1、类型转换

作业：

明天授课内容：

```
1        在执行算数运算时，计算机只能完成相同类型之间的算数运算，
2  如果两个不同的类型的数据进行算数运算就需要转换为相同类型的数据，
3  进行算数运算。比如short类型的数据和int类型的数据进行算数运算时，
```

需要先将short类型转换为int类型在进行算数运算。

类型转换分为：隐式类型转换和显式类型转换(强制类型转换/强转)。

类型转换需要注意：
1> 小类型向大类型转换时，一般是安全没有问题的。
2> 大类型向小类型转换时，一般出现数据的截断和丢失，不安全，慎用。
3> 类型转换完成之后，并不会改变变量本身的类型，
而变量本身的类型有定义变量时的类型决定
4> 在实际开发过程中，涉及到类型转换时，决定权要由程序员决定，
不要交给编译器决定。

1.1 隐式类型转换

有编译器完成类型的转换，无需程序源的介入，这类转换称为隐式类型转换。

发生隐式类型转换的情况：
1> 在算数表达式或者逻辑表达式中，操作数类型不同时会发生类型转换
2> 在赋值过程中右侧的表达式的类型和左侧变量的类型不匹配时，

```
6           也会发生类型转换，转换为左侧变量类型之后进行赋值。
7       3> 函数调用时，实参的类型和形参的类型不一致时也会发生类型转换
8       4> return返回的表达式的类型和函数的返回类型不一致时，也会发声类型转换。
9
10      C99标准的隐式类型转换：
11      如果表达式中有浮点类型的数据，转换为浮点类型数据进行运算。
12      浮点类型的优先级从低到高如下：
13          float --> double --> long double
14          举例：
15          int a; float b;
16          a+b;    // 将a隐式转换为float类型之后，在进行算数运算。
17
18          float c; double d;
19          c + d; // 将c隐式转换为double类型之后，在进行算数运算。
20      如果表达式中都不是浮点类型的，按照以下标准进行提升：
21      整数类型的优先级从低到高如下：
22      1> char / unsigned char
23      2> short / unsigned short
24      3> int / unsigned int
25      4> long / unsigned long
26      5> long long / unsigned long long
27
```

```
28      首先对两个操作数进行整数提升，如果两个操作数的类型相同，过程结束；
29      否则依照以下规则进行提升，一旦遇到可以适配的规则将不在考虑其他的规则。
30      第1种情况：如果两个操作数都为有符号数或者无符号数，将等级较低的操作数类型，
31              转换为等级较高的操作数类型。
32      举例：
33              unsigned int a; unsigned long int b;
34              a + b; // 变量a被隐式类型转换为unsigned long int类型
35      第2种情况：如果无符号操作数类型的优先级高于有符号数的类型优先级，
36              将有符号操作数的类型转换为无符号数的操作类型。
37      举例：
38              unsigned long int a; int b;
39              a + b;  // 将b变量隐式类型转换为unsigned long int类型。
40      第3种情况：如果有符号数有效范围可以包含无符号数所有的值，
41              则将无符号数类型隐式转换为有符号数的类型
42              unsigned int a; long int b; // 64位系统
43              a + b;  // 变量a被隐式类型转换为long int类型。
44
45      第4种情况 ：  如果以上三种情况都不满足，
```

```
46          将两个操作数都转换为与有符号操作数类型相对应的无符号类型。
47      举例:
48          long long int a; unsigned long int b;
49          a + b; // 将a和b都转换为unsigned long long int
50
51          unsigned int a; long int  b; // 32位系统
52          a + b;  // 变量a和b被隐式类型转换为unsigned long int类型、
53
54
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    printf("3 + 3.14 = %lf\n", 3 + 3.14);
    // int类型的3被隐式转换为double类型之后，在进行运算


    // 1. 在进行算数运算时，类型不一致时，会发生隐式类型转换
    int a = 5;
    float b = 5.12;
    printf("a + b = %f\n", a + b);  // a被隐式类型转换为float

