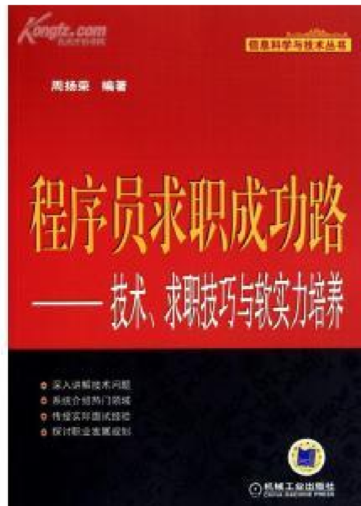
M

麦洛克菲IT求职培训

笔试面试技术技巧

受用终身的求职培训

麦洛克菲"内核，底层，安全，求职算法... www.mallocfree.com



主讲：周扬荣（麦洛克菲信息技术工作室）
主页：www.mallocfree.com
邮件：10950150@qq.com

程序员求职成功路—提纲

一，技术篇

- C
- C++
- 内存管理
- 多线程/多进程
- 数据结构与算法
- 网络
- 数据库
- Windows系统
- Linux系统
- 设计模式

二，面试篇

- 简历
- 面试
- 薪水谈判与Offer选择
- 试用期
- 跳槽

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

技术总结与提高篇

请高度集中注意力，知识点太密集

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

sizeof

sizeof(char) =?
sizeof(short)=?
sizeof(int)=?
sizeof(long)=?
sizeof(float)=?
sizeof(double)=?

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof()题目1

```
typedef struct _a
{
    char   c1;
    long   i;
    char   c2;
    double f;
}a;
```

```
typedef struct _b
{
    char   c1;
    char   c2;
    long   i;
    double f;
}b;
```

sizeof(a)=? sizeof(b)=？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof()题目2

```
#pragma pack(8)
struct s1
{
	short a;
	long b;
};
struct s2
{
	char c;
	s1 d;
	long long e;
};
struct s3
{
	char c;
	short a;
	long b;
	long long e;
};
#pragma pack()
```

1.sizeof(s1)=? sizeof(s2) = ? sizeof(s3) = ?
2.s2的c后面空了几个字节接着是d?

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof()题目3

```
typedef union
{
    long i;
    int k[5];
    char c;
} DATE;
struct data
{
    int cat;
    DATE cow;
    double dog;
} too;
DATE max;
sizeof(struct data) = ?sizeof(max))=?
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof计算大小准则

- (1)bool(BOOL)/char/short/int/long/float/double/longlong，指针，数组的大小是多少？
- (2)对齐准则
- A.自然对齐
- a）数据成员对齐规则：
- 在默认情况下，各成员变量存放的起始地址相对于结构的起始地址的偏移量:sizeof(类型)或其倍数
- b）整体对齐规则：
- 结构的总大小也有个约束条件：最大sizeof(类型)的整数倍
- B.强制对齐
- #pragma pack(push) //保存对齐状态
- #pragma pack(n) //定义结构对齐到n
-         定义结构
- #pagman pack(pop)//恢复对齐状态
- 上面的预编译语句将定义的结构体强制对齐到n。#pragma pack(n)来设定变量以n字节对齐方式。强制对齐应该遵守如下两条对齐规则：
- a）数据成员对齐规则：
- n字节对齐就是说变量存放的起始地址的偏移量：min(sizeof(类型)，n)或其倍数。
- b）整体对齐规则：
- 结构的总大小也有个约束条件：min(最大的sizeof(类型)，n)的倍数。

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

位域的大小计算

- struct 位域结构名
- {
- 位域列表
- };
- 例如：
- typedef struct _Demo
- {
- int a:4;
- int b:4;
- int c:2;
- }Demo;//sizeof(Demo)=4
- a）一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。
- b）由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度。
- c）位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。
- d）如果相邻位域字段的类型相同，且其位宽之和小于类型的sizeof(类型)大小，则后面的字段将紧邻前一个字段存储，直到不能容纳为止；
- e）如果相邻位域字段的类型相同，但其位宽之和大于类型的sizeof大小，则后面的字段将从新的存储单元开始，其偏移量为其类型大小的整数倍；
- f）如果相邻的位域字段的类型不同，则各编译器的具体实现有差异，VC6采取不压缩方式，即不同位域字段存放在不同的位域类型字节中；而GCC和DEV-C++都采取压缩方式

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof() 题目4

```
class A
{
public:
    A()
    {

    }
    void fn1();
    void fn2();
protected:
    char a;
    int b;
    static int c;
};
```

```
class A
{
public:
        A(){}
        void fn1();
        void fn2();
protected:
        int a;
private:
        int b;
};
class B:public A
{
public:
        B(){}
        void fn3();
        void fn4();
private:
        int c;
};
```

```
class A
{
public:
    A(){}
    virtual void fn1(){}
    void fn2();
protected:
    int a;
    int b;
};
class B:public A
{
public:
    B(){}
    virtual void fn1(){}
    void fn3();
    void fn4();
private:
    int c;
};
```

"麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

空类空结构的大小为？

```
class Null
{
}
struct _Null
{
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof结算5

- D．如何计算一个数组的元素个数？
  sizeof(A)/sizeof(A[0])
- E．sizeof()是动态执行还是静态编译确定？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Sizeof()计算—数组和指针

```
char *p1 = “Hello, word!”
char p2[] = “Hello, world”
char p3[] = {‘h’, ‘e’, ‘l’,’l’,’o’,’,’, ‘‘, ‘w’,’o’,’r’,’l’,’d’}
Sizeof(p1)=?sizeof(p2)=?sizeof(p3)=?

void func(int a[], int n)
{
   printf(“%d”, sizeof (a));
}
```

“麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

变量的存储位置，作用域，与生命周期

分析下列程序中每个变量的存储位置，作用域，与生命周期

```
int a = 0;
char *p1;
static float f;
void main(void)
{
    int b = 0;
    char s[] = "123";
    char *p2;
    char *p3 = "hello, world";
    static int c =0;
    p1 = (char *)malloc(128);
    p2 = (char *)malloc(256);
    free(p1);
    free(p2);
}
```

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

参数的传递

```
void fun(char c[])
{    printf("%d\n" , sizeof(c)); }
void fun2(char &c)
{    printf("%d\n" , sizeof(c)); }
void fun3(char(&c)[9])
{    printf("%d\n" , sizeof(c)); }
int main()
{
    char c[] = "12345678";
    printf("%d\n" , sizeof(c));
    fun(c);
    fun2(*c);
    fun3(c);
    return 0;
}
```

In C++, the string literal “C++”, which is the type of char,occupies exactly ………. of memory.

A. 3 bytes
B. 4 bytes
C. 5 bytes
D. 6 bytes

“麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

整数的存储

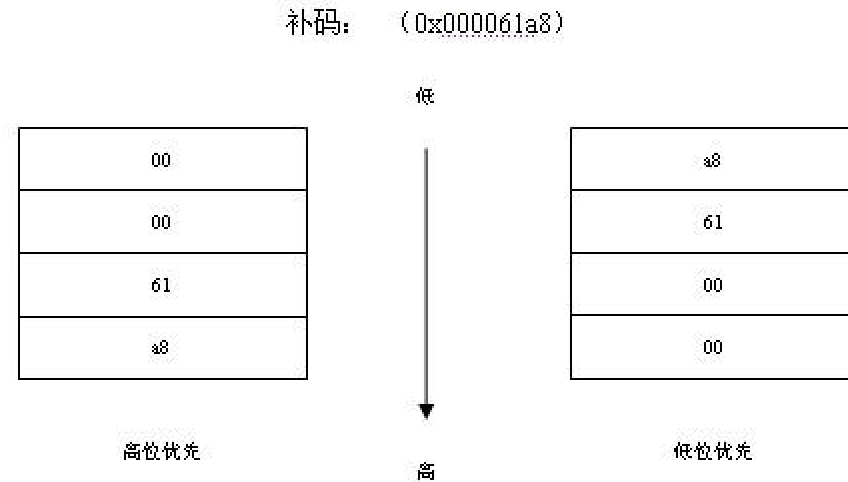
低位优先，高位优先，补码

- 补码
  - 常见数的补码（-1，128，0，127）
  - 补码计算方法（取反+1）
- 语句int i = -1;执行后，i在内存中的表现形式是什么？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

低位优先与高位优先

A.如何判断一个系统低位优先或者高位优先？
B.在网络传输过程中，为什么需要调用htonl()或ntohl()函数？网络字节序是哪种优先？
C.改变一个整数的存储格式




```
bool is_integer_lower_store()
{
    int      x = 0x1;
    char     *p = (char*)&x;
    if (*p == 1)
             return true;
    else
             return false;
}
```

```
int chgendian(int x)
{
      int      tmp = 0;
      char   *p = (char *)&x;
      int      shift =sizeof(int)*8 - 8;
      for(int i = 0; i < sizeof(int); i++)
      {
             tmp |= *p<<shift;
             shift -= 8;
             p++;
      }
      return tmp;
}
```

"麦洛克菲"内核，底层，www.malloc

位运算（1）

- 与或非取反左移右移基础运算
- 基本操作
  - #define BITN (1<<n)
  - A. 置位：a |= BITN;
  - B. 清零：a &= ~BITN
  - C. 判断：a & BITN
  - D. 取末8位：a &0xFF
  - E. a>>3 a&7
  - F.任何数与零异或不变；任何数与1异或为取反

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

位运算（2）

a与b互相交换

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

(4) 如何判断判断整数x的二进制中含有多少个1？

(5) 如何判断一个数是2的N次方？

(6)如何将字符串的IP地址转化为一个整数？

(7)计算如下表达式：

(char)(127<<1)+1

(char)(-1>>1)+1

1<<2+3

(8) Design a double linked list with only one pointer per node

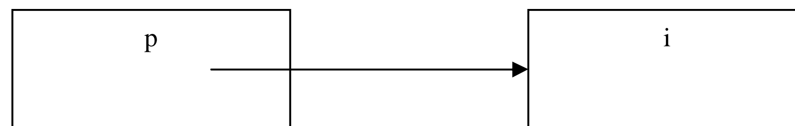
```
int func(x )
{
    int countx = 0;
    while ( x )
    {
        countx ++;
        x = x&(x-1);
    }
    return countx;
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

指针（0）

- 指针：指针其实就是一个变量，和其他类型的变量一样。在32位机器上，它是一个占用四字节的变量，它与其他变量的不同就在于它的值是一个内存地址，指向内存的另外一个地方。

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com



指针p指向i，表示它的值为i的地址

指针（0）

```
//下面是用typedef定义一个新结构最常用的定义形式
//在微软的面试中，在考查你某个算法前，一般会让你先定义一个与
        算法相关的结构。
//比如链表排序的时候，让你定义一个链表的结构。
typedef struct _node
{
        int          value;
        struct _node * next;
}node, *link;
node *pnode = NULL; //声明变量都应该初始化，尤其是指针
pnode = (node *)malloc(sizeof (node)); //内存分配
//务必检测内存分配失败情况，程序健壮性的考查
//加上这样的判断语句，会让你留给面试官一个良好的印象
//不加这样的判断，如果分配失败，会造成程序访问NULL指针崩溃
if (pnode == NULL)
{
        //出错处理，返回资源不足错误信息
}
memset(pnode, 0, sizeof(node));          //新分配的内存应该初
        始化，否则内存中含有无用垃圾信息
pnode->value = 100;
printf("pnode->value = %d\n", pnode->value);
node * ptmp = pnode;
ptmp += 1;      //指针支持加减运算，但须格外小心
free(pnode);    //使用完内存后，务必释放掉，否则会泄漏。
                //一般采取谁分配谁释放原则
pnode = NULL;//释放内存后，需要将指针置NULL，防止野指针
```

Which of the following statements creates and initializes a pointer named salesPtr?

A. salesPtr = NULL;
B. float &salesPtr = NULL;
C. float *salesPtr = "";
D. float *salesPtr = NULL;

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

指针加减法

- node *p = …;
- p += n中，p向前移动的位置不是n个字节，而是n * sizeof(*p)个字节，指针的减法运算与此类似。
- (char *)p + 1

```
#include <stdio.h>
int main(void)
{
    int a[5][10];
    printf("%d,%d,%d\n", a, a+1, &a+1);
    return 0;
}
```

其输出结果为：1310392,1310432,1310592。试分析原因。

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

指针（1）

- 含义：
- 1)　int *a[10];
- 2)　int (*a)[10];
- 3)　int (*a)(int);
- 4)　int (*a[10])(int);
- 5)　int *a, **a;
- 6)　char str[];
- 7)　char *str, **str;

- sizeof（）计算指针长度。

- char *p1 = “Hello, word!”
- char p2[] = “Hello, world”
- char p3[] = {‘h’, ‘e’, ‘l’,’l’,’o’,’,’, ‘ ‘, ‘w’,’o’,’r’,’l’,’d’}

sizeof(p1)=?sizeof(p2)=?sizeof(p3)=?

“麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

指针（2）

一，计算数组长度

```
int a[10];//sizeof (a) = 10*sizeof (int) = 40;

int a[10];
void func(int a[], int n)
{
    printf("%d", sizeof (a));//此时数组退化为指针，所以 sizeof (a) = 4
}
```

二，分析下面的代码，试指出数组指针的不同含义。

```
#include <stdio.h>
int main(void)
{
    int a[5][10];

    printf("%d,%d,%d\n", a, a+1, &a+1);
    return 0;
}
```

三，结构成员求偏移

```
#define OFFSET(TYPE, MEMBER)  (size_t) (&(((TYPE*)0)->MEMBER))
```

麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

指针（3）

- 下面的C代码在VC++6.0和低位优先的平台下的运行结果是什么？请详细说明原因。

```
#include   <stdio.h>
int main(void)
{
    int         a[5] = {1,2,3,4,5};
    int         *ptr1 = (int *)(&a+1);
    int         *ptr2 = (int *)((int )a+1);
    printf("%x,%x",ptr1[-1],*ptr2);
    return 0;
}
```

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

指针与引用的区别

- 引用必须初始化
- 引用和指针都是地址
- 引用比指针更安全，也比指针方便

```
int func(int a);      //传值
int func(int *a);   //传指针
int func(int &a);  //传引用
int x = 10;
func(x);
func(&x);
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

指针的注意事项

- 指针在声明的时候最好初始化。
- 指针的加减运算移动的是指针所指类型大小。
- 当用malloc或new为指针分配内存时应该判断内存分配是否成功，并对新分配的内存进行初始化。
- 如果指针指向的是一块动态分配的内存，那么指针在使用完后需要释放内存，做到谁分配谁释放的原则，防止内存泄漏。无法做到这一点，请用引用计数或者智能指针。
- 指针在指向的动态内存释放后应该重新置为NULL，防止野指针

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

指针的引用记数

```
class CXData
{
public:
    CXData()
    {
        m_dwRefNum = 1; //引用计数赋初值
    }
    ULONG AddRef()    //增加引用
    {
        ULONG num =
       InterlockedIncrement(&m_dwRefNum);
        return num;
    }
    ULONG Release() //减少引用
    {
        ULONG num =
       InterlockedDecrement(&m_dwRefNum);
        if(num == 0)    //当计数为0了，就释放内存
            delete this;
        return num;
    }
private:
   ULONG m_dwRefNum;    //引用计数
}
```

```
//使用实例：
CXData *pXdata = new CXData;
//使用前增加计数
pXdata->AddRef();
//使用后减少计数，如果计数为零，则释放内存
pXdata->Release();
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

C/C++语言关键字

- typedef

```
typedef 返回值 (stdcall *FuncName)(param1, param2…);
typedef struct _StructName
{
} StructName, *PStructName;
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

#define

- #define f(a,b) a/b
- f(5+3, 5-3)

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

static

- A.修饰变量
- 问题：下面的函数实现在一个数上加一个数，有什么错误？ 如何改正？

```
int add_n ( int n )
{
    static int i = 100;//局部变量或全局变量
    i += n;
    return i;
}
```

- B.修饰函数
- 现在来看一道Intel的面试题：
- 问题：A.c 和B.c两个c文件中使用了两个相同名字的static变量，编译的时候会不会有问题？这两个static变量会保存到哪里（栈还是堆或者其他的）？
- C.C++中
- 在C++里，static修饰函数和变量，表示该函数或变量属于该C++类的静态成员，为所有对象共同所有。在类中，静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。
- 静态成员函数不接受隐含的this自变量。所以，它就无法访问自己类的非静态成员。

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

extern “C”

- 声明

```
int g_iTotal = 0;
extern int g_iTotal;
```

- C++使用c里面的函数(重载与重整）

```
extern “C”
{
    int func(void);
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

volatile

```
a.#include <stdio.h>
void main()
{
    int i=10;
    int a = i;
    printf("i= %d",a);
    //下面汇编语句的作用就是改变内存中i的值
    //但是又不让编译器知道
    __asm {
    mov dword ptr [ebp-4], 20h
    }
    int b = i;
    printf("i= %d",b);
}
```

在调试版本模式运行程序，输出结果如下：

i = 10

i = 32

在release版本模式运行程序，输出结果如下：

i = 10

i = 10

```
b.#include <stdio.h>
void main()
{
    volatile int i=10;
    int a = i;
    printf("i= %d",a);
    __asm {
    mov dword ptr [ebp-4],
    20h
    }
    int b = i;
    printf("i= %d",b);
}
```

分别在调试版本和release版本运行程序，输出都是：

i = 10

i = 32

这说明这个关键字发挥了它的作用。

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

Volatile(2)

优化器在用到这个变量时必须每次都小心地重新读取这个变量的值。
C.如何让下面的循环不被优化掉？

