main contributing authors to this document:
Mu-Chu Lee, Shu-Hung You

This is just an easy introduction about overloading operators in C++. If you're familiar with C++ already, this might not be helpful for you, you can just start to do your HW1 now :)

However, if you're not that familiar with operator overloading, you might find this note helpful! Making good use of operator overloading will make your code clean and clear :)

We now start with an easy example, which keeps track of the score and the name of student's exam result.

```
class students{
    public:
        int score;
        char name[20];
};
```

In this class, there are two students. Bob and Charlie.

```
students Bob, Charlie;
strcpy(Bob.name, "Bob");
strcpy(Charlie.name, "Charlie");
```

After the exam, Bob scored 59 in the test and Charlie got 100, so we have:

```
Bob.score = 59;
Charlie.score = 100;
```

Now, we want to see whether Bob has a higher score or Charlie has a higher score. We get:

```
if(Bob.score < Charlie.score)
    cout << "Charlie!!!!!";
else
    cout << "Bob_:((((";
```

This is somehow annoying, since we just want to compare Bob and Charlie. We hope we could do something like:

```
if(Bob < Charlie)
```

However, when we try to do this, the compiler tells us "this is something undefined". Since the "<" operator doesn't mean anything to `class students`. Therefore, we'll need to *tell* what we hope to do. So we add the followings into our `class`.

```
class students{
    public:
        int score;
        char name[20];

        bool operator <(const students& st) const {
            return score < st.score;
        }
};
```

Voilà! Now we can just compare `class student` directly!

However, the same annoying thing happens when we want to say "Bob scored 59 on the test and Charlie scored 100".

```
Bob = 59;
Charlie = 100;
```

This won't work. Why? The same reason as above. The "=" operator doesn't mean anything to `class students`. Therefore, we'll need to *tell* what we hope to do again. So we add the followings into our `class`.

```
class students{
    public:
        int score;
        char name[20];

        bool operator <(const students& st) const {
            return score < st.score;
        }
        student& operator =(int num){
            (*this).score = num;
            return *this;
        }
};
```

Now, when we use "=" with an integer, `class students` would know what to do.

But another problem arises. What if I what to do "Bob's name is Bob"? This might sound weird, but when we declared a `students Bob;`, the *Bob.name* is still undefined. So we might want to say "Bob's name is Bob".

You might think the operator "=" occupied, but actually you can still use it.

```
class students{
    public:
        int score;
        char name[20];

        bool operator <(const students& st) const {
            return score < st.score;
        }
        student& operator =(char* str){
            strcpy((*this).name, str);
            return *this;
        }
        student& operator =(int num){
            (*this).score = num;
            return *this;
        }
};
```

From *char* str* and *int num*, we can easily recognize what "=" means. Therefore, we can now use:

```
students Bob, Charlie;
Bob = "Bob";
Charlie = "Charlie";
Bob = 59;
Charlie = 100;
```

Finally, we want to add up the score of the whole class, and we hope we can keep it in a `class students`. We'll have:

```
students Bob, Charlie, DSA_Class;
Bob = "Bob";
Charlie = "Charlie";
DSA_Class = "DSA_Class";
Bob = 59;
Charlie = 100;
```

```
DSA_Class = Bob.score+Charlie.score;
```

I believe you'll find **DSA_Class = Bob.score+Charlie.score** weird.

Why can't we just do **DSA_Class = Bob+Charlie**? The answer lies above. Your compiler doesn't know what `class students + class students` means. We'll need to tell what it means.

```
class students{
    public:
        int score;
        char name[20];

        bool operator <(const students& st) const {
            return score < st.score;
        }
        student& operator =(char* str){
            strcpy((*this).name, str);
            return *this;
        }
        student& operator =(int num){
            (*this).score = num;
            return *this;
        }
        int operator +(const students& st) const {
            return score+st.score;
        }
};
```

Now, we see when we do the "+" operation on `class students`, we have an integer returned!

There are still a lot of things that operator overloading can do(for example, we may feed multi-variables). Also, we can also overload operators by using function signature. There are a lot of knowledge that aren't mentioned in this note(same as courses you take in school). So if you want to learn more, you should be an active learner and seek for resources to strengthen yourself (No matter going to the TA hours or looking for information on the internet) :)