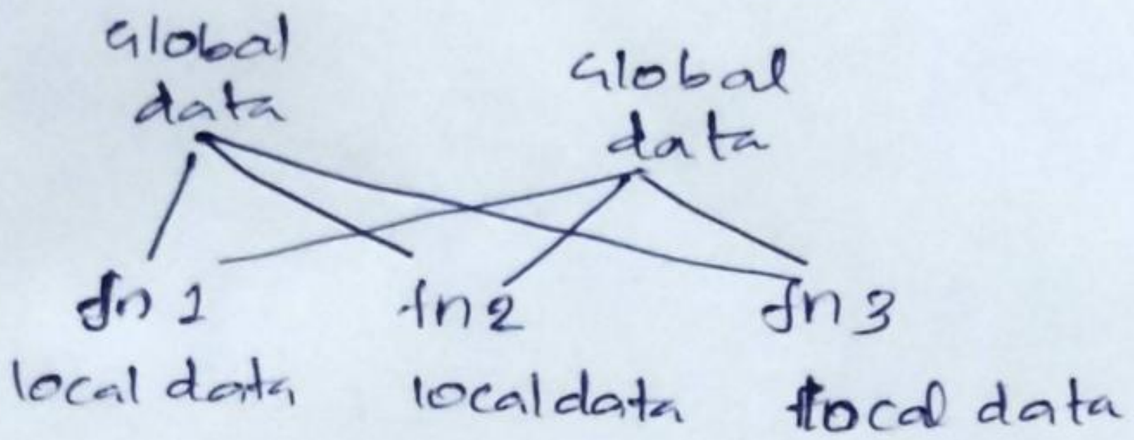
5/4/22

C

Procedural programming lang

Top-down approach



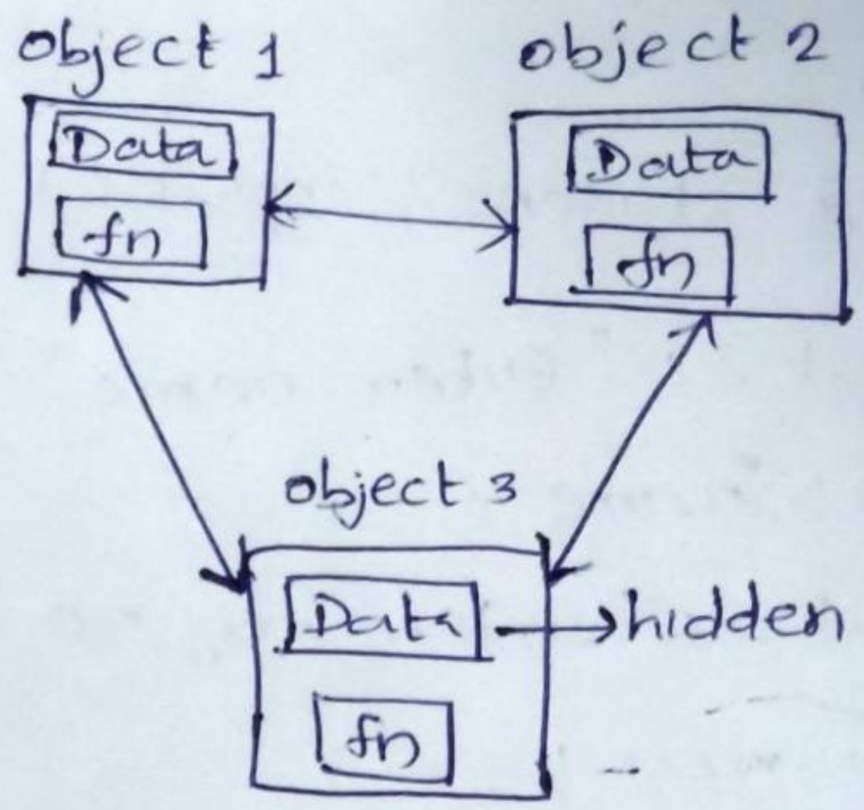
C++

Object-Oriented programming lang

Bottom-up approach

Program divided into objects

Data is hidden, can't access the external functions



Write a C++ program using class to print the details of student.

```
#include<iostream>
using namespace std;
{
  int main ()
  {
      int m1, m2, m3;
      char name;
```

