BASIC LINUX COMMANDS

Commonly Used Linux Commands	
Command	**Description**
man	display details about an instruction
ls	list contents of the current directory
ls -l	detailed listing of directory contents, shows permissions, owner, etc.
ls -a	list all files (including hidden files)
ls -la	detailed listing of all files (note that options can be combined)
cd	change directory
cd ../	backup one level from the current directory
pwd	print current working directory
touch	create an empty file
mkdir	make a new directory
rm	remove files and directories
rm -rf	recursively remove all files and directories under the specified directory
cat	list file contents
less	list file contents – one screen at a time
tail	list the end of the file (default – displays last 10 lines)
tail -n	list the last n lines of a file
cp	copy file
mv	rename a file
echo	echo values to the screen example: echo $PATH – prints the value of the PATH variable
grep	command line text search utiltity example: grep blue colors.txt – list all lines with the word blue from the colorlist.txt file
ps	list currently running processes

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

PREPROCESSOR DIRECTIVES

#define is used to declare a macro (fragment of code).

#include is used to include a file.

#ifdef , #endif , #if , #else , #ifndef (conditions before compilation)

#undef (to undefine a defined macro) , #pragma (to call a function before and after main())

[#define circleArea(r) (3.1415*(r)*(r))]

[#define pi 3.14]

HEADER FILES

#include<stdio.h> - standard input output header

#include<string.h> - string header

#include<conio.h> - console input output header

#include<math.h> - math header

.

.

.

(.h extension is necessary in C)

ERRORS IN C

- Syntax Errors (eg-missing (),{},; , printing variable without declaring , etc...)
- Run-time Errors (eg- divsion by zero , etc...)
- Linker Errors (eg- writing Main() instead of main() , etc...)
- Logical Errors (eg- Desired outputs are not obtained , etc...)
- Semantic Errors (eg- Statements are not meaningful to the compiler (void main{ int a,b,c; a+b=c; }) , etc...)

Data Types in C

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:

Following are the examples of some very common data types used in C:

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer.
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision. (6 decimal places precision)
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision. (15 decimal digits precision) (If we add 0.16f to the value, the 16th decimal won't be precise)

Different data types also have different ranges upto which they can store numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format specifiers on 32 bit gcc compiler.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	8	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	8	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%Lf

```
long double – 19 decimal digits precision
```

SIGNED AND UNSIGNED

Refer the program:

```
{
      unsigned int i=-1;
      printf("%u",i);
}
```

Output - 4294967295

MANAGING PRECISION

"%0.2f", 0.123 -> 0.12 (zero padded min. width of 0, 2 decimal places).

"%6.2f", 0.123 -> __0.12 (space padded min. width of 6, 2 decimal places).

"%06.2f", 0.123 -> 000.12 (zero padded min. width of 6, 2 decimal places).

"%0.6f", 0.123 -> 0.123000 (min width of 0, 6 decimal places).

GENERALLY, First number gives the padding element, space or zero (default space), Second number(can be two or more digits also) gives the min width (default 0) and the one after decimal gives the no. of decimals to be included.

To add 4 decimals to a value use %0.4f or %.4f

Eg:-
float num=5.46718722;
printf(“%0.4f”,num); //%0.4lf if data type is double and %0.4Lf for long double

Output – 5.4672 (rounding off)

If we add f to a number, it’ll be converted to float for that operation.
(Eg- int a=2.47372364f+2.3287466f; means the values will be converted to float, then added and then converted back to int) [Not for lf or Lf]

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*
1	1	1	1	*[START OF HEADING]*
2	2	10	2	*[START OF TEXT]*
3	3	11	3	*[END OF TEXT]*
4	4	100	4	*[END OF TRANSMISSION]*
5	5	101	5	*[ENQUIRY]*
6	6	110	6	*[ACKNOWLEDGE]*
7	7	111	7	*[BELL]*
8	8	1000	10	*[BACKSPACE]*
9	9	1001	11	*[HORIZONTAL TAB]*
10	A	1010	12	*[LINE FEED]*
11	B	1011	13	*[VERTICAL TAB]*
12	C	1100	14	*[FORM FEED]*
13	D	1101	15	*[CARRIAGE RETURN]*
14	E	1110	16	*[SHIFT OUT]*
15	F	1111	17	*[SHIFT IN]*
16	10	10000	20	*[DATA LINK ESCAPE]*
17	11	10001	21	*[DEVICE CONTROL 1]*
18	12	10010	22	*[DEVICE CONTROL 2]*
19	13	10011	23	*[DEVICE CONTROL 3]*
20	14	10100	24	*[DEVICE CONTROL 4]*
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*
22	16	10110	26	*[SYNCHRONOUS IDLE]*
23	17	10111	27	*[ENG OF TRANS. BLOCK]*
24	18	11000	30	*[CANCEL]*
25	19	11001	31	*[END OF MEDIUM]*
26	1A	11010	32	*[SUBSTITUTE]*
27	1B	11011	33	*[ESCAPE]*
28	1C	11100	34	*[FILE SEPARATOR]*
29	1D	11101	35	*[GROUP SEPARATOR]*
30	1E	11110	36	*[RECORD SEPARATOR]*
31	1F	11111	37	*[UNIT SEPARATOR]*
32	20	100000	40	*[SPACE]*
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	*[DEL]*

