Date
Page

Notes Made By

- Riti Kumari

Progeramming with C and C++

Getting started with C

Introduction to C

C was developed in 1972 at Bell Labs by Dennis Ritchie

Unix operating system is written in C language

We choose C because many languages are derived from C progeramming language.

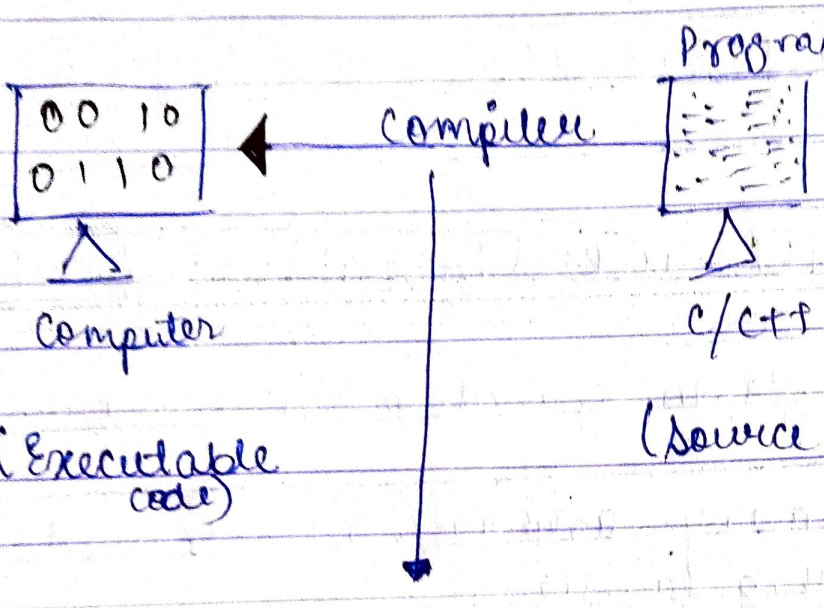
Use of C

1) windows OS is build using C & C++.
2) A part of fb appln. is written in C.
3) A part of photoshop is written in C++.

Data structure - The way we organise data
Algorithm - steps

Software requirement

Compiler + IDE (Platform for writing program & testing it)



A software which reads source code & converts to executable code.

Compiler → GCC compiler (GNU)
IDE → Eclipse IDE (is built using Java)

Java → JDK(JRE)

Creating first C-project

Workspace- where projects would be saved.

File → New → C/C++ Project → C/C++
or
Project

Next ← C project

Executable Project ← Project name

Hello World
ANSI C
Project

MinGW gcc. → Finish.

```
#. include <stdio.h>

int main()
{
  puts ("Hello World!!");

  return 0;
}
```

Exploring Eclipse IDE

for theme: Window → preferences → general → appearance.

appearance → Select your theme (Light theme) → apply

File → Restart

Window of Eclipse → Workbench.

Window - Show view → Project explorer

↓

learn-programming.c ← src

Source file → present in the source window.

file extension → .c

Problems | Tasks | Console | Properties

↓

Output

Write and Explore your first C Program

```
# include < stdio.h>
  int main() → entry point
    {
  printf (" Hello Internshala!"); → String (within double quotes)
  printf (" I am here to teach you");
    return 0;
  }
```

Run your program

Build project → Run → Local C/C++ appl^n.

O/p console.

Hello Internshala.

For new line "\n"

Replacing printf by puts

The puts() prints the text enclosed within double quotes by appending a new line character in the end.

Comments in C programming

// message

dont include comment before any code.

For multiple line comments

```
/*  - - - -
  - - - -
  - - - - */
```

Structure of C program

```
/* Documentation
              */   (comments)

// link section.  (header files) # include<stdio.h>
   // global declaration.
int main()  // function  // entry point
  {

  // Program body.

  }
  // sub-program
```

Tokens in C

```
  char size = 'M';
   ↓     ↓     ↓
datatype variable literal

int main()
  {
```

```
char name[25] = "Man is great";
char size = 'M';
int personItServes = 2;
float price = 365.8;
```

Format Specifiers

```
printf("%s", name);
printf("%c", size);
printf("%d", personItServes);
printf("%f", price);
```

"%.1f" → 365.8

"%.2f" → 365.79

"%f" → 365.799988

after rounding off

How Data is stored in a Token

Datatype

- primary
  - char
  - int
  - long
  - float
  - double
  - (integer type: char, int, long; floating point type: float, double)
- derived
  - Array
  - pointer
  - String
- User defined
  - Structure
  - Union
  - Enum

Type	Size (bytes)	Example	Format specifier
char	1	'e' '$' '#' '@' '2'	%c
int	2 or 4	2766, -3210	%d
float	4	3.12, -2.547219	%f
long	4 or 8	-2147483648, 4276878	%ld
double	8	2.612676267	%lf

Allocated memory varies from machine to machine

Variable → name of memory location. Also known as identifier

Datatype → type of data that can be stored

literal → constant value.

Constants in C.

```
#include <stdio.h>
int main()
{
const float pi = 3.14;   // convention to use capital letter for constant value.
```

const → value doesn't change.

```
int radius = 27;
float area = pi * radius * radius
printf ("The area of circle is %.2f", area);
return 0;
}
```

Constant - It is the name of the memory location where a fixed value is stored.

Reserved keywords.

In C there are 32 keywords.

Defining Tokens

* Everything in C except white space.

Eg - Variables, datatypes, literals & Constants

User Input
I/O operations
Taking input & giving output
(console)

Input data

i3 i2 i1

↓

input stream

↓

Standard input
Std in

Output data

↓

03 02 01

↓

output stream

↓

standard output
Stdout.

Taking user input <stdio.h>

char size;

scanf ("%c", &size)

↓

address of

Improving user experience.

#include <unistd.h>

usleep (100000); → Usually stops the program for seconds.

↓

1 million microsec
(1 sec)

Newline ('\n') → Enter key

char name[30];

scanf ("%.[^\n]%*c", name);

↓ ↳ for string we don't use &.

User input untill enter key is pressed.

(it stores string characters unit '\n' is pressed)

%.[^\n]%*c → it could be used.

but it can be replaced by %s.

(it stores string characters untill white space.)

Input operation

size = getchar();

Output operation

putchar(size)

↓

only a single char value.

Operators

↓

Symbols to perform operations (*, +, -, %, /, etc)

10 + 20

↓

operator

operand.

sizeof(int)

↓ ↓

operator operand.

Types of operators

1) Arithmetic
2) Relational
3) Logical
4) Increement & decrement
5) Assignment
6) Special operator

Asithmetic

(+, -, *, /, %)

3 % 2 = 1

x = 12
y = 4

\+ = 16
\- = 8
* = 48
/ = 3
% = 0.

Precedence & Associativity

( ) has the highest priority

Precedence (+ / %) > (+, -)

Asso → Left to right

PDMAS.

→ left to right

a = 3+1 - 7 * 5 %. 3 /2

= 3+1 - 35 %. 3 /2

= 4 - 2/2

= 4 - 1

= 3.

Relational Operators in C

>, >=, <, <=, ==, !=

```
int x = 2;
int y = 3;
int z = 3;

x > y;
```

o/p.

True → 1 False → 0

Logical operators

&&, ||, !

AND

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

OR

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

NOT

A	!A
0	1
1	0

Associativity : left to right

Assignment operator

=, +=, -=, *=, /=, %=

Associativity : Right to left

bill = bill + task

2 ← 1

Increament & decrement

Postfix	Prefix
a++	++a
a--	--a

```
int s=10
int ns= s++

ns = s ;
s = s+1;

ns= 10
s = 11
```

```
int s=10 ;
int ns=++s;

s = s+1;
ns = s;

ns= 11
s= 11
```

```
int age = 10;
```

	O/P
pf("%d", age++);	age = 10
pf("%d", age);	age = 11
pf("%d", ++age);	age = 12

pf("%d", age++ + ++age); left to right

$= 12 + (++13)$

$= 12 + 14$

$= 26$

age = 14

Special Operators

sizeof()	Returns size of memory location
address of &	Returns the address of memory loc
*	Pointer to a variable

```
int n = 8.
```

pf("%d", (n++) - (++n) + (--n));

$8 - (++8) + (--n)$

$8 - 9 + (--9)$

$8 - 9 + 8$

$16 - 9 = 87$

int r = 3 * 10 - 2 % 5 / 2

= 30 - 2 % 5 / 2

= 30 - 2 / 2

= 30 - 1

= 29

2) 5 (2
-4
1

2 % 5

int n = 100 % 3

= 1

5) 2 (0.
-0
2

3) 100 (33
-9
10
-9
1

int r = 3 * (10 - 2) % 5 / 2

= 3 * 8 % 5 / 2

= 24 % 5 / 2

= 4 / 2

= 2

5) 24 (4
20
4

age = age * num;

10 > 10 && 15 >= 10 || 10 == 10

0 1

0 1 1 1

Conditional statements

i) If - else ii) Ternary iii) Else-if

iv) switch v) break.

