C++

* C++ originated from C.
  - company in-charge is Bosland Corporation
  - no commands; only statements

C (originated from Unix)

- "King of Languages"
- Used to rewrite Unix

* C++ was created by Bjarne Stroustrup.

* LANGUAGES
  - SUP FEATURES → DATA
  - SUP FEATURES → FUNCTIONS
  - TYPES

PROCEDURAL	OOP
[GIVES PRIORITY TO FUNCTIONS]	(OBJECT-ORIENTED PROGRAMMING) [PRIORITY TO DATA]
→ no data-security	→ you deal with copies of the data; there is data-security
eg: C, COBOL, Pascal	eg: C++, Python, Java
– Commands	– Statements

TOKENS

KEYWORDS
(OR RESERVED WORDS)
(MOSTLY IN SMALL CASES)
→ action words
→ do a specific task
→ case-sensitive

IDENTIFIERS
(A NAME THAT IS GIVEN TO)
VARIABLES | CONSTANTS

NAMING RULES:
* Keywords cannot be used
* C++ is case-sensitive
* No special characters, except _ (underscore)
* No spaces
* No nos. at the beginning
* Max - 8 characters

STRUCTURE OF A PROGRAM

Every program starts with `#include<iostream.h>` (keyword)
→ SPACE → SPACE
Preprocessor directive

LANGUAGE INTERPRETERS

COMPILER
→ Higher Level Lang. to Machine Lang
→ translates code at one-go
→ error detection is harder (disadv.)
→ Processing speed is faster. (adv.)

INTERPRETER
→ HLL to Machine
→ translates code line-by-line
→ error detection is easier (adv.)
→ Processing speed is lower. (disadv.)

ASSEMBLERS
→ Assembly Lang. to Mach.

→ COMPILER → main( ) → BODY

C++ uses compiler language interpretation.

PREPROCESSOR DIRECTIVE #

→ is a instruction to compiler to do a specific task

~~#INCLUDE <IOSTREAM.~~
<iostream.h> → header file (in library)

→ file that is required for C++ input & output.
→ not a keyword; it is a filename

~~VOID F~~
void main( ) (main function)

→ keyword → Driver function of a program
→ 'Heart of a program' i.e, most important part of the code
→ consists the body of code

BODY OF THE CODE

{ } NOT a keyword

"PUT TO" OPERATOR
(COUT) <<

OVERALL STRUCTURE

#include <iostream.h>
void main( )
{ cout << " " ; → terminator
}

EXTRACTION OPERATORS >>
(CIN)
"GET FROM" OPERATOR

* Write a program to display your name.

```
#include <iostream.h>
void main( )
{ cout <<"Hi ! Welcome" ;
  cout <<"Thakur" ;
}
```

* Write a program to display F.R. EI.N.D.S Names.

```
#include <iostream.h>
void main( )
{ cout <<"Ross" ;
  cout <<"Joey" ;
  cout <<"Chandler" ;
}
```

* Write a program to ~~input 2 nos.~~ and find their sum of 2 nos.

```
#include <iostream.h>
void main( )
{ int A= 5 ;
  int B = 7 ;
  int C= A+B ;
  cout << C ;
}
```

Variable declaration

= → Assignment operator

OPERATOR is a symbolic representation of an operation
eg: + − / *

```
#include <iostream.h>
void main( )
{ int A, B, C ;
```

DATA TYPES

BASIC

keywords
- * int → integers (no decimal points) → 2 bytes
- * char → characters → 1 byte
- * float → decimal → 4 bytes
- * void → NOTHING

char X = 'a' (single quotes)

EXTENDED DATA TYPES

long int
- long (4 bytes) ↓ for int

double float
- double (8 bytes) ↓ for float

short int
- short (2 bytes) ↓ for int

* Write a program to input 2 nos & find their sum.

```
#include <iostream.h>
void main( )
{ int A, B, C;
  cout<<"enter that no.";
  cin >> A;
  cout<<"enter value of B";
  cin >> B

C = A + B;
cout<< "sum of the 2 nos. you have given
      is"<< C ;
}
```

2 << are used.

Cascading of operators
- When << is one operator is used on 1 line.

```
#include <iostream.h>
void main()
{  cout << "Enter side of the square";
   cin >>
{  int A, B;
   cout << "Enter side of square";
   cin >> A;
   B = A*A;
   cout << B;
   cout << "The area of square is" << C;
}
```

OPERATORS

They are symbolic representations of an operation.

eg: + − * / %

^	do not work
X	
.	

* Write a program to input principle, rate & time, & find simple interest.

Write a proo

* Mose: * → Area of

ENDL

Causes the next line.

