1. 指针和引用的区别

指针指向一块内存，它的内容是指向内存的地址；引用是某内存的别名

引用使用是无需解引用，指针需解引用

引用不能为空，指针可以为空

引用在定义是被初始化一次，之后不可变；指针可变

程序为指针变量分配内存区域，而引用不需要分配内存区域

1. memcpy和strcpy的区别

memcpy用来内存拷贝的，它有指定的拷贝数据长度，他可以拷贝任何数据类型的对象

Strcpy它只能去拷贝字符串，它遇到’\0′结束拷贝

1. new和malloc的区别，free和delete的区别

malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。

对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。

因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

1. struct和class的区别

1.成员变量

结构在默认情况下的成员是公共(public)的,

而类在默认情况下的成员是私有(private)的。

2.存储

struct保证成员按照声明顺序在内存中存储。class不保证等等

3.继承

struct A { }；

class B : A{ }; //private继承

struct C : B{ }； //public继承

这是由于class默认是private，struct默认是public。

1. struct与union的区别.(一般假定在32位机器上)

1．一个union类型的变量，所有成员变量共享一块内存，该内存的大小有这些成员变量中长度最大的一个来决定，struct中成员变量内存都是独立的

2．union分配的内存是连续的，而struct不能保证分配的内存是连续的

1. 队列和栈有什么区别？

队列先进先出，栈后进先出

1. 指针在16位机、32位机、64位机分别占用多少个字节

16位机        2字节

32位机        4字节

64位机        8字节

1. 如何引用一个已经定义过的全局变量？

extern

可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错

1. 全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

可以，在不同的C文件中以static形式来声明同名全局变量。

可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错

1. 语句for( ；1 ；)有什么问题？它是什么意思？

for( ；1 ；)和while(1)相同。

1. do……while和while……do有什么区别？

前一个循环一遍再判断，后一个判断以后再循环

1. 请写出下列代码的输出内容

#include<stdio.h>

main()

{

int a,b,c,d;

a=10;

b=a++;

c=++a;

d=10\*a++;

printf(“b，c，d：%d，%d，%d”，b，c，d）;

return 0;

}

10，12，120

1. 对于一个频繁使用的短小函数,在C语言中应用什么实现,在C++中应用什么实现?

c用宏定义，c++用inline

1. main 函数执行以前，还会执行什么代码？

全局对象的构造函数会在main 函数之前执行。

1. main 主函数执行完毕后，是否可能会再执行一段代码，给出说明？

可以，可以用on\_exit 注册一个函数，它会在main 之后执行int fn1(void), fn2(void), fn3(void), fn4 (void);

void main( void )

{

String str(“zhanglin”);

on\_exit( fn1 );

on\_exit( fn2 );

on\_exit( fn3 );

on\_exit( fn4 );

printf( “This is executed first.\n” );

}

int fn1()

{

printf( “next.\n” );

return 0;

}

int fn2()

{

printf( “executed ” );

return 0;

}

int fn3()

{

printf( “is ” );

return 0;

}

int fn4()

{

printf( “This ” );

return 0;

}

The on\_exit function is passed the address of a function (func) to be called when the program terminates normally. Successive calls to on\_exit create a register of functions that are executed in LIFO (last-in-first-out) order. The functions passed to on\_exit cannot take parameters.

This is executed next.

1. 局部变量能否和全局变量重名？

能，局部会屏蔽全局。要用全局变量，需要使用”::”

局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内

1. 描述内存分配方式以及它们的区别?

1．从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2． 在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。

3． 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

1. 类成员函数的重载、覆盖和隐藏区别？

1.成员函数被重载的特征：

（1）相同的范围（在同一个类中）；

（2）函数名字相同；

（3）参数不同；

（4）virtual 关键字可有可无。

2.覆盖是指派生类函数覆盖基类函数，特征是：

（1）不同的范围（分别位于派生类与基类）；

（2）函数名字相同；

（3）参数相同；

（4）基类函数必须有virtual 关键字。

3.”隐藏”是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

（1）如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。

（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual 关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）

