1972. C language invented by Dennis Ritchie. ①

C programming -

Page No:
Date: 14/10/22

① Data types - Represent the type of data & set of operations to be performeed on data.

Syntax - <datatype> <variablename>;

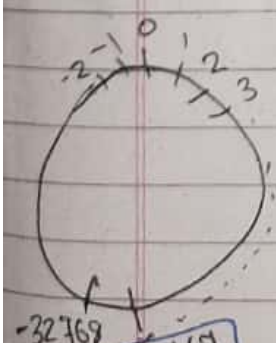
e.g - int a;

Datatype

primitive	Derived	user-defined
Integer. 2 byte	Array	Structure
float 4	String	union
Char. 1	function	Enum.

1* Integer datatypes - only allows int value to store in variable.

	Datatype	Allocation of memory
1)	int	2 byte = 16 bits
2.	Signed int	2 byte = 16 bits
3.	unsigned int	2 byte = 16 bits
4.	long int	4 byte = 32 bits.



	Range
int	-2^{15} to $2^{15}-1$ (-32768 to 32767)
Signed int	-2^{15} to $2^{15}-1$ (-32768 To 32767)
unsigned int	0 to $2^{16}-1$
long int	-2^{31} to $2^{31}-1$

int a = 32768
print(a);

Ans = -32768

Page No.:
Date: / /

2 Char data Type - Allows character values to be stored
In implementation when we require ~~register~~ char operation then go for char or unsigned char datatype.

char 1 byte = 8 bits -2^7 to 2^7-1
-128 to 127

3 Floating Datatype.

	Datatype	Allocation of memory.
1)	Float (%f, %e)	4 byte = 32 bits.
2.	double (%f, %le)	8 byte = 64 bits
3.	long double (%Lf, %le)	10 byte = 80 bits.

Q
```
void main()
{
    float x = 247.57;
    printf ("\n %f", x); // 247.570000
    printf ("\n %e", x); // 2.475700E+02
    getch();
}
```

Page No.:
Date: / /

ASCII Value

American Standard Code for Information Interchange

A - 65	a - 97	0 - 48	\n - 10
B - 66	b - 98	1 - 49	☺ - 1
C - 67	c - 99	2 - 50	♡ - 3
:	:	:	◇ - 4
:	:	:	
:	:	:	
Z - 90	z - 122	9 - 57	

Q
```
void main()
{
  printf("\n %c", 1);   // ☺
  printf("\n%c",3);     // ♡
  printf("\n %c",4);    // ◇
}
```

Q
```
voind main()
{
  char x = '\n';        // 10
  printf("%d", x);
  getch();
}
```

Q
```
void main() {
  char x = 'A';         // 65
  printf("%d", x);
}
```

Page No.
Date. / /

* Scanf & Printf

* printf(): (Print format

→ it is pre-defined output function
→ it is declared under stdio.h.
↑ header file (standard input output)

Signature of printf():
↳ Type of argument in printf()
↳ No. of argument in printf();

* Printf print the things written under double quotes..
* returns the No of character printed.

```
int x;
x = printf("kG");   // Kg
printf("%d", x);
```

assign no of character

printf("%d", x);
- printf: print No of character stored in x.
- %d: format specificer
- x: value.

e.g.1
```
#include <stdio.h>
void main(){
  int x;
  x = printf("mayur");   // mayur
  printf("\n%d", x);     // 5
}
```

2.
```
#include <stdio.h>
void main(){
  int x;
  x = 8* printf("mayur") + 8;   // mayur
  printf("\n%d", x);            // 48.
}
```

Page No:
Date: / /

3
```
#include <stdio.h>
void main () {
printf ("%d", printf ("mayur")); //mayur 5
}
```

4.
```
#include <stdio.h>
void main (){
printf ("%d", printf("mayur") *printf ("chipade"))
}
```
execution,

O/P mayur chipade 35

5.
```
#include <stdio.h>
void main () {
printf ("%d"%d ", printf("mayur"), printf("chipade"));
}
```
execution

O/P- chipademayur 57

Note -

1. if printf contain single expression then execute from left to right

2. If printf contain multiple expression then execute from right to left.

Page No.:
Date: / /

* Scanf();
↳ predefine Input function of stdio
↳ Scanf is Scanformat.

* signature -
int scanf (const char *format, <Add [Args] ...>);

* Syntax -
Scanf("format Specifiers", &var);

Scanf ("%d", &a);
↓
address of a

E.g

① #include <stdio.h>
void main (){
int x, a, b;
x = scanf ("%d %d", &a, &b);
printf ("%d", x);
3. Input

o/p - 10, 15 - 2
o/p - 10, 'A' - 1
o/p - 'A', 10 - 0

② #include <stdio.h>
void main (){
int x;
printf ("%d", scanf("%d", &x));
3. (10)

o/p - 10 - 1
o/p - A - 0

Escape Sequence (e.g. \n)

→ only used for output function [printf(), putchar(), puts()]
→ Looks like two character but always counted as single character.

	Escape Sequence	Description.
1	\a	alert
2	\n	newline
3	\b	backspace (one char)
4	\t	tab (5 char)
5	\r	return carriage
6	\"	double quate
7	\'	single quate.

Example.

① printf(" "hello" "); // Error

② printf("ABC\bD"); // ABD

3 printf("\n Computation\b\b\b\b\b\b er");
// Computation
Computerion

4. printf "Window\b\bing\r F");
// Window
winding
Finding

⑤
```
void main()
{
  printf ("\n 22");   // 22
  printf ("\b 5y");   // 25y
  printf ("\r he");   // hey
}
```

o/p - 22
25y
hey

(8)

Page No.:
Date: / /

Format Specifier.

%d → Type
↳ Beginning of f.s.

	F.S	Description
1	%d, %i	int
2	%c	char
3	%f, %e	float
4	%s	string
5	%ld, %li	long int
6.	%lf, %le	long double.
7.	%x, %X	hexadicimal
8.	%p	pointer
9	%o	octal
10	%u	unsigned.
11	%%	none (only point signal %)
12		

* Synta of F.S

<%d> [Flag] [width] [precision] <type>

↑ only use for floating point Numbers.

optional

Note - without width we cannot use flag.

e.g
```
void main(){
    int a=100;
    printf("%d", a);
    printf("\n %10d", a);
    printf("\n %010d", a);
    printf("\n %2d", a);
}
```

o/p

100

_ _ _ _ _ _ _ 1 0 0 ← width

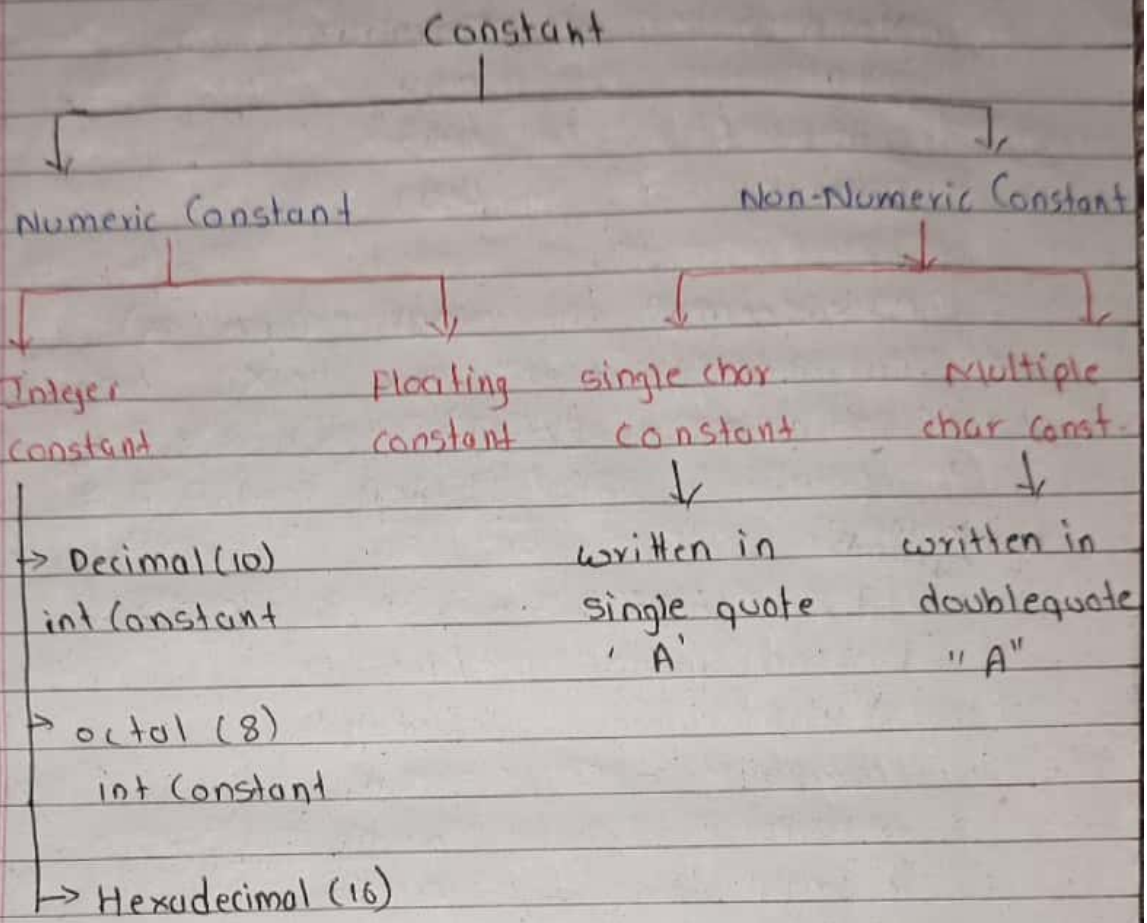
0 0 0 0 0 0 0 1 0 0 ← flag

1 [0 0]

↑ Temporary Store

C is advance compiler.

Constants.



1 Decimal int Const (Base 10)

* format Specifier : %d , %i
* Combination of digits from [0-9]

valid	Invalid
0	1,234
+2	1 234
-2	$123
10	"123"
1234	'1'

2 Octal Int Const (Base 8)

* format Specifier : %o
* Combination of digit from [0-7]
* Begin with 0 ← English Letter

Page No.
Date 15/10/22

Valid	Invalid
076	087
035	095
054	099

english letter O

3 Hexadecimal int const (Base 16)

* Format Specifier: %x, %X, %p
* Combination of [0-9] & a, b, c, d, e, f, &
  [0-9], A, B, C, D, E, F, &
  10 11 12 13 14 15

Hexadecimal number.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F,

* Begins with 0x / 0X

Valid	Invalid
0x87	0xgf
0xFc	0xhg
0xFF	0xhn

Example -

```
(1) void main()
    {
    int a=127;
    printf("%d, %x, %X, %p, %o", a, a, a, a, a);
    }
```

O/P
%d → 127
%x → 7f
%X → 7F
%p → 007f
%o → 177

Explanation.

Binary conversion

2	127	
2	63	1
2	31	1
2	15	1
2	7	1
2	3	1
	1	1

Binary.
127 = 1111111

0000000001111111
0 0 7 f
hexadecimal
$2^4 = 16$

octal number
$2^3 = 8$

001 111 111
1 7 7

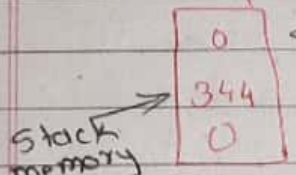
(2) void main(){ — octal number char 0
int a = 0101
printf("%d, %x, %X, %p, %o", a, a, a, a, a);

3.

O/P	explanation
%d → 65	① %d → number is octal hence
%x → 41	101 = 001 000 001
%X → 41	1 000 001 ← Binary
%p → 0041	64 + 1 = 65 Decimal
%o → 0101	② %x → 0000 0000 0001 0000 0001
	0 0 4 1

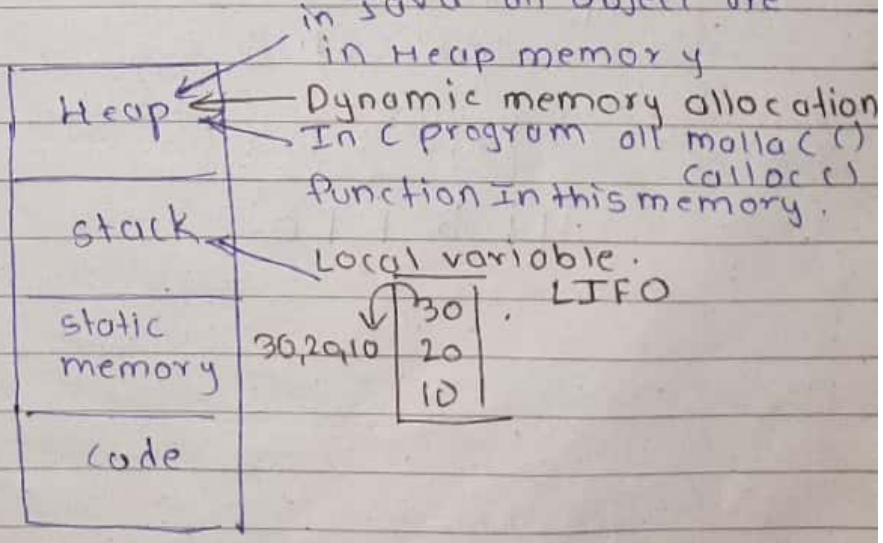
3. void main(){
printf("%d, %d, %d");
}

O/P - 0, 344, 0



← This value is only shoco in ~~turbo~~ terbo c compiler

4.



