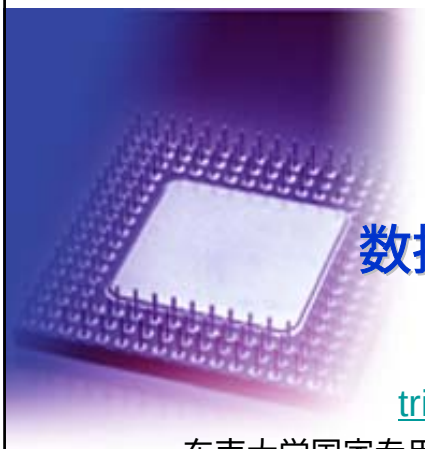
CNASIC



第 五 讲
数据结构与链表

凌 明

trio@seu.edu.cn

东南大学国家专用集成电路系统工程技术研究中心

CNASIC

目 录

CNASIC

1 结构体

结构是由若干（可不同类型的）数据项组合而成的复合数据对象，这些数据项称为结构的成分或成员。

(1) 字段

C 语言的结构还提供了一种定义字段的机制，使人在需要时能把几个结构成员压缩到一个基本数据类型成员里存放，这可以看作是一种数据压缩表示方式。

例16:

```
struct pack {
    unsigned a:2;
    unsigned b:8;
    unsigned c:6;
} pk1, pk2;
```

结构变量pk1或者pk2的三个成员将总共占用16位存储，其中a占用2 位，b占用8 位，c占用6 位。

CNASIC

(2)结构体内部的成员的对齐

在计算结构体长度（尤其是用sizeof）时，需要注意！

根据不同的编译器和处理器，结构体内部的成员有不同的对齐方式，这会引起结构体长度的不确定性。

例17：

```
#include <stdio.h>

struct a{  char a1;   char a2;  char a3; }A;
struct b{  short a2;  char a1; }B;
void main(void)

{
    printf("%d,%d,%d,%d", sizeof(char), sizeof(short), sizeof(A), sizeof(B));
}
```

在Turbo C 2.0中结果都是

1,2,3,3

在VC6.0中是

1,2,3,4

字节对齐的细节和编译器实现相关，但一般而言，满足三个准则：
1) 结构体变量的首地址能够被其最宽基本类型成员的大小所整除；
2) 结构体每个成员相对于结构首地址的偏移量（offset）都是成员大小的整数倍，
如有需要编译器会在成员之间加上填充字节（internal adding）；
3) 结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节（trailing padding）。

对于上面的准则，有几点需要说明：
1)结构体某个成员相对于结构体首地址的偏移量可以通过宏offsetof()来获得，这个宏也在stddef.h中定义，如下：

```
#define  offsetof(s,m)   (size_t)&(((s *)0)->m)
```

2) 基本类型是指前面提到的像char、short、int、float、double这样的内置数据类型，这里所说的“数据宽度”就是指其sizeof的大小。由于结构体的成员可以是复合类型，比如另外一个结构体，所以在寻找最宽基本类型成员时，应当包括复合类型成员的子成员，而不是把复合成员看成是一个整体。但在确定复合类型成员的偏移位置时则是将复合类型作为整体看待。

CNASIC

2 联合体

在一个结构（变量）里，结构的各成员顺序排列存储，每个成员都有自己独立的存储位置。联合变量的所有成员共享从同一片存储区。因此一个联合变量在每个时刻里只能保存它的某一个成员的值。

(1)联合变量的初始化

联合变量也在可以定义时直接进行初始化，但这个初始化只能对第一个成员做。例如下面的描述定义了一个联合变量，并进行了初始化：

例18：

```
union data
{
     char  n;
     float  f;
};
union data u1 = {3};                    //只有u1.n被初始化
```

3 枚举

枚举是一种用于定义一组命名常量的机制，以这种方式定义的常量一般称为枚举常量。

一个枚举说明不但引进了一组常量名，同时也为每个常量确定了一个整数值。缺省情况下其第一个常量自动给值0，随后的常量值顺序递增。

(1)给枚举常量指定特定值

与给变量指定初始值的形式类似。如果给某个枚举量指定了值，跟随其后的没有指定值的枚举常量也将跟着顺序递增取值，直到下一个有指定值的常量为止。

例如写出下面枚举说明：

```
enum color {RED = 1, GREEN, BLUE, WHITE = 11, GREY, BLACK= 15};
```

