1.1基本数据类型

基础知识

1.机器语言，汇编语言，高级语言

2.实现一个源程序 三步

1. 编辑
2. 编译（预处理，编译，连接）#预处理指令
3. 运行

3.关键字

array / catch /extern/ friend / goto / inline / nullptr / short /template / this / throw / try / typedef / typename / union / virtual

4.分隔符 便于编译系统识别

1.基本数据类型

标识符	字节数	范围
char	1	-128~127
int	4	-2 147 483 648 ~ 2 147 483 647 (亿级)
double	8	

```
const int MAX=520;  //写在最上面，然后设置数组，便于修改MAX的值

int arr[MAX];
```

浮点数判断范围

不会直接用==，而是用

```
#include<cmath>

fabs(a)<=1e-8 ;
```

减少浮点数精度误差，可以将所有数字×10 或100 .....使其变成整数

或者是可以接受的误差 如 1e-3

2.标识符：

以字母或下划线开始，由字母、数字和下划线组成的符号串

关键字小心 字母大小写敏感 （关键字都是小写，所以有一些就直接大写避免和关键字撞） Months Count.....

3.enum 枚举型

```
enum colour{red,yellow,blue,white,black};
//枚举用标识符中的序号表示数据
int main()
{
    colour c; //c是枚举类型数
    c=red;
    cout<<"red"<<c<<endl;
}
//运行结果 red 0
//枚举常量的序值可以在初始化修改，但必须是升序的
enum Months{Jan=1,Feb=2};
```

4.**进制**

进制	**备注**	**示例**	**直接以该进制输出显示结果**
十进制	后缀L表示长整数，后缀U表示无符号整数		cout<<dec<<a<<endl;
八进制	以0为前缀，只能表示正整数	023（8）=19（10）	cout<<oct<<a<<endl;
十六进制	以0x 或0X为前缀，只能表示正整数	0x3b(a~f) 0XFF （A~F)	cout<<hex<<a<<endl;

5.浮点型

（小数）13.89 / .638 /-435. 正确的

（指数表示）尾数和指数不能省略，指数必须是整数

12e8 / 314e-5 / .618e3 正确的

e-7 / .e10 / 1e2.5 错误的

6.字符型

1. '\ddd'八进制 '\xhh'十六进制

   如 '\101' 和'\x41'都表示'A'；

2. '\ +'在字符前添加 \ 输出字符

3. 字符串最后会自动添加空字符'\0'作为串结束符。所以 'x' 和 "x" 的储存形式不同

7.指针

1. 不管指针对象是什么类型，指针都是占4个字节

2.

指针	例子	
指向常量的指针	const int *p	指针的指向可以修改，但是指针指向的值不能修改
指针常量	int * const p=&a	指针的指向不可以修改，指针指向的值能修改
指向常量的常量指针	const int *const p	指针的指向不可以修改，指针指向的值不能修改

3. 常引用 const int &pa=a;

8.数据输入与数据输出

1. cin>>a>>b; 其中cin是预定义输入流对象，>>在流操作中称为流提取操作符

（输入中不能出现表达式）

ios::syns_with_stdio(false)； 关闭流同步，加快cin的读入速度

scanf();

```
#include<iomanip>

setfill(char c)                  //设置填充符号c
setpreicision(int n)              //设置浮点数输出精度
setw(int n)                       //设置输出宽度

输出小数   cout<<fixed<<setprecision(3)<<a<<endl;
填充字符 010   cout<<setw(3)<<setfill('0')<<a<<endl;
```

9.数学函数

#include< cmath >

函数原型	样例	说明
double sin(double x)	sin(3.1415)	x弧度制
double exp(double x)	exp(1)	e^x
double log(double x)	log(10)	
double pow(double x,double y)	pow(3,2)	x^y
double sqrt(double x)	sqrt()	
double fabs(double x)	fabs(-10)	绝对值
double ceil(double x)	ceil(2.1)	向上取整
double floor(double x)	floor(2.9)	向下取整

10.强制类型转换：

（把一个整数变成浮点数） 加 . 0 或者* 1 . 0

11.四舍五入精确到整数

（1） int(ans+0.5) //相当于把小数抹掉

(2) #include< cmath > round 函数就是四舍五入的

12.数字作为字符型输入

```
char c;   //若将数字作为字符型输入，如何转化为整型数字
int a;
cin>>c;
a=c-'0';
```

13.随机数

```
#include<ctime>

int ans;
srand(time(0));
ans=rand()%100 + 1;

//   rand()%(b-a+1)+a  产生一个从a到b的随机数
```

