C语言总结课程

目录

---

1. 常见头文件

C语言的标准库通过头文件组织各种功能模块。下面是一些常见的C语言头文件及其作用：

1.1 `<stdio.h>`

`<stdio.h>`（Standard Input/Output）提供了文件输入输出和标准输入输出相关的函数。

- **常用函数：**
  - `printf()`：格式化输出到标准输出（通常是屏幕）。
  - `scanf()`：从标准输入读取格式化输入。
  - `fopen()`：打开文件进行读写。
  - `fclose()`：关闭文件。
  - `fprintf()`：格式化输出到文件。
  - `fscanf()`：从文件读取格式化输入。
  - `fgets()`：从文件或标准输入读取字符串。

1.2 `<stdlib.h>`

`<stdlib.h>` 提供了通用的工具函数，包括内存管理、随机数生成、排序和转换。

- **常用函数：**
  - `malloc()`：动态分配内存。
  - `free()`：释放动态分配的内存。
  - `atoi()`：将字符串转换为整数。
  - `rand()`：生成随机数。
  - `qsort()`：快速排序函数。

1.3 `<string.h>`

`<string.h>` 提供了处理C语言字符串（字符数组）的常用函数。

- **常用函数**：
  - `strcpy()`：复制字符串。
  - `strlen()`：计算字符串的长度。
  - `strcmp()`：比较两个字符串。
  - `strcat()`：拼接两个字符串。
  - `memset()`：填充内存区域。

1.4 `<math.h>`

`<math.h>` 提供了各种数学函数，包括基本的算术运算和高级数学操作。

- **常用函数**：
  - `pow()`：计算幂。
  - `sqrt()`：计算平方根。
  - `sin()`、`cos()`、`tan()`：三角函数。
  - `log()`：计算自然对数。

1.5 `<ctype.h>`

`<ctype.h>` 用于处理字符类型和字符分类判断。

- **常用函数**：
  - `isalpha()`：判断字符是否为字母。
  - `isdigit()`：判断字符是否为数字。
  - `tolower()`：将字符转换为小写。
  - `toupper()`：将字符转换为大写。

1.6 `<time.h>`

`<time.h>` 用于时间操作，包括获取当前时间、计时和格式化时间。

- **常用函数**：
  - `time()`：获取当前时间。
  - `clock()`：获取处理器时间。
  - `strftime()`：格式化时间输出。

1.7 `<assert.h>`

`<assert.h>` 提供了程序断言功能，用于在调试时验证某些条件是否为真。

- **常用宏**：
  - `assert()`：在表达式为假的情况下终止程序并输出错误信息。

1.8 `<limits.h>` 和 `<float.h>`

- **<limits.h>**：定义了整数类型的各种限制，如 `INT_MAX`、`INT_MIN` 等。
- **<float.h>**：定义了浮点类型的各种限制，如 `FLT_MAX`、`DBL_MIN` 等。

1.9 `<stdbool.h>`

`<stdbool.h>` 提供布尔类型 `bool` 及其两个常量值 `true` 和 `false`，使得C语言更容易处理布尔逻辑。

---

2. 文件读写

C语言中文件的操作主要通过 `fopen`、`fclose`、`fscanf`、`fprintf` 等函数实现，常见操作包括文本文件的读取与写入。

示例代码：读取与写入txt文件

```c
#include <stdio.h>

int main() {
    FILE *file;
    char line[100];

    // 写入文件
    file = fopen("output.txt", "w");
    if (file == NULL) {
        printf("无法打开文件\n");
        return 1;
    }
    fprintf(file, "这是测试文本\n");
    fclose(file);

    // 读取文件
    file = fopen("output.txt", "r");
    if (file == NULL) {
        printf("无法打开文件\n");
        return 1;
    }
    while (fgets(line, sizeof(line), file)) {
        printf("%s", line);
    }
    fclose(file);

    return 0;
}
```

易错点

- 文件打开模式（如"r"、"w"等）选择错误
- 未检查文件指针是否为空
- 忘记关闭文件

3. 字符串操作函数

常见的字符串操作函数包括`strcpy`、`strlen`、`strcmp`、`strcat`等，它们主要处理C语言的字符数组。

示例代码：常用字符串操作

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

int main() {
    char str1[20] = "Hello";
    char str2[20] = "World";

    // 字符串拼接
    strcat(str1, str2);
    printf("拼接后: %s\n", str1);

    // 字符串比较
    if (strcmp(str1, "HelloWorld") == 0) {
        printf("字符串相等\n");
    }

    // 字符串长度
    printf("字符串长度: %lu\n", strlen(str1));

    return 0;
}
```

易错点

- 数组越界（未预留足够的空间）
- 使用`strcpy`时，目标数组空间不足
- 使用`strcmp`时忘记返回值含义（返回0表示相等）

---

4. qsort函数

`qsort`是C标准库中的快速排序函数，允许对任意类型的数组进行排序。

示例代码：使用qsort排序

```
cCopy code#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    int arr[] = {4, 2, 9, 7, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    qsort(arr, n, sizeof(int), compare);

    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

易错点

- 比较函数的定义错误，返回值含义不清楚
- 数组元素类型和比较函数中的类型不一致

---

5. 链表指针

链表是一种常见的数据结构，特别是动态数据存储时，链表指针的操作显得尤为重要。

示例代码：简单单向链表

```
cCopy code#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void append(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;

    new_node->data = new_data;
    new_node->next = NULL;

    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }

    while (last->next != NULL)
        last = last->next;

    last->next = new_node;
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}
```

```
int main() {
    struct Node* head = NULL;

    append(&head, 1);
    append(&head, 2);
    append(&head, 3);

    printList(head);
    return 0;
}
```

易错点

- `malloc`未检查返回值是否为空
- 忘记释放链表的动态内存
- 链表的尾节点指针未正确处理

---

6. 指针与内存管理

指针和动态内存管理是C语言中的难点，常见操作包括动态内存的分配和释放。

示例代码：动态内存管理

```
cCopy code#include <stdio.h>
#include <stdlib.h>

int *allocateArray(int size) {
    int *arr = (int*)malloc(size * sizeof(int));
    if (arr == NULL) {
        printf("内存分配失败\n");
        exit(1);
    }
    return arr;
}

int main() {
    int *arr;
    int n = 5;

    arr = allocateArray(n);

    for (int i = 0; i < n; i++) {
        arr[i] = i + 1;
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}
```

易错点

- `malloc`/`calloc`未检查返回值是否为空
- 忘记释放动态分配的内存，导致内存泄漏
- 重复释放同一块内存，导致不可预测的行为

---

7. 常见语法和逻辑错误

在编写C语言代码时，常见的语法和逻辑错误包括：

- 赋值操作`=`与比较操作`==`混淆。
- 忘记在使用指针前初始化它们。
- 在循环和条件语句中使用未定义行为（例如，修改了循环变量，或者数组越界访问）。
- 未考虑数组的边界条件，导致越界访问。
- 内存泄漏，尤其是在涉及动态内存分配时。