

C/C++ 程序设计

字符串与格式化输入/输出

张晓平

武汉大学数学与统计学院

Table of contents

1. 字符串简介
2. 常量与预处理器
3. 格式化输出

字符串简介

字符串简介 I

```
#include <stdio.h>
#include <string.h>
// density of human body: 1.04e3 kg / m^3
#define DENSITY 1.04e3
int main(void)
{
    float weight, volume;
    int size, letters;
    char name[40];
    printf("Hi! What's your first name?\n");
    scanf("%s", name);
    printf("%s, what's your weight in kilograms?\n", name);
}
```

字符串简介 II

```
scanf("%f", &weight);
size = sizeof name;
letters = strlen(name);
volume = weight / DENSITY;
printf("Well, %s, your volume is %2.2f cubic
meters.\n", name, volume);
printf("Also, your first name has %d letters,\n
n", letters);
printf("and we have %d bytes to store in it.\n
", size);
return 0;
}
```

字符串简介

```
Hi! What's your first name?
```

```
xiaoping
```

```
xiaoping, what's your weight in kilograms?
```

```
60
```

```
Well, xiaoping, your volume is 0.06 cubic meters
```

```
.
```

```
Also, your first name has 8 letters,
```

```
and we have 40 bytes to store in it.
```

字符串简介

定义

字符串（string）就是一个或多个字符的序列。例如：

```
"Once more you open the door!"
```

注意

字符串用双引号括起来，但双引号不是字符串的一部分。

字符串简介

C 字符串

- C 没有为字符串定义专门的数据类型，而是把它存储在 `char` 数组中。
- 字符串的字符存放字符数组中，每个字符占用一个单元。

```
#include <stdio.h>
int main(void)
{
    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: ");
    printf("%s\n", greeting);
    return 0;
}
```


C++ 字符串

C++ 提供了两种类型的字符串表示形式：

- C 风格字符串
 - C 风格的字符串起源于 C 语言，并在 C++ 中继续得到支持。
- C++ 引入的 `string` 类类型
 - C++ 标准库提供了 `string` 类类型。

字符串简介

```
// C style string
#include <iostream>
using namespace std;
int main()
{
    char greeting[6] = "Hello";
    cout << "Greeting message: ";
    cout << greeting << endl;
    return 0;
}
```

```
// C++ string class
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
    string greeting = "Hello";
    cout << "Greeting message: ";
    cout << greeting << endl;
    return 0;
}
```

C 字符串

- C 字符串存储在字符数组中，最后一个元素为空字符 `'\0'`，用于标记字符串的结束。
- `'\0'` 不是数字 0，它是非打印字符，其 ASCII 码的值为 0。
- `'\0'` 的存在意味着数组长度至少要比存储字符数多 1。

定义：数组（array）

数组是同一类型的数据元素的有序序列。

定义：数组（array）

数组是同一类型的数据元素的有序序列。

```
char name[40];
```

该声明语句创建一个有 40 个存储单元的数组，其中每个单元可存储一个 char 型值。

定义：数组（array）

数组是同一类型的数据元素的有序序列。

```
char name[40];
```

该声明语句创建一个有 40 个存储单元的数组，其中每个单元可存储一个 char 型值。

- [] 说明 name 是一个数组
- [] 中的 40 指出数组的元素个数
- char 标识每个元素的类型

字符串简介：字符串的使用

要使用 C 字符串，必须创建一个数组，把字符串中的字符逐个放入数组中，最后还需在结尾添加一个空字符 `'\0'`。如：

```
char greeting[10] = {'H', 'e', 'l', 'l', 'o', ' ',  
'\0'};
```

字符串简介：字符串的使用

要使用 C 字符串，必须创建一个数组，把字符串中的字符逐个放入数组中，最后还需在结尾添加一个空字符 `'\0'`。如：

```
char greeting[10] = {'H', 'e', 'l', 'l', 'o', ' ',  
'\0'};
```

但这种方法太麻烦，我们可以通过如下方式来自动完成上述过程：

```
char greeting[10] = "Hello";
```


字符串简介：字符串的使用

```
#include <stdio.h>
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    char name[40];
    printf("What's your name?\n");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    return 0;
}
```

字符串简介：字符串的使用

```
#include <stdio.h>
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    char name[40];
    printf("What's your name?\n");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    return 0;
}
```

What's your name?

