```
if (!over_zero)
{ for (int i = 0; i < 20; ++i)
  {
    intpart[i] = int_part % 10 + '0';
    int_part /= 10;
    if (int_part == 0)
    { int_len = i + 1;
      break;}
    }
    int demical_len = digit - int_len;
    for (int i = 0; i < demical_len + 1; ++i)
    {
    demical_part *= 10;
    demicalpart[i] = int(demical_part) + '0';
demical_part= demical_part - int(demical_part);
    }
    result[0] = '-';
    for (int i = 0; i < int_len; ++i)
    {
    result[int_len - i] = intpart[i];
    }
    result[int_len + 1] = '.';
    for (int i = 0; i < demical_len; ++i)
    {
    result[int_len + i + 2] = demicalpart[i];
    }
    result[int_len + demical_len + 3] = '\0';
}

void main()
{
	char str1[20];

	cout << "input string" << endl;
	cin.getline(str1,20);
	cout << "number:" << endl;
	cout << atof(str1) << endl;

	double num2;
	int digit;
	char *str2[50];
	cout << "input number and digit" <<
endl;
	cin >> num2 >> digit;
	cout << "string:" << endl;
	*str2 = dtoa(num2, digit);
	cout << *str2 << endl;
	delete[] * str2;

	system("pause");
}
```

类模板：

```
template<类型参数表>
class 类模板名
{成员变量和成员函数};
```

在类模板外编写：

```
template<类型参数表>
返回值类型  类模板名<类型参数名列表>::成员函数名(参数表)
template<class T1, class T2>
bool pair<T1, T2>::operator<(const Pair<T1,T2>&p const)
template<class T>
CArray<T>::CArray(CArray &a)
void CArray<T>::pushback(const T&v)
```

派生：

```
template<class T1,class T2>
class B:public A<T2,T1>
```

静态成员：

```
static int count;
```

(类外声明)template<> int A<int>::count = 0;

```
template<>int A<double>::count = 0;
```

```
//double  转为 string
char* dtoa(double num, int digit)
{
    bool over_zero = (num >= 0);
    num = fabs(num);
    int int_len = 0;
    char intpart[20];
    char demicalpart[20];
    int int_part = int(num);
    double demical_part = num - int(num);
    char *result = new char[digit + 4];
    if (over_zero)
  {
    for (int i = 0; i < 20; ++i)
    {
      intpart[i] = int_part % 10 + '0';
      int_part /= 10;
      if (int_part == 0)
      {
        int_len = i + 1;
        break;
        }
      }
    //小数部分
    int demical_len = digit - int_len;
    for (int i = 0; i < demical_len + 1; ++i)
    {
    demical_part *= 10;
    demicalpart[i] = int(demical_part) + '0';
    demical_part = demical_part - int(demical_part);
    }
    for (int i = 0; i < int_len; ++i)
    {
    result[int_len - 1 - i] = intpart[i];
    }
    result[int_len] = '.';
    for (int i = 0; i < demical_len; ++i)
    {
      result[int_len + i + 1] = demicalpart[i];
    }
    result[int_len + demical_len + 2] = '\0';
  }
  return result;
  delete[]intpart;
  delete[] demicalpart;
  }
```

堆栈 模板(include string)：

```
class stack
{
private:
T str[max_size];
T topnum;
public:
	void init();
	void push(T a);
	int empty();
	void pop();
	T top();
};
template <class T>
void stack<T>::init()
{topnum = -1;}
template <class T>
int stack<T>::empty()
{if (topnum <= -1) return 1; //1 为空
else return 0; //0 为不空}
template <class T>
T stack<T>::top()
{return str[topnum];}
template <class T>
void stack<T>::pop()
{    --topnum;    }
template <class T>
void stack<T>::push(T a)
{    if (topnum > max_size) exit(1);
if (topnum == max_size) return;
str[++topnum] = a;    }
```

十进制转八进制：

```
cin >> n;
while (n != 0)
{	stack.Push(n % 8);
	n /= 8;
}
int i = stack.Stackempty();
while (i == 0)
{	cout << stack.Top();
	stack.Pop();
	i = stack.Stackempty();
}
stack.~CStack();
```

```
class CHuman
{
protected:
	string name;
	char gender;
	int age;
public:
	void SetInfo(const string &name_, char gender, int age);
	void Print();
};
void CHuman::SetInfo(const string &name_, char gender_, int age_)
{
	name = name_;
	gender = gender_;
	age = age_;
}
class CStudent:public CHuman
{
protected:
	string id;
public:
void SetInfo(const string &name_, char gender_, int age_, const string &id_)
{   CHuman::SetInfo(name_, gender_, age_);
    id = id_;    }
void Print()
{   CHuman::Print();
    cout << "id: " <<id<< endl;    }
};
```

