,yy;
long dp=0;
pf("Enter day no:");
sf("%d",&dd);
pf("Enter the month no:");
sf("%d",mm);
pf("Enter the year:");
sf("%d",&yy);
dp=(yy-1)*365+(yy-1)/4;
// instead of (yy-1)/4
leap = ((yy-1)/4-(yy-1))/100+(yy-1)/400;
switch(mm)
{
case 12: dp+=30;
case 11 : dp+=31;
case 10 : dp+=30;
case 9 : dp+=31;
case 8 : dp+=31;
case 7 : dp+=30;
case 6 : dp+=31;
case 5 : dp+=30;
case 4 : dp+=31;
case 3 : dp+=28;
case 2 : dp+=31;
case 1 : dp+=dd;
}
if(yy%4==0 && mm>2)
// if((yy%4==0 && yy%100!=0 || yy%400==0) && mm>2)
 dp++;
switch(dp%7)
{
case 1 : pf("sunday"); break;
case 2 : pf("monday"); break;
case 3 : pf("Tuesday"); break;
case 4 : pf("wensday"); break;
case 5 : pf("Thursday"); break;
case 6 : pf("friday"); break;
case 7 : pf("Saturday"); break;
} getch(); }
```

* WAP a company ensures a job as follows

→ if age of the person >25, the marital status is married & gender is male.

→ if age >20, marital status unmarried, gender is male.

→ if age >20, marital status unmarried, gender is female.

If any one of criteria is satisfied he/she eligible for the job.

Note:- use if else condition. (without logical operators).

```
        <stdio.h>
Main()
{
int age;
char state, gen;
pf("Enter the age of person");
sf("%d", &age);
pf("marital status (U/M):");
fflushall();
sf("%c", &state);
fflushall();
pf("Gender:");
sf("%c", &gen);
if(age>=25)
{
if(state='M')
{
if(gen='M')
{
pf("Eligible for Job\n");
}
}
}
else
{
if (20<age>25)
{
if (state='U')
{
if(gen='M')
{
pf("Eligible for Job");
}
else if (gen='F')
{
pf("Eligible for Job")
}
else
pf("Not Eligible for Job");
}
}.
```

* WAP accept a char. find out the given char is vowels & not (using Switch).

```
Main()
{
char ch;
pfc"Enter the char.\n");
fflushall();
sfc" %c",&ch);
switch(ch)
{
case 'a': pfc" It is vowel"); break;
case 'e': pfc" It is vowel"); break;
case 'i': pfc" It is vowel"); break;
case 'o': pfc" It is vowel"); break;
case 'u': pfc" It is vowel"); break;
default: pfc" It is not a vowel");
}
```

* WAP display the o/p like this, 1 2 3 4 5.

```
<stdio.h> <conio.h>
Main()
{
int i=0;
pfc"%d", ++i)
pfc" %d", ++i)
pfc" %d", ++i)
pfc" %d", ++i)
pfc" %d", ++i)
}
```



```
*. Main()
{
pfc" 1 2 3 4 5");
}
```

*. In the above prgm we writing multiple stmts in order to display the o/p.

*. As decision making stmt can checks the condition only one time they can't check more than one time.

*. To overcome this prblms we use iterations (loops).

Unconditional Oncontrol stmts :-

* goto :- Two types of declaration.

forward declaration :

```
goto<labelname>;
Stmt 1;
Stmt 2;
----
<label name>:
Stmt 3;
-----
```

Backward declaration :

```
<labelname>:
stmt1;
stmt2;
---
goto <label name>;
```

```
* Main()
  {
  clrscr();
  X:  pfc("Goto");
      goto X;
      getch();
  }
```

The o/p infinite times takes place it is in unconditional uncontrolled stmt. It doesn't have power of condition checking.

To make this unconditional into Conditional stmt use "if" & "Ternary Operators"

```
*.Main()
  {
  int i=1;
  clrscr();
  X:  if (i<=5)
      {
      pf("Gotoin");
      i++;
      goto x;
      }
      getch();
      }.
```

o/p:. goto
goto
goto
goto
goto

* WAP to display the no. 1 to 100.

```
Main()
{
int i=1;
Natural:  if(i<=100)
          {
          pf ("%d", i);
          i++;
          goto natural; }
          getch();
          }
```

*.
```
Main()
{
int i=1; j=10;
natural: if(i<=10)
         {
         pf("%d",i)
         i++;
         pf("%2d\n", j--)
         goto natural;
         }
getch();
}
```

O/P:-
```
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 3
9 2
10 1
```

(or)
```
int i=1;
natural: if(i<=10)
         {
         pf("%2d %3d\n",
             i, 11-i);
         i++;
         goto natural;
         }
```

* WAP accept a no. find out the sum of Series upto that no's & display the sum.

```
Main()
{
int num, sum;
pf("Enter the no.");
sf("%d", &num);
natural : if (num ==0)
          {
          sum=0
          sum += num;
          num--;
          goto natural;
          }
pf("sum = %d", sum)
}
```

(Or)

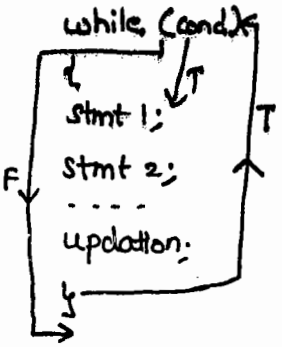
```
Main()
{
int i=1, n, sum=0;
pf("Enter the N:");
sf("%d", &n);
Series : if(i<=n)
         {
         sum = sum+i;
         i++;
         goto Series;
         }
pf("sum of series to 1+2+---+%d: %d", n, sum);
getch();
}
```

Conditional Control Stmts:-

→ Entry Control:-

*. While():-

syntax:
```
initialization;
while (cond)
{
stmt 1;
stmt 2;
----
updation;
}
```



* Main()
{
int i;
clrscr();
i=10;
while (i>=1)
{
pf("%3d", i);
i=i-1; // i--;
}
getche();
}
o/p: 10 9 8 7 6 5 4 3 2 1

* WAP find & display squares & cubes for 1 to 10 numbers.

```
Main()
{
int i=1 ; n;
clrscr();
while (i<=10)
{
n=i;
pf("square : %2d", n*n)
n=i;
pf("cubes : %3d \n", n*n*n);
i++;
}
getch();
}
pf("\n Number \t squares \t cubes \n");
pf("%3d %7d %6d \n", i, i*i, i*i*i);
i++;
}
```


* WAP accept 2 no. find the product of two no. without using multiplication oper.

```
Main()
{
int a, b, prod;
clrscr();
pf("enter the 2 values:");
sf("%d %d", &a, &b);
prod=0;
while (b>=1)
{
prod = prod + a;
b--;
}
pf("product of two no's : %d", prod);
getch();
}
```

design: pf("product of 2 no's %d * %d = %d", a, b, prod);

* WAP accept a no. display the multiplication table for the No:.

```
Main()
{
int n, i=1
clrscr();
pf ("Enter the no. for mul^n table ");
sf ("%d", n);
while (i<=10)
{
pf ("In %d * %d = %d", n, i, n*i)
i++;
}
getch();
}.
```

*WAP accept 2 no's blw the 2 no. display all mul^n table for 'n' terms.

```
Main()
{
int n1, i=1, n2, n;
clrscr();
pf("Enter the 2 no's");
sf("%d %d", &n1, &n2);
while (i<=10)
{
n=n1;
 while (n1<=n2)
{
pf ("%d * %d = %d ", n, i, n*i)
n++
}
i++;
pf("\n");
}
getch();
}.
```



```
* . Void Main()
    {
    int n, i=2, c=0;
    clrscr();
    pf(" In Enter the no:");
    sf("%d", &n);
    while (i<=n && c<=1)
    {
```

```
if (i==2)
pf("Even no's from 1 to %d": i,n);
if (i==1)
pf("In odd no's from 1 to %d:", n);
pf("%d \t",i);
i= i+2;
if (i>n)
{
i=1;
c++;
}
}
getch();
}.
```

Even no: from 1 to 10

2 4 6 8 10

odd no: from 1 to 10

1 3 5 7 9

* WAP to accept a no. find out the given no. is palindrome or not?

```
Main()
{
int num, rev=0, num1
clrscr();
pf("Enter the no:\n");
sf("%d", &num);
num1 =num;
while (num)
{
rev= rev*10 +num%10;
num /=10;
}
if (num1==rev)
pf("No. is palindrome");
Else
pf("No. is not palindrome");
getch();
}.
```


* WAP accept a no. & display the reverse of that no.

```
Main()
{
int num,rev=0;
pf("Enter the no.");
sf("%d", &num);
while (num) // num>0 or num!=0
{
rev = rev*10 + (num%10)
num /=10;
}
pf(" Reverse : %d",rev);
getch();
}.
```

*WAP accept a no. find out the given no. is Armstrong Number.

$153 = 1^3+5^3+3^3 = 153$

```
Main()
{
int num, ans=0, cube, org;
pf("Enter the nos");
sf("%d", &num);
org=num;
while (num)
{
cube = num%10;
ans = ans + (cube*cube*cube);
num /= 10;
}
pf("In Armstrong no. of the %d: %d", org, ans);
getch();
}
```

*WAP Accept a no. find out the factors of that no., display the factors. & also find out the given no. is prime or Composite no's:-

```
Main()
{
int num, halb, check, prime
pf("Enter the num");
sf("%d", &num);
halb = num/2;
pf("factors of %d :", num);
while (i<= halb)
{
check = num %i;
i++;
if (check==0)
{
pf("%d", check);
}
}
if (num==2)
{
pf("In No. is prime");
}
i=2;
while (i<=num)
{
prime = num %i;
if (prime==0)
{
if(i=num)
{
pf("It is prime\n");
}
else
{
pf("It is Composite");
i = num+1;
}
}
else
i++;
}
```

A looping Concepts Can be written in 3 ways.

1. with body 2. without body 3. with Semicolon (;)

With Body:-

```
Main()
{
int i=1;
while (i<=10).
{
pf ("\n%d",i);
i++
}
}
```

No Replacement by compiler

o/p: 1,2,3 ... 10.

whenever we written with body there is no need of any replacement for the prgm.

Without Body:.

```
Main()
{
int i=1;
while (i<=10)
pf("\n%d",i)
i++;
}
```

o/p: 1 11 11 ---- infinity.

Replacement:

```
while (i<=10)
{
pf ("\n%d",i)
}
i++;
```

At this time the Compiler makes replacement with body but within the body only one stmt takes place.

```
*. Main()
{
int i=0;
while (i++<10)
Pf ("%d",i);
getch();
}
```

o/p: 1 2 3 4 5 6 7 8 9 10

With Semicolon (;)

whenever we define the stmt with Semicolon, the Compiler replace with Empty body i.e; there is no stmts.

```
Main()
{ int i=1;
while (i<=10);
pf ("\n%d", i);
}
```

Replacement:-

```
while (i<=10)
{
}
pf ("\n%d",i);
```

——— > No stmt, no o/p infinity

```
* Main()
{
int a=5;
while(a)
{
pf("%d",a);
a=a-1;
}
pf("%d",a+10);
```

O/P: 5 4 3 2 1 10

```
*Main()
{
int a,b;
clrscr();
a=b=1;
while(a)
{
a=b<=3;
b=b+1;
pf("%d%d",a,b)
}
pf("\n%d%d",a+10,b+10);
}
```

O/P: 12 13 14 05
10 15.

```
*. Main()
{
int a;
clrscr();
a=3;
while(a--)
pf("%d",a)
pf("%d",a+10);
}
```

O/P: 2 1 0 9

Replacement:

```
while(a--)
{
pf("%d",a)
}
pf("%d",a+10)
```

```
*. Main()
{
int a;
clrscr();
while(a++<=1)
while(a++<=2);
pf("%d",a);
}
```

O/P: 5.

Replacement.

```
while(a++<=1)
{
while(a++<=2)
{
}
}
pf("%d",a)
```

```
*. Main()
{
int a,b;
clrscr();
a=b=1;
while(a)
{
a=b++<=3
pf("%d%d",a,b);
}
pf("\n%d%d",a+10,b+10);
}
```

O/P: 12 13 14 05
10 15

```
*. Main()
{
int a=1
clrscr();
while(a++<=3)
pf("%d",a);
pf("%d",a+10);
}
```

O/P: 2 3 4 15

Replacement.

```
while(a++<=3)
{
pf("%d",a);
}
pf("%d",a+10);
```

```
* Main()
{
int a=1;
clrscr();
while(a++<=1)
while(a++<=2)
pf("%d",a);
}
```

O/P:- 3

Replacement:

```
while(a++<=1)
{
while(a++<=2)
{
pf("%d",a);
}
}
```

```
*.Main()
{
int a;
clrscr();
a=1;
while(a-->=1)
while(a-->=10);
pf("%d",a);
}
```

O/P: -3

Replacement:

```
while(a-->=1)
{
while(a-->=10)
{
}
pf("%d",a);
```

```
*. Void main()
{
int a;
clrscr();
a=1;
while (a<=1)
if (a%2)
pf ("%d",a++)
else
pf ("%d",++a)
pf ("%d",a+10);
}
```

O/P: 1 2
2 3

Replacement:.

```
while (a<=1)
{
if (a%2)
{
pf ("%d",a++);
}
else {
pf ("%d",++a).
}
}
pf ("%d", a+10)
```

(*) .

```
Void main()
{
int a=1;
while (a++>=1);
pf ("%d",a);
}
```

O/P: −32768

Replacement.

```
while (a++>=1)
{
}
pf ("%d",a);
```

*.

```
Void Main()
{
int a=-1
clrscr();
while (a--);
pf ("%d",a);
}
```

O/P:. −1

Replacement.

```
while (a--)
{
}
pf ("%d",a);
```

* WAP accept 2 no's as base & Exponent find the power value of the base (similar to prod. of 2 no's).


```
Main()
{
int base, expo, pv=1
pf ("Enter the base & Exponent");
sf ("%d %d", &base, &expo);
while (expo)
{
pv= pv* base;
expo--;
}
pf ("In The power value: %d", pv);
getch();
}
```

* WAP to accept a no. find the some of that nos until that no. becomes single digit.

```
Main()
{
int num, sum=0;
pf ("Enter the number");
sf ("%d", &num);
loop: while (num)
```

```
{
sum = sum + (num % 10);
num /= 10;
}
if (sizeof(sum)==1)
{
pf ("final single digit : %d", sum)
}
else
{
num = sum;
goto loop;
}
getch();
}
```

9336
21
3

Break():-

The Break() is an unconditional controlled stmt.

The Break() will be used in switch() & inside the loop only.

The Break() "can't be used only in 'if' condition".

when the compiler reads the break stmt it immediately come out of the loop.

*. Main()

```
{
----
while (cond)
{
----
if (cond)
break;
----
} // end of loop
----
}
```

*. Main()

```
{
int i=1;
while (i<=10)
{
if (i>4)
break;
pf("%d", i);
i++; }
pf ("%d", i+10);
}
```

o/p: 1 2 3 4 15

*. void main()

```
{
int i;
i=1;
while (i<=20)
{
pf ("%d", i)
if (i>3)
break;
i++;
}
}
```

o/p: 1 2 3 4

*. Void Main()
```
int i;
i=2;
while (i<=20)
{
pf("%d",i);
i+=2 // i=i+2
 if (i>=8)
break;
}
```
O/P:- 2 4 6.

*. Void main()
```
{
int i;
i=15;
while(i>=5)
{
i--;
if(i<=10)
break;
pf("%d",i);
}
}.
```
O/P:- 14 13 12 11

*. Void main()
```
{
int i;
i=1;
while (i<=10)
{
pf("%d",i);
if(i!=i)
break;
i+=2;
}
}
```
O/P:- 1 3 5 7 9

*. Void main()
```
{
int i;
i=1;
while (i<=15)
{
pf("%d",i)
if(i!=15) // if (i!=15)
break;
i+=2;
}
}.
```
1

*. Void main()
```
{
int i;
i=50;
while(i>=10);
{
pf("%d",i)
if(i<=40)
break;
i-=5;
}
}.
```
Error., if break should not be in if.

Replacement.
```
i=50;
while (i>=10)
{
}
pf("%d",i)
if (i<=40)
break;
i-=25;
}
}
```

Continue():-

The continue() also unconditional control stmt.

Whenever the compiler read the continue() stmt it immediately re enter inside the loop.

continue() can be used only inside the loop.

```
Main()
{
----
while (cond)
{
--
...
if (cond)
... T
continue;
----
} // loop end.
}
```

* WAP accept 10 no. find out biggest no. the condition not to accept any -ve no's.

```
Main()
{
int i, num, big=0;
clrscr();
while (i<=10)
{
pf ("Enter the no. %d: ", i);
sf ("%d", &num);
if (num<0)
{
pf ("Negative no's are not allowed");
getch();
continue;
}
if (num > big)
big = num;
i++;
} // loop
pf ("Biggest : %d", big);
getch();
}
```


*.
```
Main()
{
int i=10;
while (i<100)
{
i = i+2;
if (i>40 & i<60)
continue;
pf ("%d", i)
}
}
```

op: 2 4 68 ..... 60, 62 ... 98 100

*.
```
Void Main()
{
int i;
i=80;
while (i>=2)
{
i-=2;
if (i>30 && i<60)
continue;
pf ("%d", i);
}
}
```

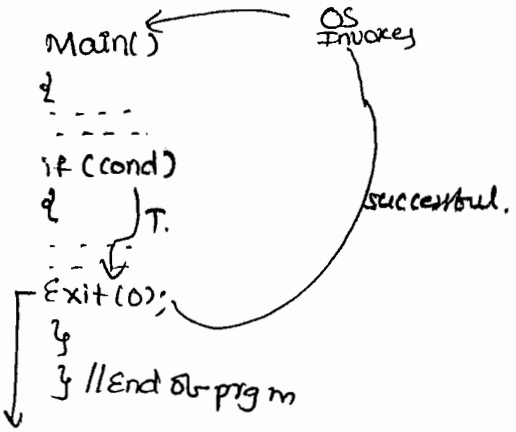
op: 78 76 74 ..... 62 60 ... 32 ... 2

*.
```
Main()
{
int i=1;
while (i<=20)
{
if (i>4 && i<=17)
continue;
pf ("%d", i);
i++;
}
}
```

keep on checking.

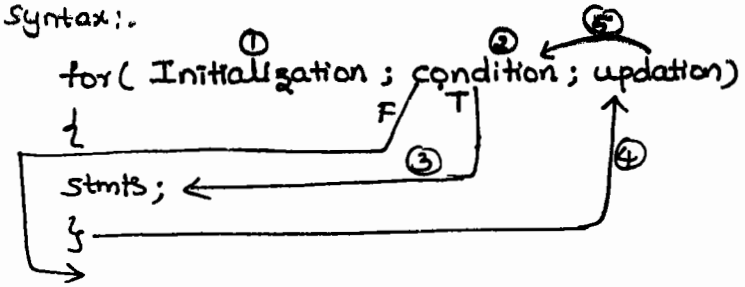
Exit():-

The Exit() stmt makes you to come out frm the prgm.



for():-

Syntax:.



The for() is also Entry control looping stmt.

The for() is more flexible when we compose with while() loop.

The for() will rotates in Anti-clockwise direction.

In for() Everything is optional i.e; we can write a stmt for(;;)

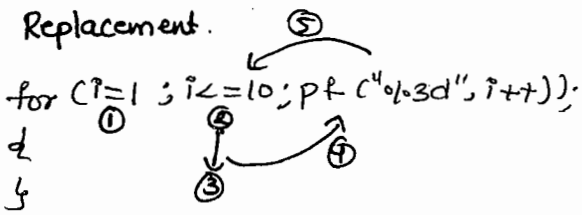
In for() we can place any no. of initializations, by seperating with comma (,) any no. of cond's & updation stmts.

Eg:. for (i=1, j=1; i<=5, j<=10; i++, j++)

The for() can also can write with body, without body & with Semicolon(;).

*.
```
Main()
{
int i;
for(i=1; i<=10; pf("%3d", i++));
getch();
}
```

Replacement.



*!
```
Main()
{
int x=0, j, k;
clrscr();
for(k=0; j=5; k<5; j>0; k=k+2, j--)
{ x++;
pf("\n The values of k, j, x is %d %d %d", k, j, x); } }
```

O/p:.

```
0 5 1
2 4 2
4 3 3
6 2 4
8 1 5
```

* in this it is checking only last condition i.e; j>0 either any no. of conditions pr. it checks only last one.

* If the loop is written with conditions,

```
for(k=0, i=5; k<5 && j>0; k=k+2, j--)
```

Then, o/p is = 0 5 1
2 4 2
4 3 3

∴ Both conditions have to satisfy.

If it is with ||(OR).

```
for (k=0, i=5; k<5 || j>0; k=k+2, j--)
```

o/p: 0 5 1.
2 4 2.
4 3 3.
6 2 4.
8 1 5.

*.
```
Main()
{
int a,b;
for(a=b=1; a; pf("\n%d %d",a,b))
a=b++<=3;
pf("\n%d %d", a+10, b+10);
}
```

o/p: 1 2 1 3 1 4 0 5
10 15

*.
```
Main()
{
int i=0;
for(;;) → rotating the loop // Infinite loop.
{
pf("%d", i);
i++;
if(i>5) → checking the condition.
break;
}
}
```

o/p: 0 1 2 3 4 5

*.
```
Main()
{
int a=1;
for(a++; a++<=2; a++)
for(a++; a++<=7; a++)
a++;
pf("%d",a);
}
```

o/p: 1 3

Replacement:-
```
for(a++; a++<=2; a++)
{
for(a++; a++<=7; a++)
{
a++;
}
}
pf("%d", a);
```

a [1] 2 3 4 5 6 7 8 9 10 11 12 13

* program :-

	C_1	C_2	C_3	C_4
R_1	1	1	1	1
R_2	2	2	2	2
R_3	3	3	3	3
R_4	4	4	4	4

Rows ↓ Top to Bottom

columns → left to Right.

→ first consider no. of Rows & no. of columns.

→ find out Initial & final values of Rows & columns

→ In this Major point the column condition.

→ The column condition can be variable & fixed value

→ In our above eg. the columns are fixed values, every row the column is changing then the condition is a variable.

→
```
Main()
{
int i,j;
for(i=1;i<=4;i++)
{
for(j=1;j<=4;j++)
pf("%d",i);
pf("\n");
}
}
```

*
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
Main()
{
int i,j;
for(i=1;i<=5;i++)
{
for(j=1;j<=i;j++)
pf("%d",j)
pf("\n");
getch();
}
}
```

*
```
00 01 02
10 11 12
20 21 22
```

```
Main()
{
int i,j;
for(i=0;i<=2;i++)
{
for(j=0;j<=2;j++)
{
pf("%d%d\t",i,j);
}
pf("\n");
getch();
}
}
```

*
```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
Main()
{
int i,j,k=5
for(i=1;i<=5;i++)
{
for(j=1;j<=k;j++)
pf("%d",j);
}
k--;
pf("\n");
getch();
}
```

2nd logic:-

```
for(i=1;i<=5;i++)
{
for(j=1;j<=6-i;j++)
pf("%d",j)
pf("\n");
}
```

3rd logic:-

```
for(i=5;i>0;i--)
{
for(j=1;j<=i;j++)
pf("%d",j);
pf("\n");
}
```

*
```
1
2 3
4 5 6
7 8 9 10
```

```
Main()
{
int i,j,K=1;
for(i=1;i<=4;i++)
{
for(j=1;j<=i;j++)
{
Pf("%d",K); K++; }
Pfc("\n");
getch();
}
```

*
```
        1
      1 2
    1 2 3
  1 2 3 4
1 2 3 4 5
```

```
Main()
{
int i,j,n;
Pf("Enter the Row value:");
Sfc("%d",&n);
for(i=1;i<=n;i++) //Row
{
j=n;
while(j>i) //spaces
{
Pfc(" ");
j--;
}
for(j=1;j<=i;j++) //col.
Pf("%d",j);
Pfc("\n");
}
getch();
}.
```

*
```
4 3 2 1
3 2 1
2 1
1
```

```
Main()
{
int i,j;
for(i=4;i>0;i--)
{
for(j=4;j>0;j--)
Pfc("%d",j);
Pfc("\n");
}
```

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

```
Main()
{
int i,j,sn;
char ch='*';
Pfc("Enter the value:");
Sfc("%d",&n);
for(i=1;i<=n;i++)
{
s=1
while(s<=38-i*2) //space
{
Pf(" ");
s++;
}
for(j=1;j<=i;j++) //col
{
Text color(9); // Text color(6+128)
                  ↳ To add colours to o/p's.
                  ↳ blinking
Cprintf("%c",ch);
}
Pfc("\n");
}
getch();
}.
```

*. while we are adding colors to o/p we have to take the cpf() stmt.

*. for textcolor() fun. we require the console.h header file.

*
```
5 5 5 5 5
5 5 5 5 5
5 5 5 5 5
```

```
Main()
{
int i,j,k=5;
for(i=1;i<=3;i++)
{
for(j=1;j<5;j++)
pf("in ",k);
pf("\n");
}
}
```

```
for(i=1;i<=15;i++)
{
pf("5");
if(i%5==0)
pf("\n");
}
```

*
```
1 1 1 1 1
1 2 2 2 2
1 2 3 3 3
1 2 3 4 4
1 2 3 4 5
```

```
for(i=1;i<=5;i++)
{
for(j=1;j<=5;j++)
if(i>j)
pf("%d",j);
else
pf("%d",i);
}
pf("\n");
```

*.
```
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

```
Main()
{
int i,j,n;
for(i=1;i<=n;i++)
{
for(j=1;j<=5;j++)
{
if(i==1||i==5||j==1||j=5)
pf(" i");
else
pf("0");
}
pf("\n");
}
```

*
```
5                   5
5 4                 4 5
5 4 5             3 4 5
5 4 3 2       2 3 4 5
5 4 3 2 1 1 2 3 4 5
```

```
Main()
{
int i,j,r,k,s;
pf(" Enter the rows:");
sf("%d",&r);
{ i=1;
while(i<=r) //row
{
k=r;
while ( k>=r-i+1 ) // left col's
{
pf("%d",k);
k--;
}
for(s=1;s<=(r-i)*2;s++)//spaces
pf(" ");
j= r-r+1;
while(j<=r) // right col's
{
pf("%d",j);
j++;
}
i++;
pf("\n");
getch();
}
```

* WAP to display all armstrong nds upto 1000 nds:

```
int ams=0; i,num;
for(i=0; i<=1000;i++)
{
a=i;
while(i)
{
cube = i%10
ams = ams + (cube*cube*cube);
num/=10
}
if(a==ams && a<=1000)
pf("%d",a)
getch();
}
```

* WAP display all the ASCII char's with a no's:

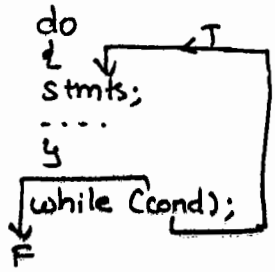
```
Main()
{
int i;
char ch;
pf("ASCII char... NO:");
for(i=0; i<=255; i++)
{
ch=i;
pf("\n %c---%d", ch, i)
}
getch();
}
```

* WAP accept a decimal no. & display the no - in binary format.

