第二次习题课讲义

T1

```
int main() {
    int m = n = a = b = c = d = 0;
    printf("%d\n", (m = a == b) || (n = c == d));
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

1. 这段代码会报什么类型的错误，应该如何修改?
2. 这段代码的运行结果如何?

答:

报错信息: `identifier "n" "a" "b" "c" "d" is undefined`

正确代码:

```
int main() {
    int m, n, a, b, c, d;
    m = n = a = b = c = d = 0;
    printf("%d\n", (m = a == b) || (n = c == d));
    printf("m=%d, n=%d\n", m, n);
    return 0;
}
```

运行结果:

```
1
m=1, n=0
```

解释:

因为 `int` 只能作用于第一个变量，而后面的变量会被解释为 `n = a = b = c = d = 0;`。因此，这些变量没有明确的类型声明，导致编译器会报错。正确的做法是先声明变量，再进行赋值。

表达式 `(m = a == b) || (n = c == d)` 的执行顺序如下:

- `a == b` 进行比较，结果为1。
- `m = (a == b)` 赋值 `m = 1`，然后 `||` 运算符短路，左边的表达式为真，所以右边就不会计算。
- 所以最后，`m = 1` 和 `n = 0`。

T2

```c
#include<stdio.h>
#define PRODUCT(a, b) a * b

void fun(int n) {
    static int x = 1;
    printf("x=%d\n", x+n);
    x += PRODUCT(x+n, x-n);
}

int main() {
    int i, x = 1;
    for (i = 1; i <= 3; i++, x++) {
        fun(x+i);
    }
    return 0;
}
```

这段代码的运行结果如何?

答:

```
x=3
x=6
x=14
```

解释:

`static` 变量:

- `static int x = 1;` 声明的 `x` 是静态变量，它只会在第一次调用 `fun()` 时初始化为 `1`。之后的每次调用都会使用上次修改后的值。
- 这意味着，`x` 的值在函数调用之间保持不变（与普通局部变量不同，普通局部变量每次调用时都会被重新初始化）。

每次调用 `fun()` 时的执行过程:

- 第一次调用:

$$main函数中x=1,i=1 \Rightarrow fun(2) \Rightarrow fun函数中n=2 \Rightarrow x+n=1+2=3$$
$$PRODUCT(x+n,x-n)=PRODUCT(1+2,1-2)=1+2*1-2=1$$
$$x+=PRODUCT(x+n,x-n) \Rightarrow x+=1 \Rightarrow x=2$$

- 第二次调用:

$$main函数中x=2,i=2 \Rightarrow fun(4) \Rightarrow fun函数中n=4 \Rightarrow x+n=2+4=6$$
$$PRODUCT(x+n,x-n)=PRODUCT(2+4,2-4)=2+4*2-4=6$$
$$x+=PRODUCT(x+n,x-n) \Rightarrow x+=6 \Rightarrow x=8$$

- 第三次调用:

$$main函数中x=3,i=3 \Rightarrow fun(6) \Rightarrow fun函数中n=6 \Rightarrow x+n=8+6=14$$

T3

```c
int main() {
    int a[2][3] = {1, 2, 3, 4, 5, 6};
    int (*p)[3] = &a[0];

    printf("%d,", (*++p)[1]);
    p = a;
    printf("%d", (*p)[1]);

    return 0;
}
```

这段代码的运行结果如何?

答:

```
5,2
```

解释:

```c
int main() {
    int a[2][3] = {1, 2, 3, 4, 5, 6};
    int (*p)[3] = &a[0];  // p 是指向包含 3 个整数的数组的指针，指向 a 的第一行（a[0]）

    printf("%d,", (*++p)[1]);  // p 指向 a[1]，然后访问 a[1][1]，为5
    p = a;  // p 指回 a 的第一行（a[0]）
    printf("%d", (*p)[1]);  // 访问 a[0][1]，为2

    return 0;
}
```

思考:

能否将`int (*p)[3] = &a[0]`替换为`int (*p)[3] = a`?

T4

```c
int main() {
    int m = 5, y = 2;
    y += y -= m *= y;
    printf("y=%d, m=%d", y, m);
    return 0;
}
```

这段代码运行结果如何?

答:

```
y=-16, m=10
```

解释:

这段代码的关键在于理解**运算符的优先级**以及**从右到左的运算顺序**。我们逐步拆解:

1. `m *= y;`:
   - 这是一个复合赋值运算，表示 `m = m * y`。
   - 初始时，`m = 5`，`y = 2`，所以执行 `m = 5 * 2`，结果 `m = 10`。
2. `y -= m;`:
   - 这是另一个复合赋值运算，表示 `y = y - m`。
   - 之前 `m` 的值已经变成了 10，所以执行 `y = 2 - 10`，结果 `y = -8`。
3. `y += y;`:
   - 这是一个复合赋值运算，表示 `y = y + y`。
   - 之前 `y` 的值已经变成了 `-8`，所以执行 `y = -8 + (-8)`，结果 `y = -16`。

T5

下面能正确进行字符串赋值操作的是（）

A：char s[5]={"ABCDE"};

B：char s[5]={'A','B','C','D','E'};

C：char *s; s="ABCDE";

D：char *s; scanf("%s",&s);

答: 选C

解释:

A: `char s[5] = {"ABCDE"};`

- **错误。**虽然 `"ABCDE"` 是一个合法的字符串常量，但数组 `s[5]` 只能容纳 5 个字符，包括字符串结束符 `'\0'`，而 `"ABCDE"` 实际上需要 6 个字符（`A, B, C, D, E, '\0'`）。因此，数组的大小与字符串的大小不匹配。

B: `char s[5] = {'A', 'B', 'C', 'D', 'E'};`

- **错误**，但这不是一个有效的字符串赋值操作。虽然这是一个合法的"字符数组"初始化语句，但是字符串要求最后必须有一个 `'\0'` 结束符，而这个选项没有添加结束符。如果添加了结束符则会与数组的大小不匹配。因此，虽然 `s` 是一个合法的字符数组，但它不是一个合法的 C 字符串。

C: `char *s; s = "ABCDE";`

- **正确。** `"ABCDE"` 是一个字符串常量，指向该常量的指针可以赋给 `char *s`。这在 C 中是合法的，因为字符串常量会自动在末尾添加 `'\0'`，并且会在程序的静态内存区域分配内存。此赋值操作是合法且常见的字符串赋值方式。

D: `char *s; scanf("%s", &s);`

- **错误。**此代码会导致错误，因为 `s` 是一个未初始化的指针（野指针），它指向的内存空间是未知的。使用 `&s` 实际上是将 `scanf` 中的指针地址传给它，但由于 `s` 没有指向一个有效的内存位置，这会导致未定义行为。正确的做法是使用 `char s[大小];` 来声明一个字符数组，或者为 `s` 分配内存空间（例如通过 `malloc`）。

T6

```
typedef struct student {
    char name[50];
    int age;
} Student;

```

```
struct student {
    char name[50];
    int age;
} Student;

```

上述两段代码中的Student的作用有何区别?

答:

第一段代码 使用了 `typedef`，使得 `Student` 成为 `struct student` 的别名。你可以直接使用 `Student` 来声明结构体变量。

- `Student` 是类型别名
- 例如：`Student s1;`，`s1` 是 `struct student` 类型的变量。

第二段代码 中，`Student` 是一个变量名，表示你定义了一个 `struct student` 类型的变量 `Student`。结构体类型本身仍然是 `struct student`。相当于语句`struct student Student;`。