2021-10-03 -- Scratchpad of CSE213 (Sec-1)
==========

Method chaining:
==========
- It is a mechanism, where we can chain calls of methods (function) in a sequence.
- We can chain calles of the same method
- We can also chain mixed calls of different methods
- We can even chani calls of methods and global functions, too

```
class Person{

        void buyFoodFromMarket(){
                cout<<"Food is bought."<<endl;
        }
        void organizeLivingRoom(){
                cout<<"Living room is organized."<<endl;
        }

        void serveFood(){
                cout<<"Food is served."<<endl;
        }

};

int main(){
        Person babu;
        babu.buyFoodFromMarket();
        babu.organizeLivingRoom();
        babu.serveFood();
        return 0;
}
```

"this" pointer:

```
int* ptr;       //ptr is an EXPLICIT pointer store address of an int
Matrix* mPtr;   //mPtr is an EXPLICIT pointer store address of an
Matrix object
```

Explicit pointer does not point to any address automatically, unless we assign
an address to the pointer
Example: ptr = &x; OR ptr = new int[10];

"this" is an IMPLICIT pointer, which we can't declare and ALSO can't assign
an address to it. "this" pointer exists for each class we define, and it ALWAYS
automatically store the address (points to) of the clint-object during the
execution of a method call. When no method of that class is being excuted,
"this" pointer will be NULL.

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

class Person{
    int tipsEarned=0;
    public:
        //void buyFoodFromMarket(){
        Person& buyFoodFromMarket(){
                cout<<"Food is bought."<<endl;
                //we need to return the client object itself
                tipsEarned += 50;
                return *this;
        }
        Person& organizeLivingRoom(){
                cout<<"Living room is organized."<<endl;
                tipsEarned += 50;
                return *this;
        }
        //void serveFood(){
        Person& serveFood(){
                cout<<"Food is served."<<endl;
                tipsEarned += 50;
                return *this;
        }
    void showTipsAmount(){
        cout<<"Total tips earned = "<<tipsEarned<<" taka"<<endl;
    }
};
int main(){
        Person babu, selim;
        //babu.buyFoodFromMarket();     //babu.organizeLivingRoom();
//babu.serveFood();
        babu.buyFoodFromMarket().organizeLivingRoom().serveFood();
        babu.showTipsAmount();
        return 0;
}
```

==========

operator overloading:
==========

```
int x;
cin is a class variable, a.k.a object of istream class

cin>>x;

cin.fieldName;
cin.methodName();

z = x+y;
```

Original form:

>> right shift operator

```
<< left-shift operator
int x = 35;     int: 4-bytes / 32-bits
binary of x:    00000000 00000000 00000000 00100011
y = x<<2;
00 <-- 000000 00000000 00000000 00100011 00<--
00 <-- 00000000 00000000 00000000 10001100<--
cout<<y;        //140

cout<<y;        //
"<<" operator is assigned additional task to read the content of RHS
operand
from memory and send it to the default console output device (screen)
using ostream object cout (LHS operand of <<)

Process of assigning additional task to an operator is called operator
overloading

class Matrix{
        int row, col;
        int **valPtr;

        operator+

};

int main(){
        x = 10 + 20;
        x = y + 3.5;

        Matrix m1,m2,m3;
        m3 = m1 + m2;   //logical, so we need to overload "+" for
Matrix class
        Student s1, s2,s3;
        //s3 = s1 + s2; //illogical, so we will NOT overload "+" for
Student
        float avgCgpa = (s1 + s2 + s3)/3; //logical
}

result = x  + y         + z
        int + int       + z
           int          + int

float avgCgpa = (s1 + s2 + s3)/3; //logical
                object + object + object
                     float      + object

                Here, "+" need to be overloaded twice:
                version-1 for : object + object producing float, for
Student class
                version-2 for : float + object producing float, for
Student class

m4 = m1 + m2 + m3;      //logical, so we need to overload "+" for
Matrix class
     object + object  + object
          object      + object
                Here, "+" need to be overloaded once:
```

```
                version-1 for : object + object producing object, for
Matrix class
-------------------------------------------------------------------
------------
Note:
=====
a)To overload a method for a class, we need to introduce a special
method
called operator method. operator method name begins with keyword
"operator"
followed by the operator symbol.

b) For unary operator, only one object is involved, and that is the
client.
   For binary operator, ALWAYS the first operand will be the cliend

3+4i
class ComplexNo{
        int real,img;
        public:
        void setComplexNo(){
                cout<<"Enter real value: "; cin>>real;
                cout<<"Enter imaginary value: "; cin>>img;
        }
        void showComplexNo(){
                cout<<real;
                if(img>=0) cout<<"+";
                cout<<img<<"i"<<endl;

                cout<<real<<(img>=0?"+":"")<<img<<"i"<<endl;
        }

         ComplexNo add(ComplexNo c){
                ComplexNo temp;
                temp.real = real + c.real;
                temp.img = img + c.img;
                return temp;
        }

         ComplexNo operator+(ComplexNo c){
                ComplexNo temp;
                temp.real = real + c.real;
                temp.img = img + c.img;
                return temp;
        }

};

int main(){
        ComplexNo c1, c2, c3, c4;
        c1.setComplexNo();      c2.setComplexNo();
c3.setComplexNo();
        c4 = c1.add(c2);                //c1+c2;
        c5 = c4 + c3;                   //c5 = c4.operaotr+(c3);
        cout<<"Complex no c1 = "; c1.showComplexNo();
        cout<<"Complex no c2 = "; c2.showComplexNo();
        cout<<"Complex no c3 = "; c3.showComplexNo();
```

```
        cout<<"Complex no c4 (c1+c2) = "; c4.showComplexNo();
        cout<<"Complex no c5 (c4+c3) = "; c5.showComplexNo();
}
-------------------
Unary operator: 1-operand:      i++
                                ...
Binary operator: 2-operands:    a+b
                                ....

Ternary operator: 3-operands:   operand-1 ? operand-2 : operand-3
                                condition ? true-Val  : false-Val
-------------------------
Q: Did you use operator overloading in your previous courses ?
A: Yes... can you tell where?
        int x=10;
        cin>>x;         //cin.operator>>(x);
        cout<<x;        //cout.operator<<(x);
                1st operand: cout (object of ostream class)
                2nd operand: x (int)

class ostream{

        public:
        void operator<<(int val){...}

};

class istream{

        public:
        void operator>>(int& val){...}

};

Note:
-----
        - return type of operator<< in ostream class could be void
          and it would be fine as long as we are happy to use it
          as a single call.
        - However, if we want to chain multiple calls, then return
type
          should be non-void.



class ostream{

        public:
        ostream& operator<<(int val){...}
        ostream& operator<<(float val){...}

};

class istream{

        public:
        istream& operator>>(int& val){...}
```

```
};


int main(){
        int x=10, y=20; float f;
        cout<<x<<y;     // cout.operator<<(x).operator<<(y);

        cin>>x>>y;      // cin.operator>>(x).operator>>(y);

        cout<<x<<f;     // cout.operator<<(x).operator<<(f);
        12      13
        int z;
        cin>>z; //      '1'             '2'             '3'
        Ascii           48              49              50
                        00110000        00110001        00110010


                        binary of 123: 64 + 32 + 16 + 8 + 4 + 1
                                        00000000 00000000 00000000
01111101
        return 0;
}
```