1、数据类型

1.1 C语言的本质
1.2 内存分配的最小单位
1.3 数据类型的作用
1.4 数据类型的分类
1.5 整数类型
    1.5.1 short类型（短整型）
    1.5.2 int类型(整型)
    1.5.3 long类型(长整型)
    1.5.4 long long类型(长长整型)
1.6 char(字符类型)
1.7 float/double(单精度浮点型/双精度浮点型)
1.8 void(空类型)

2、源码，反码，补码

3、常量

3.1 概念
3.2 常量的分类
3.3 验证整型常量
3.4 验证浮点型常量
3.5 验证字符类型的常量
3.6 验证字符串的常量
3.6 标识常量(宏定义)

1、数据类型

1.1 C语言的本质

C语言的本质就是对内存的操作。

内存的特点：内存中存储的是正在运行的程序或者数据，

内存中的数据掉电丢失，内存的大小一般为：2G,4G,8G,16G,32G.

硬盘的特点：没有运行的程序或者数据放到硬盘中，

硬盘中的数据掉电不丢失，硬盘的大小一般为：512G,1T,2T,5T

1.2 内存分配的最小单位

内存分配的最小单位为字节，1个字节占用8个bits的空间，最低位为第0位。
8bits的空间也可以这样表示[7:0]bits

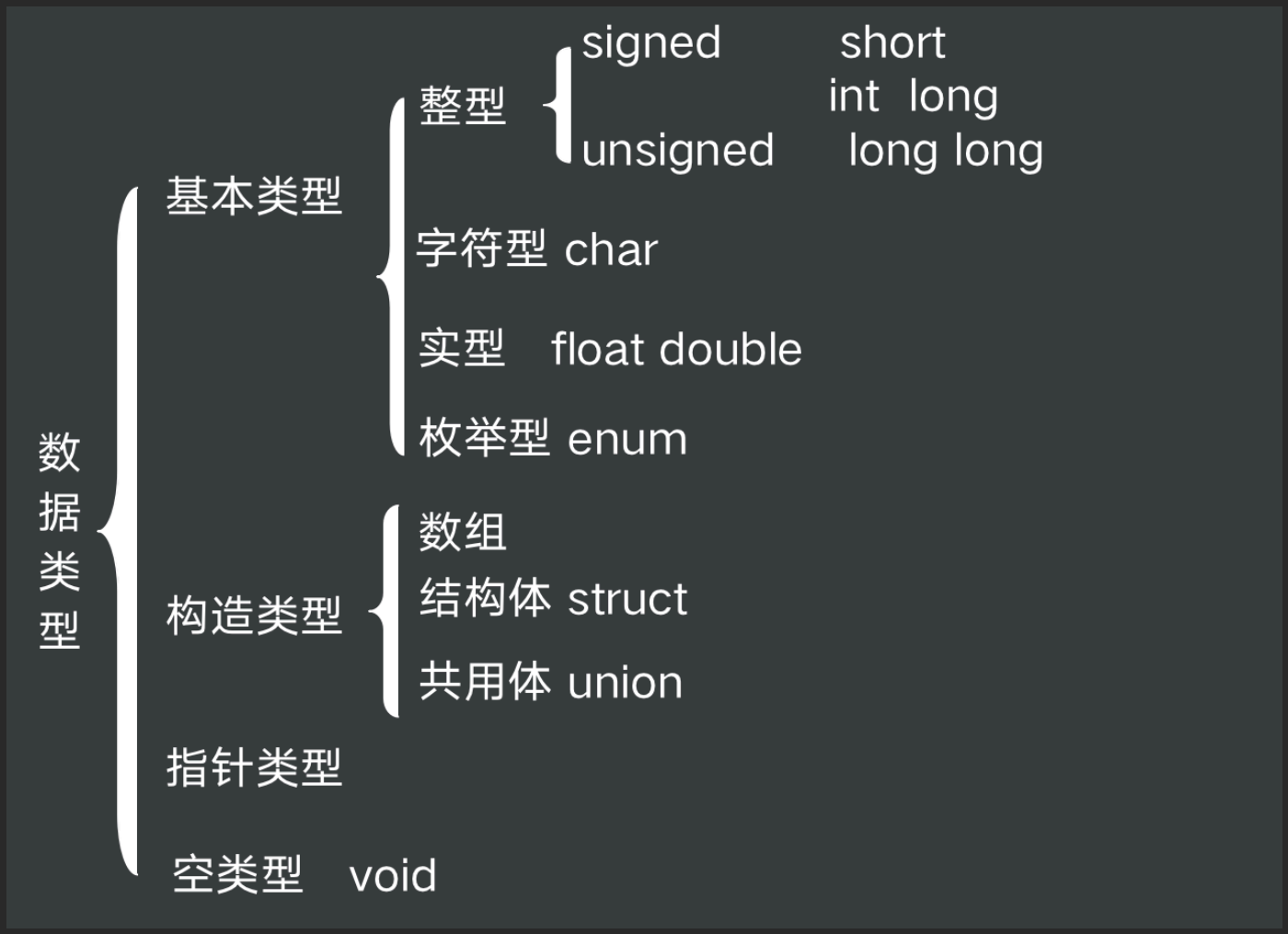
1.3 数据类型的作用

可以使用不同的数据类型定义不同类型的变量，从内存中申请不同大小的内存空间。

通过不同的数据类型可以描述不同的事物，比如：

身高：使用short
体重：使用short
性别：使用char
名字：使用字符数组

1.4 数据类型的分类



1.5 整数类型

```
1     整数类型：short / int / long / long long
2     有符号的整型：signed short / signed int / signed long / signed long long
3           定义有符号的整型变量时，可以省略signed.
4
5     无符号的整型：unsigned short / unsigned int /
6                   unsigned long / unsigned long long