1) If - else

Apply 10% if bill u >= 500.

```
if (condition)
    {
    Execute statement
    }
else
    {
    Execute statements
    }
```

2) Ternary operator (Only single expression & single statement)

condition ? exp1 : exp2.

```
payable = retailPrice >=500 ? retailPrice*0.9 : retailPrice ;
```

Else-if statements

```
if (condn)
  {
  }
else if (condn)
  {
  }
else
  {
  }
```

WAP to find out grade of a student based on marks scored in exam.

```
if (marks > 500)
  {
pf("invalid data")
  }
else if ( marks >= 450 && marks <= 50)
  {
    grade = 'A';
  }
else if
  {


  }
```

Switch statements

```
# include <stdio.h>
int main()
{
int rating;
puts ("How likely you would like to rate
      our pizza on a scale of five");

puts ("Reply with a number);

~~sf~~ ("enter your rating:");
sf ("%d", &rating);

switch (rating)
{
case 0:
                    break;
case 1:
                    break;
case 2:
                    break;
case 3:
                    break;
case 4:
                    break;
case 5:
                    break;
default:
```

In switch statement we can only use int, char & long.

Algorithm & Flowchart.

↓

Step by step approach to solve any question

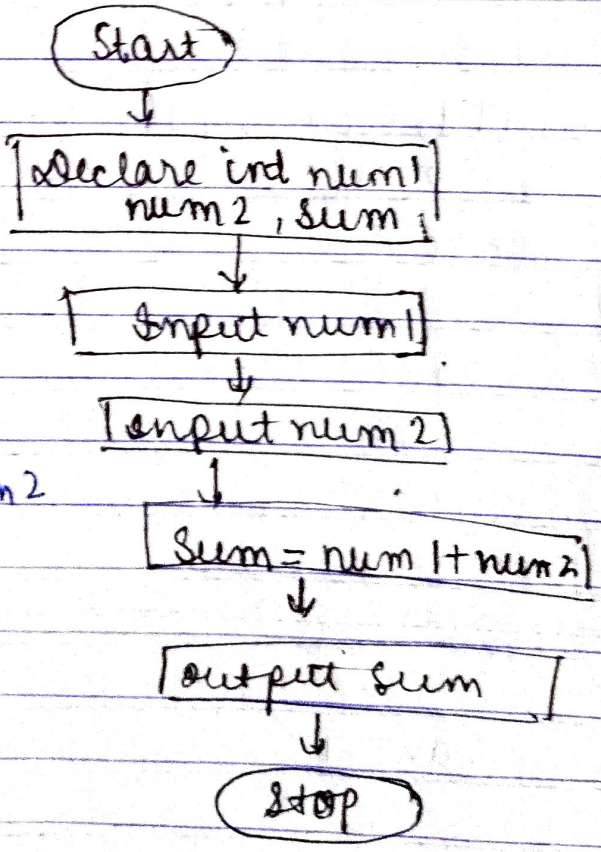
Flowchart

Algorithm → Flowchart → Code

Algorithm

1) Declare integers num1, num2 & sum.
2) Input num1
3) Input num2
4) Compute sum = num1 + num2
5) Output sum.

Flowchart



Code

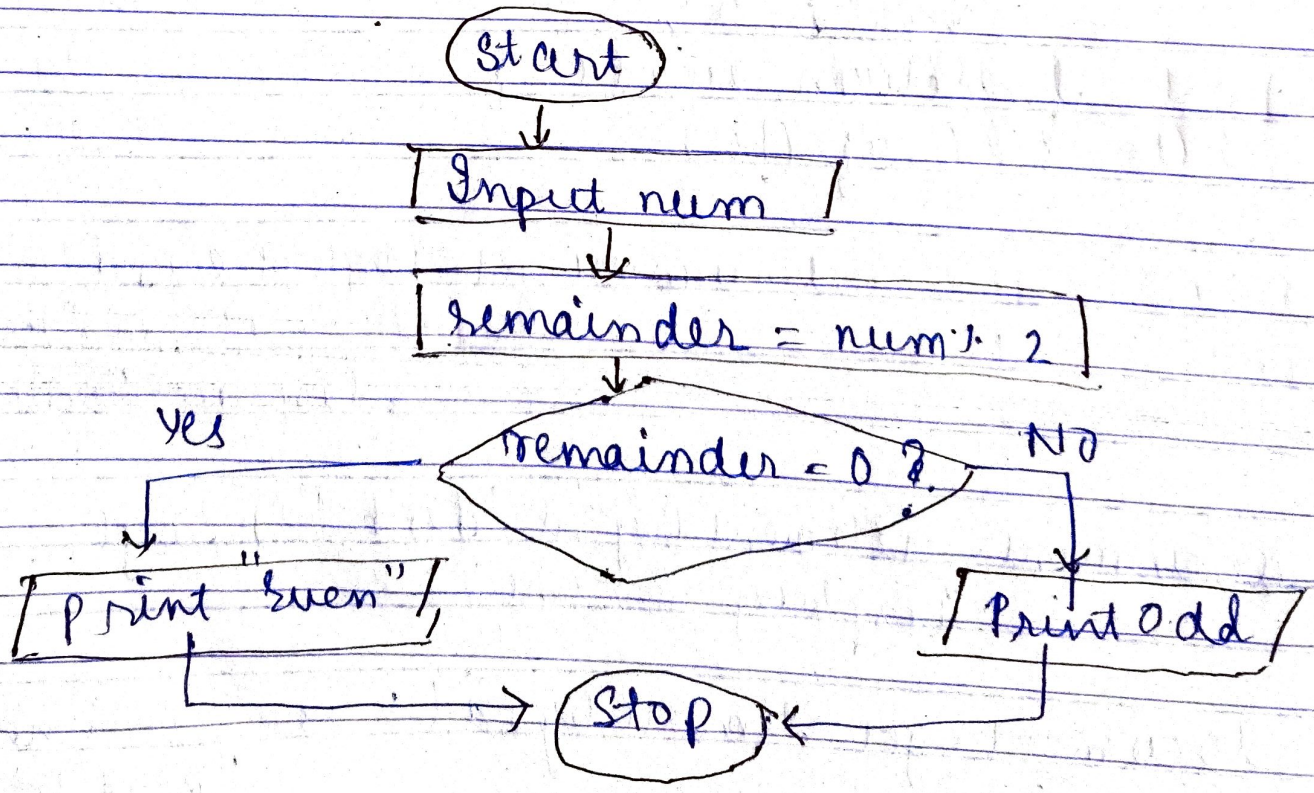
Flowchart → graphical representation of algorithm.

Oper^n → rectangle
start/stop → oval.
input → parallelgm.

Flowchart symbol

Flowline → ⟶ (to see the flow)
terminator → start / stop (oval)
I/p → any input / o/p (llgm)
Processing → such as airthmetic oper^n (rectangle)
Decision → yes/no or true false (diamond)
On-page connector → use to join diff^h. (circle)
(only alphabets (capital))

WAF to find if a no is even or odd.



Miscellaneous concepts

1) Why 'char' is categorized as 'int datatype'

```
char l = 'M';
pf("%c", l);
```

O/P → M

```
pf("%d", l);
```

O/P → 77

All characters have integer values.

When any char is assigned internally the integer or ASCII value is stored.

```
char l = '5';
pf("%d", l);
```

O/P → 53

	ASCII value
a to z	97 to 122
A to Z	65 to 90
0 to 9	48 to 57

Escape sequences. (When a character is preceded by a backslash)

'\n' → new line
'\t' → put 4 spaces
'\'' → '
'\"' → "

'\1' → error

Type conversions.

One datatype to another

Implicit TC

Smaller to larger

```
int r = 300;
double nr = r;
```

(Automatic type conversion)

Explicit TC

Larger to smaller

```
double d = 4.67;
int nd = (int) d;
```

(There is a chance of loosing some value)

Clear Input buffer

Buffer → a temporary storage area or memory to store i/p & o/p.

↗ sometimes it clears itself.

Input stream → Buffer → code → Buffer → O/p Stream

i1, i2, i3 — [1] — o_1, o_2, o_3

Sometimes we need to clear the input buffer memory

Sometimes Enter key is taken as the input.

```
int main()
  {
char name[40], g;
 int age, id;

pf ("name")
sf (name)
pf ("name")

pf ("Enter id")
sf (id)
pf ("id")

pf ("Enter your age")
sf (age)
pf ("age")

pf ("Enter gender")
sf (gender)
pf ("gender")

while ((getchar()) != '\n'); // clear the buffer memory.
```

Best Practices

Naming Conventions

1) The first letter of a variable should be a letter or an underscore
   eg- num1 _num1
2) No spaces are allowed in variable declaration
   eg - num 2 X
3) Variable names are case sensitive.
   eg- rollNumber , rollnumber
4) Except underscore no other special symbols are allowed in middle of variable name.
   eg - role_no.

