# C/C++

## C

### 基本数据类型

|  |  |
| --- | --- |
| **类型** | **符号** |
| **char** | %c |
| **unsigned char ( 0～255)** | %c |
| **short** | %hd |
| **unsigned short** | %hu |
| **int** | %d |
| **unsigned int** | %u |
| **long** | %ld |
| **unsigned long** | %lu |
| **long long** | %lld |
| **float** | %f |
| **double** | %lf |

整数格式：

%x 十六进制 %#x, %#X 带符号 uint64对应%llx

%o 八进制 %#o带符号 010表示数字8

%nd 位宽为n，高位无数时空格 uint64对应%lld

%0nd 位宽为n,高位无数时补0

%nd右对齐

%-nd左对齐

浮点数格式：

%.nf 保留小数点后n位【四舍五入】(.nf不能用于输入)

%e %E代表科学计数法格式

%g %G 不显示无意义的0，根据数据自动选择使用科学计数法

%p 地址(指针)格式

%zd size\_t输出类型

%%代表%

Unicode字符

wchar\_t c='x';

wchar\_t\* p=L"abcde";

isalpha(c)可以判断字符是否是字母（不区分大小写）

===========字面值常量

十进制： 20

八进制： 024

十六进制： 0x14

字面值整数常量类型默认为int或long（其精度类型决定于字面值大小）。通过增加后缀，可以将其转换为某种精度。

长整形：1L (long) 8UL (unsigned long)

浮点型：3.14f 3.14L（浮点型默认是double，后面加f代表单精度型，后面加L表示扩展精度型）

#include <inttypes.h>

uint16\_t val = 12;

printf("val = %" PRId16 "\n",val);

#include <float.h>

FLT\_DIG float支持的有效位个数

FLT\_MIN float最小值

&取地址 \*从地址间接访问

‘’字符 “”字符串

转译字符： \n , \t, \’, \”

左值：可改变的值

右值：不可改变的值

（左值）a=5（右值）

0代表假，其它数代表真

### 运算符

不同数据类型进行计算时，以数据精度高的为主。

如(int)x/(float)y结果为float型

#### 算术

+,-,\*,/,%,++,--

++a 先加后使用

a++ 先使用后加

+=，-=，\*=，/=

a+=1等价于a=a+1；

#### 关系

>,<,==,>=,<=,!=

#### 逻辑

&&（与） 有短路特性，前一个表达式为假，则不进行第二个表达式计算

||（或） 前一个为真，则不进行第二个表达式计算

!(非)

#### 位运算

~按位求反 一个二进制数

&按位与

取出某位(判断某位是否存在) if(a&x)

1001 1001

0010 得到0 0001 得到1

删除某位 a&=~x;

1001 （删除末尾1）使用1的反码1110

1110 得到1000

| 按位或 合成数 a|=x;

1010

0101 得到1111

^ 异或 不同时为1时得到1 a^=x;

1001 1010

0011 得到1010 0011 得到1001

#### 移位

移位运算的本质是进行了乘法操作，向左移动一位，就是乘一次2。

<< 左移

>> 右移 二进制

Exp：（9）1001<<2 得到 100100（36）

1001>>2 得到 0010（2）

#### sizeof

sizeof(int) 类型必须加括号

sizeof a 变量可以不加括号

#### 三目运算符

a>b?a:b

### 语句结构

if()…… 如果后面多条语句为一个整体，用{}括起来

if()……else……

if()……else if()……else else与最近的if配对

for(i=0;i<n;i++){……} 满足条件时，执行语句，然后i++

break; 跳出循环

continue; 取消本次循环，进行下一次循环

while(i<10){……i++;……}

do(……)while();当语句中有continue，则会停止下面语句，去执行while()。

switch（整型）

｛

case 常量1：……；break; 不加break则会在满足case条件后执行下面的语句

case 常量2：……；break;

default : ……；

｝

switch语句内部不能定义变量，需要使用变量只能在之前定义。

如果switch内要定义变量，需要使用块语句。

case 1:{ int a = 10; break; }

并行，语句之间相互顺序不影响

循环中使用goto可直接跳出循环

goto p； 可直接跳到p的位置

p: ......;

\ 可以进行折行

"abc\

def"

### 函数

int ab （int a， int \* p）｛…… return x；｝

返回值类型 函数名 形参 返回值

返回值为void类型时，只能用return；不可返回值。

主函数默认返回类型为int，可不写返回值类型和return，主函数的参数为argc，argv[],分别为命令行的字符串数量和这些字符串的指针组成的数组。

函数实现前至少要声明(若位置在调用前，则可不声明),声明后位置可任意放,声明时不可省略参数类型，可省略形参名

‘,’用来隔开参数,‘;’用来隔开语句，‘{}’用来划分语句块

函数是以形参的初始化开始的，形参就是在该函数中定义的局部变量，靠调用时传入的实参进行初始化。(传入参数为int aa，函数内产生变量int a = aa；传入为int \*pp，函数内产生变量int \*p = pp)，在一片新的内存空间运行，结束时释放这片内存空间。所以在这片空间内运行的普通形参不能改变外部实参。在使用指针时，形参为指针，通过指针的\*运算访问到了外部实参进行操作，所以可以改变外部实参。

### 变量

声明变量时不分配空间

使用变量时，是把某块内存中存储的数字按照变量类型进行解释，如int就是直接把4字节的内存上存储的数拿出来用。

不能返回自动变量的地址，自动变量在函数中使用完后会自动释放

全局变量未初始化是0,局部变量未初始化是随机数

auto int x = 10;

修饰符 数据类型 变量名 初始值

auto：自动变量,普通局部变量都是auto,一般省略

static：被限制访问范围的全局变量。在全局区定义则只能在定义位置的cpp文件中访问（使用extern不能访问）。在函数中定义则只能在该函数内使用。在类内定义则只能在类及类的对象中访问。

register：被频繁使用的变量，建议编译器将其保存到寄存器中

volatile：变量可能会被未知的因素改变，建议编译器不对这个变量进行优化

使用场合：多进程，多线程，共享的数据

硬件寄存器

一个中断服务子程序中使用的非自动变量

const int a=10; // a为常量，不可改变，只能在初始化时候赋值。

要在不同文件中访问全局变量：

int x=10; //定义变量的文件

extern int x;// 告诉编译器，该变量在其他文件中定义

对于const变量

extern const int x=10; //定义变量的文件

extern const int x;

在头文件中定义变量时，只能是static或者const

### 数组

#### 定义

a[]={…} a[]初始化后会跟据数据数量自动分配内存

a[10]

a[][10]={…}

a[10][10]

int a[n];使用时下标从0开始到n-1一共n个元素

sizeof(a)／sizeof(int) 可计算出数组中元素的个数,在函数调用中，退化为指针，无法计算出数组元素个数。

#### 初始化

==========一维数组

a[10]={1,2,3} 数量不足时补0

a[10]={[8]=5,[5]=3} 不需要考虑顺序

a[10]={0} 定义时进行清零

使用memset(a,0,sizeof(a))也可将数组清零

==========二维数组

a[m][n]={{n个数据},{…}…}

#### 数组名

一维数组：a+3等价于a[3]的指针

二维数组：a+3等价于a[3][0]的指针，\*(\*(a+i)+j)相当于a[i][j]

数组名相当于一个地址，并不真实存在，二维数组名是一个特殊的二级指针(行指针)

数组名不可以使用++运算，因为数组名是常量

可将数组名当做一个指针传给函数

int a[10];

int \*p=a时,p移动1位为int大小，即为数组下一元素的位置

(&a+1) &a移动一位的大小为sizeof(a)的大小

a与&a值相同，意义不一样

p=a或者p=&a[0],p可以像a方式一样使用

### 指针

指针存储的是内存地址，而内存地址是一个数字，所以\*p等价于

\*0xffffff00（32位系统的指针）

int a; int \*p;

a等价于\*p, p为一个int型的指针, \*p作为一个整体时为int型

可以使用typedef int\* INT 则INT p ; p为指针

typedef int a; (a指代int类型) int a；（a表示变量）

指针移动一位与指针类型相关

int a=10;

char \*p=&a; 编译会出现警告

p+1移动一个字节(char)

在32位系统中，指针占四个字节，其中存放着指向的变量的地址值

指针包含:

自身所在的地址值

自身变量名

存储的变量的地址值

指针的类型

指向的数据的类型

void \*p

通用指针，不可移动或取值，可赋值给任意类型的指针。c中可直接赋值，c++中需要类型转换

int \*p

一级指针p+1代表移动指针类型一块(一个单位)的大小，本质为一级指针中所存地址值发生变化。对\*p操作即为对p指向的变量的操作

int \*\*pp

二级指针存着一级指针的地址,pp+1代表移动四个字节(32位系统中指针四个字节，在一级指针连续存储时+1才有意义)。因为指向的是一级指针,对\*pp操作即为对pp指向的一级指针进行操作.对\*\*p操作即为对pp指向的一级指针指向的变量进行操作

pp类型是int \*\*(int型指针的指针),指向int \*。

int a = 10;

int \*p = NULL;

int \*\*pp = &p;

\*pp = &a;

int \*p[3]

指针数组，数组中元素为指针

char\* val1[100] = {0}; 等价于

char\*\* val2 = (int\*\*)malloc(sizeof(int\*)\*100);

int (\*p)[3]

数组指针，p指向数组,p+1代表移动数组大小

int \*p(int)

函数，返回值为int\*,参数类型为int

int (\*p)(int)

函数指针，返回值int，带一个参数int

函数指针使用：

void fun1(int a,char b){cout<<a<<endl;}

void (\*)(int,char) 是函数指针的类型

void (\*p)(int,char) 是定义函数指针对象p

====方式1

void (\*p)(int,char) = NULL;//定义一个函数指针p，初始化为NULL

p = fun1;//指向函数

p(2,'a');//调用

====方式2

typedef void (\*pfun)(int,char); //重定义类型

pfun p = NULL;

p = fun1;

p(2,'a');

const int \*p或者int const \*p 常量的指针

const在\*p前，\*p为一个int型的值，表示p指向的值不可改变

（不可通过p指针修改变量）

int \*const p； 常量指针（形容指针是常量，不能改变）

const 在p前，表示p不可移动，指向的值可变（p的内存地址

值不可变）

void getMemory(char \*p){ p=(char\*)malloc(10);}

该函数不能给p分配内存，而且会造成内存泄漏。因为编译器要为每个参数制作临时副本，p的副本\_p(\_p指向p)，修改\_p指向的内容，也就是p指向的内容被修改。但是，为\_p分配内存相当于让\_p的指向另一片内存，而p指向的地方不变。