这时，RED、GREEN,、BLUE 的值将分别是1、2、3，WHITE、GREY的值将分别是11、12，而BLACK 的值是15。

(2)用枚举常量作为数组长度

例19：

```
typedef enum{WHITE, RED, BLUE, YELLOW, BLACK, COLOR_NUM
            }COLOR;
… …
float BallSize[COLOR_NUM];
```

上例中当颜色数量发生变化时，只需在枚举类型定义中加入或删去颜色。无需修改COLOR_NUM的定义。与大量使用#define相比既简洁又可靠。 如：

```
typedef enum{ WHITE, RED, BLUE,COLOR_NUM }COLOR;
```

CNASIC

4 sizeof的定义和使用

sizeof 是C/C++中的一个操作符（注意！不是函数！就像return一样）。

其作用就是返回一个对象或者类型所占的内存字节数。

sizeof有三种使用形式，如下：

```
1) sizeof(var);                 // sizeof( 变量 );
2) sizeof(type_name);           // sizeof( 类型 );
3) sizeof var;                  // sizeof 变量;
```

所以，

```
int    i;
sizeof(i);                      // ok
sizeof i;                       // ok
sizeof(int);                    // ok
sizeof int;                     // error
```

为求形式统一，不建议采用第3种写法，忘掉它吧！

CNASIC

数组的sizeof

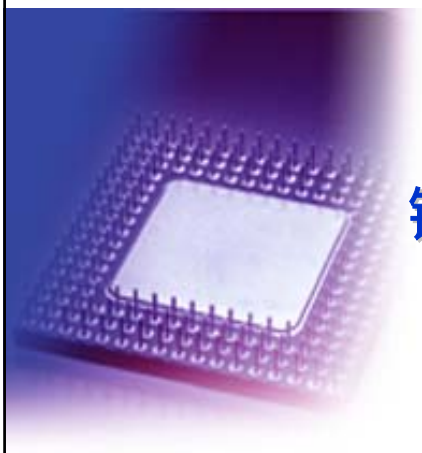
数组的sizeof值等于数组所占用的内存字节数，如：

例21：

```
char* ss = "0123456789";
sizeof(ss);         // 结果 4 , ss是指向字符串常量的字符指针
sizeof(*ss);        // 结果 1 ,*ss是第一个字符

char ss[] = "0123456789";
sizeof(ss);         // 结果 11 , 计算到‘\0’位置，因此是10＋1
sizeof(*ss);        // 结果 1 , *ss是第一个字符

char ss[100] = "0123456789";
sizeof(ss);         // 结果 100 , 表示在内存中的大小 100×1
strlen(ss);         // 结果 10 , strlen是到‘\0’为止之前的长度

int ss[100] = "0123456789";
sizeof(ss);         //结果200 , ss表示在内存中的大小 100×2
strlen(ss);         //错误,strlen的参数只能是char*且必须以‘\0’结尾
```

www.cnasic.com

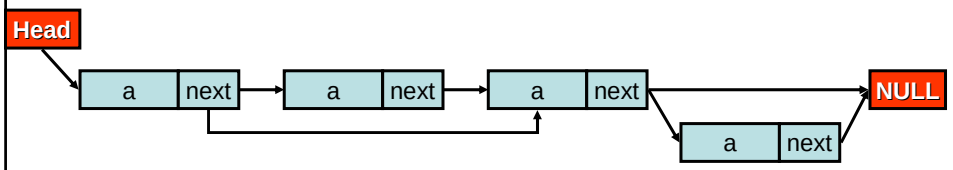
CNASIC



链 表

CNASIC

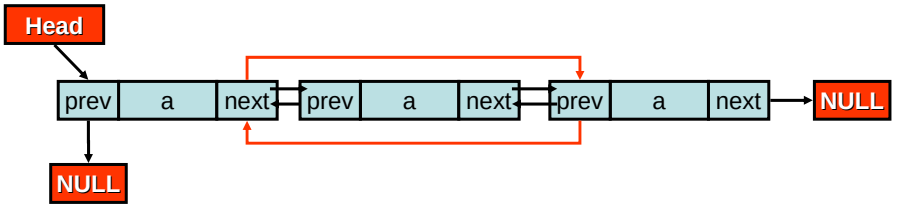
单向链表




```
Strut mylink
{
   int              a;
  struct mylink   *next;
} *Head, ptr;
```

