

C 语言高级特性手册

January 9, 2026

1 动态内存分配

1.1 动态内存分配概述

在 C 语言中，动态内存分配是在程序运行时分配内存的方法，与静态内存分配（编译时确定大小）相对。

1.1.1 为什么需要动态内存分配

- **未知数据大小**: 程序运行时才能确定需要多少内存
- **内存高效利用**: 按需分配，避免浪费
- **数据结构灵活性**: 链表、树等动态数据结构的需要
- **生命周期控制**: 手动控制内存的分配和释放

1.2 内存分配函数

1.2.1 malloc 函数

```
1 void* malloc(size_t size);
```

Listing 1: malloc 函数原型

- **功能**: 分配指定字节数的内存块
- **参数**: 需要分配的字节数
- **返回值**: 成功返回指向分配内存的指针，失败返回 NULL
- **特点**: 分配的内存不会初始化，内容是随机的

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     // 分配10个整数的空间
6     int *arr = (int*)malloc(10 * sizeof(int));
7
8     if(arr == NULL) {
9         printf("内存分配失败!\n");
10        return 1;
11    }
12
13    // 使用分配的内存
14    for(int i = 0; i < 10; i++) {
15        arr[i] = i * 10;
16    }
17
18    // 输出
19    for(int i = 0; i < 10; i++) {
```

```

20     printf("arr[%d] = %d\n", i, arr[i]);
21 }
22
23 // 释放内存
24 free(arr);
25
26 return 0;
27 }
```

Listing 2: malloc 使用示例

1.2.2 calloc 函数

```
1 void* calloc(size_t num, size_t size);
```

Listing 3: calloc 函数原型

- **功能:** 分配指定数量、指定大小的内存块
- **参数:**
 - num: 元素数量
 - size: 每个元素的大小
- **返回值:** 成功返回指向分配内存的指针, 失败返回 NULL
- **特点:** 分配的内存会初始化为 0

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     // 分配10个整数的空间, 并初始化为0
6     int *arr = (int*)calloc(10, sizeof(int));
7
8     if(arr == NULL) {
9         printf("内存分配失败!\n");
10        return 1;
11    }
12
13    printf("calloc分配的内存已初始化为0:\n");
14    for(int i = 0; i < 10; i++) {
15        printf("arr[%d] = %d\n", i, arr[i]);
16    }
17
18    free(arr);
19    return 0;
20 }
```

Listing 4: calloc 使用示例

1.2.3 realloc 函数

```
1 void* realloc(void* ptr, size_t new_size);
```

Listing 5: realloc 函数原型

- **功能:** 重新分配内存块的大小
- **参数:**
 - ptr: 指向之前分配的内存块的指针
 - new_size: 新的内存块大小
- **返回值:** 成功返回指向新内存块的指针, 失败返回 NULL
- **特点:**
 - 如果新大小小于原大小, 多余部分被丢弃
 - 如果新大小大于原大小, 新增部分不会被初始化
 - 可能返回一个新的指针地址

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     // 初始分配5个整数的空间
6     int *arr = (int*)malloc(5 * sizeof(int));
7
8     if(arr == NULL) {
9         printf("内存分配失败!\n");
10    return 1;
11 }
12
13 // 初始化
14 for(int i = 0; i < 5; i++) {
15     arr[i] = i + 1;
16 }
17
18 // 扩展到10个整数
19 int *new_arr = (int*)realloc(arr, 10 * sizeof(int));
20
21 if(new_arr == NULL) {
22     printf("内存重新分配失败!\n");
23     free(arr); // 释放原始内存
24     return 1;
25 }
26
27 arr = new_arr; // 使用新的指针
28
29 // 初始化新增的部分
30 for(int i = 5; i < 10; i++) {
```

```

31     arr[i] = i + 1;
32 }
33
34 // 输出所有元素
35 for(int i = 0; i < 10; i++) {
36     printf("arr[%d] = %d\n", i, arr[i]);
37 }
38
39 free(arr);
40 return 0;
41 }
```

Listing 6: realloc 使用示例

1.2.4 free 函数

```
1 void free(void* ptr);
```

Listing 7: free 函数原型

- **功能:** 释放之前分配的内存
- **参数:** 指向要释放的内存块的指针
- **注意事项:**
 - 只能释放由 malloc、calloc、realloc 分配的内存
 - 不能重复释放同一块内存
 - 释放后应将指针设为 NULL，避免悬空指针

1.3 malloc vs calloc vs realloc 对比

malloc	calloc	realloc
分配未初始化内存 参数: 字节数	分配并初始化为 0 参数: 数量 × 大小	调整已分配内存大小 参数: 原指针 + 新大小
内存内容随机 效率较高	内存内容全为 0 效率较低 (需初始化)	新内存内容可能改变 可能需要内存复制

Table 1: 内存分配函数对比

1.4 动态内存分配的最佳实践

1.4.1 正确的内存管理流程

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     // 1. 分配内存
7     char *str = (char*)malloc(100 * sizeof(char));
8
9     // 2. 检查是否分配成功
10    if(str == NULL) {
11        fprintf(stderr, "内存分配失败!\n");
12        return 1;
13    }
14
15    // 3. 使用内存
16    strcpy(str, "Hello, Dynamic Memory!");
17    printf("%s\n", str);
18
19    // 4. 释放内存
20    free(str);
21
22    // 5. 将指针设为NULL (避免悬空指针)
23    str = NULL;
24
25    return 0;
26 }
```

Listing 8: 内存管理最佳实践

1.4.2 常见错误与避免方法

```
1 // 错误1: 使用未初始化的指针
2 int *ptr;
3 *ptr = 10; // 错误: ptr未指向有效内存
4
5 // 正确做法:
6 int *ptr = (int*)malloc(sizeof(int));
7 if(ptr != NULL) {
8     *ptr = 10;
9 }
10
11 // 错误2: 内存泄漏
12 void func() {
13     int *ptr = malloc(100 * sizeof(int));
14     // 使用ptr...
15     // 忘记 free(ptr); // 内存泄漏!
16 }
17
18 // 正确做法:
19 void func() {
```

```
20     int *ptr = malloc(100 * sizeof(int));
21     if(ptr != NULL) {
22         // 使用ptr...
23         free(ptr);
24     }
25 }
26
27 // 错误3：重复释放
28 int *ptr = malloc(sizeof(int));
29 free(ptr);
30 free(ptr); // 错误：重复释放！
31
32 // 正确做法：
33 int *ptr = malloc(sizeof(int));
34 if(ptr != NULL) {
35     free(ptr);
36     ptr = NULL; // 设为NULL
37 }
38
39 // 再次释放时检查
40 if(ptr != NULL) {
41     free(ptr);
42 }
43
44 // 错误4：使用已释放的内存
45 int *ptr = malloc(sizeof(int));
46 free(ptr);
47 *ptr = 20; // 错误：ptr指向的内存已释放！
48
49 // 正确做法：
50 int *ptr = malloc(sizeof(int));
51 if(ptr != NULL) {
52     *ptr = 20;
53     // 使用...
54     free(ptr);
55     ptr = NULL; // 设为NULL，避免误用
56 }
```

Listing 9: 常见内存错误示例

2 结构体 (Struct)

2.1 结构体的定义与声明

2.1.1 什么是结构体

结构体是 C 语言中一种用户自定义的数据类型，它允许我们将不同类型的数据组合成一个整体。

2.1.2 定义结构体

```

1 // 定义结构体类型
2 struct Student {
3     char name[50];          // 姓名
4     int age;                // 年龄
5     float score;            // 成绩
6     char gender;            // 性别
7 };// 注意：这里需要分号
8
9 // 定义结构体的同时声明变量
10 struct Point {
11     int x;
12     int y;
13 } p1, p2; // 同时声明两个Point类型的变量
14
15 // 使用typedef创建别名
16 typedef struct Date {
17     int year;
18     int month;
19     int day;
20 } Date; // 现在可以直接使用Date作为类型名