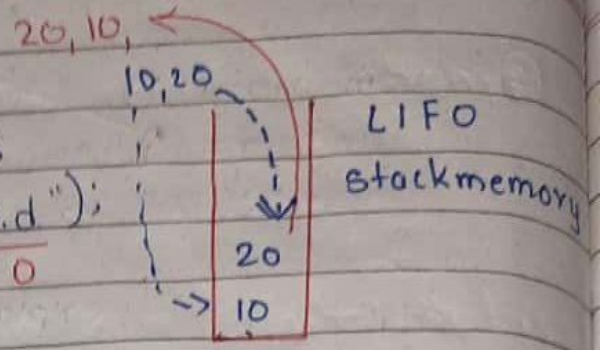
code
cimal

$2^4 = 16$

$2^3 = 8$

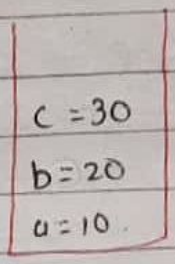
4. Void main(){
int a=10, b=20;
printf("%d, %d, %d");
}
20 10 0

o/p - 20, 10, 0



5. Void main(){
int a=10, b=20, c=30;
printf("%d, %d, %d");
}
30 20 10

o/p = 30, 20, 10

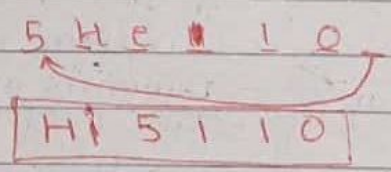


6. Void main(){
int a=10, b=20, c=30;
printf("%d, %d, %d", a, b, c);
}

o/p = 10, 20, 30

7. Void main(){
printf("5%.5s 5", strcat("Hello\r", "Hi"));
}

o/p



Hi 5 1 1 0

C Tokens :

C Tokens : it is the smallest individual unit of program are know as Token.

Types :

① Keyward -
- ↳ 32 Keywords
- ↳ written in lower case
- ↳ Each keyward has a specific meanning function
- ↳ predefined words used in programming that have special meanning to Compiler.

e.g - int a; (variable name)
for void, main()
int → Keyword
a → identifier
void → keyword

② Identifier -
- ↳ User define words for variable name, arrayname function name etc.
- ↳ Refer to name given to entities such as variable, function, array etc.
- ↳ first character should be character or underscore.
- ↳ Valid identifier can have character (both upper & lower) digit and underscore.
- ↳ name should not be a Keyword. ~~Identifier~~

e.g - int a;
a → Identifier.

- ↳ C is case sensitive. = a, A
- ↳ Length of identifier <= 32.

③ Constants: A Constant is a name given to

Page No.:
Date: / /

a variable. whose value can't be change or altered.

④ Operator -
An operator is a simple a Symbol that is use in perform operations.

⑤ Special Symbol - (~, #, %, ^, &, *, (), -, +, =, {, }, ...)
↳ Alphabet = 52 (a-z), (A-Z)
↳ Digit = 0-9 = 10
↳ Special Symbol - 255 - 62 = 193

~n = -(n+1)

```
int a=3;
~a // -4
```

or

0011

1's

1 = -
0 = +

1 | 100

Sign | magnitude

-4

Operators

Operators -

↳ An operator is a special symbol which perform particular operation

↳ In C programming language there are 44 operator and depending on the no of operands these are classified into 3 Types

e.g a + b

operand operator.

Operator

Unary op^r	Binary op^r	Ternary op^r
++a (pre incre)	a+b	a>b ? a : b
--a (pre decre)	a-b	
a++ (post incre)	a<<2	execute from right to left.
a-- (post decre)	a>>1	
+a (unary add)	a % 2	
-a (unary sub)	Binary operator execute from left to right.	
!a (Not oper)		
~a (Negation)		

unary operator execute Right to left

↳ In any expression in which operator contain highest precedence, it needs to be executed first then the operator with low precedence.

↳ if equal precedence is occour if it is binary operator then go from right to left.

Page No:
Date: / /

Types -

1. Arithmetic operator L→R +, -, *, /, %
2. Relational operator L→R <, >, >=, <=, ==
3. Logical —11— &&, ||, !
4. Assignment —11— =, *=, /=, %=, +=, -=, <<=
5. Increment / Decrement ++, --
6. Bitwise —11— &, ^, | L→R
7. Conditional —11— ? :
8. other operator

Operator Precedence.

1. (), [], →, . L→R
2. +, -, ++, --, !, ~, *, &, size of() R→L (unary operator)
3. *, /, %. L→R
4. +, - L→R
5. << (Left shift), >> (Right shift) L→R
6. <, <=, >, >= L→R
7. != , == L→R
8. & Bitwise AND
9. ^ Bitwise XOR
10. | Bitwise OR
11. && logical AND
12. || logical OR
13. ? : Conditional operator R→L
14. =, +=, -=, *=, %=, &= R→L
15. , comma operator

① Arithmetic operator +, -, *, /, %.
→ Binary operator : L → R
→ Unary operator : R → L.

① e.g- void main() {
int a = 10;
-a; ← // Dummy operator.
printf ("%d %d", a, -a);
↑ ↑
10 -10
}.
o/p - 10, -10

② void main() {
printf ("\n %d", 15 %-6); // 3
printf ("\n %d", -15 %6); // -3
printf ("\n %d", 15 %-6); // 3
printf ("\n %d", -15 %-6); // -3
}.
Note:
→ mod operator works only for int datatype
↳ you can use fmod() in <math.h>
↳ % operator works only for int.

② Relational operator → L→R <, >, <=, >=

③ void main()
{
int x, a=10, b=7, c=9
x = a<b<c;
printf ("%d", x);
}.

o/p = 0 1

explanation - 0 1
(10 < 7 < 9)

Page No.:
Date: / /

② void main () {
int x = 3;
if (x > 2 > 1)
printf ("java %d", x);
else
printf ("python %d", x);
}

Olp.—

python 3

③ ① (3 > (1) > 2) > 0; 1 1 0

② (5 > (3) > 2 > 1 > 0 > -1); 1

③ x = (4 < 5) >= 3; 1 1 0

④ x = 6 <= 6 >= 1 > 0; 1 1 0 1.

⑤ x = 10 == 10; 1 1 1

⑥ x = (2 <= 6) == (6 >= 2); 1 1 1.

⑦ x = 3 == (2 > 5) <= 3; 1 1 0

* Logical operator –
  - ↳ && Logical AND
  - ↳ || Logical OR
  - ↳ ! Logical NOT

! , && , || — high to low.

A	B	&&
0	0	0
0	1	0
1	0	0
1	1	1

A	B	||
0	0	0
0	1	1
1	0	1
1	1	1

A	!
0	1
1	0

NOTE:

AND ↳ In And operator if left expression is false then right expression is never executed.

e.g –

```
#include <stdio.h>
void main ()
{
    int a = 0, b;
    b = a && printf("Hello");
    printf("\n %d", b);
}
```

o/p – 0

explanation – 0 x 0 = 0

↳ In c and c++ Language all logical operator returns 0 or 1

↳ Every Non-zero value is called true, when the value become 0 it is false.

OR ↳ In OR operator if left expression is true then right expression will never execute.

e.g –

Page No.
Date: / /

```
Void main (){
  int a=10, b;
  b = a || printf ("know");
    Printf ("\n %d", b)
}3
```

o/p = 1

An expression containing logical operator returns either 0 or 1 depending upon whether expression result true or false Logical operator are commonly used in decision making in C programming.

NOT

```
# include <stdio.h>
void main (){
printf ("\n %d", !printf ("792"));
}
```

unary operator

second execute, first execute

o/p - 792
0

examples -

①
```
Void main ()
{
  int i=10, j=5;
  j = j || (i++ && printf ("KG"));
  printf ("\n %d ; %d", i, j);
}3.
```

o/p - 10, 1

2. Void main () {
int a, b, c, d, A, B, C;
a = printf ("A") && printf ("B") && printf ("C");
printf ("\n %d", a);
a = printf ("A") && printf ("B") || printf ("C");
printf ("\n %d", a);
}

O/P - A B C
1
A B
1

Page No.:
Date: / /

* Assignment operator

An assignment operator is used for assigning a value to a variable. The most Common assignment operator is =, +=, -= *=, /=, %=

e.g -

```
void main(){
  int a;
  printf("\n a=%d", (5==6));
}
```

relational operator returns 0 or 1

o/p = a = 1

Best example

```
void main(){
int x;
x=7,8,9;
printf("%d",x);
}
```

high precidence of =

o/p - 7

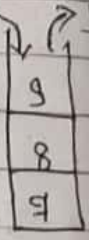
exp-

= has high precidence as compare to comma ,

```
void main(){
int x;
x=(7,8,9);
printf("%d",x);
}
```

o/p = 9

7,8,9



exp-

uses stack memory then LIFO

* Increment / Decrement operator
  ↳ When we want to change a initial value of a variable by 1 then we go for inc / dec operator
  ↳ Depending on position these operator are classified into 2 types.

Pre	post
++a	a++
--a	a--

↳ When we are working with pre-operator before evaluating the expression value need to be changed.

++a

pehle inc kro fir use karo

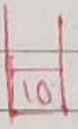
↳ When we are working with post-operator after evaluating the expression value need to be change.

a++

pehle use kro fir change kro.

examples

①
```
void main() {
    int a = 10, b;
    b = ++a;
    printf("%d, %d", a, b);
}
```



o/p = 11, 11

Page No.:
Date. / /

2. void main()
{
int a = 10, b;
b = a++;
printf("\n %d, %d", a, b);
3.

o/p = 11, 10

3. void main() {
int a = 1, b;
b = ++a + ++a + ++a;
printf("\n %d, %d", a, b);
3.

o/p = 4, 12

4. Void main(){
int a = 1, b;
b = ++a + a++ + ++a + a++;
printf("\n %d, %d", a, b);
3.

o/p = 5, 12

explanation. –

a = 1

b = ++a + a++ + ++a + a++ = 12

step 1) pre- 3 3 3 3

2) post 4 5 = 5

a = 5 b = 12

Page No.:
Date: / /

practice problem on increment decrement.

① void main ()

```
{
 int a=1;
  a= ++a * a++ * ++a;
  printf ("%d" ,a);
   getch ();
 }.
```

o/p = 28.

2. void main ()

```
{
 int a = 1;4
 a = a++ * a++ * a++ ;
  printf ("\n %d", a);
 }.
```

o/p = 4

3. void main (){

```
  int a,b;
   a=b=10;
18 a= a-- + --b;
25 b= --a + b--;
   printf (" %d,%d", a,b);
 }.
```

a: 10, 18, 17 ; b: 10, 9

o/p = 17, 25

4. void main() {
int a,b;
a=1 , b=3;
a= ++a + --b + a--;
b= b++ + --a + ++b;
printf ("\n %d , %d", a,b);
}

O/P - 3 , 10

explanation -

a = 1 5, 4, 3
b = 3 , 9 , 10

a = ++a + --b + a--;
2 2 1
5

b = b++ + --a + ++b;
3 3 3
9 → 10

5. void main() {
int a,b,c;
a=1; b=2; c=3;
a = a-- + ++b + --c;
b = ++a - b++ + c--;
c = a++ + ++b + c++;
printf ("\n %d, %d, %d", a,b,c);
}

A B C

O/P - 7 7 15

6. void main() {
int a= 1;
printf ("\n %d, %d, %d, %d", ++a, a++, ++a, a++);
}
5 3 3 1

O/P = 5, 3, 3, 1

* Bitwise operator - L → R
  - & Bitwise AND
  - | Bitwise OR
  - ^ Bitwise XOR
  - ~ Bitwise NOT
  - << leftshift left
  - >> Shift Right

↳ When we need to manipulate the data on binary representation then we go for bitwise operator.

↳ when we are working with Bitwise operator directly manipulation may take place on memory only.

↳ Bitwise operator need to be applied for int data type only.

AND → ① void main(){
```
int a=28 , b=12, c;
c = a & b;
printf("\n%d", c);
}
```
o/p - 12

explanation - $(28)_{10} = (11100)_2$

$(12)_{10} = (1100)_2$

$12 = = 01100$

② void main(){
```
int a = 0xff, b = 0xcc, c;
c = a & b;
printf("%x, %X, %p, %d", c, c, c);
}
```
(capital)

o/p - cc, CC, 00CC, 204.

$(0xff)_{16} = (1111\ 1111)_2$

$(0xcc)_{16} = (1100\ 1100)_2$

```
 1111 1111
 1100 1100
-----------
 1100 1100
```
204.

