知识点总结

一、指针函数

在C语言中，指针函数是一种特殊的函数类型，其返回值为指针，这意味着函数执行完毕后返回的是一个地址。

定义指针函数的语法格式：

```
返回值类型* 函数名(参数类型1 参数名1, 参数类型2 参数名2, ...) {
    // 函数体
    ...
    return 指针;
}
```

其中，返回值类型是函数返回的指针所指向的数据类型。

注意：要确保函数返回的指针储存的地址是有效的。地址是不是有效，跟这个地址上储存的数据的生命周期有关，如果这个数据的生命周期已经结束了，那么这个地址会被视为无效地址。例如，如果返回的是局部变量的地址，函数结束时局部变量被销毁，局部变量的地址就变成无效地址，访问会导致未定义行为。

函数返回的指针储存的是有效地址的情况包括：

1、返回全局变量的地址。

全局变量的作用域是整个程序，在程序的任何函数内部均可对其进行访问。其生命周期是从程序运行开始到结束，即贯穿程序运行的全过程。因此，在程序运行期间，全局变量的地址始终保持有效。

例：指针函数返回全局变量的地址。

```c
#include <stdio.h>
// 全局变量num
int num = 10;

int* return_var() {
    int* ptr = &num;
    return ptr;
}

int main() {
    int* ptr = return_var();
    printf("全局变量num的值: %d\n", *ptr);
}
```

2、返回静态局部变量的地址。

静态局部变量的生命周期也是从程序运行开始到结束，所以，在程序运行期间，静态局部变量的地址始终保持有效。

定义静态变量的语法格式：static 数据类型 变量名 = ...;

例：指针函数返回静态局部变量的地址。

```c
#include <stdio.h>

int* return_var() {
    // 静态局部变量num
    static int num = 10;
    int* ptr = &num;
    return ptr;
}

int main() {
    int* ptr = return_var();
    printf("静态变量num的值: %d\n", *ptr);
}
```

另外，静态局部变量只有在第一次调用函数时，会被初始化一次，之后再调用函数时，它会保留上次修改后的值。

例1：将函数内的count定义为普通局部变量。

```c
#include <stdio.h>

void calls() {
    // 局部变量count
    int count = 0;
    count++;
    printf("这是你第%d次调用函数\n", count);
}

int main() {
    for (int i = 0; i < 5; i++) {
        calls();
    }
}
```

该程序的运行结果如下：

```
这是你第1次调用函数
这是你第1次调用函数
这是你第1次调用函数
这是你第1次调用函数
这是你第1次调用函数
```

例2：将函数内的count定义为静态局部变量。

```c
#include <stdio.h>

void calls() {
    // 静态局部变量count
    static int count = 0;
    count++;
    printf("这是你第%d次调用函数\n", count);
}

int main() {
    for (int i = 0; i < 5; i++) {
        calls();
    }
}
```

该程序的运行结果如下：

```
这是你第1次调用函数
这是你第2次调用函数
这是你第3次调用函数
这是你第4次调用函数
这是你第5次调用函数
```

3、返回动态分配内存的地址。

动态分配的内存，只要没有被释放，并且程序没有结束，其地址始终保持有效。

动态分配内存最基础且常用的函数是malloc，该函数在stdlib库（即标准实用工具库）里，使用前需先包含stdlib.h这个头文件。

· malloc函数能够按我们的需求，分配指定字节数的连续内存空间，并返回一个指向该内存起始地址的指针，该指针的类型是void*，即无类型指针。

· 在C语言里，我们可以通过类型转换，把无类型指针强制转换为任何其它类型的指针。

· malloc函数的使用格式：(数据类型*)malloc(需要分配的内存大小);

其中，数据类型表示计划使用所分配内存来储存的数据的具体类型。

· 注意：

内存分配后，需检查返回指针是否为NULL。若不为NULL，表明内存分配成功，可继续进行后续的操作；若为NULL，则意味着内存分配失败。
调用指针函数（比如这里的return_var函数）后，同样要先检查返回指针是否为NULL。

另外，对于动态分配的内存，使用完毕后应及时通过free函数（位于stdlib库里）释放，避免浪费内存资源。

free函数的使用格式：free(指向动态内存的指针名)