SETW( ) → iomanip.h

ALLOCATES 10 BYTES (leaves 10 spaces)

ESCAPE SEQUENCES

→ also called as output manipulators
→ All escape sequences occupy 1 byte of space
→ character sequence
→ enclosed in ' ' or " "
→ preceded by \ (not /)
→ They work with cout only.
→ They \n → newline character (same as endl)
\b → backspace character
\t → tab character - leaves 8 character space
\a → alert character
\0 → null character

→ eg:

CODE	OUTPUT
cout<<"Hi"<<endl<<"there"<<endl<<"today"	Hi There Today

SHORTER VERSION

(OR)

CODE	OUTPUT
cout<<"Hi\n there\n today";	Hi There Today

cout << "com \b

cout << "Hi \n welcome \b to \t \t the \n first \b class \b of C \n + \n + \n + ";

↓

Hi
welcom to the
first class of C
+
+
+

CHARACTERS VS STRINGS

CHARACTERS	STRINGS
char ch = 'A'	char S = "Hello"
	↓ String is a collection of characters
→ single quotes	→ double quotes
→ 1 byte of memory	→ amount of memory = (no. of characters × 1 byte) + 1 byte (\0 (null character))
	= in this case (5 × 1) + 1 = 6 bytes

'A' → 1 byte
"A" → 2 bytes (as it is "A\0")
'\a' → 1 byte (as escape sequences are 1 byte)

COMMENTS

→ are non-executable statements
→ used to increase comprehensibility of the program → used for debugging/testing purposes.
→ do not use memory.
→ 3 types of comments

SINGLE-LINE → //
MULTI-LINE → /* */
INLINE → /* */ eg: int x=5; // Blah

OPERATORS AND EXPRESSIONS

TYPES → MATHEMATICAL

OPERATORS AND EXPRESSIONS

TOPICS
- → MATHEMATICAL
- → RELATIONAL
- → INCREMENT / DECREMENT
- → LOGICAL
- → TERNARY
- → SHORTHAND EXPRESSION

MATHEMATICAL OPERATORS

+ (add)	− (Subtract)	* (multiply)	/ (divide)	% (Remainder operator / Modulus / Mod)
			→ Division operator → Returns quotient	→ Calculates remainder → Works only with int

Smaller no. (int) divided by a bigger no. (int) → 0

Smaller no. (int) mod by a bigger no. (int) → Smaller no.

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

Quotient = 0
Rem. = 1 (smaller no.)

* Write a program to input 2 nos. and simulate (try out a calculator

Ans.

RELATIONAL OPERATORS → used to state conditions

<	>	<=	>=	== (equality)	!=
(less than)	(Greater than)	(less than or equal to)	(Greater than or equal to)	↓ (OPERATOR) checks whether the value is the no. eg: a == 10 Checks if a is equal to 10	(Not equal to)

REQUIRES ATLEAST 2 conds.

↳ if true, output is 1.
↳ if false, output is 0.

LOGICAL OPERATORS → work with relational operators to give out 1 or 0 (1 → true, 0 → false)

UNARY OPERATOR ↑ BINARY OPERATORS ↑

! (not)	&& (And)	|| (or)
→ !F = T	→ all conds. must be true, then it gives 1 (true)	→ any one of the conds. are true, output is 1 (true)
→ !T = F		

HIERARCHY
* NOT
* AND
* OR

=	VS	==
→ ASSIGNMENT OPERATOR		→ EQUALITY OPERATOR
→ used for assignment		→ checks if 2 values are equal.
→ eg: Put a = 10		→ eg: x == y

EVALUATE:

1) (x<=y) && ((y>=z) || !(z>0) && (x==z))

(i) If x=10, y=12, z=10.

Ans. → (10<=12) && ((12>=10) || !(10>0) && (10==10))

→ (T) && ((T) || !(T) && (T))

→ T && (T || F && T)

→ T && (T || F)

→ T && T

→ T

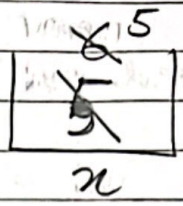
(ii) ~~x=10, y=2, z=5~~

Ans. ~~(10<=2) && ((~~

Please complete

INCREMENT / DECREMENT OPERATORS

```
int x = 5;
x = x+1;
x = x-1;
```



SHORTHAND

- Increment
  - x++ (post-fix increment)
  - ++x (pre-fix increment)
- Decrement
  - x-- (post-fix decrement)
  - --x (pre-fix decrement)

POST-FIX → updated in mem. after processing / displaying

PRE-FIX → add/subtract before processing / displaying

eg: `int x=5, y=3, z=10, A, B;`

`cout<<x<<y<<z<<endl;`

`cout<<++x<<++y<<++z<<endl;`

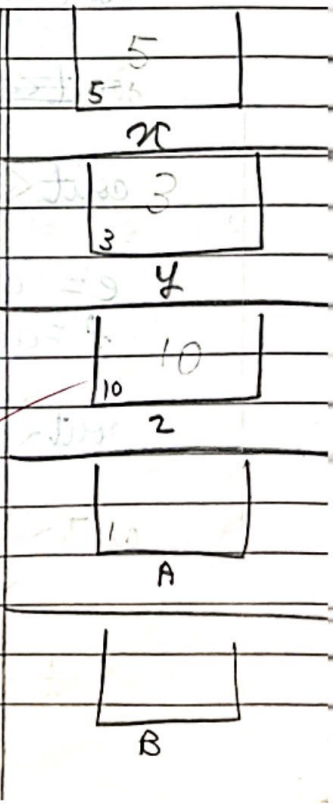
`cout<<x++<<y++<<z++<<endl;`

`A = x+y-2`

`B = y+z`

`cout<<A++<<++B<<x++ <<--y<<z--<<endl;`

`cout<<--A<<++B<<--x <<y++<<++z<<endl;`



HIERARCHY OF MATHEMATICAL OPERATIONS

EXPONENT ^

↓

BRACKETS ( )

↓

MULTIPLICATION AND DIVISION [WHATEVER COMES 1ST IS HIGHER]