Camel Case.
myAge
↓ ↓
small capital

Snake case
my_age
↓
underscore connecting

Kebab Case (for files)
learn-programming
all in lower by hyphen

Errors.
Windows → Problems view
- i info
- ⚠ warning
- ⚠ warning with quick fix
- X error

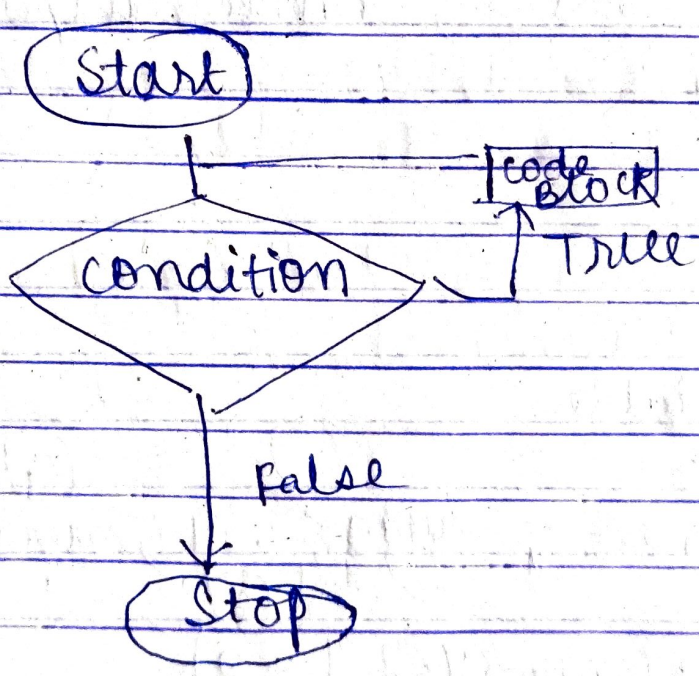
```
char x = '9'
int y = (x);
char z = 'b'

pf ("%c", y)
```

9 → 57
b → 98
155

Module 2

Diving into C-programming

Loops



A loop statement allows us to execute a statement or group of statements multiple times based on a cond^n.

Also known as iterators.

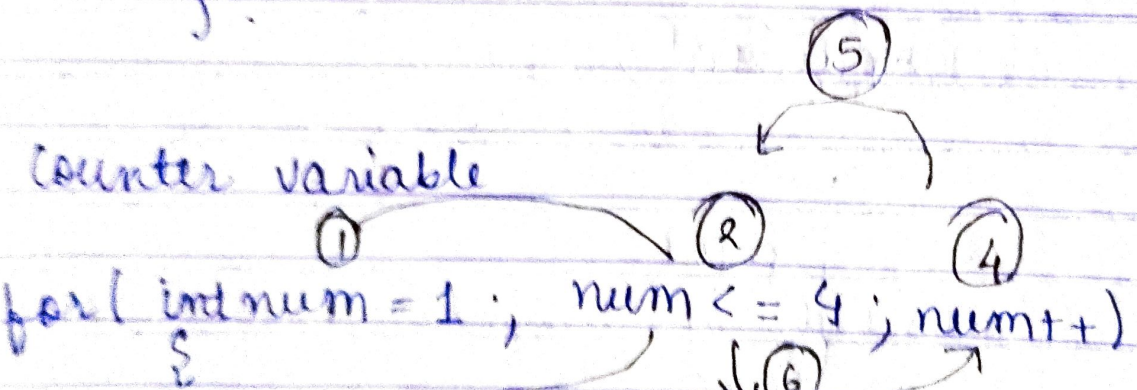
Types of loop

1) for loop
2) while loop
3) do-while loop

WAP to find out odd no from 1 to 4.

```
for(    ;    ;)
```

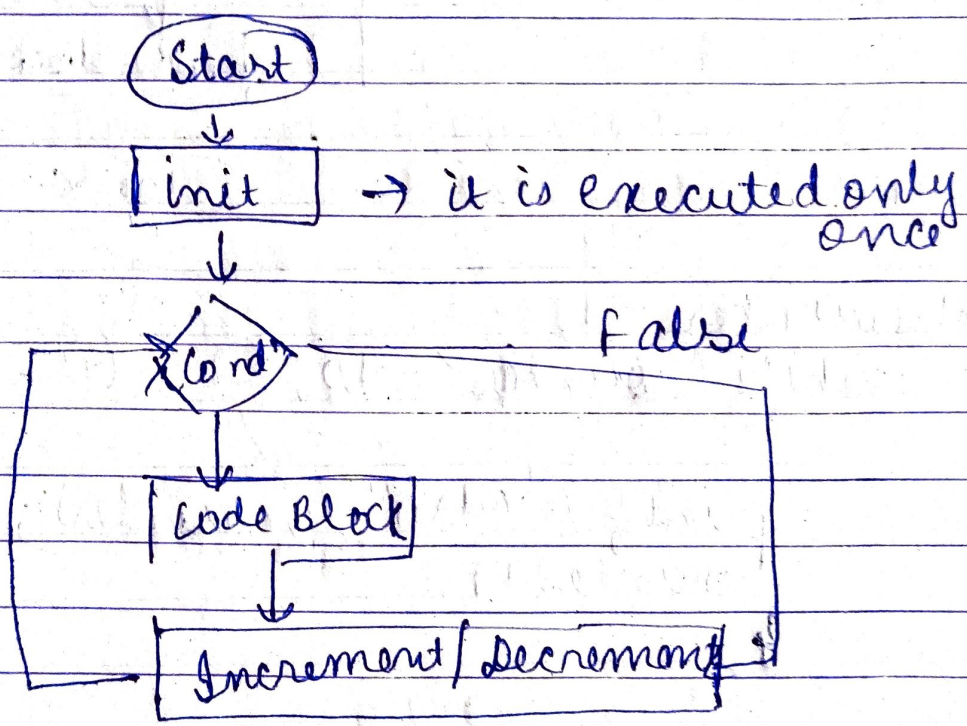
```
for( init ; condn ; increement/decreement)
  {

  }
```



until it becomes false

Syntax

```
for ( init ; condn ; increment/decrement)
  {
    //code
  }
```



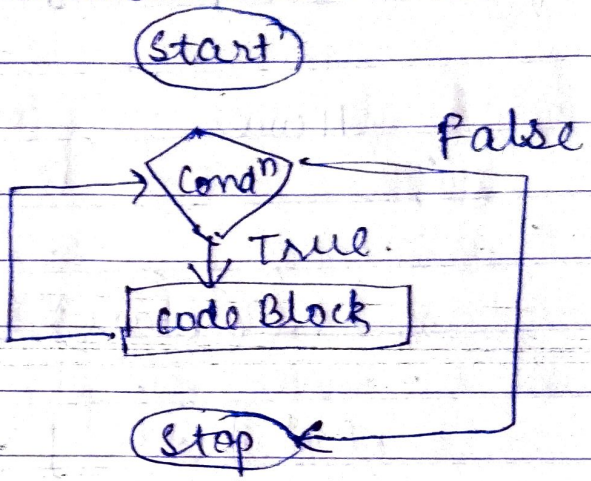
Q WAP to calc^n (1*1) + (2*2) + (3*3) + (4*4) + ... + (n*n)

```
int main()
{ int n, i ; long sum = 0;
  puts ("enter the number");
  scanf ("%d", &n);

for (i = 1 ; i <= n ; i++)
    {
    sum = sum + (i*i);
    }
printf ("sum = %ld", sum);
return 0;
```

While loop

WAP to print natural numbers till 10.

```
while (condn)
  {
  //code
  }
```




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

① ② ③ ④

WAP to find the factorial of a number.

```
int num = 5;
long factorial = 1;
while (num > 0)
  {
  factorial = factorial * num;
  num--;
  }
pf ("%ld\n", factorial).
```

while loop can be made without increment & decrement. It's not a compulsion.

