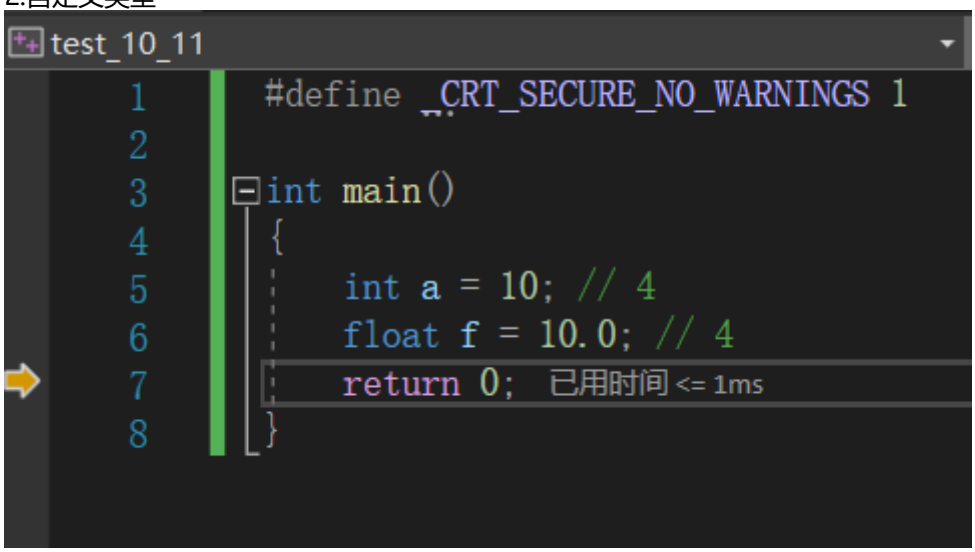
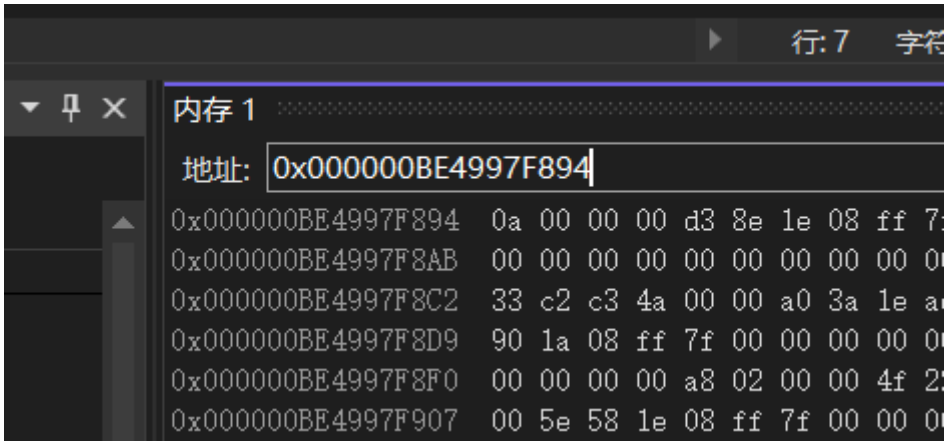
数据的存储

笔记本:	My Notebook		
创建时间:	2023/10/11 19:57	**更新时间:**	2023/10/12 19:14
作者:	dtdkc1mu		

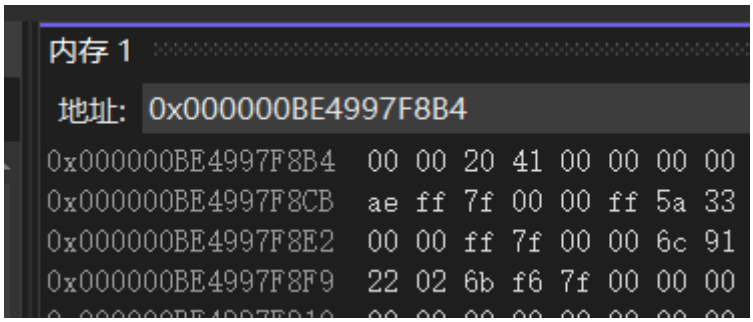
C语言类型
1.内置类型
char short int long long long float double
类型的意义:
使用这个类型开辟内存空间的大小（大小决定了使用范围）
如何看待内存空间的视角
2.自定义类型



&a - 0a000000



&b - 00002041



整型家族

```
char
    unsigned char
    signed char
short
    unsigned short [int]
    signed short [int]
int
    unsigned int
    signed int
long
    unsigned long [int]
    signed long [int]
```

浮点数家族

```
float
double
```

构造类型

```
> 数组类型
> 结构体类型 struct
> 枚举类型 enum
> 联合类型 union
```

指针类型

```
int *pi;
char *pc;
float* pf;
void* pv;
```

void表示空类型（无类型）
通常应用与函数的返回类型、函数的参数、指针类型

计算机中的有符号数有三种表示方式，即原码、反码和补码
三种表示方式均有符号位和数值位两部分，符号位都是用0表示'正'，用1表示'负'，而数值位三种表示方法不同
负整数的三种表达方式各不相同
原码：直接将二进制按照正负数的形式翻译成二进制就可以
反码：将原码的符号位不变，其它位依次按位取反即可
补码：反码+1得到补码
正数的原、反、补码都相同
对于整数来说：数据存放在内存中其实存放的是补码
why?
在计算机系统中，数值一律用补码来表示和存储。原因在于,可以将符号位和数值域统一处理；同时，加法和减法也可以统一处理（CPU只有加法器），补码和原码相互转换，其运算过程是相同的，不需要额外的硬件电路。

补码在内存中存放时为什么顺序不对?

