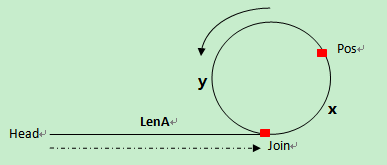
1. 怎么寻找单链表中环（环的起点）



定义两个指针p、q，p每次后移两个结点，称为快指针，q每次后移一个结点称为慢指针。假设pos为p、q相遇的结点，则此时q还未走完整个链表。

1. 证明快、慢指针相遇后，由相遇点开始慢指针向后移动LenA个结点到达环起点

慢指针移动了（LenA+x）个结点，则快指针移动了2\*（LenA+x）个结点，假设快指针延环走了n圈，则有n>=1。则可列：

**LenA+x = 2\*(LenA+x) – n\*(x+y)，**

化简有：

**n\*(x+y) = LenA+x，**

等式两侧同时加上join起点有：

**Join + n\*(x+y) = join +x+LenA**

又因为x+y为环一周的长度，所以等式左侧=join。join+x即为快慢指针相遇点Pos, 则有：

**Join = Pos+LenA，**

得证，由相遇点开始，慢指针向后移动LenA个结点到达环的起始结点。

1. LenA未知，如何得到Join结点？

注意，由上述证明可知Pos后移LenA个结点可到达Join，则由LenA==y。算法主要实现：

当快慢指针第一次相遇时，即都位于Pos位置，则使p指向链表头部，再将p、q逐步后移，则p、q会再Join处再次相遇，则得到环起点。

1. malloc/free，new/delete的区别
2. malloc/free 是c/c++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动态内存，但是new能够自动分配空间大小，而malloc需要计算字节数
3. malloc/free是库函数，malloc只能分配空间，不会自动调用构造函数。Free只能释放空间，不会自动调用析构函数。
4. new/delete是运算符，能完成动态内存分配与释放，以及自动调用构造函数与析构函数
5. new是类型安全的，而malloc不是。例如：

int\* p = new float[2]; //编译出错

int\* p = malloc(2\*sizeof(float)); //编译无法指出错误

1. malloc申请内存失败返回空指针，new申请内存失败会抛出异常。
2. new/delete的内部实现
3. new的实现

内存的分配：

CA\* p = operator new(sizeof(CA)); // 分配堆内存

CA::CA(p1)；//调用构造函数

当调用operator new[] 分配数组对象时，编译器会增加4或8字节的空间用来保存所分配的数组对象的数量。数组对象根据这个数量进行构造函数的调用

unsigned long\* ps = operator new[](20\*sizeof(CA)+sizeof(unsigned long)); //多分配unsigned long的空间

\*ps = 20; //保存对象的数量

CA\* p1 = (CA\*)(ps+1)；

CA\* pt = p1;

for (int i = 0; i < \*p; ++i)

{

CA::CA(pt);

pt += 1;

}

1. delete实现

CA::~CA(p1); //调用析构函数

operator delete(p); //释放内存

1. placement 技术

系统默认的new关键字除了分配堆内存还进行构造函数的调用，而实际中我们可能由一些已经预先分配好的内存区域，如何在这些已分配的内存来构建一个对象。

char buf1[100]; //预申请一块栈内存

CA \*p1 = (CA\*)buf1;

CA::CA(p1); //在栈内存上调用构造函数，会填充虚表指针。

p1->foo(); // foo为虚函数

p1->m\_a = 10;

char \*buf2 = new char[sizeof(CA)]; //预申请一块堆内存

CA \*p2 = (CA\*)buf2;

CA::CA(p2); //堆内存上调用构造函数

p2->foo();

p2->m\_a = 20;

p1->~CA();

p2->~CA();

delete[] buf2;

使用placement new 和placement delete

char buf1[100];

CA \*p1 = new(buf1) CA(10) //调用 operator new(size\_t, void\*)

p1->foo();

char \*buf2 = new char[sizeof(CA)];

CA \*p2 = new(buf2) CA(20); //调用 operator new(size\_t, void\*)

p2->foo();

p1->~CA();

operator delete(p1, buf1); //调用 operator delete(void\*, void\*) 还内存

p2->~CA();

operator delete(p2, buf2); //调用 operator delete(void\*, void\*)

delete[] buf2;

1. 早绑定/晚绑定与多态

编译时多态（静态多态）：通过重载函数实现，函数调用的地址在编译期间就能决定。称为早绑定。

运行时多态（动态多态）：通过虚函数实现，函数调用的地址需要在运行时才确定。称为晚绑定。

1. 死锁

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们将违法推进下去。

产生死锁的四个必要条件：

1. 互斥条件：一个资源每次只能被一个进程使用
2. 请求和保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放
3. 不可剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。
4. 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。
5. vector与list的实现原理
6. vector相当于一个动态数组，size函数返回已用空间大小，capacity函数返回总空间大小。当size与capacity相等时，说明vector目前的空间已用完。如果再添加新元素，则会引起vector空间的动态增长。这个自动增长包括重新分配内存空间、拷贝原空间、释放原空间。

reserve(n)预先分配一块指定大小的内存空间，resize只改变元素的数目，不改变vector的容量。

1. list 双向链表，信息块info，前驱指针Pre，后继指针Post。使用非连续的内存空间。可在两端push、pop。
2. 类中的默认函数

构造函数、析构函数、拷贝构造函数、默认重载赋值运算符函数、默认重载取地址运算符函数、默认重载取地址运算符const函数、默认移动构造函数（C++11）、默认重载移动赋值操作符函数（C++11）

1. 进程与线程的区别
2. 进程是程序在某个数据集合上的一次运行活动；线程是进程中的一个执行路径（进程可以创建多个线程）。
3. 在支持线程的系统中，进程是系统资源分配的单位，线程是CPU调度的单位。
4. 进程之前不能共享资源、而线程共享所在进程的地址空间和其他资源。同时线程还有自己的栈和栈指针，程序计数器等寄存器。
5. 进程有自己独立的地址空间，而线程没有，线程必须依赖于进程而存在
6. 进程切换的开销较大，线程相对较小。