```
Main()
{
 int num;
for(i= num ; i>=1 ; i=i/2)
{
a = i%2;
pf("%d",a)
}
```


Do While :-

```
initialization;
do
{
stmts;
....
}
while (cond);
```



The dowhile is an Exit controlled looping stmt first it Entry into the body, Executes at least one time the stmt.

And then it check the condition.

If condition is satisfied once again enters into the body, this process continues untill condition dissatisfied.

In dowhile the while() must have Semicolon.

Do while can also write in 3 ways.

- with body.
- without body,
- with Semicolon.

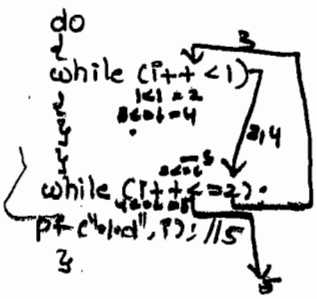
whenever the prgm is replacing after the do body must & should exist & within the body only one stmt will takesplace immediately body will close & after the body a while condition must exist with semicolon.

*. Main()
{
int i=10;
do
pf("%d", --i);
while(--i);
}

Replacement:-
do
{
pf("%d", --i); // 9,7,5,3,1.
}
while(--i);

*. Main()
{
int i=1;
do
while(i++<1);
while(i++<=2);
pf("%d", i);
}

Replacement:-



*.
Main()
{
int i=1;
do
while(i++<=1);
while(i++<=3);
while(i++<=5);
pf("%d", i);
}

do
{
while(i++<=1)
1<1=2
4<1=5
{
}
}
while(i++<=3);
while(i++<=5)
{
}
pf("%d", i); //4.
5<=3=6
2<=3=3
3<=6=4
6<=5=7

* WAP accept ano. find out the given no. is prime or not?

<stdio.h> <conio.h>
Main()
{
int num, k, f;
clrscr();
pf("Enter the element \n");
sf("%d", &num);
for(i=2; i<num; i++)
{
if(num%i==0)
{
status=1;
}
}
pf("not prime\n");
else
pf("%d prime no\n");
}

* WAP accept 2 no's blw the 2 no's display all the prime no's?

<math.h> <stdio.h> <conio.h>
int main()
{
long int n1, n2, n, cnt=0, t, flag;
clrscr();
pf("Enter the 2 no:");
sf("%ld %ld", &n1, &n2);

```
for (n=n1 ; n<=n2 ; n++)
{
//for (flag=1, t=2, t<n; t++)
for (flag=1, t=2 ; t<=sqrt(n) ; t++)
{
if (n%t ==0)
{
flag =0;
break;
} //if
} //for.
if (flag==1)
pf ("\n %2d prime = %5d", ++cnt, n);
}//for
getch();
} //main() end.
```


Number System :-

0---9 decimal base → 10

0---7 Octal base → 8

0---15 Hexadecimal base → 16 [0-9, 10-A, 11-B, 12-C, 13-D, 14-E, 15-F]

0---1 Binary base → 2

* Main()

```
{
int a=13;
clrscr();
pf ("\n %d", a); //13
pf ("\n %o", a); //15
pf ("\n %#o", a); //015
pf ("\n %x", a); //d
pf ("\n %#X", a); //0XD
pf ("\n %#x", a); //0xd
pf ("\n %b", a); // error, No format of binary.
getch();
}
```

%d	%o	%x
100	0144	0X64
30	036	0X1E
20	024	0X14

⊛
```
  0743
  0124
  0625
  ---
  111
 01714
```

⊛
```
  0476
- 0237
------
  0237
```

⊛
```
  0x174
  0xa2d
+ 0x93b
--------
  0x14dc
```

⊛
```
  0x9db
  0x6e5
+ 0xabc
--------
  0x1b7c
```

⊛
```
  0x732
  0x2ac
--------
  0x486
```

⊛
```
  0586
  0124
--------
Invalid Octal
```

1 → 1	$2^1 - 1$ → 1	→ 1
2 → 10	$2^2 - 1$ → 3	→ 11
3 → 11	$2^3 - 1$ → 7	→ 111
4 → 100	$2^4 - 1$ → 15	→ 1111
5 → 101	$2^5 - 1$ → 31	→ 11111
6 → 110	$2^6 - 1$ → 63	→ 111111
7 → 111	$2^7 - 1$ → 127	→ 1111111
8 → 1000	$2^8 - 1$ → 255	→ 11111111
9 → 1001		
10 → 1010		
97 → 1100001		

```
  11011
- 10110
-------
  00101
```

```
  11011
  10101
  11001
 -----
 1001001
```


Whenever a var. is declared it occupies some bytes of memory.
for suppose char ch; it occupies 1 byte of memory space an internally allocated with memory range of 0 to 127 & -1 to -128.
for every part of data we have binary codes which represented in 16-bit format.

```
0000 0000 0000 0000  → 0
0111 1111 1111 1111  → 32767
1000 0000 0000 0000  → -32768
1111 1111 1111 1111  → -1
```

we can easily can extract the reverse combination & also the binary formats.

```
0    reverse combination  -1
1      "          "       -2
-100   "          "       99
```

-5:
```
A  → 0000 0000 0000 0100
-5 → 1111 1111 1111 1011
```

-100:
```
99   → 0000 0000 0110 0011
-100 → 1111 1111 1001 1100
```

* 32767+1

```
0111 1111 1111 1111
0000 0000 0000 0001
-------------------
1000 0000 0000 0000  = -32768
```

* -32768-1

```
1000 0000 0000 0000
0000 0000 0000 0001
-------------------
0111 1111 1111 1111  = 32767
```

* -32768+-1

```
1000 0000 0000 0000
1111 1111 1111 1111
-------------------
0111 1111 1111 1111  = 32767
```

*
```
Main()
{
pf("%d %u %0 %2", -1,-1,-1,-1);
}
```
O/P:- -1, 65,536, 32,767, 0XFFFF.

By this binary format is same but Extenally the no's are different.

*
```
Main()
{
unsigned u;
for(u=0; u>=0; u--)
pf("%d ---- %0 ----- %0 --- %x\n", u,u,u,u);
}
```
O/P:- -1 65535 177777 ffff.

*
```
Main()
{
int a=10+010+0x10+0X10;
pf("%d", a); //50
}
```

*
```
Main()
{ clrscr();
pf("%0, %d", -32768, -32768);
}
```
O/P: 32768, -1

In this Example, -32768 its taking long byte i.e; sizeof 4 bytes & the binary code in 32-bits is,

1111 1111 1111 1111 1000 0000 0000 0000

Due to ⊖ve no.'s will be added

Due to %u 2 byte will occupy for the first 16 bits & remaining 16-bits is substituting in place of %d that no. equivalence to -1.

* pf("%u %d %0", -32768, -32768, -32768);

This comes in octal format

O/P:- 32768, -1, ...

Bitwise Operators:-

The Bitwise operators will be used in order to manipulate the bits & for digital processing, flip flops.

for all this purpose we use bitwise operators

1. 1's complement 2. Leftshift 3. Right shift
4. Bitwise AND 5. Bitwise Exclusive OR 6. Bitwise OR.

1's Complement:- Bitwise operators never work with floating type datatype.

for Encryption & decryption 1's complement will be used.

It is nothing but Reverse Combination.

Eg:-
```
Main()
{
clrscr();
pf ("%d", ~5); //-5
getch();
}
```

* ~0 → -1
* ~~0 → 0
* pf("%d", ~32768); //-32769.

Left shift (<<):-

		0000 0000 0000 0101	→ 5
5<<1	5x2¹	0000 0000 0000 1010	→ 10
5<<2	5x2²	0000 0000 0001 0100	→ 20
5<<3	5x2³	0000 0000 0010 1000	→ 40
5<<4	5x2⁴	0000 0000 0101 0000	→ 80
5<<5	5x2⁵	0000 0000 1010 0000	→ 160
11<<2	11x2²	0000 0000 0010 1100	→ 44

Right shift (>>):-

	0000	0000 0001 0100	→ 20
20>>1	0000	0000 0000 1010	→ 10 20x2¹
20>>2	0000	0000 0000 0101	→ 5 20x2²
20>>3	0000	0000 0000 0010	→ 2 20x2³
20>>4	0000	0000 0000 0001	→ 1 20x2⁴
20>>5	0000	0000 0000 0000	→ 0 20x2⁵
7>>2	0000	0000 0000 0001	→ 1 7x2²

* -10>>1

Any no. 16 times << leads to Zero(0)

If it is an odd no. 15 times left shift leads to -32768.

If it is an Even no. 15 times left shift leads to Zero(0).

If its a $\ominus^{ve}$ no. right shift above 15 times leads to -1.

Bitwise AND(&), OR(|), EX-OR(^) these operators are useful for checking the bits ON or OFF in the flip-flops & gates.. etc.

Bitwise AND(&)

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR(|)

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise EX-OR(^)

A	B	R
0	0	0
0	1	1
1	0	1
1	1	0

Eg:-

8 → 00.....00 1000 (12)

10 → 00.....00 1010

8&10 → 00.....00 1000 = 8

8|10 → 00.....00 1010 = 10

8^10 → 00.....00 0010 = 2

*. if x=10 y=12 then $x^1=y^1=x^1=y$

also x=? y=?

x - 1010

y - 1100

x 0110

y 1100

* Multiplication without using arithmetic operators:-

```
#include<stdio.h>
Main()
{
int a,b,result;
pf("\n Enter the 2nd's:");
sf("%d%d",&a,&b);
result=0
while(b!=0)
{
if(b&01)
result = result+a;
a<<=1;  in 'a' by 1.
b>>=1;  in 'b' by 1.
}
pf("\n Result : %d",result);
}.
```

* WAP accept a no. find out the given no. is Even or odd.

```
Main()
{
int num,y;
pfc" Enter the no");
sfc"%d",&num);
y=num<<15;
if (y==0)
pf c" Given no is Even");
Else
pfc" Given no. is odd");
getch();
}
```


*. Which Bitwise operator is suitable for checking wheather particular bit is ON/OFF.

Ans:- Bitwise AND (&) operator.

Eg:- Suppose in a byte that has a value 10101101.

we wish to check wheather bit no.5 is ON or OFF. Since we want to check the bit no.3, the 2nd operand for AND operation we use is binary 00001000, which is equal to 8 in decimal.

ANDing Operation.

```
10101101 — Original bit pattern.
00001000 — AND mask.
---------
00001000 — Resulting bit pattern.
---------
```

The resulting value we get in this case is 8. i.e; the value of second operand. The result turned out to be 8 since the 3rd bit of operand was ON. Had it been off. The bit no.3 in the resulting bit pattern would have evaluated to 0 & complete bit pattern would have ben 00000000.

Thus depending upon the bit no. to be checked in the first operand we decide the 2nd operand on ANDing these two operands the result.

* Which bitwise operator is suitable for turning OFF a particular bit in a no.?

Ans:- Bitwise AND operator & is complement operator.

Eg:. TO unset the 4th bit of byte-data or to turn off a particular bit in a no.

consider, char byte-data = 0b00010111;

byte_data = (byte-data) & (~(1<<4));

It can be rep^d in binary as 0b00000001 = (1<<4). << is a left bit shift Operator. It shift the bit 1 by 4 places towards left (1<<4) becomes 0b00010000. And ~ is the 1's complement operator in c-lang.

So ~(1<<4) = complement of 0b00010000 = 0b11101111.

Replacing value of byte-data & ~(1<<4) in (byte-data) & (~(1<<4)), we get (ob00010111) & (ob11101111)

* Perform AND operation to below bytes.

```
00010111
11101111
--------
00000111   Thus 4th bit is unset.
```

Applications of Bitwise Operators:-

database
6MB

Date:-
day - 2 byte
month - 2 bytes } 6 byte
year - 2 byte

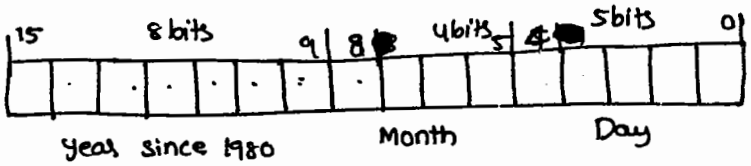
6/6 = 1MB.

char dd; } 2 byte
char mm; }
int yy; } 2 byte

6/4 = 1.5 MB

int data; } 2 byte

6/2 = 3 MB



The DOS and windows converts the actual dates into 2 bytes by using the formula.

date = 512 * (year - 1980) + 32 * month + day;

2^9 - year start from 9th bit, 2^5 - starts from 5th bit.

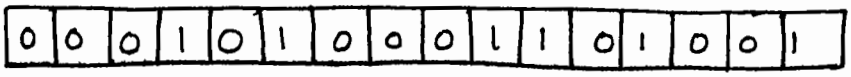
Suppose we accepted date 09/03/1990 then the conversion

date = 512 * (yy - 1980) + 32 * mm + day.

date = 512 * (1990 - 1980) + 32 * 3 + 9

= 512 * 10 + 32 * 3 + 9 = 5225.

5225 = 0001 0100 0110 1001



The binary equivalent of 5225 is passed in the date filled of the directory when we use dir in windows explorer the data format dd/mm/yy it will be takes place by using left & Right shift operator.

At this point the year, month & day is bunch of bits.

we have to separate the bits by using shift operator.

Extracting an year:.

5225 = year 0001010 month 00110 day 1001 >>9

0000 0000 0000 1010 = 10

year = 1980 + (date >> 9)

Extracting a month:-

1. first left shift 7 times to clear the year.
2. & Then Right shift by 12 times to calculate the month

```
0001 0100 0110 1001     <<7
0011 0100 1000 0000     >>12
0000 0000 0000 0011      3
```

month = (date <<7) >>12

Extracting a day:-

day = (date<<11) >>11

Eg:- 04|05|2014

date = 512 * (2014 − 1980) + 32 × 3 + 4

= 17508

17508 = 0010 0010 0011 00100

year = 01000 10001100100 >>9

0100010 − 34

year = 1980 + (date >>9)

month = 0100010001100100 <<7

0011 00100 0000000 >>12

0000 0000 0000 0011 → 3 (month)

month = (date <<7) >>12

day = 0100010001100100 <<11

0010000000000000 >>11

0000 0000 0000 0100 ← 4 (day)

day = (date<<11) >>11

FUNCTIONS

A fun. is an selb debined block it poborms predebined task.

Fun's in C-lang. are classified in two categeories:

1. pre debined functions (library functions):-

   Defined by the compiler.

   Eg: printf(), scanb(), clrscr(), getch()

2. User debined functions:-

   Debined by the user Eg:- Main().

* Why Main() is called user debined function?

We debining the functionalities ob the main() as per the user req's we providing the functionalities for the main() prgm, this fun'lities ~~for the ff~~ can be change as per the user.

This is one reason why main() is called as user debined function.

As W.K.T. Every operator returns the value similarly the fun's also returns a value but recieving the value is optional.

By debault Every fun. return type is integer.

This return types can be changes, if it is an user debined function.

```
float Main()
{
pf ("%d", sizeob (main));
getch();
}
```

4 bytes

Due to this reason also we can call the main is an user debined function.

The Return type ob main() can be changes per user requirement.

* What is the necessary to debine the "return type" for the Main():

we can also store a value for a fun.

when we are not returning a value, it raises a warning " The fun. should Return a value".

To overcome This warning, These are 2 ways:

1. Kill the nature ob the fun.
2. Return value to the fun.

To kill the nature ob the function we have void data type.

Void:- void is a data type which kills the nature ob the fun.

```
Void Main()          Main()
{                    {
}                    }
```

(Void: no return type)

Main(): by debault return type is integer.

```
Void Main()
{
return 0;      // Invalid it is not possible to return a value.
getch();
}
```

Always Do not Kill the Nature of the fun.

To write the user defined fun's, we have to follow the rules. The fun. contains 8 components.

1. Fun. Declaration (Fun. prototype).
2. Signature of a fun.
3. Fun. caller.
4. Actual Arguments.
5. Fun. callee.
6. Formal parameters.
7. Fun. Definition.
8. return type.

Function Declaration :-

The fun. Declaration is also called as prototype of fun. The declaration Tells to the compiler the return type of the fun, No. of arg's of the fun. Type of arg's & order of arg's.

This declaration is necessary for predefined fun. as well as user defined fun.

The Return type & arg's of the fun's will be changes depends upon the fun.

Syntax:-

```
return_type function_name (arg1, arg2, ---- argn);
   ↓                        ↑ I/P's
  o/p                       └ signature of a fun. ┘
```

The return type always depends on o/p of the prgm

The arg's always depends on no. of i/p's to the prgm.

Ex's:- * `float SUM(float, float);` (float: ↓o/p; float, float: ↓I/P)

* `int reverse (int);`
* `void swap (int, int);`

Note:- whenever we have to return more than one value. it is not possible to return the value at that time as per your req! we can kill nature of the fun.

Declaration Eg's:-

* void check(int);
* void prod(int, int);
* float Avg-3(float, float, float)
* void prime(int);
* void biggest(int, int, int);
* void fib(int);

Note:- Declaration of the fun. is optional in C-lang. but it is mandatory in C++.

Function Caller:-

The fun. caller is another component in that fun.

A caller can present any no. of times whenever the caller is takes place is immediately jumps to the Submodule (callee).

At the time of fun. caller we pass the arg's that arg's are called as actual arg's.

The fun. caller can pr. anywhere in the prgm.

Actual Arguments:-

The purpose of the actual arg's for the comm^n from caller to the callee.

The actual arg's can be var. types, Expression & constant no's.

The actual arg's pass values of the variable.

Valid Caller	Invalid Caller
Swap(5, 10); // constant no's	int swap(a,b);
swap(a+b, ++a); // Expression	swap(int, int);
swap(a, b); // var's.	swap(int a, int b);

Function callee:-

It is a Separate module which contains the formal parameters. It can be called as submodule.

It will be define only one time. But usage any no. of times.

The Submodule contains the defn. of the fun.

The defn. is nthg but logic of the body.

Formal parameters:-

The formal parameter purpose to recieve the values frm the actual arg's. This formal parameter is must be variable Type.

* Void swap(int, int); // declaration

```
Main()
{
---
swap(5, 10); // callee
---
}
```

(5, 10) → actual args

```
Void swap(int x, int y) // callee
{                  formal parameters
---  } fun. defn.
}
```

Functions can written in four ways :-

1. fun. with no arg., no return type. Eg:- clrscr();.
2. fun. with arg, no return type. Eg:- gotoxy().
3. fun. with arg, with return type. Eg:- pf(), sf().
4. fun. with no arg., with return type. Eg :- getch().

1. Function With No argument, No return Value :-

```
Void printline(void);
Void xyz();          // declaration.

Int main()
{
clrscr();
printline(); // callee
pf("It is a fun. oriented \n");
xyz(); // callee
pf(" Back to main()\n");
printline(); // callee.
getch();
return 0;
}
Void printline() //callee
{
int i;
for(i=1; i<=40; i++, pf("_"));
pf("\n");
}
Void xyz() // callee
{
pf("It is sub module \n");
}
```


O/P:-

```
It is a fun. Oriented.
It is Sub module.
Back to Main()
```

Advantages of functions :-

By using fun's we can find the modulasity of the prgm.
Debugging of the prgm is Easy to implement.
The Major adv. it provides Source code reusability.
Basically the pgm Execution always 4m Main(), Compilation 4m Top to Bottom.

// Test1

```
int main()
{
clrscr();
xyz(); //callee
getch();
return 0;
}
Void xyz() //callee
{
pf ("It is sub module\n");
}
```

It raises an Error since compilation happen before, before the compilation it doesn't find the declaration of that callee.

// Test 2

```
void xyz() // callee
{
pf ("It is sub module\n");
}
int main()
{
clrscr();
xyz();
getch();
return 0;
}
```

It doesn't raises any Error before Execution only the compiler already knows the defn of the fun. So, it doesn't raises any Errors.

Note:- clrscr():- 1. clear the Screen
2. Makes Cursor 1st Row & 1st coloumn.

// Test 3

```
Void xyz(); //declaration
Main()
{
xyz(); //callee
}
xyz() //callee
{
pf ("Hello");
}
```

The declaration of the callee return type is not Matching.

// Test 4

```
Void xyz(); //declaration
Main()
{
xyz();
return 0;
}
void xyz() //callee
{
pf ("Hello");
return;
}
```

There is no Errors, we are not returning any value, we are returning only control.

// Test 5

```
Void printline(); //declaration
Main()
{
int i;
printline(); //callee
return 0;
}
void printline() // callee
{
for (i=0; i<10; i++)
pf ("_");
}
```

It raises an Error of undefined symbol (i). In which module we are using the variable in the same module the variable has to declare.
(Here i scope Ends in Main Only).

Example On clrscr();-

```
Void clrscr(); //declaration
Main()
{
pf("hello\n");
clrscr(); //callee.
getch();
return 0;
}
Void clrscr()//callee
{
int i,j;
for(i=1; i<=50; i++)
{
for (j=1;j<=80;j++, pf(" "));
}
gotoxy(1,1)
}
```

col - row

Fun. With Argument & No return value :-

In this categeory there is a comm^n. from the caller to the callee but there is no comm^n 4m callee to the caller.

* WAP accept 2 no's & implement the swapping logic.

```
Void swap (int, int); //declaration
Main()
{
int a,b;
clrscr();
pf("Enter the value :");
sf("%d%d", &a,&b);
swap(a,b);
getch();
}
Void swap (int x, int y) //callee
{
x^=y^=x^=y;
pf("After swaping", x: %d y: %d ", x,y)
}
```


* Find out given no. is even or odd.

```
Void check (int); //declaration
Main()
{
int a;
clrscr();
```

```
pf ("Enter the num");
sf ("%d",&a);
check(a); //caller
getch();
}
Void check (int x)// callee
{
int y;
if (x%2 ==0)
pf ("Number is Even");
pf ("Number is odd");
}
```

```
if ( (n&1)==0)
pf ("Even");
else
pf ("odd");
```

```
y = n<<15;
if (y==0)
pf("Even");
else
pf("odd");
```

* WAP accept a no. up to that no. display all the natural no's:

```
Void natnum (int); //declaration
Main()
{
int num;
clrscr();
pf ("Enter the num");
sf("%d", &num);
Natnum (num); //caller
getch();
}
void natnum (int x) //callee
{
int i;
for (i=1; i<=x; i++)
{
pf ("%d", i);
}
}
```


* WAP accept a no. display the number in words:

```
<stdio.h>
Main()
{
int txt(long); // declaration
long int t, num;
clrscr();
pf (" Enter the num :");
sf ("%ld", &num);
if (num==0)
pf ("Zero");
if (num>=10000000)
{
t=num/10000000;
if (t<=20)
txt(t);
else
{
txt (t/10*10); // caller 2
txt (t%10); // caller 3
}
txt (10000000); // caller 4.
}
num = num%10000000;
```

```
if (num >= 100000)
{
t= num/100000;
if (t<=20)
 txt(t); //caller 5
Else
{
txt (t/10*10);//caller 6
 txt (t%10); //caller 7
}
txt (100000); // caller 8
}
num = num %. 100000;
 if (num >= 1000)
{
t= num/1000;
if(t<=20)
txt(t); //caller 9
else
{
txt (t/10*10); //caller 10
txt (t%10); //caller 11
}
txt(1000); // caller 12
}
num =num%. 1000;
if (num >= 100)
{
txt (num/100); //caller 13
txt(100) ; // caller 14
}
 num =num%. 100;
if (num<= 20)
txt (num); // caller 15
else
{
 txt (num/10 *10); //caller 16
txt (num%10); // caller 17.
}
getch();
}
txt (long num) // fun. declaration.
{
Switch (num)
{
```

```
case 1: pf ("one"); break;
case 2: pf("Two"); break;
case 3: pf ("Three"); break;
case 4: pf ("four"); break;
case 5: pf ("five"); break;
case 6: pf ("six"); break;
case 7: pf ("seven"); break;
case 8 : pf ("Eight"); break;
case 9 : pf ("nine"); break;
case 10: pf ("Ten"); break;
        :
        :
case 20 : pf ("Twenty"); break;
case 30 : pf ("Thirty"); break;
case 40 : pf ("fourty"); break;
case 50 : pf ("fifty"); break;
case 60 : pf ("sixty"); break;
case 70 : pf ("seventy"); break;
case 80 : pf("Eighty"); break;
case 90 : pf("Ninty"); break;
case 100 : pf ("Hundred"); break;
case 1000 : pf ("Thousand"); break;
case 100000 : pf ("Lakh"); break;
case 10000000 : pf ("crore"); break;
}
}
```

Fun. with Arg's & return type:-

In this Categeory comm^n takes place from the both sides, i.e; from the calles & from the callee.

* WAP accept a no. find the factorial of that numbes?

```
long fact(int); //declaration
Main()
{
int num;
long K;
pf("Enter the numbes");
sf("%d", &num);
K = fact(num); //calles
pf("factorial of %d is:%ld", num, K);
return 0;
getch();
}
long fact(int num) //callee
{
long s=1;
if(num==0||num=0)
return 1;
for( ; num>=2; s=s*num--)
return s;
}
```

Instruction	Address	Value
fact	65510 65511 65512 65513	40320
num	65514 65515	8
K	65516 65517 65518 65519	40320
num	65520 65521	8
s	65522 65523 65524 65525	40320

* Addition

```
long sum(int, int); //declaration
Main()
{
int a,b, long S;
pf("Enter the 2 nds:");
sf("%d%d",&a,&b);
S=sum(a,b); //calles
pf("sum of 2 nds is: %ld",s);
return 0;
getch();
}
long sum(int x, int y) // callee
{ long z;
z=x+y; return z; }
```

Subtraction:-

```
z = x + ~y - 1;
```

addition :-

```
z = x - ~y - 1;
```

* WAP accept a no. & display that no. in binary format.

