实验一　顺序结构程序设计

一　实验目的

1　学会使用 C 的有关算术运算符，以及包含这些运算符的表达式，特别是自加（++）和自减（--）运算符的使用。

2 掌握 C 语言中赋值语句的使用。

3　掌握 C 语言中各种数据的输入/输出方法，能正确使用各种格式转换符。

4　理解 C 语言程序的顺序结构。

二　实验内容

题目 1：长方形长为 x，宽为 y，求长方形面积和周长。

题目 2：输入一个三位数，将其加密后输出。加密方法是对该数的每一位数字将其加 6 后除以 10 取余数，作为该位上的新数字，再交换个位数字和百位数字。

三、算法描述（描述形式可采用流程图或伪代码）

题目一

```
1.  1. 从用户输入读取整数 a 和 b
2.  2. 计算 c = 2 * (a + b)
3.  3. 计算 d = a * b
4.  4. 输出 c 和 d 到控制台
5.  5. 返回 0，表示程序成功结束
```

题目二

```
1.  1. 从用户输入读取整数 a
2.  2. 使用数组 b 存储 a 的每一位数字
3.     a. 计算 b[0] 为 a 的百位数
4.     b. 计算 b[1] 为 a 的十位数
5.     c. 计算 b[2] 为 a 的个位数
6.  3. 对数组 b 中的每个元素执行如下操作：
7.     a. 将当前元素加上 6
8.     b. 对 10 取模，保留个位数
9.  4. 交换数组 b 中第一个元素和第三个元素的值
10. 5. 输出数组 b 中的每个元素到控制台
```

```
6. 返回 0，表示程序成功结束
```

四、源代码

题目一

```c
#include <stdio.h>
int main() {
    int a,b;
    scanf("%d %d",&a,&b);
    int c = 2*(a+b);
    int d = a*b;
    printf("%d\n%d",c,d);
    return 0;
}
```

题目二

```c
#include <stdio.h>
int main() {
    int a;
    scanf("%d",&a);
    int b[3];
    b[0] = a/100;
    b[1] = (a-b[0]*100)/10;
    b[2] = (a-b[0]*100-b[1]*10);
    for (int i = 0; i < 3; i++) {
        b[i] = (b[i]+6)%10;
    }
    int temp;
    temp = b[0];
    b[0] = b[2];
    b[2] = temp;
    for (int j = 0; j < 3; j++) {
        printf("%d",b[j]);
    }
    return 0;
}
```

四、实验结果及心得体会

实验一的题目主要涉及顺序结构程序设计，包括算术运算符的使用、赋值语句的应用、数据输入/输出方法的掌握以及对程序顺序结构的理解。以下是对实验的心得体会：

对于题目一（长方形面积和周长）：

- 通过这个题目，我进一步巩固了 C 语言中基本的输入输出操作和算术运算符的使用。

- 理解了如何从用户输入获取数据，进行简单的计算，然后输出结果。

- 熟悉了常见的算术运算符，如加法、乘法，以及变量的使用和赋值语句的编写。

对于题目二（三位数加密）

- 这个题目涉及到了更多的变量和数组的使用，对于数字的拆解和加密有了更深的了解。

- 掌握了如何使用数组存储数字的每一位，并且对数组进行遍历和操作。

- 在交换数组元素的过程中，我学到了如何使用临时变量来实现两个变量值的交换。

- 简单的加密算法也让我意识到在编程中可以通过一些简单的操作来实现一些基本的安全性。

总体心得：

- 实验一让我对 C 语言的基础知识有了更深的理解和熟练掌握，这对于后续的编程学习打下了基础。

- 对于顺序结构的程序设计，理解程序的执行流程，如何按照步骤执行，是编程学习中重要的一步。

- 实验过程中，通过观察和调试程序，我不仅学到了如何编写代码，还学到了如何更好地理解程序的运行过程和调试技巧。

总体来说，实验一为我提供了一个扎实的 C 语言基础，让我更加自信地迎接后续的编程挑战。通过实际动手编写代码，我深刻体会到了编程的乐趣和挑战，对于算法和逻辑的思考也让我受益匪浅。

