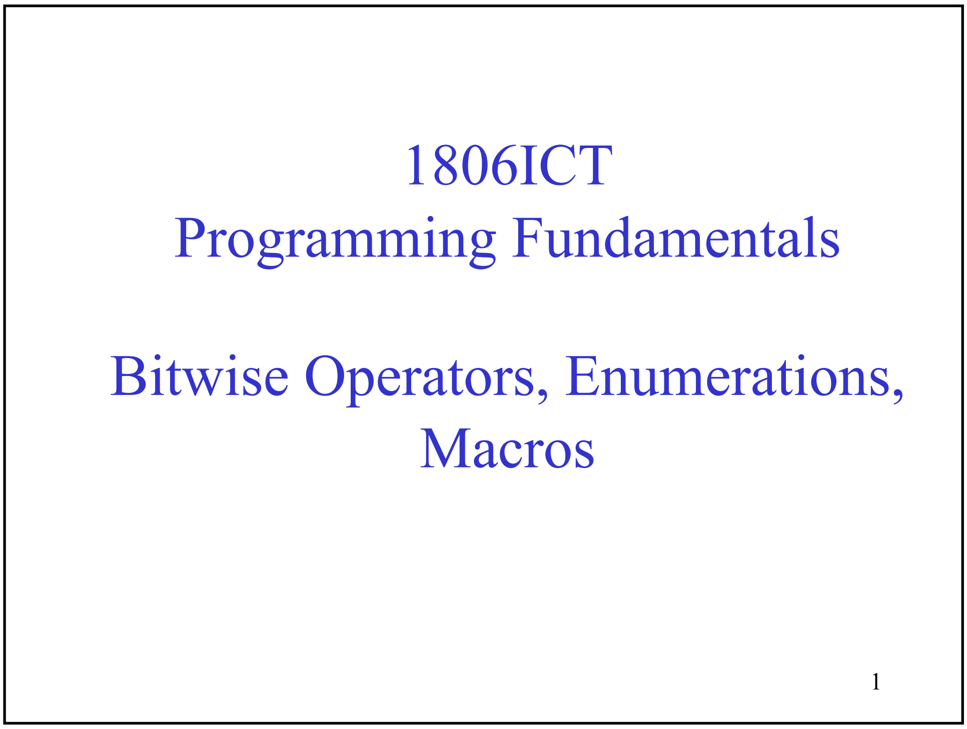
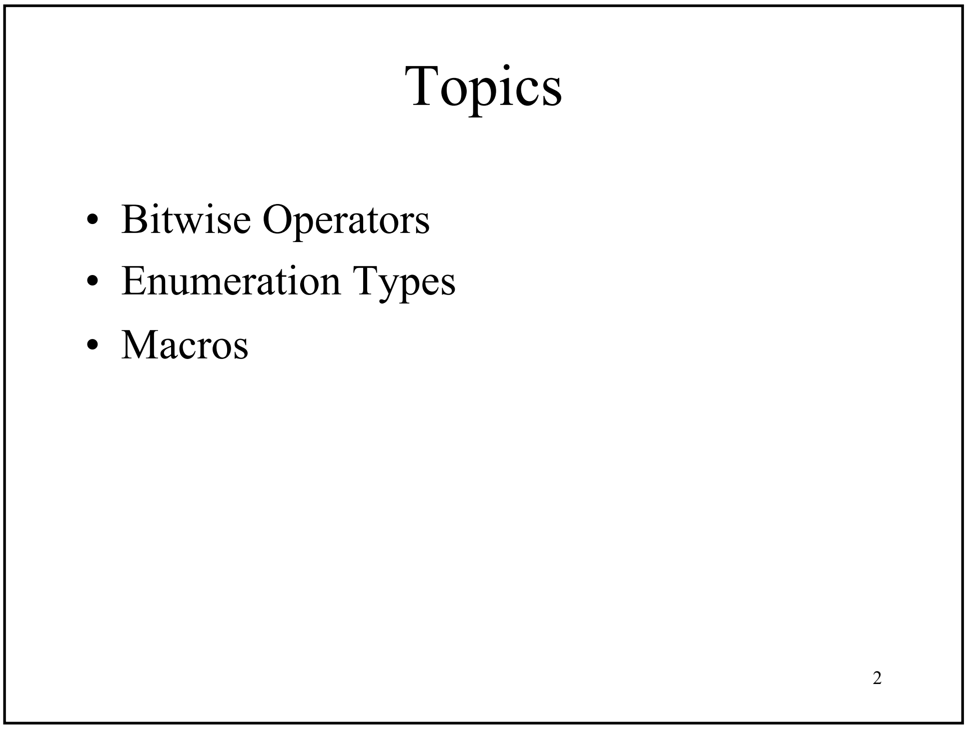


