



C

极夜酱

# 目录

|          |                              |           |
|----------|------------------------------|-----------|
| <b>1</b> | <b>Hello World!</b>          | <b>1</b>  |
| 1.1      | Hello World! . . . . .       | 1         |
| 1.2      | 数据类型 . . . . .               | 5         |
| 1.3      | 输入输出函数 . . . . .             | 8         |
| 1.4      | 表达式 . . . . .                | 11        |
| <b>2</b> | <b>分支</b>                    | <b>15</b> |
| 2.1      | 逻辑运算符 . . . . .              | 15        |
| 2.2      | if . . . . .                 | 17        |
| 2.3      | switch . . . . .             | 20        |
| <b>3</b> | <b>循环</b>                    | <b>22</b> |
| 3.1      | while . . . . .              | 22        |
| 3.2      | do-while . . . . .           | 25        |
| 3.3      | for . . . . .                | 28        |
| 3.4      | break or continue? . . . . . | 33        |
| <b>4</b> | <b>数组</b>                    | <b>35</b> |
| 4.1      | 一维数组 . . . . .               | 35        |
| 4.2      | 二维数组 . . . . .               | 37        |
| 4.3      | 字符串 . . . . .                | 40        |
| 4.4      | 字符串数组 . . . . .              | 48        |
| <b>5</b> | <b>函数</b>                    | <b>50</b> |
| 5.1      | 函数 . . . . .                 | 50        |
| 5.2      | 变量作用域 . . . . .              | 58        |
| 5.3      | 递归 . . . . .                 | 60        |
| <b>6</b> | <b>指针</b>                    | <b>71</b> |
| 6.1      | 指针 . . . . .                 | 71        |

|          |                   |           |
|----------|-------------------|-----------|
| 6.2      | 指针与数组 . . . . .   | 76        |
| 6.3      | 指针与字符串 . . . . .  | 78        |
| 6.4      | 动态内存申请 . . . . .  | 80        |
| <b>7</b> | <b>文件</b>         | <b>86</b> |
| 7.1      | 文件 . . . . .      | 86        |
| 7.2      | 文件读写 . . . . .    | 88        |
| <b>8</b> | <b>结构体</b>        | <b>97</b> |
| 8.1      | 结构体 . . . . .     | 97        |
| 8.2      | typedef . . . . . | 99        |
| 8.3      | 结构体指针 . . . . .   | 100       |

# Chapter 1 Hello World!

## 1.1 Hello World!

### 1.1.1 编程语言 (Programming Language)

程序是为了让计算机去解决某些问题，它由一系列指令构成。但是计算机并不能理解人类的语言，即使是最简单的，例如“计算一下  $1+2$  是多少”。

计算机采用的是二进制 (binary)，也就是只能够理解 0 和 1，因此编程语言用于作为人类与计算机之间沟通的桥梁。



通过使用编程语言来描述解决问题的步骤，从而让计算机一步一步去执行。流程图 (flow chat) 成为了一种程序的图形化表示方式。



图 1.1: 计算  $\sum_{i=1}^{100} i$  的流程图

### 1.1.2 Hello World!

Hello World 是学习编程的第一个程序，它的作用是向屏幕输出"Hello World!"。

**Hello World!**

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hello World!\n");
6     return 0;
7 }
```

### 运行结果

Hello World!

`#include <stdio.h>` 用于包含标准输入输出库 (standard input/output library) 的头文件 (header file), 这样才能够在程序中进行输入输出相关的操作。

`main()` 是程序的入口, 程序运行后会首先执行 `main()` 中的代码。`printf()` 的功能是在屏幕上输出一个字符串 (string), 其中 `\n` 表示输出一个换行符。最后的分号用于表示一条语句的结束, 注意不要使用中文的分号。

`return 0` 表示 `main()` 运行结束, 返回值为 0, 一般返回 0 用于表示程序正常结束。

不同编程语言的 Hello World 写法大同小异, 可以看出编程语言的基本结构是相似的。

### C++

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     cout << "Hello World!" << endl;
7     return 0;
8 }
```

### Java

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
```

```
4     }  
5 }
```

## Python

```
1 print("Hello World!")
```

### 1.1.3 注释 (Comment)

注释就是对代码的解释和说明，它并不会程序所执行。注释能提高程序的可读性，让人更加容易了解代码的功能。

注释一般分为单行注释和多行注释：

1. 单行注释：以//开头，该行之后的内容视为注释。
2. 多行注释：以/\* 开头，\*/结束，中间的内容视为注释。

## 注释

```
1 /*  
2 * Author: Terry  
3 * Date: 2022/11/16  
4 */  
5  
6 #include <stdio.h>      // header file  
7  
8 int main()  
9 {  
10     printf("Hello World!\n");  
11     return 0;  
12 }
```

## 1.2 数据类型

### 1.2.1 数据类型 (Data Types)

在计算机中，每个数据一般都有一个对应的类型，基础数据类型包括：

#### 1. 整型

- 短整型 short
- 整型 int
- 长整型 long
- 长长整型 long long

#### 2. 浮点型

- 单精度浮点数 float
- 双精度浮点数 double

#### 3. 字符型 char

不同的数据类型所占的内存空间大小不同，因此所能表示的数值范围也不同。

| 数据类型      | 大小   | 取值范围  |
|-----------|------|---|
| short     | 2 字节 | $-2^{15} \sim 2^{15} - 1$                       |
| int       | 4 字节 | $-2^{31} \sim 2^{31} - 1$                       |
| long      | 4 字节 | $-2^{31} \sim 2^{31} - 1$                       |
| long long | 8 字节 | $-2^{63} \sim 2^{63} - 1$                       |
| float     | 4 字节 | $1.2 \times 10^{-38} \sim 3.4 \times 10^{38}$   |
| double    | 8 字节 | $2.3 \times 10^{-308} \sim 1.7 \times 10^{308}$ |
| char      | 1 字节 | $-128 \sim 127$                                 |



### 1.2.2 变量 (Variable)

变量是用来存储数据的内存空间，每个变量都有一个类型，变量中只能存储对应类型的数据。

```
1 int num = 10;  
2 double wage = 8232.56;
```

变量的命名需要符合规范：

1. 由字母、数字和下划线组成，不能以数字开头
2. 不可以使用编程语言中预留的关键字
3. 使用英语单词，顾名思义

关键字是编程语言内置的一些名称，具有特殊的用处和意义，因此不应该作为变量名，防止产生歧义。

|          |         |        |          |        |
|----------|---------|--------|----------|--------|
| auto     | break   | case   | char     | const  |
| continue | default | do     | double   | else   |
| enum     | extern  | float  | for      | goto   |
| if       | int     | long   | register | return |
| short    | signed  | sizeof | static   | struct |
| switch   | typedef | union  | unsigned | void   |
| volatile | while   | inline | restrict |        |

表 1.1: 关键字

### 1.2.3 常量 (Constant)

变量的值在程序运行过程中可以修改，但有一些数据的值是固定的，为了防止这些数据被随意改动，可以将这些数据定义为常量。

在数据类型前加上 `const` 关键字，即可定义常量，常量一般使用大写表示。如果在程序中尝试修改常量，将会报错。

### 常量

```
1 #include <stdio.h>
2
3 int main()
4 {
5     const double PI = 3.1415;
6     PI = 4;
7     return 0;
8 }
```

### 运行结果

error: assignment of read-only variable "PI"

## 1.3 输入输出函数

### 1.3.1 printf()

printf() 的功能是向屏幕输出指定格式的文本，但是有些需要输出的字符在编程语言中具有特殊含义，因此这些特殊的字符，需要经过转义后输出。

| 转义字符 | 描述    |
|------|-------|
| \\   | 反斜杠 \ |
| \'   | 单引号 ' |
| \"   | 双引号 " |
| \n   | 换行    |
| \t   | 制表符   |

表 1.2: 转义字符

#### 转义字符

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("\nHello\nWorld\n");
6     return 0;
7 }
```

#### 运行结果

```
"Hello
World"
```

在对变量的值进行输出时，需要在 printf() 中使用对应类型的占位符。

| 数据类型   | 占位符 |
|--------|-----|
| int    | %d  |
| float  | %f  |
| double | %f  |
| char   | %c  |

表 1.3: 占位符

### 长方形面积

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int length = 10;
6     int width = 5;
7     double area;
8
9     area = length * width;
10    printf("area = %d * %d = %.2f\n", length, width, area);
11    return 0;
12 }
```

### 运行结果

```
area = 10 * 5 = 50.00
```

### 1.3.2 scanf()

有时候一些数据需要从键盘输入，scanf() 可以读取对应类型的数据，并赋值给相应的变量。

在被赋值的变量前需要使用取地址符 &，因为 scanf() 需要将读取到的数据保存

到该变量的内存地址中。

在使用 `scanf()`，通常会使用 `printf()` 先输出一句提示信息，告诉用户需要输入什么数据。

### 圆面积

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     const double PI = 3.14159;
7     double r;
8     double area;
9
10    printf("radius: ");
11    scanf("%lf", &r);
12
13    area = PI * pow(r, 2);
14    printf("area = %.2f\n", area);
15    return 0;
16 }
```

### 运行结果

```
radius: 5
area = 78.54
```

头文件 `math.h` 中定义了一些常用的数学函数，例如 `pow(x, y)` 可用于计算  $x$  的  $y$  次方。

## 1.4 表达式

### 1.4.1 算术运算符

大部分编程语言中的除法与数学中的除法意义不同。

当相除的两个数都为整数时，那么就会进行整除运算，因此结果仍为整数，例如  $21 / 4 = 5$ 。

如果相除的两个数中至少有一个为浮点数时，那么就会进行普通的除法运算，结果为浮点数，例如  $21.0 / 4 = 5.25$ 。

取模（modulo）运算符% 用于计算两个整数相除之后的余数，例如  $22 \% 3 = 1$ 、 $4 \% 7 = 4$ 。

#### 逆序三位数

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int num;
6      int a, b, c;
7
8      printf("Enter a 3-digit integer: ");
9      scanf("%d", &num);
10
11     a = num / 100;
12     b = num / 10 % 10;
13     c = num % 10;
14
15     printf("reversed: %d\n", c*100 + b*10 + a);
16     return 0;
17 }
```

### 运行结果

```
Enter a 3-digit integer: 520
reversed: 25
```

## 1.4.2 复合运算符

使用复合运算符可以使表达式更加简洁。例如 `a = a + b` 可以写成 `a += b`, `--`、`*=`、`/=`、`%=` 等复合运算符的使用方式同理。

当需要给一个变量的值加/减 1 时,除了可以使用 `a += 1` 或 `a -= 1` 之外,还可以使用 `++` 或 `--` 运算符,但是 `++` 和 `--` 可以出现在变量之前或之后:

| 表达式              | 含义           |
|------------------|--------------|
| <code>a++</code> | 执行完所在语句后自增 1 |
| <code>++a</code> | 在执行所在语句前自增 1 |
| <code>a--</code> | 执行完所在语句后自减 1 |
| <code>--a</code> | 在执行所在语句前自减 1 |

表 1.4: 自增/自减运算符

### 自增/自减运算符

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n = 10;
6
7     printf("%d\n", n++);
8     printf("%d\n", ++n);
9     printf("%d\n", n--);
10    printf("%d\n", --n);
```

```
11  
12     return 0;  
13 }
```