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	|	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	▉	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	ϕ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

PRECEDENCE AND ASSOCIATIVITY

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

b=++a; (incremented value is assigned)

b=a++; (value is assigned and then incremented)

a<<1; (Left shift by 1 but no change to a)

a >>= 2; (Right shift by 2 and assignment to a)

TYPE CONVERSION

- **IMPLICIT TYPE CONVERSION (done by compiler) :-**

If a and b are int, res=a/b will also be int(irrespective of data type of res).

EG:-

```
int a=4.7,b=5.7;
float res=a/b;
```

Output :- res=0.000000

If either a or b or both is float, then a/b will be float and res=a/b will depend on the data type of res.

EG:-

```
int a=4.7;
float b=5.7,res=a/b;
```

Output :- res=0.701754

```
float a=12.3,b=3.9;
int res=a/b;
```

Output :- res=3

- **EXPLICIT TYPE CONVERSION (done by programmer) :-**

```
struct person per1={‘a’,‘b’,‘c’};
struct person *p;
*p=&per1;
printf(“%d\t”, p); //memory location of structure pointer
printf(“%c %c %c”, *((char*)p), *((char*)p+1), *((char*)p+2)); //type conversion
printf(“\t%d”, p); //type conversion occurs only inside the print statement
```

Output :- 0x7ffd196da70d a b c 0x7ffd196da70d

VARIABLES (IDENTIFIERS)

- Unique
- Length is compiler dependent
- Letters, digits and underscore
- No spaces
- Upper and lowercase letters are distinguished
- First character must be a letter or underscore
- Cannot be a keyword (if,else,while,for,etc...)
- main can be used as a variable name

LITERALS

Values assigned to the variables. Can be changed if needed.

- Integers (Decimal, Octal(starts with 0) and Hexadecimal(starts with 0x))
- Floating-point literals (0.02, -2.0, 2.5E-3)
- Characters (‘a’, ‘F’, ‘2’, ‘{’)
- Escape sequences (\b, \f, \n\, \r, \t, \v, \\, \’, \”, \?, \0)
- String literals (“good”, “”, “ “, “x”, “Earth is round\n”)

CONSTANTS

Variable whose value cannot be changed. Keyword – const or using #define.

```
const float pi=3.14;
pi=4; //ERROR

#define three 3
...
result=sqrt(three);
```

ARRAYS

- Identifier points to the first element of array (printf(“%d”,*arr); gives the element in arr[0]).
- Array CANNOT be initialised as int arr[]; CAN be either, int arr[size]; or, int arr[]={1,2,3}; or, int arr[3]={1,2,3};
- Unfilled spaces are filled with 0 and undefined spaces are filled with garbage values.
- Character storing arrays are called strings in C.

STRINGS

- Properties similar to arrays. (Identifier points to the first character)
- Initialised as, char arr[]= “Hello World”; or, char arr[6]=”Hello”; or, char arr[6]={‘H’,‘e’,‘l’,‘l’,‘o’,‘\0’}; or, char arr[]={‘H’,‘e’,‘l’,‘l’,‘o’,‘\0’}; or, char arr[size];