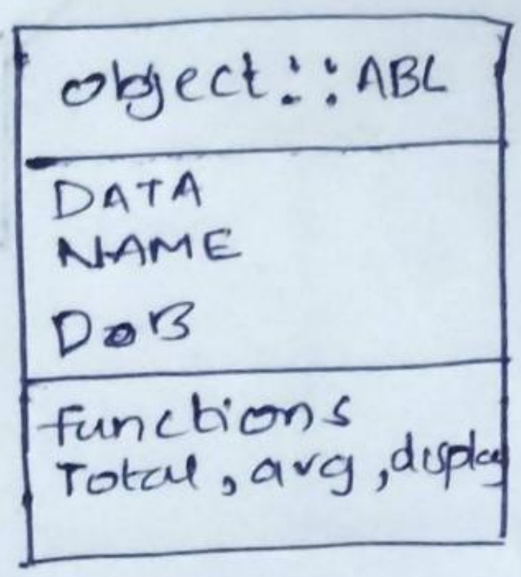
```
#include<iostream>
using namespace std;
class student {
  private :
  char name[20], reg[10], branch[10];
  int sem;

  public:
    void input();
    void display();
};
void student :: input() {
cout << "Enter name";
cin >> name;
cout << "enter reg no";
cin>>reg;
cout<<"Enter branch name"
cin >> branch;
cout<<" Enter sem";
cin>> sem;
}
void student :: display () {
cout << "\n Name" << name;
cout << "\nReg no" << reg;
cout << "\nBranch" << branch;
cout << "\nsem" << sem;
}
```

```
void main ()
{
   student s ;
   s. input( ) ;
   s. display ;
}
```

Basic Concepts of OOPs

1. Objects → Basic run time entities (may be a person, place, bank ac)
2. Classes ↳ user defined data (vectors, time, lists) - real world objec
3. Data abstraction and encapsulation ↳ takes space in mem
4. Inheritance
5. Polymorphism
6. Dynamic binding
7. Message passing



2 class → object contain, Data and a code to manipulate data

The entire ~~store~~ set of data and code of an object can be made a user defined data type with the help of a class, collection of similar objects.

3. The wrapping of data and function in to a single unit (called class) is known as encapsulation.

Normally data is not accesible to the outside world. The functions which are wrapped in the class can access.

