# 四川师范大学 实验报告

学期： 2024 至 2025 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2023 级 9 班 实验编号： 04

实验项目： 实验四 指导老师： 冯山

姓名： 刘智恒 学号： 2023060522

**一、实验题目**

字符串操作基础

**二、实验目的及要求**

掌握关字符串ADT的存储结构和基本操作算法设计与实现特点。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.设计并实现求字符串长度的算法。

2.设计并实现求子串的算法。

3.设计并实现字符串的串接合并算法。

4.设计并实现字符串的子字符串置换算法。

5.设计并实现字符串的子字符串插入算法。

**注：存储结构可以选SString或Hstring，也可以是链表或块链结构。**

**四、实验准备**

1.计算机设备;

2.程序调试环境的准备，本实验采用**Microsoft Visual Studio**环境;

3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序**Exp\_4**准备。

**注：本实验中将实验内容1至5进行了整合，故以下的实验测试用例用于5个问题的测试。**

5.实验测试用例：

（1）**主串S：ABCDEFGABCDEF；**

**待求子串Sub1：CD；(pos=3, len=2)**

**待求子串Sub2：EFG；（pos=5, len=3)**

**插入子串Sub2的位置：pos=1**

（2）**主串S：AAAAABBBBCCCDDE；**

**待求子串Sub1：AA；(pos=1, len=2)**

**待求子串Sub2：DE。(pos=14, len=2)**

**插入子串Sub2的位置：pos=15**

**五、实验过程**

**（一）问题分析**

（1）本实验中的所有实验内容均针对于**字符串**的基础操作，字符串的数据类型为基本的**字符类型（char）**，其组成为**连续的一段字符**，故可采用**数组**对字符串进行存储。由此，数据结构上采用**动态顺序表**结构更为方便，同理，采用链表结构也可行，在此不作赘述。动态顺序表类型定义如下：

**typedef struct SqString{ //字符串数据结构**

**SElemType \*data; //存储字符**

**int length; //字符串（当前）长度**

**}SqString;**

（2）各项操作基本分析：

**[1]求字符串长度**：由于在字符串数据结构中设定了长度**length**的信息，故可通过**length**进行返回得到当前字符串的长度值。

**[2]求子串**：该操作基于给定的**主串S，**以及给定的主串中的位置**pos**和给定的待求子串长度**len**对子串进行求解。在求解子串之前，还需要对给定的**pos**和**len**信息**进行合法判断**。

**[3]字符串的串接**：串接即为将一个字符串连接到另一个字符串后面。本实验中，不妨将两个字符串的串接结果**保存至一个新的字符串**中；同时，为了更明显地显示串接结果，采取**对求得的两个子串Sub1和Sub2进行串接**的操作。

**[4]子串的置换**：置换即为在主串**S**中指定一子串**Sub1**，然后再指定另一子串**Sub2**，**用Sub2对主串中出现的所有Sub1进行替换**。此时需要求得**Sub1**在**S**中**所有的出现位置**信息，然后依次进行替换；除此之外，还应该考虑子串**Sub1**和子串**Sub2**的**长度差异**：**若长度相同则直接替换，若不同，则应在替换之前先进行元素的移动。**本实验中，为了适当地简化具体操作，采取**用求得的Sub2对Sub1逐个进行替换**。

**[5]子串的插入**：插入即为在给定插入位置**pos**的情况下，将子串插入到主串**S**的**pos**位置，在此之前，需要**将主串S的pos位置及其之后的所有字符向后移动该子串长度的距离**。该操作**与顺序表的插入操作类似**，同样的，也需要对给定的pos信息进行**合法判断**。

**（二）算法描述**

（1）**Status InitString(SqString \*S);**

**//**本算法用于字符串的初始化：为顺序字符串分配内存，大小为定义的**MAXLEN**，如果分配失败，则返回错误状态。同时将字符串长度初始化为0。

（2）**Status DestroyString(SqString \*S);**

**//**本算法用于字符串的销毁：首先检查**data**是否为空，如果非空，则释放**data**所占用的内存。

（3）**void PrintString(SqString S);**

**//**本算法用于输出打印字符串：遍历字符串的每个字符，并逐个输出。

（4）**int StringLength(SqString S);**

**//**本算法用于求字符串的长度：在求长度之前，首先将长度信息初始化为0，然后通过循环遍历，逐个检查字符，直到遇到字符串结束符‘\0’；最后返回计算获得的长度。

（5）**Status FindSubString(SqString S, SqString \*Sub, int pos, int len);**

**//**本算法用于求字符串的子串：首先检查**pos**和**len**是否合法，再设置子字符串的长度为**len**；然后复制从位置**pos-1**开始的**len**个字符到**Sub**中；最后在**Sub**后添加字符串结束符‘\0’。

