

其实之前做了纸质版的笔记，但是没有带过来，不方便使用。于是就直接把黑马的笔记放了上来。
视频链接：

[黑马程序员匠心之作|C++教程从0到1入门编程,学习编程不再难](#)

需要PDF的可以下载：[C++PDF](#)

C++初识

注释

作用：在代码中加一些说明和解释，方便自己或其他程序员程序员阅读代码

两种格式

1. **单行注释：** `// 描述信息`
 - 通常放在一行代码的上方，或者一条语句的末尾，对该行代码说明
2. **多行注释：** `/* 描述信息 */`
 - 通常放在一段代码的上方，对该段代码做整体说明

提示：编译器在编译代码时，会忽略注释的内容

变量

作用：给一段指定的内存空间起名，方便操作这段内存

语法： `数据类型 变量名 = 初始值;`

示例：

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5
6      //变量的定义
7      //语法：数据类型 变量名 = 初始值
8
9      int a = 10;
10
11     cout << "a = " << a << endl;
12
13     system("pause");
14
15     return 0;
16 }
```

注意：C++在创建变量时，必须给变量一个初始值，否则会报错

常量

作用：用于记录程序中不可更改的数据

C++定义常量两种方式

1. **#define** 宏常量：`#define 常量名 常量值`
 - 通常在文件上方定义，表示一个常量
2. **const**修饰的变量 `const 数据类型 常量名 = 常量值`
 - 通常在变量定义前加关键字const，修饰该变量为常量，不可修改

示例：

```
1 //1、宏常量
2 #define day 7
3
4 int main() {
5
6     cout << "一周里总共有 " << day << " 天" << endl;
7     //day = 8; //报错，宏常量不可以修改
8
9     //2、const修饰变量
10    const int month = 12;
11    cout << "一年里总共有 " << month << " 个月份" << endl;
12    //month = 24; //报错，常量是不可以修改的
13
14
15    system("pause");
16
17    return 0;
18 }
```

关键字

作用：关键字是C++中预先保留的单词（标识符）

- 在定义变量或者常量时候，不要用关键字

C++关键字如下：

| asm | do | if | return | typedef |
|-------|--------------|-----------|-------------|----------|
| auto | double | inline | short | typeid |
| bool | dynamic_cast | int | signed | typename |
| break | else | long | sizeof | union |
| case | enum | mutable | static | unsigned |
| catch | explicit | namespace | static_cast | using |
| char | export | new | struct | virtual |
| | | | | |

| | | | | |
|--------------|--------------|------------------|------------------|-----------------|
| class asm | extern do | operator if | switch return | void typedef |
| const | false | private | template | volatile |
| const_cast | float | protected | this | wchar_t |
| continue | for | public | throw | while |
| default | friend | register | true | |
| delete | goto | reinterpret_cast | try | |

提示：在给变量或者常量起名称时候，不要用C++得关键字，否则会产生歧义。

标识符命名规则

作用：C++规定给标识符（变量、常量）命名时，有一套自己的规则

- 标识符不能是关键字
- 标识符只能由字母、数字、下划线组成
- 第一个字符必须为字母或下划线
- 标识符中字母区分大小写

建议：给标识符命名时，争取做到见名知意的效果，方便自己和他人的阅读

数据类型

C++规定在创建一个变量或者常量时，必须要指定出相应的数据类型，否则无法给变量分配内存

整型

作用：整型变量表示的是整数类型的数据

C++中能够表示整型的类型有以下几种方式，**区别在于所占内存空间不同：**

| 数据类型 | 占用空间 | 取值范围 |
|-----------------|-------------------------------------|---------------------------|
| short(短整型) | 2字节 | $(-2^{15} \sim 2^{15}-1)$ |
| int(整型) | 4字节 | $(-2^{31} \sim 2^{31}-1)$ |
| long(长整形) | Windows为4字节，Linux为4字节(32位)，8字节(64位) | $(-2^{31} \sim 2^{31}-1)$ |
| long long(长长整形) | 8字节 | $(-2^{63} \sim 2^{63}-1)$ |

sizeof关键字

作用：利用sizeof关键字可以统计数据类型所占内存大小

语法： sizeof(数据类型 / 变量)

示例：

```
1  int main() {
2
3      cout << "short 类型所占内存空间为: " << sizeof(short) << endl;
4
5      cout << "int 类型所占内存空间为: " << sizeof(int) << endl;
6
7      cout << "long 类型所占内存空间为: " << sizeof(long) << endl;
8
9      cout << "long long 类型所占内存空间为: " << sizeof(long long) << endl;
10
11     system("pause");
12
13     return 0;
14 }
```

整型结论： short < int <= long <= long long

实型（浮点型）

作用：用于表示小数

浮点型变量分为两种：

1. 单精度float
2. 双精度double

两者的**区别**在于表示的有效数字范围不同。

| 数据类型 | 占用空间 | 有效数字范围 |
|--------|------|------------|
| float | 4字节 | 7位有效数字 |
| double | 8字节 | 15~16位有效数字 |

示例：

```
1  int main() {
2
3      float f1 = 3.14f;
4      double d1 = 3.14;
5
6      cout << f1 << endl;
7      cout << d1 << endl;
8
9      cout << "float sizeof = " << sizeof(f1) << endl;
10     cout << "double sizeof = " << sizeof(d1) << endl;
11
12     //科学计数法
```

```

13     float f2 = 3e2; // 3 * 10 ^ 2
14     cout << "f2 = " << f2 << endl;
15
16     float f3 = 3e-2; // 3 * 0.1 ^ 2
17     cout << "f3 = " << f3 << endl;
18
19     system("pause");
20
21     return 0;
22 }

```

字符型

作用：字符型变量用于显示单个字符

语法： `char ch = 'a';`

注意1：在显示字符型变量时，用单引号将字符括起来，不要用双引号

注意2：单引号内只能有一个字符，不可以是字符串

- C和C++中字符型变量只占用1个字节。
- 字符型变量并不是把字符本身放到内存中存储，而是将对应的ASCII编码放入到存储单元

示例：

