@Rohay Lal
sheet-I.

Why C?

- C is simple
- C is small
- C is fast
- C offers better interaction with hardware

Historical Development of C

1960 International Committee	ALGOL 60
1963 National Committee (UK)	CPL
1967 1 Man Committee	Martin Richards - BCPL
1970 Ken Thompson, AT & T Bells Lab	B
1972 Dennis Ritchie, AT & T Bells Lab	C

ALGOL → algorithmic language.
CPL → combined prog language.
BCPL → basic combined prog language.
C cos one step ahead of B.

Where C Stands

low level lang. — high level lang.

Computer Languages

LLL	HLL
Machine Oriented Ex. Assembly, Machine	Problem Oriented Ex. Basic, Fortran, Pascal,Cobol
Better Machine efficiency	Better Programming efficiency
faster harder access	slower easier access

C → middle level lange.
slower than LLL faster than HLL
easier than LLL harder than HLL.

In The Beginning...

English	C
Alphabets, Digits	Alphabets, Digits, Sp. Symbols
Words, Numbers	Constants, Variables, Keywords
Sentences	Statements or Instructions
Paragraph	Program

Alphabets, Digits, Special Symbols

Alphabets
A - Z, a - z

Digits
0 - 9

Sp. Symbols
Total - 32

case sensitive.

; + ; & # etc.

Constants and Variables

Change — Variables — x, y

3 x + 2 y = 20

rigid. — Constants — 3 2 20

C Constants

Primary	Secondary
Integer	Pointer
Real	Array
Character	String
	Structure
	Union
	Enum
	etc.

Integer Constants

Ex. 421 -62 +45 4096

Rules:

1. No decimal point. 72 72.0
2. May be +ve or -ve. Default: +ve
3. No comma or spaces 32,500 4 7 3
4. Valid Range: -32768 to +32767

Real Constants

Ex. 427.62 +24.297 -0.00254

Rules:

1. Must contain a decimal point.
2. May be +ve or -ve. Default: +ve
3. No comma or spaces.
4. Valid Range: -3.4×10^{38} to $+3.4 \times 10^{38}$

Variables

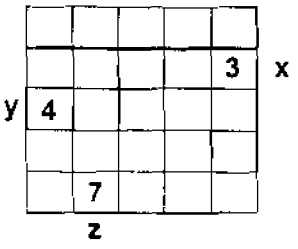
- How many types? Why?
  As many as the types of constants.
- Is it necessary to identify types?
  Yes.
  Why?

① as only those values can be stored in variables that are able to store it eg ID in int
10.2 not in int.

② to specify whether int or char variable. To define the variable.

What Happens in Memory..

Memory arbitrary location



x = 3
y = 4
z = x + y
print z
output: 7

variable is the power tool in C

So A Variable is...

- An entity whose value can change
- A name given to a location in memory

How to Identify Types

3 → Integer Constant
3.0 → Real Constant
'3' → Character Constant

a → ?
b → ?
c → ?

only specific types of data can be stored in ~~dep~~ specific data types.

C's Way of Identifying Variables

int a	float b	char c
a = 3	b = 3.0	c = '3'
(integer)	(real)	(character)

floating pt. constant = Real Constant.

Rules for Building Variable Names

- First character must be an alphabet, rest can be alphabets, digits, or underscores.

  Ex. pop98 sl_int ✗ si-int ✓ si_int

- Length <= 8 (Usually)
  eg name of variable.
- No commas or spaces
- Variable names are case sensitive
  Ex. abc ABC Abc aBc AbC

C Keywords

- How many?
- What are Keywords?
- What are Reserved words?

- 32 keywords
- words whose meaning is already explained to the machine. eg int, float, ch
- same as keywords.

Would This Work?

integer a	☒	int a
real b	☒	float b
character c	☒	char c

And How About This...

int float ☒	float char ☒
float = 3	char = 3.14

never use a keyword as a variable name.

why? 'cos float already expressed to computer. By making it a variable we are trying to confuse the PC.

Where Do We Stand?

- Alphabets, Digits, Sp. Symbols
- Constants, Variables, Keywords
- Statements or Instructions
- Program

The First C Program

p = 1000.50

n = 3

r = 15.5

si = p * n * r / 100

float P, r, Si;
int n;

Declaring Variables...

```
float  p, r, si
int  n
p = 1000.50
n = 3
r = 15. 5
si = p * n * r / 100
```

Tip: All variables must be declared.

Printing Values...

```
float p, r, si
int n
p = 1000.50
n = 3
r = 15. 5
si = p * n * r / 100
printf ( "%f", si )
```

p	n	r
1000.50	3	15.50

si
465.25

- printf is the display command.
  printf (si) only value of si.
  "%f" format specifing
- specifies the format of output variable here float : %f.

Terminology Matters

( ) Parentheses

{ } Braces

[ ] Brackets

***

General Form of *printf( )*

printf ("format string", list of variables)

can contain

%i - integer

%f - float

%c - character

Ex. printf ("%f %i %f %f", p, n, r, si)

for outputing several variables in the order of declaration.

here p = float
n = int
r = float
si = float.

Statement Terminators

I am a boy
I go to school

float p, r, si
int n

float p, r, si ;
int n ;

Statement Terminator

: colon ; semicolon

- semi colon is the terminator.
- fullstop is acting as a decimal thus not used.

Free Form Language

```
float p, r, si ;
int n ;
```

is same as

```
float p, r, si ;   int n ;
```

What To Execute

Collective Name

```
main( )
    float p, r, si ;
    int n ;
    p = 1000.50 ;
    n = 3 ;
    r = 15. 5 ;
    si = p * n * r / 100 ;
    printf ( "%f", si ) ;
```

What Belongs to *main()*

```
main( )
{
    float p, r, si ;
    int n ;
    p = 1000.50 ; n = 3 ; r = 15. 5 ;
    si = p * n * r / 100 ;
    printf ( "%f", si ) ;
}
```

indentation is a must. (format)

Comments Are Useful

```
/* Calculation of simple interest */
main( )
{
    float p, r, si ;
    int n ;
    p = 1000.50 ; n = 3 ; r = 15. 5 ;
    si = p * n * r / 100 ;
    printf ( "%f", si ) ;
}
```

Note

Remark

Tips About Comments

- Any number of comments anywhere

  /* formula */ si = p * n * r / 100 ; /* ..... */

- Multiline comments - ✓

  /* .....
  ..... */

- Nested comments - ✗

  /* ..... /* ..... */ ..... */

Single line comment // no closing mark as it is only one line.

A More General Program

```
/* Calculation of simple interest */
main( )
{
  float p, r, si ;  int n ;
  printf ( "Enter values of p, n and r " ) ;
  scanf ( "%f %i %f", &p, &n, &r ) ;
  si = p * n * r / 100 ;
  printf ( "%f", si ) ;
}
```

'Address of' operator

User specific prog:

scanf - obtains values.

syntax: ("declaration of type", &variable);

& → an address of the operator

like * multiplication operator.

Forms of Real Constants

427.62 → 4.2762E2

+24.295 → 2.4295e1

-0.00254 → -2.54e-3

Fractional Form Exponential Form

4.2762 E 2

Mantissa Exponent

Character Constants

Ex. 'A' 'm' '3' '+'

Rule : A single character enclosed within a pair of ' '.

Are They OK?

'Z' → [X]

'Nagpur' → [X]

both should slant to left

C Variables

Primary	Secondary
Integer Real Character	Pointer Array String Structure Union Enum etc.

@Rohay Lal
Sheet - II.

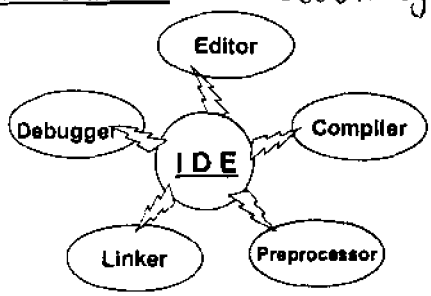
C Compilers

- ★ Turbo C
- ★ Quick C
- ★ Microsoft C
- ★ Aztech C
- ★ Zortech C
- ★ Lattice C
- ★ Watcom C
- ★ GreenLeaf C
- ★ Vitamin C

All Provide an IDE

IDE : integrated development environment

Integrated Development Environment — collection of prog.



Ctrl + F9 → compiler.

debug → removing of error.
Editor → main screen.
help to be the mainbrain.

Editor and Compiler

- ★ Editor - Helps in typing / editing of a program
- ★ Compiler - Converts C language program to machine language program

machine language → 1,0.
(low level language)

Ctrl+T → the entier word from position of cursor will be lost.

Ctrl+Y → entier line from position of cursor.

Editing Commands

Cursor Movement

- ↑ Up Arrow
- ↓ Down Arrow
- → Right Arrow
- ← Left Arrow
- Home, End
- PgUp, PgDn
- Ctrl+Home, Ctrl+End
- Ctrl+PgUp, Ctrl+PgDn

Deletion

- Del
- Backspace
- Ctrl+T
- Ctrl+Y

Computers are fun

Screen

File

Cursor → helps to type prog gives current position.

Home & end helps in acellerating movement

///ly ~~Hom~~ Page Up & Page down.

Ctrl+home / end → takes you to top/bottom of screen.

Ctrl + Pg Up & Pg down → first line of file / last line of file.

** imp short cuts.

Some More Commands

File Menu

- New
- Save
- Save As
- Open
- Exit
- Dos Shell

Permanent

Temporary

Miscellaneous

F2	Save
Ctrl+F9	Compile
Alt+F5	Output-Screen
Alt +F3	Close window

Alt + X exit.

Tip: Use meaningful filenames. Ex. CH1PR1.C

window at the bottom → Error window.

Interchanging Contents of Two Variables

```
main( )
{
   int c, d ;
   printf ( "Enter values of c and d" ) ;
   scanf ( "%i%i", &c, &d ) ;
   c = d ;
   d = c ;
   printf ( "%i %i", c, d ) ;
}
```

5 10

c	d
5	10
10	10
10	10

Method-I

Interchanging Contents of Two Variables

```
main( )
{
   int c, d , t ;
   printf ( "Enter values of c and d" ) ;
   scanf ( "%i%i", &c, &d ) ;
   t = c ;
   c = d ;
   d = t ;
   printf ( "%i %i", c, d ) ;
}
```

5 10

c	d	t
5	10	-
5	10	5
10	10	5
10	5	5

Method-II

One More Way

```
scanf ( "%i%i",&c, &d ) ;
c = c + d ;
d = c - d ;
c = c - d ;
printf ( "%i %i", c, d ) ;
```

15 10

c	d
15	10
25	10
25	15
10	15

Sum of Digits

```
main( )
{
   int  n ;
   printf ( "Enter a five digit no." ) ;
   scanf ( "%d", &n ) ;
   .. ..
   .. ..
}
```

26913

2 6 9 1 3
d1 d2 d3 d4 d5

%i & %d are both used for int
%d is better.

Calculations

26913 / 10 ⟹ 2691 - Division

26913 % 10 ⟹ 3 - Modular division

Mod

```
10 | 26913 | 2691
     26910
       3
```

The Whole Picture

```
main( )
{
   ...
   d5 = n % 10 ;  n = n / 10 ;
   d4 = n % 10 ;  n = n / 10 ;
   d3 = n % 10 ;  n = n / 10 ;
   d2 = n % 10 ;  n = n / 10 ;
   d1 = n % 10 ;  (d1=n)
   s = d1 + d2 + d3 + d4 + d5 ;
   printf ( "%i", s ) ;
}
```

10| 26/10 2

%d

		n
		26913
d5	3	2691
d4	1	269
d3	9	26
d2	6	2
d1	2	2

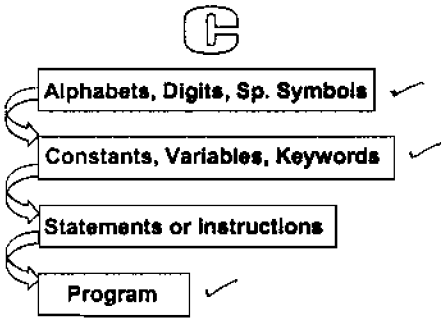
S = 21.

Use: ****

Is % (modulus) Really Useful

- Leap year or not
- Odd / Even
- Prime or not → divisible by 1 or itself.

Where Are We....



C Statements / Instructions

★ Type Declaration Instruction

Ex. int i, j, k ;
float a, b, c ;
char ch ;

Type Declaration, A few Subtleties

int a ;
a = 5 ;

is same as

int a = 5 ;

➡ int a = 5, b = 10, c = a + b * 5 % 2 ;
Here order is important

initising can also be done to expressions:

Is This Ok?

★ int a, b, c, d ;
a = b = c = d = 5 ; ✓

★ int a = b = c = d = 5 ; ✗

second is wrong as b isn't defined neither are the others.

C Statements/Instructions

★ Type Declaration Instructions

★ Arithmetic Instructions

Ex. s = d1 + d2 + d3 ;
si = j * n * r / 100 ;
c = 5.0 / 9 * f - 32 ;

+, -, *, /, % - Arithmetic Operators

For exponentiation - Use pow( )

#include <math.h> or #include "math.h"

pow(x, y)

x^y

a = pow(x, y).

Arithmetic Instructions, A Few Small Issues

① l = j * k + 3 ; ✓ Tip:
j * k + 3 = i ; ✗

② i = 3 ; ✓
3 = i ; ✗

③ a = b (c + d) ; ✗ Tip:
a = b * (c + d) ; ✓

Tip: ①

Type of Arithmetic Instructions

- ★ Integer mode AI
- ★ Real mode AI
- ★ Mixed mode AI

Integer mode AI

Ex. int a = 3, b = 4, c ;
c = a * b + 5 % 6 - b + 14 ;

* + % - + → ? arithemetic operator
= → ? assignment operator.
; → ? Terminator
c a b 5 6 14 → ? variables & constants (operands).

Legal Arithmetic Operations

Operand1	Operand2	Result
int	int	int
float	float	float
int	float	float
float	int	float

always lower datatype converts to higher.

Try This

int a ;
a = 5 / 2 ; → 2
a = 5.0 / 2 ; → 2
a = 5 / 2.0 ; → 2
a = 5.0 / 2.0 ; → 2
a = 2 / 5 ; → 0
a = 2.0 / 5 ; → 0
a = 2 / 5.0 ; → 0
a = 2.0 / 5.0 ; → 0

this is 'cos 'a' is an int variable.
(rounding off never takes place. Only trunkating takes place.).

Try This

```
float a ;
a = 5 / 2 ;      → 2.0
a = 5.0 / 2 ;    → 2.5
a = 5 / 2.0 ;    → 2.5
a = 5.0 / 2.0 ;  → 2.5
a = 2 / 5 ;      → 0.0
a = 2.0 / 5 ;    → 0.4
a = 2 / 5.0 ;    → 0.4
a = 2.0 / 5.0 ;  → 0.4
```

→ 5 & 2 are both int variable

→ 2 & 5 are both int variables.

imp

Which Is Correct?

```
float c, f = 212.0 ;

c = 5 / 9 * ( f - 32 ) ;   = 0     as 5/9 = 0 (int/int)
c = 5 / 9.0 * ( f - 32 ) ; = 100.0 as 5/9.0 int/float
c = ( f - 32 ) * 5 / 9 ;   = 100.0 as f = 212.0 = float
```

Note: Parenthesis matter

float - 32 = float

float * int / int = float.

Hierarchy/Priority/Precedence

```
c = a + b * c % 5 - d * 6
```

1, 2, 3, 4, 5, 6 (assignment).

Operators
* / %
+ -
=

which occurs earlier in the expression gets executed first.

C Instructions

- ★ Type Declaration Instructions
- ★ Arithmetic Instructions
- ★ Input / Output Instructions
  - printf( ) - Output
  - scanf( ) - Input

imp

General Form of printf( )

printf ("format string", list of variables) ;

printf ("Enter values of c and d") ;

➡ List of variables is optional.

printf ("%l %i %i", a, 35, 2 + 6 % 3) ;

➡ List can contain variables, constants or expressions.

a value
35
4

printf( )

printf ("format string", list of variables) ;

can contain

- **Format specifiers**
  - int - %l , %d
  - float - %f
  - char - %c
- **Escape sequences**
- **Any other characters** sentence.

int – %i %d

Very Imp

Numbering Systems

Decimal (0-9)	Octal (0-7)	Hexadecimal (0-9, A-F)	Binary (0-1)
0	0	0	0
1	1	1	1
.	.	.	10
9	7	9	11
10	10	A — 10	100
11	11	.	...
..	..	F — 15	111
99	77	10	1000
100	100	..	....
...	...	FF	....
...	...	100	

*

Conversions

Decl 473	$\xrightarrow{\text{decimal}}$	$4 * 10^2 + 7 * 10^1 + 3 * 10^0 = 473$
[redacted]		
Hexa 11	$\xrightarrow{\text{decimal}}$	$1 * 16^1 + 1 * 16^0 = 17$
Binary 11	$\xrightarrow{\text{decimal}}$	$1 * 2^1 + 1 * 2^0 = 3$

Octal 11 $\xrightarrow{\text{decimal}}$ $1 * 8^1 + 1 * 8^0$ = 9

**

More Conversions

Dec. 9 $\xrightarrow{\text{Octal}}$ 11	Dec. 17 $\xrightarrow{\text{Hex.}}$ 11	Dec. 3 $\xrightarrow{\text{Binary}}$ 11
8 | 9 R	16 | 17 R	2 | 3 R
8 | 1 1	16 | 1 1	2 | 1 1
0 1	0 1	0 1

R= remainder.

%d = decimal equivalent
%o = octal equivalent
%x = hexadecimal equivalent } → for small caps
%X = ~~binary~~ hexadecimal } → for capitals

Very Imp

printf( ) Makes It Handy

★ printf ("%d %o %x %X", 10, 10, 10, 10) ; → 10 12 a A
↳ o for orange

★ printf ("%d %d %d %d", 10, 12, a, A) ; → Error a & A are undeclared variable.

★ printf ("%d %d %d %d", 10, 012, 0xa, 0XA) ; → 10 10 10 10.
0-zero

★ printf ("%x %o", 10.55, 3.14) ; (error) → as they work only for integers not decimals.

★ printf ("%x %d", 088, 088) ; → in octal 1-7 no 8.
(Error)

Printf() → Print f = format. → helps to control output.

Escape Sequences

★ \n - Newline
★ \t - Tab
★ printf ("%d\n%d\t%d", 10, 20, 30) ;

10
20 30.
\n → cursor to begning of new line
\t → 8 col to the right.

Any Other Characters

★ printf ("Enter values of c and d") ;
★ printf ("Hello") ;
★★★★★ printf ("\nSimple Interest = Rs. %f", si) ;

Escape sequence
Any other character
Format specifier

→ –
Simple Interest = Rs. (value of si).

scanf() → scan f=format.

scanf()

★ scanf ("format string", list of variables) ;
★ scanf ("%d %f %c", &l, &a, &ch) ;

Only format specifiers

a space can be given btw %d & %f
not btw " & %d.

C Instructions

- Type Declaration
- Arithmetic Instruction
- Input/Output Instruction
- Control Instructions

@Koshay hal sheet -III

C Instructions

- Type Declaration Instructions
- Arithmetic Instructions
- Input / Output Instructions
- Control Instructions

Control Instructions

- What are they?
  - Control the sequence of execution of instructions
- What different types?
  - Sequence
  - Decision
  - Repetition(loop)
  - Case

Normal C Program

```
main( )
{
   int ..... ; float ..... ;
   printf ( ..... ) ;
   scanf ( ..... ) ;
   a = .... ;
   b = .... ;
   printf ( .....) ;
}
```

Tip: Sequence CI is the default CI

Decision Control Instruction

```
/* Calculation of total expenses */
main( )
{
    int qty ;
    float price ;
    printf ( "Enter quantity and price" ) ;
    scanf ( "%d%f", &qty,  &price ) ;
    ...
    ...
}
```

```
/* Calculation of total expenses */
main( )
{
    int qty ; float price ;  int dis ; float totexp ;
    printf ( "Enter quantity and price" ) ;
    scanf ( "%d%f", &qty, &price ) ;
    if ( qty >= 1000 )
        dis = 10 ;
    else
        dis = 0 ;
    totexp = qty * price - qty * price * dis / 100 ;
    printf ( "Total expenses = Rs %f", totexp ) ;
}
```

tab

tab

200 15.50

1200 15.50

Tips

- if, else - Keywords
- General form:

```
if ( condition )
    statement1 ;
else
    statement2 ;
```

Relational Operators

- Cond. is *usually* built using <, >, <=, >=, = =, !=

a = b Assignment

a = = b Comparison used with if or while statement.

imp

Would This Work?

```
int a = 3, b = 4, c, d ;
printf ( "%d", a + b ) ;
printf ( "%d", a <= b ) ;
c = a * b ;
printf ( "%d", c ) ;
d = a == b ;
printf ( "%d", d ) ;
```

Tip: Condition - True - Replaced by 1
Condition - False - Replaced by 0

Printf("%d", a<=b); a=3 b=4.
a is <= b ∴ output = 1
as true = 1 false = 2.

a==b will be done first. This is false
∴ equal to zero ∴ d = 0

Is it Monday or Tuesday

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

garbage value

Tip: Unless specifically initialised a variable contains a garbage value.

always changes. no control over the value.

```
/* Calculation of Total Expenses */
main( )
{
    int qty ; float price ; int dis = 0 ; float totexp ;
    printf ( "Enter quantity and price" ) ;
    scanf ( "%d%f" , &qty, &price ) ;
    if ( qty >= 1000 )
        dis = 10 ;
    totexp = qty * price - qty * price * dis / 100 ;
    printf ( "Total expenses = Rs %f", totexp ) ;
}
```

200 15.50

1200 15.50

Tip: else block is optional

here else isn't required as if condition is false default value of dis is used. Here dis is initialised to zero. If dis not initialized to zero then garbage value will be used.

```
/* Calculation of gross salary */
main( )
{
    float bs, hra, ca, da, gs ;
    printf ( "Enter basic salary" ) ;
    scanf ( "%f", &bs ) ;

    if ( bs >= 1500 )
        da = bs * 95 / 100 ;
        hra = bs * 20 / 100 ;
        ca = bs * 12 / 100 ;
    else
        da = bs * 92 / 100 ;
        hra = bs * 15 / 100 ;
        ca = 200 ;
    gs = bs + hra + da + ca ;
    printf ( "Gross salary = Rs. %f", gs ) ;
}
```

Tip: - Hanging else not allowed

would be zero if as/100 * bs as int/int * float → 0 * float = 0

only this belongs to if block

only this belongs to else block

here division is first ∴ 0.

Error in prog as no braces for If or Else. Here Else is hanging as it is not clear as to which all statement ~~then~~ belongs to Else or If. After If 2 more statements are included & then Else. Now Else can't be without if.

imp

```
/* Calculation of gross salary */
main( )
{
    float bs, hra, ca, da, gs ;
    printf ( "Enter basic salary" ) ;
    scanf ( "%f", &bs ) ;
    if ( bs >= 1500 )
    {
        da = ... ; hra = ... ; ca = ... ;
    }
    else
    {
        da = ... ; hra = ... ; ca = ... ;
    }
    gs = bs + da + hra + ca ;
    printf ( "Gross salary = Rs %f", gs ) ;
}
```

adding braces specifies the limit of each block.

One More Form

```
if ( condition )
{
    statement1 ;
    statement2 ;
}
else
{
    statement3 ;
    statement4 ;
}
```

Tip: - Default scope of if and else is only the next statement after them.

- To override the default scope use { }

(**)

Leap Year or Not

```
main( )
{
   int y ;
   printf ( "Enter year" ) ;
   scanf ( "%d", &y ) ;
   ...
}
```

4	1996	499	4	2000	500	4	1900	475
	1996			2000			1900	
	0			0			0	
	Leap			Leap			Not Leap	

Rule → to find leap year
① find whether it is a century year (ending in 2 zeros)
② if so div by 400
else div by 4.

Imbedded if-else is okay.

```
main( )
{
   ...
   if ( y % 100 == 0 )
   {
      if ( y % 400 == 0 )
         printf ( "Leap" ) ;
      else
         printf ( "Not Leap" ) ;
   }
   else
   {
      if ( y % 4 == 0 )
         printf ( "Leap" ) ;
      else
         printf ( "Not Leap" ) ;
   }
}
```

Nested *if-else* statements are legal

Arithmetic operators enjoy higher priority than assignment

(***)

First Day of Any Year

```
main( )
{
   int y ;
   printf ( "Enter year" ) ;
   scanf ( "%d", &y ) ;  ⇐ 1998
   ...
}
```

The Logic Behind It

1/ 1/ 1 ⇨ Monday
8/ 1/ 1 ⇨ Monday
15/ 1/ 1 ⇨ Monday
22/ 1/ 1 ⇨ Monday
29/ 1/ 1 ⇨ Monday
...
...
...
1/ 1/ 1998 ⇨ ?

1 /1 / 1 Monday
29/ 1 / 1 ?

1/ 1/ 1 to 28/1 / 1

28 % 7 ⇨ 0

1/ 1/ 1 to 31/ 12/ 1997

? % 7 ⇨ ?

Imp

here all int cos no decimal. but in ① we are exceeding the range ∴ all variables shouldn't be int.

```
main( )
{
  int y ; firstday, normaldays, leapdays, totaldays ;
  printf ( "Enter year" ) ;
  scanf ( "%d", &y ) ;
  normaldays = ( y - 1 ) * 365 ; →①
  leapdays =( y - 1 ) / 4 - ( y - 1 ) / 100 + ( y - 1 ) / 400 ;
  totaldays = normaldays + leapdays ; ③
  firstday = totaldays % 7 ;
  if ( firstday == 0 )
      printf ( "Monday" ) ;
  ...
  ...
}
```

1/ 1/ 1 to 28/ 1/ 1

28 % 7

0

1/ 1/ 1 to 31/ 12/ 1997

? % 7 ⇨ ?

① since every year has atleast 365 days

② (y-1)/4 → choosing leap year.
-(y-1)/100 → 'cos 1900 isn't leap ∴ eliminating all century years
+(y-1)/400 → 2000, etc also get eliminated ∴ adding them back.

③ calculating Total no. of days.

```
main( )
{
  int y, firstday, leapdays ;
  long int normaldays, totaldays ;
  ...
  normaldays = ( y - 1 ) * 365 ;
  ...
}
```

int	-32768 to +32767
long int	-2147483648 to +2147483647

What's Wrong Here?

```
normaldays = ( y - 1 ) * 365 ;

float a ;          float a ;
a = 5 / 2 ;        a = 5 / 2.0 ;
[a = 2 ]           [a = 2.5]
```

Soln: normaldays = (y - 1) * 365L ; → 365 becomes a long integer. (capital or small case (L, l))

Soln: normaldays = (y - 1L) * 365 ; → 1 becomes a long integer.

Soln: long int y ;
scanf ("%ld", &y) ;
normaldays = (y - 1) * 365 ; → y becomes a long integer (declaring Y as ~~an~~ a long int)

→ %ld long int.

→ worst solution as y for only one step occupies a larger range throughout the prog.
Solution 1 & 2 are better.
Also Y will NEVER exceed 32767.

The sizeof( ) Operator

```
int i ;
long int j ;
printf ( "%d %d", sizeof ( i ), sizeof ( j ) ) ;
printf ( "%d %d", sizeof ( int ), sizeof ( long int ));
```

Keyword — Operator

Size off is not a Keyword but an operator.

Size off [how many bits of memory is being occupied]

Printf ("%d", sizeof (float))
is ok as sizeof is ~~a pred~~ always going to give a one digit no
∴ int is enough ie %d.

IMP

Are You Sure?

4 % 3 ⇒ 1
3 % 4 ⇒ 3
-3 % 4 ⇒ -3
3 % -4 ⇒ 3
-3 % -4 ⇒ -3

Tip: Sign of remainder is same as sign of numerator

sign dependent on Nr: sign.

```
int a = 5, b = 5, c = 5 ;
if ( a == b == c )
   printf ( "Hello" ) ;
else
   Printf("Hi");
```

output is Hi

a==b is correct ∴ True ∴ 1

now 1==c is false
∴ else will be called.

square root of any no. sqrt(n).

% func doesn't work with float

```
Printf ("\" ");     output = "
Printf ("\\ ");     output = \
```

braces are scope delimitors.

Arithmetic operators have higher priority than assignment operator (=).

Else matches with the nearest If above it.

1/01/0001 → was a Monday.

-ve $\xrightarrow{\text{exceed}}$ +ve $\xrightarrow{\text{exceed}}$ -ve ⟶ +ve. (like a circle).
if +ve range exceeded -ve range begins.

sign of answer wrt modulus is always the same as numerator.

Abhay Lal
Sheet IV

What Will Be The Output

```
main( )
{
  int a = 5, b = 10 ;
  if ( a >= 20 ) ;
     b = 30 ;
  printf ( "%d", b ) ;
}
```

→ if condition is true or false dont do ntn.

; - Null statement
Doesn't do anything on execution

output:-
b=30

-no error will be observed.
-but prog dont work lik it should

What Will Be The Output

```
main( )
{
  int a = 10 ;
  if ( a = 5 )
     printf ( "Hi" ) ;
}
```

Tip : Truth in C is nonzero, whereas,falsity is zero.

if (a = 5) → if (a) → if (5)

value of a gets turned to 5.

```
main( )
{
   int m1, m2, m3, m4, m5 ;  int per ;
   printf ( "Enter marks in five subjects" ) ;
   scanf ( "\n%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5 );
   per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
   if ( per >= 60 )
      printf ( "First division" ) ;
   else
   {
      if ( per >= 50 )
         printf ( "Second division" ) ;
      else
      {
         if ( per >= 40 )
            printf ( "Third division" ) ;
         else
            printf ( "Fail" ) ;
      }
   }
}
```

no '\n'

Three Problems

```
main( )
{
    int m1, m2, m3, m4, m5, per ;
    printf ( "Enter marks in five subjects" ) ;
    scanf ( "%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5 ) ;
    per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
    if ( per >= 60 )
        printf ( "First division" ) ;
    if ( per >= 50 )
        printf ( "Second division" ) ;
    if ( per >= 40 )
        printf ( "Third division" ) ;
    else
        printf ( "Fail" ) ;
}
```

Anything Wrong?

if value >60
all 3 printf's executed

```
main( )
{
    int m1, m2, m3, m4, m5, per ;
    printf ( "Enter marks in five subjects" ) ;
    scanf ( "%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5 ) ;
    per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
    if ( per >= 60 )
        printf ( "First division" ) ;
    if ( per >= 50 && per < 60 )
        printf ( "Second division" ) ;
    if ( per >= 40 && per < 50 )
        printf ( "Third division" ) ;
    else
        printf ( "Fail" ) ;
}
```

logical AND operator

Still Anything Wrong?

for a value of 65
the last if statement ends up
failing the poor sod.

```
main( )
{
    int m1, m2, m3, m4, m5, per ;
    printf ( "Enter marks in five subjects" ) ;
    scanf ( "%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5 ) ;
    per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
    if ( per >= 60 )
        printf ( "First division" ) ;
    if ( per >= 50 && per < 60 )
        printf ( "Second division" ) ;
    if ( per >= 40 && per < 50 )
        printf ( "Third division" ) ;
    if ( per < 40 )
        printf ( "Fail" ) ;
}
```

Is This Better?

Yep its much better.

Logical Operators

&& - And
|| - Or
! - Not

Useful in

- Checking Ranges — if variable greater or smaller than given length.
- Yes / No Problem — counter check

```
main( )
{
   int age ; char s, ms ;
   printf ( "Enter age, sex, marital status" ) ;
   scanf ( "%d%c%c", &age, &s, &ms ) ;
   if ( ms == 'm' )
      printf ( "Insured" ) ;
   else
   {
      if ( s == 'm' )
      {
         if ( age > 30 )
            printf ( "Insured" ) ;
         else
            printf ( "Not Insured" ) ;
      }
      else
      {
         if ( age > 25 )
            printf ( "Insured" ) ;
      else
         printf ( "Not In." ) ;
      }
   }
}
```

26 m u

m is a char

to compare char variable with anything use single quotes.

Output → not insured.

here too many if's & elses ; too many indentations; too many braces.

Program Using Logical Operators

```
main( )
{
   int age ;
   char s, ms ;
   printf ( "Enter age, sex, marital status" ) ;
   scanf ( "%d%c%c", &age, &s, &ms ) ;
   if ((ms == 'm') || ( ms == 'u'  &&  s == 'm'  &&  age > 30 )
      || ( ms == 'u'  &&  s == 'f'  &&  age > 25 ))
      printf ( "Insured" ) ;
   else
      printf ( "Not Insured" ) ;
}
```

26 m u

① ② ③

Parenthesis not really required. only for better readability.

either ①, ② or ③

2a + 2b + 2c must be true

///ly 3a + 3b + 3c must be true.

Output :→ Not Insured.

Working of && And ||

cond1	cond2	cond1 && cond2	cond1 || cond2
True	True	True	True
False	False	False	False
True	False	False	True
False	True	False	True

A	B	AND (•)	OR (+)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Imp

Hierarchy of Operators

uniary operator

Operators	Type
!	Logical Not
* / %	Arithmetic
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical And
||	Logical Or
=	Assignment

Increasing Priority

Other Unary Operators → - + & sizeof

Exchanging Blocks

```
main( )
{
   int a = 3, b = 4 ;
   if ( a <= b )
       printf ( "A" ) ;
   else
       printf ( "B" ) ;
}
```

Output: A

```
main( )
{
   int a = 3, b = 4 ;
   if ( a > b )
       printf ( "B" ) ;
   else
       printf ( "A" ) ;
}
```

Output: A

error:—

One More Way

```
main( )
{
  int a = 3, b = 4 ;
  if ( a <= b )
      printf ( "A" ) ;
  else
      printf ( "B" ) ;
}
```

Output: A

```
main( )
{
  int a = 3, b = 4 ;
  if ( ! a > b )
      printf ( "A" ) ;
  else
      printf ( "B" ) ;
}
```

Output: B

if a NOT greater than b.

a is a value besides zero

∴ not a = zero

0>b is false

∴ output B.

The Correct Way

```
main( )
{
  int a = 3, b = 4 ;
  if ( ! ( a > b ) )
      printf ( "A" ) ;
  else
      printf ( "B" ) ;
}
```

Output: A

if (a>b) NOT.

Yet Another Way

```
main( )
{
  int a = 3, b = 4 ;
  if ( a > b )
      printf ( "B" ) ;
  else
      printf ( "A" ) ;
}
```

Output: A

```
main( )
{
  int a = 3, b = 4 ;
  if ( ! ( a <= b ) )
      printf ( "B" ) ;
  else
      printf ( "A" ) ;
}
```

Output: A

What Would be The Output

```
main( )
{
  int a = 3, b = - 4, c ,d ;
  c = !a ;
  d = !b ;
  printf ( "%d %d %d %d", c, d, a, b ) ;
}
```

~~b=~~
c=0
d=0
a=3
b=-4

Point Out The Error

```
main( )
{
  int a = 3, b = 4, c ;

  c = !a || b = 7 ;
  printf ( "%d %d", b, c ) ;
}
```

assignment ~~when ad~~ ~~should be~~ comparing.

[error: L value is required]

L value required:-
c equals to !a (0) OR b
(which is 4).
∴ c = 0 || 4 = 7
false OR true = true
1 0 or 1 = 1
∴ c = 1 = 7
∴ Left hand side value req.

What Would be The Output

```
main( )
{
  int a = 3, b = 4, c ;
  c = !a || ( b = 7 ) ;
  printf ( "%d %d", b, c ) ;
}
```

b=7
c=1
c = 0 || 7
false or true
true
c=1.

Yet Another Way

```
main( )
{
   int a = 3, b = 4 ;
   if ( a <= b )
       printf ( "A" ) ;
   else
       printf ( "B" ) ;
}
```

Output: A

```
main( )
{
    int a = 3, b = 4 ;
    a <= b ? printf ( "A" ) : printf ( "B" ) ;
}
```

Conditional operators

How are they different?

if True False

Expression ? state 1 : state 2

→ better ① Compact
② no indentations
③ no braces.

? and : are conditional operators.
they are tertiary operator as 3 thingys required.

How Would You Convert

```
main( )
{
   int a = 3, b = 4 ;
   if ( a <= b )
       printf ( "A" ) ;
}
```

①

```
main( )
{
   int a = 3, b = 4 ;
   a <= b ? printf ( "A" ) ;
}
```

②

if True ~~on~~ then do something else get lost

but for expression ②
Error as 3rd (state 2) expression is missing

Remove The Error

```
main( )
{
   int a = 3, b = 4 ;
   a <= b ? printf ( "A" ) ;
}
```

~~or what~~ ~~3rd~~ 3 expression missing

→ ~~is okay~~

as if ntn is to be done null statement is to be added after : else Error.

No Errors At Last

also wrong

```
main( )
{
  int a = 3, b = 4 ;
  a <= b ? printf ( "A" ) : ;
}
```

Null Statement

terminator

Null statement

Terminator

Dummies

is right

Soln: a <= b ? printf ("A") : printf (" ") ;
a <= b ? printf ("A") : (z = 3) ;

no functional values ∴ Dummies

assignment

1st ; as terminator

2nd ; as Null statement

Moral of The Story

- Use one, use another
- Dummy statement
- Not a replacement for *if-else*

Only 1 statement in ? part and one in : part

Some More Different Ways

```
main( )
{
    int a = 3, b ;
    a <= 5 ? ( b = 10 ) : ( b = 20 ) ;
    printf ( "%d", b ) ;
}
```

( ) Necessary

1

2

```
b = a <= 5 ? 10 : 20 ;
printf ( "%d", b ) ;
```

L value required if ~~brackets not given in Expression~~ 1

```
printf ( "%d", a <= 5 ? 10 : 20 ) ;
```

3

2 is fine a < has greater priority than = .

Important Sheet

@Rohay Lal
Sheet - V

(imp)

Conversions

```
main( )
{
   int  a = 10, b = 20 ;
   float  c = 3.14 , d = 6.28  ;
   printf ( "%d %d %f %f ", a, b, c, d ) ;
   printf ( "%d %d %f %d ", a, b, c, d ) ;
   printf ( "%f %d %f %f ", a, b, c, d ) ;
}
```

10 20 3.14 6.28 — 3.140000 (6 decimal) 6.280000

10 20 3.14 0

2.68e154 | 7864 | 0.000 | -1.827e259

garbage value.

(if one goes wrong all further go wrong).

(imp)

Conversions Continued...

```
main( )
{
   int  i = 65, j = 90 ;
   char  ch = 'A', dh = 'Z' ;
   printf ( "%d %d %c %c", i, j, ch, dh ) ;
   printf ( "%c %c", i, j ) ;
   printf ( "%d %d", ch, dh ) ;
}
```

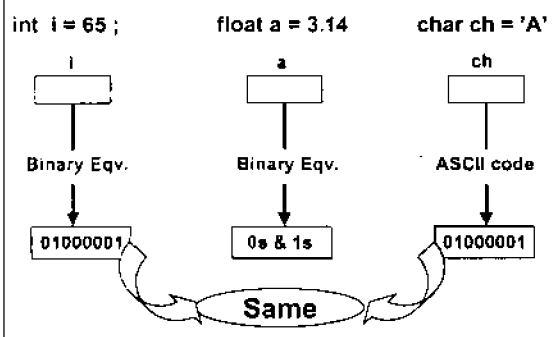
65 90 A Z

A Z

65 90

∴ 90 & Z } are related
65 & A

Representation



American Standard Code for Information Interchange.
(helps in std all binary values for all characters).

ASCII Codes

01000001	A
01000010	B
01000011	C
01000100	D
01000101	E
....	..
....	..

ASCII codes.

ASCII values

Binary Is Difficult

Character	Binary	Decimal
A	01000001	65
B	01000010	66
C	01000011	67
D	01000100	68
E	01000101	69
..	....	..

Z 90

ASCII code – 8 bit (as 8 digits)

Methods Are Different

int i = 65 ; char ch = 'A' ;

i: 01000001 ch: 01000001

printf ("%c", i) ; ⟶ A
printf ("%d", ch) ; ⟶ 65 .

① what ever binary is present in given variable output the respective char/int/float.

(imp)

ASCII codes are present for the following:

Characters to be Represented

+ 26	Capital letters (A- Z)
+ 26	Smallcase letters (a - z)
+ 10	Digits (0 - 9)
+ 32	Special Symbols (+ - * / . : ; etc.)
+ 34	Non-printable chars - control chars
+ 128	Graphic characters
256	Characters

00000000 To 11111111

combinations are different.

eg: tab, return(enter), spacebar, escape key,

All ASCII codes is 8 bits.
[1 binary digit = 1 bit]

$\therefore 2^8$ combination = 256.

00000000 $\xrightarrow{\text{convert}}$ 0

11111111 $\xrightarrow{\text{convert}}$ 255

0 → 255
= 256.

(imp)

What About More Than 255

```
int i = 300, j = 65 ;
printf ( "%c", i ) ;
printf ( "%c", j ) ;
```

binary equivalent of i

Higher	Lower
00000001	00101100

Only lower byte gets used

j

higher	lower
00000000	01000001

(16 bit)

8bit 8bit

There are only 256 char as ASCII is an 8 bit code. $\therefore 2^8 = 256$ diff combinations ∴ not more than 256.

a char corresponding to the lower level 8bit code of 300.

→ A

Chooses the lower byte & prints a char corresponding to that 8 bit.

Why Unreliable Conversions

```
float i = 3.14 ;
printf ( "%c", i ) ;
printf ( "%d", i ) ;
```

i

0s and 1s	0s and 1s	0s and 1s	0s and 1s

```
int i = 35 ;
printf ( "%f", i ) ;
```

i

0s and 1s	0s and 1s		

on doing %c lowest byte will be used

on doing %i lowest & second lowest byte will be used

on doing %f → problem as float is 4 bit & available only 2 byte.

Surprised?

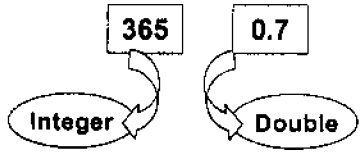
```
main( )
{
    float a = 0.7 ;
    if ( a < 0.7 )
       printf ( "A" ) ;
    else
       printf ( "B" ) ;
}
```

output = A.

why? →

Appearances Are Misleading

```
main( )
{
    float a = 0.7 ;
    printf ( "%d%d", sizeof ( a ), sizeof ( 0.7 ) ) ;
}
```

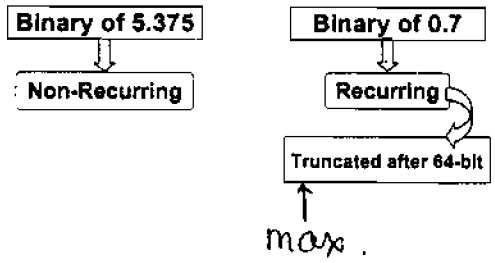


∴ output = (4, 8)

0.7 isn't a float variable but a double.

365 → default = Integer
0.7 → default = double.
0.7f → float
365L → long int.

Binary of A Float



value wise 32bit value < 64bit.

32 - bit Recurring binary equivalent of 0.7	<	64 - bit Recurring binary equivalent of 0.7

```
float  a = 0.7 ;
if ( a < 0.7 )
   printf ( "A" ) ;
else
   printf ( "B" ) ;
```

1.33 < 1.3333

double is 64 bit

a is a float 32 bit.

∴ if (a<0.7)

(float binary value) < (double binary value)

True ∴

Output = A.

Overriding Defaults

365 ⟷ 365L

0.7 ⟷ 0.7f

```
float  a = 0.7 ;
if ( a  <  0.7f )
   printf ( "A" ) ;
else
   printf ( "B" ) ;
```

converting double to float.

output → B

0.7 binary is recurring.

Exact Binary

```
main( )
{
    float a = 5.375 ;
    if ( a < 5.375 )
       printf ( "A" ) ;
    else
       printf ( "B" ) ;
}
```

(binary double / float < binary)

output : B

1.5 = 1.5000

5.375 binary is non recurring.

5.375 is double

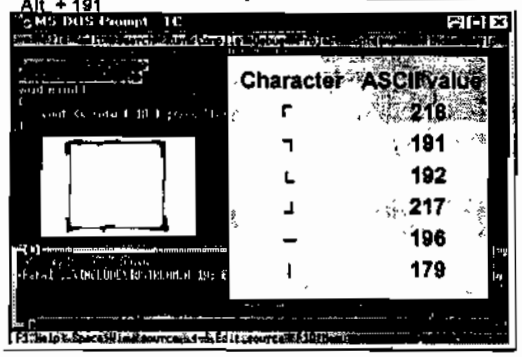
a is float.

but since 1.5 = 1.5000

∴ output B

Graphics Characters

Generate using Alt + 191



Character	ASCII value
┌	218
┐	191
└	192
┘	217
─	196
│	179

characters through generation of ASCII codes ~~characters~~
Alt + keypad no.