7           定义无符号的整形变量时，不可以省略unsigned.
8
9     数据在内存中存储时，通过最高位的值区分有符号数为正数或者负数。
10    有符号数的最高位也称为符号位，最高位为0表示正数，最高位为1表示负数。
```

1.5.1 short类型（短整型）

```
1     短整型在内存中占用2字节的空间，共计16bits，  [15:0]bits.
2     有符号数(short/signed short)：-32768 ~ 32767
3     无符号数(unsigned short)：0 ~ 65535
```

1.5.2 int类型(整型)

```
    整型在内存中占用4字节的空间，共计32bits，[31:0]bits.
    有符号数(int/signed int)：-2^31 ~ 2^31-1
    无符号数(unsigned int)：0 ~ 2^32-1
```

1.5.3 long类型(长整型)

```
32位操作系统：
    长整型在内存中占用4字节的空间，共计32bits，  [31:0]bits.
    有符号数(long/signed long)：-2^31 ~ 2^31-1
    无符号数(unsigned long)：0 ~ 2^32-1

64位操作系统：
    长整型在内存中占用8字节的空间，共计64bits，  [63:0]bits.
    有符号数(long/signed long)：-2^63 ~ 2^63-1
    无符号数(unsigned long)：0 ~ 2^64-1
```

1.5.4 long long类型(长长整型)

```
32位操作系统:
    长长整型在内存中占用8字节的空间，共计64bits，[63:0]bits.
    有符号数(long/signed long)：-2^63 ~ 2^63-1
    无符号数(unsigned long)：0 ~ 2^64-1

64位操作系统:
    长长整型在内存中占用8字节的空间，共计64bits，[63:0]bits.
    有符号数(long/signed long)：-2^63 ~ 2^63-1
    无符号数(unsigned long)：0 ~ 2^64-1
```

