NWEN241
Low-Level Programming

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

This Lecture

- Bitwise operators, masks and how to use them

Bitwise Operators

- Logical operators
  - ~: complement (turns “1” to “0” and “0” to “1”)

```
int a = 0;   /* 00000000 ... 00000000 */
int b = ~a;  /* 11111111 ... 11111111 */
```

  - &: and
  - ^: exclusive or
  - |: inclusive or

a	b	a & b	a ^ b	a | b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	0	1

Bitwise Operators

- Logical operators
  - ~, &: examples

```
int a = 0;      /* 00000000 ... 00000000 */

int b = ~a;     /* what is b’s value? */
```

Bitwise Operators

- Logical operators
  - ~, &: examples

```
int a = 0;      /* 00000000 ... 00000000 */

int b = ~a;     /* 11111111 ... 11111111 */

int c = a & b;
            /* a: 00000000 ... 00000000 */
            /*    &&&&&&&& ... &&&&&&&& */
            /* b: 11111111 ... 11111111 */
            /* c: 00000000 ... 00000000 */
```

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 00000000 00000000 00000000 00000000 */
/* leftmost bit is called high-order bit */
/* rightmost bit is called low-order bit */

/* 00000000 00000000 00000000 00000000 */
/* leftmost byte is called high-order byte */
/* rightmost byte is called low-order byte */
```

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 00000000 00000000 00000000 00000000 */
/* 10000000 00000000 00000000 00000000 */
/* leftmost bit set to 0 means... */
/* leftmost bit set to 1 means... */
```

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 00000000 00000000 00000000 00000000 */
/* 10000000 00000000 00000000 00000000 */
/* leftmost bit set to 0 means positive */
/* leftmost bit set to 1 means negative */
```

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 00000000 00000000 00000000 00000000 */
/* 10000000 00000000 00000000 00000000 */
/* leftmost bit set to 0 means positive */
/* leftmost bit set to 1 means negative */
```

  - Is the leftmost bit simply a sign bit?

```
/* assume an integer has only 4 bits */
/* what decimal int does 1011 represent? */
```

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 1011 represents:
 * 1*(-8) + 0*4 + 1*2 + 1*1 = -5
 */
/* 11111111 11111111 11111111 11111011 ? */
```

  - Sounds a bit difficult to figure out a negative integer in binary notation

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 1011 represents:
 * 1*(-8) + 0*4 + 1*2 + 1*1 = -5
 */
/* 11111111 11111111 11111111 11111011 ? */
```

  - Sounds a bit difficult to figure out a negative integer in binary notation - not really, two’s complement

Bitwise Operators

- Complement operator (~)
  - Representing negative integer in binary notation

```
/* 1011 represents:
 * 1*(-8) + 0*4 + 1*2 + 1*1 = -5
 */
/* 11111111 11111111 11111111 11111011 ? */
```

  - Sounds a bit difficult to figure out a negative integer in binary notation - not really, two’s complement

```
int a = n;   /* a = -(~a + 1) */
              /* ~a = -(a + 1) */
```

Bitwise Operators

- Complement operator (~)
  - Sounds a bit difficult to figure out a negative integer in binary notation - not really, two’s complement

```
int a = n;   /*  a = -(~a + 1) */
```

Bitwise Operators

- Complement operator (~)
  - Sounds a bit difficult to figure out a negative integer in binary notation - not really, two’s complement

```
int a = n;   /*  a = -(~a + 1) */


 a = 11111111 11111111 11111111 11111011
~a = 00000000 00000000 00000000 00000100
 a = -(~a + 1) = -(101) = -5
 a = -5
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
a = a << 31;
/* 10000000 00000000 00000000 00000000 */
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
a = a << 31;
/* 10000000 00000000 00000000 00000000 */
a = a >> 4;
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
a = a << 31;
/* 10000000 00000000 00000000 00000000 */
a = a >> 4;
/* 11111000 00000000 00000000 00000000 */
/* or */
/* 00001000 00000000 00000000 00000000 */
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
a = a << 31;
/* 10000000 00000000 00000000 00000000 */
a = a >> 4;
/* 11111000 00000000 00000000 00000000 */
/* or */
/* 00001000 00000000 00000000 00000000 */
/* for int, system dependent */
```

Bitwise Operators

- Shift operators
  - <<
  - >>

```
int a = 1;
/* 00000000 00000000 00000000 00000001 */
a = a << 31;
/* 10000000 00000000 00000000 00000000 */
a = a >> 4;
/* 11111000 00000000 00000000 00000000 */
/* or */
/* 00001000 00000000 00000000 00000000 */
/* for unsigned, it is */
/* 00001000 00000000 00000000 00000000 */
```

Masks

- Two popular masks
  - 1:   00000000 00000000 00000000 00000001

```
int a = 5, low_order_bit, mask = 1;
low_order_bit = a & mask;        /* ? */
/* only the low-order bit is retained */

/* if we want to find the value of a */
/* particular bit... */
```

Masks

- Two popular masks
  - 1:   00000000 00000000 00000000 00000001

```
int a = 5, low_order_bit, mask = 1;
low_order_bit = a & mask;        /* 1 */
/* only the low-order bit is retained */

/* if we want to find the value of a */
/* particular bit, shift the bit to the  */
/* low-order/rightmost bit */

int second_bit = (a >> 1) & mask;/* ? */
```

Masks

- Two popular masks
  - 1:   00000000 00000000 00000000 00000001

```
int a = 5, low_order_bit, mask = 1;
low_order_bit = a & mask;        /* 1 */
/* only the low-order bit is retained */

/* if we want to find the value of a */
/* particular bit, shift the bit to the  */
/* low-order/rightmost bit */

int second_bit = (a >> 1) & mask;/* 0 */
```

