Introduction to Scientific and Engineering Computation (BIL 104E)

Lecture 5
Conditional Operators
&
Data Modifiers and Math Functions

Conditional Operators: Measuring Data Sizes

Question: How do you know the size of a data type on your machine?

Answer: You can measure the data type size by using the **sizeof** operator provided by C.

The general form of the **sizeof** operator is

sizeof (expression)

Expression indicates the data type or variable you want to measure their size.

The **sizeof** operator evaluates the size, in bytes, of its operand.

The **operand** of the **sizeof** operator may be a data type (such as int, char, or float), or a constant or the name of a variable.

Conditional Operators: Measuring Data Sizes

Using the sizeof Operator

```
1:  /* 08L01.c: Using the sizeof operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     char   ch = ' ';
7:     int    int_num = 0;
8:     float  flt_num = 0.0f;
9:     double dbl_num = 0.0;
10:
11:    printf("The size of char is: %d-byte\n", sizeof(char));
12:    printf("The size of ch is: %d-byte\n", sizeof ch );
13:    printf("The size of int is: %d-byte\n", sizeof(int));
14:    printf("The size of int_num is: %d-byte\n", sizeof int_num);
15:    printf("The size of float is: %d-byte\n", sizeof(float));
16:    printf("The size of flt_num is: %d-byte\n", sizeof flt_num);
17:    printf("The size of double is: %d-byte\n", sizeof(double));
18:    printf("The size of dbl_num is: %d-byte\n", sizeof dbl_num);
19:    return 0;
20: }
```

Conditional Operators: Measuring Data Sizes

Computer Screen

```
The size of char is: 1-byte
The size of ch is: 1-byte
The size of int is:4-byte
The size of int_num is:4-byte
The size of float is: 4-byte
The size of flt_num is: 4-byte
The size of double is: 8-byte
The size of dbl_num is:8-byte
```

Conditional Operators: Logical operators

There are three logical operators in the C language:

&&	The logical **AND** operator
‖	The logical **OR** operator
!	The logical **NEGATION** operator

The **AND** and **OR** operators, are binary operators; that is, they both take two operands.

Conditional Operators: AND Operator (&&)

A general format of the logical **AND** operator is:

exp1 **&&** exp2

The truth table of the AND operator

The Values Returned by the AND Operator		
exp1	exp2	&& Yields
nonzero	nonzero	1
nonzero	0	0
	nonzero	0
0	0	0

Conditional Operators: AND Operator (&&)

```
#include <stdio.h>

int main( )
{
   int num;

   num = 1;
   printf("%d && %d yields %d\n",(num%2 == 0), (num%3 == 0), (num%2 == 0) && (num%3 == 0));
   num = 2;
   printf("%d && %d yields %d\n",(num%2 == 0), (num%3 == 0), (num%2 == 0) && (num%3 == 0));
   num = 3;
   printf("%d && %d yields %d\n",(num%2 == 0), (num%3 == 0), (num%2 == 0) && (num%3 == 0));
   num = 6;
   printf("%d && %d yields %d\n",(num%2 == 0), (num%3 == 0), (num%2 == 0) && (num%3 == 0));
   getchar();

   return 0;
}
```

```
0 && 0 yields 0
1 && 0 yields 0
0 && 1 yields 0
1 && 1 yields 0
```

Conditional Operators: OR Operator (||)

A general format of the logical **OR** operator is:

exp1 || exp2

The truth table of the OR operator

The Values Returned by the OR Operator		
exp1	exp2	|| Yields
nonzero	nonzero	1
nonzero	0	1
0	nonzero	1
0	0	0

Conditional Operators: OR Operator (||)

Using the Logical OR Operator ||

```
1:  /* 08L03.c: Using the logical OR operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   num;
7:
8:     printf("Enter a single digit that can be divided\nby both 2 and 3:\n");
9:     for (num = 1; (num%2 != 0) || (num%3 != 0); )
10:       num = getchar() - '0';
11:    printf("You got such a number: %d\n", num);
12:    return 0;
13: }
```

```
Enter a single digit that can be divided
by both 2 and 3:
2
3
4
5
6
```

Conditional Operators: NEGATION Operator (!)