```

1  int main() {
2
3      char ch = 'a';
4      cout << ch << endl;
5      cout << sizeof(char) << endl;
6
7      //ch = "abcde"; //错误，不可以用双引号
8      //ch = 'abcde'; //错误，单引号内只能引用一个字符
9
10     cout << (int)ch << endl; //查看字符a对应的ASCII码
11     ch = 97; //可以直接用ASCII给字符型变量赋值
12     cout << ch << endl;
13
14     system("pause");
15
16     return 0;
17 }

```

转义字符

作用：用于表示一些不能显示出来的ASCII字符

现阶段我们常用的转义字符有： `\n` `\\` `\t`

| 转义字符 | 含义 | ASCII码值（十进制） |
|------|---------------------------|--------------|
| \a | 警报 | 007 |
| \b | 退格(BS)，将当前位置移到前一个 | 008 |
| \f | 换页(FF)，将当前位置移到下页开头 | 012 |
| \n | 换行(LF)，将当前位置移到下一行开头 | 010 |
| \r | 回车(CR)，将当前位置移到本行开头 | 013 |
| \t | 水平制表(HT)（跳到下一个TAB位置） | 009 |
| \v | 垂直制表(VT) | 011 |
| \\ | 代表一个反斜线字符" | 092 |
| ' | 代表一个单引号（撇号）字符 | 039 |
| " | 代表一个双引号字符 | 034 |
| \? | 代表一个问号 | 063 |
| \0 | 数字0 | 000 |
| \ddd | 8进制转义字符，d范围0~7 | 3位8进制 |
| \xhh | 16进制转义字符，h范围0~9, a~f, A~F | 3位16进制 |

示例：

```
1  int main() {
2
3
4      cout << "\\\" << endl;
5      cout << "\tHello\" << endl;
6      cout << "\n\" << endl;
7
8      system("pause");
9
10     return 0;
11 }
```

字符串型

作用：用于表示一串字符

两种风格

1. **C风格字符串：** `char 变量名[] = "字符串值"`

示例：

```

1  int main() {
2
3      char str1[] = "hello world";
4      cout << str1 << endl;
5
6      system("pause");
7
8      return 0;
9  }

```

注意：C风格的字符串要用双引号括起来

1. C++风格字符串： `string` 变量名 = "字符串值"

示例：

```

1  int main() {
2
3      string str = "hello world";
4      cout << str << endl;
5
6      system("pause");
7
8      return 0;
9  }

```

注意：C++风格字符串，需要加入头文件#include<string>

布尔类型 bool

作用：布尔数据类型代表真或假的值

bool类型只有两个值：

- true --- 真（本质是1）
- false --- 假（本质是0）

bool类型占1个字节大小

示例：

```

1  int main() {
2
3      bool flag = true;
4      cout << flag << endl; // 1
5
6      flag = false;
7      cout << flag << endl; // 0
8
9      cout << "size of bool = " << sizeof(bool) << endl; //1
10
11     system("pause");
12
13     return 0;

```

数据的输入

作用：用于从键盘获取数据

关键字：cin

语法：cin >> 变量

示例：

```
1  int main(){
2
3      //整型输入
4      int a = 0;
5      cout << "请输入整型变量: " << endl;
6      cin >> a;
7      cout << a << endl;
8
9      //浮点型输入
10     double d = 0;
11     cout << "请输入浮点型变量: " << endl;
12     cin >> d;
13     cout << d << endl;
14
15     //字符型输入
16     char ch = 0;
17     cout << "请输入字符型变量: " << endl;
18     cin >> ch;
19     cout << ch << endl;
20
21     //字符串型输入
22     string str;
23     cout << "请输入字符串型变量: " << endl;
24     cin >> str;
25     cout << str << endl;
26
27     //布尔类型输入
28     bool flag = true;
29     cout << "请输入布尔型变量: " << endl;
30     cin >> flag;
31     cout << flag << endl;
32     system("pause");
33     return EXIT_SUCCESS;
34 }
```

运算符

作用：用于执行代码的运算

本章我们主要讲解以下几类运算符：

| 运算符类型 | 作用 |
|-------|---------------------|
| 算术运算符 | 用于处理四则运算 |
| 赋值运算符 | 用于将表达式的值赋给变量 |
| 比较运算符 | 用于表达式的比较，并返回一个真值或假值 |
| 逻辑运算符 | 用于根据表达式的值返回真值或假值 |

算术运算符

作用：用于处理四则运算

算术运算符包括以下符号：

| 运算符 | 术语 | 示例 | 结果 |
|-----|--------|-------------|-----------|
| + | 正号 | +3 | 3 |
| - | 负号 | -3 | -3 |
| + | 加 | 10 + 5 | 15 |
| - | 减 | 10 - 5 | 5 |
| * | 乘 | 10 * 5 | 50 |
| / | 除 | 10 / 5 | 2 |
| % | 取模(取余) | 10 % 3 | 1 |
| ++ | 前置递增 | a=2; b=++a; | a=3; b=3; |
| ++ | 后置递增 | a=2; b=a++; | a=3; b=2; |
| -- | 前置递减 | a=2; b--a; | a=1; b=1; |
| -- | 后置递减 | a=2; b=a--; | a=1; b=2; |

示例1：

```
1 //加减乘除
2 int main() {
3
4     int a1 = 10;
5     int b1 = 3;
6
7     cout << a1 + b1 << endl;
8     cout << a1 - b1 << endl;
9     cout << a1 * b1 << endl;
10    cout << a1 / b1 << endl; //两个整数相除结果依然是整数
11
12    int a2 = 10;
13    int b2 = 20;
14    cout << a2 / b2 << endl;
15 }
```

```

16     int a3 = 10;
17     int b3 = 0;
18     //cout << a3 / b3 << endl; //报错，除数不可以为0
19
20
21     //两个小数可以相除
22     double d1 = 0.5;
23     double d2 = 0.25;
24     cout << d1 / d2 << endl;
25
26     system("pause");
27
28     return 0;
29 }

```

总结：在除法运算中，除数不能为0

示例2:

```

1 //取模
2 int main() {
3
4     int a1 = 10;
5     int b1 = 3;
6
7     cout << 10 % 3 << endl;
8
9     int a2 = 10;
10    int b2 = 20;
11
12    cout << a2 % b2 << endl;
13
14    int a3 = 10;
15    int b3 = 0;
16
17    //cout << a3 % b3 << endl; //取模运算时，除数也不能为0
18
19    //两个小数不可以取模
20    double d1 = 3.14;
21    double d2 = 1.1;
22
23    //cout << d1 % d2 << endl;
24
25    system("pause");
26
27    return 0;
28 }
29

```

总结：只有整型变量可以进行取模运算

示例3:

```

1 //递增
2 int main() {
3
4     //后置递增
5     int a = 10;
6     a++; //等价于a = a + 1
7     cout << a << endl; // 11
8
9     //前置递增
10    int b = 10;
11    ++b;
12    cout << b << endl; // 11
13
14    //区别
15    //前置递增先对变量进行++, 再计算表达式
16    int a2 = 10;
17    int b2 = ++a2 * 10;
18    cout << b2 << endl;
19
20    //后置递增先计算表达式, 后对变量进行++
21    int a3 = 10;
22    int b3 = a3++ * 10;
23    cout << b3 << endl;
24
25    system("pause");
26
27    return 0;
28 }
29

```

总结：前置递增先对变量进行++，再计算表达式，后置递增相反

赋值运算符

作用：用于将表达式的值赋给变量

赋值运算符包括以下几个符号：

| 运算符 | 术语 | 示例 | 结果 |
|-----|-----|------------|-----------|
| = | 赋值 | a=2; b=3; | a=2; b=3; |
| += | 加等于 | a=0; a+=2; | a=2; |
| -= | 减等于 | a=5; a-=3; | a=2; |
| *= | 乘等于 | a=2; a*=2; | a=4; |
| /= | 除等于 | a=4; a/=2; | a=2; |
| %= | 模等于 | a=3; a%=2; | a=1; |

示例:

```
1  int main() {
2
3      //赋值运算符
4
5      // =
6      int a = 10;
7      a = 100;
8      cout << "a = " << a << endl;
9
10     // +=
11     a = 10;
12     a += 2; // a = a + 2;
13     cout << "a = " << a << endl;
14
15     // -=
16     a = 10;
17     a -= 2; // a = a - 2
18     cout << "a = " << a << endl;
19
20     // *=
21     a = 10;
22     a *= 2; // a = a * 2
23     cout << "a = " << a << endl;
24
25     // /=
26     a = 10;
27     a /= 2; // a = a / 2;
28     cout << "a = " << a << endl;
29
30     // %=
31     a = 10;
32     a %= 2; // a = a % 2;
33     cout << "a = " << a << endl;
34
35     system("pause");
36
37     return 0;
38 }
```

比较运算符

作用：用于表达式的比较，并返回一个真值或假值

比较运算符有以下符号：

| 运算符 | 术语 | 示例 | 结果 |
|-----|------|--------|----|
| == | 相等于 | 4 == 3 | 0 |
| != | 不等于 | 4 != 3 | 1 |
| < | 小于 | 4 < 3 | 0 |
| > | 大于 | 4 > 3 | 1 |
| <= | 小于等于 | 4 <= 3 | 0 |
| >= | 大于等于 | 4 >= 1 | 1 |

示例：

```
1  int main() {
2
3      int a = 10;
4      int b = 20;
5
6      cout << (a == b) << endl; // 0
7
8      cout << (a != b) << endl; // 1
9
10     cout << (a > b) << endl; // 0
11
12     cout << (a < b) << endl; // 1
13
14     cout << (a >= b) << endl; // 0
15
16     cout << (a <= b) << endl; // 1
17
18     system("pause");
19
20     return 0;
21 }
```

注意：C和C++ 语言的比较运算中，“真”用数字“1”来表示，“假”用数字“0”来表示。

逻辑运算符

作用：用于根据表达式的值返回真值或假值

逻辑运算符有以下符号：

| 运算符 | 术语 | 示例 | 结果 |
|-----|----|--------|-------------------------------|
| ! | 非 | !a | 如果a为假，则!a为真；如果a为真，则!a为假。 |
| && | 与 | a && b | 如果a和b都为真，则结果为真，否则为假。 |
| | 或 | a b | 如果a和b有一个为真，则结果为真，二者都为假时，结果为假。 |

示例1：逻辑非

```

1 //逻辑运算符 --- 非
2 int main() {
3
4     int a = 10;
5
6     cout << !a << endl; // 0
7
8     cout << !!a << endl; // 1
9
10    system("pause");
11
12    return 0;
13 }
```

总结：真变假，假变真

示例2：逻辑与

```

1 //逻辑运算符 --- 与
2 int main() {
3
4     int a = 10;
5     int b = 10;
6
7     cout << (a && b) << endl; // 1
8
9     a = 10;
10    b = 0;
11
12    cout << (a && b) << endl; // 0
13
14    a = 0;
15    b = 0;
16
17    cout << (a && b) << endl; // 0
18
19    system("pause");
20
21    return 0;
22 }
23
```

总结：逻辑与运算符总结：同真为真，其余为假

示例3：逻辑或

```
1 //逻辑运算符 --- 或
2 int main() {
3
4     int a = 10;
5     int b = 10;
6
7     cout << (a || b) << endl; // 1
8
9     a = 10;
10    b = 0;
11
12    cout << (a || b) << endl; // 1
13
14    a = 0;
15    b = 0;
16
17    cout << (a || b) << endl; // 0
18
19    system("pause");
20
21    return 0;
22 }
```

逻辑或运算符总结：同假为假，其余为真

程序流程结构

C/C++支持最基本的三种程序运行结构：顺序结构、选择结构、循环结构

- 顺序结构：程序按顺序执行，不发生跳转
- 选择结构：依据条件是否满足，有选择的执行相应功能
- 循环结构：依据条件是否满足，循环多次执行某段代码

选择结构

if语句

作用：执行满足条件的语句

if语句的三种形式

- 单行格式if语句
- 多行格式if语句
- 多条件的if语句

1. 单行格式if语句： `if(条件){ 条件满足执行的语句 }`

示例:

```
1  int main() {
2
3      //选择结构-单行if语句
4      //输入一个分数，如果分数大于600分，视为考上一本大学，并在屏幕上打印
5
6      int score = 0;
7      cout << "请输入一个分数: " << endl;
8      cin >> score;
9
10     cout << "您输入的分数为: " << score << endl;
11
12     //if语句
13     //注意事项，在if判断语句后面，不要加分号
14     if (score > 600)
15     {
16         cout << "我考上了一本大学!!! " << endl;
17     }
18
19     system("pause");
20
21     return 0;
22 }
```

注意: if条件表达式后不要加分号

2. 多行格式if语句: `if(条件){ 条件满足执行的语句 }else{ 条件不满足执行的语句 };`

示例:

```
1  int main() {
2
3      int score = 0;
4
5      cout << "请输入考试分数: " << endl;
6
7      cin >> score;
8
9      if (score > 600)
10     {
11         cout << "我考上了一本大学" << endl;
12     }
13     else
14     {
15         cout << "我未考上一本大学" << endl;
16     }
17
18     system("pause");
19
20     return 0;
21 }
```


3. 多条件的if语句: `if(条件1){ 条件1满足执行的语句 }else if(条件2){条件2满足执行的语句}... else{ 都不满足执行的语句}`

示例:

```
1  int main() {
2
3  int score = 0;
4
5  cout << "请输入考试分数: " << endl;
6
7  cin >> score;
8
9  if (score > 600)
10 {
11     cout << "我考上了一本大学" << endl;
12 }
13 else if (score > 500)
14 {
15     cout << "我考上了二本大学" << endl;
16 }
17 else if (score > 400)
18 {
19     cout << "我考上了三本大学" << endl;
20 }
21 else
22 {
23     cout << "我未考上本科" << endl;
24 }
25
26 system("pause");
27
28 return 0;
29 }
```

嵌套if语句: 在if语句中, 可以嵌套使用if语句, 达到更精确的条件判断

案例需求:

- 提示用户输入一个高考考试分数, 根据分数做如下判断
- 分数如果大于600分视为考上一本, 大于500分考上二本, 大于400考上三本, 其余视为未考上本科;
- 在一本分数中, 如果大于700分, 考入北大, 大于650分, 考入清华, 大于600考入人大。

示例:

```
1  int main() {
2
3  int score = 0;
4
5  cout << "请输入考试分数: " << endl;
6
7  cin >> score;
8
```

```

9      if (score > 600)
10     {
11         cout << "我考上了一所大学" << endl;
12         if (score > 700)
13         {
14             cout << "我考上了北大" << endl;
15         }
16         else if (score > 650)
17         {
18             cout << "我考上了清华" << endl;
19         }
20         else
21         {
22             cout << "我考上了人大" << endl;
23         }
24     }
25
26     else if (score > 500)
27     {
28         cout << "我考上了二所大学" << endl;
29     }
30     else if (score > 400)
31     {
32         cout << "我考上了三所大学" << endl;
33     }
34     else
35     {
36         cout << "我未考上本科" << endl;
37     }
38
39     system("pause");
40
41     return 0;
42 }

```

三目运算符

作用： 通过三目运算符实现简单的判断

语法： 表达式1 ? 表达式2 : 表达式3

解释：

如果表达式1的值为真，执行表达式2，并返回表达式2的结果；

如果表达式1的值为假，执行表达式3，并返回表达式3的结果。

示例：

```

1  int main() {
2
3      int a = 10;
4      int b = 20;
5      int c = 0;
6
7      c = a > b ? a : b;
8      cout << "c = " << c << endl;

```

```

9
10 //C++中三目运算符返回的是变量,可以继续赋值
11
12 (a > b ? a : b) = 100;
13
14 cout << "a = " << a << endl;
15 cout << "b = " << b << endl;
16 cout << "c = " << c << endl;
17
18 system("pause");
19
20 return 0;
21 }

```

总结：和if语句比较，三目运算符优点是短小整洁，缺点是如果用嵌套，结构不清晰

switch语句

作用：执行多条件分支语句

语法：

```

1 switch(表达式)
2
3 {
4
5     case 结果1: 执行语句;break;
6
7     case 结果2: 执行语句;break;
8
9     ...
10
11     default:执行语句;break;
12
13 }
14

```

示例：

```

1 int main() {
2
3     //请给电影评分
4     //10 ~ 9    经典
5     // 8 ~ 7    非常好
6     // 6 ~ 5    一般
7     // 5分以下 烂片
8
9     int score = 0;
10    cout << "请给电影打分" << endl;
11    cin >> score;
12
13    switch (score)
14    {
15        case 10:
16        case 9:

```

```

17     cout << "经典" << endl;
18     break;
19     case 8:
20         cout << "非常好" << endl;
21         break;
22     case 7:
23     case 6:
24         cout << "一般" << endl;
25         break;
26     default:
27         cout << "烂片" << endl;
28         break;
29 }
30
31 system("pause");
32
33 return 0;
34 }

```

注意1: switch语句中表达式类型只能是整型或者字符型

注意2: case里如果没有break, 那么程序会一直向下执行

总结: 与if语句比, 对于多条件判断时, switch的结构清晰, 执行效率高, 缺点是switch不可以判断区间

循环结构

while循环语句

作用: 满足循环条件, 执行循环语句

语法: `while(循环条件){ 循环语句 }`

解释: 只要循环条件的结果为真, 就执行循环语句

示例:

```

1  int main() {
2
3      int num = 0;
4      while (num < 10)
5      {
6          cout << "num = " << num << endl;
7          num++;
8      }
9
10     system("pause");
11
12     return 0;
13 }

```

注意：在执行循环语句时候，程序必须提供跳出循环的出口，否则出现死循环

do...while循环语句

作用： 满足循环条件，执行循环语句

语法： `do{ 循环语句 } while(循环条件);`

注意： 与while的区别在于do...while会先执行一次循环语句，再判断循环条件

示例：

```
1  int main() {  
2  
3      int num = 0;  
4  
5      do  
6      {  
7          cout << num << endl;  
8          num++;  
9  
10     } while (num < 10);  
11  
12  
13     system("pause");  
14  
15     return 0;  
16 }
```

总结：与while循环区别在于，do...while先执行一次循环语句，再判断循环条件

for循环语句

作用： 满足循环条件，执行循环语句

语法： `for(起始表达式;条件表达式;末尾循环体) { 循环语句; }`

示例：

```
1  int main() {  
2  
3      for (int i = 0; i < 10; i++)  
4      {  
5          cout << i << endl;  
6      }  
7  
8      system("pause");  
9  
10     return 0;  
11 }
```

详解：

```

int main() {
    执行一次 → 0      1      3
    for (int i = 0; i < 10; i++)
    {
        2 cout << i << endl;
    }

    执行顺序: 0123123123...

    system("pause");

    return 0;
}

```

注意：for循环中的表达式，要用分号进行分隔

总结：while, do...while, for都是开发中常用的循环语句，for循环结构比较清晰，比较常用

跳转语句

break语句

作用: 用于跳出选择结构或者循环结构

break使用的时机：

- 出现在switch条件语句中，作用是终止case并跳出switch
- 出现在循环语句中，作用是跳出当前的循环语句
- 出现在嵌套循环中，跳出最近的内层循环语句

示例1:

```

1  int main() {
2      //1、在switch 语句中使用break
3      cout << "请选择您挑战副本的难度: " << endl;
4      cout << "1、普通" << endl;
5      cout << "2、中等" << endl;
6      cout << "3、困难" << endl;
7
8      int num = 0;
9
10     cin >> num;
11
12     switch (num)
13     {
14     case 1:
15         cout << "您选择的是普通难度" << endl;
16         break;
17     case 2:
18         cout << "您选择的是中等难度" << endl;
19         break;
20     case 3:

```

```

21     cout << "您选择的是困难难度" << endl;
22     break;
23 }
24
25 system("pause");
26
27 return 0;
28 }

```

示例2:

```

1  int main() {
2      //2、在循环语句中用break
3      for (int i = 0; i < 10; i++)
4      {
5          if (i == 5)
6          {
7              break; //跳出循环语句
8          }
9          cout << i << endl;
10     }
11
12     system("pause");
13
14     return 0;
15 }

```

示例3:

```

1  int main() {
2      //在嵌套循环语句中使用break, 退出内层循环
3      for (int i = 0; i < 10; i++)
4      {
5          for (int j = 0; j < 10; j++)
6          {
7              if (j == 5)
8              {
9                  break;
10             }
11             cout << "*" << " ";
12         }
13         cout << endl;
14     }
15
16     system("pause");
17
18     return 0;
19 }

```

continue语句

作用：在循环语句中，跳过本次循环中余下尚未执行的语句，继续执行下一次循环

示例：

```
1  int main() {
2
3      for (int i = 0; i < 100; i++)
4      {
5          if (i % 2 == 0)
6          {
7              continue;
8          }
9          cout << i << endl;
10     }
11
12     system("pause");
13
14     return 0;
15 }
```

注意：continue并没有使整个循环终止，而break会跳出循环

goto语句

作用：可以无条件跳转语句

语法： goto 标记;

解释：如果标记的名称存在，执行到goto语句时，会跳转到标记的位置

示例：

```
1  int main() {
2      cout << "1" << endl;
3      goto FLAG;
4      cout << "2" << endl;
5      cout << "3" << endl;
6      cout << "4" << endl;
7      FLAG:
8      cout << "5" << endl;
9      system("pause");
10     return 0;
11 }
```

注意：在程序中不建议使用goto语句，以免造成程序流程混乱

数组

概述

所谓数组，就是一个集合，里面存放了相同类型的数据元素

特点1：数组中的每个数据元素都是相同的数据类型

特点2：数组是由连续的内存位置组成的

一维数组

一维数组定义方式

一维数组定义的三种方式：

1. 数据类型 数组名[数组长度];
2. 数据类型 数组名[数组长度] = { 值1, 值2 ...};
3. 数据类型 数组名[] = { 值1, 值2 ...};

示例

```
1  int main() {
2
3      //定义方式1
4      //数据类型 数组名[元素个数];
5      int score[10];
6
7      //利用下标赋值
8      score[0] = 100;
9      score[1] = 99;
10     score[2] = 85;
11
12     //利用下标输出
13     cout << score[0] << endl;
14     cout << score[1] << endl;
15     cout << score[2] << endl;
16
17
18     //第二种定义方式
19     //数据类型 数组名[元素个数] = {值1, 值2 , 值3 ...};
20     //如果{}内不足10个数据，剩余数据用0补全
21     int score2[10] = { 100, 90,80,70,60,50,40,30,20,10 };
22
23     //逐个输出
24     //cout << score2[0] << endl;
25     //cout << score2[1] << endl;
26
27     //一个一个输出太麻烦，因此可以利用循环进行输出
28     for (int i = 0; i < 10; i++)
29     {
30         cout << score2[i] << endl;
31     }
32
33     //定义方式3
34     //数据类型 数组名[] = {值1, 值2 , 值3 ...};
35     int score3[] = { 100,90,80,70,60,50,40,30,20,10 };
36 }
```

```

37     for (int i = 0; i < 10; i++)
38     {
39         cout << score3[i] << endl;
40     }
41
42     system("pause");
43
44     return 0;
45 }

```

总结1：数组名的命名规范与变量名命名规范一致，不要和变量重名

总结2：数组中下标是从0开始索引

一维数组数组名

一维数组名称的用途：

1. 可以统计整个数组在内存中的长度
2. 可以获取数组在内存中的首地址

示例：

```

1  int main() {
2
3      //数组名用途
4      //1、可以获取整个数组占用内存空间大小
5      int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
6
7      cout << "整个数组所占内存空间为： " << sizeof(arr) << endl;
8      cout << "每个元素所占内存空间为： " << sizeof(arr[0]) << endl;
9      cout << "数组的元素个数为： " << sizeof(arr) / sizeof(arr[0]) << endl;
10
11     //2、可以通过数组名获取到数组首地址
12     cout << "数组首地址为： " << (int)arr << endl;
13     cout << "数组中第一个元素地址为： " << (int)&arr[0] << endl;
14     cout << "数组中第二个元素地址为： " << (int)&arr[1] << endl;
15
16     //arr = 100; 错误，数组名是常量，因此不可以赋值
17
18
19     system("pause");
20
21     return 0;
22 }

```

注意：数组名是常量，不可以赋值

总结1：直接打印数组名，可以查看数组所占内存的首地址

总结2：对数组名进行sizeof，可以获取整个数组占内存空间的大小

冒泡排序

作用：最常用的排序算法，对数组内元素进行排序

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素做同样的工作，执行完毕后，找到第一个最大值。
3. 重复以上的步骤，每次比较次数-1，直到不需要比较

示例：将数组 { 4,2,8,0,5,7,1,3,9 } 进行升序排序

```
1  int main() {
2
3      int arr[9] = { 4,2,8,0,5,7,1,3,9 };
4
5      for (int i = 0; i < 9 - 1; i++)
6      {
7          for (int j = 0; j < 9 - 1 - i; j++)
8          {
9              if (arr[j] > arr[j + 1])
10             {
11                 int temp = arr[j];
12                 arr[j] = arr[j + 1];
13                 arr[j + 1] = temp;
14             }
15         }
16     }
17
18     for (int i = 0; i < 9; i++)
19     {
20         cout << arr[i] << endl;
21     }
22
23     system("pause");
24
25     return 0;
26 }
```

二维数组

二维数组就是在一维数组上，多加一个维度。

二维数组定义方式

二维数组定义的四种方式：

1. 数据类型 数组名 [行数] [列数] ;
2. 数据类型 数组名 [行数] [列数] = { {数据1, 数据2 } , {数据3, 数据4 } } ;
3. 数据类型 数组名 [行数] [列数] = { 数据1, 数据2, 数据3, 数据4};
4. 数据类型 数组名 [] [列数] = { 数据1, 数据2, 数据3, 数据4};

建议：以上4种定义方式，利用第二种更加直观，提高代码的可读性

示例：