Xiaoping Zhang

Hello, Xiaoping. What a super marvelous name!

字符串简介：字符串的使用

```
#include <iostream>
#include <string>
using namespace std;
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    string name;
    cout << "What's your name?" << endl;
    cin >> name;
    cout << "Hello, " << name << ". " << PRAISE <<
        endl;
    return 0;
}
```

字符串简介：字符串的使用

```
#include <iostream>
#include <string>
using namespace std;
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    string name;
    cout << "What's your name?" << endl;
    cin >> name;
    cout << "Hello, " << name << ". " << PRAISE <<
        endl;
    return 0;
}
```

What's your name?

Xiaoping Zhang

Hello, Xiaoping. What a super marvelous name!

C 字符串的输入

- 无须把 `'\0'` 插入 `name` 数组中，`scanf()` 会在读取输入时完成此任务。
- `name` 前无须加 `&`，因 `name` 本身就表示地址。
- 使用 `%s` 的 `scanf()` 语句会在遇到的第一个空格、制表符或换行符处停止读取，它只会把第一个单词而不是把整条语句作为字符串读入。

C++ 字符串的输入

- 使用 `cin` 读取字符串时，也按单词读取，即遇到第一个空格、制表符或换行符时会自动忽略后面的内容。

"x" 与 'x' 的差别

- 'x' 为字符，而 "x" 为字符串
- "x" 由两个字符 'x' 和 '\0' 组成

字符串简介：字符串长度

C 字符串

若使用字符数组来存储字符串 `char name[40] = "Hello";`，则

- `sizeof name` 将 `name` 数组所占空间的字节大小，即 40；
- 使用 `strlen(name)` 统计字符串 `name` 的字符个数，不包含 `'\0'`，即 5。

C++ 字符串

若使用字符串对象 `string name = "Hello";`，则

- `sizeof` 计算 `name` 对象所占空间的字节大小；
- 使用类方法 `name.size()` 来统计字符串的字符个数，不包含 `'\0'`，即 5。

字符串简介：字符串长度

```
#include <stdio.h>
#include <string.h>
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    char name[40];
    printf("What's your name?\n");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    printf("Your name of %ld letters occupises %ld
        memory cells.\n", strlen(name), sizeof name);
    printf("PRAISE has %ld letters and occupies %
        ld memory cells.\n", strlen(PRAISE), sizeof
        PRAISE);
    return 0;
}
```


字符串简介

What's your name?

Xiaoping

Hello, Xiaoping. What a super marvelous name!

Your name of 8 letters occupied 40 memory cells.

PRAISE has 30 letters and occpied 31 memory cells.

字符串简介：字符串长度 I

```
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
#define PRAISE "What's a super marvelous name!"
int main(void)
{
    string name;
    cout << "What's your name?" << endl;
    cin >> name;
    cout << "Hello, " << name << ". "
         << PRAISE << endl;
    cout << "Your name of " << name.size()
```

字符串简介：字符串长度 II

```
        << " letters occupies " << sizeof name
        << " memory cells." << endl;
    cout << "PRAISE has " << strlen(PRAISE)
        << " letters and occupies " << sizeof
        PRAISE
        << " memory celss." << endl;
    return 0;
}
```

字符串简介：字符串长度

What's your name?

Xiaoping

Hello, Xiaoping. What's a super marvelous name!

Your name of 8 letters occupies 32 memory cells.

PRAISE has 30 letters and occupies 31 memory
cells.