1806ICT
Programming Fundamentals

Bitwise Operators, Enumerations, Macros

1

1

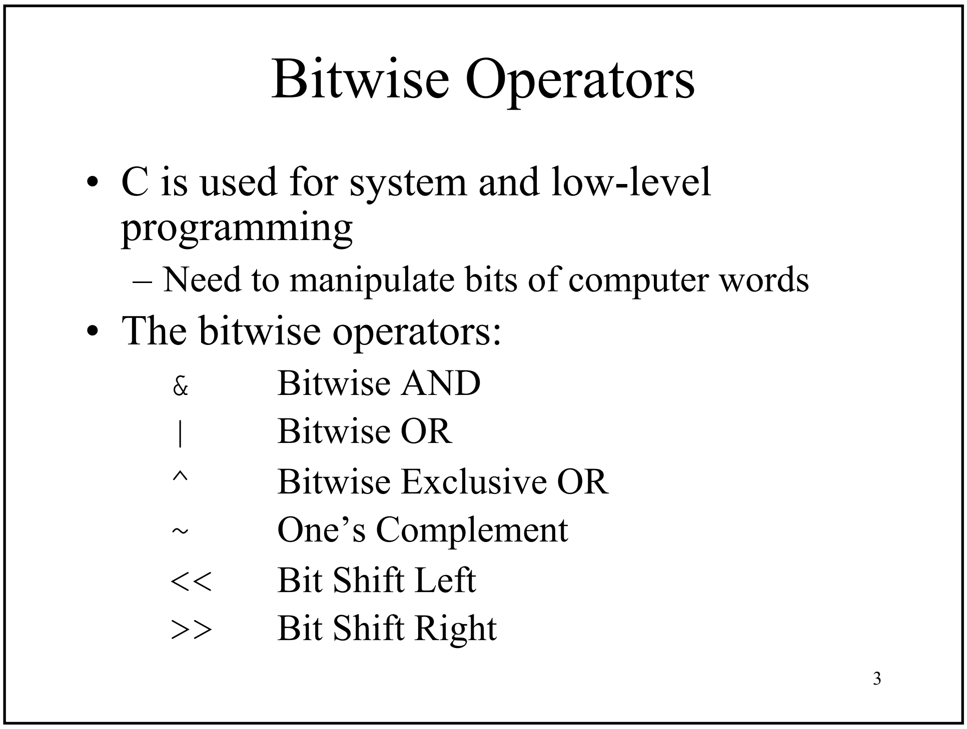


Topics

- Bitwise Operators
- Enumeration Types
- Macros

2

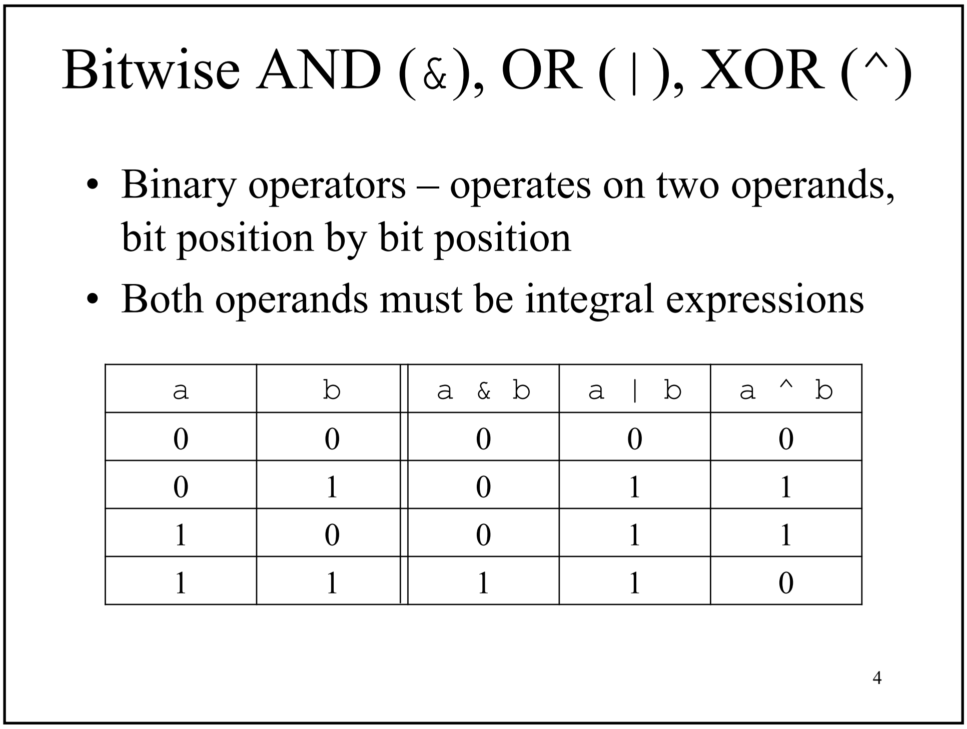
2



Bitwise Operators

- C is used for system and low-level programming
  - – Need to manipulate bits of computer words
