2021-09-28 -- Scratchpad of CSE213 (Sec-1)
==========

function and method overloading:
==========

- if we have multiple definitions of a function, then when a call is encountered, it is
important for thr compiler to be able to decide which definition to execute for that
specific call (binding definition to a call).
- To make this happen, the signature of ALL definitions must be unique
- signature is function with parameter-list without considering return type

```
int getSum(int a, int b){       //version-1
        return a+b;
}

int getSum(int a[], int sz){    //version-2
        int i,sum=0;
        for(i=0;i<sz;i++) sum += a[i];
        return sum;
}

int main(){
        int p=10, q=20;, result;
        int arr[] = {11,23,44,1,56};
        result = getSum(p,q);
        result = getSum(arr,5);
        return 0;
}
```

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

struct Employee{
        int empId; string empName; float salary; //member data, a.k.a field

        void setEmpInfo(){      //member function, a.k.a method
                cout<<"Enter employee ID: "; cin>>empId; cin.ignore();
                cout<<"Enter employee Name: "; getline(cin, empName);
                cout<<"Enter employee Salary: "; cin>>salary;
        }
        void showEmpInfo(){
                cout<<"Id="<<empId<<", Name="<<empName<<", Salary="
<<salary<<endl;
        }
};

int main(){
        Employee farid, luna; int x; cin>>x;
        //cout<<"Enter employee ID: "; cin>>farid.empId; cin.ignore();
        //cout<<"Enter employee Name: "; getline(cin, farid.empName);
```

```
        //cout<<"Enter employee Salary: "; cin>>farid.salary;
        farid.setEmpInfo();     //      luna.setEmpInfo();
        //cout<<"Id="<<farid.empId<<", Name="<<farid.empName<<",
Salary="<<farid.salary<<endl;
        farid.showEmpInfo();
        return 0;
}
```

Note: By default, ALL members of a struct are public

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

class Student{

        int studId; string studName; float cgpa;
        string grades[50];
        int credits[50];

        void setStudInfo(){
                cout<<"Enter student ID: "; cin>>studId; cin.ignore();
                cout<<"Enter student Name: "; getline(cin, studName);
                //cout<<"Enter cgpa: "; cin>>cgpa;
        }
        void showStudInfo(){
                cout<<"Id="<<studId<<", Name="<<studName<<", Cgpa="
<<cgpa<<endl;
        }
        void calcAndSetCgpa(){
                ....
                cgpa = ....;
        }
};

int main(){
        Student asif, luna;
        //cout<<"Enter student ID: "; cin>>asif.studId; cin.ignore();
        //cout<<"Enter student Name: "; getline(cin, asif.studName);
        //cout<<"Enter cgpa: "; cin>>asif.cgpa;
        //cout<<"Id="<<asif.studId<<", Name="<<asif.studName<<",
Cgpa="<<asif.cgpa<<endl;
        asif.setStudInfo();
        asif.showStudInfo();
        return 0;
}
```

Note: By default, ALL members of a class are private

==========

==========

e) 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 nned to overload "+" for Matrix class
```

```
        Student s1, s2,s3;
        //s3 = s1 + s2; //illogical
}
```