Function provide the interface b/w object data and program

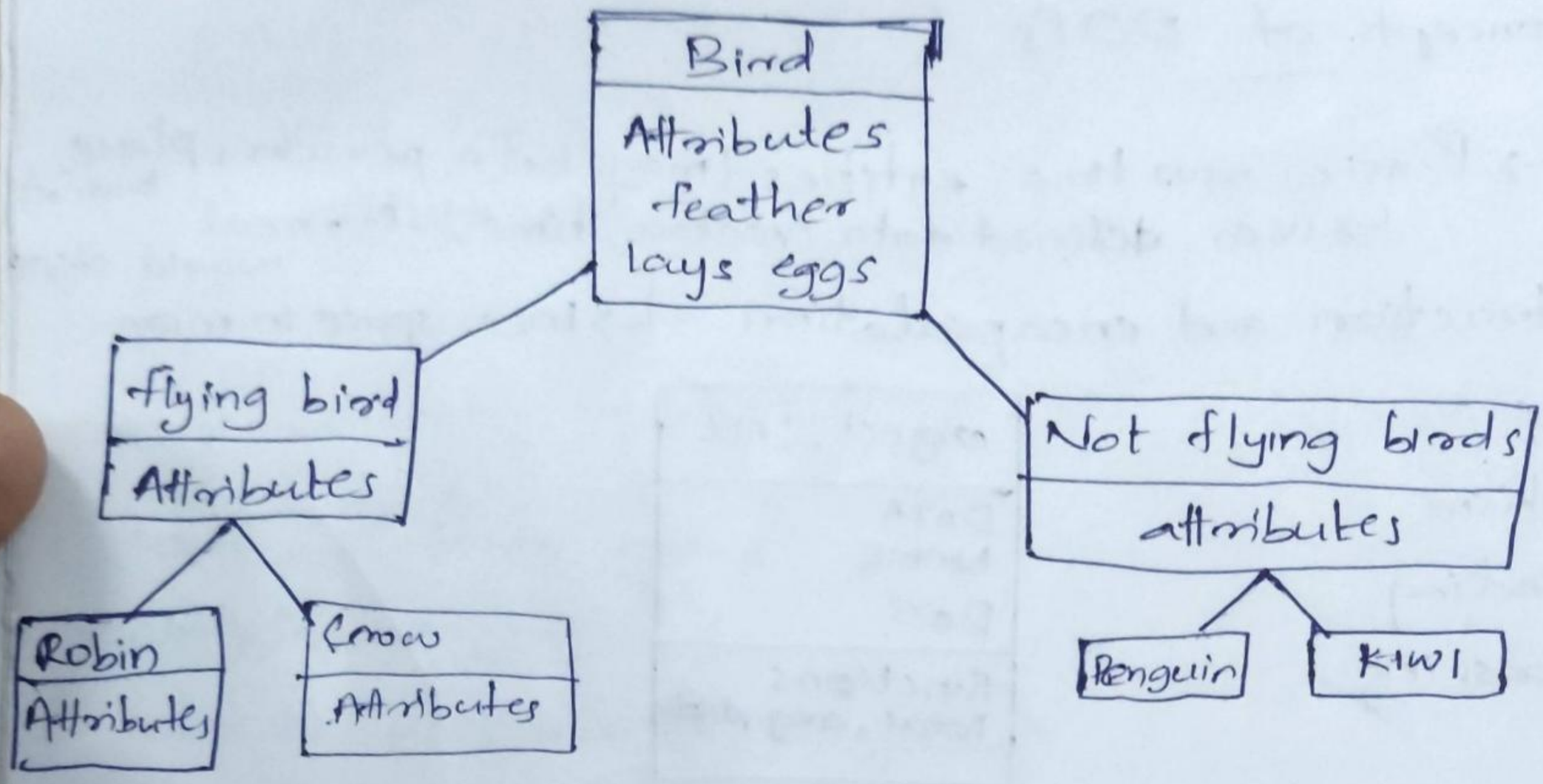
Insulation of data from the direct access by a program is

called data hiding or informating hiding.

4. Inheritance

Process by which objects of one class acquire the properties of objects of ~~authors~~ another class

It supports hierarchial classification.