```

Listing 10: 结构体定义

2.1.3 结构体变量声明

```

1 // 方法1：先定义类型，再声明变量
2 struct Student stu1, stu2;
3
4 // 方法2：定义类型的同时声明变量
5 struct Employee {
6     int id;
7     char name[50];
8     float salary;
9 } emp1, emp2;
10
11 // 方法3：使用typedef别名
12 Date today, tomorrow;

```

Listing 11: 结构体变量声明

2.2 结构体的初始化与访问

2.2.1 结构体初始化

```

1 // 方法1：声明时初始化
2 struct Student stu1 = {"张三", 20, 85.5, 'M'};
3
4 // 方法2：部分初始化（未指定的成员自动初始化为0或空）
5 struct Student stu2 = {"李四"}; // age=0, score=0.0, gender='0'
6

```

```
7 // 方法3：指定成员初始化（C99标准）
8 struct Student stu3 = {
9     .name = "王五",
10    .age = 22,
11    .score = 90.5
12};
13
14 // 方法4：数组初始化
15 struct Student class[3] = {
16     {"张三", 20, 85.5, 'M'},
17     {"李四", 21, 78.0, 'F'},
18     {"王五", 19, 92.0, 'M'}
19};
```

Listing 12: 结构体初始化

2.2.2 结构体成员访问

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct Student {
5     char name[50];
6     int age;
7     float score;
8 };
9
10 int main() {
11     struct Student stu;
12
13     // 使用点运算符(.)访问结构体成员
14     strcpy(stu.name, "张三");
15     stu.age = 20;
16     stu.score = 85.5;
17
18     // 读取结构体成员
19     printf("姓名: %s\n", stu.name);
20     printf("年龄: %d\n", stu.age);
21     printf("成绩: %.1f\n", stu.score);
22
23     // 结构体数组
24     struct Student class[3];
25     for(int i = 0; i < 3; i++) {
26         printf("请输入第%d个学生的信息:\n", i+1);
27         printf("姓名: ");
28         scanf("%s", class[i].name);
29         printf("年龄: ");
30         scanf("%d", &class[i].age);
31         printf("成绩: ");
32         scanf("%f", &class[i].score);
33     }
34 }
```

```
34     return 0;
35 }
36 }
```

Listing 13: 结构体成员访问

2.3 结构体指针

2.3.1 指向结构体的指针

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 typedef struct Student {
6     char name[50];
7     int age;
8     float score;
9 } Student;
10
11 int main() {
12     Student stu = {"张三", 20, 85.5};
13     Student *p = &stu; // 指向结构体的指针
14
15     // 通过指针访问结构体成员 (两种方法)
16     printf("方法1 - 使用(*p).成员: %s\n", (*p).name);
17     printf("方法2 - 使用p->成员: %d\n", p->age);
18
19     // 动态分配结构体内存
20     Student *p2 = (Student*)malloc(sizeof(Student));
21     if(p2 != NULL) {
22         strcpy(p2->name, "李四");
23         p2->age = 21;
24         p2->score = 78.0;
25
26         printf("动态分配: %s, %d, %.1f\n",
27               p2->name, p2->age, p2->score);
28
29         free(p2); // 释放内存
30     }
31
32     return 0;
33 }
```

Listing 14: 结构体指针

2.3.2 结构体作为函数参数

```
1 #include <stdio.h>
2 #include <string.h>
3
```

```

4 typedef struct {
5     char name[50];
6     int age;
7 } Person;
8
9 // 值传递 (复制整个结构体)
10 void printPersonByValue(Person p) {
11     printf("姓名: %s, 年龄: %d\n", p.name, p.age);
12 }
13
14 // 指针传递 (更高效)
15 void printPersonByPointer(const Person *p) {
16     printf("姓名: %s, 年龄: %d\n", p->name, p->age);
17 }
18
19 // 修改结构体内容
20 void updatePerson(Person *p, const char *name, int age) {
21     strcpy(p->name, name);
22     p->age = age;
23 }
24
25 int main() {
26     Person p1 = {"张三", 20};
27
28     printPersonByValue(p1);      // 值传递
29     printPersonByPointer(&p1);   // 指针传递
30
31     updatePerson(&p1, "李四", 25);
32     printPersonByPointer(&p1);
33
34     return 0;
35 }
```

Listing 15: 结构体作为函数参数

2.4 结构体嵌套

```

1 #include <stdio.h>
2
3 // 定义日期结构体
4 typedef struct {
5     int year;
6     int month;
7     int day;
8 } Date;
9
10 // 定义学生结构体, 嵌套日期结构体
11 typedef struct {
12     char name[50];
13     Date birthday; // 嵌套结构体
14     float score;
```

```

15 } Student;
16
17 // 定义班级结构体，包含学生数组
18 typedef struct {
19     Student students[50];
20     int count;
21     char className[20];
22 } Class;
23
24 int main() {
25     // 创建学生
26     Student stu = {
27         "张三",
28         {2000, 5, 15}, // 初始化嵌套结构体
29         85.5
30     };
31
32     printf("学生: %s\n", stu.name);
33     printf("生日: %d年%d月%d日\n",
34             stu.birthday.year,
35             stu.birthday.month,
36             stu.birthday.day);
37
38     // 创建班级
39     Class csClass;
40     strcpy(csClass.className, "计算机1班");
41     csClass.count = 0;
42
43     // 添加学生
44     strcpy(csClass.students[0].name, "李四");
45     csClass.students[0].birthday.year = 2001;
46     csClass.students[0].birthday.month = 8;
47     csClass.students[0].birthday.day = 20;
48     csClass.students[0].score = 90.0;
49     csClass.count++;
50
51     return 0;
52 }

```

Listing 16: 结构体嵌套

2.5 结构体大小与对齐

```

1 #include <stdio.h>
2
3 // 结构体大小受内存对齐影响
4 struct Example1 {
5     char a;      // 1字节
6     int b;       // 4字节
7     char c;      // 1字节
8 }; // 总大小可能是12字节 (取决于编译器对齐)

```

```

9
10 struct Example2 {
11     int b;          // 4字节
12     char a;         // 1字节
13     char c;         // 1字节
14 };// 总大小可能是8字节 (内存对齐后)
15
16 // 使用#pragma pack改变对齐方式
17 #pragma pack(push, 1) // 设置为1字节对齐
18 struct PackedStruct {
19     char a;
20     int b;
21     char c;
22 };// 总大小为6字节
23 #pragma pack(pop) // 恢复默认对齐
24
25 int main() {
26     printf("Example1大小: %lu字节\n", sizeof(struct Example1));
27     printf("Example2大小: %lu字节\n", sizeof(struct Example2));
28     printf("PackedStruct大小: %lu字节\n", sizeof(struct PackedStruct));
29 }
30
31 return 0;
}

```

Listing 17: 结构体大小与内存对齐

3 链表 (Linked List)

3.1 链表的基本概念

3.1.1 什么是链表

链表是一种动态数据结构，由一系列节点组成，每个节点包含数据和指向下一个节点的指针。

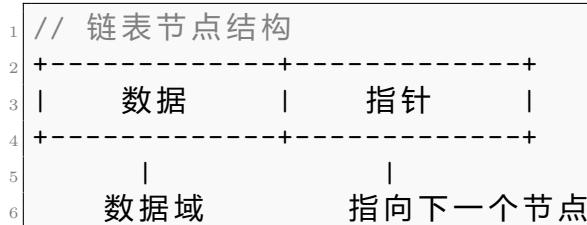


Figure 1: 链表节点结构示意图

3.1.2 链表与数组的比较

3.2 单向链表的实现

3.2.1 链表节点定义

链表	数组
动态分配内存，大小可变	静态分配内存，大小固定
插入删除效率高 ($O(1)$)	插入删除效率低 ($O(n)$)
内存不连续	内存连续
访问元素效率低 ($O(n)$)	随机访问效率高 ($O(1)$)
需要额外空间存储指针	不需要额外指针空间