```

```
13      int i = -10;
14      unsigned int j = 5;
15      unsigned int k = i + j;  // i会被隐式类型转换为unsigned int类型
16      printf("k = %u\n", k); // 输出很大的数
17      printf("k = %d\n", k); // 输出-5 将无符号整型k,隐式类型转换为int类型
18
19      // 数据在内存中存放的是二进制的补码
20      // -10的源码，补码，反码
21      // 源码：1000 0000 0000 0000 0000 0000 0000 1010
22      // 反码：1111 1111 1111 1111 1111 1111 1111 0101
23      // 补码：1111 1111 1111 1111 1111 1111 1111 0110
24      // 5的源码，反码，补码一样
25      // 补码：0000 0000 0000 0000 0000 0000 0000 0101
26      // 计算机在计算时将i转换为unsigned int无符号类型
27      // i = 1111 1111 1111 1111 1111 1111 1111 0110
28      // j = 0000 0000 0000 0000 0000 0000 0000 0101
29      // 和= 1111 1111 1111 1111 1111 1111 1111 1011
30
31      unsigned int x = 5;
32      char y = -5;
```

```
33      printf("x + y = %u\n", x + y);  // 输出0
34      printf("x + y = %d\n", x + y);  // 输出0
35      // x的源码，反码，补码
36      // 补码：0000 0000 0000 0000 0000 0000 0000 0101
37      // y的源码，反码，补码
38      // 源码：1000 0101
39      // 反码：1111 1010
40      // 补码：1111 1011
41      // 将y提升为unsigned int类型，y是一个有符号的字符类型，高位补充符号位
42      // 提升之后的结果：1111 1111 1111 1111 1111 1111 1111 1011
43      //                 + 0000 0000 0000 0000 0000 0000 0000 0101
44      //                 = 0000 0000 0000 0000 0000 0000 0000 0000
45
46      unsigned int p = 10;
47      short q = -5;
48      // 将q提升为unsigned int类型
49      printf("p + a = %u\n", p + q);  // 输出为5
50      printf("p + a = %d\n", p + q);  // 输出为5
51      // p 源码，反码，补码
52      // 补码：0000 0000 0000 0000 0000 0000 0000 1010
53      // q 源码，反码，  补码：
```

```
54      // 源码：1000 0000 0000 0101
55      // 反码：1111 1111 1111 1010
56      // 补码：1111 1111 1111 1011
57      // 将q提升为unsigned int类型，高位补符号位
58      // 提升之后数据为：1111 1111 1111 1111 1111 1111 1111 1011
59      //                 + 0000 0000 0000 0000 0000 0000 0000 1010
60      //                 = 0000 0000 0000 0000 0000 0000 0000 0101
61
62      // 2. 赋值过程也会发生隐式类型转换，将右侧操作数的类型，
63      //       转换为左侧操作数的类型，不需要考虑数据类型的优先级的问题
64      //       只需要考虑左侧操作数的类型
65
66      unsigned int s;
67      unsigned short ss = 1000;
68      s = ss;    // 将ss隐式类型转换为unsigned int类型的
69                 // 小的类型赋值给大的类型，安全。
70
71      unsigned short u = 0xFFFF;
72      unsigned char  c;
73      c = u;  // 将u隐式类型转换为unsigned char类型
74                 // 大的类型赋值给小的类型，数据会被截断和丢失，不安全，慎用。
```

```
75
76      // 3. 首先对两个操作数进行整数提升，如果两个操作数的类型相同，过程结束
77      // 否则依照以下规则进行提升，一旦遇到可以适配的规则将不在考虑其他的规则。
78      //
79      // 第1种情况：如果两个操作数都为有符号数或者无符号数，将等级较低的操作数类型，
80      //  转换为等级较高的操作数类型。
81      // 举例：
82          unsigned int ui = 1000;
83          unsigned long int uli = 2000;
84          printf("ui + uli = %lu\n",ui + uli); // 变量ui被隐式类型转换为unsigned long int类型
85      //第2种情况：如果无符号操作数类型的优先级高于有符号数的类型优先级，
86      //  将有符号操作数的类型转换为无符号数的操作类型。
87      //举例：
88          unsigned long int uli1 = 10000;
89          int i1 = 2000;
90          printf("uli1 + i1 = %lu\n", uli1 + i1);  // 将i1变量隐式类型转换为unsigned long int类型。
91      // 第3种情况：如果有符号数有效范围可以包含无符号数所有的值，
92      //  则将无符号数类型隐式转换为有符号数的类型