Page No.:
Date: / /

③ void main(){
int a, b;
printf("%d, %d", -1&3, -1|3);
}

o/p - 3, -1

$(-1)_{10} = (0000\ 0000\ 0000\ 0001)_2$

2^{15} compliment

$(-1)_{10} = (1111\ 1111\ 1111\ 1110)_2 + 1$

$(-1)_{10} = (1111\ 1111\ 1111\ 1111)_2$

$(3)_{10} = 0000\ 0000\ 0000\ 0011$

OR | = 1111 1111 1111 11 11 = (-1)$_{10}$

AND & = 0000 0000 0000 0011 = (3)$_{10}$

XOR ^ → void main(){
int a = 0xff, b = 0x34, c;
c = a^b;
printf("\n %x, %X, %p, %d", c, c, c, c);
}

XOR → Trooth Table

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

$(ff)_{16}$ = 1111 1111

$(34)_{16}$ = 0011 0100

1100 1011

c b

o/p = cb CB 00cb, 203

```
void main(){
  int a;
  a = 2^(3+5)^7;
  printf("%d", a);
}
```

(2) = 0010
(8) = 1000
XOR 1010
(7) = 0111
1101
8 4 2 1

O/P = 13

NOT ① void main(){
  int a = 5, b;
  b = ~a;
  printf("%d", b);
}

O/P = -6

~5 → 0101
↓ 1's
1010
↓ 2's
value
-6

1 आने पर value negative
0 आने पर value positive

010
101 5
+ 1 +1
110 6
6

Note →
~n = -(n+1) ← short cut.
~5 = -(5+1)
~5 = -6

② void main(){
  int a;
  a = ~print("KG");
  printf("%d", a);
  getch();
}

unary operator R→L

(~2)=-3

O/P - KG -3

③ void main(){
  int a;
  a = -(~(!(~print("KG"))));
  printf("%d", a);
}

=1 1
0
-3
2

KG 1

* Bitwise Left shift Operator

↳ a<<2 = shift left a by 2 bits.

Example

①
```
void main (){
int a= 0xff, b;
b= a<<2;
printf ("\n %x, %X, %p, %d", b, b, b, b);
}.
```

$(ff)_{16} = (0000\ 0000\ 1111\ 1111)_2$

(leftmost bits dropped ✗, remaining bits shifted left, 00 appended on the right)

0000 0011 1111 1100 = 1020

0 3 f C

O/p = 3fc, 3FC, 03fc, 1020

②
```
void main (){
int a= -10;
b= a<<4;
printf ("\n %x, %X, %p, %d", b, b, b, b);
getch();
}.
```

O/p - ff60, FF60, FF60, -160

Solution -

1's & 2's compliment.

e.g. -

1's	2's Compliment
10100	10100
01011	01100

← Left side થી જો પહેલા 1 મળે ત્યાં સુધી તો પછીના As it is આવે પછી ઉલટા

$(-10)_{10} = (1111\ 1111\ 1111\ 0110)_2$

(leftmost 1111 dropped ✗, remaining bits shifted left, 0000 appended on the right)

1111 1111 0110 0000

f f 6 0

* Bitwise Right shift operator.
↳ a>>2 = shift right 'a' by 2 bits.

Example -

① void main() {
int a=100, b;
b = a >> 3;
printf("\n %d", b);
getch();
}

Solution -

$(100)_{10} = (1100100)_2$

$\frac{1100}{12}$

O/P = 12.

2. void main(){
int a= -10, b;
b = a >> 4;
printf("\n %x, %x, %p, %d", b, b, b, b);
getch()
}

Solution -

$(-10)_{10} = (0000, 0000\ 0000\ 1010)$

$2^{'s}$

Note- जर नंतर Negative असेल तर '1' नाहीतर zero '0'

1111 1111 1111 0110

1111 1111 1111 1111

f f f f.

O/p -

ffff, FFFF, ffff, -1

NOTE :

↳ If No is negative and we perform shift right operator then left empty spaces should be filled with 1's.
So Sign Bit will not be modify.

↳ For any positive no, right shift 15 bit makes the valve 0 becouse positive data representation is available in 15 bit towards right side.

$(3)_{10} = (0000\ 0000\ 0000\ 0011)_2$

right shift केले तरी पण Ans '0'

Page No.
Date / /

* * * * * * * * * * *

* Conditional operator- [Right → Left]

(? :) → Control flow statement.

Ternary operator means it contain 3 argument i.e left middle, right side argument.

e.g: a>b ? a : b
left middle right

↳ In Conditional operator if Condition is true then return with middle argument if Condition is false then return with right argument & left expression is treated as Condition.

Point you need to remember:

- No of question marks & colon should be equal.
- Every colon should match with question mark.

Examples ← बरोबर असेल तर पहिली otherwise second

① a = 10 ? 20 : 30; // 20

② a = 5<2 ? 20 : 30; // 30

③ a = 1 != 2<5 ? 10 : 30; // 30
1 != 1

④ a = 2>5 ? 10 : 20 : 30; error

⑤ a = 2<5 ? 10 : 20 : 5 ? 30; error.

⑦ a = 5>8 ? b ? 5 ?; error.

7 a = 5<2 ? 10 : 2 != 2>5 ? 20 : 30 20 // 20

Page No.:
Date: / /

(8) a = 2<5 ? (1 != 2>8 ? 10 : 20) : 30;

0

a = (2 < 5) ? 10 : 30

1

a = 10

(9) a = 5<2 ? 10 : 1 != 2<5 ? 20 : ((5 != 0) ? 30 : 40);

5<2 ? 10 : (1 != 2<5) ? 20 : 40

1

0

5<2 ? 10 : 40

0

40

(10) a = 2<6 != 1 ? 10 : 5 > 2 ? ((8 != 0) ? 20 : 30) : 40

a = 2<5 != 1 ? 10 : (5 > 2) ? 30 : 40

1

(2 < 5) != 1 ? 10 : 30

1

0

30

(11) void main() {
int a, b = 7;
a = b>10 ? b++ : b<14 ? b>>1 : ++b;
printf ("\n %d, %d", a, b);
getch();
}

Solution –

OIP = 4, 9

b 8>10 ? b++ : 8<14 ? b>>1 : ++b;

1

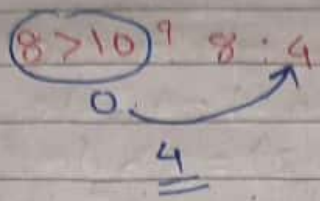
4

8

right shift

8/2 = 4

a	b
7	7
	8
	9



* Other Operator :

① Size of : size of an operator and a keyward in C & C++ language. By using size of () operator we can find size of the given input data.

e.g -

```
void main (){
    clrscr ();
    printf("%d, %d, %d", sizeof(int), sizeof(char),
            sizeof(float));
    getch ();
}
```

(2 → int, 1 → char, 4 → float)

O/p - 2, 1, 4

→ sizeof ('A') → size of (65) = 2 byte
→ size of ("A") → 2 bytes

e.g -
```
void main ()
{
    char x = 'A';
    printf ("\n %d", size of (x));
    getch ();
}
```

O/p = 1

Page No:
Date: / /

* size of string = size of (char) + 1
* size of (500) = 4 byte
* size of (5.2) = 8 byte.
  ↳ System By default consider floating point no. as double.

Example -

① void main(){
{
clrscr();
printf("\n %d, %d", sizeof("ABC"), strlen("ABC"); (4, 3)
printf("\n %d, %d", sizeof("ABC\n\n"), strlen("BC\n\n"); (6, 5)
printf("\n %d, %d", sizeof("ABC\0\0"), strlen("ABC\0\0"); (6, 3)
getch();
}

o/p - 4, 3
6, 5
6, 3

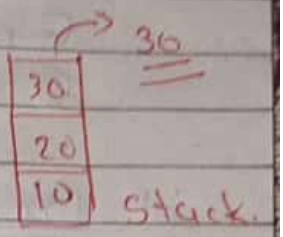
② void main(){
int a = 10;
printf("%d\n", sizeof(++a)); (2)
printf("%d\n", a); (10)
getch();
}

In size of operator modifications are not allowed.

o/p = 2
10

2 Comma Operator:
↳ it is the lowest precedence operator in C & C++ language.

```
void main (){
  int a, b;
  a = 10, 20, 30;
  b = (10, 20, 30);
  printf ("\n %d, %d", a, b);
  getch();
}
```



o/p = 10, 30

);
{ Scanf("%d");

(38) * Iteration (looping):

There are 3 types of iteration:
① For loop
② while loop
③ do while loop.

1. For loop:

Syntax: for (initialization; test expression; updat expre)
{
// loop body
}

* steps of execution -

```
for ( i=0 ; i<=5 ; i++)
  {
        body
  }
```

(① i=0 ② i<=5 ③ body ④ i++)

Example -

```
void main (){
  int i;
  for (i=1 ; i<=5 ; i++) {
        printf ("mayur");
  }}
```

o/p - mayur mayur mayur mayur mayur.

* How money times for loop will execute.

if for(i=1; i<=3; i++)

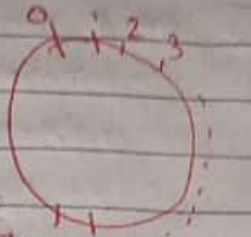
i=1	i=2	i=3	i=4
i<=3	2<=3	3<=3	4<=3
✓	✓	✓	✗

4 times
3 1
T F

for(i=1; i<=n; i++) = n+1

Example -

① void main()
{
int i=1;
for(;①;)
i++;
printf("\n%d",i);
getch();
}



i=0 ✗
✓=0

expre)

o/p = 0.

② void main(){
int a=1;
for(a++; a++<=2; a++)
a++;
printf("\n%d",a);
}

a=1, 2, 3, 4, 5, 6

o/p = 6

③ void main()
{
int a;
for(a=1; a<=5; a++);
printf("\n%d",a);
}

for loop with empty Body.

o/p = 6.

Solution- 1<=5 ✓
2<=5 ✓
3<=5 ✓
4<=5 ✓
5<=5 ✓
6<=5 ✗

(40)

Page No.:
Date: 16/10/22

(4) void main(){
int i;
for(i=16; i>=1; i/=2)
printf("%d\n", i);
getch();
}.

o/p - 16
8
4
2
1

5. void main(){
int i;
for(i=1; i<=2; printf("knowledge"), i++)
printf("Gate");
3.

o/p - Gate knowledge Gate knowledge.

6. void main(){
int i;
for(i=1; i<=2; printf("knowledge"), i++);
printf("Gate");
}.

o/p
knowledge
knowledge
Gate.

7. void main (){
int i,j;
for(i=0, j=-10; i<=5; i++, j++)
printf("\n%d %d", i, j);
}

o/p - 0 -10
1 -9
2 -8
3 -7
4 -6
5 -5

8) void main (){
int i, j;
for(i=1; i<=5; i++){
for(j=1; j<=3; j++){
printf("\n %d, %d", i, j); }.
getch();
}

o/p.

1 1
1 2
1 3
2 1
2 2
2 3
.
.
.
5 1
5 2
5 3.

i=1 i=1 i=1 i=1
j=1 j=2 j=3 j=4
✓ ✓ ✓ ✗
i++
i=2 i=2 i=2 i=2
j=1 j=2 j=3 j=4
✓ ✓ ✓ ✗
.
.
.
i=5 i=5 i=5 i=5
j=1 j=2 j=3 j=4
✓ ✓ ✓ ✗

(2.2)

Page No.:
Date: / /

9. void main (){
int a;
a = 1;
for (a++; a++ <= 2; a++);
a++;
printf ("\n %d", a);
}

a 1 2 3 4 5 6

o/p - 6

10 void main (){
int a = 1;
for (a++; a++ <= 2; a++)
for (a++; a++ <= 3; a++)
printf ("\n %d", a);
}

o/p = 7.

11 void main (){
int a = 1;
for (a++; a++ <= 2; a++)
for (a++; a++ <= 6; a++)
printf ("%d", a);
}

o/p - 11

12 void main (){
int a;
for (a = 255; a; a = 0; printf ("\n%d", a));
}

o/p - 0

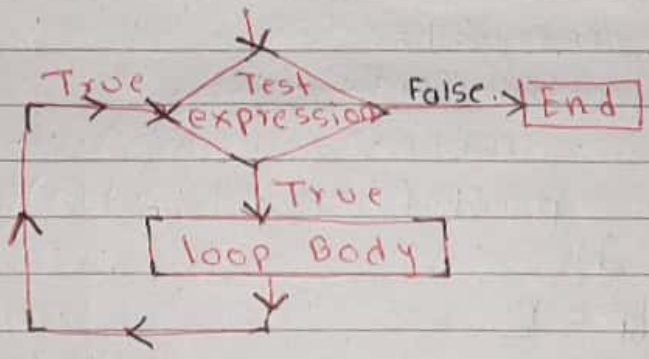
2. While loop -

↳ In for loop c compiler calculate in advance how many times loop will be executed.

↳ In while loop, c compiler can not calculate in advance how many times loop will be executed.

↳ In do while loop similar to while loop but body execute at least onetime.

