结构体、共用体和枚举类型

中山大学计算机学院

讲课人：万海

目录

CONTENTS

结构体 - 聚合数据类型

现在世界有各种各样的信息和数据，基本数据类型包括：

- 整数（long long, long, int, short, char, _Bool等）
- 浮点数（float, double,long double等）

聚合数据类型

- 数组，相同类型的数据连续排放
- 结构体、联合体，多个类型数据聚合在一起表示某类事物。

例如：一个学生可能由学号、姓名、年龄、身高、成绩等数据表示；复数由实部数据和虚部数据构成；一个平面上的点由 x 和 y 构成；……

结构体 - 由用户定义的类型

结构体（struct）是由一序列的数据成员组成的类型。用户可自由组合定义。

```
struct STUDENT {
    char id[6];
    char name[10];
    int age;
    struct ADDRESS *home;
} john;

struct STUDENT anna;

struct ADDRESS {
    char area_id[6];
    … …
};
```

定义结构体

成员声明列表

声明变量

声明变量

定义结构体

变量

类似 int a 的 a

使用关键字 **struct** 完成类型定义等操作:

1. 定义结构体类型与变量。"**struct** *identifier*{ **成员变量**表 }" 完成结构体定义。然后直接定义该类型的**实例变量**，注意 "**;**" 结束。
2. 仅定义结构体类型变量。注意：c语言中，**结构体类型**是 "struct *STUDENT*" 不是 " *STUDENT*"，与C++不同。
3. 仅定义结构体类型。以后再申明它的**实例变量。注意：**定义格式 "struct *identifier*{ *members*_list } *vars*_list **;**"，**"}" 后分号不能省。**
4. 习惯上 struct **标识符 用大写风格**

结构体 -指针与数组

结构体（struct）类型，可定义指向该类型变量的指针，或定义由该类型元素组成的数组

```
struct CLASS {
    char id[10];
    int  members;  //结构体中不支持 VLA
    struct STUDENT *students;
};

struct CLASS c_class[6];
```

定义结构体

声明指向学生或学生数组的指针

指针在哪里定义都可以

定义结构体类型的数组

结构体（struct）类型是派生类型，它可继续派生出数组、指针、甚至结构体。即结构体类型中可以包含结构体成员。

结构体 - 嵌套与多层嵌套

结构体嵌套是指结构体类型中包含结构体成员。

```
struct STUDENT {
    char id[6];
    char name[10];
    int age;
    struct ADDRESS {
        char area_id[6];
        …
        struct STUDENT receiver;
    } home;
} john;
```

声明结构体

嵌套结构体ADDRESS

多层嵌套，形成循环嵌套，编译错误

结构体嵌套体中不能出现自身，编译提示 `struct STUDENT` 定义未完成。为了保证不发生自身嵌套，A 结构体中如果包含 B 结构体类型（非指针）成员，则 B 结构体必须在 A 之前定义。

结构体 -定义匿名类型

结构体可以是**匿名类型**。

```
struct {
    char id[6];
    char name[10];
    int age;
} john;
```

匿名结构体

声明变量

程序员不知道结构体（struct）类型的名字，除了定义时申明的变量、数组等外，就无法再次定义该类型及其派生类型的变量。那有什么用呢？嵌套里可以用

结构体 - 练习

请完成程序中指定的两个任务。

```
/*struct definition*/
#include<stdio.h>

struct STUDENT {
    char id[6];
    char name[10];
    int age;
    //任务1：去除 * 号 ，按 F12
    struct ADDRESS *home;
} john;

…
```

typedef 关键字 - 类型别名

C语言很多派生类型定义很长，不利于阅读、理解；不方便识别、定义变量；也可能存在与现代计算机语言习惯不一致等。例如：

- 结构体类型 struct STUDENT
- 数组 int a[M][N]
- 指针 int (*p)[N]; int * (*p)(int,int)

关键字 typedef 可以为类型起一个新的别名。typedef 的用法一般为：

$$typedef\ \ oldName\ \ newTypeName;$$

例如：

typedef _Bool bool;
typedef unsigned int size_tt;

typedef 关键字 - 定义结构体

给**匿名结构体**赋予**别名**。简化结构体名称，与C++等定义变量风格一致。

```
typedef struct {
    char id[6];
    char name[10];
    int age;
} Student;

Student anna,john,*person;
```

