



# 第五章 程序设计复合类型

## 模块5.2：字符数组与字符串 (C风格的字符串)

主讲教师：同济大学电子与信息工程学院 陈宇飞  
同济大学电子与信息工程学院 龚晓亮



# 目录

- 字符数组
- 字符串
- 编译预处理



# 目录

- 字符数组
  1. 基本概念
  2. 字符数组的初始化



# 1.1 字符数组的基本概念

- ✓ 数据类型为字符型的数组（理解同模块5.1）

`char` arrayName[arraySize]



## 1.2 字符数组的初始化

### ✓ 全部元素赋初值

```
char dog[8]={' b', ' e', ' a', ' u', ' x', '', ' I', ' I'};
```

```
char dog[]={ ' b', ' e', ' a', ' u', ' x', '', ' I', ' I'}; //8
```

```
char dog[3]={' b', ' e', ' a', ' u', ' x', '', ' I', ' I'}; //illegal
```



## 1.2 字符数组的初始化

### ✓ 全部元素赋初值

```
char dog[2][4]={' b',' e',' a',' u',' x',' ',' I',' I'};
```

//也可写为{{' b',' e',' a',' u'}, {' x',' ',' I',' I'}};

```
char dog[][4]={' b',' e',' a',' u',' x',' ',' I',' I'}; //2
```

➤ 只可缺省行数，不可省略列数



## 1.2 字符数组的初始化

### ✓ 部分元素赋初值

```
char dog[8]={' b', ' e', ' a', ' u' };
```

b	e	a	u	\0	\0	\0	\0
---	---	---	---	----	----	----	----



# 1.2 字符数组的初始化

## ✓ 部分元素赋初值

```
char dog[2][4]={' b', ' e' };
```

b	e	\0	\0
\0	\0	\0	\0

```
char dog[2][4]={{' b'}, {' e' }};
```

b	\0	\0	\0
e	\0	\0	\0





# 目录

- 字符串
  1. 基本概念
  2. 字符串的处理方式
  3. 拼接字符串常量
  4. 在数组中使用字符串
  5. 字符串输入
  6. 每次读取一行字符串



## 2.1 字符串的基本概念

✓ 字符串是存储在内存的连续字节中的一系列字符

存储在char数组中 每个字符都是数组元素

```
char cat[8]= "fatessa";
```

f	a	t	e	s	s	a	\0
---	---	---	---	---	---	---	----



## 2.2 字符串的处理方式

### ✓ 字符串的处理方式

❖ C-风格字符串 (C-style string)

**特殊性质：**空字符结尾 `\0`

❖ 基于string类库 (模块5.3)



## 2.2 字符串的处理方式

### ❖ C-风格字符串

```
char dog[8]={ 'b', 'e', 'a', 'u', 'x', '', 'I', 'I' }; //not a string
```

```
char cat[8]={ 'f', 'a', 't', 'e', 's', 's', 'a', '\0' }; //a string!
```



```
char cat[8]= "fatessa"; //the \0 is understood
```

```
char cat[]="fatessa"; //let the compiler count
```



字符串常量 (string constant)

字符串字面值 (string literal)



## 2.2 字符串的处理方式

### ❖ C-风格字符串

```
char dog[8]={'b','e','a','u','x','','I','I'};
```

//不是字符串，字符数组长度为8

```
char cat[8]={'f','a','t','e','s','s','a','\0'};
```

//字符串长度为7，字符数组长度为8

```
char cat[8]={'f','a','t','\0','e','s','s','a'};
```

//字符串长度为3，字符数组长度为8

→ 字符串的长度：在'\0'之前的实际长度 **strlen**

字符数组的长度：数组的大小 **sizeof**



## 2.2 字符串的处理方式

### ❖ C-风格字符串

```
char s[10] = "I am fine";
```

□ s是字符串，' \0 ' 系统自动添加

□ 应确保数组足够大，能够存储字符串中所有字符（包括空字符）

```
char s[9] = "I am fine"; //illegal
```

□ 字符串常量（双引号）不能与字符常量（单引号）互换

```
char shirt_size = 'S'; //this is fine
```

```
char shirt_size = "S"; //illegal type mismatch
```