Table 2: 链表与数组比较

```

1 typedef struct Node {
2     int data;           // 数据域
3     struct Node *next; // 指针域，指向下一个节点
4 } Node;

```

Listing 18: 链表节点定义

3.2.2 链表基本操作

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 链表节点定义
5 typedef struct Node {
6     int data;
7     struct Node *next;
8 } Node;
9
10 // 创建新节点
11 Node* createNode(int data) {
12     Node *newNode = (Node*)malloc(sizeof(Node));
13     if(newNode == NULL) {
14         printf("内存分配失败!\n");
15         exit(1);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // 在链表头部插入节点
23 Node* insertAtHead(Node *head, int data) {
24     Node *newNode = createNode(data);
25     newNode->next = head;
26     return newNode; // 新节点成为新的头节点
27 }
28
29 // 在链表尾部插入节点
30 Node* insertAtTail(Node *head, int data) {
31     Node *newNode = createNode(data);

```

```
32     if(head == NULL) {
33         return newNode;
34     }
35
36     Node *current = head;
37     while(current->next != NULL) {
38         current = current->next;
39     }
40     current->next = newNode;
41
42     return head;
43 }
44
45 // 查找节点
46 Node* findNode(Node *head, int data) {
47     Node *current = head;
48     while(current != NULL) {
49         if(current->data == data) {
50             return current;
51         }
52         current = current->next;
53     }
54     return NULL;
55 }
56
57 // 删除节点
58 Node* deleteNode(Node *head, int data) {
59     if(head == NULL) {
60         return NULL;
61     }
62
63     // 如果要删除的是头节点
64     if(head->data == data) {
65         Node *temp = head;
66         head = head->next;
67         free(temp);
68         return head;
69     }
70
71     // 查找要删除的节点
72     Node *current = head;
73     while(current->next != NULL && current->next->data != data) {
74         current = current->next;
75     }
76
77     if(current->next != NULL) {
78         Node *temp = current->next;
79         current->next = current->next->next;
80         free(temp);
81     }
82 }
```

```
83     return head;
84 }
85
86 // 遍历链表
87 void printList(Node *head) {
88     Node *current = head;
89     printf("链表: ");
90     while(current != NULL) {
91         printf("%d -> ", current->data);
92         current = current->next;
93     }
94     printf("NULL\n");
95 }
96
97 // 获取链表长度
98 int getLength(Node *head) {
99     int length = 0;
100    Node *current = head;
101    while(current != NULL) {
102        length++;
103        current = current->next;
104    }
105    return length;
106 }
107
108 // 释放链表内存
109 void freeList(Node *head) {
110     Node *current = head;
111     while(current != NULL) {
112         Node *temp = current;
113         current = current->next;
114         free(temp);
115     }
116 }
117
118 int main() {
119     Node *head = NULL;
120
121     // 插入测试
122     head = insertAtHead(head, 10);
123     head = insertAtHead(head, 20);
124     head = insertAtTail(head, 30);
125     head = insertAtTail(head, 40);
126
127     printList(head);
128     printf("链表长度: %d\n", getLength(head));
129
130     // 查找测试
131     Node *found = findNode(head, 30);
132     if(found != NULL) {
```

```

134     printf("找到节点: %d\n", found->data);
135 }
136
137 // 删除测试
138 head = deleteNode(head, 20);
139 printList(head);
140
141 // 释放内存
142 freeList(head);
143
144 return 0;
145 }
```

Listing 19: 链表基本操作

3.2.3 有序链表插入

```

1 // 在有序链表中插入节点 (保持升序)
2 Node* insertInOrder(Node *head, int data) {
3     Node *newNode = createNode(data);
4
5     // 如果链表为空或新节点应插入头部
6     if(head == NULL || data < head->data) {
7         newNode->next = head;
8         return newNode;
9     }
10
11    // 查找插入位置
12    Node *current = head;
13    while(current->next != NULL && current->next->data < data) {
14        current = current->next;
15    }
16
17    // 插入节点
18    newNode->next = current->next;
19    current->next = newNode;
20
21    return head;
22 }
```

Listing 20: 有序链表插入

3.3 双向链表

3.3.1 双向链表节点定义

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 双向链表节点
5 typedef struct DNode {
```

```
6     int data;
7     struct DNode *prev; // 指向前一个节点
8     struct DNode *next; // 指向后一个节点
9 } DNode;
10
11 // 创建双向链表节点
12 DNode* createDNode(int data) {
13     DNode *newNode = (DNode*)malloc(sizeof(DNode));
14     if(newNode == NULL) {
15         printf("内存分配失败!\n");
16         exit(1);
17     }
18     newNode->data = data;
19     newNode->prev = NULL;
20     newNode->next = NULL;
21     return newNode;
22 }
23
24 // 在双向链表尾部插入
25 DNode* insertAtTail(DNode *head, int data) {
26     DNode *newNode = createDNode(data);
27
28     if(head == NULL) {
29         return newNode;
30     }
31
32     DNode *current = head;
33     while(current->next != NULL) {
34         current = current->next;
35     }
36
37     current->next = newNode;
38     newNode->prev = current;
39
40     return head;
41 }
42
43 // 遍历双向链表 (向前)
44 void printForward(DNode *head) {
45     DNode *current = head;
46     printf("向前遍历: ");
47     while(current != NULL) {
48         printf("%d <-> ", current->data);
49         current = current->next;
50     }
51     printf("NULL\n");
52 }
53
54 // 遍历双向链表 (向后)
55 void printBackward(DNode *tail) {
56     DNode *current = tail;
```

```

57     printf("向后遍历: ");
58     while(current != NULL) {
59         printf("%d <-> ", current->data);
60         current = current->prev;
61     }
62     printf("NULL\n");
63 }
64
65 // 查找尾节点
66 DNode* findTail(DNode *head) {
67     if(head == NULL) return NULL;
68
69     DNode *current = head;
70     while(current->next != NULL) {
71         current = current->next;
72     }
73     return current;
74 }
75
76 int main() {
77     DNode *head = NULL;
78
79     head = insertAtTail(head, 10);
80     head = insertAtTail(head, 20);
81     head = insertAtTail(head, 30);
82
83     printForward(head);
84
85     DNode *tail = findTail(head);
86     printBackward(tail);
87
88     return 0;
89 }
```

Listing 21: 双向链表

3.4 循环链表

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 循环链表节点
5 typedef struct CNode {
6     int data;
7     struct CNode *next;
8 } CNode;
9
10 // 创建循环链表节点
11 CNode* createCNode(int data) {
12     CNode *newNode = (CNode*)malloc(sizeof(CNode));
13     if(newNode == NULL) {
```

```
14     printf("内存分配失败!\n");
15     exit(1);
16 }
17 newNode->data = data;
18 newNode->next = newNode; // 指向自己, 形成循环
19 return newNode;
20 }
21
22 // 在循环链表尾部插入
23 CNode* insertAtTail(CNode *head, int data) {
24     CNode *newNode = createCNode(data);
25
26     if(head == NULL) {
27         return newNode;
28     }
29
30     CNode *current = head;
31     while(current->next != head) {
32         current = current->next;
33     }
34
35     current->next = newNode;
36     newNode->next = head;
37
38     return head;
39 }
40
41 // 遍历循环链表
42 void printCircularList(CNode *head) {
43     if(head == NULL) {
44         printf("空链表\n");
45         return;
46     }
47
48     CNode *current = head;
49     printf("循环链表: ");
50     do {
51         printf("%d -> ", current->data);
52         current = current->next;
53     } while(current != head);
54     printf("(回到头部)\n");
55 }
56
57 int main() {
58     CNode *head = NULL;
59
60     head = insertAtTail(head, 10);
61     head = insertAtTail(head, 20);
62     head = insertAtTail(head, 30);
63
64     printCircularList(head);
```