```
<stdio.h> < math.h>
unsigned long dtob(int);
Main()
{
int n;
clrscr();
pf("enter the decimal no:");
sf("%d", &n);
pf("Binary equivalent of %d is %ln", dtob(n), n);
getch();
return 0;
}
unsigned long bin=0, p=0;
while (n)
{
bin = bin + (n%2)* pow(10,p);
n/=2;
p++;
}
return bin;
}
```

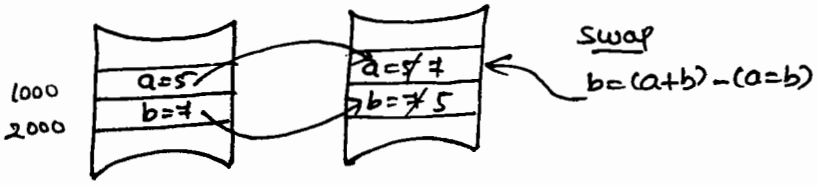
Function calls:-

A fun. calls in C- long. will be takes place in two ways.

1. Call by value (default) 2. Call by Address.

1. Call by Value:-

Sending the value of var. frm caller to the callee is called call by value. Any changes made in the fun. it will not effect to the caller fun. var. values.



*. void swap(int,int)
void main()
{
int a,b;
pf("Enter the no. a&b");
sf("%d %d", &a,&b);
swap(a,b);
pf("a: %d & b: %d", a,b);
getch();
}
void swap(int a, int b)
{ b = (a+b) - (a=b);
pf("after swaping a: %d b: %d", a,b);
}

*. char c1, c2, c3, c4;
int i1, i2, i3, i4;
float f1, f2, f3, f4;

Declaration	calling
int sum(int, int);	i1 = sum(i2, i3);
Void sum(int, int);	sum(i1, i2);
float sum(float, float);	f1 = sum(f2, f3);
char sum(char, int, float);	c = sum(c2, i1, f4)
Void Anusha();	Anusha();
int Anusha(void);	i1 = Anusha();
Void abc(int, char);	abc(i1, c1);
float Anu(char, int, float, float);	f1 = Anu(c1, i1, f2, f3);

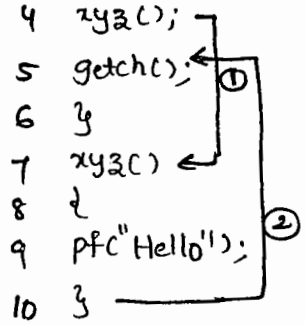
Trying to return more than One value by using return Statement

```
Calsum(a,25,d);
calsum(10+2, 25-13, d);
Calsum(a, calsum(25,10,4), d); // Nested fun. call.
Calsum(a,25,d);
calsum(a,25,d)+28;
// Returning More than 1 value
Main()
{
int a=10, b=20, c=30;
int s,p;
s,p = sumprod(a,b,c); // Invalid left side only one variable.
pf("%d %d", s,p);
}
Sumprod(int x, int y, int z)
{
int ss, pp;
ss = x+y+z;
pp = x*y*z;
return(ss, pp); // not possible.
}
// One More Try
Main()
{
int a=10, b=20, c=30;
int s,p;
s = sumprod(a,b,c);
```

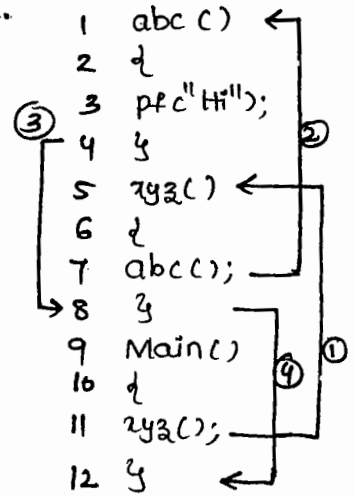
```
p= sumprod (a,b,c);
pf ("%d %d %d", s,p);
}
sumprod (int x, int y, int z)
{
int ss, pp;
ss = x+y+z;
pp = x*y*z;
return (ss); // Reduclant
return (pp);
}
// The only Way out
Main()
{
int a=10, b=20, c=30; int s,p;
s = sumprod (a,b,c,1);
p = sumprod (a,b,c,2);
pf ("%d %d", s,p);
}
sumprod (int x, int y, int z, int code)
{
int ss, pp;
ss = x+y+z;  pp = x*y*z;
if (code == 1)
return ss;
Else
return (pp);
}
// better Way
return (code == 1 ? x+y+z : x*y*z);
// Invalid.
code == 1 ? return (x+y+z) : return (x*y*z);
```

Recursion :-

*.
```
Line 1  Main()
     2  {
     3  clrscr();
     4  xyz();
     5  getch();
     6  }
     7  xyz()
     8  {
     9  printf("Hello");
    10  }
```

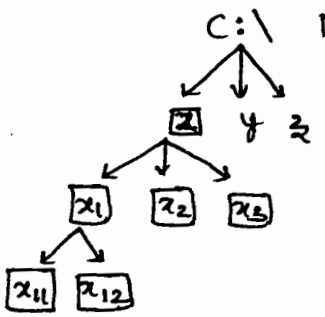


*.
```
 1  abc()
 2  {
 3  printf("Hi");
 4  }
 5  xyz()
 6  {
 7  abc();
 8  }
 9  Main()
10  {
11  xyz();
12  }
```



The function which calls by itself is called as Recursion.

Recursion is one of the beautiful process whenever we doesn't know some particular condition recursion will be implemented.



Recursion

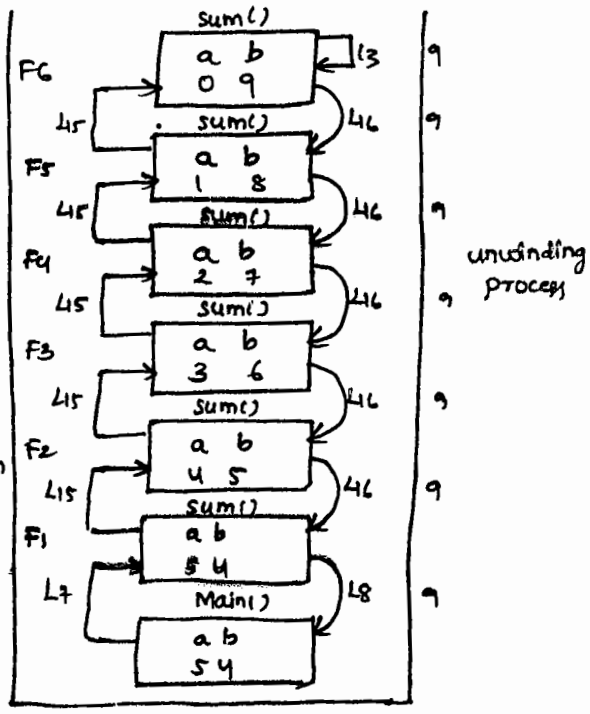
* WAP Addition of two nos Using Recursion :-

```
 1  int sum(int, int);
 2  Main()
 3  {
 4  int a,b;
 5  printf("Enter the values:");
 6  scf("%d %d", &a, &b);
 7  printf("sum of two nos: %d", sum(a,b));
 8  getch();
 9  }
10  int sum(int a, int b)
11  {
12  if(a==0)
13  return b;
14  else
15  sum(--a, ++b);
16  }
```

winding process

Recursion

unwinding process



Runtime stack

"The internal discipline of recursion will be takes place on stack process. It is called as Runtime Stack".

Any normal fun. will also works on similar way of Recursion.

Fun. Execution will be takes place based on CPU (os).

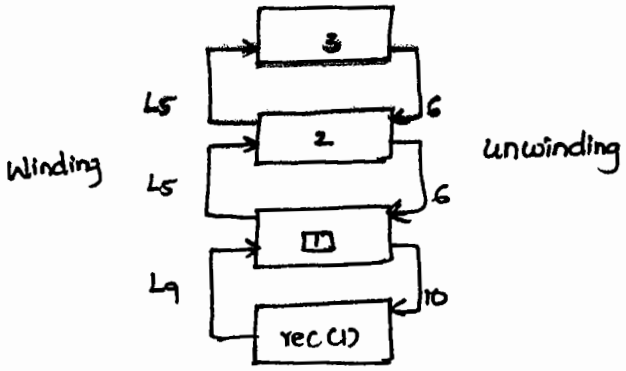
The drawback of the fun. when more no. of fun. calls are increasing, the peformance of the system will be degrade

The real time appn of a recursion.

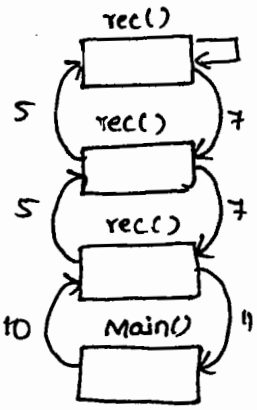
Start → Search → Files & Folders.

when Recursion Examples implementing solve the prgms with Line nos.

*
```
1  Void rec (int i)
2  {
3  pfc("\n%d",i);
4  if (i<==2)
5  rec (i+1);
6  }
7  Void main()
8  {
9  rec(1);
10 }
```
o/p:- 1 2 3



*.
```
1  Void rec (int i)
2  {
3  pfc("\n%d",i);
4  if (i<=2)
5  rec (i+1)
6  pfc("%d",i);
7  }
8  Void main()
9  {
10 rec(1);
11 }
```



1 2 3 3 2 1

Differences between iterations and recursion:-

I→ It is a process of Execution a stmt or set of stmt repeatedly untill Specified condition.

R→ Recursion is the technique of defining anything in terms of itself.

I→ Iterations involve four clear steps initialization, condition, Execution & Updation.

R→ There must be an include if stmt inside the recursion function specifying stoping condition.

I→ Any Recursive prblm can be solved iteratively.

R→ Not all the prblms have Recursive solutions.

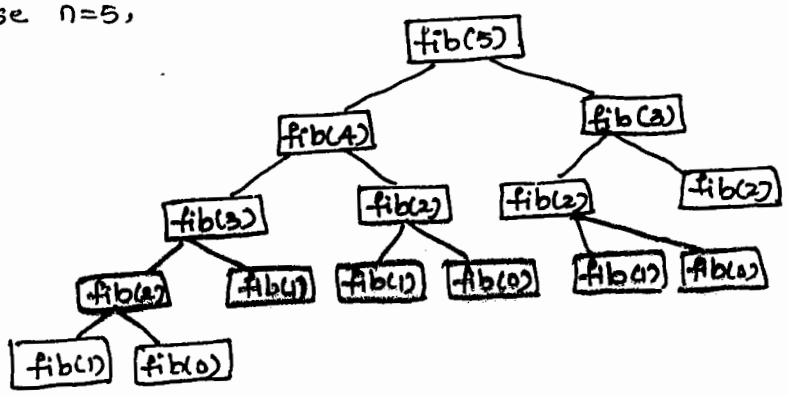
I → Iterative counter part of a prblm is more efficient in terms of memory, initialization & execution speed.

R → Recursion is generally worst option to go for simple prgms are prblms not recursive in nature.

* WAP all the recursion slow down the performance of the slm.

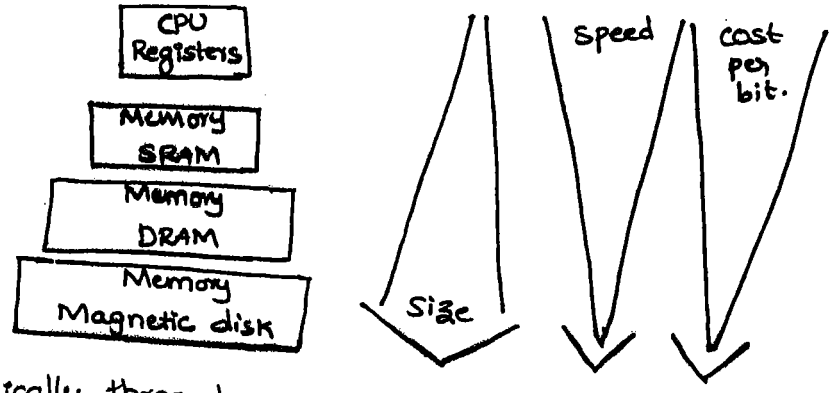
```
long fibo(int);
Main()
{
int n,r;
pf("Enter the row size:");
sf("%d",&r);
for(n=0; n<r; n++)
pf("%ld\n", fibo(n));
getch();
}
long int fibo(int n)
{
long fib;
if(n==0)
fib=0;
Else
 if(n==1)
fib=1;
 Else
fib= fibo(n-1) + fibo(n-2);
return fib;
}
```


Suppose n=5,



Memory Organization:

Structure of Memory Heirarchy:-



Basically three temporary memory areas Reg's. RAM and Cache Memory.

Data will be store in RAM area as we consider RAM area there are diff. parts available in the RAM area.

Our data will be takes place usually in the RAM process

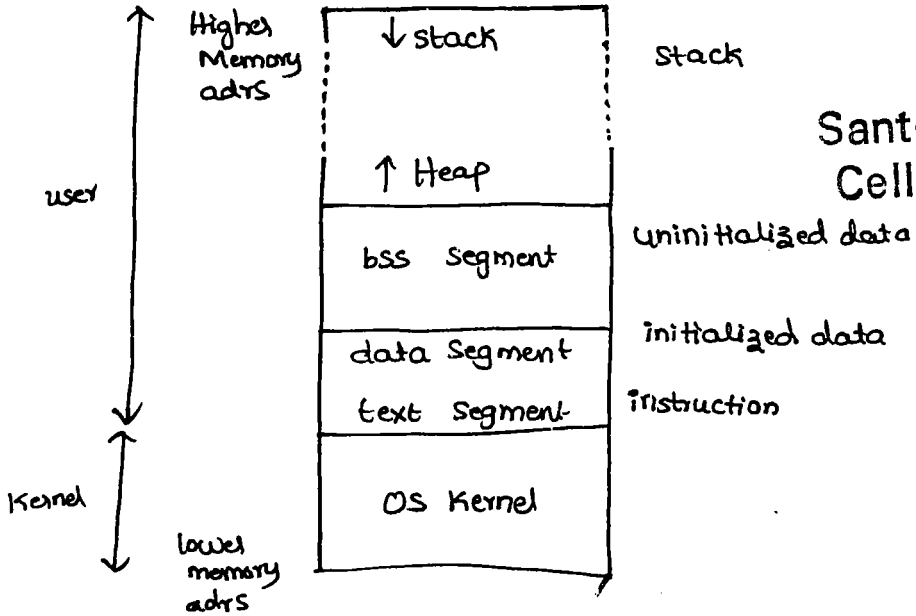
It can be static RAM or dynamic RAM.

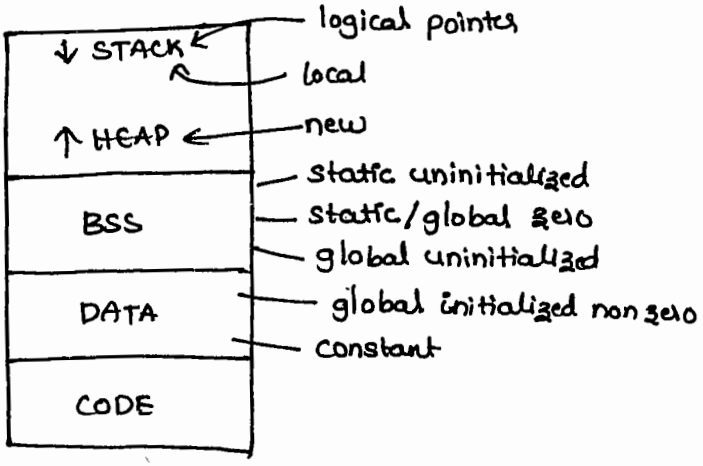
What is RAM memory?

The Entire RAM has devided in no. of Equal parts which are known as "Memory cells".

Basically a RAM contains diff. types of areas diff. parts of data of the C-prgm is loaded in the Memory.

Every diff. types of variables is located in diff. locations.





As per the Orgn in diff. parts data will be Store & diff. initialization will be takes place.

The RAM contains each cell, can store 1 byte of data, data will be stored in binary format, the char. data reserves one byte of memory space floating point of data will reserve 4 memory cell, each memory cell has unique adrs. Adrs is always in ↑ing order of Hexadecimal format, with combination of Integers.

for suppose, int a=4;

In this, variable Stores in the memory in the following way.

00000100	00000000
5000	?

Actual binary bits for 4

0000 0000 0000 0100
higher memory Lower memory

But in the Memory storage first lower memory is storing & then higher Memory Stores.

Since it is under the concept of "Little Endian" (concept of Storing lower byte 1st & higher Memory).

Little Endian:- In which intel processors will arrange

If data is stored first higher byte & then lower bytes then that is "Big Endian".

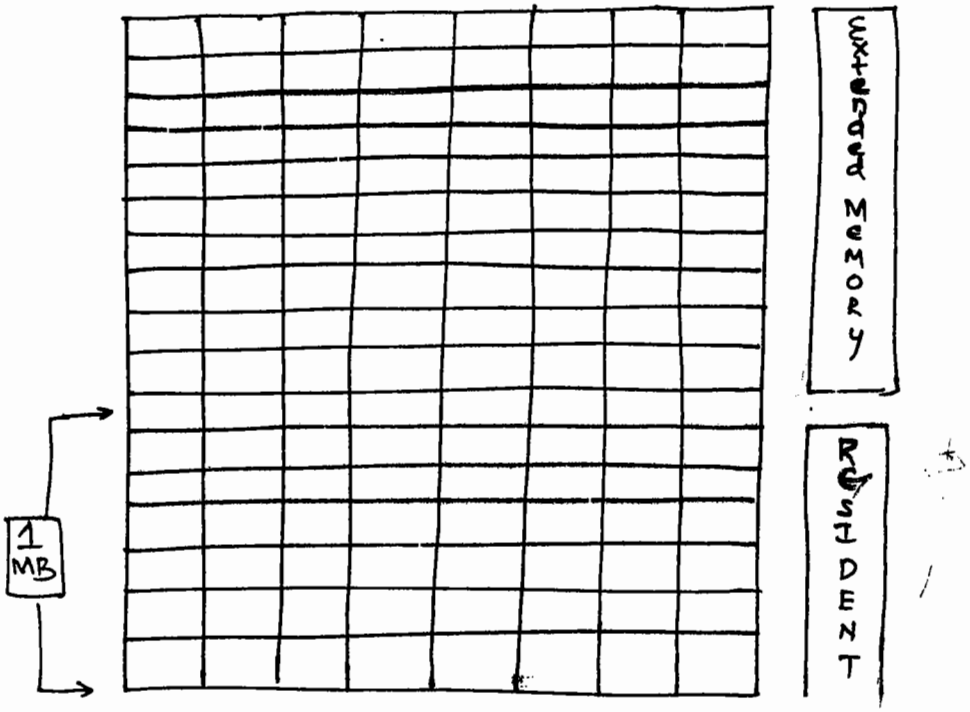
Eg:- Motorola processors (big Endian).

In this Memory adrs of the 1st cell 0X5000 then next memory adrs 0X5001 since integer data always stores in continuous memory location as we know every adrs always in ↑ing order.

What is an Resident Memory!

RAM is devided into 2 parts

1. Extended Memory (useless).
2. Resident Memory.



In TURBOC 3.0 the compiler size of Resident memory is 1 MB.

When any prgm is executed it is store in Resident Memory for TURBOC 3.0, it has 1 MB resident memory which store in the RAM when we open TURBOC.

Physical Adress of the Computer:-

All the C-val's are stored in the Resident Memory in TURBOC 3.0, 20 bits adrs of the memory cell is known as "physical Address". & "Real Address".

In 20 bits we can represent the address from 0X0000 to 0XFFFF. i.e; all C-variables must have the Memory adrs within this range.

A C-prgmr can't decide what will be the memory adrs. it is decided by compiler.


```
Main()
{
int a;
pf("%x", &a);
return 0;
}
```

we can't predict but we can see in 16-bit compilers adrs must be within 0X0000 to 0XFFFF.

In 32 bits Compilers 0X00000000 to 0XFFFFFFFF,

Suppose your C-compiler based on the Microprocessor total.

Then it total size of the addressable Memory,

$$\text{Addressable Memory} = 2^{(\text{Adrsbus})} \text{ bytes}$$

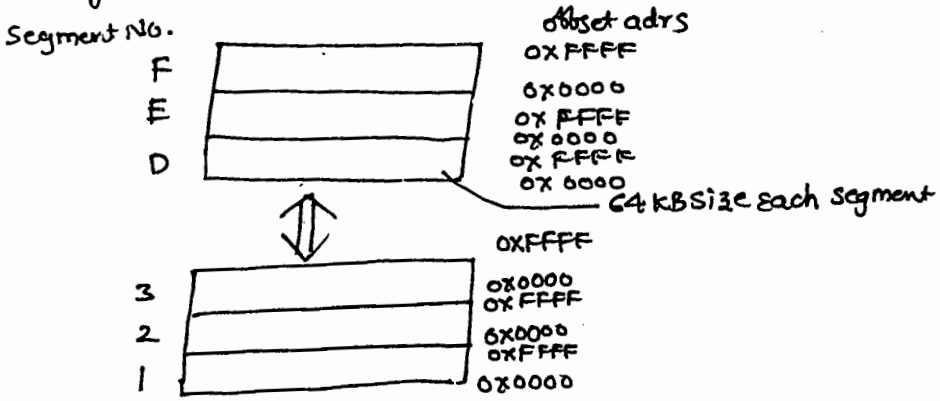
i.e; 2^{32} bytes = 4 GB

8 bits = 1 byte
1024 bytes = 1 KB
1024 KB = 1 MB
1024(1024x1024x1024) MB = 1 GB.

Memory Segments:-

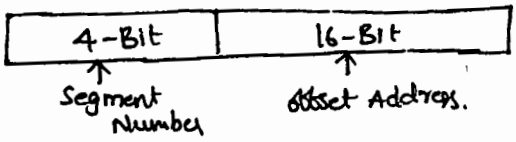
Resident Memory of RAM of size 1MB is divided into 16 equal parts. This parts is called "Segment".

Each Segment size is 64 KB, 16X64 KB = 1024 KB = 1MB.



This process of division is called Segmentation.

What is an offset address?



So, in other words we can say that the Memory adrs in C has 2 parts, Segment number & offset adrs.

Suppose physical adrs of any var. in C is 0X 500F1 then segment number is 5 and offset adrs is 00F1.

*.
```
Main()
{
int x;
clrscr();
pf("%u", &x);    // offset adrs
pf("\n %p", x);  // Segment adrs
pf("\n %p", &x); // offset adrs in hexadecimal.
getch();
}
```

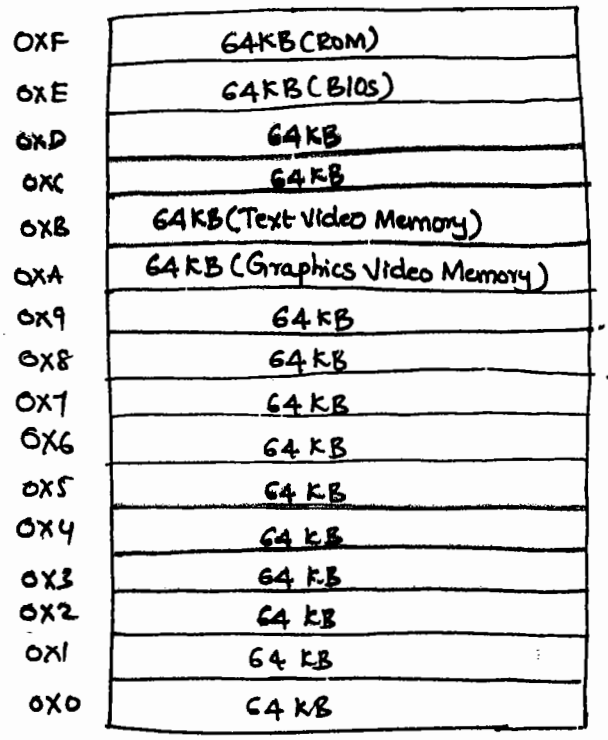
O/P: 65524
8FEB
FFF4.

Data Segment in C :-

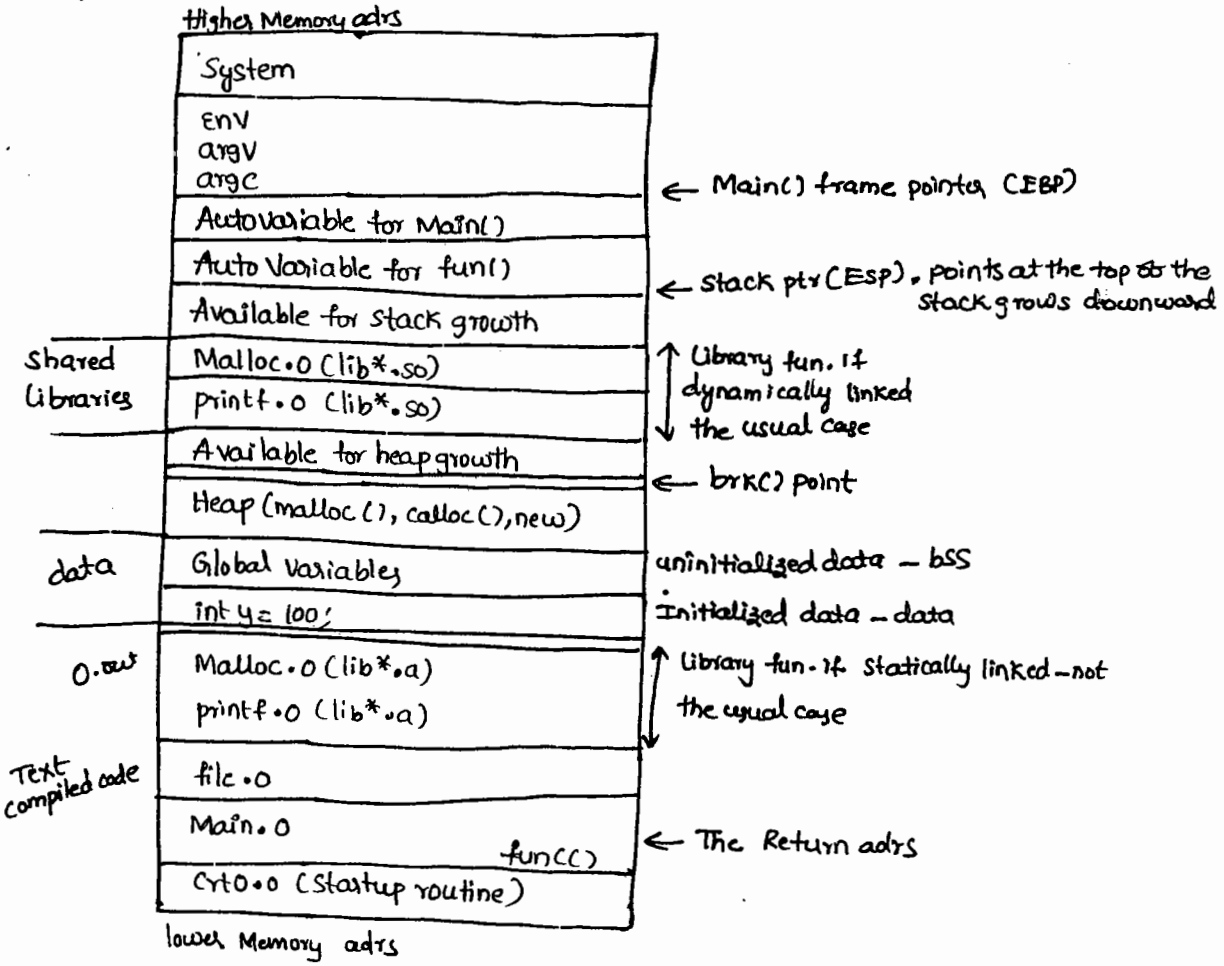
All the Segments are used for specific purpose like Segment NO.15 is used for ROM: 15 → ROM 14 → BIOS, 8 → data segment ···· Etc.

Segment no.8 which is special name, known as data segment, This segment is devided into major four parts. Thus Representation we already Mentioned in below diagram.

16 Segments of Resident Memory



The Major areas Code Area, Data Area, Heap Area & Stack Area.



1. CODE AREA:.

The prgm code is where the executable code available for the execution. This area also known as Text Segment & it is fixed size.

It can be access only by the fun. pointers not by other data pointers.

2. DATA AREA:

All the static & global variables stored in data area, whatever the data pr. in the data area throughout the life time of prgm will be takes place.

The Data area can be called as permanent memory adrs.

whatever the data pr. in the data area there will be available throughout the End of the prgm.

In this segment there are 2 parts are present. They are initialized & Non initialized.

If the var. is initialized to some values other than zero, they allocated with initialized segment.

When var's uninitialized they allocated uninitialized segment.

when we want to run an Executable prgm the OS starts a prgm known as "Loader". when it loads the file into memory it takes the bss segment & initialized to zero's.

This area can be choosed by the user depends on the Storage classes.

Basically there are 4 types of storage classes.

1. Automatic storage class
2. Static storage class.
3. Global (External) Storage class
4. Register storage class.

They are five types of " Storage class Specifiers".

1. Auto 2. Static 3. Extern 4. Register & 5. typedef.

Storage class	Specifier	Memory location	Initial value	scope	Life
Automatic	Auto	Stack (Temp. area)	Garbage	within the body	Entire body
Static	Static	data area (PMA)	Zero	within the function	Entire body pgm
External	Extern	data area	Zero	All the functions	Entire prgm
Register	Register	register	Garbage	within the body	Entire body

What is a Scope?

Scope refers to where in a prgm a variable may be access.

Eg:-
```
Void Sample()
{
a++;
}
Void main()
{
int a=10;
Sample();
Sample();
Sample();
pf("\n%d",a);
}
```

o/p:-
error, since scope of "a" is pr. only in the Main() module.

This Scope can changes if the storage class changes by default it is automatic Storage class.

*.
```
Main()
{
→ fun'al scope.
}
```

*.
```
Main()
{
if()
{ → Conditional scope
}
}
```

*.
```
Main()
{
for(i=0;...)
{ → Iteration Scope
}
}
```

*.
```
Main()
{
{
int a; → Anonymous Scope.
}
}
```

Example for life:-

Life refers to how long a variable exist or retains its value.