14.质数

```
int is_prime=1;
for(int j=2;j*j<=i;j++)
{
   if(i%j==0)
   {
      is_prime=0;
      break;
   }
}
if(is_prime)
```

二、程序控制结构

1.算术运算

1. 取模 % 计算两个整数的余数（注意不能%0 或者 %2.5小数）
2. 自增自减 i++; ++i; i--; --i; ++(++i);

3. 强制类型转换 2.0/4

2.逻辑表达式

1. ==是等于，=是赋值
2. 优先级从高到低：！ -> 算术运算符 -> 关系运算符 -> && -> ||
3. && 和 || 都是从左开始，！ 是从右开始

   （如果左边已经满足条件了右边可以不计算直接退出）
4. max=a > b ? a : b;
5. (a=b)=10;首先执行 a=b,把b的值写入a，表达式的值确实于a，然后在执行a=10.上述对a做了两次写操作，对b做了一次读操作。
6. 逗号表达式
   1. 每个表达式按顺序执行
   2. 逗号表达式也表达一个值，这个值是最后一个表达式的值（逗号运算级别最低）
   3.

表达式	**结果**
x=a=3,2*6 ;	x=3
x=(a=3, 2*6)	x=12

3. if 语句

1. 计算三角形面积

   area = $\sqrt{s * (s-a) * (s-b) * (s-c)}$

   s=(a+b+c)/2.0
2. 注意作用域 { }
3. 注意特殊情况 特判 a<0; a=0;
4. 注意判断 if (! answer)中的 !
5. *浮点数判断范围*

   不会直接用==，而是用

   ```
   #include<cmath>

   fabs(a)<=1e-8 ;
   ```
6. == 与 =

   ```

   ```

```
 if(c=a-b)  //问a-b是否等于0
 if(c==a-b)   // 二者是否相等
```

4. switch 语句

```
switch(表达式)    //表达式只能是整型、字符型、枚举型
{
    case 常量表达式1: 语句1; break;
    case 常量表达式2: 语句2; break;
    ……
    defalut:语句
}

    //如果语句一有很多，并且需要设置变量，需要加{}
    // 如case 1 :{int a=10; ……}
     // 有没有加break；加了则退出，没有则会继续往下走。
    /*
     switch(m){
     case 1:cout<<"one ";
     case 2:cout<<"two";
     case 3:cout<<"three";
     }
     */
    //   如果 m==1 那么输出结果为 one two three
```

5. while循环

根据条件执行操作而不需要关心循环次数

循环的初始化

```
先判断条件再看是否要执行
while (条件)
{
      int a=0;    //加法器
      循环体
      int b=1;     //乘法器初始
}
```

do while 语句

```
先执行一次再判断是否满足条件
do
{
```

```
    循环体
}while（条件）；            //  ;不要漏了
```

for 循环

for(初始化表达式； 循环条件表达式； 最后一句执行语句) { }

三、函数

1.值传递

1. return 会有匿名对象储存返回结果，这个匿名对象再返回调用之后撤销。

2. 如果实参类型和形参类型不同，则按形参的类型进行强制类型转换后赋给形参。

```
int max(a,b)
{return (a>b?a:b);}

m=max(5.2/2,0.5);
//输出结果
m=2;
```

3. 默认参数

```
double dist(double,double,double =0,double =0);
```

默认参数只能是最右边（尾数）的参数。

默认参数应该再函数名第一次出现时指定，后面函数定义不用重复给出

默认值可以是常量、全局变量或函数调用，但不能是局部变量。

可以用于内联函数。

2. 指针传递（地址调用）

1. 改变实参

```
void swap(int *a,int *b)
{
 int temp=*a;
 *a=*b;
 *b=temp;
}

swap(&x,&y);
```

传递常对象的地址,形参用const约束

```
int func(const int *p);

const int m=10;
cout<<func(&m)<<endl;
```

3.引用参数

形参不需要开辟新的存储空间,改变实参

```
void swap(int &a,int &b)
{
    int temp=a;a=b;b=temp;
}

swap (a,b);
```

const限定引用

```
void display(const int &rk);

int main()
{
    const int m=10;
    display(m);               //实参是变量
    display(m+10);
    display(4549);              //实参是常量
}
```