```
65     return 0;
66 }
67 }
```

Listing 22: 循环链表

4 文件指针 (File Pointer)

4.1 文件操作基础

4.1.1 文件指针概念

```
1 #include <stdio.h>
2
3 int main() {
4     // 声明文件指针
5     FILE *fp;
6
7     // 打开文件
8     fp = fopen("test.txt", "w");    // 以写入模式打开
9
10    if(fp == NULL) {
11        printf("无法打开文件!\n");
12        return 1;
13    }
14
15    // 写入数据
16    fprintf(fp, "Hello, World!\n");
17    fprintf(fp, "这是C语言文件操作示例\n");
18
19    // 关闭文件
20    fclose(fp);
21    printf("文件写入成功!\n");
22
23    return 0;
24 }
```

Listing 23: 文件指针基础

4.1.2 文件打开模式

4.2 文件读写操作

4.2.1 字符读写

```
1 #include <stdio.h>
2
3 int main() {
4     FILE *fp;
5     char ch;
```

模式	描述
”r”	只读方式打开文本文件
”w”	只写方式创建/清空文本文件
”a”	追加方式打开文本文件
”rb”	只读方式打开二进制文件
”wb”	只写方式创建/清空二进制文件
”ab”	追加方式打开二进制文件
”r+”	读写方式打开文本文件
”w+”	读写方式创建/清空文本文件
”a+”	读写方式打开文本文件 (追加)
”rb+”	读写方式打开二进制文件
”wb+”	读写方式创建/清空二进制文件
”ab+”	读写方式打开二进制文件 (追加)

Table 3: 文件打开模式

```

6 // 写入字符
7 fp = fopen("char.txt", "w");
8 if(fp != NULL) {
9     fputc('A', fp);
10    fputc('B', fp);
11    fputc('C', fp);
12    fclose(fp);
13 }
14

15 // 读取字符
16 fp = fopen("char.txt", "r");
17 if(fp != NULL) {
18     printf("文件内容: ");
19     while((ch = fgetc(fp)) != EOF) {
20         printf("%c ", ch);
21     }
22     fclose(fp);
23     printf("\n");
24 }
25

26 return 0;
27 }
28 }
```

Listing 24: 字符读写

4.2.2 字符串读写

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     FILE *fp;
```

```
6     char buffer[100];
7
8     // 写入字符串
9     fp = fopen("string.txt", "w");
10    if(fp != NULL) {
11        fputs("第一行\n", fp);
12        fputs("第二行\n", fp);
13        fputs("第三行\n", fp);
14        fclose(fp);
15    }
16
17    // 读取字符串
18    fp = fopen("string.txt", "r");
19    if(fp != NULL) {
20        printf("文件内容:\n");
21        while(fgets(buffer, sizeof(buffer), fp) != NULL) {
22            printf("%s", buffer);
23        }
24        fclose(fp);
25    }
26
27    return 0;
28 }
```

Listing 25: 字符串读写

4.2.3 格式化读写

```
1 #include <stdio.h>
2
3 typedef struct {
4     char name[50];
5     int age;
6     float score;
7 } Student;
8
9 int main() {
10     FILE *fp;
11     Student stu1 = {"张三", 20, 85.5};
12     Student stu2;
13
14     // 格式化写入
15     fp = fopen("student.txt", "w");
16     if(fp != NULL) {
17         fprintf(fp, "%s %d %.1f\n", stu1.name, stu1.age, stu1.score);
18         fclose(fp);
19     }
20
21     // 格式化读取
22     fp = fopen("student.txt", "r");
23     if(fp != NULL) {
```

```

24     fscanf(fp, "%s %d %f", stu2.name, &stu2.age, &stu2.score);
25     printf("读取: %s, %d, %.1f\n", stu2.name, stu2.age, stu2.
26         score);
27     fclose(fp);
28 }
29
30     return 0;
31 }
```

Listing 26: 格式化读写

4.3 二进制文件操作

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     char name[50];
6     int age;
7     float score;
8 } Student;
9
10 int main() {
11     FILE *fp;
12     Student stu1 = {"张三", 20, 85.5};
13     Student stu2;
14
15     // 二进制写入
16     fp = fopen("student.dat", "wb");
17     if(fp != NULL) {
18         fwrite(&stu1, sizeof(Student), 1, fp);
19         fclose(fp);
20     }
21
22     // 二进制读取
23     fp = fopen("student.dat", "rb");
24     if(fp != NULL) {
25         fread(&stu2, sizeof(Student), 1, fp);
26         printf("读取: %s, %d, %.1f\n", stu2.name, stu2.age, stu2.
27             score);
28         fclose(fp);
29     }
30
31     return 0;
32 }
```

Listing 27: 二进制文件操作

4.4 文件定位函数

```
1 #include <stdio.h>
2
3 int main() {
4     FILE *fp;
5     char buffer[100];
6
7     fp = fopen("test.txt", "w+");
8     if(fp != NULL) {
9         // 写入数据
10        fprintf(fp, "这是第一行\n");
11        fprintf(fp, "这是第二行\n");
12        fprintf(fp, "这是第三行\n");
13
14        // 回到文件开头
15        rewind(fp);
16
17        // 读取并显示
18        printf("文件内容:\n");
19        while(fgets(buffer, sizeof(buffer), fp) != NULL) {
20            printf("%s", buffer);
21        }
22
23        // 使用fseek定位
24        fseek(fp, 0, SEEK_SET); // 定位到文件开头
25        fseek(fp, 10, SEEK_CUR); // 从当前位置向前移动10字节
26        fseek(fp, -20, SEEK_END); // 从文件末尾向前移动20字节
27
28        // 获取当前位置
29        long position = ftell(fp);
30        printf("当前位置: %ld\n", position);
31
32        fclose(fp);
33    }
34
35    return 0;
36 }
```

Listing 28: 文件定位

4.5 错误处理与文件状态

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp;
6     char ch;
7
8     fp = fopen("nonexistent.txt", "r");
9 }
```

```
10     if(fp == NULL) {
11         perror("打开文件失败"); // 输出错误信息
12         return 1;
13     }
14
15     // 读取文件直到结束
16     while(1) {
17         ch = fgetc(fp);
18
19         if(feof(fp)) { // 检查是否到达文件末尾
20             printf("\n到达文件末尾\n");
21             break;
22         }
23
24         if(ferror(fp)) { // 检查是否发生错误
25             printf("\n读取文件时发生错误\n");
26             break;
27         }
28
29         printf("%c", ch);
30     }
31
32     // 清除错误标志
33     clearerr(fp);
34
35     fclose(fp);
36     return 0;
37 }
```

Listing 29: 文件错误处理

4.6 文件操作综合示例

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_STUDENTS 100
6 #define FILENAME "students.dat"
7
8 typedef struct {
9     int id;
10    char name[50];
11    int age;
12    float score;
13 } Student;
14
15 Student students[MAX_STUDENTS];
16 int studentCount = 0;
17
18 // 保存数据到文件
```

```
19 void saveToFile() {
20     FILE *fp = fopen(FILENAME, "wb");
21     if(fp == NULL) {
22         printf("无法打开文件进行保存!\n");
23         return;
24     }
25
26     // 写入学生数量
27     fwrite(&studentCount, sizeof(int), 1, fp);
28
29     // 写入所有学生数据
30     fwrite(students, sizeof(Student), studentCount, fp);
31
32     fclose(fp);
33     printf("数据已保存到文件!\n");
34 }
35
36 // 从文件加载数据
37 void loadFromFile() {
38     FILE *fp = fopen(FILENAME, "rb");
39     if(fp == NULL) {
40         printf("文件不存在或无法打开!\n");
41         return;
42     }
43
44     // 读取学生数量
45     fread(&studentCount, sizeof(int), 1, fp);
46
47     // 读取学生数据
48     fread(students, sizeof(Student), studentCount, fp);
49
50     fclose(fp);
51     printf("数据已从文件加载!\n");
52 }
53
54 // 添加学生
55 void addStudent() {
56     if(studentCount >= MAX_STUDENTS) {
57         printf("学生数量已达上限!\n");
58         return;
59     }
60
61     Student *s = &students[studentCount];
62
63     printf("请输入学号: ");
64     scanf("%d", &s->id);
65
66     printf("请输入姓名: ");
67     scanf("%s", s->name);
68
69     printf("请输入年龄: ");
```