1. static有什么用途？（请至少说明两种）

1.限制变量的作用域

2.设置变量的存储域

1. 请说出const与#define 相比，有何优点？

1．const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

2．有些集成化的调试工具可以对const 常量进行调试，但是不能对宏常量进行调试。

堆栈溢出一般是由什么原因导致的？

没有回收垃圾资源

1. 简述数组与指针的区别？

数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。

(1)修改内容上的差别

char a[] = “hello”;

a[0] = ‘X’;

char \*p = “world”; // 注意p 指向常量字符串

p[0] = ‘X’; // 编译器不能发现该错误，运行时错误

(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

char a[] = “hello world”;

char \*p = a;

cout<< sizeof(a) << endl; // 12 字节

cout<< sizeof(p) << endl; // 4 字节

计算数组和指针的内存容量

void Func(char a[100])

{

cout<< sizeof(a) << endl; // 4 字节而不是100 字节

}

1. There are two int variables: a and b, don’t use “if”, “? :”, “switch”or other judgement statements, find out the biggest one of the two numbers.

( ( a + b ) + abs( a – b ) ) / 2

1. 冒泡排序算法的时间复杂度是什么？

O(n^2)

1. 什么函数不能声明为虚函数？

构造函数（Constructor）

1. 变量在内存中存放的位置

全局变量                    全局静态区

全局静态变量                全局静态区

全局常量

        有初始化            代码区

        无初始化            全局静态区

局部变量                    堆栈区

局部静态变量                静态区

局部常量                    堆栈区

    new和malloc分配空间    堆区

1. 进程间通信方式

管道（有名管道，无名管道），共享内存，消息队列，信号量，socket通信

1. 线程同步方式

临界区：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问

互斥量：为协调共同对一个共享资源的单独访问而设计

信号量（PV操作）：为控制一个具有有限数量用户资源而设计

事件：用来通知线程有一些事件已

1. 进程和线程的区别

资源：进程是拥有资源的一个独立单位，线程是不拥有资源。

调度：线程作为调度和分配的基本单位，进程是作为资源的基本单位

并发性：进程之间可以有并发性进行，同一个进程中的多个线程是可以并发执行

系统开销：进程在创建和撤销的时候，由于系统要分配和回收资源，导致系统的开销明显大于线程

一个进程可以拥有多个线程。

1. 局部变量和全局变量能否重名

能，局部屏蔽全局。在C++里使用全局，需要使用”：：”。在C语言里，extern

1. 虚函数和纯虚函数的区别

虚函数必须实现，纯虚函数没有实现

虚函数在子类里可以不重载，但是纯虚函数必须在每一个子类里去实现

在动态内存分配的时候，析构函数必须是虚函数，但没有必要是纯虚函数

1. 面向对象的三大特性（四大特性）

封装、继承、多态（抽象）

封装：把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏

继承：子类可以拥有父类的属性和方法，但父类没有子类的属性和方法

多态：允许将子类类型的指针赋值给父类类型的指针

实现多态，有二种方式，覆盖，重载

覆盖，是指子类重新定义父类的虚函数的做法

重载，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）

1. vi编辑器打开时跳到指定的行

vi +5000 filename

1. int型在Touble C里占多少个字节

2个字节

1. 判断一个单链表是否有环

两个指针指向链表头，一个指针每次走一步，另一个指针每次走两步，若有一个指针先指向为NULL表示这个链表无环。若两个指针重合表示链表有环

1. 刷新缓冲区方式?

换行刷新缓冲区

printf(“\n”);

使用函数刷新缓冲区

fflush(stdout);

程序结束刷新缓冲区

return 0;