(Imp)

ASCII Values

+ 26	Capital letters	65 - 90
+ 26	Smallcase letters	97 - 122
+ 10	Digits	48 - 57
+ 32	Special Symbols	34 – 47, 58 – 64, 91 – 96, 123 – 127
+ 34	Non-printable chars	0 - 33
+ 128	Graphic characters	128 - 255
256	Characters	0 - 255

diff categories:

Control Instructions

- Sequence
- Decision - if - else
  - ? :
  - < > <= >= == !=
  - && || !
- Repetition / Loop control instruction

To Begin With...

```
main( )
{
   int p, n ;
   float r, si ;
   printf ( "Enter values of p, n, r" ) ;
   scanf ( "%d%d%f", &p, &n, &r ) ;
   si = p * n * r / 100  ;
   printf ( "Simple interest = Rs %f", si ) ;
}
```

Repeat with *while*

Tip : while - Keyword

```
main( )
{
  int  p, n ;
  float  r, si ;

  while
      printf ( "Enter values of p, n, r" ) ;
      scanf ( "%d%d%f", &p, &n, &r ) ;
      si = p * n * r / 100  ;
      printf ( "Simple interest = Rs %f", si ) ;
}
```

while = so long as condition remains true.

Any Condition

```
main( )
{
  int  p, n, i ;
  float  r, si ;

  while ( i <= 10 )
      printf ( "Enter values of p, n, r" ) ;
      scanf ( "%d%d%f", &p, &n, &r ) ;
      si = p * n * r / 100  ;
      printf ( "Simple interest = Rs %f", si ) ;
}
```

i value is not initialised
∴ logical error.

Initialization Necessary

```
main( )
{
  int  p, n, i ; float  r, si ;
  i = 1 ;
  while ( i <= 10 )
      printf ( "Enter values of p, n, r" ) ;
      scanf ( "%d%d%f", &p, &n, &r ) ;
      si = p * n * r / 100  ;
      printf ( "Simple interest = Rs %f", si ) ;
}
```

loop executed – Error (infinite) as i value not incremented.

So Also Incrementation

```
main( )
{
  int  p, n, i ; float  r, si ;
  i = 1 ;
  while ( i <= 10 )
      printf ( "Enter values of p, n, r" ) ;
      scanf ( "%d%d%f", &p, &n, &r ) ;
      si = p * n * r / 100  ;
      printf ( "Simple interest = Rs %f", si ) ;
      i = i + 1 ;
}
```

no braces for while
∴ Error.
infinite loop
Here also default only one statement related to while.

Scope And Default Scope

```
main( )
{
  int  p, n, i  = 1 ;  float  r, si ;
  while ( i <= 10 )
  {
      printf ( "Enter values of p, n, r" ) ;
      scanf ( "%d%d%f", &p, &n, &r ) ;
      si = p * n * r / 100  ;
      printf ( "Simple interest = Rs %f", si ) ;
      i = i + 1 ;
  }
}
```

Logic Doesn't Matter

```
main( )
{
  int  i  ;
  i = 1 ;  /* Initialisation of loop counter */
  while ( i <= 10 ) /* Testing of  loop counter */
  {
     /* any logic */
     i = i + 1 ;   /* Incrementation of loop counter*/
  }
}
```

Loop Counter / Index variable

the counter logic is the most important logic in the prog as the no. of times the prog gets executed depends on the counter

i = loop counter / index variable

Char i = 'A'

(i <= 'z') i = i+1

is OR

as in i binary stored as 65.
z = 90
+1 will make binary of B
+1 " " " " " C
⋮ z.

More On Loop Counters

- Loop counters can be integers, long integers, floats or characters
- Loop counters can be incremented or decremented by any suitable step value

Another Program

```
/* Print  numbers  from 1 to 10 */
main( )
{
   int  i = 1 ;
   printf ( "%d", i ) ;
}
```

Put it in a Loop

```
/* Print numbers from 1 to 10 */
main( )
{
    int  i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d", i ) ;
        i = i + 1 ;
    }
}
```

Another Way

```
/* Print numbers from 1 to 10 */
main( )
{
    int  i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d", i ) ;
        i ++ ;
    }
}
```

Same as ++i ;

i++ /// i=i+1.
++i /// i=i+1
} they both increment value of i

Incrementation / Decrementation Operators

- ++ Increases value of a variable by 1
- -- Decreases value of a variable by 1
- **
  //
  %%

Don't make sense

Are They OK

- i = i + 1 → i++ ✓
- i = i + 2 → i+++ ✗
- i = i + 10 → ? ✗

Tip : +++ doesn't exist

***. Imp

Would This Work

- i = j---2 ; ☑
  - i = j-- - 2 ;
- i = (j + k)++ ; ☒

- int i, j = 3 ;
  - i = --j ; ☑
  - i = --3 ; ☑
- i = j----2 ; ☑
- i = j----2 ; ☒

① i = j---2 /// (j--) -2.

② i = --3 /// i = - (-3) don't work

--j ⇒ j = j-1

--3 ⇒ 3 = 3-1 — ? nonsence.

③ i = (j+k)++ (j+k =) j+k+1 → wrong.

→ space btw 3rd & 4th minus.

→ j-- (--2) — wrong.

Space Matters

i = j----2 ; ☒

i = j-- - -2 ; ☑

Optional — Complusory

i = j-- - (-2)

int → char conversion is predictable
char → int conversion is predictable.

All characters have Ascii codes.

@Rohan Lal VI th sheet.

Is i++ Same As ++i

```
main( )
{
  int  i = 1 ;
  while ( i <= 10 )
      printf ( "%d", i++ ) ;
}
```

1 st oper. - Printing
2 nd oper. - Incrementation

Output : 1 2 3 4 .....9 10

i++ :- first printing value of i then incrementation.

Using ++i

```
main( )
{
  int  i = 1 ;
  while ( i <= 10 )
      printf ( "%d", ++i ) ;
}
```

1 st oper. - Incrementation
2 nd oper. - Printing

Output : 2 3 4 5 .....9 10 11

++i :- hose first incrementation then printing.

The Correct Way

```
main( )
{
  int  i = 0 ;
  while ( i < 10 )
      printf ( "%d", ++i ) ;
}
```

Output : 1 2 3 4 .....9 10

Moral?

```
main( )
{
  int i = 1 ;
  while ( i <= 10 )
    printf ( "%d", i++ ) ;
}
```

```
main( )
{
  int i = 0 ;
  while ( i < 10 )
    printf ( "%d",++ i ) ;
}
```

Two More Ways

```
main( )
{
  int i = 0 ;
  while ( i++ < 10 )
      printf ( "%d", i ) ;
}
```

1 st oper. - Testing
2 nd oper. - Incr

```
main( )
{
  int i = 0 ;
  while ( ++i <= 10 )
      printf ( "%d", i ) ;
}
```

1 st oper. - Incr
2 nd oper. - Testing

o/p 1 .. 10 ÷ o/p 1 .. 10

i++ post increment operator
++i pre-increment operator.

Compare

```
main( )
{
  int i = 1 ;
  while ( i <= 10 )
      printf ( "%d", i++ ) ;
}
```

```
main( )
{
  int i = 0 ;
  while ( i < 10 )
      printf ( "%d",++ i ) ;
}
```

```
main( )
{
  int i = 0 ;
  while ( i++ < 10 )
      printf ( "%d", i ) ;
}
```

```
main( )
{
  int i = 0 ;
  while ( ++i <= 10 )
      printf ( "%d", i ) ;
}
```

diff ways of writting all no. from 1 to 10.

Print Nos. From 1 to 10

```
main( )
{
    int i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d", i ) ;
        i += 1 ;
    }
}
```

Same as i++, ++i, i = i + 1

a = a * 10 ; → a *= 10 ;

b = b % 5 ; → b %= 5 ;

+= -= *= /= %= Compound Assignment Oper.

i+=1 ; < i++ / ++i } depending on value.

a*=10 → a = a*10 ;

b%=5 → b = b%5 ;

One More Way

```
main( )
{
    int i = 1 ;
    while ( 1 )  /* Repeat infinite times */
    {
        printf ( "%d", i ) ;
        i++ ;
    }
}
```

Any Non-Zero number

as true = 1 or any other value +ve/-ve
false = 0
now if always 1 or any other no.
it will always occur.

Applying Brakes

```
main( )
{
    int i = 1 ;
    while ( 1 ) /* Repeat infinite times */
    {
        printf ( "%d", i ) ;
        i++ ;
        if ( i > 10 )
            break
    }
}
```

break - Terminates Loop
exit( ) - Terminates Program

if (i > 10) exit( ) ;

Keyword

break :- termination of loop.
exit() ; → terminator of program.
↑
more powerful

imp

Calculate

$n!$ ⟹ $1 * 2 * 3 * 4 * \ldots n$ — Running product

$\sum_{i=1}^{n} i$ ⟹ $1 + 2 + 3 + 4 + \ldots n$ — Running sum

x^n ⟹ $2 * 2 * 2 * 2 * \ldots n$ — Running product

imp

Running Sum & Products

```
main( )
{
    int s, p, pr, i, n ;
    scanf ( "%d", &n ) ;
    i = 1 ; s = 0 ; p = 1 ; pr = 1 ;
    while ( i <= n )
    {
      s = s + i ;
      p = p * i ;
      pr = pr * 2 ;
      i++ ;
    }
    printf ( "%d %d %d", s, p, pr ) ;
}
```

5

i	s
1	0
2	1
3	3
4	6
5	10
6	15

terminator.

dry run of prog.

1 + 2 + 3 + 4 + ...n	RS
1 * 2 * 3 * 4 * ....n	RP
2 * 2 * 2 * 2 * ....n	RP

120

15 32

$$\frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots\ldots\ldots\ldots 10 \text{ terms}$$

```
main( )
{
    scanf ( "%f", &x ) ; i = 1 ; s = 0 ;
    while ( i <= 10 )
    {
        calculate numerator
        calculate denominator
        term = numerator / denominator
        Add / subtract alternate terms to/from s
        i++ ;
    }
    print ( s ) ;
}
```

good method !!

declaration of n is done twice.

```
main( )
{
  float x, s, n, d, t ; int i, j, k, n ;
  scanf ( "%f ", &x ) ;
  i = 1 ;  s = 0 ;
  while ( i <= 10 )
  {
      j = 2 * i - 1 ;
      k = n = d = 1 ;
      while ( k <= j )
      {
          d = d * k ;  n = n * x ;
          k++ ;
      }
      t = n / d ;
      ( i % 2 == 0 ) ? s = s - t : ( s = s + t ) ;
      i++ ;
  }
  printf ( "%f ", s ) ;
}
```

OK ?

$$\frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} -$$

Associates from R to L

i	j
1	1
2	3
3	5
4	7
5	9
..	..

Anything wrong ?

OK ?

1 goes to d then n then k.

while assigment parenthesis required.

(s = s-t) : (s = s+t);

// working happens in a Calculator.

Prime Number or Not

```
main( )
{
  int n ;
  printf ( "Enter any number" ) ;
  scanf ( "%d", &n ) ;
  ...
  ...
}
```

13

2 3 4 12

Prime no. [divisible by 1 or itself ONLY]

good Method !!

Prime No.

```
main( )
{
    int n ; int i = 2 ;
    printf ( "Enter any number" ) ;
    scanf ( "%d", &n ) ;
    while ( i <= n - 1 )
    {
        if ( n % i = = 0 )
            printf ( "Not a prime number" ) ;
        else
            i++ ;
    }
}
```

13

2 3 4 12

```
main( )
{
  int n, i = 2 ;
  printf ( "Enter any number" ) ;
  scanf ( "%d", &n ) ;
  while ( i <= n - 1 )
  {
     if ( n % i == 0 )
     {
        printf ( "Not a prime number" ) ;
        break ;
     }
     else
        i++ ;
  }
  if ( i == n )
     printf ( "Prime number" ) ;
}
```

13

when → if (i==n) → dont exits

should be exit();

as if Not a Prime No. is printed then break takes you out of loop & Prime no. gets printed.

Generate Prime Numbers

2	17
3	19
5	23
7	.
11	.
13	199

→ H.W.

if ++b is true.

i = ++a && ++b ~~&&~~ ++c

goes to work if ++a is true

Very Imp ***

What Would Be The O/P

```
b = ( ++i && ++j ) || ++k ;
main( )
{
  int i = 5, j = 4, k = -1, a, b ;
  a = i && !j ;
  b = ++i && ++j || ++k ;
  printf ( "%d %d %d %d %d", a, b, i, j, k ) ;
}
```

always

```
b = ++i && ( ++j || ++k );
```

```
0  1  6  5  -1
```

```
i = ++a && ++b && ++c ;
j = ++a || ++b || ++c ;
```

!j = 0 (! of true = 0)

i = true = 1 1 && 0 true && 0 = false.

∴ a = false = 0.

b = (++i && ++j) || ++k

6 && 5

true && true

true || 0

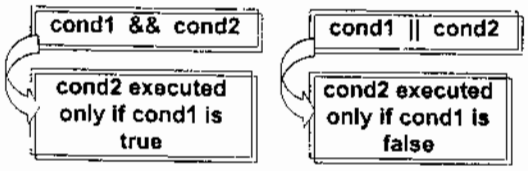
true || false

true

b = 1.

→ ++k never goes to work as in (++i && ++j) || ++k if ++i && ++j = true the or part doesn't need to get evaluated cos True || anything is true.

In General



Loop Control Instructions

- while
- for
- do-while

The *while* Loop

Init	Test	Incr
	Test	Incr
	Test	Incr
	Test	Incr
	..	..
	..	..

```
main( )
{
  int  m1, m2, m3, avg, i ;
  i = 1 ;
  while ( i <= 10 )
  {
      scanf ( "%d%d%d", &m1, &m2, &m3 ) ;
      avg = ( m1 +m2 + m3 ) / 3 ;
      printf ( "%d", avg ) ;
      i++ ;
  }
}
```

The *for* Loop

```
main( )
{
  int m1, m2, m3, avg, i ;
  for ( i = 1 ; i <= 10 ; i++)
  {
      scanf ( "%d%d%d", &m1, &m2, &m3 ) ;
      avg = ( m1 + m2 + m3 ) / 3 ;
      printf ( "%d", avg ) ;
  }
}
```

init	Test	Incr
	Test	Incr
	Test	Incr
	Test	incr
	..	..
	..	..

for (initialization ; condition ; increment/decrement)

Comparision - I

Infinite Loop

```
main( )
{
  int i = 1 ;
  while ( i <= 10 )
      statement1 ;
      statement2 ;
      statement3 ;
      i++ ;
}
```

Finite Loop

```
main( )
{
  int i ;
  for ( i = 1 ; i <= 10 ; i++ )
      statement1 ;
      statement2 ;
      statement3 ;
}
```

→ incrementation, declaration & condition all given in beginning.

no braces for while ∴ incrementation is later on & not in loop.

Comparison II

Infinite Loop

```
main( )
{
  int i = 1 ;
  while ( i <= 10 ) ;
  {
      statement1 ;
      statement2 ;
      statement3 ;
      i++ ;
  }
}
```

Finite Loop

```
main( )
{
  int i ;
  for ( i = 1 ; i <= 10 ; i++ ) ;
  {
      statement1 ;
      statement2 ;
      statement3 ;
  }
}
```

→ here ~~even if no~~ statement is executed 1 the loop ends due to limitation ~~on condition~~ & incrementation ~~all given~~ in ~~the syntax~~.

a semi colen after while ∴ i initially=1 1<=10 always ∴ always true & no execution of statement.

~~Here~~ no checking of condition the 2nd time & no incrementation

for (; ; i++);

out of loop

⋮ → normal execution

Print Nos. From 1 to 10

```
main( )
{
    int i ;
    for ( i = 1 ; i <= 10 ; i++ )
        printf ( "\n%d", i ) ;
}
```

Dropping Initialisation

```
main( )
{
    int i = 1 ;
    for ( ; i <= 10 ; i++ )
        printf ( "\n%d", i ) ;
}
```

by default i is used cos i<=10 & i++

Dropping Incrementation

```
main( )
{
    int i = 1 ;
    for ( ; i <= 10 ; )
    {
        printf ( "\n%d", i ) ;
        i++ ;
    }
}
```

by default ~~incrementation~~. no incrementation or decrementation

Dropping Condition

```
main( )
{
    int i = 1 ;
    for ( ; ; )
    {
        printf ( "%d", i ) ;
        i++ ;
        if ( i > 10 )
            break ;
    }
}
```

; ; Necessary

or

printf ("%d", i++) ;

combining

or

if (++i > 10)

point less
here condition default = 1 ie always true
infinite loop

break stops loop from becoming an infinite loop

Comparison III

Error

```
main( )
{
    int i = 1 ;
    while ( )
    {
        statement1 ;
        statement2 ;
        statement3 ;
    }
}
```

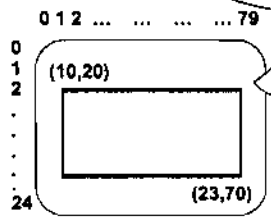
Infinite Loop

```
main( )
{
    int i = 1 ;
    for ( ; ; )
    {
        statement1 ;
        statement2 ;
        statement3 ;
    }
}
```

a condition is mandatory

Drawing Boxes

25 rows x 80 cols



(10, 20) ⟶ (row number, col number)

25 rows
80 columbs.

(row, col)

panning → horizontal movement
scrolling → vertical movement

To Begin With...

```
main( )
{
    clrscr( ) ;
    printf ( "%c", 218 ) ;
    ...
    ...
}
```

```
int ch = 65 ;
printf ( "%c", ch ) ;
printf ( "%c", 65 ) ;
```

shortcut.

218 → graphic character :- ┌

clrscr(); Positions the cursor at (0,0) default.

```
main( )                                  (10,20)
{
    int r, c ;
    clrscr( ) ;
    gotorc ( 10, 20 ) ; printf ( "%c", 218 ) ;          (23,70)
    gotorc ( 10, 70 ) ; printf ( "%c", 191 ) ;
    gotorc ( 23, 20 ) ; printf ( "%c", 192 ) ;
    gotorc ( 23, 70 ) ; printf ( "%c", 217 ) ;
    for ( r = 11 ; r < 23 ; r++ )
    {
        gotorc ( r, 20 ) ; printf ( "%c", 179 ) ;
        gotorc ( r, 70 ) ; printf ( "%c", 179 ) ;
    }
    for ( ... ... ... .. )
    {
    }
}
```

r = row
c = column

→ 11 → 22 23 not included
10 not included } → as ┌ 23 └

→ 20 → 70
(as no line b/w 20 & 70)

#include → pre processor director

```
#include  "goto.c"
main( )
{
    int  r, c ;
    clrscr( ) ;
    gotorc ( 10, 20 ) ; printf ( "%c", 218 ) ;
    .. ..
    .. ..
    for ( r = 11 ; r < 23 ; r++ )
    {
      .. ..
    }
    for ( ... ... ... )
    {
    }
}
```

Tip : include is not a keyword

Size of float variable = 4 bytes
Size of real const. = 8 bytes.

\n = new line
\t = tab to new line

When a value is supplied to scanf it goes to memory — keyboard buffer.
Scanf() then retrieves these values & stores it in the variable

fflush(stdin) — used before scanf to remove any residing enter (↵) in the keyboard buffer.

scanf("%d", i); ⟶ 35 ↵

keyboard buffer
35 ↵

scanf("%c", ch);

but if fflush(stdin) is used

35 → i
fflush(stdin) → ↵
ch ⟶ can now take any value.

@Rohay Lal VII th sheet.

3^3 combinations of 3 diff nos.

Combinations

1	1	1
1	1	2
1	1	3
1	2	1
1	2	2
1	2	3
1	3	1
1	3	2
1	3	3

2	1	1
2	1	2
2	1	3
2	2	1
2	2	2
2	2	3
2	3	1
2	3	2
2	3	3

3	1	1
3	1	2
3	1	3
3	2	1
3	2	2
3	2	3
3	3	1
3	3	2
3	3	3

Starting Off...

```
main( )
{
    int i, j, k ;
    i = 1 ;
    j = 1 ;
    for ( k = 1 ; k <= 3 ; k++ )
    {
        printf ( "\n%d%d%d", i, j, k ) ;
    }
}
```

i	j	k
1	1	1
1	1	2
1	1	3
1	2	1
1	2	2
1	2	3
1	3	1
1	3	2
1	3	3

output :- 111
112
113

Adding One More Loop

```
main( )
{
    int i, j, k ;
    i = 1 ;
    for ( j = 1 ; j <= 3 ; j++ )
    {
        for ( k = 1 ; k <= 3 ; k++ )
            printf ( "\n%d%d%d", i, j, k ) ;
    }
}
```

i	j	k
1	1	1
1	1	2
1	1	3
1	2	1
1	2	2
1	2	3
1	3	1
1	3	2
1	3	3

output :- 111
112
113
121
122
123
131
132
133

Finishing Off...

```
main( )
{
    int i, j, k ;
    for ( i = 1 ; i <= 3 ; i ++ )
    {
        for ( j = 1 ; j <= 3 ; j ++ )
        {
            for ( k = 1 ; k <= 3 ; k++ )
                printf ( "\n%d%d%d", i, j, k ) ;
        }
    }
}
```

1	1	1
..	..	..
..	..	..
1	3	3

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

3	1	1
..	..	..
..	..	..
3	3	3

full series
with common digits
eg. 111
122
133 etc.
(122
133)

Unique Combinations

```
main( )
{
    int i, j, k ;
    for ( i = 1 ; i <= 3 ; i++ )
    {
        for ( j = 1 ; j <= 3 ; j++ )
        {
            for ( k = 1 ; k <= 3 ; k++ )
            {
                if ( i != j && j != k && k != i )
                    printf ( "%d %d %d", i, j, k ) ;
            }
        }
    }
}
```

```
if ( i != j != k )
    printf ( "%d %d %d", i, j, k ) ;
```

wrong as 0 is being compared with k.

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

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

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

the only 6 unique solutions.

One More Way

```
main( )
{
    int i, j, k ;
1   for ( i = 1 ; i <= 3 ; i++ )
    {
2       for ( j = 1 ; j <= 3 ; j++ )
        {
            for ( k = 1 ; k <= 3 ; k++ )
3           {
                if ( i == j || j == k || k == i )
                    break ;
                else
                    printf ( "%d %d %d", i, j, k ) ;
            }
        }
    }
}
```

i	j	k
1	1	1
	2	1
	3	1

break sends to 2
as If isn't a loop its
a conditional statement.

111 i=j || not checked
as True || False = True.
anything

121 i=k || → not checked.
as True || False = True

continue = Keyword.

The Correct Way

```
main( )
{
    int i, j, k ;
    for ( i = 1 ; i <= 3 ; i++ )
    {
        for ( j = 1 ; j <= 3 ; j++ )
        {
            for ( k = 1 ; k <= 3 ; k++ )
            {
                if ( i == j || j == k || k == i )
                    continue ;
                else
                    printf ( "%d %d %d", i, j, k ) ;
            }
        }
    }
}
```

```
for ( .. ; .. ; k++ )
{
    ..
    if ( condition )
        continue ;
    else
        ...
    printf ( "Hi" ) ;
    ...
}
```

Continue : takes you to ③ the incrementation part of FOR (k++).

→ abandones any other statement present in FOR

here i value = 1
j value = 1

i = j || → not checked.

dry run →

i	j	k
1	1	1
		2
		3
		4
2		1
		2
		⋮

i	j	k
1	2	3
		4
3		1
		2
		3
		4

→ Bint ed

instead of continue a ; can be used as ; will (NULL statement) come out of if & increment K by k++.

Multiple Initializations

```
main( )
{
    int i, j, k ;
    i = 1 ; j = 3 ;
    for ( k = 1 ; k <= 5 ; k++ )
    {
        ...
        ...
        i += 2 ;
        j *= 4 ;
    }
}
```

Better Way

```
for ( i = 1, j = 3, k = 1 ; k <= 5 ; i += 2, j *= 4 )
{
    ...
    ...
}
```

Multiple Conditions?

Can I Drop k++

Connect using && and ||

and put k++ in the for loop.

multiple initializations & incrementati separated by (,) Comma.

multiple conditions separated by && ||.

Type of Loops

- while
- for
- do - while

Which is better

do = keyword.

The do-while Loop

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

Tip1: { } - Necessary
Tip2: ; after while - Necessary

in a do-while loop
Braces are a Must even if one statement is present.

At first it enters the loop then it checks the condition.
Printf executed 11 times. (1 by default 10 by loop).

Compare

do { printf ("Hi") ; } while (4 < 1) ;	for (; 4 < 1 ;) printf ("Hi") ;	while (4 < 1) printf ("Hi") ;
Hi	—	—

Imp

Effects of break & continue

```
while ( .... )
{
  if ( cond1 )
      break ;
  if ( cond2 )
      continue ;
}
```

Condition

```
for ( .. ; .. ; .. )
{
  if ( cond1 )
      break ;
  if ( cond2 )
      continue ;
}
```

first increment then Condition

```
do
{
  if ( cond1 )
      break ;
  if ( cond2 )
      continue ;
} while ( .... ) ;
```

Condition

Which To Use When

- for - Finite no. of times
- while - Unknown no. of times
- do - while - At least once then as many no. of times as possible!

Recommended, Not compulsory

Unknown No. of Times

Foolproof?

```
main( )
{
    char  ch = 'y' ;
    while (  ch == 'y' )
    {
        /* some logic */
        printf ( "Continue Y/N" ) ;
        scanf ( "%c", &ch ) ;
    }
}
```

```
while ( ch == 'y' || ch == 'Y' )
while ( toupper ( ch ) == 'Y' )
while ( tolower ( ch ) == 'y' )
```

toupper & to lower add 30 to the inputted digit

ie ASCII (A) $\xleftrightarrow{30}$ ASCII (a)

Control Instructions

- Sequence
- Decision
- Repetition
- Case

- while, for, do-while
  break, continue
- ++, --
- += -= *= /= %=

```
main( )
{
    int n ;
    printf ( "Enter no. between 1 and 3" ) ;
    scanf ( "%d", &n ) ;
    if ( n = = 1 )
        printf ( "You  entered 1" ) ;
    else
    {
        if ( n = = 2 )
            printf ( "You entered 2" ) ;
        else
        {
            if ( n = = 3 )
                printf ( "You entered 3" ) ;
            else
                printf ( "Wrong choice" ) ;
        }
    }
}
```

too many if
braces
indentation.

Alternatives

- Use Logical Operators [x]
- Use case Control Instruction [ ]

not Yes no problem
not checking range

switch } Keyword.
case

```
main()
{
    int n ;
    printf ( "Enter no. between 1 and 3" ) ;
    scanf ( "%d", &n ) ;
    switch ( n )
    {
        case 1 :
            printf ( "You Entered 1" ) ;
        case 2 :
            printf ( "You Entered 2" ) ;
        case 3 :
            printf ( "You Entered 3" ) ;
        else
            printf ( "Wrong Choice" ) ;
    }
}
```

→ hanging else

imp very imp.

```
main()
{
    int n ;
    printf ( "Enter no. between 1 and 3" ) ;
    scanf ( "%d", &n ) ;
    switch ( n )
    {
        case 1 :
            printf ( "You entered 1" ) ;
        case 2 :
            printf ( "You entered 2" ) ;
        case 3 :
            printf ( "You entered 3" ) ;
        default :
            printf ( "Wrong Choice" ) ;
    }
}
```

Tip: If a case fails control jumps to the next case.

Output:
You entered 2
You entered 3
wrong Choice

default → Keyword.

→ as every next case also executed

→ when every other case other than 1 2 3.

In switch even if only one statement is given braces are required.

```
main ( )
{
    int n ;
    scanf ( "%d", &n ) ;
    switch ( n )
    {
        case 1 :
            printf ( "You entered 1" ) ;
        case 2 :
            printf ( "You entered 2" ) ;
        case 3 :
            printf ( "You entered 3" ) ;
        default :
            printf ( "Wrong choice" ) ;
    }
}
```

Tip: If a case is satisfied all statements below it upto } of switch are executed

Output:
You entered 2
3
wrong Cho

reason for this is
Once one case is given true all following cases get executed as then "case" keyword is ignored. & inside statements executed.

Switch Case works step by step ie Case by Case

The Solution

```
main ()
{
   int n ;
   scanf ( "%d", &n ) ;
   switch ( n )
   {
      case 1 :
            printf ( "You Entered 1" ) ;  break ;
      case 2 :
            printf ( "You Entered 2" ) ;  break ;
      case 3 :
            printf ( "You Entered 3" ) ;  break ;
      default :
            printf ( "Wrong choice" ) ;
   }
}
```

Tip: { } are optional even if there are multiple statements in a case

in Case no need of braces even for mutiple statements

Its scope extends till the next case or (break).

What If *continue*

```
main ()
{
   int n ;
   scanf ( "%d", &n ) ;
   switch ( n )
   {
      case 1 :
            printf ( "You Entered 1" ) ;  break ;
      case 2 :
            printf ( "You Entered 2" ) ;  continue ;
      case 3 :
            printf ( "You Entered 3" ) ;  break ;
      default :
            printf ( "Wrong choice" ) ;
   }
}
```

infinite loop as continue ~~tak~~ will give an Error

⇒ Illegal useage of Continue.

(Continue will take back to switch. Compiler dont allow this as Switch isn't a loop its only a verifier).

imp

```
main ()
{
   int n ;
   scanf ( "%d", &n ) ;
   switch ( n )
   {
      case 2 :
            printf ( "You Entered 2" ) ; break ;
      case 1 :
            printf ( "You Entered 1" ) ; break ;
      case 3 :
            printf ( "You Entered 3" ) ; break ;
      default :
            printf ( "Wrong choice" ) ;
   }
}
```

Tip: Order of cases is unimportant

Tip: { } are optional even if there are multiple statements in a case

→ cos switch acts step by step.

can be written anywhere not reccessarily last.

```
main ( )
{
    int n ;
    scanf ( "%d", &n ) ;
    switch ( n )
    {
        default :
          printf ( "Wrong choice" ) break ;
        case 2 :
          printf ( "You Entered 2" ) ; break ;
        case 1 :
          printf ( "You Entered 1" ) ; break ;
        case 3 :
          printf ( "You Entered 3" ) ; break ;
    }
}
```

Tip: Even default can be the very first case

is also required.

not required

if n = 5

case 2
1
3

then default → always executed last.

```
main ( )
{
    int n ;
    scanf ( "%d", &n ) ;
    switch ( n )
    {
        case 2 :
          printf ( "You Entered 2" ) ;  break ;
        case 1 :
          printf ( "You Entered 1" ) ;  break ;
        case 3 :
          printf ( "You Entered 1" ) ;
    }
}
```

Tip: default case is optional

default is optinal:

```
main( )
{
    char  ch ;
    printf ( "Enter alphabet between A and C" ) ;
    scanf ( "%c", &ch ) ;
    switch ( ch )
    {
        case 'A' :
              printf ( "You entered A" ) ;
              break ;
        case 'B' :
              printf ( "You entered B" ) ;
              break ;
        case 'C' :
              printf ( "You entered C" ) ;
              break ;
    }
}
```

What if 'a', 'b', 'c'

without 'A' it turns out to be a variable not a char.

for small case. (one way)
— switch (touppes (ch)).

can be dropped.

```
main( )
{
    char ch ;
    printf ( "Enter alphabet between A and C" ) ;
    scanf ( "%c", &ch ) ;
    switch ( ch )
    {
        case 'A' || 'a' :
            printf ( "You entered A" ) ;
            break ;
        case 'B' || 'b' :
            printf ( "You entered B" ) ;
            break ;
        case 'C' || 'c' :
            printf ( "You entered C" ) ;
            break ;
    }
}
```

Become case 1 :

drop.

conditional operators are also not allowed.
conditional will get executed first
'A' || 'a' ⇒ true = 1
∴ all ⇒ Case 1
Case 1

```
main( )
{
    char ch ;
    printf ( "Enter alphabet between A and C" ) ;
    scanf ( "%c", &ch ) ;
    switch ( ch )
    {
        case 'a' :
        case 'A':
            printf ( "You entered A" ) ;
            break ;
        case 'b' :
        case 'B':
            printf ( "You entered B" ) ;
            break ;
        case 'c' :
        case 'C':
            printf ( "You entered C" ) ;
            break ;
    }
}
```

a A

Case 'a' will print all values of ~~Pat~~ Case 'A' as no break after Case 'a'.

Would This Work

```
main( )
{
    int n = 2 ;
    int a = 1, b = 2 ;
    switch ( n )
    {
        case a :
            ...
        case b :
            ...
        case 3 :
            ...
    }
}
```

Error.

wrong as a & b will act as a variable & a variable's value goes on changing.
if a = 3.
the switch gets confused b/w case a & case 3.

General Form

```
main( )
{
    switch ( expression )
    {
        case constant expression :
            ...
        case constant expression :
            ...
        case constant expression :
            ...
        default:
            ...
    }
}
```

n + 3 / a + 2

4 + 3 / 5 + 2

3 + 2 % 5

Checking switch

- int ✓
- char ✓
- long int ✓
- float ✗
- * Double ✗

Tip: float cannot be checked using switch

no float or double.
as : 00 → till what limit.
1.3 ≠ 1.3000 in switch
thus won't work.

Is switch a replacement for if ?

Ctrl + break → immediate break in execution.

to remove Blue bar → Ctrl + F_2.

for step-by-step execution press F_7.

to inspect a value during step-by-step execution Ctrl + F_7.

Imp Sheet

@Kshay Lal
Sheet - VIII.

Menu Management

20

10

1. Odd / Even
2. Leap Year
3. Prime No.
0. Exit

Your Choice?

10r 20c

```
# include "goto.c"
main( )
{
    int choice ;
    clrscr( ) ;
    gotorc ( 10, 20 ) ;   printf ( "1. Odd / Even" ) ;
    gotorc ( 11, 20 ) ;   printf ( "2. Leap year" ) ;
    gotorc ( 12, 20 ) ;   printf ( "3. Prime number" ) ;
    gotorc ( 13, 20 ) ;   printf ( "0. Exit" ) ;
    gotorc ( 15, 20 ) ;   printf ( "Your choice?" ) ;
    scanf ( "%d", &choice ) ;
    ..
    ..
}
```

```
main( )
{
   int choice ;
   /* display menu */
   scanf ( "%d", &choice ) ;
   switch ( choice )
   {
      case 1 :
         /* odd / even logic */
         break ;
      case 2 :
         /* leap year logic*/
         break ;
      case 3 :
         /*prime number logic*/
         break ;
      case 0 :
         break ;
      default :
         printf ( "\a" ) ;
   }
}
```

\n \t \a

Escape Sequences

printf ("\a\a\a") ;

Long Beep

a = alert.

ideally it should give out 3 beeps but we get 3 beeps as no time lag btw the beeps.

Block Commands

```
switch ( choice )
{
   case 1 :
      break ;
   case 2 :
      break ;
   case 3 :
      break ;
   case 0 :
      break ;
   default :
      printf ( "\a" ) ;
}
```

Ctrl KR — CH2PR3.C — Odd / Even logic

Ctrl KR — CH2PR4.C — Leap year logic

Ctrl KR — CH2PR5.C — Prime no. logic

block

Ctrl + KR — Read

will ask for file name
it will be read & pasted at the position of the cursor.

Other Block Commands

beginning

Imp ****.

Ctrl KB
Ctrl KK — Mark block → set of statements

end.

Ctrl KV	Move block
Ctrl KC	Copy block
Ctrl KW	Write block to file
Ctrl KY	Delete block
Ctrl KH	Hide or unmark block

→ asks for new file name.

Readjustments Necessary

- case 1 :
  main( )
  ...
  ...
  break ;
- Shift declarations

Delete

Delete

Only Shifting Won't Do → modification is required.

menu will get execute as soon as prog is executed.

for re-executing Menu !!

No while, No Menu

```
#include "goto.c"
main( )
{
    int choice ;
    while ( 1 )
    {
        /*display menu*/
        scanf ( "%d", &choice ) ;
        switch ( choice )
        {
            case 1 :
                ...
                break ;
            case 0 :
                break ;
        }
    }
}
```

- Infinite loop
- switch

out of switch-case:1

exit( ) ; → will terminate execution of program

Break From The Outermost Loop

```
main( )
{
    for ( i = 1 ; i <= 22 ; i+= 2 )
    {
        for ( j = 5 ; j <= 50 ; j++ )
        {
            for ( k = 1 ; k <= 9 ; k += 3 )
            {
                ... ...
                if ( i + j % k >= 2 )
                    break ;
            }
            break ;
        }
        break ;
    }
}
```

Go to work only once

if (i + j % k >= 2) break ;

out of for (k).

out of for (j)

out of for (i).

imp

Better Way...

```
main( )
{
    for ( i = 1 ; i <= 22 ; i+= 2 )
    {
        for ( j = 5 ; j <= 50 ; j++ )
        {
            for ( k = 1 ; k <= 9 ; k += 3 )
            {
                ... ...
                if ( i + j % k >= 2 )
                    goto out ;
            }
        }
    }
    out :
        ;
}
```

goto → keyword (always followed by label).

out → label.

goto 1.

will take control to 1.

Suggestion:— Never Use A Goto

goto is apparently too powerful.

Where Am I...

```
main()
{
    in :
        goto there ;
        for ( i = 1 ; i <= 22 ; i+= 2 )
        {
            for ( j = 5 ; j <= 50 ; j++ )
            {
                there :
                for ( k = 1 ; k <= 9 ; k += 3 )
                {
                    .....
                    if ( i + j % k >= 2 )
                        goto out ;
                }
            }
        }
        out :
            goto in ;
}
```

excess goto will confuse users in execution.

Control Instructions

(ALL of them)

- Sequence
- Decision
- Repetition
- Case
- goto ← never use!!

BASICS

FUND Shit

Functions

```
main()
{
    printf ( "\n I am in main" ) ;
}
bombay()
{
    printf ( "\n I am in Bombay" ) ;
}
kanpur()
{
    printf ( "\n I am in Kanpur" ) ;
}
hell ()
{ Printf("\n I am in Hell");
}
```

Output:

```
Functions
main()
printf()
scanf()
getch()
exit()
gotorc()
clrscr()
for()
while()
if()
switch()
```

(imp to know)

clrscr() → ntn in parenthesis

while() → somethin in parenthesis.

① they are loops / conditional statements

② they are keywords.

Output: I am in main.

**** as execution ends at closing braces of main.

Functions followed by () & name is not a keyword.

function defination.

Output: I am in Main.
I am in Bombay
I am in Kanpur.

Calling Functions

```
main( )
{
    printf ( "\n I am in main" ) ;
    bombay( ) ; kanpur( ) ;
}
bombay( )
{
    printf ( "\n I am in Bombay" ) ;
}
kanpur( )
{
    printf ( "\n I am in Kanpur" ) ;
}
```

Output :

Function Call

Function Def.

calling of function (function call).

function declaration.

after execution of function cursor/ execution returns to pt. after call of function.

Functions

standard library func

readily made (predefined)

Std. Library	User-Defined
printf( )	kanpur( )
scanf( )	bombay( )
exit( )	gotorc( )
clrscr( )	main( )

Read (imp)

Tips

- A C program is nothing but a collection of 1 or more functions
- If C program contains 1 function its name must be main( )
- If C program contains more than 1 function then one of them has to be main( )
- Execution of any C program always begins with main( )
- Function names in a program must be unique

Execution Starts from { Main
Execution Ends at } of Main

in Compiler.
{ main();
} → after call of func main shutdown the dam prog.

reason why we write Main() for execution as when Execution compiler calls the function main(); that is why.
now at closing brace of main execution cursor returns to function call. that is in the language/compiler.

Order, Order !

```
bombay( )
{
    printf ( "\n I am in Bombay" ) ;
}
main( )
{
    printf ( "I am in main" ) ;
    bombay( ) ;
}
```

Tip: Functions can be defined in any order

More Calls, More Bills

```
main( )
{
    printf ( "\n I am in main" ) ;
    bombay( ) ;
    bombay( ) ;
}
bombay( )
{
    printf ( "\n I am in Bombay" ) ;
}
```

Tip: More the calls, slower the execution

→ For multiple calls put a loop.

Nobody is Nobody's Boss

```
main( )
{
    printf ( "\n I am in main" ) ;
    bombay( ) ; kanpur( ) ;
}
bombay( )
{
    printf ( "\n I am in Bombay" ) ;
    kanpur( ) ;
}
kanpur( )
{
    printf ( "\n I am in Kanpur" ) ;
    bombay( ) ;
}
```

Tip: Any function can call any other function

Another way of achieving an infinite loop.
One function calling another function which itself is calling the former.

Local v/s STD v/s ISD Calls

```
main( )
{
    printf ( "\n I am in main" ) ;
    main( ) ;
}
```

Local Call - Recursive Function
Process - Recursion

→ from within a func. calling itself → local call.

Another infinite loop.
One function calling itself.

Process of calling itself is called Recursion.

Main() → Kanpur() (STD call)
Main() → to another program (ISD call).

Communication

```
main( )
{
    int a = 10, b = 20, c = 30 ; int s ;
    calsum ( ) ;
    printf ( "%d", s ) ;
}
calsum( )
{
    int a, b, c, s ;
    s = a + b + c ;
    printf ( "%d", s ) ;
}
```

→ local variables.
redeclaration in diff function.
→ local variables.

Output :- garbage, garbage.
(calsum) (main)

Useage of functions

Passing Values

```
main( )
{
    int a = 10, b = 20, c = 30 ; int s ;
    calsum ( a, b, c ) ;
    printf ( "%d", s ) ;
}
calsum ( int x, int y, int z )
{
    int s ;
    s = x + y + z ;
    printf ( "%d", s ) ;
}
```

Actual Arguments
Formal Arguments

→ (passing of values)
→ x y z abc or anything need not be the same as above
Preferably diff names.

Output 60 ~~60~~ garbage :
calsun main.

a	b	c	x	y	z
10	20	30	10	20	30.

As in gotorc(10,20);

gotorc(int x, int y)
{ ==
}

Rule — Actual arguments should always match Formal arguments in Number, order & type.

ie sending 3 (int int float) accepting 3 (int int float)

always a 1 → 1 relation.

Returning Values

```
main( )
{
    int a = 10, b = 20, c = 30, s ;
    s = calsum ( a, b, c ) ;
    printf ( "%d", s ) ;
}
calsum ( int x, int y, int z )
{
    int ss ;
    ss = x + y + z ;
    return ( ss ) ;
}
```

return ; Returns only control

Return control and value

return (ss) ;
return (60) ;
return (x + y + z) ;

return → Keyword.

not a function call.

for returning only control return;

we can never return only a value Only using return;.

///ly

```
a = pow (2, 5);
pow (int x, int y)
{
    return (value);
}
```

Are These Calls OK?

```
calsum ( a, 25, d ) ;
calsum ( 10 + 2, 25 % 3, d ) ;
calsum ( a, calsum ( 25, 10, 4 ), d ) ;
d = calsum ( a, 25, d ) * calsum ( a, 25, d ) + 23 ;
```

```
calsum ( int x, int y, int z )
{
    int ss ;
    ss = x + y + z ;
    return ( ss ) ;
}
```

Returned value can be ignored

Nested calls are legal.
Call within an expression is legal.

x = a y = 25 z = d.
x = 10+2 y = 25%3 z = d.
x = a y = value of Calsum(25,10,4) z = d
(nested).
(x = a y = 25 z = d) then (x = a, y = 25, z = d) then + 23.

sin (cos (0.5). => sin of value of cos(0.5)

[Whenever there is a nested call we usually don't use a semi colon]

Returning More Than 1 Value

```
main( )
{
    int a = 10, b = 20, c = 30 ;
    int s, p ;
    s, p = sumprod ( a, b, c ) ;
    printf ( "%d%d", s, p ) ;
}
sumprod ( int x, int y, int z )
{
    int ss, pp ;
    ss = x + y + z ;
    pp = x * y * z ;
    return ( ss, pp ) ;
}
```

A function can return only 1 value at a time

separate declaration for each variable.

**** while passing value, sending is important & so is collecting while returning value, returning or sending a value is arbitrary.

A func can return only a single value.

One More Try

```
main( )
{
    int a = 10, b = 20, c = 30 ;
    int s, p ;
    s = sumprod ( a, b, c ) ;
    p = sumprod ( a, b, c ) ;
    printf ( "%d%d", s, p ) ;
}
sumprod ( int x, int y, int z )
{
    int ss, pp ;
    ss = x + y + z ;
    pp = x * y * z ;
    return ( ss ) ;
    return ( pp ) ;
}
```

→ only return(ss); is getting activated as after return function stops. As control is gone back to Main

→ redundant.

