gcc a.c -o hello

.\hello.exe

Page No.
Date 19/08/22

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("my world");
    getch();
}
```

preprocessor directive → Header lines

void → return type

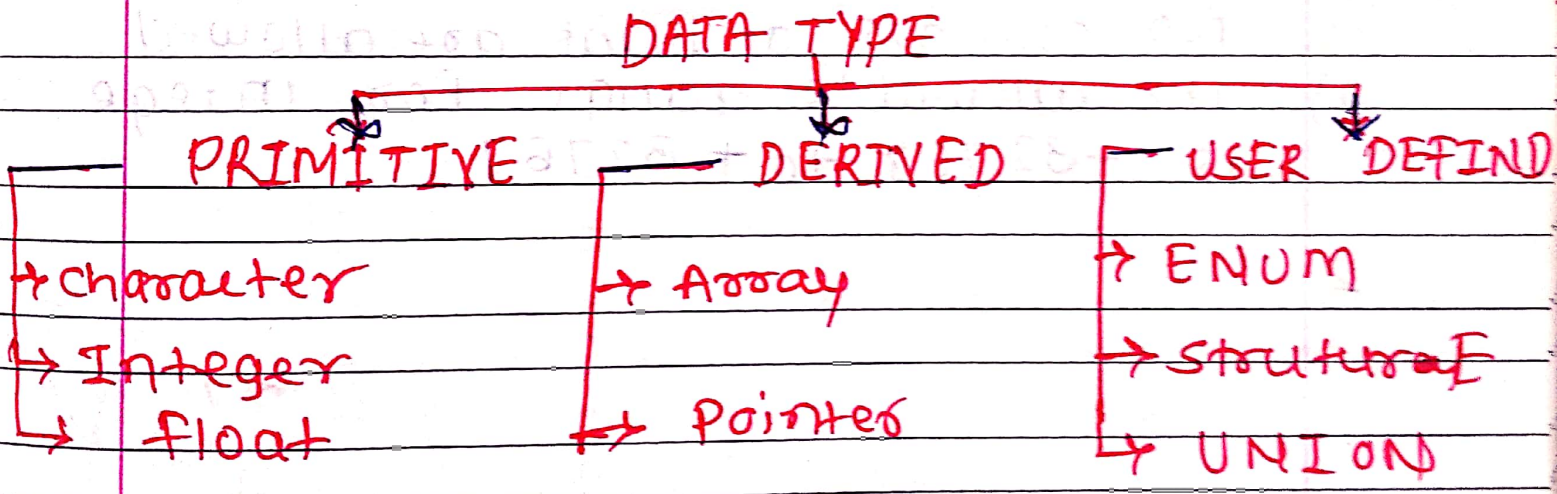
main() → standard funct.

clrscr(); → function

printf("my world"); → instruction terminator

20/08/22.

Leetcode 8)



every Datatype divide in ① variable ② Constant

*Rules for constructing variable Names

① A variable name is any combination of alphabates, digit or underscore. Some compiler allow variable name with maximum length upto 31 characters and some compiler allow upto 247 characters.

Page No.
Date

② The first character in variable name must be an alphabate or underscore.

③ No commas or blank are allowed within a variable name

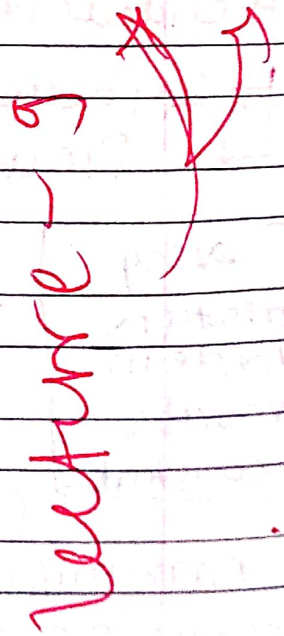
④ No special symbom other than an underscore can be used in variable name

* Rules for constructing Integer constants.

~~① An integ~~

1) An integer constant must have at least one digit.
2) It must not have a decimal point.
3) No commas or blank are allowed.
4) The allowable range for intege is -32768 to +32767

*



* Rule for constructing Real constant or floating point constant.

1) A real constant must have at least on digit
2) It must have decimal point.
3) It could be (+ve) or (-ve)
4) No commos or blank are allowed.

* Rules for representing real constant in exponential form.

1) The "M" part & "E" part should be seperated by a letter 'e or E'.
2) It may have (-ve) or (+ve)
3) At ~~test~~ least one digit.
4) Range is $-3.4e38$ to $+3.4e38$

mEE → exp. one
↓
mantissa

* Lecture - 10

* Rules for ~~define~~ constructing character constant

Page No.
Date

1) A character const. is single alphabate, single digit, single sysmbol enclosed within single inverted commas.

2) Both inverted commas should point to Left.

3) The maximum length of character constant can be 1 character.

LECTURE-11

* format Specifier / Place Holder %d. → for integer

LECTURE-13

Datatype	key word	size in Bytes	format Specifier	Range
Integer	int	2Bytes (16-bit compiler) eg TurboC++ / 4 bytes (32-bit compiler) e.g. VS code	%d	-32768 to +32767 (If 16 bit) / -2147483648 to +2147483647 (If 32 bit)

Datatype

float	float	4 bytes	%f	-3.4e38 to +3.4e38 ((-3.4e*10^38 to +3.4*10^38))

*LECTURE-14

float - Datatype

(i) if a = 5.123456789 then By default → it shows upto 6 decimal
i.e 5.123456
& if printf("a=%.3f", a) then
if →
printf("a=%.3f", a) then
a = 5.123

(ii) after (i)
printf("a=%.5f", a) then →
a=5.12345

point number pahije tikte point ghetli shaktat i.e "%.7f"

LECTURE 16 — TYPE-CONVERSION

Implicite Type conversion (complies स्वतःच काम करतो)

- Promotion
- Demotion (float ची value int मध्ये convert होताना .0 (point) च्या समोर च्या value ~~ingnore~~ ignore)

Explicite Type conversion (आपल्याला काम करावं लागते)

→ Typecasting in some specific way.

LECTURE-17

* Rules that are used for implicite of floating point & integer value in C.

(1) An arithmetic operation betn int / int then result is in int.

(2) —"— betn float / float then result is in float.

(3) —"— betn int / float then result is always in float (real number). (In these process comp. itself promote integer in real number.)

LECTURE-18

EXPLICIT TYPE CONVERSION

when,