1. 类和对象的两个基本概念什么？

对象就是对客观事物在计算机中的抽象描述。

类就是对具体相似属性和行为的一组对象的统一描述。

类的包括：类说明和类实现两大部分：

类说明提供了对该类所有数据成员和成员函数的描述。

类实现提供了所有成员函数的实现代码。

1. 数据库三范式

第一范式：没有重复的列

第二范式：非主属的部分依赖于主属部分

第三范式：属性部分不依赖于其他非主属部分

1. ASSERT( )是干什么用的

是在调试程序使用的一个宏，括号里面要满足，如果不满足，程序将报告错误，并将终止执行。

1. 如果只想让程序有一个实例运行，不能运行两个。像winamp一样，只能开一个窗口，怎样实现？

用内存映射或全局原子（互斥变量）、查找窗口句柄

FindWindow，互斥，写标志到文件或注册表,共享内存

1. 如何截取键盘的响应，让所有的’a’变成’b’？

键盘钩子SetWindowsHookEx

1. 网络编程中设计并发服务器，使用多进程 与 多线程 ，请问有什么区别？

1．进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。

2．线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。

两者都可以提高程序的并发度，提高程序运行效率和响应时间。

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。

**编程**

**字符串实现**

**strcat**

char \*strcat(char \*strDes, const char \*strSrc)

{

assert((strDes != NULL) && (strSrc != NULL));

char \*address = strDes;

while (\*strDes != ‘\0′)

++ strDes;

while ((\*strDes ++ = \*strSrc ++) != ‘\0′)

NULL;

return address;

}

**strncat**

char \*strncat(char \*strDes, const char \*strSrc, int count)

{

assert((strDes != NULL) && (strSrc != NULL));

char \*address = strDes;

while (\*strDes != ‘\0′)

++ strDes;

while (count — && \*strSrc != ‘\0′ )

\*strDes ++ = \*strSrc ++;

\*strDes = ‘\0′;

return address;

}

**strcmp**

int strcmp(const char \*s, const char \*t)

{

assert(s != NULL && t != NULL);

while (\*s && \*t && \*s == \*t)

{

++ s;

++ t;

}

return (\*s – \*t);

}

**strncmp**

int strncmp(const char \*s, const char \*t, int count)

{

assert((s != NULL) && (t != NULL));

while (\*s && \*t && \*s == \*t && count –)

{

++ s;

++ t;

}

return (\*s – \*t);

}

**strcpy**

char \*strcpy(char \*strDes, const char \*strSrc)

{

assert((strDes != NULL) && (strSrc != NULL));

char \*address = strDes;

while ((\*strDes ++ = \*strSrc ++) != ‘\0′)

NULL;

return address;

}

**strncpy**

char \*strncpy(char \*strDes, const char \*strSrc, int count)

{

assert(strDes != NULL && strSrc != NULL);

char \*address = strDes;

while (count — && \*strSrc != ‘\0′)

\*strDes ++ = \*strSrc ++;

return address;

}

**strlen**

int strlen(const char \*str)

{

assert(str != NULL);

int len = 0;

while (\*str ++ != ‘\0′)

++ len;

return len;

}

**strpbrk**

char \*strpbrk(const char \*strSrc, const char \*str)

{

assert((strSrc != NULL) && (str != NULL));

const char \*s;

while (\*strSrc != ‘\0′)

{

s = str;

while (\*s != ‘\0′)

{

if (\*strSrc == \*s)

return (char \*) strSrc;

++ s;

}

++ strSrc;

}

return NULL;

}

**strstr**

char \*strstr(const char \*strSrc, const char \*str)

{

assert(strSrc != NULL && str != NULL);

const char \*s = strSrc;

const char \*t = str;

for (; \*t != ‘\0′; ++ strSrc)

{

for (s = strSrc, t = str; \*t != ‘\0′ && \*s == \*t; ++s, ++t)

NULL;

if (\*t == ‘\0′)

return (char \*) strSrc;

}

return NULL;

}

**strcspn**

int strcspn(const char \*strSrc, const char \*str)

{

assert((strSrc != NULL) && (str != NULL));

const char \*s;

const char \*t = strSrc;

while (\*t != ‘\0′)

{

s = str;

while (\*s != ‘\0′)

{

if (\*t == \*s)

return t – strSrc;

++ s;

}

++ t;

}

return 0;

}

**strspn**

int strspn(const char \*strSrc, const char \*str)

{

assert((strSrc != NULL) && (str != NULL));

const char \*s;

const char \*t = strSrc;

while (\*t != ‘\0′)

{

s = str;

while (\*s != ‘\0′)

{

if (\*t == \*s)

break;

++ s;

}

if (\*s == ‘\0′)

return t – strSrc;