在函数中改变指针的指向无效。

在函数中分配的正确方法为：

1. void getMemoryA(char \*\*p){\*p=(char\*)malloc(10);}
2. char \*getMemoryB(){ return (char\*)malloc(10);}

调用时：

char \*p = NULL;

getMemoryA(&p); 或 p = getMemoryB();

野指针：指向垃圾内存，没有指向null的指针。

产生原因：1）指针变量没有被初始化为null；2）free内存后没有指向null；3）指针操作超过了变量生存周期。

### 字符串

字符串由一组连续字符变量表示

有三种表现形式

1）“abcd”，字面值常量，存于只读常量区

2）char a[]，数值可改变，数组字符串的本质是末尾带’\0’的数组

3）指针字符串，可指向字面值，也可以指向数组字符串

连续两个字面值可以合并成一个

printf(“%p”,”abc””de”);

输出“abcde”的首地址

char \*p = “abc”; //字面值字符串，不可变，长度4

char a[] = ”abc”；//数组字符串（末尾自动加’\0’），可变，长度4

char b[4] = {‘a’,’b’,’c’}；//数组字符串，可变，长度4

char c[] = {‘a’,’b’,’c’};//字符数组，可变，长度3

char \*p=”xxx”; (字面值）

“xxx”为字面值（代表字母首地址），不可改变，只能用p=”yyy”,改变指向

不可返回局部变量的地址(因为局部变量在栈中，被释放后有可能写入其他数据，这样局部变量的地址取值输出时会是垃圾数据)，但可以返回字面值：char\* fun(){return “xxxx”;}

char a[]=”abcd”;

数组字符串内容可以改变，初始化后只能用strcpy方式赋值

“=”改变地址，strcpy改变内容

char \*p=(char \*)malloc(sizeof(char));

可写为char \*p=malloc（1）；

不可改变p的指向（内存泄漏），只能用strcpy。

可向p输入字符串（指针必须指向某片空间）

p=”abc”; 堆分配后可以延长使用，若p指向的是栈空间，则不可以延长使用

### 内存

程序：在硬盘上可以执行的文件

进程：在内存中运行起来的程序

一个进程的内存空间包括以下几个部分：

也可分为代码段，数据段，堆栈段

1. 代码区：程序的代码（函数为基本单位）存入，只读
2. 只读常量区：常量和字符串的字面值
3. 全局区：存放全局变量和static变量，main函数执行前分配全局区
4. bss段：存放未初始化的全局变量，main函数执行前清空bss段（清空为0）
5. 堆：自由分配和回收内存
6. 栈：局部变量，包括函数的参数，自动分配和回收内存

在全局区定义 const int i=5； 存于只读常量区

在函数中定义 const int i=5； 存于栈

在函数内 str[5]=”abc”; 栈（未制造常量）

\*str=”abc”; 只读常量区(制造字面值)

代码区

只读常量区

全局区

bss段

堆

栈

内存的地址（虚拟内存地址）

每个进程都设定0到4g的虚拟内存地址（不是内存的真实物理地址，只是一个编号）。0到3g是用户空间，3g到4g是内核空间。虚拟内存地址开始时不对应任何物理内存，直接使用会引发段错误，虚拟内存地址必须映射物理内存（或硬盘上的文件[文件不能小于需要使用的大小]）才能存放数据

虚拟内存地址相当于为物理内存地址取名，存放的只是数字

内存分配其实指的是映射物理内存的过程，内存回收是解除映射的过程

0

int 4字节

进程1 char

4g-1

0 虚拟内存 物理内存

进程2

4g-1 1字节

内存管理的最小单位是一个内存页（4k，4096bit），虚拟内存地址连续时，物理内存地址可以不连续

内存分配<stdlib.h>

malloc:分配内存块，不会对分配内存进行初始化

void \*malloc(size\_t size);

分配成功返回指针，失败返回NULL

exp：

int \*p=(int \*)malloc(3\*sizeof(int));

可使用 p[0]=1,p[2]=3

calloc:分配内存块，对内存进行清零

void \* calloc(size\_t number ,size\_t size);

为number个元素分配内存，每个大小size，成功则返回指针，

失败返回null

exp：

int \*p=(int \*)calloc(3,sizeof(int));

realloc:调整先前已经分配的内存块大小

void\* realloc(void \*p , size\_t size);

p指向需要调整的内存块，size为调整后的大小

1. 内存扩张，realloc不会初始化扩张的内存
2. size为0时，释放原来分配的内存
3. 失败返回NULL，不影响原来的数据

释放： void free(void \*p);

malloc一次映射33个内存页，多余页留下次用，使用完后再次映射，除了数据区，还需额外保存一些信息（如分配内存大小，方便free释放时使用）。所以分配时，上一次malloc与下一次malloc不紧密连接。free是把映射解除，相当于将使用状态切换到未使用状态（数据还在原内存位置，只是此时可以当作空白来覆盖）

如果一次申请页数超过33页，实际分配的内存页数会稍大于申请页数。

getpagesize()可以获取当前页大小

malloc 不能返回动态内存：

void ab(char \*p){ p=malloc(1);}错误

char\* ab(){char \*p=malloc(1);return p;} 正确

int \*p=10;或者int \*p; int a; \*p=a;不可以这样赋值（段错误）

p有虚拟内存地址，但没有映射物理内存，不能存放数据

### const

常用于定义常量，修饰函数的输入参数、返回值。

void ab(const int \*p);

接收时可扩展接收范围：

const int \*p1=const int \*p2；

const int \*p1=int \*p2；

对于函数void ab(int a); const修饰无接收范围意义，因为函数中，a接收一个值时，a=x是复制一个变量，const int a=x时，传入参数时是在做const型的a的初始化。

当一个变量为const型时，可通过指针修改

1. const int a=10;

int \*p=&a;

\*p=20;可修改a的值，编译时有警告，无错误

1. int a=10;

const int \*p=&a;

int \*\*pp=&p;

\*\*pp=20;不可通过p修改变量，可以使用二级指针

1. int a；

const int b；

int \*p=&a；

a，b地址连续，p+1可访问到b的位置

<c++不可修改成功const变量>

### 结构体

#### 定义

struct A{……};

定义:struct A a;

初始化:struct A a={\*,\*,……}

访问或初始化或修改:a.\*;

typedef:给类型取别名

exp: typedef int INT;

与宏的区别:#define INT int

typedef struct{…}A;

此时A为类名，struct A等价于A

结构体指针: struct A \*p=&a;

访问成员:(\*p).i或者p->i

#### 结构体对齐

结构体成员在内存的边界上按照对齐量进行内存块对齐,连续多个成员合计内存小于对齐量则放一个内存块。

整个的结构体结构体长度为对齐量的整数倍

#pragma pack(1)//修改为单个字节对齐

struct INFO{}

#pragma pack()

|  |  |  |  |
| --- | --- | --- | --- |
| 1字节对齐 | 2字节对齐 | 4字节 | 8字节 |
| char(1)  double(8)  short(2)  char(1) | char(2)  double(8)  short(2)  char(1)  \*(1) | char(4)  double(8)  short(2)  char(1)  \*(1) | char(8)  double(8)  short(2)  char(1)  \*(5) |

#### 结构体位段

可以指定每个成员的大小，节约内存(大小指二进制数)

exp: int i:3;指定占三个二进制位

struct{int a:5;int b:2};

结构体大小4个字节，向结构体写入“1234”

内存中是对应ASICII码：00110000,00110001,00110010,00110011

a:5 b:2 则分别取：00110000,00110001,00110010,00110011

a:5 b:4 则分别取：00110000,00110001,00110010,00110011

b取出为：1001

### 联合

联合:和结构体用法相同(关键字用union)

所有成员共用起始地址相同的一片空间

exp:union A{int m;char data[4];}

m=0x 44 43 42 41

输出data的四个元素为4个十六进制数的ASCII码代表的字符

int占4个字节,1个字节8位。8个1代表数字255

char占1个字节,该内存大小可存8位二进制，2位十六进制，int相当于4个char

m: 44(data[0]) 43 42 41(data[3])

大端模式：低字节存高位数据

小端模式: 低字节存低位数据

union { int i; char x[2]; }a; int main() { a.x[0]=10; a.x[1]=1; printf("%d\n",a.i); return 0; }

0x010A 266

1. x[0]=10; a.x[1]=1;a.x[2]=2;

Ox02010A

### 枚举

枚举:为数值(整数)起名字，类似宏

enum 枚举名{枚举常量1,常量2,……};

enum color{R,G,B}; 分别代表0,1,2

int x=G; 或使用color c=G;

可以使用匿名枚举

枚举的默认值从0开始，可以在定义时赋G=5,则其后的B变为6,R仍为0

### 预处理

#预处理命令 在编译的第一个阶段被处理

宏:#define A a

将a替换为A,在写程序时，可用A,编译时自动转换为a

宏定义时，\用来续行

宏是原样替换，无类型检查

宏只能用三目运算完成复杂运算

编写宏时，应在每个代表数字的参数及整个计算结果外都加括号，用来保证运算顺序

#### 宏函数

宏的参数都没有数据类型,个数任意

#define Max(x,y) (x)>(y)?(x);(y)

不可使用x++语法

# 将宏参数转义为字面值常量

#define Txt(a,b) #a"def"#b

int x = 10;

char \*p = Txt(x,2.2);

得到字符串xdef2.2

## 将两个标识符连在一起形成一个新的标识符

#define M(type) type##max 变成typemax

void FunA(){}

void FunB(){}

#define Fun(X) Fun##X()

Fun(A); 调用FunA函数

#### 条件编译

#if 和#endif (#elif,#else)