93          unsigned int ui1 = 30000;
```

```
94          long int li = 40000;    // 64位系统
95          printf("ui1 + ui = %ld\n", ui1 + li);  // 变量ui1被隐式类型转换为long int类型。
96
97      // 第4种情况 ：  如果以上三种情况都不满足，
98      //  将两个操作数都转换为与有符号操作数类型相对应的无符号类型。
99      // 举例:
100         long long int lli = 10000;
101         unsigned long int uli2 = 20000;
102         printf("lli + uli2 = %llu\n", lli + uli2); // 将lli和uli都转换为unsigned long long int
103
104     return 0;
105 }
106
```

```
1  练习题1：以下代码的输出结果为 no
2  #include <stdio.h>
3
4  int main(int argc, const char *argv[])
5  {
6      // 0xFF编译器当成一个整数看待，在内存中存储为1111 1111
7      // 但是a是一个char类型，因此最高位为符号为，
```

```
8       // 1111 1111 在内存存储的补码，最高位为1，表示负数
9       // 1111 1111(补码) --> 1111 1110(反码) --> 1000 0001(源码)
10      char a = 0xff;
11      printf("%d\n" a);  // 结果为-1
12
13      // 0xFF是一个int类型，
14      // a是一个char类型的，需要进行整型提升，提升为int类型，高位补符号位
15      // if (0xffffffff == 0xFF)
16      if (a == 0xFF)
17      {
18          printf("yes\n");
19      }
20      else
21      {
22          printf("no\n");
23      }
24      return 0;
25  }
26
27  练习题1：以下代码的输出结果为 yes
28  #include <stdio.h>
29
30  int main(int argc, const char *argv[])
31  {
32      char a = 0xff;
33      if (a == (char)0xFF)
34      {
35          printf("yes\n");
```

```
36      }
37      else
38      {
39          printf("no\n");
40      }
41      return 0;
42  }
43  练习题2：以下代码的输出结果为 yes
44  #include <stdio.h>
45
46  int main(int argc, const char *argv[])
47  {
48      char a = -10;
49      unsigned char b = 5;
50      // a和b都会被整数提升为int类型，
51      // 如果是有符号数，高位补符号位
52      // 如果为无符号数，高位补0
53
54      // -10的源码，补码，反码
55      // 源码：1000 0000 0000 0000 0000 0000 0000 1010
56      // 反码：1111 1111 1111 1111 1111 1111 1111 0101
57      // 补码：1111 1111 1111 1111 1111 1111 1111 0110
58      // 5的源码，反码，补码一样
59      // 补码：0000 0000 0000 0000 0000 0000 0000 0101
60      // 计算机在计算时将a和b转换为int类型
61      // a = 1111 1111 1111 1111 1111 1111 1111 0110
```

```
62      // b = 0000 0000 0000 0000 0000 0000 0000 0101
63      // 和= 1111 1111 1111 1111 1111 1111 1111 1011
64      if ((a + b) < 0)
65      {
66          printf("yes\n");
67      }
68      else
69      {
70          printf("no\n");
71      }
72      return 0;
73  }
74
75  练习题3：以下代码的输出结果为  no
76  #include <stdio.h>
77
78  int main(int argc, const char *argv[])
79  {
80      int a = -10;
81      unsigned int b = 5;
82      // a 被隐式类型转换为unsigned int类型
83
84      // -10的源码，补码，反码
85      // 源码：1000 0000 0000 0000 0000 0000 0000 1010
86      // 反码：1111 1111 1111 1111 1111 1111 1111 0101
87      // 补码：1111 1111 1111 1111 1111 1111 1111 0110
```

```
88      // 5的源码，反码，补码一样
89      // 补码: 0000 0000 0000 0000 0000 0000 0000 0101
90      // 计算机在计算时将a和b转换为unsigned int类型
91      // a = 1111 1111 1111 1111 1111 1111 1111 0110
92      // b = 0000 0000 0000 0000 0000 0000 0000 0101
93      // 和= 1111 1111 1111 1111 1111 1111 1111 1011
94      if ((a + b) < 0)
95      {
96          printf("yes\n");
97      }
98      else
99      {
100         printf("no\n");
101     }
102     return 0;
103 }
```