Q. WAP to input a number & then count the number of digits present in the number using while loop.

123

```
# include <stdio.h>
int main() {

    int n , c=0;
puts ("enter the number")
scanf ("%d", &n);

while ( n>0) {
        c++ ;
        n = n/10 ;
      }

    printf ( " Total no of digits is %d", c);
return 0;
}
```

do-while loop

```
do {
// code
  }
while (condn) ;
```

WAP to print natural no from 1 to 10

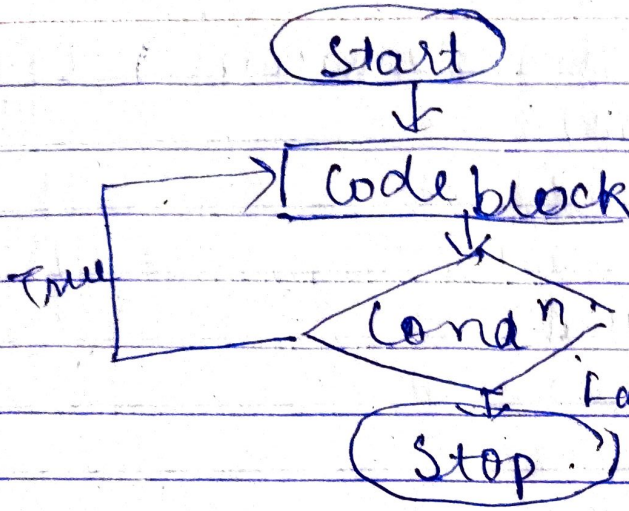
```
int c=1;
do{
  pf("%d", c); (1)
  c++;
} while (c>=10); (2).
```

Difference bet^n while or do while

- while → cond^n → we execute a code. (Entry-controlled)
- do while → we first execute code → cond^n (Exit controlled)

WAP to add numbers until user enters 0;

```
float n, s=0;
do
{
  puts("enter a number");
  pf("%f", &n);
  sum = sum + n;
}
while (n != 0);
```



Which loop is Better ?

for	while	do while
↓	↓	↓
no of iteratⁿ is fixed	no of iteration is not known	no of iteration is not known & code need to execute once

Q. WAP to check a no. is palindrome or not using do-while loop.

If original no = reversed no.

for eg 121, 1331

121 = 121

1331 = 1331

```
int main()
{
    int num, original num, remainder, reversed no=0;
    pf("Enter the num");
    sf("%d", &num);
    original num = num;

    do
    {
        ~~remainder = num%10~~
        ~~reverse num = rev*10~~
        remainder = num % 10;
        reversed num = (reversed num * 10) + remainder
        num = num / 10;
    }
        while (n > 0);

    if (original num == reversed num)
        pf("palindrome");

    else
        pf("Not palindrome");

    return 0;
}
```

10) 121 (12
10
21
20
1

10) 12 (1
10
2

10) 1 (0
0
1

10) 121 (
10
21
20
1

Let say 121

① R = 121 % 10 = 1 — 12 % 10 = 2

② R.N = 0 * 10 + 1 = 1 . 1 * 10 + 2 = 12

③ n = 121/10 = 12 . 20/10 = 0

i = 1 i=2 1,2,3 i=2

4<4 i=4 1++ 2<4. 1 2<4

3<4

i=1. i=3. i=5.

(1++) <=5. (3++) <=5

2++ 4++

(6++)

2 4 6

i=49 i<57 i++

49<57

49 - 1

50 - 2

51 - 3

52 - 4

53 - 5

54 - 6

55 - 7

56 - 8.

1 → true

0 → false.

i=5.

10)12(1 (5--)

-10

2

10)12(1

-10

2

10)121(

-10

21

20

1

5.4.3.2.1

R = 121%10 = 1

R.N = 0*10+1 = 1

N = 121/10 = 12.

R = 2

R.N = 1*10+2 = 12

N = 12/10 = 1.

R = 1%10 = 1

R.N = 12X10+1 = 121

N = 1/10 = 0

Loop control statements

Statements that are used to change normal execution of loop

'Break' statement
↓
It jumps over the part of code
↓
If user enters a negative number then terminate the loop

continue
↓
It skip that code.
↓
If user enters a number greater than 1000, skip that number.

goto.
↓
We need to define label

```
int i;
double n, s=0;

for(i=1; i<=10; i++)
  {
    pf("Enter number %d:", i);
    sf("%lf", &n);

    if (n<0)
      break;  // goto my_Label;

    if (number>1000)
      {
      puts("Number greater than 1000")
        continue;
      }
    sum+=number;
  }
```

mylabel:

printf ("Sum = %.2lf", s);
return 0;

goto is not used because it becomes difficult for the programmer to determine the flow.

Arrays (collection of value of same datatype)

int number[3] = {40, 7, 12};

array index: 0 1 2

Various way of initializing array

Method 1:

long scores[5] = {1, 2, 4, 5, 3};

Method 2:

long scores[] = {1, 2, 3, 4, 5};

Method 3:

long scores[3];

longScore[0] = 1;

```
#include <stdio.h>
int main()
{
long scores[5] = {9, 10, 11, 12, 13};

// Print all the elements using a loop
for(int i = 0; i<5; i++)
{
printf("%ld\n", scores[i]);

// Modify elements:

scores[1] = 9;
scores[2] = 10;

// User input

for (int i = 0; i<5; i++)
{
printf("%d", &scores[i]);
printf("%d", scores);
```

WAP to enter 10 elements in array & find the largest one.

```
# include<stdio>
int main()
  {
  int a[10], i = 0, largest = 0;

  while (i<10)
   pf ( Enter number:%d", i);
  sf ("%d", &a[i])
    i++ ;

    if (a[i] > largest)
       largest = a[i];
      i++;
    }
```

Memory Allocation.

Elements are stored at contiguous memory location

```
int num[4] = { [34] [51] [23] [12] };
```

0 1 2 3

100 104 108 112

Exploring 2-D Arrays

↓

An array containing diff^{n} arrays

row ↓ column ↓

```
int arr[ ][ ] = {{arr1}, {arr2}, {arr3}};
```

↓ it should always be filled

row/col^{m}	0	1	2	3
0	1	4	7	8
1	2	5	9	10
2	3	6	11	12

Print 2D Array (Matrix) → An array of arrays.

```
// row   for (int r = 0; r<3; r++)
      {
//column  for (int c = 0; c<5; c++)
          pf ("%d\n", arr[r][c]);
      }
```

Q. WAP to input a 2D array of size 2*3 print all the elements in form of a matrix and then find sum of all the elements.

```
    0  1  2
0
1
2
```

// for input

```
for( int r = 0 ; r < 2 ; r++)
  {
    for(int c = 0 ; c < 3 ; c++)
      scanf ( "%d\n", &arr[r][c]);

          Sum = Sum + arr[r][c]
  }
}
```

```
for (int r = 0 ; r < 2 ; r++)
    {
      for (int c = 0 ; c < 3 ; c++)
        {
          pf ("%d", a[r][c])
        }
      pf ("\n")
    }
```

* Arrays have a fixed size

Strings

A string is represent as an array of characters

```
char str[6] = "hi";
```

```
        0   1   2
char str[3] = {'h', 'i', '\0'}
```

A character array that doesn't contain a null character is not a string.

Various ways to declare a string

```
char s[3] = "hi"

char s[3] = {'h', 'i', '\0'};
```

```
// user input

char name[40];
puts("Enter your name:");
scanf("%s", name);
```

space until it encounters a [redacted]

We don't use & because it refers to the base address of array.

`[^\n]%c` → reads until it enters a enter key

String operations

#include<string.h>

// Append a copy of the source string to end of the destination string

Strcat();

```
char a[50] = "Hello";
char b[50] = "World";
Strcat(a, b);
```

a = Hello World
b = World

// Compares 2 strings lexigographically

Strcmp();

strcmp(a, b);

O/P (if true) 0

O/P (if false) non-zero no. (difference bet^n ASCII value of 2 characters i.e. unequal)

// Copy one string to another.

Strcpy();

Strcpy(a, b);

O/P

a = World
b = world

// length of string

strlen(); O/p = 5

strlen(a);

WAP to

a) append second string at the end of first with addn. space

strcat(a, " ");

strcat(a, b);

Functions

Built in (library funcn)	User-defined
Scanf() printf() of stdio.h	Created by programmer while writing
Strlen() strcat() of string.h	

→ return type

→ formal arguments / parameters

```
void sum ( int num1, int num2 )
    {
    int sum1 = num1 + num2;
    pf ( sum1 );
    }
```

} Func^n definition

```
void sum ( int , int ) ;
```

→ Func^n prototype / declaration

```
int main()
    {
    sum(p, q);
    }
```

} Func^n call

→ actual arguments / parameters

Use of func^n

1) It increases reusability
2) Clean, clear & Modular.
3) Reduces code redudancy

```
int sum(int , int)
int main()
    {
    p = sum(11, 12);
    }
```

```
int sum(int n1, int n2)
    {
    int s = n1 + n2;
    pf ( s );
    return s;
    }
```

2 func^n with same name is not allowed in C.

function- It is a set of statements

input → process data → output

Q. WAP to find the perimeter of rectangle.

```
#include <stdio.h>

int p (int , int);
int main()
  {
  int l, b,
pf (" Enter the length");
sf ("%d\n" &l);

pf (" Enter the breadth");
sf (" %d \n" , & b);

   p = peri (l, b)
pf ("%d", p );
```

```
int p (int a, int b)
  {
  peri = 2 * a * b;
  return peri;
  }
```

Passing Array as an argument.

```
double getAvg (float[] , int);

int main()
  {
```

```
float n[5] = {1, 2, 3, 4, 5};

double avg = getAvg(n, 5);
  pf (avg)
  return 0;
}

double getAvg ( float arr[ ], int size)
    {
      double avg, sum=0;

    for ( int i=0; i<size; i++)
        {
          s= s + arr[i]
    {
    avg = s/size;
    return avg;
    }
```

↑

WAP Using Array with funcn to find no. of odd no's

```
#include <stdio.h>
void c(int [ ], int)

int main( )
  {
int n[10] = {12, 3, 4, 5, 6, 7, 8, 9, 10, 11};
  c ( n , 10)
    return 0;
```

```
void c( int n1[ ], size)
    {
  int co = 0;
for(int i=0; i<size;
                  i++)
    {
if (n1%2 != 0)
  cot +
  }
pf ( c);
```

x = 10

i = 10

i++;	=	10
++i;	=	(++11) = 12
i--;	=	12--
i = i * 10;	=	(11 * 10 = 110
--i;	=	--110.
i = i/10;	=	109/10

50.8 * 10
508

10) 109 (
10
9
0
9

Recursion

A process where func^n calls itself

Stack overflow exception (error) → when stack memory get full and gets exhausted.