预处理遇到#if会判断后面的宏目前的数值，如果为0,则排除if与endif之间的内容

#if 0 ................ #endif 可用来做注释，0改为1取消注释

#if define 简写为#ifdef #ifndef

#undef 删除宏

#error 报告错误信息，挂起程序

#warning 警告信息

exp:#if (w>3)

#warning “w>3”

#endif

#### 可变参数

#define LOG(format,...) printf(format,\_\_VA\_ARGS\_\_);

LOG("%s%d", "x=", 3);

#### NDEBUG

relase时编译器选项

#ifndef NDEBUG 可以定义在debuge是才运行的代码

#### 函数声明

\_declspec(deprecated("xxx")) void Fun()

该函数调用时，在编译会提示xxx

#### 头文件

自定义头文件: test.h

#ifndef TEST\_H 防止重复定义

#define TEST\_H

……

#endif

防止头文件重复包含还可以使用

#pragma once

#include <\*\*.h> 到系统指定路径找

#include ＂\*\*.h＂ 到当前目录找，适合自定义的头文件

## C\_Fun

### 常用方法

#### 全局宏定义

\_\_FILE\_\_ 文件名

\_\_LINE\_\_ 行号

\_\_TIME\_\_ 代码编译生成的时间

\_\_DATE\_\_ 代码编译生成的日期

#### 字符判断

0 != isdigit(c) 满足条件则字符是一个数字

isalpha(c) 判断是否为字母

#### 字符转数字

‘5’-’0’ 5转成数字0～9

使用函数：

char \*x=”123\_12”;

int b=atoi(x); //函数会自动在非数字的字符截断，结果为123

char \*x="23.34";

double b=atof(a);

char \*x="-10"; //可以是正数也可以是负数

long l=strtol(x,&x,2);

//第二个参数用来接收指向函数结束后char\*的位置

//2代表x是一个二进制的数字，函数将字符数字转换为10进制数

#### 数字转字符

5-0+’0’ 数字5转成字符

#include <stdlib.h>

\_itoa(数字，存放数字的字符串或者字符变量，数字的进制)

int x=18; char b[5];

\_itoa(x,b,16); //可以将x转成16进制的字符串

#### 小写变大写

-’a’+’A’

#### 进制转换

十进制转二进制：

1）整数部分除2的余数倒排

2）小数部分乘2的积正排

1）余数为0或者1

2）将小数点前的0或者1取走

0.a\*2 得到1.b（取1）

0.b\*2 ……

二进制转十进制：按位乘以2的n次方后，所有项相加。

……2^2,2^1,2^0,2^-1,2^-2……

二进制转十六进制：4位一份，每份为一位十六进制

4位1111为15

简易方式：

16的二进制是10000（1后面四个0）

转十六进制快速法：

exp：28

减去一个16余12

16代表第二位为1，12代表C

所以结果为0x1C

### 中文支持

#include <locale.h>

//char\* old\_locale = \_strdup( setlocale(LC\_CTYPE,NULL) );

setlocale( LC\_CTYPE, "chs" );

//setlocale( LC\_CTYPE, old\_locale );

//free( old\_locale );

### math.h

z=pow(x,y); //计算x的y次方，X与z必须同时是float或double。

z = pow((float)x,(float)1.0/3); //计算x的三次开方

int a=(int)sqrt((double)25); //对25开平方，得到5。

abs(x)； //求绝对值。

float a=sin(3.14/2); //参数是弧度，计算sin90度的值。

ceil向上取整，floor向下取整

### 内存拷贝

1）void \*memset(void \*dest, int c, size\_t count);

将dest前面count个字符设置为字符c，返回dest的值。

2）void \*memcpy(void \*dest, const void \*src, size\_t count);

从src复制count字节的字符到dest. 返回dest的值.

void \*memmove(void \*dest, const void \*src, size\_t count);

如果src和dest出现重叠, 函数会自动处理.

char \*a=malloc(1); char \*p = a;

memcpy(p,"hello",5); 可以正常输出(也可以写大于5的数字)

char a; char \*p = &a;

memcpy(p,”hello”,5);

不可以正常输出，会多出乱码（且不能写大于5的数字，会出现段错误）

因为p只有指向堆对象时才可正常无限延伸

3）void \*\_memccpy(void \*dest, const void \*src, int c, size\_t count);

从src复制0个或多个字节的字符到dest. 当字符c被复制或者count个字符被复制时, 复制停止.

如果字符c被复制, 函数返回这个字符后面紧挨一个字符位置的指针. 否则返回NULL.

4）void \*memchr(const void \*buf, int c, size\_t count);

在buf前面count字节中查找首次出现字符c的位置. 找到了字符c或者已经搜寻了count个字节, 查找即停止.

操作成功则返回buf中首次出现c的位置指针, 否则返回NULL.

5）int memcmp(const void \*buf1, const void \*buf2, size\_t count);

比较buf1和buf2前面count个字节大小.

返回值< 0, 表示buf1小于buf2;

返回值为0, 表示buf1等于buf2;

返回值> 0, 表示buf1大于buf2.

int memicmp(const void \*buf1, const void \*buf2, size\_t count);

比较buf1和buf2前面count个字节. 与memcmp不同的是, 它不区分大小写.

### 字符串

#### 函数

char a[m];

scanf(“%s\n”,a); 无法读入字符串中的空格，遇到空格会截断

gets(s);可以读入输入字符串中的空格

fgets（a,n,stdin);用n限制输入字符的数量，如果输入的字符少于n，可以将’\n’也读入，该函数比gets安全

c=getchar() 从键盘读入字符，可读入空格

格式化字符串函数需要格式精确匹配，如float一定是用%f，double一定是用%lf。返回值为成功匹配的个数。

sprintf（str，“输出格式字符串”，输出列表）；

sscanf（str，“输入格式字符串”，输入列表）；

char \*str="10,1.20576,3.1,abc";

int a = 0;

double b = 0;

float c = 0;

char s[100] = {0};

sscanf(str,"%d,%lf,%f,%s",&a,&b,&c,s);

#### <string.h>

|  |  |  |
| --- | --- | --- |
| **函数** | **原型** | **使用** |
| **strlen** | size\_t strlen(const char \*string) | 获取字符串长度, 字符串结束符NULL不计算在内，没有返回值指示操作错误.  char s[10]=”abcd”;  sizeof(s) 10  strlen(s) 4 |
| **strcpy** | char \*strcpy(char \*dst, const char \*src) | 把src复制到dst中（清空dst） |
| char \*strncpy(char \*dst, const char \*src, size\_t count) | 复制指定长度  不清空dst（覆盖） |
| **strcmp** | int strcmp(const char \*s1, const char \*s2) | 返回值< 0, 表示s1小于s2;  返回值为0, 表示s1等于s2;  返回值> 0, 表示s1大于s2 |
| int strncmp(const char \*s1, const char \*s2, size\_t count)  只比较count个字符 |
| **strcat** | char \*strcat(char \*dst, const char \*strSource) | 将src添加到目标串dst后面, 并在得到的新串后面加上NULL结束符. src会覆盖dst后面的结束符NULL. 在字符串的复制或添加过程中没有溢出检查, 所以要保证目标串空间足够大. 不能处理源串与目标串重叠的情况.  函数返回dst值 |
| char \*strncat(char \*dst, const char \*src, size\_t count) | 将src开始的count个字符添加到dst后. 如果count大于src长度, 则会用源串的长度值替换count值 |
| **strset** | char \*strset(char \*str, int c) | 将str的所有字符设置为字符c, 遇到NULL结束符停止.  函数返回内容调整后的str指针 |
| char \*strnset(char \*str, int c, size\_t count) | 将str开始count个字符设置为字符c |
| **strstr** | char \*strstr(const char \*string, const char \*strSearch) | 在字符串string中查找strSearch子串.  返回子串strSearch在string中首次出现位置的指针. 如果没有找到子串strSearch, 则返回NULL. 如果子串strSearch为空串, 函数返回string值 |
| **strchr** | char \*strchr(const char \*string, int c) | 查找字符c在字符串string中首次出现的位置, NULL结束符也包含在查找中.  返回一个指针, 指向字符c在字符串string中首次出现的位置, 如果没有找到, 则返回NULL |
| char \*strrchr(const char \*string, int c) | 查找字符c在字符串string中最后一次出现的位置, 也就是对string进行反序搜索 |
| **strpbrk** | char \*strpbrk(const char \*string, const char \*strCharSet) | 从str的第一个字符开始查找，返回第一个出现在strSet中的字符位置指针.  如果两个字符串参数不含相同字符, 则返回NULL值. |
| **strcspn** | size\_t strcspn(const char \*string, const char \*strCharSet) | 从str的第一个字符开始查找，返回第一个出现在strSet中的字符位置下标  如果一直不满足条件则会返回字符串NULL的位置 |
| **strspn** | size\_t strspn(const char \*str, const char \*strSet) | 从str的第一个字符开始查找，返回第一个没有出现在strSet中的字符位置下标 |
| **strrev** | char \* strrev(char \*Str) | 将字符串string中的字符顺序颠倒过来. NULL结束符位置不变.  返回调整后的字符串的指针(可以不使用返回值) |
| **strupr** | char \*\_strupr(char \*string) | 将string中所有小写字母替换成相应的大写字母, 其它字符保持不变.  返回调整后的字符串的指针 |
| **strlwr** | char \*\_strlwr(char \*string) | 将string中所有大写字母替换成相应的小写字母, 其它字符保持不变.  返回调整后的字符串的指针 |
| **strdup** | char \*strdup(const char \*strSource) | 函数运行中会自己调用malloc函数为复制strSource字符串分配存储空间, 然后再将strSource复制到分配到的空间中. 注意要及时释放这个分配的空间.  返回一个指针, 指向为复制字符串分配的空间; 如果分配空间失败, 则返回NULL值. |

.

char \*strtok(char \*strToken, const char \*strDelimit);

在strToken 串中查找下一个标记, strDelimit字符集则指定了在当前查找调用中可能遇到的分界符.