```
int a=5, b=2;
float c,d;
c = a/b = 2.
```

but

```
→ d = (float)a/b = 2.500000
```

LECTURE-19 Re-usability of variable

(can use variable after it finish first work)

i.e int a, b, c;

c = a + b; → c = 4

printf(c);

c = c/2 → c = 2

printf(c);

LECTURE-20 - scanformant -

① scanf("%d", &a); (Here & means ampersand)

② How ~~memory~~ (Bytes) affect/use memory.

LECTURE-23 - (float) - type

printf("Enter your mark");

scanf("%d %d %d", &a, &b, &c);

printf("Your Result is", (float)((a+b+c)/300*100)); ⇐

or ((a+b+c)/300.0*100); ⇐

Lecture-24 - swapping value

int a, b, x;

I/P → Enter value of a, b. (Printf & scanf ~~काम~~.))

I/P → x = a; (Here x = [ ] (empty)

a = b;

b = x;

so value interchange (swap) होगा।

LECTURE-25 (4/may)

~~swap~~ swapping number without using 3rd variable

void main;

{

```
int C,D;
printf("(Enter two numbers)");
scanf("%d %d",&C,&D);
C = C+D;
D = C-D;
C = C-D;
printf(——);
```

(see as a compiler) [illegible]

eg:-
C=4, D=5
∴ C = C+D = 9
D = C-D = 9-5 = 4
C = C-D = 9-4 = 5

LECTURE 26

- Garbage value :- **Define** :- By default there is an unpredicatable value is present in variable of some storage classes. (Random value)

Lecture 27

01/09/2003 → ① To print each digit entered by user in seperate line

② Least significant digit to most significant digit

eg. 771 → 1, 7, 7	eg. 417 → 7, 1, 4	eg. 120 → 0, 2, 1

Program

① →
```
void main()
{
int n;
printf("Enter three digit number")
scanf("%d",&n);
printf("\n%d",n%10);
n=n/10;
printf("\n%d",n%10);
n=n/10;
printf("\n%d",n%10);
getch();
}
```

* modular div. [illegible] reminder [illegible] result [illegible] i.e. 10%7 = 3

((modular division operation [illegible])) i.e. 234%10 = 4

((आता n=n/10 होतोय, [illegible] rule [illegible] int/int=int [illegible] 234/10 = 23))

→ contin:

* Modular division [illegible] no. [illegible] divide [illegible] remainder [illegible] no. [illegible] i.e. 2%10 = 2

LECTURE NO. 28

Que.) If five digit no. is input through keyboard, wap to calculate sum of it's digit.

```
void main()
{
  int main n, sum=0;
  printf(enter 5 number);
  scanf("%d", &n);
  sum = sum + (n%10);
  n = n/10;
  sum = sum + (n%10);
  n = n/10;
  sum = - - - -
  - - - - -
  - - - - -      3 more time
  - - - - -
  - - - - -
  sum = sum + (n%10);
  printf("\n Answer is %d", sum)
```

((with out () नसेल honar)) ((ब्रॅकेट))

((Add pn hott ohanar & digit seprate pn honar)) i.e sum = sum ± (n%10)

output explanation :- e.g 12712 ⇒ 2+1+7+2+1 ⇒ 13

LECTURE 29

Que.) write a program to make 4 digit no. reverse

i.e
1. n=4235, r=5
2. n=423 ⇒ r=5*10+3 = 53
3. n=42 ⇒ r=53*10+2 = 532
4. n=4 ⇒ r=532*10+4 = 5324

((multiply karun ho tay nit ba))

((pan program असा लिहायचा आहे))