```
for ( int i=0; i<100000; i++);
```

这个语句用来测试空循环的速度的
但是编译器肯定要把它优化掉，根本就不执行
如果写成

```
for ( volatile int i=0; i<100000; i++);
```

它就会执行了
D.一个参数既可以是const还可以是volatile吗？可以。只读寄存器。
E.一个指针可以是volatile吗？可以。
下面的函数有什么错误：

```
int square(volatile int *ptr)
{
    return *ptr * *ptr;
}
```

→

```
int square(volatile int *ptr)
{
    int a = *ptr;
    int b  = *ptr;
    return a * b;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

Char /SWITCH

- **char**
- A.sizeof(char) = 1
- B.下面的代码是否具有可移植性

```
char tmp;
if(tmp<0){...}
else{...}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

include

- **#include**
- #include “x.h”
- #inlcude <x.h>
- #include <x>
- <x.h>：C标准库或者老的C++标准库
- <x>：新的C++标准库，放入了 std名字空间。需要结合using namespace std;来使用
- <cx>：C++标准库统一C标准库，也纳入了std名字空间中
- “x.h”：用户的头文件，从当前目录开始搜索

“麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

const

- **const**
- A．const修饰变量（常变量）
- B．const修饰函数
- const int func(const char*str) const

试指出下面的指针的含义与区别：

1）const int *a;
2）int const *a;
3）int * const a;

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

操作符

- (1)优先级与结合律
- (2)计算下面的结果：1<<2+3=?
- (3)“= =”与“=”号的区别
- 一个合格的程序员必定会反复犯的一个错误。
- (4)”+=/*=/<<=”
- (5)a>b?a:b
- (6)i++与++i求值

<table>
<tr><th>运算符</th><th>结合性</th></tr>
<tr><td>() [] -> .</td><td>从左到右</td></tr>
<tr><td>! ~ ++ -- +(正号) -(负号) *(指针取值符) (type) sizeof</td><td>从右到左</td></tr>
<tr><td>* / %</td><td>从左到右</td></tr>
<tr><td>+ -</td><td>从左到右</td></tr>
<tr><td><< >></td><td>从左到右</td></tr>
<tr><td>< <= > >=</td><td>从左到右</td></tr>
<tr><td>== !=</td><td>从左到右</td></tr>
<tr><td>&</td><td>从左往右</td></tr>
<tr><td>^</td><td>从左往右</td></tr>
<tr><td>|</td><td>从左往右</td></tr>
<tr><td>&&</td><td>从左往右</td></tr>
<tr><td>||</td><td>从左往右</td></tr>
<tr><td>?:</td><td>从右往左</td></tr>
<tr><td>= += -= *= /= %= &= ^= |= <<= >>=</td><td>从右往左</td></tr>
<tr><td>,</td><td>从右往左</td></tr>
</table>

口诀	解释
扩建点（扩建新的地点）	扩（大、中、小括号）建（->符号）点（.符号）
单算易比较（单独计算容易比较）	单（单目运算符）算（算术运算符）易（移位运算符）比较（比较运算符）
胃饥三等点（胃很饥饿，等到3点多）	胃（位运算符）饥（逻辑运算符）三（三目运算符）等（赋值运算符，＋＝、－＝等缩写运算符）点（逗点运算符）

“麦洛克菲”内核,底层,安全,求 www.mallocfree

请写出如下代码的运行结果

```
int main()
{
        int a,b,c,d;
        a=0;
        b=1;
        c=2;
        d=3;
        printf(“%d”,a+++b+c+++d++);
         //(a++)+b+(c++)+(d++)
        return 0;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

溢出

- 数组

```
void print_array(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        a[i] = a[i+1];
        printf(“%d\n”, a[i]);
    }
}
```

上面的循环判断应该改为：

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

```
void ReverseString(char * str)
{
    int n;
    char c;
    n = strlen(str);
    for (int i = 0; i < n/2; i++)
    {
        c = str[i];
        str[i] = str[n-i];
        str[n-i] = c;
    }
}
```

```
int  a[10] ={0};
void print(int x[])
{
    for(int i = 0;i<10;i++)
    {
        print(“%d\n”, x[i]);
    }
}
void main(void)
{
        int a[9] ={0};
        print(a);
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

数组防溢出

```
typedef int (&A_10)[10];

void print_a(A_10 x)
{


    for(int i = 0; i < 10;i++)
    {
        printf("%d\n", x[i]);
    }
}


int _tmain(int argc, _TCHAR* argv[])
{
    int a[11]= {0};
    print_a(a);
    return 0;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

数溢出

```
void BuildToLowerTable( void )          /* ASCII版本*/
{
        unsigned char ch;
        /* 首先将每个字符置为它自己 */
        for （ch=0; ch <= UCHAR_MAX;ch++）
           chToLower[ch] = ch;
          /* 将大写字母改为小写字母 */
          for( ch = 'A'; ch <= 'Z'; ch++ )
                     chToLower[ch] = ch +'a' – 'A';
}
void * memchr( void *pv, unsigned char ch, size_t size )
{
        unsigned char *pch = (unsigned char *) pv;
        while( -- size >=0 )
        {
                    if( *pch == ch )
                           return (pch );
                    pch++;
        }
        return( NULL );
}
//下面的代码：
char c*= (-1);
```

麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

缓冲区溢出

```
void func1(char* s)
{
    char buf[10];
    strcpy(buf, s);
}
void func2(void)
{
   printf("Hacked by me.\n");
   exit(0);
}
int main(int argc, char* argv[])
{
    char badCode[] = "aaaabbbb2222cccc4444ffff";
    DWORD* pEIP = (DWORD*)&badCode[16];
    *pEIP = (DWORD)func2;
    func1(badCode);
    return 0;
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

栈溢出

```
#include <linux/module.h>
int init_module(void)
{
    char buf[10000];
    //所以发生溢出
    memset(buf,0,10000);
    printk("kernel stack.\n");
    return 0;
}
void cleanup_module(void)
{
    printk("goodbye.\n");
}
MODULE_LICENSE("GPL");
//应用栈的大小对少？（1M）
//内核栈的大小多少？（4K或8K）
//什么时候容易栈溢出？
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

指针溢出

```
void* memchr( void *pv, unsigned char ch, size_t size )
{
    unsigned char *pch = ( unsigned char * )pv;
    unsigned char *pchEnd = pch + size;
    while( pch < pchEnd )
    {
        if( *pch == ch )
            return ( pch );
        pch ++ ;
    }
    return( NULL );
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

C++

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

const，inline与#define区别

- A.数据上的区别
- #define PAI 3.1415926 //宏定义
- const float pai = 3.1415926 //常变量，拥有类型，可调试，可做类型安全性检查
- B．代码上
- a.#define
- 预处理的地方把代码展开，不需要额外的空间和时间方面的开销，所以调用一个宏比调用一个函数更有效率。但是宏容易产生二义性，也不能访问对象的私有成员，这是宏的局限，没有类型检查。
- b.inline
- 内联函数和宏的区别在于，宏是由预处理器对宏进行替代，而内联函数是通过编译器控制来实现的。而且内联函数是真正的函数，只是在需要用到的时候，内联函数像宏一样的展开，所以取消了函数的参数压栈，减少了调用的开销。你可以像调用函数一样来调用内联函数，而不必担心会产生于处理宏的一些问题。
- 加上了inline关键字的函数，编译器将用它的代码直接进行替换。对于那些**代码较少而调用频率较高**的函数适合用inline关键字，减小了函数在执行时候的堆栈维护开销。
- c. 用宏定义一个x2的计算。
- #indefine POWER(x) x*x
- #indefine POWER(x) (x)*(x)
- #indefine POWER(x) (x*x)
- #indefine POWER(x) ((x)*(x))

- .c和.cpp文件的区别？
- new和malloc区别
- struct 和 class区别

麦洛克菲"内核，底层，求职算法培训 www.mallocfree.com

C++面向对象的三大特性

- **C++**面向对象的三大特性及其设计应用
- (1)封装
- (2)继承
- (3)多态

- 构造函数为什么不要求返回类型？

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

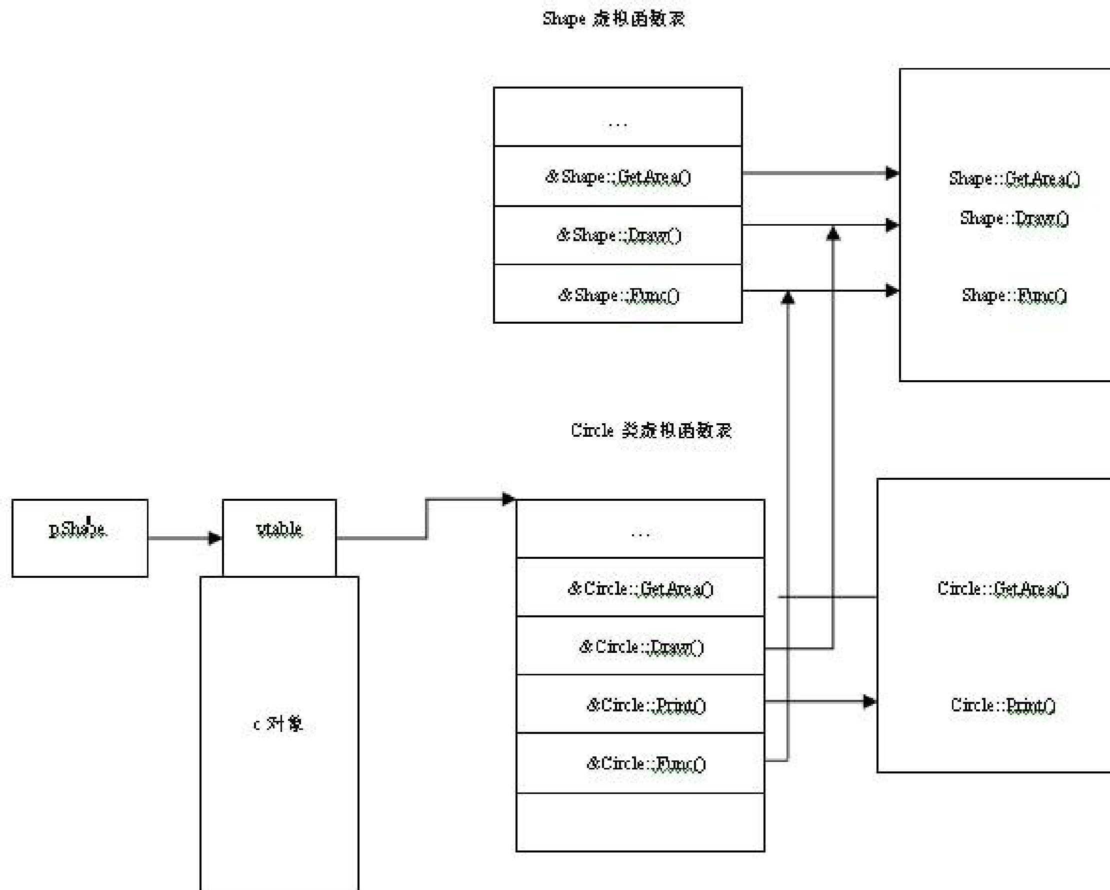
M

什么是多态？

- (1)多态概念
- 派生类对象对于同一个消息会进行不同的操作。多态性是通过虚拟函数实现的。通过基类指针（或引用）来请求使用虚拟函数时，C++会在与对象关联的派生类中选择正确的改写过的函数。
- 通过例子来举例

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

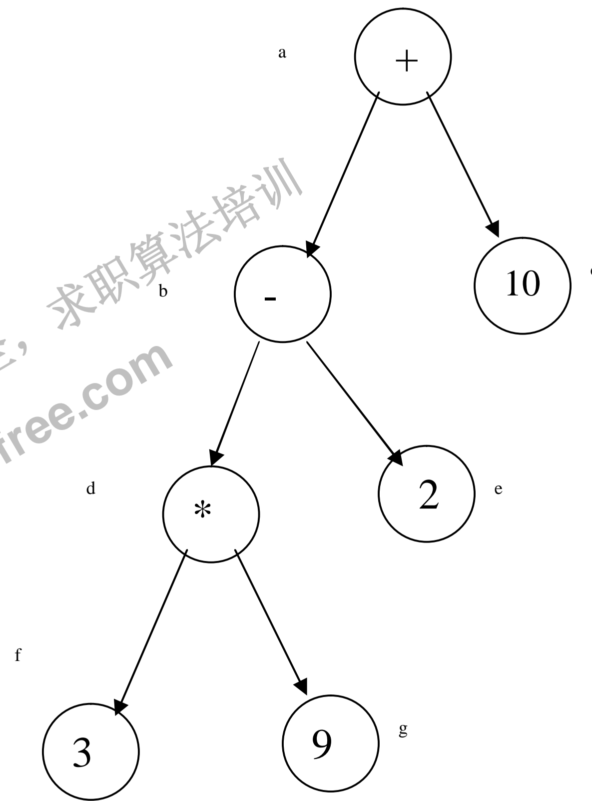
多态的实现



为什么构造函数不用使用virtual?

多态的应用

- 如右图所示，为一个算术表达式树结构。试设计一个算法，求每个结点的值。如果结点的类型为值结点，则返回其本身；如果为操作符结点，则返回其左子树的值与右子树值的计算结果。如结点f的返回值为3。结点d的值为27，结点b的值为25，结点a的值为35。



"麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

static_cast

- dynamic_cast: 通常在基类和派生类之间转换时使用；
- const_cast: 主要针对const和volatile的转换.
- static_cast: 一般的转换，如果你不知道该用哪个，就用这个。
- reinterpret_cast:用于进行没有任何关联之间的转换，如一个字符指针转换为一个整形数。reinterpret_cast 只能在指针之间转换，或者指针与整数之间

Static的转换：
A.用于类层次结构中基类和子类之间指针或引用的转换。进行上行转换（把子类的指针或引用转换成基类表示）是安全的；进行下行转换（把基类指针或引用转换成子类表示）时，由于**没有动态类型检查**，所以是不安全的。
B.用于基本数据类型之间的转换。如把int转换成char，把int转换成enum。这种转换的安全性也要开发人员来保证。
C.把空指针转换成目标类型的空指针。
D.把任何类型的表达式转换成void类型。
E.static_cast不能转换掉expression的const、volitale等属性。

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

static_cast(1)

```
class B
{
        public:
            int m_iNum;
            virtual void foo();
};
class D:public B
{
        public:
            char *m_szName[100];
};
void func(B *pb)
{
   D *pd1 = static_cast<D *>(pb);
   D *pd2 = dynamic_cast<D *>(pb);
}
```

```
class A
{
        public:
            int m_iNum;
            virtual void f(){}
};
class B:public A
{
};
class D:public A
{
};
void foo()
{
            B *pb = new B;
            pb->m_iNum = 100;
            D *pd1 = static_cast<D *>(pb); //compile error
            D *pd2 = dynamic_cast<D *>(pb); //pd2 is NULL
            delete pb;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

static_cast(2)

```
//const_cast
class B
{
        public:
            int m_iNum;
}
void foo()
{
        const B b1;
        b1.m_iNum = 100; //comile error
        B b2 = const_cast<B>(b1);
        b2. m_iNum = 200; //fine
}
```

麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

空类的实质内容是什么？

```
class Nothing
{
};
```

和下面类的声明是等价的：

```
class Nothing
{
public:
  Nothing();
  Nothing(const Nothing& rhs);
  ~Nothing();
  Nothing& operator=(const Nothing& rhs);
  Nothing* operator&();
  const Nothing* operator&() const;
};
```

Which items (item) are (is) not in the list of the default methods a class has?

