使用调试技巧

笔记本:	My Notebook		
创建时间:	2023/10/10 18:22	**更新时间:**	2023/10/11 17:43
作者:	dtdkc1mu		

调试的基本步骤
发现程序错误的存在
以隔离、消除等方式对错误进行定位
确定错误产生的原因
提出纠正错误的解决办法
对程序错误予以改正，重新测试

Debug和Release的介绍
Debug成为调试版本，它包含调试信息，并且不作任何优化，便于程序员调试程序
Release称为发布版本，它往往是进行了各种优化，使得程序在代码大小和运行速度上都是最优的，以便用户很好的使用。

常用快捷键
F5
启动调试，经常用来直接跳到下一个断点处。
F9
创建断点和取消断点
断点的重要作用，可以在程序的任意位置设置断点。
这样就可以使得程序在想要的位置随意停止执行，继而一步步执行下去。
F10
逐过程，通常用来处理一个过程，一个过程可以是一次函数调用，或者是一条语句。
F11
逐语句，就是每次都执行一条语句，但是这个快捷键可以使我们的执行逻辑**进入函数内部**（这是最
长用的）。
CTRL + F5
开始执行不调试，如果你想让程序直接运行起来而不调试就可以直接使用。

```
//debug下vs2022编译器编译以下代码时出现死循环
//release下会改变变量的内存布局，从而使得死循环消失了
int main()
{
        int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
        int i = 0;
        for (i = 0; i <= 15; i++)
        {
               printf("hehe\n");
               arr[i] = 0;
        }
        return 0;
}
```

1.栈区的默认使用：
先使用高地址处的空间
再使用低地址处的空间
2.数组随着下标的增长，地址是由低到高变化

如何写出好(易于调试)的代码

常见的coding技巧
1.使用assert
2.尽量使用const
3.养成良好的编码风格
4.添加必要的注释
5.避免编码的陷阱

```
//以下代码能够编译通过，const变量 num被修改，因为我将&num给了指针p，p并不知道num不能修改
int main()
{
	const int num = 10;
	int* p = &num;
	*p = 20;
	printf("%d\n", num);
	return 0;
}

//修改后,const修饰指针变量的* 左边时，修饰的是*p，也就是说不能通过p修改*p，也就是num的值
// const如果修饰指针变量* 的右边时，如int* const p， 修饰的时p，不能修改p，也就是p的地址(指向)不能修改了
int main()
{
	const int num = 10;
	const int* p = &num;
	printf("%d\n", num);
	return 0;
}
```

```
//复制字符串
char* my_strcpy(char* dest, const char* src)
// const 用来修饰src，表示src为要复制的源头数据，不能被修改
{
	char* ret = dest;
	assert(dest && src); // 断言
	 // 把src指向的字符串拷贝到dest指向的空间，包含'\0'字符
	while (*dest++ = *src++)
	{
		;
	}
	return ret;
	/*while (*src != '\0')
	{
		*dest = *src;
		dest ++;
		src++;
	}
	*dest = *src;*/
}
```

注：
1.分析参数的设计（命名，类型），返回值类型的设计
2.注意野指针，空指针的危害
3.assert的使用
4.参数部分const的使用

```
// 模拟实现求字符串长度的函数
int my_strlen(const char* str)
{
	int count = 0;
	assert(str != NULL);
	while (*str != '\0')
	{
		count++;
```

```
                str++;
        }
        return count;
}
```

编程时常见的错误
1.编译时错误
2.链接型错误 - 看错误提示信息，一般是标识符名不存在或者拼写错误
3.运行时错误 - 借助调试，逐步定位问题