```
#include<iostream>
conio.h cstdlib stdio
using namespace std;
int a,b,c;
for (a=1;a<=20;a++)
  for (b=1;b<=33;b++)
    for (c=3;c<=99;c++)
      if (5*a+3*b+c/3==100)
        if (a+b+c==100)
          if (c%3==0){
            cout<<"公鸡数为："<<a<<"母鸡数为："<<b<<"小鸡数为："<<c<<endl;
}

template<class Type>    //冒泡排序
void BubbleSort(Type* array,int start,int end){
    for(int i = 0;i < end - start;i++)
        for(int j = 0;j < end - start;j++){
            if(array[j] > array[j + 1])
                swap(array[j],array[j + 1]); } }
template<class Type>    //选择排序
void SelectionSort(Type* array,int start,int end){
    for(int i = 0;i < end - start;i++)
        for(int j = i + 1;j < end - start + 1;j++){
            if(array[i] > array[j])
                swap(array[i],array[j]); } }
template<class Type>    //插入排序
void InsertionSort(Type* array,int start,int end){
    for(int i = 1;i < end - start + 1;i++)
        for(int j = 0;j < i;j++){
            if(array[j] > array[i]){
                int temp = array[i];
                for(int k = i;k > j;k--)
                    array[k] = array[k - 1];
                array[j] = temp;          // 此时array[j]是最大的了          break;       } } }
template<class Type>    //快速排序
int Partition(Type *a,int start,int end){
    Type x = a[start];
    int i = start;
    for(int j = start + 1;j <= end;j++){
        if(a[j] <= x){
            i++;
            swap(a[i],a[j]); }
    }
    swap(a[i],a[start]); //把基准的值放在中间，则左边都小于他，右边都大于他        return i; }
template<class Type>
void QuickSort(Type *array,int start,int end){
    if(start < end){
        int q = Partition(array,start,end); //分割成两
        QuickSort(array,start,q - 1);
        QuickSort(array,q + 1,end);          }    }
template<class Type>    //希尔排序
void ShellPass(Type* array,int start,int end,int d){
    for(int i = 0;i < d;i++){
        for(int j = start + i + d;j < end - start + 1;j += d)
            for(int k = start + i;k < j;k += d){
                if(array[k] > array[j]){
                    Type temp = array[j];
                    for(int l = j;l > k;l -= d)
                        array[l] = array[l - d];
                    array[k] = temp; } } } } }
template<class Type>    //希尔排序
void ShellSort(Type* array,int start,int end){
    int d = 10;
    while(d > 0){
        d = (d + 1) / 2;
        ShellPass(array,start,end,d);
        if(d == 1)
            break;       }    }
int main(){       //海盗
  int pirate[30];
  int i,j,survived;
  for(i=0;i<30;i++)
    pirate[i]=0;
  i=0;j=0;
  for(survived=30;survived>1;){
    if(pirate[i]==0){
      j++;
      if(j%7==0){
        pirate[i]=1;
        cout<<"No."<<i+1<<" private jump"<<endl;
        survived-=1;
      }    }    i=(i+1)%30; }
  for(i=0;i<30;i++)
    if(pirate[i]==0)
      cout<<"No."<<i+1<<" private survive"<<endl;
      return 0;}
int f(int x)          //兔子生兔子
{if (x == 1 || x == 2)
        return 1;
else
        return f(x - 1) + f(x - 2);
} int main(void)
{         int a;                 scanf("%d", &a);
          a = f(a);
          printf("\n%d", a);
          return 0;}
int BinarySearch(int *array, int aSize, int key)/不递归
{     if ( array == NULL || aSize == 0 )    //二分查找
          return -1;
    int low = 0;
    int high = aSize - 1;
    int mid = 0;