实验二　选择结构程序设计

一　实验目的

1　掌握 C 语言表示逻辑量的方法(0 代表“假”，1 代表“真”)；

2　学会正确使用逻辑运算符和逻辑表达式；

3　熟练掌握 if 语句和 switch 语句。

二　实验内容

题目 1：判断一个三位数是否为回文数（回文数即左右对称数，例 232,585）。

题目 2：用 switch 语句完成改题目。某高速公路收费标准，小汽车每公里 0.5 元，大客车每公里 1.5 元，轻型货车每公里 2.5 元，重型汽车每公里 3.5 元，请计算实际收费金额。

三、算法描述（描述形式可采用流程图或伪代码）

题目一

```
读取字符数组 b
如果 b 的第一个字符等于第三个字符
    输出 "YES"
否则
    输出 "NO"
```

题目二

```
输出 "Car is 1."
输出 "Bus is 2."
输出 "Small Truck is 3."
输出 "Large Truck is 4."
输出 "Please enter a number:"
读取整数 a
调用函数 fact(a)
    根据 a 的值执行 switch 语句：
        - 如果 a 等于 1，设置 b 为 0.5
        - 如果 a 等于 2，设置 b 为 1.5
        - 如果 a 等于 3，设置 b 为 2.5
        - 如果 a 等于 4，设置 b 为 3.5
```

```
13.             - 如果 a 不在上述范围内，输出错误信息并重新读取 a，并再次调用 fact(a)
14.     读取距离值 c
15.     输出 "The cost is " 后接 c 乘以 b 的值
16.     返回 0，表示程序成功结束
```

四、源代码

题目一

```
1.    #include <stdio.h>
2.    int  main(){
3.        char b[3];
4.        scanf("%s",b);
5.        if(b[0]==b[2]){
6.            printf("YES");
7.        }
8.            else{
9.                printf("NO");
10.           }
11.       return 0;
12.   }
```

题目二

```
1.    #include <stdio.h>
2.    int  main(){
3.        char b[3];
4.        scanf("%s",b);
5.        if(b[0]==b[2]){
6.            printf("YES");
7.        }
8.            else{
9.                printf("NO");
10.           }
11.       return 0;
12.   }
```

五、实验结果及心得体会

实验二主要涉及选择结构程序设计，包括 if 语句和 switch 语句的应用。以下是对两个题目的心得体会：

对于题目一（判断回文数）

在这个题目中，我使用了 if 语句来判断输入的三位数是否为回文数。

学到了如何正确使用逻辑运算符和逻辑表达式，通过比较字符数组的首位和末位来判断是否回文。

注意到在 C 语言中，字符数组的比较不能直接使用==，而应该使用 strcmp 函数，但这里只是对字符进行单个字符的比较，所以直接使用==进行比较是合适的。

对于题目二（高速公路收费计算）

在这个题目中，我使用了 switch 语句来计算不同类型车辆的收费金额。

学到了如何通过 switch 语句根据不同的情况执行不同的代码块，提高了程序的可读性和灵活性。

注意到在字符数组中存储数字时，需要注意字符和数字之间的转换，可以使用 atoi 函数将字符数组转换为整数。

总体心得：

实验二让我更深入地了解了选择结构在程序设计中的应用，if 语句和 switch 语句是控制程序流程的关键工具。

此次实验中，我对逻辑运算符和表达式的理解更加深入，能够根据具体需求构建合适的逻辑条件。

使用字符数组处理输入数据时，需要注意字符和数字之间的差异，确保在比较和计算时能够得到正确的结果。

实验结果：

题目一的实验结果是通过 if 语句判断输入的三位数是否为回文数，并输出相应的结果。

题目二的实验结果是通过 switch 语句计算不同类型车辆的实际收费金额。

通过实验二的完成，我对 C 语言中的选择结构有了更加深刻的认识，并能

够灵活运用这些结构来解决实际问题。这为我的编程能力提供了一定的提升。

实验三　循环结构程序设计

一　目的和要求

1　掌握在设计条件型循环结构时，如何正确地设定循环条件，以及如何正确地控制计数型循环结构的次数。

2　熟悉用 while 语句,do-while 语句和 for 语句实现循环的方法。

3　掌握在程序设计中用循环的方法实现各种算法(如穷举、选代、递推等)。

4　掌握选择结构与循环结构的嵌套。

二　实验内容

题目：日本中学生发现一个奇妙的”定理”请角谷教授证明，而教授无能为力，因此产生了角谷猜想，猜想的内容是：任给一个自然数，若为偶数则除以 2，若为奇数则乘 3 加 1，得到一个新的自然数后按照上述规则继续验算，若干次后得到的结果必然是 1。请编程验证。