**"S"表示的是字符串所在的内存地址**

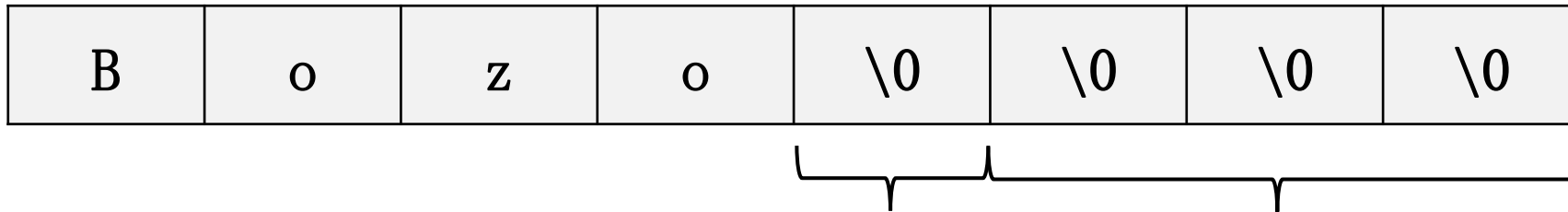


## 2.2 字符串的处理方式

### ❖ C-风格字符串

#### □ 字符串常量初始化字符数组

```
char boss[8] = "Bozo";
```



空值字符自  
动加到结尾

其余的元素  
设置为\0



# 2.2 字符串的处理方式

❖ C-风格字符串

□ 字符串常量初始化字符数组

```
char course[4][8] = {"C++", "Java", "Python", "VB"};  
char course[4][8] = {{ "C++"}, { "Java"}, { "Python"}, { "VB"} };
```

C	+	+	\0	\0	\0	\0	\0
J	a	v	a	\0	\0	\0	\0
P	y	t	h	o	n	\0	\0
V	B	\0	\0	\0	\0	\0	\0





## 2.2 字符串的处理方式

### ❖ C-风格字符串

❑ 字符数组不允许整体进行赋值操作，只能单个元素依次赋值

```
char s[10];
```

```
s = "I am fine";           //illegal 数组名s表示的是字符串所在的内存地址
```

```
s = {'I', ' ', 'a', 'm', ' ', 'f', 'i', 'n', 'e'}; //illegal
```

```
s[2] = 'A';                //this is fine
```

❑ 字符串使用专用函数进行操作（见补充材料）



## 2.3 拼接字符串常量

```
cout<< "I'd give my right arm to be" " a great violinist.\n";
```

```
cout<< "I'd give my right arm to be a great violinist.\n";
```

```
cout<< "I'd give my right ar"
```

```
"m to be a great violinist.\n";    //三者等效
```

- ❖ 任何两个由空白（空格、制表符和换行符）分隔的字符串常量将自动拼接
- ❖ 拼接时不会在被连接的字符串之间添加空格
- ❖ 第一个字符串中的\0字符将被第二个字符串的第一个字符取代



## 2.4 在数组中使用字符串

### ✓ 字符串存储到数组的方法

#### ❖ 将数组初始化为字符串常量

```
char name2[15]= "C++owboy";
```

#### ❖ 将键盘或文件输入读入到数组中

```
cin >> name1; //键盘输入其中的一种方式
```

文件输入（后续课程介绍）；



## 2.4 在数组中使用字符串

```
...  
  
#include <cstring> //for strlen()  
  
int main()  
{  
  
    const int Size=15;  
  
    char name1[Size]; //empty array  
  
    char name2[Size]="C++owboy";  
  
                                //initialized array  
  
    ...cin >> name1;  
  
    cout << strlen(name1) << ...;
```

```
    cout << ... << sizeof(name1) << ...;  
  
    name2[3] = ' \0' ;  
  
    ...  
  
    cout << name2 << endl;  
  
    return 0;  
  
}
```

```
Howdy! I'm C++owboy! What's your name?  
Basicman  
well, Basicman, your name has 8 letters and is stored  
in an array of 15 bytes  
Your initial is B.  
Here are the first 3 characters of my name: C++
```



