

Notes Made By

- RITI Kumari

Programming 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 programming language.

Use of C

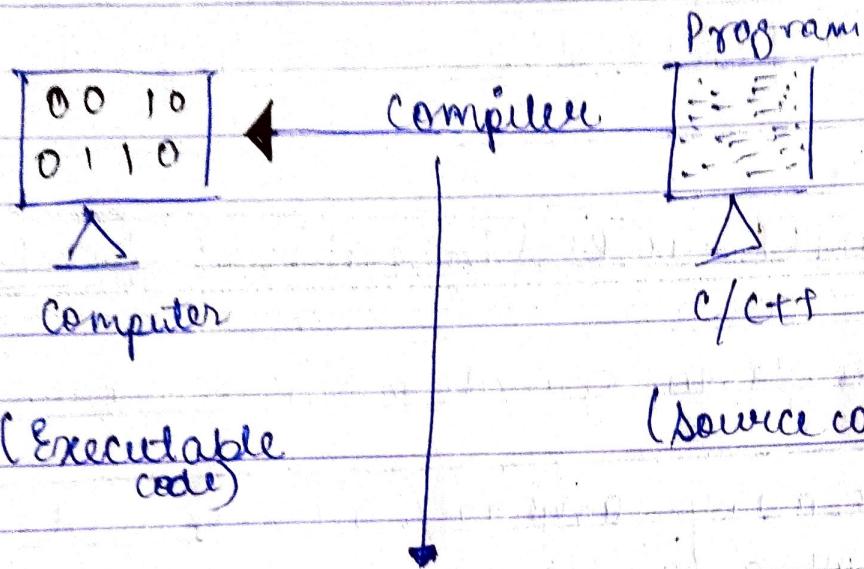
- 1) windows os is build using c & C++.
- 2) A part of fb applⁿ. is written in c.
- 3) A part of photoshop is written in c++.

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 it to executable code.

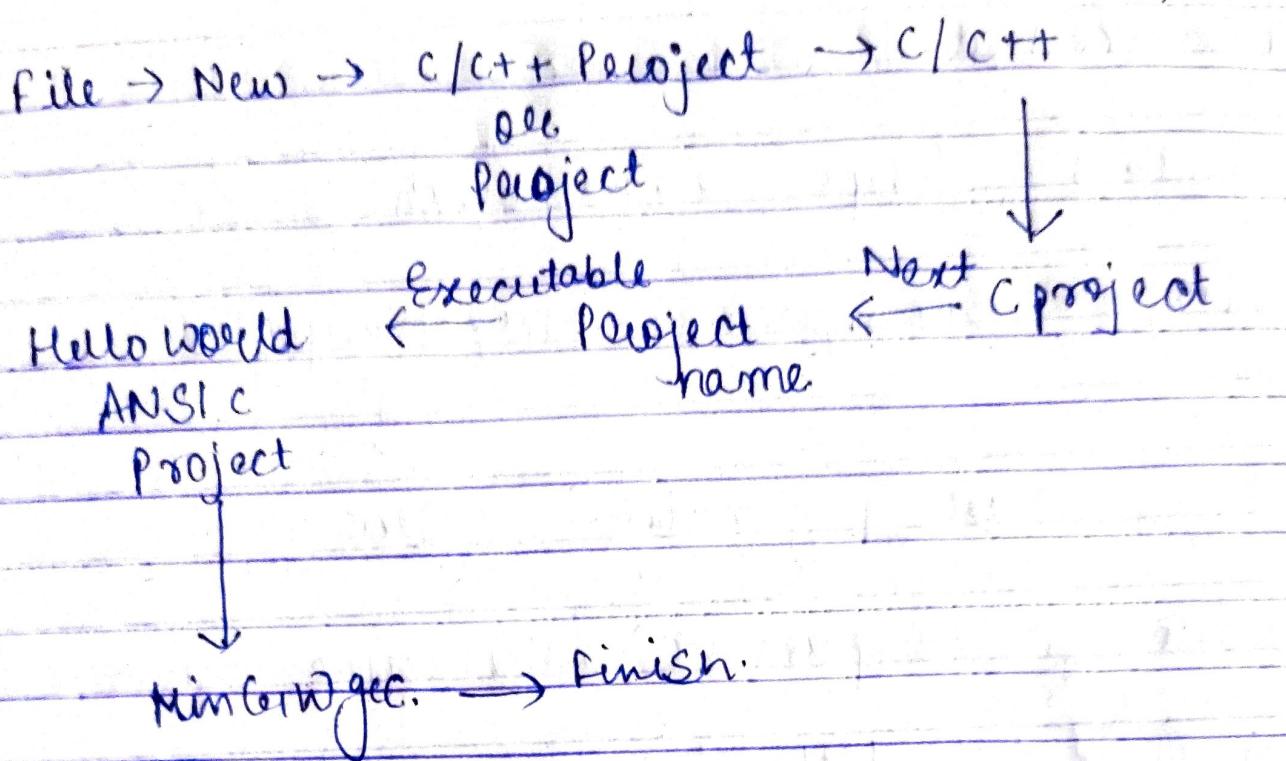
Compiler → GCC compiler (GNU)

IDE → Eclipse IDE (is built using Java)

Java → JDK (JRE)

Creating first C-project

Workspace - where projects would be saved.



#. include < stdio.h >

```

int main()
{
    puts ("Hello World!!");
    return 0;
}
  
```

Exploring Eclipse IDE

for theme: Window → preferences → general

apply ← Select your ← appearance.
theme
(light theme)

File → Restart

Window of Eclipse → Workbench.

Window → Show view → Project explorer



learn-programming.c → .c

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
{
    → string (within
    printf ("Hello Internshala!"); quotes)
    printf ("I am here to teach you");
    between 0;
}
```

Run your program
Build project → Run → local C/C++ appl.

% 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

don't 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] = " Nam is great";

char size = 'M';

int personItSees = 2;

float price = 365.8;

format specifier

printf ("%s", name);

printf ("%c", size);

printf ("%d", personItSees);

printf ("%f", price);

"%1f"

365.8

"%2f"

365.79

"%f"

365.79988

after rounding off

How Data is stored in a Token

Datatype

primary

char

derived

array

User defined

Structure

integer type

int

pointer

Union.

long

string

enum

floating point type

float

double

Type	Size (bytes)	Example,	format specifier
char	1	'e' '\$' '#' '@' '2'	.1. C
int	2 or 4	2768, -3210.	.1. d
float	4	3.12, -2.547219	.1. f
long	4 or 8	-2143648, 427688	.1. ld
double	8	2.6126762676	.1. lf

Allocable 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;

+ value doesn't change

|| convention to use capital

letter for constant value.