匿名结构体

类型别名 Student，不是变量。使用 **Camel 风格**

建议这样声明变量

(不用 struct 了)

程序员事实上并不关心结构体类型（struct XXX）的名字，直接用类型别名定义变量或派生其他类型如数组等，可改善程序的可读性，程序也更 modern。

typedef 关键字 - 定义数组或指针

如果指针或数组的定义要在多处使用或有特别含义，使用类型别名可简化程序或提高程序可读性。

```
typedef int, MATRIX_3[3][3]

MATRIX_3 a,b={{1},{4,5},{7,8,9}};

typedef int, (*CMP_F)(int,int);
typedef int, (*PTR_3)[3];

void mysort(int *base, int n, CMP_F f);
PTR_3 p1,p2,p3,*p;
```

申明数组类型别名，习惯**大写风格**

申明函数指针类型别名

申明指针类型别名

上述案例中，别名使得程序更易于阅读，特别是复杂的指针强制转类型时更具价值。

结构体操作 - 实例变量初始化

结构体的实例变量采用初始值列表依次初始化每个成员，未初始化成员则填零值，对嵌套结构体成员同样使用初始值列表初始化。

```
typedef struct {
    int year, month, day;
} Date;

typedef struct {
    char id[6];
    char name[10];
    Date birthday;
    int age;
} Student;

Student anna = {"0101","anna",
    {1995,10,1}};
```

定义结构体

嵌套结构体成员

初始化列表

结构体操作 - &*=运算符

结构体的实例变量，成员变量都可以使用 取地址、解引用和赋值运算。其中**实例变量赋值**一次可复制所有成员的值，包括其中的数组类型成员

```
Student anna,john,*person;
```
← 定义结构体的实例变量和指针

```
person = &anna;
```
← 取anna地址，赋值给指针

```
john = *person;
```
← **解引用得到变量/对象** anna 并将所有成员值赋予john对应成员

结构体实例是变量，而数组是地址。变量可以取地址。

结构体第一个成员的地址值，一定等于该成员所属实例变量的地址值。

结构体操作 - 成员运算符

结构体的实例变量使用原点成员运算符（.）和指针成员运算符(->) 访问结构体成员变量。

```
strcpy(anna.id,"0101");
strcpy(person->id,"0101");

anna.birthday.year = 1995;
person-> birthday.year = 1995;
(*person).birthday.year = 1995;
```

实例变量用原点成员运算符

指针变量用指针成员运算符

结构体嵌套可接连使用成员运算

注意：原点运算符优先级高于解引用，括号不能省

结构体 字符串

注意：(void*)&anna == (void*)anna.id 是地址相等，&anna 和 anna.id 是**不同类型指针**。例如

scanf("%s", anna.id); 不能写成 scanf("%s",&anna);

结构体操作 - 作为函数参数或返回值

结构体类型作为函数参数，如果是传值会导致实参赋值初始化形参，当结构体成员很多时会产生性能问题。**原则上采用传引用**，除非结构体成员明确且较少。

```
Student* StuCpy(Student st);
```

← 不好习惯，隐含实参复制操作

替换为：

```
Student* StuCpy(const Student *st_ptr);
```

← 用常指针保护实参对象，是否复制交给函数编程人员

```
Student* StuCpy(Student *st_ptr);
```

← 正常传引用

(传一个地址过去)

const 限定返回指针和形式参数是传引用必须考虑的重要问题。

结构体操作 - 案例研究

实现有理数及其运算，如加法

```
/*rational*/
#include<stdio.h>
#include<assert.h>

typedef struct RATIONAL {
    int x,y;
} Rational;

int gcd(int x, int y);
//注：本案例合适用传值和返回值
Rational* ADD(Rational *r1,const Rational *r2);
//请完成乘法，并测试函数正确性
Rational* MULTI(Rational *r1,const Rational *r2);
… …
```

结构体操作 - 动态内存

结构体类型数据多数需要动态管理且生命周期长。与指针配合，使用 malloc() 申请内存保存结构体对象/变量，在不需要时使用 free() 释放内存 。

```
CLASS_PTR createClass(char *name, int cnt) {
    assert(name && cnt);

    CLASS_PTR cls = malloc(sizeof(C_Class));
    strcpy(cls->id, name);
    cls->members = cnt;
    cls->students = malloc(cnt * sizeof(Student));
    memset(cls->students, 0, cnt * sizeof(Student));
    return cls;
}
```

```
void freeClass(CLASS_PTR p) {
    assert(p);

    free(p->students);
    free(p);
}
```