↓

ADDITION AND SUBTRACTION [WHATEVER COMES 1ST IS HIGHER]

eg: int a=5, b=10, c=8, d, e, f;

```
cout << a << b << c << endl;

cout << ++a << b++ << --c << endl;

cout << a++ << b++ << --c << endl;

cout << ++a << ++b << c-- << endl;

cout << a-- << --b << c++ << endl;

d = a+b-c;

cout << d++ << a++ << ++b << --c;

e = a+b-d;
f = a+b+c;

cout << e++ << --f << ++d;

cout << a++ << ++b << --c
     << d-- << e++ << --f;

cout << a++ << ++b << --c
     << d--
```

```
cout << ++a << --b << c++ << --d
     << e++ << ++f;

cout << a << b << c << d << e << f;
}
```

SHORT HAND EXPRESSIONS

```
{int x=10;
 x= x+5}
```
Same as
```
{int x= 10;
 x += 5}
```
* means that 5 is added to x, and the result is stored in x
* in other words, the no. to be added to x is 5.

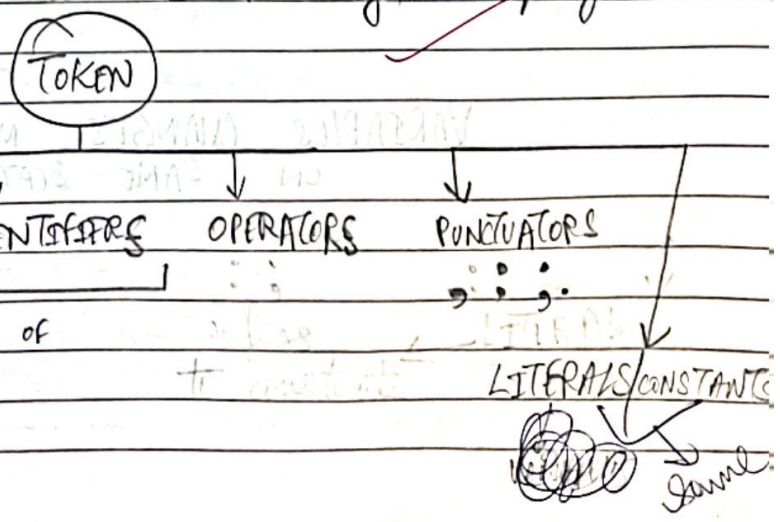
Make sure using shorthand each. has an assigned value

```
{int x=5, y;
 y += x;
 cout << x << y;}
```
→ 5 -356 some random no as y is not assigned

```
{int x = 5, y=10;
 y+= x; cout << x << y;}
```
→ 5 10 ✓

TOKENS

Token is the smallest block of C++ program.



const int a;
cin>>a;

same

CONSTANTS / ~~LITERALS~~ LITERALS

→ Constants are values that cannot be changed.
→ declared using keyword "const".
→ declaration must be in one go.
→ eg: const int a = 5; ✓ || const int a; a=5; ✗ || int a; const a=5; ✗

* Write a code to input radius of circle, and find and output the radius.

Ans.
```
#include <iostream.h>
void main(){
    const float pi = 3.14;
    float r, area;
    cout << "Enter radius"; cin >> r;
    area = pi*(r*r);
    cout << area;
}
```

EXTENDED DATA TYPES

long int → long (4 bytes) → int
EXTENDED DATA TYPES
long double → double (8 bytes) → float
int → short (2 bytes) → short int

IMP

VARIABLES CHANGES MULTIPLE TIMES ON SAME STATEMENT

When the same variable changes 2 or more times in the statement (not necessarily same line) the execution goes from right-to-left but display goes from left-to-right.

eg. `int x = 2, y = 3;` `cout << x++ << y++;` } Execution from left to right, display from left to right

```
int x = 2, y = 3;
cout << x++ << ++x << ++x << y-- << y--;
```
→ Execution from right to left, display from left to right

* Write a program to input a, b, c of a quadratic equation, and hence find its discriminant.

Ans.
```
#include <iostream.h>
#include <math.h>
void main() {
    int a, b, c, d;
    cin >> a >> b >> c;
    d = pow(b, 2) - (4 * a * c);   // arguments
    int r1 = (-b + sqrt(d)) / (2*a);
    int r2 = (-b - sqrt(d)) / (2*a);   // for ±
```

* Write a program to input P, r, and n, and find the compound interest.

$$I = P \times \left(1 + \frac{r}{100}\right)^n$$

Ans.
```
#include <iostream.h>
#include <math.h>
void main() {
    float
```

~~SOME SPECIAL CASTING~~

IMP HEADER FILES

→ <iostream.h> → cin cout
→ <math.h> → pow( ) sqrt( )
→ <conio.h> → clrscr( ) getch( )
→ <process.h> → exit( )
→ <stdlib.h> → exit( ) random( )
→ <ctype.h> → isdigit( )

TYPE CASTING

IMPLICIT (AUTOMATIC)

→ Type casting refers to the concept in which the data type of a given variable for a momentary process is changed.

eg: int x = 5;
float y = 2;
cout << x/y ⇒ 2.5
(float)

→ Changes to data type of higher memory.