(All info of func^n. each time when the func^n is called)

```
void greetUser() {
    prints ("Hello");
    greetUser();
}
```

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

```
void greetUser(int num)
{
if (num > 0)
{
puts("Welcome");
greetUser(--num);
}
}
```

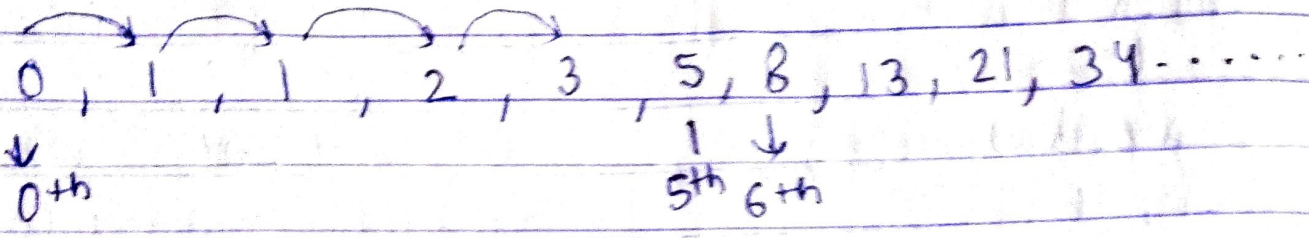
Understanding Recursive Calls.

WAP to find the factorial of a number.

```
#include <stdio.h>

long fact(int);
pf("Enter the number");
sf("%ld", &n);

long y = fact(n);

printf("The factorial is %ld", y)
```

```
long fact(int n) {
if (n == 0 || n == 1)
return 1;

else
if (n > 0)
return n * fact(n-1);
}
```

WAP to display the n^{th} number in the fibonacci series




```
#include <stdio.h>
int fib(int);

int main()
{
int n;
pf(n)
sf(n)
int result = fib(n);
pf(result)
return 0;
}
```

```
int fib(int a)
{
if(a==0 || a==1)
return a;
else
return(fib(n-1)+fib(n-2
}
```

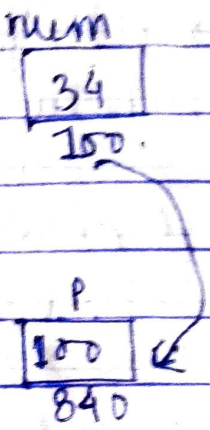
Advantage:
1) used in data structure tree traversal, sorting

Disadvantage
1) Program gets complexed
2) It consumes more stack
3) Use loops instead of recursion

World.

Pointers

* → value at
(Indirection operator)



datatype should be same → int num = 34;

int *p;
↓
pointer variable

p = # // Address of num
(p is the pointer to num variable)

pf(num) = 34
pf(&num) = 100
pf(p) = 100.
pf(*p) = *(&num) = 34.
↓
value at the (100)
pf(&p) = 840.

Pointer - A pointer is variable whose value is the address of another variable.

WAP to print sum of 2 no. using pointer

int *a int *b

a = &x b = &y

Now only replace x & y by *a and *b

Increment & decrement operator.

0	1	2	3
34	51	23	12
100	104	108	112

int num[4] = {34, 51, 23, 12};

int *p;

p = num; (since it points to the base address)

or

p = &num[0]

// p = 100 *p = 34

p++; p = 104 *p = 51

↳ It shift to next memory location.

p = p+2; p = 112 *p = 12

p++; no value are there so undefined values.

Decreementing operator

ptr--; ptr = ptr-1. // immediate previous memory location in array.

Passing Pointer to a function.

```
#include <stdio.h>

int gA(float, int);

int main() {
  float n[5] = {9,8, 3, 4, 2}
  float *p;
    p = n;
  double avg = gA(p, 5);
   pf (avg)
   return 0;
}.
```

```
double gA(float *p, int
{
double avg, sum=0
for(int i=0; i<size;
                i++)
{
sum = sum + *ptr
 ptr++;
}
return avg;
}
```

WAP to find the largest number in an array using pointer.

int *p
p = &arr[10]

```
#include <stdio.h>

int L (int* , int);
int *p , arr[10], n , l

    printf ("Enter the size of array");
    scanf ("%d" , &n);
for(i=0;  i < n ; i++)
    {
      scanf ("%d" , &a[i]);
    }

    l = L( p , 10)

    printf ("%d", l) ;


int L ( int *ptr, int size) .
    {
    int largest = *ptr ;

for (int i=0 ; i <size; i++)
    {
      if (*ptr > largest)
```

```
        largest = *ptr;
    }
    ptr++;
}
    return largest;
}
```

Advantages of Pointer

1). Useful to access elements in an array
2) Used in dynamic memory location
3) To implement O.S.

Memory Management

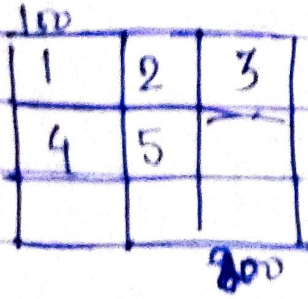
Static Memory allocation

(At compile-time i.e. before execution)

int x = 24; 1*4 bytes

float score [100]; 100 * 8 = 800 bytes

(size of array is fixed)



whole memory is consumed.

Dynamic Memory Allocation

(Allocation during run-time i.e. during execution)

↓

#include<stdlib.h>

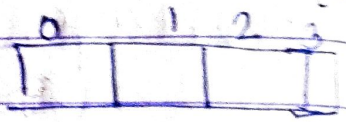
(Standard library)

null = no memory exists

Stdlib
- memory allocation — malloc()
- contiguous allocation — calloc()
- De-allocate memory — free()
- Re-allocation — realloc()

malloc()

int* ptr = (int *) malloc (4*



16 bytes

ptr[0] ptr[1] ptr[2] ptr[3]

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

int main ( )
    {
    int * ptr, n;
   puts (n)
    if (n)
    ptr = (int*) malloc(n * sizeof(int));
else {
   for(int i=0; i<n; i++){
            ptr[i] = i+1;   }

   pf ("The values present in array are");
   for (int i = 0; i <n ; i++)
            {
              pf ("%d" , ptr[i]);
        }
```

Advantage No memory wastage

For demanding too much memory it might show error

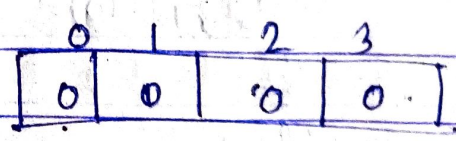
So for this

```
if(ptr==NULL)
  puts("Memory allocation failed");
  exit(0);
```

calloc()

```
int *ptr=(int*) Calloc(n, sizeof(int))
```

4,



The value of each element is initialized to 0 but in case of malloc you receive garbage value.

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

int main()
  {
  int n;
  float *p, sum=0;

  puts ("Enter n:");
  sf ("%d", &n);

  p=(float*) calloc (n, size of (float));

  if (p== = NULL)
    {
    puts ("Memory allocation failed");
    exit (0);
    }
  else
    {
    puts ("Enter data:")

  for (int i = 0;  i<n ;  i++)
      {
      scanf ("%f  , p+i);
      }
```

```
for (int i = 0; i < n ; i++)
    {
    sum = sum + p[i];
    }
printf ("The sum is %.2f ", sum);
}
return 0;
}
```

free() → It is used when dynamic memory allocation takes place.

free(ptr); → for both calloc() & malloc()

Realloc()

```
int c, newsize;
puts("Enter the no: of more value)
if( c)

    newsize = c + n;
        (int*)
    ptr = realloc (ptr, newsize * sizeof(int));

for (int i = n; i < newsize ; i++)
    pf ("%d_", ptr[i])
```

Use of realloc() & free() for sum of elements of array.

