20/1/23 (21CSC101T) Object Oriented Design & Programming

Page :
Date :

C - platform dependent
C++ - platform independent

History of C++

* It is the extension of C language. C++ is an object oriented language. C++ is a case sensitive language.

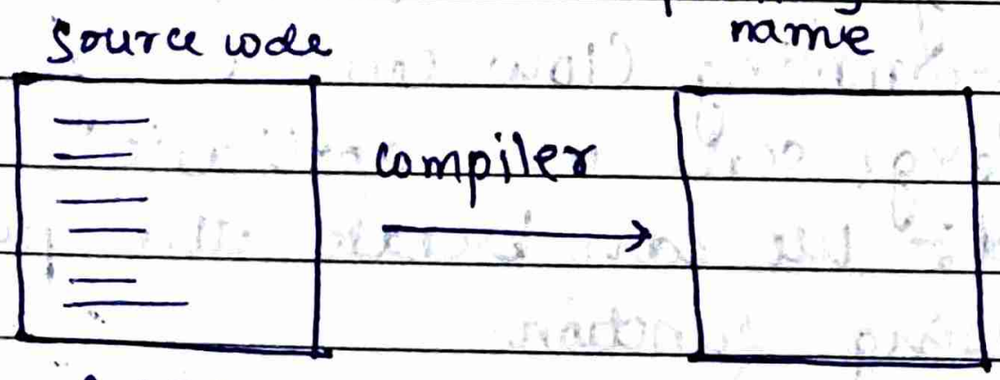
* C++ father is Bjarne.
1979, in Bell Laboratory developed.

* C++ first name is C with classes.
In 1983, the name is replaced by C++.

* C++ is compiled language.
C++ has also has exception handling.

A.C → C

A.CPP → C++
↓ primary name
↘ extention Secondary name



Page :
Date :

Template

pre processor directory

```
#include <iostream.h>      → input output stream → for I/O
#include <conio.h>         → header file → for (clrscr or getch)
void main()                → parenthesis → function
{
  clrscr();                → clear screen
  cout << "Hello";         → variable. → end of statement
  getch();                 → for holding screen o/p
}
```

Pre processor — return type — body open

→ print statement

Feature of C++

(1) Simple language
(2) Machine Independent or portable:
WORA (write once run anywhere)
(3) Mid-level language:- we can ~~use~~ do both system programming (low level & high level) And Build large scale user application.
(4) Structure Based:- We can break the program into parts using function.
It is easy to understand & modify.
(5) Rich library : C++ provide a lot of in-built function that makes the development fast
(6) Memory management : It support the feature of dynamic memory allocation in C++.
We can free the allocated memory at any time by calling the free function.

malloc
calloc
realloc
] new

Free () → delete

(7.) Speed : The compilation and execution is fast

(8.) Pointer : C++ provide the feature of pointer. We can directly interact with the memory by using a pointer.

(9.) Object Oriented
(10.) Extensible
] classes in C

Page :
Date :

OOPS Concept

(1.) Class
(2.) Object
(3.) Ncaptulation
(4) Abstraction
(5) Polymorphism
(6) Inheritance
(7) Dynamic Binding
(8) Message Passing

Class -

- Class is a blueprint, template or imaginary part.
- Class is a collection of object.
- Class is a passive entity. (objects work) not (class work)
- Every class ends with a semicolon.
- Class is a user defined data type which has data member and function.

```
class classname
{
   data member
   function
};
```

Object -

- Object is a real part.
- An entity that has state and behaviour is known as object.

- Object is an active entity.
- Object is instance of a class. (part)

Imp When a class is defined no memory is allocated but when object is created then memory is allocated.

```
class sports
{
  data member
  functions };
& sports volleyball;
  sports chess ;
  sports

};
```

```
Class A
{
  (1)
  (2)
  (3)
}
Class B
{
  (4)
  (5)
  (6)
}
class C
{
void main()
{ A, obj1,
```

Page :
Date :

Obj 2;
B : obj (2);
Obj26;
}

Naptulation

Wrapping of data or binding of data & collection of data and protect code. & code maintain.

Abstraction

show function, hide background task.

Polymorphism

poly + morphism , many (many) forms
atomatically change behaviour.

Eg → int a = 10, b = 20;
c = a + b;
c = 30
char c1[20] = 'Sanya';
char c2[20] = 'Vohra';
c1 + c2
=

Page :
Date :

Inheritance - (Code reusability) Advantage

Child class access all the member of parent class.

A (parent, super, base)

↓

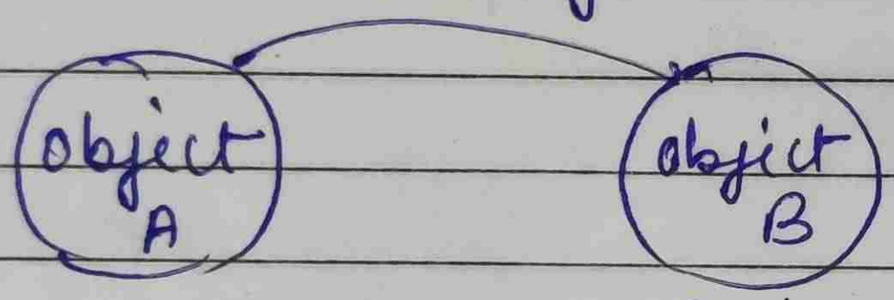
B (child, sub, derived)

```
Class A
{ (1)
  (2)
  (3)
}
                    inheritance operator
Class B (:) PublicA
{
 (4)
 (5)
 (6)
}
class C
{
void main()
{
  B obj1;
  obj1.(1);
  obj2.(4);
}
}
```

Page :
Date :

Dynamic Binding
The code to be executed in response to function call is decided at runtime.

Message Passing
Object communicate with one another by sending and recieving information to each other through message passing.



Sender obj Receiver obj.

Abstraction	Encapsulation (wrapping up data)
(1.) It is a process/method to gain information	(1) It is a process/method to contain information object
(2.) Problems are solved at interface level	(2.) Problems are solved at implementation level.
(3.) Method to hide unwanted info. Ex: to fetch, hide background.	(3.) Method to hide data in single entity along with method to protect info from outside
(4.) We implement abstr. using abstract class & interference	(4.) Encapsulation can be implemented by access modifier, ppp.
(5.) Implementation complexities are hidden using abstract classes.	(5.) Encap. data in hidden using getters, setters
(6.) Obj. that help to perform abstraction etc.	(6.) Obj. that result in encap need not be abstract

Page :
Date :

Input / Output

```
#include <iostream.h>
#include <conio.h>
void main()
{
clrscr();
cout << "Hello";
getch();
}
```

cin → console input

cout → console output

istream → for input stream

ostream → for output

cout → object, output operator,

int a; → declaration

→ variable name / identifier

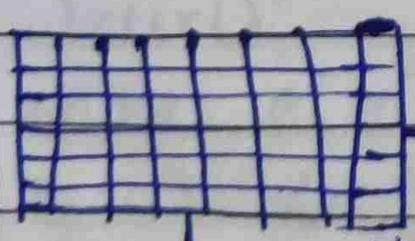
cin >> a;