```
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
Std out.

Taking user input

char size;
scanf ("%c", &size)
↓
address of

<stdio.h>

Improving user experience.

#include <unistd.h>

usleep (100000); → Usually stops the
↓
1 million ~~microsec~~,
(1 sec) program for
seconds.

Newline ('\n') → Enter key

```
char name[30];
```

```
scanf ("%[^\\n]s", name);
```



↳ for string we
don't use &.

User input will enter key is

pressed.

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

%[^\\n]s → it could be used.

but it can be replaced by %s.

(it stores string characters
until white space.)

Input operation

Output operation

size = getchar();

putchar(size)

→ only a single
char value.

Operators

symbols to perform operations (*, +, -, /, etc)

10 + 20

size of(int)

etc)

operator

operator operand

operand

Types of Operators

- 1) Arithmetic
- 2) Relational
- 3) Logical
- 4) Increment & Decrement
- 5) Assignment
- 6) Special operator

Arithmetic

(+, -, *, /, %)

$$3 \% 2 = 1$$

$$x = 12$$

$$y = 4$$

$$+ = 16$$

$$- = 8$$

$$* = 48$$

$$/ = 3$$

$$\% = 0$$

Precedence & associativity

) has the highest priority

precedence of $/ \cdot \% > +, -$

assoc → left to right

P D M A S.

→ 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

OR

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

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

NOT

A | !A

0 | 1

Associativity = left to right

Assignment operator.

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

Associativity : Right to left

bill = bill + task
 $\frac{\downarrow}{\downarrow}$
 $a \longleftrightarrow i$

Increment & decrement

Postfix
a++
a--

Prefix
++a
--a

int s=10;
int ns=s++

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

ns=s;
s=s+1;

s=s+1;
ns=s;

ns=10
s=11

ns=11
s=11

int age = 10;

O/P

pf("1.d", age++);
pf("1.d", age);
pf("1.d", +age);

age = 10
age = 11
age = 12

• pf("1.d", age++ + +age); left to right

$$\begin{aligned}&= 12 + (++13) \\&= 12 + 14 \\&= 26\end{aligned}$$

age = 14

Special Operators

size of()
address &
of *

Returns size of memory location
Returns the address of memory loc.
pointer to a variable

int n = 8.

pf("1.d", (n++) - (++n) + (--n))

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

$$8 - 9 + (-9)$$

$$8 - 9 - 8$$

$$16 - 9 = 7$$

$$\text{int } r = 3 * 10 - 2 \cdot 1.5 / 2$$

$$\begin{array}{r} 0 \\ 2) 5 (2 \\ \underline{-4} \end{array}$$

$$= 30 - 2 \cdot 1.5 / 2$$

$$\begin{array}{r} 1 \\ 2) 4 \\ \underline{-2} \end{array}$$

$$= 30 - 2 / 2$$

$$\begin{array}{r} 2 \\ 2) 5 \end{array}$$

$$= 30 - 1$$

$$= 29$$

$$\text{int } n = 100 / 3$$

$$= 3$$

$$\begin{array}{r} 5) 2 (9 \\ \underline{-0} \end{array}$$

$$3) 100 (33$$

$$\begin{array}{r} 9 \\ 10 \\ \underline{-9} \\ 1 \end{array}$$

$$\text{int } r = 3 * (10 - 2) \cdot 1.5 / 2$$

$$\begin{array}{r} 5) 24 (4 \\ 20 \\ \underline{-4} \end{array}$$

$$= 3 * 8 \cdot 1.5 / 2$$

$$= 24 \cdot 1.5 / 2$$

$$\begin{array}{r} 4 \\ 2 \\ \underline{-2} \end{array}$$

$$\text{age} = \text{age} * \text{num};$$

$$10 > 10 \quad 88 \quad 15 > = 16 \quad 11 \quad 10 = = 10$$

0

0 1 1 1

Conditional statements

- i) If - else
- ii) Ternary
- iii) Else-if
- iv) Switch
- v) break

) If - else

Apply 10% if bill $'u' \geq 500$.

if (condition)

{

 1 Execute statement

{

 else

{

 2 Execute statements

}

2) Ternary operator (Only single expression & single statement)
Condition ? expl : exp 2.

payable = retailPrice ≥ 500 ? retailPrice * 0.9 : retailPrice ;

Else-if statements

if (cond n)

{

else if (cond "n")

{

}

else

{

}

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

if (marks > 500)

{

printf("invalid data")

{

else if (marks >= 450 && marks <= 56)

{

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");
```

```
    if ("enter your rating:");
```

```
    if ("1.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.

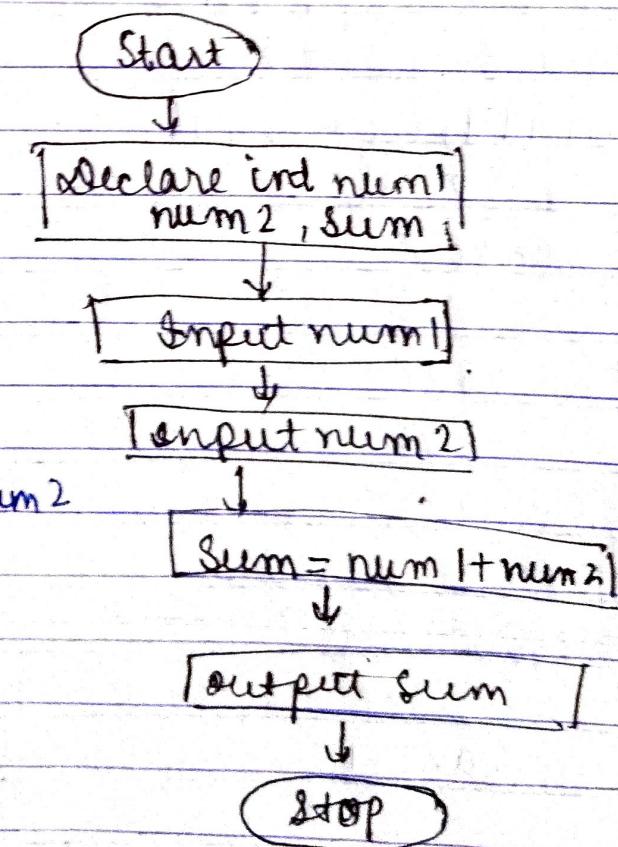
2) Input num1

3) Input num2

4) Compute sum = num1 + num2

5) Output sum.

Flowchart



Code

Flowchart \rightarrow graphical representation of algorithm.

Operⁿ \rightarrow rectangle

start/stop \rightarrow oval.

input \rightarrow parallelogram.