（6）**Status ConnectString(SqString \*Str, SqString Sub1, SqString Sub2);**

**//**本算法用于连接两个字符串：先设置串**Str**的长度为**Sub1**和**Sub2**长度之和，然后依次将**Sub1**和**Sub2**的数据复制到**Str**中。

（7）**Status SeekSubPos(SqString S, SqString Sub1, int a[]);**

**//**本算法用于查找并存储待替换子串的所有出现位置：在主串**S**中查找子串**Sub1**的所有出现位置**pos**，将子串**Sub1**的第一个字符位置信息存储至数组**a[]**中。

（8）**Status ReplaceSubString(SqString \*S, SqString Sub1, SqString Sub2, int a[]);**

**//**本算法用于子串的替换：在查找到**Sub1**在**S**中的所有位置信息之后，首先检查**Sub1**和**Sub2**的长度差异，然后计算替换后**S**的新长度，同时检查新长度是否超出**MAXLEN**限制，接着从后向前替换**Sub1**为**Sub2**以避免覆盖；最后更新S的长度。

（9）**Status InsertSubString(SqString \*S, SqString Sub, int pos);**

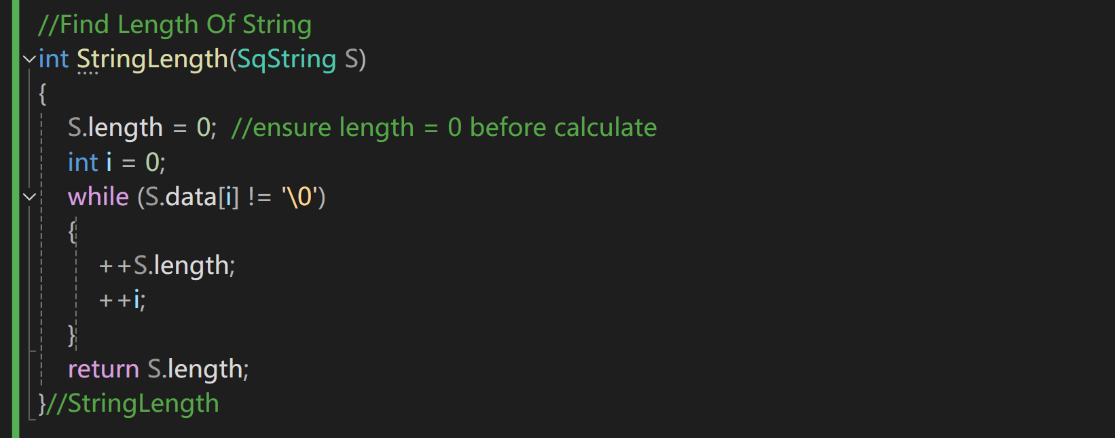
**//**本算法用于插入子串：首先检查插入位置是否合法，再将S中的元素从插入位置开始向后移动，以腾出空间；然后将Sub的内容插入到位置pos；最后更新字符串S的长度。