构造函数：

Complex(double r)

Complex(double r, double i)

Complex(Complex c1, Complex C2)

复制构造函数（注意循环时量是否加上了）：

Complex(const Complex &c){};

析构函数：~CDemo();

静态成员变量（=全局变量）：

private:static int totalArea;

public:static void PrintTotal(); //CRectangle::PrintTotal();

类外声明：int CRectangle::totalArea = 0;

常量成员函数：

int GetValue() const{return n}; //const Ctest objTest1

int GetValue(){return 2*n}; //Ctest objTest2

初始化列表:(成员对象也用复制构造函数初始化)

CTyre(int r,int w):radius(r),width(w);

CCar(int p,int tr, int tw):price(p),tyre(tr, tw);

辗转相除(计算两个整数 a,b 的最大公约数):

```
int Gcd_2(int a,int b)
{if (a<=0 || b<=0)return 0;
int temp;
while (b > 0) {temp = a % b;
a = b;      b = temp;}
return a;}
```

牛顿迭代:

x(n+1) = x(n) - f(xn)/f'(xn)

斐波那契数列

递归:

时间 O(2^N)递归次数*每次递归中执行基本操作的次数

空间:O(N) 递归的深度*每次递归所需的辅助空间的个数

```
int fib2(int n)
{if(n == 0)    return 0;
  if(n == 1)    return 1;
  return fib2(n-1)+fib2(n-2); }
```

非递归:

时间:时间复杂度为 O(n)

空间:O(1)

```
int fib(int n)
{ int result[2] = {0,1};
if(n < 2)    return result[n];
int fibOne = 0; int fibTwo = 1;
int fibN = 0; int i = 0;
for(i = 2; i <= n; i++)
{ fibN = fibOne + fibTwo;
  fibOne = fibTwo;    fibTwo = fibN; }
  return fibN; }
```

二分法插入排序:

在插入第 i 个元素时,对前面的 0～i-1 元素进行折半,先跟他们中间的那个元素比,如果小,则对前半再进行折半,否则对后半进行折半,直到 left<right,然后再把第 i 个元素前 1 位与目标位置之间的所有元素后移,再把第 i 个元素放在目标位置上。

最好的情况是当插入的位置刚好是二分位置 所用时间为 O(log₂n);

最坏的情况是当插入的位置不在二分位置 所需比较次数为 O(n),无限逼近线性查找的复杂度。

平均时间 O(n^2) 稳定 空间复杂度 O(1)

时间复杂度

1.适合记录数较多的场景,与直接插入排序相比,在寻找插入位置上面所花的时间大大减少,但是折半插入排序在记录移动次数方面和直接插入排序是一样的

2.记录比较次数与初始序列无关。因为每趟排序折半寻找插入位置时,折半次数是一定的,折半一次就要比较一次,所以比较次数也是一定的。

冒泡排序 优点:稳定 缺点: 慢,每次只能移动相邻两个数据

快排:优点:极快,数据移动少 缺点:不稳定 O(log2n)~O(n)

