ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
32	(space)	57	9	82	R	107	k
33	!	58	:	83	S	108	l
34	"	59	;	84	T	109	m
35	#	60	<	85	U	110	n
36	$	61	=	86	V	111	o
37	%	62	>	87	W	112	p
38	&	63	?	88	X	113	q
39	,	64	@	89	Y	114	r
40	(	65	A	90	Z	115	s
41	)	66	B	91	[	116	t
42	*	67	C	92	/	117	u
43	+	68	D	93	]	118	v
44	,	69	E	94	^	119	w
45	-	70	F	95	_	120	x
46	.	71	G	96	、	121	y
47	/	72	H	97	a	122	z
48	0	73	I	98	b	123	{
49	1	74	J	99	c	124	|
50	2	75	K	100	d	125	}
51	3	76	L	101	e	126	`
52	4	77	M	102	f	127	DEL
53	5	78	N	103	g		
54	6	79	O	104	h		
55	7	80	P	105	i		
56	8	81	Q	106	j		

一、数学问题

1. 精度计算——大数阶乘

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    int c[10000];
    c[0] = 1;
    int di = 1;
    cin >> n;
    int num = 0;
    for(int i = 2; i <= n; i++)
    {
        for(int j = 0; j < di; j++)
        {
          num =  c[j] * i + num;
          c[j] = num % 10;
          num /= 10;
        }
        while(num)
        {
            c[di] = num % 10;
            num = num / 10;
            di ++ ;
        }
    }
    for(int i = di - 1; i >= 0; i--)
    {
        cout<<c[i];
    }
    return 0;
}
```

2. 快速幂取模

```
LL pow(LL a, LL n, LL p)    //快速幂 a^n % p
{
    LL ans = 1;
    while(n)
    {
        if(n & 1) ans = ans * a % p;          //若不取模就去掉p
        a = a * a % p;
        n >>= 1;
    }
    return ans;
}
```

3.模运算

1.取模运算：a % p（a mod p），表示a除以p的余数。 2.模p加法：(a + b) % p = (a%p + b%p) % p 3.模p减法：(a - b) % p = (a%p - b%p) % p 4.模p乘法：(a * b) % p = ((a % p)*(b % p)) % p 5.幂模p： (a^b) % p = ((a % p)^b) % p 6.模运算满足结合律、交换律和分配律。

4. 欧拉函数

欧拉函数，一般记为 ϕ(n)，表示小于等于 n 的数中与 n 互质的数的个数。

```
 求 n  以内与 n 互质数的个数
```

```
int primes[N], euler[N], cnt;
bool st[N];

// 质数存在primes[]中, euler[i] 表示
// i的欧拉函数
void get_eulers(int n)
{
    euler[1] = 1;
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i])
        {
            primes[cnt ++ ] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; j < cnt && i * primes[j] <= n; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0)
            {
                euler[i * primes[j]] = euler[i] * primes[j];
                break;
            }
            euler[i * primes[j]] = euler[i] * (primes[j] - 1);
        }
    }
}
```

5.扩展欧几里得算法

裴蜀定理：若 a,b 是整数,且 (a,b)=d，那么对于任意的整数 x,y,ax+by都一定是 d的倍数，特别地，一定存在整数 x,y，使 ax+by=d 成立。扩展欧几里得算法可以在O(logn) 的时间复杂度内求出系数 x,y。

```
int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1; y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a/b) * x;
    return d;
}
```

6.线性筛素数

```
int primes[N], cnt;
bool st[N];
void get_primes(int n)
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; j < cnt && i * primes[j] <= n; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

7.数论简单小结论

- 根据费马小定理推得：a^x % p = a^(x%(p-1)) % p

8.快速读入

```
#include<iostream>

using namespace std;

//关闭流同步 ios_base::sync_with_stdio(false)
inline int read() {
    int x = 0, f = 1;
    char c = getchar();
    while(c < '0' || c > '9') {
        if(c == '-') f = -1;
        c = getchar();
    }
    while(c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
```

```
        c = getchar();
    }
    return x * f;
}
int main(){
    int a;
    a = read();
    cout<<a;
    return 0;
}
```