A. Copy constructor
B. Copy-assignment operator
C. stream insertion << operator
D. All of above

麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Deep copy和shallow copy

The default copy constructor performs:
A. Deep copy
B. Shallow copy
C. Hard copy
D. Soft copy

- (1)浅复制（浅克隆）
- 被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。换言之，浅复制仅仅复制所考虑的对象，而不复制它所引用的对象。
- (2)深复制（深克隆）
- 被复制对象的所有变量都含有与原来的对象相同的值，除去那些引用其他对象的变量。那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。换言之，深复制把要复制的对象所引用的对象都复制了一遍。

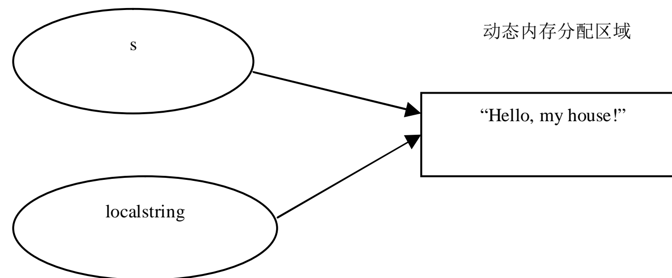
麦洛克菲"内核,底层，安全，求职算法培训 www.mallocfree.com

类中含有动态内存分配

```
class String
{
public:
    ……
private:
    char *m_data;
}
```

```
void DoNothing(String localstring)
{
}
String s = "Hello, my house!";
DoNothing(s);
//这个时候的s？
String b = “hello world!”;
String a = b;
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com



赋值运算符的设计

(1) 不设计一个专门的

```
string a("hello");
string b("world");
b = a;
string a("hello, world");
DoSomething(a);
```

a: data——> "hello\0"
b: data——> "world\0"

```
a:   data --------> "hello\0"
                 |
b:   data -------+  "world\0"
```

```
String & String::operator =(const String &other)
{
        //防止自我赋值
        if(this == &other)
                return *this;
        //释放原来的内存
        delete [] m_data;
        //分配新内存，拷贝数据
        int length = strlen(other.m_data);
        m_data = new char[length+1];
        strcpy(m_data, other.m_data);
        return *this;
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

构造函数的实现

```
String::String(const char *str)
{
    //字符串为空
    if(str==NULL)
    {
            m_data = new char[1];
            *m_data = '\0';
    }
    else
    {
            int length = strlen(str);
            m_data = new
                char[length+1];
            strcpy(m_data, str);
    }
}
```

```
//拷贝构造函数
String::String(const String &other)
{
            int length = strlen(other.m_data);
            m_data = new char[length+1];
            strcpy(m_data, other.m_data);
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

成员变量的初始化的区别？

- 顺序？静态成员的初始化？
- (1)内建型别（如int/float/char等）在初始化列表与函数体内初始化无区别；非内建型别的初始化应该在初始化列表中实现；const 常量必须在初始化列表中实现。
- (2) 成员初始化顺序应该与声明的顺序一致
- (3) C++类中的静态成员不能在类定义里边初始化，只能在类定义外初始化，并且初始化只能做一次。如果静态成员是私有成员，那么除此以后，将无法在类外使用这个私有静态数据成员，除非定义了一些函数来访问。

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

成员初始化列表顺序

```
template<class t>
    class array {
    public:
    array(int lowbound, int highbound);
    ...
private:
    vector<t> data;  // 数据存储在vector对象中
    size_t size;       // 数组中元素的数量
    int lbound, hbound;    // 下限，上限
    };
   template<class t>
    array<t>::array(int lowbound, int highbound)
    : size(highbound - lowbound + 1),
    lbound(lowbound),
    hbound(highbound),
    data(size)
    {}
```

```
class MyClass
{
public:
  MyClass();
private:
  int m_a;
  int m_b;
  int m_c;
};
MyClass::MyClass()
: m_c(0), m_b(0), m_a(0)
{
}
```

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

Static成员的初始化

```
class myclass
{
public:
    myclass();
private:
    static int x;
    static float y;
}
//仅在初始化的时候可以在类外访问这个两个私有成员
int myclass::x = 0;
float myclass::y = 1.1;
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

设计一个类，只生成该类一个实例

```
class Singleton
{
public:
    static Singleton* Instance();
protected:
    Singleton();
private:
    static Singleton* _instance;
};
Singleton* Singleton::Instance()
{
    if (_instance == 0)
    {
        _instance = new Singleton();
    }
    return _instance;
}
Singleton* sgn = Singleton::Instance();
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

基类析构函数的设计

- Class A{};
- Class B:public A {}

- A *pa = new B();
- delete pa;
- pa = NULL;
- 构造函数为什么不设置为virtual?
- 因为构造函数本来就是为了明确初始化对象成员才产生的，然而virtual function主要是为了再不完全了解细节的情况下也能正确处理对象。另外，virtual 函数是在不同类型的对象产生不同的动作，现在对象还没有产生，如何使用virtual函数来完成你想完成的动作。自己还没有构造好, 多态是被disable的

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

构造、析构、赋值函数调用顺序。

- 基类构造→本构造→赋值→本析构→基类析构
- 设计同步类CAutoLocker？

```
class A
{
public :
        A(int i = 0)
        { m_i = i;cout << m_i << "A::A()" << endl;}
        ~A(){cout << m_i << "A::~A()\n";}
        void f(){cout << m_i << "A::f()\n";}
private:
        int m_i;
};
class B:public A
{
public:
        A m_A;
        B(int i):m_A(i)
        {
                cout << "B::B()\n";
        }
        ~B(){cout << "B::~B()\n";}
        void f(){cout << "B::f()\n";}
};
void main( void )
{
    A *pa = new B(10);
    pa->f();
    delete pa;
    return;
}
```

"麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

CAutoLocker

```
class CLock
{
public:
      void Lock()
      {EnterCriticalSection(&m_sec);}
      void Unlock()
      {LeaveCriticalSection(&m_sec);}
      CLock ()
      {InitializeCriticalSection(&m_sec);}
      ~ CLock ()
      {DeleteCriticalSection(&m_sec);}

private:
   CRITICAL_SECTION m_sec;
};
```

```
class CAutoLock
{
public:
     CAutoLock(CLock  * lpLock) :
     m_pLock (lpLock)
     {
        m_pLock ->Lock();
     }
     ~CAutoLock()
     {
        m_pLock ->Unlock();
     }
private:
    CLock * m_pLock;
};
```

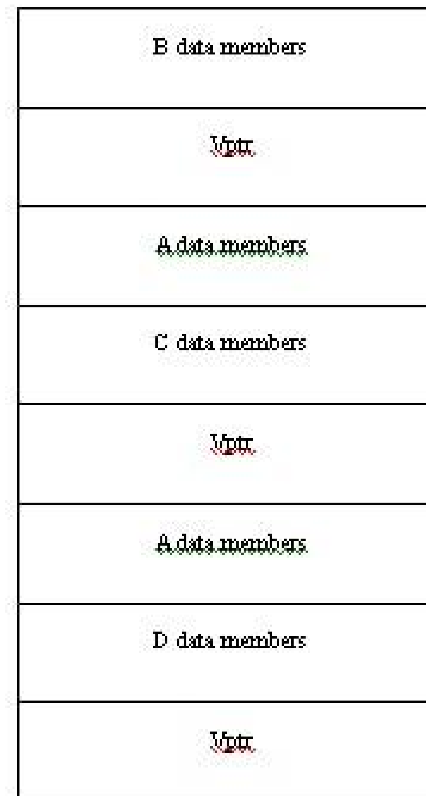
使用例子：

```
{
   CLock m_lock;
   CAutoLock(&m_lock);
   ….
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

继承与菱形继承

```
class A
{
public:
                virtual void f();
      …
private:
       int m_data;
}
class B: public A
{
public:
                virtual void f();
      …
}
class C: public A
{
public:
                virtual void f();
}
//类D多重继承自B，C
class D: public B, public C
{
public:
                void g() {f();}
}
```



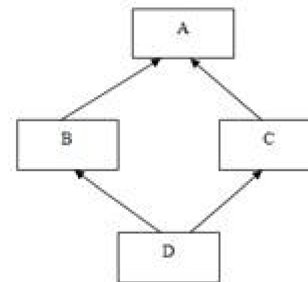
二义性

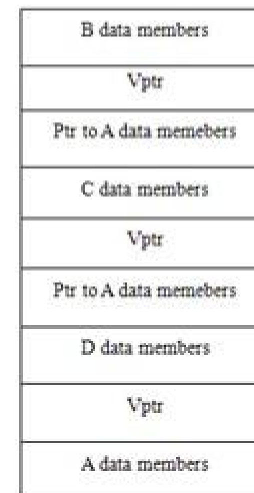
基类数据多分拷贝

“麦洛克菲”内核，底层，安全，求
www.mallocfree

继承与菱形继承

```
class A
{
public:
            virtual void f();
    …
}
class B: virtual public A
{
public:
            virtual void f();
    …
}
class C: virtual public A
{
public:
            virtual void f();
}
//类D多重继承自B，C
class D: public B, public C
{
public:
            void g();
}
```





“麦洛克菲”内核，底层，安
www.mallo

设计一个不能被继承的类

```
class FinalClass1
{
public:
    static FinalClass1* GetInstance()
    {
        return new FinalClass1;
    }
    static void DeleteInstance( FinalClass1* pInstance)
    {
        delete pInstance;
        pInstance = 0;
    }
private:
    FinalClass1() {}
    ~FinalClass1() {}
};
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

禁止对象产生于堆或非堆中

- (1)禁止堆

```
class Demo
{
public:
        ……
private:
        static void *operator new(size_t size);
        static void operator delete(void *ptr);
}
Demo *pDemo = new Demo(); //错误
Demo demo; //正确
```

- (2)禁止非堆

```
class Demo
{
public:
        ……
protected:
        ~Demo();
}
Demo *pDemo = new Demo(); //正确
Demo demo; //错误
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

虚函数/纯虚函数/虚基类/接口

- virtual void func() = 0;
- virtual void func();
- class IXyzName
- {
-     virtual void fun1() = 0;
-     virtual int fun2() = 0;
-     …;
- }
- 类对象实例化

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

重载(overload),重整/重写(override)

- (1)重整
- int func(int x, int y);
- int func(char x, int y);
- 经过编译后，函数变成为：
- @func_int4int4
- @func_char1int4
- 从此处可以看出，如果把返回值也做为一个可区别的特征话，即：
- int func(int x, int y);
- char func(int x, int y);
- 那么在做如下调用的时候：
- int a, b;
- func(a, b);
- extern “C”
- {
- }
- (2)重载与重写区别
- 方法重载和方法重写的相同点和不同点：
- 相同点：都要求方法名称相同
- 不同点：
- 方法重载发生在同一个类的内部的方法内,而方法重写发生在继承关系中，派生类重写了基类的方法
- 方法重载不要求方法的返回类型一致与否 而方法重写要求返回类型必须一致
- 方法重载要求方法的参数签名必须不一致，而方法重写要求方法的参数签名必须一致
- 方法重载对方法的访问权限和抛出异常没有限制，而方法重写对方法的访问权限和抛出异常有限制
- 决不要重新定义继承而来的非虚函数。

The feature that allows you to use the same function name for separate functions that have different argument lists is called ________

A. overriding
B. overloading
C. constructing
D. destructing

麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

M

模板

- **(1)模板与虚函数（继承）代码重用的选择**
- (1)类型T是否影响行为，如果影响，虚函数。如计算面积，三角形，正方形，圆。
- (2)T不影响，操作完全一样，用模板。比如计算和，求最大值等。
- **(2)函数模板**

```
template < typename  T >
T max ( T a , T b )
{
return  a > b ? a : b ;
 }
```

- 宏：没有类型检查，无法调试
- 重载：考虑的类型太多
- 模板：简洁

```
template <typename T>
T min(T a[ ],int n)
{
    T min=a[0];
    for( int i=1;i<n;i++)
        if(min>a[i])  min = a[i];
        return min;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

类模板

```
template <typename T>
class ClassName
{
public:
    void func(int index, const T &value);
protected:
private:
    T *data;
}
template <typename T>
void ClassName<T>::func(int index, const T &value)
{
}
ClassName<int> a;
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

STL标准模板库

- STL模板库提供了如下几种通用容器：
- 序列容器：vector, list, deque；
- 关联容器：set, map,hashmap；
- 适配器：stack, queue, priority_queue
- STL模板库还提供了两种迭代器（类似于指针）：
- iterator为读/写模式
- const_iterator为只读模式
- STL提供了一些常用的算法：
- sort()
- reverse()
- find()
- copy()

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

STL标准模板库

- 容器vector的底层数据结构为数组，为连续内存，支持[]操作符，支持尾部操作。
- 容器list的底层数据结构为链表，为非连续内存，不支持[]操作符，支持任意位置操作
- 容器deque的底层数据结构为队列，支持头部和尾部操作，支持[]操作符
- 容器se/multiset的底层数据结构为红黑树。set的内部元素依据其值自动排序，每个元素值只能出现一次，不允许重复，multiset跟set相同，只不过它允许重复元素。
- 容器map/multimap的底层数据结构为红黑树，map是键值对（key-value），每一个元素都有一个键，是排序的基础，每个键只能出现一次，不允许重复，multimap跟map相同，只不过允许重复键，可用来当作字典。
- stack适配器的默认的参数中容器是用deque实现的，可选择的容器有list，deque，vector，实现后进先出的值的排列（栈结构）
- 适配器queue可以采用deque，或者list作为底层数据结构，不能用vector（因为vector没有提供front()函数）。默认的底层数据结构为deque。

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

HASH_MAP

```
hash_map<int, string> mymap;
//hash_map<int, string, hash<int>, equal_to<int> >
//mymap;

mymap[9527]="唐伯虎点秋香";
mymap[1000000]="百万富翁的生活";
mymap[10000]="白领的工资底线";
...
if(mymap.find(10000) != mymap.end()){
...
}
```

补充阅读：http://yujiawei.iteye.com/blog/409774

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

hash_map和map的区别

- 构造函数。hash_map需要hash函数，等于函数；map只需要比较函数(小于函数).
- 存储结构。hash_map采用hash表存储，map一般采用红黑树(RB Tree)实现。因此其memory数据结构是不一样的。

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

hash_map与map应用

- 总体来说，hash_map 查找速度会比map快，而且查找速度基本和数据数据量大小，属于常数级别;而map的查找速度是log(n)级别
- 如果考虑效率，特别是在元素达到一定数量级时，考虑hash_map。但若对内存使用特别严格，希望程序尽可能少消耗内存，那么一定要小心，hash_map可能会让你陷入尴尬，特别是当你的hash_map对象特别多时，你就更无法控制了，而且 hash_map的构造速度较慢。

实现一个String类

```
//String类的声明
class String
{
public:
    String(const char *str = NULL);
    String(const String &other);
    ~ String(void);
    //注意赋值操作符返回的是对象的
    引用
    String & operate =(const String
    &other);
private:
    char  *m_data;
};
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

智能指针

```
template <class T>
class SmartPtr
{
public:
	SmartPtr (T * realPtr = 0): pointee(realPtr)
	{
	}
	SmartPtr(SmartPtr<T> &rhs);
	~SmartPtr()
	{
		delete pointee;
	}
	SmartPtr &operator = (SmartPtr<T> rhs);
	T * operator->() const;
	T& operator*() const;
	bool operator!() const;
private:
	T *pointee;
};
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

智能指针使用

A. 使用

auto_ptr<T> ptr (new T);

shared_ptr<T> ptr (new T);

B. 局限

a. auto_ptr不能共享所有权。

auto_ptr <type> pt1(new type);

auto_ptr<type> pt2 = pt1; //拷贝

auto_ptr<type> pt3;

pt3 = pt1; //赋值

b. auto_ptr不能指向数组。

c. auto_ptr不能作为容器的成员。

d. auto_ptr不能通过赋值操作来初始化

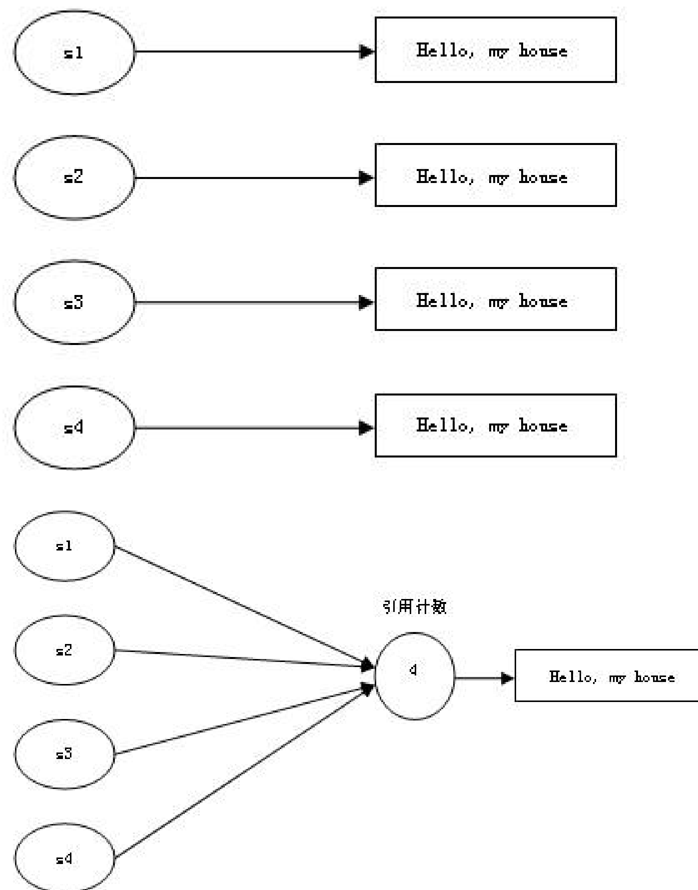
std::auto_ptr<int> p(new int(42)); //对

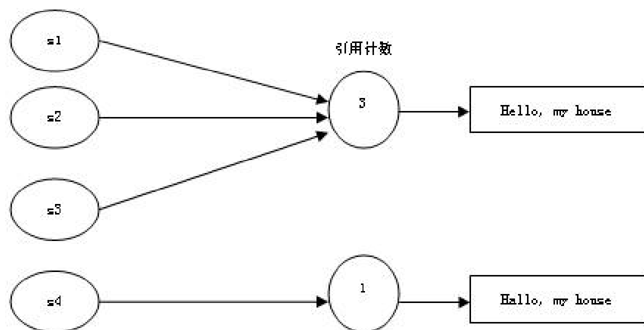
std::auto_ptr<int> p = new int(42); //错

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

写时拷贝（copy on write）

- 写时拷贝
- String s1, s2, s3, s4;
- s1 = s2 = s3 = s4 = “Hello, my house”;





麦洛克菲内核，底层，安全 www.malloc

this指针类型

- If the class name is X, what is the type of its "this" pointer (in a nonstatic, non-const member function)?
  - A. const X* const
  - B. X* const
  - C. X*
  - D. X&