- The bitwise operators:

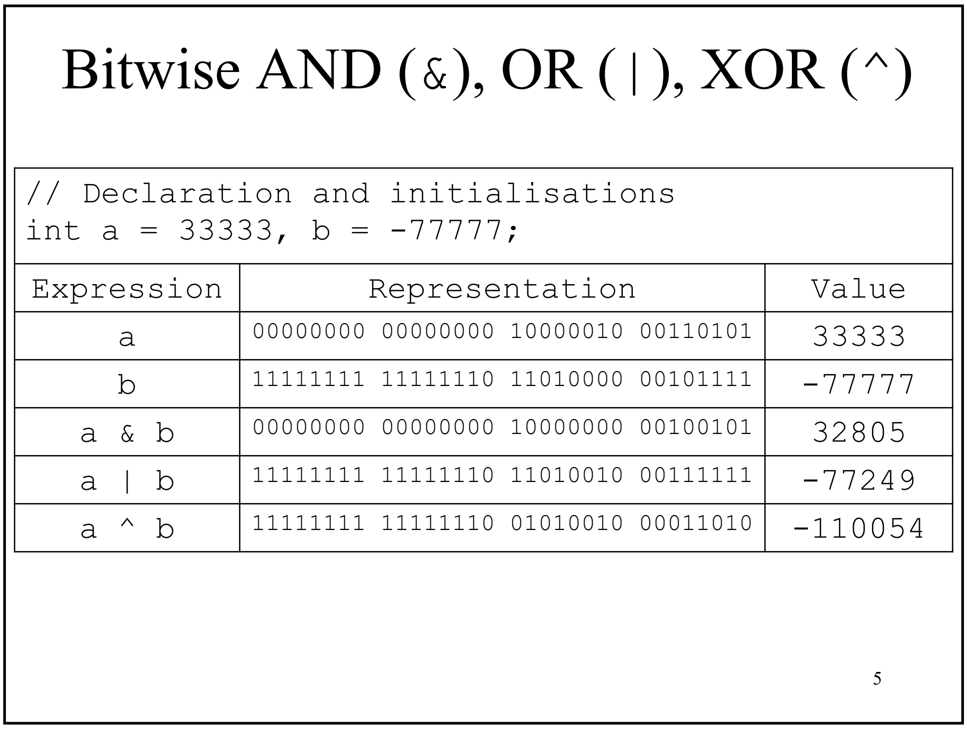
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
~	One’s Complement
<<	Bit Shift Left
>>	Bit Shift Right



Bitwise AND (&), OR (|), XOR (^)

- Binary operators – operates on two operands, bit position by bit position
- Both operands must be integral expressions

