

《计算机程序设计》实验报告

姓名： 宋亦寒 学号： PB25000200 实验日期： 2025 年 12 月 15 日

实验名称	C 程序设计入门
<div>一、 上机程序（有注释）</div> <div>1. 实验 P177-2-子数组的查找</div> <pre>#include <stdio.h> int FindMaxThree(int *a, int n) { // 检查数组长度 n 是否在有效范围[3, 100]内 if (n<3 n>100) return -1; // 长度无效则返回-1 // 初始化最大三元素和为前三个元素的和 int max_sum=*a+*(a+1)+*(a+2); // 初始化最大和子数组的起始索引为 0 int max_index=0; // 遍历所有可能的连续三个元素的子数组，i 是起始索引 for (int i=0; i<=n-3; i++){ // 计算当前连续三个元素的和 int current_sum=*(a+i)+*(a+i+1)+*(a+i+2); // 如果当前和大于最大和 if (current_sum>max_sum){ max_sum=current_sum; // 更新最大和 max_index=i; // 更新最大和子数组的起始索引 } } return max_index; // 返回最大和子数组的起始索引 } int main() { int n; // 读取数组长度 n scanf("%d", &n); int a[100]; // 读取数组元素 for (int i=0; i<n; i++){ scanf("%d", a+i); // 等价于 scanf("%d", &a[i]); } // 调用函数并打印结果（最大和子数组的起始索引） printf("%d", FindMaxThree(a, n)); }</pre>	

2. 实验 P177-4-求两个向量的内积

```
#include <stdio.h>
// 计算两个 n 维向量 a 和 b 的内积
double InnerProduct(double *a, double *b, int n)
{
    double sum=0.0; // 初始化内积和为 0.0
    // 遍历向量的每个元素
    for (int i=0; i<n; i++) {
        // 累加对应元素的乘积
        sum+=(*a+i)*(*b+i);
    }
    return sum; // 返回内积结果
}

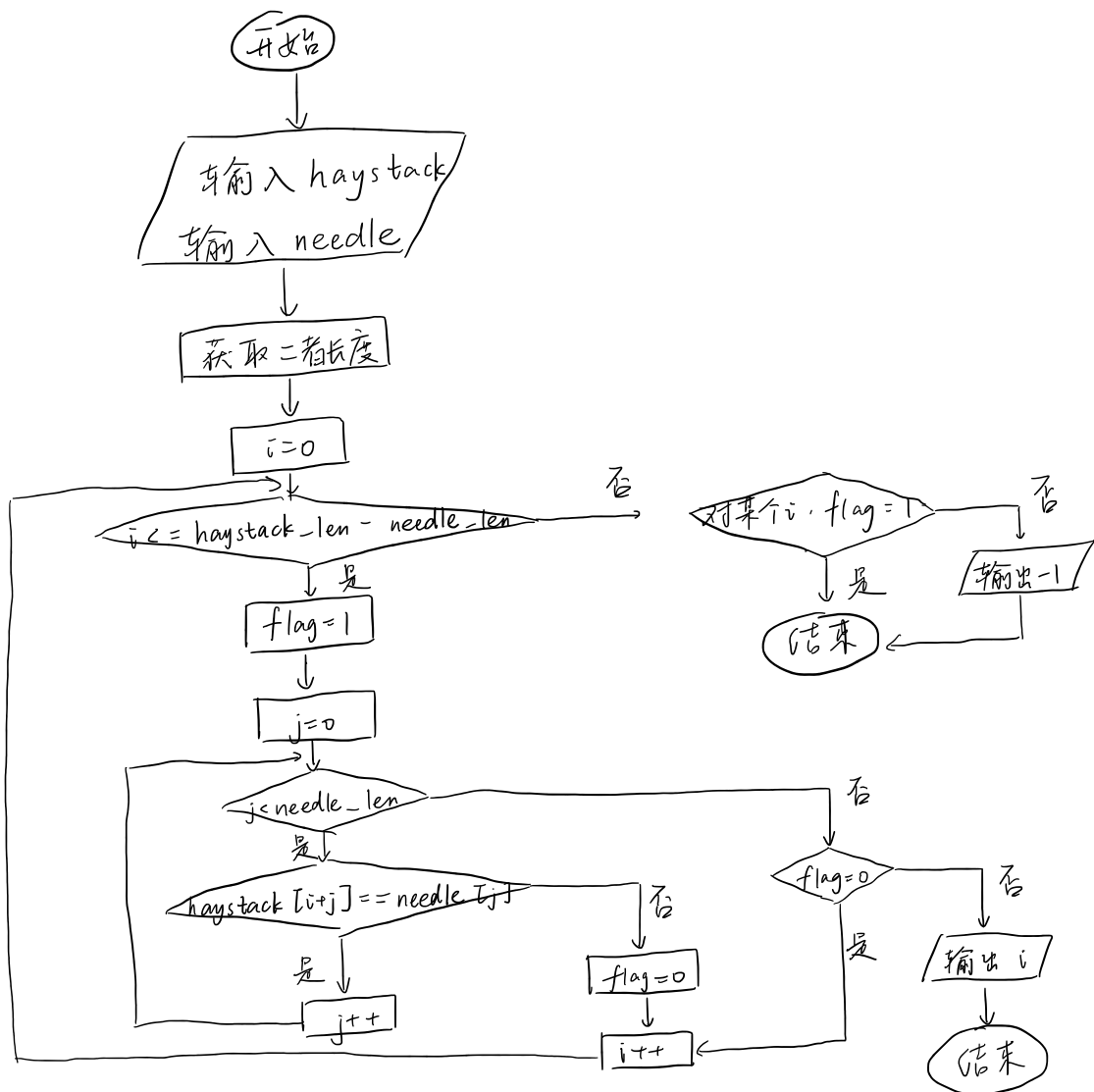
int main()
{
    int n;
    // 读取向量维度 n
    scanf("%d", &n);
    // 定义两个最大长度为 50 的双精度浮点数数组作为向量 a 和 b
    double a[50], b[50];
    // 读取向量 a 的元素
    for (int i=0; i<n; i++)
        scanf("%lf", a+i); // 等价于 scanf("%lf", &a[i]);
    // 读取向量 b 的元素
    for (int i=0; i<n; i++)
        scanf("%lf", b+i); // 等价于 scanf("%lf", &b[i]);
    // 调用函数计算内积并打印结果
    printf("%lf", InnerProduct(a, b, n));
}
```