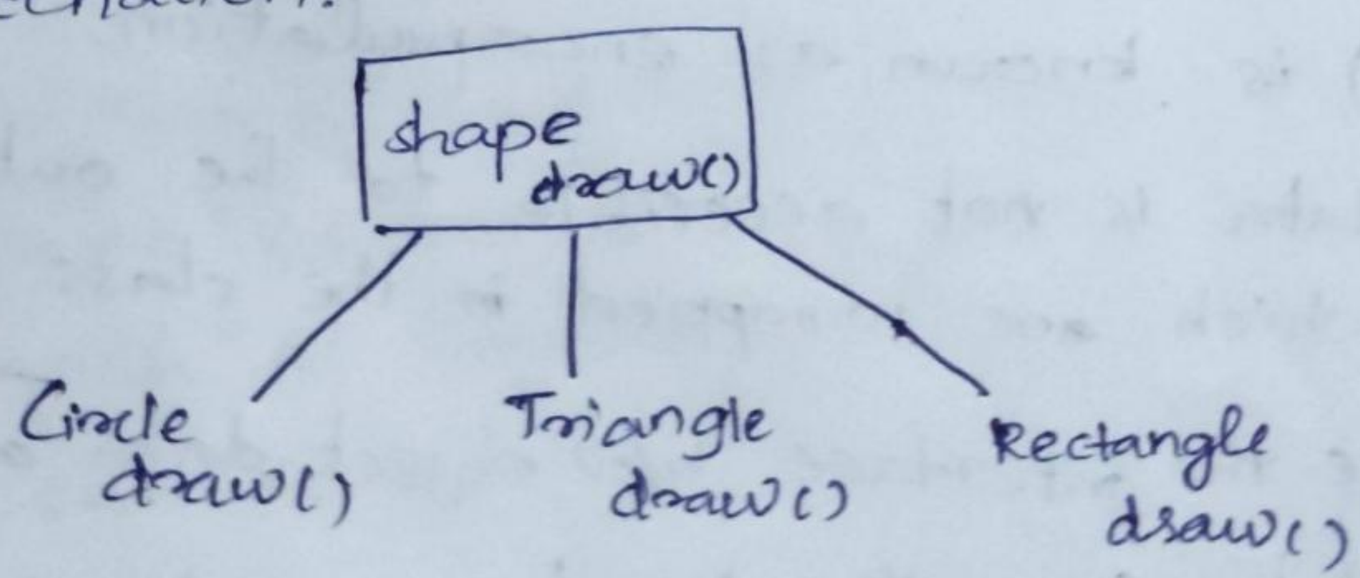


11/4/22 Polymorphism

An operation may exhibit different behaviour in different instance is called polymorphism.

Behaviour depends up on the data used in the operation

eg: In addition we use sum, another case we use concatenation.



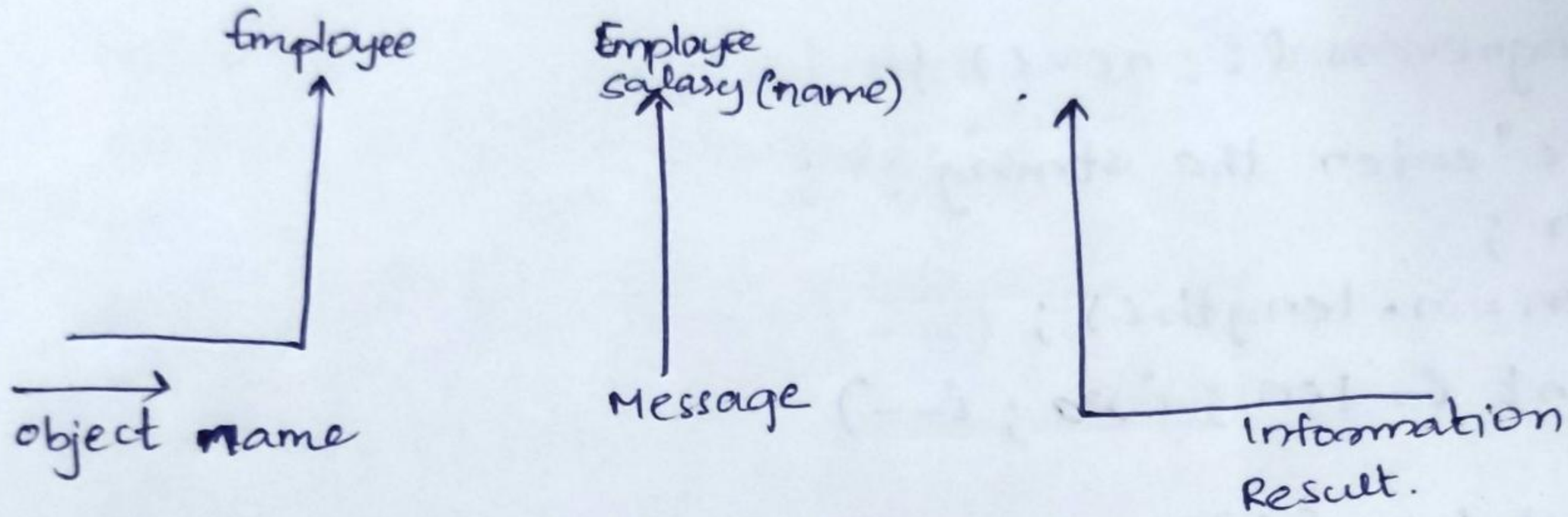
Dynamic building

A code associated with a given procedure call is not known until the time of the call at runtime.

Consider procedure 'Draw' in fig by inheritance, every object will have the same procedure.

At runtime, the code matching the object under the current reference will be called.

Message Passing



1) Creating a class that define object and their behaviour

1) Area of circle
2) Reverse a string
3) Prime or not

18/4/22

C++ functions

→ A block of code which only runs when it is called.