    while ( low <= high )
    {     mid = (low + high )/2;
        if ( array[mid] < key)
            low = mid + 1;
         else if ( array[mid] > key )
            high = mid - 1;
          else
              return mid;    }
     return -1;    }

int BinarySearchRecursive(int *array, int low, int high, int key)    //递归//二分查找
{    if ( low > high )
          return -1;
     int mid = ( low + high )/2;
     if ( array[mid] == key )
          return mid;
     else if ( array[mid] < key )
          return BinarySearchRecursive(array, mid+1, high, key);
     else
          return BinarySearchRecursive(array, low, mid-1, key);    }
int main()
{    int array[10];
     for (int i=0; i<10; i++)
          array[i] = i;
     cout<<"No recursive:"<<endl;
     cout<<"position:"<<BinarySearch(array, 10, 6)<<endl;
     cout<<"recursive:"<<endl;
cout<<"position:"<<BinarySearchRecursive(array, 0, 9, 6)<<endl;
     return 0; }

# include <iomanip>//二分法解方程
double func(double x)
{return (x*x*x - 6 * x - 3);}
void root(double a, double b, double e, double *pResult)
{          while (b - a >= e)
           {          *pResult = (a + b) / 2;
                      if (func(*pResult) * func(a) < 0)
                      {b = *pResult;}
                    else if (func(*pResult) * func(a) > 0)
                      {a = *pResult;}
                      else
                      {break;}}}
int main()
{double e = (double)0.00000001;
           double a = (double)2;
           double b = (double)3;
           double Result;
           root(a, b, e, &Result);
           cout << setiosflags(ios::fixed);
           cout << "所求实根为：" << setprecision(9) << Result << endl;
           return 0;}
//复化梯形公式求定积分
#include<iostream> <cmath> std;
int main()
{double up,down,a,b,c,n= 100; //积分区间分为 n 份
   cout << "依次输入积分上限 up、积分下限 down 和函数参数 a、b、c 的值：" << endl;
   cin >> up >> down >> a >> b >> c;    cout << endl;
   double h = (up -down) / n; //迭代步长
   double result, fx=0, fa, fb;     //result 积分最终结果
   double x = 0;
   for (int i = 1; i <= n - 1;i++)
   {    fx += c / (1 + a*pow(x, b)*sqrt(x));
      x += (up - down) / n; }
   fa = c / (1 + a*pow(a, b)*sqrt(a));
   fb = c / (1 + a*pow(b, b)*sqrt(b));
   result = (h / 2)*(fa+2*fx+fb);
   cout << "步长 h:" << h << endl;
   cout << "f(a):" << fa << endl;
   cout << "f(b):" << fb << endl;
   cout <<"result:"<< result << endl;
   return 0;    }
#include <iostream> <cmath> std;
double f(double x)
{ return 1.0/(1+x*x);}
double SnSum(double a, double b, double n)
{ double sum = 0;
 for(int i=1; i<=n; i++){
   sum += 2*f(1.0/n*i)+4*f((1.0/n*(i-1)+1.0/n*i)/2); }
 return sum;}
double Sn(double a, double b, int n)
{ double h = (b-a)/n;
 return h/6*(f(a)+f(b)+SnSum(a,b,n));}
int main()
{ double n;
 cout<<"等分次数 n:"; cin>>n;
 cout<<"计算结果:"<<Sn(a,b, n)<<endl;}
//牛顿迭代法求一元三次方程组根
#include <stdio.h>
#include<math.h>
double solut(double a,double b,double c,double d)
{      double x=1,x1=2,f,f1;
     while(fabs(x1-x)>=0.00000001)
     {     f=((a*x+b)*x+c)*x+d;
           f1=(3*a*x+2*b)*x+c;
           x=x1;
           x1=x-f/f1;
     }     return x1;}
int main()
{ double solut(double ,double ,double ,double );
     double a,b,c,d;
     scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
     printf("%.2f",solut(a,b,c,d));;