只有const引用对应的实参可以是常量或表达式，非约束的引用参数对应的实参必须是对象名。

二、函数的返回类型

1.返回基本类型

```
int function(int x)
{
    ……
    return x;
}
```

2.返回指针类型

指针函数，返回一个对象的地址表达式

int * maxpoint(int *x,int *y);

```
int *maxpoint(int *x,int *y)
{
    if(*x>*y)return x;
    return y;
}

main函数中
cout<<*maxpoint(&a,&b)<<endl;
//注意 * 间址访问，因为maxpoint返回一个地址
```

使用指针函数可以节省匿名对象造成的时空消耗

指针函数不能返回局部变量的指针

3.返回引用类型

int & maxreff(int &x,int &y);

```
int & maxreff(int &x,int &y)
{
    if(x>y)return x;
    return y;
}

cout<<maxreff(a,b)<<endl;
```

三、函数调用

1. 嵌套调用

main函数由操作系统调用

函数调用信息管理堆栈

2. 递归调用

递推和回归

构成递归函数的两个基本要素： （1）描述问题规模逐步减小的递归算法

（2）描述基本情况的递归终止条件

1. 求正整数a,b的最大公约数

```
// 默认 (a>b)
int gcd(int a,int b)
{
 return ( a%b ? gcd( b ,a % b): b);
}

//求a,b,c的公约数
int num=gcd(gcd(a,b),gcd(b,c));
```

2. 逆序输出

```
void reverse()
{
 char c;
 cin>>ch;
 if(ch!='.')
 {
     reverse();
 }
 cout<<ch;
}
```

巧妙利用堆栈

输入 abcd.

输出 .dcba

3. 斐波那契数列 0 , 1 , 1 , 2 , 3 , 5

Fibonacci(0)=0;

Fibonacci(1)=1;

Fibonacci(n)=Fibonacci(n-1)+Fibonacci(n-2);

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

4. 汉诺塔问题

```
 void move(int n, char a, char b, char c)
  {
  if(n==1){ cout << a << "->" << c << endl; }
  else
  {
      move(n - 1, a, c, b);
      cout << a << "->" << c << endl;
      move(n - 1, b, a, c);
  }
  }
```

四、函数地址与函数指针

1. 函数名、函数地址都是函数的入口地址

```
simple
&simple
*&simple
```

函数调用，后两种方式中括号不能省略

```
simple();
(&simple)();
(*&simple)();
```

2. 函数的类型

1. 函数类型指的是函数的接口，包括函数的参数定义和返回类型

   double (double,double)

   ```
   double max(double,double);
   double min(double,double);
   double average(double,double);
   //三个都是一样的函数类型
   ```

2. 用typedef来命名函数类型

   typedef 类型 函数类型名 （形参表）

   ```
   typedef double functiontype (double,double);

   functiontype max,min,average;
   ```

typedef可以用于定义标准类型的别名

typedef int integer;

integer 就是int的别名

typedef long long LL； //把long long替换成LL节约录入时间

3. 函数指针

```
double (*fp)(double,double);

typedef double (*ftype) (double,double);
或者 typedef functiontype *ftype;

ftype f1,f2;
f1=max;          //指向函数max
f2=min;          //指向函数Min
```

用函数指针调用函数

```
fp=max;
x=fp(0.5,3.92);

//从函数的调用性质可知，（*fp）（0.5，3.92）与fp(0.5,3.92)是等价的调用方式。但是（&fp）（0.5，3.92）就是错误的，fp是指针变量，它的地址不是函数的地址，它的值才是。
```

```
#include<iostream>
using namespace std;
typedef double funtype(double);
function circlePerimeter,circleArea;
double callFun(funtype*,double);
int main()
{
 double r;
 cin>>r;
 //用函数地址作为实参调用函数callfun
cout<<"the perimeter of circle is :"<<callFun(circlePerimeter,r)<<endl;
cout<<"the area of circle is :"<<callFun(circleArea,r)<<endl;
}
const double PI=3.14159;
double circlePerimeter(double r)
 { return 2*PI*r;}
 double circleArea(double r)
 { return PI*r*r;}
 double callFun(funtype * fp,double r)
 { return fp(r);}
```

五、内联函数

函数调用都需要建立堆栈空间来保存调用时的现场状态和返回地址

函数定义时前面加inline ， 后面不能重复给出

内联函数 1-5行小程序，不能有递归

```
inline int isnumber(char);
int main()
{
    ……
}
int isnumber(char c)
{
    return ch>='0' && ch<='9' ? 1 : 0;
}
```