++ t;

}

return 0;

}

**strrchr**

char \*strrchr(const char \*str, int c)

{

assert(str != NULL);

const char \*s = str;

while (\*s != ‘\0′)

++ s;

for (– s; \*s != (char) c; — s)

if (s == str)

return NULL;

return (char \*) s;

}

**strrev**

char\* strrev(char \*str)

{

assert(str != NULL);

char \*s = str, \*t = str, c;

while (\*t != ‘\0′)

++ t;

for (– t; s < t; ++ s, — t)

{

c = \*s;

\*s = \*t;

\*t = c;

}

return str;

}

**strnset**

char \*strnset(char \*str, int c, int count)

{

assert(str != NULL);

char \*s = str;

for (; \*s != ‘\0′ && s – str < count; ++ s)

\*s = (char) c;

return str;

}

**strset**

char \*strset(char \*str, int c)

{

assert(str != NULL);

char \*s = str;

for (; \*s != ‘\0′; ++ s)

\*s = (char) c;

return str;

}

**strtok**

char \*strtok(char \*strToken, const char \*str)

{

assert(strToken != NULL && str != NULL);

char \*s = strToken;

const char \*t = str;

while (\*s != ‘\0′)

{

t = str;

while (\*t != ‘\0′)

{

if (\*s == \*t)

{

\*(strToken + (s – strToken)) = ‘\0′;

return strToken;

}

++ t;

}

++ s;

}

return NULL;

}

**strupr**

char \*strupr(char \*str)

{

assert(str != NULL);

char \*s = str;

while (\*s != ‘\0′)

{

if (\*s >= ‘a’ && \*s <= ‘z’)

\*s -= 0×20;

s ++;

}

return str;

}

**strlwr**

char \*strlwr(char \*str)

{

assert(str != NULL);

char \*s = str;

while (\*s != ‘\0′)

{

if (\*s >= ‘A’ && \*s <= ‘Z’)

\*s += 0×20;

s ++;

}

return str;

}

**memcpy**

void \*memcpy(void \*dest, const void \*src, int count)

{

assert((dest != NULL) && (src != NULL));

void \*address = dest;

while (count –)

{

\*(char \*) dest = \*(char \*) src;

dest = (char \*) dest + 1;

src = (char \*) src + 1;

}

return address;

}

**memccpy**

void \*memccpy(void \*dest, const void \*src, int c, unsigned int count)

{

assert((dest != NULL) && (src != NULL));

while (count –)

{

\*(char \*) dest = \*(char \*) src;

if (\* (char \*) src == (char) c)

return ((char \*)dest + 1);

dest = (char \*) dest + 1;

src = (char \*) src + 1;

}

return NULL;

}

**memchr**

void \*memchr(const void \*buf, int c, int count)

{

assert(buf != NULL);

while (count –)

{

if (\*(char \*) buf == c)

return (void \*) buf;

buf = (char \*) buf + 1;

}

return NULL;

}

**memcmp**

int memcmp(const void \*s, const void \*t, int count)

{

assert((s != NULL) && (t != NULL));

while (\*(char \*) s && \*(char \*) t && \*(char \*) s == \*(char \*) t && count –)

{

s = (char \*) s + 1;

t = (char \*) t + 1;

}

return (\*(char \*) s – \*(char \*) t);

}

**memmove**

void \*memmove(void \*dest, const void \*src, int count)

{

assert(dest != NULL && src != NULL);

void \*address = dest;

while (count –)

{

\*(char \*) dest = \*(char \*) src;

dest = (char \*) dest + 1;

src = (const char \*)src + 1;

}

return address;

}

**memset**

void \*memset(void \*str, int c, int count)

{

assert(str != NULL);

void \*s = str;

while (count –)

{

\*(char \*) s = (char) c;

s = (char \*) s + 1;

}

return str;

}

**strdup**

char \*strdup(const char \*strSrc)

{

assert(strSrc != NULL);

int len = 0;

while (\*strSrc ++ != ‘\0′)

++ len;

char \*strDes = (char \*) malloc (len + 1);

while ((\*strDes ++ = \*strSrc ++) != ‘\0′)