→ input operator (extraction or get operator)

object

cout << "Hello";

output operator or put operator



Page :
Date :

Multiple Inputs in C++

```
void main()
{
int age;
char sec;
string name; / char name[10];
float per;
cin >> age;
cin >> sec;
cin >> name;
cin >> per;
cout << age;
cout << sec;
cout << name;
cout << per;
}
```

OR

```
void main()
{
int age;
char sec;
string name; / char name[10];
float per;
cin >> age >> sec >> name >> perc;
cout << age << sec << name << per;
}
```

for space

```
cout << age << " " << sec << " " << name <<
        " " << per;
```

for space b/w outputs

for next line

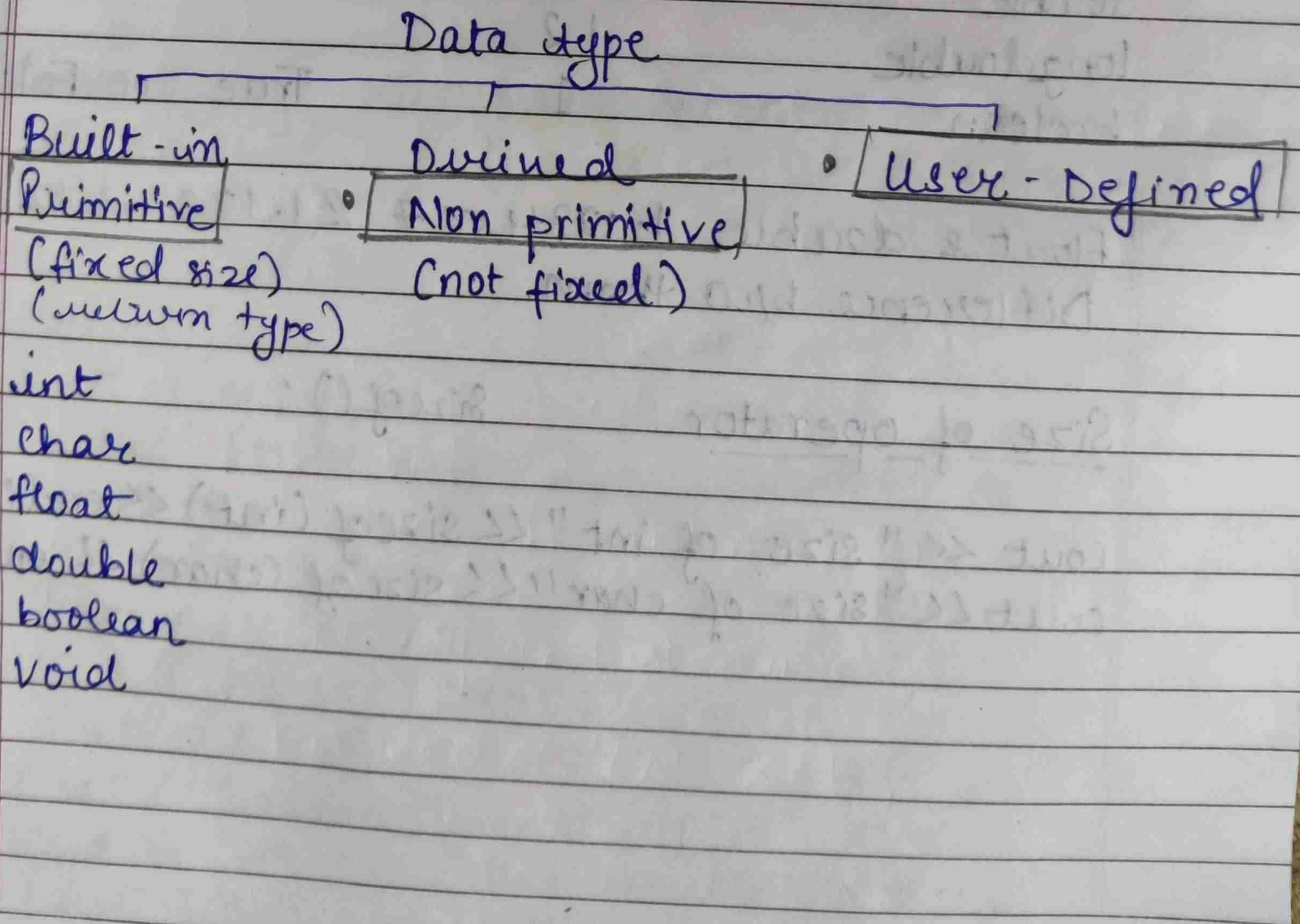
cout << age << "\n" << sec << "\n" << name << "\n" << per;

for end of line

cout << age << endl << sec << endl << name << endl << per;

→ endl & \n are used to break line. Both are same.

DATA TYPES IN C++



1 byte = 8 bits

Page :
Date :

```
int main()          char main()          void main()
{                   {                    {
  return 1;           return "A";          
}                   }                    }
```

→ Primitive Data Type

Data type	Size (byte)	Range / Eg
char	1	-128 to 127 or -2^7
short	2	-2^{15} to $2^{15}-1$
int	2/4	-2^{15} to $2^{15}-1$
long	4	-2^{31} to $2^{31}-1$
float	4	
double	8	
long double	12	
boolean	1	True or False

Float & double 1.1234567 & 1.123---12
Difference b/w them

Size of operator : sizeof();

```
cout << "size of int" << sizeof(int) << endl;
cout << "size of char" << sizeof(char) << endl;
```

(1.)

(2.)

Page :
Date :

Comments

Comment can be used to explain C++ code.
Makes it more readable.
It can also be used to prevent execution when testing alternate code

2 Types of comments

1) Singel line
- Singel line comments start with 2 forward slashes (//).

- Any text between // & and the end of the line is ignored by compiler.

```
// addition of two number
c = a+b;    // c is the storage variable
```

2) Multi line
- Multi line start with /* and end with */ any text between multiline is ignored by the compiler.

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

Page :
Date :

⇒ Identifier

All c++ variable must be identified with unique name.
Identifier can be short name (a&b) and more discriptive name (age & result)

Rules of identifier -

- Name can contain letter, digit, underscore.
- Name must begin with letter or underscore.
- Not contain white space or special character.
- Cannot be Keywords.

⇒ Constant

When you do not want other or yourself to change existing variable value. Then use const keyword.
Its value is fixed.

```
int a=5;
a++;  → 6
a=6;  → 6
```

```
const int pi=3.14;
pi++;   — error
pi=8.56;  — error
```

cannot be modified or updated.