```
1  int main() {
2
3      //方式1
4      //数组类型 数组名 [行数][列数]
5      int arr[2][3];
6      arr[0][0] = 1;
7      arr[0][1] = 2;
8      arr[0][2] = 3;
9      arr[1][0] = 4;
10     arr[1][1] = 5;
11     arr[1][2] = 6;
12
13     for (int i = 0; i < 2; i++)
14     {
15         for (int j = 0; j < 3; j++)
16         {
17             cout << arr[i][j] << " ";
18         }
19         cout << endl;
20     }
21
22     //方式2
23     //数据类型 数组名[行数][列数] = { {数据1, 数据2 } , {数据3, 数据4 } };
24     int arr2[2][3] =
25     {
26         {1,2,3},
27         {4,5,6}
28     };
29
30     //方式3
31     //数据类型 数组名[行数][列数] = { 数据1, 数据2 ,数据3, 数据4 };
32     int arr3[2][3] = { 1,2,3,4,5,6 };
33
34     //方式4
35     //数据类型 数组名[][列数] = { 数据1, 数据2 ,数据3, 数据4 };
36     int arr4[][3] = { 1,2,3,4,5,6 };
37
38     system("pause");
39
40     return 0;
41 }
```

总结：在定义二维数组时，如果初始化了数据，可以省略行数

二维数组数组名

- 查看二维数组所占内存空间
- 获取二维数组首地址

示例：

```
1  int main() {
2
3      //二维数组数组名
4      int arr[2][3] =
5      {
6          {1,2,3},
7          {4,5,6}
8      };
9
10     cout << "二维数组大小: " << sizeof(arr) << endl;
11     cout << "二维数组一行大小: " << sizeof(arr[0]) << endl;
12     cout << "二维数组元素大小: " << sizeof(arr[0][0]) << endl;
13
14     cout << "二维数组行数: " << sizeof(arr) / sizeof(arr[0]) << endl;
15     cout << "二维数组列数: " << sizeof(arr[0]) / sizeof(arr[0][0]) << endl;
16
17     //地址
18     cout << "二维数组首地址: " << arr << endl;
19     cout << "二维数组第一行地址: " << arr[0] << endl;
20     cout << "二维数组第二行地址: " << arr[1] << endl;
21
22     cout << "二维数组第一个元素地址: " << &arr[0][0] << endl;
23     cout << "二维数组第二个元素地址: " << &arr[0][1] << endl;
24
25     system("pause");
26
27     return 0;
28 }
```

总结1：二维数组名就是这个数组的首地址

总结2：对二维数组名进行sizeof时，可以获取整个二维数组占用的内存空间大小

函数

概述

作用：将一段经常使用的代码封装起来，减少重复代码

一个较大的程序，一般分为若干个程序块，每个模块实现特定的功能。

函数的定义

函数的定义一般主要有5个步骤：

- 1、返回值类型
- 2、函数名
- 3、参数表列
- 4、函数体语句
- 5、return 表达式

语法：

```
1 返回值类型 函数名 （参数列表）
2  {
3
4      函数体语句
5
6      return表达式
7
8  }
```

- 返回值类型：一个函数可以返回一个值。在函数定义中
- 函数名：给函数起个名称
- 参数列表：使用该函数时，传入的数据
- 函数体语句：花括号内的代码，函数内需要执行的语句
- return表达式：和返回值类型挂钩，函数执行完后，返回相应的数据

示例：定义一个加法函数，实现两个数相加

```
1  //函数定义
2  int add(int num1, int num2)
3  {
4      int sum = num1 + num2;
5      return sum;
6  }
```

函数的调用

功能：使用定义好的函数

语法： 函数名 (参数)

示例：

```
1  //函数定义
2  int add(int num1, int num2) //定义中的num1,num2称为形式参数，简称形参
3  {
```

```

4     int sum = num1 + num2;
5     return sum;
6 }
7
8 int main() {
9
10    int a = 10;
11    int b = 10;
12    //调用add函数
13    int sum = add(a, b); //调用时的a, b称为实际参数, 简称实参
14    cout << "sum = " << sum << endl;
15
16    a = 100;
17    b = 100;
18
19    sum = add(a, b);
20    cout << "sum = " << sum << endl;
21
22    system("pause");
23
24    return 0;
25 }

```

总结：函数定义里小括号内称为形参，函数调用时传入的参数称为实参

值传递

- 所谓值传递，就是函数调用时实参将数值传入给形参
- 值传递时，如果形参发生，并不会影响实参

示例：

```

1 void swap(int num1, int num2)
2 {
3     cout << "交换前: " << endl;
4     cout << "num1 = " << num1 << endl;
5     cout << "num2 = " << num2 << endl;
6
7     int temp = num1;
8     num1 = num2;
9     num2 = temp;
10
11    cout << "交换后: " << endl;
12    cout << "num1 = " << num1 << endl;
13    cout << "num2 = " << num2 << endl;
14
15    //return ; 当函数声明时候, 不需要返回值, 可以不写return
16 }
17
18 int main() {
19
20    int a = 10;
21    int b = 20;

```

```

22
23     swap(a, b);
24
25     cout << "mian中的 a = " << a << endl;
26     cout << "mian中的 b = " << b << endl;
27
28     system("pause");
29
30     return 0;
31 }

```

总结：值传递时，形参是修饰不了实参的

函数的常见样式

常见的函数样式有4种

1. 无参无返
2. 有参无返
3. 无参有返
4. 有参有返

示例：

```

1  //函数常见样式
2  //1、 无参无返
3  void test01()
4  {
5      //void a = 10; //无类型不可以创建变量,原因无法分配内存
6      cout << "this is test01" << endl;
7      //test01(); 函数调用
8  }
9
10 //2、 有参无返
11 void test02(int a)
12 {
13     cout << "this is test02" << endl;
14     cout << "a = " << a << endl;
15 }
16
17 //3、无参有返
18 int test03()
19 {
20     cout << "this is test03 " << endl;
21     return 10;
22 }
23
24 //4、有参有返
25 int test04(int a, int b)
26 {
27     cout << "this is test04 " << endl;
28     int sum = a + b;
29     return sum;
30 }

```


函数的声明

作用：告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。

- 函数的**声明**可以多次，但是函数的**定义**只能有一次

示例：

```
1 //声明可以多次，定义只能一次
2 //声明
3 int max(int a, int b);
4 int max(int a, int b);
5 //定义
6 int max(int a, int b)
7 {
8     return a > b ? a : b;
9 }
10
11 int main() {
12
13     int a = 100;
14     int b = 200;
15
16     cout << max(a, b) << endl;
17
18     system("pause");
19
20     return 0;
21 }
```

函数的分文件编写

作用：让代码结构更加清晰

函数分文件编写一般有4个步骤

1. 创建后缀名为.h的头文件
2. 创建后缀名为.cpp的源文件
3. 在头文件中写函数的声明
4. 在源文件中写函数的定义

示例：

```
1 //swap.h文件
2 #include<iostream>
3 using namespace std;
4
5 //实现两个数字交换的函数声明
6 void swap(int a, int b);
7
```