    return 0;     }
class complex{       //复数类重载有关
public:
     complex(double real = 0.0, double imag = 0.0): m_real(real), m_imag(imag){ };
public:
     friend complex operator+(const complex & A, const complex & B);
     friend complex operator-(const complex & A, const complex & B);
     friend complex operator*(const complex & A, const complex & B);
     friend complex operator/(const complex & A, const complex & B);
     friend istream & operator>>(istream & in, complex & A);
     friend ostream & operator<<(ostream & out, complex & A);
private:
     double m_real;    //实部
     double m_imag;    //虚部};
//重载加法运算符 complex operator+(const complex & A, const complex &B){
     complex C;
     C.m_real = A.m_real + B.m_real;
     C.m_imag = A.m_imag + B.m_imag;
     return C;}
//重载减法运算符 complex operator-(const complex & A, const complex &B){
     complex C;
     C.m_real = A.m_real - B.m_real;
     C.m_imag = A.m_imag - B.m_imag;
     return C;}
//重载乘法运算符 complex operator*(const complex & A, const complex &B){
     complex C;
     C.m_real = A.m_real * B.m_real - A.m_imag * B.m_imag;
     C.m_imag = A.m_imag * B.m_real + A.m_real * B.m_imag;
     return C;}
//重载除法运算符 complex operator/(const complex & A, const complex & B){
     complex C;
     double square = A.m_real * A.m_real + A.m_imag * A.m_imag;
     C.m_real = (A.m_real * B.m_real + A.m_imag * B.m_imag)/square;
     C.m_imag = (A.m_imag * B.m_real - A.m_real * B.m_imag)/square;
     return C;}
//重载输入运算符 istream & operator>>(istream & in, complex & A){
     in >> A.m_real >> A.m_imag;
     return in;}
//重载输出运算符 ostream & operator<<(ostream & out, complex & A){
     out << A.m_real <<" + "<< A.m_imag <<" i ";;
     return out;}
int main(){
     complex c1, c2, c3;
     cin>>c1>>c2;c3 = c1 + c2;
     cout<<"c1 + c2 = "<<c3<<endl;
     c3 = c1 - c2;
     cout<<"c1 - c2 = "<<c3<<endl;
     c3 = c1 * c2;
     cout<<"c1 * c2 = "<<c3<<endl;
     c3 = c1 / c2;
     cout<<"c1 / c2 = "<<c3<<endl;
     return 0;}
#include<Cmath>       //////重要
fabs(x)   ceil(x)  不小于 x 的最小整数
floor(x)  不大于 x 的最大帧数  log(x)
log10(x)   exp(x)       //e^x
//万扣灭灯
int t[6][8], w[6][8];
void fuck();
bool u();
int main(void)
{          //输入数组
           for (int i = 1;i < 6;++i)
                      for (int j = 1;j < 7;++j)
                                 cin >> t[i][j];
           cout << endl;
           fuck();
           for (int i = 1;i < 6;++i)
           {          for (int j = 1;j < 7;++j)
                                 cout << w[i][j]<<" ";
                      cout << endl;}
getch();
           return 0;
}void fuck()
{          for (int a = 0;a < 2;++a)
              for (int b = 0;b < 2;++b)
                  for (int c = 0;c < 2;++c)
                      for (int d = 0;d < 2;++d)
                         for (int e = 0;e < 2;++e)
                            for (int f = 0;f < 2;++f)
           {w[1][1] = a;       w[1][2] = b;
           w[1][3] = c;        w[1][4] = d;
           w[1][5] = e;        w[1][6] = f;
if (u())     {return;}
bool u()
{           for (int r = 1;r <= 4;++r)
                      for (int k = 1;k <= 6;++k)
                                 w[r + 1][k] = (t[r][k] + w[r][k] + w[r - 1][k] + w[r][k - 1]    + w[r][k + 1]) % 2;