```
Main()
{
int a; // fun'al scope.
a=10;
pf("\n%d",a); //10
{
int b; // anonymous scope
b=20;
a++; //11
b++; //21
pf("\n%d",a); //11
pf("\n%d",b); //21
}
pf("\n%d",a); //11
pf("\n%d",b); //error.
}
```

3. STACK AREA :-

All automatic variables & constants are stored into stack area,

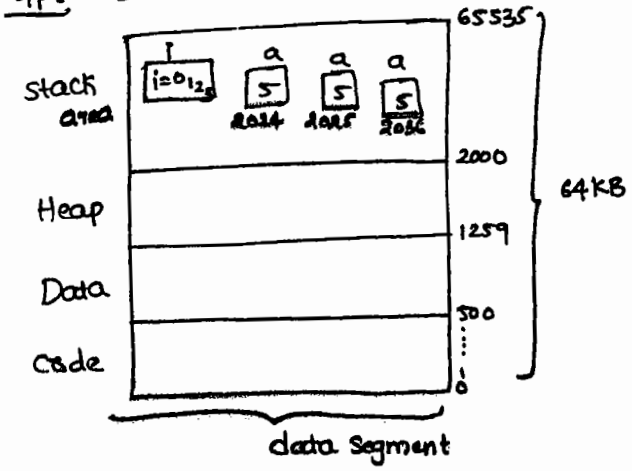
Automatic Variables & constants are

1. All the local var's dr default storage class
2. Var's dr storage classes auto,
3. Integer constant, character constants, String constants, float constants ---- etc, in any Expressions.
4. fun. parameters & fun. return value var's in the stack area are always deleted when the prgm controlled reaches to the out scope.

Due to this "stack area" also called as "Temporary memory area".

```
Main()
{
int i; // fun. Scope.
for (i=0; i<3; i++)
{
int a=5; // iteration Scope.
pf ("%d",a);
}
getch();
}
```


o/p:- 5 5 5



Once body close iteration scope will die & when "i" value increase it Creates / initializes the variable.

Variable is automatic variable storing in stack area, scope variable 'a' is within the for loop so, after each iteration variable 'a' is will be deleted from stack and in each iteration variable will initialize.

*.

```
<stdio.h>
int main()
{
int a=5, b=6, c=7, d=8;
pf("%d %d %d %d");
return 0;
}
```

↳ format Specification

o/p:- 8 7 6

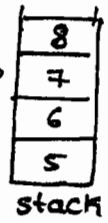
The TURBO compiler and based

The default storage classes if all the var's is auto, since it is automatic

Var. is stored in the stack area of the Memory.

stack follows the discipline LIFO

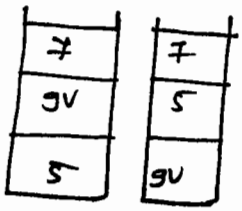
*. It will store in stack as →



In the pf() name of the var's is not written explicitly so the default o/p contain of stack which will be in LIFO order. i.e; 8 7 6.

has 2 parts one for initialization var. & other one non-initialized var., All initialized Var. are more near than non initialized variable.

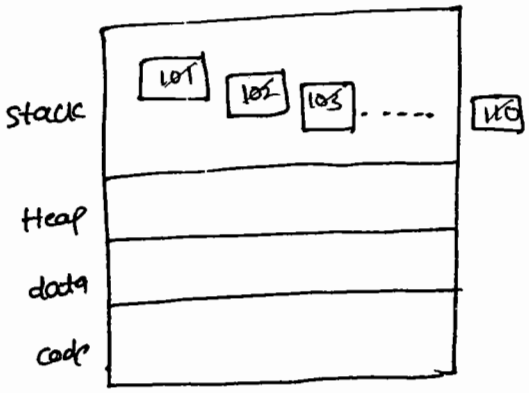
*
```
<stdio.h>
int main()
{
int a=5, b, c=7;   → non-initialized.
pf ("%d %d %d");
return 0;
}.
```
o/p:: 7 5 gv.



Automatic var. a&c are initialized & b is not initialized.

Initialized var's are more near than uninitialized, They will be stored in the stack, due to "LIFO" 7&5. since 'a' is more near than b w.r.to 'c'.

*.
```
Main()
{
int i;
for(i=1; i<=10; i++)
pf ("%d %d\n", i, fun(i));
getch();
}
fun (int i)
{
int s;
s=100;
st=i;
return s;
}
```
o/p:. 101 102 103 104 105 106
107 108 109 110



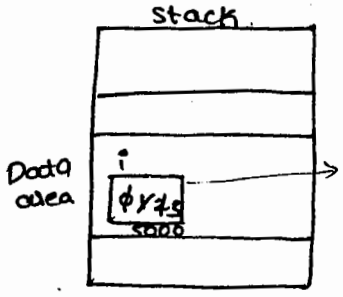
Note:. Automatic var's every time will initialize (every time scope will die).

Static :-

Eg for life on static:-

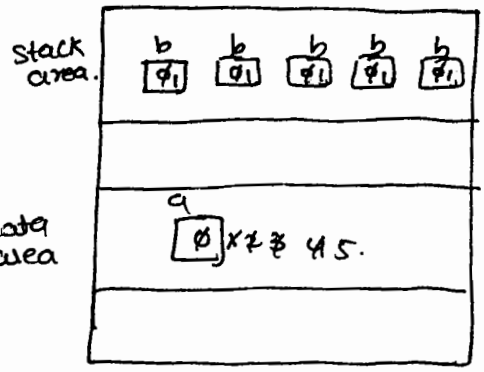
```
Main()
{
static int i;
pf("%d", i);
}
```
o/p:. '0' → data area, value will be 0.

*.
```
Main()
{
Stat();
Stat();
Stat();
}
Stat()
{
static int i;
pf("\t%d",i);
i++;
}
```



Diff. blw scope of auto (non-static) and static :-

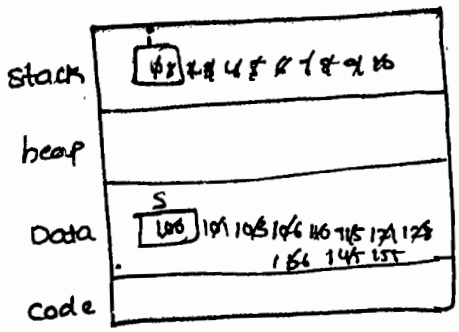
*
```
Main()
{
int i;
for(i=0;i<5;i++)
{
static int a=0;
int b=0;
a++;
b++;
pf("\n a=%d",a);
pf("\n b=%d",b);
}
return 0;
}
```



O/P:
```
1 1
2 1
3 1
4 1
5 1
```

Static variable every time will not initialize if other than zero, it has to initialize at the time of declaration only.

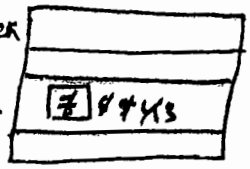
*.
```
Main()
{
int i;
for(i=1;i<=10;i++)
pf("%d %d\n", i, fun(i));
getch();
}
fun(int i)
{
static int s=100;
s+=i;
return s;
}
```



O/P:
```
1 101
2 103
3 106
4 110
5 115
6 121
7 128
8 136
9 145
10 155
```

*.
```
#include<stdio.h>
int r();
int main()
{ for(r();r();r());
{ pf("%d",r()); }
return 0; }
```

```
int r()
{
int static num=7;
return num--;
}
```



External storage class:-

```
auto int i; } integer.
  auto i;
```

*.
```
int i; // global
Main()
{
auto i; // local
pf("%d",i);
}
```

In this local var. & global var's are preserve pr. in the same area, Always prefelence given to the local rather than global var. by this reason, if we want global & local value at a time then "change the name of the variables".

*
```
Main()
{
extern int a;
pf("%d",a);
}
```
o/p:. Error

*.
```
int a; //debinition
Main()
{
extern int a; // declaration.
pf("%d",a);
}
```
o/p:. 0

Difference blw Declaration & Debinition?

Declaration:- only gives the type, status & nature of var's without reseaving any space for the variable.

Debinition:- Actual space is reserved for the variables & some initial value is given.

*.
```
int a; //debinition
Main()
{
extern int a=10; // declaration
pf("%d",a);
}
```
o/p:. Error, Here we can't assign Values to var. bcoz, it doesn't have Memory Space.

*.
```
int a;
Main()
{
extern int a=10; //declaration
a=10;
pf("%d",a);
}
```
o/p:. 10

*.
```
int a=20;
Main()
{
extern int a;
a=10;
pf("%d",a);
fun();
}
fun()
{
pf("%d",a);
}
```
o/p 10 - local var. acting as global variable.

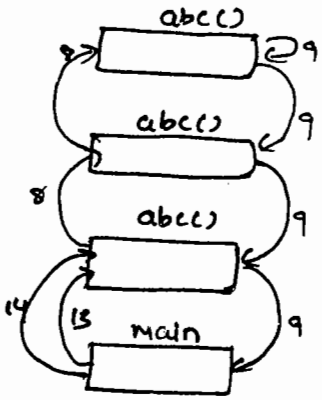
o/p:. 10 10

*.
```
int g;
Void xyz()
{
++g;
}
void main()
{
++g;
xyz();
}
```
o/p:. 2

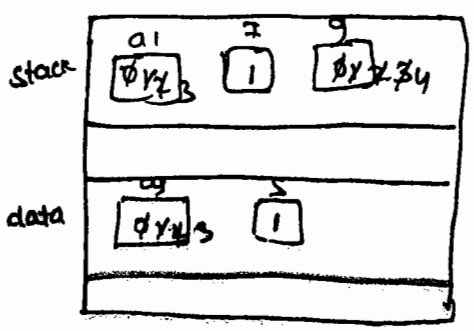
*.
```
void xyz()
{
++g;
}
intg;
void abc()
{
++g;
}
Void Main()
{
abcc();
xyz();
}
```
o/p:. Error, compilation 4m top to bottom, so, it raises an Error

*:
```
Void Anu()
{
Extern int y;
++g;
}
int g; // definition with physical Memory.
Void xyz()
{
++g;
}
Void abc()
{
++g;
}
Void Main()
{
++g; //1
xyz(); //2
abc(); //3
Anu(); //4
pf("%d",g);
}
```
local variable as global.

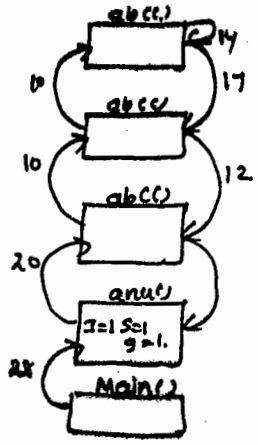
(*).
```
Void abc()
{
int l;
Static int s;
l= ++s;
pf("\n%d %d",l,s);
if (l<=2)
abc();
pf("\n%d %d %d",l,s);
}
Void Main()
{
abcc();
abc();
}
```
o/p:.
1 1
2 2
3 3
2 3
1 3
4 4
4 4.



*
```
1 int g;
2 void abc();
3 {
4 int al;
5 static int as;
6 al = ++as;
7 ++g;
8 pf("%d %d %d %d", al, as, g);
9 if (al<=2)
10 abc();
11
12 pf("\n %d %d %d %d", al, as, g);
```



```
13 Void anu();
14 {
15 int i;
16 static int s;
17  i = ++s;
18  ++g;
19 pf("\n%d %d %d", i,s,g);
20 abc();
21 if (i<=2)
22 Anu();
23 pf("\n%d %d %d %d", i,s,g);
24 }
25 Void main();
26 {
27 Anu();
28 }.
```



O/P:-

```
1 1 1
1 1 2
2 2 3
2 3 4
3 3 4
3 3 4
2 3 4
1 3 4
2 2 5
4 4 6
4 4 6
3 3 6
5 5 7
3 3 7.
```

*.
```
Main()
{
static int i=5
if(--i)
{
Main();
}
Pf("%d", i);
}
```

*.
```
Void anu(int a, int b)
{
pf("\n %d %d", a,b);
}
Void Main()
{
int i=1;
anu(++i, i++);
anu(++i, ++i);
Pf("\n%d", i);
}
```

POINTERS

& -----> address of

* -----> Value at address.
(indirection operator)
(object at that location)

```
Main()
{
int a=10;
Pf ("value of a: %d\n", a);
Pf ("Adrs of a: %u\n", &a);
Pf ("Adrs of a: %x\n", &a);
Pf ("Adrs of a: %p\n", &a);
Pf ("Adrs of a: %d\n", *a); //Error.
Pf ("value of a: %d", *&a);
getch();
}
```

→ *&a = *65524 = 10

→ *a //Invalid *10 pointer not work on value

O/p:-
Value of a: 10
Adrs of a: 65524
Adrs of a: fff4
Adrs of a: FFF4
Value of a: 10

Note: The value at adrs symbol indirectly or directly works with adrs only.

Definition:- "A ptr is an derived data type which can hold adrs."

The adrs can be ordinary var., array var., uses defined data type var, or any other ptr var.

It can also holds adrs of a fun.

Syntax:- datatype *Var-name = < Initialization>;

Eg:- int *s; // s is a var. of type int*
float *k; // k is a var. of type float*.
char *ch; // ch is a var. of type char*.

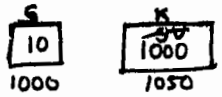
primary datatypes	Derived datatypes
int	int *
float	float*
char	char *
double	double*
etc.	etc.

In this both var. names are same but datatypes are different. one var. holds value & other var. holds Address.

int s — holds value

int *s — holds adrs

1st Method



2nd Method

Variable	Adrs	value
S	1000	10
K	1050	~~9~~ 1000

* Main()
{
int S;
int *K;
S=10;
K=&S;
pf C"value of S: %d\n",S);//10
pf C"adrs of S: %u\n",S);// 1000
pf C"value of S: %u\n",*&S);//10
pf C"value of K: %u\n",K);// 1000
pf C"Adrs of K: %u\n",&K);// 1050
pf C"value of K: %u\n",*&K);//1000
pf C"value of S: %d\n",*K);//10
pf C"%d",*S); // Invalid.
}

→ *&K → *K
*1050 *1000
100 10

* int i ;
int *ptr;

→ ptr is a var. of type int *.
→ *ptr gives integer.
→ Int ptr we can store adrs of integer var.
→ ptr is called as pointer var. of type integer or integer ptr or ptr to integer.
→ This operator(*) gives the value at the adrs pointed by the pointer.

*. Main()
{
int i=10;
int *ptr;
ptr = &i;
pf C"%d %d\n",i,*ptr);//10 10
i= 45;
pf C"%d %d\n",i,*ptr);// 45 45
*ptr =12;
pf C"%d %d\n",i,*ptr);// 12 12
}

As per the above process *ptr is replaced with object (i) at that location

*ptr is an alias to i i.e; alias to the object of the specific location (Adrs).

43 operators returns a value, only * returns the object (location).

*. Main()
{
int a=10;
int *P1,*P2;
P1= &a;
P2= &a;

Var.	Adrs	Value
a	1000	10 45
P_1	2000	1000
P_2	3000	1000

```
pf("%d %d %d %d\n", a, *P1, *P2); // 10 10 10
*P2 = 45;
pf("%d %d %d %d\n", a, *P1, *P2); // 45 45 45
P1 = P2;
b = *P1;
pf("%d %d %d %d", a, b, *P1, *P2); // 45 45 45 45
*P2 = b;
}
```

Var.	adrs	value
a	1000	01
b	2000	X2
c	3000	X2
pval1	4000	1000
Pval2	5000	2000
pval	6000	3000

```
* main()
{
int a=0, b=1, c;
int *pval1, *pval2, *pval;
int i;
pval1 = &a;
pval2 = &b;
pval = &c;
for(i=1; i<6; i++)
{
*pval = *pval1 + *pval2;
*pval1 = *pval2;
*pval2 = *pval;
}
pf("%d %d %d", *pval1, *pval2, *pval);
}
```


Levels of pointers:-

we can face any no. of levels of pointers, There is no restrictions on the levels of pointers.

No. of levels are ↑ing, the prgmng code can reduce

```
* main()
{
int a=10, *P, **P2P, ***P2P2P;
P = &a;
(*P) = (*P)+10;
P2P = &P;
(**P2P) = (**P2P)+5
P2P2P = &P2P
pf("%d %d %d %d", a, *P, **P2P, ***P2P2P);
}
```

Var.	Adrs	value
a	100	16 2625
P	200	100
P2P	300	200
P2P2P	400	300

o/p: 25, 25, 25, 25

*. Void Main()
{
int i;
int * ptr;
int **pptr;
int *** ppptr;
int i=25;
ptr = &i;
pptr = &ptr;
ppptr = & pptr;
}

i → 25
&i → 1000
*ptr → 25
ptr → 1000
&ptr → 2000
* pptr → 1000
pptr → 2000
& pptr → 3000
** pptr → 2000

p pptr → 3000
& ppptr → 4000
** pptr → 1000
*** ppptr → 25.

Var.	Adrs	Value
i	1000	25
ptr	2000	1000
pptr	3000	2000
ppptr	4000	3000

Main()
{
int i=10, *j, **k;
pf("%d\n", i); // 10
pf("%d\n", &i); // 404
pf("%d\n", *(&i)); // 10
j = &i;
pf("%d\n", &j); // 1040
pf("%d\n", j); // 404
pf("%d\n", &j); // 404
k = &j;
pf("%d %d %d %d", i, &k, *k, *&k); // 1040 1042 404 1040
pf("%d %d %d %d %d %d", i, *&i, *j, **&j, **k, ***&k);
10 10 10 10 10 10
getch();
}

properties of pointer:-

1. The size of the pointer irrespective of data type w.r.to mission processor (some time called as operating system).

```
Main()
{
float *s;
pf("%d %d", size of(s), size of (*s)); // 2 4
}
```

depends upon the mission processor.

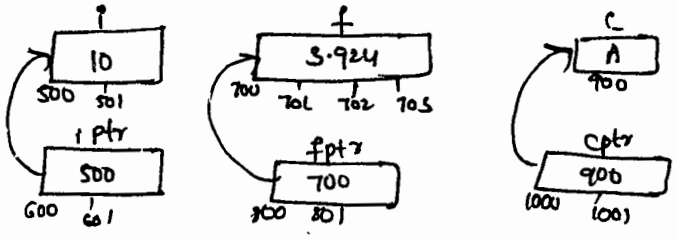
2. An integer pointer will accept integer type float pointer will accept float type i.e; Same type of pointer will accept same type.

```
int i=10;
int *iptr;
iptr=&i;
```

```
float f = 3.924;
float *fptr;
fptr = &f;
```

[illegible]

```
char c=A;
char *cptr;
cptr = &c;
```



In order to hold any type pointers we have generic pointer possible by Void pointers.

* Main()
{
Void *Vptr;
Pf ("%d", sizeof(Vptr)); //2
}

Void Vptr; //Invalid. → Normal variable.

*. Main()
{
Void *Vptr;
int i=10;
float f=3.123;
Pf("%d", sizeof(Vptr));
Vptr=&i;
Pf("In %d", *(int*)Vptr);
Vptr=&f;
Pf("In %f", *(float*)Vptr);
getch();
}

(int)Vptr → Typecasting; value at adrs.

3 Void pointer:-

"The concept of void pointers will be used to read the device adrs of the ports.

Eg: pendrive.

4. A pointer var. at the time of declaration it can be initialize with zero or any adrs or null.

If we initialize with null its basically called as "Null pointer".

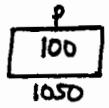
Eg: int a;
int *p=&a;
int *p=0;
int *p = Null;

What is a Null pointer?

A null ptr has a reserved value, often but not necessarily the value zero, indicating that it refers to no object. Null pointers are used routinely participating in C & C++, where compiler-time constant null is used. Null pointer means it is not referring to any object but uninitialized pointer can contain any value (just like any other uninitialized variable), which we call garbage.

Can't convert integer to integer pointer but by making exclusive type casting by user we can assign.

int *P = 100 ⟹ int *p = (int *) 100;



*. Main()
{
int *P1, *P2;
P1 = &P2; // Invalid, It raises & Error can't convert from integer * * (P2P) to integer *.
}

Eg:- Assume both are ptr variable.

P1 = 500 P2 = 600

Assume that P1 & P2 are pointer variables.

*. Addition of pointer variable in constant no. addition is possible.

*. <stdio.h>
Main()
{
int a;
int *P1 = &a;
pf ("%u\n", P1); // 65524
pf ("%u\n", P1+6); //0
}

we are performing addition of memory location the o/p will be the diff. of Memory location depends upon the operator.

P1 = P1 + 1 = 502

Addition of 2 ptrs is not possible. since the compiler will not support.

P1 + P2 = invalid.
(both are pointers)

Substraction of pointer & a constant number is possible.

P1 - 1 = 498

subtraction of two ptrs is possible.

P2 - P1 = 50

The Remaining other operators of arithmetic is not possible.

* P1 * P2
* P1 / P2
* P1 | 2
* P1 % P2

// Invalid.

Comparing relational operators with pointers is possible.

Valid :- P1 > P2
P1 < P2
P1 == P2 / comparing of 2 memory locations.
P1 != P2
P1 ++
P1 --

Advantages of Pointers :-

As w.k.t. return stmt can return only one value at a time but by using ptrs we can also return more than one value by the concept of "Call by reference" (Call by adrs).

pointers allowing to pass value to fun. using call by adrs. This is very usefui when large size of arrays are passing as an arg. to fun's.

Dynamic memory allocation can be possible only by pointers.

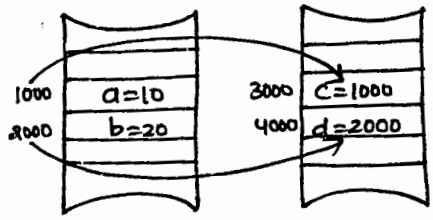
pointers points to physical memory so we can access the data very fastly.

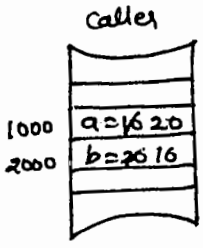
Call by Address:-

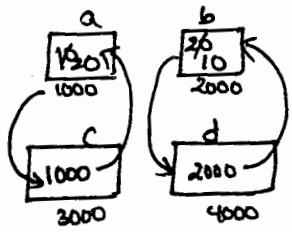
Sending the adrs of the variable from caller to the callee is called as call by adrs.

In this if any changes made in the Subfun. it will affect to the caller fun. var. value.

Eg:-







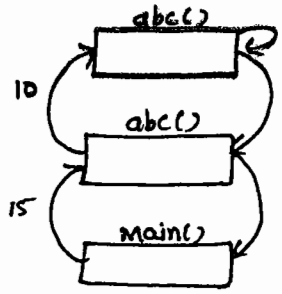
*d = (*c + *d) - (*c = *d)
= 10+20-20
*d = 10


```
*. Void swap (int *, int*)
Main()
{
int a,b;
pf("Enter the values:");
sf("%d%d",&a,&b);
swap(&a,&b); //caller
pf("After swapping in caller fun the value a:%d b:%d \n",a,b);
}
Void swap (int*c ,int *d)
{
*c^= *d^= *c^= *d;
pf("After swaping in sub fun. the value *c:%d *d: %d", *c, *d);
}
```

*. Main()
{
int i=20, *j=&i;
f1(j);
*j += 10;
f2(j);
pf("\n%d%d", i, *j); //45 45
}
f1(int *k)
{
*k += 15; //25
}
f2(int **x)
{
int m = **x, *n=&m;
*n += 10;
}

*. Void Main()
{
int a=2, b=4;
Jump(&a,b);
pf("\n%d %d", a,b); // 4 4
}
Void Jump(int *x, int y)
{
*x = *x * *x;
y = y*y;
}

*.
Void abc(int P1, int *P2)
{
++P1;
++*P2;
pf("\n%d%d", P1, *P2);
if(P1<=2)
abc(P1,P2)
pf("\n%d%d", P1, *P2);
}
Void Main()
{
int a=1, b=1;
abc(a,&b);
pf("\n%d%d", a,b);
}



O/P:
2 2
3 3
3 3
2 3
1 3

Note: Static = call by adrs means both will affect to main mean keeps updated value.

*. Main()
{
int a;
int *P;
P=&a;
pf("Enter the values:");
sf("%d", P);
pf("a: %d", a);
}

* P itself is an address.

*. Main() [better to not Execute this prgm]
{
pf("Enter the 2 values:");
sf("%d%d", (int*)65510, (int*)65512);
pf("%d+%d=%d", *(int*)65510, *(int*)65512,
(int)65510 + *(int*)65512);
getch();
}

O/P: Enter the values 100 200
100+200=300

Os is multitasking System if the locations are not free then prblm is Started. we implementing nameless location. i.e; not a reserved location. if operating system is itself is a prgm anything happen security settings will change the Same prgm is taken place in window 3.5. the system will be corrupted, In windows 95 System will hang.

present os abnormally Teminated.

Function Returning Pointers:-

we have already learnt that a fun. can return an int, a double or any other data type Similarly it can return a pointer.

However, to make a fun. return a pointer it has to be Explicitly mentioned in the calling fun. as well as in the fun. declaration. while retaining pointers, return the pointer to global vars & static & dynamically allocated adrs, do not return any adrs's of local var's bcoz stop to Exit after the fun. call.

return_type * fun-name (arg1, arg2 ---- argn)

Eg:- int *pi(void);

declares "pi" to be a fun. with no parameters that returns a pointer to an int.

* int *pi(void);
Main()
{
int *K;
K = pi();
pf ("%d", *K);//gv
}
int * pi()
{
int a=25;
return &a;
}

* int *pi(void);
Main()
{
int *K;
K = pi();
pf ("%d", *K); //25 (due to static keyword)
}
int *pi()
{
static int a=25;
return &a;
}

Note:-

when we are returning local var's the functionality will closes, whenever it is out of the scope. when control is back the life of the var. dies. pointing to a dead location is called as Dangling Pointer.

Eg: for fun. Returning pointer.

// prgm accept 2 no's & find greater number
<Stdio.h>
void main(void)
{
int a,b,*c;

```
int * check (int*, int*); // check fun. takes 2 integers as arg's & returns a ptr to an integer.
Pf("Enter 2 no's:");
Sf("%d %d", &a, &b);
c = check (&a, &b);
Pf("In Greater no's : %d", *c);
}
int *check (int *p, int *z)
{
if (*p >= *z)
return (p);
Else
return (z);
}.
```


pointer to function:-

Fun's have adrs's just like data items

A ptr to a fun. can be defined as the adrs of the code executed when the fun. is called.

A fun's adrs is the starting adrs of the Machine lang. code of the fun. stored in the memory.

ptrs to fun's are used in writing memory resident prgms writing viruses, or vaccines to remove viruses.

declaration of a ptr to fun.:-

The declaration of a ptr to a fun. requires the fun's return type & fun's arg's list to be specified along with the pointer variable.

Syntax:- return_type (* pointer variable) (fun's arg's list);

Thus, the declaration int (*fp) (int, int);

declares fp to be a var. of type "pointer to a fun. that takes two integer arg's & return an integer as its value".

Thus declarations:-

Normal

*. int i(void);

declares 'i' to be a fun. with no parameters that return an int.

pointer to function

*. int (*ip) (void)

declares 'ip' to be a ptr to a fun. that returns an integer value & takes no arg's.

Eg:

```
int fun1 (int i)
{
return i;
}
float fun2 (float f)
{
return f;
}
```

```
Main()
{
int (*p) (int);
float (*z) (float);
int i=10;
float f=1.5;
p = fun1; // adrs of fun1 assign to p.
z = fun2; // adrs of fun2 assign to z.
Pf("%d %f", p(i), z(f));
getch(); //10 1.500000
}
```

// declaration of ptr to fun.

after declaring the fun. prototypes & 2 ptrs p & q to the fun's p is assigned the adrs of fun. func1 & q is assigned the adrs of fun. func2.

Invoking a fun. by using pointers.

In the ptr declaration to fun's, the ptr. var. along with the operator(*) plays the role of the fun. name. Hence, while invoking fun. by using ptrs, the fun. name is replaced by the pointer variable.

ARRAYS

As wkt, An ordinary var. can hold only one value at a time, it is not possible to hold no. of values. for suppose if i want to store 10 subject marks of the students, we need to declare 10 variables. declaring the variables is possible but it is not a recommended process.

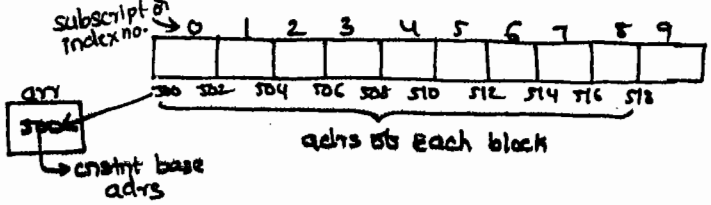
Since, the logical stmts no. of times will be takes place unnecessarily, the steps of the prgmng ↑ses. To overcome this prblms we use arrays.

Definition of Array:-

An Array is an derived data type which is single subscripted var. which can hold 'n' no. of values.

Syntax:- data type var-name[size]; → dimension.

Eg:- int arr[10];



whenever an array var. is declaring the size of the var., must be defined once we define the size, it is never possible either to ↑se & ↓se.

An Array is an homogeneous type. i.e; it can hold only same type of Elements.

The array always contains the 1st block adrs. i.e; called as base adrs.

The array Elements will be identify by the Subscript no's Externally, but internally every block will identify bytes own adrs.

The array adrs's of each block in a Sequential memory block.

They are 3 types of arrays:

1. Single dimension. 2. Two-Dimension. 3. Multi Dimension.

Declaration of Single Dimension Arrays:-

→ int a[5];
→ float arr[7];
→ char name[25];
→ int a[5] = {11,-9,22,44,7};
→ float s[5] = {49,92,-11.09,97};
→ char name[25] = {"Anusha"}; & char name[25] = {'A','N','U','S','H','A'};
→ int s[]; // invalid.
int s[n];
→ int arr[] = {10,11,12};
→ int arr[];
arr[] = {7,9,12}; // invalid
→ int a=10; // invalid, 'a' is a var.
int arr[a]; array size is fixed.

→ const int a=10;
int arr[a];

→ int arr[20000];

→ int a[40000]; // invalid, as per 16 bit not possible.
long a[40000]; 32 bit is possible, bcoz 16 bitsize = 65535
32 bit size = millions of space.

→ int a[5]={10};

0	1	2	3	4
10	0	0	0	0

, at the time of declaration of an array atleast 1 element is initialized, remaining are zero's.

→ #define MAX 10
int arr[MAX];

→ int a[0]; // invalid, bcoz index starts with zero, array size is not zero.

→ int a[5];

0	1	2	3	4
gv	gv	gv	gv	gv

→ int a[5<2]; //invalid bcoz size '0'

→ int a[5>2]; // valid, bcoz size 1

* WAP accept an array of Elements, display the elements:

```
<stdio.h>   <conio.h>
Main()
{
int arr[100];
int size, i;
clrscr();
pf("Enter the size of an array");
sf("%d", &size);
// Accepting
for(i=0; i<size; i++)
{
pf("Enter the Element at arr[%d]:", i);
sf("%d", &arr[i]);
}
// display
for(i=0; i<size; i++)
pf("%d", arr[i]);
getch();
return 0;
}
```

i → indicating index no.
arr[i] → array element
arr → constnt base adrs.

O/P:-
Enter the size of an array: 5
Enter the Element at arr[0]: 45
[1]: 79
[2]: 61
[3]: 83
[4]: 99
45 79 61 83 99.

* WAP accept some Elements in array find the Sum of all the elements & display the Sum;

```
<stdio.h>   <conio.h>
Main()
{
int arr[10];
int n, i, sum=0;
clrscr();
```

```
pf("Enter the size of an array:");
sf("%d",&n);
//Accepting
for(i=0;i<n;i++)
{
pf("Enter the elements of arr[%d]:",i);
sf("%d",&arr[i]);
}
for(i=0;i<n;i++)
sum=sum+arr[i];
pf("sum of the array elements:",sum);
getch();
}.
```


* WAP accept some elements of an array display the elements frm n to 0 !

```
<stdio.h> <conio.h>
Main()
{
int arr[10],n,i;
clrscr();
pf("Enter the size of array:"); //4
sf("%d",&n);
for(i=0;i<n;i++)
{
pf("Enter the elements at arr[%d]:",i); //10 20 30 40
sf("%d",&arr[i]);
}
for(i=n-1;i>=0;i--)
pf("%d",arr[i]); // 40 30 20 10
getch();
}
```

0	1	2	3
10	20	30	40

* WAP accept the elements of an array & also accept one element find out that element how many times it occurs, display the count.!

```
< stdio.h> <conio.h>
Main()
{
int a[20],n,i,count=0,ele;
clrscr();
pf("Enter the size of n:");
sf("%d",&n); //4
for(i=0;i<n;i++)
{
pf("Enter the elements of a[%d]:",i);
sf("%d",&a[i]);
}
pf("Enter the Element to search :",ele);
sf("%d",&ele); //10
for(i=0;i<n;i++)
{
if(a[i]==ele)
{
count++;
}
}
if(count==0)
pf("number not exist",ele);
Else
pf("Element %d is found %d times",ele,count);
getch();
}.
```

0	1	2	3
10	30	10	40

* WAP find out the 2nd max. Element from an array?

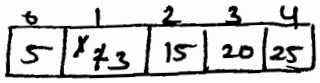
```
<stdio.h> <conio.h>
Main()
{
int a[20], i;
clrscr();
pfc" Enter the elements of an array [0-4]", i);//3 6 9 7 1
for(i=0; i<5; i++)
sf ("%d", &a[i])
fm= a[0]; //3
id=0;
for(i=1; i<5; i++)
{
if (fm < a[i])
{
fm=a[i];
 id = i;//4
}
}
if(id !=0)
sm= a[0];
Else
sm= a[1];
for(i=0; i<n; i++)
{
if( fm>a[i] && sm<a[i])
sm=a[i];
}
pf C"second max is %d", sm);
getch();
}.
```


*. WAP accept an elements delete that Element 4m an array & display the Elements!

```
Main()
{
int a[50], n, i, de;
Clrscr();
pfc"Enter the size of array:");
sfc"%d",&n);
for(i=0; i<n; i++)
{
pfC"Enter the Element of array[%d]:", i);
sfc"%d", &a[i]);
}
pfC"Enter the Element to delete:");
sfC"%d", &ele);
```

```
pf C"\n Elements Before deletion\n");
for (i=0; i<n ; i++)
pf("%3d", a[i])
for(i=0; i<n; i++)
{
if (a[i]==ele)
{
ck=1;
for(j=i; j<n; j++)
{
a[j] = a[j+1]
}
n--;
i--; //consecutive duplicates
}
}
if (ck==0)
pf C"Element not found");
Else
{
pf C" After deletation"\n);
for(i=0; i<n; i++)
Pf C"%3d", a[i]);
getch();
}
```

* Main()
{
int a[5] = {5,1,15,20,25};
int i,j,k=1,m;
i = ++a[1];
j = a[1]++;
m = a[i++];
pf("\n%d %d %d %d", i,j,m,i[a]); //3,2,15,20
}



j=3 j=2(3)

* Main()
{
Int a[] = {10,20,30,40,50};
int i=3, x;
x = 1 * a[--i] + 2*a[--i] + 3*a[--i];
pf("%d", x);
}

passing Array as a fun. parameters :-

In a fun. we can pass a single dimensional, a multidimension as a fun. parameters whenever a single dimensional array is passing as a fun. parameter it require min. 2 args. The first arg can be any type it indicates an array, the 2nd arg. indicates the size must be integer type.

Syntax:- returntype fun-name (arg1[], arg2[], ---);

* WAP accept an array of elements & display the elements by passing as a fun. parameters?

```
Void read (float[], int);
Void write (float[], int);
Main()
{
float a[10];
int size;
pf("enter the size of the array:");
sf("%d", &size);
pf("Enter the elements :\n");
read(a, size); // calle //read (&a[0], size);
pf("In The elements are ---:");
write(a, size); //calle
getch();
}
Void read (float a[], int n) // callee
{
int i;
for(i=0; i<n; i++)
sf("%f", &a[i]);
}
Void write (float a[], int n)
{
int i;
for(i=0; i<n; i++)
pf("%.2f\n", a[i]);
}
```

size is optional leftmost dimensional.

Whenever an array is passing as a fun. parameter by default it pass the adrs of the array [base adrs].

As the Array is passing adrs the concept is call by adrs.

*? In this prgm as an subscript is recieving the adrs, so it is * and [ ] both are same or not?

*? As an array use the adrs then y can't we say array is a pointer?

→ *. Yes, but the diff. is,

* *a=25;
  a++; //valid
  25++; // invalid, (integer cnst)

* int *P;
  p++; //valid

* int a[5];
  a++; // invalid, bcoz a contains cnstn base adrs.

Here, a → cnstnt pointer p → is a integer pointer.

Note:- Array name always gives the base adrs of the array (arr & &arr[0]) both are same.

Either we writing the prgm with subscript nds also, internally they will be replace by the ptrs.

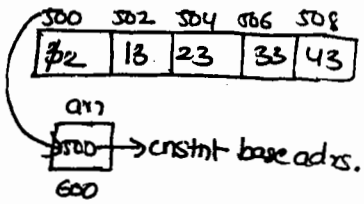
Void read (float *a, int n)

The def^n of the read fun. internally working with pointers rather than subscript the def. of read can also be written as,

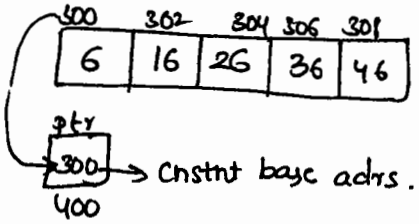
* Void read (float *a, int n)
  a[]
  {
  int i;
  for (i=0; i<n; i++)
  pf ("%d", *(a+i));
  }

So, finally * = [ ]
* * = [ ][ ]
* * * = [ ][ ][ ] --- Etc.

*. Void Main()
{
int arr[5] = {3,13,23,33,43};
++arr;
--*arr; //2
--*arr; //1
pf("%d", *arr); //1
}

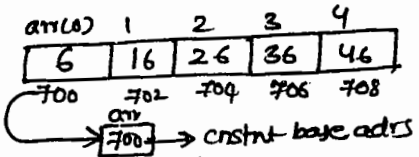


* Void Main()
{
int arr[5] = {6,16,26,36,46};
int *ptr;
ptr = &arr[0];
++ptr; // pre incrementation of pointer.
++ptr;
++*ptr; //27
pf("%d", *ptr);
}



++ptr → pre increment pointer
ptr++ → post
--ptr → pre decrement pointer
ptr-- → post
++*ptr → pre increment of object
--*ptr → pre decrement of object
(*ptr)++ → post increment
(*ptr)-- → post decrement
*++ptr → pre increment of pointer & accessing the data.
*ptr++ → accessing the data & post incrementation of pointer.
*--ptr → pre decrementation of pointer & accessing the data.
*ptr-- → accessing the data & post decrement the pointer.

*. Void Main()
{
int arr[5]={6,16,26,36,46};
pf("%d %d %d %d",arr[2],*(arr+2),*(2+arr),2[arr]; //26 26 26 26
}



External	Internal
arr[2]	*(arr+2)
	*(700+2)
	*704
	=26
2[arr]	*(2+arr)
	*(2+700)
	*704
	=26

When we construct an array sequential space is received & a cnstnt ptr is generated with the name of array as a base adrs. we comfortably locating all the elements with "Index".

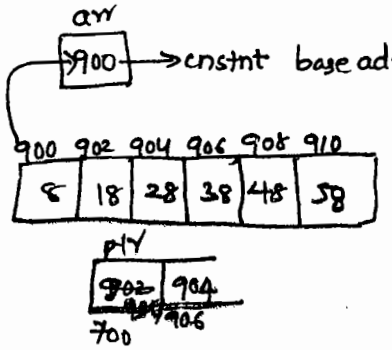
* Why the Array index Starting with Zero:-

arr[0] → *(arr+0)
*(700+0) = *706

arr[1] → *(arr+1)
*(700+1) = *708

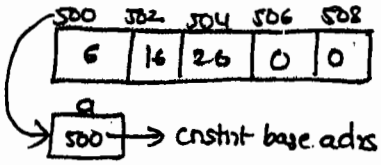
This is result array start with zero, physically 1st location.

*. Void main()
{
int arr[] = {8,18,28,38,48,58};
int *ptr;
ptr = arr+1;
++ptr;
--*ptr;
++ptr;
++*ptr;
pf("%d %d %d", ptr[-1], ptr[0], ptr[1]); }

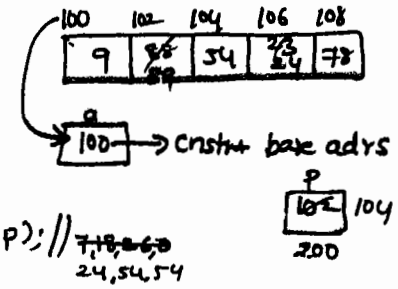


28 39 48

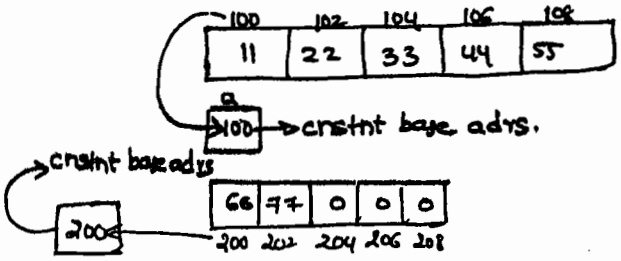
*. Main()
{
int a[5] = {6,16,26};
++*a;
++*(a+1);
++a[1];
pf("%d %d %d %d %d", a[0], *(a+1), *(2+a), a[3]); //7 18 26 0.
}



*. Main()
{
int a[5] = {9,88,54,23,78};
int *p;
p=a+1;
pf("%d %d %d %d", ++*p, *p++, *++p); // 24,54,54
}



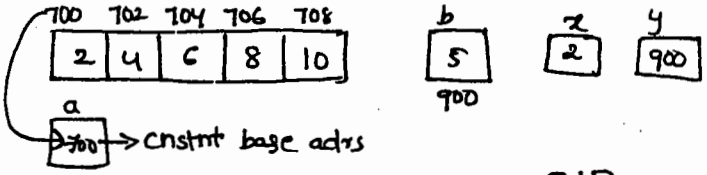
* Main()
{
int a[5] = {11,22,33,44,55};
int b[5] = {66,77};
b=a;
pf("%d", b[1]); //error.
}



b[0], we didn't assign one array values with another array.

*. Main()
{
static int a[5] = {2,4,6,8,10};
int i, b=5;
for(i=0; i<5; i++)
{
f(a[i], &b); //caller
pf("%d %d", a[i], b);
}
}
f(x,y) //callee //K&R Notation.
int x, *y;
{
x = *y += 2;
}

O/p: 2 7 4 9 6 11 8 13 10 15



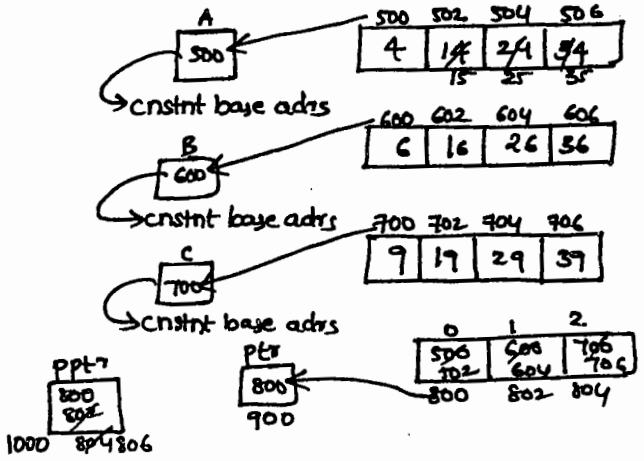
*. Main()
{
int a[2];
int i;
for(i=0; i<10; i++)
a[i] = i*i;
pf("%d %d", a[3], a[4]);
}.

In this size of the array is not satisfied only loop will satisfy. But still C & C++ doesn't raises any error. since they are not supporting bounds checking.

Array of pointers :-

int *a[5]; // This array can hold 5 adrs's in each block.

```
Void Main()
{
int A[]={4,14,24,34};
int B[]={6,16,26,36};
int C[]={9,19,29,39};
int *ptr[3];
 int **pptr;
 int i;
ptr[0]=A; // ptr[0]=&A[0];
 ptr[1]=B;
ptr[2]=C;
pptr=ptr;
for(i=1; i<=3; i++)
{
 *pptr+=i;
 **pptr+=i;
 ++pptr; //preincrement ptr.
 }
 -- pptr; // predecrement ptr.
 pf("in=%d", **pptr);
 for(i=0; i<3; i++)
  pf("%d", *ptr[i]);
 for(i=0; i<4; i++)
 pf("in=%d %d %d %d", A[i], B[i], C[i]);
 }
 getch();
 }
```



* WAP accept some elements of an array implement the operations of intersection, union, sorting.

```
<stdio.h> <conio.h>
Main()
{
 Void accept (int[], int);
 Void display (int[], int);
 Void sort (int[], int);
 int intersection (int[], int[], int[], int, int);
 int union_arr( int[], int[], int[], int, int);
 int a[20], b[20], in[20], un[40], m, n, ie, ue;
 clrscr();
 pf("\n Enter no. of elements for a:"); //5
 sf("%d", &m);
```

```
pf ("\n Enter no. of Elements in b:"); //4
sf("%d", &n);
pf ("\n Enter Elements into a:");
accept (a,m);
pf("\n Enter Elements into b:");
accept (b,n);
clrscr();
sort (a,m);
sort (b,n);
pf ("\n Elements of a:");
display (a,m);
pf("\n Elements of b:");
display (b,n);
ie= intersection (a,b,in,m,n);
pf ("\n Intersection of 2 arrays:").
if (ie==0)
pf ("No Elements");
else
display (in, ie);
ue = union_arr (a,b,un,m,n).
pf ("\n union of two arrays:
display(un,ue);
getch();
}
void accept (int *a, int n)// Restriction of
{                          accepting duplicating
                                    elements.
int i,j;
for (i=0; i<n ;i++)
{
scanf ("%d", a+i); //56 34 24
for (j=0; j<i ; j++)
{
if (*(a+i) == *(j+a))
{
pf ("Elements already Exists\n");
pf ("Enter new Element \n");
i--;
break;
}
}
}
}
```

```
Void display (int a[], int n)
{
int i;
for(i=0; i<n; i++)
pf ("\t%d ", *(a+i));
}
}
Void sort (int *a, int n)
{
int i,j,t;
for(i=0; i<n-1; i++)
{
for (j= i+1; j<n ; j++)
{
if (*(a+i) > j[a])
{
t= *(a+i);
*(i+a) = *(j+a);
*(a+j) = t;
}
}
}
}
int intersection (int a[], int *b, int *c,
                                int m , int n)
{
int i,j, k=0;
for(i=0; i<m; i++)
{
for (j=0; j<n; j++)
{
if (*(a+i)) == * (b+j))
{
*(c+k) = *(a+i);
k++;
break;
}
}
}
return k;
}
int union_arr(int *a ,int b[], int *c, int m,
                                int n)
{
int i=j=k=0;
while (i<m && j<n)
{
```

```
if (a[i]<b[j])
{
c[k]=a[i];
i++;
}
else if (a[i]>b[j])
{
c[k]=b[j];
j++;
}
else
{
c[k]=a[i];
i++;
j++;
}
k++;
}
while (i<m)
{
c[k]=a[i];
i++;
k++;
}
while (j<n)
{
c[k]=b[j];
j++;
k++;
}
return k;
}
```


Double Dimensions:-

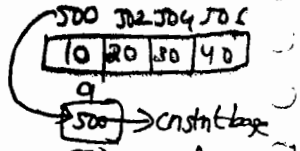
double dimension arrays is nothing but combination of raws & columns many app's can also be designed by using double dimensions. Mathmatics, matrix app's are designed by using DD's. Business oriented app's are designed by double dimension arrays & the gaming prgmngs are also designed by DD arrays.

How the Elements are arranged in Multidimension array?

```
* Main()
{ int a=5;
pf("%d",a,&a); //5,100
pf("%d",*a); //Error
}
```

a: 5 (1000)

```
* Main()
{ int a[5]={10,20,30,40};
pf("%d %d",a,&a); //500,500
pf("%d",*a); //10
}
```



Syntax for DDs:-

datatype var-name [row][col];

Eg: int a[3][2];
↳ (ebt most can be optional when you initialize

* int a[3][3] = { {4,5,9}, {10,11,12}, {-1,3,4} };
* int a[3][2] = { 10,11,12,-9,8,7 };
* int a[ ][3] = { 11,12,56,78 }; ⇒ { {11,12,66}, {78,0,0} }.
↳ optional ⇒ a[2][3].

depends on the column value, rows will be calculated

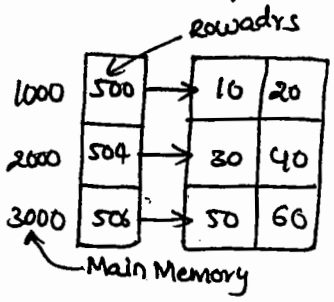
* int a[3][ ] = {11,12,13}; // invalid, bcos of without columns.
* int a[3][2];

size of array = datatype * col * row

= 2 * 2 * 3 = 12 bytes.

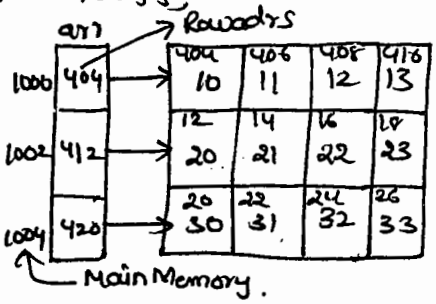
Either any type of dimensions, the adrs always will be sequential.

* Main()
{
int a[3][2] = { 10,20,30,40,50,60 };
pf("%d", a); // 500
pf("%d %d", &a, *a); // 500 500 bcos it is not a single dimension array.
pf("%d", **a); // 10.
}



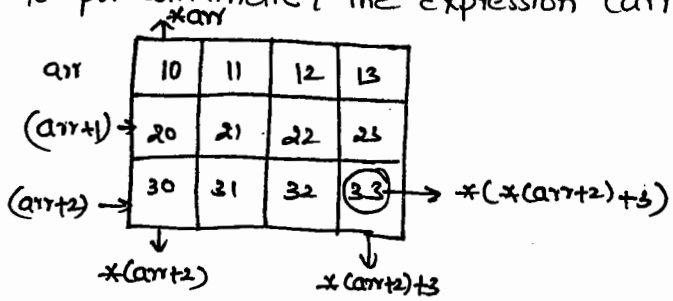
so, we require 2 pointers to show the value of 2-D, ly for 3-D, we require 3 pointers.

*. Main()
{
static int arr[ ][4] = { {10,11,12,13}, {20,21,22,23}, {30,31,32,33} };
pf("%d\n", arr); // 404
pf("%d\n", arr+0); // 404
pf("%d\n", *arr); // 404
pf("%d\n", *arr); // 412
pf("%d\n", arr+1); // 420
pf("%d\n", arr+2); // 404
pf("%d\n", *(arr+0)); // 412
pf("%d\n", *(arr+1)); // 420
pf("%d\n", *(arr+2)); // 420
pf("%d\n", *(arr+2)+3); // 426
pf("%d\n", *(arr+0)+1); // 406
pf("%d\n", *arr+1); // 406
pf("%d\n", *arr+3); // 414
pf("%d\n", *(*(arr+2)+3)); // 33
}

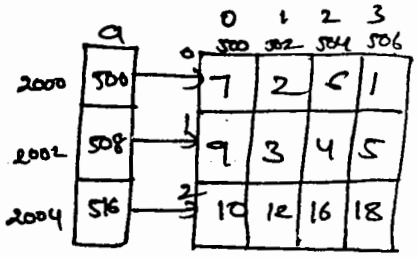


Each row is considered as a 1-D array, so, a 2-D array can be considered as a collection of 1D array that are placed one above another. In other words we can say that a 2D array is an array of arrays. so here arr is an array of 3 Elements where each Element is a 1D array of 4 integers.

W.K.T, the name of an array is a constant ptr that points to the 0^{th} Element of the array. In case of 2D arrays, 0^{th} Element is a 1D array so, the name of a 2D array rep's a ptr to a 1D array for eg. in the above case, arr is a ptr to 0^{th} 1D array contains adrs 404. Since arr is a pointer to an Array of 4 Integers so, according to ptr arithmetic, the Expression (arr+1) will rep's adrs 412----

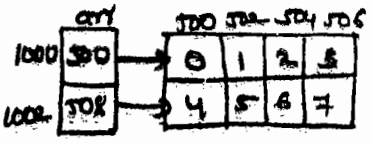


*. Void Main()
{
Static int a[][4] = {7,2,6,1,9,3,4,5,10,12,16,18};
pf("%d %d", a); //500
pf("%d %d", *a); //500
pf("%d %d %d %d", a+0, a+1, a+2); //500, 508, 516
pf("%d %d %d %d", *(a+0), *(a+1), *(a+2)); //500 508 516
pf("%d %d %d %d", a[0], a[1], a[2]); //500 508 516
pf("%d %d %d %d", a[0]+1, a[1]+2, a[2]+3); //502, 512, 522
pf("%d %d %d %d", *(a[0]+1), *(a[1]+2), *(a[2]+3)); //2 4 18
pf("%d %d %d %d", a[0][1], a[1][2], a[2][3]); //2 4 18
}



a[0] = *(a+0)
a[0][1] = *(*(a+0)+1)
a=500

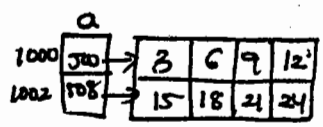
* Void Main()
{
long arr[2][4] = {0l, 1l, 2l, 3l, 4l, 5l, 6l, 7l};
pf("%ld", arr[1][2]); //6
pf("%ld %ld", *(arr[1]+3), 3[arr[1]]); // 7 7
pf("%ld %ld %ld", *(*(arr+1)+2), *(1[arr]+2), 3[1[arr]]); //6 6 7
}



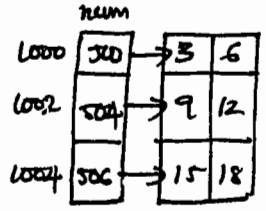
arr=1000

3[arr[1]) = *(3 + *(arr+1))
3[1[arr]] = *(3 + *(1+arr))

* int Main()
{
int a[2][4] = {3,6,9,12,15,18,21,24};
pf("%d %d %d", *(a[1]+2), *(*(a+1)+2), 2[1[a]]); //21 21 21
return 0;
}

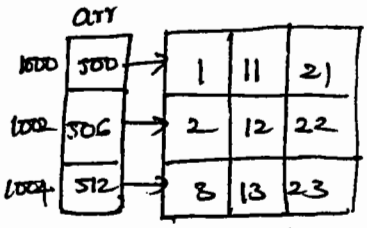


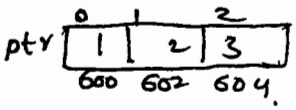
Void Main()
{
short num[3][2] = {3,6,9,12,15,18};
pf("%d %d", *(num+1)[1], **(num+2)); // 15, 12
}



* Void Main()
{
long double a;
signed char b;
int arr[sizeof(a+b)]; //arr[8+1] = arr[9].
pf("%d", sizeof(arr)); //4
}

* Void Main()
{
Static int arr[3][3] = {{1,11,21}, {2,12,22}, {3,13,23}};
int *ptr[3];
int **pptr;
clrscr();
ptr[0] = &arr[0][0];
ptr[1] = &arr[1][0];
ptr[2] = &arr[2][0];
pptr = ptr;
++*pptr;
++ptr[0];
++**pptr;
++*ptr[0];
++pptr;
++*pptr;
--ptr[1];
--**pptr;
++*ptr[1];
++pptr;
++*pptr;
--*ptr[2];
++**pptr;
pf("\n%d %d %d", arr[0][2], arr[1][2], arr[2][0]); // 23 22 3
}





```
pf("\n%d %d %d", *(*(arr+1)+2), *(*(arr+2)+0), *(*(arr+0)+2)); //23,3,23
pf("\n%d %d %d", *ptr[0], *ptr[1], *ptr[2]); //23 2 13
pf("\n%d %d %d", **(ptr+0), **(ptr+1), **(ptr+2)); // 23,2,13
}
```

* WAP accept 2D array display the Elements in Matrix format.