The Only Way Out

```
main( )
{
    int a = 10, b = 20, c = 30 ; int s, p ;
    s = sumprod ( a, b, c, 1 ) ;
    p = sumprod ( a, b, c, 2 ) ;
    printf ( "%d%d", s, p ) ;
}
sumprod ( int x, int y, int z, int code )
{
    int ss, pp ;
    ss = x + y + z ; pp = x * y * z ;
    if ( code == 1 )
        return ( ss ) ;
    else
        return ( pp ) ;
}
```

Sum, Product, Average, Variance Standard Deviation

→ here 1 & 2 are the flags.

⇒ At a time a function can return ONLY one value at a time.

⇒ by increasing the no. of flags diff values can be returned.

A Better Way

```
sumprod ( int x, int y, int z, int code )
{
    int ss, pp ;
    ss = x + y + z ;
    pp = x * y * z ;
    if ( code == 1 )
        return ( ss ) ;
    else
        return ( pp ) ;
}
```

✗ `code == 1 ? return ( x + y + z ) : return ( x * y * z ) ;` → won't work.

✓ `return ( code == 1 ? x + y + z : x * y * z ) ;` → will work.

→ return statement can't be used with a ? : statement. This is a rule.

Floyd's triangle

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

```
int i, j, flag, first_no=0;
clrscr();
for (i=0 ; i< 5 ; i++)
{
        First_no = !first_no ;
        flag = first_no ;
        for (j=0 ; j <= i ; j++)
        {
            printf ("%d", flag);
            flag = !flag;
        }
        printf ("\n")
}
```

[Pointer Basics] @Pohay Lal

sheet – IX

ANSI V/s K & R

```
calsum ( int x, int y, int z )
{
    ..
}
```

ANSI

```
calsum ( x, y, z )
int x, y, z ;
{
    ..
}
```

Kernighan & Ritchie

American National Standard Institute → method of defining variables of functions.

→ method of defining variables of a function.

Roman Equivalent

```
main( )
{
    int y ;
    printf ( "Enter year" ) ;
    scanf ( "%d", &y ) ;
    romanize ( y ) ;
}
romanize ( int yy )
{
    int n, i ;
    n = yy / 1000 ;
    for ( i = 1 ; i <= n ; i ++ )
        printf ( "m" ) ;
}
```

1998

1998 ⟹ mdcccclxxxxviii

Decimal	Roman
1000	m
500	d
100	c
50	l
10	x
5	v
1	i

A More General Call

```
main( )
{
    int y ;
    printf ( "Enter year" ) ;
    scanf ( "%d", &y ) ;
    y = romanise ( y, 1000, 'm' ) ;
}
romanize ( int yy, int j , char ch)
{
    int n, i ;
    n = yy / j ;
    for ( i = 1 ; i <= n ; i ++ )
        printf ( "%c", ch ) ;
    return ( yy % j ) ;
}
```

a general call for printing no. of 1000 in a given Year representing 1000 → m.

—: more logical way

```
main()
{
    ...
    y = romanise ( y, 1000, 'm' ) ;
    y = romanise ( y, 500, 'd' ) ;
    y = romanise ( y, 100, 'c' ) ;
    y = romanise ( y, 50, 'l' ) ;
    y = romanise ( y, 10, 'x' ) ;
    y = romanise ( y, 5, 'v' ) ;
    romanise ( y, 1, 'i' ) ;
}
romanize ( int yy, int j, char ch )
{
    int n, i ;
    n = yy / j ;
    for ( i = 1 ; i <= n ; i ++ )
      printf ( "%c", ch ) ;
    return ( yy % j ) ;
}
```

1998 ⟹ mdcccclxxxxviii

→ no. of 1000
→ no. of 500
→ 100's
→ 50's
→ 10's
→ 5's
→ 1's

→ works eventhough collection of value isn't done.

Advanced Features of Functions

- Returning a non-integer value
- Call by value / Call by reference
- Recursion

Pow(double, double)
it also returns a double.
∴ its prototype declaration is previously done in MATH.H.
Thus we have to include MATH.H so that its previously created prototype declaration may be made available.

Returning a Non-Int Value

```
main()
{
    float a, b, c ; float square ( float ) ;
    a = square ( 2.0 ) ;
    b = square ( 2.5 ) ;
    c = square ( 1.5 ) ;
    printf ( " %f %f %f ", a, b, c, ) ;
}
float square ( float x )
{
    float y ;
    y = x * x ;
    printf ( " %f ", y ) ;
    return ( y ) ;
}
```

Function Prototype

without prototype (4.0, 6.0, 2.0)

output :- 6.25, 2.25

[2.0 2.5 1.5 → are Doubles not floats by default]
(to convert to float 2.0f 1.5f 2.5f)

*** ⇒ by default a func returns an integer.

→ Square is a func which accepts a float & returns a float.

*** Prototype ensures the compiler that further down the line there is a func. which recieves a float and returns a float.

What's Wrong?

```
main( )
{
    float square ( float ) ;
    float a = 0.5 ;
    f ( a ) ;
}
f ( float x )
{
    float y ;
    y = square ( x ) ;
    printf ( "%f", y ) ;
}
float square ( float x )
{
    return ( x * x ) ;
}
```

from where the func. is being called from there the declaration of prototype must be present. [local].

⇒ above main if we declare prototype it becomes available to all other func. [global].

∴ Prototypes to be declared as Global ALWAYS.

What Would Be The Output

```
main( )
{
    int a = 10 ;
    printf ( "\n a = %d" , a ) ;   — 10
    a = f( ) ;
    printf ( "\n a = %d" , a ) ;   — garbage
}
f( )
{
    printf ( "Hello !" ) ;
}
```

the func is returning a ~~gab~~ garbage integer to 'a'.

Solution 1

```
main( )
{
    int a = 10 ;
    printf ( "\n a = %d" , a ) ;   — 10
    f( ) ;
    printf ( "\n a = %d" , a ) ;   — 10
}
f( )
{
    printf ( "Hello !" ) ;
}
```

Returned value is ignored.

→ not collecting the return value of function.

to make global declare outside main.

Solution 2

```
main()
{
    int a = 10 ; void f() ;
    printf ( "\n a = %d" , a ) ;
    f() ;
    printf ( "\n a = %d" , a ) ;
}
void f()
{
    printf ( "Hello !" ) ;
}
```

prototype as func not default (int).

void prevents returning of value.

void → means emptyness nothingness

thus void func name wont return any value.

void → keyword

Advanced Features of functions

- Returning a non-integer value
- Call by Value / Call by Reference
- Recursion

POINTERS

fundu shit

↓

will give any asbitary value depending on fancy of compiler.

imp

Things Are Simple

```
main()
{
    int i = 10 ;
    printf ( "\n value of i = %d" , i ) ;
    printf ( " \n address of i = %d" , &i ) ;
    printf ( " \n value of i = %d" , *( &i ) ) ;
}
```

10 4080 10

Memory

			10
			4080
0	1	2	3

location value.
location name
location no → address of i
memory location of i
reference of i.

Pointer Operators

& - 'Address Of' Operator
* - 'Value at Address' Operator
- 'Indirection' Operator

Address
Reference
Memory location
Cell number

(value present at memory location 4080)

&i = add. of ~~location~~ variable i

*(&i) = value at add of variable i.

*i = value of variable i.

*** Scanf ("%d", &k) sending value at add of value k
(add of variable)

Would This Work...

```
int i = 10 ;
j = &i ;
j = &23 ;
j = & ( i + 34 ) ;

printf ( "%d", *j ) ;
printf ( "%d", *4568 ) ;
printf ( "%d", *( 4568 + 1 ) ) ;
```

& can be used only with a variable

* can be used with variable, constant or expression

Some Change Necessary

→ add of variable i

→ wont work

∴ (And) & work ONLY with variables.

→ if 4568 *4568 should give value at add no. 4568.

→ value at mem location j.

Very imp

The Next Step

```
main( )
{
int i = 10 ; int *j ; int **k ;
printf ( " \n value of i = %d " , i ) ; value of i.
printf ( " \n address of i = %d " , &i ) ; add of i.
printf ( " \n value of i = %d " , *( &i ) ) ; value of i.
j = &i ; → add of i stored in j.
printf ( " \n address of j = %d " , &j ) ; → add of j
printf ( " \n value of j = %d " , j ) ; → value of j
printf ( " \n value of j = %d " , *&j ) ;
k = &j ; → add of j store in k. → value of j.
printf ( " \n %d %d %d %d " , k , &k , *k , *&k ) ;
6010, 5112, 4080, 6010.
printf ( " \n %d %d %d %d %d %d " , i, *&i, *j, **&j,
*k, **k, ***&k ) ;
10, 10, 10, 10, 10, 10.
```

i	j	k
10	4080	6010
4088	6010	5112

& k

* k. *(&i) = *&i parenthesis isn't required ① add of i obtained ② value at add obtained

→ *j isn't required printf j is the easiest & simplest way.

*j means value at address j

j → 4080 * = value at add 4080 = 10

** k

k → 6010 * 6010 = 4080
* 4080 = 10

**&j &j = 6010.
* 6010 = 4080
* 4080 = 10

In Essence ...

- & - Address of operator
- * - value of address operator

} Pointer Operators

- variable ⟷ *&variable

int &k ;
k = && j ;

- Pointers are variables which hold addresses of other variables

Integer Pointer → holding add of integer

→ not integer pointer as pointing to add.

- int i = 10 ; int *j, **k, ***l , ****m ; j = &i ; k = &j ; l = &k ; m = &l ;

*j = pointer to i
** k = pointer to pointer j which is pointer to i

the printing of int and add is the same. But they are completely diff. things.

int k & int j will be wrong. as j & k are storing add. As Addresese ≠ Integers

Proof:- we screw around with integer value. on an address it doesnt make sence to + − * / etc.

∴ declaration for j ⇒ int *j ; (*j = value at add 4080)
int *k ; (*k = value at add 6010 is an int which is wrong.
int **k ; ***k = value at 4080 is an int. Yes.

Doll Dancing ~~Fan~~ Virus

VIRUS!!

→ character pointer as holding address of 1000 which is a char.

At a time on the screen there can be 2000 char (25×80=2000).

Accessing Screen

```
main()
{
  int i; char *s;
  while ( 1 )
  {
    s = 1000;
    for ( i = 0 ; i <= 1999 ; i++ )
    {
      if ( *(s+i) >= 'A' && *(s+i) <= 'Z' )
            *(s+i) = *(s+i) + 32 ;
      else
      {
        if ( *(s+i) >= 'a' && *(s+i) <= 'z' )
              *(s+i) = *(s+i) - 32 ;
      }
    }
  }
}
```

char far *s ;

0xB8000000

Screen: M | 1000 | 1000 +1 | 1000 +2 | 1000 +3 | 1000 + 1999

```
char ch ;
ch = 'A' ;
ch = ch + 32 ;
```

output:-

Changes every char from upper case to lower very fast

*(s+i) = value at
1000 + 0 = 1000
1000 + 1 = 1001
1000 + 2 = 1002
1000 + 1999 = 1999

{Changing position of cursor on screen}

0xB8000000 → first ~~char~~ address of every screen

↑ hexadecimal can be in any system internally everything in binary

Accessing Screen

```
main()
{
  int i ; char far *s ;
  while ( 1 )
  {
    s = 0xB8000000 ;
    for ( i = 0 ; i <= 1999 ; i++ )
    {
      if ( *(s+i) >= 'A' && *(s+i) <= 'Z' )
            *(s+i) = *(s+i) + 32 ;
      else
      {
        if ( *(s+i) >= 'a' && *(s+i) <= 'z' )
              *(s+i) = *(s+i) - 32 ;
      }
    }
  }
}
```

3998

i += 2

Screen: M | + 2 | + 4 | + 6 | 0xB8000000 | + 3998

Keyword

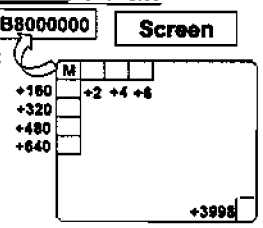
- far = far char pointer
- all char on screen are 2 bytes apart.

ALL VIRUSES

@Rohan Lal
Sheet - X

To Make Characters Fall

```
main()
{
    char far *s ; int r ; char ch ;
    char far * v ;
    s = 0xB8000000 ;
    ch = * s ;
    for ( r = 1 ; r <= 24 ; r ++ )
    {
        v = s + r * 160 ;
        * v = ch ;
    }
}
```

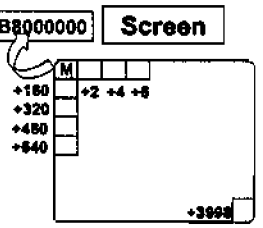


Every col +2
Every row +160.

xB8 ... +160
+320

Make All Characters Fall

```
main()
{
    char far *s ; int r ;
    char far *v ; char ch ;
    s = 0xB8000000 ;
    int c ;
    for ( c = 0 ; c <= 79 ; c++ )
    {
        ch = *( s + c * 2 ) ;
        for ( r = 1 ; r <= 24 ; r++ )
        {
            v = s + r * 160 + c * 2 ;
            * v = ch ;
        }
    }
}
```

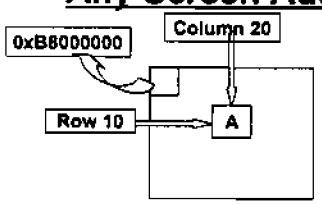


c → col
r → row

without these - pt. only the char in first col will keep falling.

moves cursor each step to the right thus letting each char from each pt. on the screen fall down.

Any Screen Address




```
v = 0xB8000000 + 10 * 160 + 20 * 2 ;
*v = 'A' ;
```

for 10 row 20 col.
let value at v position = A.

In general,

```
v =0xB8000000 + r * 160 + c * 2 ;
*v = ch ;
```

r = row
c = col.

fastest way of writing anything on the screen

gotoxy is very slow compared ∴ Printf is very slow & useless.

The above method is the fastest & from now on the only way to print things on the screen.

RAINDROP VIRUS.

Pseudorandom no. = false random no (always same random no)

Bells And Whistles

```
main ()
{
   char far *s ; char ch ;
   char far *v ; int r, c ;
   s = 0xB8000000 ;
   for ( c = 0 ; c <= 79 ; c++ )
   {
      ch = * ( s + c * 2 ) ;
      for ( r = 1 ; r <= 24 ; r++ )
      {
         v = s + r * 160 + c * 2 ;
         *v = ch ;
         delay ( 60 ) ;
         * ( v - 160 ) = ' ' ;
         sound ( 350 ) ; delay ( 100 ) ;
         nosound ( ) ;
      }
   }
}
```

0xB8000000 | Screen

M | +2 +4 +6 | +160 | +320 | +480 | +640 | +3998

randomize() (stdlib.h)

r = random (24) ; c = random (79) ;

erases previous character

delay() sound() nosound() } – library functions → (stdio.h)

as 25 rows 80 col

interesting sound.

shuts up the sound.

erasing command.

delays the falling of the fall (millisec) → used cos the prog runs at a superdooos speed. So we delay to see it fall.

Are They Same

```
char far *s ;
char far *v ;

char far *s, *v ;        char far *s, far *v ;

char far *s ;
s = 0xB8000000 ;   <=>   char far *s = 0xB8000000 ;
```

far only works for variable just preceeding it

different

here 0xB is assigned to far *s.

here 0xB.. is assigned only to s

to do something on the screen FAR has to be used. (any i/o device)

Every pointer by default is a NEAR pointer. *j, *k,

Why Two Bytes Apart

```
main()
{
   char far *s ; int i ;
   int color = 0 ;
   s = 0xB8000000 ;
   while ( 1 )
   {
      for ( i = 1 ; i <= 3999 ; i += 2 )
         * ( s + i ) = color ;
      color++ ;
      if ( color > 255 )
         color = 0 ;
   }
}
```

for I/O device

Screen: M | +2 +4 +6 | 0xB8000000

VDU Memory: ASCII | Color

Color Byte: —8 Bits—

Min value 00000000 | Max value 11111111

Range 0 to 255

every byte has one of these thingys every alt byte is either the ASCII value (0x38) or Colour.

now 1 byte = 8 bits

∴ range of colour = 0 → 255

256 colours are possible. Cos range (0–255)

use a delay to view colour screen

screwing with colour.

220 library functions

library function (variable/command) Ctrl+F1 ← cursor

context sensitive help.

alternate btw 2 states ON OFF

Is The Caps Lock On

```
main( )
{
    char far *kb ;
    kb = 0x417 ;
    *kb = 64 ;
}
```

Toggle Keys: Caps lock, Num lock, Scroll lock, Insert

01000000

7 6 5 4 3 2 1 0

0x417

Caps Lock
1 - On
0 - Off

How would you put off the caps lock?

*kb = 0
all 0's will be sent.

64 binary

we can't use binary direct as C dont get binary. If binary used it will be Octal.

→ toggle key byte.

if 1 present Caps
0 present no caps.

If touching this byte all values must be entered.

Auto Reboot

Don't Do Delete

```
main( )
{
    char far *kb ;
    kb = 0x417 ;
    printf ( "Pl. press delete key" ) ;
    *kb = 12 ;
    getch( ) ;
}
```

00001100

7 6 5 4 3 2 1 0

Alt Ctrl

Input Fun.
scanf( )
getch( )

getch → we don't follow it by an Return key.

Ctrl & Alt aren't toggle keys as they don't remain on/off when released.
They are shift keys.

Will restart the machine as Ctrl & Alt are already present
Del ⇒ Ctrl + Alt + Del

VIRUS

Happy Birthday Joshi

7 6 5 4 3 2 1 0

0 0

Our Program ⇒ Del → Ctrl + Alt + Del

Happy Birthday Joshi ⇒ Ctrl + Alt + Del → Del

→ first virus to Enter India.

→ reducing Del to Ctrl + Alt + Del

→ when Ctrl + Alt is pressed the 1 1 is converted to 0 0 before the delete key gets activated.

add of a float diff from add of int?

should a value be previously printed before (a+b)...

how is printing done through ptr.

Ram not Rom:

What would be the output

```
main( )
{
    int i = 10 ; float a = 3.14 ; char ch = 'z' ;
    int *j, **k ; float *b, **c ; char *dh, **eh ;
    j = &i ; b = &a ; dh = &ch ;
    k = &j ; c = &b ; eh = &dh ;
    printf ( " %d %d %d" , j , b , dh ) ;
    1000 2000 3000.
    printf ( " %d %d %d" , *k , *c , *eh ) ;
    1000 2000 3000.
    printf ( " %d %d %d" , sizeof ( i ), sizeof ( a ), sizeof ( ch ) ) ;
    2 4 1
```

i	a	ch
10	3.14	z
1000	2000	3000
j	**b**	**dh**
1000	2000	3000
4000	5000	6000
k	**c**	**eh**
4000	5000	6000
500	1500	2500

Continued...

Printing of a pointer is always done by %d.
Sizeof always requires %d.

...Continued

i	a	ch
10	3.14	z
1000	2000	3000
j	**b**	**dh**
1000	2000	3000
4000	5000	6000
k	**c**	**eh**
4000	5000	6000
500	1500	2500

```
    printf ( "%d %d %d ", sizeof ( j ), sizeof ( b ), sizeof ( dh ) ) ;
    printf ( "%d %d %d", sizeof ( k ), sizeof ( c ), sizeof ( eh ) ) ;

    printf ( "%d %f %c" , **k, **c, **eh ) ;
}
```

(10 3.14 z).

why → Since declaration is diff..

2 2 2 → size of any type of pointer is 2 bytes

→ 2 2 2.

Why *far* and *near*

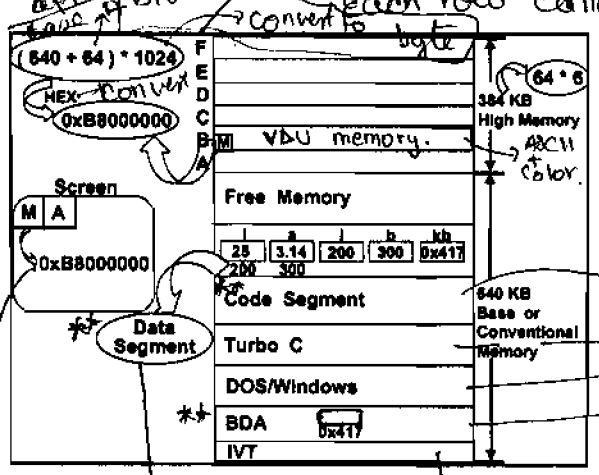
```
main( )
{
    int i = 25 ; int *j ;
    float a = 3.14 ; float *b ;
    char far *kb ;
    kb = 0x417 ;
    j = &i ;
    b = &a ;
}
```

mathematical approach.

Size = 64 Kilo
6 blocks
∴ 64×6 = 384 Kilo

(Base + block A)* byte Conversion
Hex conversion = 0xB8000000.

each row called block.



1024 bytes = 1 Kilobyte.

balance Memory = Extended Memory
|64-1 = Extended memory|

when executing prog, it goes here. (holds the code)

software.

O.S.

Bios Data Area → (Basic Input Output System)

Interrupt Vector Table.

screen has no add. as it has no memory.

holds data/variables of prog.

Thus a far pointer is holding a address away from the data segment.
(0x417 is in BDA.).

Pointer Sizes

```
main( )
{
   int i = 25 ; int *j ;
   float a = 3.14 ; float *b ;
   char far *kb ;
   kb = 0x417 ;
   j = &i ;
   b = &a ;
   printf ( "%d %d %d", sizeof ( j ), sizeof ( b ), sizeof ( kb ) )
}
```

2 2 4 → as far pointer.

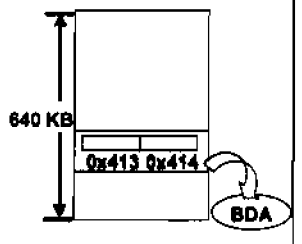
every near ptr = 2
far ptr = 4.

for pointers 4 bytes as storing add which is away from Data segment

VIRUS DETECTOR

How Much Memory

```
main( )
{
   int far *m ;
   m = 0x413 ;
   printf ( "%d" ,*m ) ;
}
```



Ideally → 640

? → 632 634 636

a virus present in base memory.

Size reduced by size of virus (in bytes).

int float char far* m.
- all depends on what you propose to do
- depends of no. of bytes you plan to access.

on retriving any add of variable we only gain the ASCII of that variable's location. Not the colour.

virus is not stationary in memory it only goes there when comp is on. Disk is cause & home for virus. Memory is the place from where it acts.

Memory is Volatile.

Cleaner for base Virus

*** → Switching off comp will remove virus from memory
A virus cleaner should clean virus from Disk not memory

```
main()
{
    int a = 10, b = 20 ;
    printf ( "%d%d", a, b ) ;
    swapv ( a, b ) ;
    printf ( "%d%d", a, b ) ;
}

swapv ( int x, int y )
{
    int t ;
    t = x ; x = y ; y = t ;
    printf ( "%d%d", x, y ) ;
}
```

	a	b
1020	10	20

	x	y	t
1020	10	20	g
	10	20	10
20 10	20	20	10
	20	10	10

a=10 b=20

→ CALL by Value : Copies of variables are being manipulated.

Using pointer concepts :-

```
main()
{
    int a = 10, b = 20 ;
    printf ( "%d%d", a, b ) ;
    swapr ( &a, &b ) ;
    printf ( "%d%d", a, b ) ;
}
swapr ( int *x, int *y )
{
    int t ;
    t = *x ;
    *x = *y ;
    *y = t ;
}
```

Call by reference → [and add is required]. eg: Printf → scanf

x=&a ∴ x should be a ptr. ∴ int *x.

	a	b
1020	10	20
20 10	200	300

	x	y	t
	200	300	10

value at x
value at 300 = value at 200

```
main()
{
    int a = 10, b = 20, c = 30, s, p ;
    sumprod ( a, b, c, &s, &p ) ;
    printf ( "%d%d", s, p ) ;
}
sumprod ( int x, int y, int z, int *ss, int *pp )
{
    *ss = x + y + z ;
    *pp = x * y * z ;
}
```

(60, 6000)

a	b	c	s	p
10	20	30	G	G
			100	200

x	y	z	ss	pp
10	20	30	100	200

→ value at address ss store x+y+z;

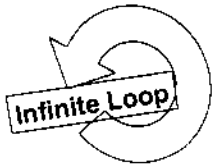
@Ashray Lal
Sheet XI

Advanced Features of Functions

- Returning Non-integer Value
- Call By Value / Reference
- Recursion

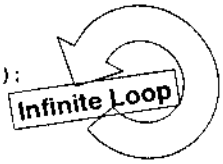
Simple Form

```
main( )
{
    printf ( "Hi" ) ;
    main( ) ;
}
```



One More Form

```
main( )
{
    f( ) ;
}
f( )
{
    printf ( "Hi" ) ;
    f( ) ;
}
```



More General

```
main( )
{
    int num, sum ;
    printf ( "Enter a number" ) ;
    scanf ( "%d", &num ) ;
    sum = sumdig ( num ) ;
    printf ( "%d", sum ) ;
}
sumdig ( int n )
{
    int d ; int s = 0 ;
    while ( n != 0 )
    {
        d = n % 10 ;
        n = n / 10 ;   s = s + d ;
    }
    return ( s ) ;
}
```

327

31698 → d5

372 → d3

n	s	d
327	0	7
32	7	2
3	9	3
0	12	

can also use while (n / 10 != 0)

```
main( )
{
    int num, sum ;
    printf ( "Enter a number" ) ;
    scanf ( "%d", &num )
    sum = rsum ( num ) ;
    printf ( "%d", sum ) ;
}
rsum ( int n )
{
    int d ; int s ;
    if ( n != 0 )
    {
        d = n % 10 ; n = n / 10 ;
        s = d + rsum ( n ) ;
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

327

(self/local call)

3

2+3

7+5

12

if one digit no. — use do while.
more than one → while is enough.
∴ in general use do while.

method is ↑ called <u>iteration</u> (for while)

← called <u>recursion</u> (for if).

Explanation of Recursion:-

```
rsum ( int n )
{
    if ( n != 0 )
    {
        d = 327 % 10 ;
        n = 327 / 10 ;
        s = 7 + rsum ( 32 ) ;
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

327

```
rsum ( int n )
{
    if ( n != 0 )
    {
        d = 3 % 10 ;
        n = 3 / 10 ;
        s = 3 + rsum ( 0 ) ;
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

```
rsum ( int n )
{
    if ( n != 0 )
    {
        d = 32 % 10 ;
        n = 32 / 10 ;
        s = 2 + rsum ( 3 ) ;
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

```
rsum ( int n )
{
    if ( n != 0 )
    {
        d =
        n =
        s =
    }
    else
        return ( 0 ) ;
    return ( s ) ;
}
```

Factorial Value

```
main( )
{
    int num, fact ;
    printf ( "Enter a number" ) ;
    scanf ( "%d", &num ) ;
    fact = factorial ( num ) ;
    printf ( "%d", fact ) ;
}
factorial ( int n )
{
    int p = 1 ;
    while ( n != 0 )
    {
        p = p * n ;
        n-- ;
    }
    return ( p ) ;
}
```

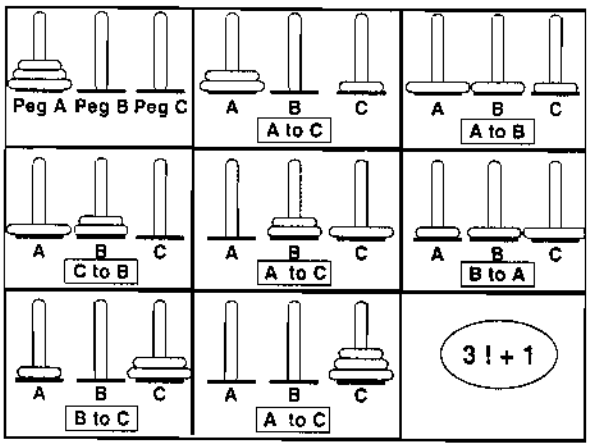
Recursive Factorial

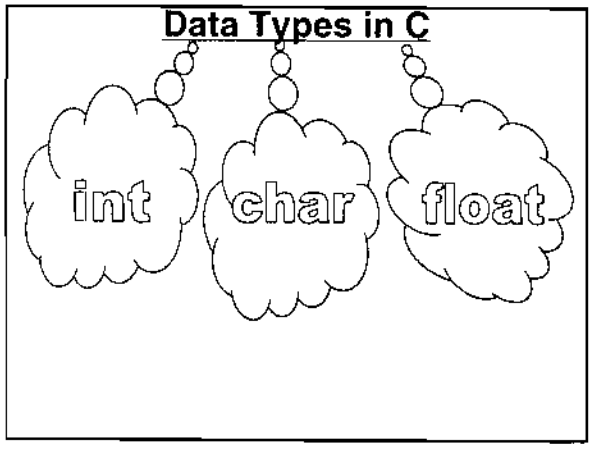
```
main( )
{
    int num, fact ;
    printf ( "Enter a number" ) ;
    scanf ( "%d", &num ) ;
    fact = refact ( num ) ;
    printf ( "%d", fact ) ;
}
refact ( int n )
{
    int p ;
    if ( n != 0 )
        p = n * refact ( n - 1 ) ;
    else
        return ( 1 ) ;
    return ( p ) ;
}
```

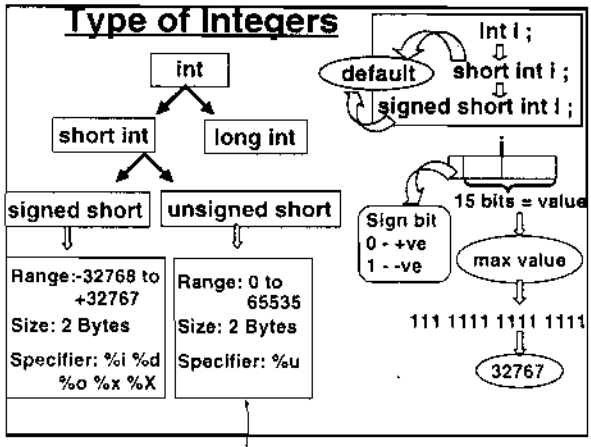
Advantages of Recursion

- Ease
- Speed

Exploring C solution



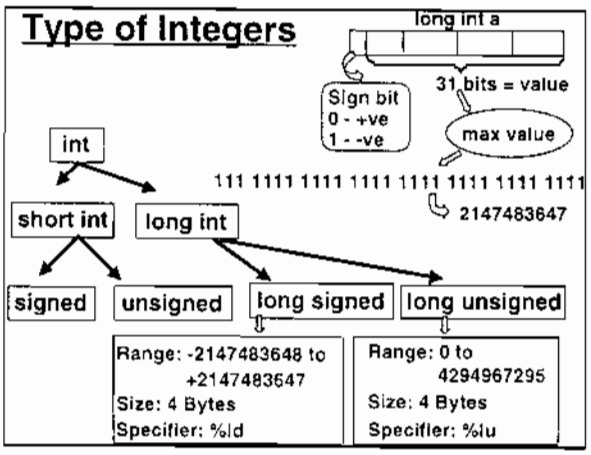


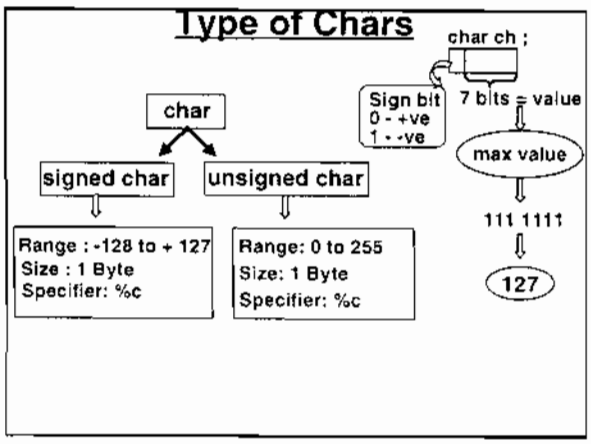


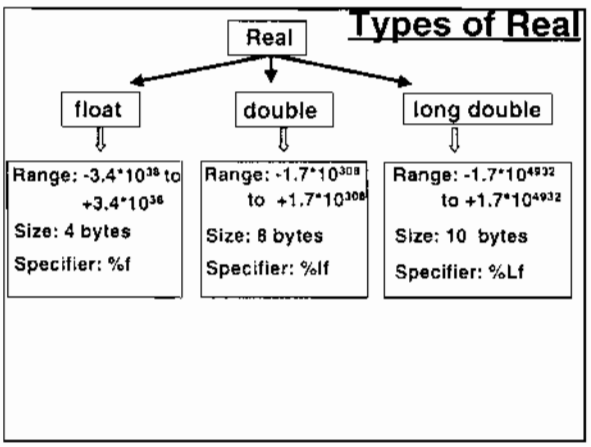
default short int.
1 bit for sign
no 15 bit storage
max 1 → is time.
converted into decimal — 32767.

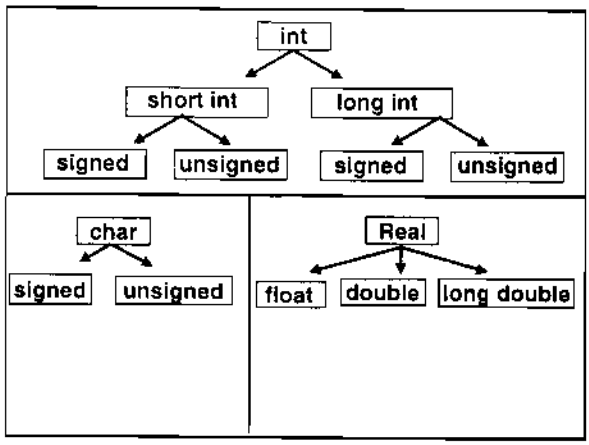
default for short int
= signed short int.

no bit saved for storage of sign. $(2 \times 32767 + 1)$


Type of Integers
long int a
Sign bit
0 - +ve
1 - -ve
31 bits = value
max value
111 1111 1111 1111 1111 1111 1111 1111
2147483647
int
short int
long int
signed
unsigned
long signed
long unsigned
Range: -2147483648 to +2147483647
Size: 4 Bytes
Specifier: %ld
Range: 0 to 4294967295
Size: 4 Bytes
Specifier: %lu


Type of Chars
char ch ;
Sign bit
0 - +ve
1 - -ve
7 bits = value
max value
111 1111
127
char
signed char
unsigned char
Range : -128 to + 127
Size : 1 Byte
Specifier: %c
Range: 0 to 255
Size: 1 Byte
Specifier: %c


Types of Real
Real
float
double
long double
Range: -3.4*10^38 to +3.4*10^38
Size: 4 bytes
Specifier: %f
Range: -1.7*10^308 to +1.7*10^308
Size: 8 bytes
Specifier: %lf
Range: -1.7*10^4932 to +1.7*10^4932
Size: 10 bytes
Specifier: %Lf



Chars With a Sign?

```
char ch = 'A' ;
```

ch

65 → binary

```
char dh = -15 ;
```

dh

-15 — binary of -15

now since one byte

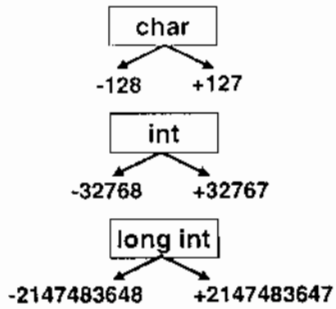
(-) (+) → gone ASCII.

What If We Exceed The Range?

```
signed char ch = 128 ;
printf ( "%d", ch ) ;
```

—128

Why This Bias?



signed

char ch = -128 ;	unsigned char dh = 128 ;	char eh = 128 ;
⇩	⇩	⇩
Binary of 128	Binary of 128	Binary of 128
⇩	⇩	⇩
10000000	10000000	10000000
⇩	printf ("%u", dh) ;	⇩
1's complement		Add 0 for +ve Sign
⇩		⇩
01111111		010000000
⇩		⇩
2's complement	01111111 +1 = 10000000	Store right 8 bits ⇩ lowest
⇩		
10000000		10000000
printf ("%d", ch) ;		printf ("%d", eh) ;

what gets stored in ch is 2'compliment of no.

signed 1 00 ↑ represents the sign.

↳ compiler goes backwards since 1 present
→ sets 1 → 1's comp
→ 1 at (7) = 128.
now 1 in 2' comp
∴ −128.

Various Forms

signed short int i ; short signed int i ; short int i ; signed int i ; int i ; short i ; signed i ;	long signed int k ; signed long int k ; long int k ; long k ;	float a ; double d ; long double e ;
	long unsigned int i ; unsigned long int i ; unsigned long i ;	
unsigned short int j ; short unsigned int j ; unsigned int j ; unsigned j ;	signed char ch ; char signed ch ; char ch ;	
	unsigned char dh ; char unsigned dh ;	

What Is 365?

int →	365
long int →	365L 365l
unsigned int →	365u
unsigned long int →	365lu 365ul

What Is 3.14?

double →	3.14
float →	3.14f
long double →	3.14L

Would This Work

```
main()
{
    char i ; unsigned char i ;
    for ( i = 0 ; i <= 255 ; i++ )
        printf ( "%d %c", i, i ) ;
}
```

infinite loop as after 127 → −128 which is lesser than 255

Stick To Your Guns

```
main()
{
    unsigned char i ;
    for ( i = 0 ; i <= 255 ; i++ )
        printf ( "%d %c", i, i ) ;
    printf ( "%d %c", i, i ) ;
}
```

best → int i;

inite loop.

255 + 1 = −0

0 < 255 ✓

∴ continues.

if i < 255 then unsigned char works. but last value (255) won't be printed

∴ ⇒ outside loop printf.

Storage Classes In C

A Complete definition of variable :

- ⇨ Type
- ⇨ Storage Class

A Storage Class signifies

- ⇨ Storage → Place where it is store.
- ⇨ Default Initial Value → stores range of variable.
- ⇨ Scope of variable → avalible (local, global)
- ⇨ Life of variable → how long it will occupy space.

Types of Storage Classes

- ⇨ Automatic → default.
- ⇨ Register
- ⇨ Static
- ⇨ External

$a=5$ $b=6$ $c=7$.

$a+=b+=c;$

sequence of execution

$b=b+c$
$a=a+b$

$a=18, b=13, c=7$

Order of execution of all *, / (same priority) left to right.

Expressions separated by a comma are evaluated left to right

$a=2$
$b=a+1$ } $b=(a=2, a+1);$

order
$a=2 \rightarrow$ 2 assig to a
$a+1 \rightarrow$ 1 added to a $\rightarrow 3$
$b=3 \rightarrow$ 3 ass to b.

Associativity (order of execution) of += is from right to left.

---

the value of the left expression is discarded.

$s=(5,6,7)$

7 is ass to s after discarding 5 & 6.

The ascii characters corresponding to the following ascii values don't get printed using printf();

7 8 9 10, 13, 26

S=0
for i<255
*(s+i*i)

kbhit() keyboard hit.
Std library func.
returns a 1 or 0.

Axay—12

Automatic Storage Class

Storage	⇨	Memory
Default Initial Value	⇨	Garbage
Scope	⇨	Local to the block in which the variable is defined
Life	⇨	Till the control is in the block in which the variable is defined

block = pair of braces.

Automatic Storage Class

```
main( )
{
    int a ;                  → Automatic int a
    static int b ;           → static
    automatic int c ;        → same as 1 (default no need to includ
    printf ( "%d %d %d", a, b, c ) ;
}
```

↑ error cause keyword is auto
∴ auto int c;

Default Initial Value

```
main( )
{
    int a ;
    static int b ;
    auto int c ;
    printf ( "%d%d%d", a, b, c ) ;
}
```

output
garbage 0 garbage.

∴ default initial value for static int b = 0.
∴ storage class defines the initial value.

Automatic initial value = garbage.

Which Is Correct?

redefination

```
main( )
    int a = 10 ;
    float a = 3.14 ;
    .......
    .......
}
```

```
main( )
{
    int a = 10 ;
    int a = 20 ;
    .......
    .......
}
```

redeclaration.

Tip: Redefinition not allowed

Redefinition?

← No

```
main( )
{
  int a = 10 ;
  {
     int a = 20 ;
     {
        int a = 30 ;
        printf ( "%d", a ) ;
     }
     printf ( "%d", a ) ;
  }
  printf ( "%d", a ) ;
}
```

(1) (2) (3)

Error ! Why?

```
main ( )
{
  int a = 20 ;
  f( ) ;
}
f( ) :
{
  printf ( "%d", a ) ;
}
```

cos a is local to main. f() no a.

scope of variable is dependent on block (braces).

Cos a has 3 diff address ∴ diff memory location.

all 3 values of a are available at every printf ~~before~~ ~~in the NEXT~~ brace. ∴ The most local value will be printed out.

for (2) 30 won't be available 'cos life time of 30 is over

(1) 20 isn't available.

Death, But When?

```
main( )
{
    int i =10, j = 20, k, l ;
    k = f ( ) ;
    l = i + j + k ;
    printf ( "%d", l ) ;
}
f( )
{
    int m = 5, n ;
    n = m * 2 ;
    return ( n ) ;
}
```

Won't die when the control goes to f( )

Would die when control goes back

40

~~Just after funct call all the previously~~

when a call is made all previously declared variables don't die. But after f() is over m & n are dead.

∴ Whenever branching off takes place variables don't die but whenever we return from the branch all local variables to the branch die.