三、算法描述（描述形式可采用流程图或伪代码）

```
定义函数 fanc(a)：
    如果 a 除以 2 的余数等于 0，则执行以下操作：
        设置 a 为 a 除以 2 的商
    否则，执行以下操作：
        设置 a 为 3 乘以 a 加 1
    如果 a 的值等于 1，则返回 1，否则返回 fanc(a)

在主函数 main() 中：
    读取整数 a
    如果调用 fanc(a) 的结果等于 1，则输出 "RIGHT"
    否则，输出 "WRONG"
    返回 0，表示程序成功结束
```

四、源代码

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

```
int fanc(int a){
    if(a%2==0)
        a=a/2;
    else
        a=3*a+1;
    if(a==1)
        return 1;
    else
        return fanc(a);
}
int  main(){
    int a;
    scanf("%d", &a);
    if(fanc(a)==1)
        printf("RIGHT");
    else
        printf("WRONG");
    return 0;
}
```

五、实验结果及心得体会

实验三的题目是关于角谷猜想的程序设计，主要目标是验证角谷猜想是否成立。以下是对该实验的心得体会：

在本次实验中，我通过编写 C 程序来实现对角谷猜想的验证。这个猜想涉及到对给定的自然数进行一系列的操作，直到最终结果变为 1。通过实验，我深刻体会到了循环结构在算法设计中的重要性和灵活运用。

学到的知识点：

循环结构的运用： 通过这个实验，我更深刻地理解了 while 循环的使用方法。在验证角谷猜想的过程中，循环结构帮助程序不断进行迭代，直到满足退出条件。

递归的思想： 角谷猜想的验证可以使用递归来实现。在函数中调用自身，不断缩小问题的规模，这种递归思想在编程中是一种常见的思

维方式。

实验心得：

算法设计思维： 实验要求验证一个猜想，这涉及到对问题的抽象和算法设计。我学到了如何将一个自然数经过一系列操作变为 1 的算法设计过程。

调试与测试： 在编写程序的过程中，我遇到了一些逻辑错误和边界情况，这促使我学会了如何进行调试和测试。通过多次运行程序，逐步发现问题并加以解决。

代码的简洁性： 在不影响程序功能的前提下，我尽量使代码简洁易懂。这有助于他人阅读和理解我的代码，同时也更方便日后的维护。

总结：

通过这个实验，我不仅学到了关于循环结构、递归和算法设计的知识，同时提高了编程的实际能力。实验不仅仅是对课堂知识的应用，更是对计算机思维和解决问题能力的培养。这次实验让我更深入地感受到编程的乐趣和挑战，为我今后的学习和工作奠定了更加坚实的基础。

实验四　数组

一　实验目的

1　掌握一维数组的定义、赋值和输入输出的方法；

2　掌握字符数组和字符串函数的使用；

3　掌握与数组有关的算法

二　实验内容

现有 21 根火柴，两人轮流取，每人每次可以取走 1 至 4 根，不可多取，也不能不取，谁取最后一楔火柴谁输。请编写一个程序进行人机对弈，要求人先取，计算机后取；计算机一方为“常胜将军”。最后输出人、机操作步骤及每次取火柴数量。

三、算法描述（描述形式可采用流程图或伪代码）

```
// 定义函数，返回参数 a 和 4 的较小值
function func(a):
    if (a 大于等于 4)
        返回 4
    否则
        返回 a

// 主函数
function main():
    // 初始化变量
    a = 21
    b 是一个 21 行 2 列的数组
    sum = 0
    round = 0

    输出提示信息："Please enter a number from 1 to 4:"

    // 循环直到 sum 大于 21
    for 每一轮:
        输入用户输入的数存入 b[i][0]

        // 检查用户输入的合法性
```

```
            如果 (b[i][0] 大于 func(a - sum) 或者 b[i][0] 小于 1)
                输出提示信息："Please enter a number from 1 to func(a - sum):"
                再次输入用户输入的数存入 b[i][0]

            // 处理计算机输入
            如果 (b[i][0] 大于 func(a - sum) 或者 b[i][0] 小于 1)
                继续循环
            否则
                sum 增加 b[i][0]
                b[i][1] 被计算为 5 减去 b[i][0]
                sum 增加 b[i][1]
                round 增加 1
                输出计算机输入："The computer entered b[i][1]"
                输出当前总和："The sum is sum"
                输出剩余数字："The remaining number is (a - sum)"
                输出提示信息："Please enter a number from 1 to func(a - sum):"

            如果 (a - sum 小于等于 4)
                输入用户输入的数存入 b[i][0]
                sum 增加 b[i][0]
                b[i][1] 设置为 0
                sum 增加 b[i][1]
                round 增加 1
                如果 (a - sum 等于 0)
                    输出 "You lose!"
                    跳出循环

        输出 "Game over!"
        输出 "The following is the game process:"

        // 输出游戏过程
```

```
54.         for 每一轮:
55.             输出 "The (j + 1)th round: You entered b[j][0], the computer entered b[j][1]"
```