⇒ Keyword in C++ (Same as C)

char int float short long
double auto void if else
for while do default break
continue const switch case return
go to static register extern enum
struct union signed unsigned typedef
volatile

⇒ New Keywords in c++;

try catch finally throw,
default public protected private,
bool true false, class, virtual
new delete using //dynamic
explicit inline mutable friend
template operator

Page :
Date :

Pointer

→ The pointer in C++ language is a variable.

→ It is also known as locator or indicator that points to the address of value.

→ It makes us able to access any memory location in the computer memory.

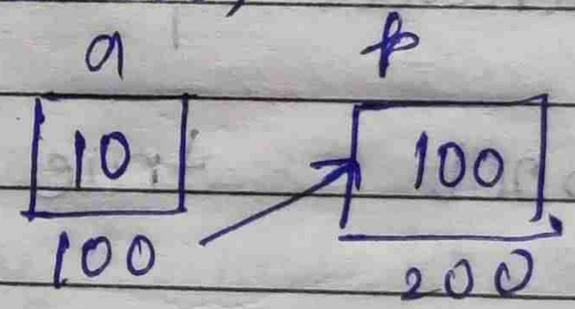
Syntax -

datatype *variable name;

Eg→ int *p;

(determine address of a variable)

Symbol - & → address operator
* → indirection operator
(access the value of an address)




```
void main()
{
int a=10;
int *p=&a;
cout << "address of a " << &a;
" ---------- " << p;
cout << "value of a " << a; → referencing
" ---------- " << *p; → dereferencing
}
```

riable

tor

mory.

f a
mabc()

an

g
ring

Page :
Date :

Swapping 2 numbers

```
void main()
{
int a=10, b=20;
cout << a << " " << b;
a = a+b;
b = a-b;
a = a-b;
cout << a << " " << b;
}
```

Pointer through

```
void main()
{
int *p1=&a, *p2=&b;
cout << *p1 << " " << *p2;
*p1 = *p1 + *p2;
*p2 = *p1 - *p2;
*p1 = *p1 - *p2;
cout << *p1 << " " << *p2;
}
```

Page :
Date :

Decision Making

⇒ **If-Else**

```
int a=10;
if (a%2==0)
{
  cout << "Even" << endl;
}
else
{
  cout << "Odd" << endl;
}
```

or

User define

```
void main()
{ int a;
cout << "Enter no" << endl
 cin >> a;
```

⇒ **Nested If-**

(Password & login)

```
void main()
{
if ( a == "Hiiw")
  {
    if (b == "1234")
      {
        cout << " login" << endl;
      }
else
    {
      cout << " Incorrect password" << endl;
    }
}
else
  {
    cout << "Incorrect mail" << endl;
  }
}
```

⇒

earlier in password & mail id

```
if (a == "Him" && b == "1234")
  {
    cout << "login" << endl;
  }
else
  {
    cout << " login failed " << endl;
  }
```

⇒ **If - Elseif**

```
int a = 4, b = 5, c = 6;
if (a == 4 && b == 5 && c == 7)
  {
    cout << "login" << endl;
  }
else if (b == 5 && c == 6)
  {
    cout << "login failed" << endl;
  }
else {
  {
    cout << "bye";
  }
```

⇒ **Switch case**

```
int a = 2;
Switch (a)
{
case 1:
cout << "Hi" << endl;
break;
case 2:
cout << "Bye" << endl;
break;
default:
cout << "Sorry";
}
```

⇒ or for multiple switch case

```
case "ABC":
case "X42":
cout << "Hello" << endl;
break;
case "DEF":
case "NHG":
cout << "Bye";
break;
default:
cout << "OK";
}
```

or User Defined

```
int a;
cout << "Enter" << endl;
cin >> a;
```

Char →

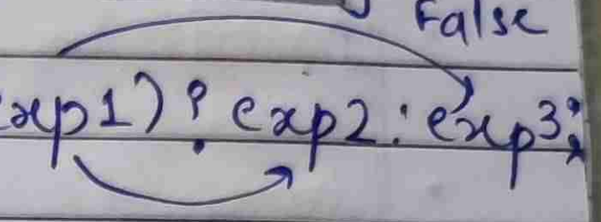
String →

```
String a;
cout << "Enter";
cin >> a;
switch (a)
{
case "ABC";
cout << "Hi" }
break;
case "X42";
cout << "Bye";
break;
default;
cout << "Sorry";
}
```

Page :
Date :

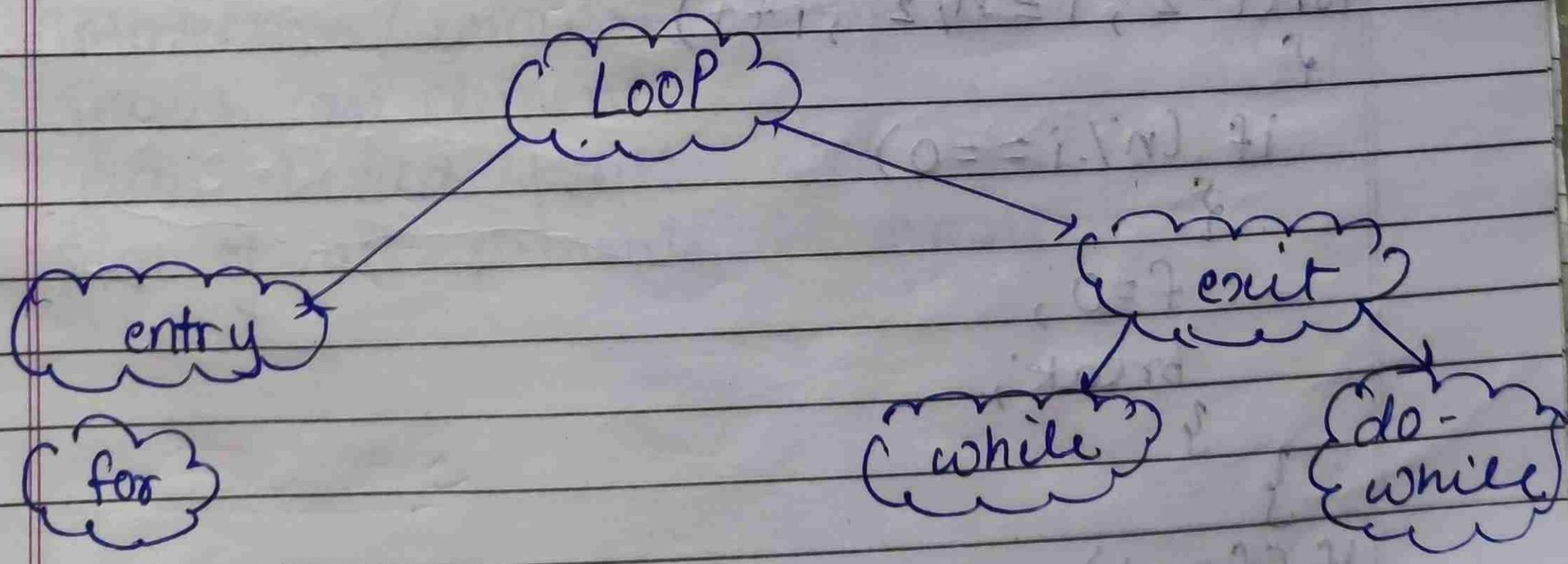
Operators

- Unary
  ++ --
  increment-decrement
- Binary
  (+, -, *, /, %.)
  - Arithmatic
  - Assignment
    (=, +=, -=, *=, /=)
  - Relational
    (<, >, <=, >=, ==, !=)
  - Logical
    (&&, ||, !)
  - Bitwise
    (&, |, ^, ~)
- Ternary



