Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

* Human readable code → Sourse code
* Machine readable code → Object Code

* Translator is needed to convert humand readable language into machine readable language.

There are three types of translator →

* Compiler
* Interpreter
* Assembler

* Compiler: Reads and translate the entire program together at a time.
C, C++, C#, Java, Cobal are compiler based programming language.

* Interpreter: It translates source code line by line into machine language.
Phython, JavaScript are Interpreter based language.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

* Assembler: It translates assembly language into machine language.

* Compiler works on 2 steps:

Step - 1

Source code → Compiler → List of errors
                       → Object Code generate

Step - 2

Input → Object Code (.obj) → Output (.exe)

* Interpreter works on one step:

Source Code → Interpete → Output (.exe)

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Difference between Compiler and Interpreter

Compiler	Interpreter
* Read and tramlate the whole code at a time.	* Read and translate the whole code line by line.
* Takes less time to execute the program.	* Takes more time to execute the program.
* Take more memory	* Take less memory.
* Show all the errors at a time.	* Show all the errors line by line.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

What is C++ ?

* C++ is known to be a very powerful computer programming language.
* C++ is a general purpose (FORTRAN → Mathematical purpose, COBLE → Business purpose). Can be used for any purpose like - to develop games, desktop apps.
* Case - Sensitive
* Object oriented programming language.
* C++ can be used instead of C but C can't be.

Usage of C++ :

* Used to develop game engine, games, desktop apps, art applications, music player etc.
* C++ being highly used to write device drivers and others software.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

C++ was developed in 1980 by Strousstrup.

↓

Father/ founder of C++

C++ was derived from C and is largely based on it.

Features of C++:

* Simple, high-level, rich library, memory managment.
* Fast speed, Object-Oriented, Compiler based.

C	C++
* C is a procedural programming language.	* C++ is both procedural and object oriented programing language.
* C is a subset of C++	* C++ is superset of C

Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

IDE : Integrated Development Environment

↓

Tool Software

C

#include<stdio.h> → Header File

- Header file has 3 part :
  - * Function Definitions
  - * Data type Definitions
  - * Macros Definitions

- There are 2 types of Header File :
  - * Pre-existing header file. (Eup - #include<stdio.h>)
  - * User-Defined header file.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

C++

```
# include <iostream>
using name space std;
int main ( )
{


return 0;
}
```

* A lot of package of files are included in # include <iostream> and by using the code using name space std; we are commanding that we're using std package file from them.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

cout << → extraction operator

cin >> → insertion operator

Escape Sequence / Backslash character:

\a → (Alert) Bell
\b → BackSpace
\f → Form Feed
\n → NewLine
\r → Carrige Return
\t → Horizontal Tab
\v → Vertical Tab
\0 → Null character
\' → Single Quote
\" → Doble Quote
\\ → Backslash
\? → Questionmark

Token

→ Keywords
→ Constants
→ Identifier
→ Strings
→ Special Symbols
→ Operators

Sub:

SAT SUN MON TUE WED THU FRI

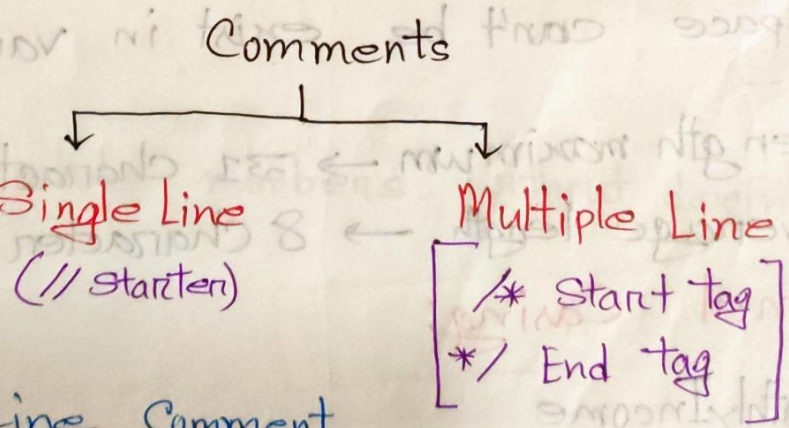
DATE: / /

Key words:

int	float	double	char	short	typedef
long	break	continue	if	else	union
switch	case	void	return	public	unsigned
default	private	for	goto	protected	virtual
do	while	enum	getch	operator	Volatile
do while	class	inline	delete	register	asm
auto	const	throw	friend	template	new
catch	extern	this	struct	sizeof	signed
			static		try

Comment:

Only for personal use. It's not mendetory. It's used to explaid about code in short sentence.




```
//Single Line Comment.
/* Multiple Line
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Comment
*/

Variable:

It's user defined.
Symbolic name for a memory location
Variable declaration syntax: data_type Variable;

Variable writing rules:

* Alphabats (A—Z) (a-z)
* Digit (0-9)
* Special Character $(Dollar Sign) __(UnderLine)
* Can't be start with digit.
* Can't be any keyword, function
* Space can't be exist in variable name.
* Length maximum → 31 character
* Average length → 8 character

Camel Casing:

monthlyIncome
classRoll

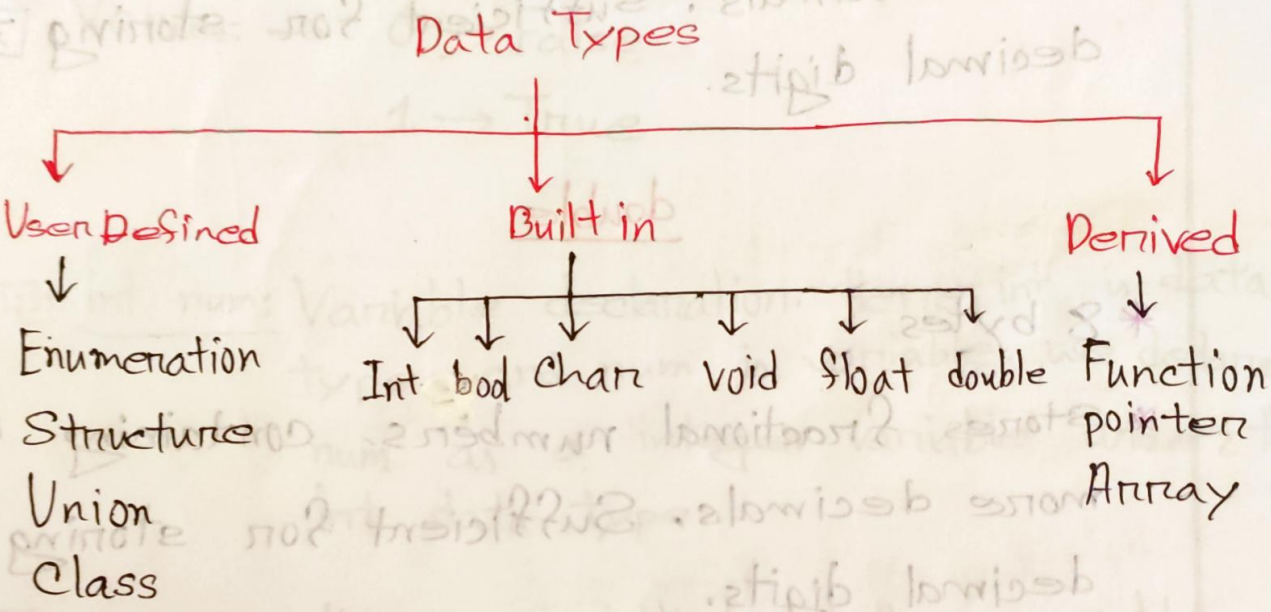
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Data Type:

Data_type Variable_name



int

* 4 bytes
* stores whole numbers, without decimals

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

float

* 4 bytes
* Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits.

double

* 8 bytes
* Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits.

char

* 1 byte
* Stores a single character/Letter/Number
  exp. → ?*#
* Has Ascii Values
  - Decimal
  - HexaDecimal

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

bool

* 1 byte
* Stores true and false
* Values → 0 → False
  1 → True

`int num;` Variable declaration. Here int is data type and num is variable. we declared num as a integer variable under the int data type.

`num = 10;` Here we initialized the variable.

variable Value will must be int type. because the data type is int.

`int num=5, num2=10;` Here we declared and initialized the variable at a time in one line.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

In "char" data type if we use the Ascii value of character then it will show the output. if we don't use the Ascii value and use the direct character then we must write the single character insid single Quote. ('A')

Operator:

There are 8 types of operator.

1. Arithmetic Operator
2. Assingment "
3. Relational "
4. Logical "
5. Conditional "
6. Unary "
7. Biwise "
8. Special "

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Arithmetic Operator

+ Addition

- Subtraction

* Multiplication

/ Division

% Modulus

++ increment

-- Decrement

int / int = int

float / int = float

int / float = float

float / float = float

Convert a variable data type forcely in the middle of program. its called type Casting.