           for (int r = 1;r <= 6;++r)
                      if ((w[5][r] + w[5][r - 1] + w[5][r + 1] + w[4][r]) % 2 != t[5][r])
                                 return false;
           return true;}
//链表 template<class T>
class slistNode
{
     public:
     slistNode(){next=NULL;}
     T data;//值
     slistNode* next;//指向下一个节点的指针};
template<class T>
class myslist
{     private:
      unsigned int listlength;
      slistNode<T>* node;//临时节点
      slistNode<T>* lastnode;//头结点
      slistNode<T>* headnode;//尾节点
      public:
            myslist();//初始化
            unsigned int length();//链表元素的个数
            void add(T x);//表尾添加元素
            void traversal();//遍历整个链表并打印
            bool isEmpty();//判断链表是否为空
            slistNode<T>* find(T x);//查找值为 x 的节点,返回节点的地址,找不到返回 NULL
            void Delete(T x);//删除值为 x 的节点
            void insert(T x,slistNode<T>* p);//在 p 节点后插入值为 x 的节点
            void insertHead(T x);//链表头部插入节点};
template<class T>
myslist<T>::myslist()
{     node=NULL;
      lastnode=NULL;
      headnode=NULL;
      listlength=0;}
template<class T>
inline unsigned int myslist<T>::length(){return listlength;}
template<class T>
void    myslist<T>::add(T x)
{     node=new slistNode<T>();//申请一个新的节点
      node->data=x;//新节点赋值为 x
      if(lastnode==NULL)//如果没有尾节点则链表为空,node 既为头结点,又是尾节点
      {     headnode=node;
            lastnode=node;}
      else//如果链表非空
      {lastnode->next=node;//node 既为尾节点的下一个节点
       lastnode=node;//node 变成了尾节点,把尾节点赋值为 node}
      ++listlength;//元素个数+1}
template<class T>
void    myslist<T>::traversal()
{node=headnode;//用临时节点指向头结点
   while(node!=NULL)//遍历链表并输出
      {cout<<node->data<<ends;
       node=node->next;}
      cout<<endl;}
template<class T>
bool    myslist<T>::isEmpty()
{return listlength==0;}
template<class T>
slistNode<T>* myslist<T>::find(T x)
{node=headnode;//用临时节点指向头结点
 while(node!=NULL&&node->data!=x)//遍历链表,遇到值相同的节点跳出
 {node=node->next;}
   return node;//返回找到的节点的地址,如果没有找到则返回 NULL}
template<class T>
void    myslist<T>::Delete(T x)
{slistNode<T>* temp=headnode;//申请一个临时节点指向头节点
 if(temp==NULL) return;//如果头节点为空,则该链表无元素,直接返回
 if(temp->data==x)//如果头节点的值为要删除的值,则删除投节点
      {headnode=temp->next;//把头节点指向头节点的下一个节点
        if(temp->next==NULL)   lastnode=NULL;//如果链表中只有一个节点,删除之后就没有节点了,把尾节点置为空
            delete(temp);//删除头节点
            return;}
while(temp->next!=NULL&&temp->next->data!=x)//遍历链表找到第一个值与 x 相等的节点,temp 表示这个节点的上一个节点
      {temp=temp->next;}
      if(temp->next==NULL) return;//没有找到则返回
      if(temp->next==lastnode)//如找到的时候尾节点
      { lastnode=temp;//把尾节点指向他的上一个节点
            delete(temp->next);//删除尾节点
            temp->next=NULL; }
      else//如果不是尾节点
      { node=temp->next;//用临时节点 node 指向要删除的节点
         temp->next=node->next;//要删除的节点的上一个节点指向要删除节点的下一个节点
            delete(node);//删除节点
            node=NULL; }}
template<class T>
void    myslist<T>::insert(T x,slistNode<T>* p)
{     if(p==NULL) return;
      node=new slistNode<T>();//申请一个新的空间
      node->data=x;
      node->next=p->next;
      p->next=node;
      if(node->next==NULL)//如果 node 为尾节点
      lastnode=node;}
template<class T>
void    myslist<T>::insertHead(T x)
{     node=new slistNode<T>();
      node->data=x;
      node->next=headnode;
      headnode=node;}
//链表结束
string str="abc";
char *p=str.data();
```