插入: 最坏情况为输入序列是降序排列的,此时时间复杂度 O(n^2) 空间复杂度 O(1) 最优时间复杂度 ---- 最好情况为输入序列是升序排列的,此时时间复杂度 O(n)

```
void BinarySort(int a[],int n )
{for (int i = 0; i < 10; i++)
{int start = 0; int end = i - 1;
int middle = 0; int temp = a[i];
while (start <= end){
middle = (start + end) / 2;
if (a[middle] > temp)
//要排序元素在已经排过序的数组左边
{end = middle - 1}
else{start = middle + 1}    }
for (int j = i - 1; j > end; j--)
//找到了要插入的位置
然后将这个位置以后的所有元素向后移动
{a[j + 1] = a[j]; }
a[end + 1] = temp;}    }
void BubbleSort(int arr[], int n)
{ for (int i = 0; i < n; ++i)
for (int j = 0; j < n - i; ++j)
if (arr[j] < arr[j + 1])
swap(&arr[j], &arr[j + 1]);}
void InsertionSort(int A[], int n)
{ for (int i = 1; i < n; i++)
{ int get = A[i];    int j = i - 1;
while (j >= 0 && A[j] > get)
{A[j + 1] = A[j];      j--; }
A[j + 1] = get; } }
void main()
{srand(time(0)); int *arr;
arr = new int[10];
for (int i = 0; i < 10; i++)
{arr[i] = rand() * 30 / RAND_MAX;
cout << arr[i] << " ";}
BinarySort(arr, 10);//从小到大
BubbleSort(arr, 10);//从大到小
delete []*arr;
system("pause");}
```

```
//char  转为  double
double atof(const char *str)
{
  double num = 0;
  double d = 10;
  bool flag = true; //正数为  ture,  负数为 false
  if (*str == ' ' || *str == '+') //若为正数
  {    str++;    }
  if(*str == '-') //若为负数
  { flag = false;    str++; }
  if (!(*str >= '0' && *str <= '9'))
  //如果开始非数字,返回 0
  return num;
  while(*str >= '0' && *str <= '9' && *str != '.')
  //小数点之前
  { num = num * 10 + (*str - '0'); str++; }
  if (*str == '.')      str++;
  while (*str >= '0' && *str <= '9')//小数点之后
  {num = num + (*str - '0') / d;
  d *= 10;
  str++;}
  return num*(flag ? 1: -1); //正数负数
}
```

```
public: void setName(char *name);
void CEmployee::setName(char *name)
{ strcpy(name, szName);}
s.setName("TOM");

swtich(n)
{case 1: …;break; case 2:…; break;}

void fun1(class t){t.num =1;}    // fun1(t);
void fun2(class stu t[]){t[0].num =1;}    // fun2(t);
void fun3(class stu *t){t->num =1; (*t).num1 = 2;}    // fun3(&t);
void fun4(class stu &t){t.num =1;}    // fun4(t);
```

构造函数:

```
Complex(double r)
Complex(double r, double i)
Complex(Complex c1, Complex C2)
```

+-重载:

```
class CComplex
{
public:
double real, imag;
CComplex(double r = 0, double i = 0) :real(r), imag(i) {};
CComplex operator - (const CComplex &c);
};
CComplex operator +(CComplex &a, CComplex &b)
{return CComplex(a.real + b.real, a.imag + b.imag);}
CComplex CComplex ::operator - (const CComplex &c)
{return    CComplex(this->real - c.real, this->imag - c.imag);}
```

=重载:

```
String & String::operator = (const String &s)
{    if (str == s.str)    return *this;
if (str != NULL)    delete[]str;
if (s.str != NULL) { str = new char[strlen(s.str) + 1]; strcpy(str, s.str); }
else str = NULL;
return *this;    }
```

流插入提取

```
#include<iostream>
#include<string>
#include<cstdlib>
using namespace std;
class Complex
{
double real, imag;
public:
Complex(double r=0, double i=0) :real(r), imag(i) {};//默认构造函数
friend ostream & operator <<(ostream &os, const Complex &c);
friend istream & operator >>(istream &is, Complex &c);
};
ostream &operator<<(ostream &os, const Complex &c)
{ os << c.real << "+" << c.real << "i";return os;    }
istream & operator >>(istream &is, Complex &c)
{    string s;    is >> s;
int pos = s.find("+", 0); cout << pos << endl;
string sTmp = s.substr(0, pos);//获得从第 0 位开始的长度为 5 的字符串
c.real = atof(sTmp.c_str());//将 string 对象转化为 char 对象
sTmp = s.substr(pos + 1, s.length() - pos - 2);
c.imag = atof(sTmp.c_str());
return is;    }
int main()
{Complex c;
int n;    cin >> c >> n;
cout << c << "," << n;
system("pause");    }
```

友元(类的外部访问对象私有成员):

友元函数(友元函数内部访问该类对象的私有成员):

pivate: int price;

friend int MostExpensiveCar(CCar cars[],int total);//全局函数

friend void CDriver::ModifyCar(CCar *pCar);

友元类: friend class CDriver (CDriver 的所有成员函数可以访问 CCar 的私有成员)

this 指针:

CComplex Add(CComplex *a){CComplex temp; temp.real = this->real + a->real;}