- Strings can also be initialised using pointers, char *str = “Hello World”; but there is difference between *str (a pointer and so, a variable) and str[] (An array and so, not a variable).
- Terminated by \0 (NULL).
- Declaring by char arr[] gives garbage values for undefined %c but no problem for %s. (MAY BE COMPILER DEPENDENT)
- If string is filled character-wise, \0 must be included at the end. Otherwise, garbage values will be stored in %s also.
- Can be inputted by scanf, gets(), fgets(), getline() [similar to fgets but more reliable. Difference is that it uses address instead of identifier – getline(&str, &size, stdin);] or getchar(). (scanf %s will only input a single word nothing after a space is included in it so use “%[^\n]%*c”) (gets reads until a new line.) (getchar reads the same as reading an array) (fgets reads until a new line or end of file (fgets(str, 20, stdin);) (gets may show warning, ignore it)
- fgets(str); //strlen(str) gives length including a newline at the end.
- Can be outputted by printf, puts() or putchar() (puts(str) outputs the string with a new line at the end).
    - printf() can also be carried out as follows :

```
char  a='a', *str= “Hello World %c %d”, *strr=”Hello World”;
int b=1;
printf(str, a, b); //Gives ouput Hello World a 1
printf(strr, a, b); //Error, format specifiers required for
                        non-string variables
printf(a,b); //Error, this works only for one string variable
printf(a); //Error, this works only for string variables
```

- String handling fuctions – strlen(), strlwr(), strupr(), strcpy(), strcat(), strcmp()
- strcmp() after checking both strings (int result=strcmp(str1,str2);) until characters are unmatched or \0 is encountered in any, gives 3 outputs
    1. 0 – If all characters inlcuding \0 are equal.
    2. >0 – If ASCII value of first string character is higher.
    3. <0 – If ASCII value of first string character is lower.
- strlwr() - Lowercase all characters.
- strupr() - Uppercase all characters.
- strlen() - Length of string excluding \0. (printf(“%zu”,(strlen(str)) or store it in an int l=strlen(str);)
- strcpy() - Copies contents of second string to first. (puts(str1); gives the copied string)
- strcat() - Concatenates second string to the first string and result is stored in first.
- 2D strings are also there. (char str[20][20];) (char str[]={“Hello”,”World”};)

```
for(i=0;i<20;i++)
{
	for(j=0;j<20;j++)
		scanf(“%s”,str[j]); //Only the row no. needs to be specified
}
```

STRUCTURE & UNION

- Definition should end with “;”.
- struct – all values can be retrived at once, size is sum of all variables.
- union – only one value (finally intialised value) can be retrieved at a time, size is that of largest variable.
- Variables can be accessed using “.”.

FUNCTIONS

- Data type is of return value. (default - int)
- No return value means void.
- If return value; is included, program control is suddenly transferred out of the function to wherever it was called.
- Definition can be written anywhere but declaration (void func(); is enough. No need to mention the arguments during declaration) must be written before calling a function.
- Array can be passed as a parameter by passing the identifier name alone (reverse(arr)). In the function declaration, data type of the array and parameter name must be specified but array size need not be specified (void reverse(char arr[])).
- While passing 2D array, max no. of elements in each row has to be specified (ie, no. of columns has to be specified) (void transpose(int a[][30])).
    - Read(a,b);
    - if(Read(a,b)==0)
        - .....

    Here, the function is executed twice.

INFINITE LOOPS

- while(1)

- for(;;)
- while(0) doesn't enter the loop

STORAGE CLASSES

GLOBAL or EXTERNAL VARIABLES

- Variables that are alive throughout a program and can be used anywhere (in any function)
- Either defined outside of all functions (#include<stdio.h> float pi=3.14)
- Or we can also define it inside any function using extern keyword (void main() { extern float pi=3.14; })
- Used when we need a variable with the same value for all functions
- Constants (const) can also be made globally accessible
- Eg:-

```
#include<stdio.h>
int Gvar=0;
void main()
{
        int Gvar=5;
}
```

Here, the value of Gvar changes from 0 to 5 inside the function but if it's called by another function it's value will be 0 but to retain the value from one function to another simple lose the datatype. (Refer below)

```
#include<stdio.h>
float pi=3.1415;
void func()
{
        printf("%f ",pi);
        pi=3.14; //refer below
}
void main()
{
        printf("%f ",pi);
        pi=3; //refer below
        printf("%f ",pi);
        func();
        printf("%f\n",pi);

}
```

Output – 3.141500 3.000000 3.000000 3.140000
If float pi=3.14; and float pi=3; output will be 3.141500 3.000000 3.141500 3.000000