大小端介绍

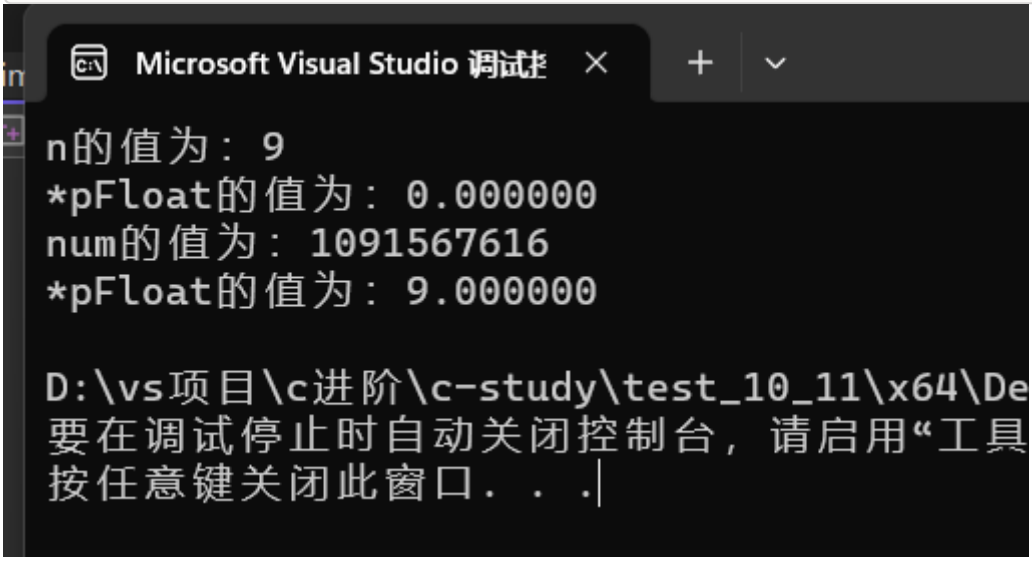
大端(存储)模式，是指数据的低位保存在内存的高地址中，而数据的高位，保存在内存的低地址中
小端(存储)模式，是指数据的低位保存在内存的低地址中，而数据的高位，保存在内存的高地址中
为什么要有大端和小端?
这是因为在计算机系统中，我们是以字节为单位的，每个地址单原都对应着一个字节，一个字节为8bit。但是在C语言中除了8bit的char之外，还有16bit的short型，32bit的long型（要看具体的编译器），另外，对于位数大于8位的处理器，例如16位或者32位的处理器，由于寄存器宽度大于一个字节，那么必然存在着一个如何将多个字节安排的问题。
因此就导致了大端存储模式和小端存储模式。

笔试题

```
//写一段代码，告诉我们当前机器的字节序是什么
#include <stdio.h>
int check_sys()
{
        int i = 1;
        return (*(char*)&i);
}
int main()
{
        int ret = check_sys();
        if (ret == 1)
        {
               printf("小端\n");
        }
        else
        {
               printf("大端\n");
        }
        return 0;
}
```

浮点型在内存中的存储

```
int main()
{
    int n = 9;
    float *pFloat = (float *)&n;
    printf("n的值为：%d\n",n);
    printf("*pFloat的值为：%f\n",*pFloat);
    *pFloat = 9.0;
    printf("num的值为：%d\n",n);
    printf("*pFloat的值为：%f\n",*pFloat);
    return 0;
}
```



num和*pFloat在内存中明明是同一个数，为什么浮点数和整数的解读结果会差别这么大?

根据国际标准IEEE，任意一个二进制浮点数V可以表示下面的形式
(-1)^S*M*2^E
(-1)^S表示符号位，当S=0，V为正数；当S=1，V为负数
M表示有效数字，大于等于1，小于2
2^E表示指数位
32位浮点数

单精度浮点数存储模型

64位浮点数

双精度浮点数存储模型

IEEE 754对有效数字M和指数E，还有一些特别的规定
1<=M<2,也就是说，M可以写成1.xxxxxxx的形式，其中xxxxxxx表示小数部分。
IEEE 754规定，在计算机内部保存M时，默认这个数的第一位总是1，因此可以被舍去，只保存后面的xxxxx部分。
等到读取的时候再把第一位的1加上去，这样做的目的，是节省1位有效数字；
至于指数E，首先E为8位，它的取值范围为0~255；如果E为11位，它的取值范围为0~2047.但是，科学技术法的E是可以出现负数的，所以IEEE 754规定，存入内存时E的真实值必须再加上一个中间数，对于8位的E，这个中间数时127；对于11位的E，这个中间数是1023.比如，2^10的E是10，所以保存成32位浮点数时，必须保存成10+127 = 137；即10001001.
当指数E从内存中取出还可以再分成三中情况：
E不全为0或不全为1
这时，指数E的计算值减去127（或1023），得到真实值，再将有效数字M前加上第一位的1.
E全为0
这时，浮点数的指数E等于-126，有效数字M为0，这样做是为了表示+-0，以及接近于0的很小的数字
E全为1
这时，E=128

```
#include <stdio.h>
int main()
{
        float f = 5.5;
        // 5.5
        // 101.1
        // (-1)^0 * 1.011*2^2
        // S = 0
        // M = 1.011
        // E = 2
        // 0 10000001011000000000000000000000
        // 0X40b00000
        return 0;
}
```