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

array, size

    puts(" Enter the size");
    scanf ("%d", &size)

    for (i=0 ; i<size, i++)
    {
        array[n] = i
```

```
int main()
{
    int n, int *p, sum = 0;

puts (" Enter n:");
scanf ("%d", &n);
    p = (int *) malloc( n, sizeof(int));
for (i=0 ; i<size ; i++)
    {
    scanf ("%d", p+i);
    }
```

```
int c, newsize

puts (" Enter the more values to be Stored");
  Scanf ("%d", &c);

    newsize = c+n;

  p= (int *) realloc (p, newsize * sizeof(float

  puts (" Enter data: ")

for (int i= n ; i < newsize ; i++)
        {
        Scanf ("%f", p+i);

for (int i = 0 ; i < newsize , i++)
                {
                sum = sum + p[i];
        }
    pf (" The sum is %d", sum);

  free (p);

  return 0;

}
```

User defined Datastructures

1) Structure
2) Union
3) Enum

We can't modify string literal after its declaration so we use

Structure

(grouping of entities of diff^{n} datatype)

```
Struct Employee{
    int id;
    char name[20];
    float salary;
};

int main()
{
    Struct Employee e1;
    e1.id = 1;
    e1.name = "Peter"   X Not allowed
    e1.salary = 23000;

    strcpy(e1.name, "Peter");

printf("%d %s %f", e1.id, e1.name
                        e1.salary);
    return 0;
}
```

Union (to save memory)

```
union Employee {
    int id          // 2 bytes  } 26 bytes
    char name[20];  // 20 bytes }
    float salary;   // 4 bytes
};
```

But memory occupied = the greatest memory occupied by the member.

As limited memory is there so the last value entered by us is shown.

At a time only one member variable can hold value so use

```
e1.id = 1;
pf("%d", e1.id)
```

→ put printf statements just after declaring.

Enum (Enumeration) → It is used to assign names to integer values.

```
enum Trafficlight {
    RED, YELLOW, GREEN   // in capital letters
};
```

RED → 0, YELLOW → 1, GREEN → 2

RED = 10 , YELLOW, GREEN
↓ ↓ ↓
10 11 12

```
int main()
{
enum Traffic light signal;
Signal = RED ;
Switch (signal)
{
Case RED : puts ("stop") ; break ;
Case YELLOW : puts (" wait") break ;
Case GREEN : puts (" Go") break ;
}
```

RED , YELLOW = 10 , GREEN.
↓ ↓ ↓
0 10 11

RED , YELLOW , GREEN
↓ ↓ ↓
0 1 10

Miscellaneous

%p format specifiers

↓

to print the actual memory address

%d → 13267$ (hexadecimal) → 0x1750

%p (hexadecimal no)

Passing by value and Pass by Refrence

```
void changeValue(int);
int main() {
    int x 10;
  printf("The value of x before %d", x);
   changeValue(x);
  printf("The value of x after %d", x);
   return 0;
}

Void changeValue(int num)
{
    num++;
}
```

O/P

10

10

? cannot be used in switch statement as it is a reserved word

Passing By reference

```
void changeValue(int *);
int x = 10;
pf("The value of x before %d", x);
changeValue(x);
pf("The new x is %d", x);

void changeValue(int* num)
{
(*num)++;
}
```

O/P

before 10
after 11

Local variables vs Global variable

```
int y;  ----> global
int main()
{
int x;  ----> local
```

inside a func"

↑ outside.
↑ a funcn

local variable prioritised more than global.

Random function.
↓
to generate random numbers.

Q. WAP to roll a dice
Random function: rand(), srand(time(NULL))

```
#include <stdlib
int
#include<stdio.h>
#include<time.h>
#include<unistd.h>
srand(time(NULL));
random value = (rand() %6)+1;
pf("You got: %d", randomValue);

int main()
{
int randomValue;
srand(time(NULL));
randomValue = (rand() %6) +1;
puts("Rolling the dice..");
usleep(2000000);
puts("You got the random value %d"
                    randomValue);
return 0;
```

0 1 2
3 4 5.

30.

30.

n1 = 10 n2 = 20

30.

10+ 10 = 20

n1 = 10 n2 = 20.

30 = 30

20+20 = 40

-1 20.

-1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

X X X X X X X

16 17, 18, 19, 20.

22 12

7 5

16 0.

5<5

1/s → Hello world

1/s → Hello world & internshala

1/s → Internshala

Hello (Hello Internshala)

CPP (C++)

↓

Advance version on C.

Getting started with C++

Bjarne Stroustrup

C → C with classes

↓

1983 C++ ++ → increment operator

C vs C++.

C++ > C

Creating C++ project (.cpp)

```
# include <iostream>
using namespace std;
int main()
{
cout << "! Hello world !" << endl;
return 0;
}
```

build your project atleast once before running

#include <iostream> → i/o operations on streams
↳ no .h file extension.

```
int main() {
    std::cout << "Hello Internshala" << std::endl;
    return 0;
}
```

std:: → Standard
cout → cout
<< → insertion operation
endl → endl is used for new line

std is getting repeated again & again so we use

```
using namespace std;
```

↓
use std whenever required in program

documentation section
link section
global declaration section
the main function.

Migrating C to C++

```
#include <string>
string name = "Person";
```

Char name[15] = "Person";

for float the precision control takes place.

User input in C++

```
cin >> size;
```
↓ extraction operator

To get string as user input

```
cin >> name;
```
↓ space as a terminating char

i/p: farm House
o/p: farm

```
getline(cin, name);
```

'\0' is not counted as length value.

Strings in C++

String msg = "hi"

0	1	2
'h'	'i'	'\0'

A String is an object represented as an array of char ended by '\0'.

String operations.

// Accessing a character

```
char letter = firstName[1];
```

// User input

```
String nickName;
cout << "Enter nickname";
getLine (cin, nickName);
```

// length.

```
int length = firstName.length();
```

// compare (lexicographically)

int result = firstName.compare(lastName);

// append

firstName.append(lastName);

// modify

firstName = "Rahul";

// String array

String names[3] = {"Rahul", "Ravi", "Raj"};

Boolean values in C++ (true/false)

↓

True or false values.

(1) (0)

datatype bool

bool isCodingFun = true;

int b1 = x > y;

bool b1 = x > y; o/p → 1

↓

it stores either 1 or 0

cout << true + 7; o/p → 8

Default parameters

void display (int, float, string message = "hi");

display (10, 7.8);

← Right to left

Function Overloading

void add (int, int);
void add (string, string);
void add (int, int, int)

Same function name but different parameters

↓ the return type doesn't play a role

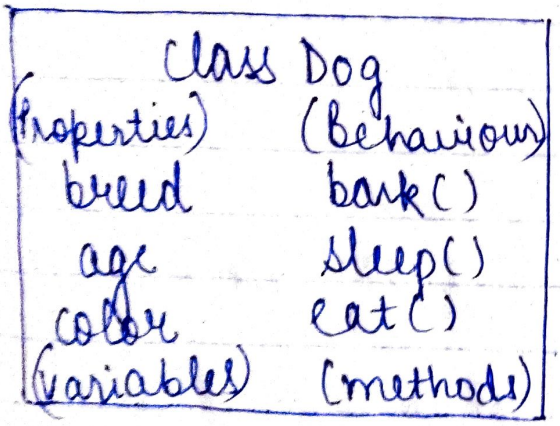
add (10, 20);
add ("Hi", "you");
add (10, 20, 30);

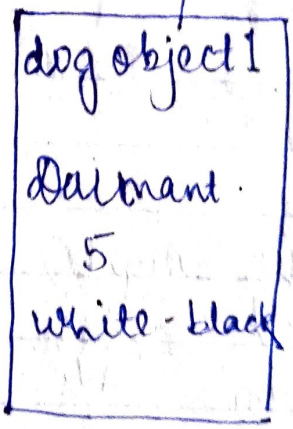
(1+1) * 20

Datatypes

Primary	User defined	Derived
char	Array	Structure
int	Pointers	Union
long	String	Enum
float		Class
double		
bool		

Class And Object





One class, many objects

A class is a blue print

→ (capital letter)

Creating a class in C++

(It allows code reusability)

```
class Employee {
  public:                    // access specifier
    int id;                  // data members
    string name;


};
```

Creating objects

```
int main() {

  Employee emp1;
  emp1.id = 1;
  emp1.name = "Hi";
  cout << emp1.id << emp1.name << endl;
```

Methods in a class
(functions)

Inside a class

Outside a class

```
public:
  int id;
  string name;
  string getFullName() { //No use of func^n. declaration
      return = first name + last name; }
  };
```

```
emp1.getfull Name()
```

Outside a class.

~~void displayDetails~~

Inside

void displayDetails();

↓

func^n declaration

Outside.

void Employee::displayDetails()

↓

scope resolution

since it is out of class

```
cout << id ;
```

↑ it can't return data

Constructors & Destructors

↓ ↓

create objects | Destruct or destroy objects

↓

Special method which is executed when a object is created

name of constructor = class name

no return type.

Should be defined in public section.