## 2.4 在数组中使用字符串

```
...  
#include <cstring> //for strlen()  
int main()  
{  
    const int Size=15;  
    char name1[Size]; //empty array  
    char name2[Size]="C++owboy";  
                                //initialized array  
    ...cin >> name1;  
    cout << strlen(name1) << ...;
```

```
    cout << ... << sizeof(name1) << ...;  
    name2[3] = '\\0';  
    ...  
    cout << name2 << endl;  
    return 0;  
}
```

- sizeof运算符指出整个数组的长度
- strlen() 函数返回的是存储在数组中的字符串的长度，并且只计算可见的字符，不把空字符计算在内



## 2.4 在数组中使用字符串

```
...  
#include <cstring> //for strlen()  
int main()  
{  
  
    const int Size=15;  
  
    char name1[Size]; //empty array  
    char name2[Size]="C++owboy";  
                        //initialized array  
  
    ...cin >> name1;  
  
    cout << strlen(name1) << ...;
```

```
    cout << ... << sizeof(name1) << ...;  
    name2[3] = ' \0' ;  
    ...  
  
    cout << name2 << endl;  
    return 0;  
}
```

- 程序多次使用了数组长度，用符号常量表示数组长度后，便于修改和维护



## 2.4 在数组中使用字符串

```
...  
#include <cstring> //for strlen()  
int main()  
{  
    const int Size=15;  
    char name1[Size]; //empty array  
    char name2[Size]="C++owboy";  
                                //initialized array  
    ...cin >> name1;  
    cout << strlen(name1) << ...;
```

```
    cout << ... << sizeof(name1) << ...;  
    name2[3] = '\0';  
    ...  
    cout << name2 << endl;  
    return 0;  
}
```

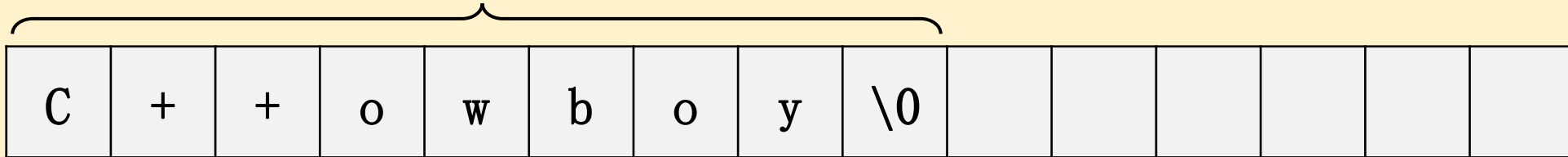
- 可以用索引来访问数组中各个字符，如：  
name2[3]即为数组name2的第3个元素
- name2[3] = '\0' 语句使用\0 截短字符串



## 2.4 在数组中使用字符串

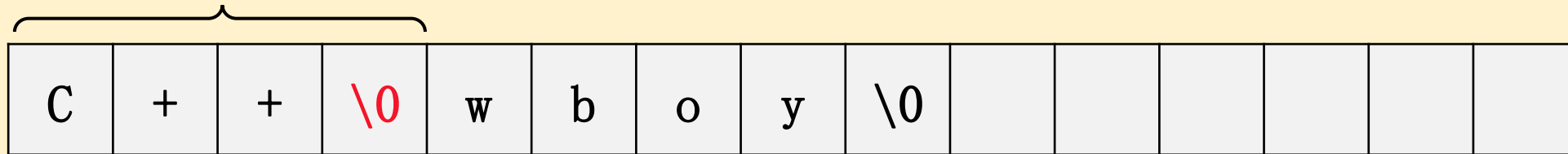
```
const int ArSize = 15;  
char name2[ArSize] = "C++owboy";
```

字符串



```
name2[3] = '\0'; //使用\0 截短字符串
```

字符串



忽略





## 2.4 在数组中使用字符串

```
...  
#include <cstring> //for strlen()  
int main()  
{  
  
    const int Size=15;  
    char name1[Size]; //empty array  
    char name2[Size]="C++owboy";  
                                //initialized array  
    ...cin >> name1;  
    cout << strlen(name1) << ...;
```

```
    cout << ... << sizeof(name1) << ...;  
    name2[3] = ' \0' ;  
    ...  
    cout << name2 << endl;  
    return 0;  
}
```