1、如果要将 string 转换为 char*，可以使用 string

提供的函数 c_str() ， 或是函数 data()，data 除了返回字符串内容外，不附加结束符'\0'，而 c_str()返回一个以‘\0’结尾的字符数组。

2、const char *c_str();

c_str()函数返回一个指向正规 C 字符串的指针，内容与本 string 串相同..

注意：一定要使用 strcpy()函数 等来操作方法 c_str()返回的指针：

```
char c[20];
string s="1234";
strcpy(c,s.c_str());
```

再举个例子

c_str() 以 char* 形式传回 string 内含字符串

如果一个函数要求 char*参数，可使用 c_str()方法：

```
string s = "Hello World!";
printf("%s",s.c_str()); //输出 "Hello World!"
```

char *转换成 string 可以直接赋值。

```
string s; char *p = "adghrtyh"; s = p;
```

深拷贝浅拷贝区别

简单的来说就是，在有指针的情况下，浅拷贝只是增加了一个指针指向已经存在的内存，而深拷贝就是增加一个指针并且申请一个新的内存，使这个增加的指针指向这个新的内存，采用深拷贝的情况下，释放内存的时候就不会出现在浅拷贝时重复释放同一内存的错误！

```
class string
{ char *m_str;
  public:
        string(char *s)
        {m_str=s;}
        string() {};
String & operator=(const string s)//浅拷贝
{m_str=s.m_str;
return *this}};
int main()
{string s1("abc"),s2;
s2=s1;
cout<<s2.m_str;}
string&operator=(const string&s)//深拷贝
{       if(strlen(m_str)!=strlen(s.m_str))
                m_str=new char[strlen(s.m_str)+1];
        if(*this!=s)
                strcopy(m_str,s.m_str);
return *this;}
//类封装
#include <iostream>    #include <string>
/**  定义类：Student
 *  数据成员：m_strName
 *  无参构造函数：Student()
 *  有参构造函数：Student(string _name)
 *  拷贝构造函数：Student(const Student& stu)
 *  析构函数：~Student()
 *  数据成员函数：setName(string _name)、getName() */
class Student
{       public:
      Student() { } //无参构造函数
      Student(string _name) { }有参构造函数：
      Student(const Student& stu)    { } 拷贝构造函数
      ~Student() {}析构函数
      void setName(string _name)
      {m_strName=_name;     }
      string getName()
      {return m_strName;    }
      private:
      string m_strName; };
int main(void)
{   //  通过 new 方式实例化对象*stu
      Student *stu = new Student();
      //  更改对象的数据成员为“慕课网”
      stu->setName("慕课网");
      //  打印对象的数据成员
     cout<<stu->getName()<<endl;
     delete stu;
      return 0;    }
```

动态内存：

```
double* pvalue = new double[?];
delete []pvalue; //  释放内存
```

字符串互转

```
double atof(const char *str)
{double s = 0.0;//每一位数
double d = 10.0;//十进制
bool flag = false;//标记是否为正数
while (*str == ' ')
{str++;}
if (*str == '-')//记录数字正负
{flag = true;
str++;}
if (*str < '0'&&*str > '9')//如果一开始非数字则退出，返回0.0
return 0;
while (*str >= '0'&&*str <= '9'&&*str != '.')//计算小数点前整数部分
{s = s * 10.0 + *str - '0';
str++;}
if (*str == '.')//以后为小数部分
str++;
while (*str >= '0'&&*str <= '9')//计算小数部分
{s = s + (*str - '0') / d;
d *= 10.0;
str++;}
return s * (flag ? -1.0 : 1.0);}
string to_String(double num)
{char _str[20];
sprintf_s(_str, 20, "%f", num);
return _str;}
```