Register Storage Class

Storage : CPU Registers → 14 → Each of 2 bytes

Default Initial value :

Scope :

Life :

Central Processing Unit ⇒ Microprocessor ⇒ µp

Different Speeds?

```
main( )
{
   auto int i ;
   for ( i = 1 ; i <= 250 ; i++ )
      printf ( "%d %c", i, i ) ;
}
```

```
main( )
{
   register int i ;
   for ( i = 1 ; i <= 250 ; i++ )
      printf ( "%d %c", i, i ) ;
}
```

Register Storage Class

Storage : CPU Registers

Default Initial Value : Garbage

Scope : Local to the block in which the variable is defined

Life : Till the control remains in the block in which the variable is defined

Be Judicious

```
main( )
{
    register int i ; register int j = 20 ;
    for ( i = 1 ; i <= 250 ; i++ )
        printf ( "%d %c", i, i ) ;
    printf ( "%d", j ) ;
}
```

Static Storage Class

Storage	⇨	Memory
Default Initial Value	⇨	0
Scope	⇨	Local to the block in which the variable is defined
Life	⇨	Variable persists between different function calls

```
main( )
{
   Increment( ) ;
   increment( ) ;
   increment( ) ;
}
increment( )
{
   auto int i = 1 ;
   register int j = 1 ;
   static int k = 1 ;
   i++ ; j++ ; k++ ;
   printf ( "%d %d %d", i, j, k ) ;
}
```

When to Use *Static*

```
main( )
{
      int *p ; int * f( ) ;
      p = f( ) ;
      printf ( "%d", *p ) ;
}
int * f( )
{
      static int a = 20 ;
      return ( &a ) ;
}
```

a	p
20	250
250	

```
char ** f ( int *, float * ) ;
```

External Storage Class

```
int a = 10 ;
main( )
{
    printf ( "%d", a ) ;
    increment( ) ;
    increment( ) ;
    decrement( ) ;
    printf ( "%d", a ) ;
}
increment( )
{
    a++ ; printf ( "%d", a ) ;
}
decrement( )
{
    a- - ; printf ( "%d", a ) ;
}
```

Output
10
11
12
11
11

Declaration V/s Definition

```
main( )
{
    extern int a ;
    printf ( "%d", a ) ;
    increment( ) ;
    increment( ) ;
    decrement( ) ;
    printf ( "%d", a ) ;
}
increment( )
{
    extern int a ;
    a++ ; printf ( "%d", a ) ;
}
decrement( )
{
    extern int a ;
    a- - ; printf ( "%d", a ) ;
}
int a = 10 ;
```

Declaration

Declaration

Declaration

Definition

```
float square ( float ) ;

float square ( float )
{
    ..
    ..
}
```

Definition

Declaration V/s Definition

```
main( )
{
    extern int a ;
    printf ( "%d", a ) ;
    increment( ) ;
    increment( ) ;
    decrement( ) ;
    printf ( "%d", a ) ;
}
increment( )
{
    extern int a ;
    a++ ; printf ( "%d", a ) ;
}
int a = 10 ;
decrement( )
{
    extern int a ;
    a- - ; printf ( "%d", a ) ;
}
```

Declaration V/s Definition

```
main( )
{
        extern int a ;
        printf ( "%d", a ) ;
        increment( ) ;
        increment( ) ;
        decrement( ) ;
        printf ( "%d", a ) ;
}
int a = 10 ;
increment( )
{
        extern int a ;
        a++ ; printf ( "%d", a ) ;
}
decrement( )
{
        extern int a ;
        a- - ; printf ( "%d", a ) ;
}
```

Two Types Of Conflicts

```
int a = 10 ;
main( )
{
        int a = 20 ;
        {
            int a = 30 ;
            printf ( "%d", a ) ;
        }
        printf ( "%d", a ) ;
}
printf ( "%d", a ) ;
```

External Storage Class

Storage	⇨	Memory
Default Initial Value	⇨	0
Scope	⇨	Global
Life	⇨	Till execution of the program doesn't end

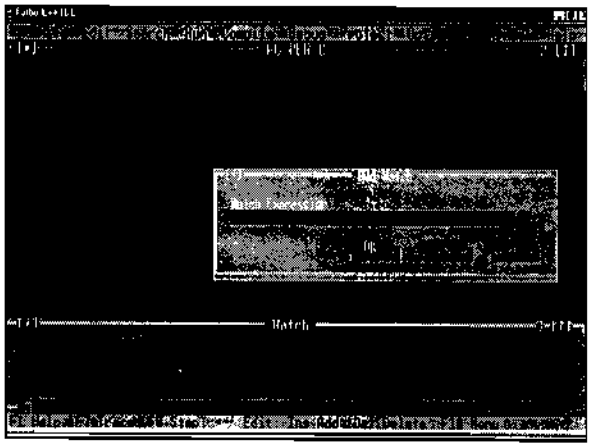
Which is The Most Powerful

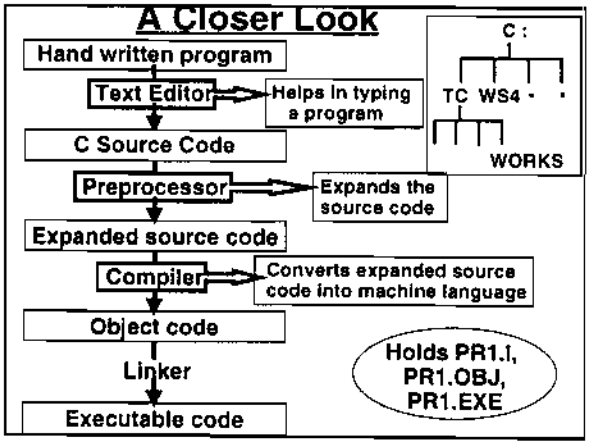
- Automatic — All other cases
- Register — For frequently used variables
- Static — If variable is to live across function calls
- External — If variable is required by all functions

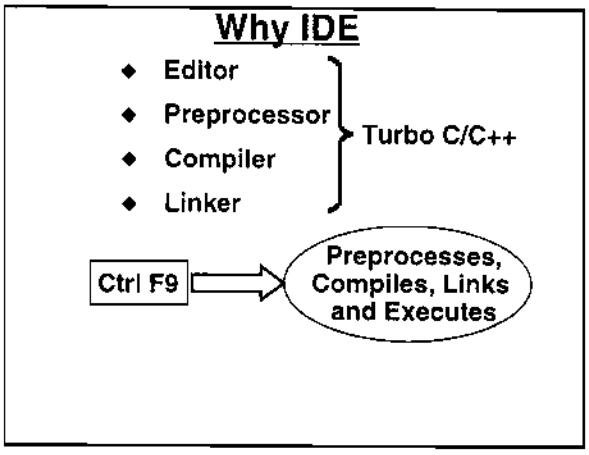
C Preprocessor

Ctrl F9	Compile and Execute
F9	Compile
F7	Step into
F8	Step over

```
main()
{
  int a = 10;
  float b = 3.5;
  display(a);
  printf("Hello");
}
display()

  printf("Nagpur");
}
```




A Closer Look
Hand written program
Text Editor
Helps in typing a program
C Source Code
Preprocessor
Expands the source code
Expanded source code
Compiler
Converts expanded source code into machine language
Object code
Linker
Executable code
Holds PR1.I, PR1.OBJ, PR1.EXE
C :
TC WS4
WORKS


Why IDE
Editor
Preprocessor
Compiler
Linker
Turbo C/C++
Ctrl F9
Preprocesses, Compiles, Links and Executes

Types of Preprocessor Directives

- File Inclusion
- Macro Expansion
- Conditional Compilation
- Miscellaneous Directives

Preprocessor In Action

```
# include "goto.c"
main( )
{
    clrscr( ) ;
    gotorc ( 10, 20 ) ;
    printf ( "Hello !" ) ;
}
```

Source Code

Ctrl F9 → Preprocessor

```
gotorc( - , - )
{
    -
    -
    -
}
main( )
{
    clrscr( ) ;
    gotorc ( 10, 20 ) ;
    printf ( "Hello !" )
}
```

Expanded source code

Compiler →

Unresolved Externals

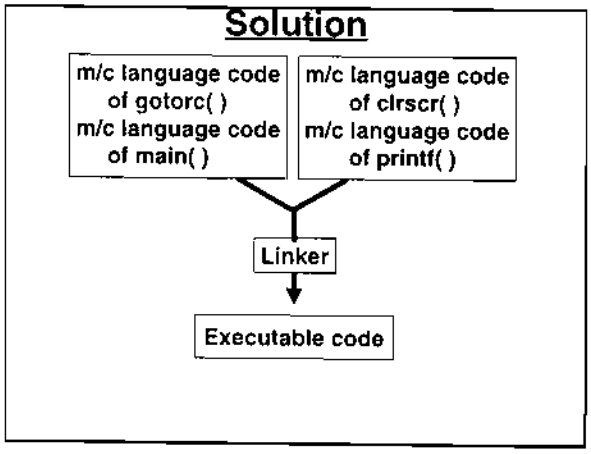
```
gotorc( - , - )
{
    -
    -
    -
}
main( )
{
    -
    -
    -
}
```

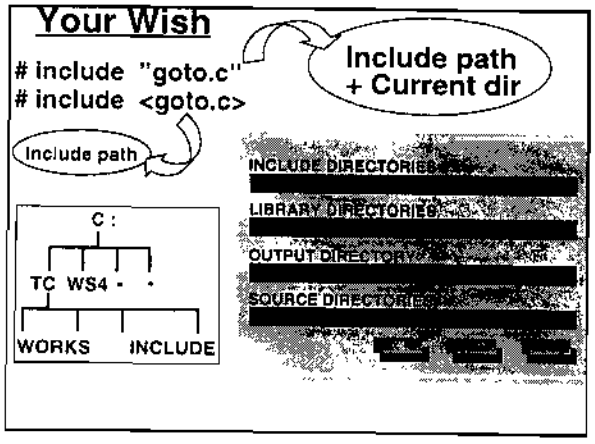
Compiler →

machine language code of gotorc( )

machine language code of main( )

Error → Why?





diff func Prototype & func call :-

Proty → type of argument & type of return values to be to be used in func.

float fun(int, float); ← proto

z = fun (int, float ; ← call

float fun ()
{ ← defination.
}

Axay Lal

Sheet – 13

Macro Expansion

```
# define LOWER 1
# define UPPER 10
main( )
{
    int i ;
    for ( i = LOWER ; i <= UPPER ; i++ )
        printf ( "\n%d", i ) ;
}
```

Macro Expansion

Macro Template

```
for ( i = 1 ; i <= 10 ; i++ )
```

They are deleted after just the substitution has been done

If you are using a variable no. of times it is better to use this

Macro Expansion

```
# define PI 3.14
main( )
{
    float r, a ;
    printf ( "Enter radius" ) ;
    scanf ( "%f", &r ) ;
    a = PI * r * r ;
    printf ( "\n%f", a ) ;
}
```

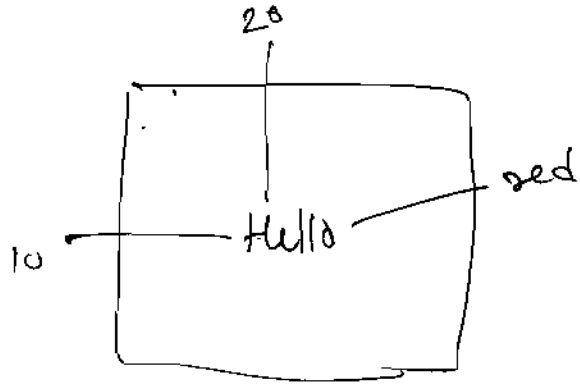
3.141528

```
float PI = 3.14 ;
PI = 6.28 ;
```

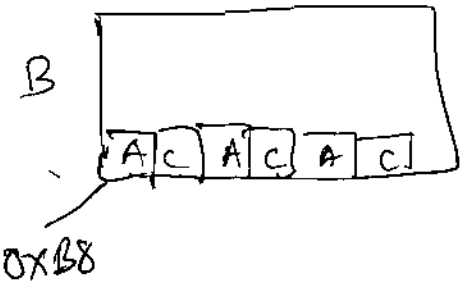
For constant values

```
# define PI 3.14
PI = 6.28 ;
```

```
a = 3.14 * r * r ;
```



*v = 'H';
*(v+1) = color;



Don't Remember Constants

```
# define PLANK 6.634E-24
main( )
{
    a = PLANK * ... ;
    b = PLANK / .... ;
    c = PLANK + ... ;
    d = PLAN - .... ;
}
```

Plank's Const.
6.634 * .....

Avogadro's No.
6.023 * .....

char mess[] = {.....};

for (————)
{
    *v = mess[i];
    *(v+1) = color;
}

Types Of Macros

- Simple Macros
- Macros With Arguments

Macros With Arguments

```
# define  PI  3.14
# define  AREA ( x )  PI * x * x
main( )
{
        float  r, a ; float  area ( float  rr ) ;
        scanf ( "%f", &r ) ;
        a = AREA ( r ) ;
        printf ( "\n%f", a ) ;
        a = area ( r ) ;
}
float area ( float  rr )
{
        return ( PI * rr * rr ) ;
}
```

Order is unimportant

a = 3.14 * r * r ;

Macros - Faster
Functions - Less Space

Macros may not always be in capital.
Preprocessor ~~compiler~~ directly substitutes the expression, hence no calling as # already deleted by then
give a \ after the define.
define A = ...

Macros With Arguments

```
# define S ( x )  x * x
main( )
{
        int  i, j, k, l, m, n = 2 ;
        i = S ( 4 ) ;
        j = S ( 2 + 2 ) ;
        k = S ( 3 + 1 ) ;
        l = S ( 1 + 3 ) ;
        m = S ( ++n ) ;
        printf ( "%d %d %d %d %d %d", i, j, k, l, n, m ) ;
}
```

i = 4 * 4 ;
j = 2 + 2 * 2 + 2 ;
k = 3 + 1 * 3 + 1 ;
l = 1 + 3 * 1 + 3 ;
m = ++n * ++n ;

Alt F Dos Shell
C> CPP PR1.C
C> exit

there should be no space

→ 16 8 7 7 4 16 Borland.

In Microsoft. — 12

C preprocessor.

PR1.I
↓
pass it only to ~~compiler~~ preprocessor not compiler as it will again delete it as it has error.
So do this ⇒

Solution...

```
# define S( x ) ( x * x )
k = S ( 3 + 1 ) ;
```

```
k = ( 3 + 1 * 3 + 1 ) ;
```

```
# define S( x )  x * x
k = S ( ( 3 + 1 ) ) ;
```

```
k = ( 3 + 1 ) * ( 3 + 1 ) ;
```

```
# define S( x ) ( x ) * ( x )
k = S ( 3 + 1 ) ;
```

```
k = ( 3 + 1 ) * ( 3 + 1 ) ;
```

Conditional Compilation

```
main( )
{
        ..
        ..
        ..

        /*
        scanf ( .. ) ; /* input */
        ..
        .. /* formula */
        ..  */
}
```

Conditional Compilation

```
main( )
{
        ..
        ..
        ..

        # ifdef YES
          scanf ( .. ) ; /* input */
          ..
          .. /* formula */
          ..
        # endif
}
```

2 Solutions :
- # define YES
- Delete # ifdef, # endif

Miscellaneous Directives

```
# define YES 10
main( )
{
        a = YES + .. ;
        # define YES 20
        a = YES + .. ;
        printf ( "YES" ) ;
        # undef YES
}
f( )
{
        int  YES ;
}
```

Redefinition OK

Never Replaced

undefined macros

pragma → misc. directory

```
# pragma inline
main( )
{
        ..
        ..
        ..
        ..
        asm stc
        asm pushf
        asm pop flags
        ..
}
```

F7, F8, F9, Ctrl F9 Won't Work

Extended Keyword (assembly line instr.)

flag register int14 CPU registers

Preprocessor cmd

Arrays

```
main( )
{
    int  m1, m2, m3, per ;
    int  i ;
    for ( i = 1 ; i <= 10 ; i++ )
    {
        printf ( "Enter Marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per = ( m1 + m2 + m3 ) / 3 ;
        printf ( "%d", per ) ;
    }
    printf ( "%d", per ) ;
}
```

Choices Available

- Use 10 variables each holding 1 value
- Use 1 variable holding all 10 values

  Array

- What is an Array?
  Array is a variable capable of holding more than 1 value at a time

Nothing Different

per = { 32, 62, 65, 42, 48, 70, 80, 86, 92, 68 }

per_1 per_3 per_6 per_{10}

In General per_i

H_2O — subscript

H^2O — superscript

Screen

per

i per[i]

per (i) per [i] — subscripted variable

```
main( )
{
    int m1, m2, m3, per[ 10 ] ;
    int i ;
    for ( i = 1 ; i <= 10 ; i++ )
    {
        printf ( "Enter Marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per[ i ] = ( m1 + m2 + m3 ) / 3 ;
    }
    for ( i = 1 ; i <= 10 ; i++ )
        printf ( "%d", per[ i ] ) ;
}
```

Array?

0 9

0 9

0 A

Screen

size or dimension of array.

Initializing Arrays

```
main()
{
    int a[ ] = { 7, 6, 11, -2, 26 } ;
    int b[ 10 ] ;
    int c[ 10 ] = { 16, 13, -8, -7, 25 } ;
    printf ( "%d%d", sizeof ( a ), sizeof ( b ) ) ;
    printf ( "%d%d", a[ 0 ], b[ 0 ] ) ;
    scanf ( "%d%d%d", &c[ 7 ], &c[ 8 ], &c[ 9 ] ) ;
    c[ 5 ] = 3 + 7 % 2 ;
    c[ 6 ] = c[ 1 ] + c[ 3 ] / 16 ;
}
```

(optional)

(compulsory)

int i ; i = 2 ; ⇒ int i = 2 ;

→ {0, 1, 2, 3, 4} here we initialize the array at time of declaration ∴ size not req.

→ 10, 20

7 garbage

Moral

- ⇨ Arrays can be initialized
- ⇨ Arrays have storage classes
- ⇨ Array elements can be scanned
- ⇨ Array elements can be calculated
- ⇨ Arithmetic on array elements Is allowed

Then how are they different?

Arrays can hold several values at a time whereas a variable cannot.

Storage

i	j	k	l	m
3	20	-5	7	11
100	400	500	700	600

```
main( )
{
    int i = 3, j = 20, k = -5, l = 7, m = 11 ;
    int a[ ] = { 3, 20, -5, 7, 11 } ; int ii ;
    printf ( "%u %u %u %u %u", &i, &j, &k, &l, &m ) ;
    for ( ii = 0 ; ii <= 4 ; ii++ )
        printf ( "%u", &a[ ii ] ) ;
}
```

→ size optional

(int, double, char, int) → on initializing array as int all inside variables get stored as ~~arrays~~ integ

```
int a[ ] = { 2, 1.4, 'A', 6 } ;
```

- Adjacency
- Similarity

a[0]	a[1]	a[2]	a[3]	a[4]
3	20	-5	7	11
502	504	506	508	510

why %u over %d : use of ptr : pointer function . diff ways of declaring a func

Bounds Checking

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
3	60	-5	7	11		
500	502	504	506	508	510	512

```
main()
{
    int a[] = { 3, 60, -5, 7, 11 } ;
    int i ;
    for ( i = 0 ; i <= 40 ; i++ )
        a[ i ] = a[ i ] * 2 ;
    for ( i = 0 ; i <= 40 ; i++ )
        printf ( "%d", a [ i ] ) ;
}
```

~~Subscript out of range~~

all languages but C.

at i=5.
value at A[5] gets multiplied by 2 & gets stored. (Garbage*2).

2 outputs: ① as above till i=4
② hang

② cos sometimes in array ~~sometimes~~ after the bounds of ~~array~~ it we store over the base of TC. Thus disrupting the prog./compiler.

So ...

- Arrays are variables capable of storing multiple values
- Array elements are stored in adjacent memory locations
- Array elements are always similar
- Checking the bounds of an array is programmer's responsibility

Useage of Arrays

compare one position with rest.

I. **Selection Sort**

```
main()
{
    int a[] = { 17, 6, 13, 12, 2 } ;
    int i, j, t ;
    for ( i = 0 ; i <= 3 ; i++ )
    {
        for ( j = i + 1 ; j <= 4 ; j++ )
        {
            if ( a[ i ] > a[ j ] )
            {
                t = a [ i ] ; a[ i ] = a[ j ] ;
                a[ j ] = t ;
            }
        }
    }
    for ( i = 0 ; i <= 4 ; i++ )
        printf ( "%d", a[ i ] ) ;
}
```

condition

17	6	13	12	2	i	j
6	17	13	12	2	0 - 1	
6	17	13	12	2	0 - 2	
6	17	13	12	2	0 - 3	
2	17	13	12	6	0 - 4	
2	13	17	12	6	1 - 2	
2	12	17	13	6	1 - 3	
2	6	17	13	12	1 - 4	
2 6	13	17	12		2 - 3	
2 6	12	17	13		2 - 4	
2 6 12	13	17			3 - 4	

basic funda.

i varies from 0 – 3
j varies from 1-4 for every change in i. (i+1 → end)

output statement.

↑ here the smallest gets positioned in the first run then the larger, till lagest.

keep comparing adjacent no. only

11.

Bubble Sort

```
main()
{
  int a[ ] = { 17, 6, 13, 12, 2 } ;
  int i, j, t ;
  for ( j = 0 ; j <= 3 ; j++ )
  {
    for ( i = 0 ; i <= 3 - j ; i++ )
    {
      if ( a[ i ] > a[ i + 1 ] )
      {
        t = a[ i ] ; a[ i ] = a[ i + 1 ] ;
        a[ i + 1 ] = t ;
      }
    }
  }
  for ( i = 0 ; i <= 4 ; i++ )
    printf ( "%d", a[ i ] ) ;
}
```

					i	i+1
17	6	13	12	2		
6	17	13	12	2	0	1
6	13	17	12	2	1	2
6	13	12	17	2	2	3
6	13	12	2	17	3	4
6	13	12	2	17	0	1
6	12	13	2	17	1	2
6	12	2	13	17	2	3
6	12	2	13	17	0	1
6	2	12	13	17	1	2
2	6	12	13	17	0	1

here the largest gets the correct position in the first run then smaller till smallest gets stored in first position.

$i \rightarrow 0-3$
$j \rightarrow 1-4$ (∵ i+1)

float **f( )
{
float a;
~~float *b, *c;~~
~~return (&f)~~
b = &a;
c = &b;
return (c);
}

n^2 comparison

Sorting Procedures

n^2 — For n = 10, Order = 100

- ★ Selection Sort
- ★ Bubble Sort
- ★ Shell Sort
- ★ Shuttle Sort
- ★ Heap Sort
- ★ Merge Sort
- ★ Radix Sort
- ★ Quick Sort ✓

$\log_2 n$ — For n = 10, Order = 3

qsort( ) → Library function

Recursion → uses recursion

Fastest? → Yes even though recursion is slow cos no of call are more but comparison are lesser. Thus faster.

lesser comparison

best procedure

Useage of qsort:-

Quick Sort At Work

```
main()
{
  int a[ ] = { 17, 6, 13, 12, 2 } ;
  int i ;
  int fun ( int *, int * ) ;
  qsort ( a, 5, sizeof ( int ), fun ) ;
  for ( i = 0 ; i <= 4 ; i++ )
    printf ( "\n%d", a[i] ) ;
}
fun ( int *x, int *y )
{
  return ( *x - *y ) ;
}
```

Comparison Function

Called $\log_2 n$ times

Called by qsort( )

for descending order (*y - *x)

here if 17-12 = +ve qsort interchanges

if 6-13 = -ve leaves it.

must be there

① which array, ② no of elements in array, ③ size of each element, ④ function name (comparison function)

logic of qsort if the change -ve leave it

** qsort is generic. So the type of /way of sorting is provided by the comparison function.

like in a list of info. sort first by name or date or salary or whatever so this logic must be supplied to the qsort( ).

Array Lal sheet-14

IMP

base element

Another Form...

a[0]	a[1]	a[2]	a[3]	a[4]
7	9	16	-2	8
102	104	106	108	110

```
main( )
{
  int a[ ] = { 7, 9, 16, -2, 8 } ;
  int i ;
  printf ( "%u %u %u", &a[ 0 ], &a[ 1 ], &a[ 2 ]) ;
  printf ( "%u %u %u", a, a + 1, a + 2 ) ;
  printf ( "%d %d %d", *a, *( a +1 ), *( a + 2 )) ;
  for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d", *( a + i ) ) ;
}
```

(102, 104, 106)

(102, 104, 106)

7 9 16

value at add.

OR

```
for ( i = 0 ; i <= 4 ; i ++ )
    printf ( "%d", a[ i ] )
```

Pointer Notation — faster

Subscript Notation — easier to understand

slower as internally it anyway converts to *(a+i)

Thus slower

* mentioning name of array alwa gives base address of array.

normally on printf a will give address of base ad of a.

* if it was an array of floats a+1 will be a+4 space jump as float is 4 bytes big

IMP

More Forms...

```
main( )
{
  int a[ ] = { 7, 9, 16, -2, 8 } ;
  int i ;
  for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d", a[ i ] ) ;
  for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d", *( a + i ) ) ;
  for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d", *( i + a ) ) ;
  for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d", i[ a ] ) ;
}
```

a[i] ⟷ i[a]

*(a + i) → *(i + a)

a[i] = *(a+i) = i[a] = *(i+a) (all same)

same

*** Moral

without actually seeing the array declaration we cannot be sure which is the subscript & which is the array name.

Flexible Arrays

int a[5] ; ⟷ int 5[a] ; ✗

printf ("%d", a[2]) ; ⟷ printf ("%d", 2[a]) ; ✓

```
int a[ n ] ;
scanf ( "%d", &n ) ;
```

✗ Scanf above int will never work

*** Size of an array must always be mentioned as a positive, non-zero, integer constant.

*** on incrementing / decrementing a ptr. it points to the next location of its type.

Pointer Arithmetic

```
main( )
{
   float  a = 3.14, *b ;
   char  ch = 'z', *dh ;
   int  i = 25, *j ;
   b = &a ; dh = &ch ; j = &i ;
   printf ( "%u %u %u", b, dh, j ) ;
   b++ ;  dh++ ;  j++ ;
   printf ( "%u %u %u", b, dh, j ) ;
   b += 3 ; dh += 8 ; j -= 3 ;
   printf ( "%u %u %u", b, dh, j ) ;
}
```

a	ch	i
3.14	z	25
1008	2009	6002

b	dh	j
1008	2009	6002

printf: — 1008 2009 6002

b++ ; dh++ ; j++ ; printf: — 1012 2010 6004 → +4 float, +1 char, +2 int

b += 3 ; dh += 8 ; j -= 3 ; printf: — 1024 2018 5998 → +12 float, +8 char, -6 int

Legal Pointer Arithmetic

Pointer + number	→	Pointer
Pointer - number	→	Pointer
Pointer - Pointer	→	Number

Ptr + Ptr is illegal

* / % — illegal

→ holds base add of a. it becomes a ptr. (integer ptr)

Access Using Pointers

```
main( )
{
   int  a[ ] = { 7, 9, 16, -2, 8 } ;
   int  *p ; int  i ;
   p = a ; /* same as &a[ 0 ] */
   printf ( "%d", *p ) ;
   p++ ;
   printf ( "%d", p ) ;
   printf ( "%d", *p ) ;
   p = a ;
   for ( i = 0 ;  i <= 4 ; i++ )
   {
      printf ( "%d", *p ) ;
      p++ ;
   }
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
7	9	16	-2	8
102	104	106	108	110

p (102) —p++→ p (104)

*p ++ ; (Ist IInd)

* ++p ; (IInd Ist)

++ *p ; 8 9 10 11 12

Once the base add of the array is know a ptr can be used to print or manipulate the values in the array.

*p = *p + 1

p remains fixed at 102. thus (7+1) (8+1) (9+1)
8 9 10

[ ] For Notation...

```
main( )
{
    int a[ ] = { 7, 9, 16, -2, 8 } ;
    int *p ; int i ;
    p = a ;
    for ( i = 0 ; i <= 4 ; i ++ )
      printf ( "%d %d %d %d", *( p + i ), *( i + p ),
                              p[ i ], i[ p ] ) ;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
7	9	16	-2	8
102	104	106	108	110

output.

7 7 7 7
9 9 9 9
16 16 16 16
-2 -2 -2 -2
8 8 8 8

here P is pointing to the array
i is just an integer
∴ without lookin' at the declaration we cannot state which is the array & which isn't.

Dancing Dolls Revisited

```
main( )
{
    char far *s = 0xB8000000 ; int i ;
    for ( i = 0 ; i <= 3999 ; i += 2 )
    {
        if ( *( s + i ) >= 'A' && *( s + i ) <= 'Z' )
    }
}
```

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

a [i]
- One is pointer or array
- Other is int

a[i] i[a] (a+i)(i+a)
out of a & i one has to be a no the other has to give an add thus has to be either an array or a pointer.

Changing Array Address

```
main( )
{
    int a[ ] = { 7, 9, 16, -2, 8 } ;
    int *p ; int i ;
    p = a ;
    for ( i = 0 ; i <= 4 ; i ++ )
    {
        printf ( "%d", *p ) ;
        p++ ;
    }
    for ( i = 0 ; i <= 4 ; i ++ )
    {
        printf ( "%d", *a ) ;
        a++ ;
    }
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
7	9	16	-2	8
102	104	106	108	110

a--;
a = a + 2 ;
a = a - 2 ;
a += 2 ;
a -= 2 ;

all will give errors.

a++ → error

a = a+1

(a) = 102+1 = 104

5 ways to print out array elements through ptr.
1) (a+i)
2) (i+a)
3) a[i]
4) i[a]
5) *a
a++

we are trying to restore the address of a as 104 from 102.

P++ is OK as P is a ptr. so it doesnt alter the array in any way as such

Passing Array Elements

```
main( )
{
    int a[ ] = { 7, 9, 16, -2, 8 } ;
    int i ;
    display ( a[ 0 ], a[ 1 ], a[ 2 ], a[ 3 ], a[ 4 ] ) ;
    for ( i = 0 ; i <= 4 ; i++ )
        display1 ( a[ i ] ) ;
}
display ( int i, int j, int k, int l, int m)
{
    printf ( "%d %d %d %d %d", i, j, k, l, m ) ;
}
display1 ( int n )
{
    printf ( "%d", n ) ;
}
```

Which is good?

→ faster longish

→ shorter slower

Passing Entire Array

```
main( )
{
    int a[ ] = { 7, 9, 16, -2, 8 } ;
    display2 ( a ) ;
    display3 ( a, sizeof ( a ) / 2 - 1 ) ;
}
display2 ( int *p )
{
    int i ;
    for ( i = 0 ; i <= 4 ; i ++ )
        printf ( "%d", *( p + i ) ) ;
}
display3 ( int *p, int n)
{
    int i ;
    for ( i = 0 ; i <= 4 ; i ++ )
        printf ( "%d", *( p + i ) ) ;
}
```

a[0]	a[1]	a[2]	a[3]	a[4]
7	9	16	-2	8
102	104	106	108	110

qsort (a, 5, , ,) ;

in general ($i \leq n$)

to get ~~siz~~ no of elements in the array.

sizeof (a) / 2 = for exact size (1 - n)

(a) / 2 - 1 for exact size (0 - (n-1))

Remember...

Any time an entire array is to be passed to function, it is necessary to pass

(1) Base address of array

(2) No. of elements present in the array

Two Dimensional Array

```
main( )
{
  int a[ ][5] = {
                  { 2, 6, 1, 8, 4 },
                  { 1, 2, 5, 6, 8 },
                  { 7, 9, 8, 7, 21 }
                  { 4, 5, 6, 8, 10 }
                 };
  int i, j ;
  printf ( "%d", a[ 2 ][ 4 ] ) ;
  printf ( "%d %d", sizeof ( a ), a ) ;
  for ( i = 0 ; i <= 3 ; i++ )
  {
    for ( j = 0 ; j <= 4 ; j++ )
      printf ( "%d", a [ i ][ j ] ) ;
    printf ( "\n" ) ;
}
```

optional

compulsory

optional

compulsory

optional

int b[ ][1][2][3]

compulsory

row, Column

no. of rows are usually optional

Columns are <u>never</u> optional.

21 → 3rd row but 0 1 2 thus 2

5th column but 0 1 2 3 4 thus 4

base address.

2 loops for traversing of whole array.

2x20 integers = 40 * When using sizeof we get the size of the whole array & not only the base address.

Find Biggest...

```
main( )
{
    int a[ ][4 ] = {
                     7, 2, 6, 1,
                     3, 5, 4, 8,
                     6, 2, 9, 50,
                     1, 2, 3, 8
                    };
    int i, j, big ; int r, c ;
    big = a[ 0 ][ 0 ] ; r = 0 ; c = 0 ;
    for ( i = 0 ; i <= 3 ; i++ )
    {
        for ( j = 0 ; j <= 3 ; j++ )
        {
            if ( a[ i ][ j ] > big )
            {
                big = a[ i ][ j ] ;
                r = i ; c = j ;
            }
        }
    }
    printf ( "%d ", big ) ; printf ( "%d %d ", r, c ) ;
}
```

Find Second Biggest and its Position

initial

scaning

check

value

position.

C is always Row Major

Pascal is always Column major.

```
main( )
{ int a[ ][4 ] = {
                   7, 2, 6, 1,
                   9, 3, 4, 5,
                   10, 12, 16, 18
                  };
```

Row Major

all values of array stored in rows

7	2	6	1	9	3	4	5	10	12	16	18
502	504	506	508	510	512	514	516	518	520	522	524

Similar elements

adjacent locations.

```
502 ⇦ printf ( "%d ", a ) ;
  7 ⇦ printf ( "%d ", *a ) ;
~~502 504 506~~ ⇦ printf ( "%d %d %d ", a+0 , a+1 , a+2 ) ;
~~7 2 6~~ ⇦ printf ( "%d %d %d ", *(a+0) , *(a+1) , *(a+2) ) ;
~~7 2 6~~ ⇦ printf ( "%d %d %d ", a[0] , a[1] , a[2] ) ;
~~8 4 9~~ ⇦ printf ( "%d %d %d ", a[0]+1 , a[1]+2 , a[2]+3 ) ;
~~Garbage/~~ Error ⇦ printf ( "%d %d %d ", *(a[0]+1) , *(a[1]+2) ,
                                           *(a[2]+3) ) ;
2 4 18 ⇦ printf ( "%d %d %d ", a[0][1] , a[1][2] , a[2][3] ) ;
}
```

```
int a[ ][4] = {
                7, 2, 6, 1,
                9, 3, 4, 5,
                10, 12, 16, 18
              };
```

7	2	6	1	9	3	4	5	10	12	16	18

502 510 518

printf ("%d", a) ; 502

printf ("%d", *a) ; 502

```
int b[ ] = { 7, 2, 6, 1 } ;
```

7	2	6	1

402 404 406 408

printf ("%d", b) ; 402

printf ("%d", *b) ; 7

```
int i = 5 ;
printf ( "%d", i ) ;
```

5

i
5

200

name of array always ~~points to zero~~th acts as a pointer to zero^th element of the array.

*a will give 502. each grp of 4 act as 1 element. & as read above we still are looking at element 1 but that one element has 4 values. These each 2D is a collection of 1D arrays.

↑ vimp

```
main( )
{ int a[ ][4] = {
                7, 2, 6, 1,
                9, 3, 4, 5,
                10, 12, 16, 18
              };
```

Row Major

7	2	6	1	9	3	4	5	10	12	16	18

502 510 514 518 524

```
        502 ⇐ printf ( "%d ", a ) ;
        502 ⇐ printf ( "%d ", *a ) ;
502 510 518 ⇐ printf ( "%d %d %d ", a+0 , a+1 , a+2 ) ;
502 510 518 ⇐ printf ( "%d %d %d ", *(a+0) , *(a+1) , *(a+2) ) ;
502 510 518 ⇐ printf ( "%d %d %d ", a[0] , a[1] , a[2] ) ;
504 514 524 ⇐ printf ( "%d %d %d ", a[0]+1 , a[1]+2 , a[2]+3 ) ;
    2  4  18 ⇐ printf ( "%d %d %d ", *(a[0]+1) , *(a[1]+2) ,
                                     *(a[2]+3) ) ;
    2  4  18 ⇐ printf ( "%d %d %d ", a[0][1] , a[1][2] , a[2][3] );
}
```

a+0 I^st element
a+1 IInd element
a+2 IIIrd element.

a[0] point at 502 a[0]+1 ~~in the first element +1 = 5~~ in I^st element second value = 504

a[1] IInd element

Moral...

a[2][3] ⇒ * (a[2] + 3) ⇒ * (* (a + 2) + 3)

Array	Subscript	Pointer
1-D	a[i]	*(a+i)
2-D	a[i][j]	*(*(a+i) + j)
3-D	a[i][j][k]	*(*(*(a + i) + j) + k)
4-D	a[i][j][k][l]	*(*(*(*(a + i) + j) + k)+ l)

↑ easier slower

↑ harder faster

3 Ways

```
main( )
{
    int a[ ][ 4 ] = {
                    7, 2, 6, 1,
                    9, 3, 4, 5,
                    10, 12, 16, 18
                  } ;
    int i, j ;
    for ( i = 0 ; i <= 2 ; i++ )
    {
        for ( j = 0 ; j <= 3 ; j++ )
            printf ( "%d %d %d", a[ i ][ j ],
                     *( a[ i ] + j ), *( *( a + i ) + j ) ) ;
    }
}
```

a[i][j][k]

*(a[i][j] + k)

*(*(a[i]) + j) + k)

*(*(*(a + i) + j) + k)

2D
a = 502
*a = 502
**a = 7.

1D
a = 502
*a = 7

3D
a = 502
*a = 502
**a = 502
***a = 7

Array of Pointers

Different?

```
main( )
{
    int a[ ][ 4 ] = {
                    7, 2, 6, 1,
                    9, 3, 4, 5,
                    10, 12, 16, 18
                  } ;
    int *p[ 4 ] ; int *r ;
    int ( *q )[ 4 ] ;
    p[ 0 ] = &a[ 0 ][ 0 ] ;
    p[ 1 ] = &a[ 2 ][ 2 ] ;
    p[ 2 ] = p[ 3 ] = &a[ 1 ][ 1 ] ;
    printf ( "%d", p ) ; 130
    r = a ; q = a ;
    r++ ; q++ ;
    printf ( "%d %d", r, q ) ; 504, 510
}
```

7	2	6	1	9	3	4	5	10	12	16	18
502				510				518			

p[ ]

502	522	512	512
130	132	134	136

r	q
502	502

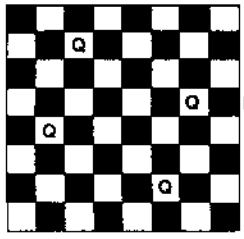
→ pointer to an array of 4 integers

→ array of ptrs.

→ as ptr of array of 4 integers
∴ (after 4 integers) increment ~~on~~ based on no. of integers associated with ptr.

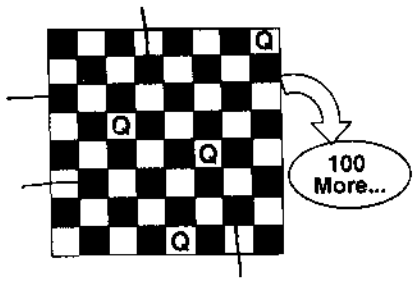
Applications of 2-D Arrays

- Matrices - Addition, Multiplication, Determinant, Transpose, Inverse, etc .
- Chess



→ 8 × 8 Board

The Solution



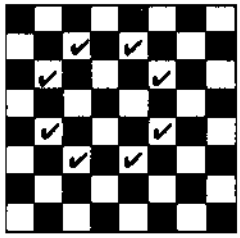
printf (" ", ++i * ++i)

*S = ++i * ++i 30

The Program...

```
int v, j, k, l, s, a[ 99 ] ;
main( )
{
   for ( scanf ( "%d", &s ) ;
         *a - s ; v = a[ ]*=v ] - a[i], k = i < s,
         j += ( v = j < s && ( !k && !! printf (
         2 + "\n\n%c" - ( !l << !j ),
         "Q" [ i^v ? ( l ^ j ) & 1 : 2] ) &&
         ++l !!a[l] < s && v && v - i + j && v + i - j ) )
         && !'( l % = s ), v !! ( l == j ? a[l += k] =
         0 : ++a[j] >= s *k && ++a[--i] )
            ;
}
```

Knight's Tour



Conditions :
- Visit every house
- Don't revisit any house

8	1	6
3	5	7
4	9	2

for odd sized matrices. only.

One Up
One right

Axay Lal
sheet XVII

Puzzle

```
# include "goto.c"
int a[ ][4 ]= {
                10, 14, 13, 11,
                 9,  7,  6,  5,
                 3,  4,  8, 15,
                 1,  2, 12,  0
              } ;
main( )
{
   clrscr( ) ;
   boxes( ) ;
   display( ) ;
   . .
}
```

(10, 20)

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

(18, 32)

gotorc(row, column).

using graphic characters

```
display( )
{
   int  i, j, r = 11, c = 21 ;
   for ( i = 0 ; i <= 3 ; i++ )
   {
      for ( j = 0 ; j <= 3 ; j++ )
      {
         gotorc ( r, c ) ;
         if ( a[ i ][ j ] ! = 0 )
            printf ( "%d",  a[ i ][ j ] ) ;
         else
            printf ( "- -" ) ;
         c = c + 3 ;
      }
      r = r + 2 ;
      c = 21 ;
   }
}
```

(10, 20)

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

(18, 32)

2 Spaces

All keys on board have ASCII & Scan codes are calculated
↑ may be same — never same
for arrow ASCII all same.

```
# include "goto.c"
int a[ ][ ]= {
                . .
              } ;
main( )
{
   int ch ;
   clrscr( ) ;
   boxes( ) ;
   display( ) ;
   gotorc ( 20, 25 ) ;
   printf ( "Use arrow keys to move nos." ) ;
   ch = getkey( ) ;
   . .
   . .
}
```

Options

```
scanf ( "%c", &ch ) ;
ch =  getch( ) ;
```

collects the ASCII values of the hit key.

Scan Codes

72
75 ← ↑ ↓ → 77
80

returns the scan code (require goto.c from lab).

```
# include "goto.c"
int a[ ][ 4 ]= {
                10, 14, 13, 11,
                 9,  7,  6,  5,
                 3,  4,  8, 15,
                 1,  2, 12,  0
              } ;
main( )
{
   ..
   ch = getkey( ) ;
   switch ( ch )
   {
      case 80 :
         t = a[ 3 ][ 3 ] ; a[ 3 ][ 3 ] = a[ 2 ][ 3 ] ;
         a[ 2 ][ 3 ] = t ;
         display( ) ;
         break ;
      ...
   }
}
```

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

down arrow key

switching 15 & 0 in array.

basically alter the value right next to zero.

```
int a[ ][ 4 ]= {
                10, 14, 13, 11,
                 9,  7,  6,  5,
                 3,  4,  8, 15,
                 1,  2, 12,  0
              } ;
main( )
{
   int r = 3, c = 3 ;
   ch = getkey( ) ;
   switch ( ch )
   {
      case 80 :
         t = a[ r ][ c ] ;  a [ r ][ c ] = a[ r - 1 ][ c ] ;
         a [ r - 1 ][ c ] = t ;
         display( ) ;
         r-- ;
         break ;
   }
}
```

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

row & column of array

to get row position of 0.

checking for legallity of arrow key movement.

```
ch = getkey( ) ;
switch ( ch )
{
   case 80 :
      if ( r ! = 0 )
      {
         t = a[ r ][ c ] ;
         a[ r ][ c ] = a[ r - 1 ][ c ] ;
         a[ r - 1 ][ c ] = t ;
         display( ) ;
         r-- ;
      }
      else
         printf ( "\a" ) ;
      break ;
}
```

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

```
int a[ ][4]= {
                10, 14, 13, 11,
                 9,  7,  6,  5,
                 3,  4,  8, 15,
                 1,  2, 12,  0
              } ;
```

```
ch = getkey( ) ;
switch ( ch )
{
   case 80 :
      : :
   case 72 :
      if ( r != 3 )
      {
         t = a[ r ][ c ] ;
         a[ r ][ c ] = a[ r + 1 ][ c ] ;  a[ r + 1 ][ c ] = t ;
         display( ) ;
         r++ ;
      }
      else
         printf ( "\a" ) ;
      break ;
   : :
}
```

10	14	13	11
9	7	6	5
3	4	8	15
1	2	12	

{ scan Code / Positional code are successive & logical & unique }

```
main( )
{
   . .
   gotorc ( 20, 25 ) ;
   printf ( "Use arrow keys... Esc to Exit" ) ;
   ch = getkey( ) ;
   switch ( ch )
   {
      case 80 :
      case 72 :
      case 75 :
      case 77 :
      case 1 :
         exit( ) ;
      default :
         printf ( "\a" ) ;
   }
}
```

```
#include "goto.c"
main( )
{
   int ch ;
   ch = getkey( ) ;
   printf ( "%d", ch ) ;
}
```

[Prog to find out scan code of any key]

scan code = 1

exit

required for every game.

```
main( )
{
   . .
   . .
   while ( 1 )
   {
      ch = getkey( ) ;
      switch ( ch )
      {
         case 80 :
         case 72 :
         case 75 :
         case 77 :
      }
      check( ) ;
   }
}
```

infinite while loop.

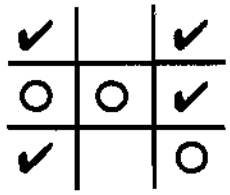
loop. to see if answer is reached.

after solution obtained Exit();

Static int counter = 0 was to output no. of times prog finished.

www.funducode.com

Tic-Tac-Toe



Center
Corner
middle

no of arrays.

3-D Arrays

```
main( )
{
   int a[ ][ 2 ][ 4 ] = {
                          {
                             3, 7, 2, 5,
                             6, 1, 4, 8
                          },
                          {
                             4, 5, 3, 2,
                             0, 0, 1, 7
                          },
                          {
                             4, 8, 5, 12,
                             6, 0, 0, 100
                          }
                       };
}
```

row
column

0
1
2

base add 400

Position in memory

```
main( )
{
   int a[ ][ 2 ][ 4 ] = {
                          ..
                       };
   printf ( "%d%d", sizeof ( a ), a[ 2 ][ 1 ][ 3 ] );
   printf ( "%d", * ( * ( * ( a + 2 ) + 1 ) + 3 ) );
   printf ( "%d%d%d%d", a, *a, **a, ***a );
}
```

		4	8	5	12
	4	5	3	2	100
3	7	2	5	7	
6	1	4	8		

2nd two D array 1st row 3rd column.

48, 100.

100

400, 400, 400, 3.

a[0][0][0]

((*(a+0)+0)+0)

Character Array.

Strings

```
main( )
{
    char name[ ] = { 'S', 'a', 'n', 'j', 'a', 'y', '\0' } ;
    int i ;
    for ( i = 0 ; i <= 5 ; i++ )
        printf ( "%c", name[ i ] ) ;
    i = 0 ;
    while ( name[ i ] ! = '\0' )
    {
        printf ( "%c", name[ i ] ) ;
        i++ ;
    }
}
```

zero → string terminator.

name[i++]

name[i]++ will give an infinite loop
Print → S, T, U, V,

Two More Ways

```
main( )
{
    char name[ ] = { 'S', 'a', 'n', 'j', 'a', 'y', '\0' } ;
    int i ;
    printf ( "%d%d", '\0', '0' ) ;
    i = 0 ;
    while ( name[ i ] ! = 0 )
        printf ( "%c", name[ i++ ] ) ;
    i = 0 ;
    while ( name[ i ] )
        printf ( "%c", name[ i++ ] ) ;
}
```

ASCII value of \0 = 0
0 = 48

while (name[0]) → ascii non-zero ie true.
while (name[1]) → S. ascii vaule non-zero ie true
:
while (name[6]) → a ascii value = 0 ∴ false.
Stop

Which Is Best?

- for (i = 0 ; i <= 5 ; i++)
- while (name[i] ! = '\0')
- while (name[i] ! = 0)
- while (name[i])
- printf ("%s", name) ;

passing base add of array.

%s = string.
bulk of characters

```
int a[10] ; char name[20] ;
display3 ( a, sizeof ( a ) / 2 - 1 )
display4 ( name ) ;
```

Why?

don't need the length of array as we know the last char is always backslash 0.

reason why integer array dont have \0.
in char ASCII values are stored all > 0 in Integers \0 ASCII = 0. Thus a zero should never be stored in the array ie 103 will store only 1 as 0 is terminator)

- Octal from decimal using recursion

modular div by 8 is req.