Flowchart symbol

flowline \rightarrow  (to see the flow)

terminator \rightarrow start/stop (oval)

I/O \rightarrow any input/Op (llgm)

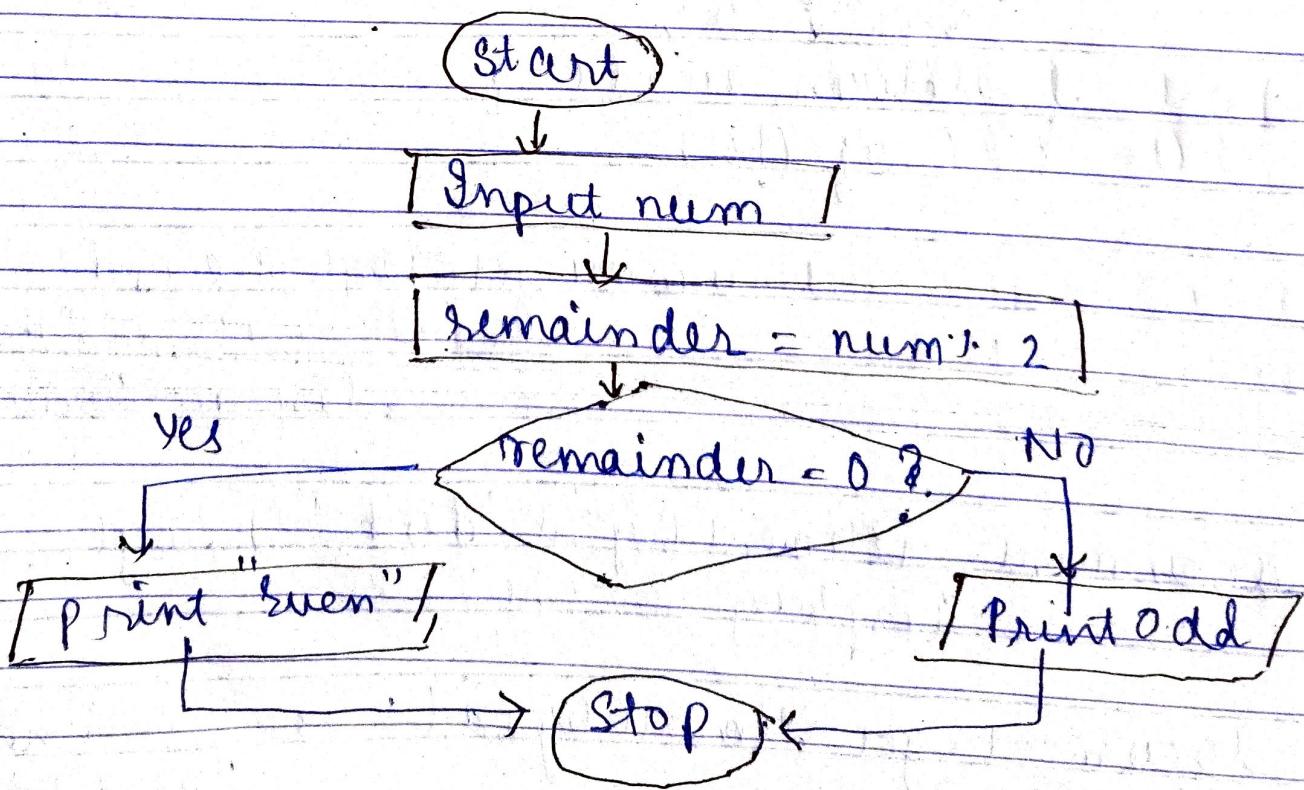
processing \rightarrow such as arithmetic operⁿ (rectangle)

decision \rightarrow yes/no or true/false (diamond)

On-page connector \rightarrow use to join diff. (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("i.c", l);

O/P → M

pf("i.d", l);

O/P → 77

All character have integer values.

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

char l = '5';

pf("i.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' → indent 4 spaces

'\' → \

'\\' → "

Type conversions -

One datatype to another

Implicit TC

Smaller → Larger

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

(Automatic type conversion)

Explicit TC

Larger → Smaller

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

(There is a chance of losing some value)

Clear Input buffer

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

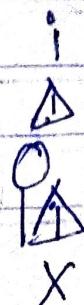
my_age
 ↓
 underscore connecting

Kebab Case (for files)

learn-programming
 all in lower by hyphen

Errors .

windows → Problems view



info

warning

warning with quick fix

error

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

9 → 57

b → 98

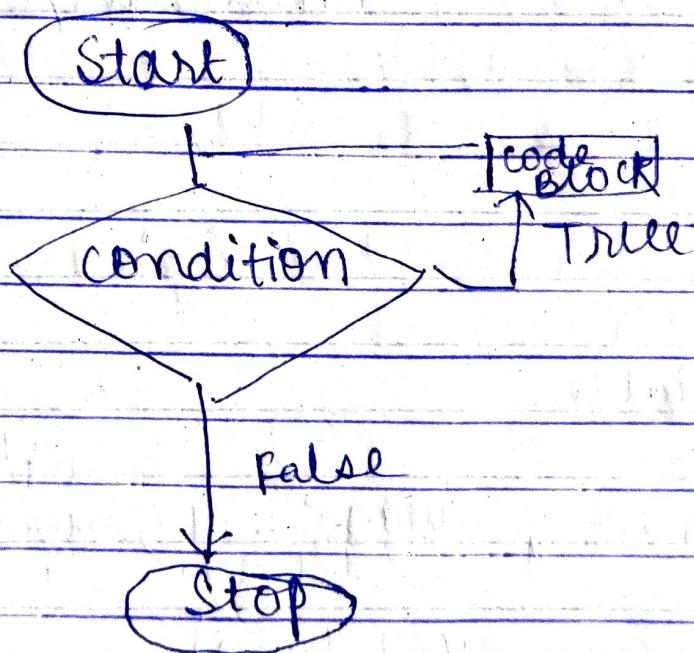
155

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

Module 2

Wing 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 9.

for (; ;)

for (init ; condⁿ ; increment/decrement)

}

(5)

counter variable

(6)

(4)

(1)

(2)

for (int num = 1 ; num <= 4 ; num++)

{ (3) if (num % 2 != 0)