AUTOMATIC AND STATIC VARIABLES

- Both **auto** and **static** variables are local variables (available only in their respective functions).
- **static** variables can retain the value of the variable between different function calls.
- But, scope of **auto** variable is within the function only. It can't retain the value of the variable between different function calls.
- **auto** is the default variable type in C. (So **auto** keyword is optional)
- **auto** is filled of garbage values by default while **static** is filled with 0.
- **auto** is initialised everytime when the function (in which it's declared) is called and is destroyed after program control leaves the function.
- **static** is initialised once and is destroyed only after the program execution.
- Eg:-

```
#include <stdio.h>
void fun(void)
{
    auto int a=0;
    static int b=0;
    printf("a=%d,b=%d\n",a,b);
    a++;
    b++;
}

void main()
{
    int loop;
    for(loop=0; loop<5; loop++)
        fun();
}
```

Output

```
a = 0, b = 0
a = 0, b = 1
a = 0, b = 2
a = 0, b = 3
a = 0, b = 4
```

REGISTER VARIABLES

- Registers are faster than memory to access, so the variables which are most frequently used in a C program can be put in registers using **register** keyword.
- Properties similar to **auto** but has faster access.
- It can also store pointers like **auto**.

COMPARISON -

- **extern** and **static** has lifetime until the program ends, while **auto** and **register** has until the function ends.
- **extern** has scope througout the program (global scope), while **auto**, **static** and **register** has scope in the function only (local scope). (**static** also has scope between function calls)
- **extern** is declared outside or inside a function, while **auto**, **static** and **register** are declared inside a function alone.
- **extern** has visibility throughout the program, while **auto**, **static** and **register** has visibility throughout the function alone. (SCOPE AND VISIBILITY ARE ALMOST THE SAME)

POINTERS

- char *ptr; and char* ptr; is the same. (Data type is not of address but of the variable)
- &(ampersand) is used to assign and *(asterisk) is used to access.
- *ptr gives the value of variable while ptr gives its address.
- scanf(“%d”,*ptr); gives segmentation fault. (Same for int *ptr = 1;)
- Scanf(“%d”,&a); a=*ptr; is the right way.
- %p is used to print pointer (address)
- Passing to a function is carried as, void swap(int *ap,int *bp); and swap(&a,&b). Since the memory address is passed the variable is changed in both functions. [Known as call by reference]
- NULL pointers *ptr=NULL; here, value of ptr is 0. (Actually, 0 is taken by OS but by convention, pointers that have value 0 are meant to point to nothing)
- if(ptr) proceeds if *ptr is not NULL. Similarly if(!ptr) proceeds if *ptr is NULL.
- Operations that can be performed are ++,--,+ and -.
- ptr++ and ptr-- points to the next or previous location (supposedly in an array). (char pointer moves from 1000 to 1001 while int pointer moves from 1000 to 1004)
- Pointers can be compared (ptr<=&a[i]).
- Pointer to pointer (int **p) accessed by **.
- func(int *ptr) is the declaration while passing an pointer to a function. (This pointer can point to the various elements of an array using ptr++[Note that ptr is not the identifier of the array])

- arr points to the first element, so func(arr); can be used to pass the array pointer that points to first element but this pointer cannot be incremented or decremented (error is lvalue required, maybe it can be incremented or decremented too in printf (“%d”, *(ptr+i));) (but string pointer can be incremented or decremented).
- int *ptr[size]; is used to declare array using pointer.
- **ptr is equivalent to *ptr[0] and arr[0]. We cannot use ptr++ here since it is used to point to the array’s first element (It should not be modified). (This is not for string pointer given below)
    - char *ptr = “Hello World”; can be used to declare string using pointer. * is not required to access the string (printf(“%s %p %c”, ptr, ptr,*ptr);)
    - ptr++ gives ello World.
- To create an list of strings using pointer:-

```
char *names[] = {
"Zara Ali",
"Hina Ali",
"Nuha Ali",
"Sara Ali"
};
```

- Here,

Value of names[0] = Zara Ali (When stdin(gets or fgets (exclude scanf %s)) to 2D array, the first string sometimes becomes “\n”. Use scanf(“\n”); before stdin as a solution to this)

Value of names[1] = Hina Ali

Value of names[2] = Nuha Ali

Value of names[3] = Sara Ali

- {“%s”, names[i]} gives the entire string at i^{th} row.
- {“%s”,*names} gives the first string.
- You cannot scanf() these strings using %s because it is similar to reading a pointer directly using scanf(). (Segmentation fault)
- Instead allocate memory for it using malloc() OR store the string in a normal char array and then copy it to the string using strdup().