四、源代码

```
1.    #include <stdio.h>
2.    int func(int a) {
3.        if (a >= 4)
4.            return 4;
5.        else
6.            return a;
7.    }
8.    int  main(){
9.        int a = 21;
10.       int b[21][2];
11.       int sum = 0;
12.       int round = 0;
13.       printf("Please enter a number from 1 to 4:\n");
14.       for (int i = 0;sum <= 21; ++i) {
15.
16.           scanf("%d", &b[i][0]);
17.           if (b[i][0] > func(a - sum) || b[i][0] < 1)
18.           {
19.               printf("Please enter a number from 1 to %d:\n", func(a - sum));
20.               scanf("%d", &b[i][0]);
21.           }
22.           if(b[i][0] <= func(a - sum) || b[i][0] >= 1)
23.           {
24.               sum += b[i][0];
25.               b[i][1] = 5 - b[i][0];
26.               sum += b[i][1];
27.               round++;
28.               printf("The computer entered %d\n", b[i][1]);
```

```
                printf("The sum is %d\n", sum);
                printf("The remaining number is %d\n", a - sum);
                printf("Please enter a number from 1 to %d:\n", func(a - sum));

                }if(a - sum <= 4){
                scanf("%d", &b[i][0]);
                sum += b[i][0];
                b[i][1] = 0;
                sum += b[i][1];
                round++;
                if(a - sum == 0){
                    printf("You lose!\n");
                    break;
                }
            }

        }
        printf("Game over!\n");
        printf("The following is the game process:\n");
        for (int j = 0; j < round - 1; ++j) {
            printf("The %dth round: You entered %d, the computer entered %d\n", j + 1, b[j][0], b[j][1]);
        }
        return 0;
    }
```

四、实验结果及心得体会

在完成这个实验的过程中，我学到了很多关于数组、字符串处理和算法设计的知识。以下是我的一些心得体会：

数组的使用： 通过这个实验，我更深刻地理解了一维数组的定义、赋值、输入输出等基本操作。数组在处理有序数据集时非常方便，而在这个实验中，数组用于记录每轮游戏的信息，包括玩家和计算机每次取火柴

的数量。

字符串处理和输入验证： 实验中要求用户输入数字，并进行合法性验证。这促使我学会了如何使用字符串数组和字符处理函数，以及如何验证用户输入的合法性。在这里，通过函数 func 来限制用户输入的范围，确保其在 1 到 4 之间。

算法设计： 实验中设计了一个简单的算法，让计算机能够保持胜利。通过巧妙的计算，计算机可以根据当前的火柴数量选择最优的取法，确保在最后一轮将火柴留给玩家。这体现了算法设计在编程中的重要性。

循环结构的嵌套： 游戏的进行是通过嵌套的循环来实现的。外层循环用于整个游戏的进行，而内层循环用于处理玩家输入和计算机的选择。循环结构的嵌套是程序设计中常见的一种形式，实验中加深了我对其使用的理解。

总体而言，通过这个实验，我不仅学到了一些基础的 C 编程技能，还提高了对算法设计和循环结构的理解。这对于日后更复杂的编程任务和项目将是非常有帮助的。

实验五　函数

一　实验目的

1　掌握定义函数的方法；

2　掌握函数实参与形参的对应关系以及“值传递”的方式；

3　掌握函数的调用方法。

二　实验内容

题目：设 x 取值为区间[1,20]的整数，求函数 f(x)=x-sin(x)-cos(x)的最大值。要求：使用自定义函数实现 f（x）功能。

三、算法描述（描述形式可采用流程图或伪代码）

```
函数 func(a):
    b = a - sin(a) - cos(a)
    返回 b

主函数 main():
    max = 0
    循环 i 从 1 到 20:
        b = 调用 func(i)
        如果 b 大于 max:
            max = b
    输出 max
    返回 0
```