- 字符串name2在第三个字符后即结束，虽然数组中还有其他的字符



## 2.4 在数组中使用字符串

```
//instr1.cpp
#include <iostream>
using namespace std;
int main()
{
    const int ArSize = 20;
    char name[ArSize];
    char dessert[ArSize];
    cout << "Enter your name:\n";
    cin >> name;
```

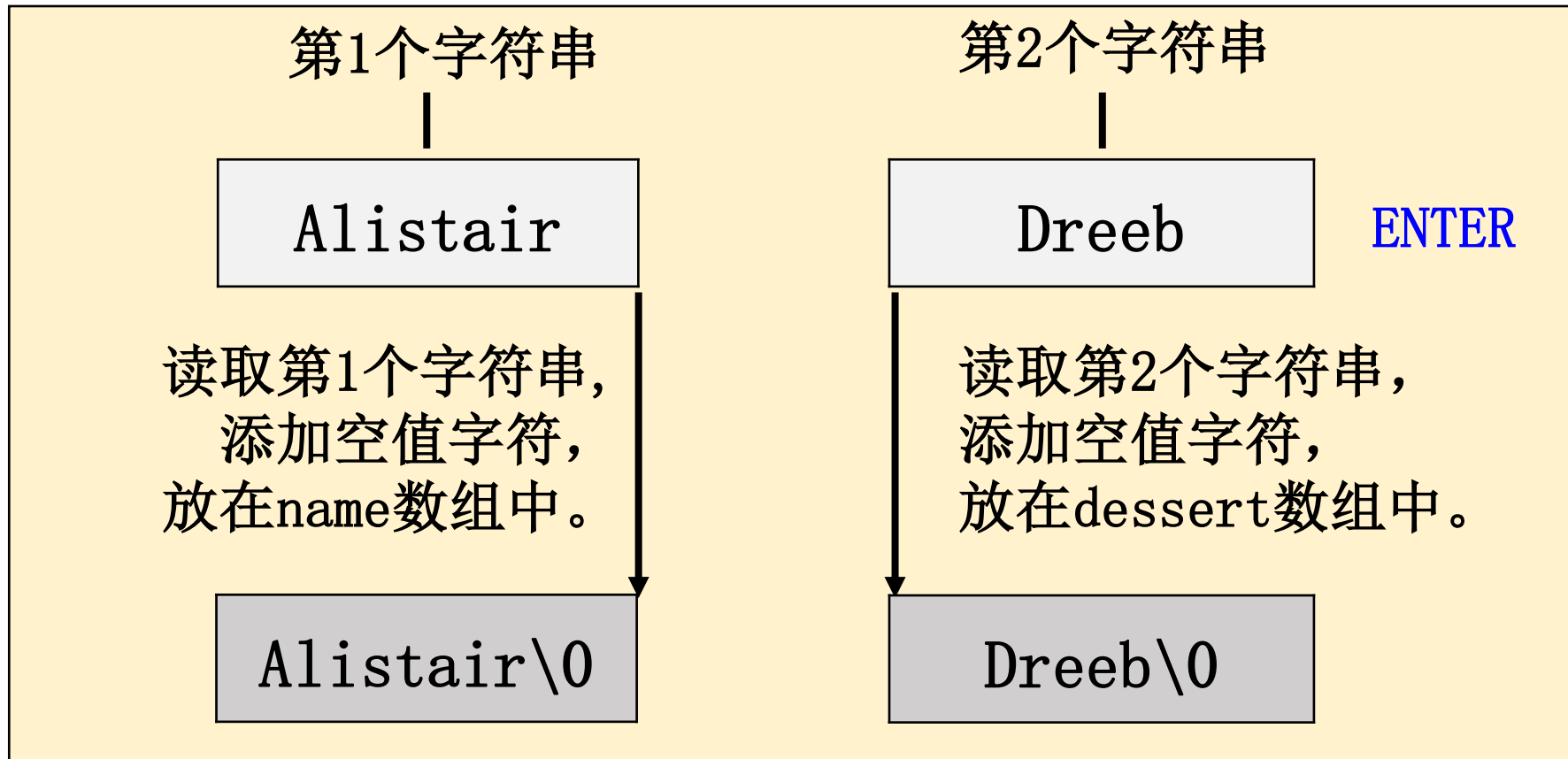
```
    cout << "Enter your favorite
            dessert:\n";
    cin >> dessert;
    cout << "I have some delicious " <<
            dessert << " for you. " <<
            name << ".\n";
    return 0;
}
```

```
Enter your name:
Alistair Dreeb
Enter your favorite dessert:
I have some delicious Dreeb for you. Alistair.
```