模板堆栈类表达式运算

```
#define _CRT_SECURE_NO_WARNINGS    //解决139行strtok和strtok_s参数引用过少
#include <cstdlib>
#define MAXSIZE 100
struct Stack//定义一个顺序存储栈
{double data[MAXSIZE];
int top;//栈顶指针  };
template<class T>
class Caculation
{public:
int CreateStack(Stack *s);
int stack_empty(Stack s);
int push(Stack *s, T x);
int pull(Stack *s, T *x);
int quit();
int CreateExpression(char *inorder);
int TransmitExpression(char *inorder, char *postorder);
int   EvaluateExpression(char *postorder, T *result);
void Begin();};
template<class T>
int Caculation<T>::CreateStack(Stack *s)//栈顶指针初始化，建立一个空栈。
{s->top = -1;return 1;}
template<class T>
int Caculation<T>::stack_empty(Stack s)//判断栈是否为空。
{if (s.top == -1) return 1;
else     return 0;}
template<class T>
int Caculation<T>::push(Stack *s, T x)//入栈
{if (s->top == MAXSIZE - 1)//栈满
return 0;
else      s->top++;
s->data[s->top] = x;//将x推入栈
return 1;}
template<class T>
int Caculation<T>::pull(Stack *s, T *x)//出栈
{if (s->top == -1)//栈空     return 0;
else      *x = s->data[s->top];//栈顶元素弹出赋予x
s->top--;   return 1;}
template<class T>
int Caculation<T>::quit()
{exit(0);   return 0;}
template<class T>
int Caculation<T>::CreateExpression(char *inorder)//输入中缀表达式
{cout << "请输入表达式：" << endl;
cin >> inorder;   return 0;}
template<class T>
int Caculation<T>::TransmitExpression(char *inorder, char *postorder)//中缀表达式转为后缀表达式
{ Stack str;
double e = 0;//进行出栈入栈操作
int i = 0, j = 0;//分别进行循环数组的下标,i为中缀下标，j为后缀下标。
int flag = 0;
if (CreateStack(&str) != 1)     //判断栈是否为空
return 0;
while (inorder[i] != '\0')     //说明栈中有元素。
{while (inorder[i] >= '0' && inorder[i] <= '9')//若是数字则输出
{if (flag) //考虑负数的情况
{postorder[j++] = '-';}
postorder[j++] = inorder[i];//让存放后缀表达式的数组存放字符
i++;
if (inorder[i]<'0' || inorder[i]>'9') //判断是否为操作符，如果是，让后缀表达式中存放一个' '
postorder[j++] = ' ';}
if (inorder[i] == ')')           //如果是关于括号的符号，则进行出栈{pull(&str, &e);
while (e != '(')
{postorder[j++] = e;   postorder[j++] = ' ';
pull(&str, &e);}}
else if (inorder[i] == '+' || inorder[i] == '-') //对于同运算级的+和-操作
{if (inorder[i] == '-' && (i == 0 || (i != 0 && (inorder[i - 1]<'0' || inorder[i - 1]>'9'))))   //当'-'号处于第一位，或前面是符号时，为负号标志
flag = 1;
else if (stack_empty(str))//如果栈空
push(&str, inorder[i]);
else
{while (!stack_empty(str) && e != '(')
{pull(&str, &e);
if (e == '(')   //优先级最大，比较以后入栈
push(&str, e);
else
{postorder[j++] = e;   //进行后缀表达式的添加
postorder[j++] = ' '; //最后添加‘ ’字符分隔开   }}
push(&str, inorder[i]);}}
else if (inorder[i] == '*' || inorder[i] == '/' || inorder[i] == '(')//对乘除以及左括号的进行入栈
push(&str, inorder[i]);
else if (inorder[i] == NULL)//如中序当前读取位为空
break;
else     return 0;
i++;}
while (!stack_empty(str))//栈非空
{pull(&str, &e);
postorder[j++] = e;
postorder[j++] = ' ';}
return 1;}
template<class T>
int Caculation<T>::EvaluateExpression(char *postorder, T *result)//计算结果
{Stack s; char *op; //存放后缀表达式中的每个因数或运算符
char *buf = postorder; //声明buf，strtok函数的需要
double d;double e, f;
if (CreateStack(&s) != 1)   return 0;
while ((op=strtok(buf, " ")) != NULL)字符串分割函数
{buf = NULL;                    //把指针置空
switch (op[0])
{case '+':
pull(&s, &d);pull(&s, &e);
f = d + e;push(&s, f);break;
case '-':
if (op[1] >= '0' && op[1] <= '9')
{d = atof(op);push(&s, d);break;}
pull(&s, &d);pull(&s, &e);
f = e - d;push(&s, f);break;
case '*':
pull(&s, &d);pull(&s, &e);
f = e * d;push(&s, f);break;
case '/':
pull(&s, &d);pull(&s, &e);
f = e / d;push(&s, f);break;
default: //考虑数字的情况，进行atof函数进行转化
d = atof(op);push(&s, d); //进行压栈   break;}}
pull(&s, result);return 0;}
template<class T>
void Caculation<T>::Begin()
{char inorder[MAXSIZE] = { 0 };//中缀表达式
char postorder[MAXSIZE] = { 0 };//后缀表达式
double result;
CreateExpression(inorder);
TransmitExpression(inorder, postorder);
cout << "转化后的后缀表达式是：" << endl;
cout << postorder << endl;
EvaluateExpression(postorder, &result);
cout << "计算结果：" << endl;
cout << result << endl;}
int Continue()
{char wait;
puts("\n\n继续吗？是请按Enter键，否请按任意键");
getchar();wait = getchar();
if (wait != '\n')   return 0;
else return 1;}
```