二、字符串处理

1.字符串标准库

C标准库（针对字符数组）

`int strlen(const char *str)`：返回从 `str[0]` 开始直到 `'\0'` 的字符数。

`sscanf(const char *__source, const char *__format, ...)`：从字符串 `__source` 里读取变量，比如 `sscanf(str,"%d",&a)` 。

`sprintf(char *__stream, const char *__format, ...)`：将 `__format` 字符串里的内容输出到 `__stream` 中，比如 `sprintf(str,"%d",i)` 。

`int strcmp(const char *str1, const char *str2)`：按照字典序比较 `str1 str2` 若 `str1` 字典序小返回负值，一样返回 0，大返回正值 请注意，不要简单的认为只有 `0，1，-1` 三种，在不同平台下的返回值都遵循正负，但并非都是 `0，1，-1`

`char *strcpy(char *str, const char *src)`：把 `src` 中的字符复制到 `str` 中， `str` `src` 均为字符数组头指针，返回值为 `str` 包含空终止符号 `'\0'` 。

`char *strcat(char *str1, const char *str2)`：将 `str2` 接到 `str1` 的结尾，用 `*str2` 替换 `str1` 末尾的 `'\0'` 返回 `str1` 。

C++标准库（针对字符串，兼容字符数组）

std::string 头文件string

- 访问运算符 `s[cur]` 返回 `cur` 位置的字符（引用）。
- 访问函数 `data()/c_str()` 返回一个 `const char*` 指针，内容与该 `string` 相同。
- 容量函数 `s.size()` 返回字符串长度。
- 判空函数 `s.empty()` 返回字符串是否为空（布尔型）
- `s.resize(int len,char c)` 把字符串当前大小置为len，并用字符c填充不足的部分
- `s.copy(char *s, int n, int pos = 0) const` 把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目
- `getline(cin, st1)` 读取字符直到换行，单纯的cin遇到空格停止

- `getline(cin, st1, 'a')` 一个直到'a'结束，其中任何字符包括'\n'都能够读入
- `st1.length()` 返回字符个数
- `st1 + st2` `st1 = st2` 可直接连接、替换
- `tolower(c)` `toupper(c)` 大小写转换（字符）
- `s.substr(i, j)` 截取s串中从i开始长度为j的子串
- `s.insert(it, p)` 把字符串p插入到it的位置
- `s.erase(3)` `s.earse(0, 4)` 删除对应下标开始对应个数的元素
- `s.compare("good")` s与"good"比较相等返回0，比"good"大返回1，小则返回-1（字典序）
- `s.reverse(s.begin(), s.end())` 字符串反转
- `s.swap(string &s2)` 交换当前字符串与s2的值

2.字符串替换/插入

替换函数

`s.replace(int p0, int n0, const char *s2)` 删除从p0开始的n0个字符，然后在p0处插入串s2

`s.replace(int p0, int n0,const string &s2, int pos, int n)` 删除p0开始的n0个字符，然后在p0处插入串s2中从pos开始的n个字符

`s.replace(int p0, int n0,int n, char c)` 删除p0开始的n0个字符，然后在p0处插入n个字符c

插入函数

`s.insert(int p0, const char *s2)` 在p0位置插入字符串s

`s.insert(int p0,const string &s2, int pos, int n)` 在p0位置插入字符串s中pos开始的前n个字符

`s.insert(int p0, int n, char c)` 此函数在p0处插入n个字符c

整体替换

使用STL的replace函数实现，结果是母串中所有相同的都被替换

用法：`string_replace(要替换的母串，要替换掉的子串，用来替换的字符串)`

如：`string_replace(st1, "Winter", "AutoWA")` 把st1中的winter换成AutoWA

```
void string_replace(string & strBig, const string & strsrc, const string &strdst) {
        size_t pos=0;
        size_t srclen=strsrc.size();
        size_t dstlen=strdst.size();
        while( (pos=strBig.find(strsrc, pos)) != string::npos){
                strBig.replace(pos, srclen, strdst);
                pos += dstlen;
        }
}
```