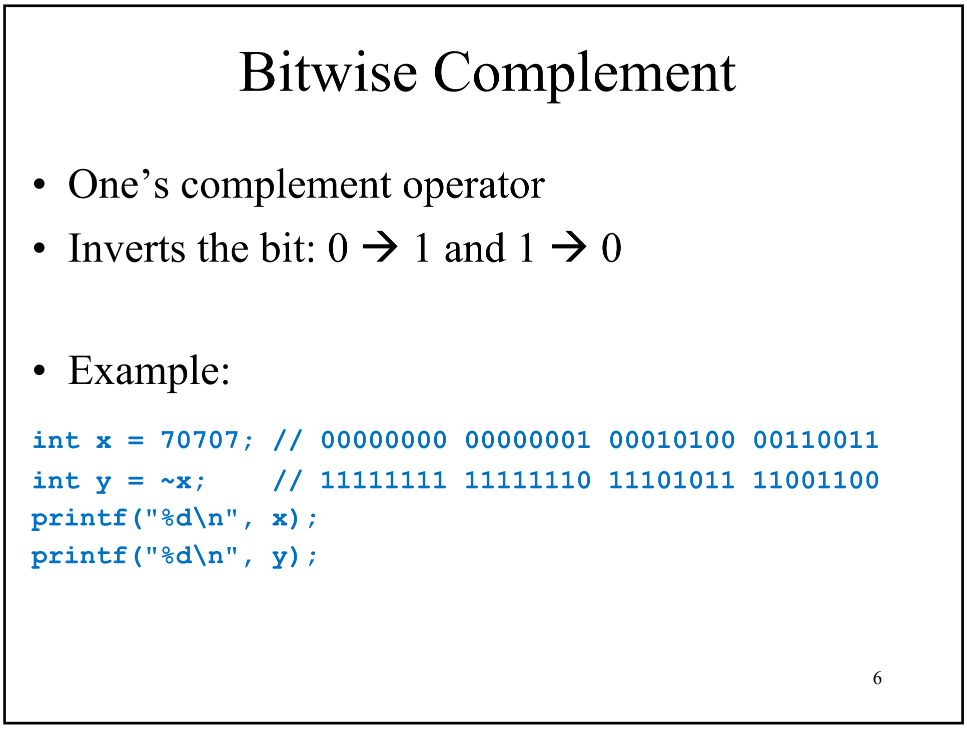
a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



Bitwise AND (&), OR (|), XOR (^)

```
// Declaration and initialisations
int a = 33333, b = -77777;
```

Expression	Representation	Value
a	00000000 00000000 10000010 00110101	33333
b	11111111 11111110 11010000 00101111	-77777
a & b	00000000 00000000 10000000 00100101	32805
a | b	11111111 11111110 11010010 00111111	-77249
a ^ b	11111111 11111110 01010010 00011010	-110054