char a = 0x0000 0000FF; // 将0xFF常量赋值给a变量对应的空间

0x0000 0000FF ----> 整数，源码反码补码一样 0x0000 00FF

将0x0000 00FF值赋值给a变量，会发生隐式类型转换，将0xFF写到a变量对应的内存空间中。

而a变量是一个有符号的char类型，最高位为符号位，由于最高位为1，此时变量a内存空间中存储的就是一个负数的补码，将此补码转换为源码：

1111 1111 ---> 补码，a变量对应内存空间存储的数据

1111 1110 ---> 反码

1000 0001 ---> 源码， 使用%d打印的结果为-1.

1.2 显式类型转换

```
格式：
    (类型名)表达式;

举例：
    unsigned char c;
    (int)c;

    int a, b;
    (long int)a + b;    // ()优先级高于+运算符的优先级
                // 先将a强转为long int类型，
                // b被隐式类型转换为long int类型，然后在进行加法运算。
```

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

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

    // 1. 如一个浮点类型，提取浮点类型数据的小数部分
    float f = 3.1415, f_part;
    f_part = f - (int)f;  // f被强转为int类型  (int)f = 3
                          // (int)f 在进行隐式类型转换得到 3.000000
    printf("f_part = %f\n", f_part);

    // 2. 两个整型进行除法运算时，结果可能为小数

    int a = 100, b = 3;
    float c = a / b;  // 33.000000
                      // 先使用int类型进行除法运算，再将结果隐式类型转换为float
    printf("c = %f\n", c);  // 33.000000

    c = (float)a / b;  // 将a强转为float类型，b也会被隐式类型转换为float
    printf("c = %.6f\n", c);   // 33.333332

    c = (float)(a / b); // 对运算的结果进行强制类型转换
    printf("c = %f\n", c);  // 33.000000
```

```
23
24      // 3. 两个数进行算数运算时，有可能会发生越界，
25      // 此时可以使用强转赋值给一个更大类型的变量
26      int x = 10000, y = 20000;
27      int z = x * y;    // 看似没有问题，实际上如果结果很大时可能会越界
28
29      long int s = (long int)x * y; // 通过类型转换解决越界的问题。
30
31      return 0;
32  }
33
```

2、运算符使用

2.1 运算符分类

```
算数运算符:  + - * / %(模除/取余)
自增自减运算符: ++ --
关系运算符: > < >= <= !=  ==
逻辑运算符: && || !
位运算符:  &  | ~ ^  << >>
赋值类运算符: =  += -= *= /= %= <<= >>= &= |= ~=  ^=
条件运算符(三目运算符) : ? :
sizeof运算符:  sizeof(数据类型)  ---> 计算数据类型占用内存空间的大小,
                                                以字节为单位。
逗号运算符: ,  (开发中基本不用, 笔试题常见)
```

2.2 算数运算符

```
格式:
    左操作数 # 右操作数;
              |----> 算数运算符: + - * / % (模除/取余)

    不可以对浮点型数据进行模除运算
```

```c
#include <stdio.h>
#define PI  3.14
int main(int argc, const char *argv[])
{
    int a = 10, b = 20;
    printf("a + b = %d\n", a + b);
    int r = 3;
```