#### 运行结果

```
10  
12  
12  
10
```

### 1.4.3 隐式类型转换

在计算机计算的过程中，只有类型相同的数据才可以进行运算。例如整数 + 整数、浮点数/浮点数等。

但是很多时候，我们仍然可以对不同类型的数据进行运算，而并不会产生错误，例如整数 + 浮点数。这是由于编译器会自动进行类型转换。在整数 + 浮点数的例子中，编译器会将整数转换为浮点数，这样就可以进行运算了。

编译器选择将整数转换为浮点数，而不是将浮点数转换为整数的原因在于，浮点数相比整数能够表示的范围更大。例如整数 8 可以使用 8.0 表示，而浮点数 9.28 变为整数 9 后就会丢失精度。

隐式类型转换最常见的情形就是除法运算，这也是导致整数/整数 = 整数、整数/浮点数 = 浮点数的原因。

### 1.4.4 显式类型转换

有些时候编译器无法自动进行类型转换，这时就需要我们手动地强制类型转换。

#### 显式类型转换



```
1 #include <stdio.h>
2
3 int main()
4 {
5     int total = 821;
6     int num = 10;
7     double average = (double)total / num;
8     printf("average = %.2f\n", average);
9     return 0;
10 }
```

#### 运行结果

82.10

# Chapter 2 分支

## 2.1 逻辑运算符

### 2.1.1 关系运算符

编程中经常需要使用关系运算符来比较两个数据的大小，比较的结果是一个布尔值 (boolean)，即 True (非 0) 或 False (0)。

在编程中需要注意，一个等号 = 表示赋值运算，而两个等号 == 表示比较运算。

| 数学符号 | 关系运算符 |
|------|-------|
| <    | <     |
| >    | >     |
| ≤    | <=    |
| ≥    | >=    |
| =    | ==    |
| ≠    | !=    |

表 2.1: 关系运算符

### 2.1.2 逻辑运算符

逻辑运算符用于连接多个关系表达式，其结果也是一个布尔值。

1. 逻辑与 &&: 当多个条件全部为 True，结果为 True。

| 条件 1 | 条件 2 | 条件 1 && 条件 2 |
|------|------|--------------|
| T    | T    | T            |
| T    | F    | F            |
| F    | T    | F            |
| F    | F    | F            |

2. 逻辑或 ||: 多个条件至少有一个为 True 时, 结果为 True。

| 条件 1 | 条件 2 | 条件 1    条件 2 |
|------|------|--------------|
| T    | T    | T            |
| T    | F    | T            |
| F    | T    | T            |
| F    | F    | F            |

3. 逻辑非!: 条件为 True 时, 结果为 False; 条件为 False 时, 结果为 True。

| 条件 | ! 条件 |
|----|------|
| T  | F    |
| F  | T    |

## 2.2 if

### 2.2.1 if

if 语句用于判断一个条件是否成立，如果成立则执行 if 语句块中的语句，否则不执行。

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int age;
6     printf("Enter your age: ");
7     scanf("%d", &age);
8
9     if(age > 0 && age < 18)
10    {
11        printf("minor\n");
12    }
13
14    return 0;
15 }
```

### 2.2.2 if-else

if-else 的结构与 if 类似，只是在 if 语句块中的条件不成立时，执行 else 语句块中的语句。

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int age;
6     printf("Enter your age: ");
```

```

7     scanf("%d", &age);
8
9     if(age > 0 && age < 18)
10    {
11        printf("minor\n");
12    }
13    else
14    {
15        printf("adult\n");
16    }
17
18    return 0;
19 }

```

### 2.2.3 if-else if-else

当需要对更多的条件进行判断时，可以使用 if-else if-else 语句。

```

1  #include <stdio.h>
2
3  int main()
4  {
5      int age;
6      printf("Enter your age: ");
7      scanf("%d", &age);
8
9      if(age > 0 && age < 18)
10     {
11         printf("minor\n");
12     }
13     else if(age >= 18 && age <= 30)
14     {
15         printf("young adult\n");
16     }
17     else if(age >= 31 && age <= 60)

```

```
18     {
19         printf("middle-aged\n");
20     }
21     else if(age >= 61 && age <= 120)
22     {
23         printf("senior\n");
24     }
25     else
26     {
27         printf("invalid age\n");
28     }
29
30     return 0;
31 }
```

## 2.3 switch

### 2.3.1 switch

switch 结构用于根据一个整数值，选择对应的 case 执行。需要注意的是，当对应的 case 中的代码被执行完后，并不会像 if 语句一样跳出 switch 结构，而是会继续向后执行，直到遇到 break。

#### 计算器

```
1 #include <stdio.h>
2
3 int main() {
4     int num1, num2;
5     char operator;
6
7     printf("Enter an expression: ");
8     scanf("%d %c %d", &num1, &operator, & num2);
9
10    switch (operator)
11    {
12        case '+':
13            printf("%d + %d = %d\n", num1, num2, num1 + num2);
14            break;
15        case '-':
16            printf("%d - %d = %d\n", num1, num2, num1 - num2);
17            break;
18        case '*':
19            printf("%d * %d = %d\n", num1, num2, num1 * num2);
20            break;
21        case '/':
22            printf("%d / %d = %d\n", num1, num2, num1 / num2);
23            break;
24        default:
25            printf("Error! Operator is not supported\n");
26            break;
```

```
27     }  
28  
29     return 0;  
30 }
```

#### 运行结果

Enter an expression: 5 \* 8

5 \* 8 = 40



# Chapter 3 循环

## 3.1 while

### 3.1.1 while

在 while 循环中，当条件满足时重复循环体内的语句。如果条件永远为真，循环会永无止境的进行下去（死循环），因此循环体内要有改变条件的机会。

控制循环次数的方法就是设置循环变量：初值、判断、更新。

while 循环的特点是先判断、再执行，所以循环体有可能会进入一次或多次，也有可能一次也不会进入。

```
1 while(条件)
2 {
3     // code
4 }
```

计算 5 个人的平均身高

```
1 #include <stdio.h>
2
3 int main()
4 {
5     double height;
6     double total = 0;
7     double average;
8     int i = 1;
9
10    while(i <= 5)
11    {
12        printf("输入第%d个人的身高: ", i);
```

```

13     scanf("%lf", &height);
14     total += height;
15     i++;
16 }
17
18 average = total / 5;
19 printf("平均身高: %.2f\n", average);
20 return 0;
21 }

```

### 运行结果

输入第1个人的身高: 160.8  
 输入第2个人的身高: 175.2  
 输入第3个人的身高: 171.2  
 输入第4个人的身高: 181.3  
 输入第5个人的身高: 164  
 平均身高: 170.5

### 计算元音、辅音个数

```

1  #include <stdio.h>
2
3  int main()
4  {
5      char c;
6      int vowel = 0;
7      int consonant = 0;
8
9      printf("输入一句英语句子: ");
10
11     while((c = getchar()) != '\n')
12     {
13         if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u'
14            || c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')

```

```
15     {
16         vowel++;
17     }
18     else if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
19     {
20         consonant++;
21     }
22 }
23
24 printf("元音 = %d\n", vowel);
25 printf("辅音 = %d\n", consonant);
26 return 0;
27 }
```

#### 运行结果

输入一句英语句子: Hello World

元音 = 3

辅音 = 7

## 3.2 do-while

### 3.2.1 do-while

do-while 循环在进入循环的时候不做检查，而是在执行完一轮循环体的代码之后，再来检查循环的条件是否满足，如果满足则继续下一轮循环，不满足则结束循环，即至少执行一次循环。

do-while 循环的主要特点是先执行、再判断。

```
1 do {  
2     // code  
3 } while(条件);
```

#### 计算整数位数

```
1 #include <stdio.h>  
2  
3 int main() {  
4     int num;  
5     int n = 0;  
6  
7     printf("输入整数: ");  
8     scanf("%d", &num);  
9  
10    do {  
11        num /= 10;  
12        n++;  
13    } while(num != 0);  
14  
15    printf("位数: %d\n", n);  
16    return 0;  
17 }
```

### 运行结果

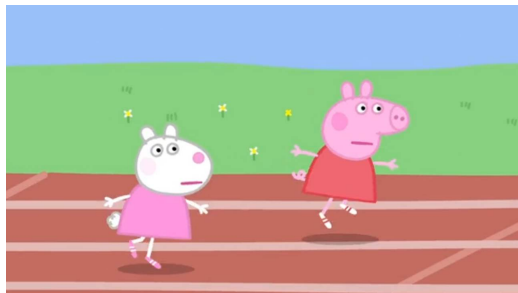
输入整数：123

位数：3

### 3.2.2 while 与 do-while 区别

while 循环与 do-while 循环有以下区别：

1. 执行顺序不同。
2. 初始情况不满足循环条件时，while 循环一次都不会执行，do-while 循环不管任何情况都至少执行一次。
3. do-while 循环的 while 语句后有 **【;】**。



### 猜数字

```
1 #include <stdio.h>
2 #include <stdlib.h>    // 标准库
3 #include <time.h>
4
5 int main() {
6     srand(time(NULL));    // 时间种子
7     int answer = rand() % 100 + 1; // 产生1-100之前的随机数
8     int num = 0;
9     int cnt = 0;
```

```
10
11     do {
12         printf("猜一个1-100之间的数字: ");
13         scanf("%d", &num);
14         cnt++;
15
16         if(num > answer) {
17             printf("猜大了\n");
18         } else if(num < answer) {
19             printf("猜小了\n");
20         }
21     } while(num != answer);
22
23     printf("猜对了! 你一共用了%d次猜对! \n", cnt);
24     return 0;
25 }
```

### 运行结果

猜一个1-100之间的数字: 50  
猜大了!  
猜一个1-100之间的数字: 25  
猜小了!  
猜一个1-100之间的数字: 37  
猜小了!  
猜一个1-100之间的数字: 43  
猜小了!  
猜一个1-100之间的数字: 46  
猜小了!  
猜一个1-100之间的数字: 48  
猜小了!  
猜一个1-100之间的数字: 49  
猜对了! 你一共用了7次猜对!