* Syntax : while (test expression)
{
// loop Body
}



Example -

① void main(){
int a = 1;
while (a++ <= 10) But increment ++
printf ("\n %d", a);
}

O/p - 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Solution -

a	1	2	3						10	11

1 <= 10 ✓
2 <= 10 ✓
3 <= 10 ✓
.
.
.
10 <= 10 ✓
11 <= 10 ✗

2. void main() {
int a=1;
while(++a <= 10)
printf("\n%d", a);
getch();
}

o/p = 2, 3, 4, 5, 6, 7, 8, 9, 10

Solution

2,3,4 ... 10, 11 (x) a = 1

while(++a < 10)

2,3,4 ... 10 2

3. ~~while~~ void main() {
int a=1;
while(a++ <= 2)
a++;
printf("\n%d", a+10);
}

o/p = 14

Solution -

a = 1 2 3 4

while(a++ <= 2)

a++

4+10 = 14

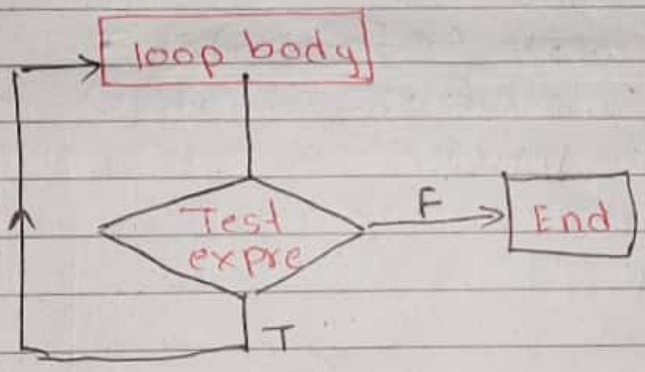
4. void main() {
int a=1;
while(a <= 32767)
{
printf("\n%d", a);
a++;
}
getch();
}

o/p - Infinite loop -

5.
```
# include <stdio.h>
void main (){
  int a = 1;
  while (a++ <= 2)
      while (a++ <= 3)
          a++;
      printf ("%d", a);
      getch();
  }
```

o/p = 6

3. Do - While loop : // exit status loop.



Syntax :

```
do
{
   // loop body
} while (test condition );
```

Example :-

① Void main() {
int a=1;
do ; ← require otherwise Error.
while (a++ <=2);
printf ("\n %d", a);
getch ();
}
O/P = 4

2. void main () {
int a =1;
do;
while (a++ <=2);
while (a++ <=4);
printf ("\n %d", a);
getch();
}
O/P - 7

✱ Decision Making Statement :

① if Statement :

i) for single statement

```
if (condition)
    stmt 1;
```

ii) For multiple statement

```
if ( condition)
 {
     stmt 1;
 }
```

जर if च्या पुढे सेमीcolon दिला नसेल आणि एकच Statement असेल तर valid आणि जर if च्या पुढे सेमी colon दिला असेल आणी खाली नवीन statement असेल तर ते seperate statement असते.

Note :

① if with empty body

```
if (condition);
    stmt;
```

↓

```
if (condition)
 {;
    // empty
 }
```

② condition → 0 = false
→ Non zero = True

Examples —

Note - जर if stmt zero value आली तर execute होत नाही

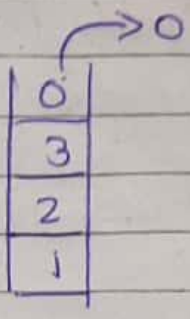
①
```
void main() {
 int a=4, b, c;
 if (b= a%2)
    c=10; X
 printf("\n%d, %d, %d", a, b, c);
}
```

o/p - 4, 0, 0

Page No.:
Date: / /

② void main(){
  if (printf("Hi"))
    printf("Hello");
  3.
o/p - Hi Hello.

3. void main()
{
  if (1,2,3,0)
  printf("Hello"); ×
  printf("Bye");
}
o/p — Bye



② <u>if-else statement</u> =

Syntax = if (condition)
{
  stmt;
}
else {
  stmt;
}.

if with empty body

if( );
else
{
  stmt;
}.

जर if empty असेल तर else execute होते.

Example -

① void main(){
  int a=14;
  if (!a==2);
  else
    printf("mayur");
}.

o/p - <u>mayur</u>

0
0 == 2 ×
!14 == 02

Page No.:
Date: / /

② void main() L→R
{
  if ((3>2)>1)
    printf("java");
  else
    printf("python");
}
o/p : python.

③ void main()
{
  if ((1!=(2>5)) && ((2<5)!=0))
  {
    printf("B");
    printf("A");
  }
  printf("\n BE");
}
o/p - BA
BE

④ void main(){ ← semicolon नाही म्हणून एकदा printf
  if (!5)o
  printf("B");
  printf("A");
  printf("\n Hello");
}
o/p - A
Hello.

Page No.:
Date: / /

⑤
```
void main (){
  if (!5 != 5)
  {
    printf ("mayur");
    printf (" chipade");
  }
  else
  {
    printf ("BE");
  }
}
```
o/p - mayur chipade

⑥
```
void main()
{
 float x = 1.2;  // 1.200000
 double y = 1.2; // 1.200000000000000
  if (x == y) ×
    printf("In OK");
 else
    printf (" In NOT OK");
}
```
o/p = NOT OK

7
```
void main (){
  if ((!1 != 2>5) || (10 != 2>5))
  {
   printf ("mayur");}
  else
    printf (" chipade");
}
```
o/p - mayur

⑧
```
void main () {
  if ( printf (" ") )
     printf ("x");
  else
     printf ("y");
}
```

Note -
एक space Allocate होती अगर जगह Allocate होती नहीं तो "" तब y print होता अन्यथा

o/p = x

3 SWITCH case :

↳ Switch is a Keyword.
↳ By using Switch we can create selection statement with multiple choice can be created.

Syntax -

```
switch (Expression)
{
   case Label 1 : Stmt 1;
   case Label 2 ; Stmt 2;
   default : Sn
}
```

e.g -

```
Switch (1)
case 1 : ------
   break;
case 2 : ------ ✓
case 3 : ------ ✓
case 4 :
default :  ✓
```

Page No:
Date: / /

Examples:

①
```
void main()
{
  float a;
  a=8;
  switch (a)
  {
    case 1: printf ("A");
            break;
    default : printf ("Not ok");
  }
}
```

o/p - Error

explanation - Only integer value are Allowed in switch.

②
```
void main(){
  int a;
  a=2;
  switch(a)
  {
    case 1.0: printf("A");
    break;
  }
}
```

→ only int
Not allowed float

o/p - Error.

3. Void main()
{
int i;
i = 2/3;
Switch (i)
{
case 0.2 : printf ("A");
break;
case 2+2 : printf ("B");
}}.

O/p - A

4. Void main (){
int i;
i = 5>2;
Switch (i)
{
case 6>3: pf ("A");
break;
case 5>2: pf ("B");
break;
}}.

Ambiguity problem
error

O/p - Error

5. Void main (){
int i=2, j=3;
Switch (i)
{
Case 2: printf ("2");
break;
case (j): printf ("3");
break;
}}.

variable Not allowed

O/p - Error.

Page No.:
Date: / /

6. Void main (){
int a;
a = 2;
Switch (a)
{
case 2: printf ("2");
case 1: printf ("1");
case 0: printf ("0");
case 4: break;
default: printf ("ok");
}}
O/p - 2,1,0
Note - जर Switch मध्ये कोणतीही value नसेल तर default print होते.

7. void main (){
int i = 2;
Switch (i)
{
case 2: printf ("2");
break;
case 1: continue; continue are Not allowed in switch case
}
}
O/p - error.

8. void main (){
int a;
a = 2;
Switch (a)
{
case 1: printf ("1");
break

```
        case 2: goto A;
    }
    A: printf (" Kg");
}
```

o/p - Kg.

9.
```
void main () {
    switch (i)
    {
        int a;
        a = 3;
    case 1: printf ("A %d", a);
            break;
    }}
```

default value are zero

o/p - A0

↳ When we are working with Switch statement, at the time of compilation condition or expression return value will be map with case constant value.

↳ At the time of execution, if the corresponded block is available then control will pass to corresponded case.

↳ From matching case upto break everything will be executed, if break is Not available then execute the next case.

↳ if At the Line of execution of switch statement if the matching labell is not available then control is passed to default block.

Page No.:
Date: / /

↳ Default is a special kind of case and it is mapped automatically when case not matched
↳ Default is an optional case.
↳ break is a keyword which is used to terminate the execution of switch case.
↳ expression must be int / char type. if it is float / double then compiler shows the error.

```
e.g- Switch ('A')
     {
        case A: ---;
        Case B: ---;
     }
```

↳ Inside Switch we can create cases in random order, because here the Compiler check the equality.
↳ According to ANSI-C we can create about 257 cases.
↳ we can also define default within Switch block anywher.

Page No.:
Date: / /

4 Jump Statement:

1) break
2) continue
3) go to
4) exit

① break :

↳ break is a keyword, by using break we can terminate the loop body or switch body.

e.g -

```
① void main()
  {
     int i;
     for (i=1; i<=5; i++)
      {
        printf("%d", i);
        break;
      }
     getch();
  }
```

o/p - 1

```
② void main()
  {
     int a;
     for (a=0; a<=100; a=a+5)
      {
        printf("%d ", a);
        if (a==50)
           break;
      }
  }
```

o/p - 0 5 10 15 20 25 30 35 40 45 50

Page No.:
Date: / /

Note - using break is always exceptional but it is recommended to place it within the loop body or switch body only.

3.
```
void main ()
{
  int a;
  a=2;
  while(a<=20)
    {
      printf("%d", a);
      a+=2;
      if(a>=8)
        break
    }
}
```
o/p - 2, 4, 6, 8

4.
```
void main ()
{
    int a;
    a=15;
    while (a>=1)
      {
        a-=2;
        if(a<=5)
          break
        printf("\n%d", a);
      }
}
```
o/p - 13, 11, 9, 7

5.
```
void main (){
  int a=2;
  while (a<=20)
  {
    printf("%d", a);
    if(!a)
    break
    a+=2;
  }
  getch();
}
```
o/p - 2, 4 6 8 10 12 14 16 18 20

6.
```
void main(){
  int a;
  a=1;
  while (a<=15)
  {
    printf("\n%d", a);
      if(a>=5);
      break;
      a+=2;
  } getch();
}
```
← if with empty body.

o/p = 1.

Page No.:
Date: / /

2. Continue:

↳ Continue is a keyword using continue we can control the next iteration.
↳ without iteration we can use continue statement
↳ using continue is optional. it should be placed with the loop body only.
↳ In implementation where we know the maximum no. of representation but in some condition is available where we required to skip the statement's from the loop.

Example -

①
```
void main (){
  int a;
  a = 0;
  while (a<100)
  {
    a+=2;
    if (a>40 && a<=80)
    continue;
    printf ("%d", a);
  }
}
```

Skip the statement.

one condition जहाँ भी condition असत्य वहाँ continue होत जाही.

continue.

o/p - 2, 4 6 8 10 ..... 38. 82 84 100

②
```
void main (){
  int a = 75;
  while (a>1)
  {
    a -= 2;
    if (a>25 && a<50)
      continue;
```

```
    printf ("%d", a);
  }
}
```

o/p - 73 71 69 67 ..... 51 25 23 21 19
17 15 13 11 9 7 5 3 1

3.
```
void main(){
  int i=1;
  while (i<=15)
  {
    printf("%d", i);
    if (i>=15 && i<=45)
      continue;
    i=i+2;
  }
}
```
o/p - 1 2 4 6 8 10 12 14 15 15 15 ... ∞. infinite

3. **Go to statement :**

↳ it is a keyword in C & C++ language.

↳ using goto statement we can transfer the control anywhere in the program

Syntax :

→
```
goto label;
....
....
...
label:
```

Example -

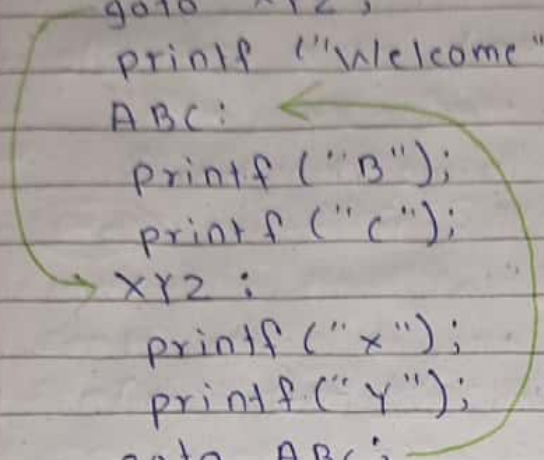
① void main(){
printf("A");
printf("B");
goto ABC;
printf("Hello");
printf("Welcome");
ABC:
printf("C");
}
o/p - A B C

② void main(){
printf("KG");
printf("Wel");
ABC:
printf("A");
printf("B");
goto XYZ;
printf("Hello");
XYZ: printf("Com");
getch();
}
o/p - KG Wel A B Com

3. void main(){
int i=2;
Even:
printf("%d", i);
i=i+2;
if(i<=20)
goto Even;
getch();
}
// o/p = 2 4 6 8 10 12 14 16 18 20

Page No.:
Date: 17/10/22

```
1. void main () {
     printf ("A");
     goto XYZ;
     printf ("Welcome");
     ABC:
      printf ("B");
      printf ("C");
     XYZ:
      printf ("x");
      printf ("Y");
      goto ABC;
     getch ();
   }
```



o/p - AXYBCXYBCXY ------ ∞

infinity

*4 Exit function():

↳ it is a predefined function of <process.h> header file & is used for termination of execution of program.

↳ exit(0);

→ indicate successfull Termination.