3. CS1003L. 03-第九次课堂作业 T4

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

// 查找子字符串 needle 在主字符串 haystack 中首次出现的位置（自定义实现 strstr 函数）
char *udf_strstr(char *haystack, char *needle)
{
    // 获取子字符串的长度
    int needle_len=strlen(needle);
    // 获取主字符串的长度
    int haystack_len=strlen(haystack);
    // 外层循环：遍历主字符串中所有可能的起始位置 i
    // 循环条件保证剩余长度足够容纳 needle
    for (int i=0; i<=haystack_len-needle_len; i++){
        int flag=1; // 匹配标志，1 表示当前认为匹配成功
        // 内层循环：比较从 haystack[i] 开始的子串和 needle
        for (int j=0; j<needle_len; j++){
            // 比较对应位置的字符是否相等
            if (*(haystack+i+j) != *(needle+j)){
                flag=0; // 字符不匹配，设置标志为 0
                break; // 跳出内层循环，进行下一个起始位置 i 的比较
            }
        }
        // 如果 flag 仍为 1，说明内层循环中所有字符都匹配成功
        if (flag==1)
            return (haystack+i); // 返回匹配成功的起始位置指针
    }
    return NULL; // 如果遍历完所有位置都没有匹配，则返回 NULL
}

int main()
{
    char haystack[1000];
    char needle[1000];
    // 读取主字符串
    scanf("%s", haystack);
    // 读取子字符串
    scanf("%s", needle);
    // 调用自定义函数进行查找
    char *pos = udf_strstr(haystack, needle);
    // 打印结果：如果找到，打印起始位置的偏移量；否则打印 -1
    // %td 用于打印 ptrdiff_t 类型（指针相减的结果）
    printf("%td\n", pos ? (pos - haystack) : -1);
}
```



4. 实验基础版 P161-2-仿真生命

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