重载函数

函数参数表（参数类型和个数）

```
//参数类型相同，个数不同
int max(int ,int);
int max(int ,int ,int);

//参数类型不同，个数相同
int abs(int a);
double abs(double a);

//返回类型无关,如下面两个不是重载函数
int max(int ,int );
double max(int ,int);
```

重载函数中使用默认参数可能产生二义性

六、变量存储特性和标识符作用域

1. 自动存储 auto 和 register ， 默认状态
2. 静态存储

   全局说明的标识符默认为extern

   static说明的局部变量只能在定义该变量的函数体中使用，默认初始化为0，系统退出后保持其存储空间和数值。

static int b; //静态变量，再次调用保留原值

如可以用来计算次数，递归算次数

eg。测试密码输入，本程序使用静态变量记录用户输入密码的次数，若连续三次输入错误则显示错误信息，并结束程序。

```
#include<iostream>
using namespace std;
int password(const int&);
int main()
{
    if (password(123456))cout << "Welcome." << endl;
    else cout << "Sorry,you are wrong." << endl;
    return  0;
}
int password(const int& key)
{
    static int n = 0;
    cout << "Please input the password.\n";
    int k;
    cin >> k;
    n++;
    if (n < 3)
    {
        if (k == key)return 1;
        password(key);      //递归，重新输入
    }
    else                    //连续输入三次错误
    {
        if (k != key)return 0;
    }
}
```

1. 具有文件作用域的变量称为 全局变量，具有块作用域的称为局部变量。
2. 块作用域 是由花括号 { }
3. 函数作用域

   语句标号“：”（后面带冒号的标识符）是唯一具有函数作用域的标识符。语句标号一般用于switch结构中的case标号，以及goto 语句转向入口的语句标号。标号可以在函数体中任何地方使用，不能再函数体外使用。
4. 文件作用域

   任何在函数之外说明的标识符都具有文件作用域。这种标识符对于从说明起到文件尾的任何函数都是可见的。
5. 具有文件作用域的是全局变量，具有块作用域的是局部变量。

6. 全局变量说明时默认初值为0，当局部变量和全局变量同名时，在块内，全局变量被屏蔽。在块内要访问全局变量，可以使用 作用域标识符“：： ”/

7. 终止程序执行

    1. abort函数 void abort(void)
    2. assert函数

```
#include<cassert>

void assert(int expression);
```

    3. exit函数 这三个函数都会直接退出整个应用程序，通常用于处理系统的异常错误。

1.一维数组

1.一维数组的访问

数组名就是数组的首地址

1. 下标访问 a[i]
2. 以指针方式访问数组 * (a+i)

```
int a[10];
a=&a[0];
a+1=&a[1];
a+i=&a[i];
```

a的值是内存分配的直接地址，常指针，a++来移动指针是错误的。 正确应该如下

```
int a[10]={1,2,3,4,5,6,7,8,9,10},*p;

for(int i=0;i<10;i++)
{cout<<*(a+i)<<'\t';}

for(p=a;p<a+10;p++)
{cout<<*p<<'\t';}

//a+i和p++计算的偏移单位是一个整形数据的长度
```

2.指针数组

类型 * 标识符 [下标表达式]

指针 都是 四个字节（无论什么类型数据）在32位操作系统下

1. 指向基本数据类型的指针数组

离散的数据，可以用指针数组来管理他们的地址，通过指针数组进行访问

```
int a=11,b=22,c=33;
int *p[3];
p[0]=&a; p[1]=&b; p[2]=&c;
for(int i=0;i<3;i++)
{cout<<*p[i]<<" ";}
```

2. 指向数组的指针数组

```
int main()
{
 double aa[2]={1.1,2.2};          //aa是一维数组名，内存直接地址
 double bb[2]={3.3,4.4};
 double cc[2]={5.5,6.6};
 double (*pf[3])[2];        //逻辑上二级指针
 pf[0]=&aa;               //aa 是一级指针  pf[0]=&aa才是同级操作
 pf[1]=&bb;                   //&aa 没有实质意义
 pf[2]=&cc;
 for(int i=0;i<3;i++)
 {
     for(int j=0;j<2;j++)
     {
         cout<<*(*pf[i]+j)<<" ";
     }
     cout<<endl;
 }
}
```

用指针数组调用函数

```
double (*pfun[4])(double);
pfun[0]=max;
pfun[1]=min;

for(int i=0;i<4;i++)
  cout<<(*pfun[i](x))<<endl;

 // *pfun[i](x)  等效于 pfun[i](x);调用函数
```