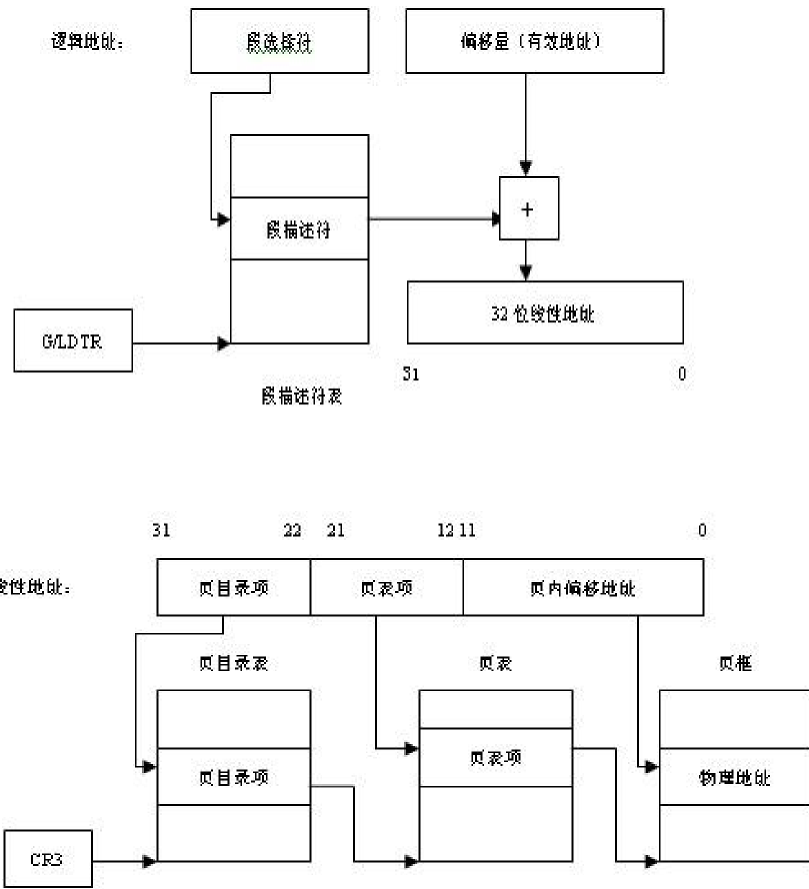
"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

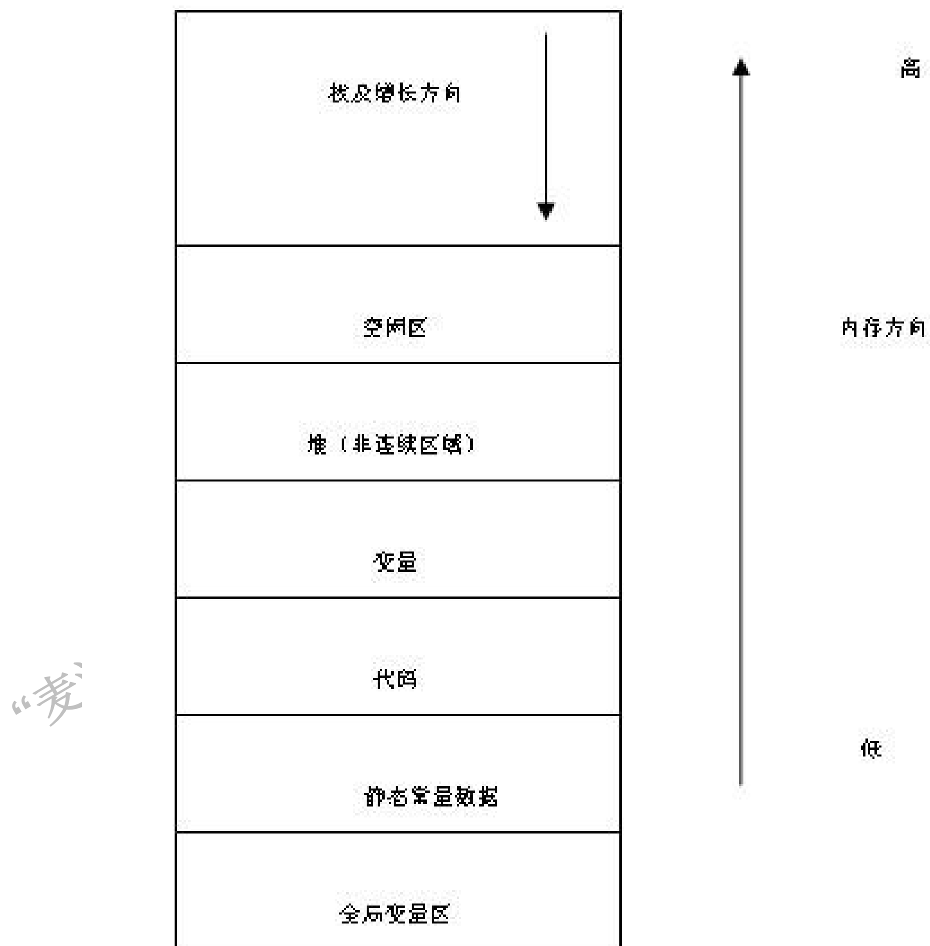
内存

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

虚拟内存空间
逻辑地址到物理地址转换



程序内存布局



调用约定

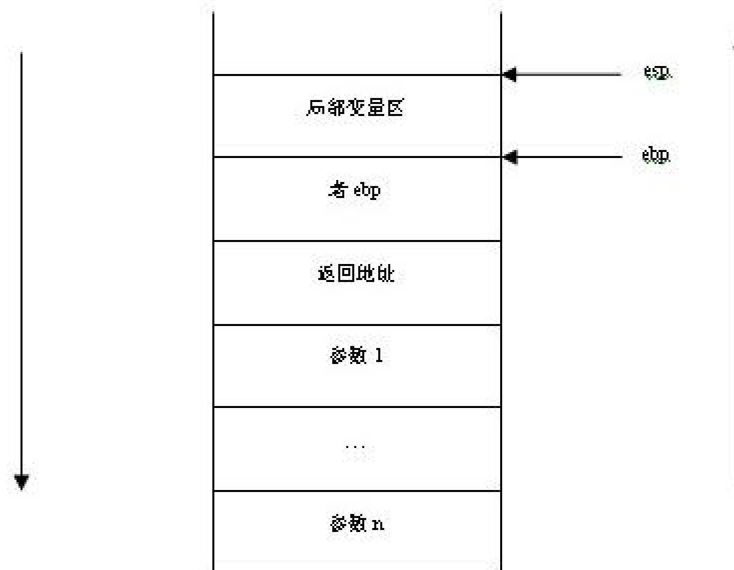
- Cdecl
- Stdcall
- Fastcall
- Thiscall
- Nakedcall

	Variable	Offset	
esp →	z	[esp]	local variables
	y	[esp+4]	
	buffer	[esp+8]	
	x	[esp+72]	
	saved eip	[esp+76]	
	a	[esp+80]	parameters
	b	[esp+84]	
	c	[esp+88]	

	Variable	Offset	
esp →	z	[ebp-76]	local variables
	y	[ebp-72]	
	buffer	[ebp-68]	
	x	[ebp-4]	
ebp →	saved ebp	[ebp]	saved register(s)
	saved eip	[ebp+4]	
	a	[ebp+8]	parameters
	b	[ebp+12]	
	c	[ebp+16]	

内存增长方向

栈增长方向



“麦洛克菲”

实例分析

```
1 #include <stdio.h>

2 void main(void)
3 {
4       char x,y,z;
5       int i;
6       int a[16];

7       for(i=0;i<=16;i++)
8       {
9               a[i]=0;
10              printf("\n");
11      }
12      return 0;
13 }
```

“麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

实例分析

一个C语言程序如下：

```
void func(void)
{
    char    s[4];

    strcpy(s, "12345678");
    printf("%s\n", s);
}
void main(void)
{
    func();
    printf("Return from func\n");
}
```

该程序在X86/Linux操作系统上运行的结果如下：

```
12345678
Return from func
Segmentation fault(core dumped)
```

试分析为什么会出现这样的运行错误。

D. 下面程序在SPARC/SUN工作站（整数存放方式是高位优先）上运行陷入死循环，试说明原因。如果将第7行的long *p改成short *p，并且将第22行long k改成short k后，loop中的循环体执行一次便停止了。试说明原因。

```
void main(void)
{
    addr();
    loop();
}
long    *p;
void loop(void)
{
    long    i, j;

    j = 0;
    for (i = 0; i < 10; i++)
    {
        (*p)--;
        j++;
    }
}
void addr(void)
{
    long  k;
    k = 0;
    p =& k;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

内存泄漏

- 泄漏分析
- void GetMemory(char *p)
- {
- p = (char *)malloc(100);
- }
- void Test(void)
- {
- char *str = NULL;
- GetMemory(str);
- strcpy(str, "hello world");
- printf(str);
- }
- void GetMemory(char **p, int num)
- {
- *p = (char *)malloc(num);
- }
- void Test(void)
- {
- char *str = NULL;
- GetMemory(&str, 100);
- strcpy(str, "hello");
- printf(str);
- }
- (2)那么如何防止内存泄漏呢？内存分配应该遵循下面的原则：
- A．谁分配，谁释放。在写下new/malloc时，要马上写下配对的delete/free以此释放掉。
- B．出错处理需释放。在函数错误处理分支中，记得释放掉已经分配的内存。
- C．网络上拷贝的代码，要仔细检查内存使用情况，预防内存泄露。
- D．引用计数
- E．智能指针
- F.已经发生了泄露如何检测？#define mynew {}

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

多线程/多进程

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

同步机制

- 同步机制(critical_section/mutex/event/semaphore)
- 内核态同步：
  - KSPIN_LOCK
  - KEVENT 阻塞一个线程直到其它线程检测到某事件发生
  - KSEMAPHORE 与事件对象相似，但可以满足任意数量的等待
  - KMUTEX 执行到关键代码段时，禁止其它线程执行该代码段
  - KTIMER 推迟线程执行一段时期
  - KTHREAD 阻塞一个线程直到另一个线程结束
  - ERESOURCE
  - FAST_MUTEX

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Critical_section

```
struct RTL_CRITICAL_SECTION
{
    PRTL_CRITICAL_SECTION_DEBUG DebugInfo;
   LONG LockCount;
   LONG RecursionCount;
   HANDLE OwningThread;
   HANDLE LockSemaphore;
   ULONG_PTR SpinCount;
};
```

DebugInfo 此字段包含一个指针，指向系统分配的伴随结构，该结构的类型为 RTL_CRITICAL_SECTION_DEBUG
LockCount 这是临界区中最重要的一个字段。它被初始化为数值 -1；此数值等于或大于 0 时，表示此临界区被占用。当其不等于 -1 时，OwningThread 字段包含了拥有此临界区的线程 ID。此字段与 (RecursionCount -1) 数值之间的差值表示有多少个其他线程在等待获得该临界区。
RecursionCount 此字段包含所有者线程已经获得该临界区的次数。如果该数值为零，下一个尝试获取该临界区的线程将会成功。
OwningThread 此字段包含当前占用此临界区的线程的线程标识符。此线程 ID 与 GetCurrentThreadId 之类的 API 所返回的 ID 相同。
LockSemaphore 它实际上是一个自复位事件，而不是一个信号。它是一个内核对象句柄，用于通知操作系统：该临界区现在空闲。操作系统在一个线程第一次尝试获得该临界区，但被另一个已经拥有该临界区的线程所阻止时，自动创建这样一个句柄。应当调用 DeleteCriticalSection（它将发出一个调用该事件的 CloseHandle 调用，并在必要时释放该调试结构），否则将会发生资源泄漏。
SpinCount 仅用于多处理器系统。在多处理器系统中，如果该临界区不可用，调用线程将在对与该临界区相关的信号执行等待操作之前，旋转 dwSpinCount 次。如果该临界区在旋转操作期间变为可用，该调用线程就避免了等待操作。旋转计数可以在多处理器计算机上提供更佳性能，其原因在于在一个循环中旋转通常要快于进入内核模式等待状态。此字段默认值为零，但可以用 InitializeCriticalSectionAndSpinCount API 将其设置为一个不同值。

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

参考阅读：http://www.cppblog.com/weidongmao/archive/2011/05/20/146839.aspx

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

```
int g_iTotal = 0;

void func()//7个线程调用
{
    g_iTotal++;
}
void func()//3个线程调用
{
    g_iTotal--;
}
```

"麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

进程通信

- 共享内存
- 管道（有名管道和无名管道）
- 信号量
- 事件
- 文件
- 消息

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

生产者消费者队列

生产者消费者队列

```
HANDLE ghSemaphore; //信号量
const int gMax = 100; //生产(消费)总数
std::queue<int> q; //生产入队,消费出队
//生产者线程
unsigned int __stdcall producerThread(void* pParam)
{
    int n = 0;
    while(++n <= gMax)
    {
        //生产
        q.push(n);
        cout<<"produce "<<n<<endl;
        ReleaseSemaphore(ghSemaphore, 1, NULL); //增加信号量
        Sleep(300);//生产间隔的时间,可以和消费间隔时间一起调节
    }
    _endthread(); //生产结束
    return 0;
}
```

```
//消费者线程
unsigned int __stdcall customerThread(void* pParam)
{
    int n = gMax;
    while(n--)
    {
        WaitForSingleObject(ghSemaphore, 10000);
        //消费
        q.pop();
        cout<<"custom "<<q.front()<<endl;
        Sleep(500);//消费间隔的时间
    }
    //消费结束
    CloseHandle(ghSemaphore);
    cout<<"working end."<<endl;
    _endthread();
    return 0;
}
```

```
//信号量来维护线程同步
ghSemaphore =
    CreateSemaphore(NULL, 0, gMax, NULL);
```

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

M

算法

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

算法设计一般思路

- 算法就是一个或多个函数（确定原型，输入检查（严进宽出），边界考虑，出错处理，性能优化）
- (1)输入输出
  - 所有变量必须初始化
  - typedef _PARAM
  - {
  -     CString szInfo;
  -     LPCTSTR *lpszAction;
  -     LPCTSTR *lpszParam;
  - }PARAM, *PPARAM;
  - PARAM param ={L"", 0};
  - 包括局部变量，全局变量，局部静态变量，C++内对象成员变量（包括STATIC）。内存在使用前，需要清零初始化。
- (2)断言的使用
- (3)各种输入，各种边界考虑
- 例子：memove()
- (4)出错处理
- 各种错误，内存分配失败等
- 函数返回结果考虑
- A.必须判断的：
  - 内存分配
  - 文件打开与读写
  - 等等
  - 使用之前，最好查阅MSDN
- B.不需判断的：
  - Free()
  - Delete()
  - Close()

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

算法的一般考虑

```
// 第一个
for (i=0; i<N; i++)
{
    if (condition)
        DoSomething();
    else
        DoOtherthing();
}
// 第二个
if (condition)
{
    for (i=0; i<N; i++)
        DoSomething();
}
else
{
    for (i=0; i<N; i++)
        DoOtherthing();
}
```

```
//阶乘实现
s = 1!+2!+3!+…+n!
//算法1：
int sum(n)
{
    int s = 0;
    for (i = 1; i < =n; i++)
    {
        s+= fac(i);
    }
    return s;
}
//算法2：
int sum(n)
{
    int s = 0;
    int t = 1;
    for (i = 1; i < =n; i++)
    {
        t *=  i;
        s += t;
    }
    return s;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com
麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

字符串

- strstr ()
- strtok()
- strcpy/strcat /memcpy/memmove
- Memcmp()
- Strcmp()
- Reversestr()
- Tolower ()
- Delchar/delchars
- IP地址转换
- 字符到整数/实数转换
- 字符串左旋

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

strstr

```
char * strstr (const char * str1, const char * str2)
{
        char *cp = (char *) str1;
        char *s1, *s2;
        if ( !*str2 )
            return((char *)str1);
        while (*cp)
        {
                s1 = cp;
                s2 = (char *) str2;
                while ( *s1 && *s2 && !(*s1-*s2) )
                        s1++, s2++;
                if (!*s2)
                        return(cp);
                cp++;
        }
        return(NULL);
}
```

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

strtok

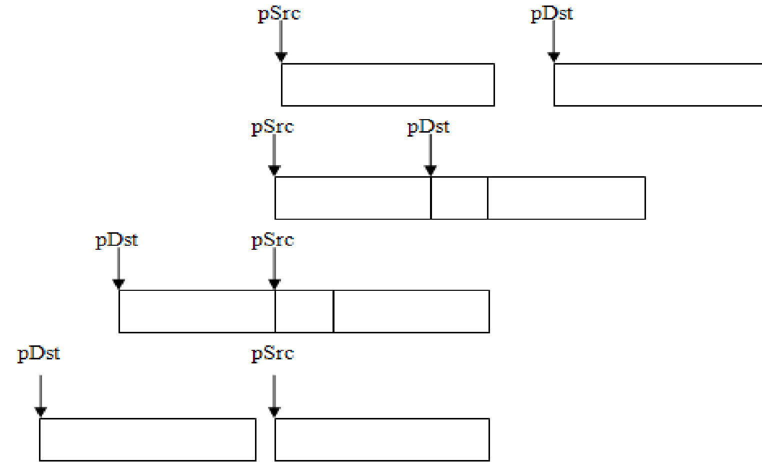
```
char *  strtok (char * string, const char *
      control)
{
        unsigned char *str;
        const unsigned char *ctrl = control;
        unsigned char map[32];
        int count;
        static char *nextoken;
        for (count = 0; count < 32; count++)
                map[count] = 0;
        do
        {
                map[*ctrl >> 3] |= (1 << (*ctrl & 7));
        } while (*ctrl++);
        if (string)
                str = string;
        else
                        str = nextoken;
        while ( (map[*str >> 3] & (1 << (*str & 7)))
       && *str )
                str++;
```

```
        string = str;
        for ( ; *str ; str++ )
        {
                if ( map[*str >> 3] & (1 << (*str & 7)) )
                {
                        *str++ = '\0';
                        break;
                }
         }
        nextoken = str;
        if ( string == str )
                return NULL;
        else
                return string;
}
```

麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

void * memmove (void * dst,const void * src,size_t count)

```
void * memmove (void * dst,const void * src,size_t count)
{
        void * ret = dst;
        if (dst <= src || (char *)dst >= ((char *)src + count))
        {
            while (count--)
                                {
                        *(char *)dst = *(char *)src;
                        dst = (char *)dst + 1;
                        src = (char *)src + 1;
            }
        }
        else
       {
                dst = (char *)dst + count - 1;
                src = (char *)src + count - 1;
                while (count--)
                {
                        *(char *)dst = *(char *)src;
                        dst = (char *)dst - 1;
                        src = (char *)src - 1;
                }
        }
        return(ret);
}
```



“麦洛克菲”内核，底层，安全，
www.mallocfree.com

Reversestr

```
void ReverseString(char * str)
{
    int n;
    char c;
    n = strlen(str);
    for (int i = 0; i < n/2; i++)
    {
        c = str[i];
        str[i] = str[n-i-1];
        str[n-i-1] = c;
    }
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

char tolower(char ch)

```
char tolower(char ch)
{
    if (ch >= ‘A’ && ch <= ‘Z’)
        return (ch + ‘a’ – ‘A’);
    else
        return ch;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

Delchar/delchars

```
char *DeleteChar(char *str, char c)
{
        assert(str != NULL);
        int iDes = 0, iSrc = 0;
        do
        {
                if (str[iSrc] != c)
                        str[iDes++] = str[iSrc];
        } while(str[iSrc++] != '\0')
}
char *DeleteChar(char *str, char chr[], int n)
{
        assert(str != NULL);
        char tmp[256] = {0};
        for (int i = 0; i < n; i++)
        {
                tmp[chr[i]] = 1;
        }
        int iDes = 0, iSrc = 0;
        do
        {
                if (!tmp[str[iSrc]])
                        str[iDes++] = str[iSrc];
        } while(str[iSrc++] != '\0')
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

IP地址转换

```
int ipstrtoint(const char *ip)
{
	int		result = 0;
	int		tmp = 0;
	int		shift = 24;
	const char *pEnd = ip;
	const char *pStart = ip;
	while(*pEnd != '\0')
	{
		//找到地址符里的’.’
		while(*pEnd != '.' && *pEnd != '\0')
			pEnd++;
		tmp = 0;
		//计算每个’.’之间的数值
		while(pStart < pEnd)
		{
			tmp = tmp * 10 + (*pStart - '0');
			pStart++;
		}
		//将计算好的数值分别左移24位，16位，8位，0位
		result += (tmp << shift);
		shift -= 8;
		if (*pEnd == '\0')
			break;
		pStart = pEnd + 1;
		pEnd++;
	}
	return result;
}
	return buf;
}
```

```
void int2ipstr (const int ip, char *buf)
{
	sprintf (buf, "%u.%u.%u.%u",
		(uchar) * ((char *) &ip + 0),
		(uchar) * ((char *) &ip + 1),
		(uchar) * ((char *) &ip + 2),
		(uchar) * ((char *) &ip + 3)
	);
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

问题：

去掉字符串最前面的星号，如*A*BC*DEF*G***最终显示为ABCDEF*******

思路：

设置读写指针，然后，遇到*不移动，并记录下星号的个数，然后，在字符串结尾处补齐。

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

字符串左旋问题

- 定义字符串的左旋转操作：把字符串前面的若干个字符移动到字符串的尾部。如把字符串abcdef左旋转2位得到字符串cdefab。请实现字符串左旋转的函数，要求对长度为n的字符串操作的时间复杂度为O(n)，空间复杂度为O(1)。
- 以abcdef为例
- 1. ab->ba
- 2. cdef->fedc
- 原字符串变为bafedc
- 3. 整个翻转：cdefab
- //只要俩次翻转，且时间复杂度也为O（n）。

麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

int atoi (const char *s)

```
int atoi (const char *s)
{
        int i, n, sign;
        for(i = 0; isspace(s[i]); i++)
                ;
        sign = (s[i] == '-') ? -1 :1 ;
        if(s[i] == '+ '|| s[i] == ' -')
        i++;
        for(n = 0; isdigit(s[i]); i++)
            n = 10 * n + (s[i] - '0');
         return sign * n;
}
void itoa (int n, char *s)
{
        int i, j, sign;
        if((sign = n) < 0)
        n = -n;
        i = 0;
        do
        {
           s[i++] = n % 10 + '0';
        }while ((n /= 10) > 0);
        if(sign < 0)
           s[i++] = '-';
        s[i] = '\0';
        for(j = i; j >= 0; j--)
            printf("%c", s[j]);
}
```

```
double atof(char  *s)
{
    double val, power;
     int  i,  sign;
    for (i = 0; isspace(s[i]); i++)
                                                ;
                   sign = (s[i] == '-') ?-1:1;
if (s[i] == '+' || s[i] == '-')
       i++;
//处理实数中整数部分
 for (val = 0.0; isdigit(s[i]); i++)
val = 10.0 * val + (s[i] - '0');
 if (s[i] == '.')
                   i++;
//处理实数小数部分
for (power = 1.0; isdigit(s[i]); i++)
{
                   val = 10.0 * val + (s[i] - '0');
                   power *= 10;
 }
return sign * val / power;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

算法改错

Identify as many bugs and assumptions as you can in the following code.

```
// NOTE that there is/are (at least):
// 1 major algorithmic assumption
// 2 portability issues
// 1 syntax error
// Function to copy 'nBytes' of data from src to dst.
voidmyMemcpy(char* dst, const char* src, intnBytes)
{
        // Try to be fast and copy a word at a time instead of byte by byte
        int* wordDst = (int*)dst;
        int* wordSrc = (int*)src;
        int numWords = nBytes>> 2;
        for (inti=0; i<numWords; i++)
         {
                *wordDst++ = *wordSrc++;
         }
        intnumRemaining = nBytes - (numWords<< 2);
        dst = (char*)wordDst;
        src = (char*)wordSrc;
        for (inti=0 ; i<= numRemaining; i++);
        {
                *dst++ = *src++;
        }
}
```

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

链表

- LinkList基本操作
  - 插入，删除，合并
- 链表排序
- 链表逆向
- 循环链表判断
- 栈和队列的实现（数组，链表）
- 栈队互模拟

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

链表的基础操作：插入，删除，合并

- 一个链表，知道它的头指针为head，尾指针为tail。试写出链表的合并与删除算法，并考虑各种特殊情况。

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

链表排序

```
void SortList(node *head)
{
        node *p, *q, *s;
        int t;
        p = head;
        while(p)
       {
                s = p;
                q = p->next;
                while(q)
                {
                        if (q->value < s->value)
                                s = q;
                        q = q->next;
                }
                if (s !=p)
                {
                        t = s->value;
                        s->value = p->value;
                        p->value = t;
                }
                p = p->next;
        }
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

链表逆向

```
void ReverseList(node **head)
{
    node *p, *q, *r;
    p = *head;
    q = p->next;
    while(q!=NULL)
   {
        r = q->next;
        q->next = p;
        p = q;
        q = r;
    }
    (*head)->next = NULL;
    *head = p;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

循环链表

```
int FindLoop(node *head)
{
      node *p;
      node *q;
      if (head == NULL)
            return 0;
      p = head;
      q = head->next;
      while (q!=NULL&&
          q->next!=NULL&&p!=q)
      {
            p = p->next;
            q = q->next->next;
      }
      if (p==q)
            return 1;
      else
            return 0;
}
```

```
node *FindLoop(node *head)
{
   node *pc = head;
   node *pf = NULL;
   if (!pc)
      return NULL;
   while (pc)
   {
      pf = head;
      while(pf && pf != pc)
      {
         if (pc->next == pf || pc->next == pc)
              return pf;
         pf = pf->next;
      }
      pc = pc->next;
   }
   return NULL;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

删除链表中的结点P

删除链表中的结点P（P非最后一个结点）或者在P之前插入一个新结点O(1)

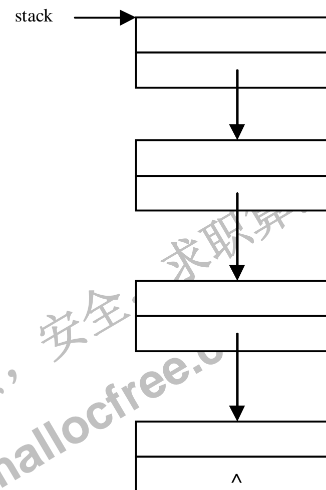
```
BOOL DeleteNode(node *list, node *delNode)
{
    if (list == NULL || delNode == NULL)
    {
            return FALSE;
    }
    if (delNode->next)
    {
            delNode->value = delNode->next->value;
            node *pTmp = delNode->next;
            delNode->next = delNode->next->next;
            delete pTmp;
            return TRUE;
     }
     return FALSE;
}
```

《麦洛克菲》内核，底层，安全，求职算法培训
www.mallocfree.com

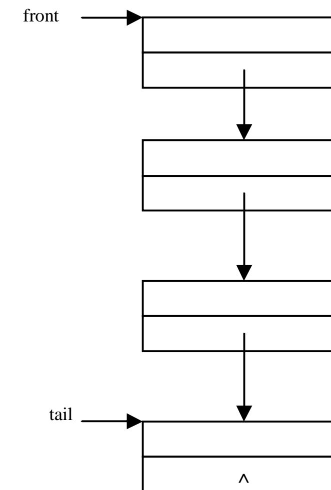
栈和队列的实现

```
int CreateStack(node **stack);
int DeleteStack(node **stack);
int Push(node **stack, void *data);
int Pop(node **stack, void **data);
BOOL IsStackEmpty(node *stack);


Qnode  *front;  //队头指针
Qnode  *rear;   //队尾指针
//队列接口
int CreateQue(Qnode **que);
int DeleteQue(Qnode **que);
int DeQueue(int *e);
int EnQueue(int *e);
BOOL IsQueEmpty(Qnode *que);
```



链栈结构



队列结构

麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

用2个栈来模拟队列

(1)入队

```
while(!S2.empty())
{
    S1.push(S2.top());
    S2.pop();
}
S1.push(c);//C是入队数据
```

(2)出队

```
while(!S1.empty())
{
    S2.push(S1.top());
    S1.pop();
}
c = S2.top();
S2.pop();
```

麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

树

- 树的遍历
- 二叉排序树：查找/插入/删除/建立
- 把二元查找树转变成排序的双向链表
- 在二元树中找出和为某一值的所有路径
- 怎样从顶部开始逐层打印二叉树结点数据
- 公共祖先

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

M

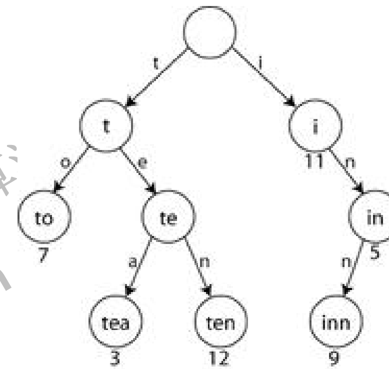
树的遍历

- 前序中序、后序中序确定一颗树
- 深度优先和广度优先遍历一颗树

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

字典树

- 1. 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
- 2. 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
- 3. 每个节点的所有子节点包含的字符都不相同。
- 给你a、b两个文件，各存放50亿条url，每条url各占用64字节，内存限制是4G，让你找出a、b文件共同的url。
- 8k的手机电话号码管理？



麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

二叉查找树

- 二叉排序树（Binary Sort Tree）又称二叉查找树。 它或者是一棵空树；或者是具有下列性质的二叉树：
- （1）若左子树不空，则左子树上所有结点的值均小于它的根结点的值；
- （2）若右子树不空，则右子树上所有结点的值均大于它的根结点的值；
- （3）左、右子树也分别为二叉排序树；

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

二元查找树转变成排序的双向链表

```
void change(Node *p, Node *&last) //中序遍历
{
    if (!p)
        return;
    change(p->left, last);
    if (last)
        last->right = p;
    p->left = last;

    last = p;
    change(p->right, last);
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

和为某一值的所有路径

输入一个整数和一棵二元树。
从树的根结点开始往下访问一直到叶结点所经过的所有结点形成一条路径。
打印出和与输入整数相等的所有路径。

```
void PrintTree(btree *t, int ivalue)
{
      If (t == NULL)
      {
            return;
      }
      if (t->left == NULL && t->right == NULL && t->value == iValue)
      {
            printf("%d\n", t->value)
      }
      if (t->left != NULL)
      {
            PrintTree(t->left, ivalue-t->value);
      }
      if (t->right != NULL)
      {
            PrintTree(t->right, ivalue-t->value);
      }
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

判断是否为二元查找树的后序遍历

```
bool verifySquenceOfBST(int squence[], int length)
{
    if(squence == NULL || length <= 0)
        return false;
    // root of a BST is at the end of post order traversal squence
    int root = squence[length - 1];
    // the nodes in left sub-tree are less than the root
    int i = 0;
    for(; i < length - 1; ++ i)
    {
        if(squence[i] > root)
            break;
    }
    // the nodes in the right sub-tree are greater than the root
    int j = i;
    for(; j < length - 1; ++ j)
    {
        if(squence[j] < root)
            return false;
    }
    // verify whether the left sub-tree is a BST
    bool left = true;
    if(i > 0)
        left = verifySquenceOfBST(squence, i);
    // verify whether the right sub-tree is a BST
    bool right = true;
    if(i < length - 1)
        right = verifySquenceOfBST(squence + i, length - i - 1);
    return (left && right);
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

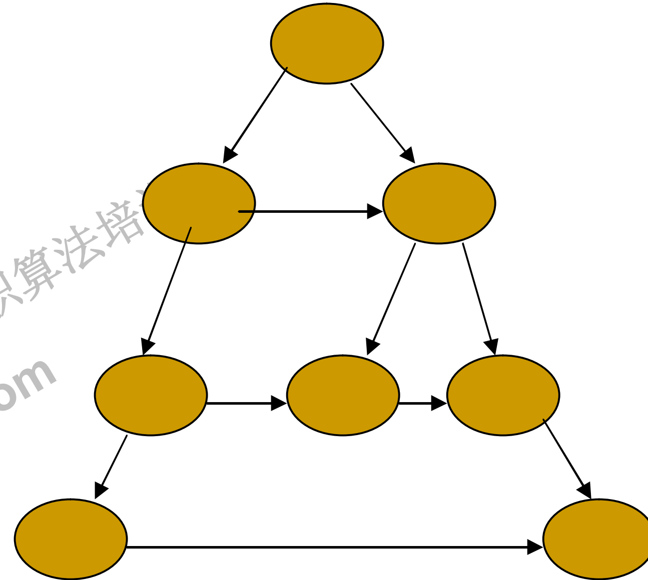
公共祖先

```
int FindLowestSharedAncestor(node *root, int value1, int value2)
{
	node   *curNode = root;
	while (1)
	{
		if (curNode->value > value1 && curNode->value > value2)
		{
			curNode = curNode->left;
		}
		else if (curNode->value < value1 &&
		   curNode->value < value2)
		{
			curNode = curNode->right;
		}
		else
		{
			return (curNode->value);
		}
	}
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

树的兄弟

```
typedef struct _btree
{
   struct _btree *left;
   struct _btree *right;
   struct _btree *sibling;
}btree, *pbtree;
```



开始,sibling是随机的

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

红黑树

- 红黑树是每个节点都带有颜色属性的二叉查找树，颜色或红色或黑色。在二叉查找树强制一般要求以外，对于任何有效的红黑树我们增加了如下的额外要求:
- 性质1. 节点是红色或黑色。
- 性质2. 根节点是黑色。
- 性质3 每个叶节点是黑色的。
- 性质4 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)
- 性质5. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

红黑树

“麦洛克菲内核，底层，安全，求职算法
www.mallocfree.com

数

- 素数
- 筛法

```
#define MAXN 40000
int SPrime(void)
{
    int i, j;
    int b_size;
    int prime[MAXN], b[MAXN];

    for (i = 2 ; i < MAXN; i ++ )
    prime[i] = 1 ;
    for (i = 2 ; i < MAXN; i ++ )
    {
      if (prime[i] == 1 )
                                {
         for (j = 2 ; i * j < MAXN; j ++ )
            prime[i * j] = 0 ;
                                }
    }
    for (i = 0 , j = 0; j < MAXN; j ++ )
    {
      if (prime[j] == 1 )
      {
         b[i] = j;
         i ++ ;
      }
    }
    b_size = i;
    return b_size;
}
```

```
bool IsPrime(int n)
{
        int i;
        for(i = 2; i <= sqrt(n+1);   i++)
        {
            if(n % i == 0)
                return 0;
        }
        return  1;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

M

证明题

证明5个任意数中必有3个数和能被三整除

证明：

把自然数按除以3后的余数分为：0，1，2
现任意抽取5个数,如果5个数中出现了3个同类的,则这3个数相加必然被3整除；如果5个数中找不到3个同类的数,那么必然是其中两类数各有两个,还有一个数在剩下的那类里面,比如1个余0的,2个余1的,2个余2的或者2个余0的,2个余1的,1个余2的,等等组合.不论组合怎样,总能找到1个余0的,1个余1的,1个余2的,这三个数相加就能被3整除

所以总能找到3个数的和被3整除

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

数组中重复的数

```
#define     MAXMUM 65536
void FindRepeated(int a[], in n)
{
        int tmp[MAXMUM];
        int i;
        for (i = 0; i < MAXMUM; i++)
        {
                tmp[i] = 0;
        }
        for (i = 0; i < n; i ++)
        {
                tmp[a[i]]++;
        }
        for (i = 0; i < MAXMUM; i++)
        {
                if (tmp[i]>1)
                    printf("%d", i);
        }
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