```

1 //swap.cpp文件
2 #include "swap.h"
3
4 void swap(int a, int b)
5 {
6     int temp = a;
7     a = b;
8     b = temp;
9
10    cout << "a = " << a << endl;
11    cout << "b = " << b << endl;
12 }

```

```

1 //main函数文件
2 #include "swap.h"
3 int main() {
4
5     int a = 100;
6     int b = 200;
7     swap(a, b);
8
9     system("pause");
10
11    return 0;
12 }
13

```

指针

指针的基本概念

指针的作用： 可以通过指针间接访问内存

- 内存编号是从0开始记录的，一般用十六进制数字表示
- 可以利用指针变量保存地址

指针变量的定义和使用

指针变量定义语法： `数据类型 * 变量名；`

示例：

```

1 int main() {
2
3     //1、指针的定义
4     int a = 10; //定义整型变量a
5
6     //指针定义语法： 数据类型 * 变量名 ；
7     int * p;

```

```

8
9      //指针变量赋值
10     p = &a; //指针指向变量a的地址
11     cout << &a << endl; //打印数据a的地址
12     cout << p << endl;   //打印指针变量p
13
14     //2、指针的使用
15     //通过*操作指针变量指向的内存
16     cout << "*p = " << *p << endl;
17
18     system("pause");
19
20     return 0;
21 }

```

指针变量和普通变量的区别

- 普通变量存放的是数据,指针变量存放的是地址
- 指针变量可以通过" * "操作符, 操作指针变量指向的内存空间, 这个过程称为解引用

总结1: 我们可以通过 & 符号 获取变量的地址

总结2: 利用指针可以记录地址

总结3: 对指针变量解引用, 可以操作指针指向的内存

指针所占内存空间

提问: 指针也是种数据类型, 那么这种数据类型占用多少内存空间?

示例:

```

1  int main() {
2
3      int a = 10;
4
5      int * p;
6      p = &a; //指针指向数据a的地址
7
8      cout << *p << endl; // * 解引用
9      cout << sizeof(p) << endl;
10     cout << sizeof(char *) << endl;
11     cout << sizeof(float *) << endl;
12     cout << sizeof(double *) << endl;
13
14     system("pause");
15
16     return 0;
17 }

```

总结: 所有指针类型在32位操作系统下是4个字节

空指针和野指针

空指针：指针变量指向内存中编号为0的空间

用途：初始化指针变量

注意：空指针指向的内存是不可以访问的

示例1：空指针

```
1  int main() {
2
3      //指针变量p指向内存地址编号为0的空间
4      int * p = NULL;
5
6      //访问空指针报错
7      //内存编号0 ~255为系统占用内存，不允许用户访问
8      cout << *p << endl;
9
10     system("pause");
11
12     return 0;
13 }
```

野指针：指针变量指向非法的内存空间

示例2：野指针

```
1  int main() {
2
3      //指针变量p指向内存地址编号为0x1100的空间
4      int * p = (int *)0x1100;
5
6      //访问野指针报错
7      cout << *p << endl;
8
9      system("pause");
10
11     return 0;
12 }
```

总结：空指针和野指针都不是我们申请的空间，因此不要访问。

const修饰指针

const修饰指针有三种情况

1. const修饰指针 --- 常量指针
2. const修饰常量 --- 指针常量
3. const即修饰指针，又修饰常量

示例：

```
1  int main() {
2
3      int a = 10;
4      int b = 10;
5
6      //const修饰的是指针，指针指向可以改，指针指向的值不可以更改
7      const int * p1 = &a;
8      p1 = &b; //正确
9      /*p1 = 100; 报错
10
11
12     //const修饰的是常量，指针指向不可以改，指针指向的值可以更改
13     int * const p2 = &a;
14     //p2 = &b; //错误
15     *p2 = 100; //正确
16
17     //const既修饰指针又修饰常量
18     const int * const p3 = &a;
19     //p3 = &b; //错误
20     /*p3 = 100; //错误
21
22     system("pause");
23
24     return 0;
25 }
```

技巧：看const右侧紧跟着的是指针还是常量,是指针就是常量指针，是常量就是指针常量

指针和数组

作用：利用指针访问数组中元素

示例：

```
1  int main() {
2
3      int arr[] = { 1,2,3,4,5,6,7,8,9,10 };
4
5      int * p = arr; //指向数组的指针
6
7      cout << "第一个元素: " << arr[0] << endl;
8      cout << "指针访问第一个元素: " << *p << endl;
9
10     for (int i = 0; i < 10; i++)
11     {
12         //利用指针遍历数组
13         cout << *p << endl;
14         p++;
15     }
16
17     system("pause");
```

```
18
19     return 0;
20 }
```

指针和函数

作用：利用指针作函数参数，可以修改实参的值

示例：

```
1  //值传递
2  void swap1(int a ,int b)
3  {
4      int temp = a;
5      a = b;
6      b = temp;
7  }
8  //地址传递
9  void swap2(int * p1, int *p2)
10 {
11     int temp = *p1;
12     *p1 = *p2;
13     *p2 = temp;
14 }
15
16 int main() {
17
18     int a = 10;
19     int b = 20;
20     swap1(a, b); // 值传递不会改变实参
21
22     swap2(&a, &b); //地址传递会改变实参
23
24     cout << "a = " << a << endl;
25
26     cout << "b = " << b << endl;
27
28     system("pause");
29
30     return 0;
31 }
```

总结：如果不想修改实参，就用值传递，如果想修改实参，就用地址传递

指针、数组、函数

案例描述：封装一个函数，利用冒泡排序，实现对整型数组的升序排序

例如数组：int arr[10] = { 4,3,6,9,1,2,10,8,7,5};

示例：

```

1 //冒泡排序函数
2 void bubblesort(int * arr, int len) //int * arr 也可以写为int arr[]
3 {
4     for (int i = 0; i < len - 1; i++)
5     {
6         for (int j = 0; j < len - 1 - i; j++)
7         {
8             if (arr[j] > arr[j + 1])
9             {
10                 int temp = arr[j];
11                 arr[j] = arr[j + 1];
12                 arr[j + 1] = temp;
13             }
14         }
15     }
16 }
17
18 //打印数组函数
19 void printArray(int arr[], int len)
20 {
21     for (int i = 0; i < len; i++)
22     {
23         cout << arr[i] << endl;
24     }
25 }
26
27 int main() {
28
29     int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };
30     int len = sizeof(arr) / sizeof(int);
31
32     bubblesort(arr, len);
33
34     printArray(arr, len);
35
36     system("pause");
37
38     return 0;
39 }

```

总结：当数组名传入到函数作为参数时，被退化为指向首元素的指针

结构体

结构体基本概念

结构体属于用户自定义的数据类型，允许用户存储不同的数据类型

结构体定义和使用

语法: `struct 结构体名 { 结构体成员列表 };`

通过结构体创建变量的方式有三种:

- `struct 结构体名 变量名`
- `struct 结构体名 变量名 = { 成员1值, 成员2值...}`
- 定义结构体时顺便创建变量

示例:

```
1  //结构体定义
2  struct student
3  {
4      //成员列表
5      string name; //姓名
6      int age;      //年龄
7      int score;    //分数
8  }stu3; //结构体变量创建方式3
9
10
11 int main() {
12
13     //结构体变量创建方式1
14     struct student stu1; //struct 关键字可以省略
15
16     stu1.name = "张三";
17     stu1.age = 18;
18     stu1.score = 100;
19
20     cout << "姓名: " << stu1.name << " 年龄: " << stu1.age << " 分数: " <<
stu1.score << endl;
21
22     //结构体变量创建方式2
23     struct student stu2 = { "李四", 19, 60 };
24
25     cout << "姓名: " << stu2.name << " 年龄: " << stu2.age << " 分数: " <<
stu2.score << endl;
26
27
28     stu3.name = "王五";
29     stu3.age = 18;
30     stu3.score = 80;
31
32
33     cout << "姓名: " << stu3.name << " 年龄: " << stu3.age << " 分数: " <<
stu3.score << endl;
34
35     system("pause");
36
37     return 0;
38 }
```


总结1：定义结构体时的关键字是struct，不可省略

总结2：创建结构体变量时，关键字struct可以省略

总结3：结构体变量利用操作符 "." 访问成员

结构体数组

作用：将自定义的结构体放入到数组中方便维护

语法： `struct 结构体名 数组名[元素个数] = { {}, {}, ... {} }`

示例：

```
1 //结构体定义
2 struct student
3 {
4     //成员列表
5     string name; //姓名
6     int age;     //年龄
7     int score;   //分数
8 }
9
10 int main() {
11
12     //结构体数组
13     struct student arr[3]=
14     {
15         {"张三",18,80 },
16         {"李四",19,60 },
17         {"王五",20,70 }
18     };
19
20     for (int i = 0; i < 3; i++)
21     {
22         cout << "姓名: " << arr[i].name << " 年龄: " << arr[i].age << " 分数: "
23         << arr[i].score << endl;
24     }
25
26     system("pause");
27
28     return 0;
29 }
```

结构体指针

作用：通过指针访问结构体中的成员

- 利用操作符 `->` 可以通过结构体指针访问结构体属性

示例：

```
1 //结构体定义
2 struct student
3 {
4     //成员列表
```

```

5     string name; //姓名
6     int age;     //年龄
7     int score;   //分数
8 };
9
10
11 int main() {
12
13     struct student stu = { "张三",18,100, };
14
15     struct student * p = &stu;
16
17     p->score = 80; //指针通过 -> 操作符可以访问成员
18
19     cout << "姓名: " << p->name << " 年龄: " << p->age << " 分数: " << p->score
20     << endl;
21
22     system("pause");
23
24     return 0;
25 }

```

总结：结构体指针可以通过 -> 操作符 来访问结构体中的成员

结构体嵌套结构体

作用： 结构体中的成员可以是另一个结构体

例如： 每个老师辅导一个学员，一个老师的结构体中，记录一个学生的结构体

示例：

```

1 //学生结构体定义
2 struct student
3 {
4     //成员列表
5     string name; //姓名
6     int age;     //年龄
7     int score;   //分数
8 };
9
10 //教师结构体定义
11 struct teacher
12 {
13     //成员列表
14     int id; //职工编号
15     string name; //教师姓名
16     int age; //教师年龄
17     struct student stu; //子结构体 学生
18 };
19
20
21 int main() {
22

```

```

23     struct teacher t1;
24     t1.id = 10000;
25     t1.name = "老王";
26     t1.age = 40;
27
28     t1.stu.name = "张三";
29     t1.stu.age = 18;
30     t1.stu.score = 100;
31
32     cout << "教师 职工编号: " << t1.id << " 姓名: " << t1.name << " 年龄: "
    << t1.age << endl;
33
34     cout << "辅导学员 姓名: " << t1.stu.name << " 年龄: " << t1.stu.age << " 考
    试分数: " << t1.stu.score << endl;
35
36     system("pause");
37
38     return 0;
39 }

```

总结：在结构体中可以定义另一个结构体作为成员，用来解决实际问题

结构体做函数参数

作用：将结构体作为参数向函数中传递

传递方式有两种：

- 值传递
- 地址传递

示例：

```

1  //学生结构体定义
2  struct student
3  {
4      //成员列表
5      string name; //姓名
6      int age; //年龄
7      int score; //分数
8  };
9
10 //值传递
11 void printStudent(student stu )
12 {
13     stu.age = 28;
14     cout << "子函数中 姓名: " << stu.name << " 年龄: " << stu.age << " 分数: "
    << stu.score << endl;
15 }
16
17 //地址传递
18 void printStudent2(student *stu)
19 {
20     stu->age = 28;

```

```

21     cout << "子函数中 姓名: " << stu->name << " 年龄: " << stu->age << " 分
    数: " << stu->score << endl;
22 }
23
24 int main() {
25
26     student stu = { "张三", 18, 100 };
27     //值传递
28     printStudent(stu);
29     cout << "主函数中 姓名: " << stu.name << " 年龄: " << stu.age << " 分数: "
    << stu.score << endl;
30
31     cout << endl;
32
33     //地址传递
34     printStudent2(&stu);
35     cout << "主函数中 姓名: " << stu.name << " 年龄: " << stu.age << " 分数: "
    << stu.score << endl;
36
37     system("pause");
38
39     return 0;
40 }

```

总结: 如果不想修改主函数中的数据, 用值传递, 反之用地址传递

结构体中 const 使用场景

作用: 用const来防止误操作

示例:

```

1 //学生结构体定义
2 struct student
3 {
4     //成员列表
5     string name; //姓名
6     int age;     //年龄
7     int score;   //分数
8 };
9
10 //const使用场景
11 void printStudent(const student *stu) //加const防止函数体中的误操作
12 {
13     //stu->age = 100; //操作失败, 因为加了const修饰
14     cout << "姓名: " << stu->name << " 年龄: " << stu->age << " 分数: " << stu-
    >score << endl;
15
16 }
17
18 int main() {
19
20     student stu = { "张三", 18, 100 };
21
22     printStudent(&stu);

```

```
23  
24     system("pause");  
25  
26     return 0;  
27 }
```