CNASIC

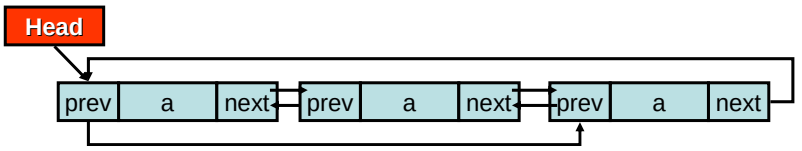
双向链表




```
Strut mylink
{
    int             a;
   struct mylink   *next;
   struct mylink   *prev;
} *Head, ptr;
```

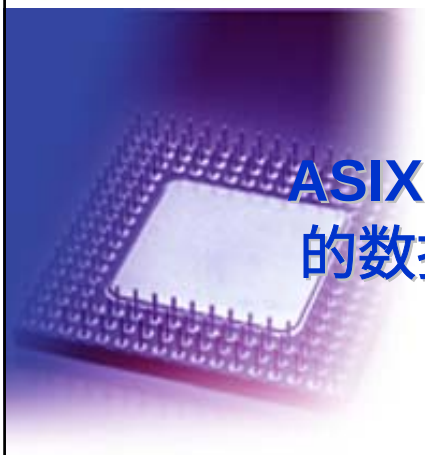

CNASIC

循环双向链表




```
Strut mylink
{
    int             a;
   struct mylink   *next;
   struct mylink   *prev;
} *Head, ptr;
```

CNASIC



ASIX Window窗口系统的数据结构与代码分析

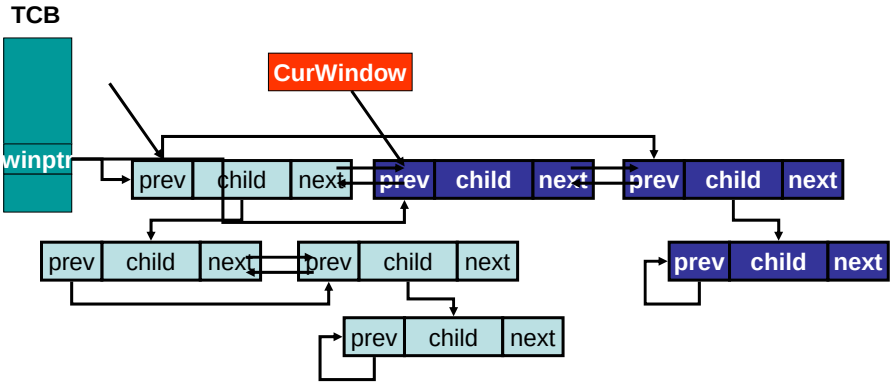
CNASIC

ASIX Window 说 明

- 窗口分为两类，但是在系统内部采用统一的数据结构
  - 窗体(Form)－也就是我们看到的窗口
  - 控件(Control)－按钮，下拉框，菜单，滚动条
- 某一个控件一定是一个窗体或则另一个控件的子窗口；而窗体只能作为父窗口存在；
- 一个任务可以有多个窗体，但是只有最前面的窗体（也就是所谓当前窗口）才能接受用户的输入消息；