NULL;

return strDes;

}

**strchr\_**

char \*strchr\_(char \*str, int c)

{

assert(str != NULL);

while ((\*str != (char) c) && (\*str != ‘\0′))

str ++;

if (\*str != ‘\0′)

return str;

return NULL;

}

**strchr**

char \*strchr(const char \*str, int c)

{

assert(str != NULL);

for (; \*str != (char) c; ++ str)

if (\*str == ‘\0′)

return NULL;

return (char \*) str;

}

**atoi**

int atoi(const char\* str)

{

    int x=0;

    const char\* p=str;

    if(\*str==’-’||\*str==’+’)

    {

     str++;

    }

    while(\*str!=0)

    {

        if((\*str>’9′)||(\*str<’0′))

        {

         break;

        }

        x=x\*10+(\*str-’0′);

        str++;

    }

    if(\*p==’-’)

    {

        x=-x;

    }

    return x;

}

**itoa**

char\* itoa(int val,char\* buf,unsigned int radix)

{

char \*bufptr;

char \*firstdig;

char temp;

unsigned int digval;

assert(buf != NULL);

bufptr = buf;

if (val < 0)

{

\*bufptr++ = ‘-’; val = (unsigned int)(-(int)val);

}

firstdig = bufptr;

do

{

digval =(unsigned int) val % radix; val /= radix;

if (digval > 9)

{

\*bufptr++ = (char)(digval – 10 + ‘a’);

}

else

{

\*bufptr++ = (char)(digval + ’0′);

}

} while(val > 0);

\*bufptr– = ‘\0′;//设置字符串末尾，并将指针指向最后一个字符

do //反转字符

{

temp = \*bufptr; \*bufptr = \*firstdig; \*firstdig = temp;

–bufptr; ++firstdig;

} while(firstdig < bufptr);

return buf;

}

**String实现**

已知String原型为：

class String

{

public:

//普通构造函数

String(const char \*str = NULL)

//拷贝构造函数

String(const String &other)

//析构函数

~String(void);

//赋值函数

String & operator=(String &other) //oh,原题目打错了,string可是一个关键字

private:

char\* m\_str;

unsigned m\_uCount;

};

分别实现以上四个函数

//普通构造函数

String::String(const char\* str)

{

    if(str==NULL)                //如果str为NULL，存空字符串

{

        m\_str = new char[1];        //分配一个字节

        \*m\_str = ‘\0′;            //赋一个’\0′

}else

    {

        m\_str = new char[strlen(str) + 1];//分配空间容纳str内容

        strcpy(m\_str, str);         //复制str到私有成员m\_str中

    }

}

//析构函数

String::~String()

{

    if(m\_str!=NULL)    //如果m\_str不为NULL，释放堆内存

{

        delete [] m\_str;

        m\_str = NULL;

}

}

//拷贝构造函数

String::String(const String &other)

{

    m\_str = new char[strlen(other.m\_str)+1];    //分配空间容纳str内容

    strcpy(m\_str, other.m\_str);            //复制other.m\_str到私有成员m\_str中

}

//赋值函数

String & String::operator=(String &other)

{

    if(this == &other)                //若对象与other是同一个对象，直接返回本身

{

        return \*this

}

    delete [] m\_str;                //否则，先释放当前对象堆内存

    m\_str = new char[strlen(other.m\_str)+1];    //分配空间容纳str内容

    strcpy(m\_str, other.m\_str);            //复制other.m\_str到私有成员m\_str中

    return \*this；

}

**编写一个二分查找的功能函数**

int BSearch(elemtype a[],elemtype x,int low,int high)

/\*在下届为low，上界为high的数组a中折半查找数据元素x\*/

{

int mid;

if(low>high)

return -1;

mid=(low+high)/2;

if(x==a[mid])

return mid;

if(x<a[mid])

return(BSearch(a,x,low,mid-1));

else

return(BSearch(a,x,mid+1,high));

}

2) 非递归方法实现：

int BSearch(elemtype a[],keytype key,int n)

{

int low,high,mid;

low=0;high=n-1;

while(low<=high)