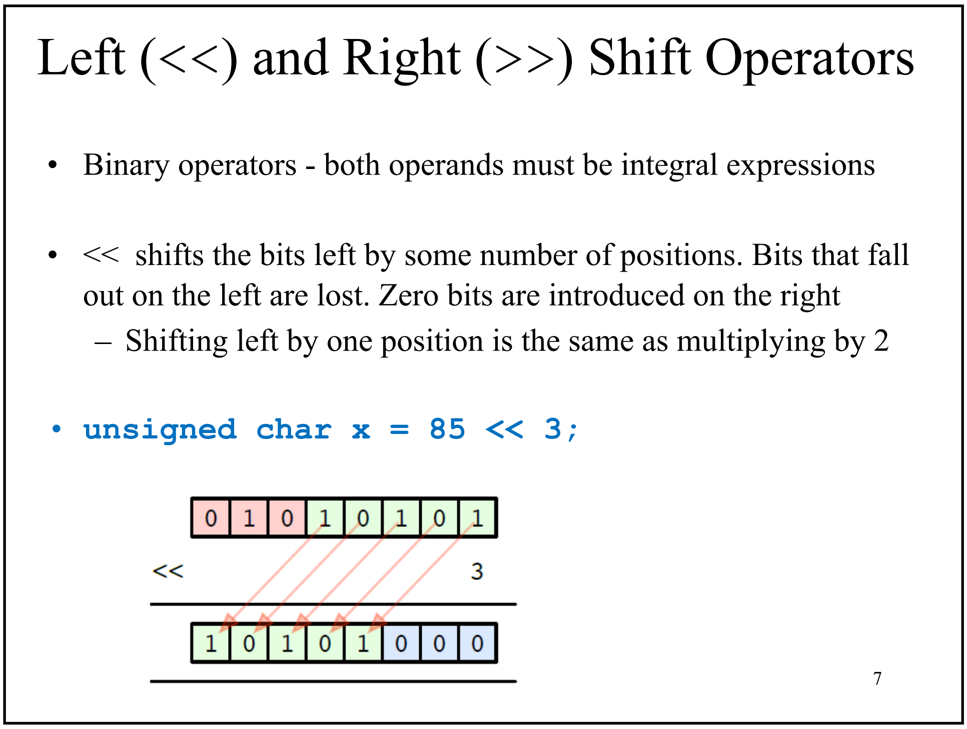


Bitwise Complement

- One’s complement operator
- Inverts the bit: 0 → 1 and 1 → 0

- Example:

```
int x = 70707; // 00000000 00000001 00010100 00110011
int y = ~x;    // 11111111 11111110 11101011 11001100
printf("%d\n", x);
printf("%d\n", y);
```



Left (<<) and Right (>>) Shift Operators

- Binary operators - both operands must be integral expressions

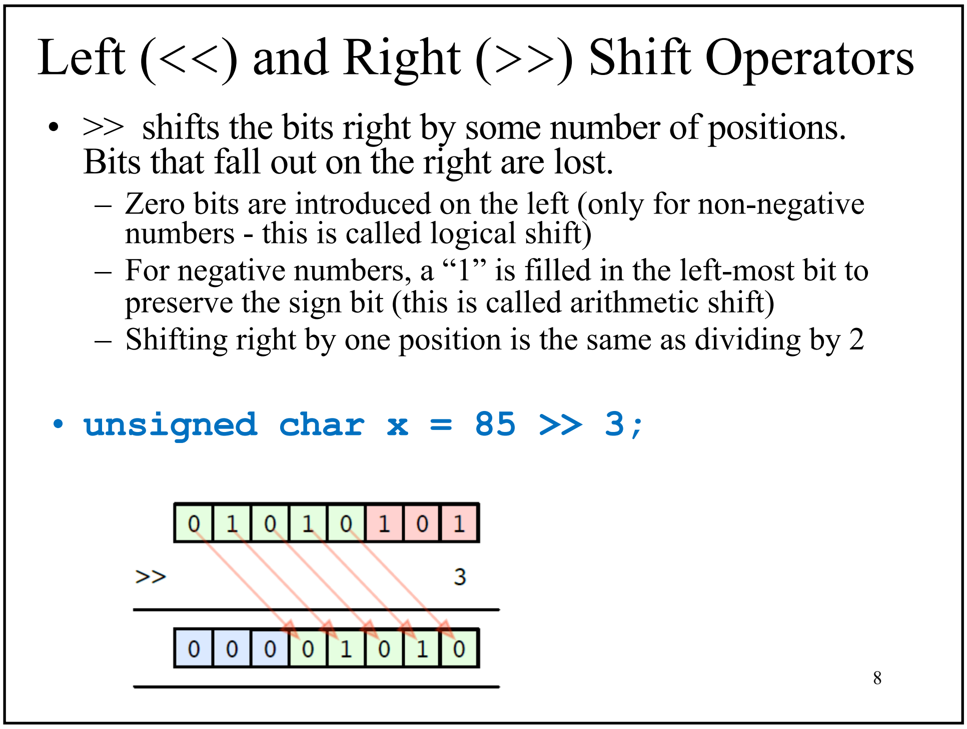
- << shifts the bits left by some number of positions. Bits that fall out on the left are lost. Zero bits are introduced on the right
  - Shifting left by one position is the same as multiplying by 2

- **`unsigned char x = 85 << 3;`**

```
  0 1 0 1 0 1 0 1
<<              3
  1 0 1 0 1 0 0 0
```