```
void main();
{
  int n, r;
  printf("o---
  scanf("%d", &n);
  r = n%10;
  n = n/10;
  r = (r*10) + (n%10);
  n = n/10;
  r = (r*10) + (n%10);
  -              2 more time
  r = (r*10) + (n%10);
```

i.e → 4326

6
432 --- (6*10) + (432%10); = 62
43 --- (62*10) + (43%10); = 623
(623*10) + (4%10) = 6234

printf("Answer is %d", r)

LECTURE 32 - Associativity of Operator

Precedence Table with associativity

operator	Associativity
* / %	Left to Right ((operation क्रमानुसार))
+ -	Left to Right ((-11-))
=	Right to Left ((-11-))

i.e a = 3*2+1 = 6+1 = 7

but
i.e

4/2*3-1

(4 = unambiguous operand; 2 = ambiguous operand; 3 = unambiguous operand) \\ ambiguous म्हणजे confuss करणारा (आता प्रश्न असा आहे की आता आधी आम करायचं))

** Note :- आता associativity क्रमामध्ये Left to right ahe मग ambiguous च्या Left side ला जो unambiguous operand आहे त्या सोबत operation होणार. \\ आत "2" च्या left la division ahe मग ती आधी solve होणार

i.e. ① 5%2*7-10/5*3+4
= 1*7-2*3+4
= 7-6+4
= 1+4
= 5

LECTURE 33

▸ Let take a=5, b=10, c=3, y=?

① y= a<b
= 1 \\ True=1, false=0

② y = b<c
= 0

③ y= a<b && c<b
= 1 && 1
= 1

④ a<b>c || a<=c
= 1>c || 0
= 0||0
= 0

All OPRATOR TABLE

OPERATOR	ASSOCIATIVITY
( )	L to R
!	R to L
* / %	L to R
+ -	L to R
< <= > >=	L to R
== !=	L to R
&&	L to R
||	L to R
=	R to L

LECTURE 34 EXPRESSION EVALUATION PRACTICE

1] value of x & y are 10 & 2 find value of z

① $z = x != y \Rightarrow z = 1 \Rightarrow$ z=1

② $z = (x+y)\%2 != 0 \Rightarrow z = 12\%2 != 0 \Rightarrow z = 0$

③ $z = !((x>y)\&\&(x>5)\&\&(y>10))$

$= !(1\&\&1\&\&0)$

$= !(1\&\&0) = !0 = 1$

"!" जर Binary digit च्या आधी आलं तर त्याला invert करतं

2] value of a, b & c are 10, 5, 50. find

① $a\&\&b\&\&c$ // All digit are consider as 1 (True)

$= 1\&\&1\&\&1 = 1$ (except "0")

② $!(a\&\&(b<c)) || ((a||b)*0)$

$= !(a\&\&1) || (1*0)$

$= !1 || 0$

$= 0 || 0 = 0$

LECTURE 35 → MCQ question

LECTURE 36 → 'If' statement.

LECTURE-37 → 'IF ELSE'.

LECTURE 40 → Nested if else

if(0) - Aasel tr ram hott nahi tr nahi Hott.

```
void main()
{ int x=5, y=7;
  if(x=5) { printf("open Up");
    if(x<y) { printf("closed Up");
      if(x=y) { printf("zero Up");
        if(x-y=2) { printf("Hero Up");
        }
        printf("level Up");}
      printf("upgrade it");}
    printf("LaLaLa");}
  printf("zero is hero");
```

Output

→ Open Up
closed Up
LaLaLa
zero is Hero

&& → AND
|| → OR
!= → NOT equal

Lecture 41 → check greatest no. among 3 no. entered by user without logical operator.

Lecture 42 → —"— with logical operator.

Lecture 43+44 → if else ladder \\ means मतलब else if use करना

```
{ int a=5;
  if (a>10) printf("Greater 10");
else if (a>7) printf("greater than 7");
else if (a>3) printf("greater than 3");
else if (a>1) printf("greater than 1");
  getch(); }
```

O/P → greater than 3

LECTURE 45 p.g. to disp. name of item when price is entered by user.

```
main()
{ int a;
  printf("Enter Your Price bet'n 100,25,50");
  scanf("%d",a);
  if(a%100==0) printf("Ink pen");
  else if(a%50==0) printf("Pen");
  else if(a%25==0) printf("Pencil");
  else {printf("Item not available");}
  getch(); }
```

LECTURE 46 check leaf year

```
main()
{ int a;
printf("Enter Year");
Scanf("%d,a");
if (a%4==0) if(a%100!=0)
{printf("Year is leap");}
else {printf("Entered Year is not leaf");}
  getch(); }
```

LECTURE 47 // check if triangle is valid or not
// if addition of any two side of triangle is greater than 3rd side then it is valid triangle i.e $\begin{pmatrix} a+b>c \\ b+c>a \\ a+c>b \end{pmatrix}$

```
main()
{ int a,b,c
  printf(" ");
  scanf(" ");
  if(a+b>c) if(b+c>a) if(a+c>b) {printf("valid");
  else {printf("Not valid");
  getch();
  }
```

* With Logical operator

```
if((a+b>c) && (b+c>a) && (a+c>b))
{printf("valid");}
else {printf("not valid");
getch();
}
```

Lecture 48-49-50 - Number system

Lecture 51 - 3 way to store integer.

Today generally Two complement repre.s use hoti

① Sign magnitude representation

In this way of integer representation most significant bit (MSB) is used as sign bit and remaining bit in number are used to represent magnitude.

- for +ve integer sign bit is zero
- for -ve integer sign bit is one.

* If there is n bit space is available than range of integer that we can represent is SMG is

$$-(2^{n-1}-1) \text{ to } +(2^{n-1}-1)$$

i.e is 4 bit space

$$= -(2^{4-1}-1) \text{ or } +(2^{4-1}-1)$$

$$= -7 \text{ or } +7$$

② +30 in 6 bit space

$+(2^{6-1}-1) = 32-1 = 31$

32	16	8	4	2	1
0	1	1	1	1	1

- Advantage of SMR → Integer representation is simple and straightforward.

- Disadvantage of SMR → ① There are two representation of zero which create ambiguity.
② Each of sign & magnitude part has to be processed separately which complicate the design of logic circuit that handle arithmetic operation,

i.e ① 1000 → −0 | 0000 → +0   ((but zero has no sign.

NOT using nowdays

52 Lecture

— One's Complement Representation of int..

One's complement → Manje invert ~~karo~~

i.e 1) +15 = 001111 , now one's complement → 110000
in 6 bit

2) −15 = 110000 , now one's complement ⇒ 101111
in 6 bit

3) −100 = 01100100 —one's com→ 10011011
in 8 bit

Range :-

Formula − $(2^{n-1}-1)$ to $+(2^{n-1}-1)$

1] 00110 → 6   checking if it is correct → $2^{5-1}-1 = 15$ correct

2] 100101 → 37   —||— $2^{6-1}-1 = 31$, it is not correct ∴ it is inverted/negative
∴ one's complement
011010 = −26.

3] 1000000 → 64   —||— $2^{7-1}-1 = 63$ it is not correct ∴ it is negative
(64 32 16 8 4 2 1)
now one's complement
0111111 = −63

If there is 1 in most significant digit then it is −ve

if 1010 = -ve because $2^{4-1}-1 = 7$

significant = 7 >> 2 bit

// one's complement is easy

* ADVANTAGE :- 1] simple to represent Integer
2] Circuit designing for addition & substraction is simpler.

* DISADVANTAGE :- 1] There are two representation zero
2] Circuit designing is relatively simple than that in sign magnitude representation but still complex.

LECTURE 53 :- Two's complement Representation

1] +3 in 8 bit = 00000011

2] -17 in 8 bit = 00010001 → one's complement