{

mid=(low+high)/2;

if(a[mid].key==key)

return mid;

else if(a[mid].key<key)

low=mid+1;

else

high=mid-1;

}

return -1;

}

**字符串逆序**

方法一

#include <stdio.h>

#include <string.h>

void main()

{

    char str[]=”hello,world”;

    int len=strlen(str);

    char t;

    int i;

    for(i=0; i<len/2; i++)

    {

        t=str[i];

        str[i]=str[len-i-1];

str[len-i-1]=t;

    }

    printf(“%s\n”,str);

    return 0;

}

方法二

#include <stdio.h>

int main(){

char\* src = “hello,world”;

int len = strlen(src);

char\* dest = (char\*)malloc(len+1);//要为\0分配一个空间

char\* d = dest;

char\* s = &src[len-1];//指向最后一个字符

while( len– != 0 )

\*d++=\*s–;

\*d = 0;//尾部要加\0

printf(“%s\n”,dest);

free(dest);// 使用完，应当释放空间，以免造成内存汇泄露

return 0;

}

**排序**

**冒泡排序**

void bubble\_sort(int a[],int n)

{

int i,j;

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

{

bool x=ture;

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

{

int temp;

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

x=false;

}

}

if(x) break;

}

}

时间复杂度O(N^2)

**选择排序**

void select\_sort(int a[],int n)

{

int i,j;

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

{

int min=i;

for(j=i+1;j<n;j++)

{

if(a[j]<a[min])

min=j;

if(min!=i)

{

int temp=a[j];

a[j]=a[min];

a[min]=temp;

}

}

}

}

时间复杂度O(N^2)

**插入排序**

void insert\_sort(int a[],int n)

{

int i,j;

for(i=1;i<n;i++)

{

int x=a[i];

for(j=i;j>0&&x<a[j-1];j–)

a[j]=a[j-1];

a[j]=x;

}

}

时间复杂度O(N^2)

**快速排序**

void quick\_sort(int a[],int ileft,int iright)

{

int iPivot=(left+right)/2;

int nPivot=a[iPivot];

for(int i=ileft,j=iright;i<j;)

{

while(!(i>=iPivot||nPivot<a[i]))

i++;

if(i<iPivot)

{

a[iPivot]=a[i];

iPivot=i;

}

while(!(j<=iPivot||nPivot>a[j]))

j–;

if(j>iPivot)

{

a[iPivot]=a[j];

iPivot=j;

}

}

a[iPivot]=nPivot;

if(iPivot-ileft>1)

quick\_sort(a,ileft,iPivot-1);

if(iright-iPivot>1)

quick\_sort(a,iPivot+1,iright);

}

时间复杂度O(NlogN)

**链表**

**单链表**

**双链表**

**循环链表**

**单链表逆置**

void reverse(link \*head)

{

    link \*p, \*s, \*t;

    p = head;

    s = p->next;

    while(s->next!=NULL)

    {

        t = s->next;

        s->next = p;

        p = s;

        s = t;

    }

    s->next = p;

    head->next->next = NULL;    //尾指针置为空

    head->next = s;         //赋值到头指针后一位

}

**链表合并**

Node \* Merge(Node \*head1 , Node \*head2)

{

if ( head1 == NULL)

return head2 ;

if ( head2 == NULL)

return head1 ;

Node \*head = NULL ;

Node \*p1 = NULL;

Node \*p2 = NULL;

if ( head1->data < head2->data )

{

head = head1 ;

p1 = head1->next;

p2 = head2 ;

}else

{

head = head2 ;

p2 = head2->next ;

p1 = head1 ;

}

Node \*pcurrent = head ;

while ( p1 != NULL && p2 != NULL)

{

if ( p1->data <= p2->data )

{

pcurrent->next = p1 ;

pcurrent = p1 ;

p1 = p1->next ;

}else

{

pcurrent->next = p2 ;

pcurrent = p2 ;

p2 = p2->next ;

}

}

if ( p1 != NULL )

pcurrent->next = p1 ;

if ( p2 != NULL )

pcurrent->next = p2 ;

return head ;

}

递归方式：

Node \* MergeRecursive(Node \*head1 , Node \*head2)