## 3.3 for

### 3.3.1 for

for 循环有三个表达式，中间用 **【;】** 分隔，**【;】** 不可省略。

```
1 for(表达式1; 表达式2; 表达式3)
2 {
3     //code
4 }
```

- 表达式 1 通常是给循环变量赋初值，可省略。
- 表达式 2 是循环条件，判断是否继续执行循环，可省略。
- 表达式 3 为更新循环变量的值，可省略。

#### 计算 1-100 的累加和

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int sum = 0;
6     for(int i = 1; i <= 100; i++)
7     {
8         sum += i;
9     }
10    printf("%d\n", sum);
11    return 0;
12 }
```

#### 运行结果

5050

计算  $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$

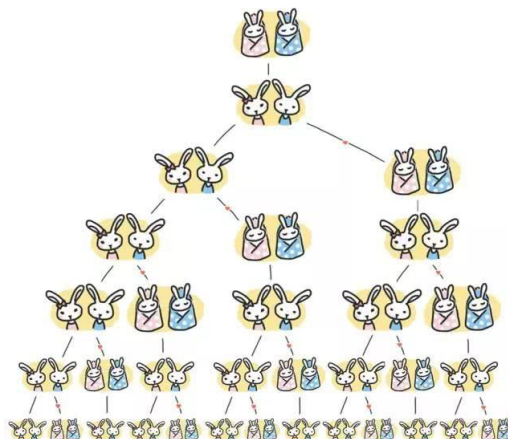
```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6     double sum = 0.0;
7     printf("输入n: ");
8     scanf("%d", &n);
9
10    for(int i = 1; i <= n; i++)
11    {
12        sum += 1.0 / i;
13    }
14    printf("%f\n", sum);
15    return 0;
16 }
```

### 运行结果

输入n: 10

2.928968

### 斐波那契数列





```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6     int num1, num2, val;
7
8     printf("输入斐波那契数列长度: ");
9     scanf("%d", &n);
10
11     if(n == 1)
12     {
13         printf("1\n");
14     }
15     else if(n == 2)
16     {
17         printf("1, 1\n");
18     }
19     else
20     {
21         num1 = 1;
22         num2 = 1;
23         printf("1, 1");
24         for(int i = 3; i <= n; i++)
25         {
26             val = num1 + num2;
27             printf(", %d", val);
28             num1 = num2;
29             num2 = val;
30         }
31         printf("\n");
32     }
33     return 0;
34 }
```

### 运行结果

输入斐波那契数列长度：10

1, 1, 2, 3, 5, 8, 13, 21, 34, 55

### 3.3.2 嵌套循环

循环也可以进行嵌套使用。

### 九九乘法表

|       |        |        |        |        |        |        |        |        |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1*1=1 | 1*2=2  | 1*3=3  | 1*4=4  | 1*5=5  | 1*6=6  | 1*7=7  | 1*8=8  | 1*9=9  |
| 2*1=2 | 2*2=4  | 2*3=6  | 2*4=8  | 2*5=10 | 2*6=12 | 2*7=14 | 2*8=16 | 2*9=18 |
| 3*1=3 | 3*2=6  | 3*3=9  | 3*4=12 | 3*5=15 | 3*6=18 | 3*7=21 | 3*8=24 | 3*9=27 |
| 4*1=4 | 4*2=8  | 4*3=12 | 4*4=16 | 4*5=20 | 4*6=24 | 4*7=28 | 4*8=32 | 4*9=36 |
| 5*1=5 | 5*2=10 | 5*3=15 | 5*4=20 | 5*5=25 | 5*6=30 | 5*7=35 | 5*8=40 | 5*9=45 |
| 6*1=6 | 6*2=12 | 6*3=18 | 6*4=24 | 6*5=30 | 6*6=36 | 6*7=42 | 6*8=48 | 6*9=54 |
| 7*1=7 | 7*2=14 | 7*3=21 | 7*4=28 | 7*5=35 | 7*6=42 | 7*7=49 | 7*8=56 | 7*9=63 |
| 8*1=8 | 8*2=16 | 8*3=24 | 8*4=32 | 8*5=40 | 8*6=48 | 8*7=56 | 8*8=64 | 8*9=72 |
| 9*1=9 | 9*2=18 | 9*3=27 | 9*4=36 | 9*5=45 | 9*6=54 | 9*7=63 | 9*8=72 | 9*9=81 |

表 3.1: 九九乘法表

```
1 #include <stdio.h>
2
3 int main()
4 {
5     for(int i = 1; i <= 9; i++)
6     {
7         for(int j = 1; j <= 9; j++)
8         {
```

```

9         printf("%d*%d=%d\t", i, j, i*j);
10    }
11    printf("\n");
12 }
13 return 0;
14 }

```

### 输出图案

```

1 *
2 **
3 ***
4 ****
5 *****

```

```

1 #include <stdio.h>
2
3 int main()
4 {
5     for(int i = 1; i <= 5; i++)
6     {
7         for(int j = 1; j <= i; j++)
8         {
9             printf("*");
10        }
11        printf("\n");
12    }
13    return 0;
14 }

```

## 3.4 break or continue?

### 3.4.1 循环控制

循环控制语句的作用是控制当前的循环结构是否继续向下执行，如果不进行控制，那么会根据既定的结构重复执行。如果有一些特殊的情况导致循环的执行中断，就称为循环的控制语句。循环控制语句的关键字有 `break` 和 `continue`。

`break` 的作用是跳出当前循环，执行当前循环之后的语句。`break` 只能跳出一层循环，如果是嵌套循环，那么需要按照嵌套的层次，逐步使用 `break` 来跳出。`break` 语句只能在循环体内和 `switch` 语句内使用。

`continue` 的作用是跳过本轮循环，开始下一轮循环的条件判断。`continue` 终止当前轮的循环过程，但它并不跳出循环。

#### break

```
1  #include <stdio.h>
2
3  int main()
4  {
5      for(int i = 1; i <= 10; i++)
6      {
7          if(i == 5)
8          {
9              break;
10         }
11         printf("%d ", i);
12     }
13     return 0;
14 }
```

### 运行结果

1 2 3 4

### continue

```
1 #include <stdio.h>
2
3 int main()
4 {
5     for(int i = 1; i <= 10; i++)
6     {
7         if(i == 5)
8         {
9             continue;
10        }
11        printf("%d ", i);
12    }
13    return 0;
14 }
```

### 运行结果

1 2 3 4 6 7 8 9 10

# Chapter 4 数组

## 4.1 一维数组

### 4.1.1 数组 (Array)

一个变量只能存储一个内容，如果需要存储更多数据，就需要使用数组解决问题。一个数组变量可以存放多个数据，数组是一个值的集合，它们共享同一个名字，数组中的每个变量都能被其下标所访问。

```
1 int number[10];  
2 float grade[50];
```

|      |      |      |      |      |
|------|------|------|------|------|
| a[0] | a[1] | a[2] | a[3] | a[4] |
|------|------|------|------|------|

- 元素：数组中的每个变量
- 大小：数组的容量
- 下标 / 索引 (index)：元素的位置，下标从 0 开始，必须为非负整数

### 4.1.2 数组初始化

一维数组可以在声明时进行初始化：

```
1 int arr[] = {3, 6, 8, 2, 4, 0, 9, 7, 1, 5};
```

很多时候在使用数组之前需要将数组的内容全部清空，这可以利用循环来实现。

一维数组初始化

```
1 int arr[100];
2 for(int i = 0; i < 100; i++)
3 {
4     arr[i] = 0;
5 }
```

### 数组最大值和最小值

```
1 #include <stdio.h>
2
3 int main() {
4     int num[] = {7, 6, 2, 9, 3, 1, 4, 0, 5, 8};
5     int n = sizeof(num) / sizeof(num[0]);
6     int max = num[0];
7     int min = num[0];
8
9     for(int i = 1; i < n; i++) {
10         if(num[i] > max) {
11             max = num[i];
12         } else if(num[i] < min) {
13             min = num[i];
14         }
15     }
16
17     printf("max = %d\n", max);
18     printf("min = %d\n", min);
19     return 0;
20 }
```

### 运行结果

max = 9

min = 0

## 4.2 二维数组

### 4.2.1 二维数组 (2D Array)

二维数组包括行和列两个维度，可以看成是由多个一维数组组成。

|         |         |         |         |
|---------|---------|---------|---------|
| a[0][0] | a[0][1] | a[0][2] | a[0][3] |
| a[1][0] | a[1][1] | a[1][2] | a[1][3] |
| a[2][0] | a[2][1] | a[2][2] | a[2][3] |

二维数组可以在声明时进行初始化：

```
1 int arr[2][2] = {{1, 2}, {3, 4}};
```

#### 初始化二维数组

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[3][4];
6     for(int i = 0; i < 3; i++)
7     {
8         for(int j = 0; j < 4; j++)
9         {
10             arr[i][j] = 0;
11         }
12     }
13     return 0;
14 }
```

#### 矩阵运算

矩阵的加法/减法是指两个矩阵把其相对应元素进行加减的运算。



矩阵加法：两个  $m \times n$  矩阵 A 和 B 的和，标记为  $A + B$ ，结果为一个  $m \times n$  的矩阵，其内的各元素为其相对应元素相加后的值。

矩阵减法：两个  $m \times n$  矩阵 A 和 B 的差，标记为  $A - B$ ，结果为一个  $m \times n$  的矩阵，其内的各元素为其相对应元素相减后的值。

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1-0 & 3-0 \\ 1-7 & 0-5 \\ 1-2 & 2-1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -6 & -5 \\ -1 & 1 \end{bmatrix}$$

```
1 #include <stdio.h>
2
3 int main() {
4     int A[3][2] = {
5         {1, 3},
6         {1, 0},
7         {1, 2}
8     };
9     int B[3][2] = {
10        {0, 0},
11        {7, 5},
12        {2, 1}
13    };
14    int C[3][2];
15
16    printf("矩阵加法\n");
17    for(int i = 0; i < 3; i++) {
18        for(int j = 0; j < 2; j++) {
19            C[i][j] = A[i][j] + B[i][j];
```

```

20         printf("%3d", C[i][j]);
21     }
22     printf("\n");
23 }
24
25 printf("矩阵减法\n");
26 for(int i = 0; i < 3; i++) {
27     for(int j = 0; j < 2; j++) {
28         C[i][j] = A[i][j] - B[i][j];
29         printf("%3d", C[i][j]);
30     }
31     printf("\n");
32 }
33
34 return 0;
35 }

```

### 运行结果

矩阵加法

1 3

8 5

3 3

矩阵减法

1 3

-6 -5

-1 1

## 4.3 字符串

### 4.3.1 字符串 (String)

由字符组成的数组成为字符串。字符串有两种初始化的方式。第一种就是普通的数组初始化形式，另一种是直接使用双引号。

```
1 char str[8] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
2 char str[8] = "program";
```

字符串结尾需要添加一个字符 `\0` 表示结束符，字符串遇到 `\0` 结束。`\0` 占一个字符的大小，记入字符数组的大小。

通过占位符 `%s` 可以对字符串进行输入输出操作：

```
1 printf("%s", str);
2 puts(str);
3
4 scanf("%s", str);
5 gets(str);
```

使用 `scanf()` 读取字符串的时候，字符串会读到空格为止，空格后的内容不会被保存到字符串中。如果需要能够读取字符串直到回车键为止，可以使用 `gets()`。

#### 字符串输入输出

```
1 #include <stdio.h>
2
3 int main() {
4     char str[32];
5     printf("输入字符串: ");
6     gets(str);
7     printf("%s\n", str);
8     return 0;
9 }
```

### 运行结果

输入字符串: hello world

hello world

### 统计字符串中某个字符出现的次数

```
1 #include <stdio.h>
2
3 int main() {
4     char str[32];      // 字符串
5     char c;            // 待统计字符
6     int cnt = 0;       // 出现次数
7     int i = 0;
8
9     printf("输入字符串: ");
10    gets(str);
11    printf("输入待统计字符: ");
12    c = getchar();
13
14    while(str[i] != '\0') {
15        if(str[i] == c) {
16            cnt++;
17        }
18        i++;
19    }
20
21    printf("%c在%s中出现了%d次\n", c, str, cnt);
22    return 0;
23 }
```