```
   11101110
+         1
   11101111
```

Now add one in 2's com

3] -32 in 8 bit → 00100000

```
   11011111   ← 1st com
+         1   ← 2's com
   11100000
```

4] -39 in 16 bit →

```
   0000000000100111
+  1111111111011000   ← 1st comple
+                 1   ← 2st com
   1111111111011001
```

* Range :-

$$\left[-(2^{n-1}) + 0 + (2^{n-1}-1)\right]$$

i.e. 16 bit.

$$= -(2^{16-1}) + 0 + (2^{16-1}-1)$$

$$= -2^{15} + 0 + 2^{15}-1$$

$$= -32768 + 0 + 32767$$

① 001101 = 13 -2^5 to 2^5-1
= -32 to 31

② 11011 = -5 x 2^4 to 2^4-1
00100   = -16 to 15
+ 1
00101 = -5

③ 10011101
01100010
+ 1
01100011 = -195

TIP
If most significant (first) Digit is one then no. is -ve

ADVANTAGE :- ① There is only one representation of zero
② Circuit design for two's complementation is simple

LECTURES 54 → trick to solve 2's complement

Short trick :- 0000000011000110
मेल नाही जो पर्यंत।
मेल नाही तो पर्यंत
आहे तसं write
करत राहो →
(Right कडून Left)   1111111100111010

e.g. ② find -320 in 16 bit
:- 0000000101000000
1111111011000000

you have to write first one as it is, then you have to change remaining bits/integer.

lecture 55 :-

1) what is 2's complement of -15
→ ① 1111 ② 11111 ③ 111111 ④ 10001

2^{n-1} to $2^{n-1}-1$ = 2^{5-1} to $2^{5-1}-1$ = -16 to 15

: 01111

LECTURE 56 → ~~Remem~~ Bitwise operator in C

Logical op. and bitwise op. are not same

& (bitwise ~~AND~~) | (BW OR) ^ (BW XOR) ~ (BitW Not) << (left shift) >> (right shift)

LECTURE 57 Bitwise AND operate

note: ① Logical operate work on two condition

② bitwise operation work on bit.

e.g

1] int main()
{
int a=15, b=39, c;
c= a & b;
printf("%d", c);
getch();
}

a = 15 = 0000 0000 0000 1111
b = 39 = 0000 0000 0010 0111
c = 0000 0000 0000 0111

∴ c = 7

2] eg:
int a=12, b=-5, c;
c= a & b;
printf("%d", c);
getch()

a = 12 = 0000 0000 0000 1100
b = -5 = 1111 1111 1111 1011
0000 0000 0000 1000

∴ c = 8

LECTURE 58 Bitwise OR operation

e.g
int a=13, b=23, c;
c= a | b
Pf("%d", c);

a = 0000 0000 0000 1101
b = 0000 0000 0001 0111
0000 0000 0001 1111

∴ c = 31

e.g 2
int a=-9, y=-17, z
z= a | b;
Pf("%d", z);

a = 0000 0000 0000 1001 (2's)
1111 1111 1111 0111
b = 1111 1111 1110 1111
1111 1111 1111 1111

∴ z = -1

LECTURE 59 → Bitwise XOR (^)

0	0	0
0	1	1
1	0	1
1	1	0

e.g
```
int a=15, b=64, c;
c = a^b;
pf("%d", c);
```
a = 0000 0000 0000 1111
b = 0000 0000 0100 0000 ^
c = 0000 0000 0100 1111

c = 79

e.g
```
int a=-1, b=-22, c
c = a^b
Pf("%d", c)
```
a = 1111 1111 1111 1111
b = 0000 0000 0001 0110 ^
1111 1111 1110 1001 → 2's
0000 0000 0001 0111

c = -23

LECTURE 60 → Bitwise NOT operator (~)

e.g
```
int a=15, b
b = ~a;
pf("%d", b);
```
a 0000 0000 0000 1111 ~
b 1111 1111 1111 0000
2's 0000 0000 0001 0000

b = -16

Logical NOT operator change logic (T→F), F→T

LECTURE 61 → Left shift (<<) operator

e.g
```
int a=5, b;
b = a<<1;
printf("%d", b);
getch();
```
a = 0000 0000 0000 0101
आता <<1 हा value shift करायचा left ला

b 0000 0000 0000 1010

∴ b = 10

जे left लागेल ते बाहेर जाणार

जेवढी value shift करायची ते << मारायचे

e.g
```
int a=5, b;
b = a<<3;
Printf("%d", b);
```
a = 0000 0000 0000 0101
b = 0000 0000 0010 1000
b = 40

Trick: a<<n → a*2^n but only when it is in range

<< लावले 0 येतो
>> लावले काही येतात

LECTURE 62

here we see most significant digit and put value, ie if most significant value is 1 then put 1 in right shift.