返回一个指针, 指向在strToken中找到的下一个标记. 如果找不到标记, 就返回NULL值. 每次调用都会修改strToken内容, 用NULL字符替换遇到的每个分界符，所以字符串不能是字面值。

char input[16]="abc,d";

char \*p;

p=strtok(input,",");

if(p) printf("%s\n",p);

p=strtok(NULL,",");

if(p) printf("%s\n",p);

第一次要设定参数，第二次参数可以NULL  
  
函数第一次调用需设置两个参数，strtok(str,",")

str 需要分割的串，根据“,”分割  
第一次分割的结果，返回串中第一个，之前的字串,也就是上面的程序第一次输出abc  
  
第二次调用该函数strtok(NULL,"."),第一个参数设置为NULL，第二个参数还是分割的依据  
结果返回分割依据后面的字串，即上面的程序输出d

====按照Split分割字符串

char \*\_Context;

char \*p = strtok\_s(pBuf, Split,&\_Context);

while (NULL != p)

{

p = strtok\_s(NULL, Split,&\_Context);

}

### I/O

<stdio.h>

printf("格式及内容",参数列表)

参数表达式多个时，从右向左计算

%后为参数输出格式,\后跟字符表示转译字符,其他内容原样输出

printf(“a=%d\n”,a);

可以直接输出格式字符串：char \*p=”abc”; printf(p);

printf("%.\*f",2,a); 输出参数列表中定义输出宽度

puts(字符串)

scanf(“格式”，地址);

scanf在按下回车时结束输入，在读到空格后会读入空格之前的内容，放入输入位置，剩余内容留在输入缓冲区（无空格就全读入，%c可读入空格）

scanf("%\*d %d",&a); 忽略输入的第一个数字

清空输入缓冲区：

缓冲区内有内容时，会输入到下一次要输入的位置

scanf(“%\*[^\n]”);

scanf(“%\*c”); 读走所有非\n的字符，然后读走\n。

当这两条语句前无输入时，会要求输入。

Exp：

scanf（“%d”,&a);

scanf(“%\*c”);

scanf不读入\n,会留在缓冲区，此句将\n读走

输入缓冲区显示条件

1. 遇到\n
2. 函数结束时
3. 输出缓冲区满
4. fflush(stdout) 强制刷新显示

获取单个字符：int c = getchar();

获取字符串（可以读取到空格）

char s[1024] = {0};

gets(s);

该函数不安全，使用fgets替代

char a[100] = {0};

//fflush(stdin);

fgets(a,100,stdin);

a[strlen(a)-1] = 0;

### 文件操作

文件就是存储介质上的数据的集合

操作系统都是以文件为单位对数据进行管理，每打开一个文件，都有一个结构体来保存，这个结构体是系统定义：FILE

文件指针就是指向结构体的指针

#### 打开关闭

FILE \*fp 定义一个文件指针fp

fopen(“文件名”，”打开方式”)

成功则返回文件指针，失败则返回NULL

fclose(“文件指针”) 文件指针=NULL；

成功则返回0，失败返回-1（EOF）

文件打开方式：

r 只读

w 只写 （文件存在时，打开会清空）

a 追加 （在原有内容上追加新内容）

r+ ， w+ ， a+ 都是读写

二进制读写

rb，wb, ab

rb+（r+b） wb+（w+b） ab+（a+b）

#### 读

fread(\*p，psize,读取次数，文件指针） 二进制读取

fgets(str,n,fp)

返回值str首地址。

从fp中读出n-1个，最后加上’\0’,放入str。

如果读取到\n或者遇到EOF,则读入结束。因此读取的行内容中最后一个字符可能是’\n’。

char str[100] = { 0 };

while (fgets(str,100,pf))

{

int posEnd = strlen(str) - 1;

if(str[posEnd] == '\n') str[posEnd] = 0;

puts(str);

}

fgetc(文件指针) 从文件中读出一个字符出来

失败返回EOF，成功则返回读出的字符

fscanf（文件指针，格式字符串，输入列表）

返回从文件中格式化到输入的参数列表的个数

返回0表示没有格式化成功

如果返回值为-1表示读取到末尾

#### 写

fwrite(p，psize，写入次数，文件指针)

fputs(pstr,fp) 成功返回非负数，失败返回EOF

fputc(‘字符’，文件指针） 将字符写入文件

失败返回EOF，成功则返回字符的ASCII码

fprintf(文件指针，格式字符串，输出列表)

将输出列表中的值按格式字符串变成字符串然后写到文件中。相当于printf不将结果输出到屏幕，而是输出到文件中。

#### 刷新

fflush(文件指针) 刷新缓冲区，也就是将缓冲区内容立即写入或者读取

#### 指针位置

文件读写时注意文件指针的位置

ftell（文件指针） 获得指针距离文件开头的距离

rewind(文件指针) 将指针移动到文件开头

fseek（文件指针，移动大小，基准位置）

SEEK\_SET 0 以开头为基准

SEEK\_CUR 1 以当前位置为基准

SEEK\_END 2 以文件末尾为基准

fseek(fp,0 ,SEEK\_END);

int fileLen = ftell(fp); 将指针移动到文件尾，然后得到文件大小。

#### 文件操作

remove(“a.txt”);

rename(“a.txt”,”b.txt”);

### 随机数

stdlib.h

//一个函数只能设置一个随机种子，不能写在循环内部重复设置

srand(time(0));

rand();//读取掉第一个数据，第一个产生的数据总是不断增大

int x=rand()%100;得到100以内的随机数

### 系统命令

stdlib.h

相当于在cmd中运行命令行

system("pause"); //让cmd窗口不关闭

system("cd d:");

system("dir"); //等价于在命令行连续执行两个命令

### time.h

time\_t t=time(0); 可获得当前时间(从1970年1月1日0:0:0开始的秒差)

显示时间：

while(1){

time\_t t=time(0);

printf("…\r＂,t);

fflush(stdout);同一位置显示时间，不断刷新

while(t==time(0));等待1秒

}

s=t%60;

m=t%3600/60;

h=(t%(3600\*24)/3600+8)%24

ctime(&t) 转成美式时间 %s

char buf[100]={0};

struct tm\* cur=localtime(&t);

sprintf(buf,”%4d”,cur->tm\_year+1900);

tm\_year+1900

tm\_mon+1

tm\_mday

tm\_hour

tm\_min

tm\_sec

sleep()

Exp: int a=sleep(3)

使程序休眠3秒，中断时返回剩余为休眠的秒数

### <errno.h>

errno为全局变量，存放错误编号，出错则errno的位置改变，不出错则不变化

perror(“str”) 自动找到错误编号，并打印错误信息，str为额外的提

示信息，可以不加

strerror()把错误编号转换成错误信息

printf(“%m\n”,errno); 打印错误信息

### <stdarg.h>

int max(int cnt,...){

int maxValue=0;

int i=0; //在va\_list v和va\_end(v)之间不可定义变量，所有变量的定义必须放在前面

va\_list v; //定义一个变长参数列表

va\_start(v,cnt); //将参数列表定位到第二个参数位置（cnt是对…包含几个参数进行说明】）

maxValue=va\_arg(v,int); //给maxValue赋的值为列表中当前位置的值，同时指针指向下一个值

for(i=1;i<cnt;i++){

int data=va\_arg(v,int);

if(data>maxValue)

maxValue=data;

}

va\_end(v); //释放变成参数列表

return maxValue;

}

调用max()函数时，可用max(2,34,123,23),内部参数数量可以任意

### <assert.h>

assert函数在NDEBUG下自动屏蔽

### 环境变量

每个程序都会收到一张环境表，是一个字符指针数组，存着各种数据的指针，以null结束。所有的环境变量在程序中可以通过环境表获取。

获取方式：

(1)extern char \*\* environ；

存放连续的一级指针（字符串），用二级指针可遍历

environ就是环境表的首地址，是全局变量。

(2)主函数的第三个参数：

main(int argc, char \*\*argv, char \*\* env)

env用来接收environ

遍历环境表：

for(;\*env;env++)

printf("%s\n",\*env);

<stdlib.h>

putenv(“name=value”)

将形式为name=value的环境变量放入环境表

getenv(“name”)

返回name关系的value的指针（value为字符串）

setenv(“name”,”value2”,1)

将name关系的内容设置为value2，第三个参数决定是否替代已有变量（0代表不替代，1替代）

unsetenv(“name”)

删除定义

clearenv()

删除环境表中所有项

## c++

### 特性

c++头文件无.h,需要使用c语言头文件时：

<stdio.h>或<cstdio>

std命名空间有：cin，cout，endl，string，fstream等

c++后缀为：.cpp .C .cxx

赋值可以用： int i(3);

c++中结构、联合、枚举

struct、union、enum不再是类型的一部分，名字就是类型名

struct A{......}; A a;

c++允许使用匿名联合

c++的struct可以像class那样用，不同的是结构体默认是public，而class默认是private。

没有任何字段的结构体，c++大小1，c中为0

结构体封装数据，函数封装代码，结构体大小与定义的函数数量无关，只与变量有关。

### 命名空间

一组相关的类型、变量、函数、对象等组织到一个逻辑结构中，按逻辑划分模块，可避免命名冲突

namespace 空间名{.............}

命名空间内可以定义函数，函数内不可定义函数。

同一命名空间可以分开写，定义后会整合成看作一个空间，在分开写时，如果要定义函数，则需在另外的空间声明

1. 访问命名空间元素或者函数

空间名::a 空间名::b()

1. 若访问使不想使用::语法

using namespace n;

可以直接使用n中所有元素

using n::i;

可以直接使用n中的变量i

using使用时，不能引用同名的变量，会产生冲突，使用::语法解决

无名命名空间： namespace{int a；...............}

使用时： ::a 或者a

全局变量与无名命名空间同名变量时会冲突，::a不会和局部变量冲突，全局变量与局部变量重名时，局部优先。

匿名空间不可以跨文件访问

命名空间内套有命名空间时：

n1::n2::a 访问到n1内n2里的a

命名空间的别名：

namespace n4=n1：：n2

将n1中的n2命名为n4

嵌套的命名空间：

命名空间A中包含另一个命名空间B，则A相当于B的全局区，所以A不能访问B中的元素，B可以访问A中的元素。

重命名类：

class ABC{public:void fun() { cout << "xxx" << endl; }};

namespace NA {

class AAA : public ABC {};