H.W

(Q.) = Prime no.
Fibonacci
Armstrong
Palindrome
Factorial
Table (no.)



```
int ii;
for(i=1; i<=5; i++)
{
 cout << i;
}
```

```
uint i=1;
while(i<5)
{
  cout<<i;
  i++;
}
```

```
int i=1;
do
{
  cout<<i;
  i++;
}
while(i<=5);
```

(1.) Factorial (5x4x3x2x1)

```
int i, n=5, f=1;
for(i=1; i<=n; i++)
 {
   f=f*i;
 }
cout << f;
```

(2.) Prime

```
int i,n,f=1;
fin>>n;
for(i=2; i<=n/2; i++)
  {
    if (n%.i==0)
    {
      f=0;
      break;
    }
  }
if (f==1)
```

```
{
 cout << "Prime";
else
 {
  cout << "Not Prime";
 }
```

logic eg - 123

$10\overline{)123}\,12$ $\quad$ $10\overline{)12}\,1$

120 $\quad$ 10

(3) $\quad$ (2)

Armstrong → 1*1*1, 2*2*2, 3*3*3.

Palindrome → arrange

(Q.) Array Questions

- ✓ Sort an array
- ✓ minimum array find
- ✓ maximum array find
- intersection / common element
- search an element
- even & odd position add
- Sum of all elements in array

multiply
↳ initialise
↳ 1

Addition
↳ initialise
↳ 0

Page :
Date :

Array

- collection of similar data type.
- Based on indexing, first 0 last (n-1).
- type of a data structure.
- Initialisation → int a[5];

```
for(i=0; i<5; i++)
{ cin>> a[i];
}
```

(1.) Maximum (2.) minimum

```
void main()
{
int a[5];
for(i=0; i<5; i++)
  {
    cin >> a[i];
  }
int c = a[0];
for(i=0; i<=5 ; i++)
  {
   if(a[i]>=c)
     {
       c = a[i];
     }
  }
cout<<c;
}
```

or (a[i]<=c)

(3.) sort

```
void
{
int
for
{
}
for
{
fo
}
}
```

(4.) S

Page :
Date :

(3.) sorting

0	1	2	3	4
5	7	1	2	3

```
void main()
{
int a[5], i, j, t;
for (i=0 ; i<5 ; i++)
 {
  cin>> a[i];
 }
for (i=0; i<4 ; i++)
 {
 for(j=1+i ; j<5 ; i++)
   {
     if (a[i] >=a[j])
       {
          t = a[i];
          a[i] = a[j];
          a[j] = t;
       }
   }
}
for (i=0; i<5 ; i++)
{
  cout << a[i];
}
}
```

if a[i] > a[j]
t = a[i]
a[i] = a[j]
a[j] = t

print

(4.) Searching element

```
void main()
{
int a[i], i, f=1, n;
for (i=0; i<5; i++)
```

Page :
Date :

```
{
cin >> a[i];
}
}
int n=5;
for(i=0; i<5; i++)
{
  if (a[i] == n)
  {
    j=0;
    break;
  }
}
if (j==0)
{
  cout << "Element found";
}
else
{
  cout << "Not found";
}
}
```

(3.) Fibona

```
int n
{
int
cout
for (i
{
  c
  co
  a
  b
}
```

(4.) Arm

```
int
{
in
c
C=
wh
{
a
b
n
if
{
}
el
```

(3.) Fibonacci Series (0, 1, 2 3 5 8 13 - -)

```
int main()
{
int a=0, b=1, i, c, n;
cin >> n;
cout << a << b;
for (i=2; i<=n; i++)
{
  c=a+b;
  cout << c;
  a=b;
  b=c;
}
```

(4.) Armstrong no. (eg 153)

```
int main()
{
int a, b=0, c, n;
cin >> n;
c=n;
while (n>0)
{
 a = n%10;
 b = b+ (a*a*a);
 n = n/10;
if (c==b)
{
 cout << "Armstrong";
}
else
 { cout << "not";
 }
```

(5.) Palindrome (2002)

```
int main()
{
int a, b=0, c, n;
cin >> n;
while (n>0)
{
 a = a%10;
 b = (b*10)+a;
 n = n/10;
}
if (c==b)
{
  cout<<"Palindrome";
}
else
 {
  cout<<"Not Palindrome";
 }
```

(6.) Multiplication table

```
int main()
{
int n,i;
cin >> n;
for (i=1; i<=10; i++)
{
   cout<<n<<"*"<<i<<"="<<n*i<<endl;
}
return 0;
}
```

Array

(5) Search

```
int m
{
 int i
cout<
for (i
{
   cin
}
cout
for(i=
{
  if
  {
  }
}
if(c=
{
 cou
}
else
{
cou
}
retu
}
```

Page :
Date :

Array

(5) Search an element in array

```
int main()
{
int i,n, c=0; a[5];
cout<<"Enter an array:";
for(i=0; i<5; i++)
{
  cin>> a[i];
}
cout<<"Enter element";<< endl;
for(i=0; i<5; i++)
{
  if(a[i] == n)
    {
      c=1;
    }
}
if(c==1)
{
 cout<<"Element Found";
}
else
{
 cout<<"Element not found";
}
return 0;
}
```

Page No. :
Date :

(6.) Even - Odd

```
int main(){
int a[5], i, d=0, c=0;
cout<<"Enter array:";
for(i=0;i<5;i++)
{
  cin>>a[i];
}
for(i=0;i<5;i++)
{
 if(a[i]%2==0)
    {
      c=c+a[i];
    }
 else
    {
      d=d+a[i];
    }
}
cout<<"even";
cout<<c;
cout<<"odd";
cout<<d;
return 0;
}
```

(7.) Sum of all elements in array

```
int main()
{
```

Page :
Date :

```
int a[5], i, d=0, c=0;
cout<<" Enter array :";
for(i=0; i<5; i++)
{
   cin>>a[i];
}
for(i=0; i<5; i++)
{
    c=c+a[i];
}
cout<<c;
return 0;
}
```

Page :
Date :

OOPS

Class & object

```
class A                          O/P
{                                18
public:
int age;
};
int main()
{
A obl;
obl age =18;
cout<<age;
}
```

How to create a class?