Masks

- Two popular masks
  - 1:   00000000 00000000 00000000 00000001

```
int a = 5, low_order_bit, mask = 1;
low_order_bit = a & mask;   /* 1 */
/* only the low-order bit is retained */

/* if we want to find the value of a */
/* particular bit, shift the mask */

int second_bit = (a & (mask << 1)) >> 1;
```

Masks

- Two popular masks
  - 1:   00000000 00000000 00000000 00000001

```
int a = 5, low_order_bit, mask = 1;
low_order_bit = a & mask;   /* 1 */
/* only the low-order bit is retained */

/* if we want to find the value of a */
/* particular bit, shift the mask */

int second_bit = (a & (mask << 1)) ? 1 : 0;
```

Masks

- Two popular masks
  - 255:   00000000 00000000 00000000 11111111

```
int a = 261, low_order_byte, mask = 255;
low_order_byte = a & mask;  /* ? */
/* only the low-order byte is retained */
```

Masks

- Two popular masks
  - 255:   00000000 00000000 00000000 11111111

```
int a = 261, low_order_byte, mask = 255;
low_order_byte = a & mask;  /* 5 */
/* only the low-order byte is retained */
```

Masks

- An example (print an int in bits)

Masks

- An example (print an int in bits)

```
int i, a, n = 32;
scanf("%d", &a);
int mask = 1<<(n-1); /* mask = ... */
for (i=1; i<=n; i++)
{ putchar((a & mask)? '1':'0');
  a <<=1;  /* shift next bit to where? */
  if (!(i%8))        /* for decoration */
  { if (i<n) putchar(' ');
    if (i==n) putchar('\n');
  }
}
```

Masks

- An example (print an int in bits)

```
int i, a, n = 32;
scanf("%d", &a);
int mask = 1<<(n-1); /* mask = 1000... */
for (i=1; i<=n; i++)
{ putchar((a & mask)? '1':'0');
  a <<=1;  /* shift next bit to where? */
  if (!(i%8))        /* for decoration */
  { if (i<n) putchar(' ');
    if (i==n) putchar('\n');
  }
}
```

Masks

- An example (print an int in bits)

```
int i, a, n = 32;
scanf("%d", &a);
int mask = 1<<(n-1); /* mask = 1000... */
for (i=1; i<=n; i++)
{ putchar((a & mask)? '1':'0');
  a <<=1;  /* shift next bit to leftmost*/
  if (!(i%8))        /* for decoration */
  { if (i<n) putchar(' ');
    if (i==n) putchar('\n');
  }
}
```

Masks

- Packing/unpacking data

```
/* pack a structure into an integer */
```

Masks

- Packing/unpacking data

```
/* pack a structure into an integer */
typedef struct student {
  int id;
  int age;
  char gender;
  char classrep;
} Student;

int pack(int id, int age, char cr, char g);
Student unpack(int stu);
void bitprint(int a);
```

Masks

- Packing/unpacking data
  - Packing

```
/* id(23bits), age(7bits), */
/* gender(1bit), classrep(1bit) */
int pack(int id, int age, char g, char cr)
{ int student = 0; /* 00000000 ... 00000000 */
  student |= (g=='M' || g=='m')? 1:0;
  student |= ((cr=='Y' || cr=='y')? 1:0) << 1;
  student |= age << 2;
  student |= id << 9;
  return student;  /* contain packed info */
}
```

Masks

- Packing/unpacking data
  - Packing (for you to do)

```
/* id(23bits), age(7bits), */
/* gender(1bit), classrep(1bit) */

/* implement pack with following prototype */

int pack(Student *);
```

Masks

- Packing/unpacking data
  - Unpacking

```
/* id(23), age(7), gender(1), classrep(1) */
Student unpack(int stu)
{ Student a;
  int mask1=1, mask2 = 127, mask3 = 8388607;
     /* what should the masks look like??? */
  a.gender = ((stu&mask1)? 'M':'F');
  a.classrep = ((stu&(mask1<<1))>>1)? 'Y':'N';
  a.age = ((stu&(mask2<<2))>>2);
  a.id = ((stu&(mask3<<9))>>9);
  return a;
}
```

Masks

- Packing/unpacking data
  - Unpacking

```
/* id(23), age(7), gender(1), classrep(1) */
Student unpack(int stu)
{ Student a;
  int mask1=1, mask2 = 127, mask3 = 8388607;
     /* 1 ‘1’, 7 ‘1’s, 23 ‘1’s */
  a.gender = ((stu&mask1)? 'M':'F');
  a.classrep = ((stu&(mask1<<1))>>1)? 'Y':'N';
  a.age = ((stu&(mask2<<2))>>2);
  a.id = ((stu&(mask3<<9))>>9);
  return a;
}
```

Masks

- Packing/unpacking data
  - Unpacking

```
/* id(23), age(7), gender(1), classrep(1) */
Student unpack(int stu)
{ Student a;
  int mask1=1, mask2 = 127, mask3 = 8388607;
     /* 1 ‘1’, 7 ‘1’s, 23 ‘1’s */
  a.gender = ((stu&mask1)? 'M':'F');
  a.classrep = (stu&(mask1<<1))? 'Y':'N';
  a.age = ((stu&(mask2<<2))>>2);
  a.id = ((stu&(mask3<<9))>>9);
  return a;
}
```

Next Week/Lecture

- File handling
- Writing large programs