**（三）程序代码**

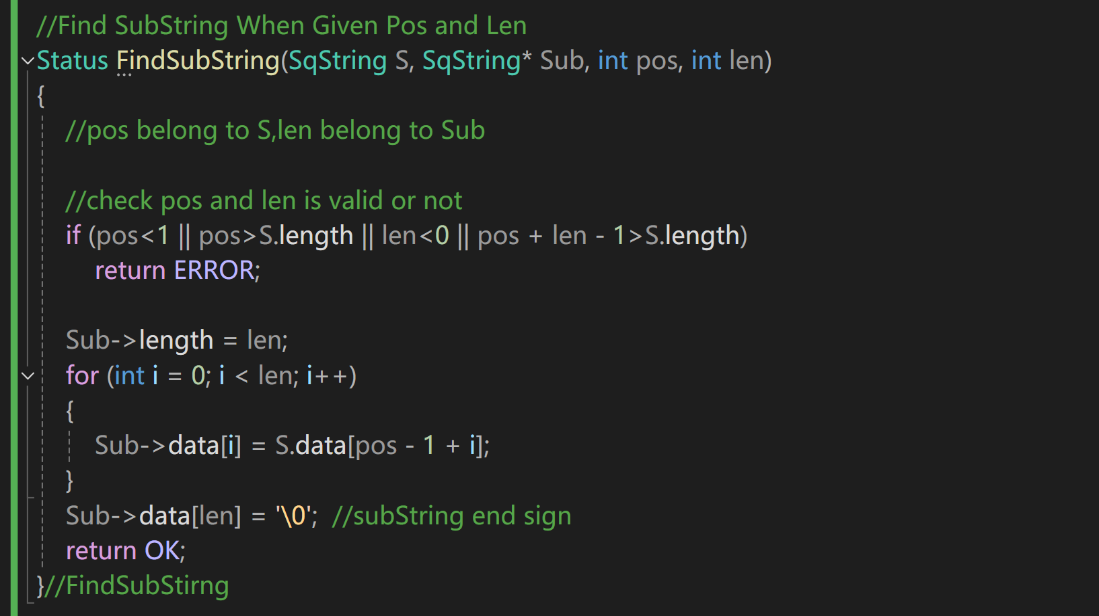
根据以上对**实验内容**的算法描述，求解的程序代码如下：

****

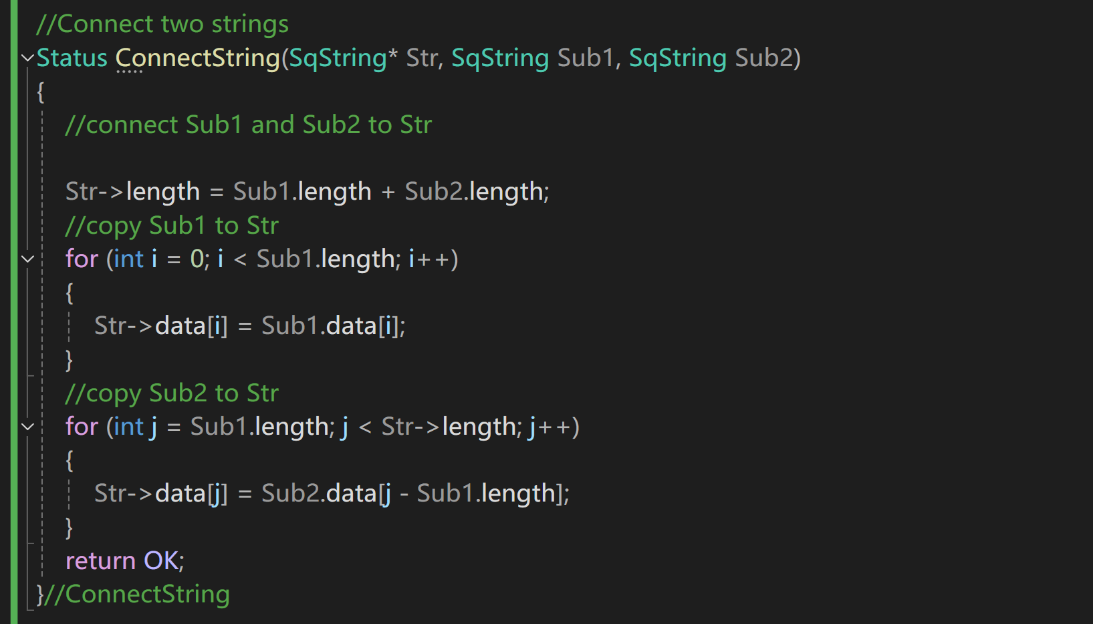
**注：此处只展示主要算法的程序代码图。**

****

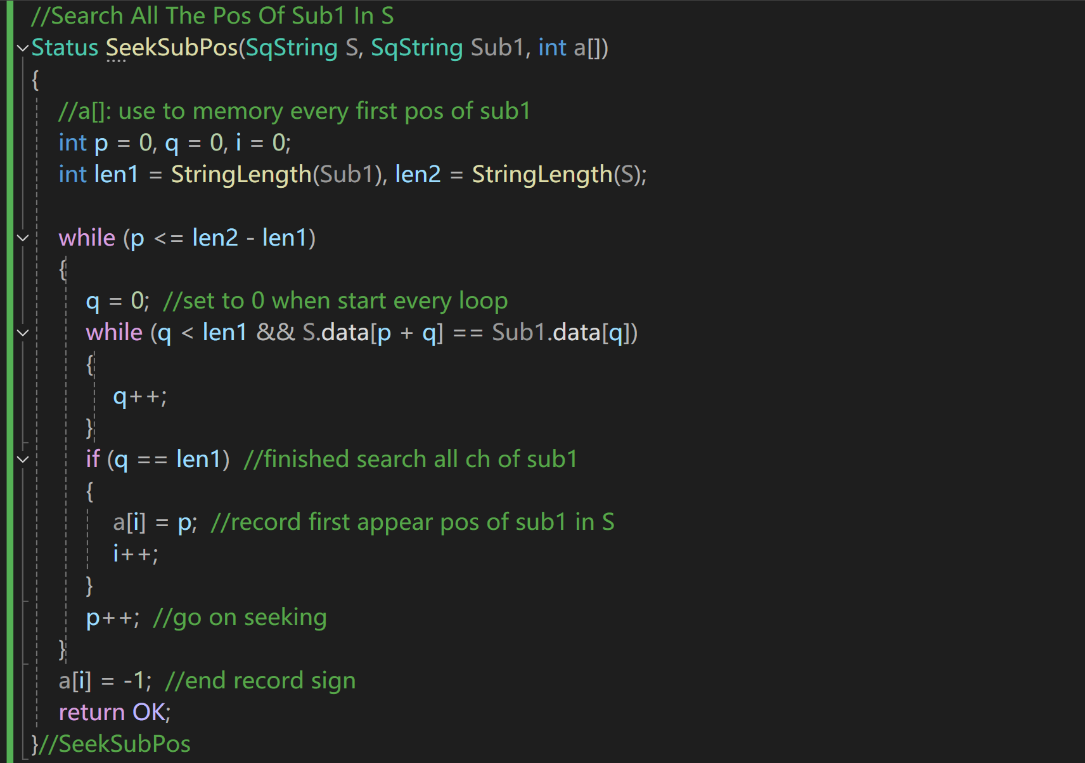
**图5.1求字符串长度part**

****

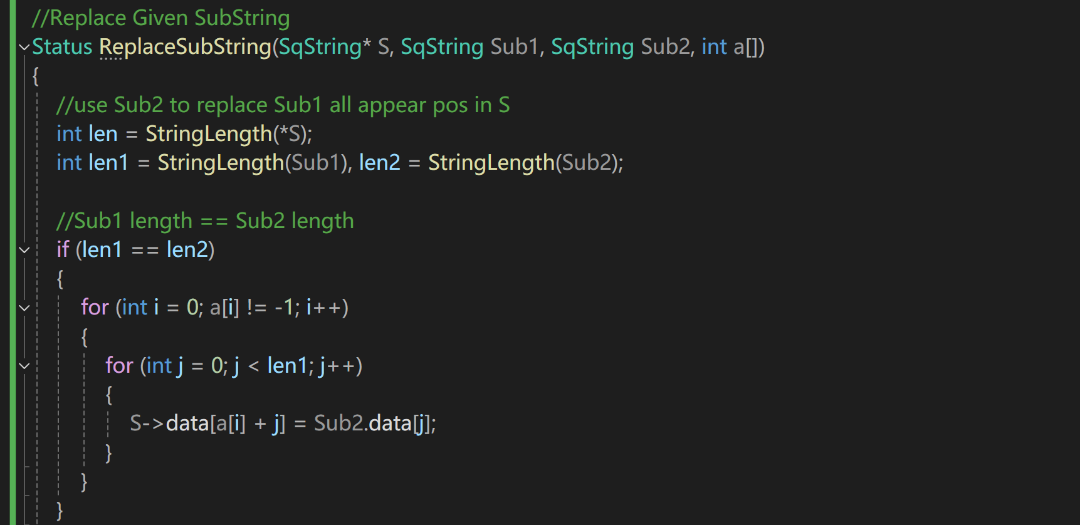
**图5.2求字符串子串part**

****

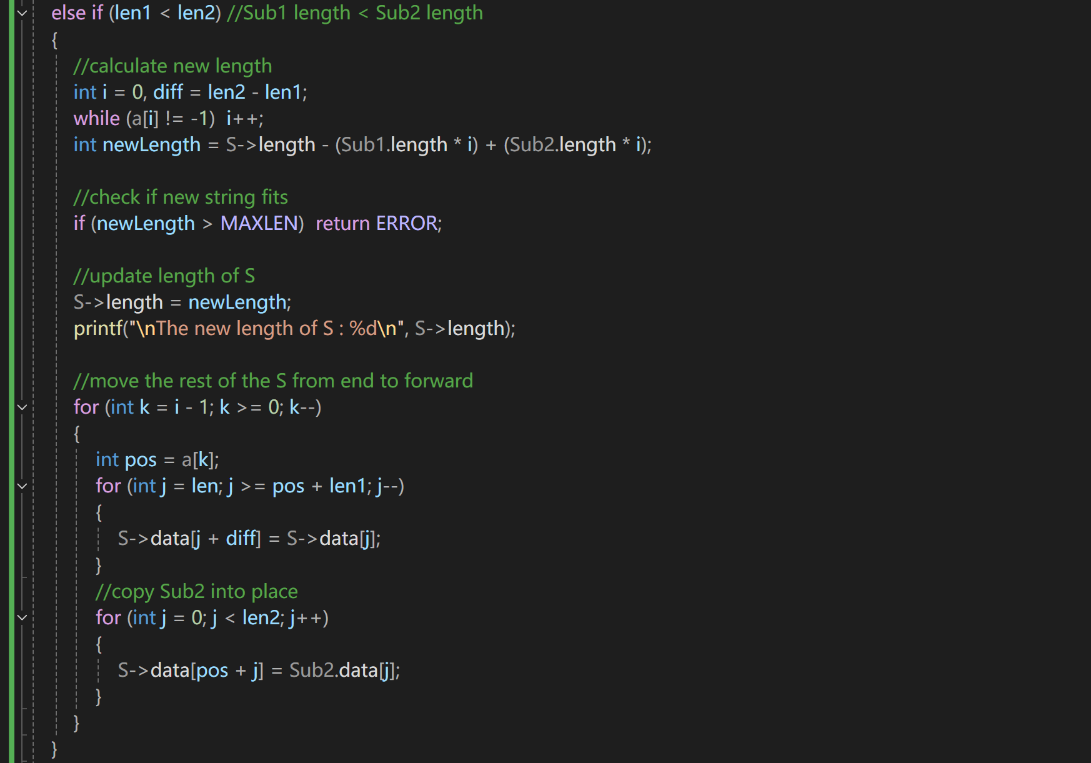
**图5.3连接字符串part**

****

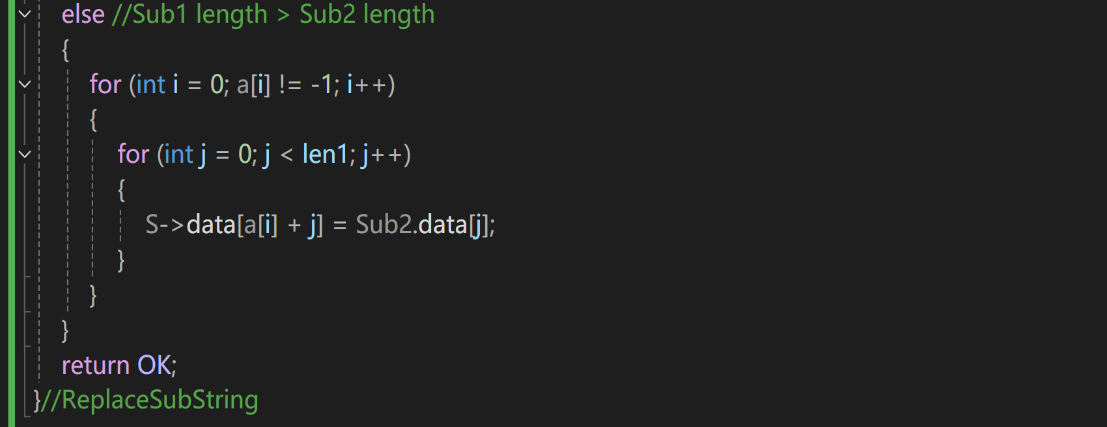
**图5.4查找子串位置**

****

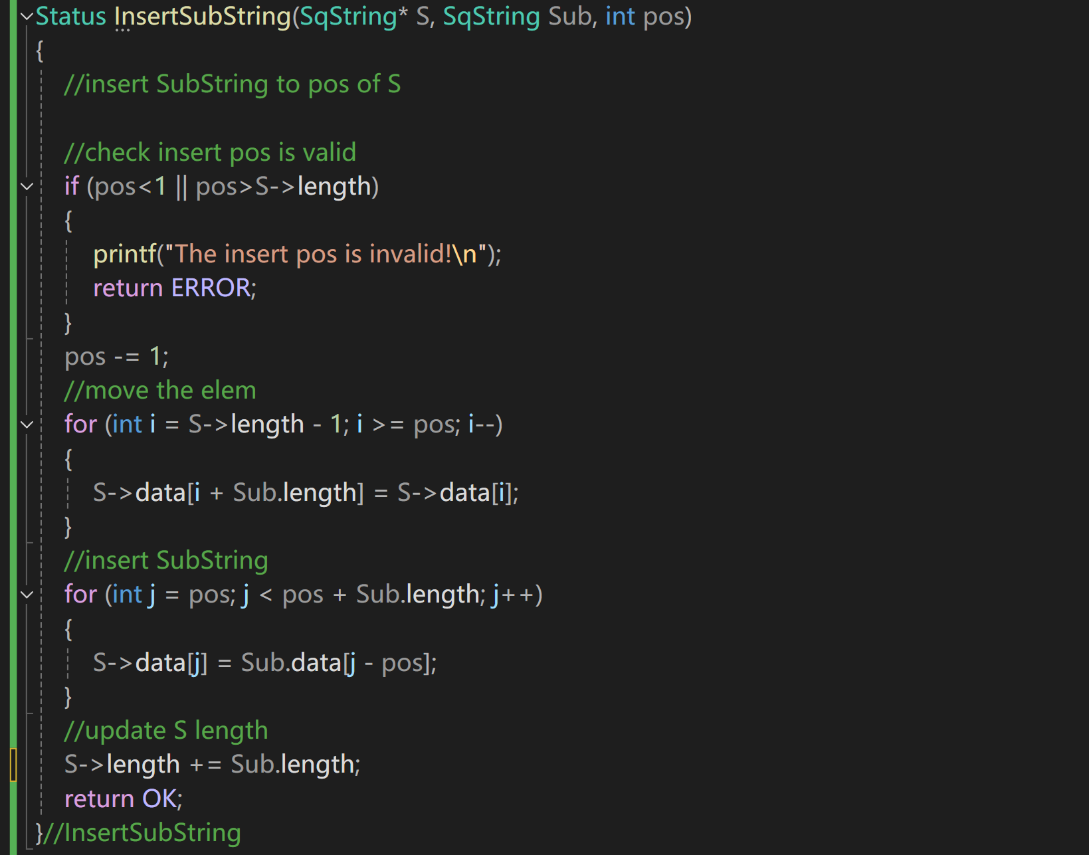