}

使用：

NA::AAA aa;

aa.fun();

### 匿名联合

不需要定义联合名，全局区联合需要是static

static union

{

int VAL;

char CH;

};

VAL = 123;

int \*p = (int\*)&CH;

int x = \*p;

### 函数重载

函数重载：用同一作用域中，函数名相同，参数列表不同（个数，顺序，类型）的函数构成重载，返回值不构成重载。

因为gcc编译时不会给函数改名，g++编译时根据名字和参数改名

在c++代码中的函数名前加extern “C”可以在编译时不改变函数名，这样就可以让c与c++互调函数

1. c调用c++函数：

g++ -c tc.cpp 生成 tc.o文件

gcc t.c tc.o -lstd++

1. c++调用c的函数

gcc -c t.c 生成t.o文件

g++ tc.cpp t.o

### 函数默认值

int ab(int x=1, int y=10){}

这样写可以在调用时不传入参数，如果传入参数，则使用传入的值替代默认值

靠右原则：默认一个参数值后，后面的参数都需要默认（左边的可以不默认）

当函数声明和实现分开时，参数的默认值在声明部分指定

ab(int x=1)与ab()在使用ab()调用时会产生冲突

### bool

c++中boo是一个特定的类型

0 ‘\0’ false NULL 代表false

其他都是true

### 哑元

哑元：在实现函数时，函数的参数只有类型，无形参名

exp：

原函数： void ab(int a);

改变后： void ab(int);

使用后ab()或者ab(4)都可以，可以保存函数之前的使用方式

当void ab()和void ab(int)同时存在，ab(3)会使用ab(int)函数，ab()会使用ab()函数

### 内联函数

inline void ab(){}

inline关键字必须放在函数实现处，放在函数声明处无效。

如果类内函数在声明的同时实现，未分开写，就会变成内联函数。

头文件中只能放函数的声明，不能放实现。否则因为多个文件包含了头文件，会出现重定义。如果要在头文件中放函数的声明及实现，可以直接定义为内联函数。

直接把函数的二进制代码复制到调用位置，适合调用频繁，且代码量小的函数。主要是为了减少函数跳转调用的时间。内联只是一种优化策略，对编译器的权限只是一个建议，宏预处理时替换，内联编译时替换。

### 内存分配

new——delete new[]——delete[]

new：

\*p=new 类型； int \*p=new int；

\*p=new 类； A \*a=new A();

\*p=new 类型(初值)； int \*p=new int(3); \*p的值为3

delete：

1. delete 指针

delete p；

p=NULL；

1. int \*p=new int[5]; （可使用p[1]=10）

delete[] p;

int (\*p)[3]=new int[4][3];

p为()[3]数组的指针，分配4个数组大小的空间

定位内存分配：

char data[100];

int \*p=new(data) int[25];

直接从栈中分配内存（data[100]中），不需要释放内存

### 引用

引用:变量的别名

c++中函数可以使用引用改变外部函数的值（类似指针）

void fun(int& a){}

不能返回局部变量的引用，加static可以让变量被返回

定义引用： int x=100; 引用必须初始化

int &y=x；

一个变量可以有多个引用，引用绑定后不可作为其他变量的引用

引用前加const就可以为引用赋常数：

const int &ab=10；

或者void ab(const int &a); 调用时ab(10);

不可以有常量的引用：如&3

int (&a)[5] 数组的引用

c++中引用和指针的区别：

1. 引用代表变量本身，而不是变量地址
2. 引用必须初始化，而指针不用
3. 一个变量的引用建立联系后不可以关联到别的变量，而指针可以随时改变指向
4. 有指针的指针，无引用的引用
5. 有指针的引用，无引用的指针 int \*&b=p；
6. 有指针的数组，无引用的数组

### 类型转换

新类型 变量名 = \*\*\*\_cast<新类型>(原始变量)

1. 隐式类型转换 static\_cast

避免自动类型转换

int a = 10; float b = static\_cast<float>(a);

不能让指针类型互相转换，比如char\*转换为int\*。对于指针只能从void\*转为其他指针。

1. const\_cast 去掉常属性

volatile const int y=100；

int \*p=const\_cast<int \*>(&y);

\*p=20;

如果不加volatile，y为10，加了以后y才能被改变。

1. dynamic\_cast

当A有虚函数时才能使用，否则编译报错。

A \*pa = new B(); B \*pb = dynamic\_cast<B\*>(pa);

1. reinterpret\_cast

允许任何类型指针类型的转换，包括指针变整数，整数变指针

### 类

#### 概述

面向对象的三大特征

封装：把不需要用户知道的隐藏起来（private），然后提供公开的访问接口(public)。提供公开的接口一般是在public下写函数，通过函数访问到类内私有成员。可以保护数据，防止不必要的扩展。对于调用者而言，使用简单，便于协同开发。

继承：可以把一个类相关数据传到下一个类中，便于扩展（可以增加新特征和功能）。

多态：可以通过子类改写从基类中继承的方法。

类：是对象的抽象描述

对象：对类的实例化

A \*a=new A();

A a=A();

A a; 定义一个类A的对象a

A().fun(); 构造一个临时对象，并调用成员函数

对象就是实际内存中的内容，引用（指针）保存对象的地址，变量保存引用

#### 内存分布

一个空的类对象占1个字节

类对象的大小由非静态的成员变量大小决定，函数和静态成员不占用对象的大小，如果类中有virtual函数，则产生虚函数表，会增加一个指针大大小。

#### 与结构体区别

struct包装数据，函数包装代码，类既包装数据，也包装代码

行为：类中的函数

特征：类中的变量

class和struct都可以描述对象

struct可以使用{..............}初始化对象，class只有在最新的标准才可以使用这种初始化的方法（需要public）

struct A

{

void fun(){cout<<"xxx"<<endl;}

int a;

double b;

};

可以使用A a={1,1.2};

如果把struct换成class则可能无法使用该种赋值方式

#### 权限

struct默认是完全公开的，class默认是私有的

private：只有在类内可以使用

public：在类内外都可以使用

在class中加public:则此语句下的内容都可以在类外和类内访问，直到遇到下一个权限的修饰语

class A{..................};

A a; 构造函数无参时，此行不可写成A a()

A \*p=new A();

new 后放的是A的构造函数

#### 类的基本函数

class A{......

public: (如果定义构造函数，则不能省略)

A(){......}

构造函数重载 A(int x， int y)：a(3),b(x),y(y)初始化参数{.....}

A(int x=1){....}

拷贝构造 A(const A&a){.....}

稀构函数 ~A(){........}

}

声明和实现分开时：初始化参数列表放在实现部分，参数的默认值在声明部分指定，函数前的static修饰放在声明处，函数的const修饰在声明和实现部分都写。

构造、析构都有系统默认的，如过要自定义，会覆盖系统的。此类函数无返回值

拷贝构造内的A&a可以不写a，如果拷贝构造的操作中有需要使用a，才必须写上

##### 构造函数

构造函数在对象创建时调用一次

A a； 创建一个栈对象，同时调用无参构造

A a(10); 创建一个栈对象，调用有参构造（传入参数10）

A \*p=new A(); 创建一个堆对象，同时调用无参构造（无参构造时也可用 A \*p=new A）

A(x=1)和A()在使用A()调用时会产生冲突

const变量只能在初始化时赋值（初始化参数列表中进行），初始化参数列表中写成A(),如果A是基本类型,则A的值变为0，如果A是类类型，则为无参构造（继承中用）

##### 拷贝函数

拷贝构造函数调用时间：在使用同类型的对象构建一个对象(复制)；在函数参数值传递；函数返回对象一旦有拷贝操作就调用

使用时：A a; A b=a;

不可写成A a； A b； b=a；

1. A get(){A a；return a;}

A c=get();

不用拷贝构造，系统自动优化

2）A get(){A \*a=new A(); return \*a;}

A c=get();

会调用拷贝构造，因为使用了堆

自定义拷贝构造时，必须同时定义构造函数，不可以只定义拷贝构造（可以只定义构造，不定义拷贝）

##### 析构函数

析构函数中执行释放内存可能会造成内存泄漏。

在栈中将class创建出对象后，会在程序某行之后不再出现该对象时，自动调用析构函数（在最后一次出现对象的行调用）

析构函数用来释放内存

可单独调用~A() 堆内层时必须单独调用析构，因为堆内存不会自动调用析构函数

exp：A \*a=new A();

a->~A()

##### 虚析构函数

（在析构前加virtual），基类有虚函数，基类对象的指针指向子类对象(多态条件时)，释放基类对象的该指针时，如果基类析构函数不是虚函数，则子类析构函数的调用行为未定（不调用），把基类中析构变为虚函数，则会调用子类构造，同时也会调用基类的析构。

构造函数不能为虚函数（拷贝构造，构造）

##### 深拷贝

当要拷贝的对象中有指针变量时，会使用拷贝前后两个对象中的指针指向同一块堆内存，在其中一个对象释放堆内存后，另一对象中指针指向的是一块已经释放的内存，再次释放会出错，解决此问题需要使用深拷贝方式（自己定义拷贝过程）

class A{

int m；

int \*n;

A():m(10),n(new int(8)){}

A(const A&a):n(new int){

m=a.m; \*n=\*(a.n);

}

~A(){

if(n！=NULL){delete n; n=NULL;}

}

}；

1. 构造函数为指针分配堆内存
2. 拷贝构造函数为拷贝出的对象的指针分配新的堆内存，并进行其他数据的拷贝
3. 自定义析构函数释放堆内存（也可以不使用if语句进行判断，因为有时候没有给类的指针成员分配内存，而是指向null，可以不用释放,delete NULL也不会出问题）

因为有了1)和3)所以才需要2)的步骤。

##### 拷贝构造和赋值函数

编译器会自动生成缺省的拷贝构造和赋值函数，如果不想编写这两个函数，也不希望默认的被调用，则可以将这两个函数声明为私有的。

##### c++11初始化

class ABC

{

int x = 10;

int y{2};

};