字符串简介：字符串函数

- 头文件 `string.h` 包含许多与字符串相关的函数的原型，包括 `strlen` 函数。
- C 把函数库分成多个相关函数的序列，并为每个序列提供一个头文件。比如：
 - (1) `printf` 和 `scanf` 属于标准输入输出序列，使用 `stdio.h`。
 - (2) `strlen` 和其它一些与字符串相关的函数同属一个系列，使用 `string.h`。

字符串简介：printf 函数处理长字符串

- 一条 printf 语句占用两行，但只能在参数之间断行，不允许在字符串中间断行。
- 使用两个 printf 语句输出一行，换行符只出现在第二条语句。

字符串简介: sizeof 运算符与 strlen 返回值

设 name = "Morgan", 则

```
sizeof name : 40
```

```
strlen(name): 6
```

M	o	r	g	a	n	\0							
---	---	---	---	---	---	----	--	--	--	--	--	--	--

字符串简介：sizeof 运算符与 strlen 返回值

```
sizeof PRAISE : 29  
strlen(PRAISE): 28
```

sizeof 运算符在处理字符串变量时，会将空字符也计算在内。

字符串简介：sizeof 运算符后的圆括号

- 圆括号对于数据类型是必需的，而对于具体量则是可选的。

```
sizeof(float)  
sizeof(char)
```

```
sizeof name  
sizeof 2.15
```

- 建议在所有情况下都使用圆括号。

```
sizeof(name)  
sizeof(2.15)
```

常量与预处理器

```
// circle1.c:
#include <stdio.h>
int main(void)
{
    float radius, circum, area;
    radius = 1;
    area = 3.1415926 * radius * radius;
    circum = 2 * 3.1415926 * radius;
    printf("radius = %f, circum = %f, area = %f\n"
        , radius, circum, area);
    return 0;
}
```

常量与预处理器

```
// circle2.c:
#include <stdio.h>
int main(void)
{
    float radius, circum, area;
    float pi = 3.1415926;
    radius = 1;
    area = pi * radius * radius;
    circum = 2 * pi * radius;
    printf("radius = %f, circum = %f, area = %f\n"
        , radius, circum, area);
    return 0;
}
```

```
// circle3.c:
#include <stdio.h>
#define PI 3.1415926
int main(void) {
    float radius, circum, area;
    radius = 1;
    area = PI * radius * radius;
    circum = 2 * PI * radius;
    printf("radius = %f, circum = %f, area = %f\n"
        , radius, circum, area);
    return 0;
}
```

```
// circle4.c:
#include <stdio.h>
int main(void) {
    float radius, circum, area;
    const float PI = 3.1415926;
    radius = 1.;
    area = PI * radius * radius;
    circum = 2 * PI * radius;
    printf("radius = %f, circum = %f, area = %f\n"
, radius, circum, area);
    return 0;
}
```

常量与预处理器：宏定义

宏定义的一般形式

```
#define NAME value
```

常量与预处理器：宏定义

宏定义的一般形式

```
#define NAME value
```

注

- 不是 C 语句，不需要分号；是一条预处理指令
- 宏名使用大写，以示跟普通变量的差别。
- 宏名的命名请遵循变量命名规则。

常量与预处理器：宏定义

宏定义的一般形式

```
#define NAME value
```

注

- 不是 C 语句，不需要分号；是一条预处理指令
- 宏名使用大写，以示跟普通变量的差别。
- 宏名的命名请遵循变量命名规则。

`#define` 也可用于定义字符和字符串变量，前者用 `' '`，后者用 `" "`。

```
#define BEEP '\a'  
#define TEE 'T'  
#define ESC '\033'  
#define OOPS "Now you have done it!"
```