→ Pass the data, know as parameters in to the functions.

→ It is used to ~~pro~~ perform certain actions and they are important for reusing code. Define the code once, and use it many times.

Have some predefined functions

* main()

- Can create your own functions to perform certain action.

Syntax

```
void myfunction() {
    // code to be executed }
```

→ Name of the func

call a function

- It will be executed later, they are called

To call a function

- Write the function's name followed by the two parenthesis () and a semicolon.

Inside the main, call myfunction();

```
// create a function
void myfunction() {
 cout << "I just got executed!";
}

int main() {
// call the function
    myfunction();
    myfunction();
    return 0;
}
```

C++ function consists of two parts

- Declarsation
- Definition

<u>Declarsation</u>

Return type, name of func and parameters

Definition → the body of the fn.

<u>Parameters</u>

- informations can be passed to function parameters
- It act as variables inside the function.
- It is specified after the function name inside the parenthesis

- It can add many parameters separate them with a coma.

Syntax

```
void myfunction ( parameter 1 , parameter 2 , parameter 3)
{
  // code }
```

function takes a string called fname as parameter

```
void myfunction ( string fname) {
  cout << fname << "~~reference~~ \n";
  }
  int main ()
  {
    myfunction ("Liam");
    myfunction(" Roby ");
    return 0;
  }
```

file name

Default parameter value

Also use a default parameter value, by using equal =

we call the function, without an assignment it uses a default value

eg: void myfunction (string country = "Norway")
{
cout << country << "\n"; }

int main () {
myfunction ("sweeden");
myfunction ("India");
return 0;
}

19/4/22
C++ multiple parameters
Inside the fn you can add as many parameters
void myfunction (stringfname, int age)
{ cout << fname << "Aneen" << age "\";

}
main ()
{
myfunction ("fname", 3);

}

```
#include <iostream>
using namespace std;
void circle()
{
        float r;
        cout << "enter the radius: "
        cin >> r;
        cout << "area is " << 3.14 * r * r;
}
void rectangle ()
{
        float len, breadth;
        cout << "enter the length & breadth: ";
        cin >> len >> breadth;
        cout << "area of ▭ is " << len * breadth;
}
void triangle ()
{
        float base, height;
        cout << "enter the height & base of Δ: ";
        cin >> height >> base;
        cout << "area of Δ is " << 0.5 * base * height;
}
```

```
int
~~void~~ main()
{
    circle();
    rectangle();
    triangle();

return 0;
}

int choice;
cout<< "enter the choice: \n 1. circle \n 2. rectangle
                          \n3. triangle \n";
cin >> choice;
for (int i=1 ; ; i++) {
switch (choice)

{
    case 1 : circle(); break;
    case 2 : rectangle(); break;
    case 3 : triangle(); break;
    default : cout "Invalid";

}

}

}
```

Return Keyword

If the function to return a value, you can use a data type such as int, string instead of void, and use the keyword return inside the function.

```
int myfunction (int x) {

    return 5+x;
}
int main () {
    cout<<myfunction (3);
    return 0;
}
```

```
int myfunction (int x, int y) {

        return x+y;
}

int main()
    cout << myfunction (5,3);
    return 0;
```

Pass by Reference

We pass normal variables to the fun. Also pass a reference to the function. This can be useful when you need to change the value of arguments

```
void swapname (int &x, int &y) {
    int z = x
        x = y
        y = z
}
int main() {
    int firstnum = 10;
    int secondnum = 20;
    cout << "Before swap" << "\n";
    cout << "first num << secondnum << "\n";
```

// call the fn which will change the value of first num and second num.

```
    swapname (firstnum, secondnum)
    cout << "After swap" << "\n";
    cout << first num << secondnum << "\n";

    return 0;
}
```

Pass array as function parameter.

```
void myfunction (int mynumber[5]) {
    for (int i=0 ; i<5 ; i++)
    {
        cout << mynumbers [i] << "\n";
    }}
```

```
int main()
{
    int mynumber [5] = {10, 20, 30, 40, 50};
    myfunction (mynumber);
    return 0;
}
```

```
void max (int num1, int num2)
{
    if (num1>num2)
    {
        cout<<"num1 is the max number \n";
    }
    else
    {
        cout<<"num2 is the maximum \n";
    }
}
int main()
{
    int num1, num2;
    cout<<" Enter two numbers: ";
    cin>>num1>>num2;
    max (num1, num2);
    return 0;
}
```

21/4/22 Inline functions

→ Place the keyword inline before the function name and define the function before any calls are made.