#### 声明与实现

类的声明和定义在开发时，是需要分开的，定义写在头文件中

返回类型 类名：：函数名（参数）{........}

=============声明

#ifndef T\_H

#define T\_H

class Date{

Date();

void ShowDate();

};

#endif

=============实现

#include “t.h”

Date::Date(){..........}

void Date::ShowDate(){..................}

#### this指针

this是指向当前对象的指针，\*this代表当前对象。构造对象时，指向正在构建的对象，成员函数中，指向这个函数的对象

当函数形参与成员变量重名时，可以用this区分：

class A{

int b;

public:

A(int b){this->b=b;}

};

可使用this返回数据，可作为参数传递

#### 静态成员、函数

静态成员：类和类的对象所共享的函数、数据（相当于全局变量、函数，存在于全局区，不属于对象，只是访问范围限制在类内），不需要对象，只需要类型就可以访问。

声明和实现分开时，函数前的static修饰放在声明处。

静态成员必须在类外（全局区）进行初始化操作，不能在类的初始化中进行赋值。

class A{ public: static int x;};

int A::x=0;

类内函数或类的对象，能直接访问类内的静态变量、函数。

类外调用静态成员、函数[需要是public]

A::x;

A::fun();

静态函数只能直接访问静态成员，不能直接访问非静态成员，因为无this指针。如果想在静态函数中访问非静态成员，可以传入一个指针：

static void ab(A \*a){ cout<<a->m<<endl;}

#### 成员指针

typedef void (A::\* FUN)(int a);

类型是：void (A::\*)(int);

//赋值

FUN f = &A::f1;

使用：

(a.\*f)()

(this->\*f)()

成员指针直接输出时为1

用联合可以输出指针指向的地址

union{int A::\*p; void \*pr;};

用pr可以输出指向的地址

#### const对象和函数

const对象只能调用const函数，声明和实现分开时，函数的const修饰在声明和实现都需要写。

1. const A a; const对象
2. void ab() const{..........} const函数（类内）

void ab(){.................}

以上两个函数构成重载（非const函数优先调用非const函数，如果无非const函数，就用const）

const函数不能修改普通成员变量，如需修改，则在成员前加mutable修饰

exp： mutable int m；

void ab()const{m=10;}

#### 友元

友元函数：在全局函数中要调用类中私有成员，需要在类中用friend声明一下这个全局函数。则该全局函数可访问到类中私有成员（通过类的对象）

exp:

class A{int x;};

A a;

void ab(A a){cout<<a.x<<endl;}

把class改为：

class A{

int x;

friend void ab();

};

友元类：

friend class B；

在B继承A时，使用友元类，可以使B能访问到A的private成员

扩展用法：

template <typename T>

class AAA{friend T;int x;};

class BBB

{

void fun()

{

AAA<BBB> aa;

aa.x = 10;

}

};

#### 其它

在类中调用其他函数，需要在类前声明要使用的函数。

对象创建过程，分配这个对象内存

对象没有任何成员，大小是1

对象大小由数据成员决定，和函数无关（虚函数除外）

如果对象成员是class类型，调用这个对象的无参构造。基本类型不做任何操作，有初始化参数列表时，先执行初始化参数列表

不能在定义时对数据进行初始化

class A{int a;}不能写成int a=10；

当class中无数据需初始化，有时需要用a={}或者写一个空参构造

A& b(){..........}

A a;

1. b().b().b(); 返回类型是A&，所以可以让函数多次调用

explicit A(){} 构造函数

防止A a=10; 语句将10转成A类型

class中每个函数的第一个参数实际为类的指针

void ab(int \*) 等价于void ab(A a, int x)

因此使用 a.ab(y);

在全局区定义时： void (\*p)(A,int)

### 内存分配

new与malloc区别：

1. new会在创建对象时自动调用对象的构造函数
2. 自动处理类型转换

delete与free区别：

delete释放内存前会自动调用析构函数

### 运算符重载

#### 定义

运算符重载(一种特殊的函数)

双目运算符重载：

对象1 运算符 对象2

类内重载：

void operator+(对象2){.......}

void为返回值类型，可任意

+为要重载的运算符，相当于函数名

在类内重载可以少写第一个参数，因为类内有this指针（this指针指向当前对象1）

对象1必须是类，对象2类型任意

全局区重载：

void operator+(对象1，对象2){...........}

运算符重载后，即可用运算符对类进行运算，一般重载尽量定义在类内

输入输出重载只能在类外，不能打开输入输出类

istream& operator>>(istream&is, A &a){

return is>>a.x或者return cin>>a.x

}

is等价于cin，可任意名

A &a 不能加const，输入是做改变，&a引用a，因为相当于在函数中改变外部的值

istream& 返回输入流，即返回的值后面可以用流的语法。

ostream& operator<<(ostream& os, const A &a){..........}

#### 单目运算符重载

运算符 对象1 -, !, ~, ++, --

与双目运算符重载类似，只是少了一个对象2

++，--重载时默认为前++、--：

类内定义： void operator++(){.............}

使用哑元int可变为后++、--

void operator++(int){............}

#### ()运算符重载

class AA

{

int x;

public:

AA(){x=10;}

int operator()(int a,int b)

{//函数方式调用

return a+b+x;

}

//类型转换

operator double(){return x;}

};

AA aa;

int x = aa(10,20);

double y = aa;

#### 特殊重载

有些运算符只能重载为类内的成员形式：

=， []

“=”重载时注意自赋值：

if(this==&a) return \*this;

可以重载new，delete，重载时至少有一个类类型

void\* operator new(size\_t t){............}

void operator delete(void \*p){...........}

\*,->希望按照指针方式操作

\*返回对象，->返回地址

A& operator\*(){return \*p;}

A\* operator->(){return p;} 迭代器用

注意：(\*p).m “.”优先级高

#### 不能进行的重载

不能对基本类型数据进行重载

不能发明新的运算符

不能被重载的运算符：

::

.

sizeof

.\* 到成员的指针

？ ： 三目运算符

typeid 类型识别

### 继承

#### 定义

B继承自A，则B中有A的一切，而且对A进行了拓展。所以B可以强制类型转换成A，而A类的指针可以指向B，反之却不可以。

1. 组合继承：在子类中定义一个基类的对象作为成员(不能实现多态)，不可以在B的初始化参数列表调用A的构造：

class B{A a;...........};

1. class B:public A{.......};

或者protect，private

可实现多态

在类中定义并实现另一个类时：

class A{class B{};};

1. B不能直接互访成员，A相当于B的类外全局区

#### 权限

public：可被任意实体访问到

protected：只允许本类成员及子类访问

private：只允许本类成员访问

1. public公有继承：public----------->public

子类内和外都可以访问

protected------------->protected

子类可访问（子类的子类可访问）类外不能访问

private------------->无权

子类内外都不可访问

1. protected保护继承：public、protected------------>protected

private------------->无权

1. private私有继承：public、protected---------->private

子类可访问（子类的子类不可访问），类外不可

访问到

private-------------->无权

子类无权访问时，如果在基类中将子类定义为友元类，则可访问。

或者在基类提供公有的接口让子类可以通过接口访问到

子类继承基类后，基类相当于子类的一个成员，等同于把基类中的成员写在子类中

子类中没有和基类相同名的数据时，可以直接使用，若有重名，则会隐藏基类数据，使用A::ab调用(有访问权限前提下)

#### 函数调用顺序

继承中，先调用基类的构造，后调用子类的构造（析构顺序相反），在子类定义后，不会影响基类的相应构造函数和析构函数

如果基类没有无参构造，则需要在子类构造函数的初始化参数列表中调用基类的构造。

子类默认先调用基类的拷贝构造，后调用子类的拷贝构造(赋值运算符)自定义后将不再调用基类的

子类自定义拷贝构造时：

class B:public A{

B(const B&b):A(b){.............}

调用基类的拷贝构造

剩余内容（b中特有内容）的拷贝

};

构造函数调用顺序：

虚基类构造--->普通基类构造---->子类中的子类---->子类

#### 多重继承

class B:public A1,public A2{..........};

class B:public A1,A2{..........}; //A2前不加public表示private继承

基类有重名时，在子类定义此名，使基类的隐藏

#### 钻石继承

一个基类有多个子类，而这多个子类又是另一个类的基类

class A{public:int val;};

class B:public A{};

class C:public A{};

class D:public B,public C{};

访问D对象的val时，无法确定是B中的val还是C中的val，需要类名作用域：

D d;

int x = d.B::val;

子类通过不同的基类访问到最高层的类时，会有歧义

为了消除歧义，引入虚继承，这样最高层代码只有一份，不会产生复制，最下层可直接访问到最上层

class A{public:int val;};

class B:virtual public A{};

class C:virtual public A{};

class D:public B,public C{};

使用时：

D d;

int x = d.val;

### 多态

基类函数必须是virtual（子类可以不加），使用基类的指针或者引用指向子类，实现基类对象调用子类的多态。

当子类调用基类的函数a()，而这个函数又调用基类和子类都有的b()时，会调用基类的b()，只有重写时，才会调用子类的b()。

#### 多态的应用

继承是多态的基础，虚函数重写是多态的关键

基类有虚函数，基类的指针或引用指向子类的对象时，基类掉调用这个虚函数就可以有不同表现（这个虚函数在子类中可以重写）

子类重写基类虚函数时，前面可以加virtual，也可以不加

在子类定义和基类相同的函数，这种机制为名字隐藏，但如果基类函数是虚函数，则称为函数重写（overwrite）

class A{public: virtual void ab(){1}};

class B:public A{public:virtual void ab(){2}};

1. A \*p=new B();

p->ab(); 结果为2

1. B b;

A& a = b;

A指向B后，通过指针pA只能访问A的成员内容，但是pA通过虚函数可以访问到B的函数，这样就相当于可以对B的内容进行操作。

#### 虚函数表

class中数据占内存，函数不在class内存储，不占class内存，但在函数前加virtual，就产生一个纯虚指针（指向虚函数表），该指针相当于class中数据，会占开头位置的内存