```
# include <iomanip>
        ↓
   1. set precision()
        ↓
   2. setw()
```

To use the number 1 and 2 function we must

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

use th iomanip Header file.

n = 1 (value of n is 1)

n == 1 (value of n and 1 is equal)

L.H.S = R.H.S

#Assignment Operator:

	expl:	same as
=	→ x = 5	→ x = 5
+=	→ x += 3	→ x = x + 3
-=	→ x -= 3	→ x = x - 3
*=	→ x *= 3	→ x = x * 3
/=	→ x /= 3	→ x = x / 3
%=	→ x %= 3	→ x = x % 3

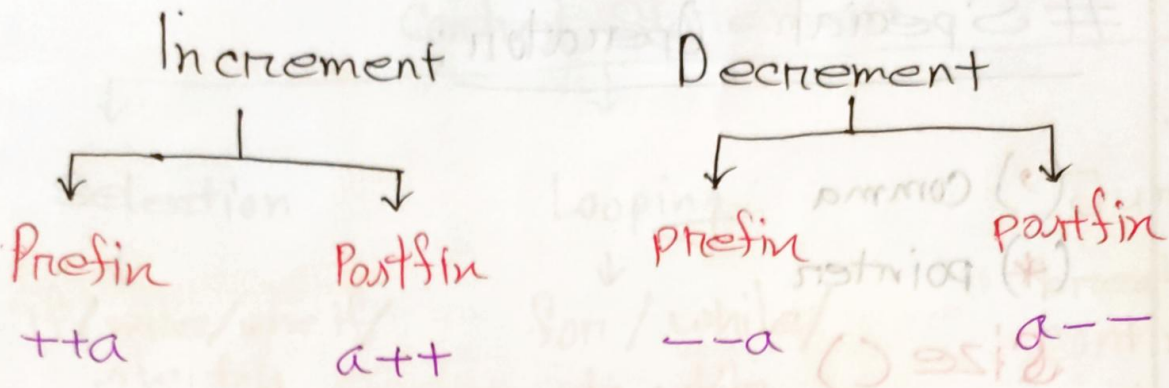
|= → x |= 3 → x = x | 3 ^= → x ^= 3 → x = x ^ 3

&= → x &= 3 → x = x & 3 >>= → x >>= 3 → x = x >> 3

#Unary Operator:

\+ → Unary plus
− → Unary minus
} used for changing sign + or −

++ → increment
−− → Decrement
} to increase or decrease value of a variable −1 or +1



++a → increments a by one - before it is used

a++ → increments a by one - after it is used

--a → decremets a by one - before it is used

a-- → decrements a by one - after it is used

Precidence level of operator is

(*, /, %)

(+, -)

→ #1st one serve first

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Special Operator:

(,) comma

(*) pointer

Size()

Relational Operator:

> → x>y → x is greater than y

>= → x>=y → x is greater and equal to y

< → x<y → x is less than y

<= → x<=y → x is less than and equal to y

== → L.H.S=R.H.S → x and y are equal(same)

!= → x!=y → x and y are not equal

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Control Statement

- Selection → if / else / else if / switch
- Looping → for / while / do while
- Jump → break / continue / return

if

```
if (condition)
{ codes to be executed; True/if's body
}
Rest of the code
```

variable = tolower(variable);
use to convert a single character upper to lower.

variable = toupper(variable);
use to convert a single character lowe to upper

if / else

```
if (condition)
{ True statement;
}
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

```
else
{ false statement;
}
```

Bitwise operator;

& → And. Bitwise

| → OR. "

^ → XOR. "

>> → Shift Right. " → Division by 2

<< → " Left " → Multiplication by 2

~ → Not. Bitwise

Decimal → Binary → + − * /

→ Decimal → output

(ans binary to decimal)

Sum, multiplication. work in true chart.

convert the value of variable in binary then work with it then convert it in decimal and finally show the output.

- f

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

```
if (condition)
{ Codes;
}
else if ();
{ codes;
}
else if ();
{ codes;
}
else
```

else if

break Statement

*break statement is used to break loop or switch statement.

```
break;
```

* It breaks the current flow of the program at specified condition.

*In case of innerloop, it breaks only inner loop.

```
for (int i=1; i<=100; i++)
  {
   if (i==10)
      {                    // it will print 1 to 9
       break;
      }
      cout << i << endl;
  }
```

*Can be use for switch or loop.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Switch statement:

```
switch (expression)
{
case value1: //codes;
             break;
case value2: //codes;
             break;
case valuen: //codes;
             break;
default : //codes;
}
```

continue statement

The continue statement skips the current iteration of a loop.

```
continue;
```

* Can be used in loop except switch.

```
for (int i=1; i<=100; i++)
  { if (i==10)
    {
     continue;
    }
    cout<< i<<endl;
  } //it will print 1 to 100 except 10
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Logical operator

Logical And ⟶ &&

" OR ⟶ ||

" NOT ⟶ !

Logical And

```
x=13, Y=19;
if (x>15 && Y<2)
{cout<< "yes";}
if(x>11 && Y<2)
{cout<< "No";}
if(x>11 && Y>2)
{cout<< "Hi";
```

if the all the condition are true then it will be true and enter the body.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Logical OR:

```
if (condition1 || condition2)
{
  // coded;
}
```

```
x=13, Y=19;
if (x>15 || Y<2)
{
 cout<< "yes";
}
if (x>11 || Y<2)
 { cout<<"NO";
}
```

if one condition is true then its true and it'll enter the body.

~~if (x) ||~~

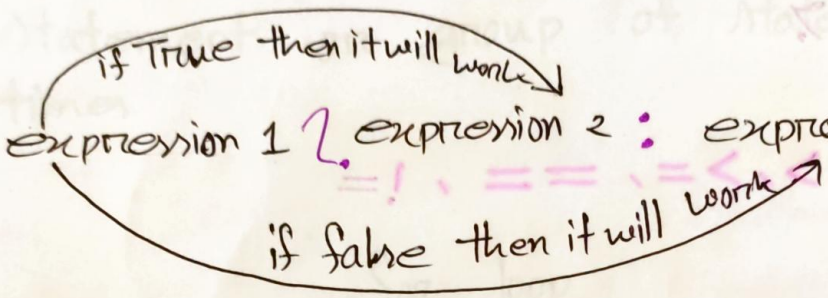
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

#Conditional Operator/Ternary Operator

?: // it works like if/else



exp:

```
int x=20, Y=15;
int large=x>Y?x:Y;
cout<<large;
```

output will be x that means 20

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Presidence level of operators

!, ++, --, ( )

*, /, %

+ -

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

&&

||

=

First come first serve.

Formatting Output: cout<< objectives;

showpoint; //to showpoint in output

noshowpoint; // will not showpoint after this objectives

setprecision; // to show a fixed amount of digit after point

fixed; // show 6 digit after point

setw(); // to align the output in a serial

Headerfile for those objectives = #include<iomanip>

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Every loop has 3 part – initialization/condition/increament/decreament

Looping/Iteration:

A loop statement allows us to execute a statement or group of statements multiple times.

For loop

Initialization ↑ increament/decrement ↑

```
for (conditions and others)
{
  cout<<    ;
}
```

```
(int=1; i<=5; i++)
```

↓ condition

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

while loop

```
initialization;
while (condition)
{
  Statements
  increment/decrement;
}
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

do while loop

```
initialization;
do {
    Statements
    increment/decrement;
} while(condition);
```

print first then check the condition

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Difference between while and do while loop:

The do while loop will execute at least one time even if the condition is false.

```
int x=10;
while (x<5)
{
 cout<<"Hello"<<endl;
  i++;
}
```

```
int x=10;
do {
 cout<<"Hello"<<endl;
  i++;
}while (x<5);
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Array

Declaration of array

* An array is a collection of variables of same ~~type~~ data type.

```
data_type array_name [array_size];
```

Variable (array name)

```
int marks[100]  // marks 1 to marks 100
```

int → data type

marks[100] → array size

```
int marks[100];
float marks[100];
double marks[100];
char marks[100];
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Initialization of array

int marks[5] → Array size 5

↓

variable are marks[0] marks[1] marks[2]

marks[3] marks[4]

Index number

Initialization way:

1:

```
marks[0]=80;
marks[1]=65;
marks[2]=75;
marks[3]=82;
marks[4]=45
```

2:

```
int marks[5] = {80, 65, 75, 82, 45};
```

3:

```
int marks[] = {80, 65, 75, 82, 45};
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Printing an Array

Normal

```
int marks[ ] = {80, 65, 75, 82, 45};
```

index number

```
cout << marks[0]; // to print 80
cout << marks[1]; // print 65
cout << marks[2]; // print 75
cout << marks[3]; // print 82
cout << marks[4]; // print 45
```

Output → 80 65 75 82 45

In loop

```
int marks[ ] = {80, 65, 75, 82, 45};
for (i=0; i<4; i++)
 {
   cout << marks[i];
 }
```

Output → 80 65 75 82 45

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Getting user input

Normally

```
int marks [5];

cin >> marks [0];
cin >> marks [1];
   :
cin >> marks [4];
```

In loop

```
int marks [5];
for (i=0; i<=4; i++)
{
  cin >> marks [i];
}
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Getting input and show output:

```
int marks[5];
//input
for (int i=0; i<5; i++)
{
  cin>>marks[i];
}
//output
for (int i=0; i<5; i++)
{
  cout<<marks[i]<<" ";
}
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Types of Array

① One dimensional (1-D array/Linear array)

example:

```
int marks[10];
```

⑪ Multi dimensional array.

ⓐ Two dimensional array (2-D array/matrix array).

example:

```
int marks[2][3];
```

ⓑ Three dimensional array (3-D array)

example:

```
int marks[2][3][2];
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

String

A string is a sequence of characters.

example:

"This is a string"

"a"

"" → empty string

"string"

String presentation:

There are 2 ways to present a string.

