**结构体的自引用：**

struct Node
{
int data;
struct Node n;
}; //这种做法是错误的
struct Node
{
int data;
struct Node\* next;
}; //这种做法是正确的

**typedef 类型**

typedef struct Node
{
int data;
struct Node\* next;
}node; //将Node重命名为node

**结构体内存对齐（计算大小）**

对齐规则：

1. 第一个成员在与结构体变量偏移量为0的地址处

2.其他成员变量要对齐到某个数字（对齐数）的整数倍的地址处

对齐数----编译器默认的一个对齐数 与 该成员大小的较小值

vs中默认的值为8

gcc没有默认对齐数

3.结构体总大小为最大对齐数（每个成员变量都有一个对齐数）的整数倍

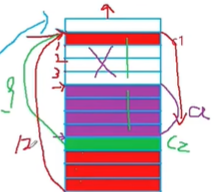
4.如果嵌套了结构体的情况。嵌套的结构体对齐到自己的最大对其数的整数倍处，结构体的整体大小就是所有最大对齐数（含嵌套结构体的对齐数）的整数倍

**为什么由内存对齐？**

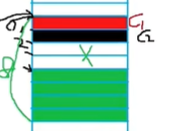
1.平台原因（移植原因）：不是所有的硬件平台都能访问任意地址上的任意数据；某些平台智能在某些地址处取特定类型的数据，否则抛出硬件异常

2.性能原因：数据结构（尤其是栈）应该尽可能的在自然边界上对其，原因在于，为了访问未对其的内存，处理器需要做两次内存访问，而对齐的内存访问仅需要一次访问

struct s1
{
char c1;
int a;
charc2;
}; //大小12



struct s2
{
char c1;
char c2;
int a;
}; //大小8



设置默认对齐数：

#pragma pack(4)

取消默认对齐数：

#pragma pack()

**结构体传参：**

void Init(struct S\* ps)
{
ps->a = 100;
ps->c = 'w';
ps->d = 3.14;
}
//传值
void print(struct s tmp)
{
printf("%d %d %lf\n", tmp.a, tmp.c, tmp.d);
}
//传址 比较好的方式
void print(const struct S\* ps)
{
printf("%d "%d "%lf\n", ps->a, ps->c, ps->d);
}
int main()
{
struct S s = {0};
Init(&s); //传入结构体的地址
return 0;
}