**图5.5子串置换part\_1**

****

**图5.6子串置换part\_2**

****

**图5.7子串置换part\_3**

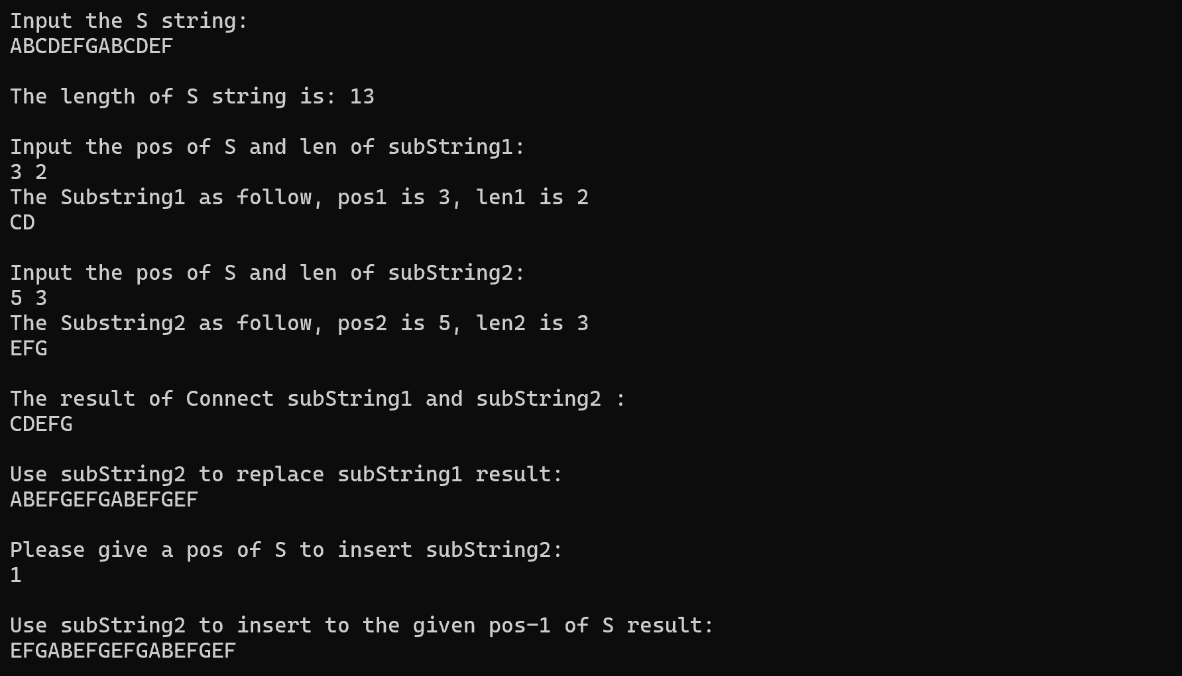
****

**图5.8子串插入**

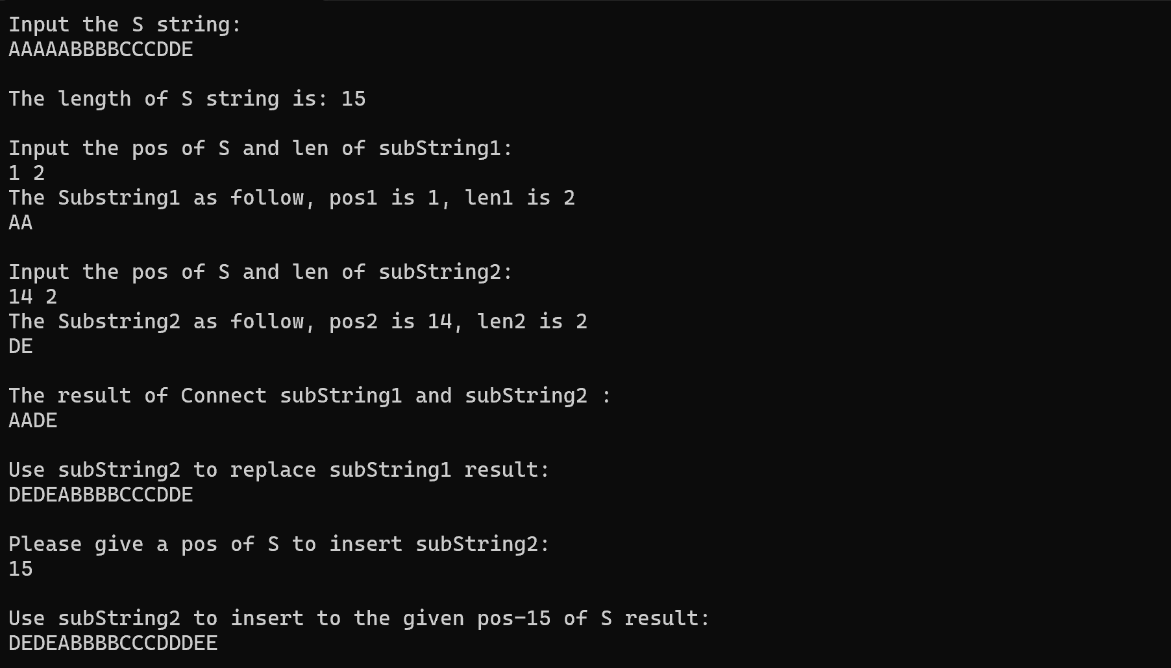
**六、结果分析**

**（一）结果呈现：**

1.根据以上**实验内容**的程序代码运行后进行多次测试，以下是根据**实验内容测试用例**进行测试后的结果示意图：



**图6.1测试用例1结果**

****

**图6.2测试用例2结果**

**（二）结果分析：**

**1.时间复杂度分析**

**【InitString】—— 初始化字符串**

该函数仅进行内存分配和初始化，不涉及循环操作。故时间复杂度为**O(1)**。

【**DestroyString**】**—— 销毁字符串**

该函数只是释放内存，没有循环操作。故时间复杂度为**O(1)**。

**【PrintString】—— 打印字符串**

需要遍历字符串的每个字符进行打印。故时间复杂度为**O(n)**，其中**n**为字符串的长度。