```
int x=18, y;
y = x>>1;
printf("%d", y);
getch();
```

a = 0000 0000 0001 0010
b = 9

if x>>2

0000 0000 0000 0100
= 4

eg 2)

```
int x=-18, y
y = 18>>1;
pf("%d", y);
```

a = 1111 1111 1111 1110
x>>1

if x>>2

1111 1111 1111 0111 1111 1111 1111 1011
if = -9 = -5

Trick :-
x>>n = x/2^n, but if answer is in decimal point, consider small near no. ie 4.5 = 4 & -4.5 = -5

4.5 = 4 & -4.5 = -5

LECTURE 62.1 Endian.

Note :- Every bit has their own address

Endian :- the term endian refer to order of storing bytes in computer memory

* Endian *

big endian

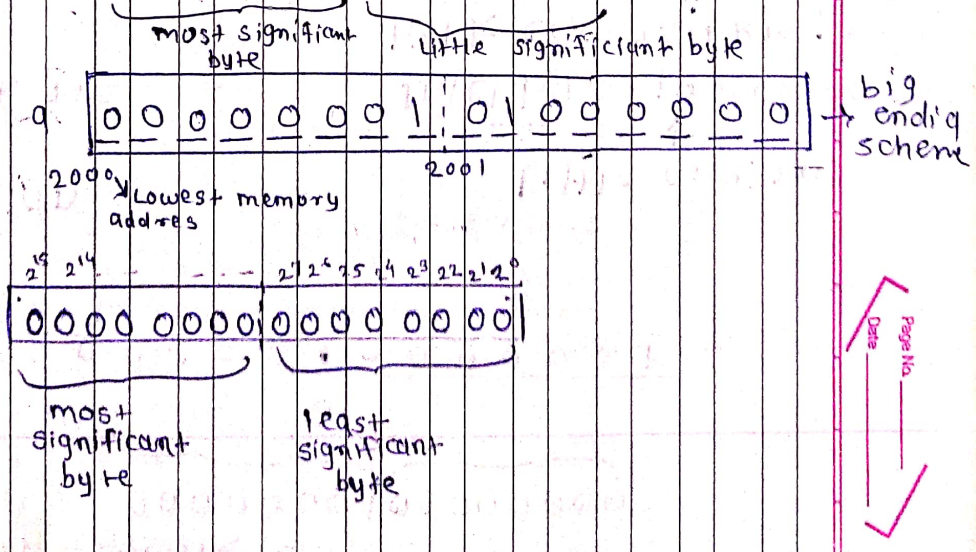
In big endian scheme the most significant bit store in lowest memory address

Little endian

In Little endian scheme the least significant bit store in lowest memory address

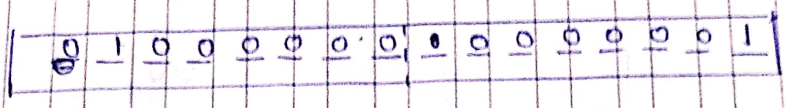
Big Endian eg

eg. int 320; → 0000000 1 0100000



Little endian Scheme :-

e.g a = 320 = 00000000 | 01000000



Big endian

motorala 68K are big endians.

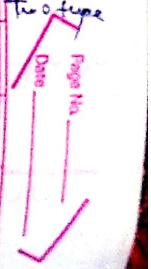
Power PC (by motorla) and SPARCK (by sun) processor were big endian current version of these processor are ~~bi-end~~ bi-endian

That support two type.

Little endian

Intel based processor are little endian. ARM proce-ssor were little endian. But currgen ARM generati-on processor are Bi-Endian

That support Two type



LECTURE 63 → character & their ASCII value

Storage of character constant is in one byte

8 bits

ASCII (American Standard Code for Information Interchange)

→ There is an integer is associate with each character constan which is known as Ascii value or that character constant.

Capital Letter						Small leter						Digits					
A	B	C	...	X	Y	Z	a	b	c	...	x	y	z	0	1	2	...	8	9
↓	↓	↓		↓	↓	↓	↓	↓	↓		↓	↓	↓	↓	↓	↓		↓	↓
65	66	67		88	89	90	97	98	99		120	121	122	48	49	50		56	57

In case, if you forget value

```
{ char ch = 'B';
  pf("%d", ch); }
```

LECTURE 7 !! Decision control statement switch-case-default.

Syntax :-
```
switch(Integer Expression)
{
  case constant 1:
   statement 1;
   statement 2;
      :
  case constant 2:
   statement 1;
   statement 2;
      :
  default :
   statement 1;
   statement 2;
}
```

ie:
```
void main()
{
 int a = 3
 switch(a)
 {
 case 1:
  pf("H");
 case 2:
  pf("No");
  pf("\nYes");
 case 3:
 pf("right");
 case 4:
 pf("Dear");
 default:
 pf("Thank");
 }
getch();
}
```

जर instruction-case सोबत match झाली तर त्या खालची सगळी statement run होणार

पण जर each case मध्ये जर थांबवायचे असेल तर break; statement use करणे

LECTURE 8 Increment and Decrement

Increment :- ++
- → Post increment :- a++ ; i.e. a=5, b; i.e. b=a++; pf(b=5; a=6)
- → Pre increment :- ++a → ie a=5, b; ie b=++a; pf(b=6; a=6)

Decrement :- --
- → post Decrement :- a--
- → pre increment :- --a

Post increment e.g :- int a = 4;
Pf ("%d", a++); = 4
Pf ("%d", ++a); = 5

Post Decremen eg :- int x = 9;
Pf ("%d", x--); = 9
Pf ("%d", x); = 8

LECTURE 99

→ Introduction to loops.

syntax of for loop :-