```
public : {
Employee ()
    {
    cout << " Employee is created";
    }
};
```

Outside the class

```
Employee(); // inside // declaration
```

```
Employee :: Employee () {
cout << "Employee"
    }
```

// outside // defination.

Types of constructor

Default | Parameterized | Copy

Purpose: perform some action as a object is created.

2) Parameterized constructor

```
Public:
Employee (int ID, string name)
    {
    id = ID;    if using   id = id;
    }
};
```

Use.

this → id = id;

this → attribute present at the class

points to the id of the ongoing class.

In main

Employee e1(1, "Hi")
// parameterized

Employee e2 ;
//default constructor

Copy constructors

↓

copy values of datamember from O_1 to O_2

```
Employee emp3 = emp2; // copy constructor
emp3.id = 3;
```

↓

no same id should be used.

Constructor Overloading

↓

multiple constructor having same name but diff parameters.

Destructors (destroy objects)

↓

deletes an object to free up memory (it is internally present)

i) no parameters
ii) no return value

```
~Employee() {
    cout << "Employee Object deleted"
}
```

for outside.

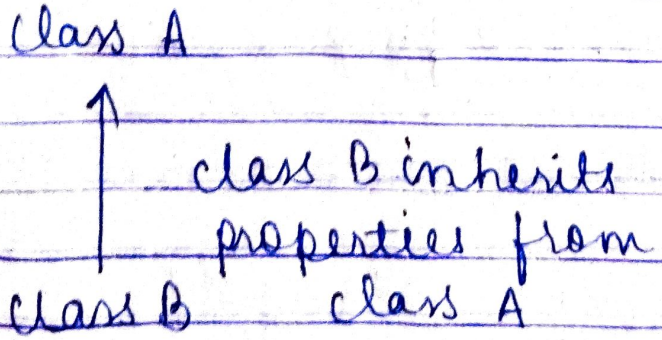
declaration

~Employee();

defination.

```
Employee :: ~Employee()
{
cout << "Employee object with id:" << this->id
                << "being deleted";
}
```

Inheritance



(Parent / base / superclass)

Animal.

```
class Animal {
  public:
    int age;
    string color;

    void sun() {

    }
};
```

(Derived / child)

Dog.

```
class Dog : public Animal {
  public:
    string petName;
    void bark()
    {
    }
};
```

Lion

```
class Lion : public Animal
{
public:
  string name;
  void roar() { }
};
```

```
#include <iostream>
#include <string>

using namespace std;

class Animal {
  public:
    int age;
    string color;

    void run {
        cout << "Running" << endl;
    }
};

class Lion : public Animal {

    public:
        string petName.
        void roar()
            {
            cout << "Roaring" << endl;
            }
};

int main()
  Lion lion;
  lion.roar().
```

Super class can't access sub-class properties

Types of inheritance

Single Multilevel hierarchical

Multiple Hybrid

Single in.

```
class Animal
   ↑
class Dog
```

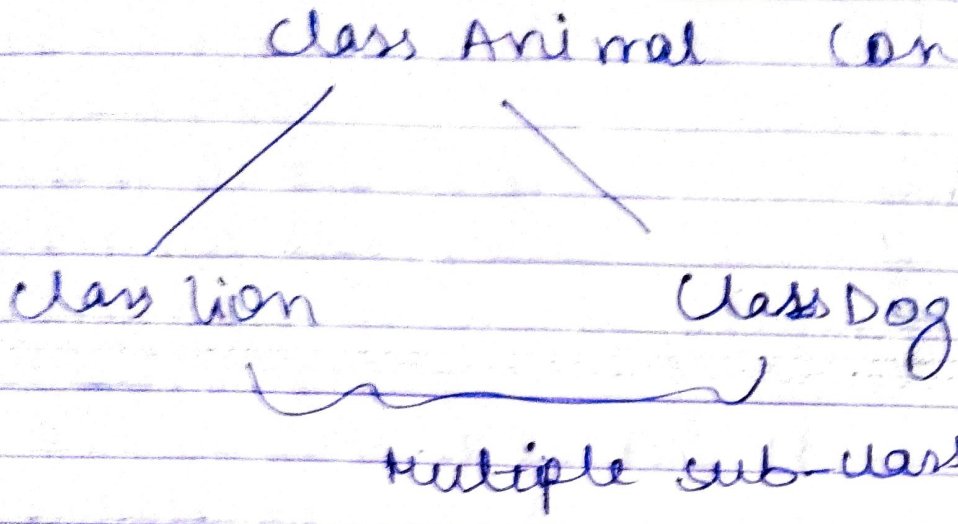
One superclass & One Subclass

Multilevel

```
Class Animal
   ↑
class Dog
   ↑
class puppy
```

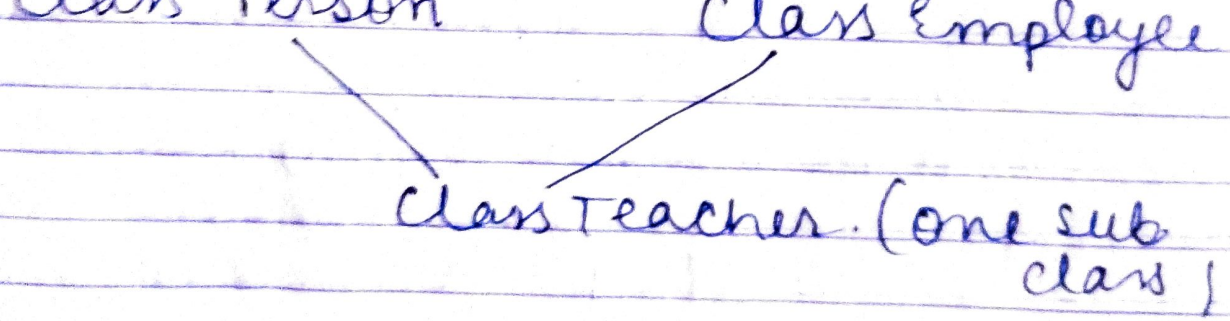
SuperClass
Subclass
child class.

Hierarchical



Multiple

(Multiple superclass)

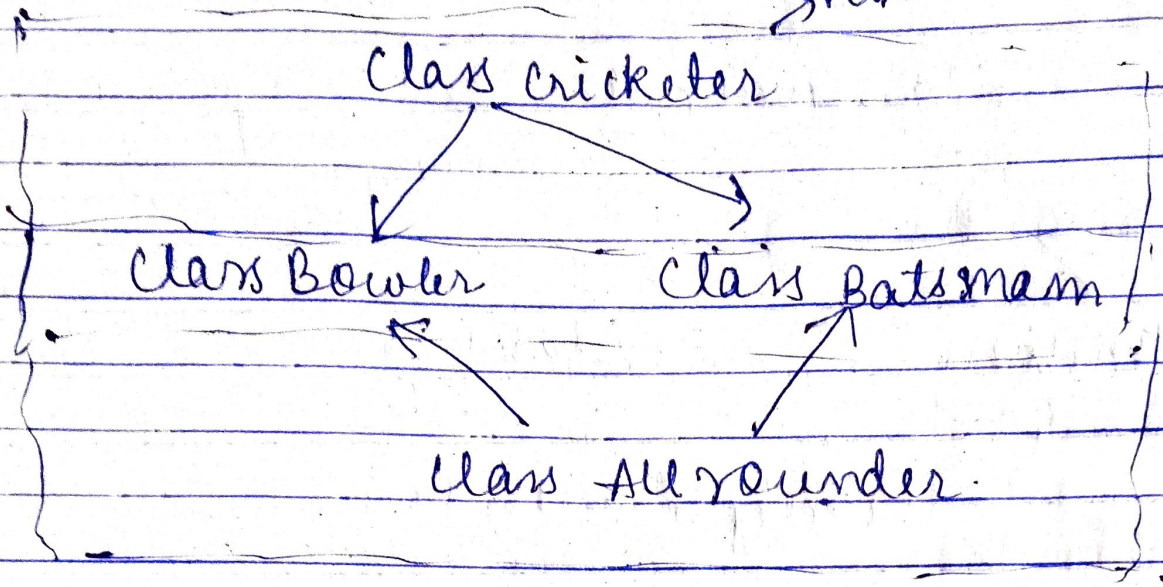


Hierarchical

```
class Person          class Employee{
{                             };
};

class Teacher : public Person, public Employee
{
};
```

Hybrid (2 or more inheritance

→ Hibrarchical



↓ Multiple

class <Subclass Name> : public <superclass Name>
{ };

→ modifiers

Access specifiers (whether a func^n can access the attributes of other)

Public Private Protected

Private & Public access specifiers

```
public:    // accesible outside the class.
float radius;
double area();

private:   // is not accesible outside
                the class

Class Circle {
  private:
    float radius;

  public:
    void SetRadius (float radius)
      {
        this -> radius = radius;
      }
    double getArea ()
        {
          return 3.14 * radius * radius;
        }
};

int main ()
  {
    Circle. setRadius (5);
```

Users can set the radius but not access the radius.

default is private

Data hiding	Abstraction	Encapsulation
↓	↓	↓
As we made it private (Outside world has no idea about that attribute)	method getArea() set wrns area of the object.	grouping similar data & function in a group like we wrap similar data & methods in a single unit
Hiding data outside the class	Just hiding how the things are implemented.	
	Hiding the implementation details	

Protected Access specifier

```
class Person {

private:
  string phoneNumber;  // it is private

public:
  string FullName;

};

class Student : public Person {
  }                          // it can be accessed here

protected:
  string phoneNumber;
```

↓

It can we accessed within the child class but can't be accessed outside the class.

It can be accessed by inherited class

Polymorphism

Object (behave differently in different situations)

poly + morphs

many forms

One name & many forms

Women → Mother, Employee, customer

Types of polymorphism

- Compile Time
  - function overloading
  - (Method)
- Run-time
  - Function Overridi
  - Virtual functio

Function Overriding

Same function in the child class present in the parent class.

We override function of child class by parent class.