常见错误

```
#define B = 20
```

常量与预处理器：宏定义

常见错误

```
#define B = 20
```

此时，B 将会被 `= 20` 而不是 `20` 代替。若使用以下语句

```
c = a + B;
```

会被替换成如下错误的表达：

```
c = a + = 20;
```

定义

在宏定义中，也可以像一个“函数”一样实现某种功能，这种用法叫函数宏。

常量与预处理器：函数宏

定义

在宏定义中，也可以像一个“函数”一样实现某种功能，这种用法叫函数宏。

```
#include <stdio.h>
#define MAX(a,b) ((a) > (b) ? (a) : (b))
int main(void)
{
    printf("max(1, 2) = %d\n", MAX(1, 2));
    printf("1.0 + max(1.1+2.2, 3.0) = %.2f\n", 1.0
        + MAX(1.1+2.2, 3.0));
    return 0;
}
```

常量与预处理器：函数宏

定义

在宏定义中，也可以像一个“函数”一样实现某种功能，这种用法叫函数宏。

```
#include <stdio.h>
#define MAX(a,b) ((a) > (b) ? (a) : (b))
int main(void)
{
    printf("max(1, 2) = %d\n", MAX(1, 2));
    printf("1.0 + max(1.1+2.2, 3.0) = %.2f\n", 1.0
        + MAX(1.1+2.2, 3.0));
    return 0;
}
```

```
max(1, 2) = 2
1.0 + max(1.1+2.2, 3.0) = 4.30
```

错误的用法

```
#include <stdio.h>
#define MAX(a,b) (a) > (b) ? (a) : (b)
int main(void)
{
    printf("max(1, 2) = %d\n", MAX(1, 2));
    printf("1.0 + max(1.1+2.2, 3.0) = %.2f\n", 1.0
        + MAX(1.1+2.2, 3.0));
    return 0;
}
```

常量与预处理器：函数宏

错误的用法

```
#include <stdio.h>
#define MAX(a,b) (a) > (b) ? (a) : (b)
int main(void)
{
    printf("max(1, 2) = %d\n", MAX(1, 2));
    printf("1.0 + max(1.1+2.2, 3.0) = %.2f\n", 1.0
        + MAX(1.1+2.2, 3.0));
    return 0;
}
```

```
max(1, 2) = 2
1.0 + max(1.1+2.2, 3.0) = 3.30
```


常量与预处理器：函数宏

```
#include <stdio.h>
#define PRINT_SQUARE(x) printf("the square of "
#x " is %d.\n", (x) * (x));
int main(void)
{
    PRINT_SQUARE(3);
    PRINT_SQUARE(3+2);
    return 0;
}
```

常量与预处理器：函数宏

```
#include <stdio.h>
#define PRINT_SQUARE(x) printf("the square of "
#x " is %d.\n", (x) * (x));
int main(void)
{
    PRINT_SQUARE(3);
    PRINT_SQUARE(3+2);
    return 0;
}
```

```
the square of 3 is 9.
the square of 3+2 is 25.
```

错误的用法

```
#include <stdio.h>
#define PRINT_SQUARE(x) printf("the square of "
#x " is %d.\n", x * x);
int main(void)
{
    PRINT_SQUARE(3);
    PRINT_SQUARE(3+2);
    return 0;
}
```

常量与预处理器：函数宏

错误的用法

```
#include <stdio.h>
#define PRINT_SQUARE(x) printf("the square of "
#x " is %d.\n", x * x);
int main(void)
{
    PRINT_SQUARE(3);
    PRINT_SQUARE(3+2);
    return 0;
}
```

```
the square of 3 is 9.
the square of 3+2 is 11.
```

常量与预处理器：函数宏

```
#include <stdio.h>
#define PRINT(a) \
    do { \
        printf("%s: %d\n", #a, a); \
        printf("%d: %d\n", a, a); \
    } while(0);
int main(void)
{
    PRINT(3);
    PRINT(3+2);
    return 0;
}
```

常量与预处理器：函数宏

```
#include <stdio.h>
#define PRINT(a) \
    do { \
        printf("%s: %d\n", #a, a); \
        printf("%d: %d\n", a, a); \
    } while(0);
int main(void)
{
    PRINT(3);
    PRINT(3+2);
    return 0;
}
```