```
70     scanf("%d", &s->age);
71
72     printf("请输入成绩: ");
73     scanf("%f", &s->score);
74
75     studentCount++;
76     printf("学生添加成功!\n");
77 }
78
79 // 显示所有学生
80 void displayStudents() {
81     if(studentCount == 0) {
82         printf("没有学生信息!\n");
83         return;
84     }
85
86     printf("\n%-10s %-20s %-10s %-10s\n",
87            "学号", "姓名", "年龄", "成绩");
88     printf("-----\n");
89
90     for(int i = 0; i < studentCount; i++) {
91         printf("%-10d %-20s %-10d %-10.1f\n",
92                students[i].id,
93                students[i].name,
94                students[i].age,
95                students[i].score);
96     }
97 }
98
99 // 菜单
100 void menu() {
101     int choice;
102
103     do {
104         printf("\n==== 学生信息管理系统 ====\n");
105         printf("1. 添加学生\n");
106         printf("2. 显示所有学生\n");
107         printf("3. 保存到文件\n");
108         printf("4. 从文件加载\n");
109         printf("0. 退出\n");
110         printf("请选择: ");
111         scanf("%d", &choice);
112
113         switch(choice) {
114             case 1:
115                 addStudent();
116                 break;
117             case 2:
118                 displayStudents();
119                 break;
120             case 3:
```

```
121         saveToFile();
122         break;
123     case 4:
124         loadFromFile();
125         break;
126     case 0:
127         printf("感谢使用!\n");
128         break;
129     default:
130         printf("无效选择!\n");
131     }
132 } while(choice != 0);
133 }
134
135 int main() {
136     menu();
137     return 0;
138 }
```

Listing 30: 学生信息管理系统

5 综合实例：通讯录管理系统

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FILENAME "contacts.dat"
6
7 typedef struct {
8     char name[50];
9     char phone[20];
10    char email[50];
11} Contact;
12
13 typedef struct Node {
14     Contact contact;
15     struct Node *next;
16} Node;
17
18 Node *head = NULL;
19
20 // 创建新节点
21 Node* createNode(Contact contact) {
22     Node *newNode = (Node*)malloc(sizeof(Node));
23     if(newNode == NULL) {
24         printf("内存分配失败!\n");
25         exit(1);
26     }
27     newNode->contact = contact;
```

```
28     newNode->next = NULL;
29     return newNode;
30 }
31
32 // 添加联系人
33 void addContact() {
34     Contact contact;
35
36     printf("请输入姓名: ");
37     scanf("%s", contact.name);
38
39     printf("请输入电话: ");
40     scanf("%s", contact.phone);
41
42     printf("请输入邮箱: ");
43     scanf("%s", contact.email);
44
45     Node *newNode = createNode(contact);
46
47     // 插入到链表头部
48     newNode->next = head;
49     head = newNode;
50
51     printf("联系人添加成功!\n");
52 }
53
54 // 显示所有联系人
55 void displayContacts() {
56     Node *current = head;
57     int count = 0;
58
59     printf("\n==== 通讯录 ===\n");
60     printf("%-20s %-15s %-30s\n", "姓名", "电话", "邮箱");
61     printf("-----\n");
62
63     while(current != NULL) {
64         printf("%-20s %-15s %-30s\n",
65                current->contact.name,
66                current->contact.phone,
67                current->contact.email);
68         current = current->next;
69         count++;
70     }
71
72     printf("总共 %d 个联系人\n", count);
73 }
74
75 // 查找联系人
76 void searchContact() {
```

```
77     char name[50];
78     printf("请输入要查找的姓名: ");
79     scanf("%s", name);
80
81     Node *current = head;
82     int found = 0;
83
84     while(current != NULL) {
85         if(strcmp(current->contact.name, name) == 0) {
86             printf("找到联系人:\n");
87             printf("姓名: %s\n", current->contact.name);
88             printf("电话: %s\n", current->contact.phone);
89             printf("邮箱: %s\n", current->contact.email);
90             found = 1;
91             break;
92         }
93         current = current->next;
94     }
95
96     if(!found) {
97         printf("未找到联系人: %s\n", name);
98     }
99 }
100
101 // 保存通讯录到文件
102 void saveContacts() {
103     FILE *fp = fopen(FILENAME, "wb");
104     if(fp == NULL) {
105         printf("无法打开文件!\n");
106         return;
107     }
108
109     Node *current = head;
110     while(current != NULL) {
111         fwrite(&current->contact, sizeof(Contact), 1, fp);
112         current = current->next;
113     }
114
115     fclose(fp);
116     printf("通讯录已保存到文件!\n");
117 }
118
119 // 从文件加载通讯录
120 void loadContacts() {
121     FILE *fp = fopen(FILENAME, "rb");
122     if(fp == NULL) {
123         printf("文件不存在或无法打开!\n");
124         return;
125     }
126
127     // 清空当前链表
```

```
128     Node *current = head;
129     while(current != NULL) {
130         Node *temp = current;
131         current = current->next;
132         free(temp);
133     }
134     head = NULL;
135
136 // 从文件读取
137 Contact contact;
138 while(fread(&contact, sizeof(Contact), 1, fp) == 1) {
139     Node *newNode = createNode(contact);
140     newNode->next = head;
141     head = newNode;
142 }
143
144 fclose(fp);
145 printf("通讯录已从文件加载!\n");
146 }
147
148 // 释放链表内存
149 void freeList() {
150     Node *current = head;
151     while(current != NULL) {
152         Node *temp = current;
153         current = current->next;
154         free(temp);
155     }
156 }
157
158 // 菜单
159 void menu() {
160     int choice;
161
162     do {
163         printf("\n==== 通讯录管理系统 ===\n");
164         printf("1. 添加联系人\n");
165         printf("2. 显示所有联系人\n");
166         printf("3. 查找联系人\n");
167         printf("4. 保存到文件\n");
168         printf("5. 从文件加载\n");
169         printf("0. 退出\n");
170         printf("请选择: ");
171         scanf("%d", &choice);
172
173         switch(choice) {
174             case 1:
175                 addContact();
176                 break;
177             case 2:
178                 displayContacts();
```

```
179         break;
180     case 3:
181         searchContact();
182         break;
183     case 4:
184         saveContacts();
185         break;
186     case 5:
187         loadContacts();
188         break;
189     case 0:
190         printf("感谢使用!\n");
191         break;
192     default:
193         printf("无效选择!\n");
194     }
195 } while(choice != 0);
196
197 freeList();
198 }
199
200 int main() {
201     menu();
202     return 0;
203 }
```

Listing 31: 通讯录管理系统

6 字符串常用方法

6.1 字符串基础回顾

在 C 语言中，字符串是以空字符 ('\0') 结尾的字符数组。

6.1.1 字符串声明与初始化

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     // 方法1：字符数组
6     char str1[] = "Hello";
7
8     // 方法2：指定大小的字符数组
9     char str2[20] = "World";
10
11    // 方法3：字符指针（指向字符串常量）
12    char *str3 = "Hello World";
13
14    // 方法4：逐个字符初始化
15    char str4[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
16
17    printf("str1: %s\n", str1);
18    printf("str2: %s\n", str2);
19    printf("str3: %s\n", str3);
20    printf("str4: %s\n", str4);
21
22    return 0;
23 }
```

Listing 32: 字符串声明与初始化

6.2 字符串输入输出函数

6.2.1 gets 与 fgets

gets()	fgets()
不安全，可能导致缓冲区溢出	安全，可以指定最大读取长度
不留换行符	保留换行符（如果读取到）
已从 C11 标准中移除	推荐使用

Table 4: gets 与 fgets 比较