```
    printf("圆的面积 = %f\n", r * r * PI);

    // %% : 输出一个%
    printf("b %% a = %d\n", b % a);
    // printf("b %% 1.5 = %d\n", b % 1.5); // 编译报错，不能对浮点数进行模除运算
    return 0;
}

```

```
练习题：
    给定一个100 ~ 999之间的数，提取个位，十位，百位，
    并对个位，十位，百位，进行求和。

```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int num = 867;
    int g, s, b;
    g = num % 10;
    s = num / 10 % 10;
    b = num / 100;
    printf("g + s + b = %d\n", g + s + b);
    return 0;
}

```

2.3 自增自减运算符

```
单目运算符:
格式:
     #操作数;          ---> 前自增自减，  先自增自减之后再使用。
     操作数#;          ---> 后自增自减，  先使用，再进行自增自减。

     自增自减运算符每次运算完成之后都自动加1，或者减1.

自增自减运算符在笔试题中比较常见，同时在开发中也经常使用。
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int a = 10;             // a = 10
    int b = a++;            // b = 10; (先使用变量a的值)  a = 11 (a变量在自动加1)
    int c = ++a;            // c = 12; (a先自动加1，然后在使用) a = 12

    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);

    // 不可以对常量进行自增自减运算，比如 10++ --10;
```

```
    return 0;
}

```

```
1. 有以下程序
main()
{
    int m = 12, n = 34;
    printf ("%d%d", m++, ++n);
    printf ("%d%d\n", n++, ++m );
    printf ("%d%d\n", n, m );          //
3614
}
程序运行后的输出结果是 [单选题] [必答题]
   ○ A)12353514 正确
   ○ B)12353513
   ○ C)12343514
   ○ D)12343513

2. 有以下程序
#include <stdio.h>

int main(int argc, const char *argv[])
{
    int m=3,n=4,x;
    // 单目运算符进行结合时，从右到左进行结合
    // m++是后加加，先使用m的值，对3取负运
算，结果赋值给x,x = -3
    // 然后m中的值在加1，m = 4
    x=-(m++);
    x=x+8/++n;
```

```
26      printf("%d\n",x);
27      return 0;
28  }
29
30  程序运行后的输出结果是 [单选题] [必答题]
31      ○ A) 3
32      ○ B) 5
33      ○ C) -1
34      ○ D) -2 正确
35  3. 以下选项中，与k=n++完全等价的表达式是 [单选题] [必答题]
36      ○ A) k=n,n=n+1  正确
37      ○ B) n=n+1,k=n
38      ○ C) k=++n
39      ○ D) k+=n+1
40
41  4. 若有说明和语句：
42  int a=5;
43  a++;
44  此处表达式a++的值是 [单选题] [必答题]
45      ○ A) 7
46      ○ B) 6
47      ○ C) 5(正确)
48      ○ D) 4
```

2.4 关系运算符

```
格式:
    左操作数 # 右操作数
              |---> 关系运算符: > < >= <= != ==

    关系表达式的运算结果为bool类型，非0表示真，0表示假。

    关系表达式一般和if分支语句，循环语句配合使用。
```

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int a = 10, b = 20;

    printf("a > b = %d\n", a > b);  // 0
    printf("a < b = %d\n", a < b);  // 1


    if (a > b)  // 条件判断语句和分支语句结合使用
    {
        printf("Yes\n");
    }
    else
    {
        printf("No\n");
    }
```

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

2.5 逻辑运算符

```
1       左操作数 # 右操作数
2                 |---> 逻辑运算符 ：  && ||
3
4       #操作数
5       |----> 逻辑运算符 ： !(单目运算符)
6
7       && ：两个表达式都为真则结果为真(结果为1)，只要有一个为假则结果为假(结果为0)；
8       || ：两个表达式只要有一个为真，则结果为真(结果为1)，
9               只有两个表达式都为假时，结果为假(结果为0).
10      !: 真变假(结果为0)，   假变真(结果为1)
```