## 2.5 字符串输入

- ✓ `cin`使用空白（空格、制表符和换行符）来确定字符串的结束位置





## 2.5 字符串输入

- ✓ cin使用空白（空格、制表符和换行符）来确定字符串的结束位置
- ✓ 输入字符串不能比目标数组长，否则越界报错

```
cout << "Enter your name:\n";  
cin >> name;  
cout << "Enter your favorite dessert:\n";  
cin >> dessert;  
cout << "I have some delicious " << dessert << " for you. " << name << ".\n";  
return 0;
```

已引发异常

Run-Time Check Failure #2 - Stack around the variable 'name' was corrupted.

```
Enter your name:  
Aliiiiiiiiiiiiiistair Dreeb  
Enter your favorite dessert:  
I have some delicious Dreeb for you. Aliiiiiiiiiiiiiistair.
```



## 2.6 每次读取一行字符串

### ✓ 面向行的类成员函数

#### ❖ `getline()` 函数

读取一行输入，直到到达换行符，并**丢弃**换行符；

#### ❖ `get()` 函数

读取一行输入，直到到达换行符，将换行符**保留**在输入序列；

### ✓ C语言的函数

#### ❖ `gets()`, `gets_s()`, `fgets()` 函数



## 2.6 每次读取一行字符串

### ❖ getline() 函数

- ❑ 通过回车键输入的换行符来确定输入结束;
- ❑ 函数有两个参数:

`cin.getline(name, 20)`

输入行的数组的名称    要读取的字符数（最多读取19个字符）

第三个可选参数（后续课程内容）；

- ❑ 自动在结尾处添加空字符\0;



## 2.6 每次读取一行字符串

```
//instr2.cpp
#include <iostream>
using namespace std;
int main()
{
    const int ArSize = 20;
    char name[ArSize];
    char dessert[ArSize];
    cout << "Enter your name:\n";
    cin.getline(name, ArSize);
```

```
    cout << "Enter your favorite
            dessert:\n";
    cin.getline(dessert, ArSize);
    cout << "I have some delicious " <<
            dessert << " for you. " <<
            name << ".\n";
    return 0;
}
```

```
Enter your name:
Dirk Hammernose
Enter your favorite dessert:
Radish Torte
I have some delicious Radish Torte for you. Dirk Hammernose.
```



## 2.6 每次读取一行字符串

- ❑ `getline()` 函数通过换行符来确定行尾，但不保存换行符。存储字符串时，用空字符来替换换行符

```
char name[10];  
cout << "Enter your name: ";  
cin.getline(name, 10);
```

用户键入Jud来作出响应，然后按下

Enter your name: Jud

`cin.getline()` 读取“Jud”以及用户按ENTER键而生成的换行符，并将换行符替换为\0

J	u	d	\0						
---	---	---	----	--	--	--	--	--	--





## 2.6 每次读取一行字符串

### ❖ get() 函数

➤ 该函数有多种变体：无参，一参，两参

```
cin.get(name, ArSize);  
cin.get(dessert, ArSize); //a problem
```

get(...)与getline(...)类似，  
但不再读取并丢弃换行符，而  
是将其留在输入队列中

```
cin.get(name, ArSize); //read first line  
cin.get(); //read new line  
cin.get(dessert, ArSize); //read second line
```

不带参的get()可读取下一个  
字符（即使是换行符）

```
cin.get(name, ArSize).get(); //concatenate member functions
```



