member functions defined inside class are implicitly inline.

inline function definition must be put in header file:

The compiler need to see the definition of the inline function whenever it finds any use of that inline function. That is typically possible if the inline function is placed in a header file.

if function is defined outside class, then `inline` keyword is only needed in definition.

---

functions of template class: member function, friend function(forward declaration)

member functions of template class defined outside class: (**template is still needed**)

```
template<int N>
inline Complex& Matrix<N>::operator()(int i, int j)
{
xxx
```

For friend function:

```
template<int N> class Matrix;
template<int N> Matrix<N> operator+(const Matrix<N> &m1, const Matrix<N>

template<int N>
class Matrix{
    Complex m[N][N];
public:
  friend Matrix<N> operator+<>(const Matrix<N> &m1, const Matrix<N> &m2)
};

template<int N>
inline Matrix<N> operator+(const Matrix<N> &m1, const Matrix<N> &m2)
{
  Matrix<N> ret;
  for(int i=0; i<N; ++i)
    for(int j=0; j<N; ++j)
      ret[i][j] = m1[i][j] + m2[i][j];
  return ret;
}
```

1. operator**+**<>: The **<> after the name of the friend function is needed to make clear that the friend is a template function. Without the <>, a non-template function would be assumed.**(Compiler will think operator**+** is a

   non-template function)
2. the name `operator+` refers to a function template, which should be declared (as template) in advance.

p.s.

```
template<int N>
class Matrix{
    Complex m[N][N];
public:
  template<int NN>
  friend Matrix<NN> operator+<>(const Matrix<NN> &m1, const Matrix<NN> &
};
```

this will make a lot of friend. operator+<4> is also a friend of Matrix<3>.

---

return type

return type: (zyd: almost never return const reference)

zyd: principle: **return a temporary object, then return by value(and define move operation); if want to use return as lvalue, then return by reference**(e.g. operator=, operator+=, operator(), operator[])

1. Returning a pointer or a reference to a newly created object (temporary object) is usually a very bad idea. **Return by value, and define move operations to make such transfer efficient.**
   (p.s. return const-reference is allowed. The lifetime of a temporary is extended via a ref-to-const. But is not recommended. Do not know why. Seems like it may result in memory management problems)

```
Matrix operator+(const Matrix& a, const Matrix& b) // return-by-value
```

1. When function returns one of their argument objects (or *this for a class), return a reference.

```
Matrix& Matrix::operator+=(const Matrix& a);
Matrix& Matrix::operator=(const Matrix& a) {return *this;} // (a=b)=c is allowed
Matrix& Matrix::operator()(int i) const;
```

p.s. unlike most operators, assignment "=" is right-associative. i.e. a=b=c is equivalent to a=(b=c)

p.s. associativity: when two operators have the same precedence, how to calculate. e.g. left-associative: a+b-c is equivalent to (a+b)-c; a=b=c is equivalent to a=(b=c)

---

const member function

define member function const if 1 or 1&&2 are true:

1. the member function doesn't change the object
2. want temporary object to call that method

---

operator[] only takes one argument. Use operator()