Subscript Operator

The subscript operator must be a member function.

To be compatible with the ordinary meaning of subscript, the subscript operator usually returns a reference to the element that is fetched. Also, subscript can be used on either side of an assignment.

```
int arr[]={1,2,3};
arr[0];// arr[0] return a reference to the first element in the arr.
int b=arr[0]; // subscript is used on the right-hand side
arr[0]=3;// subscript is used on the left-hand side
```

Consequently, it is also usually a good idea to define both `const` and `nonconst` versions of this operator. When applied to a `const` object, subscript should return a reference to `const` so that it is not possible to assign to the returned object.

An example:

```
class StrVec{
    public:
        // run on nonconst objects of StrVec
        string & operator[](std::size_t n){
            return elements[n];
        }

        // run on any kinds of StrVec, but it is the best to run on const
        // object of StrVec
        const string &operator[](std::size_t n)const{
            return elements[n];
        }
        // other members
    private:
        string *elements;// pointer to the first element in the array
};
```

We can use these operators similarly to how we subscript a `vector` or array. Because subscript returns a reference to an element, if the `StrVec` is `nonconst`, we can assign to that element; if we subscript a `const` object, we can't:

```
// assume sves is a StrVec
const StrVec cvec=svec;// copy elements from svec into cvec

// if svec has any elements, run the string empty function on the first one
if(svec.size()&&svec[0].empty()){
    svec[0]="zero";// ok, subscripting returns a reference to a string
    cvec[0]="zip";// error, subscripting cvec returns a reference to const
}
```