1.6 char(字符类型)

```
    字符类型在内存中占用1字节的空间，共计8bits，[7:0]bits.
    有符号字符类型(char/signed char) : -128 ~ 127
    无符号字符类型(unsigned char) : 0 ~ 255

    也可以使用ascii码表中的字符对字符类型的变量赋值。
```

1.7 float/double(单精度浮点型/双精度浮点型)

```
1       float : 在内存中分配4字节的空间
2       double :  在内存中分配8字节的空间
3
4       float类型和double类型在内存中采用科学计数法进行数据的存储。
5       比如:  3.14e3  ==> 3.14 * 10^3
6
7       float类型的数据的存储:
8            符号位(S)占1位, 第[31]位,
9            指数位(E)占8位, 第[30:23]位,
10           有效位(M)占23位, 第[22:0]位。
11
12      double类型的数据的存储:
13           符号位(S)占1位, 第[63]位,
14           指数位(E)占11位, 第[62:52]位,
15           有效位(M)占52位, 第[51:0]
```

1.8 void(空类型)

```
    void空类型占用的内存空间为0，
    但是在linux系统中使用sizeof(void)进行测试时，返回的结果为1.
    在linux系统中1只是一个占位作用，起始是没有分配内存空间。

    void关键字经常和函数，指针配合使用，后续进行讲解。
```

测试64位系统值数据类型的大小：

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    // sizeof() 是一个运算符，专门用来测试数据类型的大小
    // 在64操作系统中，sizeof返回值是一个long类型，
    // long类型的占位符为%ld

    printf("char size = %ld\n", sizeof(char));

    printf("unsigned short size = %ld\n", sizeof(unsigned short));

    // unsigned short <=等价于=> unsigned short int （int一般省略不写）
    printf("unsigned short int size = %ld\n", sizeof(unsigned short int));

```

```c
    printf("int size = %ld\n", sizeof(int));

    printf("long size = %ld\n", sizeof(long));
    // long <=等价于=> long int (int可以省略不写)
    printf("long int size = %ld\n", sizeof(long int));

    printf("long long  size = %ld\n", sizeof(long long));
    // long long<=等价于=> long long int (int可以省略不写)
    printf("long long int size = %ld\n", sizeof(long long int));


    // 浮点类型没有无符号的 unsigned float (错误)
    printf("float size = %ld\n", sizeof(float));
    printf("double size = %ld\n", sizeof(double));

    printf("void size = %d\n", sizeof(void));
    // void a = 10;  // 假设void类型分配1字节空间，可以使用void定义变量，
                  // 并进行赋值操作，但是实际并不可以这样写，编译器报错
```

```
33
34      return 0;
35  }
36
37
38  gcc 01typesize.c     ----> 默认生成64位的
    可执行程序
39
```

测试32位系统值数据类型的大小：

```
 1  #include <stdio.h>
 2  int main(int argc, const char *argv[])
 3  {
 4      // sizeof() 是一个运算符，专门用来测试数
    据类型的大小
 5      // 在32操作系统中，sizeof返回值是一个
    int类型，
 6      // int类型的占位符为%d
 7
 8      printf("char size = %d\n",
    sizeof(char));
 9
10      printf("unsigned short size =
    %d\n", sizeof(unsigned short));
11
12      // unsigned short <=等价于=>
    unsigned short int （int一般省略不写）
13      printf("unsigned short int size =
    %d\n", sizeof(unsigned short int));
14
```

```
15      printf("int size = %d\n", sizeof(int));
16
17      printf("long size = %d\n", sizeof(long));
18      // long <=等价于=> long int (int可以省略不写)
19      printf("long int size = %d\n", sizeof(long int));
20
21      printf("long long  size = %d\n", sizeof(long long));
22      // long long<=等价于=> long long int (int可以省略不写)
23      printf("long long int size = %d\n", sizeof(long long int));
24
25
26      // 浮点类型没有无符号的 unsigned float (错误)
27      printf("float size = %d\n", sizeof(float));
28      printf("double size = %d\n", sizeof(double));
29
30      printf("void size = %d\n", sizeof(void));
31      // void a = 10;  // 假设void类型分配1字节空间，可以使用void定义变量，
32                  // 并进行赋值操作，但是实际并不可以这样写，编译器报错
```

