

# 《C语言程序设计》

丁盟  
C语言课程组

```
#include <iostream.h>
#include "bignumb.h"

void main(void){
    big_number a(50);
    long five=5;
    double pi=3.1415926;
    cout << "\n\n";
    cin >> a;
    cout << "b=";
    cin >> b;
    cout
    if (a<b)
        cout << "\na<b";
    if (a>b)
        cout << "\na>b";
    if (a==b)
        cout << "\na=b";
    cout << "\na+b=" << a+b;

    f_in1.unsetf(rskipws);
    getline(f_in1,s);
    try
    {
        s.erase(0,s.find("]",1));
        s.erase(0,(s.find("]",1)+10));
        str= s.substr(0,s.find("]",1));
    }
    return 1;
}

size=str.compare(ip);
if (size==0)
{
    try{
        str=s.substr((s.find("]",1))
```

# 上一讲知识复习

- ◆理解“函数”与“面向过程的编程”的关系。
- ◆掌握函数原型声明与原型定义的方法。
- ◆深入理解参数传递，尤其是“传递地址”的情况。
- ◆掌握嵌套调用、递归函数的分析方法。

# 本讲教学目标

- ◆掌握结构体数据类型的定义语法。
- ◆掌握三种结构体变量定义语法。
- ◆掌握结构体对象成员的访问方式。
- ◆掌握结构体对象数组的定义、使用方法。
- ◆掌握共用体类型定义、变量的使用。
- ◆掌握枚举类型变量的使用。

# 本讲授课程内容



结构体



结构体的内存分配

共用体

枚举

# 结构体

- ❖ 编写学生管理系统，程序需要处理100个学生的数据，每个学生需要存储学号、姓名、性别、入学时间、C语言成绩、数学成绩、英语成绩、音乐成绩。
  - ◆ 打印数学成绩和英语成绩都在90分以上的学生的信息
  - ◆ 按照数学成绩对这些学生进行排序
  - ◆ 学号为100的学生转学走了，如何删除它的信息。
  - ◆ 如何新加一个学生，并且添加完成后学生信息按数学成绩有序

# 结构体

- ❖ 计算机如何表示一个学生的信息？
- ❖ 如何表示多个学生的信息？
- ❖ 如何用计算机处理如下表格：

**某学校学生成绩管理表**

学号	姓名	性别	入学时间	C语言	英 语	数 学	音 乐
1	令狐冲	男	1999	90	83	72	82
2	林平之	男	1999	78	92	88	78
3	岳灵珊	女	1999	89	72	98	66
4	任莹莹	女	1999	78	95	87	90
5	... ..						
6	... ..						



# 结构体

```
// 每个学生的学号用数组的下标表示
char studentId[100][10];
char studentName[100][10];
char studentSex[100][2];
int timeOfEnter[100]; // 入学时间用int表示
int scoreC[100];      // C语言的成绩
int scoreEnglish[100]; // 英语课的成绩
int scoreMath[100];   // 数学课的成绩
int scoreMusic[100];  // 音乐课的成绩
```

# 结构体

1	令狐冲	男	1999
2	林平之	男	1999
3	岳灵珊	女	1999
4	任盈盈	女	1999
.....	.....	.....	.....
90	83	72	82
78	92	88	78
89	72	98	66
78	95	87	90
.....	.....	.....	.....

- ❖ 数组个数太多
- ❖ 结构显得比较零散，不容易管理
- ❖ 分配内存不集中，寻址效率不高



# 结构体

## 采用一个整体描述学生相关信息

1	令狐冲	男	1999	90	83	72	82
2	林平之	男	1999	78	92	88	78
3	岳灵珊	女	1999	89	72	98	66
4	任莹莹	女	1999	78	95	87	90

# 结构体

```
struct student
{
    char studentID[10]; // 学生的序号
    char studentName[10]; // 学生的姓名
    char studentSex[4]; // 学生的性别
    int timeOfEnter; // 学生的入学时间
    int scoreC; // 学生的C语言成绩
    int scoreEnglish; // 学生的英语成绩
    int scoreMath; // 学生的数学成绩
    int scoreMusic; // 学生的音乐成绩
};
```

- ❖ `struct student` 是一个自定义的数据类型
- ❖ `struct student s1` ; 是一个结构体变量
- ❖ `struct student stu[100]`; 是一个结构体数组

# 结构体

	普通变量	结构体变量
数据类型	固定（死板）	自定义（灵活）
定义方式	Datatype varName	struct student stu1;
访问方式	通过变量名访问	整体->部分 { _' >
存储方式	整体	部分->整体，有内存漏洞
数组	Datatype arrName[5]	struct student stu[50];
指针	Datatype * pointer	普通结构体指针 结构体内的自引用指针（关系）