(1) Class Keyword is used to create a class.

(2) Public Keyword is an access specifier or modifier which specify that member (everything inside class, variable & function) of the class are accessible from outside the class.

(3) End of class defination with EOS or ;

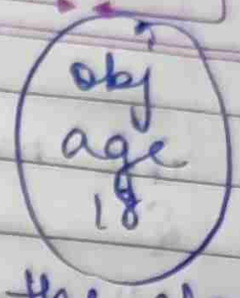
(4) Inside the class when variable are declared they are called attributes

- Keyword
- access specifier
public, private
specify member of class
can accessible from
outside of class
- inside a class variable
are called attribute

Obj
To
and
3+
cl
{
Eg
A obl;
a.obl;

Page :
Date :

Object

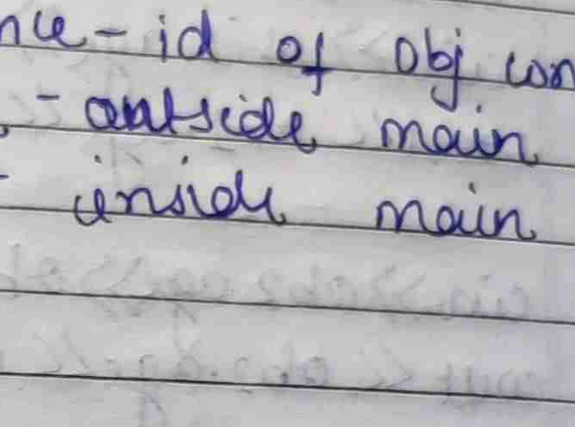


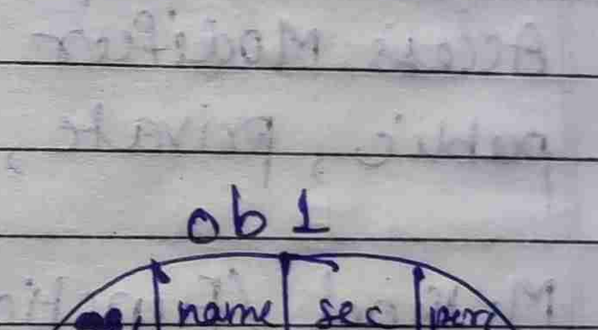
To create an object of class, specify the class name and reference value.
3 types of variable { reference - id of obj contain
global - outside main
local - inside main

```
class A
{
 public:
 int age;
};
int main()
{
A obj;
cin >> obj.age;
cout << obj.age;
}
```

Program →

Eg:
```
class A
{
public:
int Age;
String name;
char sec;
float perc;
};
```





```
int main()
{
 A ob1;
cout << "Enter data:" << endl;
cin >> ob1.age >> ob1.name >> ob1.sec >> ob1.perc;
cout << ob1.age << ob1.name << ob1.sec << ob1.perc;

cin >> ob2.age >> ob2.name >> ob2.sec >> ob2.perc;
cout << ob2.age << ob2.name << ob2.sec << ob2.perc;
return 0;
}
```

Access Modifier

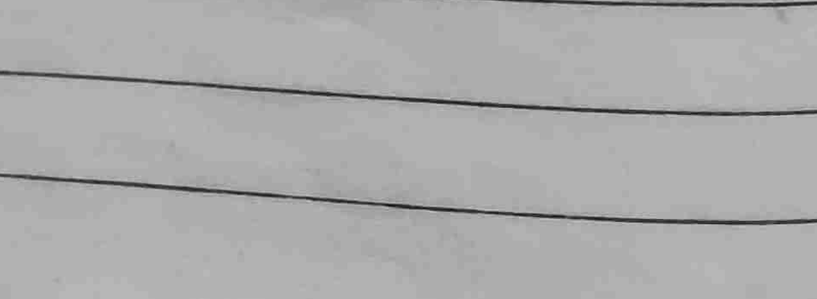
public, private, protected.

Method / function in class

```
class A
{
 public: show
 void main()
 {
  cout << "Hello";
 }
};
int main()
{
 A ob1;
 ob1.show();
}
```

O/P

Hello



Page :
Date :

```
class A
{
 public:
  void show()
  {
    cout << "Hello";
  }
};
class B
{
 public;
  void play()
  {
    cout << " Bye";
  }
};
int main()
{
 A ob1;
 ob1.show();
 Bob2;
 ob2.play();
 return 0;
}
```

7/2/23

Eg→ class A
{
public:
int show()
{
cout << "Hello";
return 0;
}
int play()
{
cout << "Bye";
return 0;
}
class B
{
public:
int red()
{
cout << "go";
return 0;
}
int black()
{
cout << "exit";
return 0;
}
}

O/P Hello
Bye
go
exit

Eg→ cl
{
public

int main()
{
A ob1;
ob1.show();
ob1.play();
B ob2;
ob2.red();
ob2.black;
return 0;
}

Eg

Page :
Date :

Eg→ class A
{
private: { attributes / variables }
int age;
public:
int show() { func }
{
cout << "Enter age:";
cin >> age;
cout << age;
return 4;
}
}
int main()
{
A ob1;
//ob1.show();
cout << "Result:" << ob1.show();
return 0;
}

O/P 4

Pre-define :

Multiple function.

Eg- class A
{
private:
int age;

y ob1.show();

O/P
Enter age:
18
18

Page :
Date :

Eg → class A
{
private:
int age;
public:
int show()
{
age = 18; → global variable
cout << age;
return 0;
}
int play()
{
local variable ← int age = 20;
cout << age;
return 0;
}
int rd(int age) → local variable
                  → formal parameter
{
cout << age;
return 0;
}
};

int main()
{
A.ob1;
ob1.show();
ob1.play();
ob1.rd(89);

obj. show();
obj. fun();
obj. red(22);
}

eg →

class A
{
private:
int age;
string name;
char sec;
float per;

public:
int show()
{
cin >> age >> name >> sec >> per;
cout << age << name << sec << per;
return 0;
};
int red()
{
cin >> age >> name >> sec >> per;
cout << age << name << sec << per;
return 0;
}
int main()
{ A obj;
obj. show();
obj. red();
return 0;
}

18
20
29
18
20
22

class is heterogenous data type

obj 1.
{ 2 baaki ke liye
2 func.