```
33
34      return 0;
35  }
36
37
38  gcc -m32 01typesize.c    ----> 指定生成32位的可执行程序
```

2、源码，反码，补码

```
1   数据在内存中存储的都是补码，数据的补码需要使用源码，反码进行转换。
2
3   无符号数的源码，反码，补码都是一样。
4       比如：unsigned char ch = 0x78;
5       0x78源码：  0b0111 1000
6       0x78反码：  0b0111 1000
7       0x78补码：  0b0111 1000   ----> 在内存中存放的是数据的补码。
8
9   有符号数(正数)：源码，反码，补码都是一样。
10      比如：char ch = 0x78;
11      0x78源码：  0b0111 1000
12      0x78反码：  0b0111 1000
13      0x78补码：  0b0111 1000   ----> 在内存中存放的是数据的补码。
14
15  有符号数(负数)：源码，反码，补码不一样。
16      比如：char ch = -7;
17      -7源码：符号位为1，有效位为负数的绝对值.
```

```
                0b1000 0111
    -7返码：符号位不变，有效位按位取反，每个bit位进行取反操作(0变1，1变0)
                0b1111 1000
    -7补码：在反码的基础知识加1，符号位不变。
                0b1111 1001  ----> 在内存中存放的是数据的补码。

练习题：
    short类型：  short s = -0x567;
        源码：1000 0101 0110 0111
        反码：1111 1010 1001 1000
        补码：1111 1010 1001 1001

    int类型：   int i = -10;
        源码：1000 0000 0000 0000 0000 0000 0000 1010
        反码：1111 1111 1111 1111 1111 1111 1111 0101
        补码：1111 1111 1111 1111 1111 1111 1111 0110
```

3、常量

3.1 概念

```
在程序的运行过程中，常量的值不可以被改变。
```

3.2 常量的分类

整型常量表示	举例	输出对应的格式化字符	常量分配内存空间的大小
十进制数	1234	%d(有符号数) %u(无符号数)	4字节
八进制数	01234	%o %#o(自动补充前导符)	4字节
二进制数	0b0101	无	4字节
十六进制数	0x1234	%x %#x(自动补充前导符)	4字节

浮点型常量(实型)	比如	输出对应的格式化字符	浮点类型的常量默认分配内存空间的大小
一般的写法	3.1415	%lf %f	8字节
科学计数法([+/-]m.ne[+/-]T)	+3.14e+2 （3.14 * 10^2） -3.14e-2 （-3.14 * 10^-2）	%le %e	8字节

```
1       字符常量:字符类型的常量一般使用单引号（英文的单引号）括起来，
2       字符‘A’在内存中采用ascii码值进行存储，占用1字节的空间。
3       但是使用sizeof('A')计数大小时，结果为4字节，原因sizeof计算的字符'A'
4       的ASCII码值的的大小，而ASCII码值是一个整型常量，因此sizeof('A')
5       输出的结果就是4。要想计算字符类型的常量的大小只能通过变量进行测试。
6       char ch = 'A';
7       sizeof(ch);
8
9       比如：'A'   '\n'   'a'   '9'   .....
```

```
    字符串常量：由多个字符组成的字符串，字符串常量需要使用双引号括起来，
            字符串的结尾有一个隐藏的尾'\0',表示字符串的结束，因此判断字符串的
            是否到结尾通过判断'\0'字符。

