2021-09-26 -- Scratchpad of CSE213 (Sec-1)
==========

Overhauling Required concepts in C++:
==========

a) Reviewing "function"

- Jargon:
  - function call
  - function prototype
  - function definition

- pass-by-value
  ```
  int doSome(int x){....}
  ```

- pass-by-reference
  ```
  int doSome(int& x){....}
  ```

- pass-by-address
  ```
  int doSome(int* ptr){....}
  ```

- return-by-value
  ```
  int doSome(){....}
  ```

- return-by-reference
  ```
  int& doSome(){....}
  ```

- function vs method

In this course:
"function" --> Global function
"method" --> Member function of struct/class

b) Difference between struct and class

c) function and method chaining

d) function and method overloading

e) operator overloading

==========

```
int     int     int     int     --> array can be an option

int     string  float           --> struct/class
id      name    cgpa
```

- if your purpose is merely to form a heterogeneois collection, they you can use either struct/class

- But, if you want to exploit full potential of Object Orientation,

class is the defacto choice

==========

a) Reviewing "function"

- Jargon:
  - function call
    - function will be called by its name and TYPICALLY the call terminates with semi-colon
    - parametr will be passed (if any) without type

```
int main(){
        int x, y=20; cin>>x; //10
        doSome(x);      //function call
        return 0;
}
```

  - function prototype
    - It establishes the identity of the function
    - Also ends with semi-colon
    - Parameter names are optional

```
returnType fnName(paraType1, paraType2, paraType3);

void
non-void

int main(){
        int x, y=20; cin>>x; //10
        doSome(x);      //function call
        return 0;
}
```

  - function definition
    - Its the body of the function
    - A block "{ }" of code is associated with the definition

```
returnType fnName(paraType1 paraName1, paraType2 paraName2, paraType3 paraName3){

}
```

    - pass-by-value

```
int doSome(int x){....}
```

    - pass-by-reference

```
int doSome(int& x){....}
```

    - pass-by-address

```
int doSome(int* ptr){....}
```

    - return-by-value

```
int doSome(){....}
```

    - return-by-reference

```
int& doSome(){....}
```

```
        - function vs method

        In this course:
                "function"      --> Global function
                "method"        --> Member function of struct/class

b) Difference between struct and class

c) function and method chaining

d) function and method overloading

e) operator overloading

=============================================================================
==================

1) retutn type void, with zero parameter
        Ex:


2) retutn type void, with one parameter
        void printMessage(String msg){  //definition
                cout<<msg<<endl;
        }

        int main(){
                printMessage("Success");
                doSome();       //doSome is called, and main is the
caller
                return 0;
        }

3) retutn type void, with >1 parameters
        int main(){
                int arr[] = {11,23,44,1,56};
                sort(arr,5);
                return 0;
        }

4) retutn type nonVoid, with zero parameter


5) retutn type nonVoid, with one parameter


6) retutn type nonVoid, with >1 parameters


        int globalArr[10];

        void doSome(){.... int w;...}
        int getSum(int a, int b){
                doSome();
                return a+b;
```

```
}
int getSumOfArray(int a[], int sz){
        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 = getSumOfArray(arr,5);
        return 0;
}
```

---

d) 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;
}
```