10 进制转 8 进制

```
int * Stack::Single(int number)
{          int *single_eight;
           int m;//余数
           int n = 0;
           int fake = number;
           while (fake != 0)
           {     fake /= 8;    n++;}//求位数
           single_eight = (int*)malloc(n + 1);
           int s = 1;
           while (number != 0)
           {           m = number % 8;
                       single_eight[s] = m;
                       number /= 8;
                       s++;}
           single_eight[0] = n;
           return single_eight;        }
```

圆周率

```
#include<iostream>include<time.h>
#define N 30000;
class Point {
private:
           double x;
           double y;
public:
void FillPoints(int n0)
{for (int i = 0; i < n0; ++i)
       {(this+i)->x = 1.0 * rand() / RAND_MAX;
       (this+i)->y = 1.0 * rand() / RAND_MAX;}}
int CacuInPoints(int n, double r, int n0)
{double r0;
for (int i = 0; i < n0; ++i)
{   r0 = ((this + i)->x) * ((this + i)->x) + ((this + i)->y) * ((this + i)->y);
if (r0 < +r * r) n++;}
return n;}
double CacuPI(int n0, int n, double r)
{double PI;
PI = 4 * (1.0 * n / n0) / (r * r);
return PI;}                          };
void main()
{           int n0;//总点数double r;//圆半径
           puts("请输入圆半径(小于或等于1.0)：");
           cin >> r;
           double PIVAL;//最终PI值
           Point pts[N] = {};//保存n0个点的坐标
           srand(time(NULL));
           for (int i = 0; i < 10; ++i)
           { int n = 0;//落入圆内的点数
           n0 = 10000 + rand() % 20000;
           cout << "总点数：" << n0 << endl;
           (*pts).FillPoints(n0);
           n = (*pts).CacuInPoints(n, r, n0);
           cout << "圆内点数：" << n << endl;
           PIVAL = (*pts).CacuPI(n0, n, r);
           cout << "PI的值：" << PIVAL << endl;}
           system("pause");}
```