# 结构体

- ❖ 结构体是一种**自定义数据类型**，结构体变量代表的结构体类型的数据对象作为一个**整体存储**在内存中。
- ❖ 结构体类型定义的基本语法形式为：

```
struct 结构体名  
{  
    类型关键字  成员名1;  
    类型关键字  成员名2;  
    .....  
    类型关键字  成员名n;  
};
```

构成结构体的变量称结构体**成员 (member)** 也称**域 (field)**

**struct 结构体名**定义了**新的数据类型**，是一个模板，不分配内存，用于生成结构体变量

# 结构体

```
struct student
{
    char studentID[10]; // 学生的序号
    char studentName[10]; // 学生的姓名
    char studentSex[4]; // 学生的性别
    int timeOfEnter; // 学生的入学时间
    int scoreC; // 学生的C语言成绩
    int scoreEnglish; // 学生的英语成绩
    int scoreMath; // 学生的数学成绩
    int scoreMusic; // 学生的音乐成绩
};
```

❖ **struct student**是一个自定义的数据类型

# 结构体

❖ 请用结构体描述以下的学生信息

学号	姓名	性别	年龄	学分	出生年月			住址
					年	月	日	

// 出生日期的结构体

```
struct DATE
```

```
{
```

```
    int    year;
```

```
    int    month;
```

```
    int    day;
```

```
};
```

结构体定义  
可以嵌套

```
struct student
```

```
{
```

```
    char    ID[10];
```

```
    char    name[20];
```

```
    char    sex;
```

```
    int     age;
```

```
    float   score;
```

```
    struct DATE birthday;
```


```
    char    addr[30];
```

```
};
```

# 结构体

❖ 分析下面结构体定义语句，找错。

```
struct Complex
{
    double real;
    double imag;
};
struct Node
{
    int data_id;
    struct Complex data;
    struct Node nextNode;
    struct Node * next;
};
```



◆ 成员变量的类型**不能和本结构体类型相同，即不能递归定义。**

◆ 结构体的成员变量可以是**本结构体类型的指针变量。**  
这种结构也叫**自引用结构。**



# 结构体

❖ 什么不允许出现在结构体成员中？

➤ 本身结构体类型的成员变量

➤ 函数

➤ 静态变量

➤ 如果在结构体类型定义时使用了typedef，则使用别名创建的本身结构体类型指针也不允许出现在结构体成员当中

# 结构体

## ❖在定义结构体的同时定义结构体变量

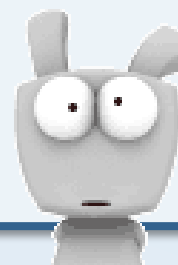
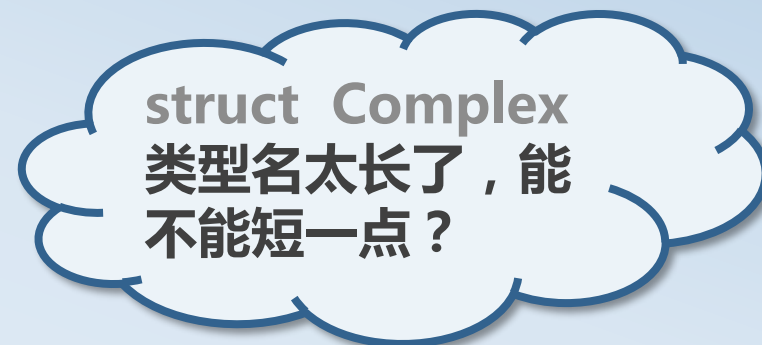
```
struct Complex  
{  
    double real;  
    double imag;  
}op1,op2;
```

```
struct  
{  
    double real;  
    double imag;  
}op3;
```

## ❖先定义结构体类型后定义结构体变量

```
struct Complex  
{  
    double real;  
    double imag;  
};
```

```
struct Complex op1,op2;
```



# 结构体

❖ 请用typedef简化struct Complex变量的定义形式

```
typedef struct Complex Complex;
```

```
Complex stu1,stu2;
```

❖ 更通用的做法：

```
typedef struct Complex  
{  
    double real;  
    double imag;  
} Complex;
```



```
typedef struct student Complex;
```

Complex就是struct Complex  
的同义词

```
Complex stu1,stu2;
```

# 结构体

- ❖ 在定义结构体变量的时候可以按照成员的顺序和类型对成员整体赋初值。

```
struct Complex c1 = {1.4, 2.5};
```

- ❖ 或者在定义结构体变量后，单独给成员变量赋值。

```
struct Complex c1 ;  
c1.real = 1.4;  
c1.imag = 2.5;
```