7

7



Left (<<) and Right (>>) Shift Operators

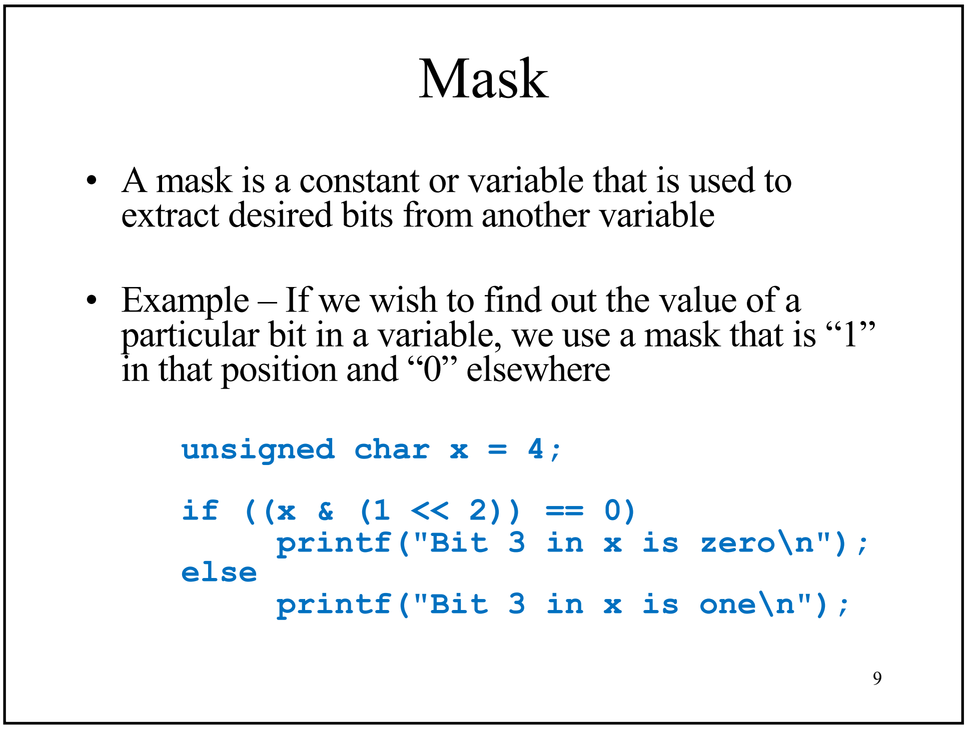
- >> shifts the bits right by some number of positions. Bits that fall out on the right are lost.
  - Zero bits are introduced on the left (only for non-negative numbers - this is called logical shift)
  - For negative numbers, a “1” is filled in the left-most bit to preserve the sign bit (this is called arithmetic shift)
  - Shifting right by one position is the same as dividing by 2

- **`unsigned char x = 85 >> 3;`**

```
  0 1 0 1 0 1 0 1
>>              3
  0 0 0 0 1 0 1 0
```

8

8



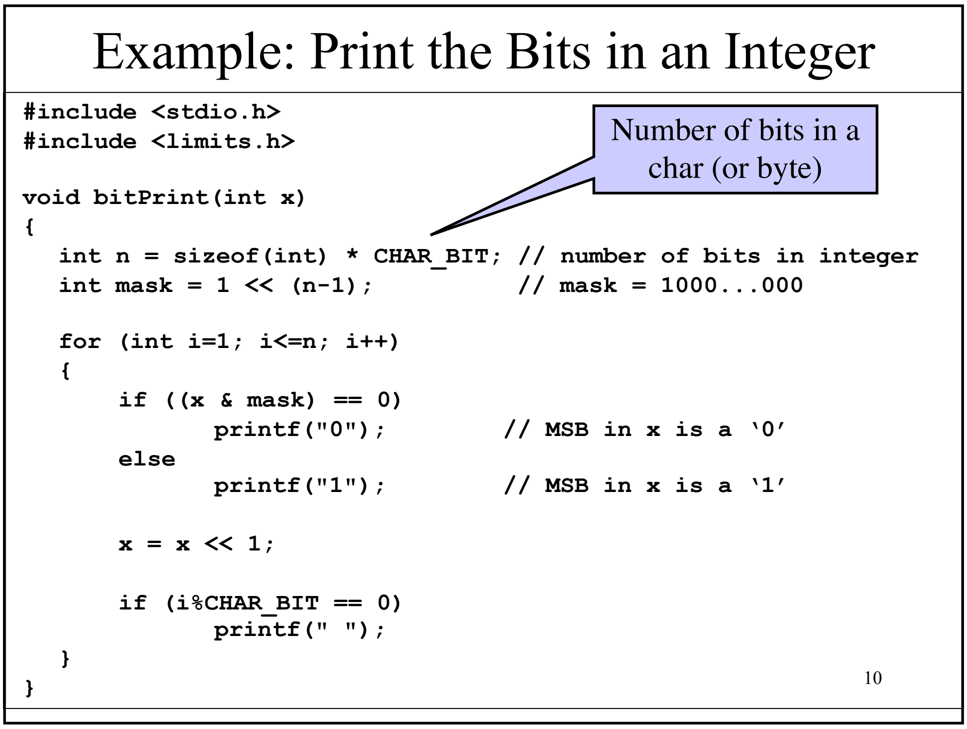
Mask

- A mask is a constant or variable that is used to extract desired bits from another variable