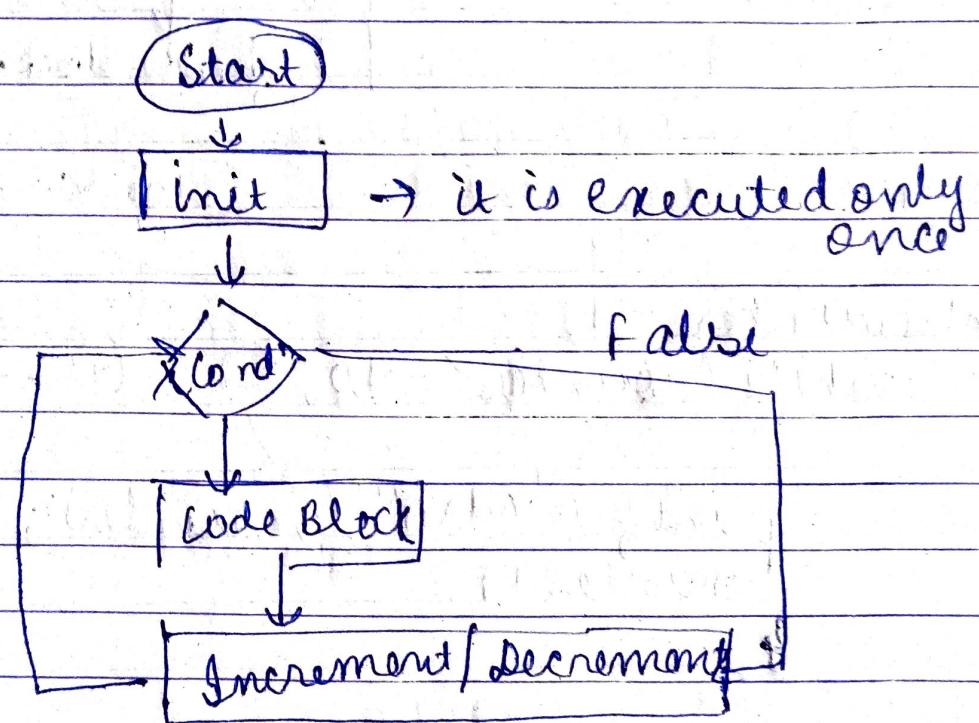
 printf("%d\n", num); }

}

until it becomes false

Syntax

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



Q WAP to calcⁿ: $(1*1) + (2*2) + (3*3) + \dots + (n*n)$

```
int main()
```

```
{ int n, i ; long sum = 0 ; }
```

```
gets ("Enter the number");
```

```
scanf ("%d", &n);
```

```
pf ("sum %.ld", sum);
```

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

```
{
```

```
    sum = sum + (i*i) ;
```

```
return 0;
```

while loop

WAP to print natural numbers till 10.

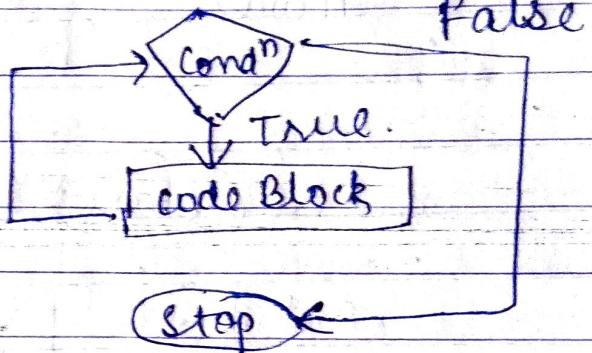
while (cond^n)

{

 || code

}

(Start)



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--;

 printf ("%d\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.

12.3

```
# 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
```

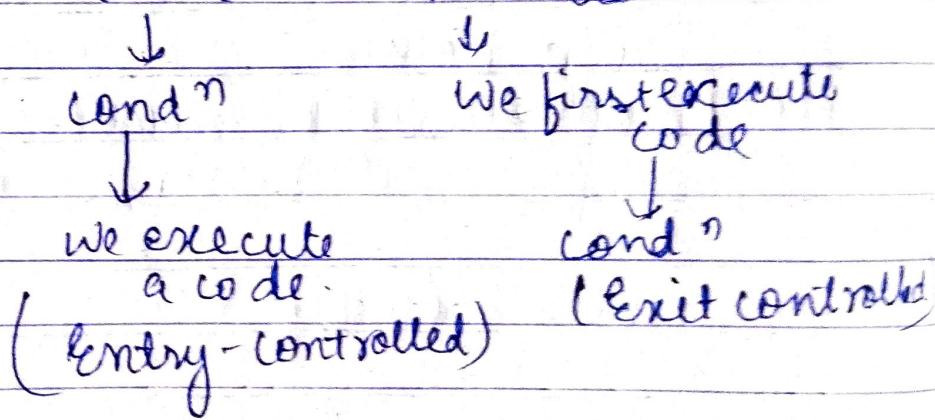
```
}
```

```
while (cond);
```

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

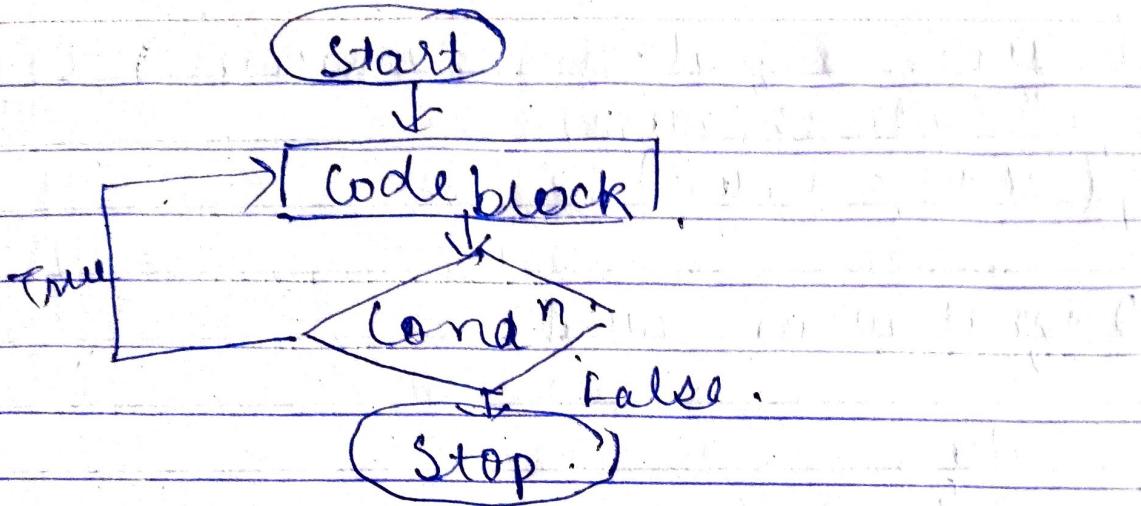


WAP - to add numbers until user enters 0;