查词频

```
#include<iostream>#include<string>
#include<fstream>#define N 1000
class CString
{public:
           void choose(char ** single, int words_num)
{          string input;
           int feak_num = 0;
           int num = 0;
           puts("请输入你所需查找频数的单词");
           cin >> input;
           cout << input.length() << endl;
           for (int i = 0; i < words_num; ++i)
           {   for (int j = 0; j < input.length(); ++j)
               { if (single[i][j] == input[j])
                       feak_num++;
                 else   break; }
              if (feak_num == input.length())
               {num++;feak_num = 0;}
              else      feak_num = 0;       }
              puts("查找结果为：");
              cout << input << " : " << num << endl;}
void user_continue(char **single, int words_num)
//参照上面
void punctuation(char *content)//标点符号转为空格
{for (int i = 0; content[i]; ++i)
   {if (content[i] < 'A' || content[i] > 'Z')
           {if (content[i] < 'a' || content[i] > 'z')
           content[i] = ' ';}}         }
void words_num(char *content)
{char seps[] = " ";
char *taken1 = NULL;
char *taken2 = NULL;
char *next_taken = NULL;
punctuation(content);
int i = 0;    char *single[N] = {};
taken1 = strtok_s(content, seps, &next_taken);
while (taken1 != NULL)
{    single[i] = taken1;
taken1 = strtok_s(NULL, seps, &next_taken);
i++;}
user_continue(single, i);}
void File_num(char *content)
{int i, j;    int kind[128] = {};
puts("字母频数：");
for (i = 0; content[i]; ++i)
   {if (content[i] >= 'A' && content[i] <= 'Z')
           content[i] += 32;//将大写转换为小写}
for (i = 0; content[i]; ++i)
{ if (content[i] >= 'a' && content[i] <= 'z')
       kind[content[i]]++;}
for (j = 0; j < 128; j++)
{ if (kind[j])
   {cout << (char)j << " : " << kind[j] << endl;}}    }
char *file_read(ifstream &File, char *content)
{if (File)
{while (!File.eof())
{ File.read(content, N);
   char *copy = content;
   cout << "读入文章为：\n" << content << endl;
   return copy;   } }
else cerr << "wrong" << endl;
File.close();}     };
void main()
{CString Begin;
char content[N] = {};
ifstream File;     File.open("file.txt", ios::in);
char *c_content = Begin.file_read(File, content);
Begin.File_num(c_content);//数字母个数
Begin.words_num(c_content);//数单词个数
system("pause");}
```

虚函数继承

```
class GrandFather
 {public:
GrandFather():i_G(5)
           {cout<<"GrandFather() is
              called!"<<endl;}
virtual ~GrandFather()
            {cout<<"~GrandFather() is called!"<<endl;}
public:
     virtual void Test()
     {cout<<"GrandFather::Test() is called!"<<endl;}
private:   int i_G;   };
class Father: virtual public GrandFather //虚拟继承              {public:
   Father():i_F(7)
         {cout<<"Father() is called!"<<endl;};
   virtual ~Father()
         {cout<<"~Father() is called!"<<endl;}
 public:
        virtual void Test()
         {cout<<"Father::Test() is called!"<<endl;}
private:    int i_F;     };
class Uncle: virtual public GrandFather//虚拟继承
 { public:
        Uncle():i_U(3)
         {cout<<"Uncle is called!"<<endl;}
       virtual ~Uncle()
         {cout<<"~Uncle   is called!"<<endl;}
   public:
         virtual void Test()
         {cout<<"Uncle ::Test() is called!"<<endl;}
   private:   int i_U; };
class Son:public Father,public Uncle
{public:
     Son():i_S(9)
     {cout<<"Son is called!"<<endl;};
     virtual ~Son()
     {cout<<"~Son   is called!"<<endl;}
public:
      virtual void Test()
      {cout<<"Son ::Test() is called!"<<endl;}
private:int i_S;   };
int main(void)
 {Son p;     p.Test();
cout<<sizeof(Son)<<endl;
cout<<sizeof(Father)<<endl;
cout<<sizeof(GrandFather)<<endl;
return 0;   }
```

运行情况：

```
GrandFather() is called!
Father() is called!
   Uncle is called!
   Son is called!
   Son ::Test() is called!
   32     20     8
   ~Son is called!
   ~Uncle is called!
   ~Father() is called!
   ~GrandFather() is called!
```