四、源代码

```c
#include <stdio.h>
#include <math.h>
int func(int a){
    int b = a - sin(a) - cos(a);
    return b;
}
int  main(){
```

```
        int max = 0;
        for (int i = 1; i < 21; ++i) {
            int b = func(i);
            if(b > max){
                max = b;
            }
        }
        printf("%d", max);
        return 0;
    }
```

五、实验结果及心得体会

实验五主要涉及定义函数、实参与形参的关系以及函数的调用方法。以下是一些可能的心得体会：

函数定义与调用： 通过实验，我学会了如何定义函数以及如何调用它们。在这个实验中，我们定义了函数 func(x)，并在主函数中通过调用这个函数来获取对应的函数值。

形参与实参： 实验中的函数 func 接受一个形参 x，而在主函数中调用这个函数时，我们传递具体的值作为实参。这展示了函数中形参与实参的关系，以及函数是如何通过实参获取值的。

值传递： 实验中的函数参数采用的是“值传递”的方式。这意味着在调用函数时，实参的值会被复制到函数的形参中。这也解释了为什么在函数中修改形参的值不会影响到实参。

循环与条件判断： 实验中使用循环从 1 到 20 迭代变量 x，并通过条件判断找到函数的最大值。这展示了如何使用循环和条件语句来实现对特定区间的计算。

数学函数的使用： 在实验中，使用了数学函数 sin 和 cos。这展示了如何在程序中使用标准库提供的数学函数来进行复杂的计算。

问题解决能力： 实验要求找到给定函数在特定区间内的最大值，这锻炼了我分析问题和设计解决方案的能力。

总的来说，通过这个实验，我不仅加深了对函数的理解，还学会了如何在程序中应用数学函数和进行问题求解。这为我今后的编程学习奠定了基础。

实验六　指针

一　实验目的

1、掌握指针的概念，会定义和使用指针变量；

2、能正确使用变量的指针和指向变量的指针变量；

3、能正确使用数组的指针和指向数组的指针变量；

4、能正确使用字符串的指针和指向字符串的指针变量；

5、正确使用指针作函数参数的方法。

二　实验内容

删除一维数组中相同的多余元素，即相同的值只保留一个

三、源代码

```c
#include <stdio.h>

void removeDuplicates(int arr[], int *size) {
    if (*size <= 1) {
        // 如果数组长度小于等于1，无需删除重复元素
        return;
    }

    int index = 1; // 从第二个元素开始比较
    for (int i = 1; i < *size; i++) {
        int j;
        for (j = 0; j < index; j++) {
            if (arr[i] == arr[j]) {
                break; // 找到重复元素，跳出内层循环
            }
        }

        if (j == index) {
            // 如果内层循环完成，说明当前元素不重复
            arr[index] = arr[i];
            index++;
        }
    }
```

```
24.
25.         *size = index; // 更新数组大小
26.     }
27.
28.     int main() {
29.         int arr[] = {1, 2, 3, 2, 4, 3, 5};
30.         int size = sizeof(arr) / sizeof(arr[0]);
31.
32.         // 打印原始数组
33.         printf("Original array: ");
34.         for (int i = 0; i < size; i++) {
35.             printf("%d ", arr[i]);
36.         }
37.         printf("\n");
38.
39.         // 调用函数删除重复元素
40.         removeDuplicates(arr, &size);
41.
42.         // 打印删除重复元素后的数组
43.         printf("Array after removing duplicates: ");
44.         for (int i = 0; i < size; i++) {
45.             printf("%d ", arr[i]);
46.         }
47.         printf("\n");
48.
49.         return 0;
50.     }
```

四、实验结果及心得体会

在实验六中，你学习了指针的概念以及指针在不同情境下的应用，包括变量指针、数组指针、字符串指针，以及指针作为函数参数的使用。在具体实验内容中，你需要删除一维数组中相同的多余元素，即相同的值只保留一个。