Page :
Date :

8/2/23

Or

void Read (int age, string name, float per,
~~float~~. only cout not cin.

int main()
{ Abl;
obl.read (18, "Hina", 20.5, 'N');
} return 0;

function print karte hai to uski return type print hoti hai

Page :
Date :

8/2/23

Function (swap) for calling. func
encaptulation - collection of data input
types (wrapping) logic
output

```
class A
{
private
int a,b,c ; public:
void input()
{
  cout << "Enter a & b values:";
  cin >> a >> b;
  cout << a << b;
}
void logic()
{
  c = a;
  a = b;
  b = c;
}
void output()
{
  cout << a << " " << b;
}
};
int main()
{
A obj;
obj.input();
obj.logic();
obj.output();
return 0;
}
```

Func for swap

```
class A
{ void input()
  void logic()
  void output()
};
int main()
{ A obj;
  obj.input();
  obj.logic();
  obj.output();
  return 0;
}
```

Accessing func outside class,

Function inside & outside the class access it

```
class A
{
public:
void show()
{
cout<<"Hello"<<endl;
}
void play();
};
void A :: play()
{
cout<<"Bye";
}
int main()
{
A obj;
obj.show();
obj.play();
return 0;
}
```

Declared inside

Defined outside

Prog,

Encapsulation : Collection of data type member.

wrapping up of data/ binding data

set/get

```
class A
{
private:
int age;
```

→ private variable by encapsulation & abstraction

```
void setAge (int a)
 {
   age = a;
 }
int getAge ()
 {
   return age;
 }
};
int main()
{
  A ob1;
  ob1.setAge(20);
  cout << ob1.getAge();
  return 0;
}
```

Prog,

```
class A
{
 private:
 int age;
 string name;
 public:
 void setAge (int A)
 {
   age = a;
 }
 int getAge()
 {
   return age;
 }
```

```
void setname (string n)
{
  name=n';
}
string getname ()
{
  return name';
}};
int main()
{
A obl';
obl.setAge(20)';
cout<< obl.getAge();
ob.setname("Him");
cout.<< obl.getNum();
return 0';
}
```

By default all the member in class are private

Access Specifier :

Public : Member are accessable from outside the class.

Private : Member cannot be access from outside the class.

Protected : Member cannot be access from outside the class, they can be accessed in inherit class.

Page :
Date :

```
class A
{
public:
data member
function

private:
data member
function

protected:
data member
function
};
```

object Exception handling | Pointers Exception handling

Page :
Date :

Encapsulation:

Encapsulation is to make sure that sensitive data is hidden from user. To achieve this you must declare class variable private, collection of data member and function.

Abstraction: Show function hide background task.

c++ classes provide great level of data abstraction.

They provide sufficient public method to the outside world.

To play with the functionality of the object. and to (manipulate object data).

Data abstraction refers to provide only essential information to the outside world or hidden their background details.

Access labels -

* Same for public, private.

```
class Bank
{
private:
int balance, atmpin;
public:
int accno;
string bname, ifsc;
void userdata()
{
  balance = 50000;
  atmpin = 1234;
  accno = 123456789;
  bname = "CUB";
  ifsc = "CIUB00102";
  cout << balance << atmpin << Accno << balance << ifsc;
};
int main()
{
  Bank b1;
  b1.userdata();
  cout << b1. balance << b1.atmpin << b1.accno
       << b1.ifsc;
  return 0;
}
```

error ← [

bus

private

so here

abstraction

is used.

Page :
Date :

Type conversion
convert one datatype data to another datatype data.
There are 2 types of type conversion.

Implicit type conversion (automatic type conversion)

Explicit type conversion (type casting)

Done by the compiler on its own. Also known as automatic type.

① ⇒ Implicit :

Eg → bool → char → short int → long int → float → double

```
int a = 10;
char b = 'a';
a = a + b;
cout << a
```

o/p
10 + 97

② ⇒ Explicit : Also known as type cast. When the user manually change data from one type to another. It is known as Explicit conversion.

There are 3 major ways :-

1)
2)
3)

Page :
Date :

1) C style type casting (cast notation)
2) Function notation.
3) Type conversion operation

static	Non static
retain value	Do not retain value

long double ⇒ double ⇒ float ⇒ long int ⇐
bool ⇐ char ⇐ short int ⇐ int

```
double a = 5.97;
int b = (int) a + 1';
cout << b';
```

Higher data type

long double
↓
double
↓
float
↓
long int
↓
int
↓
short int
↓
char
↓
bool

data loss

no data loss

Page :
Date :

Type conversion operators : 4 types of type conversion operators

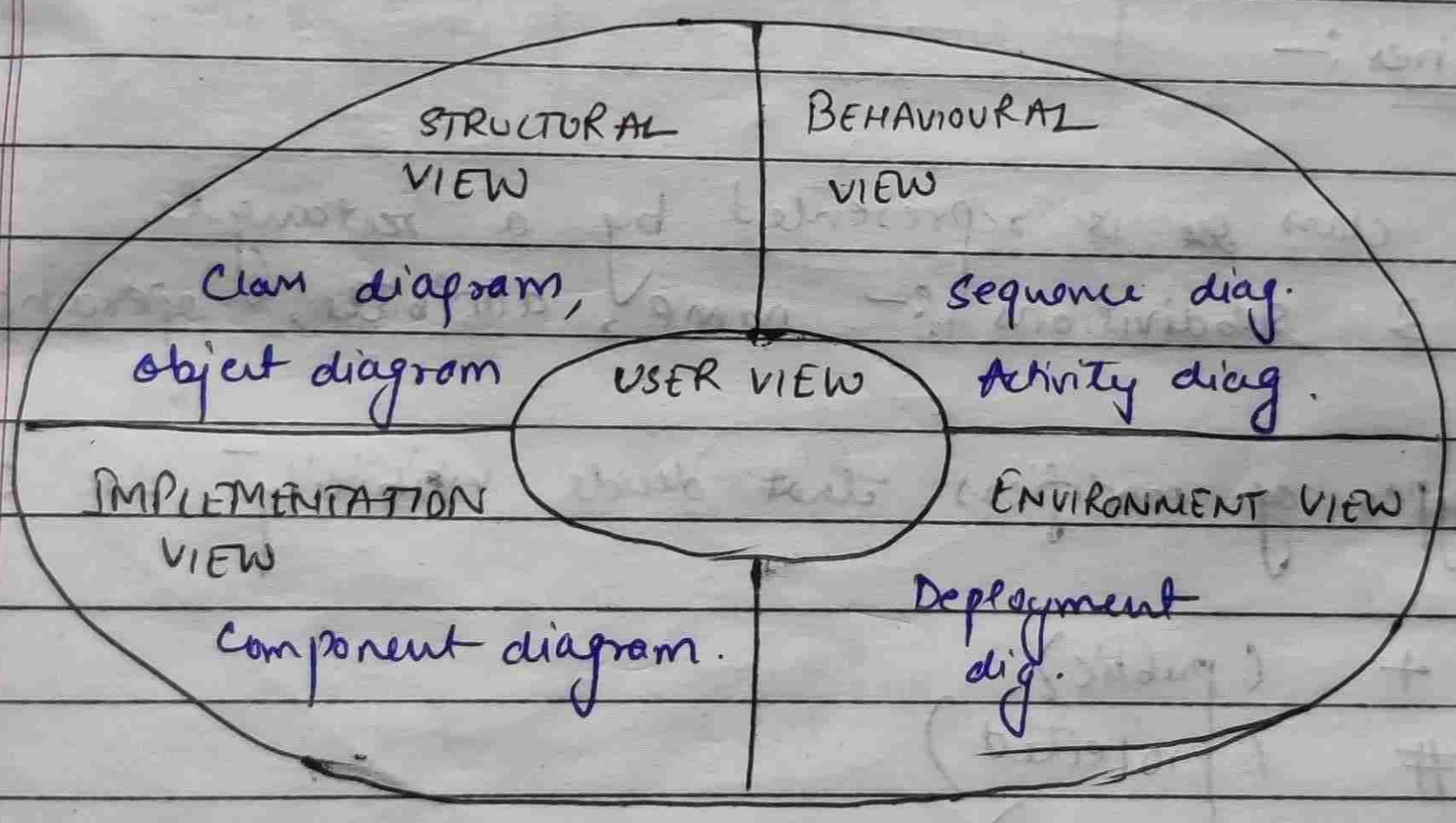
1) static_cast
2) dynamic_cast
3) const_cast
4) reinterpret_cast