- ❖ 同类型结构体变量可相互赋值(不涉及到指针成员)

```
struct Complex c1, c2 ;  
c1.real = 1.4;  
c1.imag = 2.5;  
c2 = c1;
```

# 结构体

- ❖ 即使两种结构体类型的成员列表完全相同，他们对应的变量也不可以使用 = 进行赋值。

```
struct a
{
    char c;
    int n;
} A;
```

```
struct b
{
    char c;
    int n;
} B;
```

```
A = B;
```



# 结构体

❖根据结构体的类型，可以定义相应的结构体指针变量.

```
struct Complex c1 = {1.4, 2.5};  
struct Complex * p = &c1;
```

总结：

1.结构体也是一种数据类型，所以也有相应的指针类型.

2.结构体指针和其他类型的指针性质基本相同，因为它也是指针.

# 结构体

- ❖ 若是普通**结构体变量**，则访问其成员的方式为：
  - 结构体变量名.成员名
- ❖ 若是通过**结构体指针**访问结构体对象，则形式为：
  - 结构体指针名->成员名
  - ( \*结构体指针名 ).成员名

❖ 例：

```
struct Complex c1 ;  
struct Complex * c2;  
c2 = (struct Complex *)malloc(sizeof(struct Complex ));  
  
c1.imag = 2.1;  
c1.real = 3.4;  
  
c2 -> real = 4.5;  
c2 -> imag = 5.6;  
(*c2).real = 6.7;  
(*c2).imag = 5.9;
```



# 结构体

- ❖ 可以将结构作为参数传递给函数，也可以定义返回结构值的函数。
- ❖ 结构作为函数参数有三种不同方法：
  - ◆ 将**结构成员的值**传递给函数处理。
  - ◆ 将**整个结构作为参数值**传递给函数。
  - ◆ 将**结构指针变量**做函数的参数。

把结构作为整体来处理，  
但作用方式和效果不同。

# 结构体

## ❖ 结构体也是一种数据类型

- ◆ 可以定义该类型的变量、数组、指针.....
- ◆ 可以做函数的参数类型和返回值类型
- ◆ 它的成员可以是任意类型
  - 基本类型、数组、指针、结构体、共用体.....

## ❖ struct 类型的变量

- ◆ 两个结构体变量之间可以相互赋值
- ◆ 可以做函数的形参
- ◆ 可以通过.或者 -> 访问成员
- ◆ 可以取地址&
- ◆ 不可能直接参与算术和比较运算

## ❖ 面向对象和数据库是struct的思想的发展

# 本讲授课程内容



结构体

结构体的内存分配



共用体

枚举

# 结构体的内存分配

```
struct
{
    char a;
    int b;
    double c;
} A;
```

`sizeof(A) = ?`

`sizeof(A.a) = 1`

`sizeof(A.b) = 4`

`sizeof(A.c) = 8`

`sizeof(A) = 13 ?`

`sizeof(A) = 16`

- ❖ 结构所占的实际空间一般是按照编译器默认对齐方式（对齐系数）来进行内存分配的。

# 结构体的内存分配

## 成员的对齐步骤：

- ❖ 1 判断结构体成员的大小是否小于等于对齐系数，小于等于执行到2，否则到3；
- ❖ 2 判断预存放结构体成员的首地址相对于结构体首地址的偏移量是否是本成员大小的整数倍，若是则存放本成员，否则到4；
- ❖ 3 判断预存放结构体成员的首地址相对于结构体首地址的偏移量是否是对齐系数的整数倍，若是则存放本成员，否则到4；
- ❖ 4 在本成员和上一个成员之间填充一定的空字节以达到整数倍的要求再存放本成员。

# 结构体的内存分配

## 整体的对齐步骤：

- ❖ 1 判断结构体成员中最大成员的大小是否小于等于对齐系数，小于等于执行到2，否则到3；
- ❖ 2 判断结构体各成员成员对齐后长度之和是否是最大成员大小的整数倍，是则不进行整体对齐，否则到4；
- ❖ 3 判断结构体各成员成员对齐后长度之后是否是对齐系数的整数倍，是则不进行整体对齐，否则到4；
- ❖ 4 在最后一个成员后填充一定的空字节以达到整数倍的要求，整体对齐完成。

# 结构体的内存分配

❖使用伪指令 `#pragma pack (n)` 设置对齐系数

$n = (1, 2, 4, 8, 16)$

例：`#pragma pack (4)`

C编译器将按照4个字节进行对齐。

**\*当n取值非法时伪指令无效。**



# 结构体的内存分配

- ❖ 当结构体中某成员为数组时，则在处理结构体的对齐时，该成员数组在求整数倍关系时使用数组类型的大小。
- ❖ 当结构体中某成员为结构体时，则在处理结构体的对齐时，该成员结构体在求整数倍关系时使用该成员结构体中最大成员的大小。