EXPLICIT (USER-DEFINED)

int x = 5;
float y = 2;
cout << x / int(y);

TYPE → EQUIVALENT INTEGER → made into Binary form by OS

ASCII VALUE
American Standard Code for Information Interchange

```
char ch=65;
cout<<ch;
```
A

```
int a=65;
cout<<(char)a
```
A

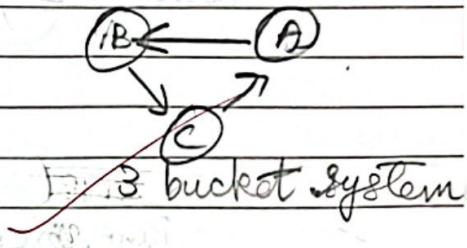
A→65
B→66
C→67
...
a→97
b→98
c→99
...
0→48
1→49

IMP

* Write a program to input 2 nos. and exchange them.

Ans Method 1

```
...
...{
int A,B,C;
cin>>A>>B;
C=B;
B=A;
A=C;
}
```

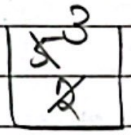


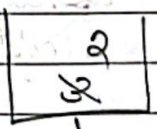
3 bucket system

Method-2

```
...
...{
int a=2, b=3;
a=a+b;
b=a-b;
a=a-b;
}
```

* ~~Find value of:~~





* Find value of:

(i) cout << 8 + (6/3/2) + 4 - (4 * 5/2) - 2;

value = 8 + 1 + 4 - 10 - 2
= 13

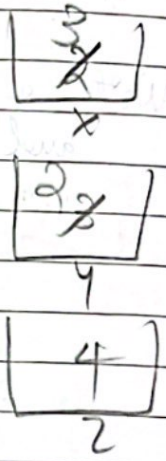
* ~~Find~~

(ii) int x = 2, y = 3, z = 4;

z += x++ + --y;
4 2 2

cout << z;

z = 4 + 2 + 2
= 8



~~SIZE~~ sizeof() function

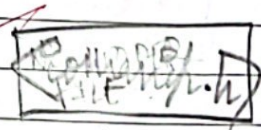
sizeof() function

→ used for finding memory usage of something

→ eg: int a = 5; float b = 5.5;

cout << sizeof(a) → 2

cout << sizeof(b) → 4



DATA TYPE MODIFIERS

UNSIGNED ← data that can't take signs

SIGNED → data that can take signs

TERNARY OPERATOR

→ also known as if/else operator

→ ? → if ; : → else

IF/ELSE	TERNARY OPERATORS
if(expression) {	(expression) ? {
} else {	} : {
}	}

* Write a program to check if a no. is odd/even.

Ans.

```
#include <iostream.h>
void main( ){
    int A;
if (a%2==0){
    cout<<"Even"
}else{cout<<"odd"}
}
```

```
(a%2=0)? cout<<"even";
       : cout<<"odd";
}
```

<=10
>10 and <=20
>20 and <=30
>30 and <=40
>40

* Write a program to calculate salary received by a person given by formula: Total Sal = Basic salary + overtime, given basic sal = 2000;

* Write a program to input 2 nos. and output bigger no.
(A>B) ? cout<<A :

* Write a program to input 3 ∠s of a Δ, and check if a triangle can be formed. (A+B+C)==180

* Write a program to input age of person, and check if he can vote or not.

TMP

* Write a program to input the electricity units of a household, and hence, calculate bill payable.

TMP

* Calculate no. of notes a user has by inputting amount.

cin>>num; rem = num;
n1000 = rem/1000;
rem = n1000 * 1000;
n100 = rem/100

5839/1000 = 5

FLOW OF CONTROL

TOPICS → CONDITIONAL STATEMENTS / SELECTIONAL
→ LOOPS
→ JUMP STATEMENTS
* switch(case)
→ LOOPS
- WHILE
→ - DO... WHILE
- FOR

CONDITIONAL / SELECTIONAL STATEMENTS

if | if...else | nested if...else | if else if else | switch(case)

IF STATEMENT

→ if (expression) {
...;
...;
}

→ eg: int n=5, y=3;
if (n > y) { cout << "No. 1 is bigger"; }

This block of code only runs when the expression is true.

* Give a discount of 5%. off to everyone who buys for more than 100 Dhs.

Ans.
```
#include <iostream.h>
void main() {
float payment, cost;
cout << "Enter cost";
cin >> cost;
if (cost > 100) {
payment = (95/100) * cost
}
cout << payment;
}
```

IF... ELSE STATEMENT

→
```
if (expression) {
... ;
... ;
} else {... ;
... ;
}
```

if cond. is true, if block gets executed or else, else block gets executed.

```
if ( ) {
...
} else { ; } null statement
```

* → eg.
```
int x=5 y=3;
if (x=y) {
cout << "both are equal";
} else {
if (x>y) {
cout << "x is greater";
} else {
cout << "y is greater";
}
}
```

IF... ELSE IF... ELSE → Space

```
if (cond1){
  ...;
} else if (cond2){
  else if (condn){
  else{
  }
```

* Write a program to input a no. from user to display the day of week.

```
#include <iostream.h>
void main(){
int n;
cin >> n;
if (n=1){
    cout<<"Sunday";
} else if (n=2){ cout<<"Monday";}
  else{ cout<<"wrong no.";
}
```

* Write a program to input a letter and then display if it is a vowel or consonant.

```
# include <iostream.h>
void main(){
  char letter;
  cin >> letter;
  if((letter=='A' || letter=='a' || ...){
      cout<< letter<<"is a vowel";
  } else { cout<< letter<<"is a consonant";
  }
```

Div by: 4 True

2100 ✗ 2016 ✓ 200 false

2400 ✓ 2014 ✗

* WAP to accept 3 nos. from the user, and to print the smallest no.

