C语言问题汇总

printf返回值

根据printf函数原型

```
int printf(const char* restrict format, ...);
```

知，函数返回写入字符的数量。

```
    int returnValue = printf("输入%d\n", 2);

    printf("printf返回%d", returnValue);
```

```
输入2
printf返回6
--------------------------------
Process exited after 0.1519 seconds with return value 0
请按任意键继续. . .
```

通过结果可知，printf返回的可理解为字符串中的字符个数，由于汉字占两个字符，故返回2 * 2 + 1 + 1 = 6

return返回值如何显示

利用变量接收，子程序函数返回主调函数，宣告函数的一次执行结束，在调用期间所分配的变量单元被撤消

```
#include <stdio.h>
int max(int, int);          // 获取两数较大值，函数声明

int main()
{
    int ans = max(1, 2);    // 调用函数，利用返回类型变量接收函数返回值
```

```
    printf("%d", ans);

    return 0;
}
int max(int a, int b)
{
    return a > b ? a : b;
}
```

返回值一般可保存函数处理结果或处理状态等信息，同时立即终止当前函数

数据类型*的使用

不是数据类型，但可以与基本数据类型结合形成复合的指针类型，例如int a表示a为一个指向整型变量的一维指针变量

```
int a = 10;
int* pa = &a;           // 指向a的内存地址
printf("*pa = %d", *pa); // 通过指针获取变量值
printf("pa = %p", p);    // 通过指针获取变量地址
```

```
*pa = 10
pa = 000000000061FE14
```

这里只讲述简单的指针应用，随着后续学习的跟进，会重点讲述。

自定义函数的调用

若需要调用自定义函数，首先需要声明和定义函数

```
#include <stdio.h>
void sayHello();        // 函数声明

int main()
{
    sayHello();
```

```
    return 0;
}
/*
 * 函数定义
 * 功能：输出Hello, World
*/
void sayHello()
{
    printf("Hello, world");
}
```

注意：函数声明一般位于头文件之后，主函数之前，函数定义位于主函数之后。也直接在头文件后实现函数定义（此时不必声明）；但是为了突出主函数。让主要逻辑直观体现，工程代码仍建议体现函数声明部分。

调用函数时只需要体现函数名及函数实参，如sayHello()就是对自定义函数的定义。若需要利用返回值，则创建变量接收返回值，否则返回值会被丢弃（对非静态函数）。

指针怎么用

指针被称为C语言的灵魂，其用于存放地址的变量，是无符号整型

- 提高程序的编译效率和执行速度：由于直接针对内存地址，故执行效率更高，但也更危险
- 通过指针可使用主调函数和被调函数之间共享变量或数据结构，便于实现双向数据通讯：

  由于函数一般通过值传递，故只能实现主函数（调用函数）向被调函数信息的传递，被调函数只能透过返回值实现简单的应答。由于指针直接通过地址，所以不用担心由函数创建的临时（局部变量）被销毁，可以直接通过地址获取由被调函数改变的信息。以changeValue函数为例：

  ```
  #include <stdio.h>

  void changeValeu#include <stdio.h>

  void changeValueByNormal(int);
  void changeValueByPointer(int*);
  ```

```c

int main(void)
{
    int a = 4;

    changeValueByNormal(a);

    printf("changeValueByNormal: %d\n", a);

    changeValueByPointer(&a);

    printf("changeValueByPointer: %d\n", a);

    return 0;
}

void changeValueByNormal(int num)
{
    num = 5;
}
void changeValueByPointer(int *num)
{
    *num = 5;
}
```

```
changeValueByNormal: 4          // 修改失败
changeValueByPointer: 5
```

- 可以实现动态的存储分配：动态内存在堆（heap）上创建，堆是向高地址扩展的数据结构，是不连续的内存区域，理论上可分配内存更大，同时更灵活，最常用于数组的创建。

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

int main(void)
{
    int* a = malloc(sizeof(int) * 4);       // 创建4个int存储单元
```

```
    int* current = a;
    int count = 1;
    while (count <= 4) {
        *current = count++;
        printf("%d ", *current);
        current++;
    }
    return 0;
}
```

- 便于表示各种数据结构，编写高质量的程序

  基本的数据结构都涉及指针，例如链表等等。

什么时候用scanf

scanf是一个针对所有基本数据类型均可输入的函数，对于字符串数组或字符串指针变量，由于数组名可以转换为数组和指针变量名本身就是地址，因此使用scanf()函数时，不需要在它们前面加上&操作符。

针对字符型变量可以选用效率更高的getchar()直接读取字符，对于字符串可使用gets()，与scanf()输入不同的是，gets()允许接收空格字符。对于两者而言，读取字符串由于不指定字符串长度，会产生风险，故可自定义字符串读入函数readLine()。

有时出于时间的考量，会使用快读函数读取数字（int），速度快于scanf

```
int read()
{
    int x = 0,f = 1;
    char c = getchar();
    while (c <'0'|| c > '9') {
        if(c=='-') f=-1;
        c=getchar();
    }
    while(c>='0'&&c<='9') {
        x =x * 10 + c - '0';
    }
```

```
    c = getchar();

    return x * f;
}
```