每个类只要有虚函数，则会有自己的虚函数表，如果没有重写基类的虚函数，则虚函数表中的函数地址是基类的，如果重写了虚函数，则虚函数表中的虚函数地址是自己的

调用虚函数表中的函数，需要得到虚函数表中函数的地址，虚函数表中都是虚函数的地址

多态时，基类的纯虚指针指向子类的相应虚函数表的位置

### 抽象类

不能被实例化的类（类中有纯虚函数就不能被实例化）

纯虚函数：virtual void ab()=0;

除了不能被实例化，和其他类无区别

只能用于继承后，使用多态的方式实现出抽象类

A \*a=new B(); A为抽象类

子类继承抽象类时，如果不实现纯虚函数（定义同名函数，将纯虚函数隐藏），则子类也是抽象类

### 异常

当throw出现时，程序会直接跳转到catch处（一直向函数上层搜索catch块，如果没有catch则直接崩溃）

throw 处会创建或复制一个对象，throw上面所有变量进行了析构，然后在该处调用catch函数

class A{ public: int x; A(int a){x = a;} };

try

{

int condition = 2; //根据condition，抛出不同类的对象

if(1 == condition){ throw 1;}

else if(2 == condition){throw A(2); }

}catch(int i)//接收抛出的对象

{ cout<<i<<endl;

}catch(A &a)

{ cout<<a.x<<endl;

}catch(...)//接收前面catch没有枚举出的异常类型

{ cout<<"other exception"<<endl;

}

### C++11

\_\_cplusplus c++编译器版本

\_\_func\_\_ 当前函数名,可用于初始化参数列表标志类名

constexpr int a = 10; //编译时确定值

static\_assert(8 == sizeof(void\*), "指针长度错误");//编译时确保指针长度为8

void fun()noexcept(true){} 表示函数不抛出异常，如果有异常，结束程序，阻止异常扩散

成员变量定义时可以初始化

template<typename T> class A

{

friend T;

};

A<B> a; 可以在B中访问A的私有成员

using INT = int;

class A final 类不能被继承

virtual void fun()final 虚函数不允许重载

void fun()override 显示声明该函数是对基类函数重载

int a[] = {1,2,4,6,7,8};

for (auto i : a)

{

if (i == 6) break;

cout << i << endl;

}

## c++StdLib

### 流

#### I/O流

cin、cout都属于<iostream>中类的对象

类 对象

istream cin;

ostream cout;

cout可以被重定向，命令行启动程序时加入：

>>a.txt 可以将cout内容输出到a.txt，不在屏幕显示

cerr不带缓冲，直接输出到屏幕

<iostream>

io默认是格式化，会忽略空格和换行，>>对输入进行格式化，使之与特定数据类型匹配

流操作在两个数据之间不论多少空格或者换行，都会忽略

cout<<x<<endl; 等价于 cout<<x;cout<<endl;

因为<<流操作符返回的是左边的对象，即执行完后返回cout。

cout<<cin<<endl; 流不出错时输出一个地址，出错会输出0，拒绝io，但不影响其他操作，输入流会出错的情况：

cin.getline(char \*p, size\_t len);最多能接收len-1个字符，与\*p内存大小无关，输入个数超过时输入流会出错，此时无法再接受输入

.clear();纠正流状态，让出错的流变正常，如文件到末尾，不会清空流

.ignore(n,’\n’)；清空缓冲区，最多清空n个字符，当遇到’\n’立即结束(如果不加入’\n’，会一直等待输入区输入，直到清空够n个字节)

exp：

if(!cin){

cin.clear();

cin.ignore(200,’\n’);

}

将结束符去掉，剩余内容留在缓冲区，以字符串形式输入到data(格式化输入)（该函数型式只能用cin）cin相当于输入缓冲区

.get() 读一个字符 exp: c=cin.get();

.put() 写一个字符 cout.put(c);

//输出时只保留小数点后两位

#include <iomanip>

double x = 21.2323;

cout<<fixed<<showpoint<<setprecision(2)<<x<<endl;

#### 文件流

<fstream>

只读 ifstream 文件必须存在

只写ofstream 文件可以不存在，存在则清空

读写fstream 可以指定模式

fstream iof("D:\\1.txt",fstream::out|fstream::app);每次写之前定位到末尾）

fstream iof("a.txt"); 构造函数直接打开文件，也可使用.open()函数打开，打开时也可指定模式，不指定模式时，文件必须存在

if(!iof) return; 判断文件未打开

.close() 关闭文件

==========读取所有数据

std::ifstream ifs("D:\\1.txt");

string strval;

std::getline(ifs, strval, (char)EOF);

==========逐行操作

iof<<"hello"<<endl<<"abc"<<endl;

向文件中写入两行内容（读/写指针移动）

iof>>str1>>str2; 从文件中读出两行内容

(空格或者换行自动分，两段内容之间不论多少空格或者换行)

string strline;

while (getline(ifs,strline))

char str[100] = { 0 };

while (ifs.getline(str, 100))

==========逐个操作

char c；

while((c=iof.get()!=EOF))

cout<<c<<endl;

逐个输出全部字符（读入前调整指针）

==========文件指针操作

.seekp(n,参考位置) 从参考位置调整写指针位置(大小为n)

参考位置： ios::beg ios::cur ios::end .tellp()写指针当前位置

seekg、tellg用于读指针，seekp、tellp用于写指针。

fstream中，读写指针一体，任意用一个即可，在ifstream中只能用读指针，ofstream中只能用写指针

======获取文件大小

iof.seekg(0,ios::end);

fileLen = iof.tellg();

iof.seekg(0,ios::beg);

===========二进制读写

ifstream ifile(filePath,ifstream::binary);

.write iof.write(char\*,n)

从str中向文件写入n个字符

.read iof.read(char\*,n)

从文件中读出n个字符放到str中

.gcount

返回最近一次非格式化读取的字符的数量

如read函数时n为10，但只有6个字节，则返回6

如果没有读取到内容，返回0

#### 字符流

<sstream>

istringstream 从流中读出字符串

ostringstream 向流中写入字符串，相当于sprintf

wostringstream对应wsting

stringstream同时具备istringstream和ostringstream功能

ostringstream ostr; //构造ostringstream类的对象

string name;

int age;

ostr<<name<< " "<<age;

string str=ostr.str(); //构造出string

istringstream istr(str); //构造istringstream类的对象

istr>>name>>age; //空格或者换行自动分

清空操作：ss.str("");

#### 流操作

>>操作是格式化按照字段之间的空格或换行来区分

getline(stream, string , ‘结束符’)；

Steam 可以是cin，也可以是stringsream，也可以是fstream

结束符代表读取结束的位置，可以不使用，默认为‘\n’

### <string>

using namespace std;

wstring 需要使用wcout和wcin

使用前需要进行初始化：locale::global(locale(""));

或者wcout.imbue(locale(""));wcin.imbue(locale(""));

#### wstring之间转换

string str = "abc";

wstring wstr(str.length(),L' ');

std::copy(str.begin(), str.end(), wstr.begin());

#### 函数实现

class string

{

public:

char \*m\_data;

string(const char \*p=NULL)

{

if(p!=NULL)

{

int len=strlen(p);

m\_data=new char[len+1];

strcpy(m\_data,p);

}

else

{

m\_data=new char;

\*m\_data='\0';

}

}

string(const string& other)

{

int len=strlen(other.m\_data);

m\_data=new char[len+1];

strcpy(m\_data,other.m\_data);

}

~string(void)

{

delete[] m\_data;

}

string& operator=(const string& other)

{

if(this==&other)

return \*this;

else

{

delete[] m\_data;

int len=strlen(other.m\_data);

m\_data=new char[len+1];

strcpy(m\_data,other.m\_data);

}

}

};

#### 输入输出

string类重载运算符operator>>用于输入，同样重载运算符operator<<用于输出操作。

getline(istream &in,string &s);用于从输入流in中读取一行字符串到s中，以换行符'\n'分开（可以读入空格，cin>>s遇到空格会截断字符串）

输入ab cd

getline(cin,str); //得到str为ab cd

cin>>str1>>str2;//得到str1为ab，str2为cd

#### 构造函数

当构造的string太长而无法表达时会抛出length\_error异常

string(const char \*s); //用c字符串s初始化

string(int n,char c); //用n个字符c初始化

string s1；

string s2="hello"；

#### 字符操作

const char &operator[](int n)const;

const char &at(int n)const;

operator[]和at()均返回当前字符串中第n个字符的位置，但at函数提供范围检查，当越界时会抛出out\_of\_range异常，下标运算符[]不提供检查访问。

const char \*c\_str()const;//返回一个以null终止的c字符串

const char \*data()const;//返回一个非null终止的c字符数组

如果string类析构了，就会指向垃圾数据（必要时使用strcpy）

int 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

#### 特性描述

int size()const; int length()const; //返回当前字符串的长度

bool empty()const; //当前字符串是否为空

void 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

int capacity()const; //返回当前容量（即string中不必增加内存即可存放的元素个数）

max\_size()const; //返回string对象中可存放的最大字符串的长度，结果是一个很大的数值

#### 子串

string substr(int pos = 0,int n = npos) const;//返回pos开始的n个字符组成的字符串，n默认值时代表子串是pos到末尾。

#### 查找

string::npos可以指代-1，当未查找到时，可以用来判断返回值。

pos = s.find(str,int pos = 0);//在s中查找str第一次出现的位置。默认从下标为0 的位置开始查找

s.rfind(str);//在s中查找str最后一次出现

s.find\_first\_of(str);//在s中查找str中任意字符第一次出现

s.find\_last\_of(str);//在s中查找str中任意字符最后一次出现

s.find\_first\_not\_of(str);//在s中查找第一个不属于str的字符

s.find\_last\_not\_of(str);//在s中查找最后一个不属于str的字符

#### 替换

s. replace(pos,len,str);//从pos位置开始的len个字符替换为str

s.replace(itA,itB,str);//将迭代器AB之间的字符删除替换为str

#### 插入

string &insert(int p0, const char \*s);

string &insert(int p0, const char \*s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);

//前4个函数在p0位置插入字符串s中pos开始的前n个字符

string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c

iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置

void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在it处插入n个字符c

#### 删除

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置

iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置

string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串

### <typeinfo>

可以进行运行时类型识别

typeid(变量或类型)