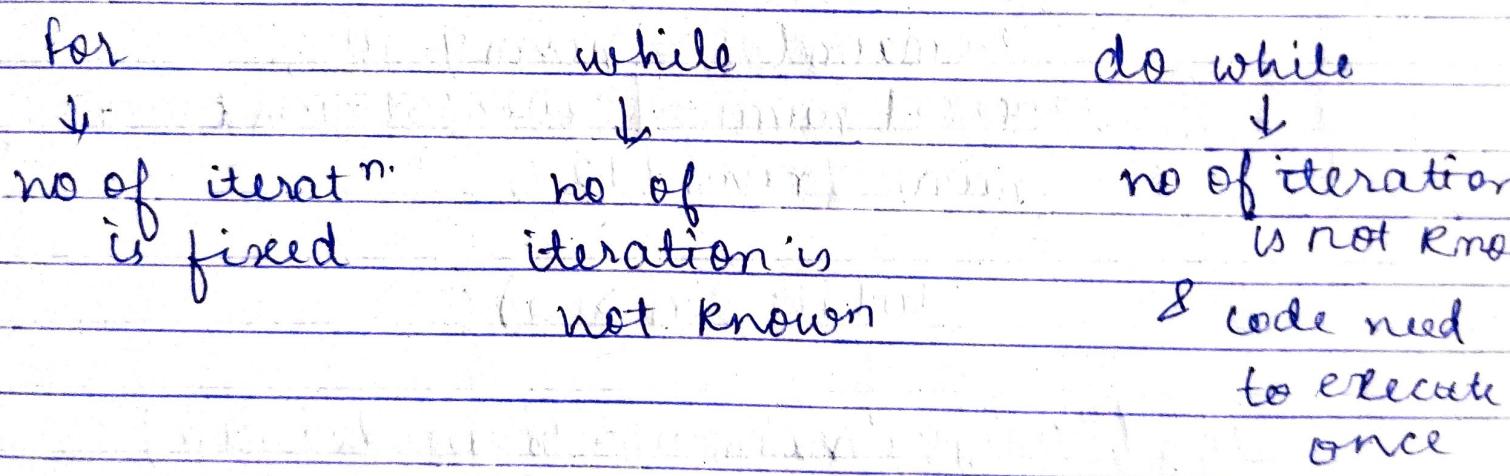
```
float n, s = 0;  
do {  
    puts ("Enter a number");  
    sf ("%f", &n);
```

sum = sum + n;

```
}  
while (n != 0);
```



which loop is better?



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

If original no = $\boxed{23}$ reversed no.

for eg (21, 133)

$$121 = 121$$

$$1331 = 1331$$

```
int main()
```

int num, original num, remainder; reversed
no=0;

```
if ("Enter the num");
```

```
if ("1.d", &num);
```

```
else
```

original num = num;

```
do
```

```
{
```

~~reversed num = 0~~

~~original num = 10 * no~~

remainders = num % 10;

reversed num = (reversed num * 10) + remainders;

num = num / 10;

```
}
```

```
while (n > 0);
```

```
if (original num == reversed num)
```

```
pf ("Palindrome");
```

```
else
```

```
pf ("Not palindrome");
```

```
return 0;
```

```
}
```

Let say 121

$$① R = 121 \% 10 = 1$$

$$121 / 10 = 12 \quad \underline{1}$$

$$② R.N = 0 * 10 + 1 = 1$$

$$1 * 10 + 2 = 12 \quad \underline{12} \quad 1$$

$$③ n = 121 / 10 = 12$$

$$2 * 10 = 0 \quad \underline{0}$$

$i < 4$ $i = 4$ $i = 1$ $i = 2$ $i = 3$ $i = 4$ $i = 5$
 $i++$ $2 < 4$ $3 < 4$ $i = 3$ 1 $2 < 4$
 $i = 1$ $i = 5$

$(i++) \leq 5$ $(3++) \leq 5$
 $2++$ $4++$

$(8++)$

2 4 6

$i = 49$ $i \leq 57$ $i++$

$49 \leq 57$

$49 - 1 \rightarrow \text{true}$
 $50 - 2 \rightarrow \text{false}$.

$51 - 3$

$52 - 4$

$53 - 5$

$54 - 6$

$55 - 7$

$56 - 8$

$1 \rightarrow \text{true}$

$0 \rightarrow \text{false}$.

$\frac{12}{10} \frac{1}{2}$

$\frac{10}{10} (1) \quad (5--)$

$\frac{-10}{2}$

$\frac{12}{10} \frac{1}{2}$
 $\frac{2}{20}$

5 4 3 2 1

$$R = 1210 \div 10 = 1$$

$$R = 2$$

$$R = 1 \div 10 = 1$$

$$R \cdot N = 0 * 10 + 1 = 1$$

$$R \cdot N = 1 * 10 + 2 = 12$$

$$R \cdot N = 12 \div 10 = 1$$

$$N = 121 \div 10 = 12$$

$$N = 12 \div 10 = 1$$

$$N = 1 \div 10 = 0$$

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++)
```

```
{
```

```
    if ("Enter number : .d:", i);
```

```
    if ("%lf", &n);
```

```
    if (n < 0)
```

```
        break; // goto my_Label;
```

```
    if (number > 1000)
```

```
{
```

```
        puts ("Number greater than 1000")  
        continue;
```

```
}
```

```
    sum += number;
```

```
?
```

myLabel:

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

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

Arrays: (Collection of values of
same datatype)

```
int numbers[] = { 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[ 7 ] = { 1, 2, 3, 4, 5 };
```

Method 3:

```
long scores[ 3 ] ;
```

```
long score[ 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("1. d", &scores[i]);
        scanf("%d", &scores);
    }
```

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

```

#include <stdio.h>
int main()
{
    int a[10], i = 0, largest = 0;
    while (i < 10)
        pf ("Enter number: %d", i);
        if ("%d", &a[i])
            i++;
        if (a[i] > largest)
            largest = a[i];
        i++;
}

```

Memory Allocation:

Elements are stored at contiguous memory location

```

int num[4] = { 0 | 34 | 51 | 23 | 12 | } ;
              1   2   3
              100 104 108 112

```

Exploring 2-D Arrays

↓
An array containing diffⁿ 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.

```
1/ row for (int r = 0; r < 3; r++)  
{
```

```
1/ column for (int c = 0; c < 5; c++)  
{ pf (" %d ", 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

// for input

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

~~Sum = Sum + arr[r][c]~~
} .

```
for( int r=0 ; r<2 ; r++ )
```

```
{  
    for( int c=0 ; c<3 ; c++ )
```

~~scanf("%d", arr[r][c])~~
} .

```
    pf( "%d" , arr[r][c] )
```

```
{  
    pf( "\n" )
```

```
.
```

* Arrays have a fixed size.