```
Main()
{
int a[5][5];
int i,j;
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
pf("Enter the element at a[%d][%d]:", i,j);
sf("%d", (*(a+i)+j));
}
}
for(i=0;i<3;i++, pf("\n"));
{
for(j=0;j<3;j++)
pf("%3d", *(a[i]+j));
}
getch();
}
```


* A[2][3] → *(a[2]+3) → *(*(a+2)+3)

Array	Subscript	pointer
1D	a[i]	*(a+i)
2D	a[i][j]	*(*(a+i)+j)
3D	a[i][j][k]	*(*(*(a+i)+j)+k)
4D	a[i][j][k][l]	*(*(*(*(a+i)+j)+k)+l)

* WAP implement Matrix Addition & Multiplication

```
Void accept(int[][30], int, int);
Void display(int[][30], int, int);
Void prod(int[][30], int[][30], int[][30], int, int, int);
int mat1[30][30], mat2[30][30], mat3[30][30];
int i,j, k, sum, r1, r2, c1, c2;
Main()
{
clrscr();
pf("Enter the size of Mat1\n");
```

$$\begin{bmatrix}1&2\\3&4\\5&6\end{bmatrix}\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$$

r1=3
r2=2
c1=2
c2=3

```
    sf("%d%d",&r1,&c1);
    pf("Enter the size of mat2\n");
if (c1==r2)
{
pf("\n Enter the first matrix\n");
accept (mat1, r1, c1);
pf("\n Enter the second matrix\n");
accept (mat2, r2, c2);
pf("\n The Elements in matrix1\n");
display (mat1, r1, c1);
pf("\n The Elements in matrix 2\n");
display (mat2, r2, c2);
pf("\n product of 2 matrices ....\n");
prod (mat1, mat2, mat3, r1, c2, r2);
display (mat3, r1, c2);
}
else
pf("\n Multiplication not possible ...\n");
getch();
}
void accept (int mat[][50], int r, int c)
{
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
sf("%d", *(mat+i)+j);
}
}
void display (int mat[][50], int r, int c)
{
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
pf("%3d", *(mat[i]+j));
pf("\n");
}
}
void prod (int m1[][50], int m2[][50], int m3[][50],
                                int r1, int c2, int r2)
{
for(i=0; i<r1; i++)
{
for(j=0; j<c2; j++)
{
sum=0;
for(k=0; k<r2; k++)
{
sum=sum+m1[i][k]*m2[k][j];
}
m3[i][j]=sum;
}
}
}
```

* WAP accept a 3x3 matrix find the Sum of principal diagonal Elements.
* WAP accept a matrix display the Transpose of matrix
* WAP accept a 5x3 dimension matrix display the mirror image & water image of the Matrix.

*. int arr[7][8];

1 → a[0][0]
28 → a[3][3]
16 → a[1][7]
45 → a[5][4]
36 → a[4][3]
52 → a[6][3].

8|27|3-R
24/3-C

8|15|1-R.
8/7-C

* Main()
{
int arr[5][8];
pf("%d%d%d", sizeof(arr), sizeof(arr[2]), sizeof(arr[1][0])); //80,16,2
}

→ indicate a element
sizeof(arr[1][0]). → Row contain 8 column so, 8x2=16

* Main()
{
int a[4][8] = {1,5,55,555};
pf("In %u", &a[3][4] - &a[1][2]);
}

a[3][4] = 538
a[1][2] = 502
36 → devide by 2 bytes.
36/2 = 18

&2 - &2.

Row=4, col=8,

diff. in Memory location. = 8*3+4 = 28
8*1+2 = -10
18 bytes.

column.
a[1][2]

Multidimension arrays:-

Basically more than one dimensions will be take place as a multidimension arrays, for Easyness of the priming, we use multidimensional arrays (nothing but multipointers).

*. int a[5][3][4]; → left most can be optional, when you initialize.
Blocks ↓ Rows ↓ Columns

size = 2*5*3*4 = 120 bytes.

Elements = 5*3*4 = 60 Elements.

* int a[3][2][3] = { { {1,2,3}, {4,5,6} }, { {7,8,9}, {10,11,12} }, { {13,14,15}, {16,17,18} } };
  (0th block, 1st block, 2nd block.)

* int arr[5][4][5][2]
  (5 ↓ set, 4 ↓ Block, 5 ↓ row, 2 ↓ column.)

* int arr[3][4][5];

  1 → arr[0][0][0];
  37 → arr[1][3][1];
  14 → arr[0][2][3];
  59 → arr[2][5][3];
  29 → arr[1][1][3]
  11 → arr[0][2][0]

$5\left|\begin{matrix}36\\32\end{matrix}\right|7$ – R ⇒ $4\left|\begin{matrix}7\\4\end{matrix}\right|1$ – B ⇒ 3 – R
1 – C

* int arr[5][3][4][5];

  1 → a[0][0][0][0]
  147 → arr[2][1][1][1]
  234 → arr[3][2][2][3]
  96 → arr[1][1][3][0].

$5\left|\begin{matrix}146\\145\end{matrix}\right|29$ ⇒ $4\left|\begin{matrix}29\\24\end{matrix}\right|7$ ⇒ $3\left|\begin{matrix}8\\6\end{matrix}\right|2$ – B
1 – C, 5 – R, 2 – B

$5\left|\begin{matrix}95\\95\end{matrix}\right|19$ ⇒ $4\left|\begin{matrix}19\\16\end{matrix}\right|4$ ⇒ $3\left|\begin{matrix}4\\3\end{matrix}\right|1$ – set.
0 – C, 3 – R, 1 – B

$5\left|\begin{matrix}233\\230\end{matrix}\right|46$ ⇒ $4\left|\begin{matrix}46\\44\end{matrix}\right|11$ ⇒ $3\left|\begin{matrix}11\\9\end{matrix}\right|3$ – S
3 – C, 2 – R, 2 – B

* <stdio.h> <conio.h>

```
Void main()
{
int marks[3][2][5][2], i,j,k,l;
int subsum, semsum, yearsum, grandtotal=0;
clrscr();
for(i=0; i<3; i++)
{
yearsum=0;
for(j=0; j<2; j++)
{
semsum=0;
for(k=0; k<5; k++)
{
subsum=0;
for(l=0; l<2; l++)
{
pf("Enter %d yr %d sem %d sub %d marks :", i+1, j+1, k+1, l+1);
sf("%d", &marks[i][j][k][l]);
subsum = subsum + marks[i][j][k][l];
}
semsum = semsum + subsum;
yearsum = yearsum + semsum;
grandtotal = grandtotal + yearsum;
}
pf("Grand total = %d", grandtotal);
getch();
}.
```

Strings

collection of chars makes to form strings.

unformatted I/O fun's:-

getch()	puts()
getchar()	putchar()
getche()	

ASCII codes:-

'\n' → 10	' ' → 32
'\r' → 13	'\0' → 0
'\t' → 9	EOF → 26
'\b' → 8	End of file.

Getchar():- The getchar() fun. is similar to sf() fun. both this fun's will accept the i/p untill the user press Enter key. Both can capture only datatype. The getchar() fun. will read only 1 char. from the buffer.

getchar() & sf() both are from the stdio.h headfile.

*. <stdio.h>

```
Main()
{
char s;
pf("Enter the char:");
//sf("%c", &s);
s = getchar();
pf("%c", s);
getch();
}
```


It reading the chars from the buffer.

The getchar() & sf() both read the Enter key as a "\n" which have the ascii code is '10'.

* getche() & getch():-

Both this fun's are similar both will accept one byte of char. as an i/p.

These both are 4m the conio.h header file

```
<conio.h>
Main()
{
 char s;
 pf("Enter the char:");
 //sf("%c", &s);
 s = getche();
 pf("%c %d", s, s);
 getch();
}
```

* Here, no need of the Enter key after Entering the i/p values.
* The getche() fun, after accepting any i/p it doesn't wait for any key.
* The getch() fun. unlike nature doesn't show what type of char we typed on the screen. Appn:- In the pswd logic.
* By using the "getch()" fun. we can write the pswd logics.
* The getch() & getche() will read "Enter" key as a "\r"
* Only in C-lang, the 2 fun's getch() & getche() both can read data keys & Non-data keys.
* The Req. of the Non data keys in order to implement Gaming prgming we require of Non-data keys.
* Every Non data key start with a Special char. i.e. "Zero(0)". followed by Another number.

up arrow	→ 0&72	Pg up	→ 0&73
down arrow	→ 0&80	pg down	→ 0&81
Left arrow	→ 0&75	Home	→ 0&71
Right arrow	→ 0&77		

```
*. <stdio.h>
   Main()
   {
   char s1,s2;
   pf("Enter the char:");
   s1=getch();
   s2=getch();
   pf("\nd.d",s1,s2);
   getch();
   }
```


putch() & putchar():- will display 1 byte of char as an o/p.

Wherex() & wherey():- from Conio.h Headafile.

Wherex() use cnt horizontal cursor position, it returns an int. value in the range of 1-80.

wherey() gives cnt Vertical cursor position, it return an int. value in the range of 1-25, 1-43 & 1-50.

gotoxy (col,row):-

The gotoxy make you to move the cursor to any position of the screen as per the requirement column & Row values.

* WAP to move the cursor to any position of the Screen.

```
Void Main()
{
char ch1, ch2;
int x,y;
clrscr();
ch1 = getch();
while (ch1==0)
{
ch2 = getch();
x = wherex();
y = wherey();
if (ch2==72)
gotoxy (x, y-1);
if (ch2==77)
gotoxy (x+1, y);
if (ch2==80);
gotoxy (x, y+1);
if (ch2==75)
gotoxy (x-1, y);
ch1=getch();
}
}
```


update the above code in the prime number logic.

gets():- The gets() fun. is a string i/p which accepts only a strings.

The gets() fun. will accept the i/p until the user press Enter Key.

The diff. b/w the sf() & gets() fun. The sf() will not accept the char's when the space is occurs.

The gets() can read the char's either the space is occurs also.

puts():- The puts() fun. will display a string as an o/p. puts() can only display strings.

puts always shows the Next line.

```
Main()
{
char str[100];
pf ("Enter the String");
// sf ("%s", str);
gets(str);
puts(str);
pf (str);
getch();
}
```

o/p:-

```
Enter the String: This is data.
This is data.
This is data.
```

* Example On Screen Saver program (Scrolling).

```
<stdio.h>  <conio.h> <string.h> < dos.h>
Void Main()
{
char st[40], ch;
int i,s,k1c;
clrscr();
textmode();
pf C"enter the string:");
gets(st);
strupr(st);
strcat(st," ");
_setcursortype (_SOLID CURSOR);
clrscr();
textcolor(Red);
textbackground(white);
c = (40-strlen(st))/2;
gotoxy(c,12);
cpf C"%s", st);
while(!kbhit())
{
ch= st[0];
for(i=0; i< strlen(st); i++)
{
st[i]= st[i+1];
}
st[i-1] =ch;
st[i] = '\0';
gotoxy(c,12);
cpf C"%s", st);
delay(100);
}
getch();
}.
```

Kbhit = Keyboard hit.
Equal to enter key it means until press enter key we didn't do anything.

* < Ctype.h> → character type.

This is char. type.h fun's, they will accept 1 byte of char as the Ascii Code & they return integer type of dp either '1' or '0'.

Eg:
```
int isalnum(int c);
int islower(int c);
int isalpha(int c);
int isdigit(int c);
int isascii(int c);
int isupper(int c); ---etc.
```

```
*.  <ctype.h>
    Main()
    {
    char ch;
    pf("Enter the char:");
    ch=getchar();
    if(isupper(ch));
    pf("upper case");
    Else
    if(islower(ch))
    pf("lower case");
    Else if(isdigit(ch))
    pf("Digit");
    Else
    if(isascii(ch))
    pf("ASCII");
    getch();
    }.
```


toupper():- Convert from lowercase to uppercase

tolower():- Convert from uppercase to lowercase

ch='A' → ch = ch+32 = 'a'.

ch = ch-32 = 'A'.

if you want to accept more than one char. i.e; a string then string will be accepted by 2 derived datatypes, one is array & another is pointer.

```
*.  Main()
    {
    char str[]={"Anusha"};
    int i;
    i=0;
    while(i!=6)
    {
    pf("%c",str[i]); //Anusha
    i++;
    }
    }
```

This condition is not a perfect condition. since, the length of the string is increases & decreases, it is not a proper condition, to display the o/p, we need to write a condition that has to work for any no. of chars,

Basically Every String Ends with a Special char called as Null char.

```
*. Main()
{
char str[] = {"Anusha"};
int i=0;
while (str[i] != '\0')
{
pf("%c", str[i]);
i++;
}
}
```

Every String the last char. must be a null char. for the null char we need to specify one byte of Space.

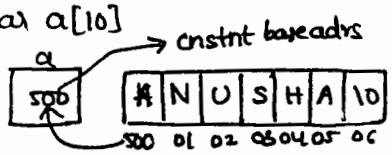
* What is the diff. b/w char * & char []?

char[]

```
Main()
{
char str[6] = "Anusha";
puts(str);
}
```

o/p: Anusha with garbage char's.

*. char a[10]




```
*. Main()
{
char a[10] = "Anusha";
puts(a);
a=a+2;
puts(a);
}
```

o/p: Error, since 'a' is an array which contains cnstnt base adrs never possible to change.

```
*. Main()
{
char a[10] = "Anusha";
char a1[10];
//a1=a; invalid
strcpy(a1,a);
puts(a1);
}
```

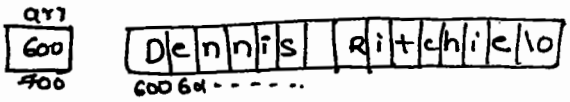
o/p: Anusha, The internal concept of Every predefined fun is written by ptrs, so, Either it is written by array also it works.

char*

```
Main()
{
char *str = "Dennis Ritchie";
puts(str);
}
```

o/p: Dennis Ritchie.

* char *arr.




```
*. Main()
{
char *a = "Dennis Ritchie";
puts(a);
a=a+7;
puts(a);
}
```

o/p: Dennis Ritchie
Ritchie.

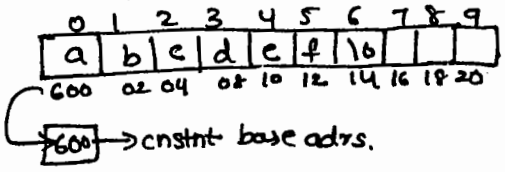
```
*. Main()
{
char *a = "Anusha";
char *a1;
a1=a;
puts(a1);
}
```

o/p: Anusha.

*. There is no concept internally written by array every thing on pointers.

*. Void main()

```
{
char str[10]="abcdef";
pf("\n %s", str+3); //def
str[3]='b';
pf("\n %s", str); //abcdef.
str[3] = 100;
pf("\n %s", str); //abcdef.
str[3]='10';
str[3]='0';
pf("\n %s", str); //abc0ef
str
Pf("\n %s", str); //abc
str[3]=97;
pf("\n %s", str); //abcaef
str[3]=0;
pf("\n %s", str); //abc.
str[3]='d';
pf("\n %s", str); //abcdef
}.
```



*. Void display (char *ptr)

```
{
pf("\n %s", ptr);
if (*ptr)
display (ptr+1);
pf("\n %s", ptr);
}
Void Main()
{
char str[10] = "Hello";
display (str);
}
```

o/p:

```
Hello
ello
llo
lo
o
.....
o
lo
llo
ello
Hello
```

*. Void Main()

```
{
char s1[50] = "Hello";
char s2[50] = "cobol";
s1 = s2;
puts(s1);
puts(s2);
}
```

o/p: Error

* Void Main()

```
{
char str[20];
str[0]='a';
str[1]='b';
str[2]='c';
puts(str);
}
```

o/p:- abc with garbage chars

Void main()

```
{
char str[20] = " ";
str[0]='a';
str[1]='b';
str[2]='c';
puts(str);
}
```

o/p:- abc

* <string.h> :-

1. strcpy(T,S); T→Target S→source ⇒ string copy.

   It copy the source to Target string.

2. strcat(T,S); String Concatenation

   It combine the source to the target i.e; concatinating.

3. strrev(T); It reverse the string.

4. strupr(T); It convert the string into uppercase.

5. strlwr(T); It convert the string into lowercase.

   All the above string fun's return type is Char*.

6. l = strlen(str); String length

   It find the length of the string including with spaces

7. l = strcmp(S1,S2); String comparision.

   It compare two strings based on the ascii codes.

8. l = stricmp(S1,S2); String ignored comparision.

   It compare 2 strings by ignoring case sensitive.

   The above 3 string fun's return type is integer.

* Void Main()
{
char S1[50] = "pascal";
char S2[50] = "windows";
strcpy(S2+4, S1+2);
puts(S1);
puts(S2);
}

p	a	s	c	a	l	\0

w	i	n	d	o	w	s	\0

s c a l.

o/p: pascal
windscal.

* Void Main()
{
char S1[50] = "Hello";
char S2[50] = "cobol";
strcat(S1, S2);
puts(S1); // Hellocobol
puts(S2); // cobol.
}

H	e	l	l	o	\0

c	o	b	o	l	\0

\0

o/p: Hellocobol cobol.

* strcat(S2+3, S1+3);

o/p: S1 = Hello . S2 = cobol\0

* strcpy will copy frm the given position & strcat always takes place frm the null char.

* strrev() → it reverse the string without null char.

```
Void Main()
{
char S1[50] = "Hello";
strrev(S1);
puts(S1);
}
```

o/p: olleH.

```
Void Main()
{
char S1[50] = "pascal";
strrev(S1+2);
puts(S1);
}
```

o/p: lacs.

KIRAN SIR
NOW WITH
Santosh Technologies
Cell : 9246392345

```
*. Void Main()
{
char s1[50]="Hello89101112131";
strrev(s1+5);
puts(s1);
}
```
o/p:- Hello31211110198

Strlen():- It finds the length of the string including with spaces.

String length never find out length of the null char.

```
* Main()
{
char s[100];
puts("Enter the string");
gets(str);
i=strlen(str);
pf("%d",i);
pf("\n%d",strlen(str));
getch();
}
```
o/p:
This is a data
14
14.

```
* Main()
{
int i;
char str[50]="abcdef g";
pfc("%d\n%d\n%d",strlen(str),sizeof(str),
        sizeof("abc"));
}
```
abcdef-g

o/p: 8 50 4

* size of operator will count the null character.

```
* Main()
{
char s[]="Rendezvous!";
pfc("\n%d",*(s+strlen(s)));
}
```

R	e	n	d	e	z	v	o	u	s	!	\0

o/p:- Zero.

* **StrCmp :-** The strcmp will compare the ascii code.

if the string1 ascii code is greater than the string2, Then by the diff's of the first 2 chars it returns a +ve value.

if the string2 ascii code is greater than the string1. Then it return the diff's of in -ve value.

if the total string is same in that chars of ascii value then it return Zero.

```
Void Main()
{
char s1[50]="abcd";
char s2[50]="abcd";
int i;
i=strcmp(s1,s2);
pfc("%d",i);
}
```
o/p: Zero.

(*) It compare 1st 2 chars if they are same it moves to the next char's. This comparision will be take place until the char. differs. it also compare the null chars.

*. "abcdef".
"abc@aa".

97	98	99	100	101	102
97	98	99	97	97	97
0	0	0	3	4	5

o/p:- 3

*. "abc"
"abcdef"
o/p:- -100

*. "Hello abc";
"Anuabc";
i = stricmp(s1+5, s2+3);
pfc"o/d", i); // zero.

H	e	l	l	o	a	b	c	\0

A	n	u	a	b	c	\0

* Stricmp :- it compare a strings by ignoring CaseSensitive.

```
Main()
{
char s1[50] = " Anusha Reddy";
char s2[50] = " Anusha Reddy";
int i;
i = stricmp(s1, s2);
pfc"o/d", i); //zero.
}
```


* WAP accept a string of paragraph find out no. of words in that paragraph, display the count;

```
Main()
{
char *ch;
int i, count = 1;
puts("Enter the String");
gets(ch);
for(i=0; ch[i] != '\0', i++)
{
if (ch[i] == 32)
count++;
}
pf("no. of words : %d", count);
getch();
}.
```

o/p: Enter the String
This is a data
No. of words : 4

* WAP accepting a string of paragraph, display the paragraph in the formate of each word for separate line.

```
Main()
{
char *s;
int i;
puts("Enter the string");
gets(s);
for(i=0; s[i]!='\0'; i++)
{
if (s[i] == 32)
s[i] = '\n';
} pf(str);
getch(); }.
```

o/p: Enter the String
This is a data
This
is
a
data

* WAP accept a decimal no. display that no. in any format like Hexa, octal, by using a String fun.

```
#include <stdio.h> <conio.h> <stdlib.h>
Main()
{
char str[40]; int n;
clrscr(); → textmode(a);
pf C"decimal no");
Sfc("%d", &n);
itoa (n, str, 2);
pfc"Binary no. given no: %s", str);
itoa (n, str, 8);
pfc"In octal no. given no: %s", str);
itoa (n, str, 16);
pfc"In Hexa decimal no. given no: %s, str);
getch();
}.
```

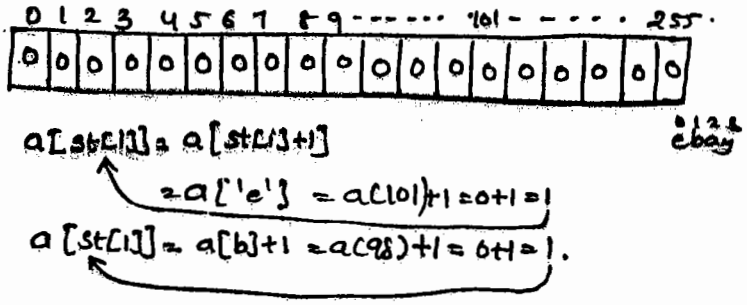
conversion 4m integer to string

itoa → integer to string (Src, destn, base)
integer string 2, 8, 10, 16

O/P: Enter decimal no: 10
Binary no. given no: 1010
octal :12
Hexa :a.

* WAP accept a string of paragraph find out the frequency of each char. How many times it occurs in the given String.

```
Main()
{
char *str; //char s[1000];
Static int a[256]; → 256 is the total no. of ascii value of total char's.
clrscr();
puts C"Enter the string");
gets(str);
for(i=0; str[i]!='\0'; i++)
{
a[str[i]] = a[str[i]]+1;
}
pfc"char \t\t freq.");
for(i=0; i<256; i++)
{
if(a[i]!=0)
{
pf C"In %c \t\t %d", i, a[i]);
}
}
getch();
}.
```



* WAP accept a word, delete the word frm the given string.

```
<stdio.h> <conio.h> <string.h>
Void main()
{
char st[60], dw[20], wd[20];
int i, j, k, id=0, ck=0;
clrscr();
pf("Enter the string:");
gets(st);
pf("Enta word to delete:");
gets(dw);
for(i=0; i<=strlen(st); i++)
{
if(st[i]=32|| st[i]='\0')
{
wd[id]='\0';
if(strcmp(dw,wd)==0)
{
ck=1;
if(st[i]=='\0')
st[i-strlen(wd)]='\0';
Else
{
for(j=i-strlen(wd), k=i+1; st[k]!='\0'; j++, k++)
{
st[j]=st[k];
}
st[j]='\0';
}
}
id=0;
}
Else
{
wd[id]=st[i];
id++;
}
}
if(ck==0)
pf("word not found");
Else
pf("Given String after deletion: %as", st);
getch();
}
```


I/P: /* A class is a group of common behaviours & common Relationships */

O/P: enter word to delete: common

Given String after deletion: A class is a group of behaviours & Relationships.

* WAP accept a string findout the given string is palindrome or not?

```
Void Main()
{
char st[30];
int k;
int palindrome (char[]);
pf ("Enter the string");
gets (st);
k = palindrome (st);
if (k==0)
pf ("given string not palindrome");
Else
pf ("given string is palindrome");
}
int palindrome (char st[])
{
int i, j, len =0;
while (st[len]!='\0')
{
len ++;
}
for (i=0; j=len-1; i<j; i++; j--)
{
if (st[i]!=st[j])
return 0;
}
return 1;
}
```


* WAP accept 2 char's replace the 1st char with the second char's from the given string.

```
<stdio.h> → Void streplace (char[], char, char);
Void main()
{
char ch1, ch2;
char st[25];
puts ("enter the string");
gets (st);
puts ("enter the 1st char:");
fflush (stdin);
ch1 = getchar();
puts ("enter the 2nd char:");
fflush (stdin);
ch2 = getchar();
streplace (st, ch1, ch2);
Puts (st);
getch(); }
```

```
Void streplace (char *ptr, char p1, char p2)
{
for (; *ptr; ptr++)
{
if (*ptr==p1)
*ptr = p2;
}
}
```

* WAP accept a string of paragraph, convert the lower case chars to uppercase & upper case to lower case from the given paragraph.