### 运行结果

输入字符串: `this is a test`

输入待统计字符: `t`

`t`在`this is a test`中出现了3次

## 4.3.2 ASCII 码

ASCII 全称 American Standard Code for Information Interchange (美国信息交换标准代码), 一共定义了 128 个字符。

| ASCII | 字符  | ASCII | 字符      | ASCII | 字符 | ASCII | 字符 |
|-------|-----|-------|---------|-------|----|-------|----|
| 0     | NUT | 32    | (space) | 64    | @  | 96    | `  |
| 1     | SOH | 33    | !       | 65    | A  | 97    | a  |
| 2     | STX | 34    | "       | 66    | B  | 98    | b  |
| 3     | ETX | 35    | #       | 67    | C  | 99    | c  |
| 4     | EOT | 36    | \$      | 68    | D  | 100   | d  |
| 5     | ENQ | 37    | %       | 69    | E  | 101   | e  |
| 6     | ACK | 38    | &       | 70    | F  | 102   | f  |
| 7     | BEL | 39    | '       | 71    | G  | 103   | g  |
| 8     | BS  | 40    | (       | 72    | H  | 104   | h  |
| 9     | HT  | 41    | )       | 73    | I  | 105   | i  |
| 10    | LF  | 42    | *       | 74    | J  | 106   | j  |
| 11    | VT  | 43    | +       | 75    | K  | 107   | k  |
| 12    | FF  | 44    | ,       | 76    | L  | 108   | l  |
| 13    | CR  | 45    | -       | 77    | M  | 109   | m  |
| 14    | SO  | 46    | .       | 78    | N  | 110   | n  |
| 15    | SI  | 47    | /       | 79    | O  | 111   | o  |
| 16    | DLE | 48    | 0       | 80    | P  | 112   | p  |
| 17    | DC1 | 49    | 1       | 81    | Q  | 113   | q  |
| 18    | DC2 | 50    | 2       | 82    | R  | 114   | r  |
| 19    | DC3 | 51    | 3       | 83    | S  | 115   | s  |

|    |     |    |   |    |   |     |     |
|----|-----|----|---|----|---|-----|-----|
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t   |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u   |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v   |
| 23 | TB  | 55 | 7 | 87 | W | 119 | w   |
| 24 | CAN | 56 | 8 | 88 | X | 120 | x   |
| 25 | EM  | 57 | 9 | 89 | Y | 121 | y   |
| 26 | SUB | 58 | : | 90 | Z | 122 | z   |
| 27 | ESC | 59 | ; | 91 | [ | 123 | {   |
| 28 | FS  | 60 | < | 92 | / | 124 |     |
| 29 | GS  | 61 | = | 93 | ] | 125 | }   |
| 30 | RS  | 62 | > | 94 | ^ | 126 | ~   |
| 31 | US  | 63 | ? | 95 | _ | 127 | DEL |

表 4.1: ASCII 码表

## ASCII 码

```

1 #include <stdio.h>
2
3 int main() {
4     for(int i = 0; i < 128; i++) {
5         printf("%d - %c\n", i, i);
6     }
7     return 0;
8 }
```

### 4.3.3 字符串操作函数

C 的系统库中提供了一些对字符串的常用操作函数，这些函数都定义在 `string.h` 头文件中。

## strlen()

计算字符串的长度，不包括 \0 结束符。

### strlen()

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char s[32] = "hello world";
6     printf("字符串长度 = %d\n", strlen(s));
7     return 0;
8 }
```

### 运行结果

字符串长度 = 11

## strcpy()

将一个字符串复制到另一个字符串中，须确保第一个字符串有足够大的长度。

### strcpy()

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char s1[32] = "hello world";
6     char s2[32] = "program";
7
8     strcpy(s1, s2);
9     printf("s1 = %s\n", s1);
10    printf("s2 = %s\n", s2);
}
```

```
11     return 0;
12 }
```

#### 运行结果

```
s1 = program
s2 = program
```

### strcat()

将第二个字符串拼接第一个字符串尾部，须确保第一个字符串有足够大的长度。

#### strcat()

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char s1[32] = "hello";
6     char s2[32] = "world";
7
8     strcat(s1, s2);    // 把s2拼接到s1后面，s2不发生改变
9     printf("s1 = %s\n", s1);
10    printf("s2 = %s\n", s2);
11    return 0;
12 }
```

#### 运行结果

```
s1 = helloworld
s2 = world
```



## strcmp()

比较两个字符串的大小，依次比较字符串中每一个字符的 ASCII 码。

- 返回负数：字符串 1 小于字符串 2。
- 返回正数：字符串 1 大于字符串 2。
- 返回 0：字符串 1 等于字符串 2。

### strcmp()

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char s1[32] = "communication";
6     char s2[32] = "compare";
7     printf("strcmp()比较结果: %d\n", strcmp(s1, s2));
8     return 0;
9 }
```

### 运行结果

strcmp() 比较结果：-1

### 登录

```
1 /**
2  * 缓冲区溢出
3  * 用户名输入：[32个任意字符] + [新用户名]
4  * 密 码输入：[32个任意字符] + [新密 码]
5  * 产生缓冲区溢出，密码被篡改
6  * 下一次登录输入新用户名和密码就能实现成功登录
7  */
8 #include <stdio.h>
```

```

9  #include <string.h>
10
11 int main() {
12     char username[16] = "admin";
13     char password[16] = "qwerty";
14     char input_username[16];
15     char input_password[16];
16
17     while(1) {
18         printf("用户名: ");
19         gets(input_username);
20         printf("密 码: ");
21         gets(input_password);
22
23         if(strcmp(input_username, username) == 0
24             && strcmp(input_password, password) == 0) {
25             printf("登录成功! \n");
26             break;
27         } else {
28             printf("用户名或密码错误! \n");
29         }
30     }
31
32     return 0;
33 }

```

### 运行结果

用户名: admin  
 密 码: qwerty  
 登录成功!

## 4.4 字符串数组

### 4.4.1 字符串数组

字符串数组就是由多个字符串组成的数组，可以看作是一个二维的字符数组，其中第一维表示字符串数组的大小，第二维表示每个字符串的最大长度。

```
1 char str[4][12] = {"C++", "Java", "Python", "JavaScript"};
```

| 0 | 1 | 2 | 3  | 4  | 5 | 6  | 7 | 8 | 9 | 10 | 11 |
|---|---|---|----|----|---|----|---|---|---|----|----|
| C | + | + | \0 |    |   |    |   |   |   |    |    |
| J | a | v | a  | \0 |   |    |   |   |   |    |    |
| P | y | t | h  | o  | n | \0 |   |   |   |    |    |
| J | a | v | a  | S  | c | r  | i | p | t | \0 |    |

- `str[0]: "C++"`
- `str[1]: "Java"`
- `str[0][0]: 'C'`
- `str[0][1]: '+'`
- `str[0][2]: '+'`

#### 遍历字符串数组

```
1 #include <stdio.h>
2
3 int main() {
4     char str[4][12] = {"C++", "Java", "Python", "JavaScript"};
5     for(int i = 0; i < 4; i++) {
6         printf("%s\n", str[i]);
7     }
8     return 0;
9 }
```

### 运行结果

C++

Java

Python

JavaScript

# Chapter 5 函数

## 5.1 函数

### 5.1.1 函数 (Function)

函数执行一个特定的任务，C 提供了大量内置函数，例如 `printf()` 用来输出字符串、`strlen()` 用来计算字符串长度等。

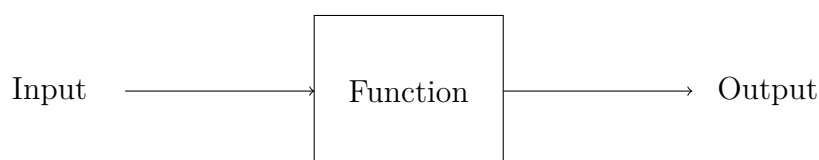


图 5.1: 函数

当调用函数时，程序控制权会转移给被调用的函数，当函数执行结束后，函数会把程序控制权交还其调用者。

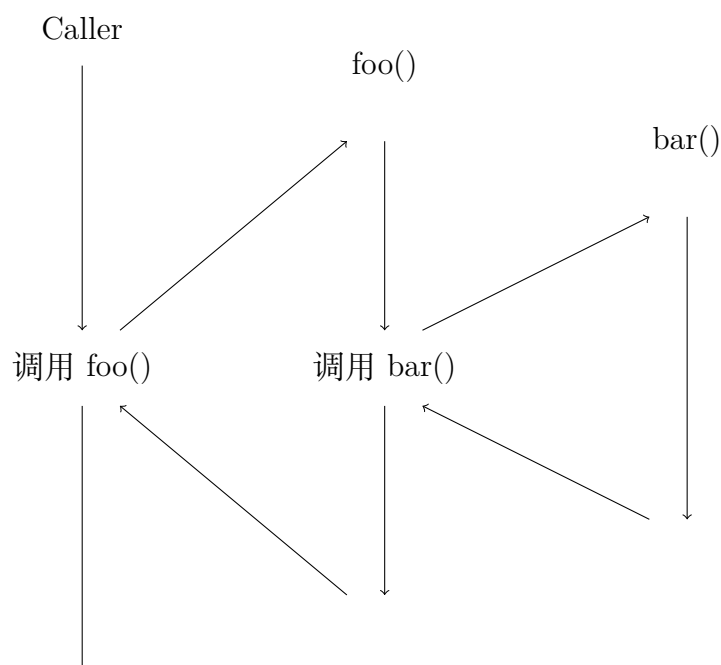


图 5.2: 函数调用

函数声明时需要指定函数的名称、返回类型和参数。在函数声明时，参数的名称

可以省略，但是参数的类型是必须的。

函数的参数列表包括参数的类型、顺序、数量等信息，参数列表可以为空。

函数可以返回一个值，函数的返回类型为被返回的值的类型。函数也可以不返回任何值，此时函数的返回类型应定义为 `void`。

```
1 return_type function_name(parameter_list);
```

```
1 data_type function_name(parameter_list) {  
2     // code  
3 }
```

### 5.1.2 函数设计方法

为什么不把所有的代码全部写在一起，还需要自定义函数呢？

使用函数有以下好处：

1. 避免代码复制，代码复制是程序质量不良的表现
2. 便于代码维护
3. 避免重复造轮子，提高开发效率

在设计函数的时候需要考虑以下的几点要素：

1. 确定函数的功能
2. 确定函数的参数
  - 是否需要参数
  - 参数个数
  - 参数类型
3. 确定函数的返回值

- 是否需要返回值
- 返回值类型

### 函数实现返回最大值

```
1 #include <stdio.h>
2
3 // 函数原型
4 int max(int num1, int num2);
5
6 int main() {
7     printf("%d\n", max(4, 12));
8     printf("%d\n", max(54, 33));
9     printf("%d\n", max(0, -12));
10    printf("%d\n", max(-999, -774));
11    return 0;
12 }
13
14 // 函数实现
15 int max(int num1, int num2) {
16     // if(num1 > num2) {
17     //     return num1;
18     // } else {
19     //     return num2;
20     // }
21
22     return num1 > num2 ? num1 : num2;
23 }
```

### 运行结果

```
12
54
0
-774
```

### 函数实现累加和

```
1 #include <stdio.h>
2
3 int sum(int start, int end) {
4     int total = 0;
5     for(int i = start; i <= end; i++) {
6         total += i;
7     }
8     return total;
9 }
10
11 int main() {
12     printf("1-100的累加和 = %d\n", sum(1, 100));
13     printf("1024-2048的累加和 = %d\n", sum(1024, 2048));
14     return 0;
15 }
```

### 运行结果

```
1-100的累加和 = 5050
1024-2048的累加和 = 1574400
```

### 函数实现输出 i 行 j 列由自定义字符组成的图案

```
1 #include <stdio.h>
2
3 void printChars(int row, int col, char c) {
4     for(int i = 0; i < row; i++) {
5         for(int j = 0; j < col; j++) {
6             printf("%c", c);
7         }
8         printf("\n");
9     }
10 }
```



```

11
12 int main() {
13     printChars(5, 10, '?');
14     return 0;
15 }