Strings

A string is represented as an array of characters.

```
char str[3] = "hi";
```

0 1 2
char str[3] = { 'h', 'i', 'o' };

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', 'o' };

// user input

```
char name[40];
puts ("Enter your name:");
scanf ("%s", name); space
until it encounters a newline
we don't use & because it refers to the
base address of array.
```

[^ \n]*%c → reads until it
enters ~~a~~ 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";

a = Hello world

char b[50] = "World";

b = World

Strcat(a, b);

// Compares 2 strings lexicographically

strcmp();

strcmp(~~a~~, a, b);

O/P (if true)

0

O/P (if false)

non-zero no.

(difference betⁿ ASCII
value of 2 character
i.e. unequal)

// Copy one string to another.

strcpy();

O/P

strcpy(a, b);

a = world

b = world

II length of string

strlen();

O/p = 5

strlen(a);

WAP to

- a) append second string at the end of
first with add n. space

strcat(a, " ");

strcat(a, b);

functions

Built in (library
funcn)

scanf(), printf() of stdio.h.

User-defined

Created by programmer
while writing

strlen(), strcat() of string.h.

↗ return type
 ↗ formal arguments / parameters
 ↗ function definition

```

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

void sum(int , int); → function prototype / declaration

```

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

↗ actual arguments / parameters

Use of funcⁿ:

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

```
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 funcⁿ: 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

* ~~int main()~~

```
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 = s + arr[i]
```

```
avg = s / size;
```

```
return avg;
```

```
}
```

↑

WAP Using array with function to find no. of odd nos.

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

```
int main()
```

```
{
```

```
int n[10] = {1, 3, 5, 7, 9, 11};
```

```
↳ c(n, 10)
```

```
, return 0;
```

```
void c(int n[], int)
```

```
int co = 0,
```

```
for (int i = 0; i < size; i++)
```

```
    if (n[i] % 2 != 0)
```

```
        co +
```

```
    pf(c);
```

~~$x = 10$~~ ~~$i = 10$~~
 ~~$i++;$~~ ~~$i = 10$~~
 ~~$t+1;$~~ ~~$i = (t+1) = 12$~~
 ~~$i--;$~~ ~~$i = 12 - 1 = 11$~~
 ~~$i = i * 10;$~~ ~~$i = 11 * 10 = 110$~~
 ~~$--i;$~~ ~~$i = --110 = 0$~~
 ~~$i = i / 10;$~~ ~~$i = 110 / 10 = 10$~~
 ~~$508 * 10$~~
~~508~~

Recursion

A process where funcⁿ calls itself

Stack overflow exception (error) \rightarrow when
 stack memory get full and gets exhausted:
 \downarrow

Call info of funcⁿ:
 each time when
 the funcⁿ
 is called)

```

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

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

~~Start~~

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

long fact(int n);
if ("Enter the number")
    if ("1.1d", &n);
long y = fact(n);
printf ("The factorial is %.1d", y)
```

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

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34 \dots$

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

int fib (int);

int main()

5

int n;

$\psi(n)$

$s^1(n)$

```
int result = fib(n);
```

of (result)

return 0;

3

int fib (int a)
{

if ($a \neq 0$ || $a = -1$)

between a;

20

```
return fib(n-1)+fib(n-2);  
}
```

Advantage

-) used in data structure
tree traversal, sorting

Disadvantage

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

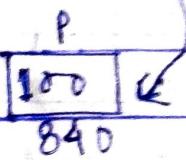
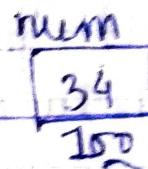
World .

Pointers

* → Value at
(Indirection operator)

~~datatype
should
be
same~~ int num = 34;

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

a = &x

int *b

b = &y

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

Increment & decrement operator

int num[4] = { 34, 51, 23, 12 }
 ^ 0 1 2 3
 | 100 104 108 112

int *p ;

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

or

p = &num[0]

// p = 100

*p = 34

p++;

p = 104 or *p = 51

↳

It shift to next memory location.

p = p+2;

p = 112 or *p = 12

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

Decrementing operator

$\text{ptr}--;$ $\text{ptr} = \text{ptr} - 1.$ // immediate previous memory location in array.

Passing Pointer to a function.

```
#include <stdio.h>
double gA(double *ptr, int size)
{
    double avg, sum = 0;
    for(int i=0; i<size; i++)
        sum = sum + *ptr;
    ptr++;
    return avg;
}
```

int main()
{
 float n[5] = {9.8, 3, 4, 2};
 float *p;
 p = n;
 double avg = gA(p, 5);
 if(avg)
 return 0;
}

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 A.S.

Memory Management

Static Memory Allocation

(At compile time

i.e. before execution)

int x = 24; // 4 bytes

float score [100]; $100 \times 8 = 800$ bytes

↓
size of array
(fixed)

100	1	2	3
	4	5	6
800			

} whole
memory
is
consumed

Dynamic Memory Allocation

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

#include <stdlib.h>

(Standard
library)

null = no memory exists

StlLib

- | | |
|-------------------------------|-----------|
| — memory allocation | malloc() |
| — cont'd. generous allocation | calloc() |
| — De-allocate memory | free() |
| — Re-allocation | realloc() |

malloc()

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

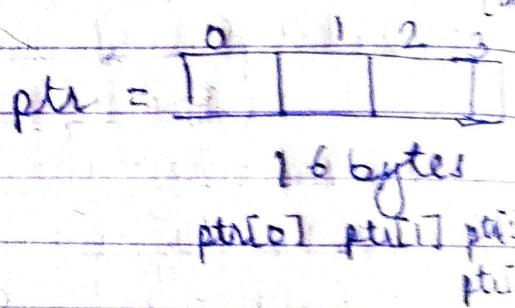
```
#include <stdio.h>
#include <stdlib.h>
```

```
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);
```

malloc()

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

The value of each element is [0 | 0 | 0 | 0].
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:");
    scanf("%d", &n);
    p=(float*)calloc(n, sizeof(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;
ptrs ("Enter the no. of more value")
sf (c)
```

$$\text{newsize} = c + n;$$

```
(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[i] = 0;
```

```
int main()
```

```
{  
    int n, *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

Structure

↳ (grouping of entities of diff'n datatypes)

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

```
int main()  
{
```

```
    struct Employee e1;  
    e1.id = 1;  
    e1.name = "Peter" × 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

char name[20]; // 20 bytes

// 26 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;

printf("%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

}

↓

↓

z

// in capital letters

RED = 10 , YELLOW = 11 , GREEN

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 specifier

to print the actual memory address

void

l-p (hexadecimal no)

14132678 → hexadecimal
0x7750

Passing by value and Pass by Reference

O/P

```
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)

{
int i = 1;

A constant can't be used in switch statement,
as it is a reserved word.

Passing By reference

void changeValue(int *x);

```
int x = 10;
```

```
if ("The value of x before *x.d", x);
```

```
changeValue(x);
```

```
if ("The old new x is *x.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 function

outside
↑ "function"

local variable prioritised more than global.