```

1 #include <stdio.h>
2
3 int main() {
4     char name[50];
```

```

5   printf("请输入您的姓名: ");
6   fgets(name, sizeof(name), stdin);
7
8   // 移除可能的换行符
9   int len = strlen(name);
10  if(len > 0 && name[len-1] == '\n') {
11      name[len-1] = '\0';
12  }
13
14  printf("您好, %s!\n", name);
15
16
17  return 0;
18 }
```

Listing 33: fgets 使用示例

6.2.2 puts 与 fputs

```

1 #include <stdio.h>
2
3 int main() {
4     char str[] = "Hello World";
5
6     // puts 自动添加换行符
7     puts(str);
8     puts("This is a test");
9
10    // fputs 不添加换行符
11    fputs(str, stdout);
12    fputs("\n", stdout);
13
14    // 写入文件
15    FILE *fp = fopen("output.txt", "w");
16    if(fp != NULL) {
17        fputs(str, fp);
18        fputs("\n", fp);
19        fclose(fp);
20    }
21
22    return 0;
23 }
```

Listing 34: puts 与 fputs 使用

6.3 字符串处理函数

6.3.1 字符串长度 - strlen

```

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

```

3
4 int main() {
5     char str[] = "Hello World";
6     char empty[] = "";
7
8     printf("'%s' 的长度: %lu\n", str, strlen(str));
9     printf("空字符串的长度: %lu\n", strlen(empty));
10
11    // 注意: strlen不包括结尾的'\0'
12    printf("sizeof(str): %lu\n", sizeof(str)); // 12 (包含'\0')
13    printf("strlen(str): %lu\n", strlen(str)); // 11 (不包含'\0')
14
15    return 0;
16 }
```

Listing 35: strlen 使用示例

6.3.2 字符串复制 - strcpy 与 strncpy

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char src[] = "Hello World";
6     char dest1[20];
7     char dest2[20];
8     char dest3[5]; // 目标缓冲区较小
9
10    // strcpy - 不安全, 可能导致缓冲区溢出
11    strcpy(dest1, src);
12    printf("strcpy: %s\n", dest1);
13
14    // strncpy - 安全, 指定最大复制长度
15    strncpy(dest2, src, sizeof(dest2)-1);
16    dest2[sizeof(dest2)-1] = '\0'; // 确保以'\0'结尾
17    printf("strncpy: %s\n", dest2);
18
19    // 处理目标缓冲区较小的情况
20    strncpy(dest3, src, sizeof(dest3)-1);
21    dest3[sizeof(dest3)-1] = '\0';
22    printf("strncpy(截断): %s\n", dest3);
23
24    return 0;
25 }
```

Listing 36: 字符串复制函数

6.3.3 字符串连接 - strcat 与 strncat

```

1 #include <stdio.h>
```

```

2 #include <string.h>
3
4 int main() {
5     char str1[50] = "Hello";
6     char str2[] = " World";
7     char str3[50] = "Hello";
8     char str4[] = " Universe!";
9
10    // strcat - 连接字符串
11    strcat(str1, str2);
12    printf("strcat: %s\n", str1);
13
14    // strncat - 安全连接, 指定最大添加长度
15    strncat(str3, str4, 8); // 只连接前8个字符
16    printf("strncat: %s\n", str3);
17
18    // 更安全的用法
19    char buffer[20] = "Hello";
20    strncat(buffer, " World and Everyone!",
21             sizeof(buffer) - strlen(buffer) - 1);
22    printf("安全连接: %s\n", buffer);
23
24    return 0;
25 }
```

Listing 37: 字符串连接函数

6.3.4 字符串比较 - strcmp 与 strncmp

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str1[] = "apple";
6     char str2[] = "apple";
7     char str3[] = "banana";
8     char str4[] = "APPLE";
9
10    // strcmp - 完全比较
11    printf("strcmp('%s', '%s') = %d\n", str1, str2, strcmp(str1, str2));
12    printf("strcmp('%s', '%s') = %d\n", str1, str3, strcmp(str1, str3));
13    printf("strcmp('%s', '%s') = %d\n", str3, str1, strcmp(str3, str1));
14
15    // 比较规则:
16    // 返回0: 字符串相等
17    // 返回<0: str1小于str2
18    // 返回>0: str1大于str2
19 }
```

```

20 // strcmp - 比较前n个字符
21 printf("strcmp('%s', '%s', 3) = %d\n", str1, str4, strcmp(str1,
22     str4, 3));
23
24 // 不区分大小写的比较 (非标准, Windows可用_strcmp)
25 #ifdef _WIN32
26 printf("_strcmp('%s', '%s') = %d\n", str1, str4, _strcmp(str1,
27     str4));
28#endif
29
30 return 0;
}

```

Listing 38: 字符串比较函数

6.3.5 字符串查找 - strchr 与 strstr

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str[] = "Hello World! Welcome to C programming.";
6
7     // strchr - 查找字符第一次出现的位置
8     char *p1 = strchr(str, 'W');
9     if(p1 != NULL) {
10         printf("找到 'W' 在位置: %ld\n", p1 - str);
11         printf("从该位置开始的字符串: %s\n", p1);
12     }
13
14     // strrchr - 查找字符最后一次出现的位置
15     char *p2 = strrchr(str, 'o');
16     if(p2 != NULL) {
17         printf("最后一个 'o' 在位置: %ld\n", p2 - str);
18     }
19
20     // strstr - 查找子字符串
21     char *p3 = strstr(str, "World");
22     if(p3 != NULL) {
23         printf("找到 'World' 在位置: %ld\n", p3 - str);
24         printf("子字符串: %s\n", p3);
25     }
26
27     // 查找所有出现的字符
28     printf("\n查找所有 'o' 的位置:\n");
29     char *p = str;
30     while((p = strchr(p, 'o')) != NULL) {
31         printf("位置: %ld\n", p - str);
32         p++; // 移动到下一个位置继续查找
33     }
34

```

```

35     return 0;
36 }
```

Listing 39: 字符串查找函数

6.3.6 字符串分割 - strtok

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char str[] = "apple,banana,orange,grape";
6     char *token;
7
8     printf("原始字符串: %s\n", str);
9     printf("分割结果:\n");
10
11    // 第一次调用, 传入原始字符串
12    token = strtok(str, ",");
13
14    while(token != NULL) {
15        printf("%s\n", token);
16        // 后续调用, 传入NULL继续分割
17        token = strtok(NULL, ",");
18    }
19
20    // 注意: strtok会修改原始字符串!
21    printf("\n修改后的原始字符串: %s\n", str);
22
23    return 0;
24 }
```

Listing 40: strtok 使用示例

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main() {
6     char str[] = "John,Doe,25,New York";
7     char backup[100];
8
9     // 备份原始字符串
10    strcpy(backup, str);
11
12    char *tokens[10];
13    int count = 0;
14
15    // 使用strtok_r (可重入版本, 更安全)
16    char *saveptr;
17    char *token = strtok_r(str, ",", &saveptr);
```

