知识点总结

·**作用域**：程序中标识符（如变量等）的有效范围。

局部作用域：在函数内部或代码块（用花括号{}括起来的一段代码）内定义的变量具有局部作用域，这些变量被称为**局部变量**。

局部变量只在其定义的函数或代码块内可见和可用。

```c
#include <stdio.h>

void func() {
    // local_var是局部变量
    int local_var = 10;
    {
        // inner_var是局部变量
        int inner_var = 20;
        printf("Inner variable: %d\n", inner_var);
    }
    // 此行会导致编译错误，因为inner_var在此处不可见
    printf("Inner variable: %d\n", inner_var);

    printf("Local variable: %d\n", local_var);
}

int main() {
    func();
    // 此行会导致编译错误，因为local_var在此处不可见
    printf("Local variable: %d\n", local_var);
    return 0;
}
```

·全局作用域：在所有函数外部定义的变量具有全局作用域，这些变量被称为全局变量。全局变量在整个程序的任何地方都可以访问。

```c
#include <stdio.h>
// global_var是全局变量
int global_var = 100;

void print_global() {
    //在print_global函数中可以直接访问全局变量global_var
    printf("Global variable: %d\n", global_var);
}

int main() {
    //在main函数中可以直接访问全局变量global_var
    printf("Global variable in main: %d\n", global_var);
    print_global();
    return 0;
}
```

·注意：当局部变量和全局变量同名时，在局部变量的作用域内，局部变量会遮蔽全局变量，即使用该变量名时，访问到的是局部变量的值，而不是全局变量的值。

```c
#include <stdio.h>

// num是全局变量
int num = 10;

void test() {
    // num是跟全局变量同名的局部变量
    int num = 20;
    // 此处访问到的是局部变量num的值20
    printf("Local num: %d\n", num);
}

int main() {
    // 此处访问到的是全局变量num的值10
    printf("Global num: %d\n", num);
    test();
    return 0;
}
```

运行结果如下：

Global num: 10

Local num: 20

·函数的实参、形参：

实参（实际参数）：出现在函数调用的括号中，是传递给函数的实际数据。可以是常量、变量、表达式或函数返回值等具体数据。

形参（形式参数）：出现在函数定义的括号中，是函数需要外部提供的数据。

例：

```c
#include <stdio.h>
// a和b是形参
int add(int a, int b) {
    int result = a + b;
    return result;
}

int main() {
    int num_1 = 5, num_2 = 3;
    // num_1和num_2是实参
    int sum = add(num_1, num_2);
    printf("The sum is: %d\n", sum);
}
```

·另外，实参和形参的命名可以相同，并不会产生混淆的后果。这是因为实参在调用函数的作用域里，而形参在被调用函数的作用域里，它们的可见范围并不重叠。

·函数的参数传递：

1）值传递：指的是在调用函数时传入外部数据的值。在函数里对形参的修改不会影响外部实参的值。

例：

```c
#include <stdio.h>
// num是形参
void change_value(int num) {
    num = num * 2;
    printf("调用了函数，num的值：%d\n", num);
}
int main() {
    int value = 10;
    printf("调用函数前，value的值：%d\n", value);
    // value是实参
    change_value(value);
    printf("调用函数后，value的值：%d\n", value);
}
```

运行结果如下：

```
调用函数前，num的值：10
调用了函数，value的值：20
调用函数后，value的值：10
```

2）**指针传递**：指的是在调用函数时**传入外部数据的地址**。在函数里通过**对指针的解引用**，可以**访问和修改原数据的值**。

例：

```c
#include <stdio.h>
// ptr是形参
void change_value(int* ptr) {
    // 通过解引用指针ptr，修改了ptr所指向的变量值。
    *ptr = *ptr * 2;
    printf("调用了函数，ptr指向的变量值：%d\n", *ptr);
}

int main() {
    int value = 10;
    printf("调用函数前，value的值：%d\n", value);
    // value是实参
    change_value(&value);
    printf("调用函数后，value的值：%d\n", value);
}
```

运行结果如下：

```
调用函数前，ptr指向的值：10
调用了函数，value的值：20
调用函数后，value的值：20
```

题目1、有以下程序:

```
#include<stdio.h>
func(int x)
{
    x = 10;
    printf("%d, ", x);
}
main( )
{
    int x = 20;
    func(x);
    printf("%d", x);
}
```

程序的运行结果为: 10, 20

题目2、有以下程序:

```
#include <stdio.h>
int a = 1;
int f(int c) {
    int a = 2;
    c = c + 1;
    return (a++) + c;
}
main() {
    int i, k = 0;
    for (i = 0; i < 2; i++) {
        int a = 3;
        k += f(a);
    }
    k += a;
    printf("%d\n", k);
}
```

程序运行后的输出结果是_____。

13

题目3、有以下程序

```
#include <stdio.h>
int k=5;
void f(int *s)
{
    s = &k;
    *s = 7;
}
main()
{
    int m = 3;
    f(&m);
    printf("%d,%d\n", m, k);
}
```

程序运行后的输出结果是（D）

A. 3,5

B. 7,7

C. 5,7

D. 3,7