```
class Animal {
    public:
        void sound() {
            cout << Animal << endl;
        };

class Dog : public Animal {
        public:
            void sound()
            {
            cout << Dog << endl;
            };

int main() {
    Dog dog;                    dog → Dog object
    dog.sound();
}
```

no of parameter should be same
same name

Virtual function

```
#include <new>
Animal *animal = new Dog();
    animal → sound();
    animal → Sleep();
```

Too many child class, then it is used

Replace every method by
public:
virtual void sleep(){
cout <<
}

expected to be overriden in child class
may/may not

ensures the execution of proper func^n.

By default in child class it would be virtual.

Abstract classes

virtual void sound() = 0; // pure virtual function

mandatory to override in the child class

Advantage: it forces the child class to override it

Abstract class

Abstract class

A class that contains atleast one pure virtual function.

We can't create object of an abstract class
You can create pointer
Child which inherits from an abstract base class is called concrete class.

Interface

An abstract class that contains only pure virtual function.

```
public:
    virtual void sleep() = 0;
    virtual void sound() = 0;
```

Shape
virtual double getArea() = 0;

mandatory to implement

Class square class Rectangle class △

* All interface are abstract classes.

Friend class & Friend function

- Not reserved keywords in C++

to access private attributes of other class

```
friend class car;

class Employee {
    private:                    // or protected
        string phNo;
    public:
        string name;
    void setPhoneNumber(string phoneNumber)
        {
        this -> phNo = phoneNumber;
        }
    friend class car;
};
```

```
class Car{

    public:
        string carName;

    void display (Employee* emp)
            {
        cout <<"Employee name: " <<empname <<
                                emp.phNo << "Car name:"
                                    << carName

            }

};

int main() {

    Employee employee;
    employee.setPhoneNumber("1111000");
    employee.name = "Rishi";

    Car car;
    car.carName = "Ferrari"
    car.display(employee);

    return 0;
}
```

Friend function

inside the class

```
friend void display();
```

outside the class.

```
void display(Employee emp)
{
    cout << "Employee name:" << emp.name << "ph No" << emp.
}                                                    endl
```

Miscellaneous Concepts

Reserved keywords (90+ reserved keywords)

Pascal Case
eg: MyAge

Diving into C++ programming

Namespaces (add^{n} info to diff^{n} bet^{n} func^{n}, variable & class)

```
using namespace std;
```

variable name (should be shorter)
→ japan

```
namespace jp {
    float dollarValue = 108;
}

namespace cn {
    float dollarValue = 7;
}

int main() {
    cout << "1 USD = " << jp::dollarValue << "Yen" << endl;
    cout << "1 USD = " << cn::dollarValue << "Yuaan" << endl;
```

We can use same function name also.
Classes can also be used inside a namespace

The using directive

```
using namespace std;
```

using → directive
namespace → reserved keyword
std → variable name

using namespace jp; ✓ for only one.
using namespace jn; ✗

local variable always gets the

Standard libraries

C++ std. library is a combo of classes & functions.

C	C++
stdio.h	iostream
stdlib.h	cstdlib
time.h	ctime
string.h	cstring or string
unistd.h	unistd.h
math.h	cmath

cmath → part of O.S not std library

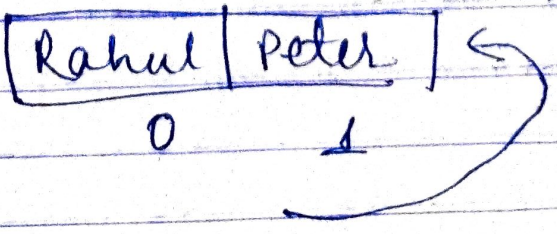
Math library

#include <cmath>

Vector library → similar to array but in vector you can insert as many elements you want to.

#include <vector> (you can store objects of a user defined class)

```
int main() {

    vector < String > names;
                ↓       ↓
            datatype  variable.

    name: push_back ("Rahul");   // Adds elements
    name. push_back ("Peter");

    cout <<"size" <<name.size() <<endl;

// Accessing an element.

    cout << names[0] <<endl;

// Remove element.
    names. pop_back();
    cout << name. size () <<endl;
```



Preprocessor Directives

(Macro)

when anything is constant.

```
#define UPPER_LIMIT 10      // Macro defination
```

- `#define` → use to define a macro
- `UPPER_LIMIT` → macro template
- `10` → Macro expansion

* All letters should be capital & joined by _ .

Macro with parameter.

```
#define AREA(r) (3.14*r*r)   // Macro with parameter
```

→ No datatype required (Any datatype is excepted)

```
cout << AREA(5);
```

func^n & Macro with parameter
↓
one-line of code
faster

Commonly used preprocessor → to preprocess the source code before compiling it.

Macro
define

File inclusion
include

File inclusion

```
#include <iostream>   // File inclusion defination
```

Creating own header file

```
#include "filename"
```

Exception Handling (Error)

Types of Error

- Compile time
  - C/C++ → [Compiler] → (Program)
  - (missing of ; or comments)
- Run-time (exceptions)
  - 0 1 0 1 0 1 0
  - 0 0 1 1 1 1 0
  - 1 1 0 0 0 0

Handling Exception

```
#include<iostream>
using namespace std;

int main()
{
int a, b;

cout << "Enter a";
cin >> a

cout << "Enter b";
cin >> b
```

int result = a/b;
cout << result << endl;

i/p

10 5

10/5 = 2

o/p

2

i/p

2 0

2/0 = ∞

o/p

no output

```
try {
    if (b == 0) {
        throw "The value shouldn't be 0";
    }
    int result = a/b;
    cout << result
}

catch (const char* msg) {
    cout << msg << endl;
}
```

```
try {
    // Try block code
    // Throw exception using throw keyword
  }
Catch ( )
  {
    // catch code.
  }
```

Debugging.
↓
(To find a bug or error)
using debugger.

Remove try & catch ip = 2/0

(Local window debugger)
↓
It shows the exception & line number.

Tracking values.

when int value is like

```
int si;
int r = 5.7;
int p = 100;
int t = 1;
```

SI = 5.

Use Breakpoints (pause the execution at that line of code)

Step-over.

When program ends debugger detach itself

when we want to debug code before or after main. (not in main)

Step in & Step out

Reach to place where function is called then step in.

Step out

Step over
↓
Step in
↓
Step out

File Handling :

How to write & Read file

<fstream>

ofstream
O/p file stream
write
file

ifstream
i/p file stream
Read
file

```
#include <fstream>
int main() {
// Create an object of ofstream
    ofstream oFile;
// Create a file & open it
    oFile.open("my-note.txt");
// Write on the file
    oFile << "Hi"
    oFile << "Internshala";
```

kebab case

if not present it would be create

```
// close the opened file
ofile.close();
return 0;
}
```

Read from a file

```
include <fstream>
int main()
  {
  string str;   // hold the text
  ifstream iFile;
  iFile.open("my-note.txt"); // ensure it is present
  // use a while loop
  while(!iFile.eof()) // end of file
      {
      getline(iFile, str); // Reads a line
       cout<<str;
      }
  iFile.close();
```

Miscellaneous Concept

Programming Paradigms

A way to classify programs based on their features

OOP's Object Oriented Programming	POP Procedural Programming
Program is divided into small parts called objects	Functions.
inheritance	no inheritance
function overloading	Not supported
Access specifiers more secure	Not present
Data is important than funcn	funcn is more imp than data
C++, Java, Python	C, FORTAN, C++ Pascal

Hybrid language

p ∠ q ! p : q
q9.1∠9901 ?

C vs C++

C	C++
i) POP	i) POP + OOPs
ii) Not supported a) inheritance b) OOP's concept	ii) Supported
iii) 32 keywords	iii) More keywords
iv) No exception handling