↳ in GCC it is a predefine function of <stdio.h> header file.

example -

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int n, i;
  printf ("no. dall bey:");
```

```
Scanf ("%d", &n);
for (i=2; i<n; i++)
{
    if (n%i==0)
    {
        printf ("prime nahi hai");
        exit(0);
    }
}
printf ("%d is prime", n);
return 0;
}
```

No. daal bay:

O/P - 47

prime nahi hai.

Pointers :

Page No.:
Date: / /

* Pointers :

↳ A pointer is a variable which hold address of same variable.

↳ When we are working with pointer we require to use the following Operator

① & = address of operator
(%p, %x, %X, %u)

② * = indirection operator / dereference operator value at address

↳ pointer is a variable which is used to store memory address of another variable

↳ pointer in c is a user defined data type. it is build by using primary data types. variable hold values and pointers hold their memory address where these variable are located in memory.

Example –

Suppose A

100 ← value
FFFF ← address.

① void main () {
int a=10;
printf ("\n Address of a= %p", &a);
printf ("\n value of a= %d, %d", a, *(&a));
getch ();
}.

O/p – Address of a = FFFF
value of a = 10, 10

&a
*(FFFF)
value at address
10

Page No:
Date: / /

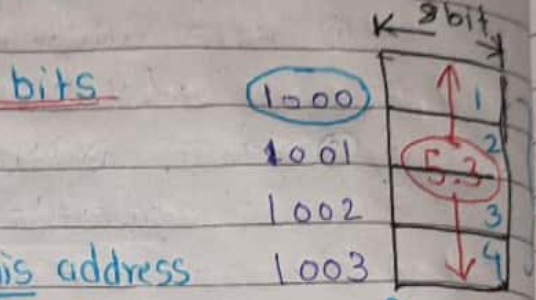
↳ When we are working with & operator, it always return this address of a variable.

↳ Starting address of a variable is called this address

4 byte = 32 bits

float a = 5.3

&a = 1000 = this address



this address is also known as first address

↳ Byte addressable memory.

↳ 52 cell चा unique address No Hota.

System
(Data Interpretation Mechanism)

Little Endian	Big Endian
L Byte — L Address	L Byte — H address
H Byte — H Address	H Byte — L Address

By default process follow little Endian mechanism.

Endian म्हणजे Lower Byte ला Lower Address मेटणार आणि Higher Byte ला higher Address मेटणार.

→ In implementation when we are printing the physical address we use.
%p, %x, %u, %lu, %lp format specifier only.

↳ %p, %x, %lp will print physical address in hexadecimal format.

↳ %u, %lu will print the address in decimal format.

4byte
→ %p, %x, %u will print the physical address in 16 bit format.

→ %lu, %lf will print the PA, in 32-bit format.

Why %d is Not use
∴ %d can go up to 32767
∴ %d is not used for address
→ for address (0 to 65535)
→ Also address must be +ve.

Page No:
Date: / /

Declaration of Pointer —

<data type> * <ptr-name>;

- <data type>: int, char, float
- <ptr-name>: any valid identifier

```
int *p;
char *c;
float *f;
void *v;
```

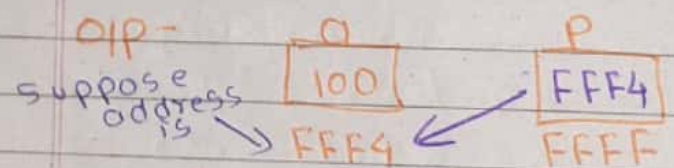
↳ Any kind of pointer size is 2 byte only because it hold address. i.e common for any datatype.

↳ The address of variable is unsigned type so size of pointer is always 2 byte.

Example

```
void main()
{
  int a=100;
  int * p; // Pointer Declaration.
    p= &a;
printf("\n Address of a= %p, %p, %p", &a, p, *(&p));
printf("\n value of a= %d, %d, %d, %d", a, *p, *(&a), **(&p));
  getch();
}
```

o/p -



printf 1 → &a, p, *(&p) → *(FFFF)

FFF4, FFF4, FFF4

value आएगी

printf 2 → a, *p, *(&a), **(&p)

100, 100, 100, 100

FFFF → FFF4 → 100

optional कभी चेतात तर कधी नाही

* When we are working with pointer in 'c' following warning.

① Suspicious pointer Conversion:
When we are assigning address of Variable into different type of pointer.

```
int a;
char *p;
p = &a;   // suspicious
          pointer
          conversion
```

② Non-portable pointer Conversion :
This warning message we get when we assign a value type of data to the pointer.

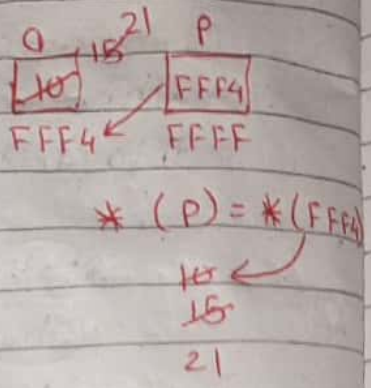
```
int a=100;
int *p;
p=a;      // Non-portable pointer Conversion
```

p=a; → p=&a required.

Page No.:
Date: / /

* Questions on pointer :-

①
```
void main(){
  int a=10, b, *p;
  p = &a;
  printf("\n %d", *p);
  *p = *p+5;
  printf("\n %d", a);
  a = 21;
  printf("\n %d", *p);
  b = *p;
  printf("\n %d", b);
}
```



O/P - 10
15
21
21

* Types of pointer

1. Wild Pointer -

↳ Pointer variable which is not initialized with any variable address or uninitialized pointer variable is called Wild pointer.

↳ Also called bad pointer because without any variable address it will point to some memory location.

```
void main()
{
  int a;
  int *p; // without address is
          wild ptr.
}
```

Page No.:
Date: / /

↳ In implementation when we are working with pointer. it is always recommed to alway initialize with any variable address or make it Null pointer.

2. Null pointer:

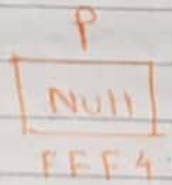
↳ pointer variable initialized with Null. it is called Null pointer

<datatype> * <ptr_name> = Null;

e.g - int * P = NULL

↳ it does Not store Any address at the time of Creating pointer.

int *p = Null;



zero
'0' पण लिहू शकतो पण IDE Typecasting करावे लागेलच

<datatype> * <ptr_name> = (Data_Type *)0;

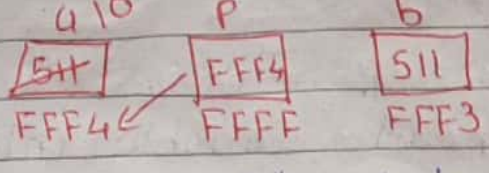
int * p = (int *)0;

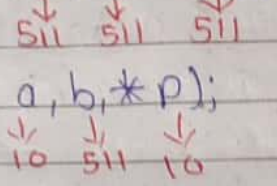
↓

Typecasting

Page No.:
Date: / /

2.
```
void main()
{
  int a,b;
  int *p;
  a = 511;
  p = &a;
  b = *p;
  printf("\n %d, %d, %d", a, b, *p);
  *p = 10;
  printf("\n %d, %d, %d", a, b, *p);
  getch();
}
```

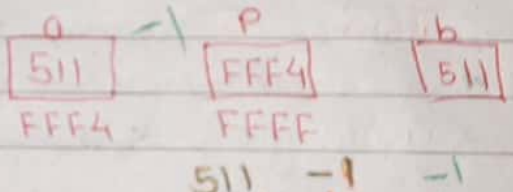




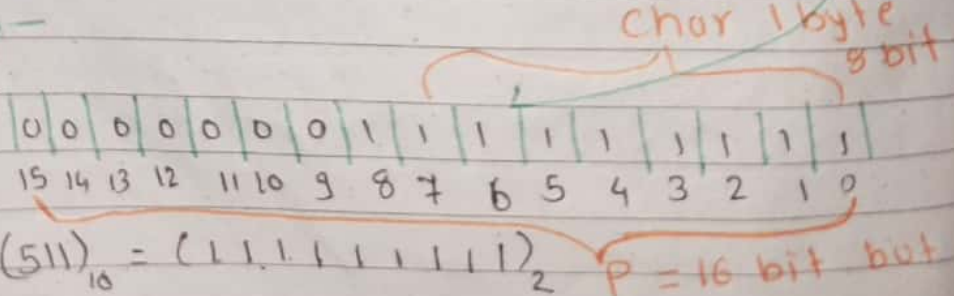
o/p - 511 511 511
10 511 10

3.
```
void main()
{
  int a,b;
  char *p;
  a = 511;
  p = &a;
  b = *p;
  printf("\n %d, %d, %d", a, b, *p);
  *p = 10;
  printf("\n %d, %d, %d", a, b, *p);
  getch();
}
```

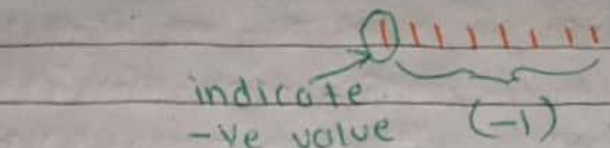
all pointer has 2 byte = 16 bit
but char has 1 byte = 8 bit



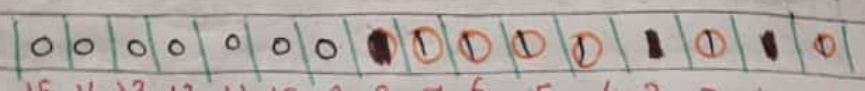
o/p -



$(511)_{10} = (1\,1\,1\,1\,1\,1\,1\,1\,1)_2$ p = 16 bit but



*p = 10; → p 10 केला म्हणून



p char आहे पण a int आहे म्हणून सर्व गोंधळ जाणार

$(1000\ 0101\ 0)_2 = (266)_{10}$

O/P- 511 -1 -1
266 10 10

Note - char आणि दुसऱ्या Datatype च्या size वर खूप फरक पडतो

bit
it

③ Void pointer - Generic pointer.

```
int a=10;
float b=5.2;
char c='A';
void *<ptr_name>
```

```
Void *vp
vp=&a;
vp=&b;   } valid.
vp=&c;
```

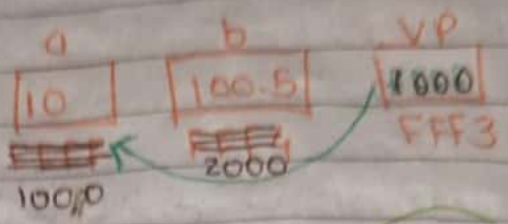
↳ size of void pointer is 2 byte
↳ The generic pointer of C & C++ is void pointer.
↳ it can access & manipulate any kind of data properly.

if you want the value of pointer void *vp void does not get any value so we need type cast

*((int *)vp) or *((float*)vp)

Example

```
void main(){
int a=10;
float b=100.5;
void *vp;
clrscr();
vp=&a;
printf("\n Address of a :%u, value of a=%d",
                    vp,*((int *)vp));
                    Type cast
vp=&b;
printf("\n Address of b :%u, value of b=%f",
                    vp, *((float *)vp));
getch();
}
```



o/p - Address of 1000 value of a = 100
Address of b 2000 value of b = 100.5

④ Dangling pointer

Dangling pointer means any pointer to pointing the address of those pointer who are not exists anywhere.

* → Astric

* Pointer to Pointer.

<datatype> ** <ptr to ptr>

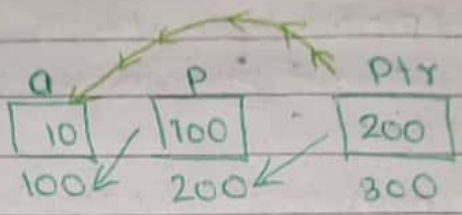
int **ptr; // pointer to integer pointer.

float **fptr; // pointer to float pointer.

↳ it is a concept of holding a ptr address into pointer variable.

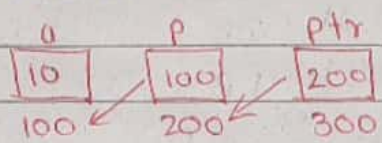
Example:

① void main()
{
int a=10;
int *p;
int **ptr;
p = &a;
ptr = &p;
}



a = 10
*p = *(100) = 10
*ptr = *(200) = 100
**ptr = **(200) = *(100) = 10

② void main(){
int a=10, *p, **ptr;
clrscr();
p = &a;
ptr = &p;



printf("\n %u, %u, %u, %u", &a, p, *(&p), **(&ptr));
100 100 100 *(200) **(300)
*(200)
100

printf("\n %u, %u, %u, %u", &p, ptr, *(&ptr), &ptr);
200 200 *(300) 300
*200

printf("\n %d, %d, %d, %d, %d, %d",
a, *p, *(&a), **ptr, **(&p), ***(&ptr));
10 10 10 10 10 10

O/P – 100, 100, 100, 100
200, 200, 200, 300
10, 10, 10, 10, 10, 10

(16)

* Arithmetic operations on Pointer:

① Address + number ⇒ Generate next address.
↳ address + number * size of (datatype)

int *p = (int x = 100;
① p+3 = 100 +3 * size of datatype (int)
= 100 +3 * 2
= 106
② p-3 = 100 - 3×2
= 94

++ Address → Generate next address
↳ Address = Address + 1
int *p = (int *) 100;
++p = p+1 * datatype size (int)
= 100 + 1 × 2
= 102

② int ptr $\xrightarrow{+1}$ 2B
float ptr $\xrightarrow{+1}$ 4B
char ptr $\xrightarrow{+1}$ 1B
double ptr $\xrightarrow{+1}$ 8B
long double $\xrightarrow{+1}$ 10B.

③ Address1 + Address2
Addr 1 * Addr 2
Addr 1 % Addr 2
Addr1 / Addr 2
} illegal operation.

* Addr 1 − Addr 2 = No. of elements
↳ Add 1 − Add 2 = (Add 1 − Add 2) / Size of Data Type

Example -