中位数

在一个文件中有 10G 个整数，乱序排列，要求找出中位数。内存限制为 2GB。

```
#define MAX_INDEX 1.25*1024*1024*1024
int findmidint()
{
	char a[MAX_INDEX];
	long value = 0;
	long total = 0;
	long index = 0;
	for(int i = 0; i < MAX_INDEX; i++)
	{
		a[i] = 0;
	}
	while ((value = GetANumFromFile()) != EOF)
	{
		if (!(a[value >> 3] & (1 << (value & 7)))
		{
			total++;
			a[value >> 3] |= (1 << (value & 7));
		}
	}
	for (int i = 0; i < MAX_INDEX * 8; i++)
	{
		if ((a[i >> 3] & (1 << (i & 7)))
		{
			index++;
			if (index == total/2)
				return i;
		}
	}
	return 0;
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

排序

- 插入（希尔）
- 选择（堆排序）
- 交换（冒泡，快速）
- 归并排序
- 基数排序

重点：

原理

实现

复杂度（平均，最坏）

基本插入排序的时间复杂度为O(n2)，是一种稳定排序算法。
希尔排序的复杂度为：O(n1.25)，它不是稳定排序。
简单选择排序的时间复杂度为O(n^2)，它是不稳定排序。
堆排序的时间复杂度为O(nlogn)，它是不稳定排序。
冒泡排序的时间复杂度为O(n^2)，它是稳定排序。
快速排序的时间复杂度为O(nlogn)，最坏情况为O(n^2),不稳定排序。
归并排序的时间复杂度为O(nlogn)，为稳定排序。需要额外的空间：O(n)。
基数排序法是属于稳定性的排序，其时间复杂度为O (d(n+rd))，其中rd为所采取的基数，而d为位数

插入、冒泡排序的速度较慢，但参加排序的序列局部或整体有序时，这种排序能达到较快的速度。反而在这种情况下，快速排序反而慢了。

当n较小时，对稳定性不作要求时宜用选择排序，对稳定性有要求时宜用插入或冒泡排序。

若待排序的记录的关键字在一个明显有限范围内时，且空间允许是用桶排序。

当n较大时，关键字元素比较随机，对稳定性没要求宜用快速排序。

当n较大时，关键字元素可能出现本身是有序的，对稳定性有要求时，空间允许的情况下。宜用归并排序。

当n较大时，关键字元素可能出现本身是有序的，对稳定性没有要求时宜用堆排序。

“麦洛克菲”内核，底层，安全，求职算法培训 www

冒泡排序

```
void bubblesort(int a[], int n)
{
      int i, j, tmp;
      for (i = 0; i < n-1; i++)
      {
            for (j = n; j >= i+1; j--)
            {
                  if (a[j] < a[j-1])
                  {
                        tmp = a[j];
                        a[j] = a[j-1];
                        a[j-1] = tmp;
                  }
            }
      }
}
```

冒泡排序的时间复杂度为*O*(*n2*)，它是稳定排序

“麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

快速排序

```
void q_sort(int numbers[], int left, int right)
{
  int pivot, l_hold, r_hold;
  l_hold = left;
  r_hold = right;
  pivot = numbers[left];
  while(left < right)
  {
    while ((numbers[right] >= pivot) && (left < right))
      right--;
    if (left != right)
    {
      numbers[left] = numbers[right];
      left++;
    }
    while ((numbers[left] <= pivot) && (left < right))
      left++;
    if (left != right)
    {
      numbers[right] = numbers[left];
      right--;
    }
  }
  numbers[left] = pivot;
  pivot = left;
  left = l_hold;
  right = r_hold;
  if (left < pivot)
    q_sort(numbers, left, pivot-1);
  if (right > pivot)
    q_sort(numbers, pivot+1, right);
}
```

```
void quickSort(int numbers[], int array_size)
{
      q_sort(numbers, 0, array_size - 1);
}
```

快速排序的时间复杂度为$O(n\log n)$，最坏情况为$O(n2)$。
如何将一个数组中的负数放在正数后面？

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

堆排序

```
void sfilter(int a[], int l, int m)
{
        int i, j x;
        i = l;
        j = 2 * i;
        x = a[i];
        while ( j <= m)
        {
                if (j < m && a[j] < a[j+1])
                        j++;
                if (x < a[j])
                {
                        a[i] = a[j];
                        i = j;
                        j = 2 * i;
                }
                else
                {
                        j = m + 1;
                }
        }
        a[i] = x;
}
void heapsort(int a[], int n)
{
        int i, w;
        for (i = n/2; i >= 1; i--)
                sfilter(a, i, n);
        for (i = n; i >= 2; i--)
        {
                w = a[i];
                a[i] = a[1];
                a[1] = w;
                sfilter(a, 1, i - 1);
        }
}
```

堆排序的时间复杂度为*O*(*n*log*n*)，它是不稳定排序。

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

查找

- 折半查找
- 二叉排序树
- HASH查找

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

折半查找

```
int BinSearch(int a[], int n, int k)
{
        int low, high, mid, find, i;

        find = 0;
        low = 1;
        high = n;
        while (low <= high && !find)
        {
                mid = (low + high)/2;
                if (a[mid] < k)
                        low = mid + 1;
                else if (a[mid] > k)
                        high = mid - 1;
                else
                {
                        i = mid;
                        find = 1;
                }
        }
        if (!find)
                i = 0;
        return i;
}
```

折半查找必须满足两个条件：一，元素必须是连续存储；二，元素必须有序。

麒麟远创的题目，马路找车。

如果数组中有重复的元素，返回最先出现的元素？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

二元查找树查找

```
btree *search(btree *b, int x)
{
	if (b == NULL)
	{
		return NULL;
	}
	else
	{
		if (b->data == x)
		{
			return b;
		}
		else if (x < b->data)
		{
			return (search(b->left));
		}
		else
		{
			return (search(b->right));
		}
	}
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

HASH查找

问题1：给你一个单词a，如果通过交换单词中字母的顺序可以得到另外的单词b，那么定义b是a的兄弟单词。现在给你一个字典，用户输入一个单词，让你根据字典找出这个单词有多少个兄弟单词。

问题2：请编写一个高效率的函数来找出字符串中的第一个无重复字符。例如，"total"中的第一个无重复字符上一"o"。

Hash表如何解决冲突？各有什么优缺点？

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

倒排表

- 一个很大的字符串，如何在里面查找某个单词？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

递归算法应用

- 不允许使用任何全局或局部变量编写
- int strlen(char *strDest)。

```
int strlen( const char* s )
{
    return *s?1+strlen(s+1):0;
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

请反向的输出一个字符串：比如”hello, world!”，打印出来是：”!dlrow, olleh”。

```
void inverse(char *p)
{
    if( *p = = '\0' )
        return;
    inverse( p+1 );
    printf( "%c", *p );
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

递归实现链表转置

```
list ResverseList(list l)
{
  if(!l || !l->next)
    return l;
  list n = reverse(l->next);
  l->next->next = l;
  l->next=null;
  return n;
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

计算一棵二叉树t的高度

```
int height(btree *t)
{
	int h, h1, h2;
	if (s == NULL)
	{
		return (0);
	}
	else
	{
		h1 = height(t->left);
		h2 = height(t->right);
		if (h1 > h2)
		{
			h = h1 + 1;
		}
		else
		{
			h = h2 + 1;
		}
		return (h);
	}
}
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