3.字符串查找

查找函数`find()`找到并返回某字符的位置

- `st1.find('a')` 输出第一个'a'的下标（0开始），查找对象可以也是一个字符串
- `st1.find('a', 3)` 从下标3开始查询a的下标，返回的下标同上
- `st1.rfind('a', 5)` 从0到5从后向前查找a所在该串的位置
- `find()` 函数返回的是一个size_t类型（或string::size_type类型）

  当没有匹配到目标的时候，返回的是str.npos

  例：定义一个size_t类型变量，可判断是否匹配到了

```
size_t position;
string s = "AutoWA";
position = s.find("WA");
cout << position << endl;  //输出4
position = s.find("ACM");
if(position == s.npos) cout << "None" << endl;  //未匹配到，输出None
```

- `st1.find_first_of(st2, 0)` 在str1中从0开始向后查找，只要在st1中遇到一个字符与st2中任意一个字符相同，就返回该字符在st1中的位置，匹配失败返回npos
- `st1.find_last_of(st2, 4)` 同上，从指定位置向前查找
- `st1.find_first_not_of(st2, 0)` 同上，遇到一个字符与st2中任何一个不同，返回位置，last同理。

4.最长公共子序列（LCS）

给定两个字符串 str1 和 str2 ，如果将 str1 去掉一些字符后得到字符串str，将str2去掉一些字符后也能得到字符串 str ，则 str 是 str1 和 str2 的一个公共子串。在得到str的时候不能改变字符的相对顺序。求字符串 str 长度的最大值。

如 ABCEF BMCGUAFB，则最长公共子序列为3: BCF

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int dp[5005][5005];
    char a[5005],b[5005];
int main()
{

    cin>>a+1>>b+1;
    int n=strlen(a+1);
    int m=strlen(b+1);
    for(int i=1;i<=n;++i)
    for(int j=1;j<=m;++j)
    {
        if(a[i]==b[j])dp[i][j]=dp[i-1][j-1]+1;
        else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
    }
    cout<<dp[n][m];
}
```

三、排序/查找

1.快速排序 一般较快-取决于元素排列顺序

说明：quickSort(数组名，起始元素下标， 终止元素下标)，默认升序排列

```
void quickSort(int *arr, int left, int right){
    int i = left, j = right;
    int mid = arr[(i+j)/2];
    while(i <= j){
        while(arr[i] < mid) i ++;
        while(arr[j] > mid) j --;
        if(i <= j){
            int tmp;
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i ++;
            j --;
        }
    }
    if(i < right) quickSort(arr,i, right);
    if(left < j) quickSort(arr,left, j);
}
```

2.归并排序 稳定-时间复杂度固定

用法同上

```
int temp[max];
void merge(int *a, int left, int mid, int right) {
    int i = left, j = mid + 1, n = 0, length = right - left;
    while(i <= mid && j <= right) {
        if(a[i] < a[j]) {
            temp[n++] = a[i++];
        } else {
            temp[n++] = a[j++];
        }
    }
    if(i > mid) {
        while(j <= right) {
            temp[n++] = a[j++];
        }
    } else {
        while(i <= mid) {
            temp[n++] = a[i++];
        }
    }
```

```
    for(int k = 0;k <= length;++k) {
        a[left+k] = temp[k];
    }
}
void mergesort(int *a, int left, int right) {
    if(left < right) {
        int mid = (left + right) / 2;
        mergesort(a,left,mid);
        mergesort(a,mid+1,right);
        merge(a,left,mid,right);
    }
}
```

3.二分查找

说明：*arr为数组名， begin为起始下标，end为终止下标， e为要查找的内容，最终返回查找的内容的下标，如果没有找到则返回比它小的第一个数的下标

数组必须已经按照升序排列好

```
int bSearch(int *arr, int begin, int end, int e) {
    int mid, left = begin, right = end;
    while(left <= right) {
        mid = (left + right) >> 1;
        if(arr[mid] > e) right = mid - 1;
        else left = mid + 1;
    }
    return right;
}
```

4.STL排序

头文件依赖：`<algorithm>`

`sort(begin, end, cmp)` 范围是左闭右开

begin数组首地址，end数组尾地址，cmp自定义比较规则，默认为升序排序。

降序排列`sort(begin, end, greater<int>())`

或自定义比较方法