```
void main()
{
  int * p1 = (int*) 100;
  int * p2 = (int*) 200;
  printf ("%d", p2 - p1); // p2-p1 = 200-100/2 = 100/2 = 50
  getch();
}
```

o/p = 50

④ We can use relational operator & conditional operator between two address.

Addr1 > Addr2
Addr1 < Addr2
Addr1 == Addr2
} valid.

>=
<=
!=
} valid ✓

⑤ We can not perform bitwise operation between two pointer.

* Array

* ↳ An array is a user define datatype in c-language which is constructed from fundamental datatype of c-language.
* ↳ An array is a collection of similar kind of data values in a single variable
* ↳ When we are working with array always compile time memory management occours
  i.e
  Static memory allocation.

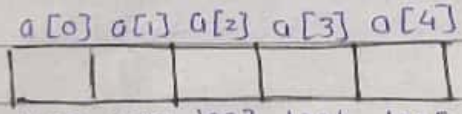
Declaration :

< Data type > < array name > [size] ;

e.g.-

```
int a [5]; // initialization of 1-D array
```

type of element — name of array — Size of array



→ address is always ascending order.

↳ When we are working with array always memory will be constructed in contiguous memory randomly, so that we can access the data in one go.

e.g- ~~a[5];~~ a(0)

↳ When we are working with array all the element share same name with unique identification value called Index or Subscript.

↳ Array indx should start from 0 to size-1
↳ When we are working with array we are required to use array subscript operator [ ]
↳ Array subscript operator requires one argument of type unsigned integer constant.
↳ In Declaration of the array, size must be require to mention.

int a[10]; ✓ must be unsigned integer constant

inta[]; ✗

int a[-5] ✗

* Example -

```
Void main () {
    int a[6];
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    a[4] = 50;
    printf ("%d", a[5]); // Garbage value
    printf ("%d", a[6]); // Segmentation fault error.
}
```

0	1	2	3	4	5
10	20	30	40	50	

Array च्या बाहेरची value Access करायला गेले कि Segmentation error येतो.

Page No:
Date 18/10/22

int a[81000]; ✗ Range 0 - 65535
int a[20000]; ✓
int a[12.4]; ✗
int a[2>3]; ✗

① void main(){
int a[10], i;
clrscr();
for(i=0; i<10; i++)
{
a[i] = i++;
printf("%d", a[i]);
}
}.

o/p - GCC = 0 2 4 .. 10
compiler

Turbo = Garbage value.

0	1	2	3	4	5	6	7	8	9
0	#	2	≡	4	≡	6	≡	8	

↑ ~~garbage~~ value.

② void main(){
int a[10], i;
clrscr();
for(i=0; i<10; i++){
a[i] = ++i;
printf("\n%d", a[i]);
}
getch();
}.

a[0] = 1
a[1] Not defined so error

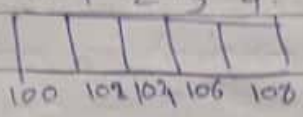
0	1	2	3	4	5	6	7	8	9

o/p- Error

जर आपण वापरतो i तोहला तर i ची value print होणार.

① void main(){
int a[5];
printf("\n %d", &a[4] - &a[1]);
}

o/p = 3



&a[4] - &a[1]

108 - 102 = 6

$\frac{6}{\text{Data type size (int)}} = \frac{6}{2} = 3$

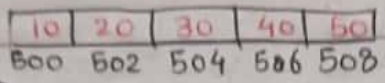
* Initialization of 1-D Array.

<data_type> Array_Name [size] = { List of elements separated by Comma };

e.g Int: int a[5] = {10, 20, 30, 40, 50};

↳ size is optional

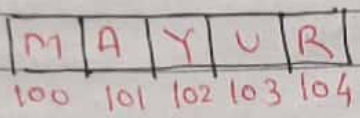
int a[] = {10, 20, 30, 40, 50};



e.g - char

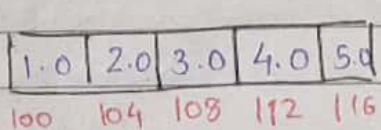
char s[] = {'M', 'A', 'Y', 'U', 'R', '\0'};

'\0' → Null char



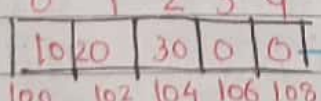
e.g - float -

float a[] = {1.0, 2.0, 3.0, 4.0, 5.0};



* Partial initialization of 1-D Array.

int a[5] = {10, 20, 30};



→ Partial initialization contain '0' zero

e.g - void main(){
int a[5] = {10, 20, 30};
printf("\n %d", a[4]);
}

O/P = 0

Note:

① int a[5] = { };
must be initialized with atleast one value

② int a[3] = {10, 20, 30, 40, 50};
// Error : To many initialization.

③ int a[2];
a[0] = 40 , a[1] = 20, a[2] = 10;

* Relation between 1-D Array & Pointer

int a[] = {10, 20, 30, 40, 50};

a[0]	10	100 ⇒ a+0 = 100+0 = 100
a[1]	20	102 ⇒ a+1 = 100+1×2 = 102 (Datatype size.)
a[2]	30	104 ⇒ a+2 = 100+2×2 = 104
a[3]	40	106 ⇒ a+3 = 100+3×2 = 106
a[4]	50	108 ⇒ a+4 = 100+4×2 = 108.

① print : a : array name ⇒ Base Address = 100

array Name always indicate base address of Array

② increment in address depend on size of (DT)

③ &a[i] = a+i
100 = 100+0 = 100
100 = 100

④ a[i] = *(a+i) जर आपल्याला Array चा address सहीत value पाहिजे असेल तर *(a+i)

a[i] = *(a+i)
100 = *(100)

a[i] = 10

⑤ a[i] = i[a] बरोबर ✓

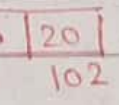
⑥ if a is array & pointer question. then

++a
a++
--a
a--
} Invalid Error.

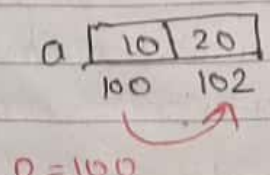
We can not modify Base address of array in pointer.

++a = a+1
= 100+1×2
= 1022 Not Allowed.

⑦ int *p;
int a[] = {10, 20};
p = a; // 100
p++;



Array to pointer मध्ये पुन्हा Address ची location बदलू शकतो



∴ =p+1
=100+1×2
=102

OT ↓

Q1
```
void main (){
int a[] = { 10, 20, 30};
int i;
for (i=0; i<3; i++)
{
printf ("%d, %d", a[i], *[a+i]);
}
getch();
}
```

10	20	30
100	102	104

a[i] → 10 ; *[a+i] → *(100)

10, 10

20 → *(a+1) → *(100+1*2) → *(102)

20, 20

30 → *(100+2*2) → *(100+4) → *(104)

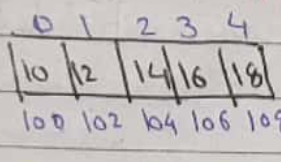
30, 30

o/p = 10, 10
20, 20
30, 30

* Accessing element of Array using pointer.

①
```
void main(){
 int a[] = {10,12,14,16,18}, i;
 int *p;
 clrscr ();
 p = a;
 for (i=0 ; i<5 ; i++)
 {
    printf ("\n %d ", *p);
     ++p;
  }
}
```
= access value using pointer.

o/p - 10, 12, 14, 16, 18.



① *p = *(100) = 10 — 10
② *p = *(10+1*2) = *(102) = 12 — 12
③ *p = *(10+2x2) = *(104) = 14 — 14
④ *p = *(10+3x2) = *(106) = 16 — 16
⑤ *p = *(10+4x2) = *(108) = 18 — 18

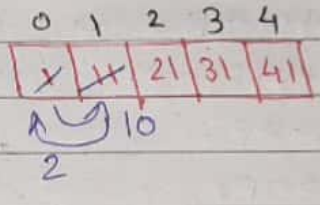
NOTE:

① ++p = pre inc.. in pointer / Generate next address (Jump in next location)
② p++ = post inc. in pointer / Generate next address (jump in next location)
③ --p = pre dec. in pointer / Generate previous address (Jump in previous location)
④ p-- = Post dec. in pointer / Generate previous address (jump in previous location)

⑤ ++*p = pre inc. of value
⑥ (*p)++ = post inc. of value
⑦ --*p = pre dec. of value
⑧ *p-- = post dec. of
⑨ *++p = pre incr. of pointer then access data.
⑩ *p++ = first access the data then post inc. of pointer
⑪ *--p = pre dec. of pointer the acc. data
⑫ *p-- = first access the data then post dec. of pointer.

problems -

① #include<stdio.h>
int main(){
int a[] = {1, 11, 21, 31, 41};
int *p;
p = a;
++p; ✓
--*p; ✓
--p;
++*p;
printf("\n %d, %d", a[0], a[1]);
return 0;
}

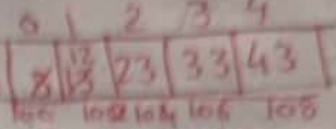


o/p - 2, 10

② void main(){
int a[] = {2, 13, 22, 32, 42, 52};
int *p;
p = a;
++p;
++*p;
--p;
--*p;
printf("\n %d, %d", a[0], a[1]);
return 0;
}

o/p - 1, 13

3. Void main() {
int a[] = {3, 13, 23, 33, 43, 2};
int *p = Null;
p = a+1; // 102
--p; // 100
++*p; // 4
++p; // 102
--*p; // 12
printf("\n %d, %d", a[0], a[1]);
}

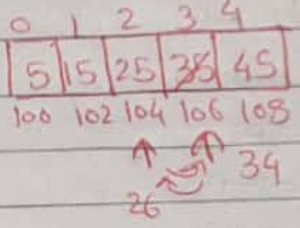


p = 10 + 1 * 2 = 102

4 12

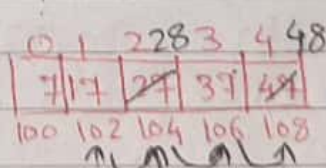
O/p = 4, 12

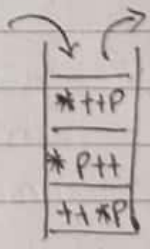
4. Void main() {
int a[] = {5, 15, 25, 35, 45};
int *p = a+2;
++p;
--*p;
--p;
++*p;
printf("\n %d, %d, %d", a[0], a[3], a[2]
}



O/p - 5, 34, 26.

5. void main() {
int a[] = {7, 17, 27, 37, 47};
int *p = a+1;
++p;
++*p;
printf("\n %d, %d, %d", ++*p, *p++, *++p);
}





48 37 37

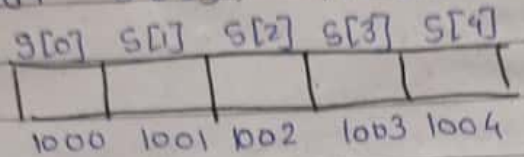
O/p - 48, 37, 37.

* String

* String : it is a sequence of character.

Declaration -

```
char arrayName [size];
char s[5]; // char array
```



→ char array is also known as string because char array is a group of character.

→ String I/O

There are Two predefine function for I/O:

① gets (); → get the i/p string

② puts (); → put string on o/p.

Syntax:

```
char s[10];
gets(s);
puts(s);
```

scanf ("%d", s); ⇒ Not allowed because scanf mayur_chipade space दिल्याशिवाय पुढे जात नाही.

But

scanf ("%[^\n]s", s) = gets()

आधी लिहलेले कि वाचते.

① void main() {
char s[10];
clrscr();
printf("\n Enter your Name:");
gets(s);
puts(s);
getch();
}

o/p:- Enter your Name: Mayur Satish Chipade

② void main() {
char s[] = "knowledgegate";
printf("\n %s", s); // knowledgegate
printf("\n %s", s+2); // owledge gate
printf("\n %s", s+4); // ledge gate
printf("\n %s", s+8); // egate.
}

o/p -

k	n	o	w	l	e	d	g	e	g	a	t	e
100	101	102	103	104	105	106	107	108	109	110	111	112

* Uninitialized Char array
char s[5];

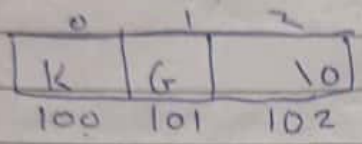
* initialized char array:
a) char s[] = {'M', 'A', 'Y', 'U', 'R'}
b) char s[] = "MAYUR"; // String.

↳ in this case C-compiler automatically insert '\0' character at the end of String.

Null char = '\0' : End of string
ASCII value = 0

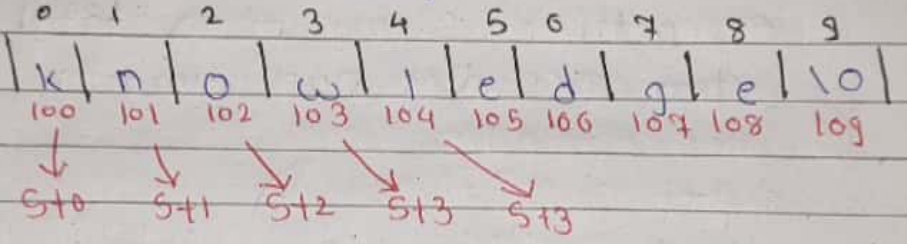
* partial Initialization :

char S[5] = "KG";



* Relation between String and pointer

char s[] = "knowledge";



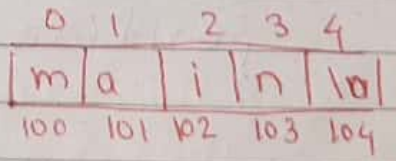
* & s[i] = s+i
* s[i] = *(s+i)
* C - Compiler automatically convert s[i] to *(s+i).

User friendly → machine friendly.

s[i] = *[s+i] = *(i+s) = i[s].

Example

①
```
void main(){
  char s[] = "main";
  int i;
  for(i=0; s[i]!='\0'; i++)
  printf("\n %c, %c", s[i], *(s+i));
}
```



s[0] *(100+0) *(100) → m

o/p - m, m
a a | n n

Page No:
Date: / /

* String Library function -

1. strlen ()
2. strcpy ()
3. strcat ()
4. strrev ()
5. strcmp ()

} string.h

(1) strlen() -

↳ it is a predefine function of string.h
↳ We can find length of string by using strlen function.
↳ Length of string mean total no of char excluding null character.

e.g -

```
chars[] = "Knowledge Gate";
strlen (s);     // 14
strlen (s+2);   // 12
strlen (s+5);   // 9
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
k	n	o	w	l	e	d	g	e		G	a	t	e	\0

```
#include<stdio.h>
void main(){
printf("\n %d, %d", sizeof("ABC \n\n"), strlen("ABC\0"));
}
```

sizeof("ABC \n\n") / 6 , strlen("ABC\0") / 3

Size of = No. of char + 1 (or. allocated block) = 5 + 1 = 6

Without strlen(); function.

```
#include <stdio.h>
void main(){
char s[]="Knowledge Gate";
int count=0;
int i;
for(i=0; s[i]; i++)
{
    count++;
}
printf("length of string = %d", count);
}
```

o/p - 14.

