知识点总结

· 在C语言中，结构体（struct）是一种用户自定义的数据类型，它允许将一系列相同类型或不同类型的数据组合在一起，支持嵌套、数组、指针操作。

· 定义一个结构体的语法格式：

```
struct 结构体标签 {
    数据类型 成员1;
    数据类型 成员2;
    ... // 可以有多个不同类型的成员
};
```

例：定义一个表示学生信息的结构体。

```
struct Student {
    char name[20];
    int age;
    float score;
};
```

· 定义结构体变量：

1）定义结构体类型的同时定义结构体变量。

例：

```
struct Student {
    char name[20];
    int age;
    float score;
} stu;
```

2）先定义结构体类型，再定义结构体变量。

例：

```
struct Student {
    char name[20];
    int age;
    float score;
};
struct Student stu;
```

· 初始化结构体变量：

1）定义结构体变量的同时，进行初始化。

例：`struct Student stu = {"Alice", 20, 85.5};`

注意：要按照结构体定义中成员的顺序来赋值。

· 可以使用成员访问符（. 或 ->）访问结构体变量中的成员：

①直接操作结构体变量本身：结构体变量名.成员名

其中，结构体变量名也可以是已定义的结构体数组的数组元素。

②使用指向结构体变量的指针：指针名->成员名

③先对指向结构体变量的指针进行解引用，再访问：(*指针名).成员名

例：

```c
// 直接操作结构体变量
struct Student stu = {"Alice", 20, 85.5};
printf("Student name: %s, age: %d, score: %.2f\n", stu.name, stu.age, stu.score);
```

例：

```c
// 使用指向结构体变量的指针
struct Student stu = {"Alice", 20, 85.5};
struct Student* p = &stu;
printf("Student name: %s, age: %d, score: %.2f\n", p->name, p->age, p->score);
```

2）定义结构体变量后，再逐个赋值。

例：定义了一个结构体变量stu，以及指向stu的指针p。

```
struct Student stu;
struct Student* p = &stu;
// 使用结构体变量操作
stu.age = 21;
strcpy(stu.name, "Bob");
stu.score = 90.0;
// 使用指针操作
p->age = 21;
strcpy(p->name, "Bob");
p->score = 90.0;
```

注意：给结构体里的字符串赋值时，不能直接用赋值运算符（=），可以借助string库里的strcpy函数来复制内容。

· typedef关键字：可以用typedef为现有数据类型创建别名，简化复杂类型声明，提升代码可读性。

· 其语法格式：typedef 原类型名 新类型名;

· typedef常用于为结构体定义一个更简洁的别名，避免每次使用结构体时都要写struct关键字

例：

```
typedef struct Student {
    char name[20];
    int age;
    float score;
} Student;  // Student是struct Student的别名

int main() {
    // 定义结构体变量时，无需写struct关键字
    Student stu = {"Charlie", 22, 78.0};
}
```

· 结构体可以嵌套使用:即一个结构体的成员可以是另一个结构体类型。

例:

```c
#include <stdio.h>

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

typedef struct Student {
    char name[20];
    int age;
    float score;
    Date birthday; // 嵌套结构体
} Student;

int main() {
    Student stu = {"Alice", 20, 85.5, {2005, 3, 11}};
    printf("出生年份: %d\n", stu.birthday.year);
    return 0;
}
```

运行结果如下:

出生年份: 2005

注意:在访问嵌套结构体成员时,需要使用多层成员访问运算符(. 或 ->),如 stu.birthday.year。

- 结构体变量可以作为函数参数：

例：

```c
#include <stdio.h>

typedef struct Student {
    char name[20];
    int age;
    float score;
} Student;

void print_student(Student stu) {
    printf("姓名：%s，年龄：%d，成绩：%.2f\n", stu.name, stu.age, stu.score);
}

int main() {
    Student stu = {"Alice", 20, 85.5};
    print_student(stu);
    return 0;
}
```

运行结果如下：

姓名：Alice，年龄：20，成绩：85.50

· 可以动态地给结构体变量分配内存：

例：

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

typedef struct Student {
    char name[20];
    int age;
    float score;
} Student;

int main() {
    Student* p = (Student*)malloc(sizeof(Student));
    if (p == NULL) {
        printf("内存分配失败\n");
        return 1;
    }

    p->age = 20;
    strcpy(p->name, "Alice");
    p->score = 85.50;
    printf("姓名：%s，年龄：%d，成绩：%.2f\n", p->name, p->age, p->score);
    free(p);  // 必须释放内存
    return 0;
}
```

运行结果如下：

姓名：Alice，年龄：20，成绩：85.50

· 补充：

1、同类型结构体的不同变量之间的赋值：

1）整体赋值：