1. The c style character string(character type array)
2. String class

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

C style character string / Character array

1st style

```
char message[6] = {'H', 'e', 'l', 'l', 'o'}
```

null character → '\0'

↳ End of a string

* Size of character type array must ~~be~~ accurate +1 / more

2nd style

```
char message[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

Without declaring size, use of \0 (null char) is ~~mandatory~~.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

3rd style

```
char message[] = "Hello";
```

* To get input including space ( ) and showing it output including space ( ) we can use two function.

① gets(variable);

↓

it's a function of C language that's why a header file of " " is mendatory which is = #include<stdio.h>

② cin.getline(variable, size of array);

↓

it's a function of c++. So no need to add any header file extra.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

#String Library Function

Header File → # include<string>

strlen()
strcpy()
strcat()
strupr()
strlwr()
strcmp()

strlen()

~~int~~ it will show the size of a string. which is a integer number. That's why to store this size value we need a integer type variable must.

example: char name[] = "Ekramul";

int length = strlen(name);

length → variable to store length size

name → variable want to show the size

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Strcpy ( )

We use this to copy ~~and replaced~~ the source elements and replace it in destinations variables.

```
char name1[], name2[];
strcpy(name1, name2);
```

name2 → Source

name1 → destination

↓

Copy the source elements and replace it in destination. Here first one is always destination and after comma (,) 2nd one is always source.

```
char name1[]= "Hello";   char name2[] = "World";
strcpy(name1, name2);   output → World (Hello will replaced)
```

name1[]="Hello"; → strcpy(,) → name2[]="World";

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Strcat (,)

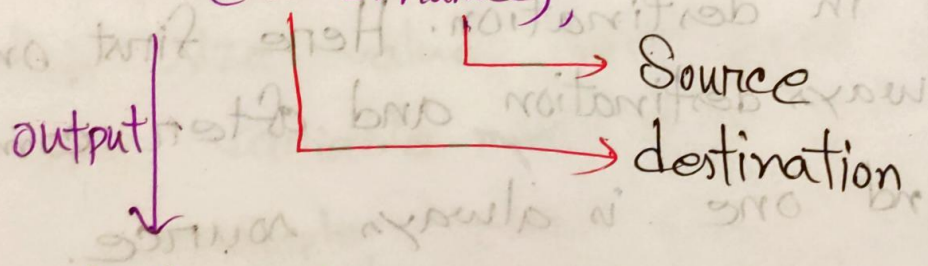
concatenate the ~~value~~ strings of two string variable.

```
strcat(destination, source);
```

Here source ones elements will add with dertinations elements.

```
name1[] = "Hello";  name2[] = "World"

strcat(name1, name2);
```



output

HelloWorld

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

strupr()

Use to convert strings from lower case to upper case.

```
name1[] = "hello";

strupr(name1);
```

↓output

HELLO

strlwr()

Use to convert string upper case to lower case.

```
name1[] = "HELLO";
```

↓output

hello

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

strcmp()

Use to compain one string with another one. we need a int type variable to store the capanision value.

* if the elements between two string matched then the value will be 0.
* if they don't, then the value will be -1 or 1.

```
Char name1[] = "Hello";   Char name3[] = "Hello";
Char name2[] = "World";
int value1, value2;

value1 = strcmp(name1, name3);
```

↓

Here value1's value will be 0, because they are matched.

* compain first string with the second string.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Copairison values →

(i) Zero (0) : All characters of string are same.

(ii) Greater than zero (>0) : If character in left string comes ~~befor~~ after the character of right string.

(iii) Less than zero (<0) : if character in left string comes before the character of right string.

We have to run a condition to check that if those strings are same or not.

if/else

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Pointer

* Pointer is a variable, that stores/points the address of another variable.

Declaration of pointer;

```
int *p;
```

`*p;` — pointer variable

Symbols: &, *

&

*it is an → address of operator

* This symbol is used to get the address of the variable.

Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

*

*It is an → value at operator

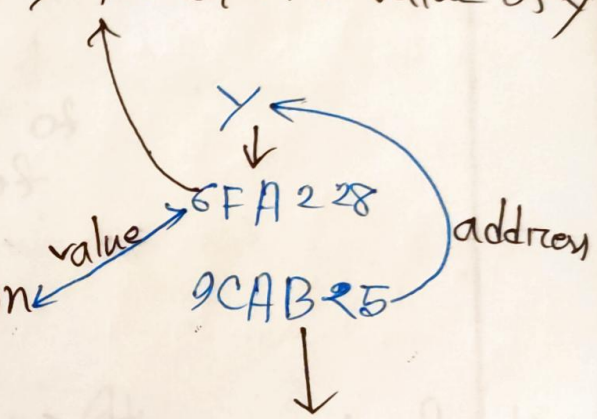
* This symbol is used to get the value of the variable that the pointer is pointing to.

in y pointer it is value of y

x = variable

5 = Value

6FA228 = memory location



as y(pointer) is a variable itself. so y has an address also. this is address of y

```
int x= 10, y= 20;
int *p, *q;
p = &x; //address of x is stored here
q = &y; //  "   "  x  "  y  "  "  "  "
cout<<*p<<*q<<p<<q<<&p<<&q;
```

*p → it's value of x

*q → value of y

p → address of x

q → address of y

&p → address of p

&q → address of q

Sub: Final Term

SAT SUN MON TUE WED THU FRI

DATE: / /

Function

*A function is a group of statements that perform a particular task.

*If you need to do same thing again and again then you may use function.
↓
in different place, middle of program, in any place of program.

```
int main ()
{
//group of statements
}
```

↳ Its a main function

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

Type of function:

① Buit in function / Library Function

- tolower()
- toupper()
- strncat()
- sizeof()
- setw()

① User defined function →→

How does a function work?

Input → function → output

A → tolower() → a

*takes an input process it and show the output.

Why do we need function?

* To do a same process again and again in different place of a single program.

example: Addition, Multiplication, Division process

Sub: _____

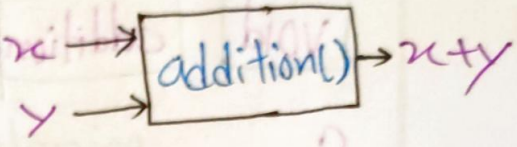
SAT SUN MON TUE WED THU FRI

DATE: / /

```
int main()
{
  int x=10, y=20;
  int sum = x+y;
  cout<< sum;

  x=20, y=20;
  sum = x+y;
  cout<< sum;

  x=20, y=30;
  sum= x+y;
  cout<<sum;
```



* We are doing addition of the value of x and y ~~again~~, and showing the result as output.

we are doing this thing again and again.

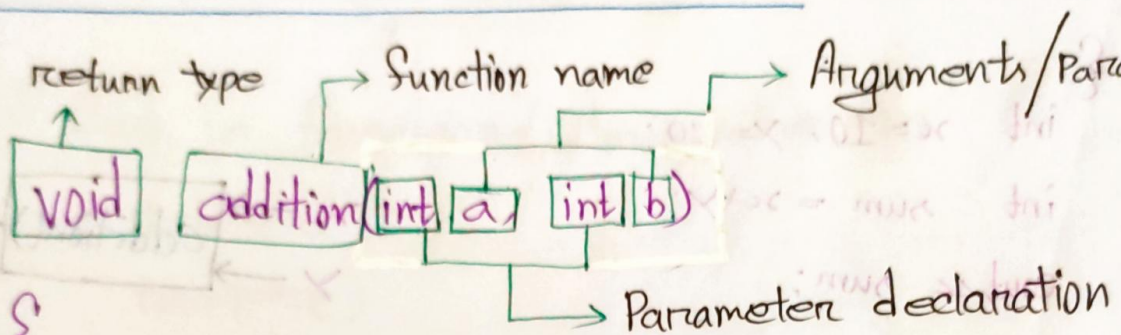
. ~~This~~ For this kind of things we can use function.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

How to declare a function:




```
{
  int sum=a+b;
  cout<< sum;
}
```

Advantage of function:

```
//function calling
int main()
{
 //function calling start
 addition(10, 20);
 addition(20, 20);
 addition(20, 30);
}
```

we are calling this function

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

① Code re-usability

② Can use the same function for different inputs.

Syntax of a function declaring