IMP
* WAP for temperature conversion from Fahrenheit to celcius, and vice-versa. Program should give option to user from which unit to which we must convert.

[C → F ⇨ $C = \frac{5}{9} \times (F - 32)$]

[F → C ⇨ $F = \left(\frac{9}{5} C\right) + 32$]

SWITCH ... CASE

→ only comparing equality (==)
→ only int and char.
* ⇒ ALWAYS put break; (except for default)
* switch(exp){ label :
case 1: { cout << ; break; }
case 2: { cout << ; break; }
default: { cout << ; }
}

label
value to be tested
no. char.
NO BREAK

eg:
```
int day;
cin>>day;
switch(day){
case 1: { cout<<"Sunday"; break; }
case 2: {
case n: {
...
default: { cout<<"enter valid no."; }
}
```
VERY IMP
NO BREAK

* Write a program to input 2 values and a symbol,

IMP
* Write a program to input ~~2 nos.~~ a date, month & year; no. no.
check if it is valid date or not.
FOR LEAP YEAR: if ((y%4==0 && (y%100!=0 || y%400==0)

* WAP to accept 3 nos. from the user, and to print the smallest no.

* WAP for temperature conversion from Fahrenheit to celcius, and vice-versa. Program should give option to user from which unit to which we must convert.

[F → C ⇨ $C = \frac{5}{9} \times (F - 32)$]

[C → F ⇨ $F = (\frac{9}{5} C) + 32$]

SWITCH ... CASE

→ only comparing equality (==)

→ only int and char.

* ⇒ ALWAYS put break; (except for default)

* switch(x){ label

case 1: { cout << ; break; }

case 2: { cout << ; break; }

default: { cout << ; }

label → value to be tested → no. / char.

NO BREAK

}

eg:

```
int day;
cin>>day;
switch(day){
case 1: { cout<<"Sunday"; break; }
case 2: {
case 3: {
.....
default: { cout<<"enter valid no."; }
}
```

VERY IMP

NO BREAK

* Write a program to input 2 values and a symbol,

IMP

* Write a program to input ~~2 nos.~~ a date, month & year, check if it is valid date or not.

no. no.

FOR LEAP YEAR: if((y%4==0 && y%100!=0) || y%400==0)

LOOPS → also called as iteration

- while (entry-controlled)
- do...while; (exit-controlled)
- for (counter-controlled)

→ 3 parts of Loops:
- initialisation (int i = 5) [, allowed]
- condition (i <= 5) [, NOT allowed]
- statements (code to run)
- updation (++i) [, allowed] (use !&& ||)

WHILE LOOP (entry-controlled)

```
int i = ...;   // initialisation
while (i <= ...) { // condition
    cout << i;
    i++; // updation
}
```

initialisation must be BEFORE while loop

* Accept a no. from the user and display all even nos. from 2 to the no. entered by the user.

Ans.

```
...
int num, i = 2;
cin >> num;
while (i <= num){
    cout << i;
    i = i + 2;
}
```

* display nos. from 50 to 2.

```
...
int i = 50;
while (i >= 2){
    cout << i;
    i--;
}
```

Accept names from a user and display them until he/she enters 'q'.
letters

* Accept letters from a user until user enters '*'.

```
char letter;
int i;
cin >> letter;
while (letter != '*'){
    cout << letter;
    cin >> letter; // for using again
}
exit(0); // quitting program
```

* Accept letters from user and display vowels, until '*' is entered.

cin >> letter;

IMP

→ Once a number loop is run, the value of int changes even outside the loop.

→ eg: for(int i=0; i<=5; i++);

cout<< i; | OUTPUT 6

DO... WHILE LOOP
(exit-controlled)

```
int i = ... ; // Initialization
do {
    ... // code to run
    ... // updation
} while (...); // cond.
```

WHILE VS DO... WHILE

WHILE	DO... WHILE
→ entry controlled loop	→ exit controlled loop
→ loop only runs when cond. is true	→ loop runs once then, loop runs more times if cond is true
→ cond. checked at entry point	→ cond. checked at exit point

* Accept letters from user & display them until user enters '*'.

```
char letter;
do { cout<<"enter a letter \n";
     cin>>letter;
     cout<< letter;
} while(letter != '*')
cout<<"loop ended";
```

* Display all letters from A to Y. (ch++)

* WAP to display a menu for rectangular operations — area, perimeter and diagonal for a rect. & whose length and breadth and choice of operation.

```
#include <iostream.h>
#  "    <conio.h>
#  "    <math.h>
#  "    <process.h>
void main() {
  char ch, exit, op;
  float l, b, peri, area, diag;
  cout<<
  "Welcome to menu" << endl <<
  "1. Area" << endl <<
  "2. Perimeter" << endl <<
  "3. Diagonal" << endl <<
  "4. Exit" << endl <<
  "Enter the length and breadth";
  cin >> l >> b;
  cout << "Enter what you want (1-4)";
  // end of menu
do{
switch (op) {
  case 1: area = l*b;
    cout << area; break;
  case 2: peri = 2*(l+b);
    cout << peri; break;
  case 3: diag = sqrt((l*l)+(b*b));
    cout << diag; break;
  case 4: exit(0); break;
  default: cout << "wrong input";
}
// end of switch
```

```
cout << "\n Want to do more (y/n) ?";
cin >> ch;
if(ch == 'y' || ch == 'Y'){
    cout << "Enter choice from 1 to 4;
}
}while(ch == 'y' || ch == 'Y'); //end of loop
}
```

FOR LOOP

(counter — controlled)

do dif. between i++ & ++i

```
for(initialisation; cond.; updation){
    //code to be run
}
```

(test cond. → code to run → updation)

* Write a loop for displaying even nos. from 2-50.

```
for(int i=2; i<=50; i+=2){
    cout << i << endl;
}
```

* ~~Show all~~ WAP for printing all multiples of 5 till 100.
* WAP to show the tables of an inputted no. from 1-10.