The general format of using the logical **NEGATION** operator is:

!expression

The truth table of the NEGATION operator

The Values Returned by the ! Operator	
expression	*Value Returned by* !
nonzero	0
0	1

Conditional Operators: NEGATION Operator (!)

Using the Logical Negation Operator (!)

```
1:  /* 08L04.c: Using the logical negation operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   num;
7:
8:     num = 7;
9:     printf("Given num = 7\n");
10:    printf("!(num < 7)  yields: %d\n", !(num < 7));
11:    printf("!(num > 7)  yields: %d\n", !(num > 7));
12:    printf("!(num == 7) yields: %d\n", !(num == 7));
13:    return 0;
14: }
```

```
Given num = 7
!(num < 7)  returns: 1
!(num > 7)  returns: 1
!(num == 7) returns: 0
```

Conditional Operators: Manipulating Bits

A bit is the smallest storage unit in the computer world.

A bit can only hold the values **0** and **1** (0 and 1 are used to represent the off and on states of electronic.)

Each digit of a hex number consists of 4 bits. It is easy to convert a decimal number into a hex or a binary number.

Conditional Operators: Manipulating Bits

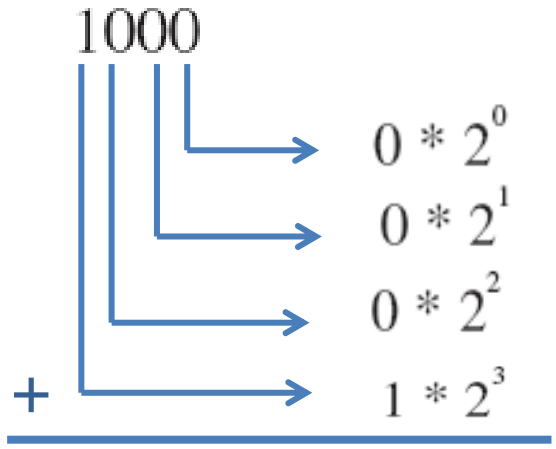
Numbers Expressed in Different Formats		
Hex	*Binary*	*Decimal*
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Conditional Operators: Manipulating Bits

How to convert a **decimal** number into a **binary** or a **hex** number

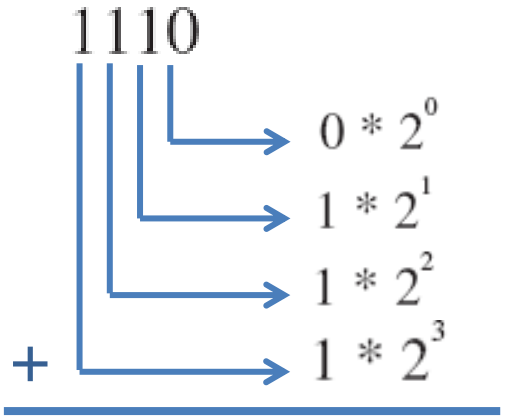
Binary is a 2-based numbering system. Each digit in a **binary** number is called a bit and can be 1 or 0.
If the position of a bit in a **binary** number is n, the bit can have a value of either 0 or 2 to the power of n.

Example 1



$2^3 \rightarrow 8$ (decimal)