一个台阶总共有n 级，如果一次可以跳1 级，也可以跳2 级。求总共有多少总跳法，并分析算法的时间复杂度。

sum(N)=1 (N=1);
sum(N)=2 (N=2);
sum(N)=sum(N-1)+sum(N-2) (N>2);

```
int jump_sum(int n)  //递归版本
{
  if (n == 1 || n == 2)
      return n;
   return jump_sum(n-1)+jump_sum(n-2);
}
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

算法设计关键

- （1）循环
  - 1. 数组
  - 2. 字符串
  - 3. 链表
  - 4. 栈空/栈满/队空/队满
  - 5. 指针大小比较
  - 6. 数为正
  - 7. 反复循环直到条件成熟
- （2）各种情况完备考虑
- （3）递归解决

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

海量数据处理

- 1G = 10亿
- 2G = 2*1000*1000*1000=20亿
- 2^32=4*2^30 = 4G = 40亿
- 1M = 1*1000*1000 = 100万

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

海量数据处理--常用算法

- 1．Bloom filter
- 适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集
- 2．Hashing
- 适用范围：快速查找，删除的基本数据结构，通常需要总数据量可以放入内存。
- 3．bit-map
- 适用范围：可进行数据的快速查找，判重，删除，一般来说数据范围是int的10倍以下
- 4．堆
- 适用范围：海量数据前n大，并且n比较小，堆可以放入内存
- 5．分区划分
- 适用范围：第k大，中位数，不重复或重复的数字
- 6．倒排索引(Inverted index)
- 适用范围：搜索引擎，关键字查询
- 7．外排序
- 适用范围：大数据的排序，去重
- 8．trie树
- 适用范围：数据量大，重复多，但是数据种类小可以放入内存
- 9．分布式处理 mapreduce
- 适用范围：数据量大，但是数据种类小可以放入内存
- 基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

海量数据处理

- 实际例子：
- 1．海量日志数据，提取出某日访问百度次数最多的那个IP。
- 2．搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的10个查询串，要求使用的内存不能超过1G。
- 3．有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。
- 4．有10个文件，每个文件1G，每个文件的每一行存放的都是用户的query，每个文件的query都可能重复。要求你按照query的频度排序。
- 5．给定a、b两个文件，各存放50亿个url，每个url各占64字节，内存限制是4G，让你找出a、b文件共同的url？
- 6．在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数。
- 7．给40亿个不重复的unsigned int的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中？
- 8．怎么在海量数据中找出重复次数最多的一个？
- 9．上千万或上亿数据（有重复），统计其中出现次数最多的钱N个数据。

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

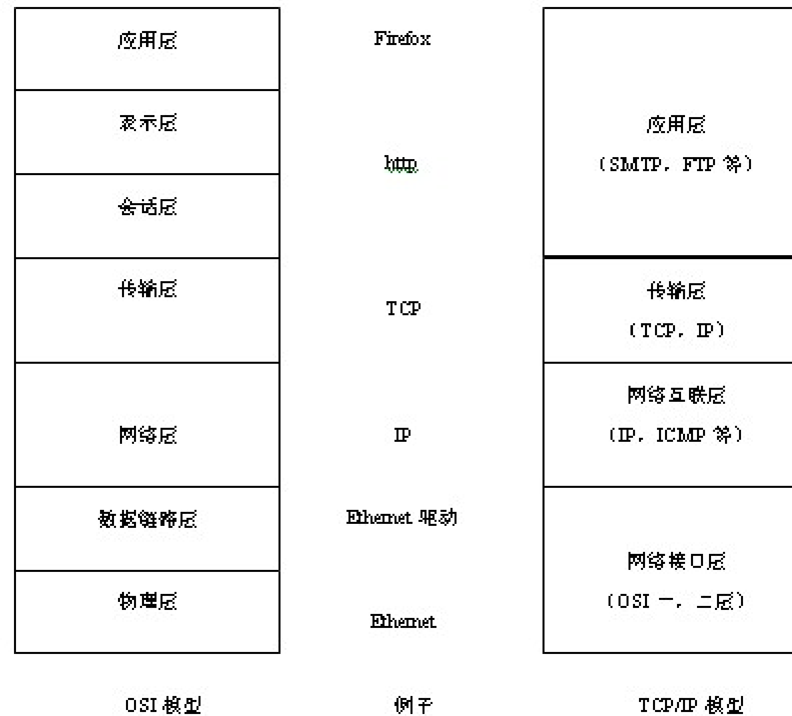
M

网络

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

网络协议模型

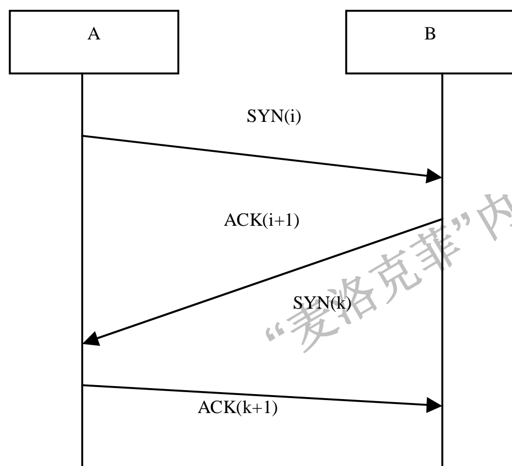
- ICMP是什么协议，处于哪一层?
- Internet控制报文协议，处于网络层（IP层）



"麦洛克菲"内核，底层
www.m

TCP三次握手（连接与断开）

- SYN FLOOD攻击



“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

TCP定时器

- (1)连接建立(connection establishment)定时器，在发送SYN报文段建立一条新连接时启动。如果没有在75秒内收到响应，连接建立将中止。
- (2)重传(retransmission)定时器，在TCP发送数据时设定。如果定时器已超时而对端的确认还未到达，TCP将重传数据。重传定时器的值(即TCP等待对端确认的时间)是动态计算的，取决于TCP为该连接测量的往返时间和该报文段已被重传的次数。
- (3)延迟ACK(delayed ACK)定时器，在TCP收到必须被确认但无需马上发出确认的数据时设定。TCP等待200 ms后发送确认响应。如果，在这200 ms内，有数据要在该连接上发送，延迟的ACK响应就可随着数据一起发送回对端，称为捎带确认。
- (4)持续(persist)定时器，在连接对端通告接收窗口为0，阻止TCP继续发送数据时设定。由于连接对端发送的窗口通告不可靠(只有数据才会被确认，ACK不会被确认)，允许TCP继续发送数据的后续窗口更新有可能丢失。因此，如果TCP有数据要发送，但对端通告接收窗口为0，则持续定时器启动，超时后向对端发送1字节的数据，判定对端接收窗口是否已打开。与重传定时器类似，持续定时器的值也是动态计算的，取决于连接的往返时间，在5秒到6 0秒之间取值。
- (5)保活(keepalive)定时器，在应用进程选取了插口的SO _KEEPALIVE选项时生效。如果连接的连续空闲时间超过2小时，保活定时器超时，向对端发送连接探测报文段，强迫对端响应。如果收到了期待的响应，TCP可确定对端主机工作正常，在该连接再次空闲超过2小时之前，TCP不会再进行保活测试。如果收到的是其他响应，TCP可确定对端主机已重启。如果连续若干次保活测试都未收到响应，TCP就假定对端主机已崩溃，尽管它无法区分是主机故障(例如，系统崩溃而尚未重启)，还是连接故障(例如，中间的路由器发生故障或电话线断了)。
- (6)FIN_WAIT_2 定时器，当某个连接从FIN_WAIT_1状态变迁到FIN_WAIT_2状态，并且不能再接收任何新数据时(意味着应用进程调用了close，而非shutdown，没有利用TCP的半关闭功能)，FIN_WAIT_2定时器启动，设为10分钟。定时器超时后，重新设为75秒，第二次超时后连接被关闭。加入这个定时器的目的是为了避免如果对端一直不发送FIN，某个连接会永远滞留在FIN_WAIT_2状态。
- (7)TIME_WAIT 定时器，一般也称为2MSL定时器。2MSL指两倍MSL，即最大报文段生存时间。当连接转移到TIME_WA I T状态，即连接主动关闭时，定时器启动

TCP/UDP协议区别

- UDP协议是一个无连结的数据报协议。它是一个 “不可靠”协议，不是因为它特别不可靠，而是因为它不检查数据包是否已经到达目的地，并且不保证它们按顺序到达。如果一个应用程序需要这些特点，它必须自己提供或者使用TCP。
- UDP的典型性应用是如流媒体（音频和视频等）这样按时到达比可靠性更重要的应用，或者如DNS查找这样的简单查询／响应应用，如果建立可靠的连结所作的额外工作将是不成比例地大。
- TCP协议是一个可靠的、面向连结的传输机制，它提供一种可靠的字节流保证数据完整、无损并且按顺序到达。TCP尽量连续不断地测试网络的负载并且控制发送数据的速度以避免网络过载。另外，TCP试图将数据按照规定的顺序发送。
- TCP协议的优缺点？

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

M

ARP协议与ARP攻击

- 以太网设备比如网卡都有自己全球唯一的MAC地址，它们是以MAC地址来传输以太网数据包的，但是它们却识别不了IP包中的IP地址，所以在以太网中进行IP通信的时候就需要一个协议来建立IP地址与MAC地址的对应关系，以使IP数据包能发到一个确定的地方去。这就是ARP(Address Resolution Protocol，地址解析协议)。从IP地址到物理地址的映射有两种方式：表格方式和非表格方式。ARP具体说来就是将网络层（IP层，也就是相当于OSI的第三层）地址解析为数据连接层（MAC层，也就是相当于OSI的第二层）的MAC地址。ARP协议是通过IP地址来获得MAC地址的。
- ARP是如何通过IP地址来获得MAC地址的呢？假如某机器A要向主机B发送报文，会查询本地的ARP缓存表，找到B的IP地址对应的MAC地址后就会进行数据传输。如果未找到，则广播 A一个ARP请求报文（携带主机A的IP地址Ia——物理地址Pa），请求IP地址为Ib的主机B回答物理地址Pb。网上所有主机包括B都收到ARP请求，但只有主机B识别自己的IP地址，于是向A主机发回一个ARP响应报文。其中就包含有B的MAC地址，A接收到B的应答后，就会更新本地的ARP缓存。接着使用这个MAC地址发送数据（由网卡附加MAC地址）。因此，本地高速缓存的这个ARP表是本地网络流通的基础，而且这个缓存是动态的。ARP 表：为了回忆通信的速度，最近常用的MAC地址与IP的转换不用依靠交换机来进行，而是在本机上建立一个用来记录常用主机IP－MAC映射表，即ARP 表。
- ARP攻击就是通过伪造IP地址和MAC地址实现ARP欺骗，能够在网络中产生大量的ARP通信量使网络阻塞，攻击者只要持续不断的发出伪造的ARP响应包就能更改目标主机ARP缓存中的IP-MAC条目，造成网络中断或中间人攻击。

WINDWOS网络5种IO模型

- A.选择（Select）
- B.异步选择（WSAAsyncSelect）
- C.事件选择（WSAEventSelect）
- D.重叠I/O（Overlapped I/O）
- E.完成端口（Completion Port)

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

MTU的大小？

- MTU是最大的TCP/IP传输单元，即Maximum Transmission Unit。Windows默认为1500字节，这是以太网的分组标准。
- EtherNet(一般上网方式，默认值):1500
- PPPoE/ADSL:1492
- Dial Up/Modem:576
- ping -f -l 1500 127.0.0.1

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

IP地址与子网掩码

- (1)IP地址分类
- 共4个字节
- A类：第1个字节表示网络，第1位为0：1-126
- B类：前2个字节表示网络，前2位为10，128-191
- C类：前3个字节表示网络，前3位为110，192-223
- 私有：10.x.x.x./172.x.x.x/192.168.x.x
- 特殊IP地址：
- 全0：主机自己
- 全1：有线广播
- 网0：主机1：网络上的所有主机
- 网：主机1：定向网络广播
- 127：XXX：回送
- (2)掩码
- A：255.0.0.0
- B：255.255.0.0
- C：255.255.255.0
- (3)子网掩码
- 子网掩码是一个32位地址，是与IP地址结合使用的一种技术。它的主要作用有两个，一是用于屏蔽IP地址的一部分以区别网络标识和主机标识，并说明该IP地址是在局域网上，还是在远程网上。二是用于将一个大的IP网络划分为若干小的子网络。用于子网掩码的位数决定于可能的子网数目和每个子网的主机数目。

A、确定哪些组地址归我们使用。比如我们申请到的网络号为 “210.73.a.b”，该网络地址为C类IP地址，网络标识为“210.73.a”，主机标识为“.b”。
B、根据我们现在所需的子网数以及将来可能扩充到的子网数，用宿主机的一些位来定义子网掩码。比如我们现在需要12个子网，将来可能需要16个。用第四个字节的前四位确定子网掩码。前四位都置为“1”（即把第四字节的最后四位作为主机位，其实在这里有个简单的规律，非网络位的前几位置1原网络就被分为2的几次方个网络，这样原来网络就被分成了2的4次方16个子网），即第四个字节为“11110000”，这个数我们暂且称作新的二进制子网掩码。
C、把对应初始网络的各个位都置为“1”，即前三个字节都置为“1”，第四个字节低四位置为“0”，则子网掩码的间断二进制形式为：“11111111.11111111.11111111.11110000”
D、把这个数转化为间断十进制形式为：“255.255.255.240”

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

掩码与子网掩码

- 一，利用子网数来计算
- 在求子网掩码之前必须先搞清楚要划分的子网数目，以及每个子网内的所需主机数目。
- 1)将子网数目转化为二进制来表示
- 2)取得该二进制的位数，为 N
- 3)取得该IP地址的类子网掩码，将其主机地址部分的的前N位置 1 即得出该IP地址划分子网的子网掩码。
- 如欲将B类IP地址168.195.0.0划分成27个子网：
- 1)27=11011
- 2)该二进制为五位数，N = 5
- 3)将B类地址的子网掩码255.255.0.0的主机地址前5位置 1，得到 255.255.248.0即为划分成 27个子网的B类IP地址 168.195.0.0的子网掩码。
- 二，利用主机数来计算
- 1)将主机数目转化为二进制来表示
- 2)如果主机数小于或等于254（注意去掉保留的两个IP地址），则取得该主机的二进制位数，为 N，这里肯定 N<8。如果大于254，则 N>8，这就是说主机地址将占据不止8位。
- 3)使用255.255.255.255来将该类IP地址的主机地址位数全部置1，然后从后向前的将N位全部置为 0，即为子网掩码值。
- 如欲将B类IP地址168.195.0.0划分成若干子网，每个子网内有主机700台：
- 1) 700=1010111100
- 2)该二进制为十位数，N = 10
- 3)将该B类地址的子网掩码255.255.0.0的主机地址全部置 1，得到255.255.255.255 然后再从后向前将后 10位置0,即为： 11111111.11111111.11111100.00000000
- 即255.255.252.0。这就是该欲划分成主机为700台的B类IP地址 168.195.0.0的子网掩码。

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

M

数据库

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

数据库设计与SQL语句编程

- 其中SQL四条最基本的数据操作语句：Insert，Select，Update和Delete，这四条语句是数据库查询中使用最频繁的四条语句，因此应该牢固掌握。
- 1. 选择：select * from table where 范围
- 2. 插入：insert into table(field1,field2) values(value1,value2)
- 3. 删除：delete from table1 where 范围
- 4. 更新：update table set field1=value1 where 范围
- 5. 查找：select * from table where field1 like '%value%'
- 6. 排序：select * from table order by field1,field2 [desc]

"麦洛克菲"内核，底层，安全，求职算法培训 www.mallocfree.com

索引

- (1)聚集索引
- 聚集索引根据数据行的键值在表或视图中排序和存储这些数据行。索引定义中包含聚集索引列。每个表只能有一个聚集索引，因为数据行本身只能按一个顺序排序。只有当表包含聚集索引时，表中的数据行才按排序顺序存储。如果表具有聚集索引，则该表称为聚集表。如果表没有聚集索引，则其数据行存储在一个称为堆的无序结构中。
- (2)非聚集索引
- 非聚集索引具有独立于数据行的结构。非聚集索引包含非聚集索引键值，并且每个键值项都有指向包含该键值的数据行的指针。从非聚集索引中的索引行指向数据行的指针称为行定位器。行定位器的结构取决于数据页是存储在堆中还是聚集表中。对于堆，行定位器是指向行的指针。对于聚集表，行定位器是聚集索引键。
- 索引是一个称为B树的数据结构。在聚簇索引中，索引树的叶级页包含实际的数据：记录的索引顺序与物理顺序相同。在非聚簇索引中，叶级页指向表中的记录：记录的物理顺序与逻辑顺序没有必然的联系。索引的作用主要是为了在查询时提高查询的效率，并且尽量减小更新时的开销。
- 聚簇索引非常像目录表，目录表的顺序与实际的页码顺序是一致的。非聚簇索引则更像书的标准索引表，索引表中的顺序通常与实际的页码顺序是不一致的。一本书也许有多个索引。例如，它也许同时有主题索引和作者索引。同样，一个表可以有多个非聚簇索引。一个表最多只能有一个聚簇索引。

M

事务

- 事务是单个的工作单元。如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分。如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除。也就是说，事务是由一系列的“原子”操作组成的，这些原子操作必须全部完成，否则所有的动作都会被取消并恢复到初始状态。
- 开始事务使用BEGIN TRANSACTION 语句显，以 COMMIT 或 ROLLBACK 语句结束。
- 事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务。
- 原子性：事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。
- 一致性：事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如B树索引或双向链表）都必须是正确的。
- 隔离性：由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。
- 持久性：事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

存储引擎

- 数据库如MySQL中的数据用各种不同的技术存储在文件(或者内存)中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。这些不同的技术以及配套的相关功能在MySQL中被称作存储引擎(也称作表类型)。
- MyISAM： Mysql的默认数据库，最为常用。拥有较高的插入，查询速度，但不支持事务
- InnoDB：事务型数据库的首选引擎，支持ACID事务，支持行级锁定
- BDB： 源自Berkeley DB，事务型数据库的另一种选择，支持COMMIT和ROLLBACK等其他事务特性
- Memory：所有数据置于内存的存储引擎，拥有极高的插入，更新和查询效率。但是会占用和数据量成正比的内存空间。并且其内容会在Mysql重新启动时丢失
- Merge：将一定数量的MyISAM表联合而成一个整体，在超大规模数据存储时很有用
- Archive：非常适合存储大量的独立的，作为历史记录的数据。因为它们不经常被读取。Archive拥有高效的插入速度，但其对查询的支持相对较差
- Ederated： 将不同的Mysql服务器联合起来，逻辑上组成一个完整的数据库。非常适合分布式应用
- Cluster/NDB：高冗余的存储引擎，用多台数据机器联合提供服务以提高整体性能和安全性。适合数据量大，安全和性能要求高的应用
- CSV： 逻辑上由逗号分割数据的存储引擎
- BlackHole：黑洞引擎，写入的任何数据都会消失，一般用于记录binlog做复制的中继
- MySQL>:show engines;
- CREATE TABLE mytable (id int, title char(20)) ENGINE = INNODB
- ALTER TABLE mytable ENGINE = MyISAM

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

锁定

- 不同的存储引擎在不同的对象级别支持锁定，而且这些级别将影响可以同时访问的信息。得到支持的级别有三种：表锁定、块锁定和行锁定。支持最多的是表锁定，这种锁定是在MyISAM中提供的。在数据更新时，它锁定了整个表。这就防止了许多应用程序同时更新一个具体的表。这对应用很多的多用户数据库有很大的影响，因为它延迟了更新的过程。
- 页级锁定使用Berkeley DB引擎，并且根据上载的信息页(8KB)锁定数据。当在数据库的很多地方进行更新的时候，这种锁定不会出现什么问题。但是，由于增加几行信息就要锁定数据结构的最后8KB，当需要增加大量的行，也别是大量的小型数据，就会带来问题。
- 行级锁定提供了最佳的并行访问功能，一个表中只有一行数据被锁定。这就意味着很多应用程序能够更新同一个表中的不同行的数据，而不会引起锁定的问题。只有InnoDB存储引擎支持行级锁定。

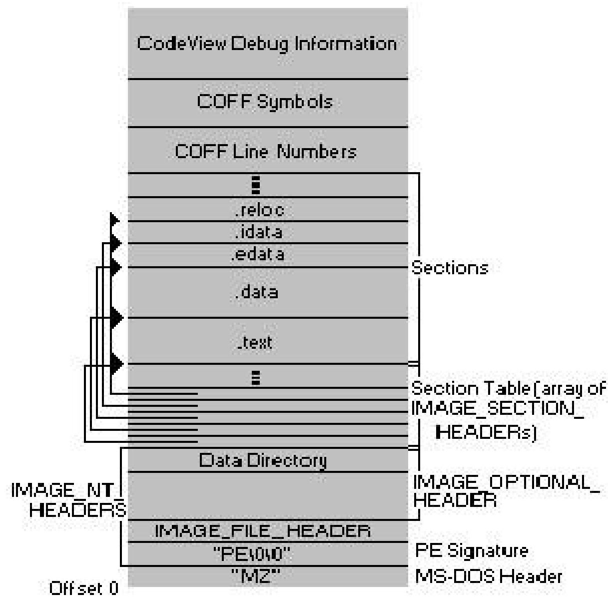
“麦洛克菲内核，底层，安全，求职算法培训” www.mallocfree.com

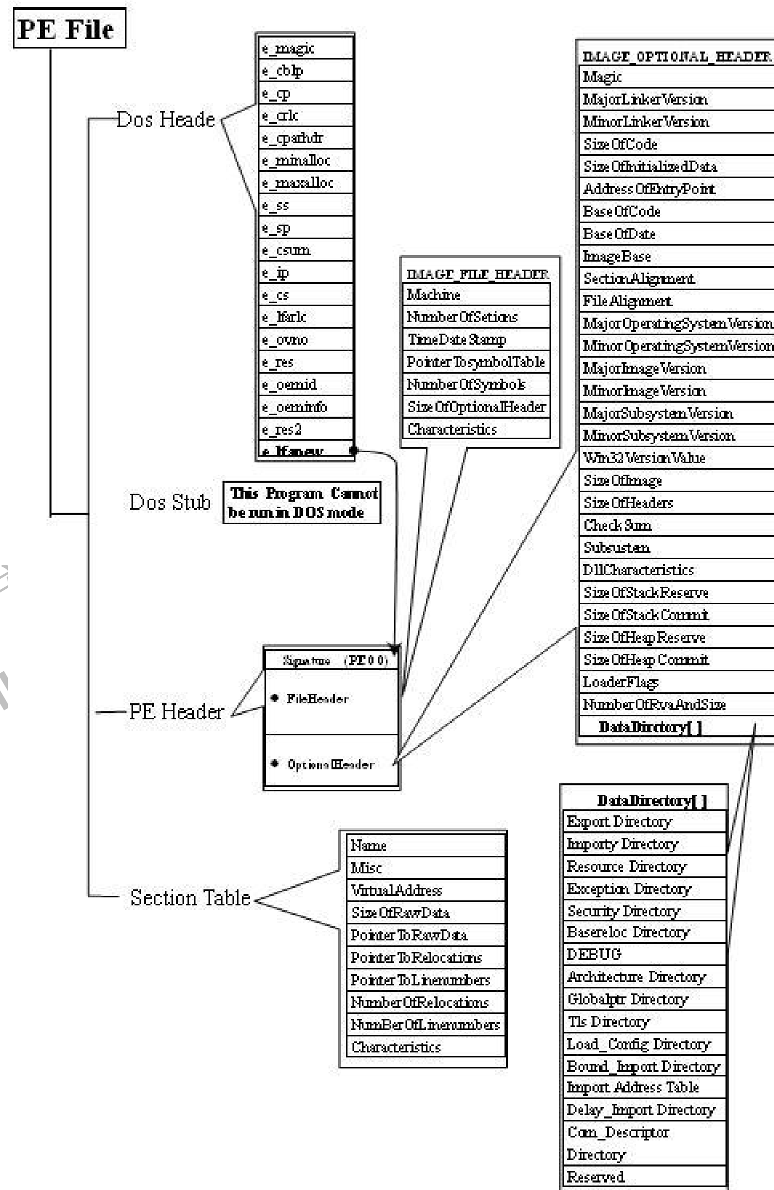
WINDOWS操作系统

- COM的线程模型
- 公寓(Apartment)有的译文译作"套间"。这个术语抽象的是COM对象的生存空间，你还真的可以想象成公寓，线程就是住在公寓里的人。套间定义了一组对象的逻辑组合，这些对象共享一组并发性和冲入限制。每个COM对象都属于某一个套间，一个套间可以包含多个COM对象。
- 单线程套间-->一个线程-->序列化访问所拥有的com对象
- 多线程套间-->多个线程-->直接访问所拥有的com对象（同步需要自己考虑）
- A.单线程套间STA
- Single-threaded Apartments，一个套间只关联一个线程，COM库保证对象只能由这个线程访问（通过对象的接口指针调用其方法），其他线程不得直接访问这个对象(可以间接访问，但最终还是由这个线程访问)。
- COM库实现了所有调用的同步，因为只有关联线程能访问COM对象。如果有N个调用同时并发，N-1个调用处于阻塞状态。对象的状态（也就是对象的成员变量的值）肯定是正确变化的，不会出现线程访问冲突而导致对象状态错误。
- B.多线程套间MTA
- Multithreaded Apartments，一个套间可以对应多个线程，COM对象可以被多个线程并发访问。所以这个对象的作者必须在自己的代码中实现线程保护、同步工作，保证可以正确改变自己的状态。

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

PE结构





M

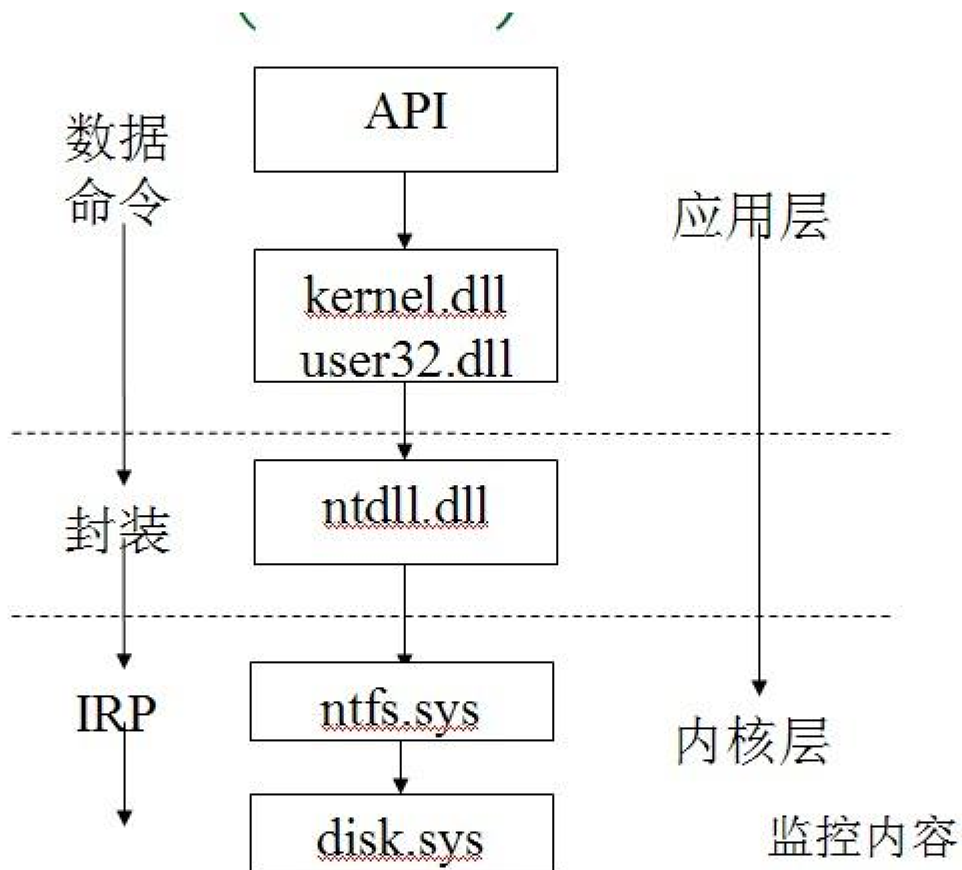
三个函数的区别

- CreateThread：是Windows的API函数(SDK函数的标准形式,直截了当的创建方式，任何场合都可以使用)，提供操作系统级别的创建线程的操作，且仅限于工作者线程。不调用MFC和RTL的函数时，可以用CreateThread，其它情况不要使用。因为：
  - C Runtime中需要对多线程进行纪录和初始化，以保证C函数库工作正常。
  - MFC也需要知道新线程的创建，也需要做一些初始化工作。
  - 有些CRT的函数象malloc(),fopen(),_open(),strtok(),ctime(),或localtime()等函数需要专门的线程局部存储的数据块，这个数据块通常需要在创建线程的时候就建立，如果使用CreateThread，这个数据块就没有建立，但函数会自己建立一个，然后将其与线程联系在一起，这意味着如果你用CreateThread来创建线程，然后使用这样的函数，会有一块内存在不知不觉中创建，而且这些函数并不将其删除，而CreateThread和ExitThread也无法知道这件事，于是就会有Memory Leak，在线程频繁启动的软件中，迟早会让系统的内存资源耗尽。
- _beginthreadex：MS对C Runtime库的扩展SDK函数，首先针对C Runtime库做了一些初始化的工作，以保证C Runtime库工作正常。然后，调用CreateThread真正创建线程。
- AfxBeginThread：MFC中线程创建的MFC函数，首先创建了相应的CWinThread对象，然后调用CWinThread::CreateThread,在CWinThread::CreateThread中，完成了对线程对象的初始化工作，然后，调用_beginthreadex(AfxBeginThread相比较更为安全)创建线程。它让线程能够响应消息，可用于界面线程，也可以用于工作者线程。

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

M

系统调用流程



M

LINUX

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

常用命令

- cp/mv/ls/rm/ifconfig/pwd/find/du/df/ln/ps/kill -9
- find . –name xx -print | [xargs] grep “yy”
- [1] XXX | grep –i ‘hello’
- [2] XXX | xargs grep –i ‘hello’[1]的情况下，grep将通过管道读取XXXX的输出结果，并在该结果中搜索hello。[2]的情况下，xargs将通过管道读取XXXX的输出结果，并将该结果作为grep的最后的FILE参数，和grep –i ‘hello’组合成完整的命令（如 grep –i ‘hello’ stdio.h stdlib.h）后，执行该命令。grep从stdio.h stdlib.h文件中搜索hello。
- 内存：
- free -m
- top
- vmstat
- cat /proc/meminfo
- CPU：
- cat /proc/cpuinfo |grep xxx
- 内核：
- Uname –a

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

GCC

- Gcc –o –O – O2 –g –I –L –l
- G++
- O0：默认，无优化
- O1：代码大小和执行时间优化，但几乎不影响编译时间，也可以省略为O
- O2：几乎打开了所有的优化选项，但除了loop-unrolling和function-inlining
- O3：打开所有的优化选项
- gcc 命令的常用选项
- 选项 解释
- -ansi 只支持 ANSI 标准的 C 语法。这一选项将禁止 GNU C 的某些特色，例如 asm 或 typeof 关键词。
- -c 只编译并生成目标文件。
- -DMACRO 以字符串“1”定义 MACRO 宏。
- -DMACRO=DEFN 以字符串“DEFN”定义 MACRO 宏。
- -E 只运行 C 预编译器。
- -g 生成调试信息。GNU 调试器可利用该信息。
- -IDIRECTORY 指定额外的头文件搜索路径DIRECTORY。
- -LDIRECTORY 指定额外的函数库搜索路径DIRECTORY。
- -lLIBRARY 连接时搜索指定的函数库LIBRARY。
- -m486 针对 486 进行代码优化。
- -o FILE 生成指定的输出文件。用在生成可执行文件时。
- -O0 不进行优化处理。
- -O 或 -O1 优化生成代码。
- -O2 进一步优化。
- -O3 比 -O2 更进一步优化，包括 inline 函数。
- -shared 生成共享目标文件。通常用在建立共享库时。
- -static 禁止使用共享连接。
- -UMACRO 取消对 MACRO 宏的定义。
- -w 不生成任何警告信息。
- -Wall 生成所有警告信息。

麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

内核锁

- (1)原子操作
- (2)自旋锁
- 不可递归
- 不可睡眠
- 中断上下文可使用
- 短期持有
- (3)信号量
- 允许睡眠
- 适合长期占有
- 只能在进程上下文使用
- (4)完成变量
- (5)读写锁
- 一个任务通知另外一个任务发生了某个特定的事件。同步两个任务。

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

软中断和工作队列

- (1)中断处理程序局限：
- 异步方式执行并且可能会打断其他重要代码，甚至是其他中断处理程序
- 往往对硬件进行操作，需要很高的时限要求，效率要高，时间要少
- 中断上下文不能阻塞
- 一般只完成必要的数据拷贝
- (3)下半部机制
- 下半部的任务是执行与中断处理密切相关但中断处理程序本身不执行的工作。在下半部运行时，可以响应所有的中断。
- a.软中断
- b.tasklet
- c.工作队列
- 大多数时候选择tasklet；需要睡眠选择工作队列；执行频率和连续性高时选择软中断。

“麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

.Core文件

- 程序运行时的内存映像，在程序发生异常时产生。
- 查看：gdb程序名 core文件名→where
- 启用：ulimint –S –c unlimited
- 禁用：ulimit –c o

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

IPTABLE与netfilter

- Netfilter-iptables由两部分组成，一部分是Netfilter的“钩子”，另一部分则是指导这些钩子函数如何工作的一套规则--这些规则存储在被称为iptables的数据结构之中。钩子函数通过访问iptables来判断应该返回什么值给Netfilter框架。
- Netfilter框架为多种协议提供了一套类似的钩子（HOOK），用一个struct list_head nf_hooks[NPROTO][NF_MAX_HOOKS]二维数组结构存储，一维为协议族，二维为上面提到的各个调用入口。每个希望嵌入Netfilter中的模块都可以为多个协议族的多个调用点注册多个钩子函数（HOOK），这些钩子函数将形成一条函数指针链，每次协议栈代码执行到NF_HOOK()函数时（有多个时机），都会依次启动所有这些函数，处理参数所指定的协议栈内容。

- 每个注册的钩子函数经过处理后都将返回下列值之一，告知Netfilter核心代码处理结果，以便对报文采取相应的动作：
- NF_ACCEPT：继续正常的报文处理；
- NF_DROP：将报文丢弃；
- NF_STOLEN：由钩子函数处理了该报文，不要再继续传送；
- NF_QUEUE：将报文入队，通常交由用户程序处理；
- NF_REPEAT：再次调用该钩子函数。

补充阅读：
https://www.ibm.com/developerworks/cn/linux/l-ntflt/
http://blog.sina.com.cn/s/blog_6988074a0100sej9.html

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

M

设计模式

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

常用设计模式

- Factory 模式（工厂模式）
- Abstract 模式（抽象工厂模式）
- Singleton 模式（单例模式）
- Adapter 模式（适配器模式）
- Proxy模式（代理模式）
- Observer模式（观察者模式）

《设计模式精解》 （美）Alan Shalloway,James R.Trott

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

Factory 模式（工厂模式）