IMP

* WAP to find a factorial of an inputted no.

(Factorial of 3 is: 1×2×3 = 6)

[Product of all nos. upto (including) given no.]

NEAT TRICK WITH FOR LOOP

```
for(int i=10; i<=20; i++);
cout << i;
```

OUTPUT WILL BE:

21

WHEN TO USE WHICH LOOP

WHILE LOOP
→ entry-controlled "
→ used when not sure how many times loop will run
→ init
cond.
statement(s)
updation

DO...WHILE LOOP
→ exit-controlled "
→ (same as while)
→ init
statement(s)
updation
cond.

FOR LOOP
→ counter-controlled
→ sure how many times loop will run
→ init
cond.
statement(s)
updation

* WAP to display: (Hollow Pyramid)

```
*
* *
* * * * * *
```

* WAP to display:

```
1 2 3 4 5
1 2 3 4 5
```
5 nos.

* WAP to display:

```
1
2 2
3 3 3
...
```
till 5

* WAP to display:

```
1 1 1 1 1
1 1 1 1 1
```
5 nos.

* WAP to find sum of: $x - \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$
(enter x & n)

* WAP to find sum of: $\frac{1}{x} - \frac{3!}{x^2} + \frac{5!}{x^3} - \frac{7!}{x^4} + \dots + \frac{x^n}{n!}$

* WAP to display:

```
*
* * *
* * * * *
```

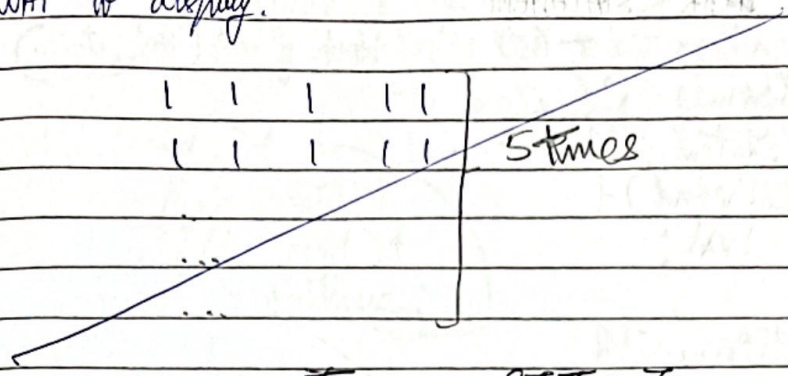
* WAP to display all prime nos. between 2 nos.
* WAP to display:

```
1
2 3
4 5 6
```

* WAP TO :

```
  --A
 -AB
-ABC
```
cout << char(64+n); → next letter

* WAP to display:



JUMP STATEMENTS

→ C++ provides us multiple statements through which we can transfer control ~~any to any~~ to anywhere in the program.

→ Jump statements changes the flow of control to a specific location by skipping statements in a program unconditionally.

→ C++ has 4 jump statements:
- ~~break~~ break – get out of block of code (loops, etc...)
- continue – skips statements below it and goes to beginning of loop
- goto – goes to particular part of code
- ~~return~~
- exit(0) – ends the program

SETW(*)

→ changes no. of characters/digits are displayed.
→ eg: int a = 12345;
setw(a) = 3;
cout << a; // output will be 123
→ HEADER FILE → iomanip.h
→ full form: setwidth
→ only works for cout <

BREAK STATEMENT

- Gets out of block of code (loop, etc...)

```
while(// cond.){
    // statements1
    if (test cond.){
        break;
    }
    // statements2
}
// statements3
```

```
for( ; ; ){
    // statements1
    if (test cond.){
        break;
    }
    // statements2
}
// statements3
```

```
do{
    // statements1
    if (test cond.){
        break;
    }
    // statements2
} while(cond.)
// statements3
```

~~WAP to display nos. from~~ from 1-5

WAP to input ~~lotto's nos~~ a no. from a user, then display nos. from that no. to 10, quit the loop when 'q' is entered.

statements

CONTINUE STATEMENT

- skip all below continue, and goes back to beginning of loop

I
```
while(cond.){
    //statements1
    if(test cond.){
        continue;
    }
    //statements2
}
```

II
```
do{
    // statements1
    if(test cond.){
        continue;
    }
    // statements2
} while(cond.)
```

III
```
for( ; ; ){
    // statements1
    if(test cond.){
        continue;
    }
    // statements2
}
```

* WAP to output nos. from 1 — 10, EXCEPT 5 & 7.

