**知识点一**

**二叉树的基本性质**

性质1 二叉树第i层上的结点数目最多为2i-1(i≥1)。

证明：用数学归纳法证明：

　 归纳基础：i=1时，有2i-1=20=1。因为第1层上只有一个根结点，所以命题成立。

　归纳假设：假设对所有的j(1≤j<i)命题成立，即第j层上至多有2j-1个结点，证明j=i时命题亦成立。

　归纳步骤：根据归纳假设，第i-1层上至多有2i-2个结点。由于二叉树的每个结点至多有两个孩子，故第i层上的结点数至多是第i-1层上的最大结点数的2倍。即j=i时，该层上至多有2×2i-2=2i-1个结点，故命题成立。

性质2 深度为k的二叉树至多有2k-1个结点(k≥1)。

证明：在具有相同深度的二叉树中，仅当每一层都含有最大结点数时，其树中结点数最多。因此利用性质1可得，深度为k的二叉树的结点数至多为：

20+21+…+2k-1=2k-1

故命题正确。

性质3 在任意-棵二叉树中，若终端结点的个数为n0，度为2的结点数为n2，则no=n2+1。

证明：因为二叉树中所有结点的度数均不大于2，所以结点总数(记为n)应等于0度结点数、1度结点(记为n1)和2度结点数之和：

n=no+n1+n2 (式子1)

　 另一方面，1度结点有一个孩子，2度结点有两个孩子，故二叉树中孩子结点总数是：

nl+2n2

　　树中只有根结点不是任何结点的孩子，故二叉树中的结点总数又可表示为：

n=n1+2n2+1 (式子2)

　　由式子1和式子2得到：

no=n2+1

**知识点二**

字节对齐

字节对齐方式分为两种:

1, 默认方式:

32位操作系统 4字节对齐

64位操作系统 8字节对齐

2, 人为设置方式:

GUN 设置方法: GNU使用\_\_attribute\_\_选项来设置

使用伪指令#pragma pack (n)，C编译器将按照n个字节对齐

内存对齐有哪些原则呢？我总结了一下大致分为三条：

第一条：第一个成员的首地址为0

第二条：每个成员的首地址是自身大小的整数倍

第二条补充：以4字节对齐为例，如果自身大小大于4字节，都以4字节整数倍为基准对齐。

第三条：最后以结构总体对齐。

第三条补充：以4字节对齐为例，取结构体中最大成员类型倍数，如果超过4字节，都以4字节整数倍为基准对齐。（其中这一条还有个名字叫：“补齐”，补齐的目的就是多个结构变量挨着摆放的时候也满足对齐的要求。）

**VS2010 可以看出内存分布情况**

对于位域得分布，不太清楚，

**知识点三**

**C语言——指针的运算**

一 ：取地址运算“&”与取内容运算“\*”：

单目运算“&”是取操作对象的地址 ， “\*”是取指针指向的对象的内容 ， 两者互为逆运算

int x , \*p ;

p = &x ;

&(\*p) = p 表示指针 ； \*（&x）= x 表示变量x .

二 ：指针的算术运算 ：

指针的运算与其基类型有关 ， 一般的，如果p是一个指针 ，n是一个正整数 ，则对指针 p 进行 +（-）操作后的实际地址是 ：

p +(-)n\*sizeof(基类型)

复制代码

char \*p ;

int \*q ;

float \*tk ;

//假设当前地址为 ： p = 2110H , q = 2231H , tk = 2478H

p += 1 ; // p = p + 1 = 2111H

q += 4 ; // q = q + 4\*4 = 2239H

tk -= 3 ; // tk = tk -3\*8 = 246CH

三 ： 指针自加 ，自减运算 ：

指针的自加 ，自减运算也是地址运算

m = \* p ++ 等价于 m = \*(p ++)

取指针 p 当前所指变量的值赋给变量 m ; p 做加 1 运算 , 指向下一个目标变量 .

m = \* ++ p 等价于 m = \* ( ++ p)

p 做加 1 运算 , 指向下一个目标变量 ; 取指针 p 当前所指变量的值赋给变量 m .

m = (\*p)++ 与 m = ++(\*p)

前者是将指针 p 所指的变量的值赋给变量 m , 然后变量 \*p 自加 1 ;

后者是将指针 p 所指的变量 \*p 的值自加 1 后赋给变量 m .

四 ：指针间的减法运算 :

指向同一组类型相同数据的指针之间可以进行减法运算 , 相减的结果表示两指针间相距的数据的个数 .

五 ：指针的关系运算 ：

p , q 指向同一个数组 , p > q 表示p 是否指向 q 所指位置的后面 , 若成立 则 表达式的值为 非 0 , 否则为 0 .

六 ：指针的下标运算 ：

p[i] = \*(p + i) ;

Example

数组合指针:

int arr[10] = {0,1,2,3,4,5,6,7,8,9};

int \*p = arr;

sizeof(arr) == 40;

sizeof(p) == 4;

**知识点四**

小知识总结： 字符类型也是数据类型，只不过是 他typedef 了一下，字符在内存中的存储方法是利用ASCII 码中的数据存储。

int aa;

char cc[100];

strcpy(cc,"0123456789abcdefghijklmnopqrstuvwxyz");

memcpy(&aa,cc,sizeof(AA));

cout << aa <<endl;

strcpy 拷贝后，四个字节的字符串， “0123” 按照 ASCII 进行存储. 即位30,31,32,33; 0001 1110 0001 1111 0010 0000 0010 0001

**知识点五**

Extern 关键字: 有两个作用:

1, Extern 和 C 连用, extern "C" void fun(int a, int b);

对于 extern "C" void fun(int a, int b); 在C++环境中使用C, 为了能够找到C语言的函数和变量.

C++语言在编译的时候为了解决函数的多态问题，会将函数名和参数联合起来生成一个中间的函数名称，而C语言则不会，因此会造成链接时找不到对应函数的情况，此时C函数就需要用extern “C”进行链接指定，这告诉编译器，请保持我的名称，不要给我生成用于链接的中间函数名。

2, Extern 声明变量和函数

extern char \*a;

extern void fun(int a, int b);

对于Extern 声明的函数和变量只是一种声明修饰，表明该函数和变量的定义可能在别处。