实验 10 符号表的构建和使用

实验难度：★★☆☆☆
建议学时：2 学时

一、 实验目的

- 了解符号表的结构。
- 掌握符号表的插入、查找和删除等基本操作。

二、 预备知识

- 学习了符号表的概念以及作用域的基本规则。
- 在这个实验中主要用到了杂凑表（哈希表）的插入和查找等操作。如果读者对这一部分知识有遗忘，可以复习一下数据结构中的相关内容。

三、 实验内容

3.1 阅读实验源代码

SymbolTable.h 文件

主要定义了与符号表相关的数据结构，这些数据结构定义了一个符号表的单链表存储形式，允许每个作用域使用独立的符号表。其中 Symbol 结构体用于定义符号的相关信息和 Symbol 单链表，SymbolTable 结构体用于定义作用域对应的杂凑表和 SymbolTable 单链表。具体内容可参见下面两个表格。

Symbol 的域	说明
SymbolName	符号名称
SymbolType	符号类型
ClashCount	冲突次数。当一个符号名称在其作用域中被重复定义时，此计数器就要增加 1。
RefCount	引用次数。当一个符号名称在其作用域中被引用时，此计数器就要增加 1。
pNext	指向下一个 Symbol

SymbolTable 的域	说明
Bucket	杂凑表（桶）
Invalid	作用域是否无效的标志。1 表示无效，0 表示有效
pNext	指向下一个 SymbolTable

main.c 文件

首先，定义了符号表链表的头指针和符号引用失败计数器两个全局变量，然后定义了 main 函数。在 main 函数中调用了 CreateSymbolTable 函数构造符号表。

在 main 函数的后面，定义了一系列函数，有关这些函数的具体内容参见下面的表格。关于这些函数的参数和返回值，可以参见其注释。

函数名	功能说明
NewSymbol	创建一个新的符号。注意，调用了 memset 函数将符号的内容清空，这与将符号结构体中的每个字段分别清空是等效的，但是使用 memset 函数可以编写更少的代码，执行效率更高。
NewSymbolTable	创建一个新的符号表。注意，也调用了 memset 函数将符号表的内容清空。
PushScope	在符号表链表的表头添加一个作用域。 **提示：** ● 可以调用 NewSymbolTable 函数创建符号表。 ● 添加一个作用域和入栈操作类似。 此函数的函数体还不完整，留给读者完成。
PopScope	将符号表链表中最内层的作用域设置成无效。 **提示：** ● 并不需要将符号表从符号表链表中移除，只需要将该符号表中的 Invalid 域置成 1 即可。 ● 设置一个作用域无效和出栈操作类似。 此函数的函数体还不完整，留给读者完成。
AddSymbol	向符号表中添加一个 Symbol。 **提示：** ● 每当定义一个新的变量时，会调用此函数向符号表中添加一个 Symbol。 ● 如果新定义变量的名称在其作用域中与已定义变量的名称重复，就不能添加 Symbol，而应将已定义变量的 ClashCount 域增加 1。 此函数的函数体还不完整，留给读者完成。
RefSymbol	对 Symbol 进行一次引用。 **提示：** ● 每当使用一个已定义的变量时，会调用此函数将所用变量的 RefCount 域增加 1。 ● 如果使用的变量未定义，此函数需将符号引用失败计数器（RefErrorCount）增加 1。 此函数的函数体还不完整，留给读者完成。
Hush	求 Symbol 的哈希值。

CreateSymbolTable	**提示：**
	● 通过调用 PushScope 函数模拟进入作用域。
	● 通过调用 PopScope 函数模拟退出作用域。
	● 通过调用 AddSymbol 函数模拟定义一个变量。
	● 通过调用 RefSymbol 函数模拟使用一个变量。
	此函数中的代码模拟了下列源代码在进入作用域、退出作用域、定义变量和使用变量时对符号表的操作。
	{
	int i, j;
	int f(); // 函数声明

```
    {
        char i;
        int size;
        char temp;
        {
            char* j;
            long j; // 重复定义
            j = NULL;
            i = ‘p’;
            size = f();
            new = 0; // 引用失败
        }
        j = 3;
    }
}
```

3.2 编写源代码并通过验证

按照下面的步骤继续实验：

1. 为函数体不完整的函数编写源代码。注意尽量使用已定义的局部变量。
2. 按 Ctrl+Shift+B,在弹出的下拉列表中选择“生成项目”。如果生成失败，根据“TERMINAL”窗口中的提示信息修改源代码中的语法错误。
3. 按 Ctrl+Shift+B，然后在下拉列表中选择“测试”，启动验证。在“TERMINAL”窗口中会显示验证的结果。如果验证失败，可以在“EXPLORER”窗口中，右击“result_comparation.html”文件，在弹出的菜单中选择“Open Preview”，可以查看用于答案结果文件与读者编写程序产生的结果文件的不同之处，从而准确定位导致验证失败的原因。

提示:如果需要通过验证，请不要修改 CreateSymbolTable 函数中的代码，只需将其他函数体补充完整即可。

四、思考与练习

1. 编写一个 FreeSymbolTable 函数，当在 main 函数的最后调用此函数时，可以将整个符号表链表的内存释放掉，从而避免内存泄露。
2. 编写一个 ScanSymbolTable 函数，当在 main 函数的最后调用此函数时，可以对整个符号表进行扫描，对于未被引用的变量发出警告。
3. 修改 PopScope 函数，不再使用符号表的无效标志，而是将作用域对应的符号表从链表头移除，并考虑此时如何对未被引用的变量发出警告。