```
octal(int n)
{
    int r
    r = n % 8
    n = n / 8
    octal(n)
} Printf("%d",r)
```

if (n == 0) return → Printf("%d",r)

Printf("%d",r) → (after octal as the corresponding values of r are to be printed in reverse order)

**String functions defination

Axay Lal sheet-16.

Multiword Strings

```
main( )
{
   char str1[  ] = { 'S', 'a', 'n', 'j', 'a', 'y', '\0' } ;
   char str2[  ] = "Sanjay" ;
   char str3[ 15 ] ;
   printf ( "%d%d", sizeof ( str1 ), sizeof ( str2 ) ) ;
   printf ( "Enter name & surname" ) ;
   scanf ( "%s", str3 ) ;
   printf ( "%s", str3 ) ;
   printf ( "Enter name & surname" ) ;
   gets ( str3 ) ;
   printf ( "%s", str3 ) ;
   puts ( str3 ) ;
}
```

String Terminator

\0 Assumed

Rahul Sood

Rahul

Rahul Sood

Rahul Sood.

giving base add of array.

Different
3
3.0
'3'
"3"

integer
double
char
string.

→ 7 7

* Only Rahul coo → space is a terminator. /// lg tab & return (\0).

* gets → get string from keyboard

* puts → puts string on screen.

* Puts faster than printf. as it does check the access specifier.

Which Is Better?

```
main( )
{
   char str1[ ] = "Amol" ;
   char str2[ ] = "Sanjay" ;
   char str3[ ] = "Rahul" ;
   printf ( "%s%s%s", str1, str2, str3 ) ;
   puts ( str1 ) ;
   puts ( str2 ) ;
   puts ( str3 ) ;
   scanf ( "%s%s%s", str1, str2, str3 ) ;
   gets ( str1 ) ;
   gets ( str2 ) ;
   gets ( str3 ) ;
}
```

all calls to puts must be unique.

all calls to gets must be unique.

Printf / scanf (better for single word strings).

Think Differently

```
main( )
{
   char str [  ] = "Sanjay" ;
   char *p ;
   p = str ;
   while ( *p != '\0' )
   {
      printf ( "%c", *p ) ;
      p++ ;
   }
}
```

Storing base.

p p p

str[ ]

S	a	n	j	a	y	\0
401	2	3	4	5	6	7

p 401 —p++→ p 402

what actually gets passed is base add of H (P+1) add of E . . .

Output ?

```
main( )
{
    printf ( "Hello" ) ;
    printf ( 2 + 3 % 2 + "Mechanical" ) ;
}
    printf ( char *p, ... )
    {
        while ( *p != '\0' )
        {
            ..
            printf ( "\nsi = Rs. %f", si ) ;
            p++ ;
        }
    }
```

p++ ; increments value of ptr P.

each char is passed to a Char pointer which continues till *p != '\0' is found true.

(3 + "Mechanical")

here output is Manical.

in *p M's base add is passed +3 = H. then (~~*p~~)(p++) ;

0+3 → M
1+3 → A

Strlen defination

```
main( )
{
    char str1[ ] = "Nagpur" ;
    char str2[ ] = "Ahmedabad" ;
    int I1 ; int I2 ; int I3 ;
    I1 = strlen( str1 ) ;
    I2 = xstrlen ( str2 ) ;
    printf ( "%d%d", I1, I2 ) ;
    I3 = xstrlen ( "Baroda" ) ;
    printf ( "%d", I3 ) ;
}
xstrlen ( char *p )
{
    int count = 0 ;
    while ( *p != '\0' )
    {
        count ++ ;
        p++ ;
    }
    return ( count ) ;
}
```

p → 401

N	a	g	p	u	r	\0
401	2	3	4	5	6	7

p → 401

6 , 7

Strlen() → string length.

Xstrlen → user defined func doing same thing as strlen.

Strlen (doesn't calculate the '/0' value).

sizeof → includes the '\0' value.

Strcpy defination

① base add = base add.

str2 = str1 ② ///ly to a = a → this is wrong. as a = 2 here str1 is a variable

Copying Strings

```
main( )
{
    char str1[ ] = "Nagpur" ;
    char str2[ 10 ] ;
    char str3[ 20 ] ;
    str2 = str1 ;
    strcpy ( str2, str1 ) ;
    printf ( "%s", str2 ) ;
    xstrcpy ( str3, "Bombay" ) ;
    printf ( "%s", str3 ) ;
}
xstrcpy ( char *t, char *s )
{
    while ( *s != '\0' )
    {
        *t = *s ;
        t++ ;
        s++ ;
    }
    *t = '\0' ;
}
```

s → N a g p u r \0

t → N a g p u r

Nagpur.

Bombay

Strcpy → string copy. (to, from).

is same as base add of both strings are going.

xstrcpy → user defined func doing similar func as Strcpy.

Copying of values.
(let value at add = value at new add).

*t = *s.

without \0 the string is incomplete as no terminator present.

terminator.

imp – working of while

More Ways...

```
xstrcpy ( char *t, char *s )
{
    while ( *s != '\0' )
    {
        *t = *s ;
        t++ ;
        s++ ;
    }
    *t = '\0' ;
}
```

```
xstrcpy ( char *t, char *s )
{
    while ( *s )
        *t++ = *s++ ;
    *t = '\0' ;
}
```

```
xstrcpy ( char *t, char *s )
{
    while ( *t++ = *s++ )
        ;
}
```

s ⇩ N a g p u r \0 s ⇩

t ⇩ N a g p u r \0 t ⇩

while (i = 2)

```
*t = *s
while ( *t )
t++
s++
```

→ as always > 0 *s = \0 = 0
∴ while(0) – false.

Run – I

→ N is copied
→ while (no zero)
{
t++ } – shifting positions of t & s to next memory location.
s++ }

① ~~Equality~~/Assignment Uniary operator.
② Test
③ increment — binary operator

last Run
\0 copied while(zero) then out of loop.

Concatenation

```
main( )
{
    char str1[ ] = "Nagpur" ;
    char str2[ ] = "Bombay" ;
    xstrcat ( str1, str2 ) ;
    printf ( "%s", str1 ) ;
}
xstrcat ( char *t, char *s )
{
    while ( *t )
        t++ ;
    while ( *s )
        *t++ = *s++ ;
    *t = '\0' ;
}
```

20

→ Nagpur Bombay.

t ⇩ N a g p u r \0 t ⇩

s ⇩ B o m b a y \0

strcat(to, from).

① Passing 2 base add
② while (no zero) increment
→ getting the position of \0.
③ while (no zero) copy 2nd string to first.
→ over writing the \0.
④ putting \0 back to 1st string.

*** this can only work if target string should be big enough to encorporate it

** * strcat → Combine *

Shorter Version

```
xstrcat ( char *t, char *s )
{
    while ( *t )
        t++ ;
    while ( *s )
        *t++ = *s++ ;
    *t = '\0' ;
}
```

t = t + strlen (t) ;

strcpy (t, s) ;

t ⇩ N a g p u r \0 t ⇩

s ⇩ B o m b a y \0

→ making it point to \0. t + 6 (7).
sizeof(t) = 2 byte. as ptr. base add + 6 = 7

t is \0 adding s till \0.

→ also gets copied.

strCat → starts from \0 only
strcpy → starts from any given position.

Concatenation

```
main( )
{
    char  str1[   ] = "Kanpur" ;
    char  str2[   ] = "Raipur" ;
    char  str3[ 30 ] ;
    strcpy ( str3, str1 ) ;
    strcat ( str3, str2 ) ;
    printf ( "%s", str3 ) ;
}
```

→ Kanpur Raipur.

Using only strcpy( ) → strcpy (str3, str1) ; strcpy (str3 + strlen (str3), str2) ;

Using only strcat( ) → str3[0] = '\0' ; xstrcat (str3, str1) ; xstrcat (str3, str2) ;

till \0 + str2

as strcat starts from \0.

strupr → string to Upper Case

Convert To Uppercase

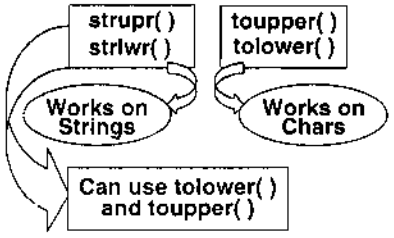
```
main( )
{
    char  str[ ] = "Nagpur" ;
    xstrupr ( str ) ;
    printf ( "%s", str ) ;
}
xstrupr ( char  *p )
{
    while ( *p )
    {
        if ( *p >= 'a'  &&  *p <= 'z' )
            *p = *p - 32 ;
        p++ ;
    }
}
```

NAGPUR.

① base add passed
② while true.
③ check if already caps.
*p = *p - 32 converting to caps.
④ p++ → next ~~ve~~ location.

strlwr() → lower Case.

What's The Difference?



String a collection of char.
thus multiple touppers & tolowers acting on a string.

Strcmp — String Compare

Comparing Strings

```
main( )
{
    char str1[ ] = "Bombay" ;
    char str2[ ] = "Nagpur" ;
    char str3[ ] = "Bombaywala" ;
    char str4[ ] = "Bombay" ;
    int i, j, k ;
    strcmp ( str1, str2 ) ;
    xstrcmp ( str1, str3 ) ;
    xstrcmp ( str1, str4 ) ;
    printf ( "%d%d%d", i, j, k ) ;
}
```

strcmp(this with the other).

Comparing 2 strings

if equal return 0.
if not equal return (non zero value)

Comparing Strings

```
main( )
{
    i = xstrcmp ( str1, str3 ) ;
}
xstrcmp ( char *t, char *s )
{
    while ( *t == *s )
    {
        if ( *t == '\0' )
            break ;
        t++ ;
        s++ ;
    }
    return ( *t - *s ) ;
}
```

t ⇩ N a g p u r \0
s ⇩ B o m b a y \0

t ⇩ B o m b a y \0
s ⇩ B o m b a y w a l a \0

t ⇩ B o m b a y \0
s ⇩ B o m b a y \0

→ passing base add of 2 string
→ checking whether *t = *s.
goon comparing till

without [if] return (ascii of \0 − ascii of w) ⇒ −109

without [if] return (ascii of N − ... B) ⇒ +ve value.

either t or s.

Useful in sorting names in some order. if +ve the second string should come first
if −ve correct order (if ascending).

Function declaration of Puts()

Outputting Strings

```
main( )
{
    char str1[ ] = "Nagpur" ;
    printf ( "%s", str1 ) ;
    xputs ( str1 ) ;
}
xputs ( char *p )
{
    while ( *p != '\0' )
    {
        printf ( "%c", *p )
        p++ ;
    }
}
```

p ⇩ N a g p u r \0

putch (*p) ;

① sending base.
② check if if not = \0
~~put~~ print value of char.

better to use putch() as (Put character) it put/prints only a character on the Screen

/// declaration of gets.

Putch ⟷ getch.

i= ignore case

Standard Library Functions

strlen
strcpy
strcat
strupr
strlwr
toupper
tolower
strcmp
puts
gets

strncpy (str1, str2, 5) ; → first 5 char of str2 to str1.
strncat (str1, str2, 6) ; → cat first 6 of str2 to str1.
strncmp (str1, str2, 6) ; → compare " " " " " " " " " " "
strncmpi (str1, str2, 4) ; → " " " " i→ ignore case.
strnicmp (str1, str2, 4) ; → same as strncmpi
strchr ("Hello", 'e') ; → searches for a character within a string. returns add of char in string. ∴ collection should be done in char*p
strstr ("I am a boy", "am") ; → Scans a string within a string. address of a in am in string is return n.
strset → Sets all char of string to some char
strnset → Sets only first n char to some specified char.
: :

*** Const

```
char *p = "Hello" ;
p = p + 1 ;   ✓
p = "Bye" ;   ✓
*p = 'M' ;    ✓
```

```
char p[ ] = "Hello" ;
p = p + 1 ;   ✗
p = "Bye" ;   ✗
*p = 'M' ;    ✓
```

both L & r are variables
as P is pointing to base add.

```
const char *p = "Hello" ;
p = p + 1 ;   ✓
p = "Bye" ;   ✓
*p = 'M' ;    ✗
```

(const string with a non const ptr)

```
char * const p = "Hello" ;
p = p + 1 ;   ✗
p = "Bye" ;   ✗
*p = 'M' ;    ✓
```

const keyword [const ptr to a non const string.]

```
char const *p = "Hello" ;
p = p + 1 ;   ✓
p = "Bye" ;   ✓
*p = 'M' ;    ✗
```

(const string non const ptr)

```
const char * const p = "Hello" ;
p = p + 1 ;   ✗
p = "Bye" ;   ✗
*p = 'M' ;    ✗
```

const ptr to a const string

Name of an array acts as a constant pointer to zeroth element of the array

Utility of const

```
main( )
{
    char str1[ ] = " Hello" ;
    char str2[ 10 ] ;
    xstrcpy ( str2, str1 ) ;
    printf ( "%s", str2 ) ;
}
xstrcpy ( char *t, char *s )
{
    *s = 'A' ;
    while ( *s != '\0' )
        *t++ = *s++ ;
    *t = '\0' ;
}
```

const char *s ; → then *s = 'A' won't be allowed. (non const ptr to a const string).

char *s → storing base add of string ∴ add of H.

Axay Lal → Sheet – XVII

Handling Several Strings

```
main( )
{
    char str1[ ] = "Sanjay" ;
    char str2[ ] = "Amol" ;
    char str3[ ] = "Sivaramakrishnan" ;
    char str4[ ] = "Sameer" ;
    char str5[ ] = "Rahul" ;
    : :
}
```

Array of Strings / 2-D Array

```
main( )
{
    char n[   ][ 20 ] = {
                          "Sanjay",
                          "Amol",
                          "Sivaramakrishnan",
                          "Sameer",
                          "Rahul"
                        } ;
    printf ( "%d", sizeof ( n ) ) ;
    printf ( "%c%c", n[ 0 ][ 1 ], n[ 1 ][ 0 ] ) ;
}
```

→ to be on safe side.

5×20 = 100 × 1 byte each = 100 bytes

aA

Sanjay\0	Amol\0	Sivar...\0	Sameer\0	Rahul\0
401	421	441	461	481

```
main( )
{
    char n[  ][ 20 ] = {
                          : :
                        } ;
    int i, j ;
    for ( i = 0 ; i <= 4 ; i++ )
    {
        for ( j = 0 ; j <= 19 ; j++ )
        {
            if ( n[ i ][ j ] == '\0' )
                break ;
            printf ( "%c", n[ i ][ j ] ) ;
        }
    }
}
```

→ prevents printing of garbage as Sanjay\0 ..... till 20 (garbage).

Exchanging Names :--

Sanjay\0	Amol\0	Sivar...\0	Sameer\0	Rahul\0
401	421	441	461	481

```
main( )
{
   char n[ ][ 20 ] = { "Sanjay", "Amol", ... } ;
   int i, j ; char t ;
   for ( j = 0 ; j <= 19 ; j++ )
   {
      t = n[ 1 ][ j ] ;
      n[ 1 ][ j ] = n[ 2 ][ j ] ;
      n[ 2 ][ j ] = t ;
   }
   for ( i = 0 ; i <= 4 ; i++ )
      printf ( "%s", &n[ i ][ 0 ] ) ;
}
```

n + i

2D array for string wastes alot of space & tedious to deal with 2D array

string at n[0][0] → base add of Sanjay

n[1][0] base add of Sivar...

n+0 → 401
n+20 → 421
n+40

as n is a ptr & points to first element here Sanjay.

actually (n+0) (n+1) (n+2) (n+3)

*n + i = 401 402

Disadvantages

Sanjay\0	Amol\0	Sivar...\0	Sameer\0	Rahul\0
401	421	441	461	481

- Wastage
- Inefficient Processing

Wastage
13
15
3
13
14
58 Bytes

not adjacent locations as now they are just strings not ptrs.

Imp

Sanjay\0	Amol\0	Sivar...\0	Sameer\0	Rahul\0
400	300	600	500	100

```
main( )
{
   char *n[ ] = {
                  "Sanjay",
                  "Amol",
                  "Sivaramakrishnan",
                  "Sameer",
                  "Rahul"
                 } ;
   int i ; char *t ;
   for ( i = 0 ; i <= 4 ; i++ )
         printf ( "%s", n[ i ] ) ;
   t = n[ 1 ] ; n[ 1 ] = n[ 2 ] ;
   n[ 2 ] = t ;
   for ( i = 0 ; i <= 4 ; i++ )
         printf ( "%s", n[ i ] ) ;
}
```

n[]

400	300	600	500	100
202	204	206	208	210

n[0] add of Sanjay
n[1] add of Amol

here names aren't the array's priority where as the ptrs are thus

→ printing all names

→ swaping Amol & Sivar

no loss of space in 2D 58 bytes lost. here only 10 bytes lost. 48% saved.

Arranging In Ascending Order (strings)

```
main( )
{
  char *n[ ] = {
                 "Sanjay", "Amol",
                 "Sivaramkrishnan", ..
               } ;
  int i, j ; char *t ;
  for ( i = 0 ; i <= 3 ; i++ )
  {
     for ( j = i + 1 ; j <= 4 ; j++ )
     {
        if ( strcmp ( n[ i ], n[ j ] ) > 0 )
        {
           t = n[ i ] ; n[ i ] = n[ j ] ; n[ j ] = t ;
        }
     }
  }
  for ( i = 0 ; i <= 4 ; i++ )
       printf ( "%s", n[ i ] ) ;
}
```

Selection Sort
i j
0 - 1 0 - 2 0 - 3 0 - 4
1 - 2 1 - 3 1 - 4
2 - 3 2 - 4
3 - 4

switching base addresses.

Project - 4

Calendar

```
main( )
{
   int m, y ; int leapdays ;
   long int normaldays ;
   long int totaldays ;
   printf ( "Enter month and year" ) ;
   scanf ( "%d%d", &m, &y ) ;
                                  8 1999
   normaldays = ( y - 1 ) * 365 L ;
   leapdays = ( y - 1 ) / 4 - ( y - 1 ) / 100
              + ( y - 1 ) / 400 ;
   totaldays = normaldays + leapdays ;
   .. → 1/1/1 → 31/12/1998
}
```

1/8/1999 - ?
1/1/1 - Mon
1/1/1999 - ?
↓
1/1/1 to 31/12/1998
↓
x % 7

1/8/1999 - ?
1/1/1 to 31/7/1999
↓
1/1/1 to 31/12/1998
+1/1/1999 to 31/7/1999 -
x % 7

→ x % 7

R=0 Mon
=1 Tues

→ each year 365 days
→ calculating leapyears → leap days

...Calendar

```
main( )
{
   int days[ ] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;
   int i, s ; int firstday ;
   ::
   totaldays = normaldays +
               leapdays ;
   s = 0 ;
   for ( i = 0 ; i <= m - 2 ; i++ )
        s = s + days[ i ] ;
   totaldays += s ;
   firstday = totaldays % 7 ;
   ..
}
```

1/1/1 to 31/12/1998
totaldays

1/1/1999 to 31/7/1999
31 + 28 + 31 + 30 + 31 + 30 + 31

m = Aug = 8
now in array
Aug = 7 ∴ m-1
we want July = 6 m-2

S = no of days btw 1/1/1999 & 31/7/1999

if R=0 next day mon.

error as this is only for non leap year thus before sum check whether it is a leap year or not

...Calendar

```
main( )
{
                0 1 2 3 4 5 6 7 8 9 10 11
  int days[ ] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;
  ::
  totaldays = normaldays + leapdays ;
  if ( ( y % 400 == 0 ) || ( y % 100 != 0 && y % 4 == 0 ) )
      days[ 1 ] = 29 ;
  s = 0 ;
  for ( i = 0 ; i <= m - 2 ; i++ )
      s = s + days[ i ] ;
  totaldays += s ;
  firstday = totaldays % 7 ;
  ::
}
```

→ check for leap year. (given year).

→ Putting Feb = 29 days

Display

specifying Month & Year.

Screen

```
        August 1999
   20    26    32    38    44    50    56
   |     |     |     |     |     |     |
10 — Mon__Tue__Wed__Thu__Fri__Sat__Sun
```

size of (month) = 24 (2×12).

...Calendar

```
main( )
{
    char *months[ ] = { "January", "February", ... } ;
    ::
    totaldays += s ;
    firstday = totaldays % 7 ;
    col = 20 + firstday * 6 ;
    clrscr( ) ;
    gotorc ( 8, 35 ) ;
    printf ( "%s %d", months[ m - 1 ], y ) ;
    gotorc ( 10, 20 ) ;
    printf ( "Mon Tue Wed Thu Fri Sat Sun" ) ;
}
```

```
      20    26    32
      |     |     |
10 — Mon  Tue  Wed   ...
```

→ if 0 go below Monday
1 go below Tuesday.

specific formula.

> Case 0: col 20 } basic funda.
Case 1: col 26.

→ make an array (~~* to array~~) (array of *).

→ Case 0 - Jan
1 - Feb
2 - March } basic fund.

Printing of no. 1 → n

...Calendar

```
main( )
{
    int days[ ] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, ... } ;
    /* print month year */
    /* print days */
    row = 12 ;
    for ( i = 1 ; i <=days[ m - 1 ] ; i++ )
    {
        gotorc ( row, col ) ;
        printf ( "%d", i ) ;
        col = col + 6 ;
        if ( col > 56 )
        {
            row++ ; col = 20 ;
        }
    }
}
```

```
      20     26     32
       |      |      |
10 — Mon   Tue   Wed     ...
```

col = col + 6 → next day.

col > 56 → to prevent it from exceed **56**

Falls

① → any optional date
any optional year.

...Calendar

```
main( )
{
    ..
    scanf ( "%d%d", &m, &y ) ;
    while ( 1 )
    {
        normaldays = ( y - 1 ) * 365L ;
        calendar
        gotorc ( 20, 35 ) ;
        printf ( "Rt-Next mth..." ) ;
        ch = getkey( ) ;
        switch ( ch )
        {
            case 77 :
                m++ ;
                if ( m > 12 )
                {
                    y++ ; m = 1 ;
                }
        }
    }
}
```

Next year Y++
Prev. mth ← m-- → Next mth
Prev. year Y--

m++ → Right arrow increase month.

if (m > 12) → check on value of month.

Check on no of days in Feb.

...Calendar

```
main( )
{
    int days[ ] = { 31, 28, 31, 30, 31, 30,
                    31, 31, 30, 31, 30, 31 } ;
    scanf ( "%d%d", &m, &y ) ;
    while ( 1 )
    {
        normaldays =
        leapdays =
        totaldays =
        if ( y % 400 = = 0 ) || ( y % 100 ! = 0  &&  y % 4 = = 0 ) )
            days[ 1 ] = 29 ;
        else
            days[ 1 ] = 28 ;
        ..
    }
}
```

9, 1998 - 30
9, 2000 - 30
9, 1700 - 30
9, 1752 - 16

→ exception
So separate if condition for 9, 1752.

as 1 2 3 (gone) → 17

14 less than normal day calculation is wrong but since 14 % 7 is 0
So adding or sub 14 is no use.

Precompiler → doesn't check for closing braces
undefined variables etc.

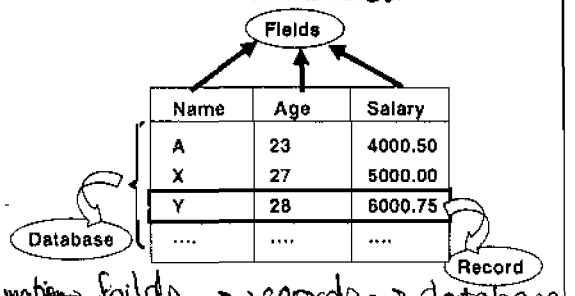
```
main()
{                                   int color=150;
    char str[]="Nagpur";            int c, p
    char *p; char for *s=0x...;
    p=str; c=20;                    Nagpur10
    while (*p)
    {
        v=s+10*150+c*2;
        *v=*p;
        *(v+1)=color;
        :
        p++; ...
    }
```

Data Structures

Axay Lal
The Ax. effect

(18)

Terminology



information → feilds → records → database

Handling Data

```
main( )
{
    char n[ ] = { 'A', 'X', 'Y', '\0' } ;
    int a[ ] = { 23, 27, 28 } ;
    float s[ ] = { 4000.50, 5000.00, 6000.75 } ;
    int i ;
    for ( i = 0 ; i <= 2 ; i++ )
        printf ( "%c %d %f", n[ i ], a[ i ], s[ i ] ) ;
}
```

A 23 4000.50

output.

by lookin at source we cannot find an corelation btw A 23 & 4000.50.

* Declaration never reserves space. Only defination reserves space.

int a — 2bits
extern int — no space.

Structures

```
main( )
{
  struct employee
  {
    char n ;
    int a ;
    float s ;
  } ;
  struct employee e1 = { 'A', 23, 4000.50 } ;
  struct employee e2 = { 'X', 27, 5000.00 } ;
  struct employee e3 = { 'Y', 28, 6000.75 } ;
  printf ( "%c %d %f", e1.n, e1.a, e1.s ) ;
  printf ( "%c %d %f", e2.n, e2.a, e2.s ) ;
  printf ( "%c %d %f", e3.n, e3.a, e3.s ) ;
}
```

. , → } structure operators

→ no memory saved.

→ bytes will be reserved here 1+2+4.

→ (. & → structure operators)

Structure → dis-similar entities are used.
natural relationship is still intact.

Array of Structures

```
main( )
{
    struct employee
    {
        char  n ;
        int  a ;
        float  s ;
    } ;
    struct  employee  e[ ]= {
                              { 'A', 23, 4000.50 },
                              { 'X', 27, 5000.00 } ,
                              { 'Y', 28, 6000.75 }
                            } ;
    int  i ;
    for ( i = 0 ; i <= 2 ; i++ )
        printf ( "%c %d %f", e[ i ].n, e[ i ].a, e[ i ].s ) ;
}
```

point less.

Terminology

Keyword ← struct employee → Structure name/tag

```
{
    char  n ;
    int  a ;
    float  s ;
} ;
struct  employee  e1, e2, e[ 10 ] ;
```

Structure elements/ members

size = 70 bytes

Structure variables

Array of structures

each size 7 bytes.

Conclusion

- A structure is usually a collection of dissimilar elements.
- *** Structure elements are always stored in adjacent memory locations.

struct employee e[3] = { .. } ;

A	23	400.50	X	27	500.00	Y	28	600.75

401 403 407 408 415

→ (i)(a)+(h)

isn't it ~~more~~ faster if calling from mem is done when all similar together / mem of struct? / fc(a) int a
data error/ segment / general failure, disaa/ -secondick.

** incrementing a ptr takes you to the next place OF ITS TYPE

Array of Structures

```
struct employee e[ ] = { ... } ;
char *p ;
struct employee *q ;
struct employee (*r )[3];
```

Ptr. to structure / structure ptr.

Ptr. to array of structures

→ r is a ptr to an array of 3 structures.

```
struct employee e[ 3 ] = { .. } ;
```

A	23	400.50	X	27	500.00	Y	28	600.75

401 408 415

```
p = e ; q = e ; r = e ;
p++ ; q++ ; r++ ;
printf ( "%u", p ) ; 402
printf ( "%u", q ) ; 408
printf ( "%u", r ) ; 422
```

outside array. next array of 3 structures.

```
struct employee *p[3] ;
```

→ array of 3 structure ptr or array of ptr.

↳ 6 bytes (2×3).

Array of Ptrs to structures

Always ptr of p = 2.

(Imp)

Declarations & Definitions

```
struct employee
{
   char n[20] ;
   int a ;
   float s ;
} ;
struct employee e = { "Rahul", 25, 4000.50 } ;
```

Compulsory

/// int i ; i = 10 ;

better

Optional

OR

```
struct employee
{
   char n[20] ; int a ; float s ;
} e = { "Rahul", 25, 4000.50 }, x, y ;
```

/// int i = 10 .

Copying

```
main( )
{
   struct emp
   {
      char n[20] ;
      int a ;
      float s ;
   } ;
   struct emp e1 = { "Rahul", 23, 4000.50 } ;
   struct emp e2, e3 ;
   e2.n = e1.n ;
   e2.a = e1.a ;
   e2.s = e1.s ;
   e3 = e1 ;
   printf ( "%s %d %f", e3.n, e3.a, e3.s ) ;
}
```

piecemeal copying

strcpy (e2.n, e1.n) ;

copying at one shot

whenever %s → base add.

→ is wrong as n will give base add. so base add = base add which is illegal.

→ all elements of e1 now in e3.

Copying Arrays

```
int a[10] = { 3, 6, 5, ... } ;
int b[10] ;
for ( i = 0 ; i <= 9 ; i++ )
      b[ i ] = a[ i ] ;
```

OR

```
struct z
{
   int arr[10] ;
} ;
struct z  a = { 3, 6, 5, ... } ;
struct z  b ;
b = a ;
```

better.

what happens.

Nested Structures

```
main( )
{
  struct address
  {
     char city[20] ;
     long  int  pin;
  } ;
  struct emp
  {
     char  n[20] ; int  age ;
     struct  address  a ; float  s ;
  } ;
  struct emp e={ "Rahul", 23,"Ngp", 44010, 4000.50 };
  printf ( "%s %d %s %ld %f", e.n, e.age, e.city, e.pin,
           e.s ) ;
}
```

f - int

printf ("%d", a.b.c.d.e.f) ;

a.city

e.a.city, e.a.pin

order is imp always 1-1 corespondence btw defi & declarations.

Calling one structure from another ∴ nested structures are Okay.

Passing Structure Elements

```
main( )
{
   struct  book
   {
      char  n[20] ; int  nop ; float  pr ;
   } ;
   struct  book  b = { "Basic", 425, 135.00 } ;
   display ( b.n, b.nop, b.pr ) ;
   show ( b.n, &b.nop, &b.pr ) ;
}
display ( char *n, int  pg, float  p )
{
   printf ( "%s %d %f", n, pg, p ) ;
}
show ( char *n, int *pg, float *p )
{
   printf ( "%s %d %f", n, *pg, *p ) ;
}
```

display ← user defined function. Not a lib func.

holding string of char. (base add).

Global.

Passing Structures

```
main( )
{
    struct book
    {
        char n[20] ; int nop ; float pr ;
    } ;
    struct book b = { "Basic", 425, 135.00 } ;
    display1 ( b ) ; show1 ( &b ) ;
}
display1 ( struct book bb )
{
    printf ( "%s %d %f", bb.n, bb.nop, bb.pr ) ;
}
show1 ( struct book *bb )
{
    printf ( "%s %d %f", (* bb ).n, (* bb ).nop, (* bb ).pr ) ;
    printf ( "%s %d %f", bb -> n, bb -> nop, bb -> pr )
}
```

→ if not Global struct then struct becom local to main. error as display1 wont know what struct book is

→ all elements of b are now in bb. /// to b=bb.

→ firstly *b reaches struct then .n, nop.

the two are the same.

with → ptr to struct.
with . structure variable

Global

Complex Nos. — Usage of structures.

```
main( )
{
    struct com
    {
        float r, i ;
    } ;
    struct com a = { 2.5, 1.3 } ;
    struct com b = { 1.2, 1.7 } ;
    struct com c ;
    struct com add ( struct com, struct com ) ;
    c = add ( a, b ) ;
    printf ( "%f %f", c.r, c.i ) ;
}
struct com add ( struct com x, struct com y )
{
    struct com z ;
    z.r = x.r + y.r ;
    z.i = x.i + y.i ;
    return z ;
}
```

→ ans returned value not int.

→ return type similar to struct com.

→ as calling values from com.

Similar prog.

Complex prog.

Complex Nos.

```
main( )
{
    float a[ ] = { 2.5, 1.3 } ;
    float b[ ] = { 1.2, 1.7 } ;
    float *c ;
    float * add ( float *, float * ) ;
    c = add ( a, b ) ;
    printf ( "%f %f", *c, * ( c + 1 ) ) ;
}
float * add ( float *x, float *y )
{
    static float z[2] ;
    z[ 0 ] = *x + *y ;
    z[ 1 ] = *x + *y ;
    return z ;
}
```

→ cos we are returing base add.

next

first value

next float.

x++ ; y++;

second value.

~~returing base add~~ ∴ declaration required.

→ cos even after ~~ans~~ func is over arrays should survive

if array is being returned → use structure where we will just place a call.

Applications of Structures

Solid Uses. – Imp Subject

- Database Management
- Positioning Cursors
- Receiving ascii and scan codes
- Displaying characters
- Printing on printer
- Mouse Programming
- Graphics Programming
- All Disk Operations
- etc.

C:\> Dir .

Array. idiot 12/06 12/01/2. . . .

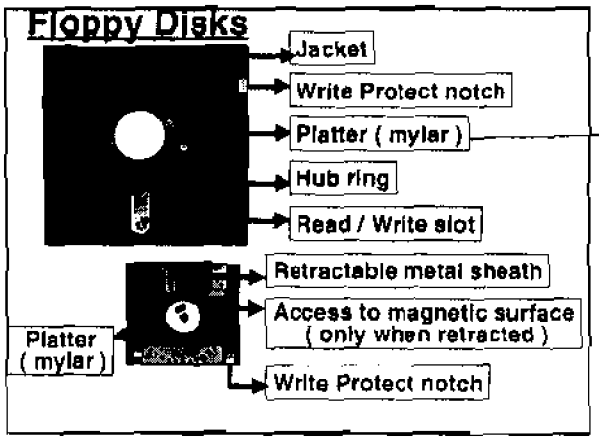
↑ is an array of structure

as

array	char	int	float

used structure.

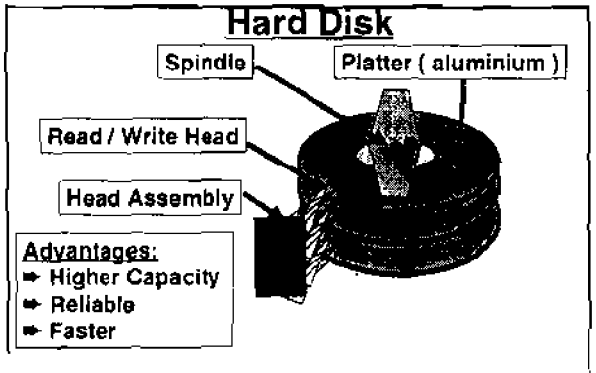
Floppy Disks



→ name of plastic mylar ///ly to tape coated with magnetic oxide

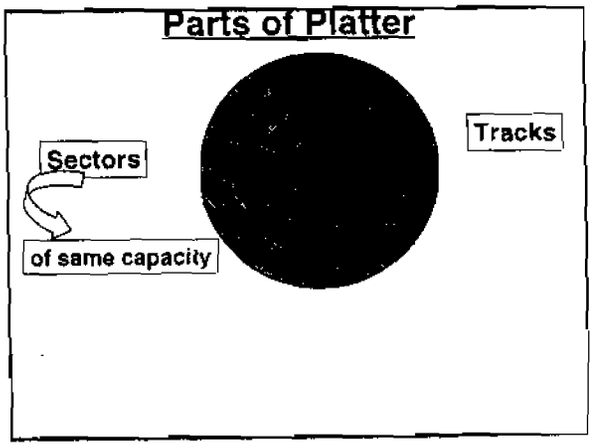
360 rpm
revolution speed of floppy.

Hard Disk



revolution speed of Hdd
7200 rpm

slower floppy → any faster speeds cause dust to accumilate.



formating leads to formation of tracks which have shit loads of sectors of same capacity = 512b

Specs Of Floppy Disks

Type	Sides	Trk/side	Sec/Trk	Dia.	TPI	Name
360 KB	2	40	9	5 1/4"	48	DSDD-D9
1.2 MB	2	80	15	5 1/4"	96	HD-QD15
1.44 MB	2	80	18	3 1/2"	135	2HD-QD18

Bytes / Sector for any disk = 512

Capacity of D9 disk = 2 * 40 * 9 * 512 / 1024

360 KB

in bytes

in kilobytes

Diameter

Tracks per inch.

double sided double disk — D9b

high density — Quad density disk D15(track)

2 high density — 2 Quad density Disk.

trade names

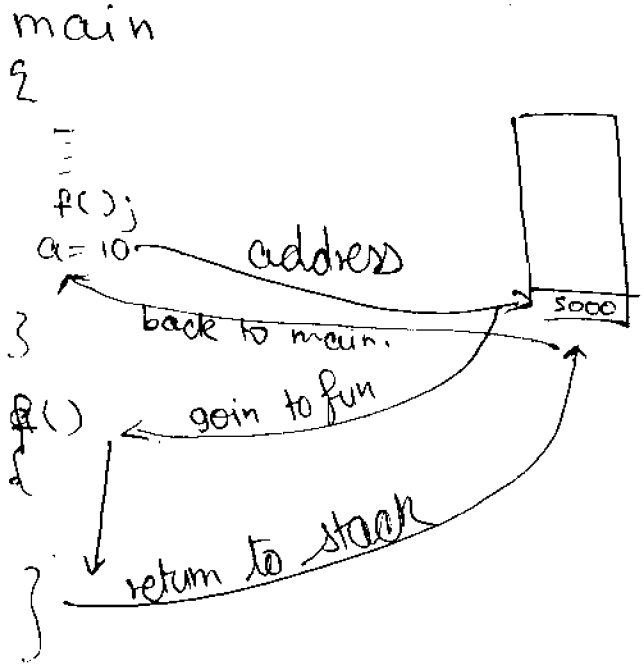
Configs

format? media descriptors; why in hexa; ~~Sys id is a string will it end in 00~~ ~~how do u read any sector~~? what goes where? when does a virus hit hdd? how does it spread.

Processor	— Intel 334-433.	(AMD-K6 3D now 350MHz)
Mother board	— SiS / intel	(Red fox)
RAM	— Siemens / Hyndi — (64)	(64)
Hdd	— Seagate — 6.3	(20)
Keyboard	— TVS Gold 104 Keys.	(104 keys — unknown)
monitor	— LG / Samsung SVGA color.	(unknown — Energy Ster Compliant)
mouse	— logitech / microsoft.	(logitech — scroller)
CD Rom	— Creative / Samsung.	(Samsung 40x).

Stack overflow

before transfering control to another func the add of the next ~~line is~~ instruction is stored.



limited storage space (stores add).

if exceeded comp goes haywire or hangs

— this is stack overflow

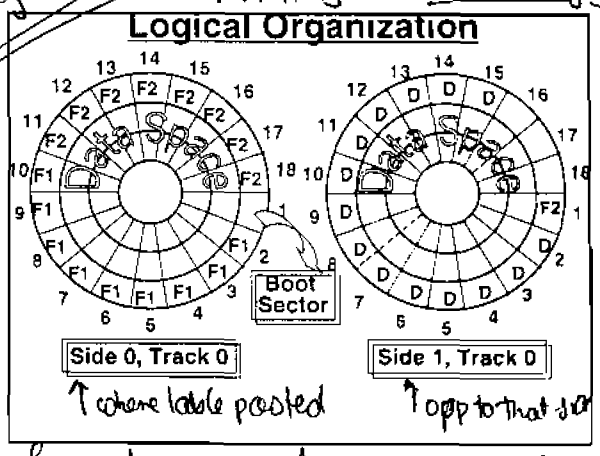
```
main()
{ add();
}
add()
{ add(); }
```

if recursion not used judiciously — stack overflow

— Infinite loop ∴ infinite times add stored ∴ overflow.

Pam Good

**** 1.44MB Disk Management [Lucky Lal sheet - 19]



↑ where lable posted ↑ opp to that side

→ for diff disk diff space occupied by the diff logical parts.

0 - 79 tracks (80)

Side (no) track (no) sector (no)

Always start circle.

1) Side 0 track 0 sector 1 — Boot sec

2) F_1 & F_2 is file Allocation table (FAT

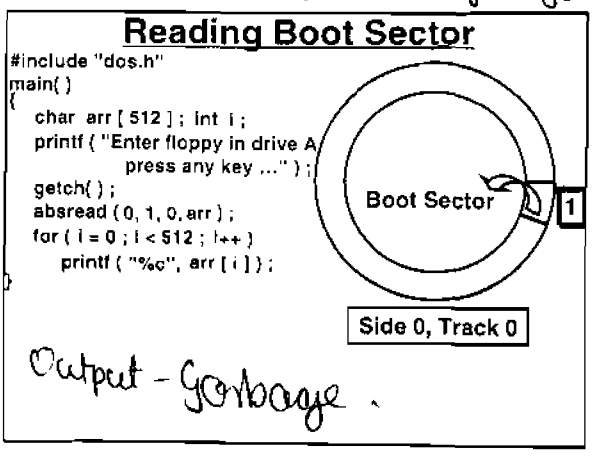
3) D — directory sector

4) rest - data structure.

format - creats tracks & sectors

2) FAT → gives location of files on disk. F_1 & F_2 2 copies as F_2 is F_1's back

3) Dir gives all info. This info goes to Directory sect. additional info.



Output - Garbage.

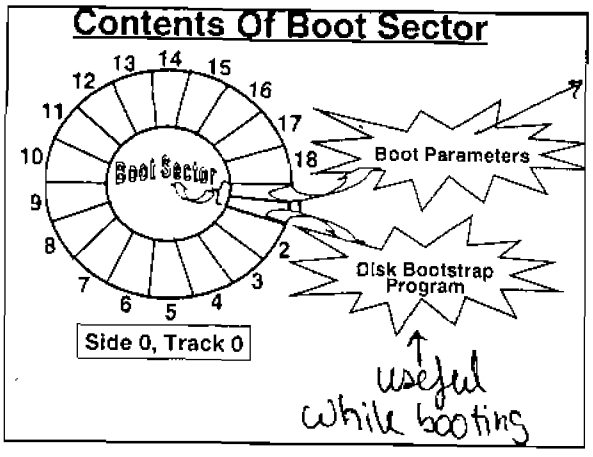
absread = absolute read. (explanation ahead)

~~0 — head side~~

~~1 — sector~~

~~0 — side track~~

arr — name of variabl. 512 as each sector is 512 bytes.



→ info abt the disk no. of sec no. of sids.

useful while booting

Boot Parameters

Description	No. of bytes	360 Kb	1.2 Mb	1.44 Mb
Jump Instruction	3	EB3490		
System ID	8	MSDOS5.0		
No. of bytes/sector	2	512	512	512
No. of sectors/cluster	1	2	1	1
No. of sectors in reserved area	2	1	1	1
No. of copies of FAT	1	2	2	2
Max. no. of root dir. entries	2	112	224	224
Total no. of sectors	2	720	2400	2880
Media descriptor	1	FD	F9	F0
No. of sectors/FAT	2	2	7	9
No. of sectors/track	2	9	15	18
No. of sides	2	2	2	2
No. of hidden sectors	2	0	0	0

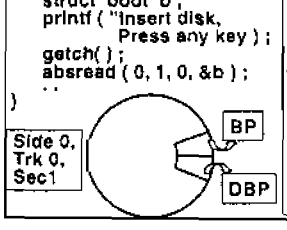
① hexa decimal digit → 24 binary digits = 3 bytes.
② under which O/S did it get formatted [got ntn with functionality of prog].
③ opp of sectors
⑥ Max files are 224 bytes.
⑦ 2 sides * 80 tracks * 2*18*18
⑧ hexa decimal for HDD F8.

Even for an hdd order of Parameters are the same.

wasnt workin previously as we created an array but we are storing diff entries in it of diff data types.

Reading Boot Sector

```
main( )
{
    struct boot
    {
        char jump[ 3 ] ;
        char sysid[ 8 ] ;
        int bps ;
        char spc ;
        ..
        char rest[ 482 ] ;
    } ;
    struct boot b ;
    printf ( "Insert disk,
             Press any key ) ;
    getch( ) ;
    absread ( 0, 1, 0, &b ) ;
    ..
}
```



Description	No. of bytes
Jump Instruction	3
System ID	8
Bytes/sector	2
Sectors/cluster	1
Sectors in reserved area	2
Copies of FAT	1
Max. root dir. entries	2
Total no. of sectors	2
Media descriptor	1
Sectors/FAT	2
Sectors/track	2
Sides	2
Hidden sectors	2

BP – 30 bytes } 512 bytes.
DBP – 482 bytes }

sizeof(b) = 30 bytes.

&b as add of container (place where to put ans) is required.

now we are reading 512 byte storing 30 bytes.

now absread can't read 30 bytes but only 512 (min)
→ this makes 'b' 512 bytes.

...Cont

```
# include "dos.h"
main( )
{
    struct boot
    {
        char jump[ 3 ] ;
        char sysid[ 8 ] ;
        int bps ;
        char spc ;
        ..
    } ;
    struct boot b ; int i ;
    absread ( 0, 1, 0, &b ) ;
    for ( i = 0 ; i <= 2 ; i++ )
        printf ( "%X", b.jump[ i ] ) ;
    for ( i = 0 ; i <= 7 ; i++ )
        printf ( "%c", b.sysid[ i ] ) ;
    printf ( "Bytes/sector = %d", b.bps ) ;
    printf ( "Sectors/cluster = %d", b.spc ) ;
}
```

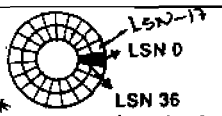
→ convert hexadecimal → decimal.
→ system id.

not %c as ultimately we are printing a no.

Actually → drive no. → which drive HDD or a:
→ no. of sectors to be read
→ beginning sector
buffer → some place in memory.

In General

absread (drive no., no. of sectors to read,
sector from where reading should begin, buffer) ;
absread (0, 1, 0, &b) ;

ROM - BIOS	DOS
	
*** Every sector referred using side, track, sector	*** Every sector referred using LSN
Drive A - 0 Drive B - 1 Drive C - 128 Drive D - 129 : : :	Drive A - 0 Drive B - 1 Drive C - 2 Drive D - 3 : : :

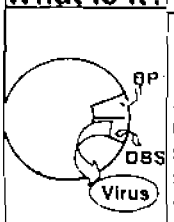
Why 1 as 1st sector is 0?
cos 2 diff methods.

LSN - logical sector no.
= after LSN 17 next side till LSN-35
then next side

<Bios.h> ← other header.

so when include Dos.h we use dos method
if wanna read 2 sectors
(0, 2, 36, &b) → 36 & 37.
↑ should be 1024 bytes.

What Is It?



1.44 Mb Disk

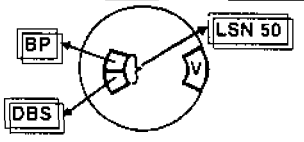
Description	Typical Values	Obtained Values
Jump Instruction	EB3490	102
System ID	IBM 3.3	....
Bytes/sector	512	20480
Sectors/cluster	1	0
Sectors in reserved area	1	1
Copies of FAT	2	-24
Max. root dir. entries	224	0
Total no. of sectors	2880	2048
Media descriptor	F0	7
Sectors/FAT	9	-2
Sectors/track	18	0
Sides	2	0
Hidden sectors	0	8

95% virus in boot.

if new values
|| — Virus ||

So all boot sectors viruses sit on the boot sector.
Why in boot sector? Cos it is called everytimes so best chance to become active.

Anti-Viral




```
# Include "dos.h"
main( )
{
    char  a[ 512 ] ;
    printf ( "Insert infected disk, Press any key"  ) ;
    getch( ) ;
    absread ( 0, 1, 50, a ) ;
    abswrite ( 0, 1, 0, a ) ;
}
```

→ say displaced to 50

The virus NEVER deletes the boot sectors cos then the disk becomes unuseable. So it displaces boot parameters to some other place.

Putting in array as we dont wanna read the shit just put it back at 0 sector.

To find exactlocation. run a loop 20880 times.
20880

Generic solution for Virus Cleaner

Better...