    比如：
    "a"   "nihao"   "hello world" ....
```

```
    标识常量(宏定义) ---> #define
    格式：
        #define   宏定义的名字    字符串

    举例：
        #define   PI   3.14
        #define   NUM  100
        #define   CH   'A'
        #define   STRING    "hello world"
    宏定义在出处理阶段进行展开，就是将宏定义的字符串将宏定义的名字进行替换。
```

3.3 验证整型常量

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    printf("常量值 = %d, 常量占用内存大小 = %ld\n", 1234, sizeof(12));
```

```
    //  1234 ----> 整型常量
    //  sizeof(12) ---> 计算常量的大小，默认分配4字节的空间
    printf("常量值 = %#o, 常量占用内存大小 = %ld\n", 01234, sizeof(012));
    printf("常量值 = %#x, 常量占用内存大小 = %ld\n", 0x1234, sizeof(0x12));
    printf("常量值 = %#x, 常量占用内存大小 = %ld\n", 0b1010, sizeof(0b1010));


    return 0;
}

```

3.4 验证浮点型常量

```
#include <stdio.h>

int main(int argc, const char *argv[])
{
    printf("浮点数值 = %f, 浮点型常量默认占用内存空间的大小 = %ld\n",
                3.1415, sizeof(3.1415));
    // 浮点型的常量默认是double类型，因此分配8字节的内存空间存储浮点型的数据

    printf("浮点数值 = %e, 浮点型常量默认占用内存空间的大小 = %ld\n",
                3.1415, sizeof(3.1415));

```

```
11
12      printf("浮点数值 = %le, 浮点型常量默认占用内存空间的大小 = %ld\n",
13                  3.1415, sizeof(3.1415));
14
15
16      return 0;
17  }
18
```

3.5 验证字符类型的常量

```
 1  #include <stdio.h>
 2  int main(int argc, const char *argv[])
 3  {
 4      printf("字符常量值 = %d\n",'A');
 5      // 'A' : 字符常量, 字符串常量占用1字节的空间
 6
 7      printf("sizeof('A') = %ld\n", sizeof('A'));
 8      // sizeof('A') : 计算的是字符'A'的ASCII码值的大小,
 9      // 而字符'A'的ASCII码值是一个整型常量, 等价于sizeof(65).
10
11
12      // 要想测量字符类型常量的大小只能通过变量赋值的方式
13      char ch = 'A';
```

```
14      printf("字符类型的数据占用内存的大小 = %ld\n",sizeof(ch));
15      return 0;
16  }
17
18
```

3.6 验证字符串的常量

```
1  #include <stdio.h>
2  int main(int argc, const char *argv[])
3  {
4
5      printf("字符串常量值 = %s\n", "A");
6      printf("字符串常量值 = %s\n", "hello world");
7
8      printf("字符串常量值 = %s\n", "hello\0world");
9      // printf在打印字符串时，遇到尾'\0'就结束输出
10     // 输出结果为hello
11
12     printf("字符串常量占用内存空间的大小 = %ld\n", sizeof("hello"));
13     // 计算字符串的长度时，一定要把隐藏的尾'\0'也计算到字符串的长度中
14
15     return 0;
16 }
```

```
17
```

3.6 标识常量(宏定义)

```c
#include <stdio.h>
// 宏定义在main函数的外边
#define PI  3.14
#define NUM 100
#define CH  'A'
#define STRING  "hello world"

int main(int argc, const char *argv[])
{

    printf("PI = %lf\n", PI);
    printf("NUM = %d\n", NUM);
    printf("CH = %c\n", CH);
    printf("STRING = %s\n", STRING);

    return 0;
}

```

gcc -E 06define.c -o 06define.i

```
1
2 801 int main(int argc, const char
  *argv[])
3 802 {
4 803
5 804   printf("PI = %lf\n", 3.14);
6 805   printf("NUM = %d\n", 100);
7 806   printf("CH = %c\n", 'A');
8 807   printf("STRING = %s\n", "hello
  world");
9 808
10 809   return 0;
11 810 }
12
```