② strcpy() -

↳ it is used to copy one string from one character array to another character array.

Syntax : strcpy(destn, source);

copy.

```
s1="MC";
s2="  ";
strcpy(s2,s1);
```

e.g -
```
char s1[]="KG";
char s2[5];
s2=s1;   // error
```

↓

Base address change नहीं कर सकते

↓use

```
strcpy(s2,s1);
```

Without strcpy

① void main(){
char s1[]="ABC";
char s2[5];
int i;
for (i=0; s1[i]; i++)
{
s2[i]= s1[i];
}
s2[i]='\0';
printf ("\n %s", s2);
}
o/p- ABC

② void main (){
char s1[]='ABC');
char s[];
int i;
for (i=0; s1[i]; i++);
{
s1[i]=s[i]; // s is not initialized
s मध्ये काहीच value नाही
}
printf ("\n %s", s1);
}
o/p Segmentation fault

③ char s1[] = "Mayur";
char s[];
~~int i;~~
~~for(i=0; s1[i]; i++);~~
strcpy (s, s1);
printf ("\n %s ", s);
}
o/p - mayur.

③ strcat():
↳ used to concatenate two string.
Syntax : strcat(destination, source);

e.g - char s1[10] = "PQR";
char s2[5] = "XYZ";
strcat(s1, s2);

	0	1	2	3	4	5	6	7	8	9	10
s1 =	P	Q	R	X	Y	Z	\0				
s2 =	X	Y	Z	\0							
	0	1	2	3							

④ strrev():
↳ used to reverse a string except null character:
Syntax: strrev(s)

```
void main()
 {
   char s[] = "ABCD";
   strrev(s); // DCBA
   printf("\n Reverse string= %s", s)
   getch();
}
```

5 strcmp:
↳ used to compare two strings.
syntax : int i = strcmp(str1, str2)

+ve -ve 0

* if i>0 +ve = str1>str2
* if i<0 -ve = str1<str2
* if i=0 = str1 = str2

* e.g

① s_1 = "a a" Compare ASCII value
s_2 = "a b" a = 97 a - b
0 -1 b = 98 97 - 98 = -1
-1
i = strcmp(s1, s2);
= -1 (-ve
s1 < s2

Compare करते length wise compare होत नाही character च्या ASCII value ने compare होते.

② s_1 = "a a"
s_2 = "a a"
i = strcmp(s1, s2)
= 0
s1 = s2

* String Reverse without using strrev();

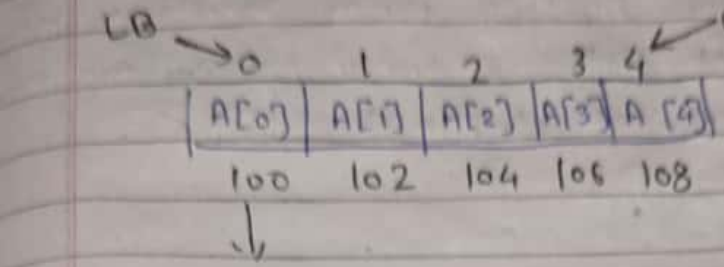
```
#include <stdio.h>
#include <string.h>
void main(){
    char s[] = "ABCD", t;
    int i, l;
    l = strlen(s);          // 4
    for(i=0; i<l/2; i++)    // i=0
    {
        t = s[i];           // A
        s[i] = s[l-1-i];    // s[(4-1-i)=(3) s[3]
        s[l-1-i] = t;
    }
    printf("\n %s", s);
}
```

0	1	2	3	
A	B	C	D	\0

O/P // DCBA

Basic of 1D array.

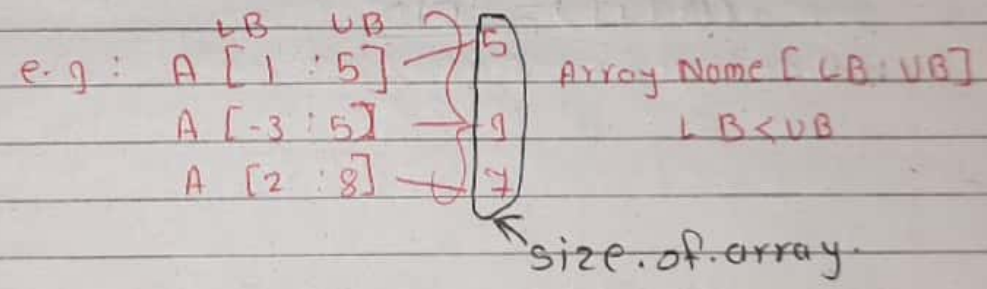
↳ array is a similar kind of DataType.



Base address: array name A
for accessing array.

→ Lower Bound (LB) = 0
→ Upper Bound (UB) = 4

Size of array = 4 - 0 + 1
= UB - LB + 1
=



* Finding Location of an Array / Address

$$Loc(A[i]) = Base\ address + w \times (i - LB)$$

BA = 200 , w = 2 Byte

datatype
int, char, float
2 1 4

① A [0 : 4] find = A[2] ?

LB = 0 UB = 4

Array Size = 4 - 0 + 1 = 5

Loc (A[1]) = 200 + 2 (2 - 0)
= 200 + 2 (2)
= 200 + 4

A[2] = 204

Page No:
Date: / /

Q Consider an array A[-2:15], which is stored in memory. starting from location 1000 Each element take 2 location in memory The location of element A[3] is?

A[3] = ?

LB = -2 UB = 15 , BA = 1000

$\omega = 2$

$Loc(A[i]) = BA + \omega(i - LB)$ → Relative index

$= 1000 + 2 \times (3 - (-2))$

$1000 + 2 \times 5$

$1000 + 10$

$(A[i]) = 1010$

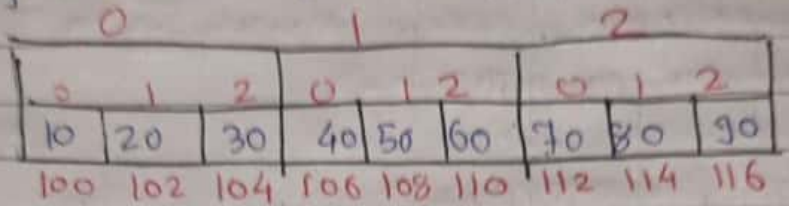
2-D array

Date: 19/10/2022 99

★ ↳ Collection of array.

row ↓ column ↓

e.g int a[3][3] = {(10,20,30),(40,50,60),(70,80,90)};



↳ arr name = a = base address of 0th row = 100

↳ arrayname+1 = a+1 = Base addr of 1st row = 106

↳ ——— 11 — a+2 = ——— 11 — 2nd row = 112

Note- in matrix there are two ways to store the values.

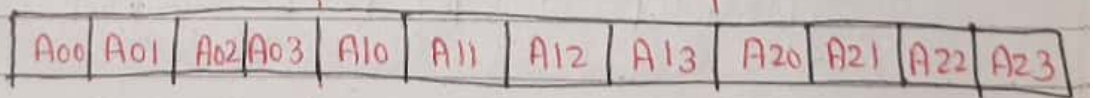
① RMO (Row Major Order)

② CMO (Collomn Major Order)

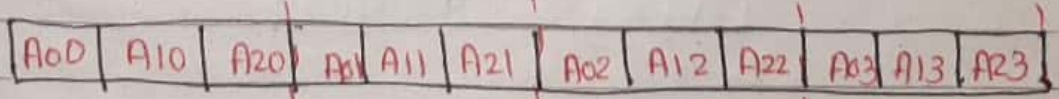
①

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23

① RMO - Row major order मध्ये पहीले Row येतात



② CMO - Collomn major order मध्ये पहीले column येतात.



100

RMO Location formula -

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23.

Loc(A[i][j]) = BA + w[(i - LBi)n + (j - LBj)]

Most IMP In C

*** Relation Between 2-D array & pointer.

Syntax -

<Data type> arrName [] [size] = {List of element seperated by ','}

(row is optional; size = column)

e.g- int a[][3] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};

	0			1			2		
	0	1	2	0	1	2	0	1	2
	10	20	30	40	50	60	70	80	90
	100	102	104	106	108	110	112	114	116

Partial initialization of 1D array

e.g- int a[3][3] = {1, 2, 3, 4, 5};

	0			1			2		
a	0	1	2	0	1	2	0	1	2
	1	2	3	4	5	0	0	0	0

e.g - float a[ ] [5] = {{1.0}, {2.0}, {3.0}}

0	1	2	0	1	2	0	1	2
1.0	0.0	0.0	2.0	0.0	0.0	3.0	0.0	0.0

Note

* Increment in Address always depends on size of element.

int → 2B
float → 4B
char → 1B

formula * &a[i] [j] = a[i]+j → address of i^{th} row & j^{th} column.

formula * a[i] [j] = *(a[i]+j)
= *(*(a+i)+j) - value of i^{th} row & j^{th} column

formula 3-D array * a[i] [j] [k] = *(a[i] [j]+k)
= *(*(a[i]+j)+k)
= *(*(*(a+i)+j)+k)

Page No.
Date / /

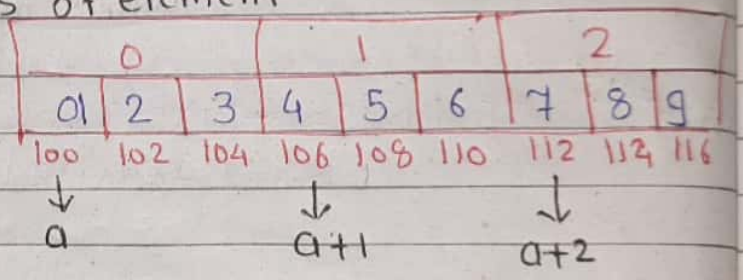
Problems :

①
```
void main(){
int a[3][3]={{1,2,3},{4,5,6},{7,8,9}},i,j;
printf("\n %u, %u, %u", a, a+1, a+2);
printf("\n address of element \n");
  for(i=0; i<3; i++)
  {
    printf("\n");                    //Nested for loop
     for(j=0;j<3;j++)
     {
       printf("\t %u", a[i]+j);
     }
  }
}
```

3.

o/p - 100, 106, 112

Address of element



100 102 104
106 108 110
112 114 116

$$
\begin{cases} a[0]+0 = 100+0 = 100 \\ a[0]+1 = 100+1*2 = 102 \\ a[0]+2 = 100+2*2 = 104 \end{cases} \quad 0
$$

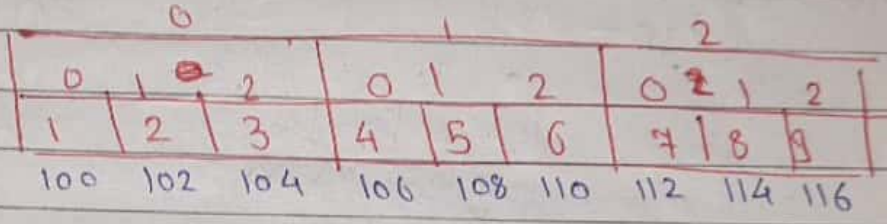
$$
\begin{cases} a[1]+0 = 106+0 = 106 \\ a[1]+1 = 106+1\times2 = 108 \\ a[1]+2 = 106+2\times2 = 110 \end{cases} \quad 1
$$

$$
\begin{cases} a[2]+0 = 112+0 = 112 \\ a[2]+1 = 112+1\times2 = 114 \\ a[2]+2 = 112+2\times2 = 116 \end{cases} \quad 2
$$

(i, j; * 2 : data type)

② void main(){
int a[][3] = {{1,2,3},{4,5,6},{7,8,9}};
for (i=0; i<3; i++)
{
printf("\n");
for (j=0; j<3; j++)
{
printf("\t %d", *(*
}
}

o/p -



1 2 3
4 5 6
7 8 9

③ void main(){
int n[25];
n[0] = 10;
n[24] = 200;
printf("\n %d, %d, %d, %d", *n, *(n+24), *(n+0));
}

*(n) → *(0) → 10
*(24) → 200
*(0) → 10

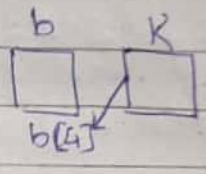
o/p - 10, 200, 10

4. void main()
{
int b[] = {10, 20, 30, 40, 50};
int i, *k;
k = &b[4] - 4;
for(i=0; i<=4; i++)
{
printf("\n%d", *k);
k++;
}

o/p =

0	1	2	3	4
10	20	30	40	50
100	102	104	106	108

k



k = &b[4] - 4 (4 → data type value)
= 108 - 4 × 2
= 108 - 8
k = 100

10, 20, 30, 40, 50

5. void main() {
char a[] = "knowledgegate";
char *b = "knowledgegate";
printf("%d, %d", sizeof(a), sizeof(b));
printf("\n%d, %d", sizeof(*a), sizeof(*b));
getch();
}

o/p - 14, 2
12, 11

कोणताही pointer 2 byte memory खातो.

sizeof(*a) — 1 byte ('k') — 1 char

sizeof(*b) — 1 byte (*100 address) — 1 byte (k) — 1 char

Problem on Array.

