Lec 15 Data Structures

1. 矩阵转置

核心内容

- **功能**：从文件读取矩阵并转置后输出到新文件。
- **步骤**：
  1. **读取文件**：打开输入文件，读取矩阵的行数 `m` 和列数 `n`。
  2. **存储矩阵**：用二维数组 `matrix[m][n]` 存储矩阵元素。
  3. **转置操作**：将矩阵的行列互换，写入输出文件。
- **代码关键点**：

```
// 读取矩阵
fscanf(in, "%d%d", &m, &n);  // 读取矩阵尺寸
int matrix[m][n];
for (i = 0; i < m; i++)
    for (j = 0; j < n; j++)
        fscanf(in, "%d", &matrix[i][j]);

// 写入转置矩阵
fprintf(out, "%d %d\n", n, m);  // 转置后行列数交换
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
        fprintf(out, "%d ", matrix[j][i]);
    fprintf(out, "\n");
}
```

- **注意事项**：
  - 文件操作需检查是否成功打开（`fopen` 返回 `NULL` 表示失败）。
  - 转置时注意行列索引的交换（原 `matrix[i][j]` 变为 `matrix[j][i]`）。

2. 数据类型

基本数据类型

- **整型**：`int`（4字节）、`short`（2字节）、`long`（8字节）。
- **浮点型**：`float`（4字节，精度约7位）、`double`（8字节，精度约15位）。
- **字符型**：`char`（1字节，表示 ASCII 字符）。

复合数据类型

- **数组**：连续内存空间存储同类型元素，如 `int arr[10]`。

- **指针**：存储变量的地址，如 `int *p = &x`。
- **结构体**：聚合多个不同类型的数据（如学生信息）。
- **联合体**：同一内存块存储不同类型数据（共享内存）。
- **枚举**：定义一组命名的整型常量。

3. 结构体（Structures）

定义与使用

- **语法**：

```
struct Student {
    char name[50];
    int id;
    float gpa;
};
```

- **变量声明**：

```
struct Student student;  // 单个结构体变量，在此之后可以用student来替代struct Student这个东西
struct Student class[100];  // 结构体数组
```

- **访问**：

```
strcpy(s1.name, "Alice");  // 字符数组需用 strcpy
s1.id = 1001;
s1.gpa = 3.9;
```

内存对齐

- **规则**：编译器会根据成员类型对齐内存，可能导致结构体实际大小大于理论值。
- **示例**：

```
struct Struct1 {
    char c;     // 1字节
    int i;      // 4字节（对齐到4字节边界）
    char c2;    // 1字节
};  // 总大小为 12 字节（1 + 3填充 + 4 + 1 + 3填充）
```

- **优化方法**：按成员大小从大到小排列，减少填充。

4. typedef 关键字

- **用途**：为数据类型定义别名，提高代码可读性。
- **示例**：

```
typedef struct Student Student;  // 结构体别名
typedef int Score;               // 基本类型别名
```

- **结构体简化**：

```
typedef struct {
    char name[50];
    int id;
} Student;  // 直接使用 Student 声明变量
```

5. 结构体数组

- **定义**：

```
Student students[100];  // 存储100个学生信息
```

- **初始化**：

```
Student class[] = {
    {"Alice", 1001, 3.9},
    {"Bob", 1002, 3.8}
};
```

- **遍历与操作**：

```
for (int i = 0; i < 100; i++) {
    printf("Name: %s, GPA: %.2f\n", students[i].name, students[i].gpa);
}
```

6. 文件处理

读取学生记录

- **步骤**：
  1. 打开文件并检查错误。
  2. 读取数据到结构体数组。
  3. 关闭文件。
- **代码示例**：

```c
int load_data(char filename[], Student s[]) {
    FILE *in = fopen(filename, "r");
    if (in == NULL) return -1;
    char temp[256];
    fgets(temp, 256, in);  // 跳过标题行
    int n = 0;
    while (fscanf(in, "%s %d %f", s[n].name, &s[n].id, &s[n].gpa) != EOF)
        n++;
    fclose(in);
    return n;
}
```

7. 排序（按 GPA）

- **算法**：冒泡排序（简单但效率低）。
- **实现**：

```c
void sort_by_gpa(Student s[], int n) {
    int flag = 1;
    while (flag) {
        flag = 0;
        for (int i = 0; i < n-1; i++) {
            if (s[i].gpa > s[i+1].gpa) {
                swap(&s[i], &s[i+1]);
                flag = 1;
            }
        }
    }
}

void swap(Student *a, Student *b) {
    Student temp = *a;
    *a = *b;
    *b = temp;
}
```

8. 指针与结构体

- **定义指针**：

```c
Student *ptr = &s1;  // 指向结构体变量
Student *arr_ptr = students;  // 指向结构体数组
```

- **访问成员**：

```c
printf("%s", ptr->name);  // 等价于 (*ptr).name
ptr->gpa = 4.0;
```

9. 嵌套结构体

- **定义**：

```
typedef struct {
    char first[20];
    char last[20];
} Name;

typedef struct {
    Name name;
    int id;
    float gpa;
} Student;
```

- **使用**：

```
Student s;
strcpy(s.name.first, "Alice");
strcpy(s.name.last, "Smith");
```

10. 联合体（Union）

- **特点**：所有成员共享同一块内存，大小为最大成员的大小。
- **示例**：

```
union Data {
    int i;
    float f;
    char str[20];
};
```

- **用途**：节省内存，适用于不同数据类型的临时存储。

11. 枚举（Enumeration）

- **定义**：

```
enum Month { JAN, FEB, MAR };  // JAN=0, FEB=1, MAR=2
enum Month current = FEB;
```

- **自定义值**：

```
enum Month { JAN=1, FEB, MAR };  // JAN=1, FEB=2, MAR=3
```

12. 内存布局

- **运行时栈**：存储局部变量和函数调用帧。
- **堆**：动态内存分配（`malloc`, `free`）。
- **静态区**：全局变量和静态变量。

关键注意事项

1. **结构体内存对齐**：通过 `#pragma pack` 可手动调整对齐方式。
2. **文件操作错误处理**：始终检查 `fopen` 返回值。
3. **指针操作安全**：避免野指针（初始化为 `NULL`，使用前检查）。
4. **结构体数组传递**：函数参数中传递数组指针，避免拷贝开销。