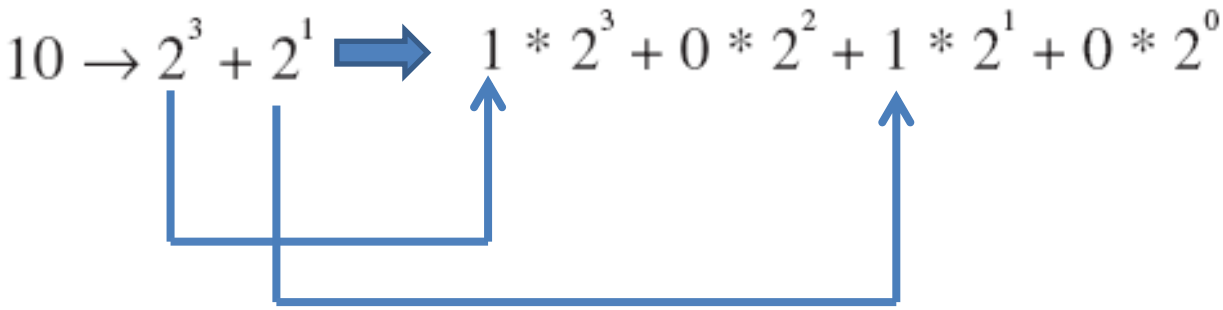
Example 2



14 (decimal)

Conditional Operators: Manipulating Bits

How to convert a **binary** number into a **decimal** number



Second way:

10 / 2 = 5 >>>>>> remainder = 0
5 / 2 = 2 >>>>>> remainder = 1
2 / 2 = 1 >>>>>> remainder = 0
1 / 2 = 0 >>>>>> remainder = 1

Least significant bit

Most significant bit

Conditional Operators: Bitwise Operators

There are six bit-manipulation operators in the C language

Operator	*Description*
&	The bitwise AND operator
¦	The bitwise OR operator
^	The bitwise exclusive OR (XOR) operator
~	The bitwise complement operator
>>	The right-shift operator
<<	The left-shift operator

The general forms of the bitwise operators are:

x & y

x | y

x ^ y

~x

Here **x** and **y** are operands.

Conditional Operators: Bitwise Operators

Truth Tables of Bitwise Operators

AND

x & y	z
0 0	0
0 1	0
1 0	0
1 1	1

OR

x | y	z
0 0	0
0 1	1
1 0	1
1 1	1

XOR

x ^ y	z
0 0	0
0 1	1
1 0	1
1 1	0

NOT

~y	z
0	1
1	0

Examples Using Bitwise Operators

Expressions			*Results*		
Decimal	*Hex*	*Binary*	*Decimal*	*Hex*	*Binary*
12 & 10	0x0C & 0x0A	1100 & 1010	8	0x08	1000
12 ¦ 10	0x0C ¦ 0x0A	1100 ¦ 1010	14	0x0E	1110
12 ^ 10	0x0C ^ 0x0A	1100 ^ 1010	6	0x06	0110
~12	~0x000C	~0000000000001100	65523	FFF3	1111111111110011

Conditional Operators: Bitwise Operators

Using Bitwise Operators

```
1:  /* 08L05.c: Using bitwise operators */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int   x, y, z;
7:
8:     x = 4321;
9:     y = 5678;
10:    printf("Given x = %u, i.e., 0X%04X\n", x, x);
11:    printf("      y = %u, i.e., 0X%04X\n", y, y);
12:    z = x & y;
13:    printf("x & y  returns: %6u, i.e., 0X%04X\n", z, z);
14:    z = x ¦ y;
15:    printf("x ¦ y  returns: %6u, i.e., 0X%04X\n", z, z);
16:    z = x ^ y;
17:    printf("x ^ y  returns: %6u, i.e., 0X%04X\n", z, z);
18:    printf("  ~x   returns: %6u, i.e., 0X%04X\n", ~x, ~x);
19:    return 0;
20: }
```

Conditional Operators: Bitwise Operators

Computer Screen

```
Given x = 4321, i.e., 0X10E1
      y = 5678, i.e., 0X162E
x & y  returns:  4128, i.e., 0X1020
x ¦ y  returns:  5871, i.e., 0X16EF
x ^ y  returns:  1743, i.e., 0X06CF
  ~x   returns: 61214, i.e., 0XEF1E
```

Conditional Operators: Bitwise Operators

Don't confuse the bitwise operators **&** and | with the logical operators **&&**
and ||.
For instance,