CNASIC

ASIX Window的数据结构



CNASIC

ASIX Win 中窗口的数据结构

```
typedef struct asix_window
{
    struct asix_window      *prev;          //point to the previous sibling link
    struct asix_window      *next;          //point to the next sibling link
    struct asix_window      *child;         //point to the children link

    WNDCLASS                *wndclass;      //point to the relevent class entry

    unsigned int            task_id;    //the task id of this window
    unsigned int            wnd_id;     //the window id which is assigned by Creatwindow()
    unsigned int            parent_id; //the winId of this windows's parent

    unsigned int            status;

    unsigned short          x;
    unsigned short          y;
    unsigned short          width;
    unsigned short          hight;
    char                    *caption;   //the caption text string
    char                    *tag;       //the help tag string
    unsigned int            style;
    unsigned int            hmenu;

    void                    *ctrl_str;  //point to the control related struct
    void                    *exdata;
} ASIX_WINDOW;
```

CNASIC

窗口类的数据结构

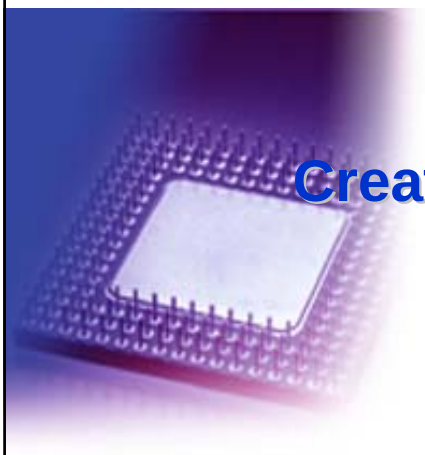
```
typedef struct window_class
{

    unsigned char                wndclass_id;

    unsigned int      (*create)(char *caption, U32 style, U16 x, U16 y,\
                       U16 width, U16 hight, U32 wndid, U32 menu, void **ctrl_str, void *exdata);

    unsigned int      (*destroy)(void *ctrl_str);
    unsigned int      (*msg_proc)( U32 win_id, U16 asix_msg, U32 lparam, void *data, U16 wparam,
    void *reserved);
    unsigned int      (*msg_trans)(void *ctrl_str, U16 msg_type, U32 areald, P_U16 data, U32 size,
    void *trans_msg);

    unsigned int (*repaint)(void *ctrl_str, U32 lparam);
    unsigned int      (*move)(void *ctrl_str, U16 x, U16 y, U16 width, U16 hight, void *reserved);
    unsigned int      (*enable)(void *ctrl_str, U8 enable);
    unsigned int      (*caption)(void *ctrl_str, char *caption, void *exdata);
    unsigned int      (*information)(void *ctrl_str, struct asix_window *wndinfo);


} WNDCLASS;
```


CNASIC

创建窗口CreateWindow函数

- 参数检查
- 分配窗口数据结构
- 如果创建的是Form，则老窗口压栈
- 窗口数据结构的填写
- 调用各窗口类的具体创建函数
- 将窗口数据结构加入到窗口链表中
- 返回窗口句柄

CNASIC



CreateWindow函数代码阅读

CNASIC

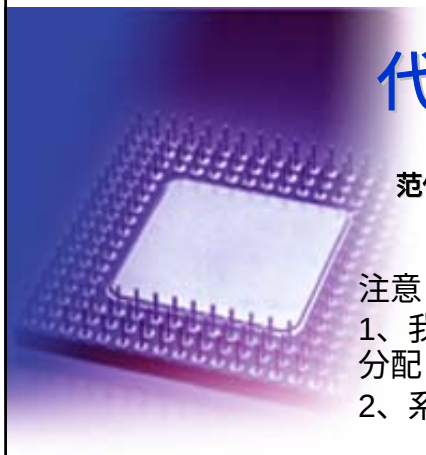
删除窗口DestroyWindow函数

- 参数检查
- 如果被删窗口的子窗口不为空，则递归调用本函数，直到子窗口为空。
- 调用窗口类的具体删除函数，
- 将被删窗口的数据结构从窗口链表中取下
- 背景重绘
- 释放被删窗口的数据结构

CNASIC

DestroyWindow函数代码阅读

CNASIC



代码的调试

范例代码已在VC＋＋编译通过，Just Try It！

注意：

1、我们在项目中包含了上次课的动态内存分配

2、系统堆的大小在malloc.h文件中定义