```
for(exp1; exp2; exp3);
{
  statement 1;
  statement n;
}
```

When loop limit extend, then loop repeat it self.

32767 + 1 = -32768

i.e -32768, -32767 ------ 0, 1, 2, --- 32767

for 16bit

∴ -32768 + 2 = 32766

LECTURE 101

ARRAY

→ Def^n :- Array is collection of similar type of data elements.

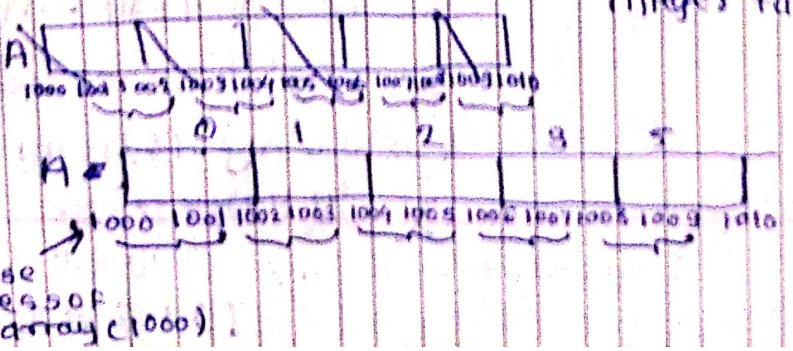
Syntax :-

Data Type Array Name [size];

Note :- memory allocated to an array is always contiguous.

i.e int A[5];

integer take 2 byte in 16 bit compiler



LECTURE 101

1) int A[5] = {10,20,30,40,50,60};
   = ERROR ✗
2) int A[5] = {10,20,30}
   = 10,20,30,0,0 ✓
3) int A[5] ✓
   = Garbage value
4) int A[] = {2,4,6,8}; ✓
   = Size = A[4]
5) int A[] ;
   = ERROR ✗

LECTURE 103 Accessing of array

```
void main();
int a[5] = {10,20,30,40,50};
int i;
for(i=1;i<4;i++)
{
printf("%d\n",a[i]);
}
```

LECTURE 104 initializing Array at runtime

```
int a[5],i;
printf("Enter 5 integer");
for(i=0;i<5;i++)
{
  scanf("%d",&a[i]);
}
getchar();
```

LECTURE 107 Two Dimentional Array

DATATYPE ARRAYNAME[ROW SIZE][COLUMN SIZE]

int a[3][4] = {{10,5,17,0},{1,2,3,5},{1,1,2,1}};

	C_0	C_1	C_2	C_3
R_0	10	5	17	0
R_1	1	2	3	5
R_2	1	1	2	1

{1,3,9,4,20,1,3,5,6,3,2,1}
R_0 R_1 R_2

Note :- adhi row ~~pas~~ stored honar.

int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12,13};
XX ERROR

int a[3][4] = {1,2,3,4};
आहि तेवडे store होणार बाकी zero मिळणार

int a[][4] = {1,2,3---};
int a[][3] = X

int a[2][] = {10,20---80}; X
row size
int a[][] = X (~~column~~) dile tr chalel pn
~~row~~ column must be their size

* Column size must be define, if only column size is define then it true
if only row size define, then it wrong

LECTURE 109 memory mapping of 2D Array

i.e a[3][4] = {10,20,30,40,50,60,70,80,90,100,110,120}

0	1	2	3	0	1	2	- - -	2	3
10	20	30	40	50	60	70	- - -	110	120
500	502	504	506	508	510	512	514	520	522

Row 0 Row 1 Row 2

i.e a[1][3] = 80

i.e a[0][0] = 10

	0	1	2	3
0	10	20	30	40
1	50	60	70	80
2	90	100	110	120

Col Row

Lecture 11 & 12

```
int a[2][3] = {1,2,3,4,5,6};
for(i=0; i<=1; i++)        // For first row then column
{
  for(j=0; j<3; j++)
  {
    printf("%d", a[i][j]);
  }
};
```

// for first colum then row

```
int a[2][3] = {1,2,3,4,5,6};
for(j=0; j<3; j++)
{
  for(i=0; i<2; i++)
  {
    Pf("%d", a[i][j]);
  }
```

Lecture 11 2 Function

Defn: A function is self-contained block of statement that are perform some task

FUNCTION

- Predefined
- User defined

(void) return type

LECTURE 115 — Passing Argument in Fun.

(Argument ≈ Parameter)

```
void fun(int);
void main()
{ int x=5;
  fun(x);          → Actual Parameter
  getch();
}
void fun(int a)    : a = x = 5
{
 pf("%d", a);      → Formal Parameter
}
```

LECTURE 117

return (t); last का लिखा की value return होती

void() का value return होता नाहि

Lecture 118 — Modular Approch in programming

- Modular programming is process of subdividing a computer program into separate sub-program
- In 'C' programming we divide the program into smaller module called funⁿ which handle a particular responsibility.

ADVANTAGE)

- It become easy to manage a program
- It become easy to work in team
- It improve quality of program
- It become easy to reuse function
- Debugging of program become easy.

Scanned with OKEN Scanner

LECTURE 121 — Storage classes of C

Storage classes
- Automatic S.C
- Static S.C
- External S.C
- Register S.C

Lecture 122 — Automatic storage class

:- Storage → Main Memory
:- Default initial value → Garbage value
:- Scope → within the block in which variable is defined { }
:- Life → Till the end of block in which variable is defined. { }
:- Keyword → auto

Lecture 123 — Static storage class

:- Storage → Main Memory
:- Default initial value → zero
:- Scope → within the block in which variable is defined
:- Life → till the end of program. ((double variable))
:- keyword → static

Note:- value of variable persists bet^n different function call.

\\ one variable can be defined one time in program
i.e if 2 variable defined then 2^nd time compiler
~~ignore~~ ignor definationi \\ static [illegible] certain variable [illegible] defined [illegible]

Lecture 124 Register storage class

- Storage → CPU register
- Default initial value → Garbage value
- Scope :- within the block in which variable is defined
- Life :- Till the end of block in which varible is defined.
- keyboword :- register.

In this process variable stored in processor
Speed is fast of program

LECTURE 125 External storage class

- storage → main memory
- Default Initial value → zero
- Scope → Global // पूरा program
- Life → Till the end of the program
- keyword → extern