(x=1) & (y=10) ⟹ result: 0

$$\overbrace{00000000}^{\text{result: 0}} = 00000001 \,\&\, 00001010$$

is a completely different expression from

(x=1) && (y=10) ⟹ result: 1

Conditional Operators: Bitwise Operators

```
#include<stdio.h>

int main()
{
   int a, b, result;

   a=1;
   b=10;
   result = a && b;
   printf(" The result of a && b is %d\n", result);
   result = a & b;
   printf(" The result of a & b is %d\n", result);
   getchar();
   return 0;
}
```

```
The result of a && b is 1
The result of a & b is  0
```

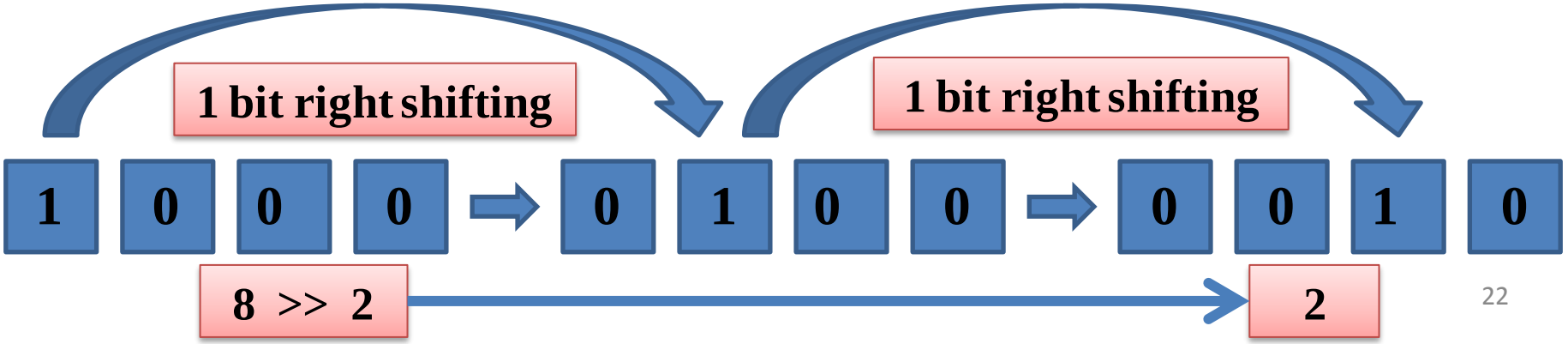
Conditional Operators: Shift Operators

The >> operator shifts the bits of an operand to the right; the << operator shifts the bits to the left.

The general forms of the two shift operators are

x >> y
x << y

Here **x** is an operand that is going to be shifted. **y** contains the specified number of places to shift.



Conditional Operators: Shift Operators

The operation of the shift-right operator (>>) is equivalent to dividing by powers of two:

x >> y

is equivalent to the following:

$x / 2^y$

Here x is a non-negative integer.

On the other hand, shifting to the left is equivalent to multiplying by powers of two:

x << y

is equivalent to

$x * 2^y$

Conditional Operators: Shift Operators

```
#include<stdio.h>
main()
{
    int num, i, power_of, temp, right_shift, divider, digit1, digit2, digit3, digit4;

    printf("please write decimal number?\n");
    scanf("%d", &num);
    printf("please write decimal number for right shifting?\n");
    scanf("%d", &right_shift);

    digit1 = num % 2;
    divider = num / 2;
    digit2 = divider % 2;
    divider /= 2;
    digit3 =divider % 2;
    divider /= 2;
    digit4 = divider % 2;
    printf("Binary representation of %d is %d %d %d %d \n",num, digit4, digit3, digit2,digit1);
    temp = num;
```

Conditional Operators: Shift Operators

```
num = num >> right_shift;
    digit1 = num % 2;
    divider = num / 2;
    digit2 = divider % 2;
    divider /= 2;
    digit3 =divider % 2;
    divider /= 2;
    digit4 = divider % 2;
    printf("Binary representation of %d right shifted number %d is %d %d %d %d
\n",right_shift, num, digit4, digit3, digit2,digit1);
    power_of = 1;
    for(i=1; i <= right_shift;i++)
          power_of *= 2;
    printf( "%d / %d gives same result as right shifting %d\n", temp, right_shift,
temp/power_of);
    getchar();
    getchar();
    return 0;
}
```

```
please write decimal number?
15
please write decimal number for right shifting?
2
Binary representation of 15 is 1 1 1 1
Binary representation of 2 right shifted number 3 is 0 0 1 1
15 / 2 gives same result as right shifting 3
```

Conditional Operators: x?y:z operation

The operator **?:** is called the conditional operator, which is the only operator that takes three operands.