```
<stdio.h>
Void convert (char[]);
Void main()
{
char st[25];
puts("enter the string");
gets(st);
convert(st)
puts(st);
getch();
}
Void convert (char *ptr)
{
for(; *ptr; ptr++)
{
if(*ptr>=65 && *ptr<=90)
*ptr+=32;
else
if (*ptr >=97 && *ptr<=122)
*ptr-= 32;
}.
```


* WAP copy one string data into another String

```
<stdio.h>
Void copy (char[], char[]);
Void main();
{
char st[25], s[25];
puts ("enter the string:");
gets (st);
copy (s, st);
puts ("After copying");
puts (s);
getch();
}
Void copy (char *ptr, char * pptr)
{
for (; *pptr ; ptr++, pptr++)
* ptr = * pptr;
*ptr = '\0';
}
```

(or)

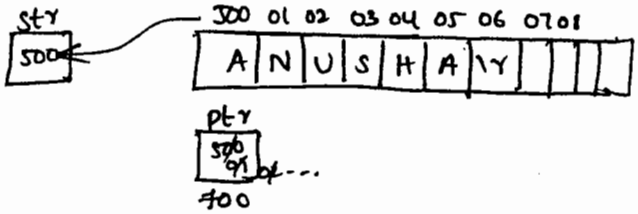
```
Void copy (char *ptr, char *pptr)
{
while (* ptr++ = *pptr++);
}.
```

String Concatenation:-

```
Void xxxx (char *T, char *S)
{
for (; *T; T++);
while (*T++ = *S++);
}.
```

Implement the logic of password:-

```
<stdio.h> <string.h>
Void Main()
{
Void password (char *);
char str[50];
int a,b;
clrscr();
pf ("Enter password");
password (str); //callee
if (strcmp (str, "Anusha") != 0)
{
pf ("In password not correct");
getchch();
Exit(1);
}.
/* any other logic Execute if password is right */
a=5, b=10;
pf ("\n old", a+b);
getch();
}
Void password (char *ptr) //callee
{
while (1)
{
*ptr = getch();
if (*ptr == '\r')
{
*ptr = '\0';
break;
}
pf ("*");
ptr++;
}
}.
```



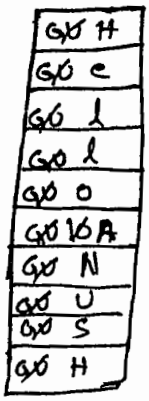
password logic with backSpace :-

```
<stdio.h> <conio.h> <process.h> (failed)
void main()
{
char st[40], ch; /* const char *exe; */
int i=0;
clrscr();
gotoxy(20,12);
pfc" enter password:");
while(1)
{
ch=getch();
if (ch==13)
break;
else
{
if (ch==8)
{
if (i>0)
{
pfc("\b");
pfc("%c", 32);
pfc("\b");
i--;
}
}
else
{
st[i]=ch;
i++;
pfc("*");
}
}
}
st[i]='\0';
if (strcmp(st,"Anusha")==0)
{
gotoxy(35,20);
pfc("please wait ----");
System("e:\\grep.exe");
}
else
{
gotoxy(35,20);
pfc("Invalid password");
getch();
}
```

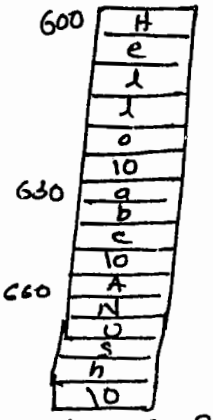
upto right now we are storing only single strings if you want to store collection of names we require double dimension.

* How the collection of strings will store in a double dimension:-

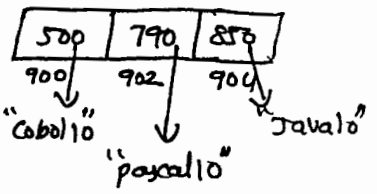
```
*. Void Main()
{
char st[2][5];
puts (str[0]); //gv
puts (str[1]); //gv
strcpy (str[0], "Hello");
puts (str[0]); // Hello
puts (str[1]); // Empty.
strcpy (str[1], "Anusha");
puts (str[0]); // Hello Anush
puts (str[1]); // Anush.
}
```




```
*. Void Main()
{
char s1[3][30]={"Hello", "abc", "Anush"};
char *s2[3] = {"cobol", "pascal", "java"};
int i;
for(i=0; i<3; i++)
puts (s1[i]);
for(i=0; i<3; i++)
puts (s2[i]);
}
```

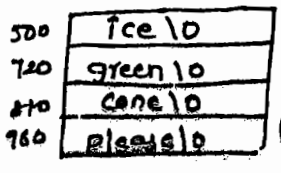


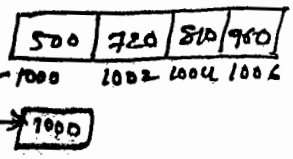
Array is Sequential.

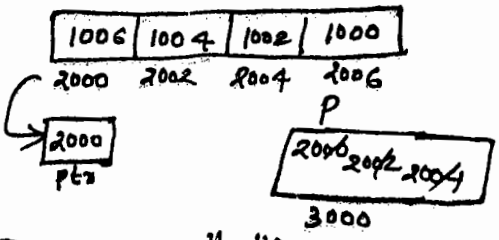


pointer is random.

```
*. Main()
{
static char *s[] = {"ice", "green", "cone", "please"};
static char **ptr[] = {s+3, s+2, s+1, s};
char **p = ptr;
ptf("\n%s", **++p);
ptf("\n%s", *--*++p+3);
ptf("\n%s", *p[-2]+3);
ptf("\n%s", p[-1][-1]+1);
}
```







**++P = **2002 = *1004 = 810 ("cone\0")

--++P+3 = *--*1004+3 = *--1002+3

= *1000+3 = 500+3 = 503 ("ice\0")

*P[-2]+3 = **(P-2)+3 ⇒ **2000+3 = *1006+3 = 963 ("age\0")

P[-1][-1]+1 = *(*(P-1)-1)+1 ⇒ *(*2002-1)+1 = *(1002)+1

= 721 ("reen\0")

Structures

Typedef:- The Typedef is used to provide alias names to the existing datatype

Syntax:- Typedef old-data typename newdatatype name;

Eg:- Typedef int Identity;
Identify BookId;
identity sid;

*. Main()
{
typedef int number;
typedef float decimal;
typedef char * string;
number s=34.56;
decimal k;
string s1, s2, s3 = "abc";
pf("In %d %d", sizeof(s), s);
pf("In %d", sizeof(k));
pf("In %d %s", sizeof(s1), s3);
getch(); o/p: 2 34
}. 4
2 abc

. #define cp1 char
typedef char *cp2;
Void Main()
{
cp1, p1, p2, p3; // it is replaced with char2 *
cp2, p4, p5, p6; // it is alias name to char *
} Here, p1 p4 p5 p6 are pointers & p2 p3 are normal variables.

char * = p1, p4, p5, p6 & char = p2, p3

* as below that array is an homogeneous type of data it can hold only same type of Elements in order to hold heterogeneous type of data we use structure concept.

* By using structures we can create a users defined data types.

* It is a type collection also called as an agregate data type of compound datatype.

* when a structure is defined we define a compound variable datatype.

*. structure is a collection of one & more variable, possibility of diff. types grouped together under a single name for convinient handling.

Array pointers & other structures, unions & Enums can also be included in structure as a Elements.

The individual members can be ordinary variables, arrays, pointers, & other structures.

By using structure & union we can create our own datatypes.

Syntax:-
```
Struct <structure-name>
{                          → tag name is user defined datatype name.
member-ele 1;
member-ele 2;
- - - - - - - -
member-ele n;
};
```

The Member elements can be accessible by using 2 operators.

1. Member access operator (.)
2. pointer to Member access operator (--->)

* Create a structure called as student?

```
<stdio.h>
Struct student → user defined datatype.
{
int sid;
char *sname;
float fees;
};
Main()
{
struct student sai;   → datatype, → variable name
clrscr();
pf("size of structure: %d %d", sizeof(sai), sizeof(struct student)); // 8 8
pf("Enter the data ---\n");
sf("%d %s %f", &sai.sid, sai.sname, &sai.fees); // sai: {121, "scott", 6789.000000}
pf(" \n The data is ---\n");
pf("%d %s %f", sai.sid, sai.sname, sai.fees); // 121 scott 6789.000000.
getch();
}
```


The sizeof the structure is, total member elements of the structure.

Structures can be defined in diff. ways by initializing the structure variables also.

*.
```
Struct Emp
{
int id;
char name[30];
int sal;
};
```

```
Void Main()
{
emp e1,e2,e3; //invalid in c (valid yes in c++)
Struct Emp e1, e2,e3;
struct Emp e4 = {4,"xyz", 45006}; //valid
e1.id=1;
e1.name="abc"; //invalid if array pointer OK
e1.name = abc; //invalid.
e2.id = e1; // (valid)
e2.name = e1.name;
e3.sal = e1.sal+e2.sal;
```

```
*. Struct Emp
{
int id;
char name[20];
int sal;
};
typedef struct Employee Emp;
Void Main()
{
struct Employee e1,e2,e3; //yes
emp e4,e5,e6; //yes
}.
```

```
*. Struct Emp
{
int id;
char name[20];
int sal;
};
e1,e2 = {2,"xyz", 12000} ,e3; //yes
// e1e2e3 are bcom global variables.
Void Main()
{
Struct emp e4,e5; //yes
//e4,e5 are local var's
}.
```

any where any type of global var's contain zero, not GV's.

Nameless Structures :-

```
* Struct
{
int id;
char name[20];
int sal;
} e1,e2,e3; //yes
Void Main()
{
Struct e4,e5; //no
}
```

```
*. Typedef Struct
{
int id;
char name[20];
int sal;
} EMP; //yes// EMP becomes alias name to the Structure
Void Main()
{
EMP e1,e2,e3; //yes
}
```


Define a Structure inside the Main():-

```
Void Main()
{
typedef Struct
{
int id;
char ename[20];
int sal;
} EMP; //yes
} //Main
```

```
Void abc()
{
EMP e1,e2,e3;//no
}
```

↕ Elements not declared outside the Main

Array of Structures :-

Array of structures concept, inorder to implement to access more than one record

```
<stdio.h>
{
int sid;
char name[20];
float fees;
};
Main()
{
struct student s[5];
int i;
clrscr();
pf("size of structure : %d %d", sizeof(s), sizeof(struct student));
for(i=0; i<3; i++)
{
pf("Enter the %d Record : \n", i+1);
sf("%d %s %f", &s[i].sid, s[i].sname, s[i].fees);
}
pf("\n The Records are --- \n");
for(i=0; i<n; i++)
pf("%d %s %f \n", s[i].sid, s[i].sname, s[i].fees);
getch();
}
```

o/p:-

```
size of structure : 140 28
Enter the 0 Record : 121
                     Scott
                     4532
Enter the 1 Record : 131
                     clark
                     5664
Enter the 2 Record : 141
                     Anu
                     7789
Enter the 3 Record : 151
                     anusha
                     3693
The Records are :    131 clark  5664
                     141 Anu    7789
                     151 Anusha 3693
```


When we run this prgm on TurboC Editor it raises an Error of floating pt. formats not linked. The compiler not supporting the fun. so, user has to define.

```
link_float()
{
float a, *b;
b=&a;
a=*b;
}
```

These are some cases in which the reference to the float is a bit obscure & the compiler doesnot detect the need for the Emulator (app^n).

The most common is using scanf() to read a float in an array of structure as shown in our prgm.

we can force the format to be linked by the using link_float() fun. it forces linking of the floating_point Emulator into an app^n. There is no need to call this fun. just include where ever necessary in ur prgm.

Dynamically allocating Memory for the Structures:-

```
<stdio.h> <conio.h>
typedef struct student
{
int sid;
char sname[20];
float fees;
}std;
Main()
{
std *s;
int i=1;
char opt;
clrscr();
pf("Allocate the Memory dynamically...\n");
do
{
s= (std *) malloc(sizeof(std));
if(s==NULL)
{
pf("\n memory not allocated propely\n");
getch();
Exit(0);
}
pf("\n Enter the details of %d Record:",i);
sf("%d%s%f",&(*s).sid, s->sname, &s->fees);
pf("\n The details of %d Record",i);
pf("%d %s %f\n", s->sid, (*s).sname, s->fees);
i++;
free(s);
pf("\n do u want to create one more record (Y/N):");
fflush(stdin);
opt=getchar();
}//do
while(opt=='Y'||opt=='y');
}
link-float()
{
float a,*b;
b=&a;
a=*b;
}.
```


So, dynamically there is no wastage of memory loss will not takeplace.

Structures with in Structures:-

One structure var. acting as a member element to the another structure is called as structures within structures.

```
<stdio.h> <conio.h>
Struct x
{
int a;
};
Struct y
{
int b;
Struct x x1;
};
Struct z
{
int c;
Struct yy1;
};
Main()
{
Struct z.z1;
z1.c=10;
z1.y1.b=20;
z1.y1.x1.a=30;
pf("\nd\nd\nd",z1.y1.x1.a,z1.y1.b,z1.c);
getch();
}
```


Nested Structures:-

```
Struct emp
{
int eno;
char cname[20];
float sal;
Struct fac
{
char pno[10];
char sub[20];
}f; // fac
   ↳ mandatory
}; // Emp
Void main()
{
Struct Emp e1; //56 byte
Struct fac f1; //30 byte
e1.eno=121 //yes
strcpy(e1.name,"Anusha"); //yes
e1.sal=10000; //yes
e1.pno="9999999999"; //Error , bcoz it is string, we
                        have to use string fun.
e1.f.pno="0000000000"; //Error
strcpy(e1.f.pno,"1234567890"); //yes
strcpy(e1.f.sub,"C&C++");
}
```

Unions:- Union is also user defined data type similar to structures concept but they have some diff's will be take place btw unions & structures.

Every app^{n} of unions is possible in structures but Vice-Versa is not possible.

Syntax:- Union <union_name>
{
Member_ele 1;
Member_ele 2;
- - - - - - -
Member_ele n;
};

Eg:-

```
Struct S
{
int a;
float b;
};
union u
{
int a;
float b;
};
Main()
{
struct S s1;
union u u1;
pf("structure : %d\n", sizeof(s1)); //6
pf("union : %d\n", sizeof(u1)); //4
getch();
}
```

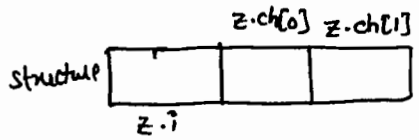
*.
```
Main()
{
Struct S s1 = {12, 56.535}; //valid
union u u1 = {34, 2.452}; // Error, it
                          can't initialize at a time
u1.a = 56;
pf("%d\n", u1.a);
u1.b = 342.535;
pf("%f\n", u1.b);
getch();
}
```

The sizeof structure is the "Total" member-Elements of the structure.

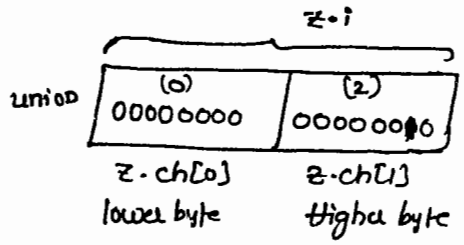
The sizeof union is the "Highest" member-Elements of the unions.

In structures we can initialize all the member ele's at a time. but In unions it is not possible, in unions at a time one Memory location used for only one member Element.

*.
```
Main()
{
Struct a
{
int i;
char ch[2];
};
struct a z;
z.i = 512;
pf("%d", sizeof(z));//4
pf("%d %d %d %d", z.i, z.ch[0], z.ch[1]); // 512, gv, gv
}
```



*. Main()
{
union a
{
int i;
char ch[2];
};
union a z;
z.i = 512;
pfc("%d", sizeof(z)); //2
pf("%d %d %d %d", z.i, z.ch[0], z.ch[1]); // 512 0 2
}.



512 = 1000 000000000

* In what situations unions better to use rather than Structures concept:-

Applications of Unions:-

if u store Employee info,

Highly scale HSK

Semi scale SSK

One way to store Structures, structures wasting 2 slots Either any 2 used but not both then we can store by unions.

*. UTILITY

Name
Grade
Age
If Grade = HSK
Hobbie Name
Credit card No.
If Grade = SSK
Vehicle No.
Distance frm company.

Ans:

```
Struct info1
{
char hobby[10];
int credit no;
};
Struct info2
{
char veh.no[10]; // 12 bytes
int dist;
};
```

```
Union info
{
Struct info1 a; // 12 bytes
Struct info2 b; // 12 bytes
};
Struct emp
{
char name[20];
char grade[4];
int age;
union info f;
};
```

Implementing by unions we are saving nearly 12 bytes of Memory.

Enumeration (enum):-

It is also similar to user defined type, the Enumerations member Elements are constants.

By using enumerations we can define not only member Elements it also tells what type of data has to hold.

Syntax:- enum Enumeration-name
{
Member-ele 1, Member-Ele 2, Member-ele 3 = <value> ..... Member-ele n;
};

The Member Elements internally accept constant values as an integer no's, Enumeration make the prgm more readable.

Eg:-
```
Enum BOOL
{
false,
True
};
```

* Enum will accept series of integer constants.
* it is an alias to integer
* Size of Enum is 2 bytes.
* Through Enum we can generate Series of integer constants.
* Major Menu based prgming will be takes place the Enumerations.
* Enum color {red, blue, white, yellow=5, green};
  color s;
  s = green;
*
```
<stdio.h>
Void main()
{
Enum color { BLACK, WHITE, GREEN};
enum color x;
Pf("\n %d . BLACK", BLACK);
Pf("\n %d . WHITE", WHITE);
pf("\n %d . GREEN", GREEN);
pf("\n Select a color !");
Sf("%d", &x);
switch (x)
{
Case BLACK: pf("\n BLACK is selected"); break;
case WHITE: pf("\n WHITE is selected"); break;
case GREEN: pf("\n GREEN is selected"); break;
}
}.
```



```
0. BLACK
1. WHITE
2. GREEN
Select a color ! 1
WHITE is selected.
```

*. enum MONTH {JAN=1, FEB, MARCH, ---- , DEC};

enum DAY { noob-Day-jan=31, noob-day-feb=28, noob-day-March= ..... };

```
Void Main()
{
//int dd;
 enum DAY dd;
 enum MONTH mm;
 //int mm;
 Switch(MM)
 Case DEC: dp+ = noob-days-NOV;
 // case 12: dp+=30;
   ----
   ----
   ----
   ----
 }
 }.
```

Projects in Technical Book

Dy!

Dynamic Memory Allocation

As wkt, static Memory allocations there is a wastage of Memory blocks. since, at the time of compilation only we have to define the size before Execution of the prgm. for this reason. Either we can't ↑se & ↓se the size. Inorder to allocate the Memory at the time of Execution with the size we require of dynamic memory concepts. what are the memory we allocated by the dynamic memory concept? it will be allocated from Heap area.

* Stack Vs Heap allocation?

The Memory allocated ints stack has unique name & adrs, therefore memory location can be accessed by name or through adrs.

The Memory allocated ints heap does not contain a name, instead contains only the starting adrs of the memory allocated, therefore can be accessed through this adrs.

The memory allocated through Heap can be local & global respective to its fun'al def^{n}.

If memory allocated as global then that memory can be accessed through a diff. fun. by locating its adrs.

In static Memory location the memory is allocated & freed by system.

In dynamic memory allocation it is the risk of the prgmr.

To allocate the memory dynamically we have separate fun's like malloc(), Calloc(), & realloc().

for deallocation we have free() fun.

All this fun's return type is void pointer (void *)

In K&R 'c return type is char *.

Malloc():- Used to allocate required no. of bytes in memory at runtime. It takes one arg. viz. size in bytes to be allocated.

Syntax:-
```
Void * malloc(size_t size);
(or)
pointer_Var = (type cast *) malloc (size of(size));
```

size_t is equivalent to the unsigned int data type malloc() can allocate a max. of 64 Kb.

Eg: `a = (int*) malloc(4);`

4 is the size (in bytes) of memory to be allocated

Calloc():- Used to allocate required no. of bytes in memory at runtime. It needs two arg's viz; 1. total no. of data 2. size of each data.

Syntax:- `Void * calloc (size_t nmemb, size_t size);`

Eg:- `a = (int *) calloc (8, sizeof (int));`

Here, sizeof indicates the size of the datatype & 8 indicates that we want to reserve space for storing 8 integers.

calloc provides 2 arg's nitems & size the parameter nitem specifies no. of times to allocate & size specifies the size of each item.

for Eg. to allocate 10 integers,

`ary = (int *) calloc (n, sizeof (int));`

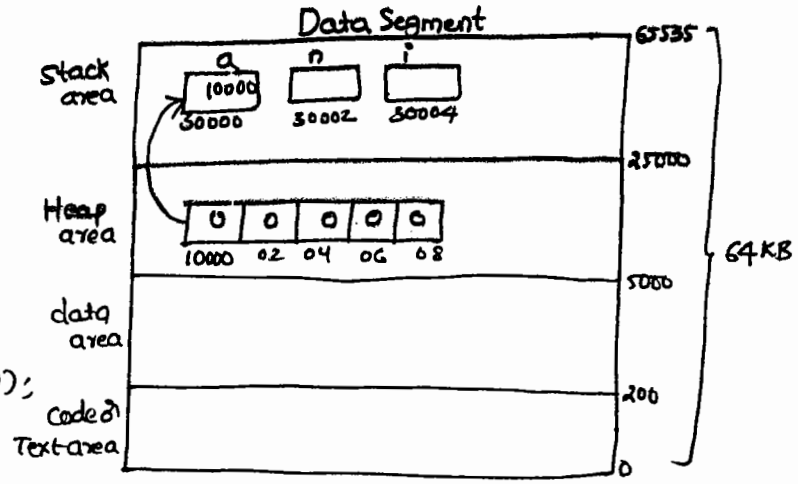
The fun. calloc allocates a blk of size (nitems * size).

* WAP which allocate the memory dynamically i.e; create dynamic single dimension array.

Note:- The diff's of malloc() & calloc(); Malloc() after allocating the memory it allocates garbage values. calloc() allocates 0's.

```
<alloc.h>
Main()
{
int *a, n, i;
clrscr();
pf("Enter no. of Elements:");
sf("%d", &n);
// memory allocation
a = (int *) calloc (n, size of (int));
// acccpting
pf("Enter Elements :\n");
for(i=0; i<n; i++)
sf("%d", a+i);
// logic

//display
for(i=0; i<n; i++)
pf("%d\t", *(a+i));
getch();
}
```

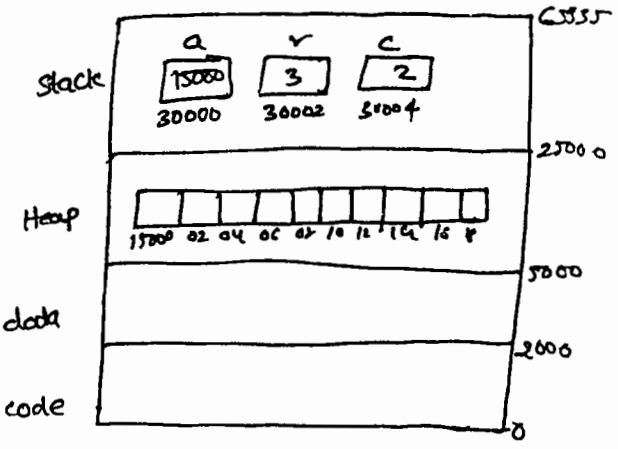


The pointer var. doesn't know what type of elements it is pointing & How many elements it is pointing.

The major diff. b/w static & dynamic memory is, for static size will be debined compilation time, dynamic memory size will be debined at Execution time. But memory allocation for both always execution time only.

Dynamic 2D-array:-

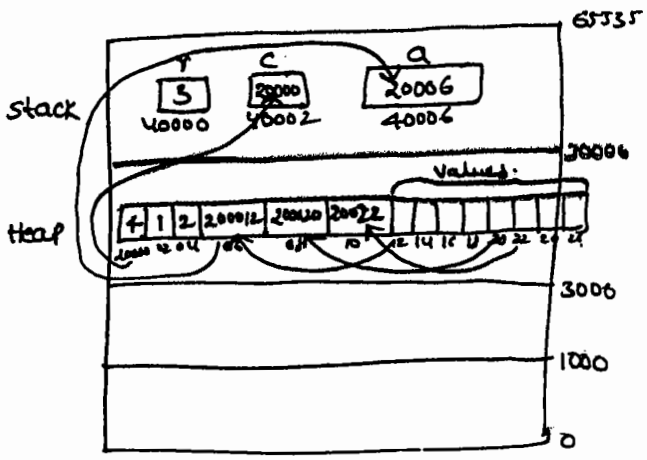
```
<alloc.h>
Main().
{
int **a, r, c, i, j;
clrscr();
pf("Enter the Rows & cols:");
sf("%d %d", &r, &c);
//row Memory
a = (int **) malloc (r * sizeof(int));
//col. Memory
for(i=0; i<r; i++)
{
*(a+i) = (int *) malloc (c * sizeof(int));
}
pf("Enter the values\n");
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
sf("%d", *(a+i)+j);
}
pf("\n Enter values\n");
for(i=0; i<r; i++)
{
for(j=0; j<c; j++)
pf("%d ", *(a[i]+j));
}
for(i=0; i<r; i++)
{
free(*(a+i));
a[i] = Null;
}
free(a);
a=Null;
getch();
}
```



* prgm on dynamic memory allocating for each row diff. column values:-

```
<alloc.h>
Main()
{
int **a, r, *c, i, j;
clrscr();
pf("Enter the no. of rows:");
sf("%d", &r);
```

```
// row memory for holding col. values
 c = (int *) malloc (r * sizeof(int));
for(i=0; i<r; i++)
{
 pf ("Enter no. of cols for row no. %d: ", i);
 sf ("%d", c+i);
}
a = (int **) malloc (r * sizeof(int));
for(i=0; i<r; i++)
{
 *(a+i) = (int *) malloc (*(c+i) * sizeof(int));   → mul. operator.
}
// Accepting
 pf ("Enter the values: \n");
 y = wherey();
 k = wherex() + 4;
for(i=0; i<r; i++)
{
 x = k;
 for(j=0; j<*(c+i); j++)
 {
  gotoxy(x, y);
  sf ("%d", *(*(a+i)+j));
  x = x + 4;
 }
 y++;
}
 pf ("\n\n Given values: \n");
 y = wherey();
 k = wherex() + 4;
for(i=0; i<r; i++)
{
 x = k;
 for(j=0; j<*(c+i); j++)
 {
  gotoxy(x, y);
  pf ("%d", *(*(a+i)+j));
  x = x + 4;
 }
 y++;
}
 getch();
}
```



o/p:

```
Enter no. of rows : 3
Enter no. of cols for row 0 : 4
                      1 : 1
                      2 : 2
Enter the values :
        12 56 67 90
        34
        23 65
 Given values :
    12 56 67 90
    34
    23 65.
```

Realloc():-

This fun. is used to increase & decrease the size of any dynamic memory which is allocated using malloc() & calloc() fun's.

Syntax:. Void realloc (Void *ptr; size_t newsize);

The 1st arg. 'ptr' is a ptr to the memory previously allocated by the malloc & calloc fun.

The 2nd arg. 'new size' is the size in bytes, of a new memory region to be allocated by realloc().

This value can be larger & smaller than previously allocated memory. The realloc() fun. adjusts the old memory region if size is smaller than the size of old memory.

If the new size is larger than the existing memory size, it uses the size by copying the contents of old memory region to new memory region. The fun. deallocate the old memory region. realloc() fun. is helpful in managing a dynamic array whose size may change during execution.

for Ex.'s of realloc() refer website.

free():-

The free() fun. deallocates the memory as per the dynamic memory deallocation will not takes place until the user explicity deallocate.

Note:. If the user not deallocated when the system shutdown it will be deallocated.

Syntax:- free (pointer-var);

Macros

The C preprocessor is a prgm that process our source prgm before it is passed to the compiler. This Macros is one type of lang. which is available mostly in C & C++.

features of C preprocessors:-

The C prgm involves so many stages frm the stage of writing a C prgm to the stage of getting executed.

The processors used in the Execution of C prgm are,

1. Text editor 2. preprocessor 3. compiler 4. linker.

preprocessor directives are:

1. Macro Expansion 2. file inclusion
3. Conditional compilation 4. Miscellaneous directives.

1. Macro Expansion :-

In this Macro Expansion directives we mostly use one type of macro i.e; #define.

#define is a preprocessor directive which is created before compilation stmts. The #define can be used in 3 ways.

① To generate constants

#define A 10

The left side will be replaced with Right side, inplace of 'A' Every time "10" will be Replaced & Substituted.

```
* #define A 10
  void main()
  {
  int i;
  i=A;
  pf("\n%d%d", i, A);
  }
```

Replacement:-

```
#define A 10
void main()
{
int i;
i=10;
pf("\n%d%d", i, 10);
}
```


*. #define size 10
#define ISEMPTY top NULL

② It is used to provide alias name for existing datatypes & funls.

*. #define LI long int
#define pf printf

It also provides some names of aliases for operators also.

*. #define AND &&
#define OR ||

3. it is used for code replacements.

The Macros can be defined with arg's also.

*. #define mul(a,b) a*b

```
Void Main()
{
int i;
i= MUL (2,3);
pf ("%d", i);
i= MUL (2+3, 2+3);
pf ("%d", i);
}
```

o/p:- 6,11

Replacement:.