```
1  初始化三角形的三个边，判断是否可以构成一个三角形；
2  条件：任意两边长度之和大于第三个边。
3  #include <stdio.h>
4  int main(int argc, const char *argv[])
5  {
6      int a = 10, b = 9, c = 6;          // 可以构成三角形
7      // int a = 2, b = 9, c = 6;        // 不可以构成三角形
```

```
8
9       printf("结果为1，可以组成一个三角形 : %d\n",
10                  (a + b > c) && (a + c > b) && (b + c > a));
11
12      if ((a + b > c) && (a + c > b) && (b + c > a))
13      {
14          printf ("可以构成三角形\n");
15      }
16      else
17      {
18          printf("不可以构成三角形\n");
19      }
20      return 0;
21  }
22
```

```
1  准备一个成绩，判断成绩是否合理，成绩的范围为0-100.
2  #include <stdio.h>
3  int main(int argc, const char *argv[])
4  {
5      int score = 200;
6
7      if (score < 0 || score > 100)
8      {
9          printf("成绩不合理\n");
10     }
11     else
```

```
    {
        printf("成绩合理\n");
    }

    if (score >= 0 && score <= 100)
    {
        printf("成绩合理\n");
    }
    else
    {
        printf("成绩不合理\n");
    }
    return 0;
}

```

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int a = 10, b = 9, c = 6;        // 可以构成三角形
    // int a = 2, b = 9, c = 6;      // 不可以构成三角形

    printf("结果为1，可以组成一个三角形 : %d\n",
           (a + b > c) && (a + c > b) && (b + c > a));

    if (!(a + b > c) || !(a + c > b) || !(b + c > a))
```

```
11      {
12          printf ("不可以构成三角形\n");
13      }
14      else
15      {
16          printf("可以构成三角形\n");
17      }
18      return 0;
19  }
20
```

```
1  总结:
2      多个表达式进行逻辑运算时，会存在截断。
3      多个表达式进行与运算时，从左到右进行运算，当有一个表达式结果为假时，
4      后边的表达式就不再进行运算。
5
6      多个表达式进行或运算时，从左到右进行于是暖，当有一个表达式结果为真时，
7      后边的表达式就不再进行运算。
8
```

2.6 位运算符

```
1       左操作数 # 右操作数;
2                   |---> 位运算符:  &   |   ^   <<
    >>
3
4       ~操作数;
5       |---> 位运算符: ~(按位取反，单目运算符)
6
7       位运算符主要用于两个数的二进制的每个位进行
    算数运算。
8       任何一个类型最低位为第0位，比如:
    unsigned int a;
9       高位
      低位
10      **** **** **** **** **** **** **** ****
    ****
11      31
      0
12
13
```