3.7 宏定义扩展的用法

3.7.1 宏定义就是在预处理阶段进行简单的替换，

```
1       如果宏定义中有算数表达式时，尽量要多加括
  号。
```

```
1 #include <stdio.h>
2 #define  SUM     100 + 200
3 #define  SUB     (200 - 100)
4 #define  D(r)    (r + r)
5 #define  MUL(a,b)  (a * b)
```

```c

int main(int argc, const char *argv[])
{
    printf("%d\n", SUM * 10); // 2100
    // 替换的结果printf("%d\n", 100 + 200 * 10)

    printf("%d\n", SUB * 10);  // 1000
    // 替换的结果: printf("%d\n", (200 - 100) * 10);

    // 宏定义可以有参数，调用用定义时传递一个值，最终会替换
    // 宏定义中对应的参数的位置
    printf("圆的周长 = %f\n", 3.14 * D(10));
    // 替换结果 :  printf("圆的周长 = %f\n", 3.14 * (10 + 10))

    printf("两个数的积 = %d\n", MUL(100,200));
    // 替换结果:  printf("两个数的积 = %d\n", (100 * 200));

    return 0;
}

```

```
练习题：定义两个宏实现求三角形的面积。
sqrt(S * (S - a) * (S - b) * (S -c));

```

```
a,b,c : 三角形3个边的长度
S = (a+b+c)/2

#include <math.h>

double sqrt(double x);
    功能: 对参数x进行开平方
    参数:
        x:对x进行开方运算
    返回:
        结果

编译程序: gcc  ***.c  -lm
```

```c
#include <stdio.h>
#include <math.h>
#define   S(a,b,c)  ((a + b + c) / 2)
#define   AREA(s,a,b,c)  sqrt(s*(s-a)*(s-b)*(s-c))

int main(int argc, const char *argv[])
{
    double S1 = S(6.0,6.0,6.0); // 替换结果 : double S1 = ((6.0,6.0,6.0 / 2)

    printf("%lf\n", AREA(S1,6.0,6.0,6.0));  // S1通过定义变量实现
    // 替换结果 sqrt(S1*(S1-a)*(S1-b)*(S1-c));
    return 0;
```

```
}

```

```
#include <stdio.h>
#include <math.h>
#define   S(a,b,c)  ((a + b + c) / 2)
#define   AREA(s,a,b,c)  sqrt(s*(s-a)*(s-b)*(s-c))

int main(int argc, const char *argv[])
{
    printf("%lf\n", AREA(S(6.0,6.0,6.0),6.0,6.0,6.0));
    //printf("%lf\n", sqrt(S(6.0,6.0,6.0)*(S(6.0,6.0,6.0)-6.0)*(S(6.0,6.0,6.0)-6.0)*(S(6,6,6)-6.0)));
    //printf("%lf\n", sqrt(((6.0 + 6.0 + 6.0) / 2)*(((6.0 + 6.0 + 6.0) / 2)-6.0)*(((6.0 + 6.0 + 6.0) / 2) - 6.0)*(((6.0 + 6.0 + 6.0) / 2)-6.0)));
    //
    return 0;
}

```

3.7.2 宏定义实现的代码注释

```
// ----> 单行注释
/* 注释内容 */  ----> 多行注释
#if 0/1
#else
#endif     ----> 多行注释

目前最主要使用的代码的注释的手段。
```

```
宏定义实现注释的方式:

// 定义宏定义
#define  宏定义名

#ifdef  宏定义名
     // 如果"宏定义名" 被定义了，则此段代码有效，否则无效
#else
     // 如果"宏定义名" 没有被被定义了，则此段代码有效，否则无效
#endif

#ifndef  宏定义名
     // 如果"宏定义名" 没有被被定义了，则此段代码有效，否则无效
#else
     // 如果"宏定义名" 被定义了，则此段代码有效，否则无效
#endif
```