Page No.:
Date: / /

105

6. Void main() {
int a[] = {12, 14, 15, 23, 45};
printf ("\n %u, %u", a, &a);
printf ("\n %u, %u", a+1, &a+1);
getch();
}

O/P

12	14	15	23	45
100	102	104	106	108

100, 100
102, 110

&a+1
↓
100+1
100 + size of arry
Base addr + Size of array
100 + 10
110

7. void main() {
int a[][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
printf ("\n %u, %u, %u", a, a+1, &a+1);
}

O/P -

0			1			2		
1	2	3	4	5	6	7	8	9
100	102	104	106	108	110	112	114	116

a = 100
a+1 = 106
&a+1 = 118 · &a+1 = a+1 × size of 2D array

106

8. void main(){
int a[] = {0,1,2,3,4};
int *p;
for (p=a; p<&a[4]; p++)
printf("\n %d", *p);
getch();
}

0	1	2	3	4
0	1	2	3	4
100	102	104	106	108

o/p - 0
1
2
3
4

p = 100

100 <= 108 ✓ = 0
102 <= 108 ✓ = 1
104 <= 108 ✓ = 2
106 <= 108 ✓ = 3
108 <= 108 ✓ = 4

9. void main (){
int a[] = { 0, 1, 2, 3, 4 };
int *p;
for (p = a+4; p >= a; p--)
printf("\n %d", *p);
}

0	1	2	3	4
0	1	2	3	4
100	102	104	106	108

o/p - 4
3
2
1
0

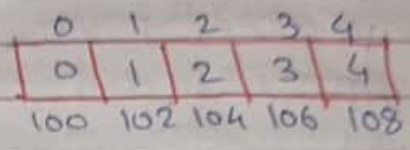
p = a+4
p = 108

p >= a
108 >= 100 ✓

p--

p = 106 *p
p = 104 4
p = 102 3
p = 100 2
1
0

10 void main(){
int a[] = {0,1,2,3,4};
int i, *p;
for(p=a+4, i=0; i<=4; i++)
printf("\n %d", p[-i]);
getch();
}



3.
O/P - 4
3
2
1
0

① p = a+4 = 108
② i = 0
③ p[-i] = *(p-i) = *(108) = 4

*(p-i)
a[i] = *[a+i]
a[-i] = *[a-i]

④ i++

② *(p-i) = *(p-1) = *(108-1×2)
= *(108-2)
= *(106)
= 3

11 void main(){
int a[] = {0,1,2,3,4};
int *p;
for(p=a+4; p>=a; p--)
printf("\n %d", a[p-a]);
getch();
}

a[p-a]
*(a+p-a)
*(p)

3.

O/P - 4
3
2
1
0

B.S = Base Address

108

12. void main(){
int a[] = {0, 1, 2, 3, 4};
int *p[] = {a, a+1, a+2, a+3, a+4};
int **ptr = p;
printf("\n %u, %d", a, *a);
printf("\n %u, %u, %d", p, *p, **p);
printf("\n %u, %u, %d", ptr, *ptr, **ptr);
getch();
}

O/P - 100, 0
200, ~~300~~ 100, 0
300, 200, 100

Step-1 O/P -

a =

0	1	2	3	4
100	102	104	106	107

B.S

Step-2 जर प्रत्येक location वा address store करायचा असेल तर pointer वा Array करावा लागतो.

*p[] = {a, a+1, a+2, a+3, a+4};

*p =

100	102	104	106	108
200	202	204	206	208

B.S

Step-3

**ptr = p;

ptr

200
300

Step-4 printf("\n %u, %u", a, *a);
100 0

Step-5

printf("\n %u, %u, %d", p, *p, **p);
200 100 0

Step-6 printf("\n %u, %u, %d", ptr, *ptr, **ptr);
300, 200, 100

string question / character array.

13 void main ()
{
printf ("9+"Knowledgegate");
}

0 1 2 3 4 5 6 7 8 9
B.S

o/p = g

B.S
9+100
109

14 void main(){
printf("Knowledge gate"[5]);
}

0 1 2 3 4 5
100
100+5

o/p - e

15. void main (){
char ch[20];
int i;
for (i=0; i<19; i++)
*(ch+i)=67;
*(ch+i)='\0';
printf("\n %s", ch);
}

o/p -

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ch C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	10

ASCII value of 67 = C (capital C)

o/p - CCCCCCCCCCCCCCCCCCC
19

16. Void main(){
    printf("%c", "abcdefg"[4]);
    }

String ↓

याचा अर्थ असा पण होतो
*(s+4) / s[4]

Olp -
e

17. Void main(){
    char s[] = {48,48,48,48,48,48,48,48,48,48};
    char *p;
    int i;
    p=&s;
    for(i=0; i<=9; i++)
    {
        if(*p)
        {
            printf("%c", *p);
            p++;
        }
    }
    }

Olp - 0000000000

48, 48, 48, 48, 48, 48, 48, 48, 48, 48.
ASCII value of 48 = 0

Note- character char Array आहे आणि त्यात, integer value आहे म्हणून तो ASCII value घेतो.

18
```
void main(){
  char s1[] = "Hello";
  char s2[] = "Hello";
  if (s1==s2)
     printf("Equal");
  else
     printf("Unequal");
}
```

Olp - Unequal

Because Base address are change in two String that's whay this is unequal.

19
```
void main(){
  char s[7] = "String";
  printf("\n %s", s);
}
```

Olp- Garbage value ~~String~~

20
```
void main(){
  char *str[] = {"sb", "moh", "maya", "hai"};
  printf("%d, %d", sizeof(str), sizeof(str[0]));
}
```

Olp- 8, 2

2	sb	← str[0] = 2
2	moh	
2	maya	
2	hai	

8

Page No.
Date 27/10/22

21. void main () {
char *m[] = {"some love one", "Some low two",
"I love one", "That is you"};
printf ("\n %d, %d", sizeof(m), sizeof(m[0]));
}

8 2

o/p - 8, 2

Some bve lone, Some love two, I love one, That is you, = 8 byte

2 byte.

22

22 #include<stdio.h>
void main() { array of pointer
char *S[] = {"ice", "green", "come", "please"};
char **ptr[] = {S+3, S+2, S+1, S};
char ***p = ptr;
printf("\n %s", **++p);
printf("\n %s", *--*++p+3);
printf("\n %s", *p[-2]+3);
printf("\n %s", p[-1][-1]+1);
}

o/p -

= = =

i	c	e	\0	g	r	e	e	n	\0	c	o	m	e	\0	p	l	e	a	s	e	\0
300	301	302	303	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

*S

300	304	310	315
100	102	104	106

*ptr

106	104	102	100
200	202	204	206

o/p = come
Blank
ase
reen

*** p=ptr 200

① prinf = **(++p) 200 → 202 104 310 come
++ * *

② printf = *--*++p+3
++p = 204 102 100 300 = 300+3×1
* - *
= 303 \0 आसल्याने बंद

③ *p[-2]+3)
*p[-2] = **(p-2) = 204 = 2×2 = 200
200 106 315 = 315+3 = 318 = ase
* *

④ p[-1][-1]+1;
((p-1)-1)+1 = 204 - 1 = 202 104 102 304 305
* -1 * +1
reen

Page No.
Date 30/10/22

* Array of pointer

<DT> * <arr_name> [size];
eg * a[5];

* pointer to array:
used for 2D array

<DT> (*ptr_name) [size];
e.g- int (*a) [5];
↳ pointer 'a' pointing to array of 5 element

Example -

```
#include <stdio.h>
void main(){
 int a[][3]={{10,20,30},{40,50,60},{70,80,90}};
 int *p;
 int (*q)[3]; → pointer to array.
 p=a;          p [100] address of a
 q=a;          q [100]
 printf("\n %u, %u", p, q); // 100, 100
  p++; //102
  q++; //106
  printf("\n %u, %u", p, q); //102 106
}
```

a/p - 0 1 2

0	1	2	0	1	2	0	1	2
10	20	30	40	50	60	70	80	90
100	102	104	106	108	110	112	114	116

p++
q++

Note - pointer to array 2D matrix मध्ये use करतात आणि ++ लावा तो पुढचा Row मध्ये जातो

Function -

* Theory
  - ↳ function is a subprogram which is designed for a specific task is called function.
  - ↳ By using function we can design the application in module format
  - ↳ When we are designing application in module format then we can easily debug the program.
  - ↳ By using function we can Reduce coding part of application.
  - ↳ By using function we can keep track what they are doing.
  - ↳ Basic purpose of function is Reusability
  - ↳ A C program is a collection of function.
  - ↳ From any function we can call any other function.
  - ↳ always compilation process will take place from top to bottom.

* Example

①
```
Void show ();   // function Declaration
void main ()
{
 clrscr();
 show();   // function call  { function defination
 getch();                   { after calling
}
void show()  // function defination
{
 printf("welcome to kg");
}
```

2.
```
void show()
{
  printf("Welcome to Pune");
}
void main()
{
  clrscr();
  show(); // function calling
  getch();
}
```
↳ function defination before calling

↳ Execution process will start from main function & ends with main function.

↳ When we are calling a function which is define after calling; for avoiding the compilation error we are required to go for forward declaration i.e prototype is required / function Declaration.

↳ Declaration of function:
means need to mention return type name of function of parameter type information.

↳ Function declaration provides incomplete details about the function to compiler

↳ Function defination provide complete details about the function to the compiler.

↳ Function defination, 1st line is called function declaration or function header

↳ Always function declaration should be matched with function declaration

* Syntax -

```
return_type function_name(type of argument); // Declaration
void main()
{
 function_name(arg); // function calling.
}
return type function_name (arg) // defination
{
 :
}
```

↳ Return type parameter and Return type statement are optional

↳ In implementation whenever a function is Not returning any values, then specify the Return type as void.

↳ void means Nothing. i.e no return values.

↳ Default parameter of any function is void

↳ In C-programming function are classified into 2 types.

1. Library function or Built in function
2. User defined function

* Library function - inbuilt function implemented
  - These function are set of pre-def function which is available with compiler
  - The implementation part of pre-define functions are available in machine readable format.
  - In implementation when we are working with pre-define function, for avoiding the compilation error, we require to go for forward declaration by using header files.
  - (.h) file does not contain any implementation part of predefined function i.e, it contain forward declaration only.

Limitation of pre-define function :
- all pre-define function contain limited tasks only i.e, for what purpose function is designed for some purpose, it should be used.
- As a programmer, we do not have any control on predefined function because coding is available in machine readable format.
- As a programmer, we cannot change or modify the behaviour of any predefined function

for e.g- printf(), scanf(); getch(), etc

User defined function -

↳ As per client Requirement or as per the user requirement or project requirement if any function is implemented by programmer. Those are called used defined function. As a programmer we have full control on user define function because coding part is available in user readable format.

↳ as a programmer, whenever we need to change the behaviour of any user define function, then it is possible to change.

Rules :-

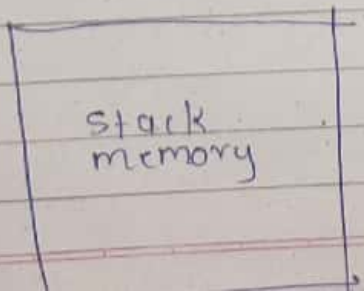
① When a function is define after calling then it must require function declaration before calling.

② A function definition is given before calling in the program, then we can skip the function declaration.

③ A function is declare & defined in the program but called then compiler show no error.

④ if function is declared & defined in the program but not called then compiler show no error

⑤ if function is declare & called but not define then compiler shows error.



stack memory ~~HEO~~ function Buntat.

* local Variable, Global, Free variable

(1). Local variable :-

↳ a variable that is declare inside the function or block is called a local variable.

↳ it must be declare at the start of the block

```
void function1(){
    int x=10;
}
```

2. Global Variable -

↳ A variable that is declare outside the function or block is called a global variable.

↳ Any function can chang the value of the global variable. it is available to all the function.

↳ it must be declare at the start of the block.

```
int value=20;  // global variable.
void function1(){
 int x=10;     // local variable.
}
```

↳ Global variable by default contain a value.

3. Free Variable -