```
#include "dos.h"
main( )
{
  char a[ 512 ] ;  char ch = 'y' ;
  char *names[ ] = { "O Jerusalem", "Yankee Doodle",
                     "Robinson Cruso", "Eddie Murphy", ... } ;
  printf ( "Insert uninfected disk, Press any key" ) ;
  getch( ) ;
  absread ( 0, 1, 0, a ) ;
  while ( ch = = 'y' )
  {
     printf ( "Insert infected disk, Press any key" ) ;
     getch( ) ;
     abswrite ( 0, 1, 0, a ) ;
     printf ( "Another disk y/n" ) ;
     ch = getche( ) ;
  }
}
```

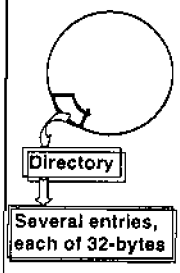
~~Never ask anyone the name of virus~~
~~Ask what it does & check if it is true~~

Ask where the virus sit then we know how to remove

echo → will display Y on screen

Never its name or origin

Directory Sector



Description	Size
Filename	8 bytes
Extension	3 bytes
Attribute	1 byte
Reserved for Future use	10 bytes
Time	2 bytes
Date	2 bytes
Starting Cluster number	2 bytes
Size	4 bytes

beginni spot in data space.

16 entries per sector

Printing Directory

```
# include "dos.h"
main( )
{
   struct entry
   {
      char n[ 8 ] ;  char ext[ 3 ] ;
      char unused[ 17 ] ;
      long int size ;
   } ;
   struct entry e[ 16 ] ; int i, j ;
   printf ( "Insert disk, Press any key" ) ; getch( ) ;
   absread ( 0, 1, 19, e ) ;
   for ( i = 0 ; i <= 15 ; i++ )
   {
      for ( j = 0 ; j <= 7 ; j++ )
         printf ( "%c", e[ i ].n[ j ] ) ;
      for ( j = 0 ; j <= 2 ; j++ )
         printf ( "%c", e[ i ].ext[ j ] ) ;
      printf ( "%ld", e[ i ].size ) ;
   }
}
```

} 32 bytes

19 cos first boot sector — 1
FAT — 9
FAT_2 — 9
19.

now always starts from 0 ∴ 19 = 0 − 18
so we start from 19

all names printed using printf %c & not %s as we aren't sure whether \0 is present or not.
Explorer/Dos-Dir — all read the dir structure sector.

Akshay Lal — XX

Directory Entry

?

Directory Entry	
Element	Bytes
Filename	8
Extension	3
Attribute	1
Reserved	10
Time	2
Date	2
Starting clus. no.	2
Size	4
Total	32

9 byte entry (10)

25/01/99

?

9 Bytes

Why 2 Bytes?

25/01/99

(1999 - 1980) * 512 = 9728
\+ 01 * 32 = 32
\+ 25 = 25
9785

yr	m	d
0010011	0001	11001
7 Bits	4 Bits	5 Bits

0010011 ⟹ 16 + 2 + 1 ⟹ 19 + 1980 ⟹ 1999

0001 ⟹ 1

11001 ⟹ 16 + 8 + 1 ⟹ 25

2 byte s = 16 bit -

we aren't able to pick up specific digits. Unless bitwise operators aren't used.

Distribution

Date

yr	m	d
7 Bits	4 Bits	5 Bits

Time

hr	m	s
5 Bits	6 Bits	5 Bits

① Select → ~~FAT~~ searches for file.
Attr. ① hidden
② sub/vol/system
③ first cluster

IMP

③ → FAT goes on putting together

Read only = 1 write protect on t/-r
Hidden = 1 t/-h

```
# include "dos.h"
main()
{
   struct entry
   {
      char n[ 11 ] ;
      char attr ;
      char rest[ 20 ] ;
   } ;
   struct entry e[ 16 ] ; int i ;
   printf ( "Insert disk, PAK" ) ;
   getch( ) ;
   absread ( 0, 1,19, e ) ;
   for ( i = 0 ; i <= 15 ; i++ )
      e[ i ].attr = 2 ;
   abswrite ( 0, 1, 19, e ) ;
}
```

Attribute Byte 514 FFF

7 6 5 4 3 2 1 0

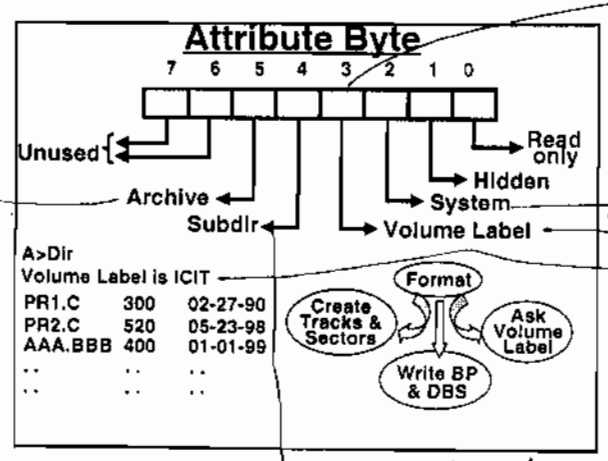
Read only

Hidden

00000010

2

tells the PC whether the file has been modified or not → used in Backup



→ if 1 then volume lable

→ 1 = then file is operating system file. (dos file windows system files)

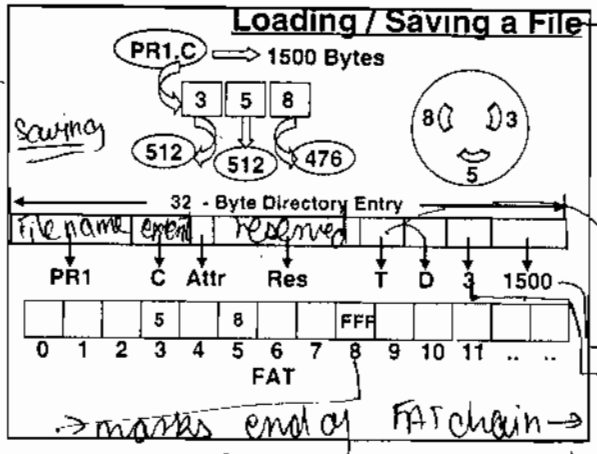
→ after formating a lable is by default put.

Always supply vol. lable.

also gets stored in directory to confirm sub = 1.

Vol Lable goes to directory. Becomes a Vol. Lable only if bit = 1.

loading



→ loading & saving is only done by OS not DOS/software.

[1 cluster = 1 sector for disk (1.44MB) in each sector 512kb]

time/date of creation

→ ~~Binary location~~ size of file.

→ begining cluster of file

→ marks end of FAT chain → no more cluster after.

at first 512 get stored but size 1500
thus second 512 get store to start again.

No cluster will ever get stored b/w 2 files.
all clusters of that file not in same one can be dif

all files are represented by 1 FAT.
but all files have separate directories

Input / Output In C

printf( )	getkey( ) — user defined func
scanf( )	absread( )
getch( )	abswrite( ) → Keywords?
putch( )	
gets( )	
puts( )	

All I/O is done using Functions

non-are keywords all are functions from library.

I/O Functions

Conio

Console I/O | Disk I/O | Port I/O

if a value ~~sent to speaker~~. predefined value in memory.

Formatted	Unformatted
char, int, float, string } printf( ), scanf( )	char - getch( ), getche( ), getchar( ), fgetchar( ), putch( ), putchar( ), fputchar( )
	int - / float - → No Std. Library Function
	string - gets( ), puts( )

f is formatted printing to screen.
scaning from keyboard

greater control on o/p & i/p..

format → ~~controls~~ where how / how much / allign can be specified.

Unformated → prints/gets from current position of string.

printf( )

printf ("format string", list of variables / constants / Expressions) ;

Can contain

Format Specifiers	- %c	
	- %d %u %ld %lu %X %x %o %i	
	- %f %lf %Lf	
	- %s	
Escape Sequences	\n - newline	\r - carriage return
	\t - tab	\'
	\a - alert	\"
	\b - backspace	\\
Any other character	Hello & other strings of its type.	

Formatting Output

```
main()
{
   int a = 35 ;
   clrscr( ) ;
   printf ( "a = %d", a ) ;
   printf ( "\na = %2d", a ) ;
   printf ( "\na = %6d", a ) ;
   printf ( "\na = %-7d", a ) ;
   printf ( "\na = %1d", a ) ;
}
```

Screen

```
0 1 2 3 4 5 6 7 8 9 0 1 2
a = 3 5
a = 3 5
a = - - - - 3 5
a = 3 5 - - - - -
a = 3 5
```

2 cols get reserved for value of a.
6 cols get reserved for value of a (gets printed Right alligned).
-7 7 cols get reserved. (-ve sign says left allign/)
only ~~both~~ one col is reserved.

%10u	%15ld	%5c	%7lu	%20s	%-35s

right allig 3 → 1 left allign
35
10 coll 15 coll 5 coll 7 coll 35 colls
reserve

* if reserved no. of rows is less it screws the limitation & just goes forward.

Six decimal values after decimal.

Formatting Output

```
main()
{
   float a = 35.6927 ;
   clrscr( ) ;
   printf ( "a = %f", a ) ;
   printf ( "\na = %7.4f", a ) ;
   printf ( "\na = %8.2f", a ) ;
   printf ( "\na = %-7.1f", a ) ;
   printf ( "\na = %1.1f", a ) ;
}
```

Screen

```
0 1 2 3 4 5 6 7 8 9 0 1
a = 3 5 . 6 9 2 7 0 0
a = 3 5 . 6 9 2 7
a = - - - 3 5 . 6 9
a = 3 5 . 7 - - -
a = 3 5 . 7
```

left allign

%10.2lf	%-15.3Lf

·4 means only 4 decimals
7 total no. of coll reserved (including L & R & decimal pt.).
rounding off takes place (& right allig
1 col. & 1 pt after decimal spot

No. of cols → get ignored if needed.
After decimal → never ignored.

0.0 → 0 colls ignored
.0 no position after decimal.

Escape Sequences

```
main()
{
   clrscr( ) ;
   printf ( "Hello\b\bHi" ) ;
   printf ( "\nHello\tHi" ) ;
   printf ( "\nHello\nHi" ) ;
   printf ( "\nHello\rHi" ) ;
   printf ( "He said \"Let us go\". " ) ;
   printf ( "\n//\\\\" ) ;
}
```

backspace
next line
Home

Screen

```
HelHi
Hello    Hi
Hello
Hi
Hillo
He said "Let us go".
//\\
```

~~\t gives a tab of 8 spaces~~

8 8

printf ("\nHello\tHi") ; → Hello Hi

printf ("\nHello\t\tHi") ; → Hello Hi

These 8 segments are called Print Zones
So on '\t' takes you to the Next Print Zone
(has no bearing on what tab setting was made in the editor).

\" → gives a " on screen.
\\ → the second \ will get printed

What's The Difference

\n - New Line
\r - Carriage Return

Enter

```
main( )
{
   char ch ;
   ch = getch( ) ;
   printf ( "%d", ch ) ;
   printf ( "%d %d", '\r', '\n' ) ;
}
```

13

13, 10

ascii value of \r also 13

Enter key hit
\r is generated.
The OS converts \r \n combinat
(\r \n) → (line feed + Carriage return)

\r takes to bigin line ; \n next line

---

optional

```
printf ( "Format String", variables ) ;
```

Format Specifiers
Escape Sequence
Any Other Chars

Variables
Constants
Expressions

```
scanf ( "Format String", List of variables ) ;
```

Format Specifiers only

compulsory

```
float a ;
scanf ( "%10.2f", &a ) ;
```

Acceptable, but rarely used

10.2f
~~0.2~~ 2 places after decimal
10 number should be accomod
within 10 places.

Unformatted Console I/O Functions

```
main( )
{
   char ch ;
   printf ( "Press any key" ) ;
   getch( ) ;

   printf ( "Another disk Y/N" ) ;
   ch = getche( ) ;

   printf ( "Delete all files Y/N" ) ;
   ch = getchar( ) ;

   /* or ch = fgetchar( ) ; */

   putch ( ch ) ;
   putchar ( ch ) ;
   fputchar ( ch ) ;
}
```

same

getch( ) → Just Element / never printed

getche( ) → echo (means get it & print) no Enter k

getchar( ) → element + Enter key + print.

/* or ch = fgetchar( ) ; */ → ~~formatted~~ function

putch, putchar, fputchar → is also a Macro.

Peculiarities

getch( ) - Doesn't wait for enter.
getche( ) - Echoes character.
getchar( ) - Waits for enter. Macro.
fgetchar( ) - Waits for enter. Function.

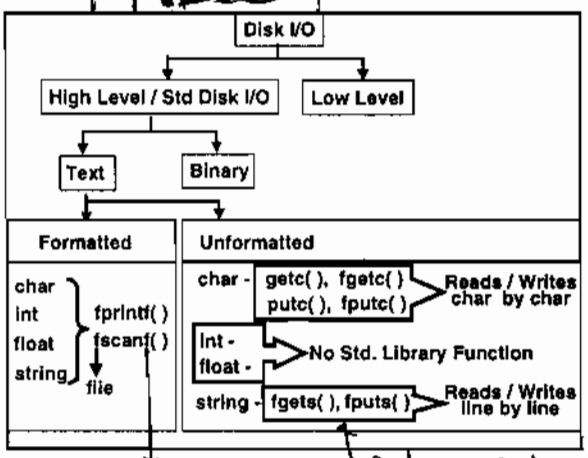
Macro faster, longer
function slower, compact.

Array like int or a float are stored in data segment of memory

absread / abswrite → sector oriented.

Array — 21

FILES



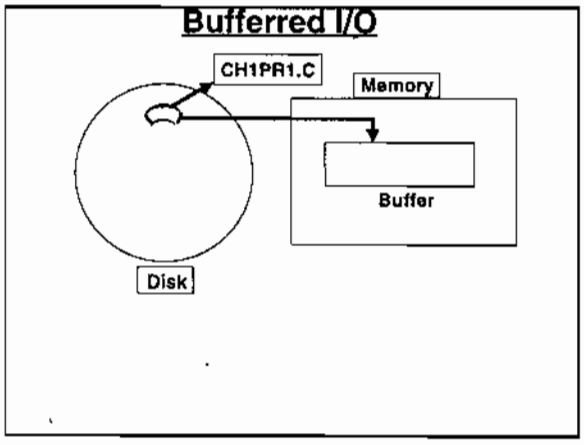
formatted

Put a string into a file
take a string outta file.

Text files — Ascii files

Binary → won't be able to understand the contents of file

Bufferred I/O



brinin' file tottay to boffer then readin' becomes faster.
reading also gets done from buffer.

EOF = end of file is a Macro.

data type

8 - filename
3 - extension
• - 1
\0 - 1

Displaying File Contents

```
# include "stdio.h"
main( )
{
    FILE *fp ;
    char fname [ 13 ] ; char ch ;
    printf ( "Enter file name" ) ;
    gets ( fname ) ;
    fp = fopen ( fname, "r" ) ;
    while ( 1 )
    {
        ch = getc ( fp ) ;
        if ( ch == EOF )
            break ;
        printf ( "%c", ch ) ;
    }
    fclose ( fp ) ;
}
```

reading.

"r" → is actually a string

fopen = std lib function.

searches for file name in dir of disk
checks whether file exists or not
whether it is a vol lable / system file.
" it is hidden
the gets ready to put it in buffer.

file pointer

load char from file being pointed to by fp.

empties buffer

returns the ascii value from file (actually buffer) charater by character.

file terminator → a character whose ascii value is 26

Keyword → redefines a type to any new name

any user specific name.

typedef

```
unsigned long int i, j ;
```

```
typedef unsigned long int ull ;
ull i, j ;
```

```
struct employee
{
        =
};
struct employee e1, e2 ;
```

```
struct employee
{
        =
};
typedef struct employee emp ;
emp e1, e2 ;
```

→ here struct employee = emp

```
struct a
{
        =
        char *p ;
};
typedef struct a FILE ;
```

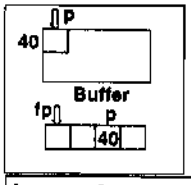
```
FILE *fp ;
```

is a ptr to a structure

Struct a = FILE

A Fresh Look

```
# include "stdio.h"
main( )
{
    FILE *fp ;
    char fname[13] ; char ch ;
    printf ( "Enter file name" ) ;
    gets ( fname ) ;
    fp = fopen ( fname, "r" ) ;
    while ( 1 )
    {
        ch = getc ( fp ) ;
        if ( ch == EOF )
            break ;
        printf ( "%c", ch ) ;
    }
    fclose ( fp ) ;
}
```




```
fp -> p = fp -> p + 1 ;
```

(for incrementation) this is also done by getc.

```
struct a
{
        =
        char *p ;
};
typedef struct a FILE ;
```

P is defined in structure as char *p

P initially remains at first entry of buffer.

fopen will create that structure & buffer

Tips

- fopen( ) - Standard Library Function to open the file. → buffer & structure.
- fopen (fname, "r") ; - Returns NULL (MEMORY) if file not found.

```
# define  NULL  0
```

- Every file ends with a char with ascii value 26.
- EOF is a macro.

```
# define  EOF  -1
```

- FILE is a structure defined in stdio.h
- getc( ) reads char & also shifts pointer to next char.

for every new file there is a new buffer & a new structure.

Copying Files

```
# include "stdio.h"
main( )
{
    FILE  *fs, *ft ;
    char  fname [ 13 ] , target [ 13 ] ;  char  ch ;
    printf ( "Enter source & target file names" ) ;
    gets ( fname ) ;   gets ( target ) ;
    fs = fopen ( fname, "r" ) ;  ft = fopen ( target, "w" ) ;
    while ( 1 )
    {
        ch = getc ( fs ) ;
        if ( ch == EOF )
            break ;
        putc ( ch, ft ) ;
    }
    fclose ( fs ) ;  fclose ( ft ) ;
}
```

writin'

terminator.

(which char, & where)

separate declarations are neccessary or fcloseall();
closes all open files

Better

Filecopy

```
# include "stdio.h"
main( )
{
    char  source [ 67 ], target [ 67 ] ;
    FILE  *fs, *ft ;
    printf ( "Enter source file name" ) ;
    gets ( source ) ;
    printf ( "Enter target file name" ) ;
    gets ( target ) ;
    fs = fopen ( source, "r" ) ;
    if ( fs == NULL )
    {
        printf ( "Can't open source file" ) ;
        exit( ) ;
    }
    ft = fopen ( target, "w" ) ;
    if ( ft == NULL )
    {
        printf ( "Can't open target file" ) ;
        fclose ( fs ) ; exit( ) ;
    }
    while ( 1 )
    {
        ch = getc ( fs )
        if ( ch == EOF )
            break ;
        putc ( ch, ft ) ;
    }
    fclose ( fs ) ;
    fclose ( ft ) ;
}
```

account for no file.

67 cos sometimes file not present in local directory but in some othe director or drive. (path required sometimes)

66 max length of path.
67 for \0

fs, ft will hold add of structure

puts the 26 at end of traget file.

Tips

- Maximum path length = 66
- fs = fopen (s, "r") - Returns NULL if file is absent
  - Returns address of structure, if present
- ft = fopen (t, "w") - Creates new file if file is absent
  - Overwrites file, if present
- fclose( ) - closes the file and adds 26 at the end, if opened for writing

Syntax

getc (<file pointer>)
putc (<character>, <file pointer>)

[getc returns a -1 when a 26 Ascii is reached.]

⟹ thus instead of EOF we can use -1

⟹ fp is a ptr to a structure.

⟹ getc returns char's ascii
also increments to next char.

⟹ file handing structure declared in std.io

Stdio. — FILE
EOF
NULL

~~while (ch = getc(fs) != EOF)~~ → ~~so long as we dont reach the last char keep storing in ch.~~

```
while ((ch=getc(fs)) != EOF)
        putc(ch, ft);
```

very imp

Files :)

akshay Lal -22

Coding/Decoding

(encryption/decryption)

Global

```
# include "stdio.h"
FILE *fs, *ft ;
main( )
{
    char s[ 67 ], t[ 67 ] ; char ch ;
    printf ( "Enter source" ) ;
    gets ( s ) ;
    printf ( "Enter target" ) ;
    gets ( t ) ;
    fs = fopen ( s, "r" ) ;
    if ( fs == NULL )
    {
        printf ( "Unable to open" ) ;
        exit( ) ;
    }
    ft = fopen ( t, "w" ) ;
    if ( ft == NULL )
    {
        printf ( "Unable to open" ) ;
        fclose ( fs ) ; exit( ) ;
    }
    puts ( "Encrypt / Decrypt
                E/D" ) ;
    ch = getche( ) ;
    switch ( ch )
    {
        case 'E' :
        case 'e' :
            encrypt( ) ;
            break ;
        case 'D' :
        case 'd' :
            decrypt( ) ;
            break ;
    }
    fclose ( fs ) ;
    fclose ( ft ) ;
    remove ( s ) ;
}
```

user defined.

delete a file (name of the file) not pointer.

we use name of file & not ptr cos on disk there is no ptr only name of file.

Cracks

(offset code) Method – I

Offset Cipher

Encryp

```
encrypt( )
{
    char ch ;
    while ( ( ch = getc ( fs ) ) != EOF )
        putc ( ch + 128, ft ) ;
}
```

(all graphic characters)

diff to understand

Graphic Characters

I am at Nagpur

J bn bu Obhqvs

too simple

```
decrypt( )
{
    int ch ;
    while ( ( ch = getc ( fs ) ) != EOF )
        putc ( ch - 128, ft ) ;
}
```

Decryp

(all graphic characters) decoded to english.

{Char e → 128 → 127 but ch = 128 is out of range thus we use int}

{unsigned char 0 - 255 but this will turn into an infinite loop as -1 will never be reached}

not necessary to make int as either way we are offsetting the characters

here the off set is constant thus crackin is simple

Method – II

Substitution Cipher

```
encrypt( )
{
    char str1[ ] = "Z a A 3 4 n M ...." ;
    char str2[ ] = "N j J x 8 9 0 M ...." ;
    char ch ; int i ;
    while ( ( ch = getc ( fs ) ) != EOF )
    {
        for ( i = 0 ; i < strlen ( str1 ) ; i++ )
        {
            if ( ch == str1[ i ] )
            {
                putc ( str2[ i ], ft ) ;
                break ;
            }
        }
    }
}
```

I am at Nagpur

all characters (unique) in both strings.

funda (strchr searches for char in string)

search for char in str1 replace char from st2.

if we make another string alt. when & where from you put the char

1 — Str1
2 — Str2
3 — Str1 etc.

Thus more no of strings more weird is the decrypt.

*****

Remove Blank Lines

```
# define NOTBLANK 1
# define BLANK 0
main( )
{
   char s[ 67 ], t[ 67 ] ; char str[ 80 ] ;
   FILE *fs, *ft ;
   int a ;
   puts ( "Enter source & target" ) ;
   gets ( s ) ; gets ( t ) ;
   fs = fopen ( s, "r" ) ; ft = fopen ( t, "w" ) ;
   while ( fgets ( str, 79, fs ) )
   {
      a = isblank ( str ) ;
      if ( a == NOTBLANK )
         fputs ( str, ft ) ;
   }
   fclose ( fs ) ; fclose ( ft ) ;
   remove ( s ) ; rename ( t, s ) ;
}
```

Contd...

target, source

renames the file.

79 → max char to read at a time.

str → stored in str.

fs → pointer

→ fgets gets string from file.

→ here != EOF

as fgets returns a NULL (=0) so if we wanted to use the != we will use != NULL (but of no use as 0 = false ∴ while(0) is exit).

...Contd

```
isblank ( char *p )
{
   while ( *p != '\0' )
   {
      if ( *p != ' ' && *p != '\t' && *p != '\n' )
         return ( NOTBLANK ) ;
      else
         p++ ;
   }
   return ( BLANK ) ;
}
```

collects base add of string

I am at Nagpur
ABCD

____
____ ____ ____
____ ____ ____
____ IJK ___

Continues for whole line.

either blank line is ① just enter
② tonnes of tabs
③ space bar ∞
④ \r \n.

so if either any other No blank
so if not (1-4) noblank.

{remove all /* or // from file}

Writin' to records:

Handling Records

```
# include "stdio.h"
main( )
{
   struct employee
   {
      char n[ 20 ] ; int a ; float s ;
   } ;
   struct employee e ;
   FILE *fp ; char ch = 'y' ;
   fp = fopen ( "emp.dat", "w" ) ;
   while ( ch == 'y' )
   {
      puts ( "Enter record" ) ;
      scanf ( "%s %d %f", e.n, &e.a, &e.s ) ;
      fprintf ( fp, "%s %d %f \n", e.n, e.a, e.s ) ;
      puts ( "Add another y/n" ) ;
      ch = getche( ) ;
   }
   fclose ( fp ) ;
```

Cont...

Printf → o/p on screen
fprintf → o/p on file.

→ prints on the screen (writes to file).

Syntax

fgets (<storage array> , <max length of char. to be read at one time> , <file pointer>)

(checkin' if shit is written back to file)

...Cont

```
fp = fopen ( "emp.dat", "r" ) ;
while ( fscanf ( fp, "%s%d%f ", e.n, & e.a, & e.s ) != EOF )
    printf ( "\n %s%d%f", e.n, e.a, e.s ) ;

fclose ( fp ) ;
}
```

fscanf → reading records from file

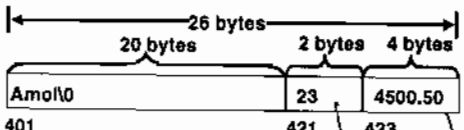
Only in Theory we use fpr & fsc
Never in Practice

why?

**** fread/fwrite

Starting at 401 write 26 bytes once into file [ptr fp]

Why The Difference




```
fprintf ( fp, "%s %d %f \n", e.n, e.a, e.s ) ;
fwrite ( &e, sizeof ( e ), 1, fp ) ;
while ( fscanf ( fp, "%s%d%f", e.n, & e.a, & e.s ) != EOF )
while ( fread ( &e, sizeof ( e ), 1, fp ) != 0 )
```

* based of files
1. while
2. some func in while as condition.
3. limit of read/write

from a file pointed to fp read once. from 401 to 26 bytes

the int is converted to a string so ASCII of 2 & ASCII of 3 is printed but thats okay cos 1 byte + 1 byte = 2 byte & int = 2.

but for float when same methodology is applied when returned to disk we get shit loads more. here 7 bytes ie 3 more

when using fwrite always open in binary mode → wb
fread → wr

File Opening Modes

imp

"w" ↔ "wt" "wb"	- If file is absent, new file is created - If file is present, existing file is overwritten - Operation - Writing
"rb" "rt"	- If file is absent, returns NULL - If file is present, loads the file into memory and sets pointer pointing to first character in file - Operation - Reading
"ab" "at"	- If file is absent, new file is created - If file is present, loads the file into memory and sets up a pointer beyond last character in file - Operation - Writing at end of file.

Cont...

Text: fscanf, fprintf } wt/rt/w

Binary: fread, fwrite } wb, rb.

Additional Parameters

the + allows both operations.

...Cont

write / read mode

"wb+" "wt+"
- If file is absent, new file is created
- If file is present, existing file is overwritten
- Operation - Reading / Writing

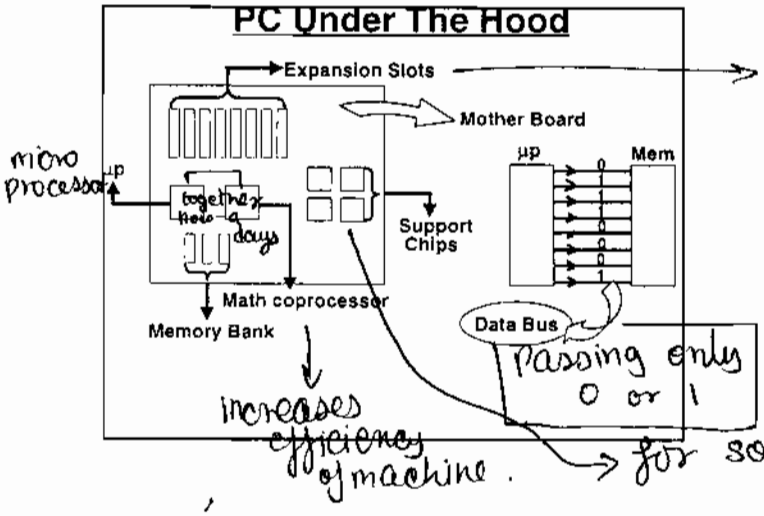
"rb+" "rt+"
- If file is absent, returns NULL
- If file is present, loads the file into memory and sets pointer pointing to first character in file
- Operation - Reading / Writing

read / write mode

"ab+" "at+"
- If file is absent, new file is created
- If file is present, loads the file into memory and sets up a pointer beyond last character in file
- Operation - Writing at the end / Reading

GRAPHICS

@xay Lal — 24



for ext. drives & hardware.

for soundblaster, drives etc.

Types of Microprocessor

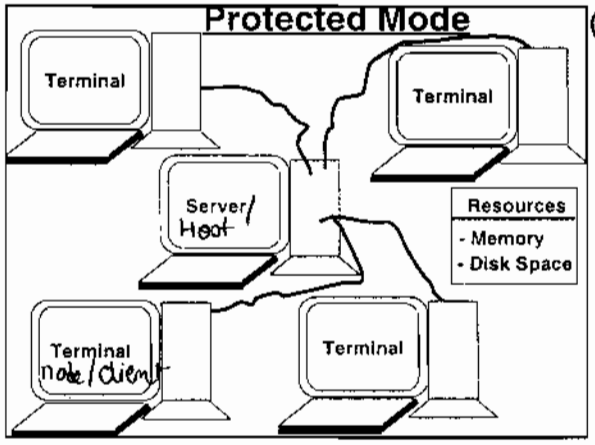
Micro processor	Data Bus	Address Bus	Max. Memory	Mode of operation
8088 (PC, XT)	8 bits	20 bits	1 mb	Real
8086 (PC, XT)	16 bits	20 bits	1 mb	Real
80286 (AT)	16 bits	24 bits	16 mb	R/P
80386 (AT386)	32 bits	32 bits	4096 mb	R/P
80486 (AT486)	32 bits	32 bits	4096 mb	R/P
Pentium	64 bits	64 bits	2^{44} mb	R/P

real/protected mode

XT = extra hdd.
AT = advance technology.

data bus — data passes
Add bus — adds pass (max no. of bytes in memory $= 2^{20}$).
gives max memory.

Dos — real (reduces add bus to 20 b)
Unix / Linux / Win → R/P.

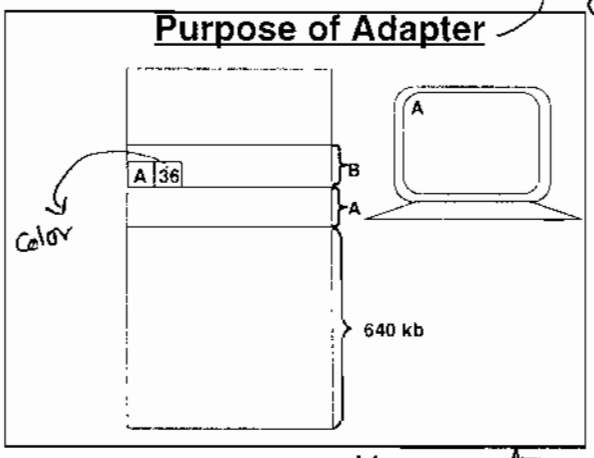


① LAN

type: total memory processor speed on server
nothing on clients

type: Tonnes of resources on server
Some local memory in client.

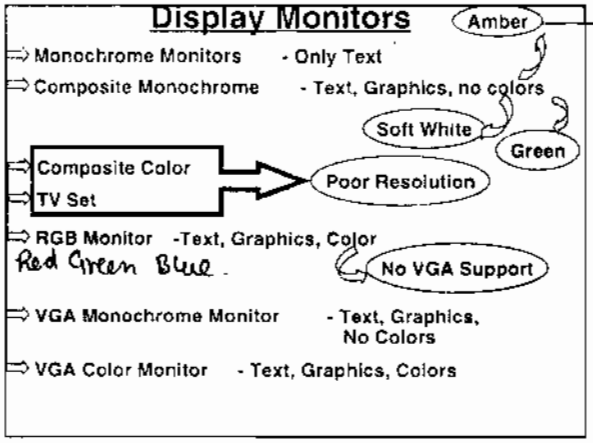
keeps picking up ASCII2 color values from screen.



refresh rate = 60 to 80 times in one sec.
↓
managed by adapter.
reason why we have to keep refreshing Phosphorescence CRO.

So the adapter keeps refreshing the screen. even if no shit is being entered

Screen saver → illuminates all pt. on screen at equal times so as to increase life of CRO / Phospor bronze screen.

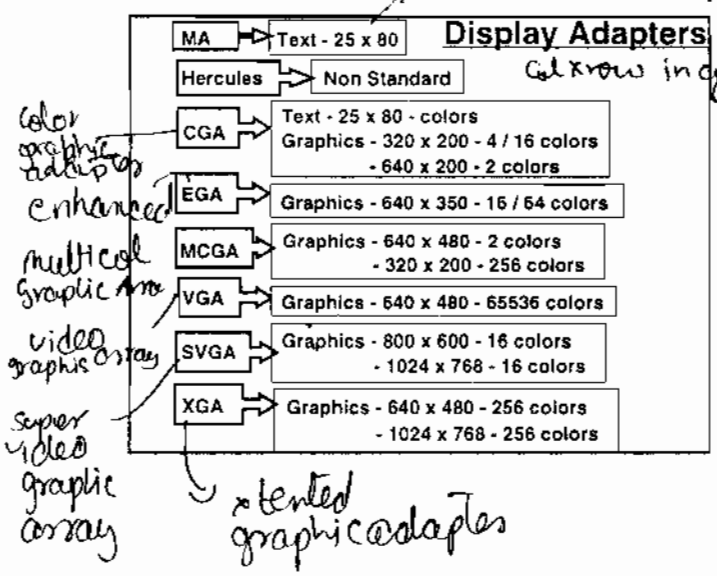


Nagpur Railway — Orange on Black, Black on Orange ↓ Hurts eye.

VGA – video graphics adapter.

row x col in normal text mode.

MA = monochrome adap



Col x row in graphs

color graphic adapter

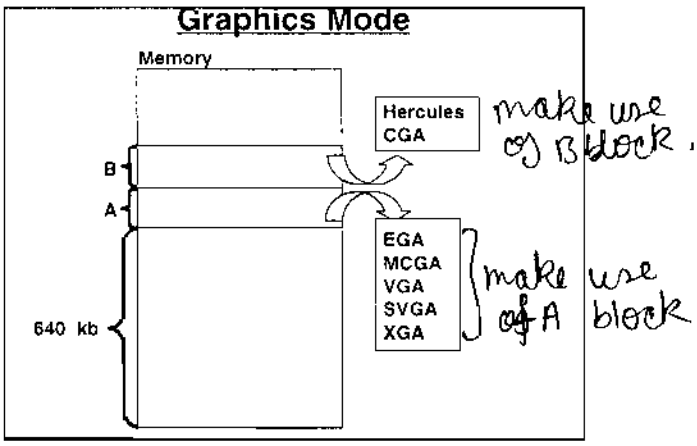
enhanced

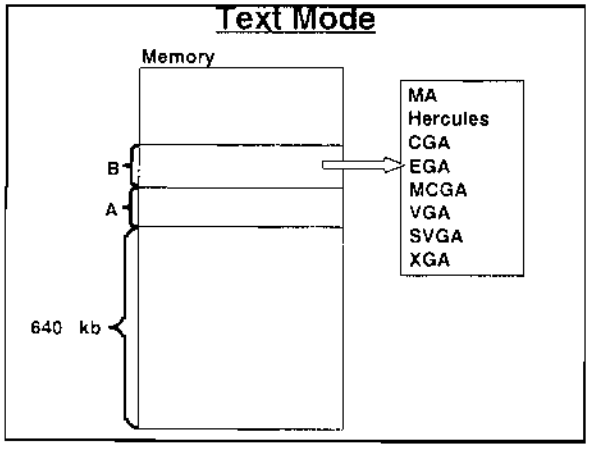
multicol Graphic Array

video graphics array

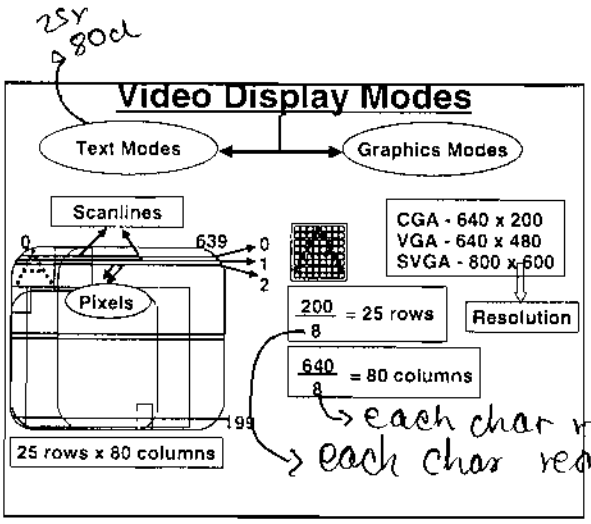
super video graphic array

extented graphic adapter





while working only in text mode always B block



Pixel → picture cell / picture element

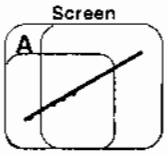
higher resolution → all pixels are packed closer together. Thus char will appear sharper.

Text (r, c)
Graphics (c, r)

To Draw

Text Mode → Graphics Mode

Screen

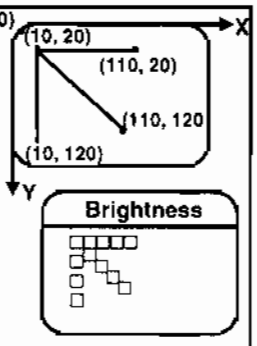


each resolution is called a mode (each of which is given a specific no.

macro

```
# include "graphics.h"
main( )
{
   int gm, gd = DETECT ;
   int x ; int y ;
   initgraph ( & gd,  & gm,
                "c:\\tc\\bgi"  ) ;
   for (  x = 10 ;  x <= 110 ; x++)
      putpixel ( x, 20, WHITE ) ;
   for ( y = 20 ; y <= 120 ; y++ )
      putpixel ( 10, y, WHITE ) ;
   for ( x = 10,  y = 20 ;  x <= 110 ;
            x++, y++ )
      putpixel ( x, y, WHITE ) ;
   getch( ) ;
   closegraph( ) ;
   restorecrtmode( ) ;
}
```

horizon
vertical
diagonal



initgraph() initializes graphic func ...
↓
loads a prog (device driver prog) which drives the monitor / adapter.

bgi → borland graphic interphase

gd = graphics drives
gm = graphics mode

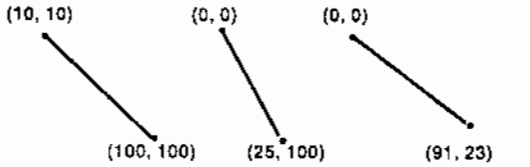
closegraph( ) → opp. to init (shuts down graphics system)

putpixel(x, y, colour) → lights up a pixel on screen.

horizontal brightest
vertical less
diagonal least

screen 640×480
640 cols ∴ more pixels combined together so brighter
480 rows ∴ lesser " " " " " lesser.
diagonal very less.

Line Drawing



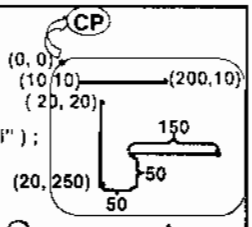
Ratio 1 : 1 | Ratio 1 : 4 | Ratio 91 : 23

```
line ( 10, 10, 100, 100 ) ;
line ( 0, 0, 25, 100 ) ;
line ( 0, 0, 91, 23 ) ;
```

→ Bresenham's Line drawing Algorithm

Current position

```
# include "graphics.h"
main( )
{
    int gm, gd = DETECT ;
    initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
    setcolor ( RED ) ;
    line ( 10, 10, 200, 10 ) ;
    moveto ( 20, 20 ) ;
    lineto ( 20, 250 ) ;
    moverel ( 50, -50 ) ;
    linerel ( 150, 0 ) ;
    getch( ) ;
    closegraph( ) ;
    restorecrtmode( ) ;
}
```



line func doesnt alter position of cursor.

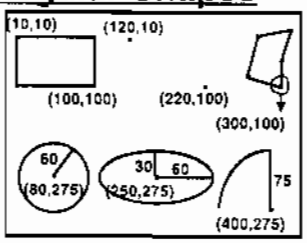
lineto → from Cp to give position → it also places Cursor to (20,250) ∴ Cp position chang

moverel → from Cp move x & y.

(from x, from y, from 90°, to 180°, 75 radius)