结构体实例创建、销毁配对

先分配动态结构体，释放时最后

再分配成员的空间；释放时先成员

使用**函数创建和释放**初始化逻辑复杂的结构体，是良好的编程习惯。

指针作为函数的返回值 - 小结

指针作为函数的返回值，必须保证所指对象没有被释放，因此**不能返回指向函数中自动变量的指针**。具体的，以下返回的指针都是安全的：

1. 返回指向函数中静态变量的指针；
2. 返回指向函数中定义的字符串字面量的指针；
3. 返回指向全局变量或函数的指针；
4. 返回函数参数引用的对象及其关联对象的指针，如指向传入数组的元素，或指向传入结构体指针的成员；
5. 返回动态申请内存的指针

结构体操作 - 综合案例

实现班级、学生结构体的定义、构建、输入、输出与释放

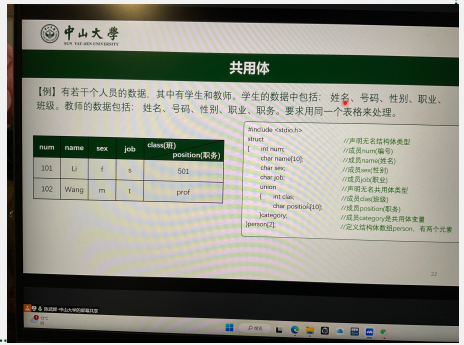
```
/*myclass*/
… …
void readStudents(CLASS_PTR p) {
    printf("id name <enter> q exit input\n");
    for(int i=0;i<p->members;i++) {
        //利用指针操作结构体对象成员
        STUDENT_PTR st_ptr = &(p->students[i]);
        scanf("%s",st_ptr->id);
        if (tolower(*(st_ptr->id))== 'q') {
            *(st_ptr->id) = 0;
            return;
        }
        scanf("%s",st_ptr->name);
    }
}
… …
```

程序要点：

1. 结构体类型定义建议放在单独的头文件中。
2. 结构体对象赋值是结构体复制，操作结构体元素用指针更方便。
3. 注意各种语言元素的风格
   - 哪些是大写风格
   - 哪些是驼峰风格
   - 哪些是小写风格
4. 空间的分配与释放
5. 简单结构体输入
6. 结构体数组内容表格输出

共用体

与结构体一样，共用体(union)也是一种派生数据类型。共用体的成员共享同一个存储空间。共用体的成员可以是任意数据类型。



共用体

声明一个共用体与声明一个结构体的格式相同，只不过是将关键词 struct 改为 union。右边这个共用体定义表示number是一个共用体类型，它的成员是 整型变量x **或者** 双精度型变量y

```
union number {
    int x;
    double y;
};
```

共用体

可对共用体执行的操作有三种：实例变量的复制；&运算符取共用体实例变量地址，**注意：实例变量地址值和所有成员变量地址值都相等**；用原点成员运算符或指针运算符访问共用体的成员。

不能用运算符==或!=来比较两个共用体。

枚举类型

C语言提供的最后一个用户自定义的数据类型，称为枚举类型(enumeration)。通过关键字enum引入的枚举类型，是一个用标识符表示的整型枚举常量的集合。除非专门定义，枚举类型中枚举值都是从0开始并且依次递增1的。

枚举类型

```
enum months {JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};
```

例如上面这条语句创建了一个新的数据类型enum months,其中标识符的值被相应地置成从0到11的整数。下面则改为1到12来为月份计数。

```
enum months {JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};
```

枚举类型

在一个枚举类型中出现的标识符必须是互不相同的。可以在定义枚举类型时，通过给标识符赋值来显式地给枚举常量赋值。一个枚举类型中的多个成员可以拥有相同的常量值。

枚举类型

```c
#include <stdio.h>

enum months {JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};

int main(){
    enum months m;
    for (m = JAN; m <= DEC; m++){
        printf("%d\n", m);
    }
    return 0;
}
```

中山大學
SUN YAT-SEN UNIVERSITY

谢 谢

中山大学计算机学院

编制人：课题组