Random function:

↓

to generate random numbers.

Q.W.A.P to roll a dice

Random function: rand(), srand(time(NULL))

#include < stdlib.h >

~~int main()~~

#include < stdio.h >

#include < time.h >

#include < unistd.h >

srand(time(NULL));

randomValue = (rand() % 6) + 1;

printf("You got: %d", randomValue);

main

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

$n_1 = 10$

$n_2 = 20$

30

$10 + 10 = 20$

(1)

$n_1 = 10$

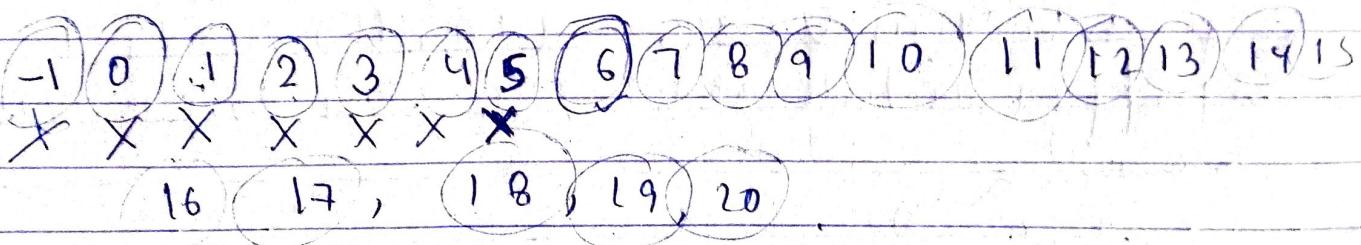
$n_2 = 20$

$30 = 30$

$20 + 20 = 40$

-1

20



5 < 5

(1)

22 12
7
16
5
0

.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 stream
↳ no . h file extension.

int main() {

 std::cout << "Hello Internshala" << endl;
 | ↓ ↓
 standard second insertion
 | ↓ operation
 return 0;
}

std::

↓

endl
is used
for new
line

std is getting repeated again & again
so we use

using namespace std;

use std wherever 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;

↑
farm House

↓
Space as a terminating char

↑
Opp
screen

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;`

`b0||b1 = x > y;`

a/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);`

same funcⁿ

`void add (string, string);`

name but

`void add (int, int, int);`

different

the return type doesn't play a role

parameters

`add (10, 20);`

`add ("Hi", "you");`

`add (10, 20, 30);`

(1+) * 20

Data types

Primary User defined Derived.

char
int
long
float
double
bool

Array
pointers
string

Structure
Union
Enum
classes

Class And Object

class Dog	
Properties	(Behaviour)
breed	bark()
age	Sleep()
color	eat()
Variables	(methods)

real world entity having state & behaviour

dog object 1
Dalmatian
5
white-black

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;       // datamembers  
    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 Outside
a class a class

```
public:  
    int id;  
    string name;  
    string getfullname() { // No use of funcn.  
        return first name + last name; } declaration  
    };
```

emp1.getfull.Name()

Outside a class.

void displayDetails

Inside

void displayDetails();
↓
funcn. declaration

Outside.

void Employee::displayDetails()
↓
scope resolution

cout << id;

since it is part of class

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 () {, //outside //definition

cout << "Employee"

}

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;

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 O₁ to O₂

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)

↓
destroys 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();`

definition:

`Employee :: ~Employee()`

`cout << "Employee object with id: " << this->id`
`& " being deleted";`

Inheritance

Class A

↑
class B inherits
properties from
Class B Class A

(Parent / base / superclass)

Animal.

class Animal {

public:

int age;

String color;

void run() {

}

};

(derived / child)

Dog :

Lion

Class Dog : public Animal {

public:

String petName;

void bark()

{

} ;

Class Lion : public Animal

{

public:

String name;

void roar() { }

{

} ;

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

```
using namespace std;
```

```
class Animal {
```

```
public:
```

```
int age;
```

```
string colour;
```

```
void run {
```

```
cout << "Running" << endl;
```

```
}
```

```
Lion
```

```
class Dog : public Animal {
```

```
public:
```

```
string petName;
```

```
void bark()
```

```
{
```

```
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
Sub class

Multilevel

Class Animal

superclass

class Dog

Subclass

class puppy

~~base~~
child class.

Hierarchical Inheritance

class Animal (one superclass)

class Lion

class Dog

multiple sub-classes

Multiple

(multiple superclasses)

class Person

class Employee

class Teacher. (one sub-class)

base

class Person

class Employee;

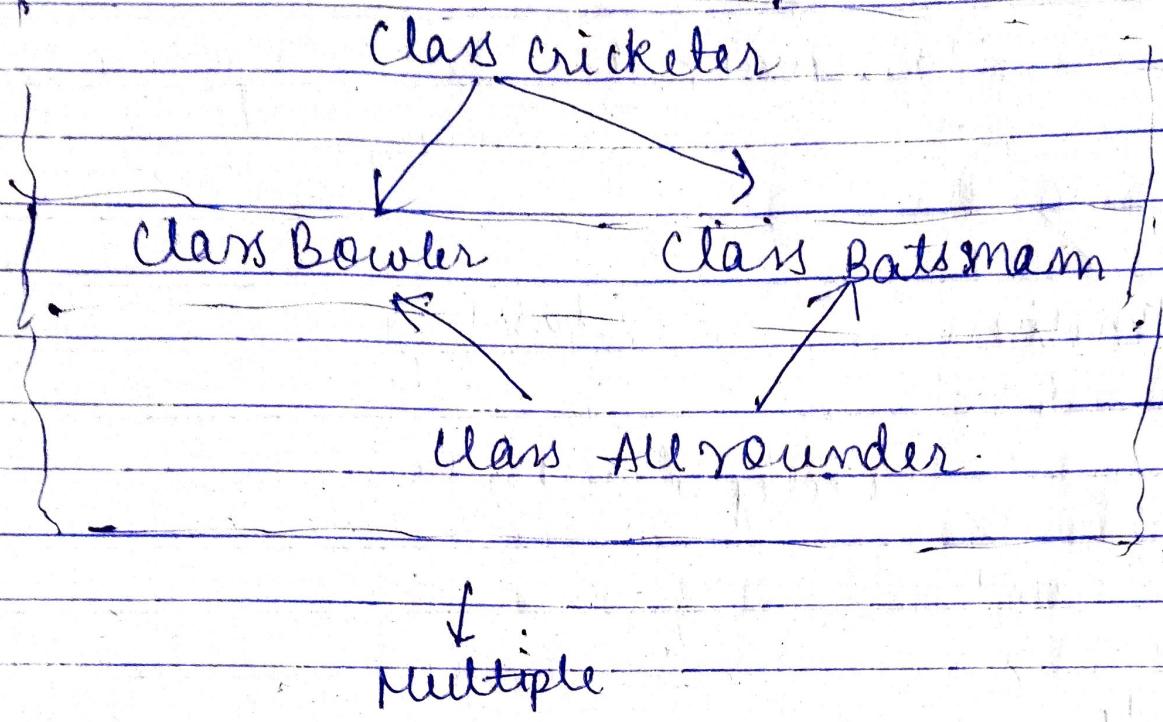
{

};

class Teacher : public Person, public Employee

{

Hybrid (2 or more inheritance)
↳ hierarchical



class <Sub class Name> : public <super class>
Name
{ };

Access Specifiers (whether a funcⁿ can access the attributes of other)
↳ modifiers
Public Private Protected

Private & Public access Specifiers

public: // accessible outside the class.

float radius;

double area();

private: // is not accessible 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);

User can set the radius but not access the radius.

default is private

Data hiding



As we made it private

(outside world object.)

has no idea about that attribute)

Hiding data outside the class

Abstraction &



method getArea()
setArea()

area of the object.

Just hiding how the things are implemented.

Hiding the implementation details

Encapsulation



grouping similar

data & function

in a group

like we wrap similar

data & methods in a single unit

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

It can be accessed by inherited class

Object

Polymorphism (behave differently
in different situations)

poly + morphs

many forms

One name & many forms

Woman → Mother, Employee, customer

Types of polymorphism

Compile Time

function overloading
(Method)

Run-time

function overriding
Virtual function

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.sound();
```