CALL BY VALUE

```
#include <stdio.h>
void swapx(int x, int y);
int main()
{
int a = 10, b = 20;
swapx(a, b);
```

```
printf("a=%d b=%d\n", a, b);
return 0;
}
void swapx(int x, int y)
{
int t;
t = x;
x = y;
y = t;
printf("x=%d y=%d\n", x, y);
}
```

Output :

x=20 y=10

a=10 b=20

CALL BY REFERENCE

```
void swapx(int*, int*);
int main()
{
int a = 10, b = 20;
swapx(&a, &b);
printf("a=%d b=%d\n", a, b);
return 0;
}
void swapx(int* x, int* y)
{
int t;
t = *x;
*x = *y;
*y = t;
printf("x=%d y=%d\n", *x, *y);
}
```

Output :

x=20 y=10

a=20 b=10

DYNAMIC MEMORY ALLOCATION

When array size has to be increased or decreased. Declared using pointers. Header <stdlib.h> has to be used.

- malloc() - memory allocation, A single large block of specified size is initialised with garbage values by default (can also be considered as non-initialised). { ptr = (cast-type*) malloc(byte-size) }
    - ptr = (char*) malloc(5*sizeof(char));
- calloc() - contiguous allocation, individual blocks of specified size are intialised with default 0 values. { ptr = (cast-type*) calloc(n, element-size); }
    - ptr = (int*) calloc(5, sizeof(int));
- free() - frees allocated memory. { free(ptr); }
    - free(ptr);
- realloc() - Re-allocates allocated memory and new blocks are initialised with garbage values. { ptr = realloc(ptr, newSize); }
    - ptr = realloc(ptr, 10*sizeof(int));

TO REFER !

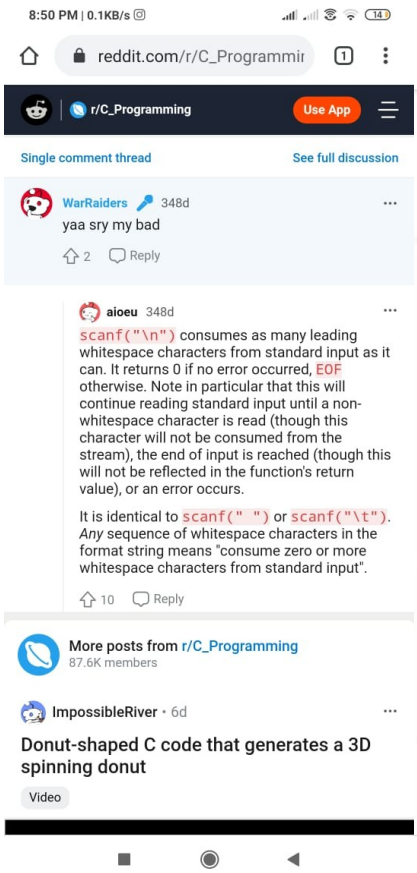
- Isalpha(), abs(), atoi(), etc...
- Large size handling
- Diff for input-output and processes
- GCC can’t store very large values other than input or output.
- %1000000007
- ((m%n)^2)%n = (m^2)%n (Check validity)
- The reason of taking Mod is to prevent integer overflows. The largest integer data type in C/ C++ is unsigned long long int which is of 64 bit and can handle integer from 0 to (2^64 – 1). But in some problems where the growth rate of output is very high, this high range of unsigned long long may be insufficient.
- __int64_t
- typedef long long int int64_t
    - from library stdint

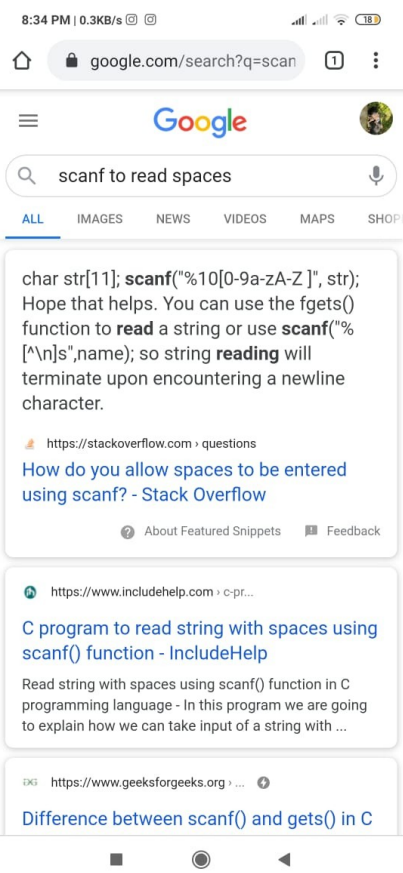
```
typedef signed char             int8_t;
typedef short int               int16_t;
typedef int                     int32_t;
# if __WORDSIZE == 64
typedef long int              int64_t;
# else
_extension_
typedef long long int           int64_t;
# endif
#endif

/* Unsigned.  */
typedef unsigned char           uint8_t;
typedef unsigned short int      uint16_t;
```