【**StringLength**】**——** **求字符串长度**

需要遍历字符串直到遇到结束符**\0**。故时间复杂度为**O(n)**，其中**n**为字符串的长度。

【**FindSubString**】**—— 求字符串子串**

在最坏情况下，可能需要遍历整个字符串来确定子字符串的位置。故时间复杂度为**O(n)**，其中**n**为字符串的长度。

【**ConnectString**】**—— 连接字符串（子串）**

该函数将两个字符串的子串连接到一个新的字符串中，分别遍历**Sub1**和**Sub2**的长度，时间复杂度为**O(m+n)**，其中**m**和**n**分别为**Sub1**和**Sub2**的长度。

【**SeekSubPos**】—— **查找子串位置**

外层循环的最大迭代次数为**len2-len1+1**，其中**len2**为字符串**S**的长度，**len1**为子字符串**Sub1**的长度。内层循环在最坏情况下需要检查完整的**Sub1**，即最多需要**len1**次比较。因此在最坏情况下，时间复杂度为**O((len2-len1+1)\*len1)=O(len2\*len1)**。

【**ReplaceSubString**】**—— 子串替换**

在处理**相等长度（len1==len2）**以及**len1>len2**的情况时，时间复杂度均为**O(n\*len1)**，其中**n**是子串**Sub1**的出现次数，**len1**为子串**Sub1**的长度。在**Sub1**的长度小于**Sub2**的情况下，需要计算新长度，找到所有出现位置的时间复杂度为**O(n)**；移动剩余字符的时间复杂度为**O(len2-len1)**，因为每次出现位置都会影响到后面字符的移动；替换字符的复杂度为**O(n\*len2)**；故此时综合时间复杂度为**O(n\*(len2-len1+len2))=O(n\*len2)**。

综上所述，**ReplaceSubString**的时间复杂度为：**O(n\*max(len1,len2))**。

【**InsertSubString**】**—— 子串插入**