```
# include "graphics.h"
main( )
{
    int gm, gd = DETECT ;
    int x ; int y ;
    int a[ ] = { 300, 100, -, -,
                 -, -, 300, 100 } ;
    initgraph ( &gd, &gm,
                "c:\\tc\\bgi" ) ;
    x = getmaxx( ) ;
    y = getmaxy( ) ;
    setcolor ( RED ) ;
    rectangle ( 0, 0, x, y ) ;
    circle ( 80, 275, 60 ) ;
    ellipse ( 250, 275, 0, 360,
              60, 30 ) ;
    arc ( 400, 275, 90, 180,
          75 ) ;
    setcolor ( YELLOW ) ;
    rectangle ( 10, 10, 100, 100 ) ;
    bar ( 120, 10, 220, 100 ) ;
    drawpoly ( 6, a ) ;
    getch( ) ;
    closegraph( ) ;
    restorecrtmode( ) ;
}
```

Regular Shapes



getmax(x y) → max x, max y.

bar3d → 3d bar charts.

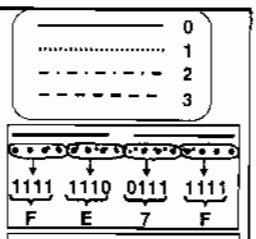
bar → fills the rectangle. (fillcolour ≠ setcolour).

drawpoly → (no. of pairs of co-ord, base add of array where co-ordinates stored)

circle → (centre x, centre y, radius)

ellipse → (centre x, centre y, from 0°, to 360°, major radius axis, minor radius axis)

```
# include "graphics.h"
main( )
{
    int gm, gd = DETECT ;
    initgraph ( &gd, &gm,
                "c:\\tc\\bgi" ) ;
    struct linesettingtype l ;
    setcolor ( RED ) ;
    line ( 10, 10, 100, 100 ) ;
    getlinesettings ( &l ) ;
    setlinestyle ( 2, 0, 3 ) ;
    line ( 10, 20, 100, 20 ) ;
    setlinestyle ( 4, 0xFE7F, 3 ) ;
    line ( 10, 30, 100, 30 ) ;
    setlinestyle ( l.linestyle,
                   l.upattern,
                   l.thickness ) ;
    line ( 10, 40, 100, 40 ) ;
    getch( ) ; closegraph( ) ;
    restorecrtmode( ) ;
}
```




```
struct linesettingtype
{
    int linestyle ;
    unsigned upattern ;
    int thickness ;
}
```

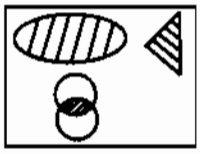
light up = 1
no light = 0

we are given only 16 bits to design.

restoring original/default line style.

[line style no., user defined pattern, thickness of line (1 or 3)]

```
# include "graphics.h"
main( )
{
    int gm, gd = DETECT ;
    int a[ ] = { 300, 100, 400, 150, 400, 50, 300, 100 } ;
    initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
    setfillstyle ( 4, BLUE ) ; setcolor ( RED ) ;
    fillellipse ( 100, 100, 50, 25 ) ;
    setfillstyle ( 5, YELLOW ) BLUE ;
    fillpoly ( 4, a ) ; Fillpoly(4,a);
    setcolor ( RED ) ;
    circle ( 100, 180, 50 ) ;
    circle ( 100, 250, 50 ) ;
    setfillstyle ( 4, GREEN ) ;
    floodfill ( 100, 210, RED ) ;
    getch( ) ;
    closegraph( ) ;
    restorecrtmode( ) ;
}
```



for an ellipse. centre

→ x radius y radius

** for no-regular shape use floodfill.

** starting from that pt keep colouring till a RED colour is reached

```
# include "graphics.h"
main( )
{
    int gm, gd = DETECT ;
    initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
    outtextxy ( 10, 10, "Hello" ) ;
    settextjustify ( CENTER_TEXT, BOTTOM_TEXT ) ;
    settextstyle ( 2, 0, 5 ) ;
    outtextxy ( 100, 150, "Hello" ) ;
    setcolor ( RED ) ;
    line ( 100 - 3, 150, 100 + 3, 150 ) ;
    line ( 100 , 150 - 3, 100, 150 + 3 ) ;
    getch( ) ; closegraph( ) ;
    restorecrtmode( ) ;
}
```

Point would be at the center and below text

Hello

Hello

settextstyle (type, orientation, pt. size)

Default(, Top_Text)

→ vertically pt above string.
horizontally = length of string.

→ centre allignment
~~the pt bottom~~ the pt will be below the text.

16 diff sytes / FONTS , 0 → horizontal , 1 → vertically

1 = 8x8 matrix
2 = 16x16 matrix
3 = 24x24 matrix

all chars made out of 8x8 matrix

Cryptography in C/C++ Apress
Data compression in hand book. - BPC

a hard times (Half life) how to change / alter | Basic requirements of an OS | rootkit is an OS | trojan.

Axay Lal — 25

Colors And Palettes

```
main( )
{
    int gd = DETECT, gm , i, j, x, y, color, startcolor, height,
                                                    width ;
    struct palettetype palette ;
    struct viewporttype vp ;
    initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
    getpalette ( &palette ) ;
    rectangle ( 0, 20, 639, 479 ) ;
    outtextxy ( 200, 10, "Palette demonstration" ) ;
    getviewsettings ( &vp ) ;
    width = ( vp.right - vp.left ) / 16 ;
    height = ( vp.bottom - vp.top - 20 ) / 16 ;
    x = 0 ;  y = 20 ;  startcolor = 0 ;
    for ( j = 0 ; j <= 15 ; j++ ) ;
    {
        color = startcolor ;
        for ( i= 0 ; i <= 15  ;i++ ) ;
        {
            setfillstyle ( SOLID_FILL, color++ ) ;
```

Cont...

..Cont

```
                bar ( x, y, x + width, y + height ) ;
                x = x + width + 1 ;
            }
            startcolor++ ;
            x = 0 ;
            y += height + 1 ;
        }
        getch( ) ;
        while ( !kbhit( ) )
            setpalette ( random ( 16 ), random ( 65 ) ) ;
        setpalette ( &palette ) ;
        getch( ) ;
        closegraph( ) ;
        restorecrtmode( ) ;
    }
```

one's compliment.

Bitwise Operators

- One's Complement ~ Tilde

```
char ch = 32 ;
printf ( "%d", ~ch )
```

-33

printf("%d", ch) ~ 32

char ch = 32 ;

7	6	5	4	3	2	1	0

check 1 or 0 — set to 0 — set to 1

00100000 ⇓~ 11011111 ⇓ -ve

11011111 − 1 = 11011110 ⇓~ 00100001 ⇓ -33

Utility

```
while (( ch = getc ( fs ) ) != EOF )
    putc ( ~ch, ft ) ;
```

Encrypt — Decrypt

encrypt

dealing with individual; not a long-string all together.

~ will print out a negative no.

to decrypt do same as

1's comp of A is say A'
1's comp of A' is A

as no assignment.

Bitwise Operators

- One's Complement - -
- Left Shift and Right Shift

<< >>

```
unsigned char ch = 32 ;
printf ( "%d", ch >> 2 ) ;
printf ( "%d", ch ) ;
printf ( "%d", ch << 1 ) ;
```

— 8
— 32
— 64

00100000 → >> 2 → 00001000

00100000 → << 1 → 01000000

→ right most 2 bits are lost.

→ empty slots filled with 0.

∴ right shift is basically division by 2

→ the left most bit is lost
default bit is 0.

∴ left shift is basically multi by 2

Utility of <<, >>

32-Byte Directory Entry	
Filename	8
Extension	3
Attribute	1
Reserved	10
Time	2
Date	2
Starting clus no.	2
Size	4

06 / 01 / 99

year

(1999 - 1980) * 512	=	9728
+ month 1 * 32	=	32
+ Day 6	=	6
		9766

Binary

yr	m	d
0010011	0001	00110
7 Bits	4 Bits	5 Bits

added.

→ day processors came into existance.

to get year if we right shift by 9 spaces we will obtain.

000 . . . 0010011
year.

// for m & d.

d, m, y

```
main( )
{
   int dt = 9766 ;
   int d, m, y ;
   y = ( dt >> 9 ) + 1980 ;
   m = ( dt << 7 ) >> 12 ;
   d = ( dt << 11 ) >> 11 ;
   printf ( "%d%d%d", d, m, y ) ;
}
```

at the end dt still equalls 9766 as assignment hasent taken place.

6 1 1999 → 9766

0010011 0001 00110 —>> 9 → 000000000 0010011

0010011 0001 00110 —<< 7 → 0001 00110 0000000 —>> 12 → 000000000000 0001

0010011 0001 00110 —<< 11 → 00110 00000000000 —>> 11 → 00000000000 00110

6, 1, 1999.

>> 9 to get 7 MSB.

<< 7 >> 12?
to get 4 bits from middle.

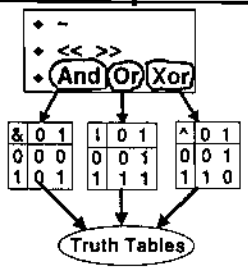
<< 11 >> 11 ?
to get 5 LSB bits.

>> 9 why?
cos $2^9 = 512$.
$(x + 1980) * 512 = 9728$.

time can also ///ly be done this way.

Bitwise used basically used for specific elements in a file.

Bitwise Operators



&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

Bitwise And

Utility of &

```
main( )
{
   char n ;
   printf ( "Enter a no." ) ;
   scanf ( "%d", &n ) ;
   if ( n & 8 == 8 )
   {
      printf ( "3rd bit is on" ) ;
      n = n & 0xF7 ;
      printf ( "%d", n ) ;
   }
   else
      printf ( "3rd bit is off" ) ;
}
```

to check whether a bit is on or off

&	0	1
0	0	0
1	0	1

	7	6	5	4	3	2	1	0	
	-	-	-	-	0/1	-	-	-	n
&	0	0	0	0	1	0	0	0	
	0	0	0	0	0/1	0	0	0	

	-	-	-	-	1	-	-	-
&	1	1	1	1	0	1	1	1
	-	-	-	-	0	-	-	-

Put off a bit.

Procedure
- make every bit 0 except 3.
ie (anding) bits with 0 = 0.

$0 \& 0 = 0$
$1 \& 0 = 1$

now in 3 0 or 1
and with 1
so whatever present in 3 gets o/p
without disturbing others.

Same
$0 \& 1 = 0$
$1 \& 1 = 1$

so if we AND n with 8 (000
o/p = 8 then 3rd bit is on
o/p = 0 the 3rd bit is off.

To put 3rd bit off, AND with 0.
so with whole n → AND with 0 only 3rd & rest with 1.

```
# include "dos.h"
main( )
{ struct entry
  {
    char n[ 11 ] ;
    char attr ;
    char rest [ 20 ] ;
  } ;
  struct entry e[ 16 ] ;
  int i ;
  absread ( 0, 1, 19, e ) ;
  for ( i = 0 ; i <= 15 ; i++ )
  {
    if ( e[ i ].attr & 1 == 1)
      e[ i ].attr &= 0xFE ;
    if ( ( e[ i ].attr & 2 ) != 2 )
      e[ i ].attr = e[ i ].attr | 2 ;
  }
  abswrite ( 0, 1, 19, e ) ;
}
```

	0/1
&	0000000 1
	0000000 0/1

	1
&	11111110
	0

	0/1
&	000000 1 0
	000000 0/1 0

	0
|	00000010
	1

32-Byte Dir Entry	
Filename	8
Extension	3
Attribute	1
Reserved	10
Time	2
Date	2
St. clus no.	2
Size	4

	7	6	5	4	3	2	1	0

Hidden

Read only

0 → Read/write
1 → Read only

0 → ~~write~~ non hidden
1 → hidden.

Bitwise Compound assignment operator. /// (n+=10)

~ = doesn't exist
Urinary operator
all others binary.

Putimage () puts image on screen. on x, y, coordinates (top left corner) from memory location (P).

Animation

```
# include "graphics.h"
# include "alloc.h"   for malloc
main( )
{
  int gm, gd = DETECT ;
  int n ; char *p ; int x ;
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  setcolor ( RED ) ;
  setfillstyle ( 4, BLUE ) ;
  circle ( 100, 120, 30 ) ;
  n = imagesize ( 70, 90, 130, 150 ) ;
  p = ( char * ) malloc ( n ) ;   base add
  getimage ( 70, 90, 130, 150, p ) ;
  for ( x = 70 ; x <= 400 ; x += 5 )
  {
    putimage ( x, 90, p, XOR_PUT ) ;   - errase
    putimage ( x + 5, 90, p, XOR_PUT ) ;   - redraw at new position
  }
  getch( ) ; closegraph( ) ;
}
```

moving

(70, 90)

(130, 150)

imaginary rectangle.

getimage → store the array in memory pass the coordinates of imaginary rectangle

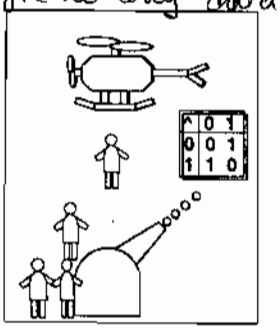
→ gets size of image.

P is a Ptr as malloc returns a base address.

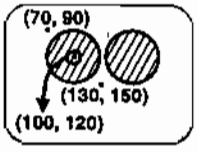
(char*) type casting.

malloc returns a ptr which is not specific to any data type (void* pointer).

malloc(n); → memory allocation allocates 'n' no. of bytes in memory also returns base address of those 'n' bytes. (it doesnt know what type of data is going to be stored. some type cast this void ptr to a char ptr.



0	0	1
0	0	1
1	1	0



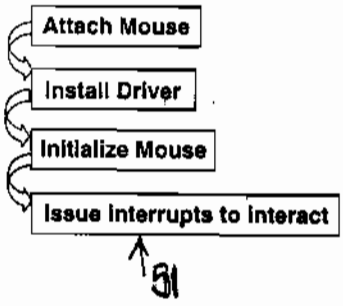
XOR_Put does an xor btw those pixels present at location with those pixels from memory.

① → will erase the previous image
eg any pixel that can be seen is ON
∴ so 1 EXOR 1 = 0 → off. } so all pixels go off.
|||ly 0 EXOR 0 = 0 → off.

② → redraws the image.
now 0 on screen EXOR with 1 from memory
|||ly 0 EXOR 1 = 1 → ON
0 EXOR 0 = 0 → OFF

to speed up motion.
for(x+=5) → increase.

Mouse Programming



→ to check if hit or no hit check if xy co-ordinates of chopper & falling dude with bullet is equal.

Freehand Drawing

```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
```

User defined

```
initmouse( )
{
    union REGS i, o ;
    i.x.ax = 0 ;
    int86 ( 51, &i, &o ) ;
}
```

Function to initialize func. for all other interrupts service no gets stored in ah.

Freehand Drawing

```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
```

User defined.

max screen boundry ⇒ drawn boundary for mouse.

```
restrictmouse ( int x1, int y1, int x2, int y2 )
{
    union REGS i, o ;
    i.x.ax = 7 ;
    i.x.cx = x1 ;
    i.x.dx = x2 ;
    int86 ( 51, &i, &o ) ;
    i.x.ax = 8 ;
    i.x.cx = y1 ;
    i.x.dx = y2 ;
    int86 ( 51, &i, &o ) ;
}
```

Service no

to restrict in x direction.

~~Service no.~~

to restrict in y direct.

Freehand Drawing

```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
  showmouseptr( ) ;
```

user defined.

```
showmouseptr( )
{
    union REGS i, o ;
    i.x.ax = 1 ;
    int86 ( 51, &i, &o ) ;
}
```

Service No for mouse

1- show
2- hide
3- returning status/position

Freehand Drawing

```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
  showmouseptr( ) ;

  getmousestatus ( &b, &x, &y ) ;
```

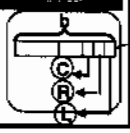

```
getmousestatus ( int *b, int *x, int *y )
{
    union REGS i, o ;
    i.x.ax = 3 ;
    int86 ( 51, &i, &o ) ;
    *b = o.x.bx ;
    *x = o.x.cx ; *y = o.x.dx ;
}
```

gets whether LCR is On or OFF if ON return X & Y coordinate
b = if ON/OFF
x = x coor
y = y coor.

Freehand Drawing

```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
  showmouseptr( ) ;

  getmousestatus ( &b, &x, &y ) ;
```

now we want to check whether L = 0 OR 1

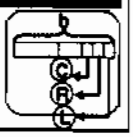


16 bits

Freehand Drawing

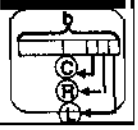
```
main( )
{
  int gm, gd = DETECT,
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
  showmouseptr( ) ;

  getmousestatus ( &b, &x, &y ) ;
  if ( b & 1 == 1 )
  {
     hidemouseptr( ) ;
```


```
hidemouseptr( )
{
    union REGS i, o ;
    i.x.ax = 2 ;
    int86 ( 51, &i, &o ) ;
}
```



Freehand Drawing

```
# include "dos.h"
# include "graphics.h"
main( )
{
  int gm, gd = DETECT, b, x, y, prevx, prevy ;
  initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
  initmouse( ) ;
  restrictmouse ( 0, 0, getmaxx( ), getmaxy( ) ) ;
  showmouseptr( ) ;
  while ( ! kbhit( ) )
  {
    getmousestatus (&b,&x,&y ) ;
    if ( b & 1 == 1 )
    {
      hidemouseptr( ) ; prevx = x ; prevy = y ;
      while ( b & 1 == 1 )
      {
        line ( prevx, prevy, x, y ) ;
        prevx = x ; prevy = y ;
        getmousestatus ( &b, &x, &y ) ;
      }
      showmouseptr( ) ;
    }
  }
  getch( ) ; closegraph( ) ; restorecrtmode( ) ;
}
```



AND

Puts. / says whether bit is on / OF

And with 1 to get undisturbed value (to check).
And with 0 to get a 0 as answer (to change values to 0)

OR

To convert on to OFF / OFF to ON

OR with 1 ⟶ OFF to ON
OR with 0 ⟶ no change.

Type cast

```
int c ,d.
float a.
c=2
d=5
```

$\boxed{a = d/c \Rightarrow 2}$

$a = (\text{float})\ d/c \Rightarrow 2.5$

fundu

Axay Lal (28)

from top of o/s
keep (0, 1000)

1000 = paragraphs
each paragraph = 16
∴ 16000 bytes.
So from top of O/S ~~p~~ store 16,000 byt for my prog
1000 isn't really a const. any no. will do.

How are TSRs Different

Terminate but Stay Resident

Remain in memory

Triggered Execution

Paragraph boundary is always a multiple of 16. where one para ends & another begins

Calling Functions

```
main( )
{
   void display( ) ;
   void ( *p )( ) ;
   display( ) ;
   p = display ;
   ( *p )( ) ;
}
display( )
{
   printf ( "Hello" ) ;
}
```

One Way

One More Way

→ return & recieve ntn.
→ ptr to a funct. which recieves/returns intn
→ getting base add of func
→ calling of function.

Some Definitions

void (*p)( ) ; ⟹ ptr to a func which recieves ntn & returns ntn.
void *p( ) ; ⟹ function which is going to recieve ntn & returns a void ptr similar to Malloc
void (*p)(int, float) ; ⟹ ptr to fun which recieve int & float & returns a void.
void (*p)(int *, char **) ; ⟹
int **(*p)(char **, float *) ; ⟹

returns a ptr to an integer point.

Size of ptr = 2 bytes

Define

p is a pointer to a function which accepts a double, an int pointer and a float pointer and returns a far long int pointer.

```
long int far *( *p )( double, int *, float * ) ;
```

Prototype of a function p which receives two doubles and a float pointer and returns a long double pointer.

```
long double * p( double, double, float * ) ;
```

```
int *( *p[ 3 ] )( int *, int ** )
```

each element is a ptr to a func which recieves 2 int ptr & int ptr to ptr & returns an int ptr.

prev = *36 also same it gets add. Thus it is a ptr to rom bios func which recieves ntn & returns ntn

replace the add in IVT with that of the func.

```
# include "dos.h"
void interrupt ( *prev )( ) ; void interrupt our( ) ;
int i ;char far *s = 0xB8000000 ;
main( )
{
  prev = getvect ( 9 ) ;
  setvect ( 9, our ) ;
  keep ( 0, 1000 ) ;
}
our( ) void interrupt
{
  for ( i = 0 ; i <= 3999 ; i += 2 )
  {
    if ( *( s + i ) >= 65 && *( s + i ) <= 90 )
      *( s + i ) += 32 ;
    else
    {
      if ( *( s + i ) >= 97 && *( s + i ) <= 122 )
        *( s + i ) = *( s + i ) - 32 ;
    }
  }
  ( *prev )( ) ;
}
```

extended keyword

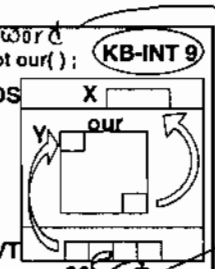
KB-INT 9

interrupt no. 9

base add of func.

our func

returns control to rom bios func



get called only when interrupt is called.

replace X (add of BIOS) with Y (add of Y) (our func)

basically ptr to a func. So IVT is ntn but an array of ptrs to functions.

sets a value in interrupt Vector table.

to make prog resident in memory.

9 = Keyboard interrupt.
8 = time interrupt

COLOR

```
# include "dos.h"
void interrupt( *prev )( ) ;  void interrupt our( ) ;
char far *s = 0xB800000 ;  int i, color, count ;
main( )
{
  prev = getvect ( 8 ) ;
  setvect ( 8, our ) ;
  Keep ( 0, 1000 ) ;
}
void interrupt our( )
{
  count++ ;
  if ( count = = 182 )
  {
    for ( i = 1 ; i <= 3999 ; i += 2 )
      *( s + i ) = color ;
    color++ ;
    if ( color > 255 )
      color = 0 ;
    count = 0 ;
  }
  ( *prev )( ) ;
}
```

(18.2x10)

1 sec — 18.2
10 sec — 182

after Colour

calls rom bios whose add stored in prev.

Colour will change only after every 10 secs. (Colour of screen).

8 occurs 18.2 ticks in one sec.

User → On Cap
Comp → *prev puts it on
*kb puts it on.
User → off cap
Comp → *prev puts it off
*kb puts it on
User → gets pissed

Caps Locked

```
# include "dos.h"
void interrupt ( *prev )( ) ;
void interrupt our( ) ;
char far *kb = 0x417 ;
main( )
{
    prev = getvect ( 9 ) ;
    setvect ( 9, our ) ;
    keep ( 0, 1000 ) ;
}
void interrupt our( )
{
    ( *prev )( ) ;
    *kb = 64 ;
}
```

7	6	5	4	3	2	1	0

Caps Lock
1 - On
0 - Off

→ anyway it is on/or OFF

→ we are alltering all values in 0x417. We should only alter 6. ntn else.

*kb OR= 64 → whatever syntax

How Many Hits

```
# include "dos.h"
void interrupt ( *prev )( ) ;
void interrupt our( ) ;
main( )
{
    prev = getvect ( 9 ) ;
    setvect ( 9, our ) ;
    keep ( 0, 1000 ) ;
}
void interrupt our( )
{
    ( *prev )( ) ;
    ( *prev )( ) ;
}
```

Rom bios gets called twice.
eg type main
we get mmaa iinn.
backspace will remove 2 chars at a time.

We can also make it so that every alt. time A is deactivated

Deactivating 'A'

```
# include "dos.h"
void interrupt ( *prev )( ) ; void interrupt our( ) ;
main( )
{
    prev = getvect ( 9 ) ;
    setvect ( 9, our ) ;
    keep ( 0, 1000 ) ;
}
void interrupt our( )
{
    if ( inportb ( 0x60 ) == 30 )
    {
        outportb ( 0x20, 0x20 ) ;
        return ;
    }
    else
        ( *prev )( ) ;
}
```

Port no → Code no, value

Scan Code of A

if not A call rom bios (normal functioning).

inportb( )
→ reading a Byte from an Input Port (pickup scan codes).

Port specific place in memory
Each port given a no. & given a specific job.
0x60 = Keyboard port.

writes a Byte to Output port.

~~0x20 = monitor port~~.
0x20 = controller port

ISR :- Interrupt Service Routine
- gets called only when Interrupt occurs
eg: all bios func.

| void interrupt (* prev)();

a func recieves/returns ntn

TSR whenever the prog terminates effect vanishes.
here even when we exit the prog effect remains in the memory.

Every Virus is a TSR. it has to get char (interrupts) screw with it & prints its on the screen (interrupts) so TSR is very imp.

Every virus which spreads gets control of interrupt no. 19. which is a disk I/O interrupt.

---

what happens when we press keyboard?

key bit → scan code → available at 0x60

↓

rombios called no. 9 → pickup scan code → scancode & corresponding ASCII gets stores in keyboard buffer

→ bios tells compu its work done.
ie put value at 0x20 from control no. 0x20.

So the processor knows that job is over if outportb(0x20, 0x20) is encounted by bios. Similar to feinda of the end of the file is (-5).

Akshay Lal

Delete

- Open a file called "WS.EXE"
- Skip past first 1701 bytes
- Copy rest into TEMP.EXE
- Delete WS.EXE
- Rename TEMP.EXE to WS.EXE

all Exe files in binary

Methodology of erradicating virus in Exe file.

{Virus in file in data structure}

The Program

```
# include "stdio.h"
main( )
{
   FILE *fs ; char  e[ 10 ] ; int  i ;
   char  s[  ] = { 0xE5, 0x92, 0x0, 0x20, 0x20, 0x20,
                   0x20, 0x20, 0x20, 0x20 } ;
   fs = fopen ( "WS.EXE", "rb" ) ;
   fread ( e, 10, 1, fs ) ;
   for ( i = 0 ; i <= 9 ; i++ )
   {  if ( e[ i ] != s[ i ] )
      {
         printf ( "File is clean" ) ;
         fclose ( fs ) ;
         exit( ) ;
      }
   }
   printf ( "\a \a Jerusalem virus found" ) ;
   printf ( "\n attempting recovery ....." ) ;
```

signature

WS.EXE

1701

virus

TEMP.EXE

Cont..

rb → read binary.

e, 10, 1, fs

e = add of array
10 = read ten bytes
1 = once
fs = read from.

Virus in an exe file in a data space

Always .C files never gets executed

Virus size 1701 bytes.

sig. of virus = first few bytes of virus code.

..Cont

```
   ft = fopen ( "TEMP.EXE", "wb" ) ;
   fseek ( fs, 1701L, SEEK_SET ) ;
   while ( ( ch = getc ( fs ) ) != EOF )
         putc ( ch, ft ) ;
   fclose ( fs ) ;
   fclose ( ft ) ;
   remove ( "WS.EXE" ) ;
   rename ( "TEMP.EXE",  "WS.EXE" ) ;
}
```

char ch ;
FILE *ft ;

WS.EXE

1701

TEMP.EXE

fseek(file ptr, 1701L, MACRO

Positions ptr

from begining go head 1701 byte i.e. stop on 1702 byte L = long int.

remove ⇄ delete

rename ⇄ remaning

** Virus always goes to first 10 or last ten.
Once it has ~~gone~~ been executed it goes to memory
then it attaches itself with
every exe file
* never goes to FAT & Directory as
either never gets executed.

Altering basic commands

Changing Commands

```
# include "stdio.h"
main( )
{
    FILE *fp ; char ch ;
    fp = fopen ( "command.com", "rb+" ) ;
    while ( ( ch = getc ( fp ) ) != EOF )
    {
        if ( ch == 'D' )
        {
            ch = getc ( fp ) ;
            if ( ch == 'I' )
            {
                ch = getc ( fp ) ;
                if ( ch == 'R' )
                {
                    fseek ( fp, -3L, SEEK_CUR ) ;
                    putc ( 'Y', fp ) ; putc ( 'P', fp ) ; putc ( 'K', fp ) ;
                    fseek ( fp, 0L, SEEK_CUR ) ;
                }
            }
        }
    }
    fclose ( fp ) ;
}
```

at this pt cursor is after R

COMMAND.COM
DIR

Necessary on alteration

fseek btw Putc & getc

-3L
{long int go 3 back by 3 bytes}

→ gets loaded every time OS is loaded. Has a list of all comman names & allowed extensions

Seek_Cur = from current position of cursor
-end = from end.

bugg must be given

Low Level Disk I/O

/* File copy prog */

C>filecopy PR1.C NEWPR1.C

```
main ( int argc, char *argv[ ] )
{
    int in, out ;
    if ( argc != 3 )
    {
        puts ( "Improper Usage !" ) ;
        puts ( "Correct Usage : c> filecopy source target" ) ;
        exit( ) ;
    }
    in = open ( argv[1], O_RDONLY | O_BINARY ) ;
    if ( in == -1 )
    {
        puts ( "Unable to open" ) ;
        exit( ) ;
    }
    out = open ( argv[2], O_WRONLY | O_BINARY| O_CREAT )
```

Cont..

argc → a count of arguments
argv → a vector of arguments (array).
(array of ptr. to strings)

→ for cpyin always 3 strings rear.
① filecopy ② source ③ target all attribute attach to filecpy. ~~so any~~ so argc = 3
not more not less

open → low level Disk I/O.
(add of first string, MACROS) both.

MACROS defined in low level header file. (O_ is a file openin' flag)

O_RD only → read only.
O_Wr only → write only
O_Binary → in binary
O_Create → if not present → create a new file

..Cont

```
    if ( out == -1 )
    {
        puts ( "Unable to open" ) ;
        close ( in ) ;
        exit( ) ;
    }
    while ( ( n = read ( in, buffer, 512 ) ) != 0 )
        write ( out, buffer, n ) ;
    close ( in ) ;
    close ( out ) ;
}
```

```
# include "fcntl.h"
# include "types.h"
# include "stat.h"
```

```
int n ;
char buffer[512] ;
```

→ not fclose

read → reads file chunk by chunk.
read 512 bytes at a time store in buffer from file ptr in.
(array array)

→ write (into buffer, (no. of bytes present in n))

read → returns no. of bytes read

⇒ (Imp)

Interaction with Hardware Through C

- Using Standard Library Functions
- Directly programming the Hardware
- Calling BIOS / DOS Routines (function)

Printf Scanf — very reliable but very slow.
absread abswrite

fastest way, but very unreliable as drive electronic is not documented very wel

resonably fast
resonably reliable.

→ interrupt is a signal given by use to the micro processor telling it to stop what its doing & do what we tell it do.

Steps in Calling BIOS / Dos routine

- Issue an interrupt
- Find Interrupt number
- Multiply interrupt no. by 4
- Pick address of BIOS / DOS routine from IVT
- Store current CPU register values in stack
- Setup CPU registers with values required by BIOS /DOS routine being called
- Transfer control to BIOS / DOS routine
- Let BIOS / DOS routine get executed
- Collect values returned by BIOS/ DOS routines into ordinary variables
- Restore values from stack back into CPU registers
- Resume original interrupted activity

Push

Pop

→ why multiply by 4.

→ as each add is 4 bytes. eg: inter no. 9 to get add 9*4 = 36 bytes from beginning. [36, 37, 38, 39]

→ if anything is to be told to bios func. These value are stored here. That is why Register don't always work.

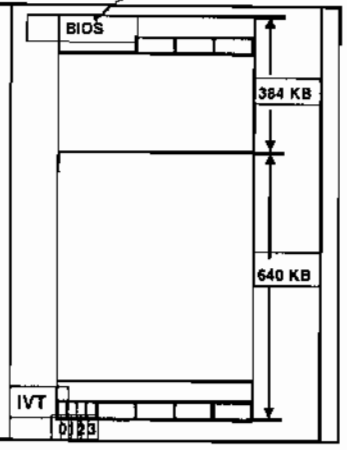
→ if we want to collect shit from bios func.

→ all stored in stack usin Push & put back into registers ~~using~~ from ~~top~~ stack using Pop.

functions no names but address. no. of func = 256. as 256 interrupts.

Interrupt no.'s

Keyboard	9
Disk	19
Printer	23
Mouse	51
VDU	16
Timer	8



IVT

interrupt vector Table → holds add of all bios func.

each add 4 bytes big (all for ptrs)

| (0123) = interrupt no. 0 | next 4 for Inter no. 1 | next 4. for Inter no. 2 |

Position Cursor

Interrupt no. 16 — related with screen.
DH - Row no.
DL - Column no.
AH - Service no.

Dx — names of CPU registers as they pass values to bios.

CPU Registers total 14

Data Registers

AX - Accumulator - AH, AL
BX - Base - BH, BL
CX - Count - CH, CL
DX - Data - DH, DL

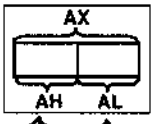
Segment Registers

CS - Code Segment
DS - Data Segment
ES - Extra Segment
SS - Stack Segment

Offset Registers

BP - Base Pointer
SP - Stack Pointer
IP - Instruction Pointer
SI - Source Index
DI - Destination index

Flags Register

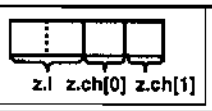


high byte low byte.

Structures

```
main( )
{
   struct a
   {
      int i ;
      char ch[ 2 ] ;
   } ;
   struct a z ;
   z.i = 512 ;
   printf ( "%d", sizeof ( z ) ) ; 4
   printf ( "%d %d %d", z.i, z.ch[ 0 ], z.ch[ 1 ] ) ;
}
```

512 0 0



Size of a structure is sum of sizes of its elements.

union — keyword.

Unions

Permits access to same memory locations in more than one way

```
main( )
{
   union a
   {
      int i ;
      char ch[ 2 ] ;
   } ;
   union a z ;
   z.i = 512 ;
   printf ( "%d", sizeof ( z ) ) ;
   printf ( "%d %d %d", z.i, z.ch[ 0 ], z.ch[ 1 ] ) ;
}
```

z.i: 00000000 00000010
z.ch[0] Low, z.ch[1] High

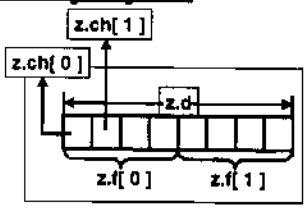
Binary of 512: 00000010 00000000
High, Low

2 is a union variable /// ly to structure

size of union = size of longest variable.

How Many Bytes

```
union a
{
   double d ;
   float f[ 2 ] ;
   char ch[ 5 ] ;
} ;
union a z ;
printf ( "%d", sizeof ( z ) ) ;
```



Size of a union is size of the longest element of the union.

how to access memory location is dependent on how we ~~want to acce~~ declare it.

z.d all 8 bytes.
z.f[0] for first 4 bytes (4 at a time).
z.ch[0] for first 1 byte (1 at a time).

4 pages = 4 screen = 4 cursor

Positioning Cursor

```
main( )
{
   struct WORDREGS
   {
      int ax, bx, cx, dx, si, di, cflag, flags ;
   } ;
   struct BYTEREGS
   {
      char al, ah, bl, bh, cl, ch, dl, dh ;
   } ;
   union REGS
   {
      struct WORDREGS x ;
      struct BYTEREGS h ;
   } ;
   union REGS i, o ;
   clrscr( ) ;
   i.h.dh = 10 ; i.h.dl = 20 ; i.h.ah = 2 ; i.h.bh = 0 ;
   int86 ( 16, &i, &o ) ;
   printf ( "Hi" ) ;
}
```

row, col, service no., video page no.

O is output screen.

10 row
20 col

i = input → passed to bios
o = output → returned from bios.

16 = interrupt no. as related with VDU.

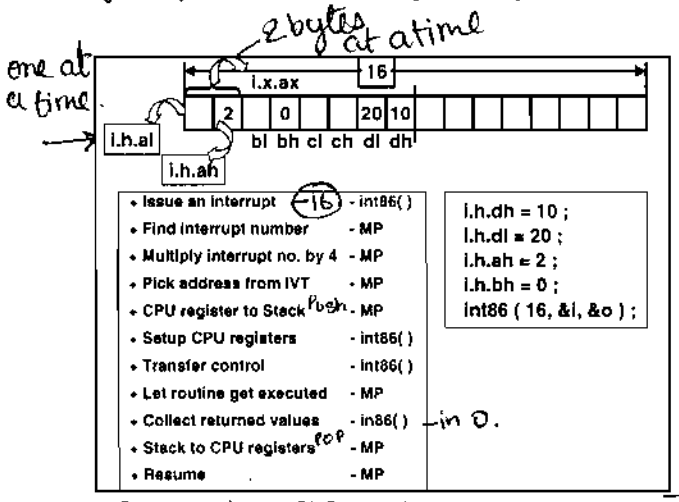
→ int86 allows to issues an interrupt issuing in ~~of~~ 8086 family of inter.

→ stdlib func.

→ allows to call any bios func

Service no like positioning, change color, clrscr(), etc. — all are associated with no. 2 is used cos we want to position cursor.

Size of union = size of biggest variable, ie 16



ax = 2 bytes
ah = 1 byte

we specifically entered values in dl dh & not in cl ch as Rom bios with the service no. 2 will look only in dl dh.

{ 2 dots doesnt always mean that there are nested structers it could be a union in struct or vice versa }.

blink. MP = micro processor.

intensity < high - red, low - pink.

0 100 0 1 00

Red on Red so for diff combinations

clscr() calls a bios func which clears screen

```
# include "dos.h"
main( )
{
    cls( ) ;
    gotorc ( 10, 20 ) ;
    printf ( "Hi" ) ;
}
gotorc ( int r, int c )
{
    union REGS i, o ;
    i.h.ah = 2 ;  i.h.bh = 0 ;
    i.h.dh = r ;  i.h.dl = c ;
    int86 ( 16, &i, &0 ) ;
}
cls( )
{
    union REGS i, o ;
    i.h.ah = 6 ;  i.h.al = 0 ;
    i.h.ch = 0 ;  i.h.cl = 0 ;
    i.h.dh = 24 ;  i.h.dl = 79 ; i.h.bh = 7 ;
    int86 ( 16, &i, &0 ) ;
}
```

gotorc()

Intensity bit of f/g

Blinking bit

Color Byte: R G B R G B — B/g, F/g

bac grnd for grnd

VDU Page no.

Lines to scroll, 0 to clear

Filler Attribute

always before issuing interrupts
① union REGS (2 variables)
② service no, othe values

all 1 - ~~black~~ white = 1
0 - ~~white~~ black = 0

color → black = 7, white = 112

color of screen — 7 = 0000 0111 — black B/n with white F/g

similar to window

top left & bottom left corner of screen

row no. & col no.

scrolls the matter up by one page depending value of al. (al no of lines by which we want to scroll up).

∴ 16 colors in text
256 color combinations (shading)

Delete Files

```
# include "dos.h"
main( )
{
    union REGS i, o ;
    char fname[ 67 ] ;
    printf ( "Enter file name" ) ;
    gets ( fname ) ;
    i.h.ah = 65 ;
    i.x.dx = fname ;
    intdos ( &i, &o ) ;
    if ( o.x.cflag == 0 )
        printf ( "Successful' ) ;
    else
        printf ( "Unable to delete" ) ;
}
```

remove don't ~~remove~~ delete file but invokes a dos func for deleting it.

service no. for deleting.

base add.

returned value stored in 0.

as we are calling dos func. we don't issue intr no. as it is taken that we have to call 32

Renaming Files

```
# include "dos.h"
main( )
{
   union  REGS  i, o ;
   char old[ 67 ], new[ 67 ] ;
   puts ( "Old Name:" ) ;
   gets ( old ) ;
   puts ( "New Name" ) ;
   gets ( new ) ;
   i.h.ah = 86 ;
   i.x.dx = old ;
   i.x.di = new ;
   intdos ( &i, &o ) ;
   if ( o.x.cflag == 0 )
      printf ( "Successful" ) ;
   else
      printf ( "Unable to rename" ) ;
}
```

→ service no. for renaming

→ base add.

→ returned value.

BOOK:- Advance Computing. Ray & Tandon.

$\boxed{200}$ BIS

net : Ralph Brown.

→ list of interrupts

[openin' a virus infected file is OK
never execute it]

[open a virus (boot) infected
never boot from it]

.C can be damaged but NEVER infected as
NO .C (non executable)/data files never get
infected. They can get screwed by the virus
found somewhere else NOT on data codes.
So no hacker ever wants to put a virus on a
non-executable file.

C:>> command line promt
dir
↑ command line arguments.

```
struct myfile
{
    char *a[4];
    char (*a)[4];
} e;

sizeof (e); ⟶ 10
```

---

int86(); for bios reading
intdos(); for dos value

soft ice — Exe file editor
hexadecimal editor
virtual CD

CD index / restart / reboot / Ctrl+Alt+Del / 10 algorith. / 2 bit algo. / hellylife. / → where is info. stored. / data format or (log) 3rd bit.

Extra - I. (linked list) :- @xay Lal :-

Imp (for all fundas read the PTR Book)

Arrays

No Flexibility

```
main( )
{
    int m1, m2, m3, i, per [ 10 ] ;
    for ( i = 0 ; i <= 9 ; i++ )
    {
        printf ( "Enter marks" ) ;
        scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
        per [ i ] = ( m1 + m2 + m3 ) / 3 ;
        printf ( "%d", per [ i ]) ;
    }
}
```

array → Static memory location.

* no variable permitted in array only const.

400
400+1 = 402 — as int

Dynamic Allocation

```
# include "alloc.h"
main( )
{
    int *p ; int m1, m2, m3, i, per ;
    printf ( "Enter no. of students" ) ;
    scanf ( "%d", &n ) ;
    p = malloc ( n * 2 ) ;
    for ( i = 0 ; i < n ; i++ )
    {
        scanf ( "%d%d%d", &m1, &m2, &m3 ) ;
        per = ( m1 + m2 + m3 ) / 3 ;
        *( p + i ) = per ;
    }
    for ( i = 0 ; i < n ; i++ )
        printf ( "%d", *( p + i )) ;
}
```

Memory: p → n * 2

→ dynamic memory. Keeps memory during execution.

* here ~~memory~~ variable or const is permitted.

P=(int*)&malloc(n*2);

if instead of *(p+i) we do *p=per; & incrementation done by p++; Then after loop p has more till the end & has to be brought back.

*(p+i) the ptr p doesn't ibely move. only the value is stored in diff memory location.

$\boxed{*(p+i) = P[i] = i(P)}$

if we want it to make it more dynamic put malloc in for loop.

Memory Allocation

Array	malloc( )
↓	↓
Static	Dynamic
↓	↓
Compile Time memory	Execution Time

Compile Time memory is alloc

Execution Time memory location

Array → when size known.

Malloc → when flexibility req.

More flexible

if we want more & more values in the memory even after length is over.

Better Still...

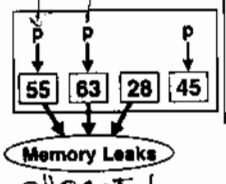
no flexibility

```
int *p[ ];
char ch = 'Y';
while ( ch = = 'Y' )
{
   scanf ( "%d %d %d", &m1, &m2, &m3 );
   per = ( m1 + m2 + m3 ) / 3 ;
   malloc ( 2 ) ;
   *p  = per ;
   printf ( "Another student y/n" ) ;
   ch = getche( ) ;
}
```

if we define *p as array of ptr.

first loop

2nd loop



Memory Leaks

memory allocated but access impossible

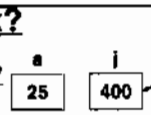
The 2nd ~~type~~ time loop is run the 2 bytes might/might not be adjacent. to previous Also P will shift from the first mem to the second.

So ~~when~~ printing will be impossible as the memory locations aren't known. Only the last value can be reached.

Memory Leak?