```
return_type  function_name(Parameter_list)
{
//function_body /////// declarations and statements
}
```

Function Prototype:

* The first line of the function syntax.

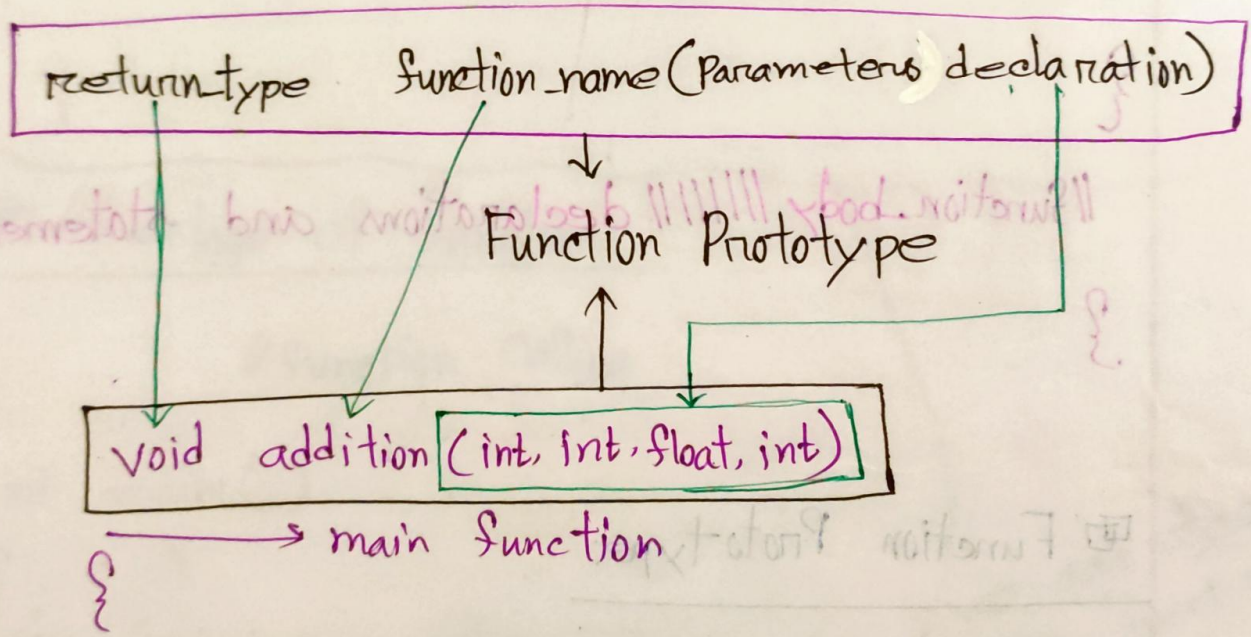
* It contains everything that others need to know about the function in order to use it.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

```
void addition(int a, int b)
{
  int sum=a+b;
     cout<< sum;
}
```




```
{
  //Function body
}
```

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

* If we declare the function body after the main function, then we must ~~use the~~ declare the prototype before the main function. Otherwise not.

Return a Value from function

* if we declare returntype, then befor(}) a return(); is must. Otherwise it will return meaningless value or unlogical value.

* returned value will be present in function.

```
int addition(int a, int b)
{
    int sum = a+b;
    return sum;
}
```

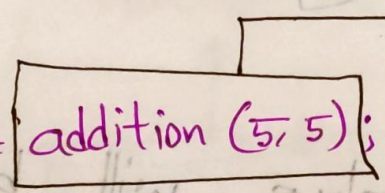
```
// addition(5, 5);
```

Sub: _____ SAT SUN MON TUE WED THU FRI ☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

* When we'll call the function addition. Then it'll add 5 and 5 the result 10 will be stored in sum and it'll return sum.

* Then in main function the value of sum will be present in addition(5,5). we have to work with this function to get the sum value.



cout<< result; → output = 10

cout<< addition(5,5); → " = 10

* Program will returned the value via addition(5,5)

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Passing Arguments to a function

There are two ways to pass arguments to a function;

1. Pass by value
2. Pass by reference

1. Pass by Value

* In case of pass by value, a copy of the argument is passed to the function

```
int main()
{
  int x=10;
  cout << "Before calling function x= "<<x<<endl;
  display(x);        // actual Parameter
  cout<<x<<endl;
  getch();
}

void display(int num)   // formal parameter
{
  num = 20;
}
```

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

⑪ Pass by reference

* In case of pass by reference copies an arguments address into the formal parameter.

```
int main()
{
  int x=10;
  cout<< "Before calling the function x= "<<x<<endl;

  display (&x);

  cout<< "After calling the function x= "<<x<<endl;
  getch();
}
```

Actual parameter

```
void display (int *num)
{
  *num=20;
}
```

Formal Parameter

* It pass the original arguments not copy, that's why the original value of the argumets will be updated and changed.

Sub: SAT SUN MON TUE WED THU FRI

DATE: / /

Local variable and Global variable.

① Local Variable

```
int main()
{
    int x= 10;
    cout<< "Inside the main function x= "<<x<<endl;
    display();
    getch();
}
```

Here x is local variable for main function body.

It's the area of x local variable. We're unable to use it, outside of main function body.

```
void display();
{
    cout<< x;
}
```

x is only for main function not for display function.

Here no x is declared. If we want to use the x of main function here, we can't.

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

⑪ Global Variable

```
using namespace std;
int globalvariable = 10; // have to declare it outside in and at first of all function
void display().
{
    int localfondisplay= 10;
    cout<< global_variable; //ok. it's public property
    cout<<localvari; // not ok. it's main function's property.
}
int main()
{
    int localvari= 90;
    cout<< localvari; //ok we can
    cout<<global_variable; //ok we can.
    getch();
}
```

→ It's global variable. for all

→ It's local variable for display

Sub:

Scope Resolution Operator

① To Access Global Variable

```
using namespace std;

int x=313;                    → global variable for all
int main()
{
  int x=1213;                 → local variable for main
  cout<< "Value of x is: "<< x;
}
```

output will be 1213
because → local variable has highest preidence level more than global variable.

local vari > global vari

* We can't access the global variable by writting normally, (if the variable name of global and local are same) in this case program will give the highest priority to the local variable.

Sub:

Use of Scope Resolution (::)

① To access global variable
② To define a function outside a class
③ To access a class's static variable

Scope Resolution Ope

① To Access Global Variable

```
using namespace std;

int x=313;                          // x → global

int main()
{
  int x = 1213;                     // x → local variable for main
  cout<< " Value of x is: "<< x;
}
```

output will be 1213

because → local variable has highest preidence level more than global variable.

local vari > global vari

* We can't access the global variable by writting normally, (if the variable name of global and local are same) in this case program will give the highest priority to the local variable.

Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

```
using namespace std;
int x=313;          → Global variable
int main()
{
   int x=1213;      → Local variable
   cout<< "Value=" << x;     → Here it will access the local variable.
   cout<< "Value=" << ::x;
}
```

Scope resolution operator

Accessing global variable

Use of scope resolution:

* To access the global variable we have to use scope resolution operator (::). If the local variable name and global variable name are same of different, it will always work.

* After accessing global variable by using scope resolution. We can do with it, whatever we want. We can show its value a

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Output. We can do arithmetic operation.

Or Even we can change the value of global variable.

```
using namespace;

int x=10;