例：指针函数返回动态分配内存的地址。

```c
#include <stdio.h>
#include <stdlib.h>
int* return_var() {
    // 动态分配内存
    int* ptr = (int*)malloc(sizeof(int));
    if (ptr != NULL) {
        *ptr = 10;
    }
    return ptr;
}
int main() {
    int* ptr = return_var();
    if (ptr != NULL) {
        printf("动态分配内存里的值: %d\n", *ptr);
        // 释放动态分配的内存
        free(ptr);
    }
    else {
        printf("内存分配失败\n");
    }
}
```

二、函数指针

在C语言中，函数指针是一种特殊的指针类型，它指向的是函数而非普通变量。

函数指针的定义与初始化：

定义语法：返回值类型 (*指针名)(参数类型1, 参数类型2...);

其中，返回值类型是指针所指向的函数的返回值类型。

后面的参数类型，要跟指针所指向的函数需要的所有参数的数据类型，在数量以及顺序上都要一样。

例：假设有一个函数add，用于计算两个整数的和。

定义指向该函数的函数指针如下：

```c
int add(int a, int b) {
    return a + b;
}

int main() {
    int (*fp)(int, int);
    // 等价于 int (*fp)(int, int) = &add;
    int (*fp)(int, int) = add;
    return 0;
}
```

· 通过函数指针调用函数：

调用格式：指针名(实参1, 实参2, ...)

例：通过指针fp调用函数add。

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main() {
    int (*fp)(int, int) = add;
    // 等价于 int result = (*fp)(3, 5);
    int result = fp(3, 5);
    printf("结果是: %d\n", result);
}
```

- 函数指针作为函数参数：

函数指针常作为函数参数使用，这使得函数具有更高的灵活性，可以根据传入指向不同函数的函数指针，来执行不同的操作。

该程序的运行结果如下：

结果是：15

结果是：5

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

void operate(int a, int b, int (*fp)(int, int)) {
    int result = fp(a, b);
    printf("结果是：%d\n", result);
}

int main() {
    int num_1 = 10;
    int num_2 = 5;
    int (*fp)(int, int) = add;
    operate(num_1, num_2, fp);
    // 以下两行代码，可以简化成：operate(num_1, num_2,
subtract);
    fp = subtract;
    operate(num_1, num_2, fp);
}
```

返回值类型 (*指针名)(参数类型1, 参数类型2...);

题目1、设有以下函数：

void fun(int n, char* s) {……}

则下面对函数指针的定义和赋值均是正确的是（ D ）

A．void（* pf）(); pf = fun;

B．void * pf(); pf = fun;

C．void * pf(); * pf = fun;

D．void(* pf)(int,char); pf = &fun;

指针函数可以返回无类型指针

指针函数返回的地址可以是全局变量的地址，或者是静态变量的地址

静态局部变量，函数调用结束后，不会像普通局部变量那样被销毁，而是一直存在，直到程序结束

题目2、关于指针函数的说法正确的是（ C ）

A. 指针函数不能返回void*类型

B. 指针函数返回的地址必须来自动态内存分配

C. 可以返回函数内static变量的地址

D. 指针函数的返回值必须立即使用

题目3、有以下程序:

```
#include <stdio.h>
int add(int a, int b)
{
    return (a + b);
}
main()
{
    int k, (*f)(int, int), a = 5, b = 10;
    f = add;
    …
}
```

则以下函数调用语句错误的是（A）

A. k = *f(a, b);

B. k = add(a, b);

C. k = (*f)(a, b);

D. k = f(a, b);

题目4、有以下程序:

```
#include <stdio.h>
int *f(int *s, int *t)
{
    if(*s < *t) *s = *t;
    return s;
}
main()
{
    int i = 3, j = 5, *p = &i, *q = &j, *r;
    r = f(p, q);
    printf("%d,%d,%d,%d,%d\n", i, j, *p, *q, *r);
}
```

程序的运行结果是（A）

A. 5,5,5,5,5

B. 3,5,5,5,5

C. 5,3,3,3,5

D. 3,5,3,5,5

题目5、有以下程序：

```
#include <stdio.h>
int* get_value() {
    static int a = 10;
    return &a;
}
int main() {
    *get_value() += 5;
    printf("%d", *get_value());
    return 0;
}
```

程序的运行结果是 15

静态局部变量只有在第一次调用函数时，会被初始化一次，之后再调用函数时，它会保留上次修改后的值