2.二维数组

1. 二维数组的定义和初始化

高维数组在内存中以高维优先的方式存放

利用初始化值表，可以省略高维数组中的最高维长度说明，低维不可以

```
int a[][3]={1,2,3,4,5,6};       对
int b[2][]={1,2,3,4,5,6};       错
```

2.以指针方式访问数组

```
int a[2][3];

a=&a[0];        a+1=&a[1];            //二维数组名逻辑上是二维指针
*a=a[0];
a[0]=&a[0][0];               //a[i]表示第i行元素的首地址，一级指针
*a[0]=a[0][0];
```

```
二维数组名 a 逻辑上是二维指针
但是
int **pp;
pp=a;   这是错误的

只有定义一个指向一维数组的指针，才可以操作逻辑上为二级指针的二维数组名
int (*pary)[4];
pary=a+i;
```

```
int main()
{
    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
    int i,j,total=0;
    int *p,(*pary)[4];
    for(p=a[0];p<a[0]+12;p++)
         total+=*p;
    cout<<"total is "<<total<<endl;
    for(i=0;i<3;i++)
       for(j=0;j<4;j++)
       {
        pary=a+i;
        cout<<setw(3)<< *(*pary+j);
       }
```

3.数组作为函数参数

1. 数组名作为函数参数

```
void fun(int x[],int a){……};

 //这里的x[]实际是指针     int x[]等效于int *x
//在fun函数中是允许修改指针x，例如，在函数内可以移动指针 x++ 来访问数组元素，但是数组名ary++就不允许

int ary[3];
fun(ary,n);
```

4.动态存储

new按照指定类型的长度分配存储空间，并返回所分配空间的首地址。

```
int *p=new int(5);

int *q=new int[4]; //申请长度为4的动态数组

delete p;
delete []q;
p=NULL; q=NULL;
```

```
void App(int *& pa,int len)        //int * & pa;注意是指针引用参数
{
  pa=new int[len];           //如果是 int *pa;那么函数调用时就只是
  if(pa==NULL)             //将ary的地址赋给形参pa，在APP函数中pa的
    {                        // new申请的地址写入pa，与ary无关。
                           //函数返回后，ary也无法获得动态数组的地址
      cout<<"allocation faiure.\n";
      return ;
    }
  for(int i=0;i<n;i++)
     pa[i]=0;
}
int main()
{
    int *ary=NULL,*t;
    int i,n;
    cout<<"n=";
    cin>>n;
    APP(ary,n);
    for(t=ary,t<ary+n;t++)
       cout<<*t<<" ";
    cout<<endl;
    for(int i=0;i<n;i++)
      ary[i]=10+i;
```

```
    cout<<endl;
    delete []ary;
    ary=NULL;
}
```

杨辉三角，记得是从右到左迭代

```
#include<iostream>
using namespace std;
void yhtriangle(int* const py, int n);
int main()
{
    int* p, n;
    do
    {
        cin >> n;
    } while (n > 20 || n < 0);
    p = new int[n + 1];
    yhtriangle(p, n);
    delete[]p;
    p = NULL;
}
void yhtriangle(int* const py, int n)
{
    py[0] = 1;
    cout << py[0] << endl;
    for (int i = 1; i < n + 1; i++)
    {
        py[i] = 1;
        for (int j = i - 1; j > 0; j--)
        {
            py[j] = py[j] + py[j - 1];
        }
        for (int k = 0; k <= i; k++)
            cout << py[k] << " ";
        cout << endl;
    }
}
```

5.C字符串

字符串存放在字符型数组中，并添加'\0'作为其结束标记

```
#include<cstring>

char str1[10]={'s','t','u','d','e','n','t','\0'};
static char str[10]={'s','t','u','d','e','n','t'};  // 自动补\0
```

```
char *str="student";
char str2[]="student";
```

iostream

1. 字符串常量、字符串数组、字符指针都表示字符串
2. 输出字符指针就是输出整个字符串
3. 输出字符指针的间接引用就是输出单个字符

```
char *s="hello";
for(int i=0;i<5;i++)
   cout<<*(s+i);
cout<<endl;
for(int i=0;i<5;i++)
   cout<<s+i<<endl;
```

4. strlen(n),计算的时有效字符的个数，空字符不包括在内
5. 字符串的直接比较就是地址比较
6. 字符串的赋值就是地址赋值

```
char str1[10]="computer",str2[10],*sp;
int i=0;
while(str1[i])
 str2[i]=str1[i++];
str2[i]='\0';
sp=str1;   //正确的赋值
```