```
// 计算给定坐标 (r, c) 处细胞的活着的 ('*') 邻居数量
```

```
int living_neighbours_count(char grid[40][40], int r, int c)
```

```
{
```

```
    int count=0;
```

```
    // 遍历以 (r, c) 为中心的 3x3 区域
```

```
    for (int i=-1; i<=1; i++){
```

```
        for (int j=-1; j<=1; j++){
```

```
            // 跳过细胞自身 (i=0, j=0)
```

```
            if (i==0 && j==0) continue;
```

```
            // 检查邻居坐标 (r+i, c+j) 是否在 40x40 网格边界内
```

```
            if (r+i>=0 && r+i<40 && c+j>=0 && c+j<40) {
```

```
// 如果邻居是活着的 ('*')
if (grid[r+i][c+j] == '*') {
    count++; // 活着的邻居计数加一
}
}
}
return count; // 返回活着的邻居总数
}

// 模拟康威生命游戏 n 代演化，并打印最终网格状态
void state_change(char grid[40][40], int n)
{
    // 定义一个临时网格用于存储下一代状态，避免在计算时影响本代状态
    char next_state[40][40];
    // 循环 n 次，进行 n 代演化
    for (int generation=0; generation<n; generation++){

        // 遍历整个 40x40 网格中的每个细胞
        for (int r=0; r<40; r++){
            for (int c=0; c<40; c++){
                // 获取当前细胞的活着的邻居数量
                int living_neighbours=living_neighbours_count(grid, r, c);
                // 获取当前细胞的状态
                char current_state=grid[r][c];

                // 应用康威生命游戏的规则
                if (current_state=='*'){ // 规则 1 & 2: 活细胞 (*)
                    // 活细胞邻居数为 2 或 3 时，存活到下一代
                    if (living_neighbours==2 || living_neighbours==3){
                        next_state[r][c]='*';
                    } else {
                        // 活细胞邻居数少于 2 或多于 3 时，死亡 ('-')
                        next_state[r][c]='-';
                    }
                } else { // 规则 3: 死细胞 (-)
                    // 死细胞邻居数恰好为 3 时，复活 ('*')
                    if (living_neighbours==3){
                        next_state[r][c]='*';
                    } else {
                        // 否则保持死亡状态 ('-')
                        next_state[r][c]='-';
                    }
                }
            }
        }
    }
}
```

```
    }
}

// 将计算出的下一代状态 next_state 复制回当前网格 grid
for (int r=0; r<40; r++) {
    for (int c=0; c<40; c++) {
        grid[r][c]=next_state[r][c];
    }
}

// 打印 n 代演化后的最终网格状态
for (int r=0; r<40; r++) {
    for (int c=0; c<40; c++) {
        printf("%c", grid[r][c]);
    }
    printf("\n"); // 每行结束后换行
}

}

int main()
{
    char grid[40][40];
    // 读取 40x40 的初始网格状态
    for (int r=0; r<40; r++) {
        for (int c=0; c<40; c++) {
            char ch=getchar(); // 读取字符
            // 只接受 '*' 或 '-' 作为有效输入
            if (ch=='*' || ch=='-') {
                grid[r][c]=ch; // 存入网格
                c++; // 移动到下一列
            }
        }
    }
    int n;
    // 读取要模拟的代数 n
    scanf("%d", &n);
    // 执行状态演化并打印最终结果
    state_change(grid, n);
}
```

二、 调试中的问题及解决方法（字数不限）

1. 题目二：向量内积

问题：未注意到函数声明中第二个指针参数的类型缺失：`double InnerProduct(double *a, *b, int n)`

备注：C 语言要求每个参数都必须单独指定类型

2. 题目三：字符串查找 `udf_strstr`

问题：字符串输入循环错误：`for (int i=0; i<1000; i++) scanf("%s", haystack+i)`

解决方案：`%s` 会读取一整个字符串，因此只需调用一次 `scanf("%s", haystack)` 来读取

问题：`printf` 语句中出现警告：`printf("%d\n", pos?(pos-haystack):-1);`，编译器提示 `pos-haystack` 的类型为 `long int (ptrdiff_t)`，与 `%d` 不匹配

解决方案：将格式说明符 `%d` 更改为正确的指针差值类型说明符 `%td`，或进行强制类型转换 `(int)(pos-haystack)`