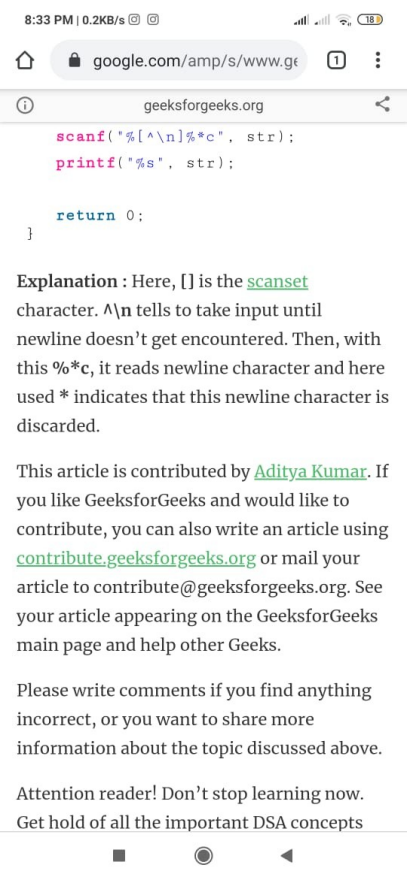
```
#ifndef __uint32_t_defined
typedef unsigned int            uint32_t;
# define __uint32_t_defined
#endif
#if __WORDSIZE == 64
typedef unsigned long int       uint64_t;
#else
_extension_
typedef unsigned long long int  uint64_t;
#endif


/* Small types.  */

/* Signed.  */
typedef signed char             int_least8_t;
typedef short int               int_least16_t;
typedef int                     int_least32_t;
#if __WORDSIZE == 64
typedef long int                int_least64_t;
#else
_extension_
typedef long long int           int_least64_t;
#endif

/* Unsigned.  */
typedef unsigned char           uint_least8_t;
typedef unsigned short int      uint_least16_t;
typedef unsigned int            uint_least32_t;
#if __WORDSIZE == 64
typedef unsigned long int       uint_least64_t;
#else
_extension_
typedef unsigned long long int  uint_least64_t;
#endif


/* Fast types.  */

/* Signed.  */
typedef signed char             int_fast8_t;
#if __WORDSIZE == 64
typedef long int                int_fast16_t;
typedef long int                int_fast32_t;
typedef long int                int_fast64_t;
#else
typedef int                     int_fast16_t;
typedef int                     int_fast32_t;
_extension_
typedef long long int           int_fast64_t;
#endif

/* Unsigned.  */
typedef unsigned char           uint_fast8_t;
#if __WORDSIZE == 64
typedef unsigned long int       uint_fast16_t;
typedef unsigned long int       uint_fast32_t;
typedef unsigned long int       uint_fast64_t;
#else
typedef unsigned int            uint_fast16_t;
typedef unsigned int            uint_fast32_t;
_extension_
typedef unsigned long long int  uint_fast64_t;
#endif


/* Types for `void *' pointers.  */
#if __WORDSIZE == 64
# ifndef __intptr_t_defined
typedef long int                intptr_t;
#  define __intptr_t_defined
# endif
typedef unsigned long int       uintptr_t;
#else
# ifndef __intptr_t_defined
typedef int                     intptr_t;
#  define __intptr_t_defined
# endif
typedef unsigned int            uintptr_t;
```

```
#endif


/* Largest integral types.  */
#if __WORDSIZE == 64
typedef long int                intmax_t;
typedef unsigned long int       uintmax_t;
#else
__extension__
typedef long long int           intmax_t;
__extension__
typedef unsigned long long int  uintmax_t;
```

- https://sites.uclouvain.be/SystInfo/usr/include/stdint.h.html
  - Nice library must read
- Argument passing to function args, argv etc...
- -> operator
- OOP keywords in C (static, abstract, etc...)
- Function pointer
- Abstraction in C
- Vectors in C/CPP