int main()
{
  cout<<::x;
  //Changing global value
  ::x=::x+5; → Here is global x is now(15)
  cout<<::x*2;
```

↓

Output 30

Sub:

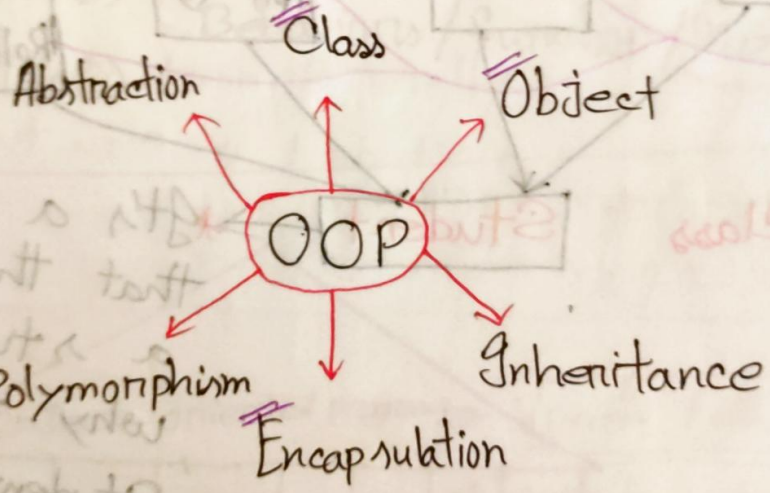
SAT SUN MON TUE WED THU FRI

DATE: / /

#Object Oriented Programming OOP

- O = Object
- O = Oriented
- P = Programming
- C + OOP → C++

Concept/Features of OOP:



* If any programming language has those features, then we can call it as OOP language.

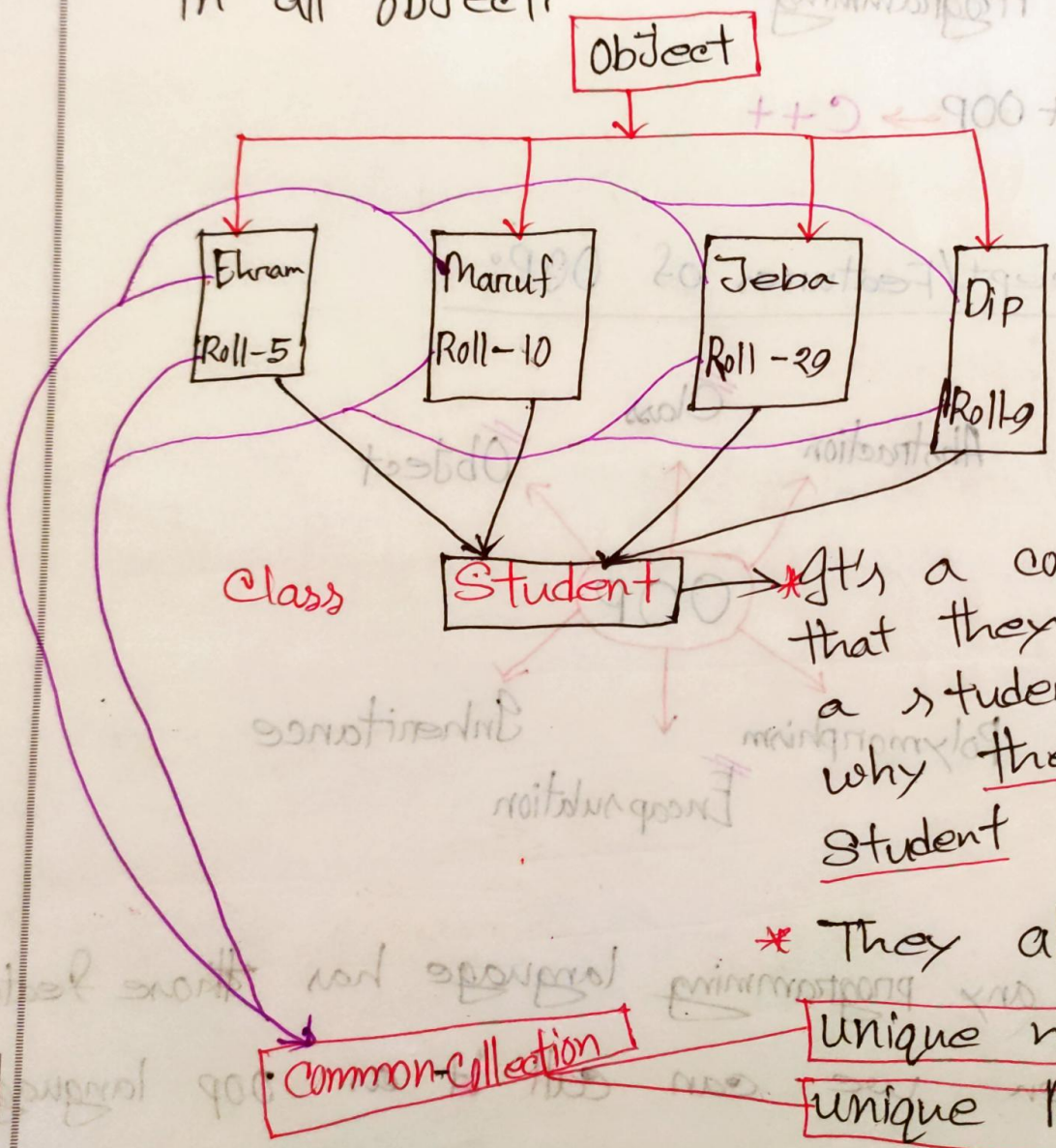
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

What is class and Object?.

* Class is a common collection of some object. that means a common things is present in all object.



* It's a common things that they all are a student. That's why their class is Student

* They all have a unique name and unique Roll.

Sub: _____

SAT SUN MON TUE WED THU FRI

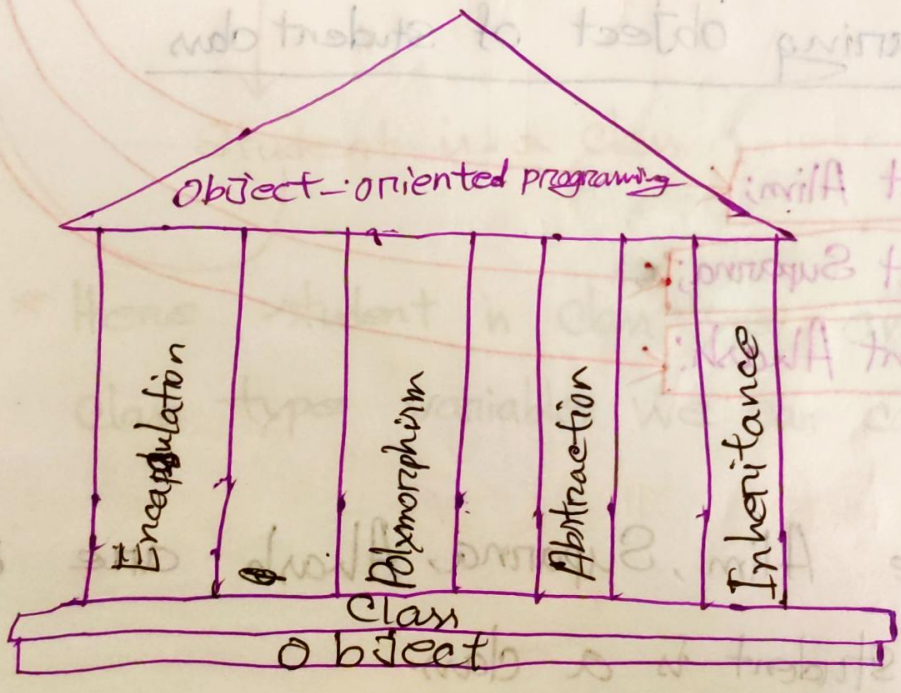
DATE: / /

* Two common things in Object is, they has
  (i) State / variable / attribute
  (ii) Behaviors / function / Method

name
Roll
} States / variable / attribute

Class member

SetName()
SetRollNo()
} Behaviors / function / Method

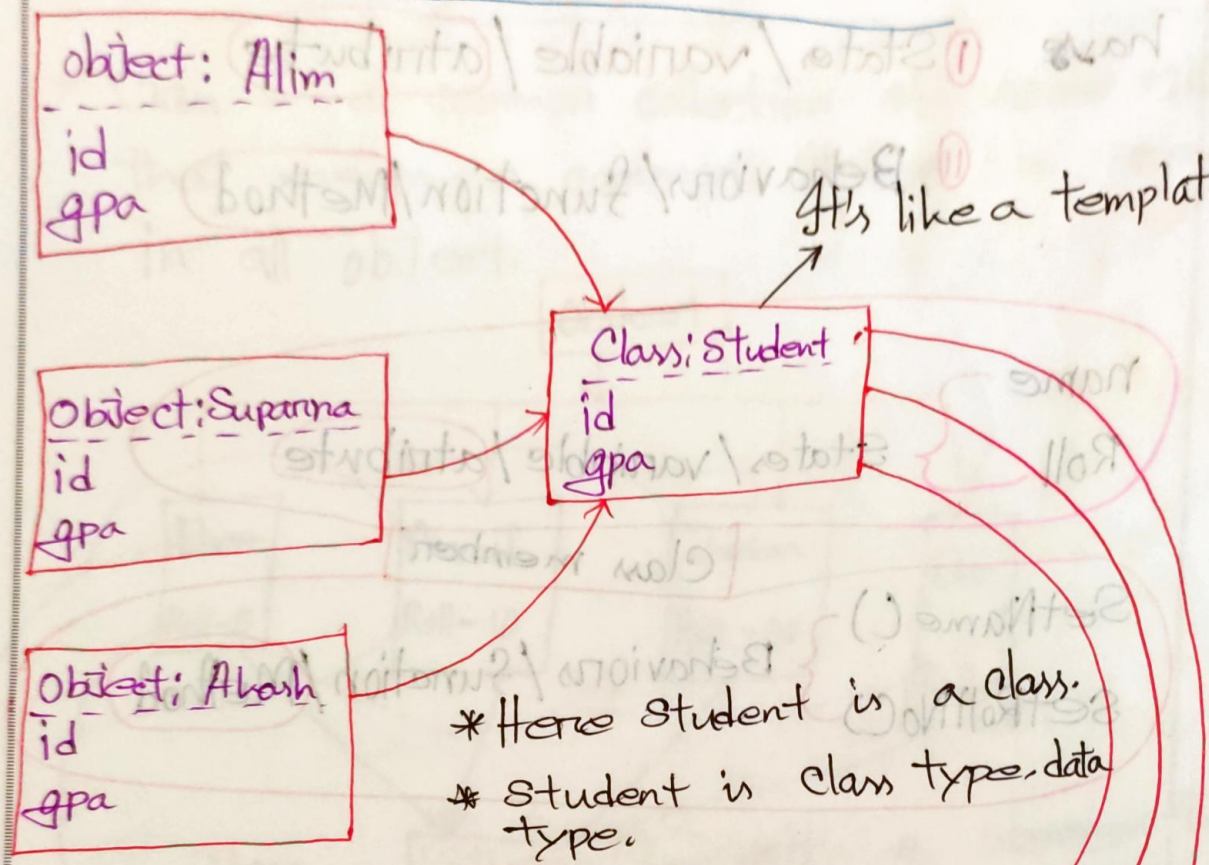


Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Understanding Class and object



* Here Student is a class.
* Student is class type, data type.

Declaring object of student class

Student Alim;
Student Suparna;
Student Akash;

* Here Alim, Suparna, Akash are object
* And Student is a class.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

* What is class?

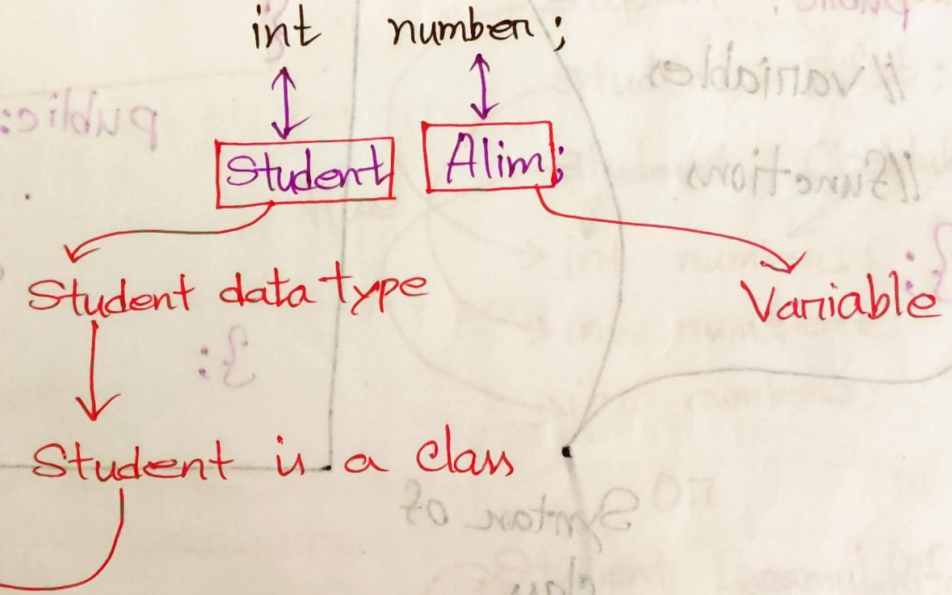
→ A class is a template from which individual object can be created.

→ Here class is user defined data type like user defined function.

* What is object?

→ Any class type variable is called an object

example:



* Here student is class type and Alim is a class type variable. we can call object also.

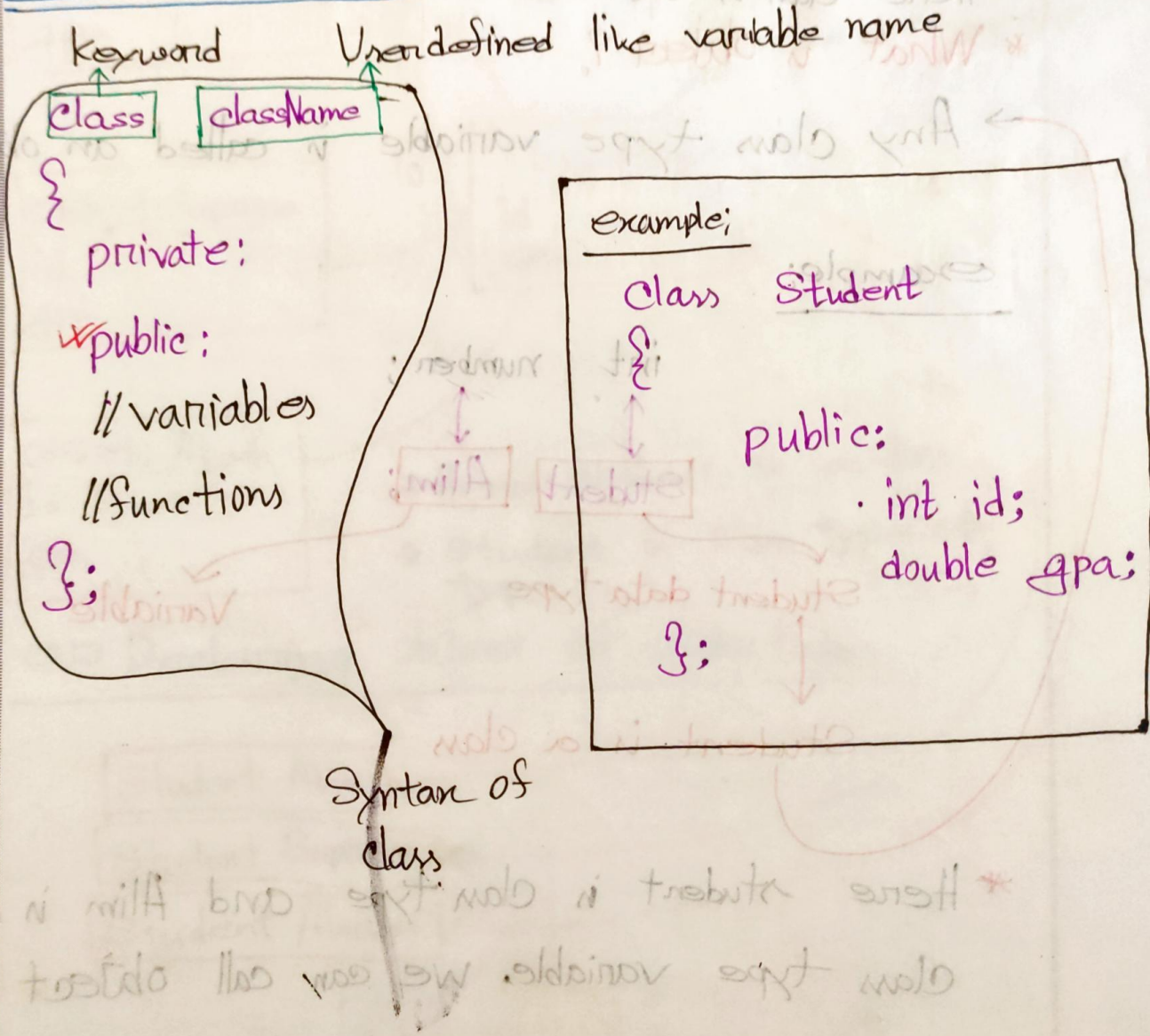
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

How to declare class and object ?

class declaration:



Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

object declaration