理解指针的概念： 在学习实验过程中，对指针的概念进行了深入的理解。指针是一个变量，其值为另一个变量的地址。

熟练使用指针和指针变量： 通过实验，掌握了定义和使用指针变量的方法，了解了指针与变量之间的关系，以及如何通过指针访问和修改变量的值。

数组指针和指向数组的指针： 学会了处理数组指针，尤其是在删除一维数组中相同元素的情境下，正确使用指向数组的指针变量，删除重复元素的算法也得到了巩固。

字符串指针的应用： 了解了字符串指针的用法，特别是在处理字符串时，通过指针更灵活地操作字符串的内容。

指针作为函数参数： 掌握了将指针作为函数参数传递的方法，这在处理数组和字符串时尤为重要，能够有效地修改原始数据。

代码调试和测试： 在实验过程中，对编写的代码进行了调试和测试，确保程序能够正确运行并得到预期的结果。

对内存的理解： 通过指针的使用，对计算机内存的工作原理有了更深入的认识，理解了指针是如何与内存地址关联的。

总体而言，通过实验六，不仅学到了关于指针的基础知识，还在实际问题中应用了这些知识。这有助于提高编程能力，更好地理解和利用指针在程序设计中的作用。

实验七　指针与字符串

一　实验目的

1、掌握字符数据、字符串在内存的存储结构。

2、掌握运用字符指针操作字符串数据的方法。

3、掌握常用的字符串处理函数。

二　实验内容

（1）自写字符串连接函数（2）实现的简单加解密系统

三、源代码

题目一

```
#include <stdio.h>
#define MAX 100
char * star1(char * star, const char * star2)
{

    char * temp;
    temp = star;
    while (*temp != '\0')
        temp++;
    while ((*temp++ = *star2++) != '\0')
        continue;
    return star;
}
char * s_gets(char * star, int n)
{
    char * ret_val;
    int i = 0;
    ret_val = fgets(star, n, stdin);
    if (ret_val)
    {
        while (star[i] != '\n' && star[i] != '\0')
            i++;
        if (star[i] == '\n')
            star[i] = '\0';
```

```
            else
                while (getchar() != '\n')
                    continue;
        }
        return ret_val;
    }
    int  main(){
        char star[MAX];
        char star2[MAX];
        s_gets(star, MAX);
        s_gets(star2, MAX);
        puts(star1(star, star2));
        return 0;
    }
```

题目二

```
    #include <stdio.h>
    #define MAX 100
    char * s_gets(char * star, int n);
    char * s_switch(char * star);
    char * s_gets(char * star, int n)
    {
        char * ret_val;
        int i = 0;
        ret_val = fgets(star, n, stdin);
        if (ret_val)
        {
            while (star[i] != '\n' && star[i] != '\0')
                i++;
            if (star[i] == '\n')
                star[i] = '\0';
            else
                while (getchar() != '\n')
                    continue;
        }
        return ret_val;
    }
    char * s_switch(char * star)
```

```
{
    int i = 0;
    while (star[i] != '\0')
    {
        if (star[i] >= 'a' && star[i] <= 'z')
            star[i] = star[i] - 31;
        else if (star[i] >= 'A' && star[i] <= 'Z')
            star[i] = star[i] + 31;
        i++;
    }
    return star;
}
//转换为大写并向后移动一个字符后输出
int  main(){
    char star[MAX];
    s_gets(star, MAX);
    puts(s_switch(star));
    return 0;
}
```

四、实验结果及心得体会

在实验中，你学到了关于字符数据和字符串在内存中存储结构的知识，以及如何使用字符指针操作字符串数据。具体而言，实验要求你自写字符串连接函数并实现一个简单的加解密系统。以下是一些心得体会：

字符数据和字符串内存结构： 通过实验，我深入了解了字符数据和字符串在内存中的存储结构。字符串以字符数组的形式存储，每个字符占用一个字节，而字符串以 null 结尾。

字符指针的应用： 通过实验内容，我学到了如何使用字符指针操作字符串数据。字符指针可以指向字符串的起始位置，通过指针进行遍历和操作，这使得对字符串的处理更加灵活。

字符串连接函数： 编写字符串连接函数的过程中，我理解了字符串连接的基本原理。通过循环遍历源字符串并逐个复制到目标字符串，同时需要注意 null 终止符的处理。

加解密系统的设计： 实现简单的加解密系统要求我考虑字符的移位和逆移位操作。这使我更加熟悉了字符的 ASCII 码以及如何通过运算对字符进行简单的加密和解密。

字符串处理函数的应用： 在实验中，我掌握了一些常用的字符串处理函数，如 strlen()、strcpy() 等，这些函数在实际编程中非常有用。

调试和测试： 编写完代码后，我进行了充分的调试和测试。通过输入不同的字符串和密钥，检查程序的输出结果，确保加解密系统能够正常运行。

对程序设计的思考： 这个实验让我思考了程序设计中的字符串处理和加解密的一些基本原理，也提高了我对字符串操作的实际能力。

总的来说，通过这个实验，我巩固了对字符串和字符指针的理解，学到了一些实用的字符串处理方法，并锻炼了编写简单加解密系统的能力。这对我后续在编程中处理字符串数据和设计简单的加解密功能都有帮助。