~~str2=str1~~; 错误的赋值

五、集合与结构

1. 位运算

^异或 ~按位取反

位运算效率最高

按二进制位串形式输出无符号整数的值

```
#include<iostream>
using namespace std;
void bitDisplay(unsigned value);
int main()
{
```

```
    unsigned n;                  //无符号整数 unsigned
    cin >> n;
    bitDisplay(n);
    return 0;
}
void bitDisplay(unsigned value)
{
    unsigned c;
    unsigned  bitmask = 1 << 31;              // bitmask掩码
    cout << value<<"=";
    for ( c = 1; c <= 32; c++)
    {
     cout << (value & bitmask ? 1 : 0);   //注意加括号整个括起来
     value <<= 1;
     if (c % 8 == 0)cout << ' ';
    }
    cout << endl;
}
```

变量a或b进行两次异或运算，不会改变原来的值。

利用异或运算的特点，可以快速进行两个变量值的交换而无需借助第三方辅助变量

```
#include<iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;
    cout << "a=" << a << " b=" << b << endl;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;
    cout << "a=" << a << " b=" << b << endl;
}
```

位运算要求操作数类型为整型，若要交换浮点型变量的数据，可以用指针方式分段处理数据，可以用于快速处理大量数据。

```
#include<iostream>
using namespace std;
int main()
{
    double a = 123.456, b = 456.789;
    int* ap, * bp;
    ap = (int*)(&a);        //  double型变为int型
    bp = (int*)(&b);
    cout << "a=" << a << " b=" << b << endl;
```

```
    *ap = (*ap) ^ (*bp); *bp = (*ap) ^ (*bp); *ap = (*ap) ^ (*bp);
    ap++; bp++;             //前半部分和后半部分
    *ap = (*ap) ^ (*bp); *bp = (*ap) ^ (*bp); *ap = (*ap) ^ (*bp);
    cout << "a=" << a << " b=" << b << endl;
    return 0;
}
```

2.集合

集合运算	对应的位运算
A∪B	A|B
A∩B	A&B
差集A-B	A& (~(A&B))
A包含于B	A|B==B
补集~A	~A
X属于A	1<<(x-1)&A==1<<(x-1)
空集A	A==0

六.链表

```
#include<iostream>
using namespace std;
struct node
{
    int data;
    node* next;
};
```

1.1建立单向链表

（挂在后面）

```
void Create_after(node* &head)  //使用指针的引用参数
{
    node* s, * p;
    s = new node;
    cin >> s->data;
    s->next = NULL;
    while (s->data!=0)
    {
        if (head == NULL)        //记得是==
            head = s;
```

```
        else p->next = s;
        p = s;
        s = new node;
        cin >> s->data;
    }
    p->next = NULL;
    delete s;
    return;
}
```

1.2建立链表（挂在前面）

```
void Create_before(node*& head)    //使用指针的引用参数
{
    node* s, * p;
    s = new node;
    cin >> s->data;
    s->next = NULL;
    while (1)
    {
        p = s;
        s = new node;
        cin >> s->data;
        s->next = p;
        if (s->data == 0)
        {
            head = s->next;
            break;
        }
    }
}
```

2.遍历链表

从头指针开始逐个输出，直到指针为空

```
void showlist(node* head)
{
    cout << "now the items of node are :\n";
    while (head)
    {
        cout << head->data << '\t';
        head = head->next;
    }
    cout << endl;
}
```

主函数调用上述函数

```
int main()
{
    node* head = NULL;
    Create_after(head);
    showlist(head);
    head = NULL;
    Create_before(head);
    showlist(head);
    return 0;
}
/*
运行效果如下:
1 2 3 4 0
now the items of node are :
1       2       3       4
4 5 6 7 0
now the items of node are :
7       6       5       4
可以作为逆序输出，挂在前面很有用
*/
```