- Example – If we wish to find out the value of a particular bit in a variable, we use a mask that is “1” in that position and “0” elsewhere

```
unsigned char x = 4;

if ((x & (1 << 2)) == 0)
     printf("Bit 3 in x is zero\n");
else
     printf("Bit 3 in x is one\n");
```

9

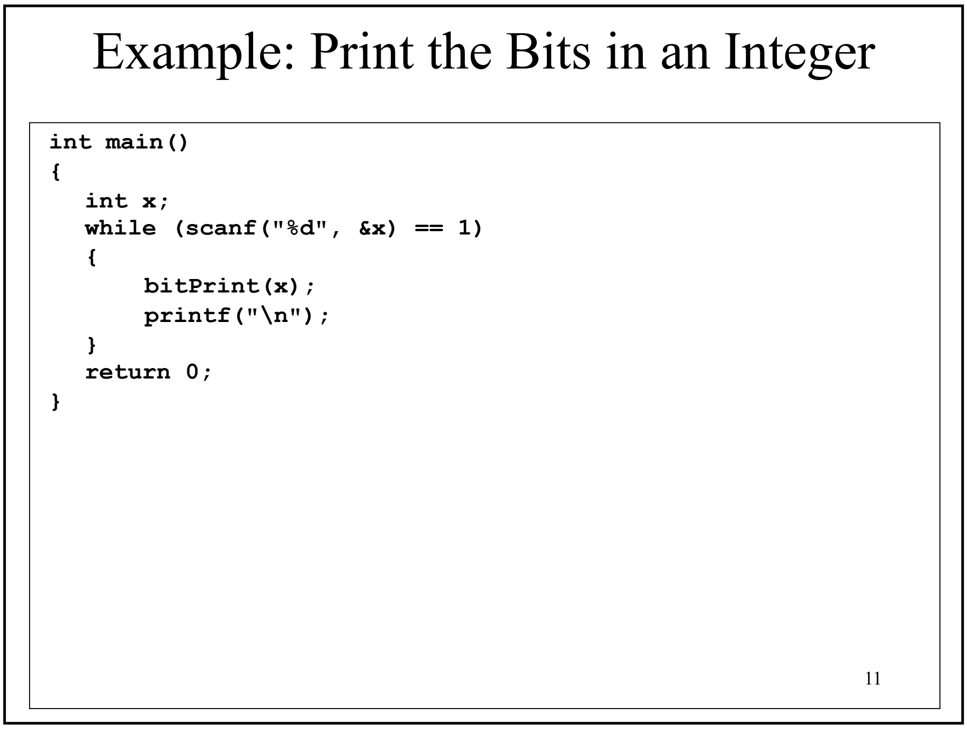


Example: Print the Bits in an Integer

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

void bitPrint(int x)
{
   int n = sizeof(int) * CHAR_BIT; // number of bits in integer
   int mask = 1 << (n-1);          // mask = 1000...000

   for (int i=1; i<=n; i++)
   {
        if ((x & mask) == 0)
             printf("0");        // MSB in x is a ‘0’
        else
             printf("1");        // MSB in x is a ‘1’

        x = x << 1;