可以获得对象的类型信息

string typeName = typeid(int).name();

B \*pb = new B(); A \*pa = pb;

cout<<typeid(\*pa).name()<<endl;

if(typeid(\*pa)==typeid(\*pb)){cout<<"=="<<endl;}

typeid和dynamic\_cast<>()在转换成相应的子类类型时，

有继承关系/需要父类有虚函数

### <ctime>

std::clock\_t timeStart = std::clock();//记录当前时间

。。。。。。。。

double t = double(std::clock() - timeStart)/CLOCKS\_PER\_SEC;

//计算程序运行到此处运行的时间，单位是秒

### 线程

#include <thread>

void thr(int i,double cc){}//线程入口函数与参数个数任意，参数必须是可以拷贝的类型

thread t1(thr, 10,20);//创建并运行线程

t1.join();//如果在下一个thread定义前调用，可以确保下一个线程在这个线程运行结束后运行

atomic<A\*> aa = new A();//定义原子类型

A\* p = \*pa;//线程中访问

## DataStructure

### 链表

由多个节点组成，每个节点中包括有效数据和至少一个节点指针

定义链表头结点时：

node \*head=new node;

head->next=NULL;

判断时：

node \*n=head->next;

while(n)

删除时：

void destroy(NODE \*head){

NODE \*temp=head->next;

while(temp){

head->next=temp->next;

delete temp;

temp=head->next;

}

}

### 堆栈

先进后出，同一端进行插入、删除

1）数组：

void stack[N]；

int size; 有效数据个数，作为数组下标

size=0相当于清空栈

2）链表：头节点 ...........................

头节点定位，在头结点后的位置进行删除和插入

### 队列

先进先出，一端用来插入，一端用来取出

1）数组：

void queue[n];

int head,tail,size;

从tail处插入，从head处取出，取出时先判断

head<tail,用size判断是否为空

push：queue[tail]=data;

tail=(tail+1)%n;

size++;

pop: head=(head+a)%n;

size--;

2)链表：头节点 ................

头节点定位，在头结点后取出，在链表尾插入

### 树

通常包括节点声明和根指针说明，逻辑结构通常用链式存储方式实现

二叉树：每个节点最多有两个子节点，p\_left,p\_right

有序二叉树：左比根小，右比根大

5

2 8

4 6 10

二叉树遍历：

1. 前序（根，左，右）

从根开始找，有左时，把左当成根再找，无左时把右当成根再找

1. 中序（左，根，右）
2. 后序（左，右，根）

中序可以用投影：

A

B G

C D H I

E F

C B E D F A H G I

从c开始找，再找到b,再到右边d时，把def当整体，所以先到左的e，再到根d

node \*\* pp\_root;

node \* p\_root;

\*pp\_root=NULL; 初始值

\*pp\_root=p\_root; 赋值

void destroy(node \*\*pp\_root){

if(\*pp\_root){

destroy(&((\*pp\_root)->p\_left));

destroy(&((\*pp\_root)->p\_right));

free(\*pp\_root);

\*pp\_root=NULL;

}

}

void show(node \*p\_root){

if(p\_root){

show(p\_root->p\_left);

printf(“%d\n”,p\_root->num);

show(p\_root->p\_right);

}

}

### 查找

int a[N]

查找：顺序查找、折半查找(用已排序好的数组)

折半查找：每次与中间值进行比较，再决定到前一半或者或一半进行再次查找。

p+N/2

奇数

N/2 N/2

p+N/2

偶数

N/2 N/2-1

int search(int \*a, int N, int num){

if(N>0){

if(a[N/2]==num) return 1; 表示找到

else if(a[N/2]>num) return search(a,N/2,num);

else{

if(N%2)

return search(a+N/2+1,N/2,num);

else

return search(a+N/2+1,N/2-1,num);

}

}

else return -1; 未找到

}

### 排序

排序时候千万不要把循环内的i，j写错，否则会出现死循环

int a[N]

#### 冒泡排序

每次循环将最小值放到最前面的位置

for(i=0;i<N-1;i++){

for(j=i+1;j<N;j++){ 每次外循环把最小数放前面

if(a[i]>a[j]) 内循环将后面每个数与外循环数比较

swap(a[i],a[j]);

}

}

#### 选择排序

冒泡基础上加一个用来选择的下标变量，每次循环找出最小元素的下标，将这个元素与最前面的进行交换。

for(int i=0;i<n-1;i++)

{

int min\_index=i; //用来存放每一圈里面的最小值的下标

for(int j=i+1;j<n;j++)//每次j一圈时扫描选择出最小项

if(arr[j]<arr[min\_index]) min\_index=j;

if(min\_index!=i)//用最小项交换，即将这一项移到列表中的正确位置

{

int temp;

temp=arr[i]; arr[i]=arr[min\_index]; arr[min\_index]=temp;

}

}

#### 插入排序

把前面当成已排好的数组，进行插入。循环从下标1开始，每次循环都是一位一位回退和插入，直到遇到前面大的（不需要插入的位置）。

for(j=1;j<N;j++){ 每次外循环将当前数插入到前面

num=a[j]; 存储外循环上的数，用来插入到前面，同时用于比较

for(i=j-1;i>=0;i--){ 内循环找到插入位置

if(a[i]>num) {

a[i+1]=a[i];

如果前面大，则将其向后移动，留出插入位置

a[i]=num; 插入到留出的位置

}

else break; 如果前面没有比num大，退出循环

}

}

#### 快速排序

每次循环先判断数组元素数量大于0，使用一个基准数num，一个头指针和一个尾指针，num这个数在头指针和尾指针之间移动，头尾指针向中间移动。循环结束后num之前的都比num小，之后额都比num大

void quick\_sort(int \*a, int N) {

if (N < 2) return;

int \*p\_start = a;

int \*p\_end = a + N - 1;

int num = \*p\_start; //基准数num为\*p\_start的值

while (p\_end > p\_start) {

if (\*p\_start > \*p\_end) swap(\*p\_start, \*p\_end);

//num在p\_start和p\_end的指向位

//置上来回移动，两个指针向中间靠拢

//移动那个没有指向num的指针

if (\*p\_start != num) p\_start++;

else p\_end--;

}// 循环结束后，p\_start和p\_end在同一位置上，且num位置排好

quick\_sort(a, p\_start - a); //排num之前的数组，计算出的数组中

quick\_sort(p\_start + 1, (a + N - 1) - p\_start); //排num之后的数组 数量也可写n - (p\_start - a + 1)

}

#### 库函数排序

int fun(const void \*a,const void \*b)

{

return \*(int\*)a - \*(int\*)b; //升序

return \*(int\*)b - \*(int\*)a;//降序

}

int a[10]={2,343,23,433,231,43,23,22,12,1};

qsort(a,10,sizeof(int),fun);

排序函数中也可以写：

if(\*(int\*)a > \*(int\*)b)

return 1; //返回1代表进行排序

return -1;

### 递归

递归函数可以将函数中的数据分成若干分支，其中一定包含一种不需要调用递归函数的情况（结束深层次调用情况），剩余则是需要将数据进行深层次调用递归函数情况

Fun

结果1 结果temp1（调用fun）

结果2 结果temp2（调用fun）

结果3 结果4

递归求阶乘

double fun(int i){

if(i<=1) return 1.0;

return i\*fun(i-1);

}

### 算法

二进制

1 2 4 8......................

可以相加组成8之前的所有数1，2，3，4，5，6，7，8

得到数字没一位：取余，取整

exp: 532 a=532%10

b=532%100/10

c=532/100

关于10取余可以得到10以内的数

判断一个整数二进制第三位置上是0还是1

（data&4）==4 满足条件则是1

与运算，4的二进制为100

取八进制的后四位

a%07777 a%11 取0到11内的数

闰年

%4==0&&%100!=0||%400==0

奇数

1. %2!=0
2. 等差数列 1、3、5...............（i+=2）
3. 通项公式 i\*2-1或者i\*2+1

num=num\*10+i

从高位数字开始，逐个加入数字组成数

统计任意输入数量数据的和

while(1){

cin>>a;

if(a==0) break;

sum+=a;

}

求整数位数

do{

a[data%10]++; 数字0到9，每出现一次，对应下标内的值加1

data/=10;

}while(data!=0);

不断向一个数字加入尾数：

i=i\*10+a;

判断一个字符是否是数字：

ch>='0'&&ch<='9'

1 2 3 5 7 13.............

。 。 。 second+=first

first=second-first p变为p1

p p1

a (+7)%10 b 加密

b (+3)%10 a 解密

计算一个数由两个数相加组成 i<=a/2

计算一个数由两个数相乘组成 i<=sqrt(a)

水仙花数： n>=3 153=1^3+5^3+3^3

a取最后八位： a&0xFF

最大int： 0x7F FF FF FF

0111 1111

最高位为符号位

### 原码/反码/补码

数字在存储时是二进制，对于有符号的类型，最高位0代表是正数，1代表是负数

反码为原码逐位取反，补码为反码加1

负数在计算和存储时，是使用补码进行的，-x为~x+1，这样有利于在计算机中对两个数进行加法运算。

正数+负数 = 模 （模就是该类型最大值）

-5+10的运算过程（假设使用8位表示数）

0000 0101 （5的二进制）

1111 1011 （-5的二进制）

0000 1010 （10的二进制）

相加过程：

1111 1011 （-5）

+ 0000 1010 （10）

= 1 0000 0101

去掉最高位溢出的1，得到0000 0101也就是5

### 线性回归方程

void Line(double\* pBufX,double\* pBufY, const int nLen)

{

double x,y,A=0.0,B=0.0,C=0.0,D=0.0,delta;

for(int i=0;i<nLen;++i)

{

x = pBufX[i];

y = pBufY[i];

A+=x\*x;

B+=x;

C+=x\*y;

D+=y;

}

delta = A\*nLen-B\*B;

if(delta < PRECISION && delta > -PRECISION)

{

TRACE("Divide Zero");

return ;

}

double k = (C\*nLen-B\*D)/delta;

double b = (A\*D-C\*B)/delta;

}