3.插入结点

```
node *s;
s=new node;
cin>>s->data;//生成新节点
```

(1)在表头插入节点

```
s->next=head;
head=s;
```

(2)在*p后插入节点

```
//需要先找到*P的位置
s->next=p->next;
p->next=s;
```

(3)在*p前插入节点

```
//双节点扫,q为p的前一个结点
s->next=p;
q->next=s;
```

eg.用插入法生成一个有序链表

```
#include<iostream>
using namespace std;
struct node
{
    int data;
    node* next;
};
void insert(node* &head,int num)             //记得引用参数 & head
{
    node* s,*p,*q;
    s = new node;
    s->data = num;
    s->next = NULL;
    if (head == NULL)
    {
        head = s; return;             //若表空，则建立一个结点的链表
    }
    if (num < head->data)
    {
        s->next = head;
        head = s;
        return;                 //若输入的数据是最小值，则在最前面
    }
    for (q = head, p = head->next; p; p = p->next, q = q->next)
    {
        if (num < p->data)                          //搜索插入
        {
            q->next = s;
            s->next = p;
            return;
        }
    }
    q->next = s;                    //若被插数据最大，插入表尾
    return;
}
void showlist(const node* head)
{
    while (head)
    {
        cout << head->data << '\t';
        head = head->next;
    }
    cout << endl;
}
```

```
int main()
{
    cout << "Input the number until 0." << endl;
    node* head = NULL;
    int num;
    cin >> num;
    while (num != 0)
    {
        insert(head, num);
        cin >> num;
    }
    showlist(head);
    return 0;
}
```

4.删除结点

删除结点也要根据结点的位置进行不同处理，还要注意 *释放被删结点*。

（1）删除头节点

```
node *p;
p=head;
head=head->next;
delete p;
```

(2) 删除*p

q为p的前驱结点

```
q->next=p->next;
delete p;
```

eg.从链表中删除等于key的结点

```
void del(node*& head,int key)
{
    if (!head)
    {
        cout << "List null!\n"; return;
    }
    if (key == head->data)
    {
        node* s;
        s = head;
```

```
        head = head->next;
        delete s; s = NULL;
        return;
    }
    node* p, * q;
    for (p = head->next, q = head; p; q = p, p = p->next)
    {
        if (p->data == key)
        {
            q->next = p->next;
            delete p; p = NULL;
            return;
        }
    }
        cout << "There is no key : " << key << endl;
}
```

Josephus环游戏

```

```

C++大作业

题目二: 长整数、高精度运算

一、问题描述： 设计一个程序实现两个任意长的整数（包括正数和负数）、任意精度实数的算术运算。

二、提示： （1）用动态链表存贮数据，每结点含一个整型变量，表示若干位数。 （2）整数输入和输出按中国对于长整数的习惯表示，每3位1组，组间用逗号隔开。 （3）实现长整数的加、减运算。 （4）程序运行界面清晰实用。

```
#include<iostream>
#include<iomanip>
using namespace std;
struct node
{
    int num1;
    node* next;
};
struct list
{
    int num2;
    list* next;
};
struct answer
{
    int data;
```

```
    answer* next;
    answer* before;
};
int sign1 = 1, sign2 = 1, flag, key = 1, a = 0;
int create_num1(node*& head);
void  create_num2(list*& head);
void create_ans(answer*& head,node*head1,list *head2)
{
    answer* a, * p;
    a = new answer;
    a->next = NULL;
    p = a;
    node* n = head1;
    list* l = head2;
    a->data = n->num1 + l->num2;
    while (1)
    {
        if(n)n = n->next;
        if(l)l = l->next;
        p = a;
        a = new answer;
        a->next = p;
        p->before = a;
        if (n && l)
            a->data = n->num1 + l->num2;
        if (!n && l)
            a->data = l->num2;
        if (n && !l)
            a->data = n->num1;
        if (!n && !l)
        {
            head = p;
            break;
        }
    }
}
void change(answer*& head)
{
    while (!head->data && head->next)
    {
        head = head->next;
    }//去掉前面的0
    answer* p = head;
    answer* q = head;
    if (p->data < 0)
    {
        key = -1;
        while (p)
        {
            p->data *= (-1);
            p = p->next;
        }
    }
    while (q->next)
```

```
        q = q->next;//找到最后一个结点

    while (q!=head)
    {
        if (q->data < 0)
        {
            q->data += 1000;
            q->before->data -= 1;
        }
        if (q->data >= 1000)
        {
            q->data -= 1000;
            q->before->data += 1;
        }
        q = q->before;
    }
}
void showlist(answer* head)
{
    if (!head->next && head->data<1000) //如果答案小于1000，则直接输出；
    {
        cout << head->data << endl;
        return;
    }
    if (head->data >= 1000)
    {
        cout << (head->data) / 1000 << ",";
        head->data %= 1000;
        cout << setw(3) << setfill('0') << head->data ;
        if (head->next)cout << ',';
        else { cout << endl; return; }
        head = head->next; a = 1;
    }
    if (!a )
        cout << head->data << ',', head = head->next;
    while (head->next)
    {
        cout << setw(3) << setfill('0') << head->data << ',';
        head = head->next;
    }
    cout << setw(3) << setfill('0') << head->data << endl;


}
int main()
{
    cout << "本程序用于计算两个两个任意长的整数（包括正数和负数）、任意精度实数的加法或减法运算\n";
    cout << "格式要求：请先输入第一个整数，输入+/-，再输入第二个整数，最后输入= \n" << "（整数输入和输出按中国对于长整数的习惯表示，每3位1组，组间用逗号隔开。）\n";
    cout << "示例输入如： 123,456+123,456=  \n";
    cout << endl;
    char ask = 'Y';
    while (ask=='Y')
```

```
    {
        sign1 = 1, sign2 = 1, flag=0, key = 1,a=0;
        node* head1 = NULL;
        list* head2 = NULL;
        answer* head3 = NULL;
        flag = create_num1(head1);
        create_num2(head2);
        create_ans(head3, head1, head2);
        change(head3);
        if (key == -1)cout << '-';
        showlist(head3);
        cout << "\n\n是否想要继续输入进行计算 Y/N \n";
        cin >> ask;
    }
    return 0;
}
int create_num1(node*& head)
{
    node* s, * p;
    s = new node;
    cin >> s->num1;
    if (s->num1 < 0)sign1 = -1;
    s->next = NULL;
    char symbol;
    cin >> symbol;
    while (1)
    {
        if (symbol == '+')
        {
            head = s; return 1;
        }
        if (symbol == '-')
        {
            head = s; return -1;
        }
        p = s;
        s = new node;
        cin >> s->num1;
        s->num1 *= sign1;
        s->next = p;
        cin >> symbol;

    }
}
void  create_num2(list*& head)
{
    list* s, * p;
    s = new list;
    cin >> s->num2;
    if (s->num2 < 0)sign2 = -1;
    s->num2 *= flag;
    s->next = NULL;
    char symbol;
    cin >> symbol;
```

```
    while (1)
    {
        if (symbol != ',')
        {
            head = s;
            break;
        }
        p = s;
        s = new list;
        cin >> s->num2;
        s->num2 *= sign2;
        s->num2 *= flag;
        s->next = p;
        cin >> symbol;

    }
}
```