→ Computer can ignore the inline qualifier in casedefined function is more than a line.

Syntax

```
inline return-type function name (parameter)
 {
     //function code
 }
```

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

inline int max ( int x , int y)
{ return (x>y) ? x:y; }
```

Default arguments and constant arguments

Default → the func called without passing arguments

working of default arg

Case 1 : No arg

```
void temp ( int =10 , float = 8.8)
    int main (){
```

```
temp()
}
void temp (int i, float f) {
    //code }
```

case 2: first argument is passed

```
void temp (int=10, float=8.8)
    int main() {
        temp(1); // if temp(1.1) is passed,
                    1 is taken (error)
    }
    void temp (int i, float f) {
        // code }
```

25/4/22

case 3: Two arguments are passed

```
void temp (int=10, float=8.8)
    int main() {
    temp(1, 10.6) }
    }
void temp (int i, float f) {
        // code }
```

Function overloading

→ multiple functions have same ~~have~~ name with different parameters.

```
int     myfunction (int x)
float   myfunction (float x)
double  myfunction (double x, double y)
```

eg: Two fns that add members of different data type.

```
int
~~void~~ sum (int i=1, ~~int j~~ int j=5)
{
    return i+j;
}
int sum (float f=5.1, float g=5.5)
{
    return f+g;
}
int main()
{
    cout << (1,5);
    cout << (2.3, 5.6);
}
```

constant arguments

→ put 'const' infront of a parameter
it means that it can't be modified in the fn

```
const int var=5;
```

→ const on function parameters passed by references or pointer

function prototype/

```
float cir ( float , float pi = 3.14)
                          ↳ const argument
{
  //main fn
  void main ()
  {
  //declare variable as float
    float r , area ;
  // Input radius
}
  code
  cout<< "enter the radius: ";
  cin>>r ;
  ~~area = pi*r*r~~;
  ~~cout<< "area is " << area~~;
}  cin (r) ;
```

Recursion

→ function call itself

→ one of the statements on the function definition makes a call to the same function, in which it is present.

→ A recursion fn also possess a base case which returns the program control from the current instances of the function to all back to the calling function.

```
//factorial
#include <iostream>
using namespace std ;
void fact()
{
    int ~~fact~~ = fact ~~int i = fact~~
                i
    ~~fact(int is~~
    return ~~fact * = fact - 1~~; fact(n) *fact(n-1);

  }
int main ()
{   cout<< "enter the no to find factorial : ";
    cin>>n;
    fact (n) ;
}
```

26/4/22

```
// strlen using recursion
#include <iostream>
using namespace std;

int strlen (str word)
{
    while ( str word [i] !=="/0")
    { count++;
      i++;
      strlen (word [i]);
    }
    return 1;
}
int main()
{
    string word;
    cout<<"enter the word :";
    cin.getline >> word;
    strlen (word); cout<< count;
    return 0;
}
```

friend function

- To handle some specific tasks related to class objects
- Two classes to share a particular fn
  eg: two classes manager and scientist have been defined, use a fn income_tax() to operate on both objects of both the classes
- Such situation c++ allows a common function to made friendly with these classes
- Such function need to be member of any these classes
  eg: Using friend function to add data objects of the different classes

```
class ABC    //
 //.....
class XYZ
{   int data;
  public:
    void setvalue (int value)
      {
        data = value; }

friend void add (xyz , ABC) ;//
  }
void add (xyz obj , ABC obj 2) // friend defn part
```

```
cout << "sum of data value of XYZ and ABC
  using friend function = " << obj.data + obj2.data