dog → Dog object

) no of parameter should be same
*) same name

Virtual function

```
#include <iostream>
```

```
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 overridden 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 Δ

* All interface are abstract classes.

Friend Class & Friends Function

(90+ reserved keywords in C++, like

to access private attribute of other class

Friend class can;

```
class Employee {
```

```
private:           // or protected:  
    string phNo;
```

```
public:
```

```
    string name;
```

```
void SetPhoneNumber(string phoneNum)
```

```
{  
    this->phNo = phoneNum;
```

```
}
```

Friend class can;

```
class Car {  
public:  
    string carName;  
  
void display (Employee emp)  
{  
    cout << "Employee name: " << emp.name <<  
    emp.phoneNo << "Car name: "  
    << carName  
}  
};  
  
int main () {  
    Employee employee;  
    employee.setPhone Number ("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 << phNo
endl

}
}

Miscellaneous Concepts

Reserved keywords (90+ reserved keywords)

Pascal Case

e.g.: MyAge

Diving into C++ programming

Namespaces (add "info" to diff. betn. func's, 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 << "Yuan"  
        << endl;
```

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

The using directive

```
using namespace std;  
↓      ↓      ↓  
directive reserved variable name  
keyword
```

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

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	<math>

part of O.S not
Std library

Math library

#include <cmath>

→ Similar to array but
in vector you can insert
as many elements you
want
to.

Vector library

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

```
int main() {
```

```
    vector<string> names;
```

↓ ↓
datatype variable.

```
    name.push_back ("Rahul");
```

```
    name.push_back ("Peter");
```

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

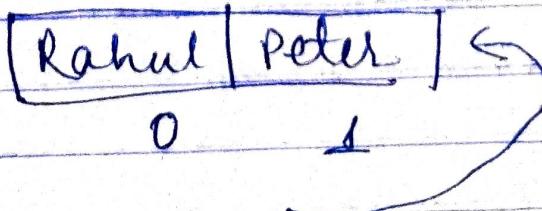
// Accessing an element.

```
cout << names[0] << endl;
```

// Remove element:

```
name.pop_back();
```

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



Preprocessor directives

(macro)

when anything is constant

define UPPER LIMIT 10 // Macro definition

↓ ↓ ↓
use to macro macro
define template expansion
a macro

* All letters should be Capital & joined by -

Macro with parameters.

define AREA(u) (3.14 * u * u) // Macro with parameter

↓
No datatype required
(Any datatype is accepted)

cout << AREA(5);

funcn & macro with parameter

↓
one line of code
faster

→ to preprocess the
source code before compiling

Commonly used preprocessor

Macro

define

file inclusion

include

File inclusion

include <iostream> // file inclusion
definition

Creating own header file

include "filename"

Exception Handling

(Error)

Types of Error

Compile-time

C/C++ → Compiler
(Program)
(missing of ; or
comments)

Run-time
(exception)

0101010
00,1110
110000

Handling Exception

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

```
int main()
```

```
{
```

```
int a, b;
```

```
cout << "Enter a";
```

```
cin >> a
```

```
cout << "Enter b";
```

```
cin >> b
```

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

i/p

10

$$10 \% 5 = 2$$

b:

0/p

2

i/p

2

$$2 / 0 = \infty$$

0

0/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 s1 ;  
int r = 5.7;  
int p = 100;  
int t = 1;
```

s1 = 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

open 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");
```

↓
kebab case

if not present
it would
be created

// Write on the file

```
ofile << "Hi"
```

```
ofile << "Internshala";
```

// 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); // Read a line
cout << str;

}
ifile.close();

Miscellaneous Concept

Programming Paradigms

A way to classify programs based on their features

↓
OOP's

Object Oriented Programming

Program is divided into small parts called objects

inheritance

function overloading

Access specifiers more secure

Data is important than funcn

C++, Java, Python

↓
P.O.P.

Procedural Programming

functions

no inheritance

Not supported

Not present

funcn is more imp than data

C, FORTRAN, C++
Pascal

Hybrid language

P L a v ! p : a
a g . 1 L a g g o l ?

C vs C++

C

- i) POP
- ii) Not supported
 - a) Inheritance
 - b) OOP's concept
- iii) 32 keywords
- w) No exception handling

C++

- i) POP + OOPS
- ii) Supported

- iii) More keywords

Syntax
(Errors)

Source code → Preprocessor → Compiler
(Removes comments)
Expand directives

↓
myfile.i

Assembler
(myfile)

myfile.obj

linker

myfile.exe

Program
running

loader