```
ie  int a;  ---> external type
    void fun();      ∴ a = 0
    void main;
    {
    pf("%d", a); → 0
    fun();
    }
    fun();
    {pf("%d", a+2); → 2
```

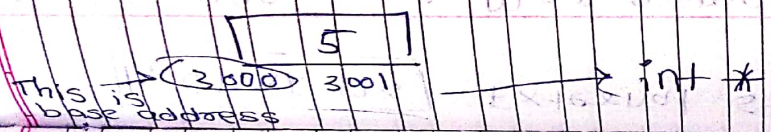
Lecture 126 Integer Type

Unsigned	signed
- In 16 bit compiler range is 0-65535	- In 16 bit compiler range is -32768 to +32767
- format :- %u	- format :- %d
- keyword :- unsigned	- keyword :- signed.

Lecture 127

** Pointer variable :- It is a special type of variable which store the address of some other variable or location.

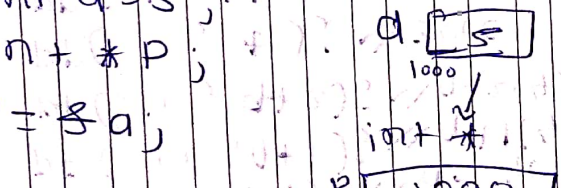
in 16 bit m/c int a=5;



Lecture 128

indirection operator (value of)

```
int a=5;
int *p;
p=&a;
```



* मध्ये value किती आहे check करणे but * च्या समोर जे आहे त्याला * address समजतो

a → 5

&a → 1000 *(&a) = 5

p → 1000 = *(1000) *(&p) *P = *1000

&p → 2000 = 5 *(2000) = 5

= 1000

address

((&p))

= *(*2000)

= *1000

= 5

Lecture 132 :- Check out 132.c file

```
int a=5, b=7;
float c='A';
int *p,*t;
float *q;
p=&a;
t=&b;
q=&c;
Pf("%d",p); \\p=2000
Pf("%d",p+1); \\p+1(size of int) ∴ p+1(4) = 2004
Pf("%d",p+2); \\p+2(4) = 2008
Pf("%d",q); \\ q= 4000
Pf("%d",q+1); \\q+1(4) = 4004
Pf("%d",q+2); \\q+2(4) = 4008
Pf("%d",t); \\3000
Pf("%d",t+1); \\t+1(size of char) t+1(1) = 3001
Pf("%d",t+5); \\t+5(1) = 3005.
getch();
}
```

a=[5] 2000

suppor &a = 2000
∴ p=2000; *p=5
t=3000; *t=7
q=4000; *q='A'.

formulae:- here size of int in 32 bit compiler is 4 & float size=4 & char=1

Lecture 133 :- What if we do ~~+~~ - betn two pointer.

:- Note :- We can only do substraction in betn 2 ptr.

```
int a=5, b=6, *p, *q;
p=&a;
q=&b;
Pf("%d is answe",&p-&q);
```

q=[5] 6000 b=[6] 4000

\\ ∴ (6000 - 4000) / size of int
= (6000 - 4000)/4 = 2000/4 = 500

Lecture 134 :-

Assesing array usin pointer.

```
int a[5]={1,2,3,4,5};
int *p;
p=&a[0];
for(int i=0;i<=5;i++)
{
Pf("%d",p[i] or *(p+i));
```

a: [1][2][3][4][5] 1000 1004 1008 1012 1016 1020

Lecture 135 String :- It is collection of character.

'\0' - Null character

At string, add \0 to every last of end.

i.e

```
char a[6]={'A','B','C','D','E','\0'}
  or
char a[6]="PAPIU";
```

Lecture 136 Printing String

```
void main
{ char a[]="shantanu";
  int i=0;
  for(i=0;a[i]!='\0';i++)
  {
   Pf("%c",a[i]);
  }
  getchar();
}
```

→ or

```
while(a[i]!='\0')
{
 Pf("%c",a[i]);
 i++;
}
```

OR

```
Printf("%s",a);
```

Lecture 137-138 check on laptop.

Lecture 139

```
int char a[20];
Pf("Enter Name");
Sf("%s",a); \\ Here we don't need to &
Pf("%s",a); \\ same here also.
```

Lecture 140 new fun.

gets(); → use to get input \\scanf()

puts(); → use to display string \\printf()

```
int char a[10];
Pf("Enter");
gets(a);
puts(a);
```

Lecture 141 New standard function String

We are going to learn new header file fun

```
#include<string.h>
void main
{
 char A[] = "SHANTANU HULWAN";
 int b;
 b= strlen(A);
 pf("%d", b);
}
```

o/p = 15

strlen( ); use to measure length of string.

Lecture 141 strcpy :- one string to another string.

```
#include<stdio.h>
#include<string.h>
void main()
{
 char a[10] = "SHANTANU";
 char b[ ];
 strcpy(b,a);
 pf("%s",a);
 pf("%s", b);
}
```

Lecture 142

```
#<string.h>}
void main
{
 a[20];
 b[30] a
 strcpy(b, "Hello world");
 strcpy(a, b);
 pf("%s", a);
 pf("%s", b);
```

o/p

Hello world
Hello world

Lecture 143 strrev : use to reverse the string

```
#include<string.h>
void main()
{
char a[10]: "Hello_yes";
strrev(a);
Pf("%s",a);
}
```

O/P: sey-olleH

Lecture 144 strcmp : use to compare the string

```
#<string.h>
void main
{
char a[10] = "Hello world";
char b[10] = "Hello world";
int x;
x = strcmp(a,b);
```

strcmp(,) if same then output is 0, else 1.

actually this fun do substraction.