实验八 结构体与动态链表的建立

一 实验目的

1、掌握结构体的定义方法。

2、掌握动态内存管理函数。

3、掌握动态链表的建立。

二 实验内容

会员管理信息系统之会员信息的输入、输出。

三、源代码

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

// 定义会员结构体
struct Member {
    int id;
    char name[50];
    int age;
    struct Member* next;
};

// 函数声明
struct Member* createMember(int id, const char* name, int age);
void addMember(struct Member** head, struct Member* newMember);
void printMembers(const struct Member* head);
void freeMembers(struct Member* head);

int main() {
    struct Member* members = NULL;

    // 添加会员
    addMember(&members, createMember(1, "John Doe", 25));
```

```
24.         addMember(&members, createMember(2, "Jane Smith",
    30));
25.         addMember(&members, createMember(3, "Bob Johnson",
     22));
26.
27.         // 打印会员信息
28.         printf("Members:\n");
29.         printMembers(members);
30.
31.         // 释放内存
32.         freeMembers(members);
33.
34.         return 0;
35.     }
36.
37.     // 创建新会员
38.     struct Member* createMember(int id, const char* name,
    int age) {
39.         struct Member* newMember = (struct Member*)malloc(
    sizeof(struct Member));
40.         if (newMember == NULL) {
41.             perror("Memory allocation failed");
42.             exit(EXIT_FAILURE);
43.         }
44.
45.         newMember->id = id;
46.         strncpy(newMember->name, name, sizeof(newMember->n
    ame) - 1);
47.         newMember->age = age;
48.         newMember->next = NULL;
49.
50.         return newMember;
51.     }
52.
53.     // 添加新会员到链表
54.     void addMember(struct Member** head, struct Member* ne
    wMember) {
```

```
        if (*head == NULL) {
            *head = newMember;
        } else {
            struct Member* current = *head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newMember;
        }
    }

    // 打印会员信息
    void printMembers(const struct Member* head) {
        const struct Member* current = head;
        while (current != NULL) {
            printf("ID: %d, Name: %s, Age: %d\n", current->id, current->name, current->age);
            current = current->next;
        }
    }