3: 3

3: 3

3+2: 5

5: 5

常量与预处理器：函数宏

```
#include <stdio.h>
#define X(n) x##n
#define PXN(n) printf("x"#n" = %d\n", x##n);
int main(void)
{
    int X(1) = 12; PXN(1);
    int X(2) = 24; PXN(2);
    int X(3) = 36; PXN(3);
    return 0;
}
```

常量与预处理器：函数宏

```
#include <stdio.h>
#define X(n)  x##n
#define PXN(n) printf("x"#n" = %d\n", x##n);
int main(void)
{
    int X(1) = 12; PXN(1);
    int X(2) = 24; PXN(2);
    int X(3) = 36; PXN(3);
    return 0;
}
```

```
x1 = 12
x2 = 24
x3 = 36
```


注意

- 宏名中不允许出现空格，因为宏定义把宏名后的第一个空格认作宏名与其替换值的分割符。
- 为了避免一些运算优先级的错误，请注意括号的使用。
- 可以跨行进行宏定义，请使用 \ 断行。

注意

- 宏名中不允许出现空格，因为宏定义把宏名后的第一个空格认作宏名与其替换值的分割符。
- 为了避免一些运算优先级的错误，请注意括号的使用。
- 可以跨行进行宏定义，请使用 \ 断行。

#与##在函数宏中的使用

- **#** 把宏参数转换为一个字符串。例如，若 `x` 是一个宏参数，则 `#x` 可把参数转换为相应的字符串，该过程称为**字符串化**。
- **##** 用于宏参数的连接。
 - 若有 `#define X(n) x##n`，则 `X(2)` 将展开为 `x2`。
 - 若有 `#define CONS(a,b) (a##e##b)`，则 `CONS(2,3)` 将展开为 `2e3`。

常量与预处理器：const 修饰符

C90 允许使用关键字 `const` 把一个变量声明转换为常量声明：

```
const int MONTHS = 12;
```

这使得 `MONTHS` 成为一个只读值。你可以显示它，并把它用于计算中，但不能改变它的值。

格式化输出

表 1: 格式说明符

格式说明符	输出
%a	浮点数、十六进制和 p-计数法
%A	浮点数、十六进制和 P-计数法
%c	字符
%d	有符号十进制数
%e	浮点数、e-计数法
%E	浮点数、E-计数法
%f	浮点数、十进制计数法
%g	根据数值不同自动选 %f 或 %e 。 %e 格式在指数小于-4 或大于等于精度时使用
%G	根据数值不同自动选 %f 或 %E 。 %E 格式在指数小于-4 或大于等于精度时使用

表 2: 格式说明符

格式说明符	输出
%i	有符号十进制整数（同%d）
%o	无符号八进制整数
%p	指针
%s	字符串
%x	使用十六进制数字 0-f 的无符号十六进制整数
%X	使用十六进制数字 0-F 的无符号十六进制整数

格式化输出

printf() 的使用格式

```
printf(Control-string, item1, item2, ...);
```

- item1, item2, ... 是要打印的项目，它们可以是变量，也可以是常量，甚至是在打印之前进行计算的表达式。
- 控制字符串 (Control-string) 是一个描述项目如何打印的字符串，它为每个要打印的项目包含一个格式说明符。

格式化输出

printf() 的使用格式

```
printf(Control-string, item1, item2, ...);
```

- item1, item2, ... 是要打印的项目，它们可以是变量，也可以是常量，甚至是在打印之前进行计算的表达式。
- 控制字符串 (Control-string) 是一个描述项目如何打印的字符串，它为每个要打印的项目包含一个格式说明符。

```
printf( "You look great in %s\n" , color );
```