# 结构体的内存分配

- ❖ 对齐的好处：便于成员寻址
- ❖ 对齐的坏处：内存浪费（存在空字节）
- ❖ 注：不同的编译器、平台，对齐方式会有变化

# 本讲授课程内容



结构体

结构体的内存分配

共用体

枚举



# 共用体

❖ 共同体的类型的定义与结构体的完全相同，只需要将union 替换struct 即可。

❖ 典型的形式：

```
union 类型名
{
    成员声明列表；
};
```

❖ 共同体变量的声明：

- ◆ 在定义共同体类型的时候声明该共同体的变量
- ◆ 在定义共同体类型的时候声明该共同体的变量
- ◆ 像声明 int 型变量一样声明共同体变量

# 共用体

## 结构体

- 成员**各自占有**内存空间
- 大小为各成员占用空间的和+填充
- 定义时可以初始化
- 可以做函数参数和返回值
- 可定义结构体指针和数组

## 共同体

- 成员**共同占有**内存空间
- 大小为成员变量中占用空间最大的
- 定义时可以初始化第一个成员
- 可以做函数参数和返回值
- 可定义共同体指针和数组

# 共用体

- ❖ 对共同体对象的成员访问与对结构体对象的成员访问方法相同。
- ❖ 共同体对象处于**栈**区，则访问其成员的方式为：
  - ◆ **对象名.成员名**
- ❖ 共同体对象处于**堆**区，则只能通过指向该对象的指针访问它的成员，形式为：
  - ◆ **对象指针名->成员名**
  - ◆ **( \*对象指针名 ).成员名**

# 共用体

- ❖ 共同体的所有成员占据**同一块内存**。
- ❖ 共同体变量和结构体变量的异同。
- ❖ 最后**接受修改的成员是起作用的成员**，其他成员的值失去了作用。
- ❖ **共同体类型的定义可以出现在结构体类型定义中，反之结构体类型的定义也可以出现在共同体类型的定义中。**
- ❖ typedef也可**简化共同体的定义**。



# 本讲授课程内容



结构体

结构体的内存分配

共用体

枚举



# 枚举

❖所谓“枚举”是指将变量的值一一列举出来，变量的值只限于列举出来的值的范围。

❖定义：

**enum 枚举类型名 { 有限集合元素列表 };**

❖例：

```
enum weekday {sun,mon,tues,wednes,thurs,fri,satur};
```

```
enum weekday {sun = 1,mon,tues,wednes,thurs,fri,satur};
```

# 枚举

```
enum 枚举名  
{  
    成员表列 ;  
};  
enum 枚举名 变量表列 ;
```

```
enum 枚举名  
{  
    成员表列 ;  
}变量表列 ;
```

```
enum  
{  
    成员表列 ;  
}变量表列 ;
```

❖例：

```
enum color { red, white };  
enum color myColor;
```

```
enum color  
{ red, white } myColor;
```

```
enum  
{ red, white } myColor;
```

# 枚举

❖ 对枚举变量赋值语句的形式为：



◆ 枚举型变量名 = 有限集合中的某个元素;

```
enum weekday  
{ sun, mon, tue, wed, thu, fri, sat } weekday, weekday;  
weekday = mon;  
weekday = sun;
```

❖ 注意

❖ **枚举成员都是常量**，不能赋值

❖ 枚举常量和整形常量类型不同，不能直接赋值

```
sun = 1;   
weekday = 2; 
```

```
weekday = (enum weekday)2 
```

# 枚举

```
#include <stdio.h>
int main(void)
{
    enum Days{Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday};
    enum Days theDay;
    int j = 0;
    printf("请输入今天是星期几: (0 to 6)\n");
    scanf("%d",&j);
    theDay = (enum Days)(j);
    if(Sunday == theDay || Saturday == theDay)
    {
        printf("It is the weekend.\n");
    }
    else
    {
        printf("It is a working day.\n");
    }
    return 0;
}
```

# 枚举

- ❖可以在**定义**枚举类型的同时，**声明**枚举型变量。
- ❖枚举元素本身由系统赋予一个表示**序号的数值**,从0开始顺序定义为0，1...
- ❖可以**自由的设置**枚举元素的数字序号
- ❖可直接把枚举元素赋值给相应的枚举变量
- ❖若要把枚举元素的数字序号赋予枚举变量则**必须用强制类型**转换.

Thank You !