在底层代码中使用此种方式非常常见。

```c
#include <stdio.h>
#define  DEBUG
int main(int argc, const char *argv[])
{
#ifdef   DEBUG
    printf("hello\n");
#else
    printf("world\n");
#endif


#ifndef   DEBUG
    printf("hello\n");
#else
    printf("world\n");
#endif
    return 0;
}

```

```c

#if defined(宏定义名)
    // 如果"宏定义名" 被定义了，则此段代码有效，否则无效
#else
    // 如果"宏定义名" 没有被定义了，则此段代码有效，否则无效
#endif
```

```
 7
 8 以上用法可以进行逻辑运算：
 9       &&(逻辑与：都为真结果为真)
10       ||(逻辑或：只要有一个为真结果为值)
11       ！(逻辑非：真变假，假变真)
12
13 #if !defined(宏定义名)
14       // 如果"宏定义名" 没有被定义了，则此段代码有效，否则无效
15 #else
16       // 如果"宏定义名" 被定义了，则此段代码有效，否则无效
17 #endif
18
19 #if defined(宏定义名1) && defined(宏定义名2)
20       // 如果"宏定义名1"和 "宏定义名2" 同时都被定义了，
21       // 则此段代码有效，否则无效
22 #else
23       // 只要有一个"宏定义名1"和 "宏定义名2" 没有被定义，
24       // 则此段代码有效，否则无效
25 #endif
26
27 #if defined(宏定义名1) || defined(宏定义名2)
28       // 只要有一个"宏定义名1"和 "宏定义名2" 被定义，
29       // 则此段代码有效，否则无效
30 #else
```

```
    // 如果"宏定义名1"和 "宏定义名2" 都没有都被定义，
    // 则此段代码有效，否则无效
#endif

在底层代码中使用此种方式非常常见。
```

```c
#include <stdio.h>
#define  DEBUG
int main(int argc, const char *argv[])
{
#if defined(DEBUG)
    printf("hello\n");
#else
    printf("world\n");
#endif


#if !defined(DEBUG)
    printf("hello\n");
#else
    printf("world\n");
#endif
    return 0;
}

```

3.7.3 宏定义跟#和##结合使用

```
1 # ：  将宏定义的参数转换为字符串
```

```
 1 #include <stdio.h>
 2 #define NAME  "zhangsan"
 3 #define NAME1(n)    n
 4 #define NAME2(n)    #n
 5 // #n 将n参数对应的字符串两边加上双引号
 6 //
 7 int main(int argc, const char *argv[])
 8 {
 9     printf("NAME = %s\n", NAME);
10     // 替换结果 printf("NAME = %s\n", "zhangsan");
11
12     printf("NAME1 = %s\n", NAME1("lisi"));
13     // 替换结果 printf("NAME1 = %s\n", "lisi");
14
15     // printf("NAME1 = %s\n", NAME1(lisi));  // 编译报错
16     // 替换结果 printf("NAME1 = %s\n", lisi);
17
18     printf("NAME2 = %s\n", NAME2(wangwu));
```

```
19      // 替换结果 printf("NAME2 = %s\n",
    "wangwu");
20
21      return 0;
22  }
23
```

```
1  ##  :  实现字符串的拼接
```

```
 1  #include <stdio.h>
 2  // 实现pri和ntf字符串的拼接
 3  #define DEBUG()  pri##ntf("test
    code\n")
 4  // 将参数a,b对应的字符串进行拼接
 5  #define SHOW(a,b)  a##b("test code\n")
 6
 7  int main(int argc, const char *argv[])
 8  {
 9      DEBUG();
10      // 替换结果  printf("test code\n");
11
12      SHOW(print, f);
13      // 替换结果  printf("test code\n");
14      return 0;
15  }
16
```

3.8 宏定义的总结

```
1> 宏定义的名字一般大写
2> 每个宏定义都需要单独写一行，如果换行进行书写需要加续行符"\"
3> 宏定义的结尾不能有分号
4> 在实际开发中一些常量尽量使用宏定义，只要修改宏定义，
        所有使用此宏定义的位置都将被修改。
```