Page :
Date :

#1 What is UML? Mention perspectives of using UML diagram. Also classify the UML diagrams.

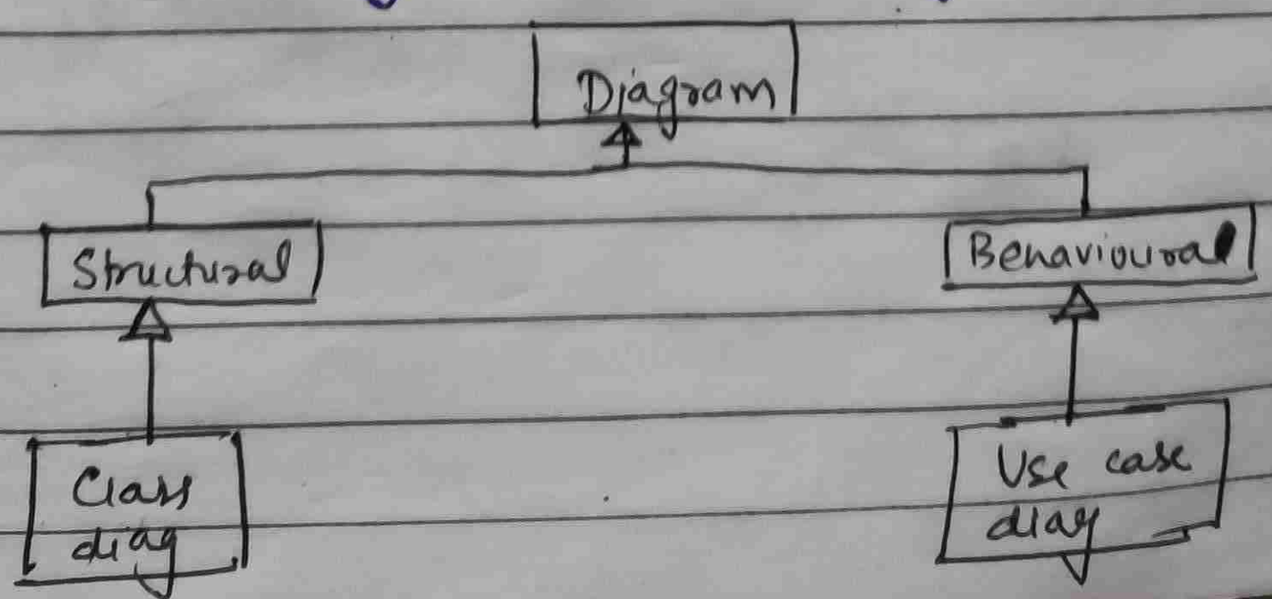
Unified Modeling language is a general purpose modelling language.
Its aim is to define a standard way of visualizing the way a system has been designed.
It is not a programming language

Perspectives that can be captured are :-



Diagrams in UML are classified as :-

- Structural Dig : Captures static aspects of a system
- Behavioural Diag : Captures dynamic aspect



Page :
Date :

#2 Describe Class diagrams with example. List down its properties and relationships in class diagrams.

Class diagrams are main building blocks of every object-oriented method. It shows relationship, collaboration & associations.

Some softwares available are :- Lucid Chart, Edraw MaX

Properties :-

Each class is represented by a rectangle with 3 subdivisions :- name, attribute, operations

3 Types of modifiers that decide visibility are :-

1. + (public)
2. # (protected)
3. - (private)