```
main( )
{
   int *j ; int *f( ) ;
   j = f( ) ;
   printf ( "%d", *j ) ;
}
f( )
{
   int a = 25 ;
   return ( &a ) ;
}
```

Prototype



~~Memory Leak~~

Dangling Pointer

int*

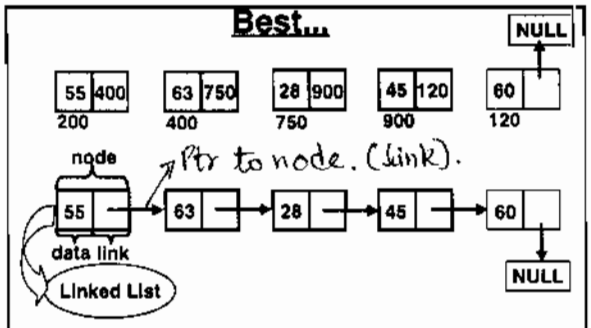
as we are returning the add of a value (integer pt).

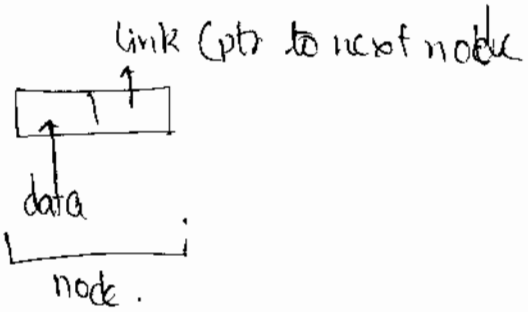
error as 'a' is local to f() so after f() finished 'a' is dead. So ptr is doing ntn & is addressing ntn.

opp to Memory location.

here we can reach address but ntn in that location.

Best...

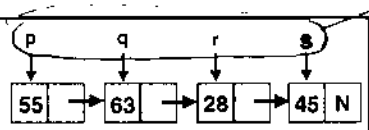




→ two 2 dissimilar entries (1) data (2) address (ptr to next node).

→ ptrs to the nodes

Linked List




```
main( )
{
  struct node
  {
     int data ;
     struct node *link ;
  } ;
  struct node *p, *q, *r, *s ;
  malloc ( sizeof ( struct node ) ) ;
  q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
  r = ...
  s = ...
  p → data = 55 ; q → data = 63 ; r → data = 28 ; s → data = 45 ;
  p → link = q ; q → link = r ; r → link = s ; s → link = NULL ;

                                                  Cont...
```

→ cyclic declaration.

→ a ptr to structure.

→ size of the structure. here 2 byte 2 int 2 ptr.

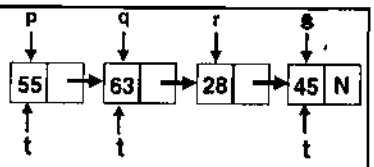
→ storing data

→ storing the linkup.

p= (struct node*) malloc (sizeof(struct node));

Imp

..Cont




```
# include "alloc.h"
main( )
{
  ...
  ...
  ...
  ...
  printf ( "%d %d %d %d", p → data, q → data,
                          r → data, s → data ) ;
  printf ( "%d %d %d", p → data, p → link → data,
                       p → link → link → data ) ;
  t = p ;
  while ( t != NULL )
  {
     printf ( "%d", t → data ) ; t = t → link ;
  }
}
```

I. II. III.

→ for defined linked list.

→ we want to go beyond last node

→ for a longer linked list.

t should be a struct node *t.

we dont do t→link != NULL the last data will not get printed for last t = NULL.

∴ we use t != NULL

→ we are actually shifting the ptr t.

t = t → link

Most General

```
# include "alloc.h"
struct node
{
   int per ; struct node *link ;
} ;
struct node *p ;
main( )
{
   char ch = 'Y' ; int pp, m1, m2, m3 ;
   p = NULL ;
   while ( ch == 'Y' )
   {
      scanf ( "%d %d %d", &m1, &m2, &m3 ) ;
      pp = ( m1 + m2 + m3 ) / 3 ;
      add ( pp ) ;
      printf ( "Another student y/n" ) ;
      ch = getche( ) ;
   }
}
```

→ self referential structure.

→ before entering loop no nodes present. P= NULL indicates an empty linked list. even after adding P should ALWAYS pt. only to the first value in list

adds value ONLY at the end.

→ putting r->link=NULL as (if) first value then link should be NULL as we dont know if more values will come

Most General

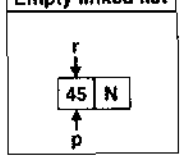
```
add ( int pp )
{
   struct node *r ; struct node *t ;
   r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
   r -> data = pp ; r -> link = NULL ;
   if ( p == NULL )
      p = r ;
   else
   {
      t = p ;
      while ( t -> link != NULL )
         t = t -> link ;
      t -> link = r ;
   }
}
```

allocating space.

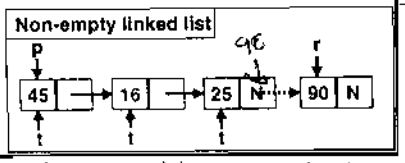
we want to stop at last node

search for previous node.

Empty linked list



Non-empty linked list



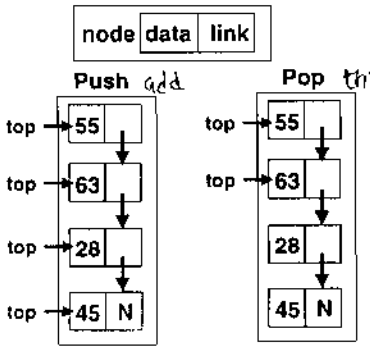
P==NULL means its the first node so p must pt to first node ∴ p=r.

We always want to keep P station so t moves. If P moves memory leak occurs as Previous node memory lost.

store the add on Next node in link of previous.

P is always at first (to prevent memory leak) & t keep moving up & down the linked list

Stack As A Linked List



Stack LIFO

Que FIFO.

Stack As A Linked List

```
#include "alloc.h"
struct node
{
   int data ;
   struct node *link ;
} ;
main( )
{
   struct node *top ;
   int i, item ;
   top = NULL ;
   push ( &top, 45 ) ;
   push ( &top, 28 ) ;
   push ( &top, 63 ) ;
   push ( &top, 55 ) ;
   displaystack ( top ) ;
   t = count ( top ) ;
   printf ( "Total items = %d" , t ) ;
   printf ( "\nPopped : " ) ;
   item = pop ( &top ) ;
   printf ( "%d ", item ) ;
   displaystack ( top ) ;
   t = count ( top ) ;
   printf ( "Total items = %d" , t ) ;
}
```

making top pt to first element.

&top as pass by reference is req cos tops value Must change as more values are inputted.

ptr to a ptr as &t is being passed.

size of(s)=2

Push Element On Stack

```
int push ( struct node **s, int item )
{
    struct node *q ;
    q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    q -> data = item ;
    q -> link = *s ;
    *s = q ;
}
```

allocates memory

value

S is pt to top which itself is/maybe pt to something else.

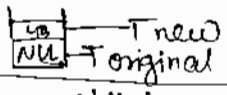
value at (*S).

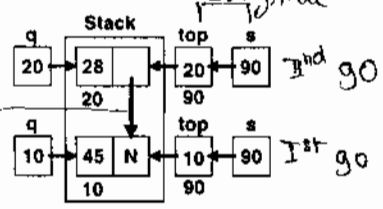
top = q isn't correct as top isn't available.

*S is pt to add of top.

∴ *S = q.

Now we have replaced top=NULL to top=10. ∴ top has moved to Next Inputted value





pointing to next node.

Top is never availible. Only its add in S is availible

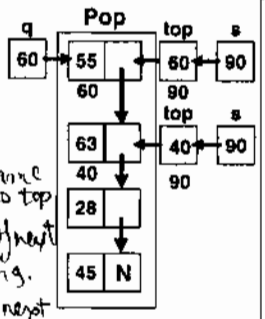
Pop Element From Stack

```
int pop ( struct node **s )
{
    int item ;
    struct node *q ;
    if ( *s == NULL )
        printf ( "Stack is empty" ) ;
    else
    {
        q = *s ;
        item = q -> data ;
        *s = q -> link ;
        free ( q ) ;
        return ( item ) ;
    }
}
```

checks if stack is empty

make q pt to same node as top

give add of next node & putting top at next node.



free opp to Malloc → ↑

↳ deallocates memory.

memory allocated/pt by q is deleted.

returning the popped value.

Display Stack Elements

```
void displaystack ( struct node *q )
{
    while ( q != NULL )
    {
        printf ( "%2d ", q -> data ) ;
        q = q -> link ;
    }
}
```

printing data

go to next node.

normal printing.

instead of printing count no. of entries in Stack

No. Of Elements In Stack

```
int count ( struct node * q )
{
   int c = 0 ;
   while ( q != NULL )
   {
      q = q -> link ;
      c++ ;
   }
   return c ;
}
```

Parenthesis in return are optional

Que → whilst using a keyboard
Stack → CPU registers

Data structure – arrangement of data. eg Array.

FAT is an example of a linked list
diff last in linked = NULL →
FAT → FFF.

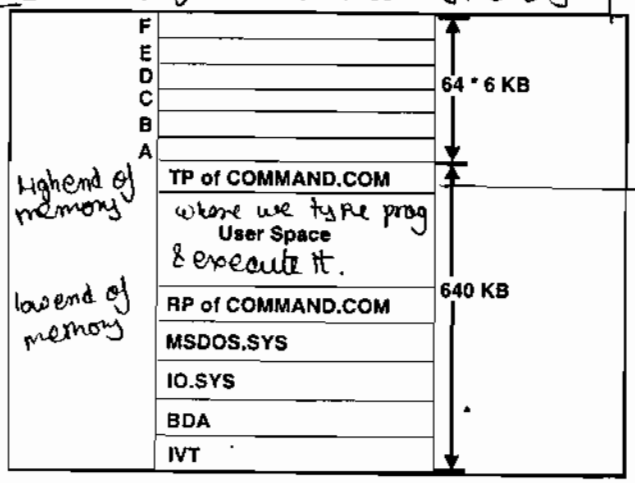
DBMS is also based on linked list.

for bring ptr in a file back to begining of file
rewind(fp).

Size of text file : put cursor at begin. of file.
→ pos = ftell (fp);
Output pos

fseek (fp, 0, SEEK_END) — end of file.
f " (fp, n, SEEK_CUR); — n place/bytes ahead
f " (fp, -n, SEEK_CUR — n place/bytes behind

① Base — 640
② hi memory — 6*64
③ extended — all the rest of memory



→ gets out if User space is req to be more. Gets reloaded when more space is present.

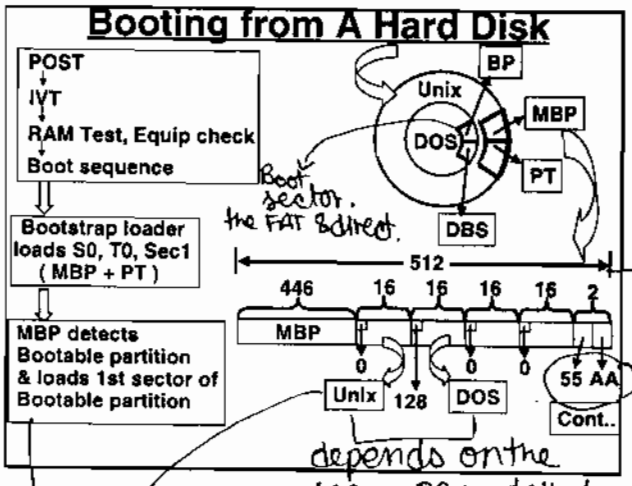
413 > 414 gives add of where RP loads TP. (414 downwards).

for floppy ① format.
for HDD ① partition
② format.

Partition Made for dual OS

Master Boot Program (MBP) detects boot partition.
Partition Table (PT). holds info of partition.

4 OS on 1 hdd (max).
(on floppy no partition as very small thus 2 OS is impossible).



→ (512 as it is a sector)

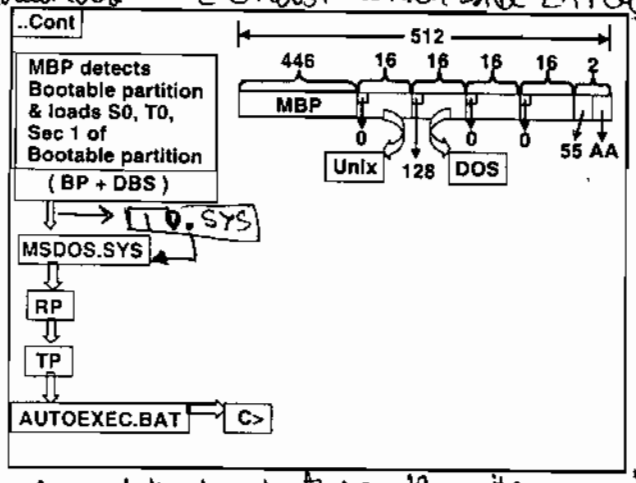
→ signature of partiontable sector
every partition table has this signature

depends on the way OS installed
reason 4 OS only as (16+16+16+16)

Partition is first priority for new HDD. FDisk asks no. of Partitions & size of each.

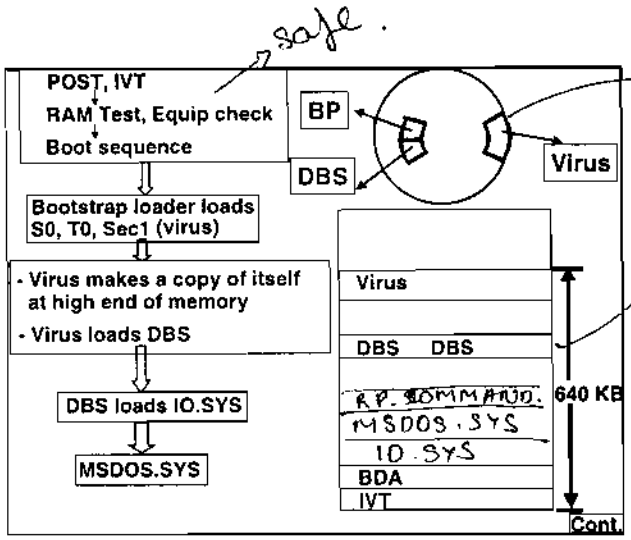
decides which OS should load.

→ info on where Unix partition starts & ends. Which side & track etc.



bios disk does both read & write

→ where every 128 present in either of 4 partitions that loads OS. All others are 0.
If we wanna do it ourselves use bios disk. (Absread will only read dos portion sector).

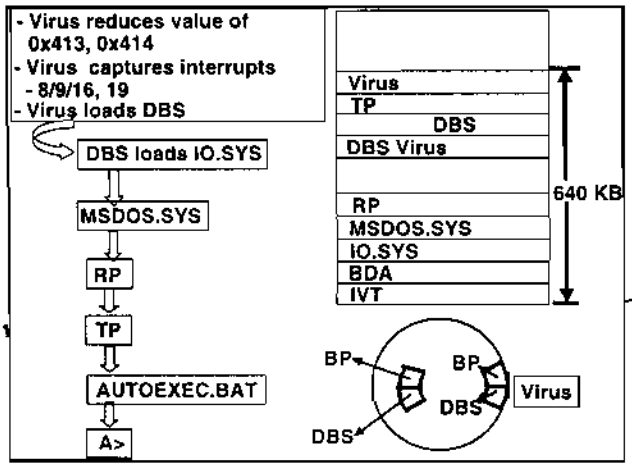


safe.

as virus resiseds in boot sector.

actually here DBS should load but since virus ~~present so virus~~ Now DBS always loads IOS, MSDOSY & gets deleted. So it first loads here makes a copy in himemory & loads DBS back to its work. Before doing so it reduces the actual size of the virus by the size of the virus. It tells DBS this so that the RP can load TP to beloco virus.

Before giving control to DBS it captures control of interrupts (so it is a TSR) - Inter no.19.



Spreading Captures 19 (disk read/writi) now when copy/delete or any other shit the ~~virus~~ Inter no.19 is called. Virus already has it so it abreads the B.P & DBS. Replaces it by Virus source code Then Carries on with normal copy.

Virus FAQs

- Is it safe to use an infected floppy? [redacted] Ok but never boot from that disk
- Can a virus infect C file? → it can but no use as the C file doesnt get execute only opened.
- Would the virus be eliminated if I format the disk? → No. Kill the patient the ~~cir~~ disease will any way go dead.
- Is TSR a virus? All viruses are TSR. → TSR dont executed on its own doesnt spread.
- How do I write a virus? We will never teach you that we will tell u how to make a virus cleaner Ethical Asshole.

WINDOWS

POST
↓
Plug & Play Bios (P&P
↓
Determined Boot sequence.
↓
Boot strap loader
↓
Master Boot Prog
↓
Finds Active (128) Partition
↓
DBS gets loaded [virus in boot sector goes here]
↓
IO.SYS gets loaded
↓
MSDOS.SYS.
↓
message comes starting Win95/98
↓
logo.sys (start up picture).
↓
System.Dat & User.Dat get loaded.
↓
(info comon to all & diff to user eg wall papers)

→ Config.sys loaded (loads device driver prog) (while in DOS) (loads real mode device drivers)
↓
Autoexec.bat → (also can have Virus)
↓
Win.Com (load windows into memory) desktop papers show up. — DOS↑
↓
Vnm32.vxd (Virtual machine manager). (Virtual devic drive)
↳ also loads drivers which are used under Win (loads Protected mode drivers)
↓
Kernel32.dll, gdi32.dll, user32.dll (dynamic linke libraries)
(is kernel for Win) (graphical device interphase).
↓
Explorer.exe
↓
Network login dialog (when it asks for pass & username & network type).
↓ ⊗

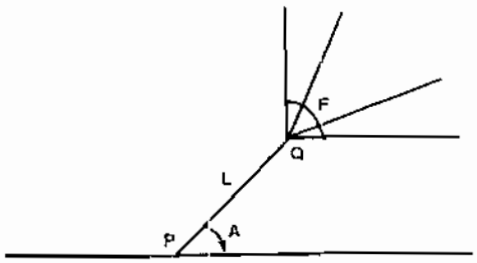
32 bit operation.

Present in registry ↖ Run key (the same function as autoexec.bat. loads the the windows Registry)
↓
now desktop shows up.
↓
Explorer error
Please restart

anything that can be expressed in terms of itself.

Anay Lal
Sheet 27

Tree Generation



F = is fanning angle. (spreadout angle).
A = b/w gnd & stem.

Tree Generation

```
main( )
{
   int gd=DETECT, gm ;
   int drawtree ( int x1, int y1, float a, float l, float f, int n ) ;
   initgraph ( &gd, &gm, "c:\\tc\\bgi" ) ;
   drawtree ( 280, 350, 270, 75, 80, 7 ) ;
   getch( ) ;
   closegraph( ) ;
   restorecrtmode( ) ;
}
drawtree ( int x1, int y1, float a, float l, float f, int n )
{
   int i, numbranch = 2, x2, y2 ,
   float delang, ang, lenratio = 0.75, spreadratio = 0.8 ,
   if ( n > 0 )
   {
      x2 = x1 + l * cos ( 0.0174 * a ) ;
      y2 = y1 + l * sin ( 0.0174 * a ) ,
      setcolor ( WHITE ) ,
      line ( x1, y1, x2, y2 ) ,
                                                    Contd...
```

degree of tree → ~~no. of branches~~

x_1, y_1, A Leng of branc, F

in radians

$x_2 = x_1 + l\cos\theta$
$y_2 = y_1 + l\sin\theta$

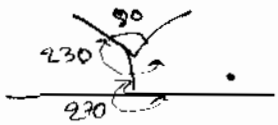
as V if M

~~length of current branch with bra~~

ratio of current with original

reducing the length of branch

fanning angle.




```
...Contd.      if ( numbranch > 1 )
                  delang = f / ( numbranch -1.0 ) ;
               else
                  delang = 0 ;

               ang = a - f / 2.0 - delang ;
               for ( i = 1 ; i <= numbranch ; i++ )
               {
                  ang += delang ;
                  drawtree ( x2, y2, ang, l * lenratio,
                              f * spreadratio, n - 1 ) ;
               }
            } // end of if
            else
            {
               setcolor ( random ( 7 ) + 1 ) ;
               ellipse ( x2, y2, 0, 276, 2, 4 ) ,
               fillellipse ( x2, y2, 2, 4 ) ;
            }
         } // end of main( )
```

delta angle (delang)

deduction in length

deduction in no. of branches.

deduction in angle

leaves

draw tree.

all figs follow some basic format.

keep drawing 'V's till a leaf is got.

Axay Lal
26/09/2001

Compression

```
# include "stdio.h"
FILE *fs, *ft ;
main( )
{
   char s[ 67 ], t[ 67 ] ; char  ch ;
   printf ( "Enter source" ) ;
   gets ( s ) ;
   printf ( "Enter target" ) ;
   gets ( t ) ;
   fs = fopen ( s, "r" ) ;
   if ( fs == NULL )
   {
      printf ( "Unable to open" ) ;
      exit( ) ;
   }
   ft = fopen ( t, "w" ) ;
   if ( ft == NULL )
   {
      printf ( "Unable to open" ) ;
      fclose ( fs ) ; exit( ) ;
   }
   puts ( "Compress/
         Decompress C/D" ) ;
   ch = getche( ) ;
   switch ( ch )
   {
      case 'C' :
      case 'c' :
            compress( ) ;
            break ;
      case 'D' :
      case 'd' :
            decompress( ) ;
            break ;
   }
   fclose ( fs ) ;
   fclose ( ft ) ;
}
```

reduction of space

Compression

```
compressfile( )
{
   int  count, ch ;
   while ( ( ch = getc ( fs ) ) != EOF )
   {
      if ( ch == ' ' )
      {
         count = 1 ;
         while ( ( ch = getc ( fs ) ) == ' ' )
            count++ ;
         putc ( count + 127, ft ) ;
      }
      putc ( ch, ft ) ;
   }
}
```

I ___ am __ at _ Nagpur
⇩
I δ am φ at θ Nagpur

~~get~~ ascii of value say x
$127 - x$ = no. of spaces (decomp)
get a space, increment count, $\boxed{127 + count}$ is the ascii of value.

$\boxed{\text{Count} = \text{no. of spaces}}$

Decompression

```
decompressfile( )
{
   int  count, ch ;
   while ( ( ch = getc ( fs ) ) != EOF )
   {
      if ( ch > 127 )
      {
         ch = ch - 127 ;
         for ( count = 1 ; count <= ch ; count++ )
            putc ( ' ', ft ) ;
      }
      else
         putc ( ch, ft ) ;
   }
}
```

Huffman Coding Algorithm

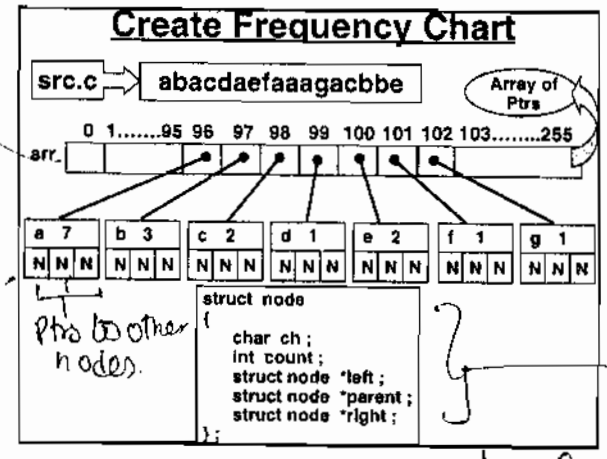
used by WinZip

- Variable length codes for each character
- More the frequency, smaller the code
- Unique prefix attribute

011 = A
0110 ≠ B
↳ prefix is never common.

is an array of structures (each structure has the value it frequency & 3 ptrs).

Create Frequency Chart

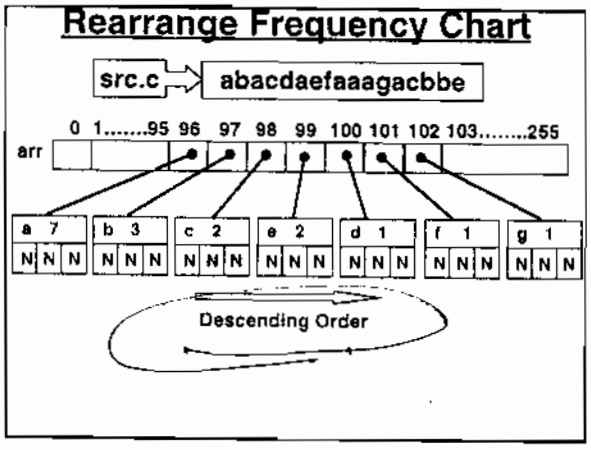


in the 96th slot store frequency of A. (ASCII of A = 97) 97 cos 0 → 96 = 97th slot.

used to design the no. of time a value occurs + 3 (ptrs to other nodes)

no. of times a occurs in the file..

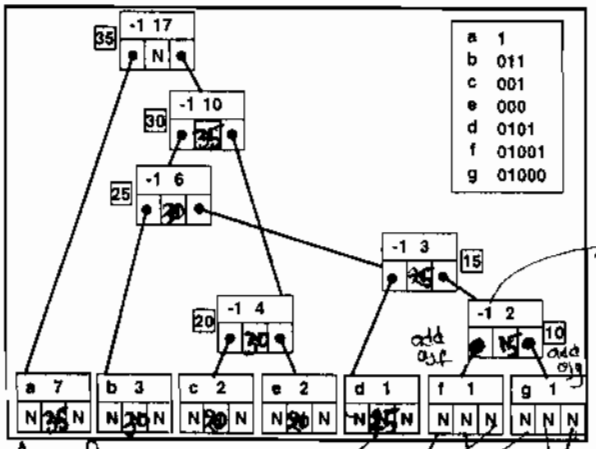
Rearrange Frequency Chart



Logic :

N N N → child
↓
parent

f & g N 10 N
d & (f+g) N 15 N



↑ leaf nodes.

denotes that it is not a leaf node.

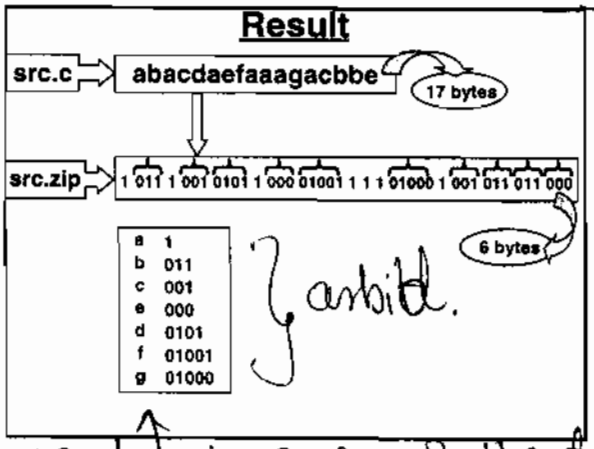
-1 cos no -1 in the arr (0-255)

add of Child node

→ this N will be 15

Ptr to parent node (the one above it containing its sum)

① travel right to left.
f, & g,
② new node.
③ now 2<3 ∴ 2 & other 2 get new node.
④ now 3<4 ∴ 3 + previous 3 get new node
⑤ ∴ do all leave nodes then stop.



replace a by 1
b by 011

at the end of the zip the specific values (frequency) of that file must be stored.

read ↑ in array & start replace. Use bitwise Operator

max frequency → lesser byte size.
prefix always unique.

Start from top
when moving left = 1
right = 0.

for a
① from top N move left
①
② for b top N move right left & left = 011.
③ right left right right right
0 1 0 0 0

Popup Menus

```
#include <stdlib.h>
#include <stdio.h>
#include <alloc.h>
char *menu[ ] =
        {
          " Mumbai   ",
          " Delhi    ",
          " Chennai  ",
          " Nagpur   ",
          " Pune     ",
          " Bangalore "
        } ;
char far *s = 0xB8000000 ;

main( )
{
   int choice, x, y ;
   char ch = 'y', far *p ;
   clrscr( ) ;
   randomize( ) ;
   draw_bkgnd( ) ;
   p = malloc ( 6 * 11 * 2 ) ;

   while ( ch == 'y' || ch == 'Y' )
   {
      x = random ( 15 ) ;
      y = random ( 60 ) ;
```

or p=(char*) malloc(6* 11* 2)

longest name

6 menu items

2 bytes in VDU memory

gives the size of the total menu.

base add of menu, no of items in menu, begining at x, y collected in char** as menu it a menu*

```
      savevideo ( x, y, x + 6, y + 11, p ) ;
      displaymenu ( menu, 6, x, y ) ;

      choice = getresponse ( menu, "MDCNPB", x ,y ,6 ) ;

      gotoxy ( 5, 24 ) ;
      printf ( "You selected %d %s\t\tPress 'Y' to continue
               and 'N' to exit" , choice , menu[choice-1] ) ;

      restorevideo ( x, y, x + 6, y + 11, p ) ;
      ch = getch ( ) ;
      erase_message( ) ;
   }
   free ( p ) ;
}
```

stores in memory in p which is allocated by malloc

x+6, y+11

display menu, hotkeys, from x, y, 6 elem

bottom helpmate.

restores the previous screen back on VDU.

erase message at bottom.

user defined

```
draw_bkgnd ( )
{
   int row , col ;
   for ( row = 0 ; row <= 21 ; row++ )
      for ( col = 0 ; col <= 79 ; col++ )
         writechar ( row, col, ' ', 145 ) ;
}

displaymenu ( char **menu, int count, int sr, int sc )
{
   int i ;
   for ( i = 0 ; i < count ; i++ )
      writestring ( menu[i], sr++, sc, 112 ) ;
}

erase_message( )
{
   int col ;
   for ( col = 0 ; col <= 79 ; col++ )
      writechar ( 22, col, ' ', 10 ) ;
}
```

writes the colour in blue display a space at r & c

writes directly to VDU memor

writes menu(string) at row & coll in (colour no 112).

erase char by displaying space in some colour

```
writestring ( char *s, int row, int col, int attr )
{
   while ( *s )
      writechar ( row, col++, *s++, attr ) ;
}
```

row col colour

writes string to vsu.

```
writechar ( int r, int c, char ch, int attr )
{
   char far *v ;
   v = s + r * 160 + c * 2 ;
   *v++ = ch ;
   *v = attr ;
}
```

writing to

v++ ;

for colour

*v = ch ;
v++ ;
*v = attr

```
getresponse ( char **menu, char *hotkeys, int r, int c,
              int count )
{
   int choice = 1 , hotkeychoice = 1 , ascii , scan ,i;
   char *s ;

   while ( 1 )
   {
      writestring ( menu[ choice - 1 ], r + choice - 1, c, 60 ) ;
      while ( !kbhit( ) )
         ;
      ascii = getch( ) ;
      if ( ascii == 0 )
         scan = getch( ) ;

      writestring ( menu[choice-1] , r + choice - 1, c, 112 ) ;
      if ( ascii == 0 )
      {
         if ( scan == 80 )
         {
```

waits till keyboard is touched.

**

base add of string, row & col, 60

colour no

from disp menu
we print the whole menu in colour blue
now we re-print only one string in a diff colour so it looks high lighted

choice = 1

menu[0], r+0, c

after moving restoring the first item back to its normal state.

```
      choice++ ;
      if ( choice > count )
         choice = 1 ;
   }

   if ( scan == 72 )
   {
      choice-- ;
      if ( choice == 0 )
         choice = count ;
   }
}
else
{
   if ( ascii == 13 )
      return ( choice ) ;

   s = hotkeys ;
   hotkeychoice = 1 ;
```

ascii = 0 then arrow keys
ascii = 13 then enter

```
         while ( *s )
         {
            if ( ascii == *s )
               return ( hotkeychoice ) ;
            hotkeychoice++ ;
            s++ ;
         }
         printf ( "\a" ) ;
      }
   }
}
```

hotkeychoice ~~returns~~ acts as a counter of hotkey.

default.

add of base of hotkey.

s = starting r = row
e = ending c = col.

store values in buffer

```
savevideo ( int sr, int sc, int er, int ec, char *buffer )
{
   char far *v ;
   int i, j ;

   for ( i = sr ; i <= er ; i++ )
   {
      for ( j = sc ; j <= ec ; j++ )
      {
         v = s + i * 160 + j * 2 ;
         *buffer = *v ;
         v++ ;
         buffer++ ;
         *buffer = *v ;
         buffer++ ;
      }
   }
}
```

ascii values & colours will be stored.

ascii

Colour

*buffer++ = *v++;
*buffer++ = *v;

VDU → buffer

```
restorevideo ( int sr, int sc, int er, int ec, char *buffer )
{
   char far *v ;
   int i, j ;
   for ( i = sr ; i <= er ; i++ )
   {
      for ( j = sc ; j <= ec ; j++ )
      {
         v = s + i * 160 + j * 2 ;
         *v = *buffer ;
         v++ ;
         buffer++ ;
         *v = *buffer ;
         buffer++ ;
      }
   }
}
```

opp of previous

ascii

colour

buffer → VDU.

Why OO

Structured Programming
Ex. C, Pascal, Ada

Features:
- 4 control Instructions
- Functional Abstraction
- Local Variables

Limitations:
- No reuse
- No importance to data
- Poor real world model

Object Oriented Programming
Ex. C++, Java, Eiffel, SmallTalk

Features:
- Reuse
- Data equally important

- 1980
- AT&T Bell Labs
- Bjarne Stroustrup

OOP Cornerstones

- Encapsulation - Hides complexity
- Inheritance - Promotes reuse
- Polymorphism - Generalizes actions

Facilitated by Classes and Objects

C++

Structures

```
struct emp
{
  char n[20] ; int a ; float s ;
} ;
```

Structure Name/Tag

Structure Members Elements

struct	emp	e1, e2,	e[20],	*p ;
keyword	structure name	structure variables	array of structures	pointer to a structure

Accessing Structure Elements

```
struct emp
{
   char n[20] ; int a ; float s ;
} ;
emp e1 = { "Ajay", 33, 6000 } ;
cout << e1.n << e1.a << e1.s ;
emp *p ;
p = &e1 ;
cout << p -> n << p -> a << p -> s ;
```

cout - console output
<< - Insertion operator

struct is optional

Overloaded operator

Structure Operators

struct is optional

```
cout << e1.n << e1.a << e1.s ;
printf ( "%s %d %f", e1.n, e1.a, e1.s ) ;
```

Are Structures Different in C++

C → Collection of Data

C++ → Collection of Data & Funs. → Classes

Classes In C++

```
# include <iostream.h>
class a
{
   int i ;
   float j ;
} ;
void main( )
{
   a z = { 10, 3.14 } ;
   cout << z.i << z.j ;
}
```

Class members are by default private

Making It Work

```
# Include <iostream.h>
class a
{
  public :
     int i ; float j ;
} ;

void main( )
{
   a z = { 10, 3.14 } ;
   cout << z.i  << z.j ;
}
```

Still Better Way...

```
class a
{
  private :
     int i ; float j ;
  public :
     void setdata ( int ii, float jj )
     {
        i = ii ; j = jj ;
     }
} ;
void main( )
{
  a e1, e2 ;
  e1.setdata ( 10, 3.14 ) ;
  e2.setdata ( 100, 2.5 ) ;
}
```

Encapsulation at work

e1	
i	j
10	3.14

e2	
i	j
100	2.5

Constructors

```
class a
{
  private :
     int i ; float j ;
  public :
     a ( int ii, float jj )
     {
        i = ii ; j = jj ;
     }
     a( )
     {
        i = 0 ; j = 0.0 ;
     }
} ;
void main( )
{
  a e1 ( 10, 3.14 ) ;
  a e2 ;
}
```

2 Argument Constructor

Overloaded Functions

0 Argument Constructor

e1	
i	j
10	3.14

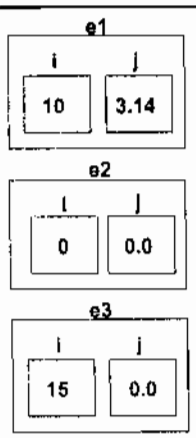
e2	
i	j
0	0.0

Either I do it or you do it

Two In One

```
class a
{
  private :
     int i ; float j ;
  public :
     a ( int ii = 0, float jj = 0.0 )
     {
        i = ii ; j = jj ;
     }
} ;
void main( )
{
  a e1 ( 10, 3.14 ) ;
  a e2 ;
}
```

What if
a e3 (15) ;




```
#include <iostream.h>
class comp
{
  private :
     double r, i ;
  public :
     comp ( double rr = 0,
            double ii = 0)
     {
        r = rr ;
        i = ii ;
     }
     void print( )
     {
        cout << r << i ;
     }
```

Return Type

2 Arg Constructor

c = a.operator + (b) ;

```
comp operator + ( comp c2 )
{
   comp t ;
   t.r = r + c2.r ;
   t.i = i + c2.i ;
   return t ;
}
} ;
void main( )
{
comp a ( 1.0, 1.0 ) ;
comp b ( 2.0, 2.0 ) ;
comp c ;
c = a + b ;
cout << "c = " ;
c.print( ) ;
}
```

References

```
# include <iostream.h>
struct data
{
   int i ; float f ;
} ;
class emp
{
  public :
     void fun1 ( data x )
     {
        x.i = 2 ; x.f = 5.5 ;
     }
     void fun2 ( data *y )
     {
        y->i = 2 ; y->f = 5.5 ;
     }
     void fun3 ( data &z )
     {
        z.i = 3 ; z.f = 10.5 ;
     }
} ;
void main( )
{
   data d = { 1, 2.2 } ;
   emp e ;
   e.fun1 ( d ) ;
   cout << d.i << d.f ;
   e.fun2 ( &d ) ;
   cout << d.i << d.f ;
   e.fun3 ( d ) ;
   cout << d.i << d.f ;
}
```

Changing a reference changes referent

Which Is Better

```
# include <iostream.h>
struct emp
{
   char n[20] ; int a ;
} ;
class ex
{
  public :
     void fun1 ( data d )
     {
        cout << d.n << d.a ;
     }
     void fun2 ( data *d )
     {
        cout << d->n << d->a ;
     }
     void fun3 ( data &d )
     {
        cout << d.n << d.a ;
     }
  } ;
  void main( )
  {
     emp e = ( "amol", 21 ) ;
     ex  z ;
     z.fun1 ( e ) ;
     z.fun2 ( &e ) ;
     z.fun3 ( e ) ;
  }
```

Are References Necessary

```
class  comp
{
  public :
     comp operator + ( comp c2 )
     {
        comp  t ;
        t.r = r + c2.r ;
        t.i = i + c2.i ;
        return t ;
     }
} ;
void main( )
{
   comp a ( 1.0, 1.0 ) ;
   comp b (2.0, 2.0 ) ;
   comp c ;
   c = a + b ;
}
```

How to avoid a copy?

Use Reference

Use Pointer

Inheritance

```
# include <iostream.h>
class  index
{
  private :   protected :
     int  count ;
  public :
     index( )
     {
        count = 0 ;
     }
     void display( )
     {
        cout << count ;
     }
     void operator ++ ( )
     {
        count ++ ;
     }
} ;
```

```
class  index1: public index
{
  public :
     void operator --( )
     {
        count -- ;
     }
} ;          index1  i ;
void main( )
{
     index  i ;
     i.display( ) ;
     i++ ;
     i.display( ) ;
     i-- ;
     i.display( ) ;
}
```

```
#include <iostream.h>
class shape
{
   public :
      virtual void draw( )
      {
         cout << "shape" ;
      }
};

class circle : public shape
{
   public :
      void draw( )
      {
         cout << "circle" ;
      }
};
```

```
class rectangle : public shape
{
   public :
      void draw( )
      {
         cout << "rectangle" ;
      }
};
void main( )
{
   shape *p ;
   circle o1 ;
   rectangle o2 ;
   p = &o1 ;  // no error
   p -> draw( ) ;
   p = &o2 ;
   p -> draw( ) ;
}
```

Pure Virtual Functions

```
#include <iostream.h>
class shape
{
   public :
      virtual void draw( ) = 0 ;
};

class circle : public shape
{
   public :
      void draw( )
      {
         cout << "circle" ;
      }
};
```

```
class rectangle : public shape
{
   public :
      void draw( )
      {
         cout << "rectangle" ;
      }
};
void main( )
{
   circle c1, c2, c3, c4, c5 ;
   rectangle r1, r2, r3, r4, r5 ;
   shape *p[10] = { &c1, &r4, .. }
   for ( int i = 0 ; i <= 9 ; i++ )
      p[ i ] -> draw( ) ;
}
```


Ric compression, interrupts,

Institute Of Computing & Information Tech.

44-A, Gokulpeth, Nagpur-10, India. Ph: 531046, 545322

STORAGE CLASSES

STORAGE CLASS	STORAGE	DEAFAULT INITIAL VALUE	SCOPE	LIFE
Automatic	Memory	Garbage	Local to the block	Till control remains in the block in which the variable is defined
Register	CPU Registers	Garbage	Local to the block	Till control remains in the block in which the variable is defined
Static	Memory	Zero	Local to the block	Variable persists between different function calls
External	Memory	Zero	Global	Till the execution of the program doesn't come to an end

PREPROCESSOR DIRECTIVES

STATEMENT	DESCRIPTION
# define id text	*text* will be substituted for *id* wherever it later appears in the program; if construct *id (a1, a2, ...)* is used, arguments *a1, a2, ...* will be replaced where they appear in text by corresponding arguments of macro call.
# if expr #endif	If constant expression *expr* is TRUE, statements up to *#else* will be processed, otherwise they will not be
# if expr # else # endif	If constant expression *expr* is TRUE, statements up to *#else* will be processed, otherwise those between the *#else* and *#endif* will be processed.
# ifdef id # endif	If *id* is defined (with *#define* or on the command line) statements up to *#endif* will be processed, otherwise they will not be processed (optional *#else*).
# ifndef id . . # endif	If *id* has not been defined, statements up to *#endif* will be processed (optional *#else* construct).
# include "file" or # include <file>	Inserts contents of *file* in program. Double quotes mean look first in same directory as source program, then in the include directory; corner brackets mean only include directory.
# undef id	Remove definition of *id*.
# pragma inline	Tells the compiler that the source program contains inline assembly language statements.

Notes: Preprocessor statements can be continued over multiple lines provided each line to be continued ends with a backslash character (\) Statements can also be nested

ARRAYS

A single-dimensional array *aname* of *n* elements of a specified type and with specified initial values (optional) is declared with:

```
type aname[n] = { val1, val2, ... } ;
```

If complete list of intial values is specified, *n* can be omitted. Char arrays can be initialised by a string of chars in double quotes. Valid subscripts of the array range from *0* through *n-1*. Multi dimensional arrays are declared as:

```
type aname[n1] [n2].. .= { init_list } ;
```

Values listed in the initialization list are assigned in 'dimension order' (i.e. as if last dimension were increasing first). Here are some examples:

```
char name[ ] = "anil"; /* char array */
/* array of char pointers */
char *days[ ] = { "Sun", "Mon", "Tue" };
int mat[2][4] = {
                 { 2,4, 5, 8},
                 {-5,0,4,7 }
               }; /* 2 x4 array of ints */
struct emp e[100] ; /* array of structs */
```

STRUCTURES

A structure *sname* of specified elements is declared with a statement of the form

```
struct sname
{
    element_declaration ,
    element_declaration ;
} ,
struct sname variable _list ;
```

Each *element_declaration* is a type followed by one or more member names. If structure declaration and variable list definition are combined then *sname* is optional. Elements of a structure are accessed using . operator or -> operator. If structure elements are to be accessed using a structure variable use the operator and if they are to be accessed using a structure pointer then use the -> operator. Example:

```
struct emp
{
    char n[20] ; int age ;
} ;
struct emp e1 = { "Anil", 30 }, *e2 = &e1
printf ( "\n%s %d", e1.n, e2.age ) .
printf ( "\n%d", e2 ->n, e2 ->age )
```

FUNCTIONS

Functions follow this format

```
ret type name ( type arg1, type arg2, .)
{
    local var_declartions
    statement ,
    statement ;
    return value .
}
```

ret_type is the return type for the function and can be *void* if the function returns non value or omitted if it returns an *int*. If the functions receives or returns some--thing other than an *int* it is necessary to mention its prototype.

THIS SPACE IS DEDICATED IN MEMORY OF THOSE WHO DO NOT USE THIS CARD REGULARLY!!

UNIONS

A union permits us to access the same locations in memory in more than one way For Example:

```
union a
{
    int i ; char ch[2] ,
} ;
union a z ;
```

The 2 bytes occupied by this union can be accessed as a two byte entity by saying *z.i* and as individual bytes by saying *z.ch[0]* and *z.ch[1]*.

Note: union cannot be initialized

typedef

typedef is used to give a new name to data type. For example, in the following code struct employee has been renamed as EMP.

```
struct employee
{
    char name[20] ; int age .
} ,
typedef struct employee EMP
```

EDITING COMMANDS

CURSOR MOVEMENT

Arrow keys

Ctrl f	Next word
Ctrl a	Previous word
Ctrl Home	Top of screen
Ctrl End	Bottom of screen
Ctrl PgUp	Beginning of file
Ctrl PgDn	End of file

DELETION

Del	Character at cursor
Backspace	Character before cursor
Ctrl t	One word
Ctrl y	One line

BLOCK COMMANDS

Ctrl kb	Mark beginning of block
Ctrl kk	Mark end of block
Ctrl kc	Copy marked block
Ctrl kv	Move marked block
Ctrl kw	Write block to file
Ctrl ky	Delete marked block
Ctrl kh	Hide marked block

F1	Help
Ctrl F1	Help on functions
F2	Saves current file
Ctrl F9	Compiles and runs program
F6	Switches between edit and other windows
Alt F7	Shows previous error
Alt F8	Shows next error
F3	Loads file
Alt x	Quits out of TC/BC
F5	Zooms and unzooms current window
F7	Runs program in debug mode with trace
F8	Runs program in debug mode without trace
Ctrl F8	Sets or clears break point
Ctrl F7	Adds watch on variable/expression
F9	Compiles a program, creates .EXE file
Alt F5	Displays output screen
Alt F9	Compiles a program, creates .OBJ file
Ctrl F4	Evaluates a C expression
Alt F3	Picks a file for loading from recently used files