## 2.6 每次读取一行字符串

```
//instr3.cpp
#include <iostream>
using namespace std;
int main()
{
    const int ArSize = 20;
    char name[ArSize];
    char dessert[ArSize];
    cout << "Enter your name:\n";
    cin.get(name, ArSize).get();
```

```
    cout << "Enter your favorite
            dessert:\n";
    cin.get(dessert, ArSize).get();
    cout << "I have some delicious " <<
            dessert << " for you. " <<
            name << ".\n";
    return 0;
}
```

```
Enter your name:
Mai Parfait
Enter your favorite dessert:
Chocolate Mousse
I have some delicious Chocolate Mousse for you. Mai Parfait.
```



## 2.6 每次读取一行字符串

```
// numstr.cpp
...

cout << "What year was your house built?\n";
int year;

cin >> year;    //此句需要修改

cout << "What is its street address?\n";
char address[80];

cin.getline(address, 80);

cout << "Year built: " << year << endl;
cout << "Address: " << address << endl;
cout << "Done!\n";

...
```

```
What year was your house built?
1966
What is its street address?
Year built: 1966
Address:
Done!
```

修改方法一:

```
cin >> year;
cin.get();    //or cin.get(ch)
```

修改方法二:

```
(cin >> year).get();
//or (cin >> year).get(ch);
```



## 2.6 每次读取一行字符串

- 每次读取一行字符串

- gets() 函数

- 从stdin流读取字符串;
    - 通过回车键输入的换行符来确定输入结束;
    - 函数原型: `char* gets(char* buffer);`
    - **丢弃换行符**, 储存其余字符, 自动在结尾处添加空字符\0;
    - 函数**不安全**, 需确保buffer的空间足够大, 才不发生溢出;



## 2.6 每次读取一行字符串

### ❖ gets\_s() 函数

- ❑ 从stdin流读取字符串;

- ❑ 函数原型: `gets_s(char *p, int n);`

//n表示其最多读取的数量, 一般为数组大小;

- ❑ 读到换行符, 会丢弃它;

- ❑ 如果输入行太长, `gets_s()` 会丢弃该输入行的其余字符;



## 2.6 每次读取一行字符串

### ❖ fgets() 函数

#### ➤ 函数原型:

```
char *fgets(char *buf, int bufsz, FILE *stream);
```

//指明了读入字符的最大数量: bufsz-1或遇到换行符;

#### ➤ 主要用于处理文件输入;

#### ➤ 若读入从键盘输入的数据, 第三个参数为stdin;

#### ➤ 读到换行符, 会把它存储到字符串中;



## 2.6 每次读取一行字符串

```
#include <iostream>
using namespace std;
int main()
{
    char s[80];
    cout << "Please input your string with spaces for tesing: \n";
    //gets(s);    //Only Dev is ok
    //gets_s(s); //Only VS is ok
    fgets(s, 20, stdin); //Both Dev and VS are ok. The result is different with gets/gets_s
    cout << "Your input is: " << s << "###" << endl;
    return 0;
}
```

在双编译环境测试程序并比较:

1. 不同编译环境下的函数运行;
2. fgets与其他两个函数的区别;



## 2.6 每次读取一行字符串

✓ 思考：如果双编译器，读入一行字符串（带空格）

□VS : gets\_s

□Dev: gets

如何共存于一个源程序中？即实现程序跨平台？

❖ 方法一：Dev/VS: fgets

但fgets 函数读到换行符，就会把它存储到字符串中，而不是像 gets 函数那样丢弃它。

❖ 方法二：条件编译





# 目录

- 编译预处理
  1. 基本概念
  2. 条件编译



## 3.1 编译预处理的基本概念

- ✓预处理是C语言特有的功能，它是在对源程序正式编译前由预处理程序完成的。程序员在程序中用预处理命令来调用这些功能
- ✓宏定义是用一个标识符来表示一个字符串，这个字符串可以是常量、变量或表达式。在宏调用中将用该字符串代换宏名
- ✓文件包含是预处理的一个重要功能，它可用来把多个源文件连接成一个源文件进行编译，结果将生成一个目标文件
- ✓条件编译允许只编译源程序中满足条件的程序段，使生成的目标程序较短，从而减少了内存的开销并提高了程序的效率