```
if(x==0)
{Pf("Both are same");}
else
{Pf("Both are not same");}
}
```

Lecture 145 This ~~fun~~ is use for check palindrome

→ Palindrom:- Left to right & right to left same,
ie ① saras ② abba.

— check program on Laptop.

Lecture 146 :- Structure is user-defined data type in C. It allow us to store data of diff^{n} types together.

*Syntax :-

```
struct stru-name
{
 datatype   var1;
   ''       var2;
   ''       var3;
    .
    .
    .
};
```

```
#include<stdio.h>
struct stu
{
 int rollno;
 float mark;
};
void main()
{
 struct stu x;
 x.rollno=5;
 x.mark=85.5;
 Pf("%d",x.rollno);
Pf("%f", x.mark);
 getchar();
}
```

Lecture 151 check laptop

remember to use fflush(stdin) to take accurate getsc() type input.

In laptop we face problem while taking input from user.

So we use fflush(stdin); in line ((21)).

Lecture 152 → Pointer to structure.

```
#include<stdio.h>
struct stu
{ int roll;
  float mark; };
void main()
{
 struct stu x={10,10.10};
 struct stu *p;
 p=&x;
 Pf("roll is %d", (*y).rollno.); \\y->ro
 Pf("mark ox%f", (*y.mark); \\y->ma
}
```

Pointer size

int *x → 2 bit
float *y → 2 bit
char *t → 2bit
struc stu *p → 2bit
because it save addresses only

Lecture 153 → New data type UNION

syntax :-

```
union unionname
{ data type 1 val1;
  data type 2 va2;
};
```

Lecture 154 | Difference betn structure & Union.

	Struct	Union
Memory	Separate memory space is alloted to each member of structure.	All member share the same memory space.
Syntax	Both nearly same	Both nearly same
Size	Size is sum of all data member bit size	Size is ~~are~~ equal to largest bit size among all variable
Stored value.	Whenever the value is assigned it does not disturb other defined vari-able value.	Here present defind value only store, once another value assigned then it's ecrse Past values

Lecture 155

* Recursion :- The process in which a function call itself directly or indirectly is called recursion and such a function is called recursive funt.

Lecture 156 : Recursion - Part 1

Well defined # recursion
:- ① There must be certain condition, called base condition for which the function stop to call itself. ② Each time the function call itself, it must be closer to base condition.

Lecture 160

* Static memory allocation :- It is allocation technique which alloca-te a fixed amount of memory during compile time.

* Dynamic memory Allocation :- In this technique memory is allocated while executing program or we can say memory is allocated at run time

Lecture 161

check laptop

malloc() : It is used to dynamically allocate the memory.

free() : It is "de-allocate".

Lecture 162

Calloc :- "contigous allocation" function is ued to dynamically allocat memory space for specified number

Note :- By default it initialize each data element with zero

Lecture 163 Preprocessor: C preprocessor is program that processes our ~~prog~~ source program before it given to compiler.

(1) Source code is given to preprocessor and then preprocessor create expanded source code.

(3) Preprocessor directive being with # symbol

(4) Command that we give to preprocessor are known as preprocessor directive.

(5) Types of preprocessor.
- (a) Macro Expansion
- (b) file inclusion
- (c) conditional compilation
- (d) miscellaneous directive

Lecture 164 Macro expansion.

```
#define  PI   3.415
Preprocessor  Macro   macro
Directive     templae expansion
```

Lecture 165

```
#define area(r) (3.14*r*r)
void main()
{
 float r1=0.17, r2=3.1;
 Pf("%f %d", area(r1), area(r2))
```

* check laptop for best info.

- To expand multiple line in macro expansion

```
#define loop  for(i=1;i<10;i++)\
              {\
               Pf("%d",i);\
```

Lecture 167 #include is preprocessor directive which is used to include a header file in our program.

enum {True, False} \\ enum count number and fixed them

or {False, True}

or {Hello, How, are, you} (0 1 2 3) ← compiler consider these value

* Time.h

<#unistd.h> मध्ये sleep(n) n sec program sleep hoto.

<time.h>

time_h variablenam = time(NULL)

:- current time save karat.

pn ते करताना ~~1970 मधून~~ ~~कर~~ करत आसतं 1 Jan 1970 पासून

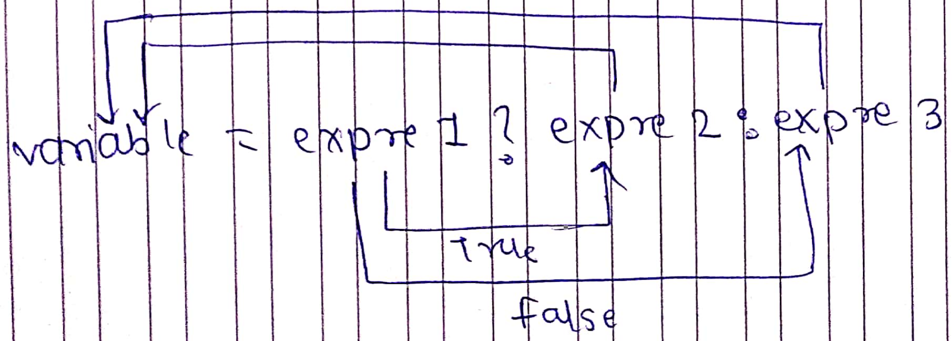
time_h m = time(NULL);

जर current time print karayche asel tr

char *st = ctime(&m);

pf(%s, st); ⟶ current time print hotti

* Conditional Operator: ?:



if (exp1) satisfy expre2 then true or false.