控制描述

变量列表

格式化输出

printf() 的使用格式

```
printf(Control-string, item1, item2, ...);
```

- item1, item2, ... 是要打印的项目，它们可以是变量，也可以是常量，甚至是在打印之前进行计算的表达式。
- 控制字符串 (Control-string) 是一个描述项目如何打印的字符串，它为每个要打印的项目包含一个格式说明符。

```
printf( "You look great in %s\n" , color );
```

控制描述

变量列表

注意

不要忘记给控制字符串后列表中的每个项目都使用一个格式说明符。

格式化输出

- 如果只想打印一个语句，则不需要任何格式说明符；
- 如果只想打印数据，则无须加入任何说明内容。
- 想打印 %，必须使用两个 %% 符号。

```
printf("Once more you open the door!\n");  
printf("%s%d\n", "area = ", area);  
printf("%d%% = %f\n", 30, 0.3);
```

格式化输出：格式说明符%d

- %d: 按整型数据的实际长度输出
- %md: 输出字段的宽度为 m ，右对齐
若数据位数 $< m$ ，左端补空格；若 $\geq m$ ，按实际位数输出。
- %-md: 输出字段的宽度为 m ，左对齐
若数据位数 $< m$ ，右端补空格；若 $\geq m$ ，按实际位数输出。
- %0md: 输出字段的宽度为 m ，右对齐
若数据位数 $< m$ ，右端补 0；若 $\geq m$ ，按实际位数输出。

格式化输出: %d

```
// width.c:
#include <stdio.h>
#define N 1000
int main(void)
{
    printf("%d*\n", N);
    printf("%2d*\n", N);
    printf("%10d*\n", N);
    printf("%-10d*\n", N);
    printf("%010d*\n", N);
    return 0;
}
```

格式化输出: %d

```
// width.c:
#include <stdio.h>
#define N 1000
int main(void)
{
    printf("%d*\n", N);
    printf("%2d*\n", N);
    printf("%10d*\n", N);
    printf("%-10d*\n", N);
    printf("%010d*\n", N);
    return 0;
}
```

```
*1000*
*1000*
*      1000*
*1000      *
*0000001000*
```

格式化输出: %f, %e, %E

```
// floats.c:
#include <stdio.h>
int main(void)
{
    const double RENT = 3852.42;
    printf("%f*\n", RENT);
    printf("%e*\n", RENT);
    printf("%.2f*\n", RENT);
    printf("%.1f*\n", RENT);
    printf("%.3f*\n", RENT);
    printf("%.3e*\n", RENT);
    printf("%.3E*\n", RENT);
    printf("%.4f*\n", RENT);
    printf("%.2f*\n", RENT);
    printf("%.0f*\n", RENT);
    printf("%.f*\n", RENT);
    printf("%.4f*\n", RENT);
    return 0;
}
```

格式化输出：%f, %e, %E

```
*3852.990000*  
*3.852990e+03*  
*3852.99*  
*3853.0*  
*  3852.990*  
* 3.853e+03*  
* 3.853E+03*  
*+3852.99*  
*3852.99   *  
*0003852.99*
```

格式化输出：%f，%e，%E

`%m.nf`

`%m.ne`

`%m.nE`

- m 为字段宽度
- n 为小数点右边数字的个数

格式化输出：%f，%e，%E

`%m.nf`

`%m.ne`

`%m.nE`

- m 为字段宽度
- n 为小数点右边数字的个数

`%.nf`

- 整数部分以实际长度输出
- n 为小数点右边数字的个数

格式化输出：%f，%e，%E

`%m.nf`

`%m.ne`

`%m.nE`

- m 为字段宽度
- n 为小数点右边数字的个数

`%.nf`

- 整数部分以实际长度输出
- n 为小数点右边数字的个数

`%m.f`

- 字段宽度为 m
- 不输出小数点后的数字

格式化输出 I

```
// flags.c:
#include <stdio.h>
int main(void)
{
    printf("%x %X %#x %#X\n", 31, 31, 31, 31);
    printf("%d*\n", 42);
    printf("% d*\n", 42);
    printf("% d*\n", -42);
    printf("%5d*\n", 6);
    printf("%5.3d*\n", 6);
    printf("%05d*\n", 6);
    printf("%05.3d*\n", 6);
    return 0;
}
```