label: ← given an identifier

GOTO STATEMENT

→ goes to a particular part of code

```
label:{
// statements
}
goto label;
```

Create a calculator.

```
{int a, b;
goto choice;
cin>>a>>b>>sum;
sum: {cout<<a+b;}
dif: {cout<<a-b;}
pro: {cout<<a*b;}
quo: {cout<<a/b;}
mod: {cout<<a%b;}
if (sym=='+') {goto sum;}
if (sym=='-') {goto dif;}
...
...
...
}
```

EXIT()

- ends the program
- in <process.h> & <stdlib.h>

exit(0); → normal termination (no error)
exit(1); → abnormal termination (error)

* WAP to exit program to display nos. from 1 and exit after displaying 5.

```
for(int i=1; i<=10; i++){
    cout<<i;
    if(i==5){
        exit(0);
    }
}
```

DIGIT PROBLEMS

OVERVIEW

* → %10 → obtain last digit
* /10 → remove last digit & get rest of nos.
* while(num!=0){ → syntax for the loop

* ARMSTRONG'S NO.
  - 370 → $(3)^3 + (7)^3 + (0)^3 = 27 + 343 + 0 = 370$
  - 371

* ANY NO.
  = (ten's digit × 10) + (one's digit)

* PALINDROME NO.
  - 121 → reversed → 121

* PERFECT NO.
  - $6 = 1 \times 2 \times 3$ || $1+2+3 = 6$

* FIBONACCI SERIES
  0 1 1 2 3 5 8 13

* PRIME NO.
  - NO. HAS ONLY 2 FACTORS:
    : 1
    : ITSELF

* FACTORIAL
  $n! = n(n-1)(n-2)\ldots(n-1)\ldots \times 1$
  eg: $3! = 3 \times 2 \times 1 = 6$

*

Ans.

* WAP to display each digit on a different line.

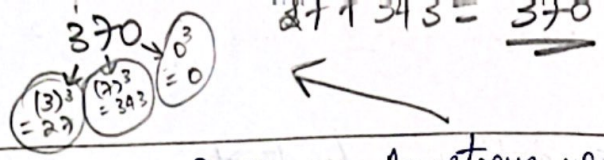
```
{ int num, digit;
  cin >> num;
  while (num != 0) {
  digit = num % 10;   // gets last digit
  cout << digit << endl;  // display digit & creates new line
  num = num / 10;   // removes last digit and gets
                    // other letters
  }
}
```

* WAP to count how many times an entered digit is present in a no.

```
{ int num, digit, digit, n;   (entered digit)
  cin >> num;
  while (num != 0) {
    digit = num % 10;
    if (digit == digit) { n++; }
    num /= 10;
  }
  cout << n;
}
```

* WAP to find sum of digits of a no.

```
{ int num, digit, sum = 0;   → make use to assign value as 0
  cin >> num;
  while (num != 0) {
    digit = num / 10;
    sum += digit;   // add value of last digit to sum
    num /= 10;
  }
  cout << sum;
}
```



* WAP to find if a given no. is an Armstrong no. or not.

```
{ int num, x, sum, t, r;
  cin >> num;
  x = num;                    // for testing at end
  while(x != 0){
    t = x % 10;
    r = pow(t, 3);
    sum += r;
    x /= 10;
  }
  if (sum == num){
    cout << " Armstrong";
  else { cout << " No";
  }
```

display if

* WAP to input a no. & a digit, and ~~count the no. of times~~ the digit occurs. (show Yes or No)

```
{ int num, udig, dig; flag=0; // udig is user digit
  cin >> num >> udig;
  while(num != 0){
    dig = num % 10;
    if (dig == udig){
      flag += 1;
      if (!= break;
    }
    num = num / 10;
    num = num/10;
  }
  if (flag == 1){ cout << " Yes"; }
    else { cout << " No"; }
}
```

~~* WAP to input a no. & reverse it.~~

of digits of an inputted no.

* WAP to find sum1, until it is a single digit no.

sum

```
while (num != 0)
{ int ld = num % 10;
  if (ld + num > 9)
    break;
  sum += ld;
  num /= 10;
}
```

Ans

* ALSO DO PRIME NOS. IN A RANGE.

IMP
* WAP to input a no. to check if it's a prime no. or not.

```
int num, i=2, flag=1;
cin>>num;
while(i<num){
    if(num%i==0){
        flag=0;
        break;
    }
    i++;
}
if(flag==1){cout<<"prime";}
else{cout<<"composite";}
```

* WAP to reverse an inputted no.

```
int n, rev=0, t;
cin>>n;
while(n!=0){
    t=n%10;
    rev=(rev*10)+t;
    n/=10;
}
cout<<rev;
```

121 → rev → 121

* WAP to check palindrome no.

... (same as rev.) → but also n=num; (at beginning)

```
if(rev==num){
    cout<<"palindrome";
}else{cout<<"no"}
```

Sum of factors of the no. (other than itself) should be equal to the no.
eg: 6 = 1×2×3
Sum = 1+2+3 = 6

* WAP to check if a no. is perfect or not.