4、变量

4.1 概念

```
在程序的运行过程中，值允许发生改变的量，称为变量
```

4.2 定义变量的格式

```
存储类型    数据类型   变量名   = 初始值;

存储类型有6个: auto  static  extern const volatile register(C高级讲解)
数据类型:
    整型(signed / unsigned)(char / short / int / long /long long)
    浮点型(float / double)
变量名: 标识符，遵循标识符的命名法则。
```

4.3 定义变量的目的

```
1       C语言的本质就是操作内存，定义变量的目的就是为了分配内存空间，
2       通过变量来操作对应的内存空间。
3       定义不同类型的变量可以在内存中分配不同大小的内存空间，通过变量
4       最终操作空间的大小跟变量的类型有关。
5
6       定义变量时，思考变量存储什么类型的数据，数据的范围。
7       性别：  定义字符类型的变量；
8       身高：  定义unsigned short类型的变量；
9       全球人口数量变化的变量：  定义long long类型的变量
```

4.4 变量的定义以及初始化

```
1  #include <stdio.h>
2  // 函数外部定义的变量为全局变量
3  // 全局变量的生命周期为整个程序
4  char sex = 'M';
5
6  // 全局变量如果没有进行初始化，默认初始化为0
7  // 修改全局变量的值，或者定义之后再给全局变量复制必须在函数中完成
8  float pi;
9
10
11 int main(int argc, const char *argv[])
```

```c
{
    // 函数内存定义的变量为局部变量
    // 局部变量的生命周期为函数内

    // 1. 定义变量的,同时进行初始化
    unsigned short heigh = 180;
    printf("heigh = %dcm\n", heigh);

    // 2. 先定义变量, 后进行初始化
    unsigned int weight;
    // 定义的局部变量如果没有进行初始化, 变量的值为随机值
    printf("weight = %dkg\n", weight);
    weight = 90;          // 初始化, 修改变量的值
    printf("weight = %dkg\n", weight);

    // 3. 相同类型的变量直接可以直接进行复制操作
    char ch1 = 'A';
    char ch2 = ch1;  // 使用相同类型的变量进行初始化
    char ch3;
    ch3 = ch1;       // 相同类型的变量直接进行赋值

    printf("ch1 = %c, ch2 = %c, ch3 = %c\n",ch1,ch2,ch3);

    // 4. 定义变量时, 可以同时定义多个相同类型的变量
```

```
36      long int  l1 = 10000, l2 = 20000, l3;
37      l3 = 30000;
38      printf("l1 = %ld, l2 = %ld, l3 = %ld\n",l1,l2,l3);
39
40      // 5. 使用全局变量
41      printf("sex = %c\n", sex);
42
43      // 6. 未初始化的全局变量默认初始化为0
44      printf("未赋值PI = %f\n", pi);
45      pi = 3.14;
46      printf("赋值PI = %f\n", pi); // 使用的是全局变量的pi
47      printf("全局变量pi地址 = %#x\n", &pi);
48
49      // 7. 局部优先原则：局部变量和全局变量定义了相同名字的变量，优先使用局部变量
50      float pi = 3.1415;
51      printf("赋值PI = %f\n", pi); // 使用的是局部变量的pi
52      printf("局部变量pi地址 = %#x\n", &pi);
53
54      return 0;
55  }
56
```

6、周四的授课内容

```
1. 数据类型转换

2. 算数运算符
     单 ： +(正数) -(负数) ++ -- ~(按位取反) !(非)
     算 ：  + - * / %
     移 : << >>
     关 : > < >= <= != ==
     与 : &(按位与)
     异 : ^(按位异或)
     或 : |(按位或)
     逻 : && ||
     条 : ?:
     赋 : = += -= *= /= %= &= |= ^= <<= >>=
     逗号 : ,

3. 控制语句
```