```

### 运行结果

```

??????????
??????????
??????????
??????????
??????????

```

### 自定义函数实现 strlen()

```

1 #include <stdio.h>
2
3 /**
4  * @brief 自定义计算字符串长度函数
5  * @param str[]: 待计算字符串
6  * @retval 字符串长度
7  */
8 int myStrlen(char str[]) {
9     int i = 0;
10    while(str[i] != '\0') {
11        i++;
12    }
13    return i;
14 }
15
16 int main() {
17     char s[32] = "hello world";
18     printf("字符串长度 = %d\n", myStrlen(s));
19     return 0;

```

20 }

### 运行结果

字符串长度 = 11

### 自定义函数实现 strcpy()

```
1 #include <stdio.h>
2
3 /**
4  * @brief 自定义字符串复制函数
5  * @param dst[]: 目标字符串
6  * @param src[]: 源字符串
7  * @retval None
8  */
9 void myStrcpy(char dst[], char src[]) {
10     int i = 0;
11     while(src[i] != '\0') {
12         dst[i] = src[i];
13         i++;
14     }
15     dst[i] = '\0';
16 }
17
18 int main() {
19     char s1[32] = "hello world";
20     char s2[32] = "program";
21
22     myStrcpy(s1, s2);
23     printf("s1 = %s\n", s1);
24     printf("s2 = %s\n", s2);
25     return 0;
26 }
```

### 运行结果

s1 = program

s2 = program

### 自定义函数实现 strcat()

```
1 #include <stdio.h>
2
3 /**
4  * @brief 自定义字符串拼接函数
5  * @param dst[]: 目标字符串
6  * @param src[]: 源字符串
7  * @retval None
8  */
9 void myStrcat(char dst[], char src[]) {
10     int i = 0;
11     int j = 0;
12
13     // 找到目标字符串尾部
14     while(dst[i] != '\0') {
15         i++;
16     }
17
18     while(src[j] != '\0') {
19         dst[i++] = src[j++];
20     }
21     dst[i] = '\0';
22 }
23
24 int main() {
25     char s1[32] = "hello";
26     char s2[32] = "world";
27
28     myStrcat(s1, s2);
```

```
29     printf("s1 = %s\n", s1);
30     printf("s2 = %s\n", s2);
31     return 0;
32 }
```

#### 运行结果

```
s1 = helloworld
s2 = world
```

## 5.2 变量作用域

### 5.2.1 局部变量 (Local Variable)

定义在块内的变量就是本地变量，在进入块之前，其中的变量不存在，离开块，变量则释放。在一个块内不能定义同名的变量，并且本地变量不会被默认初始化。

本地变量的生存周期从声明时开始到所在块结束消亡，其作用域为所在的块中。

在函数中，函数的每次调用就会产生一个独立的空间，在这个空间中的变量，是函数的这次运行所独有的，函数的参数也是本地变量。

#### 局部变量

```
1 #include <stdio.h>
2
3 int main() {
4     int a = 1;
5     printf("a = %d\n", a);
6     {
7         int a = 2;
8         printf("a = %d\n", a);
9     }
10    printf("a = %d\n", a);
11    return 0;
12 }
```

#### 运行结果

```
a = 1
a = 2
a = 1
```

### 5.2.2 全局变量 (Global Variable)

全局变量可以在程序任何地方创建，可以被本程序所有对象或函数引用。但是全局变量会占用更多的内存（因为其生命周期长），使用全局变量程序运行时速度更快一些（因为内存不需要再分配）。

全局变量的优先级低于局部变量，当全局变量与局部变量重名的时候，起作用的是局部变量，全局变量会被暂时屏蔽掉。

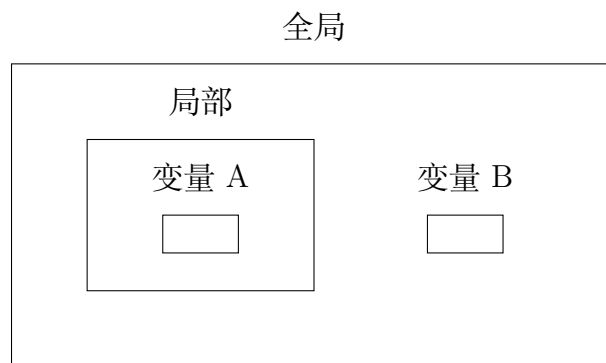


图 5.3: 全局变量

#### 全局变量

```
1 #include <stdio.h>
2
3 int a = 1;    // 全局变量
4
5 int main() {
6     int a = 2; // 本地变量
7     printf("a = %d\n", a);
8     return 0;
9 }
```

#### 运行结果

a = 2

## 5.3 递归

### 5.3.1 递归 (Recursion)

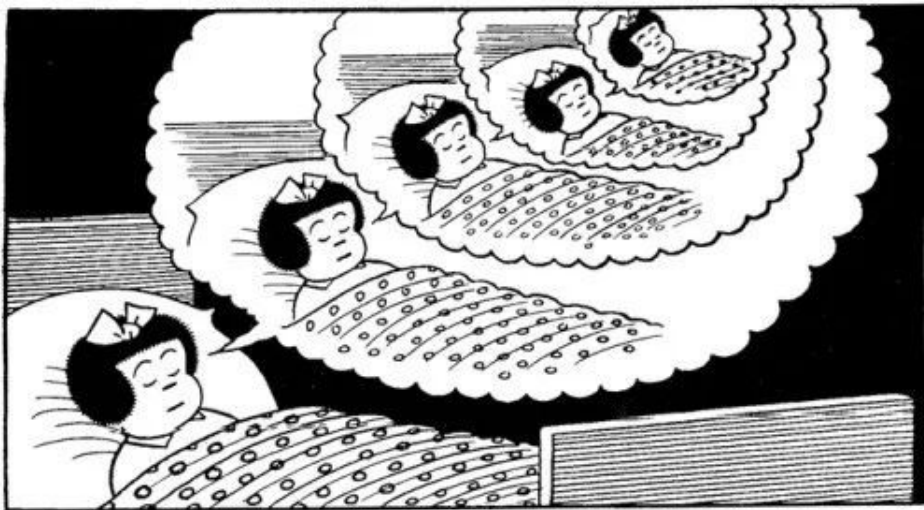
要理解递归，先得理解递归（见5.3章节）。

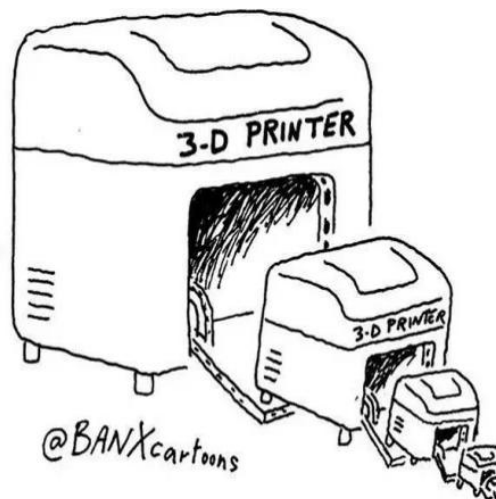
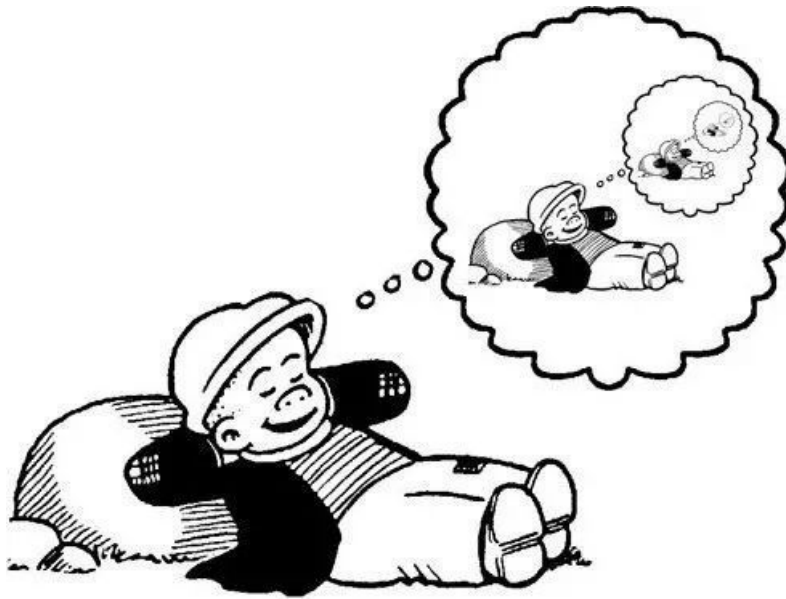
在函数的内部，直接或者间接的调用自己的过程就叫作递归。对于一些问题，使用递归可以简洁易懂的解决问题，但是递归的缺点是性能低，占用大量系统栈空间。

递归算法很多时候可以处理一些特别复杂、难以直接解决的问题。例如：

- 迷宫
- 汉诺塔
- 八皇后
- 排序
- 搜索

在定义递归函数时，一定要确定一个结束条件，否则会造成无限递归的情况，最终会导致栈溢出。









## 无限递归

```
1 #include <stdio.h>
2
3 void tellStory() {
4     printf("从前有座山\n");
5     printf("山里有座庙\n");
6     printf("庙里有个老和尚和小和尚\n");
7     printf("老和尚在对小和尚讲故事\n");
8     printf("他讲的故事是: \n");
9     tellStory();
10 }
11
12 int main() {
13     tellStory();
```

```
14     return 0;
15 }
```

### 运行结果

从前有座山  
山里有座庙  
庙里有个老和尚和小和尚  
老和尚对小和尚在讲故事  
他讲的故事是：  
从前有座山  
山里有座庙  
庙里有个老和尚和小和尚  
老和尚对小和尚在讲故事  
他讲的故事是：  
...

递归函数一般需要定义递归的出口，即结束条件，确保递归能够在适合的地方退出。

### 阶乘

```
1  #include <stdio.h>
2
3  int factorial(int n) {
4      if(n == 0 || n == 1) {
5          return 1;
6      }
7      return n * factorial(n-1);
8  }
9
10 int main() {
11     printf("5! = %d\n", factorial(5));
12     return 0;
}
```