{

if ( head1 == NULL )

return head2 ;

if ( head2 == NULL)

return head1 ;

Node \*head = NULL ;

if ( head1->data < head2->data )

{

head = head1 ;

head->next = MergeRecursive(head1->next,head2);

}

else

{

head = head2 ;

head->next = MergeRecursive(head1,head2->next);

}

return head ;

}

**写一个Singleton模式**

#include<iostream>

using namespace std;

class Singleton

{

    private:

    static Singleton\* \_instance;

    protected:

    Singleton()

{

    cout<<”Singleton”<<endl;

}

    public:

    static Singleton\* Instance()

{

if(NULL==\_instance)

{

    \_instance=new Singleton();

}

return \_instance;

}

};

static Singleton\* Singleton::\_instance=NULL;

int main()

{

    Singleton \* s =Singleton::Instance();

    Singleton \* s1=Singleton::Instance();

}

**如何对String类型数据的某个字符进行访问？**

#include<iostream>

using namespace std;

int main()

{

    string s=”abcdefg”;

    const char \*c=s.c\_str();

    while(\*c!=’\0′)

    {

        printf(“%c”,\*c++);

    }

}

**文件加密、解密**

**1.加密（encryption）：**

#include<stdio.h>

void encryption(char \*ch)

{

    (\*ch)^=0xFF; //算法可自行修改调整，使用AES加密算法

}

int main(int argc,char \*argv[])

{

    if(argc<2)

    {

        printf(“参数不足”);

        return -1;

    }

    //文件的打开（fopen函数）

    /\*

     r    read    只读

     w    write    只写

     a    append    追加

     t    text    文本文件，可省略不写

     b    banary    二进制文件

     +    读和写

     \*/

a.out a.c b.txt

argv[0] argv[1] argv[2]

    FILE\* fpr=NULL;

    FILE\* fpw=NULL;

    //文件打开失败返回一个空指针值NULL

    if(NULL==(fpr=fopen(argv[1],”r”))){printf(“%m\n”);return -1;}

    if(NULL==(fpw=fopen(argv[2],”w+”))){printf(“%m\n”);return -1;}

    char ch;

    while((ch=fgetc(fpr))!=EOF)

    {

        //putchar(ch);

        encryption(&ch);//加密函数

        printf(“%c”,ch);//加密后字符显示

        fputc(ch,fpw);//存放进文件

    }

    printf(“\n文件加密成功！\n”);

    //文件的关闭(fclose函数)

    fclose(fpr);

    fclose(fpw);

}

**2.解密（decryption）：**

#include<stdio.h>

#include<time.h>

void show()

{

    time\_t start=time(NULL);

    while(start==time(NULL));

}

void decryption(char ch)

{

    (\*ch)^=0xFF;//算法可自行修改调整

}

int main(int argc,char \*argv[])

{

    if(argc<2)

    {

        printf(“参数不足”);

        return -1;

    }

    //文件的打开（fopen函数）

    /\*

     r    read    只读

     w    write    只写

     a    append    追加

     t    text    文本文件，可省略不写

     b    banary    二进制文件

     +    读和写

     \*/

    FILE\* fpr=NULL;

    FILE\* fpw=NULL;

    //文件打开失败返回一个空指针值NULL

    if(NULL==(fpr=fopen(argv[1],”r”))){printf(“%m\n”);return -1;}

    if(NULL==(fpw=fopen(argv[2],”w+”))){printf(“%m\n”);return -1;}

    char ch;

    printf(“开始解密！\n”);

    while((ch=fgetc(fpr))!=EOF)

    {

        show();

        ch=decryption(ch);//解密函数

        printf(“%c”,ch);//解密后字符显示

        fputc(ch,fpw);//存放进文件

        fflush(stdout);//刷新显示

    }

    printf(“\n文件解密成功！\n”);

    //文件的关闭(fclose函数)

    fclose(fpr);

    fclose(fpw);

}

**斐波那契数列（Fibonacci sequence）**

int Funct( int n )

{

if( n==0 || n==1 ) return 1;

retrurn Funct(n-1) + Funct(n-2);

}