```
bool cmp(int a, int b) {
    return a > b;
}
```

结构体排序自定义cmp：

```
bool cmp(T a, T b) {
    return a.time > b.time;
}
```

四、STL库

1.栈(stack)

头文件 `#include <stack>`

定义 `stack<typename> name`

常用函数

`push()` 压栈，入栈一个元素

`top()` 访问栈顶元素 ***注意：当栈空时访问栈顶元素非法***

`pop()` 弹出栈顶元素

`empty()` 判断是否栈空，是返回true，否返回false

`size()` 返回栈内当前元素个数

2.不定长数组（vector）

头文件 `#include <vector>`

定义 `vector<typename> name`

访问

- 通过下标访问name[index]
- 通过迭代器访问

  定义方式：`vector<typename>::iterator it = name.begin()`

  得到 `it` 后通过 `*it` 来访问vector里的元素， `*(it + i)` 来访问第 `i` 个元素

  遍历vector的循环可写：

```
for(vector<typename>::iterator it = name.begin(); it != name.end(); it++){}
```

常用函数

`push_back()` 在尾部插入一个元素

`pop_back()` 在首部插入一个元素

`size()` 获取vector长度，返回unsigned类型

`clear()` 清空

`insert(it, x)` 向迭代器it处插入元素x

`erase()`

- 删除迭代器it处的单个元素 `erase(it)`

- 删除一个区间 [first, last) 的元素 `erase(first, last)` first与last都是迭代器

3.集合（set）

特点：内部自动有序且不含重复元素，默认升序

头文件 `#include <set>`

定义 `vector<typename> name`

只能通过迭代器访问set内元素

访问

只能通过迭代器访问，迭代器定义 `set<typename>::iterator it`

使用 `*it` 来访问set内的元素

采用下列方式枚举：

```
for(set<typename>::iterator it = name.begin(); it != name.end(); it++){}
```

常用函数

`insert()` 向集合内插入一个元素

`find(value)` 返回对应值是value的迭代器

`erase()` 删除单个元素 `s.earse(it)` （迭代器）或 `s.erase(value)` （对应值）

`size()` 返回集合大小

`clear()` 清空集合

通过重载小于号自定义结构体集合排序规则

重载结构体小于号，示例（id优先升序，相同时age升序）：

```
bool operator < (const Students &s) const {
    if(id != s.id) return id < s.id;
    else return age < s.age;
}
```

4.队列（queue）

头文件 `#include <queue>`

定义 `queue<typename> name`

常用函数

`front()` 访问队首

`back()` 访问队尾

`push()` 入队一个元素

`pop()` 队首元素出队

`empty()` 队列判空，返回true为空

`size()` 返回队列大小

5.优先队列（priority_queue）

头文件 `#include <queue>`

定义 `priority_queue<typename> name`

常用函数

`push()` 往堆底插入元素，向上调整

`top()` 访问队首元素，也就是优先级最高的元素

`pop()` 令队首元素出队

`empty()` 队列判空

`size()` 返回队列大小

优先级自定义

- 基本数据类型

  `priority_queue<int, vector<int>, less<int> > q` 数字大的优先级大

  `priority_queue<int, vector<int>, greater<int> > q` 数字小的优先级大

- 结构体类型（或在结构体内重载小于号）

```
struct cmp{
    bool operator (const student &s1, const student &s2){
        return s1.s_grade > s2.s_grade;//grade大的优先级高
    }
}
priority_queue<student, vector<student>, cmp> q;
```

6.键值对（map）

特点：每个键只能出现一次

头文件 `#include <map>`

定义 `map<typename1, typename2> mp`

访问

- 通过下标访问，例如`mp['key']`
- 通过迭代器访问

```
map<typename1, typename2>::iterator it;
for(map<typename1, typename2>::iterator it = mp.begin(); it != mp.end(); it++)
{//it->first; 访问键, it->second; 访问值}
```

常用函数

`find(key)` 返回key的映射的迭代器

`erase(it)` 删除迭代器指向的键值对

`erase(key)` 删除键为key的键值对

`erase(first, last)` 删除[first, last)区间元素

`size()` 返回map的大小

`clear()` map判空