    // 释放链表内存
    void freeMembers(struct Member* head) {
        struct Member* current = head;
        while (current != NULL) {
            struct Member* next = current->next;
            free(current);
            current = next;
        }
    }
```

四、实验结果及心得体会

在完成实验八，学习了结构体与动态链表的建立后，我对 C 语言中结构体和动态内存管理有了更深入的理解。

首先，通过实验，我掌握了结构体的定义方法。结构体允许我们将不同类型的数据组合在一起，形成一个新的数据类型。在这个实验中，我们定义了一个`Member`结构体，其中包含会员的 ID、姓名、年龄等信息。这样的

组织方式使得我们可以更方便地处理相关的数据。

其次，学习了动态内存管理函数，如`malloc`和`free`。动态内存分配允许我们在运行时请求和释放内存，而不是在编译时确定。这种灵活性使得我们可以更好地应对变化的数据需求。在实验中，我们通过`malloc`为每个会员创建动态内存，并通过`free`释放内存，有效地管理了内存资源。

最后，了解了动态链表的建立。链表是一种动态数据结构，允许我们按需插入、删除元素。通过动态链表，我们可以有效地组织和管理会员信息。在实验中，我们实现了向链表中添加新会员的函数，以及释放整个链表的函数。

在心得体会方面，通过实际动手完成这个实验，我深刻体会到了结构体和动态链表的实际应用价值。这些概念在软件工程中经常用于组织和管理复杂的数据结构。同时，学到的动态内存管理的知识也对我理解和解决内存泄漏等问题提供了帮助。

总的来说，这个实验对我深入理解 C 语言中结构体和动态内存管理提供了宝贵的经验，我相信这些知识将对我的编程能力产生长远的影响。

实验九 动态链表的添加、删除与修改

一 实验目的

1、掌握链表节点的添加、删除、查找和修改。

二 实验内容

会员管理信息系统之会员信息的修改、删除、添加、查找。

三、源代码

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

// 定义会员结构体
struct Member {
    int id;
    char name[50];
    int age;
    struct Member* next;
};

// 添加会员
void addMember(struct Member** head, int id, const char* name, int age) {
    struct Member* newMember = (struct Member*)malloc(sizeof(struct Member));
    newMember->id = id;
    strcpy(newMember->name, name);
    newMember->age = age;
    newMember->next = *head;
    *head = newMember;
}

// 删除会员
void deleteMember(struct Member** head, int id) {
    struct Member* current = *head;
    struct Member* previous = NULL;
```

```
27.
28.       while (current != NULL && current->id != id) {
29.           previous = current;
30.           current = current->next;
31.       }
32.
33.       if (current != NULL) {
34.           if (previous == NULL) {
35.               *head = current->next;
36.           } else {
37.               previous->next = current->next;
38.           }
39.           free(current);
40.       }
41.   }
42.
43.   // 修改会员信息
44.   void updateMember(struct Member* head, int id, const c
   har* newName, int newAge) {
45.       struct Member* current = head;
46.
47.       while (current != NULL && current->id != id) {
48.           current = current->next;
49.       }
50.
51.       if (current != NULL) {
52.           strcpy(current->name, newName);
53.           current->age = newAge;
54.       }
55.   }
56.
57.   // 查找会员
58.   struct Member* findMember(struct Member* head, int id)
    {
59.       struct Member* current = head;
60.
61.       while (current != NULL && current->id != id) {
```

```
            current = current->next;
        }

        return current;
    }

    // 显示所有会员信息
    void displayMembers(struct Member* head) {
        struct Member* current = head;

        while (current != NULL) {
            printf("ID: %d, Name: %s, Age: %d\n", current->id, current->name, current->age);
            current = current->next;
        }
    }

    // 释放链表内存
    void freeMembers(struct Member* head) {
        struct Member* current = head;
        struct Member* next;

        while (current != NULL) {
            next = current->next;
            free(current);
            current = next;
        }
    }

    int main() {
        struct Member* members = NULL;

        // 添加几个示例会员
        addMember(&members, 1, "Alice", 25);
        addMember(&members, 2, "Bob", 30);
        addMember(&members, 3, "Charlie", 22);

```

```
    // 显示所有会员信息
    printf("All Members:\n");
    displayMembers(members);

    // 查找并修改会员信息
    struct Member* foundMember = findMember(members, 2)
;
    if (foundMember != NULL) {
        printf("\nUpdating Member with ID 2:\n");
        updateMember(members, 2, "Bob Updated", 35);
        displayMembers(members);
    }

    // 删除一个会员
    printf("\nDeleting Member with ID 1:\n");
    deleteMember(&members, 1);
    displayMembers(members);

    // 释放内存
    freeMembers(members);

    return 0;
}
```

四、实验结果及心得体会

在完成实验八（结构体与动态链表的建立）的过程中，我深入学习了结构体的定义和动态链表的建立。这次实验帮助我更好地理解了如何使用结构体组织数据，并通过动态链表实现对数据的灵活管理。

首先，我学会了如何定义结构体，将不同类型的数据组织在一起，形成更为复杂的数据结构。结构体的使用使得我能够更加清晰地表示实际问题中的数据关系，提高了代码的可读性和可维护性。

其次，通过实验中对动态链表的建立，我深入理解了指针的概念和内存管理。动态链表相较于静态数组更加灵活，能够根据需要动态地分配和释放内存。这使得我们在处理数据时更具有扩展性，能够应对不确定数量的数据。

实验中，我学到了如何通过 `malloc` 函数动态地分配内存，并通过指针进行访问和操作。同时，在释放内存时使用 `free` 函数，避免内存泄漏问题。这对于实际项目中大规模数据的处理是非常重要的。

在实验中编写的简单会员管理系统，通过结构体表示会员信息，通过动态链表实现对会员的增删查改，使我更好地理解了结构体和链表的使用场景。这不仅对编程技能的提升有所帮助，也拓宽了我对数据结构的认识。

总的来说，通过这次实验，我深刻理解了结构体和动态链表的使用，这对于日后更复杂的数据管理和结构设计提供了坚实的基础。同时，对于指针和内存管理的掌握也使我更加熟练地处理内存资源，写出更健壮的程序。