```

18
19     while(token != NULL && count < 10) {
20         tokens[count] = token;
21         count++;
22         token = strtok_r(NULL, ", ", &saveptr);
23     }
24
25     printf("CSV数据:\n");
26     printf("原始字符串: %s\n", backup);
27     printf("分割结果:\n");
28     printf("    姓名: %s %s\n", tokens[0], tokens[1]);
29     printf("    年龄: %s\n", tokens[2]);
30     printf("    城市: %s\n", tokens[3]);
31
32     return 0;
33 }
```

Listing 41: 更安全的分割方法

6.3.7 字符串转换函数

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 int main() {
7     // atoi - 字符串转整数
8     char str1[] = "123";
9     char str2[] = "456.78";
10    char str3[] = "789abc";
11
12    int num1 = atoi(str1);
13    int num2 = atoi(str2);
14    int num3 = atoi(str3);
15
16    printf("atoi('%s') = %d\n", str1, num1);
17    printf("atoi('%s') = %d\n", str2, num2);
18    printf("atoi('%s') = %d\n", str3, num3);
19
20    // atof - 字符串转浮点数
21    double f1 = atof(str2);
22    printf("atof('%s') = %.2f\n", str2, f1);
23
24    // strtol - 更安全的字符串转长整数
25    char str4[] = "12345";
26    char *endptr;
27    long num4 = strtol(str4, &endptr, 10); // 十进制
28
29    if(*endptr == '\0') {
30        printf("strtol('%s') = %ld (完全转换)\n", str4, num4);
```

```

31 } else {
32     printf("strtol('%s') = %ld (部分转换, 剩余: %s)\n", str4,
33         num4, endptr);
34 }
35
36 // 大小写转换
37 char str5[] = "Hello World";
38
39 // 转换为小写
40 for(int i = 0; str5[i]; i++) {
41     str5[i] = tolower(str5[i]);
42 }
43 printf("小写: %s\n", str5);
44
45 // 转换为大写
46 for(int i = 0; str5[i]; i++) {
47     str5[i] = toupper(str5[i]);
48 }
49 printf("大写: %s\n", str5);
50
51 return 0;
}

```

Listing 42: 字符串转换示例

6.4 字符串格式化函数

6.4.1 sprintf 与 snprintf

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char buffer[100];
6     int age = 25;
7     float salary = 5000.50;
8     char name[] = "张三";
9
10    // sprintf - 格式字符串写入缓冲区
11    sprintf(buffer, "姓名: %s, 年龄: %d, 工资: %.2f",
12            name, age, salary);
13    printf("%s\n", buffer);
14
15    // snprintf - 安全版本, 指定最大长度
16    char small_buffer[20];
17    int written = snprintf(small_buffer, sizeof(small_buffer),
18                           "姓名: %s, 年龄: %d", name, age);
19
20    if(written >= sizeof(small_buffer)) {
21        printf("缓冲区不足! 需要 %d 字节, 只有 %lu 字节\n",
22              written, sizeof(small_buffer));
}

```

```

23     }
24     printf("实际写入: %s\n", small_buffer);
25
26     return 0;
27 }
```

Listing 43: sprintf 使用示例

6.4.2 sscanf

```

1 #include <stdio.h>
2
3 int main() {
4     char data[] = "John Doe 25 5000.50";
5     char first[20], last[20];
6     int age;
7     float salary;
8
9     // 从字符串中提取格式化数据
10    int result = sscanf(data, "%s %s %d %f",
11                          first, last, &age, &salary);
12
13    if(result == 4) {
14        printf("成功读取 %d 个字段:\n", result);
15        printf("  姓: %s\n", last);
16        printf("  名: %s\n", first);
17        printf("  年龄: %d\n", age);
18        printf("  工资: %.2f\n", salary);
19    }
20
21    // 提取特定格式的数据
22    char date[] = "2024-01-15";
23    int year, month, day;
24    sscanf(date, "%d-%d-%d", &year, &month, &day);
25    printf("\n日期: %d年%d月%d日\n", year, month, day);
26
27    return 0;
28 }
```

Listing 44: sscanf 使用示例

6.5 自定义字符串函数

6.5.1 实现常用字符串函数

```

1 #include <stdio.h>
2
3 // 自定义 strlen 函数
4 size_t my_strlen(const char *str) {
5     size_t len = 0;
6     while(str[len] != '\0') {
```

```
7         len++;
8     }
9     return len;
10 }
11
12 // 自定义 strcpy 函数
13 char* my_strcpy(char *dest, const char *src) {
14     char *original = dest;
15     while ((*dest++ = *src++) != '\0');
16     return original;
17 }
18
19 // 自定义 strcat 函数
20 char* my_strcat(char *dest, const char *src) {
21     char *original = dest;
22
23     // 移动到 dest 的末尾
24     while (*dest != '\0') {
25         dest++;
26     }
27
28     // 复制 src 到 dest 末尾
29     while ((*dest++ = *src++) != '\0');
30
31     return original;
32 }
33
34 // 自定义 strcmp 函数
35 int my_strcmp(const char *str1, const char *str2) {
36     while (*str1 && *str2 && *str1 == *str2) {
37         str1++;
38         str2++;
39     }
40     return *(unsigned char*)str1 - *(unsigned char*)str2;
41 }
42
43 int main() {
44     // 测试自定义函数
45     char str1[50] = "Hello";
46     char str2[] = " World";
47
48     printf("my_strlen('%s') = %lu\n", str1, my_strlen(str1));
49
50     my_strcat(str1, str2);
51     printf("my_strcat: %s\n", str1);
52
53     printf("my_strcmp('apple', 'apple') = %d\n",
54           my_strcmp("apple", "apple"));
55     printf("my_strcmp('apple', 'banana') = %d\n",
56           my_strcmp("apple", "banana"));
57 }
```

```
58     return 0;  
59 }
```

Listing 45: 自定义字符串函数实现

6.6 字符串处理综合示例

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <string.h>  
4 #include <ctype.h>  
5  
6 // 去除字符串两端的空白字符  
7 char* trim(char *str) {  
8     if(str == NULL) return NULL;  
9  
10    // 去除开头的空白字符  
11    char *start = str;  
12    whileisspace(*start)) {  
13        start++;  
14    }  
15  
16    // 去除结尾的空白字符  
17    char *end = str + strlen(str) - 1;  
18    while(end > start && isspace(*end)) {  
19        end--;  
20    }  
21    *(end + 1) = '\0';  
22  
23    // 如果字符串开头有空白，需要移动字符串  
24    if(start != str) {  
25        memmove(str, start, strlen(start) + 1);  
26    }  
27  
28    return str;  
29}  
30  
31 // 字符串反转  
32 char* reverse_string(char *str) {  
33     if(str == NULL) return NULL;  
34  
35     int len = strlen(str);  
36     for(int i = 0; i < len / 2; i++) {  
37         char temp = str[i];  
38         str[i] = str[len - i - 1];  
39         str[len - i - 1] = temp;  
40     }  
41     return str;  
42}  
43  
44 // 统计单词数量
```

```
45 int count_words(const char *str) {
46     if(str == NULL) return 0;
47
48     int count = 0;
49     int in_word = 0;
50
51     for(int i = 0; str[i] != '\0'; i++) {
52         if(!isspace(str[i])) {
53             if(!in_word) {
54                 count++;
55                 in_word = 1;
56             }
57         } else {
58             in_word = 0;
59         }
60     }
61
62     return count;
63 }
64
65 // 判断是否为回文字符串
66 int is_palindrome(const char *str) {
67     if(str == NULL) return 0;
68
69     int len = strlen(str);
70     for(int i = 0; i < len / 2; i++) {
71         if(tolower(str[i]) != tolower(str[len - i - 1])) {
72             return 0;
73         }
74     }
75     return 1;
76 }
77
78 int main() {
79     char text[] = "Hello World! Welcome to C programming. ";
80     char palindrome[] = "A man a plan a canal Panama";
81
82     printf("原始文本: '%s'\n", text);
83
84     // 去除空白
85     printf("去除空白后: '%s'\n", trim(text));
86
87     // 统计单词
88     printf("单词数量: %d\n", count_words(text));
89
90     // 测试回文
91     char test1[] = "racecar";
92     char test2[] = "hello";
93
94     printf("'%s' 是回文吗? %s\n", test1,
95           is_palindrome(test1) ? "是" : "否");
```

```

96     printf("'%s' 是回文吗? %s\n", test2,
97             is_palindrome(test2) ? "是" : "否");
98
99 // 清理句子中的空格并测试回文
100 char clean_palindrome[100];
101 int j = 0;
102 for(int i = 0; palindrome[i] != '\0'; i++) {
103     if(!isspace(palindrome[i])) {
104         clean_palindrome[j++] = tolower(palindrome[i]);
105     }
106 }
107 clean_palindrome[j] = '\0';
108
109 printf("清理后: '%s'\n", clean_palindrome);
110 printf("是回文吗? %s\n",
111         is_palindrome(clean_palindrome) ? "是" : "否");
112
113 // 字符串反转
114 char to_reverse[] = "Hello";
115 printf("'%s' 反转后: '%s'\n",
116         to_reverse, reverse_string(to_reverse));
117
118 return 0;
119 }
```