按位与运算：与0清0，与1不变

左操作数	运算符	右操作数	运算结果
0	&	0	0
1	&	0	0
0	&	1	0
1	&	1	1

按位或运算：或1置1，或0不变

左操作数	运算符	右操作数	运算结果
0	|	0	0
1	|	0	1
0	|	1	1
1	|	1	1

按位异或：异或0不变，异或1取反（相异为1，相同为0）

左操作数	运算符	右操作数	运算结果
0	^	0	0
1	^	0	1
0	^	1	1
1	^	1	0

按位取反：

```
1 1变0，0变1
```

左移和右移：

```
无符号数:
    左移: 高位移出, 低位补0
    右移: 低位移出, 高位补0

有符号数:
    左移: 高位移出, 低位补0
    右移: 低位移出, 高位补符号位

    左移每移动一位, 相当于乘以2, 右移移动一位相当于除以2.
    举例:
        0000 1000 右移一位 0000 0100
        0000 1000 左移一位 0001 0000
```

```
练习题:
    假设有一个unsigned int a = 0x12345678;

    1> 将a变量的第[5]位清0, 保持其他位不变;
    2> 将a变量的第[20]位置1, 保持其他位不变;
    3> 将a变量的第[15:8]位清0, 保持其他位不变;
    4> 将a变量的第[31:28]位置1, 保持其他位不变;
    5> 将a变量的第[7:4]位设置位0110, 保持其他位不变。

#include <stdio.h>
int main(int argc, const char *argv[])
{
```

```
13      //假设有一个unsigned int a = 0x12345678;
14      unsigned int a = 0x12345678;
15      // 0001 0010 0011 0100 0101 0110 0111 1000
16      // 1> 将a变量的第[5]位清0，保持其他位不变；
17      // 1111 1111 1111 1111 1111 1111 1101 1111
18      a = a & (unsigned int)0xFFFFFFDF;
19      a = a & (~((unsigned int)0x1 << 5));
20      // 0000 0000 0000 0000 0000 0000 0000 0001  < 5
21      // 0000 0000 0000 0000 0000 0000 0010 0000  ~
22      // 1111 1111 1111 1111 1111 1111 1101 1111
23
24
25      // 2> 将a变量的第[20]位置1，保持其他位不变；
26      a = a | ((unsigned int)0x1 << 20);
27
28      // 3> 将a变量的第[15:8]位清0，保持其他位不变；
29      a = a & (~((unsigned int)0xFF << 8));
30
31
```

```
32      // 4> 将a变量的第[31:28]位置1，保持其他位不变；
33      a = a | (unsigned int)0xF0000000;
34      a = a | ((unsigned int)0xF << 28);
35      // 0000 0000 0000 0000 0000 0000 0000 1111  << 28
36      // 1111 0000 0000 0000 0000 0000 0000 0000
37
38
39      // 5> 将a变量的第[7:4]位设置为0110，保持其他位不变。
40      // 1.先把相对应的位清0
41      a = a & (~((unsigned int)0xF << 4));
42      // 2. 再把相对应的位置1
43      a = a | ((unsigned int)0x6 << 4);
44
45      // 以下用法不推荐使用
46      // 1. 先把相对应的位置1
47      a = a | ((unsigned int)0xF << 4);
48      // 2. 再将对应的位清0
49      a = a & (~((unsigned int)0x9 << 4));
50
51      return 0;
52  }
53
54
```

2.7 赋值类的运算符

```
一般形式：
    左操作数 = 右操作数;
        |
        |----> 左操作数只能是一个变量，不可以是常量

符合类赋值运算符：
    左操作数 # 右操作数;
        |      | ----> += -= *= /= %= &= |= ^= ~= <<= >>=
        |----> 左操作数只能是一个变量，不可以是常量

      效果：比如：  a += b;    ==> a = a + b;
```

2.8 条件运算符(三目运算符)

```
格式：
    表达式1 ? 表达式2 : 表达式3;
    表达式1成立，执行表达式2，
    表达式1不成立，执行表达式3；
```

```
案例：使用条件表达式实现比较两个整数的大小。
#include <stdio.h>
```

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

    int max_value;
    int a = 10, b = 20;
    max_value = a > b ? a : b;
    printf("a和b的最大值为 max_value = %d\n", max_value);


    return 0;
}

```

使用宏定义实现求两个数的最大值?

使用宏定义和条件运算符实现三角形三个边长是否可以构成三角形。

```c
#include <stdio.h>
#define MAX_VALUE(a,b)   (a > b) ? a : b

#define TRIANGLE(a,b,c)  ((a+b>c) && (a+c>b) && (b+c>a)) ? printf("Yes\n") : printf("no\n")

int main(int argc, const char *argv[])
{
    int max_value;
    max_value = MAX_VALUE(50, 300);
```

```
14      printf("max_value = %d\n",
    max_value);
15
16      TRIANGLE(10,8,7);
17
18      return 0;
19  }
20
```

作业：

```
1  吸收今天的内容：
2      隐式类型转换；
3      算数运算符：逻辑运算，位运算，条件运算符
4
5
```

明天授课内容：

```
1  sizeof 逗号
2  输入输出的函数：printf/scanf   gets/puts
   getchar/putchar
3  分支语句(if else / switch case).
4
```