## 运行结果

5! = 120

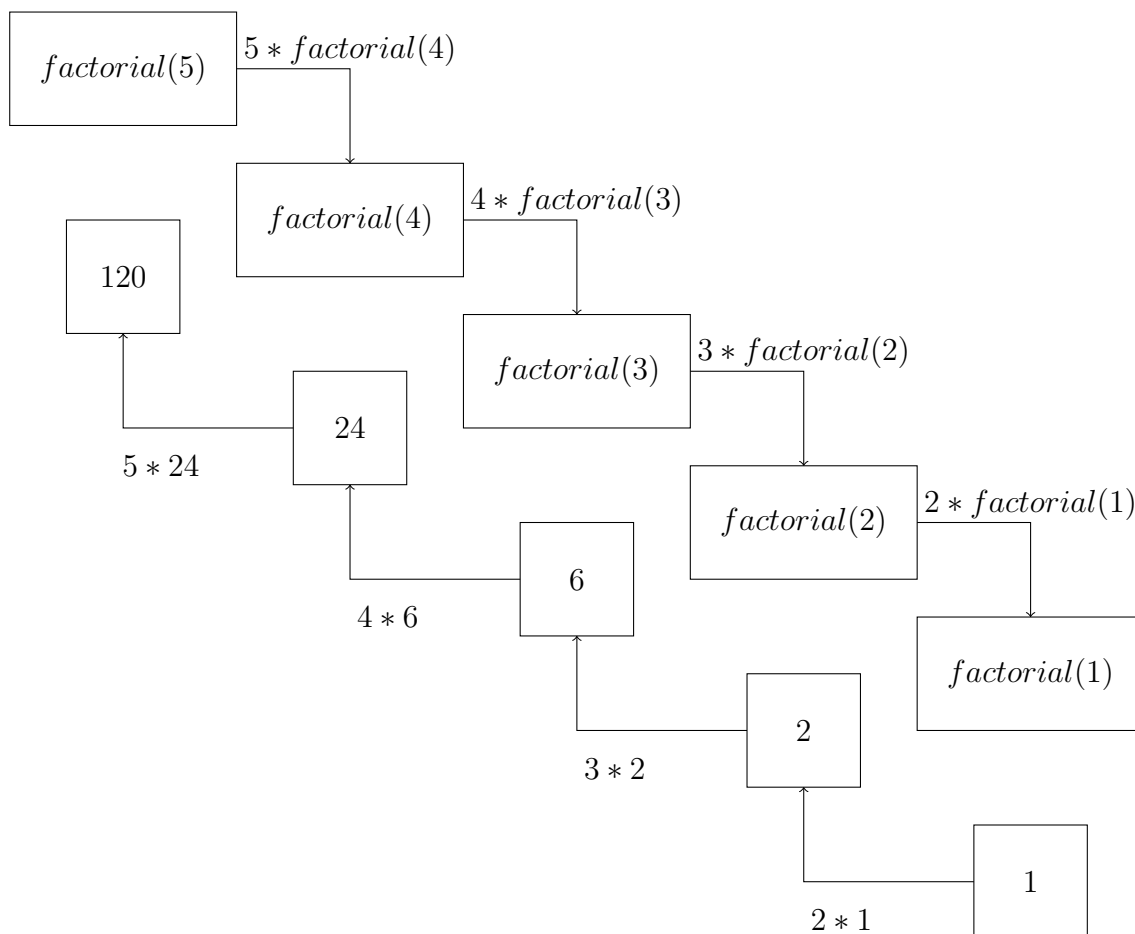


图 5.4: 阶乘

## 斐波那契数列（递归）

```

1 #include <stdio.h>
2
3 int fibonacci(int n) {
4     if(n == 1 || n == 2) {
5         return 1;
6     }

```

```

7     return fibonacci(n-2) + fibonacci(n-1);
8 }
9
10 int main() {
11     int n = 7;
12     printf("斐波那契数列第%d位: %d\n", n, fibonacci(n));
13     return 0;
14 }

```

### 运行结果

斐波那契数列第7位：13

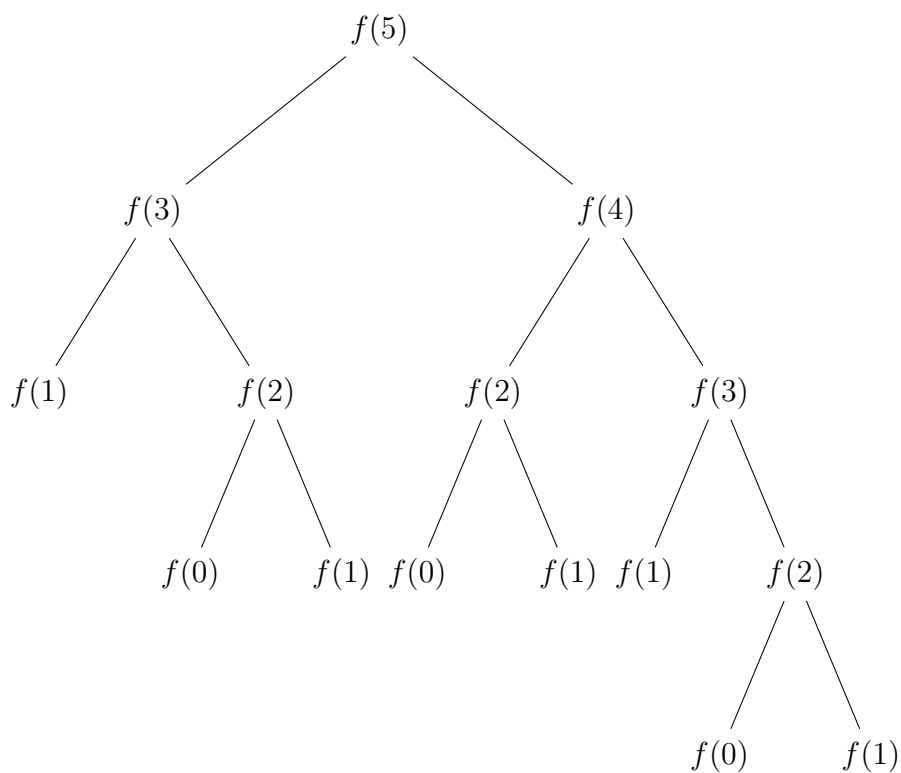


图 5.5: 递归树

### 斐波那契数列（迭代）

```

1 #include <stdio.h>
2

```

```

3  int fibonacci(int n) {
4      int f[n];
5      f[0] = f[1] = 1;
6      for(int i = 2; i < n; i++) {
7          f[i] = f[i-2] + f[i-1];
8      }
9      return f[n-1];
10 }
11
12 int main() {
13     int n = 7;
14     printf("斐波那契数列第%d位: %d\n", n, fibonacci(n));
15     return 0;
16 }

```

### 运行结果

斐波那契数列第7位: 13

### 阿克曼函数

$$A(m, n) = \begin{cases} n + 1 & m = 0 \\ A(m - 1, 1) & m > 0, n = 0 \\ A(m - 1, A(m, n - 1)) & m > 0, n > 0 \end{cases}$$

```

1  #include <stdio.h>
2
3  int A(int m, int n) {
4      if(m == 0) {
5          return n + 1;
6      } else if(m > 0 && n == 0) {
7          return A(m-1, 1);
8      } else {

```

```

9         return A(m-1, A(m, n-1));
10    }
11 }
12
13 int main() {
14     printf("%d\n", A(3, 4));
15     return 0;
16 }

```

### 运行结果

125

| $m \backslash n$ | 0         | 1               | 2               | 3                     | 4               | $n$   |
|------------------|-----------|-----------------|-----------------|-----------------------|-----------------|---|
| 0                | 1         | 2               | 3               | 4                     | 5               | $n + 1$   |
| 1                | 2         | 3               | 4               | 5                     | 6               | $2 + (n + 3) - 3$   |
| 2                | 3         | 5               | 7               | 9                     | 11              | $2(n + 3) - 3$  |
| 3                | 5         | 13              | 29              | 61                    | 125             | $2^{n+3} - 3$   |
| 4                | 13        | 65533           | $2^{65536} - 3$ | $A(3, 2^{65536} - 3)$ | $A(3, A(4, 3))$ | $\underbrace{2^{2^{\cdot^{\cdot^2}}}}_{n+3 \text{ twos}} - 3$ |
| 5                | 65533     | $A(4, 65533)$   | $A(4, A(5, 1))$ | $A(4, A(5, 2))$       | $A(4, A(5, 3))$ | ...   |
| 6                | $A(5, 1)$ | $A(5, A(5, 1))$ | $A(5, A(6, 1))$ | $A(5, A(6, 2))$       | $A(5, A(6, 3))$ | ...   |

表 5.1: 阿克曼函数

吓得我抱起了

抱着抱着抱着我的小鲤鱼的我的我的我



### 汉诺塔

给定三根柱子，其中 A 柱子从大到小套有  $n$  个圆盘，问题是如何借助 B 柱子，将圆盘从 A 搬到 C。

规则：

- 一次只能搬动一个圆盘
- 不能将大圆盘放在小圆盘上面



递归算法求解汉诺塔问题：

1. 将前  $n-1$  个圆盘从 A 柱借助于 C 柱搬到 B 柱。
2. 将最后一个圆盘直接从 A 柱搬到 C 柱。
3. 将  $n-1$  个圆盘从 B 柱借助于 A 柱搬到 C 柱。

```
1 #include <stdio.h>
2
3 int move = 0;      // 移动次数
4
5 /**
6  * @brief 汉诺塔算法
7  * @note 把 n 个盘子从 src 借助 mid 移到 dst
8  * @param n: 层数
9  * @param src: 起点柱子
10  * @param mid: 临时柱子
11  * @param dst: 目标柱子
12  */
13 void hanoi(int n, char src, char mid, char dst) {
14     if(n == 1) {
15         printf("%d号盘: %c -> %c\n", n, src, dst);
16         move++;
17     } else {
18         // 把前 n-1 个盘子从 src 借助 dst 移到 mid
19         hanoi(n-1, src, dst, mid);
20         // 移动第 n 个盘子
21         printf("%d号盘: %c -> %c\n", n, src, dst);
22         move++;
23         // 把刚才的 n-1 个盘子从 mid 借助 src 移到 dst
24         hanoi(n-1, mid, src, dst);
25     }
26 }
27
28 int main() {
29     hanoi(4, 'A', 'B', 'C');
30     printf("步数 ==> %d\n", move);
31     return 0;
```



**运行结果**

1号盘：A -> B

2号盘：A -> C

1号盘：B -> C

3号盘：A -> B

1号盘：C -> A

2号盘：C -> B

1号盘：A -> B

4号盘：A -> C

1号盘：B -> C

2号盘：B -> A

1号盘：C -> A

3号盘：B -> C

1号盘：A -> B

2号盘：A -> C

1号盘：B -> C

步数 ==> 15

# Chapter 6 指针

## 6.1 指针

### 6.1.1 指针 (Pointer)

指针是一个变量，用来保存另一个变量的地址。指针与其它变量或常量一样，在使用指针之前需使用【\*】指定一个变量是指针类型。

```
1 data_type *pointer_name;
```

通过取地址运算符【&】可以获取变量在内存中的地址。

#### 指针

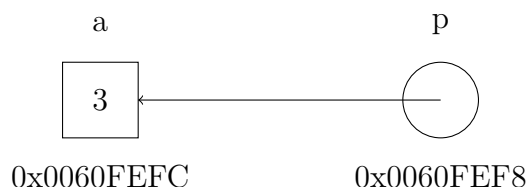
```
1 #include <stdio.h>
2
3 int main() {
4     int a;
5     int *p = &a;
6
7     printf("变量a的地址: %p\n", &a);
8     printf("指针p保存的值: %p\n", p);
9     printf("指针p的地址: %p\n", &p);
10    return 0;
11 }
```

#### 运行结果

变量a的地址: 0060FEFC

指针p保存的值: 0060FEFC

指针p的地址: 0060FEF8



### 6.1.2 取内容运算符

取内容运算符 **【\*】** 是一个单目运算符，用来访问指针所指向地址上的值。

#### 通过指针修改变量的值

```
1 #include <stdio.h>
2
3 int main() {
4     int a = 1;
5     int *p = &a;
6
7     printf("指针p所指向的地址上的值: %d\n", *p);
8     *p = 2;
9     printf("指针p所指向的地址上的值: %d\n", *p);
10    return 0;
11 }
```