Listing 46: 字符串处理综合应用

6.7 字符串安全注意事项

重要安全提醒:

- 避免使用不安全的函数: gets, strcpy, strcat 等
- 使用安全版本: fgets, strncpy, strncat 等
- 始终检查缓冲区边界
- 确保字符串以'\0'结尾
- 使用 strncpy 时手动添加终止符
- 处理用户输入时要特别小心

6.7.1 安全字符串处理示例

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 // 安全字符串复制函数
6 int safe_strcpy(char *dest, size_t dest_size, const char *src) {
```

```
7  if(dest == NULL || src == NULL || dest_size == 0) {
8      return -1;
9  }
10
11 size_t src_len = strlen(src);
12
13 // 检查是否有足够空间（包括终止符）
14 if(src_len >= dest_size) {
15     // 复制尽可能多的字符
16     strncpy(dest, src, dest_size - 1);
17     dest[dest_size - 1] = '\0';
18     return -2; // 表示被截断
19 }
20
21 strcpy(dest, src);
22 return 0; // 成功
23 }

24
25 // 安全字符串连接函数
26 int safe_strcat(char *dest, size_t dest_size, const char *src) {
27     if(dest == NULL || src == NULL || dest_size == 0) {
28         return -1;
29     }
30
31     size_t dest_len = strlen(dest);
32     size_t src_len = strlen(src);
33
34     // 检查是否有足够空间
35     if(dest_len + src_len + 1 > dest_size) {
36         // 连接尽可能多的字符
37         strncat(dest, src, dest_size - dest_len - 1);
38         return -2; // 表示被截断
39     }
40
41     strcat(dest, src);
42     return 0; // 成功
43 }

44
45 int main() {
46     char buffer[20];
47
48     // 安全复制
49     int result = safe_strcpy(buffer, sizeof(buffer), "Hello World!");
50     if(result == -2) {
51         printf("警告：字符串被截断\n");
52     }
53     printf("缓冲区：%s\n", buffer);
54
55     // 尝试复制过长的字符串
56     result = safe_strcpy(buffer, sizeof(buffer),
57                         "This is a very long string that won't fit");
58 }
```

```
58     if(result == -2) {
59         printf("警告：字符串被截断\n");
60     }
61     printf("缓冲区：%s\n", buffer);
62
63     // 安全连接
64     safe_strcpy(buffer, sizeof(buffer), "Hello");
65     result = safe_strcat(buffer, sizeof(buffer), " World!");
66     if(result == 0) {
67         printf("安全连接：%s\n", buffer);
68     }
69
70     return 0;
71 }
```

Listing 47: 安全字符串处理

7 最佳实践与常见错误

7.1 结构体最佳实践

1. **使用 `typedef` 创建别名**: 提高代码可读性
2. **合理设计结构体布局**: 考虑内存对齐, 将相关数据放在一起
3. **使用 `const` 保护数据**: 对于不需要修改的结构体参数, 使用 `const` 修饰
4. **考虑结构体大小**: 避免创建过大的结构体, 考虑使用指针
5. **初始化所有成员**: 防止未初始化错误

7.2 链表最佳实践

1. **检查内存分配**: 每次 `malloc` 后检查返回值是否为 `NULL`
2. **及时释放内存**: 使用 `free` 释放不再需要的节点
3. **处理边界情况**: 考虑空链表、单个节点等情况
4. **避免内存泄漏**: 确保每个 `malloc` 都有对应的 `free`
5. **使用哨兵节点**: 简化插入和删除操作

7.3 文件操作最佳实践

1. **检查文件指针**: 每次 `fopen` 后检查返回值是否为 `NULL`
2. **及时关闭文件**: 使用 `fclose` 关闭不再需要的文件
3. **检查读写操作**: 验证 `fread`/`fwrite` 等函数的返回值
4. **使用二进制模式处理结构体**: 避免文本模式下的格式问题
5. **处理文件错误**: 使用 `feof` 和 `ferror` 检查文件状态

7.4 常见错误

```

1 // 错误1: 忘记结构体末尾的分号
2 struct Student { // 错误: 缺少分号
3     char name[50];
4     int age;
5 }
6
7 // 错误2: 访问未初始化的结构体指针
8 struct Student *p;
9 p->age = 20; // 错误: p未初始化
10
11 // 错误3: 链表节点未初始化next指针
12 Node *newNode = malloc(sizeof(Node));
13 newNode->data = 10;
14 // 忘记设置newNode->next = NULL;
15
16 // 错误4: 文件未关闭
17 FILE *fp = fopen("test.txt", "w");
18 // ... 使用文件
19 // 忘记 fclose(fp); // 内存泄漏
20
21 // 错误5: 文件模式使用错误
22 fp = fopen("binary.dat", "r"); // 错误: 应该用"rb"

```

Listing 48: 常见错误示例

8 补充

8.0.1 动态内存分配的综合应用

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 动态数组结构
5 typedef struct {
6     int *data;        // 数组数据
7     int size;         // 当前元素数量
8     int capacity;    // 总容量
9 } DynamicArray;
10
11 // 初始化动态数组
12 DynamicArray* createDynamicArray(int initialCapacity) {
13     DynamicArray *arr = (DynamicArray*)malloc(sizeof(DynamicArray));
14     if(arr == NULL) return NULL;
15
16     arr->data = (int*)malloc(initialCapacity * sizeof(int));
17     if(arr->data == NULL) {
18         free(arr);
19         return NULL;

```

```
20     }
21
22     arr->size = 0;
23     arr->capacity = initialCapacity;
24     return arr;
25 }
26
27 // 添加元素
28 int addElement(DynamicArray *arr, int value) {
29     if(arr->size >= arr->capacity) {
30         // 容量不足，扩展
31         int newCapacity = arr->capacity * 2;
32         int *newData = (int*)realloc(arr->data, newCapacity * sizeof(
33             int));
34
34         if(newData == NULL) {
35             return 0; // 扩展失败
36         }
37
38         arr->data = newData;
39         arr->capacity = newCapacity;
40     }
41
42     arr->data[arr->size] = value;
43     arr->size++;
44     return 1;
45 }
46
47 // 获取元素
48 int getElement(DynamicArray *arr, int index) {
49     if(index < 0 || index >= arr->size) {
50         printf("索引越界!\n");
51         return -1;
52     }
53     return arr->data[index];
54 }
55
56 // 释放动态数组
57 void freeDynamicArray(DynamicArray *arr) {
58     if(arr != NULL) {
59         if(arr->data != NULL) {
60             free(arr->data);
61             arr->data = NULL;
62         }
63         free(arr);
64     }
65 }
66
67 // 使用示例
68 int main() {
69     DynamicArray *arr = createDynamicArray(5);
```

```
70
71     if(arr == NULL) {
72         printf("创建动态数组失败!\n");
73         return 1;
74     }
75
76     // 添加元素
77     for(int i = 0; i < 10; i++) {
78         if(!addElement(arr, i * 10)) {
79             printf("添加元素失败!\n");
80             break;
81         }
82     }
83
84     // 输出元素
85     printf("动态数组内容:\n");
86     for(int i = 0; i < arr->size; i++) {
87         printf("arr[%d] = %d\n", i, getElement(arr, i));
88     }
89
90     printf("数组大小: %d, 容量: %d\n", arr->size, arr->capacity);
91
92     // 释放内存
93     freeDynamicArray(arr);
94
95     return 0;
96 }
```

Listing 49: 动态数组实现