C viva questions

1. **Who developed C language?**
   C language was developed by Dennis Ritchie in 1970 at Bell Laboratories.
2. **Which type of language is C?**
   C is a high – level language and general purpose structured programming language.
3. **What is a compiler?**
   Compile is a software program that transfer program developed in high level language into executable object code
4. **What is IDE?**
   The process of editing, compiling, running and debugging is managed by a single integrated application known as Integrated Development Environment (IDE)
5. **What is a program?**
   A computer program is a collection of the instructions necessary to solve a specific problem.
6. **What is an algorithm?**
   The approach or method that is used to solve the problem is known as algorithm.
7. **What is structure of C program?**
   A C program contains Documentation section, Link section, Definition section, Global declaration section, Main function and other user defined functions
8. **What is a C token and types of C tokens?**
   The smallest individual units are known as C tokens. C has six types of tokens Keywords, Constants, Identifiers, Strings, Operators and Special symbols.
9. **What is a Keyword?**
   Keywords are building blocks for program statements and have fixed meanings and these meanings cannot be changed.
10. **How many Keywords (reserve words) are in C?**
    There are 32 Keywords in C language.
11. **What is an Identifier?**
    Identifiers are user-defined names given to variables, functions and arrays.
12. **What is a Constant and types of constants in C?**
    Constants are fixed values that do not change during the program execution. Types of constants are Numeric Constants (Integer and Real) and Character Constants (Single Character, String Constants).
13. **What are the Back Slash character constants or Escape sequence characters available in C?**
    Back Slash character constant are \t, \n, \0
14. **What is a variable?**
    Variables are user-defined names given to memory locations and are used to store values. A variable may have different values at different times during program execution
15. **What are the Data Types present in C?**
    Primary or Fundamental Data types (int, float, char), Derived Data types(arrays, pointers) and User-Defined data types(structures, unions, enum)
16. **How to declare a variable?**
    The syntax for declaring variable is
    data type variable_name-1, variable_name-2,....variable_name-n;
17. **What is meant by initialization and how we initialize a variable?**
    While declaring a variable assigning value is known as initialization. Variable can be initialized by using assignment operator (=).
18. **What are integer variable, floating-point variable and character variable?**
    A variable which stores integer constants are called integer variable. A variable which stores real values are called floating-point variable. A variable which stores character constants are called character variables.
19. **How many types of operator or there in C?**
    C consist Arithmetic Operators (+, -, *, /,%), Relational Operators (<, <=, >, >=, !=), Logical Operators (&&, ||, !), Assignment Operators (=, +=, -=, *=, /=), Increment and Decrement Operators (++, --), Conditional Operator(?:), Bitwise Operators(<<, >>, ~, &, |, ^) and Special Operators (. , ->, &, *, sizeof)
20. **What is a Unary operator and what are the unary operators present in C?**
    An operator which takes only one operand is called unary operator. C unary operators are

Unary plus (+), Unary minus (-), Increment and Decrement operators (++,--), Address of operator (&), Value at operator (*), sizeof operator, ones complement operator (~).

21. What is a ternary operator and which is called ternary operator is C?

An operator which takes three operands is called ternary operator. Conditional operator (? :) is knows as ternary operator in C.

22. What is the use of modulus (%) operator?

The modulus operator produces the remainder of an integer division. It cannot be used on floating point data.

23. What is the use of printf and scanf functions in C?

Values of variables and results of expressions can be displayed on the screen using printf functions. Values to variables can be accepted through the keyboard using scanf function.

24. What are the format codes used in printf and scanf functions in C?

%c (for reading or printing a single character), %d (for reading or printing signed integer), %u (for reading or printing unsigned integer), %ld (for reading or printing long signed integer), %lu (for reading or printing long unsigned integer), %f (for reading or printing floating point value), %lf (for reading or printing double floating point value), %Lf (for reading or printing long double value, %s (for reading or printing string value)

25. What are the decision making statements available in C?

IF statement, Switch statement and conditional statement

26. What is the use of IF statement and how it works?

The IF statement is used to control the flow of execution of statements. It first evaluates the given expression and then, depending on whether the value of the expression is true or false, it transfers the control to a particular statement.

27. Forms of IF statements?

Simple IF statement, IF-ELSE statement, NESTED IF-ELSE statement and ELSE IF ladder

28. How switch statement works?

The switch statement tests the value of a given variable against a list of case values and when a match is found, block of statement associated with that case is executed and if no match is found then the block of statements associated with the optional default is executed. If default case not present, control goes to the statement after the switch.

29. What is the difference between IF and SWITCH statement?

IF works on integers, floats and characters whereas SWITCH works on only integers and characters. Switch statement cannot perform inequality comparisons whereas IF can perform both equality and inequality comparisons.