#### 运行结果

指针p所指向的地址上的值: 1

指针p所指向的地址上的值: 2

取地址运算符 **【&】** 与取内容运算符 **【\*】** 起相反作用：

- $*\&p == *(\&p) == p$
- $\&*p == \&(*p) == p$

为什么要多此一举通过指针修改变量的值？

由于函数只能由一个返回值，如果当函数需要返回多个值时，某些值就只能通过指针进行返回。

### 交换两个变量的值（Bug 版本）

```
1 #include <stdio.h>
2
3 void swap(int num1, int num2) {
4     int temp = num1;
5     num1 = num2;
6     num2 = temp;
7 }
8
9 int main() {
10     int a = 11;
11     int b = 22;
12
13     printf("交换前: a = %d, b = %d\n", a, b);
14     swap(a, b);
15     printf("交换后: a = %d, b = %d\n", a, b);
16     return 0;
17 }
```

### 运行结果

交换前: a = 11, b = 22

交换后: a = 11, b = 22

### 交换两个变量的值（正确版本）

```
1 #include <stdio.h>
2
3 void swap(int *num1, int *num2) {
4     int temp = *num1;
```

```

5     *num1 = *num2;
6     *num2 = temp;
7 }
8
9 int main() {
10     int a = 11;
11     int b = 22;
12
13     printf("交换前: a = %d, b = %d\n", a, b);
14     swap(&a, &b);
15     printf("交换后: a = %d, b = %d\n", a, b);
16     return 0;
17 }

```

#### 运行结果

交换前: a = 11, b = 22

交换后: a = 22, b = 11

### 6.1.3 野指针

使用指针时最常见的错误就是声明了指针变量，但还没有指向任何变量，就开始使用指针。

#### 野指针

```

1 #include <stdio.h>
2
3 int main() {
4     int *p;
5     printf("%d\n", *p);
6     return 0;
7 }

```

### 运行结果

warning:

'p' is used uninitialized in this function [-Wuninitialized]

在变量声明的时候，如果没有确切的地址可以赋值，为指针变量赋一个 NULL 值是一个良好的编程习惯。赋为 NULL 值的指针被称为空指针。NULL 指针是一个定义在标准库中的值为零的常量。

```
1 #define NULL 0
```

### 空指针 NULL

```
1 #include <stdio.h>
2
3 int main() {
4     int *p = NULL;
5     printf("%p\n", p);
6     return 0;
7 }
```

### 运行结果

000000

## 6.2 指针与数组

### 6.2.1 指针与数组

数组变量本身就表达地址，所以无需使用【&】取地址。

```
1 int arr[10];  
2 int *p = arr;
```

但是数组的每个单元表达的是变量，需要使用【&】取地址。

```
1 int arr[10];  
2 int *p = &arr[0];
```

#### 指针遍历数组

```
1 #include <stdio.h>  
2  
3 int main() {  
4     int arr[] = {44, 12, 64, 78, 16, 72, 13, 98, 84};  
5     int n = sizeof(arr) / sizeof(arr[0]);  
6     int *p = arr;  
7  
8     while(p < arr + n) {  
9         printf("%d ", *p);  
10        p++;  
11    }  
12    printf("\n");  
13    return 0;  
14 }
```

#### 运行结果

44 12 64 78 16 72 13 98 84

### 6.2.2 数组与函数

在将数组作为函数参数传递的时候，在函数参数列表中的数组实际上是一个指向数组首地址的指针。

以下两种函数声明是等价的：

```
1 int func(int arr[]);
2 int func(int *arr);
```

#### 查找数组最大值

```
1 #include <stdio.h>
2
3 int getMax(int *arr, int n) {
4     int max = arr[0];
5     for(int i = 1; i < n; i++) {
6         if(arr[i] > max) {
7             max = arr[i];
8         }
9     }
10    return max;
11 }
12
13 int main() {
14     int arr[] = {76, 23, 12, 98, 5, 61, 30};
15     int n = sizeof(arr) / sizeof(arr[0]);
16     int max = getMax(arr, n);
17     printf("max = %d\n" , max);
18     return 0;
19 }
```

#### 运行结果

max = 98



## 6.3 指针与字符串

### 6.3.1 指针与字符串

指针还可以指向一个字符串常量，但是试图通过指针所指的字符串做写操作会导致程序崩溃。

#### 修改字符串常量

```
1 #include <stdio.h>
2
3 int main() {
4     char *s = "hello";
5     s[0] = 'H';
6     printf("%s\n", s);
7     return 0;
8 }
```

因此，如果需要对字符串进行修改，应该用字符数组的形式。

#### 修改字符串

```
1 #include <stdio.h>
2
3 int main() {
4     char s[] = "hello";
5     s[0] = 'H';
6     printf("%s\n", s);
7     return 0;
8 }
```

#### 运行结果

Hello

在对一个指向字符串的指针进行赋值操作的时候，并没有产生新的字符串，只是让两个指针都指向该字符串，对其中任意一个指针做的操作都会对另一个指针产生影响。

### 指向字符串的指针

```
1 #include <stdio.h>
2
3 int main() {
4     char str[] = "hello";
5     char *s = str;
6     char *t = s;
7     s[0] = 'H';
8     printf("指针s指向的字符串: %s\n", s);
9     printf("指针t指向的字符串: %s\n", t);
10    printf("指针s的地址: %p\n", &s);
11    printf("指针t的地址: %p\n", &t);
12    return 0;
13 }
```

### 运行结果

指针s指向的字符串: Hello

指针t指向的字符串: Hello

指针s的地址: 0022FE40

指针t的地址: 0022FE38

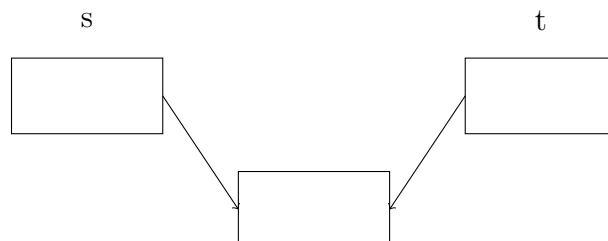


图 6.1: 指向字符串的指针

## 6.4 动态内存申请

### 6.4.1 malloc()

C99 支持声明数组时使用变量作为数组的大小。

```
1 int n = 50;  
2 int arr[n];
```

但是在 C99 之前的版本中，需要使用动态内存申请的方式进行数组空间的开辟。malloc() 的功能是向系统申请指定的内存空间（以字节为单位），使用该函数需要包含头文件 stdlib.h。

malloc() 函数原型为：

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

malloc() 的返回值为 void \* 类型，表示一个指向申请到的空间的首地址，是一个无类型的指针，开发者需要自行转换为自己需要的类型。如果 malloc() 申请内存失败，则会返回空指针 NULL。

**耗尽所有可申请到的内存空间**

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main() {  
5     void *p;  
6     int cnt = 0;  
7  
8     // 每次申请100MB的空间  
9     while((p = malloc(100 * 1024 * 1024))) {  
10         cnt++;  
11     }  
12     printf("一共分配了%dMB空间\n", cnt*100);  
13     return 0;
```

14 }

### 运行结果

一共分配了1900MB空间

通过 malloc() 申请来的空间是需要归还给操作系统的，否则程序长时间运行内存会逐渐下降。

通过 free() 可以把申请来的空间释放，但是有两点需要注意：

1. 只能释放通过 malloc() 申请得到的空间。
2. 只能通过空间的首地址进行释放。

### 动态申请内存空间

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int n;
6     printf("班级人数: ");
7     scanf("%d", &n);
8
9     int *scores = (int *)malloc(sizeof(int) * n);
10    if(!scores) {
11        fprintf(stderr, "内存申请失败\n");
12        exit(1);
13    }
14
15    int total = 0;
16    for(int i = 0; i < n; i++) {
17        printf("第%d个学生成绩: ", i+1);
18        scanf("%d", &scores[i]);
19        total += scores[i];
```

```

20     }
21
22     printf("平均分: %.2f\n", 1.0 * total / n);
23     free(scores);
24     return 0;
25 }

```

### 运行结果

```

班级人数: 5
第1个学生成绩: 67
第2个学生成绩: 98
第3个学生成绩: 100
第4个学生成绩: 53
第5个学生成绩: 65
平均分: 76.60

```

在函数中的定义的字符数组是局部变量，其作用域和生命周期仅在函数内有效，如果将其作为函数返回值返回，在函数外部无法访问到该变量的内容。

### 函数返回字符串（Bug 版本）

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  /**
6   * @brief 生成一段自我介绍
7   * @param name: 姓名
8   * @param age: 年龄
9   * @retval 指定格式字符串: 大家好, 我叫{name}, 今年{age}岁。
10  */
11 char* generateInfo(char *name, int age) {
12     char info[128] = "大家好, 我叫";
13     char age_str[8] = "";

```

```

14     strcat(info, name);
15     strcat(info, ", 今年");
16     // itoa()函数用于将整数转为字符串
17     // 把age以10进制转换为字符串保存到age_str
18     strcat(info, itoa(age, age_str, 10));
19     strcat(info, "岁。");
20     return info;
21 }
22
23 int main() {
24     printf("%s\n", generateInfo("极夜酱", 17));
25     return 0;
26 }

```

### 运行结果

warning:

function returns address of local variable [-Wreturn-local-addr]

### 函数返回字符串（正确版本）

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  /**
6   * @brief 生成一段自我介绍
7   * @param name: 姓名
8   * @param age: 年龄
9   * @retval 指定格式字符串: 大家好, 我叫{name}, 今年{age}岁。
10  */
11 char* generateInfo(char *name, int age) {
12     char *info = (char *)malloc(sizeof(char) * 128);
13     if(!info) {
14         return NULL;

```

```

15     }
16     char age_str[8] = "";
17     strcpy(info, "大家好, 我叫");
18     strcat(info, name);
19     strcat(info, ", 今年");
20     // itoa()函数用于将整数转为字符串
21     // 把age以10进制转换为字符串保存到age_str
22     strcat(info, itoa(age, age_str, 10));
23     strcat(info, "岁。");
24     return info;
25 }
26
27 int main() {
28     printf("%s\n", generateInfo("极夜酱", 17));
29     return 0;
30 }

```

### 运行结果

大家好, 我叫极夜酱, 今年17岁。

## 6.4.2 内存管理

内存通常包括了栈区 (stack)、堆区 (heap)、数据区、程序代码区：

- 栈区：由编译器自动分配和释放，存放函数的参数值、局部变量的值等。
- 堆区：一般由程序员分配和释放，若程序员不释放，程序结束后被 OS 回收。
- 数据区：存放全局变量和静态变量，程序结束后由系统释放。
- 程序代码区：存放函数体的二进制代码。

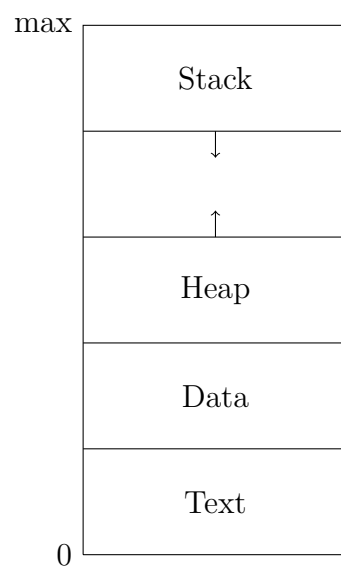


图 6.2: 内存管理



# Chapter 7 文件

## 7.1 文件

### 7.1.1 文件打开

C 具有操作文件的能力，比如对文件数据的添加、删除、修改等。为了统一对各种硬件的操作，不同的硬件设备也都被看作是文件进行管理。计算机中键盘是标准输入设备 `stdin`，显示器是标准输出设备 `stdout`。