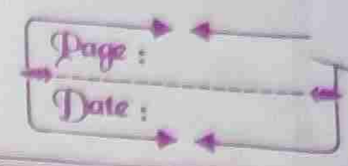
Example :-

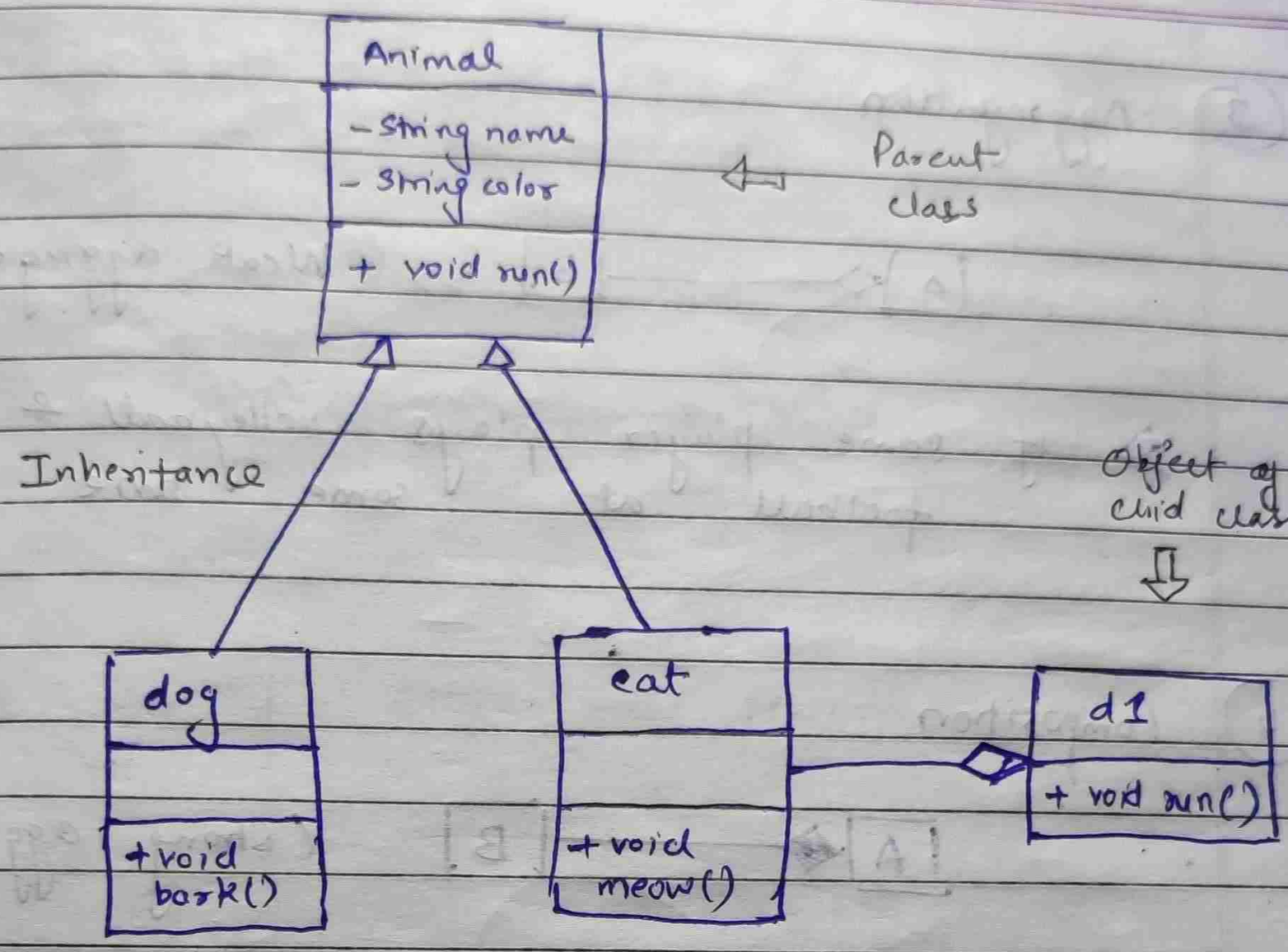
Next →

Child class

(1)

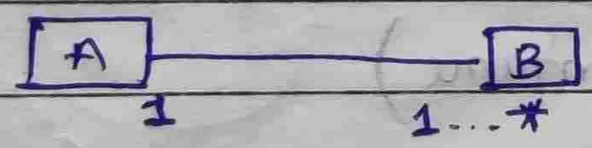
(2)





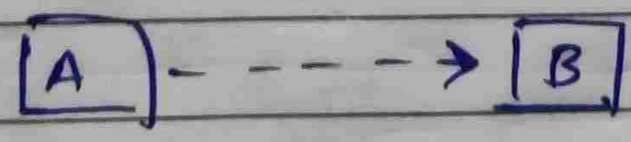
Relationship :-

(1) Association ex coach to student



type : one to one (1)
one to many (1.... *)
exact no. (2, 4, 5)
zero to star (0..* *)

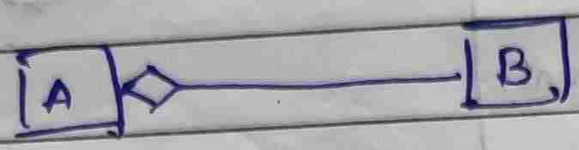
(2) Dependency ex Bike on petrol



Page :
Date :

3

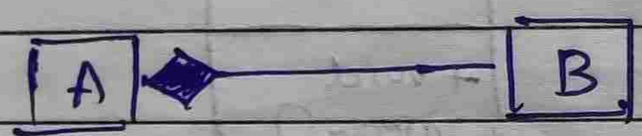
(3) Aggregation



(Weak aggregation)

ex if same player plays volleyball & football at same time

(4) Composition



(Strong aggregation)

ex Mobile & battery are strong aggreg.
Mobile & electricity are weak aggreg.

(5) Generalization (inheritance or interface)

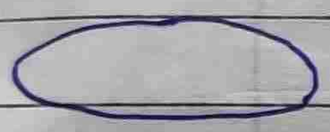
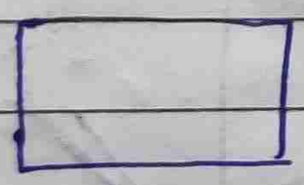
(inheritance)

Page :
Date :

#3 What is UML Use Case Diagram? Mention its components and relationships.

A use case diagram is used to represent the dynamic behaviour of a system.
It depicts high-level functionality.

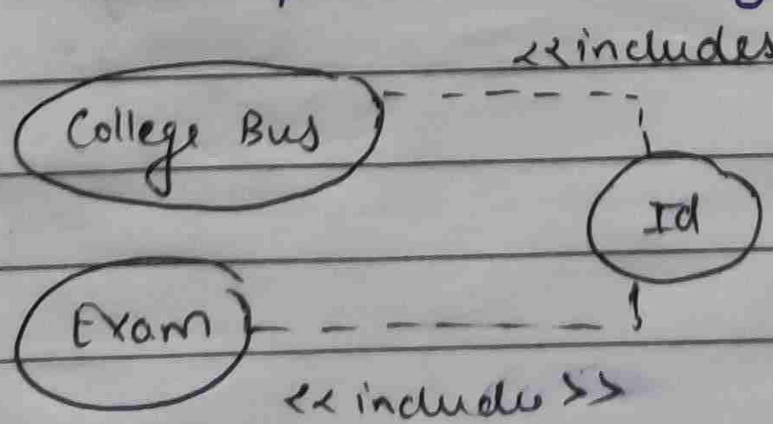
Components :—

(1)		actor / real person
(2)		use case
(3)		Communication line
(4)		System boundary
(5)		Stereo type

Relationships :—

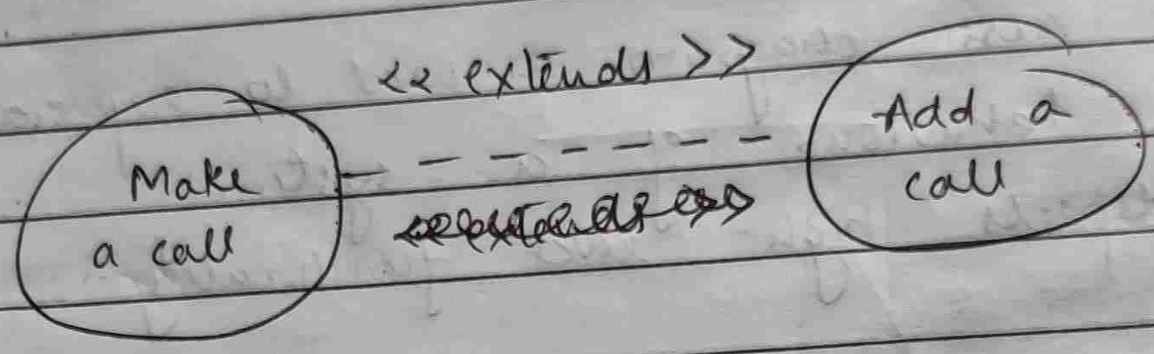
(1) <<includes>> - It is an implicit function. They are required to carry out a function.

ex



(2) <<extends>> - It is explicit function. Without this addition also the functionality works.

ex



Online Shopping Example :-