```
Student stu_1 = {"Alice", 20, 85.5};
Student stu_2 = stu_1;
```

2）逐个成员赋值：

```
Student stu_1 = {"Alice", 20, 85.5};
Student stu_2;
strcpy(stu_2.name, stu_1.name);
stu_2.age = stu_1.age;
stu_2.score = stu_1.score;
```

2、匿名结构体：是一种没有标签的结构体类型，它的核心作用是简化对嵌套成员的访问。

```c
// 传统嵌套结构体（需通过中间名称访问）
struct Student {
    char name[20];
    int age;
    float score;
    struct Date {
        int year;
        int month;
        int day;
    } birthday;
};
struct Student stu;
stu.birthday.year = 2005;  // 需要写birthday.year
```

```c
// 使用匿名结构体（直接访问）
struct Student {
    char name[20];
    int age;
    float score;
    struct {  // 匿名结构体
        int year;
        int month;
        int day;
    };
};
struct Student stu;
stu.year = 2005;    // 直接访问year，无需birthday.year
```

· 注意：

1）内层的匿名结构体不能跟外层结构体的成员同名。

2）匿名结构体需作为外层结构体的成员直接定义，不可单独声明变量。

例：

```
// ❌非法：匿名结构体单独声明变量
struct {
    int year;
    int month;
    int day;
}; // 编译报错：缺少类型名
```

可以通过使用typedef关键字为匿名结构体设置别名，将匿名结构体转换为一个可复用的独立类型，从而解决匿名结构体无法直接声明变量或重复使用的限制。

例：

```
typedef struct {
    int year;
    int month;
    int day;
} Date; // Date是该匿名结构体的别名

int main() {
    // 可以直接使用Date来定义结构体变量
    Date birthday = {2005, 3, 11};
}
```

typedef的作用是给已有的数据类型取个别名

题目1、设有以下语句

```
typedef struct TT {char c; int a[4]} CIN;
```

则下面叙述中正确的是（D）

A．CIN是structTT类型的变量

B．TT是struct类型的变量

C．可以用TT定义结构体变量　struct TT

D．可以用CIN定义结构体变量

声明结构体类型并且定义结构体变量：

1. 先声明结构体类型，再定义结构体变量
2. 声明结构体类型的同时，定义结构体变量
3. 定义匿名结构体类型的同时，定义结构体变量

题目2、下面结构体的定义语句中，错误的是（ B ）

A. `struct ord {int x; int y; int z;}; struct ord a;`

B. `struct ord {int x; int y; int z;} struct ord a;`

C. `struct ord {int x; int y; int z;} a;`

D. `struct {int x; int y; int z;} a;`

题目3、设有以下程序段：

```
struct MP3
{
    char name[20];
    char color;
    float price;
} std, *ptr;
ptr = &std;
```

要引用结构体变量std中的color成员，下列写法中错误的是（C）

A. std.color

B. ptr->color

C. std->color

D. (*ptr).color

结构体变量，我们可以直接用“.”来访问它的成员

指向结构体的指针，我们可以用由一个短横线和一个右尖括号组成的箭头“->”，来访问它所指向的结构体成员

题目4、有以下定义和语句：

```
struct workers
{
    int num;
    char name[20];
    char c;
    struct
    {
        int day;
        int month;
        int year;
    } s;
};
struct workers w, *pw;
pw = &w;
```

能给w中year成员赋1980的语句是（D）

A. *pw.year=1980; (*pw).s.year=1980;

B. w.year=1980; w.s.year=1980;

C. pw->year=1980; pw->s.year = 1980;

D. w.s.year=1980;

解引用运算符的优先级低于成员访问运算符

题目5、有以下结构体说明、变量定义和赋值语句

```
struct STD
{
    char name[10];
    int age;
    char sex;
} s[5], *ps;
ps = &s[0];
```

则以下scanf函数调用语句有错误的是（A）

A. scanf("%d",ps->age);

B. scanf("%d",&s[0].age);

C. scanf("%c",&(ps->sex));

D. scanf("%s",s[0].name);

scanf函数在接收输入的数据时，要传递变量的地址

下标运算符[]和成员访问符（.和->）的优先级高于取址符&

字符串名本身也代表了字符串的起始地址

题目6、有如下程序:

```
#include <stdio.h>
struct person
{
    char name[10];
    int age;
};
main()
{
    struct person room[2] = {{"Wang",19},{"Li",20}};
    printf("%s:%d\n",(room + 1)->name, room->age);
}
```

程序运行后的输出结果是（A）

A. Li:19

B. Wang:19

C. Li:20

D. Wang:17

room[0]:
name : Wang
age: 19

room[1]:
name : Li
age: 20