## 3.2 条件编译

✓避免在同一个文件中将同一个头文件包含多次

```
//coordin.h    使用预处理器编译指令  
  
#ifndef COORDIN_H_  
  
#define COORDIN_H_  
  
//place include file contents here  
  
#endif
```

✓编译时：让编译器只对满足条件的代码进行编译，将不满足条件的代码舍弃，使生成的目标程序变小



## 3.2 条件编译

### ✓ 条件编译

❖编译时：让编译器只对满足条件的代码进行编译，将不满足条件的代码舍弃，使生成的目标程序变小，从而减少了内存的开销并提高了程序的效率

### ✓ 分支语句

❖执行时：根据相应的条件选择不同的代码执行。在编译阶段，全部代码生成目标程序，并没有减少内存的开销



## 3.2 条件编译

### ✓ 预处理器编译指令

#define	定义一个预处理宏
#undef	取消宏的定义
#if	编译预处理中的条件命令
#ifdef	判断某个宏是否被定义，若已定义，执行随后的语句
#ifndef	与#ifdef相反，判断某个宏是否未被定义
#elif	若#if, #ifdef, #ifndef或前面的#elif条件不满足，则执行#elif之后的语句
#else	与#if, #ifdef, #ifndef对应，若这些条件不满足，则执行#else之后的语句
#endif	#if, #ifdef, #ifndef这些条件命令的结束标志
defined	与#if, #elif配合使用，判断某个宏是否被定义



## 3.2 条件编译

### ✓ 条件编译的三种形式

#ifdef 标识符

程序段1

#else

程序段2

#endif

如果标识符已被#define命令定义过，则对程序段1进行编译；否则对程序段2进行编译。

#ifndef 标识符

程序段1

#else

程序段2

#endif

如果标识符未被#define命令定义过，则对程序段1进行编译，否则对程序段2进行编译。

#if 表达式

程序段1

#else

程序段2

#endif

如常量表达式的值为真(非0)，则对程序段1进行编译，否则对程序段2进行编译。



## 3.2 条件编译

✓只定义宏而没有指定宏的值时，要检查宏不能使用#if、#elif，而应当使用#ifdef或#ifndef，或者使用#if defined(宏)、#if !defined()

```
#include<stdio.h>

#define B

#if A
    #define HELLO "hello world"
#elif B
    #define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

//编译错

abc E0029 应输入表达式  
abc E0020 未定义标识符 "HELLO"  
✗ C1017 无效的整数常量表达式

```
#include<stdio.h>

#define B

#if A
    #define HELLO "hello world"
#elif defined(B)
    #define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

//hello gaocheng



## 3.2 条件编译

✓只定义宏而没有指定宏的值时，要检查宏不能使用`#if`、`#elif`，而应当使用`#ifdef`或`#ifndef`，或者使用`#if defined(宏)`、`#if !defined()`

```
#include<stdio.h>

#define A

#ifdef A
    #define HELLO "hello world"
#else
    #define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

未被编译

//hello world

```
#include<stdio.h>

#define A

#ifndef A
    #define HELLO "hello world"
#else
    #define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

未被编译

//hello gaocheng





## 3.2 条件编译

✓ 定义宏且指定宏的值时，要检查宏可以使用#if、#elif

```
#include<stdio.h>

#define A 3

#if A
#define HELLO "hello world"
#elif B
#define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

未被编译

//hello world

```
#include<stdio.h>

#define B 3

#if A
#define HELLO "hello world"
#elif B
#define HELLO "hello gaocheng"
#endif

int main()
{
    printf("%s\n", HELLO);
    return 1;
}
```

未被编译

//hello gaocheng



## 3.2 条件编译

✓ 前面思考题：用标识不同编译器的预置宏定义来区分不同编译环境

```
...  
  
#if (__GNUC__)    //Dev (GNU GCC)  
    gets(...);  
#elif (_MSC_VER) //Microsoft Visual C/C++  
    gets_s(...);  
#endif  
...
```



# 总结

- 字符数组

1. 基本概念
2. 字符数组的初始化

- 字符串

1. 基本概念
2. 字符串的处理方式
3. 拼接字符串常量
4. 在数组中使用字符串
5. 字符串输入
6. 每次读取一行字符串

- 编译预处理

1. 基本概念
2. 条件编译