8分）本程序功能是把一个用拼音输入的名字自动生成6位数字串的密码。生成规则是把字母串的最后6位逆序，取每个字母小写的ASCII码值，其除以10的余数为该位的密码值。当输入名字的字母串不足6位，生成时以字母“z”补足。图4是程序的运行效果。请填写change函数的函数原型并编写函数。

EG：//请输入名字拼音，以#结束：

ZhongFuWa#

生成密码为：

797230#

```
#include<iostream.h>
#include<ctype.h>
struct link {char s; link * next;};
void inputName(link *& h);
void outLink(link *h);
void change(link *&hCode, link *h);   //change的函数原型

void main()
{ link *name=NULL, *code=NULL;
  cout<<"请输入名字拼音,以#结束:\n";
inputName(name);
  change(code,name);
  cout<<"生成密码为:\n";
outLink(code);
}

void inputName(link *& h)       //逆序存放字符串
{ link *p;
  p=new link;    cin>>(p->s);
  while((p->s>='a'&&p->s<='z'||p->s>='A'&&p->s<='Z')&&p->s!='#')
  { p->next=h;   h=p;
```

```
        p=new link;  cin>>(p->s);
    }
}

void change(link *&hCode, link *h)
{ char d;
  link *p=NULL;
  hCode=new link;
  hCode->next=NULL;
  p=hCode;
  d=h->s;
    for(int i=0;i<6;i++)
    { p->s=int(tolower(d))%10+'0';
      p->next=new link;
      p=p->next;
      p->s='#';
      p->next=NULL;
      if(h->next)
        {h=h->next; d=h->s;}
      else d='z';
    }
}

void outLink(link *h)
{ while(h)  {cout<<(h->s); h=h->next;}
  cout<<endl;
}
```

实现字母大小写转换

```
#include<cctype>

char s=tolower(p);  //全改为小写
char a=toupper(p);  //全改为大写

函数原型
int tolower(int c)           //改为小写
{
    if((c>='A)&&(c<='Z'))
       return c+('a'-'A');
    else  return c;
}
int toupper(int c)       //改为大写
{
    if((c>='a')&&(c<='z'))
      return c+'A'-'a';
    else return c;
}
```