The general form of the conditional operator is

x ? y : z

Here **x, y**, and **z** are three operand expressions. Among them, **x** contains the test condition, and **y** and **z** represent the two possible final values of the expression.

If **x** evaluates to nonzero (logically true), then **y** is chosen; otherwise, **z** is the result yielded by the conditional expression.

The conditional operator is used as a kind of shorthand for an **if** statement.

Conditional Operators: x?y:z operation

For instance:

x > 0 ? 'T' : 'F'

evaluates to '**T**' if the value of x is greater than **0**. Otherwise, the conditional expression evaluates to the value '**F**'.

Using the Conditional Operator

```
1:  /* 08L07.c: Using the ?: operator */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:      int   x;
7:
8:      x = sizeof(int);
9:      printf("%s\n",
10:         (x == 2)
11:         ? "The int data type has 2 bytes."
12:         : "int doesn't have 2 bytes.");
13:     printf("The maximum value of int is: %d\n",
14:         (x != 2) ? ~(1 << x * 8 - 1) : ~(1 << 15) );
15:     return 0;
16: }
```

```
int doesn't have 2 bytes.
The maximum value of int is: 2147483647
```

Data Modifiers

In order to have greater control over the data, the C keywords for the four data modifiers are:

- **signed**
- **unsigned**
- **short**
- **long**

Data Modifiers : Enabling or Disabling the Sign Bit

How does the computer represent a negative number in the binary format?

One bit can be used to indicate whether the value of a number represented in the binary format is negative. This bit is called the **sign bit**.

Two data modifiers, **signed** and **unsigned**, that can be used to enable or disable the **sign bit**.

Data Modifiers : The signed Modifier

The **leftmost** bit can be used as the sign bit.

if the **int** data type is **32** bits long and the rightmost bit is counted as bit **0**, you can use bit **31** (**leftmost**) as a sign bit.

When the sign bit is set to **1**, the C compiler knows that the value represented by the data variable is **negative**.

A data modifier, **signed**, can be used to indicate to the compiler that the **integer** data types (**char**, **int**, **short int**, and **long int**) use the sign bit.

By default, all the **integer** data types except the **char** data type are **signed** quantities

Data Modifiers : The unsigned Modifier

The **unsigned** modifier can be used to tell the C compiler that the specified data type is only capable of holding **non-negative** values.

Like the **signed** modifier, the **unsigned** modifier is meaningful only to the **integer** data types (**char**, **int**, **short int**, and **long int**).

The declaration

unsigned int x;

tells the C compiler that the integer variable **x** can only assume positive values using 32 bits.

```
unsigned int x, y;
x = 12345U;
y = 0xABCDu;
```

you can use **unsigned constant** decleration by suffixing **u** or **U**.

Data Modifiers : Changing Data Sizes

The C language gives you the flexibility to modify sizes of data types by using **short** and **long** data modifiers.

The short Modifier

A data type can be modified to take less memory by using the short modifier.

short x;

decreases the size of bits for integer **x**. (Normally x is represented by 32 bits but after short modifier x is represented by 16 bits)

Data Modifiers : Changing Data Sizes

The long Modifier

If you need more memory to keep values from a wider range, you can use the **long** modifier to define a data type with increased storage space.

For instance :
given an integer variable **x** that is **32 bits** long, the declaration

long int x;

increases the size of **x** to at least **64 bits**.

```
long int x, y;
x = 123456789l;
y = 0xABCD1234L;
```

you can use **long constant** decleration by suffixing **l** or **L**.