30. How conditional operator (? :) works?

The conditional expression is evaluated first. If the expression is true then expression after the question mark is executed otherwise expression after the colon is executed

31. What is GOTO statement?

GOTO is an unconditional branching statement which transfer control to the specified label

32. What is a LOOP?

Loop is a sequence of statements that are executed repeatedly

33. What are the types of LOOP control statement?

Entry-Controlled Loop statement and Exit-Controlled loop statement

34. What are the LOOP control statements present in C?

WHILE, DO-WHILE, FOR

35. What are the sections present in FOR loop?

Initialization section, Conditional section, increment or decrement section

36. How a FOR loop works?

In FOR loop first initialization section is executed, then given condition is checked. If condition becomes true then body of the loop is executed after that increment or decrement section is executed

37. What is the use of break statement?

Break statement is used to exit from a loop

38. What is an ARRAY?

Array is a collective name given to similar elements

39. How ARRAY elements are stored in memory?

Array elements are stored in consecutive memory locations

40. How we can initialize an ARRAY?

All the element are separated by comma and enclosed within braces

41. How to access array element?

Array elements can be accessed by using subscript

42. What is the difference between normal variable and array variable?

A variable can store only one value at a time whereas array variable can store several value at a time.

43. What are the types of Array's?

One-Dimensional array, Two-Dimensional array and Multi-Dimensional array

44. What is a TWO-DIMENSIONAL array?

A Two-Dimensional array is an array which has elements as one-dimensional arrays?

45. What is a character array?

Array which can store several characters is called character array

46. How to initialize a character array?

Character arrays can be initialized by separating character constants with comma and enclosed with in parenthesis or characters enclosed within double quotation marks.

47. What is the difference between reading strings using scanf and gets?

Scanf can not read strings with multiple words whereas gets can read strings with multiple words

48. What are the String-Handling functions available in C?

gets, puts, strcat, strcmp, strcpy and strlen.

49. What are the types of functions?

C functions are divided into two categories user-defined function and built-in functions

50. What is a function?

Function is a self contained block of statement which is used to perform certain task

51. Which are called built-in functions?

Printf, scanf, clrscr, gotoxy, string handling functions and file handling functions

52. What is function prototype declaration?

A function declaration is also known as function prototype declaration which contains function return type, function name, parameter list and terminating semicolon

53. What are formal arguments and actual arguments?

Arguments that are used in function calling are called actual arguments. Arguments that are used in function definition are called formal arguments

54. What is a recursive function?

A function calling itself is called function recursion

55. What is call by value and call by reference?

Passing values to the called function is called call by value, passing addresses to the called function is called call by reference

56. How to pass an array to a function?

Arrays are passed to a function by sending its address

57. What is a global variable and local variable?

Variables which are declared in the global section is called global variables and Variables which are declared in a function or a block are called local variables

58. What is a pointer variable?

Pointer variable is a variable which can store address of another variable

59. How can we store address of a variable in a pointer?

By using address of operator we can store address of a variable in a pointer

60. How can we access a variable value using a pointer?

By using value at operator we can access a variable value using its pointer

61. What is the use of pointers?

Pointer are used to pass array and structures from function to another function

62. How many bytes a pointer variable occupies in memory?

A pointer variable irrespective of its type it occupies two bytes in memory

63. What are the storage classes available in C?

Auto, Static, Extern and Register

64. What is a structure?

Structure is a user-defined data type. Structure is a collective name given to dissimilar elements

65. How to access structure members?

Structure members can be accessed using dot operator

66. How to initialize structure variable?

All the members are separated by comma and are enclosed within braces

67. What are the differences between structures and arrays?

Structures stores dissimilar values where as arrays stores similar values. One structure variable can assigned to another structure variable whereas one array variable cannot be assigned to another array variable

68. What is the size of a structure?

Sum of all the members size is becomes structure size

69. How to access structure member by its pointer?

We can use structure members using arrow operator with its pointer

70. What is a union?

Union is a user-defined data type which can store a value of different data types

71. What is the difference between structures and unions?

Structures can store several values at a time whereas unions can store one value at a time. A structure size becomes sum of all its members whereas a union size becomes size of a member whose size is largest

72. What are the types of files we can create using C?

We can create text and binary files using C

73. What are the file-handling functions present in C?

fopen, fclose, fgetc, fputc, fgets, fputs, fprintf, fscanf, fread, fwrite, fseek

74. What are the file opening modes present in C?

r, w, a, r+, w+, a+, rb, wb, rb+, wb+