```
Class declaration

Class Student
{
  public:
    int id;
    double gpa;
};
```

Class type/data type

object declaration

className objectName;

like a variable

Syntax of object

example;

Student Ekramul;

Student Rifat;

Student Obaidulkader;

like

int number1;

int number2;

int number3;

Or

Student Ekramul, Rifat, kaoya;

like

int num1, num2, num3;

* Write like we write the variable with declaration.

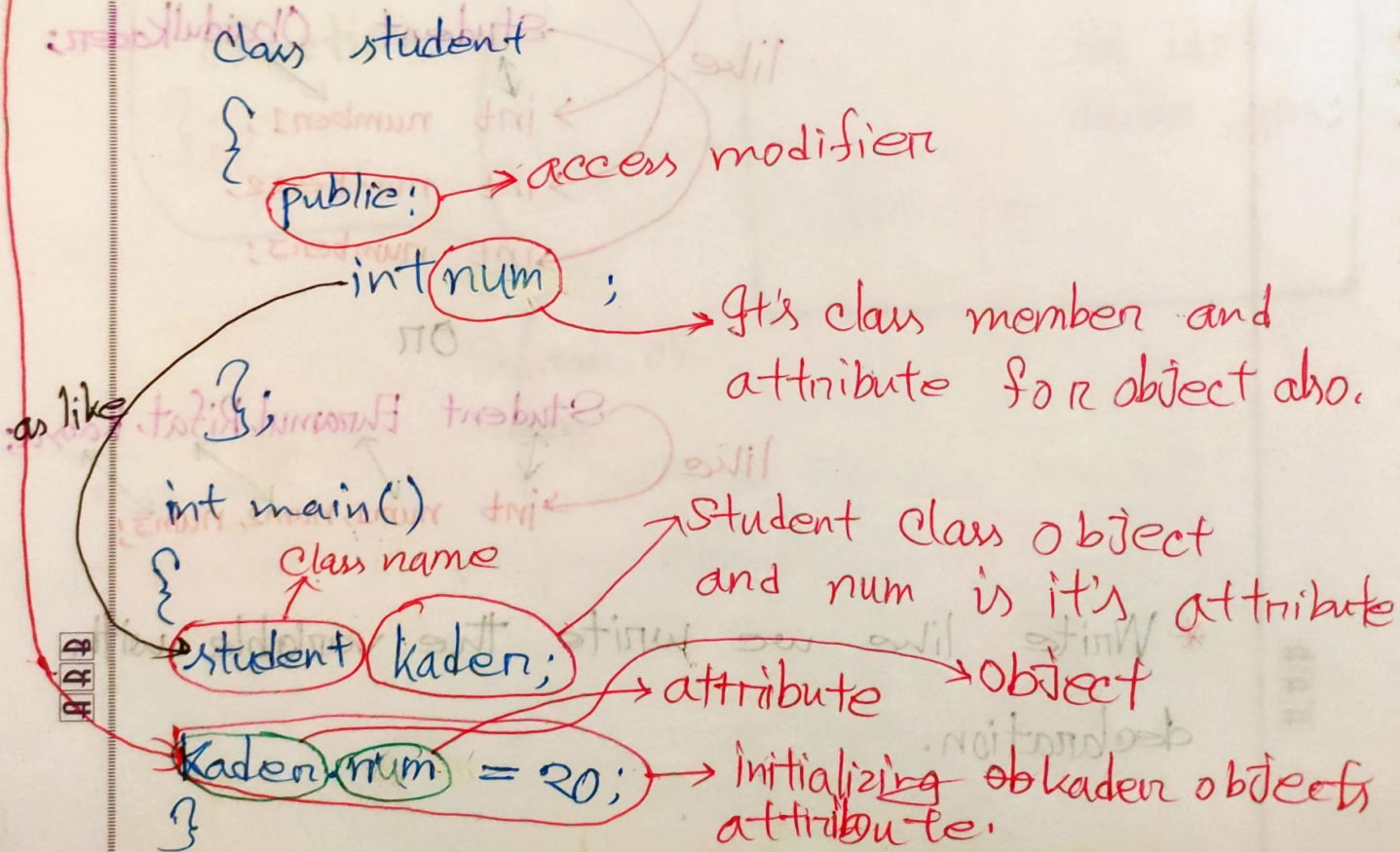
Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

Accessing object attributes:

* Class variables are attributes for each object declared in the main function body.

* To access the attributes for an object we must take the help of those objects.



Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Adding function to the class:

```
class student {
public:
    int id;
    float gpa;

    void display()
    {
        cout<< id <<gpa<< endl;
    }
};

int main()
{
    student alim;

    alim.id = 225;
    alim.gpa=5.00;

    alim.display();

    return 0;
}
```

student → declaring student as class type
class name.

will work for any object only which object is student class only

declaring object for student class

initializing attributes for alim

calling display function for alim

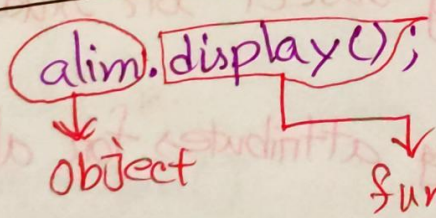
Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

* Function inside of a class, will ~~only~~ works for those classes objects only.
* Here in previous page display function will ~~only~~ works for student class types object, that means display will work for alim only. and if we want to use this display function which is inside the class. We must need to take help of object. For which object we are calling display.

example:



↓

Here we took the help of object alim, to call the display function. which is in inside the class.

Sub: _____

SAT ☐ SUN ☐ MON ☐ TUE ☐ WED ☐ THU ☐ FRI ☐

DATE: / /

Access modifier

Three types:

(i) public

(ii) private

(iii) protected

Constructor

What is a constructor?

→ Constructor is a special type of function that is used to initialize the objects attributes.

What are the properties of constructor?

(i) Constructor is a special type of function.

(ii) Constructor has the same name as that of the class it belongs. Syntax = classname() {//body}

(iii) It has no return type, not even void

(iv) It is called automatically

(v) Have to declare it inside the class body

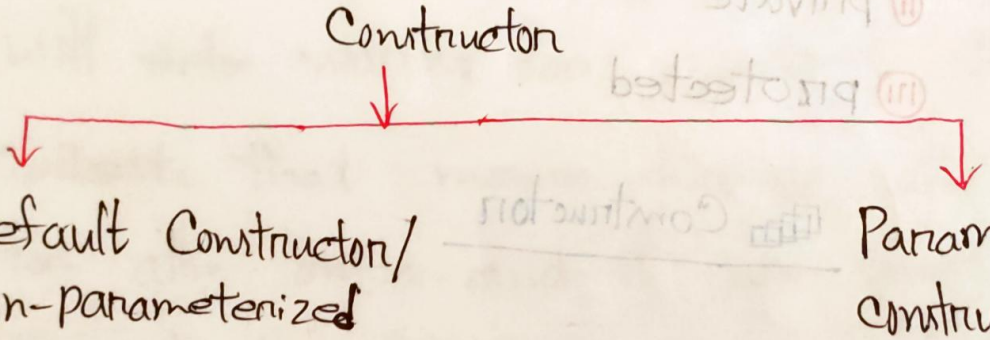
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Types of constructor

There are two types of constructor



Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Destructor

* Destructors are functions which are just the opposite of constructors. It destroys the objet whenever the object goes out of scope.

* It has the same name as that of the class, with a tilde(~) sign before it.

Syntax = ~classname()
{
//body
}

* A destructor gets automatically called when the object goes out of scope. A destructor is always non-parameterized. When it is called it destroys the object.

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

* Destructor are used to free the memory acquired by and objet during its scope (lifetime). So that the memory becomes available for further use.

Properties of destructor:

* It is automatically invoked when the objets are destroyed..
* It does not have any arguments.
* It has no return type not even void. As like constructor.
* It should be declared in the public section of the class.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

When is destructor called?

* A destructor function is called automatically.

When the object goes out of the scope.
It's called befor scope end (3).

(I) When the function ends

(II) When the program ends

(III) A block containing local variable ends

(IV) A delete operator is called

Less lifetime destroy first

Most lifetime destroy last

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Selection Operaton

```
class Selection                          → Selection Operaton
{ public:
string bhai_name;
void bol_bhai()
 {
   cout<<"bol_bhai is called"<<endl;
   ,,<<"Bhaika nam:"<<bhai_name<<endl;
 }
};
int main()
{
 Selection bhai;
 Selection *get=&bhai;//Putting bhai in get
 get=&bhai; //putting bhai object in get
 get->bhai_name="Bhaiyab"; //as like bhai.bhai_name
 get->bol_bhai(); //as like bhai.bol_bhai();

}
```

accessing

as like get=bhai

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Constant Variable

Const > Static

* In constant variable, variable value can't be change. And we can't change it it's fixed.

* Constant variable's value must be initialized during declaration. Otherwise, it is not possible to initialize it letter. Otherwise will show error without intialization.

Syntax = Const datatype variable=value

* If we want the constant variables value as input. We can take a input befor declaring a constant vari. and initialize this input value in constant value during declaration of constant variable.

```
Const int x = 10;
x = 20; //error
```

```
int m;
cin >> m;
Const int x = m;
```

```
Const int z;
z = 50; //error
```

```
Const int m;
cin >> m; //error
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

☐ Constant function

Syntax = returntype functionName() const

```
{
  //body
}
```

☐ Constant Object

Syntax = const className objectName;

* With Constant Object, cannot access non-Constant method / function
* Method must be constant for constant Objects
* With non-constant Object, we can access both (constant / nonconstant method) method.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Static keyword

① Static variable in a function

② " " " " Class

③ Class object as static

④ Static function in a class

⑤ " variable " " "

Same for All

① Static Variable in a Function

* Syntax = static data-type variable (= value) ;

optional, depend on user

* variable value will be updated always.

```
Static int x=1;
x=10;
x=20;
```

value will be updated here in start

```
void demo() { static int count=0;
              cout<<count; count++;}
int main() { demo()}
```

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

⑪ Static Variable in Class

* Can not be initialized inside class.
* Can not be first initialized from main function for any specific object.
* Can write it as static const datatype variable = value;

Then it will be considered as constant variable in class. Because const > static.

In this case this variable value can't be change, because it is constant now.

* In class static variable can be declared only, not initialized.
* To initialize a static variable which is inside the class, we need to use scope resolution operator.

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

* We have to initialize a static variable which inside class, from outside of class and outside of main function:-

};

initialization here => Syntax = variable's datatype Classname :: variable name = value;

must

int main()
{
}

optional

* Static variable value will be same for all object

* Can be accessable from any function after the first initialization / declaration from outside class Using any object and class name.

* Cannot use this system in method inside class.

* accessing static vari. value using object:

Syntax = object_name . static_variable_name ;

* accessing static vari. value using class Name:

Syntax = class_name :: static_variable_name ;

Value will be same for all object

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

local variable > other static

* can be ~~take as~~ initialize by taking input from main function.

Classname :: static_variable_name

easy / efficient way / standard / recommend

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

⑩ Class objects as static

Syntax = static className object_Name;

↓

* Static keyword befor object, it means the object is static.

* Static increase the lifetime of an object till the end of program.

```
int main()
{
int x=0;
if (x==0)
{
static OOP obj;
}
cout<< "End" <<endl;
}
```

lifetime

destructor will call here.

```
int main()
{
int x=0;
if (x==0)
{
OOP obj;
}
cout<< "End" <<endl;
}
```

lifetime ← Destructor will call here

lifetime less → Destroy first

More → Last

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

(IV) Static functions in a class

① If a function is static, then ~~et~~ variable used inside it must be static

```
static int x;
static int getx() { return x; }
```

② Same for everyone.

```
static void functionname()
{
  //body
}

int main()
{
  GfG Ob;
  Ob.functionname(); // not recommend. / will work.
  GfG::functionname(); // Standard recomonend
}
```

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

⑤ Static variable in class

* same value for all

```
Class stat
{
 public:
Static int x;
    int y;

};
int stat::x = 100; // Ok. x is static
//int stat::y = 20; //error; y is not static

int main()
{
stat obj;

obj.
cout<< obj.x; //not recommend
cout<<stat::x; // recommend
```

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

String Class

Header file → #include <string>

Nothing will happen if we don't use it. But using it is standard.

+ we can add two strings variable. and one variable with one string.

```
Str3= Str1 + Str2;
Str2= Str1 + " Haque";
```

```
Str1= "Ehramul";
Str2= "Rifat";
Str3= "";
```

= we can copy and replaced any string to another.