C 通过声明一个文件 `FILE` 类型的指针，可以对指针所指向的文件进行操作。

`fopen()` 用于打开文件，函数原型为：

```
1 FILE *fopen(const char *fname, const char *mode);
```

| 打开方式 | 描述                            |
|------|-------------------------------|
| r    | 只读，文件必须存在，否则打开失败              |
| w    | 只写，创建一个新文件                    |
| a    | 追加，如果文件不存在则创建；存在则将数据追加到末尾     |
| r+   | 读 + 写，文件必须存在，否则打开失败           |
| w+   | 写 + 读，创建一个新文件                 |
| a+   | 追加 + 读，如果文件不存在则创建；存在则将数据追加到末尾 |
| rb   | 以只读打开二进制文件                    |
| wb   | 以只写打开二进制文件                    |
| ab   | 以追加打开二进制文件                    |
| rb+  | 以读 + 写打开二进制文件                 |
| wb+  | 以写 + 读打开二进制文件                 |
| ab+  | 以追加 + 读打开二进制文件                |

表 7.1: 文件打开方式

如果文件打开失败，`fopen()` 则会返回空指针 `NULL`。

### 7.1.2 文件关闭

在对文件操作结束后，需要使用 `fclose()` 将文件关闭。

`fclose()` 负责清空缓冲区，并释放文件指针。需要特别注意的是，在对文件执行写操作以后，并不会马上写入文件，而只是写入到了这个文件的输出缓冲区中。只有当输出缓冲区满了，或者执行了 `fflush()`，或者执行了 `fclose()` 以后，或者程序结束，才会把输出缓冲区中的内容写入文件。

文件

data.txt

```
1 This is a test.
```

file\_open.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("data.txt", "r");
6     if(!fp) {
7         fprintf(stderr, "File Open Failed\n");
8         exit(1);
9     }
10    fclose(fp);
11    return 0;
12 }
```

## 7.2 文件读写

### 7.2.1 fgetc() 读字符

fgetc() 的功能是从文件读取一个字符。成功时，返回读到的字符（int 类型）；失败或读到文件尾，返回 EOF（-1）。

fgetc() 函数原型：

```
1 int fgetc(FILE *stream);
```

#### 读取并输出指定文件内容

data.txt

```
1 极夜酱 17
2 小灰 22
3 小白 19
```

fgetc.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("data.txt", "r");
6     if(!fp) {
7         fprintf(stderr, "File Open Failed\n");
8         exit(1);
9     }
10
11     char c;
12     while((c = fgetc(fp)) != EOF) {
13         printf("%c", c);
14     }
15
16     fclose(fp);
17     return 0;
```

18 }

### 统计程序源代码的字符数和行数

count\_chars\_and\_lines.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("count_chars_and_lines.c", "r");
6     if(!fp) {
7         fprintf(stderr, "File Open Failed\n");
8         exit(1);
9     }
10
11     char c;
12     int charNum = 0;        // 字符数量
13     int lineNum = 0;        // 行数
14
15     while((c = fgetc(fp)) != EOF) {
16         if(c == '\n') {
17             lineNum++;
18         } else {
19             charNum++;
20         }
21     }
22
23     printf("字符数: %d\n", charNum);
24     printf("行 数: %d\n", lineNum);
25
26     fclose(fp);
27     return 0;
28 }
```

### 运行结果

字符数：513

行 数：27

## 7.2.2 fputc() 写字符

fputc() 的功能是将一个字符写入文件中。

fputc() 函数原型：

```
1 int fputc(int ch, FILE *stream);
```

### 将程序源代码输出到指定文件

fputc.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp1 = fopen("fputc.c", "r");
6     FILE *fp2 = fopen("data.txt", "w");
7     if(!fp1) {
8         fprintf(stderr, "File Open Failed\n");
9         exit(1);
10    }
11
12    char c;
13    while((c = fgetc(fp1)) != EOF) {
14        fputc(c, fp2);
15    }
16
17    fclose(fp1);
18    fclose(fp2);
19    return 0;
```

### 7.2.3 fgetc() 读字符串

fgetc() 的功能是从文件读取一个字符串。读取成功时，返回指向字符串的指针；读取失败时，返回 NULL。

fgetc() 函数原型：

```
1 char *fgetc(char *str, int num, FILE *stream);
```

- str：用于保存字符串的变量。
- num：最多读取字符数量，由于字符串结尾需要保留 \0 结束符，因此真正只能最多读取 num - 1 个字符。

#### 读取并输出程序源代码内容

fgetc.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("data.txt", "r");
6     if(!fp) {
7         fprintf(stderr, "File Open Failed\n");
8         exit(1);
9     }
10
11     char c;
12     while((c = fgetc(fp)) != EOF) {
13         printf("%c", c);
14     }
15
16     fclose(fp);
```

```
17     return 0;
18 }
```

### 7.2.4 fputs() 写字符串

fputs() 的功能是将一个字符串写入文件中。

fputs() 函数原型：

```
1 char *fputs(const char *str, FILE *stream);
```

将程序源代码输出到指定文件

fputs.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp1 = fopen("fputs.c", "r");
6     FILE *fp2 = fopen("data.txt", "w");
7     if(!fp1) {
8         fprintf(stderr, "File Open Failed\n");
9         exit(1);
10    }
11
12    char line[128];
13    while(fgets(line, sizeof(line), fp1)) {
14        fputs(line, fp2);
15    }
16
17    fclose(fp1);
18    fclose(fp2);
19    return 0;
20 }
```

## 7.2.5 fprintf() 格式化输出

fprintf() 使用方法与 printf() 类似，只是多增加了一个参数，用于指定输出流。

fprintf() 函数原型：

```
1 int fprintf(FILE *stream, const char *format, ...);
```

将数据格式化输出到指定文件

fprintf.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("data.txt", "w");
6     char name[3][12] = {"极夜酱", "小灰", "小白"};
7     int age[3] = {17, 22, 19};
8
9     for(int i = 0; i < 3; i++) {
10         fprintf(fp, "%s\t%d\n", name[i], age[i]);
11     }
12
13     fclose(fp);
14     return 0;
15 }
```

运行结果 data.txt

极夜酱 17

小灰 22

小白 19



## 7.2.6 fscanf() 格式化输入

fscanf() 的功能是按照指定格式从文件读取数据。读取成功时返回实际读取的数据个数，失败时返回 EOF。

fscanf() 函数原型：

```
1 int fscanf(FILE *stream, const char *format, ...);
```

### 从指定文件读取指定格式数据

data.txt

```
1 极夜酱 17
2 小灰 22
3 小白 19
```

fscanf.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     FILE *fp = fopen("data.txt", "r");
6     if(!fp) {
7         fprintf(stderr, "File Open Failed\n");
8         exit(1);
9     }
10
11     char name[12];
12     int age;
13
14     while(fscanf(fp, "%s\t%d", name, &age) != EOF) {
15         printf("%s\t%d\n", name, age);
16     }
17
18     fclose(fp);
19     return 0;
20 }
```

### 运行结果

极夜酱 17

小灰 22

小白 19

## 7.2.7 feof() 检查文件结束

feof() 的功能是检查文件是否已经达到文件末尾位置，如果是就返回非零值（真）。

feof() 函数原型：

```
1 int feof(FILE *stream);
```

### 从通讯录文件中查找指定人名

data.txt

```
1 极夜酱 17
2 小灰 22
3 小白 19
```

feof.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <stdbool.h>
5
6 int main() {
7     FILE *fp = fopen("data.txt", "r");
8     if(!fp) {
9         fprintf(stderr, "File Open Failed\n");
10        exit(1);
11    }
12
13    char key[32];    // 需查找数据
```

```
14     char name[32];
15     int age;
16     bool found = false; // 是否找到
17
18     printf("查找姓名: ");
19     gets(key);
20
21     while(!feof(fp)) {
22         fscanf(fp, "%s\t%d", name, &age);
23         if(strcmp(name, key) == 0) {
24             printf("%s\t%d\n", name, age);
25             found = true;
26             break;
27         }
28     }
29
30     if(!found) {
31         printf("未找到【%s】的信息\n", key);
32     }
33
34     fclose(fp);
35     return 0;
36 }
```

### 运行结果

查找姓名: 小灰

小灰 22

# Chapter 8 结构体

## 8.1 结构体

### 8.1.1 结构体 (Structure)

C 中，数组是一种允许存储多个相同类型数据项的结构。结构体是另一种用户自定义的数据类型，它允许存储不同类型的数据项。

结构体的声明可以使用关键字 `struct`，结构体名一般首字母大写。结构体的声明以 **【;】** 结束。结构体的声明通常定义为全局变量，这样就可以被多个函数所使用的了。

```
1 struct struct_name {  
2     data_type var_name1;  
3     data_type var_name2;  
4     ...  
5 };
```

通常会用于描述同一个事物的变量定义成结构体。例如：

- 日期（年、月、日）
- 坐标（横坐标、纵坐标）
- 学生信息（姓名、年龄、学号、成绩）

定义结构体变量时，不能只使用结构体名，需要加上 `struct` 关键字。

通过成员运算符 **【.】** 可以访问一个结构体之中的成员变量。

结构体

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

```
2
3 struct Date {
4     int year;
5     int month;
6     int day;
7 };
8
9 int main() {
10     struct Date date;
11     date.year = 2021;
12     date.month = 3;
13     date.day = 12;
14
15     printf("%d年%d月%d日\n", date.year, date.month, date.day);
16     return 0;
17 }
```

#### 运行结果

2021年3月12日

## 8.2 typedef

### 8.2.1 typedef

关键字 typedef 可以用来给数据类型定义别名，通过使用 typedef 可以简化结构的声明，不用每次都加上 struct 关键字了。

```
1 typedef struct [struct_name] {  
2     data_type var_name1;  
3     data_type var_name2;  
4     ...  
5 } struct_name;
```

#### typedef 定义别名

```
1 #include <stdio.h>  
2  
3 typedef struct Coordinate {  
4     double x;  
5     double y;  
6 } Coordinate;  
7  
8 int main() {  
9     Coordinate coor;  
10    coor.x = 3.1;  
11    coor.y = 2.7;  
12    printf("(%.1f, %.1f)\n", coor.x, coor.y);  
13    return 0;  
14 }
```

#### 运行结果

(3.1, 2.7)

## 8.3 结构体指针

### 8.3.1 结构体指针

与数组不同，结构体变量的名字并不是结构体变量的地址，必须使用取地址运算符【&】。

结构体也可以作为函数参数进行传递。如果是按值传递，那么在函数中会新创建一个结构体变量，并复制调用者的结构体的值。如果是按址传递，则需要传递结构体的指针。

C 提供了一个间接引用运算符【->】，可以直接访问结构体指针所指的结构变量中的成员。

#### 倒数

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // 分数
5 typedef struct Fraction {
6     int numerator;    // 分子
7     int denominator; // 分母
8 } Fraction;
9
10 /**
11  * @brief 倒数
12  * @note 分母不能为0
13  * @param f: 分数结构体
14  * @retval None
15  */
16 void reciprocal(Fraction *f) {
17     if(f->numerator == 0) {
18         fprintf(stderr, "无法计算倒数\n");
19     } else {
```

```
20     int temp = f->numerator;
21     f->numerator = f->denominator;
22     f->denominator = temp;
23 }
24 }
25
26 int main() {
27     Fraction fraction = {2, 5};           // 2/5
28     printf("%d/%d的倒数是", fraction.numerator, fraction.denominator);
29     reciprocal(&fraction);
30     printf("%d/%d\n", fraction.numerator, fraction.denominator);
31     return 0;
32 }
```

#### 运行结果

2/5的倒数是5/2