该函数需要移动元素以插入子串**Sub**，此时的时间复杂度为**O(n)**，然后再复制**Sub**的长度，为**O(m)**，因此总复杂度为**O(n+m)**。

**综上所述：整个程序的时间复杂度主要受ReplaceSubString函数的影响，在最坏情况下可能达到O(n\*k)，其中n是字符串S的长度，k是Sub1出现的次数。**

**2.空间复杂度分析：**

**【数据结构的空间占用】**

**SqString**结构体中包含一个字符指针**data**和一个整数**length**,其中**data**指向一个大小为**MAXLEN（预定义常量）**的动态分配的数组。对于每一个**SqStrin**g对象，它消耗的空间是**O(MAXLEN)**，并且包含整数的长度**length**，即**O(1)**。因此，每个**SqString**对象的总空间复杂度为**O(MAXLEN+1)=O(MAXLEN)**。

**【动态内存分配】**

在**InitString**函数中，为**data**动态分配内存，大小为**sizeof(SElemType)\*MAXLEN**，这会占用一个额外的**O(MAXLEN)**的空间。而在最后的**DestroyString**函数中，我们释放了之前分配的内存。

【**临时变量**】

在多个函数中，定义了一些局部变量，比如**ReplaceSubStirng**函数中用数组**a[MAXLEN]**用于存储子串位置。在这些情况下，这些局部变量的空间复杂度是常量级别的，为O(1)或**O(MAXLEN)**。

**综上所述：主要的空间消耗来自于SqString对象的动态内存分配，其空间复杂度为O(MAXLEN)。因此，整个的空间复杂度为O(MAXLEN)。**

**七、实验总结**

1.通过动态内存分配**（malloc）**来管理字符串的数据，可以灵活地根据需要来分配内存，有利于提高内存使用率。

2.在各个函数中都有对输入参数的**有效性检查**。例如，**检查位置是否合法、长度是否合理**等。这种对边界条件的检查有利于确保程序的健壮性。

3.在进行**子串替换**和**子串插入**时，使用**反向遍历**的方式避免了**覆盖**问题。