Ans.
```
int num, i=1, sum=0, n;
cin>>num;
while(i<num){num ; ++i){
  if(num%i==0)        //check if it is a factor
    sum+=i;
  i++;
}
if(sum == num){
  cout<<"perfect";
}else{
  cout<<"not perfect";
}
```

* WAP for Fibonacci series. 0 1 1 2 3 5 8
(1+2) (3+5) (2+5)

```
{ int a=0, b=1, c, n, i=3;
  cin>> n;                    // no. of terms of series
  if(n<=2){
    cout<<"not possible... need more than 2";
  }else{                      // (as 0 & 1 are already there)
    while(i<=n){
      c = a+b;
      cout<< c;
      a = b;
      b = c;
      i++;
    }
  }
}
```

// change the order
0 1 1 → 1/a 1/b 2/c
1/a 2/b 3/c

*

* Find: (1) $x - \frac{x^2}{2!} + \frac{x^3}{3!} + x^4$

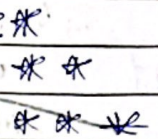
Ans. $\frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \dots + \frac{x^n}{n!}$

~~WAP to display all~~

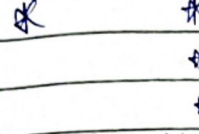
IMPORTANT LOOPS

~~for(~~

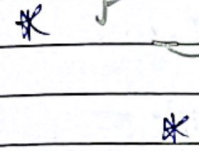
```
cin>>n;
for(int r=1, r<=n;
```



IMPORTANT LOOPS



```
cin>>n;
for(int r=1; r<=n; r++){          // no. of rows
    for(int c=1; c<=r; c++){      // no. of columns
        cout<<"*";
    }
    cout<<endl;
}
```



```
cin>>n;
for(int r=1; r<=n; r++){
    for(int s=1; s<=n-r; s++){
        cout<<" ";
    }
    for(int c=1; c<=r; c++){
        cout<<"* ";
    }
    cout<<endl;
}
```

total no. of rows − no. of current row (to calculate no spaces)

```
1
2 2
3 3 3
```

```
cin>>n;
for(int r=1; r<=n; ++r){
    for(int i=1; i<=r; i++){
        cout<<r;
    }
    cout<<endl;
}
```

row no.

```
1
1 2
1 2 2
```

```
cout<<i;
```

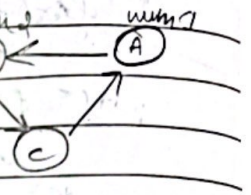
CONCEPT → check if all the nos. in between are prime nos. or not

* WAP to find all prime nos. between 2 inputted nos. (prime nos. in a range).

Ans.

```
int num1, num2; i=2, flag=0;
cin >> num1 >> num2;
if(num1 > num2) {
    int c = num2;
    num2 = num1;
    num1 = c;
    // to ensure that num1 < num2
}
else {
for(num1 += 1; num1 < num2; ++num1, i=2, flag=0) {
while(i < num1) {
if(num1 % i == 0) {
flag = 1;
break;
}
++i;
}
if(flag == 0) {
cout << num1 << " ";
}
}
}
```

NOT REQ. FOR EXAM



PRECAUTIONS

→ num1 should be atleast 2.

→ num1 < num2

Ans.

1 2 4
2 3
4 5 6

FLOYD'S TRIANGLE

```
int n, num = 1; // no. that gets displayed
cin >> n; // no. of rows user wants
for(int r=1; r<=n; ++r){
    for(int c=1; c<=r; ++c){
        cout << num;
        ++num; // to increase display no.
               // (from 1 to 2, ...)
    }
    cout << endl;
}
```

1 2 3 4 5
1 2 3 4 5
...

n times (by user)

```
int n, i, j;
cin >> n;
for(int r=1; r<=n; ++r){
    for(int c=1; c<=5; ++c){
        cout << c;
    }
}
```

A
A B
A B C
...

n times (by user)

```
int n,
cin >> n;
for(int r<=n
```

* WAP to find if a given no. is a prime or not.

```
{ int num, i=2, flag=0;
  cin>> num;
  if(num<i ?)
```

*
A
A B
A B C
...
] n-times (by user)

64+1 = 65 → A
64+2 = 66 → B

```
int n, ch;
 cin>>n;
for(int r=1; r<=n; ++r){
    for(int i=1; i<=r; ++i){
        cout<< char(64+i);
    }
    cout<<endl;
}
```

*
a
a b
a b c
...
] n times (by user)

n−r r
a
a b
a b c
n

```
int n, ch;
 cin>>n;
for(int r=1; r<=n; ++r){
    for(int s=1; s<=n−r; ++s){
        cout<<" ";
    }
    for(int i=1; i<=r; ++i){
        cout<< char(96+i);
    }
    cout<<endl;
}
```

96+1 = 97 → a
97+1 = 98 → b

```
*       * * *
        * *
        *
```

```
int n;
cin>>n;
for(r=n; r>=1; --r){
    for(int c=1; c<=r; c++){
        cout<<"*";
    }
    cout<<endl;
}
```

```
* 1 2 3
  4 5
  6
int num=1, cin>>n;
for(int r=n; r>=1; --r){
    for(int c=1; c<=r; ++c){
        cout<<num;
        ++num;
    }
    cout<<endl;
}
```

```
* 3 2 1
  2 1
  1
int n; cin>>n;
for(int r=n; r>=1; --r){
    for(int c=r; c>=1; --c){
        cout<<c;
    }
    cout<<endl;
}
```

$k! = 2^2 - 1$

~~WAP to find~~

```
int n; cin>>n;
for(int r=1, k=0; n, r<=n; ++r, k=0){
    for(int s=1; s<=n-r; ++s){
        cout<<" ";
    }
    while(k!=(2*i)-1){
        cout<<"*";
        ++k;
    }
    cout<<endl;
}
```

* Factorial

```
double fact=1;
for(int i=1; i<=n; ++i)
{   fact *= i;
}
```