```
Factory* fac = new ConcreteFactory();
Product* p = fac->CreateProduct();
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Abstract 模式（抽象工厂模式）

- AbstractFactory* cf1 = new ConcreteFactory1();
- cf1->CreateProductA();
- cf1->CreateProductB();
- AbstractFactory* cf2 = new ConcreteFactory2();
- cf2->CreateProductA();
- cf2->CreateProductB();

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Singleton 模式（单例模式）

```
Singleton* Singleton::Instance()
{
    if (_instance == 0)
    {
        _instance = new Singleton();
    }
    return _instance;
}
Singleton* sgn = Singleton::Instance();
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

Adapter 模式（适配器模式）

```
void Adapter::Request()
{
    _ade->SpecificRequest();
}
Adaptee* ade = new Adaptee;
Target* adt = new Adapter(ade);
adt->Request();
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Proxy模式（代理模式）

```
void Proxy::Request()
{
    cout<<"Proxy request...."<<endl;
    _sub->Request();
}
Subject* sub = new ConcreteSubject();
Proxy* p = new Proxy(sub);
p->Request();
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Observer模式（观察者模式）

```
ConcreteSubject* sub = new ConcreteSubject();
Observer* o1 = new ConcreteObserverA(sub);
Observer* o2 = new ConcreteObserverB(sub);
sub->SetState("old");
sub->Notify();
sub->SetState("new"); //也可以由Observer调用
sub->Notify();
```

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

Facade 模式（门面模式）

```
Facade::Facade()
{
    this->_subs1 = new Subsystem1();
    this->_subs2 = new Subsystem2();
}
Facade::~Facade()
{
    delete _subs1;
    delete _subs2;
}
void Facade::OperationWrapper()
{
    this->_subs1->Operation();
    this->_subs2->Operation();
}
Facade* f = new Facade();
f->OperationWrapper();
```

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

设计题

- 美国盲人
- 电话号码备份问题
- 等等

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

面试篇

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

简历

- 简历标题：自己名字
- 简历文件名字：自己名字+基本信息
- 基本信息
- 教育背景
- 工作/实习经历
- 专业技能
- 项目经验
- 项目标题/项目时间/项目描述/项目成果/开发环境
- 工作成绩
- 自我评价

设计原则：

防止错误
针对性强
突出重点
突出优点

照片贴不贴？

政治面貌写不写？

年龄？

身高，体重？

培训经历要不要写？

麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

英语面试

- 1．准备自我介绍
- 2．准备工作介绍
- 3．项目英语介绍
- 4．与老外交流
- 5．英语简历

My name is Zhou Yangrong. I am now woking at Sun Microsystems. I have three years work expierence. I am good at C/C++ language. I graduated from Institute of Software, Chinese academy of sciences with a master degree. I feel honored to be here for a face-to-face interview.

大家注意在英语简历中，我们用的一些有用的表达方式：band 6（英语六级），be proficient in（精通于），be familiar with（熟练于），experience in（具有某相关方面的经验）。在投递外企的时候，我们还可以把外企上的招聘要求里面的英语语句照搬下来作为自己的能力的表达句，当然前提条件就是一定要真实。

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

项目面试

- 1. 提前复习
- 2. 梳理算法
- 3. 技术点
- 4. 不足
- 5. 你有什么问题想问我的吗？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

HR面试

- 1. 为什么跳槽
- 2. 目前薪水
- 3. 期望待遇
- 4. 职业规划
- 5. 个人优缺点

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

Offer选择

- 1. 薪水
- 2. 专业特长
- 3. 潜力方向
- 4. 企业顺序
- 5. 创业公司（是否拿到风投）

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

薪酬谈判

- 良好的面试表现
- 知己知彼
- 以offer谈offer
- 讨价还价
- 她拖你也拖
- 职业发展高度来让步或拒绝

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

如何过试用期？

- 1．期限与待遇：（0-6个月），80%-100%
- 2．自信，基本都能通过
- 3．积极主动
- 4．虚心学习
- 5．出色完成任务，做出能够量化的成绩
- 6．不迟到不早退，遵守公司纪律,不犯严重错误

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

如何跳槽？

- 360员工跳槽到百度？
- 频繁跳槽？
- 跳槽流程？
- 跳槽的最佳时间？

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

总结

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

C语言

- sizeof()
- 变量存储类型，作用域，生存空间，传参数
- 整数的存储 (高位优先/低位优先）
- 位运算
- 指针，指针与引用，函数指针
- static/extern /volatile/char/pragma/include/const/typedef/namespace
- 操作符

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

C++

- const，inline与#define区别
- C++面向对象的三大特性及其设计应用
- static_cast<>/dynamic_cast<>/reinterpret_cast<>
- 空类的实质内容是什么
- 构造函数中，如果类中含有动态内存分配，该如何处理？
- 成员变量的初始化的区别？顺序？静态成员的初始化？
- 设计一个类，只能生成该类的一个实例
- 基类析构函数的设计
- 赋值运算符的设计
- 构造、析构、赋值函数调用顺序。设计同步类CLocker？
- 继承与菱形继承
- 设计一个不能被继承的类
- 虚函数/纯虚函数/虚基类/接口
- 什么是多态？
- C++类对象的大小
- 重载/重写/重整
- 模板 （函数/类）
- STL标准模板库
- 禁止对象产生于堆或非堆中
- String类/智能指针/写时拷贝/引用计数
- 智能指针使用，实现一个智能指针
- i++与++i

"麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

M

数据结构与算法

- **算法设计一般思路**
- 字符串
- 链表插入，删除，合并，排序，逆向，循环链表
- 栈和队列
- 树的遍历
- 二叉排序树查找/插入/删除/建立
- 字典树
- 数
- 数组
- 排序
- 查找
- HASH
- 递归
- 海量数据处理

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

内存

- 虚拟内存空间
- 逻辑地址到物理地址转换
- 程序内存布局
- 堆和栈的区别
- 调用约定
- 栈桢（活动记录）//calling convention
- 内存分配
- 内存泄漏

麦洛克菲内核，底层，安全，求职算法培训 www.mallocfree.com

多线程/多进程

- 同步机制(critical_section/mutex/event/semaphore)
- 进程通信
- 生产者消费者队列

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

网络

- 协议模型：OSI/TCP-IP协议模型
- TCP三次握手（连接与断开）/SYN FLOOD攻击
- TCP定时器
- TCP/UDP协议区别
- ARP协议与ARP攻击
- SOCKET编程，如何获取一个网页的数据？
- Windows Socket 五种IO模型
- MTU的大小？
- IP地址与子网掩码

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

数据库

- SQL语言SELECT，DELETE，UPDATE，INSERT编程
- 表设计
- 索引：聚集/非聚集索引
- 事务
- 存储引擎
- 锁定：表/块/行
- MYSQL的常用字段

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

WINDOWS

- COM
- PE结构
- CreateThread/
  - _begintrheadex
  - AfxBeginThread()

"麦洛克菲内核，底层，安全，求职算法培训
www.mallocfree.com

Linux

- 常用命令
- GCC与00，01，02，03优化
- .core文件
- 内核锁
- 软中断和工作队列

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

设计模式

- Factory 模式（工厂模式）
- Abstract 模式（抽象工厂模式）
- Singleton 模式（单例模式）
- Adapter 模式（适配器模式）
- Proxy模式（代理模式）
- Observer模式（观察者模式）

《设计模式精解》（美）Alan Shalloway,James R.Trott

“麦洛克菲”内核，底层，安全，求职算法培训 www.mallocfree.com

M

设计题

- 美国盲人
- 电话号码备份问题
- 等等

"麦洛克菲"内核，底层，安全，求职算法培训
www.mallocfree.com

M

智力题

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

M

Q&A

“麦洛克菲”内核，底层，安全，求职算法培训
www.mallocfree.com

发件人 branda2010@163.com
标题 猎头诚邀-C方向高端机会参考，敬请回复！ 2011-4-20 16:26
收件人 我 其他操作

周先生，您好！
我是Chance（中国）猎头顾问Branda，受IBM研发中心委托寻找产品技术高端职位负责人，主要针对IBM **X或P服务器**上相关**驱动和内核**方向得**研发&管理**两个高端机会，请您参考！
若您对此职位感兴趣，请您于24小时内针对以上职位要求将您的最新中、英文简历 及"猎头职位反馈表"以邮件的形式回复我，我将会在收到回复后与您进一步沟通。**具体职位要点：在猎头职位反馈表中有具体描述，请您查看并结合自身优势填写，这将成为推荐成功与否的重要因素。**

若您身边看机会、适合的朋友，欢迎推荐

猎头职位1：Linux & KVM Architect
Job description
The Linux Technology Center organization in Greater China STG is seeking experienced open source software engineers for design, development and test of Linux and other Open Source software products.
These are challenging and high growth opportunities for r... technical leadership growth. This team is an integral part... Center as well as the WW Linux community. In addition,... China sales team (Hardware, Software, Services) to win...

高级猎头职位反馈表.doc

发件人 seaway wei <seaway0080@yahoo.com.cn>
标题 JD FROM TINA 2011-7-6 17:43
收件人 我 其他操作

Hello,Yangrong

Nice to meet you here! I'm Tina,headhunter,the JD from EMC is as follows,hope we can take further communication for the position.

Looking forward to your reply!

Principal SW engineer

(Location:Beijing)

General summary:

...eer to join a team of software professionals to ...u would be doing systems programming where ...ant. Our customers are worldwide and include ...uickly find the "gold" in the huge amount of data ...stas of data analysis for their companies.

...are dedicated to the whole team winning, and ...o learn and develop. Pioneers who will always be ...a crucial impact in creating at Greenplum. And

Greeting from MS - 收件箱 - Mozilla Thunderbird
文件(F) 编辑(E) 查看(V) 转到(G) 消息(M) 工具(T) 帮助(H)
获取消息 新建消息 通讯录 标签 搜索全部消息… <Ctrl+K>
收件箱 | Greeting from MS - 收件箱

回复 转发 归档 垃圾 删除
发件人 Carree Ma
标题 Greeting from MS 2011-7-22 14:32
收件人 我 其他操作

Dear Zhou Yangrong,
Nice to talk with you.
Please kindly find the JD for WIN Kernel SDE

Sincerely,
Carree Ma Jie
HR Staffing Team
Contact HR for Support

Need Info?	Go to HR web	Queries?	Email PRC HR Mail Box	Job?	Access Internal Global Careers Site	Training?	Visit Learning Central

SDEII-Hua-CIG.docx