格式化输出

```
1f_1F_0x1f_0X1F
```

```
*42*
```

```
*_42*
```

```
*-42*
```

```
*_ _ _ _ 6*
```

```
*_ _ 006*
```

```
*00006*
```

```
*_ _ 006*
```

- %x, %#x, %X, %#X的用法

- %x 输出 1f
- %X 输出 1F
- %#x 输出 0x1f
- %#X 输出 0X1F

- % d的用法

- 正值之前产生一个前导空格；
- 在负值之前不产生前导空格。

这使得有效位相同的正值和负值以相同字段宽度打印输出。

- %m.nd, %0md, %0m.nd 的用法

- %5.3d 用于在整数格式中来产生足够的前导零以填满要求的最小数字位数。
- %05d 将会用前导零填满整个字段宽度。
- 在 %05.3d 中，0 标志和精度说明符同时出现，此时 0 标志将会忽略。

格式化输出: %s

```
// strings.c:
#include <stdio.h>
#define WORD "Hello World!"
int main(void)
{
    printf("%2s*\n", WORD);
    printf("%15s*\n", WORD);
    printf("%15.5s*\n", WORD);
    printf("%*-15.5s*\n", WORD);
    printf("%%.13s*\n", WORD);
    return 0;
}
```

格式化输出: %s

```
// strings.c:
#include <stdio.h>
#define WORD "Hello World!"
int main(void)
{
    printf("%2s*\n", WORD);
    printf("%15s*\n", WORD);
    printf("%15.5s*\n", WORD);
    printf("%*-15.5s*\n", WORD);
    printf("%.13s*\n", WORD);
    return 0;
}
```

```
*Hello_World!*
*   _Hello_World!*
* _ _ _ _ _ _ _ _ _ _Hello*
*Hello_ _ _ _ _ _ _ _ _ _*
*Hello_World!*

```

格式化输出：printf()的返回值

```
#include <stdio.h>
int main(void)
{
    int bph2o = 100;
    int rv;
    rv = printf("Hello\n");
    printf("the printf function printed %d
character.\n", rv);
    return 0;
}
```


格式化输出：printf()的返回值

```
#include <stdio.h>
int main(void)
{
    int bph2o = 100;
    int rv;
    rv = printf("Hello\n");
    printf("the printf function printed %d
character.\n", rv);
    return 0;
}
```

```
100 C is water's boiling point.
the printf function printed 32 character.
```

printf() 返回所有打印字符的个数，包括空格和不可见的换行符。

格式化输出：printf() 中的 %n

在 printf() 中，使用 %n 将获取 %n 出现之前的所有字符的个数，并将其传递给后面对应的变量。

格式化输出：printf() 中的 %n

在 printf() 中，使用 %n 将获取 %n 出现之前的所有字符的个数，并将其传递给后面对应的变量。

```
// printf_n.c:
#include <stdio.h>
int main(void)
{
    int c1, c2;
    printf("Hello Wuhan %nUniversity!%n\n", &c1, &c2);
    printf("c1 = %d, c2 = %d\n", c1, c2);
    return 0;
}
```

格式化输出：printf() 中的 %n

在 printf() 中，使用 %n 将获取 %n 出现之前的所有字符的个数，并将其传递给后面对应的变量。

```
// printf_n.c:
#include <stdio.h>
int main(void)
{
    int c1, c2;
    printf("Hello Wuhan %nUniversity!%n\n", &c1, &c2);
    printf("c1 = %d, c2 = %d\n", c1, c2);
    return 0;
}
```

```
Hello Wuhan University!
c1 = 12, c2 = 23
```