```
Void Main()
{
int i;
i= 2*3
pf("%d", i);
i = 2+3*2+3;
pf("%d", i);
}
```

* #define mul(a,b) (a)*(b)

```
Void Main()
{
---
Same as above
---
---
}
```

Replacement:.

```
Void main()
{
int i;
i= (2)*(3); //6
pf ("%d", i);
i= (2+3)*(2+3);
pf ("%d", i);
}
```

Testing of the Macros:-

A Macro can be tested by saving the prgm in the slw folder

c:\tc\bin\filename.C

after saving the file go to the DOS prompt, c:\tc\bin.

G:\TURBOC2 > cpp m1.c ↕ cpp -> C preprocessor

Internally intermediate code is generated.

G:\TURBOC2 > type m1.i

```
Void Main()
{
:
}
```

* #define ARRANGE (a>25 AND a<50)

```
#define AND &&
Main()
{
int a =30;
if (ARANGE)
pf("within Range");
Else
pf("out of Range");
}
```

*.
```
#define size 100
void main()
{
int a;
a=++size;
pf("%d%d%d", size, a);
}
```
Error, cnstnt no's not ↑ed

Replacement:-
```
void main()
{
int a;
a=++100;
pf("%d%d%d", 100, a);
}
```

*.
```
#define size a
void main()
{
int a;
a=100;
a=++size;
pf("%d%d%d", size, a); // 101, 101
}
```

Replacement:-
```
void main()
{
int a;
a=100;
a=++a;
pf("%d%d%d", a, a);
}
```

Macro's More than One line ☆ Stmts:-

Macro's can be defined more than one line when you defining in more than one line ☆ stmt. we have to end the every line with "backslash".

```
#define PRINT(n,c) {\
for(i=1; i<n; i++)\
{\
for(j=i; j<n; j++)\
pf(" ");\
for(j=1; j<=i; j++)\
pf("%c", c);\
pf("\n");\
}\
}\
main()\
{
int i,j;
clrscr();
PRINT(6,'*');
PRINT(5,'#');
getch();
}
```


Macro's & fun's:-
```
#define sum(x,y) x+y
void main()
{
int s;
s=sum(10,20);
pf("%d", s); //30
}
```

Replacement:-
```
void main()
{
int s;
s=10+20;
pf("%d", s);
}
```

```
*. int Sum(int x, int y)
   {
   return x+y;
   }
   void Main()
   {
   int S;
   S = sum(10,20);
   pf("%d",S); //30
   }
```


diff's of Macro & fun's:-

Macro's are like fun's but there is diff. blw those two.

1. In Macro call the preprocessor replaces the macro template with its macro expansion. whereas in fun. call the control is passed to a fun. along with certain arg's some fun's are peformed in the fun. & a usebul value is returned back frm the fun.
2. Macros make the prgm run faster but ↑ses the prgm size, whereas fun's make the prgm smaller & compact i.e, if the macro is used 100 times in the prgm, the macro expansion goes into source code for 100 times which ↑ses the prgm size, otherwise if the fun. is used 100 times in the prgm frm diff. places it takes same amount of space.

In the fun's while passing the arg's to a fun. & getting back the returned value doesn't take time & slow down the prgm. where as in macros this prblm won't occurs since they have already been Expanded & placed in the Source code before Compilation.

2. file Inclusion:-

#include. This directive causes one file to be included in another. The preprocessor cmd for file Inclusion is, #include "filename". This cmd inserts the Entire contents of filename into the source code at that pt. in the prgm. This file inclusion preprocessor is used in 2 cases they are.

1. If the prgm is very large the code is devided into several diffr. files, Each containing a set of related fun's these file inclusion preprocessor is used. These files are included at the begining of main prgm.
2. fun's & macro defn's are needed in Many prgms commonly. In such situations these commonly used fun. & macro defn's are stored in a file, & that file can be included in Every prgm. This will add all the stmnts. in this file to prgm which is written. In prgmng generally files are included with the Extension .h, this Extension stands for 'header file'. It contains stmts which when included go to the head of prgm.

There are 2 ways to write #include stmt.

They are : #include "filename".

#include <filename>.

The diff. blw these 2 stmts is as shown below.

when u place in < > it check the file only in the slw directory. i.e; Tc\include, it doesn't checks any local directories. if u place in " " first it checks the slw directory if it is not available then it go to the local directories.

Example On file Inclusion:-

```
<stdio.h> <conio.h>
#define begin main()
{
#define print(a) printf (#a);
#define end getch();
}
#define numba int
#define characta char
#define printint(a) printf ("%d",a);
#define readint(a) scanf ("%d", &a);
#define printchar(a) printf ("%c",a);
```


Save the files with.h extension, open a new file, substitute the before file save as c:\ Alg.c.

```
*. #include "c:\alg.c"
   begin
   numba n;
   characta c;
   print (Enta any value)
   readInt (n)
   c=n;
   print (Given value:)
   printint(n)
   printchar(c)
   End.
```

3. Conditional Compilation:- It is a 3rd type of macros diff. types of conditional compilation are present in order to reduce the code of Executive files & other stmts we use this conditional compilations.

Eg:- #if #ifdef
#else #undef ---- etc.

Syntax:-

```
#ifdef Macroname
Stmt 1;
Stmt 2;
Stmt 3;
#Endif;
```

Eg:-
```
#define pf printf
Void main()
{
pf C"A");
#if 7>11
pfc" B");
pfc" c");
# End if
pf C" Hello");
}
```
o/p:- AHello

When u working with this preprocessor depends on the condition it decide either the code need to execute & not. This type of conditional macros we can reduce the size of the executable files.

Satisfied → Compiled, not Satisfied → not Compiled.

```
Void Main()
{
pf C"Hello");
#if 5! =5>2
pf C"A");
pf C"B");
# else
pfc"x");
pf C"y");
# Endif
pfc"welcome");
}
```
o/p:- HelloAB welcome

```
Void main()
{
pf C" Hello");
#ifdef TEST
pfc"A");
pf C" B");
#Endif
pf C"welcome");
}
```
o/p:- Hello welcome.

- when we compiling inside the TEST defining, stmts doesn't executed. just like a cmds the stmts are skipping.

*.
```
#define pf printf
#define TEST
Void main()
{
pfc"Hello");
#ifdef TEST
pfc"A");
pf C" B");
#Endif
pf C"welcome");
}
```
* By using this (#ifdef) macro compilation time will save user flexibility

o/p:- HelloABwelcome

Undef :- By using #undef we can close the scope of existing define macro.

*.
```
#define A 10
Void Main()
{
pf("%d", A); //10
#undef A
#define A 20
pf("%d", A); //20
#undef A
pf("%d", A); // Error, bcoz of without
}                defining of Macro.
```

*.
```
#define pf printf
#define A 10
Void main()
{
pf("%d", A); //10
fun();
pf("%d", A); } //10
fun()
{
#undef A
#define A 5
}
```
Bcoz, func() inside the main, dat's y no error.

4. Miscellaneous directives :-

pragma → This directive will be used for on/off options.
This will be works depends on the Editors.

// supress Warnings.

#pragma warn-rvl //return value
#pragma warn-par // parameter not used
#pragma warn-rch // unreachable code

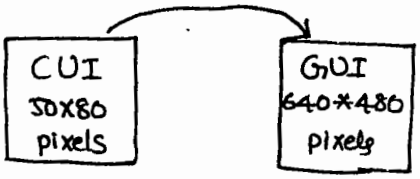
```
int f1()
{
int a=5;
}
void f2 (int x)
{
pf("inside f2\n");
}
int f3()
{
int x=6;
return x;
x++;
}
void main()
{
f1();
f2(7);
f3();
}
```


Error :- To create Error Msgs.

Graphics

In order to work with graphics first we have to convert into graphics. The graphics Environment will support both the CUI & GUI.

To convert to GUI we have to initiate the graphics system by initiate graph.



initgraph (graphic Drives, graphic mode, path); ↳optional.

The graphic drives detect what type of adopter present in ur system. The drivers can be CGA, MCGA, EGA ---- etc.

The graphic modes will depends on the drivers they supports diff. types of pixels depends on the modes.

The path of the file indicates supporting graphics file system.

BGI → Boreland Graphic Interface

C:\tc\bgi → To insert files or anything based on graphics.

* WAP to find out what type of drives is present in ur system & how many pixels it is supporting.

```
<graphics.h>
Void main()
{
int gd = DETECT, gm=0;
initgraph (&gd, &gm, "c:\tc\bgi");
pf("%d %d", getmaxx(), getmaxy());
setbkcolor(6); // background
setcolor (4); // foreground
line (0,0, 649, 472); // Create diagonal line
Rectangle (350, 220, 410, 590); // Rectangle
clear device (); // circle.
circle (300, 300, 70); // circle.
// pf("Hello");
set textstyle (1, HORIZ_DIR, 4);
outtextxy (150, 50, "GRAPHICS");
getch();
close graph(); // To close the graphics part
}
```

o/p:- 639 479

*. <graphics.h> <stdlib.h>

```
Main()
{
int gd = DETECT, gm=0, i;
Initgraph (&gd, &gm, "c:\\tc\\bgi");
Setbkcolor(4);
while (!kbhit())
{
for(i=0; i<=100; i++)
{
Setcolor (random(16));
circle (300, 300, i);
delay (10);
}
}
getch();
closegraph();
}
```

FILES

Many app's require the info. to be written to or read frm an auxilary memory device. Such info is stored on memory device in the form of data file.

Data file allows to store info. permanently & access, alter that info. whenever necessary.

There are 2 diff. types of data files. They are,

Stream Oriented or Standard data files.

System Oriented or low level data files.

There are numerous library fun's available for IIO. These are categeorized into three:

a. Console IIO fun's: fun's to recieve i/p frm Keyboard & write o/p to VDU

b. Disk IIO fun's:- fun's to peform IIO operations on a floppy disk & hard disk.

c. port IIO fun's:- fun's to peform IIO operations on various ports.

Console Oriented apps always use the terminal (keyboard & screen) as the target place. This works fine as long as the data is small. However, many real-life prblms involve large volumes of data & such situations.

The Console oriented IIO operations pose two major prblms.

1. It becomes cumbersome & time consuming to handle large volumes of data through terminals.
2. The Entire data is lost when either the prgm is terminated or the computer turned off.

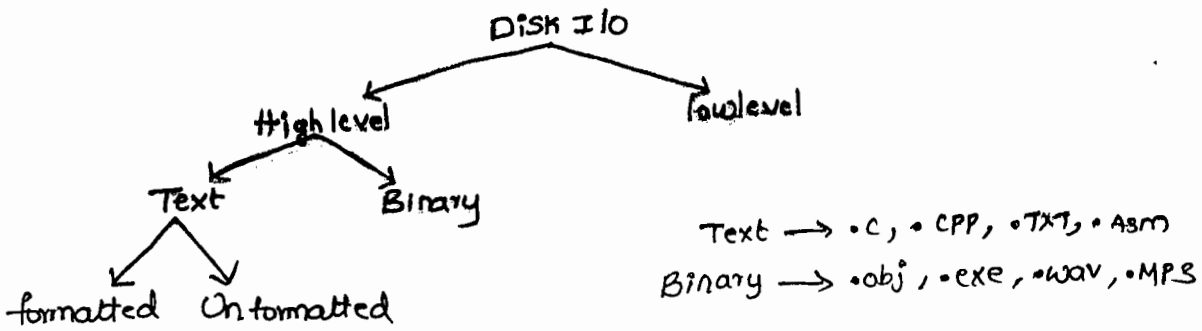
It is therefore necessary to have a more flexible approach where data can stored on disks & read whenever necessary, without destroying the data. This method employs the concept of files to store data.

Definition of file:-

A File is a place on the disk where a group of related data is stored

A File is a collection of related data stored in a particular area on the disk.

Stream Oriented data files are nothing but high level IIO files.



All kinds of Multimedia files are Binary section.

The Stream Oriented datafiles are Separated into Text files & Binary files.

Text files are devided into formatted & Unformatted.

Stream Oriented datafiles are sub divided into 2 categeories.

Text files consisting of consecutive char's, These char's can be interpretened Either by the particular library fun's used to transfer the info. or by format Specifications within the library fun's.

These char's can be interpretened as individual dataitems or Components or strings or numbers.

Unformatted datafiles ⟶ it organizes data into blocks containing conditions bytes of info. The blocks represent more complex data structures such as an array & structures. To process stream Oriented datafiles a separate set of library fun's are used.

The System Oriented data files more closely related to the computer OS. They are much complicated to work upon but more efficient for certain appls.

Separate sets of procedures with accompanying library fun's are required.

Opening of a file:-

when we want to perform any operations on a files we must perform of opening of a file.

The first step in a Stream Oriented databiles is to Establish the buffer area the buffers of buffer area is the temporary storage which is used while the info. is being transferred blw the computer memory & databiles.

To Establish the buffer area we have to use a keyword of "FILE".

It is a built in the structure which includes the val's which can reveal the attributes of the file. It is used to represent the file pointer objects.

file cntrl structures for stream.

```
Typedef struct {
    short       level;
    unsigned    flags;
    char        fd;
    unsigned char hold;
    short       bsize;
    unsigned char *buffer, *curp;
    unsigned    istemp;
    short       token;
} FILE;
```

Syntax:- FILE *file-pointer;

Eg: FILE *fp;

file-pointer = fopen (filename, filetype);

The file type indicates what type of operations we are performing on a file, Basically They are called as modes of the file (attributes).

"r" searches file. If the file exists loads it into memory & sets up a pointer which points to the 1st char in it, if file doesn't exist it returns null. Operations possible reading from a file.

"w" searches file. If the file exists its contents are overwritten. If file doesn't exists new file is created. Returns null if unable to open a file. Operations possible writing to a file.

"a" searches file. If the file exists loads it into memory & sets up a ptr to point the first char. If file doesn't exists new file is created. Returns null if unable to open a file. Operations possible Appending new contents at the end of file.

"r+" searches file. If it exists, loads it into memory setup a ptr which points to the first char. init. If does not exist it returns NULL. Operations possible reading, writing, modifying existing contents.

and w+, a+, ab+ .... etc.

File Operations:- <stdio.h>

functions:- fclose() fgets() fread() fwrite() fEOF() fopen() fscanf()

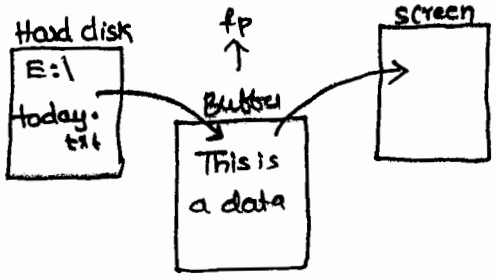
fremove() fgetc() fprintf() fseek() frename() fgetchar() fputc() ftell()

constant Datatypes & Global variables.

1. EOF 2. SEEK_CUR 3. FILE 4. SEEK_END 5. NULL 6. SEEK_SET

* WAP to read the contents from the file & display on the Screen.

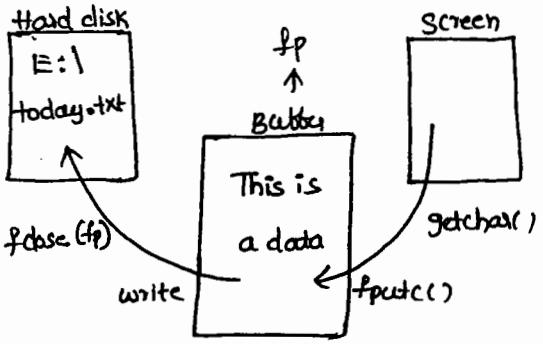
```
<stdio.h> <conio.h>
Main()
{
FILE *fp;
char fname[67], ch;
puts("Enter the file name");
gets(fname);
fp= fopen(fname,"r");
if (fp==NULL)
```



```
{
pf("file Not found");
getch();
Exit(0);
}
pf("\n File is present & the contents are----\n");
while(1)
{
ch= fgetc(fp)
if (ch== EOF)
break;
putchar(ch);
}
pf("\n File Open successfully");
fclose(fp); //save as
getch();
}
```


o/p: File is pr. & the contents are
This is a data
File open successfully.

Creating a file:-




```
Main()
{
FILE *SP;
char fname[67], ch;
puts("Enter the file name");
gets(fname);
SP= fopen(fname, "w");
if (sp==NULL)
{
pf("File not Created");
getch();
Exit(0);
}
pf("\n Enter the contents onto the file \until press F6----\n");
while(1)
{
ch= getchar();
if (ch== EOF)
break;
fputc (ch,sp);
}
pf("\n File Created successfully");
fclose(sp); //save as
getch();
}
```

* To add some more contents onto the file use Append mode.

```
SP = fopen (fname, "a");
```

The append mode will add the content at End of the file.

* fputc() It redirects a single char. to the standard file.

```
fputc (char, file*);
```

* fgetc() reads a single char. into a target var. from a specified file.

```
<char> = fgetc (file *);
```

* The fputc() & fgetc() makes you to move from one char to another position.

* WAP to delete a file?

To delete a file we unlink (fname);

```
Main()
{
FILE *fp;
char fname[20];
clrscr();
puts(" Enter file name to delete");
gets (fname);
fp = fopen (fname, "r");
if (fp == NULL)
{
puts ("File Not present");
getch();
Exit();
}
fclose (fp);
unlink (fname);
fp = fopen (fname, "r"); // for confirmation of
if (fp == NULL)                file deleting r not.
puts ("\n File Deleted successfully");
fclose (fp);
getch();
}.
```


* WAP read a file from the file display no. of chars, alphabets, digits, Tabs --- etc.

```
<stdio.h> <process.h>
void main()
{
FILE *fp;
char fname [20], ch;
int nc=0, ns=0, nt=0, nd=0;
int nl=1, nw=1, flag=0;
clrscr();
puts(" Enter the file name");
gets (fname);
```

```
fp = fopen (fname, "r");
if (fp == NULL)
{
pf ("file not created");
getch();
exit(1);
}
while()
{
ch = fgetc (fp);
if (feof (fp)) // if (ch == EOF)
break;
nc++;
if (ch>='A' && ch<='Z' || ch>='a' && ch<='z')
nal++;
if (ch==' ')
ns++;
if (ch=='\t')
nt++;
if (ch=='\n')
nl++;
if (ch==' ' || ch=='\t' || ch=='\n')
flag=1;
if (flag==1 && ch!='\t' && ch!='\n' && ch!=' ')
{
nw++;
flag=0;
}
}
pf ("\n No. of chars = %d", nc); // 14
pf ("\n No. of nal = %d", nal); // 11
pf ("\n No. of Spaces = %d", ns); // 5
pf ("\n no. of Tabs = %d", nt); // 0
pf ("\n no. of lines = %d", nl); // 1
fclose (fp); pf ("\n No. of words = %d", nw); // 4
getch();
}
```


o/p: Enter the file name
newfile.txt

* fprintf() and fscanf():-

The general format is: int fprintf (fp, format,s)

FILE *fp;
char *format;

The call fprintf() places o/p on the named o/p to which the file ptr fp points, s represents the arg's whose values are printed. format is the format specifier string. The format conversions of printf() work exactly same with fprintf().

fscanf():-

It is used to redirect or accept the data frm the standard file into given variables.

fscanf(file*, "<identifiers>", &variable);

Eg. on fprintf():-

```
<stdio.h> <conio.h>
Void Main()
{
FILE *fp;
int eno;
char ename[20];
float sal;
clrscr();
fp= fopen("e:\\ Emp.dat","a+b");
pf("Enter Enos");
sf("%d", &eno);
pf("Enter Ename:");
fflush(stdin);
gets(ename);
pf("Enter salary:");
sf("%f", &sal);
fpf(fp, "%d %s %0.2f", eno,ename,sal);
pf(" 1 Record saved");
fclose(fp);
getch();
}
```


* Read the data frm the file using fscanf():-

```
<stdio.h> <conio.h>
void main()
{
FILE *fp;
int eno;
char ename[20];
float sal;
clrscr();
fp= fopen("e:\\ Emp.dat", "r");
fscanf(fp, "%d %s %f", &eno, ename, &sal);
pf("\n Eno: %d", eno);
pf("\n Ename: %s", ename);
pf("\n Salary: %0.2f", sal);
fclose(fp);
getch();
}
```

Random Access:-

A file may be accessed seauentially or randomly. In a seauential access, all the preceding data is accessed before accessing a specific portion of a file.

Random access permits direct access to a specific portion of a file.

fseek(), ftell(), & rewind() are the fun's used in random access of a file.

fseek():-

flag = fseek(file-ptr, offset, from-where);

int flag is the value returned by the fun. fseek() if successful & 1 if unsuccessful .....

fseek() sets the position of the "next i/p or o/p operation" in the file to which the file ptr fp points too. The new position is at the signed distance offset bytes. from the begining, from the crnt position or from the End of the file depending upon the value of the ptr. name.

fseek (fp, 7L, 0);

position the file associated with fp to the eight character of the file (remember the 1st char. is at position 0).

(0) SEEK-SET → begining position
(1) SEEK-COR → crnt "
(2) SEEK-END → Ending "

If the 3rd parameter is 1 the 2nd parameter specifies an int or int to the crnt position of the FILE.

→ fseek (fp, 3L, 1); SKIPS ahead 3 char's in the file whereas moves back 3 chars.

→ fseek (fp, -3L, 1);

*. In emp table we need to retrieve the 5th record for this one we use fseek() fun.

```
Struct Emp
{
int eno;
char ename[20];   } 26 bytes.
float sal;
};
```

fseek(fp, (id-1)*sizeof(emp), SEEK-SET);

id = 5th record
fp = 1st record

0-25 → 1st record
26-51 → 2nd
52-77 → 3rd
78-103 → 4th
104-128 → 5th

If in emp. table there are n records, then the last record is,

fseek (fp, -sizeof(EMP), SEEK_END);

* WAP accept 2 chars replace the first char with 2nd char. from the file().

```
Main()
{
FILE *fp;
char fname[20], ch1, ch2, ch;
clrscr();
puts("Enter the file name");
gets(fname);
pf("Enter the first char:");
fflush(stdin);
ch1= getchar();
pf("Enter the 2nd char:");
fflush(stdin);
ch2 =getchar();
fp= fopen(fname, "r+");
if (fp== NULL)
{
pf("FILE Not present");
getch();
exit(0);
}
while(1)
{
ch= fgete(fp);
if (ch==EOF)
break;
if (ch==ch1)
{
fseek (fp, -1, SEEK-CUR);
fpf("%c", ch2);
fseek (fp, 0, SEEK-CUR);
}
}
fclose (fp);
getch();
}
```


* WAP accept a file replace the file containing of lower to upper & viceversa.

```
Main()
{
FILE *fp;
char fname[20], ch;
clrscr();
puts("Enter the file name");
gets(fname);
```

```
pfc("enter the char");
fflush(stdin);
if (ch>=65&&ch<=90)
{
fseek (fp,-1, SEEK-CUR);
fpf (fp,"%c", ch+32);
fseek (fp,0, SEEK-CUR);
}
else
if (ch >=97 && ch<=122)
{
fseek (fp, -1, SEEK-CUR);
fpfC fp, "%c", ch-32);
fseek (fp, 0, SEEK-CUR);
}
fclose (fp);
getch();
}
```


drawbacks of fprintf() & fscanf() :-

```
struct Emp
{
int eno;
char ename[20];
float sal;
};
Emp e;
fpf (fp,"%d %s %d", e.id, e.ename, e.sal);
```

we can write this but it is not recommandable.

In this situation if we want to make any modification then we need to create a temporary file & copy this one & make changes & once again copied into this one.

fwrite() :-

It is used to redirect the cmplt object of a structure into the binary file.

fwrite (< &object>, size, 1, file *);

fwrite (&e, sizeof(e), 1, fp);

1-indicate no.of records &e—source fp—target

To achieving a uniqueness of memory allocation then we go for fwrite() fun.

fread() :-

It is used to read the data frm the binary file into a structure object with a known size.

fread (< &object>, size, 1, file *);

fread (&e, sizeof (e), 1, fp);

It is used to read the Entire record at a time.

Eg- On fwrite() & fread():-

```
struct emp
{
int eno;
char ename[20];
float sal;
};
void main()
FILE *fp;
struct emp;
char ch;
clrscr();
fp=fopen ("c:\\Emp1.dat","a+b");
do
{
pf("enter no:");
sf("%d", &e.eno);
pf ("enter ename");
fflush(stdin);
gets (e.ename);
pf ("Enter salary:");
sf ("%f", &e.sal);
fwrite (&e, sizeof(e), 1,fp);
pf ("\nrecord saved successfully");
pf ("\n Enter another record (y/n):");
fflush (stdin);
sf("%c", &ch);
}
while (ch == 'y');
fseek (fp,0,0); // Reading & display
pf ("%n -10s%n -15s %s", "E.No", "E.NAME", "SALARY");
pf ("\n ----- ");
fread (&e, sizeof(e), 1,fp);
while (!feof (fp))
{
pf ("\n %n -10d %n -15s %n .2f", e.eno, e.ename, e.sal);
fread (&e, sizeof (e), 1,fp);
}
fclose (fp);
getch();
}.
```

Command line arguments:-

passing some arg's to the main & developing the cmds is nothing but Cmd line arg's.

It is a parameter supplied to a prgm when the prgm is invoked These are the arg's passed to the main at cmd prompt.

```
int main (int argc, char *argv[])
                    |      |
                    |      └→ double dimension arg. vector.
                    └→ string.
```

Main() fun. in C takes 2 arg's called argc, argv.

The info. contained in the cmd arg's, when main is called up by the stream.

argc: The var. argc is an arg. counter that counts the no. of arg's on the cmd line.

argv: The argv is an arg. vector & rep's an array of char ptrs that points to the cmd line arg's. The size of array will be equal to argc.

* WAP implement a user defined cmnd by passing some arg's to main of a Calculator?

```
int main (int argc, char *argv[])          //converting exe. file into cmd.
{
int n1,n2;
char op;
if (argc <4)
{
pf ("In syntax of cmd is incorrect");
Exit(0);
}
op=argv[2][0];
n1= atoi(argv[1]);
n2= atoi(argv[3]);
switch(op)
{
case '+': pf ("In %d", n1+n2); break;
case '-': pf ("In %d", n1-n2); break;
case '*': pf ("In %d", n1*n2); break;
case '/': pf ("In %d", n1/n2); break;
case '%': pf ("In %d", n1%n2); break;
default: pf ("In Invalid operator");
}
}
```

dp:- E:\ >kalc 5+7

12

Cmd prompt can be executed frm the DOS also self Run. arg's.

argv[0]: "...\\...\\KALC.EXE".

argv[1]: "5".

argv[2]: "+".

argv[3]: "7".

Save the file with any name for suppose, Kalc.c & generate an Executable file. Then go for the DOS prompt. Then run the cmd

*- How to do program in LINUX (VI Editor)

Application ⟶ prgmng ⟶ Vi Editor.

(or)

Application ⟶ System tools ⟶ Terminal.

1. first go to Kernal of OS
   Start ⟶ Terminal.
2. After opening Terminal (cmd prompt) U see cmd prompt $
3. Type $ vi filename.c.
4. Then Opens Vi Editor (notepad)
5. press "Insert" Button & 'i' and Type i.e; write your code.
6. finally press ESC+shift : wq (save and quit)
7. Then come out from vi Editor
8. Again u see $ (prompt)
9. Type $ gcc filename.c (compile) (Here, u see errors if any in prgmng)

Again Type $ ./a.out (RUN)
you see the Result of the prgm.

KIRAN SIR
NOW WITH
Santosh Technologies
Cell: 9246392345

Project in Technical Book

Please follow the website for present Company Questions

http://KiranSrinivas.wordpress.com

For Technical Book Seperate Book is available.

for doubts
mail to Vkiransrinivas@yahoo.co.in

KIRAN SIR
NOW WITH
Santosh Technologies
Cell: 9246392345