*using variable_name.size(); we can measure the length of a string.

```
int len = Str1.size();
```

here len value is 7

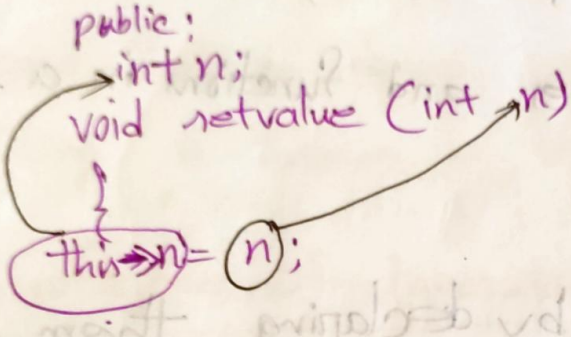
Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

this keyword

* this → used to specify / ~~the~~ indicated / accessing the class member.



* If we use only this. then it will show the location address only for those object.
* Access the currently executing object of a Class
* Access the data members of the correctly executing object
* Calling the member functions associated with the currently executing object
* To resolve the shadowing issue, when a local variable has a same name as an

Sub: _____ SAT SUN MON TUE WED THU FRI

DATE: / /

instance variable

Encapsulation

Encapsulation is a process of —

① Combining variables and functions in a single unit (class)

② Protecting data by declaring them as private.

* Private data will be hidden from other classes and they can only be accessed through public function of their current class. That is known as data hiding.

* Accessing private properties via public setter and getter method.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Inheritance

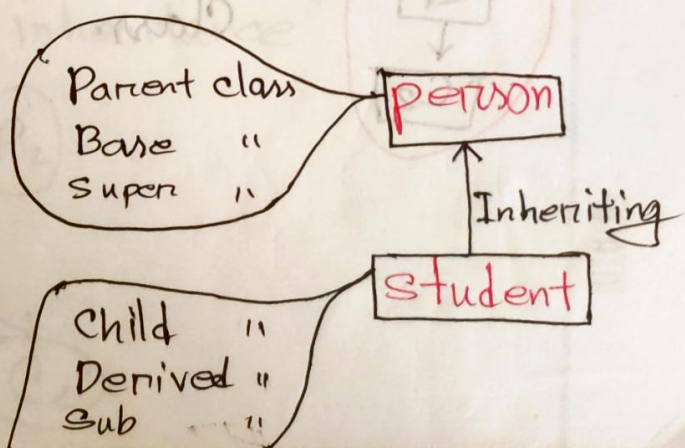
* Inheritance is a relationship.

- The process of obtaining the data members and functions from one class to another class is known as inheritance.

Importance of Inheritance:

- Code reusability
- Application development time is less
- Application takes less memory
- Application execution time is less

```
class student : public person
{
  ---
  ---
  ---
};
```



Sub: _____ SAT SUN MON TUE WED THU FRI ☐ ☐ ☐ ☐ ☐ ☐ ☐

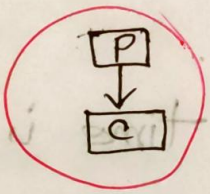
DATE: / /

Syntax:

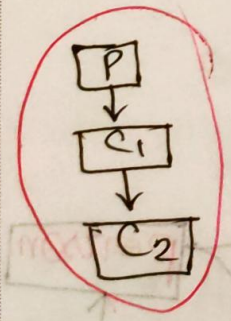
class ChildClass_Name : Access_modifier Parent_class_Name

Types of Inheritance

Single Inheritance: A child class with only one parent class.



Multilevel Inheritance: A child class with one parent class, and that parent class is a child class of another class.

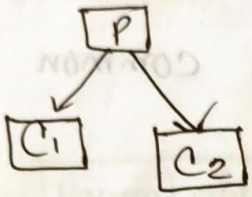


Sub:

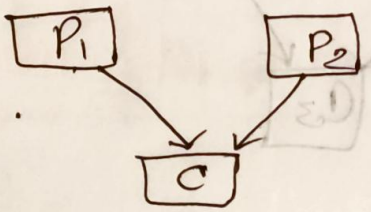
SAT SUN MON TUE WED THU FRI

DATE: / /

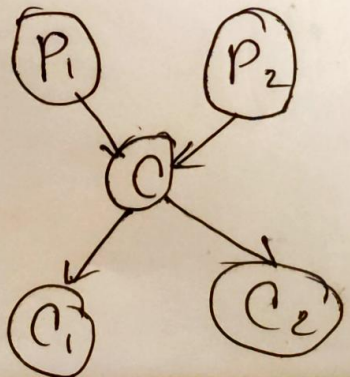
Hierarchical Inheritance: Multiple child classes with same parent class.



Multiple Inheritance: A child class with multiple ~~base~~ parent class ~~is called~~.



Hybrid Inheritance: Combination of multiple and hierarchical inheritance

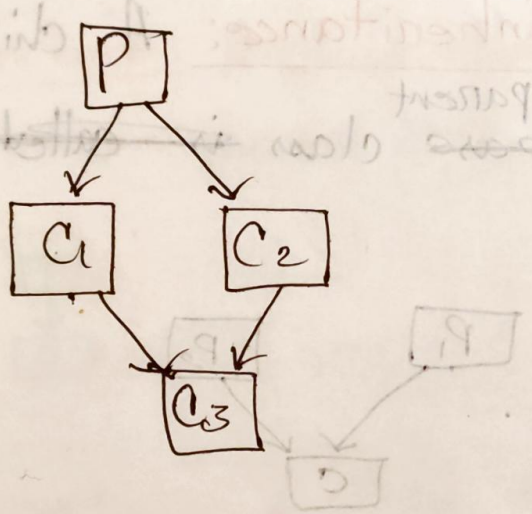


Sub: _____

SAT SUN MON TUE WED THU FRI

DATE: / /

Multipath Inheritance: A ~~derived~~ Child class with two parent class and ~~those~~ two parent class have one common ~~base~~ Parent class.



Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Modes of Inheritance

~~Public mode:~~

Parent class member access specifier	Type of Inheritence		
	public	protected	private
Public	public	protected	Private
Protected	protected	"	"
private	Hidden	Hidden	Hidden

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

Function Overloading

* Function overloading is a feature of object oriented programming. Where two or more function can have the same name but different parameters.

* When a function name is overloaded with different jobs it is called function overloading.

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Function Overriding

* It is the redefinition of base class function in it's derived class with same signature. (return type and parameters).

* In function overriding

function in Parent class → Overriden function

function in child class → Overriding function

Difference between Overloading and Overriding

Overloading	Overriding
(i) Parameter must be different	(i) Parameter must be same
(ii) It occurs within the same class.	(ii) It occurs between two classes, sub class and super class.
(iii) Inheritance isn't involved	(iii) Inheritance is involved

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

Overloading	Override
(iv) One method must not hide another	(iv) Child method hides parent method
(v) Return type may or may not be same	(v) Return type must be same

Operator Overloading

* Most of the built in operators in c++, can be redefinable or overloadable.
* Thus, a programmer can use operators with user defined types as well.
* Overloaded operators are functions with special name the keyword "operator" followed by the symbol for the operator being defined, like any other function.

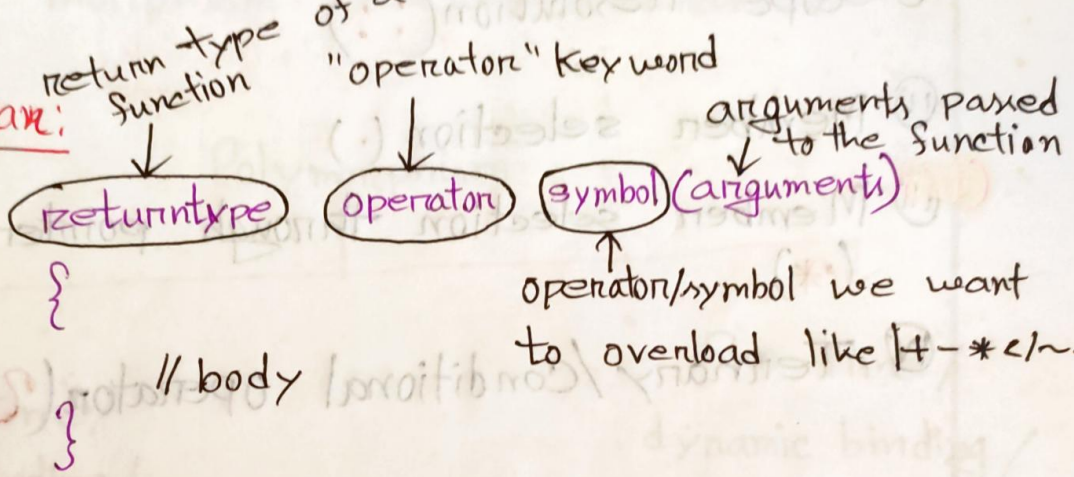
Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

* An overloaded operator has a return type and a parameter list

Syntax:



Overloadable Operators

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	( )
->	->*	new	new[]	delete	delete[]

Sub: _____

SAT SUN MON TUE WED THU FRI
☐ ☐ ☐ ☐ ☐ ☐ ☐

DATE: / /

Non-Overloaded operators:

① Scope resolution (::)

② Member selection (.)

③ Member selection through pointer to function (.*)

④ Ternary / Conditional operator (?:)

Sub:

SAT SUN MON TUE WED THU FRI

DATE: / /

Polymorphism

Poly + morphism → 2 Greek words

Poly → many

morphism → forms

Polymorphism

- Compile Time / Static binding / early binding
  - Function / Operator Overloading
- Run Time / dynamic binding / late binding
  - Function Overriding