}

int main()
{
  XYZ x;
  ABC A;
  x.setvalue(10);
  A.setvalue(20);
  add(x, A)
  return 0;
}
```

28/4/22 IInd Module

Class, Objects

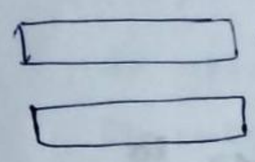
Class → New way of creating and implementing user defined data types.

Memory allocation for objects

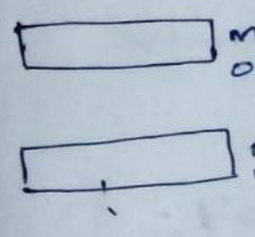
→ Member function is created and placed in the memory space only one, when they are defined as part of the class specification.

All objects belonging to the same class use same member functions, no separate space is allocated for member function, when the objects are created, separate memory locations for the objects are essential.

common for all objects



when fn defined memory created



Objects of Memory

Static Data members → class members that use Keyword 'static'

↳ It is normally used to maintain values ~~to~~ common to entire class

↳ visible only ~~in the~~ within the class, but its lifetime is entire prgm.

↳ It is initialized to zero, when the first object of the class is created.

```
#include <iostream>
#include <strigng.h>
using namespace std;

class student {   // create a class

private:
    int roll no;
    char ~~class~~ name[10];
    int marks;

Public:

  static int objectcount;  // static data member
  student ()  // (fn) constructor to define the objects
  {
     objectcount ++;
  }
  void getdata () {
    cout << "enter name"; ~~class, roll no"~~;
    ~~cin>>name~~
    cin.getline(name, 10);
    cout << "enter roll no";
    cin >> roll no;
    cout << " enter marks";
    cin>>marks;

  }
  void putdata()
  {
     :
  }
  int student :: objectcount = 0;

  int main()
  {
     student s1;  // create object for student class
      s1. getdata();
      s1. putdata();
     cout << " the total number of objects created" << endl;
     Student :: Objectcount << endl;
      return 0;
  }
```

9/5/21 Static member functions

1) A member function that is declared static has the following properties

a) A static function can have access to only other static members (functions or variables)

b) A static member function can be called using the ~~object~~ class name (instead of its objects)

class_name : : function_name;

Eg: Program illustrates the implementation of these characteristics static function show_count() displays the number of objects created till that moment

A count of number of objects created is maintained by the static variable count.

The function show code() displays the code

Static member functions

```
class test
{
  int code;
  static int count ;// static member variable
  Public :
    void setcode();
    {
      code = count++;
    }
```

```
void showcode()
 {
    cout << "object number : " << code << "\n";
    static void showcount () // static mem fn
    {
       cout << "count" << count << "\n";
    }

}
int test :: count ;
int main()
{
  test t1, t2 ;
  t1 . setcode ();
  t2 . setcode ();
  test : : showcode () // access static mem fn
  t1 . showcode ();
  t2 . showcode ();
  return 0 ;
}
```

constructor and destructor

→ It is a special member function of a class whose task is to initialize the objects of the class

→ A destructor is also member function of a class that is instataneously called whenever the object is destroyed.

It can be defined manually with asguments or without asguments.

Many constructors in class. It can be overloaded but it can't be inherited.

1) copy constructor
2) Default constructor
3) Parameterized constructor

Default constructor

Doesn't take any arguments

Parameterized constructor

Programmer can add use the parameter within a constructor of required. It helps to assign critical values to objects at the time of execution.

Copy constructor

Special type of constructor which takes an object as an argument and it is used to copy the values of data members of one object in to another object. In this case, copy constructors are used to destroying and intializing an object from another object.

```
class copcon{
    int a,b;
    public:
        copycon (int x , int y)
        {
            a = x;
            b = y;
            cout<<"\n Here is the initialization of
            constructor ";
        }

    void Display()
    {
        cout<<"\n value : \t"<< a "\t"<< b ;
    }
}
int main()
{
    copycon object(30, 40); // copy constructor
    copycon obj 2 = object;
}
```

10/5/22

Operator Overloading

→ Ability to provide operators with a special meaning for a datatype. This ability called as operator overloading.

eg: → Overload operator '+' in a class like string so that we can concatenate two strings by ~~ass~~ just using '+'.

Assement I

1) Overloading binary operator using friends
2) Manipulation of strings using operators
3) Overloading unary operators
4) Overloading binary operators