Data Modifiers : Changing Data Sizes

Modifying Data with short and long

```
1:  /* 09L02.c: Using short and long modifiers */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:      printf("The size of short int is: %d.\n",
7:          sizeof(short int));
8:      printf("The size of long int is: %d.\n",
9:          sizeof(long int));
10:     printf("The size of float is: %d.\n",
11:         sizeof(float));
12:     printf("The size of double is: %d.\n",
13:         sizeof(double));
14:     printf("The size of long double is: %d.\n",
15:         sizeof(long double));
16:     return 0;
17: }
```

```
The size of short int is: 2.
The size of long int is: 4.
The size of float is: 4.
The size of double is: 8.
The size of long double is: 10.
```

Data Modifiers : Adding h, l, or L to printf

You can add **h** into the integer format specifier (like this: **%hd** or **%hu**) to specify that the corresponding argument is a **short int** or **unsigned short int.**

Using **%ld** or **%Ld** specifies that the corresponding argument is **long int**. **%lu** or **%Lu** is then used for the **long unsigned int** data.

Mathematical Functions in C: sin(), cos(), tan()

You have to include the header file **math.h** in your C program before you can use any math functions defined in the header file.

if you do need to make mathematical calculations, you can use set of math functions.

Syntax for the sin( ) function is

```
#include <math.h>
    double    sin(double   x);
```

Syntax for the cos( ) function is

```
#include <math.h>
    double    cos(double   x);
```

Syntax for the tan( ) function is

```
#include <math.h>
    double    tan(double   x);
```

Mathematical Functions in C: sin(), cos(), tan()

The double variable **x** contains the value of an angle in radians.

The following formula can be used to convert the value of an angle in degrees into the value in radians:

radians = degree * (3.141593 / 180.0).

3.141593 is the approximate value of **pi**.

Mathematical Functions in C: sin(), cos(), tan()

Calculating Trigonometric Values with sin(), cos(), and tan()

```
1:  /* 09L04.c: Using sin(), cos(), and tan() functions */
2:  #include <stdio.h>
3:  #include <math.h>
4:
5:  main()
6:  {
7:     double x;
8:
9:     x = 45.0;                    /* 45 degree */
10:    x *= 3.141593 / 180.0;       /* convert to radians */
11:    printf("The sine of 45 is:    %f.\n", sin);
12:    printf("The cosine of 45 is:  %f.\n", cos);
13:    printf("The tangent of 45 is: %f.\n", tan);
14:    return 0;
15: }
```

```
The sine of 45 is:    0.707107.
The cosine of 45 is:  0.707107.
The tangent of 45 is: 1.000000.
```

Mathematical Functions in C: pow(), sqrt()

C has no intrinsic operator for raising a number to a power.

Syntax for the pow( ) function is

```
#include <math.h>
    double   pow(double   x,   double   y);
```

The value of the double variable **x** is raised to the power of **y**.

Syntax for the sqrt( ) function is

```
#include <math.h>
    double   sqrt(double   x);
```

The **sqrt()** function returns the **non-negative square root of x** in the double datatype. An error occurs if **x** is negative.

Mathematical Functions in C: pow(), sqrt()

Applying the pow() and sqrt() Functions

```
1:  /* 09L05.c: Using pow() and sqrt() functions */
2:  #include <stdio.h>
3:  #include <math.h>
4:
5:  main()
6:  {
7:     double x, y, z;
8:
9:     x = 64.0;
10:    y = 3.0;
11:    z = 0.5;
12:    printf("pow(64.0, 3.0) returns: %7.0f\n", pow(x, y));
13:    printf("sqrt(64.0) returns:     %2.0f\n", sqrt);
14:    printf("pow(64.0, 0.5) returns: %2.0f\n", pow(x, z));
15:    return 0;
16: }
```

```
pow(64.0, 3.0) returns: 262144
sqrt(64.0) returns:     8
pow(64.0, 0.5) returns: 8
```