        if (i%CHAR_BIT == 0)
             printf(" ");
   }
}
```

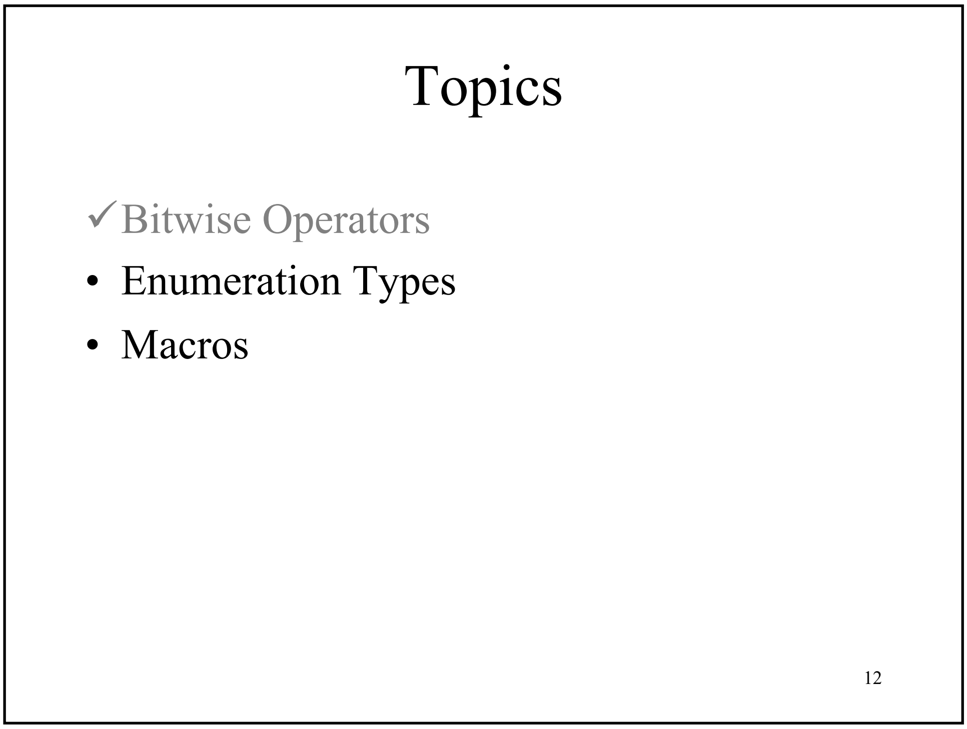
Number of bits in a char (or byte)

10



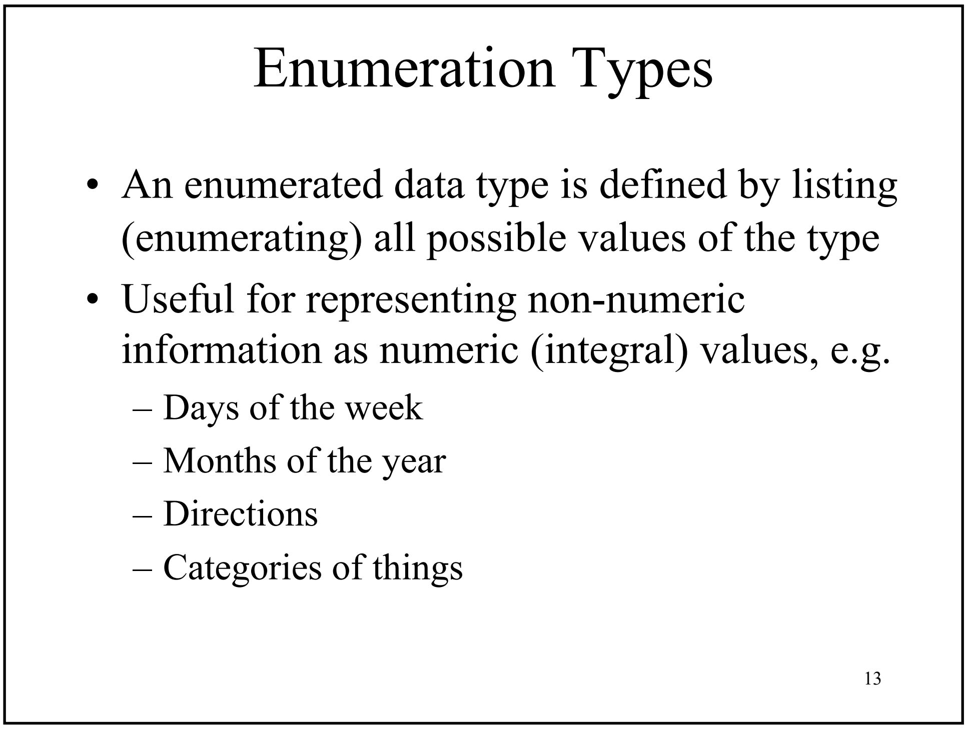
Example: Print the Bits in an Integer

```
int main()
{
   int x;
   while (scanf("%d", &x) == 1)
   {
        bitPrint(x);
        printf("\n");
   }
   return 0;
}
```



Topics

- ✓ Bitwise Operators
- Enumeration Types
- Macros

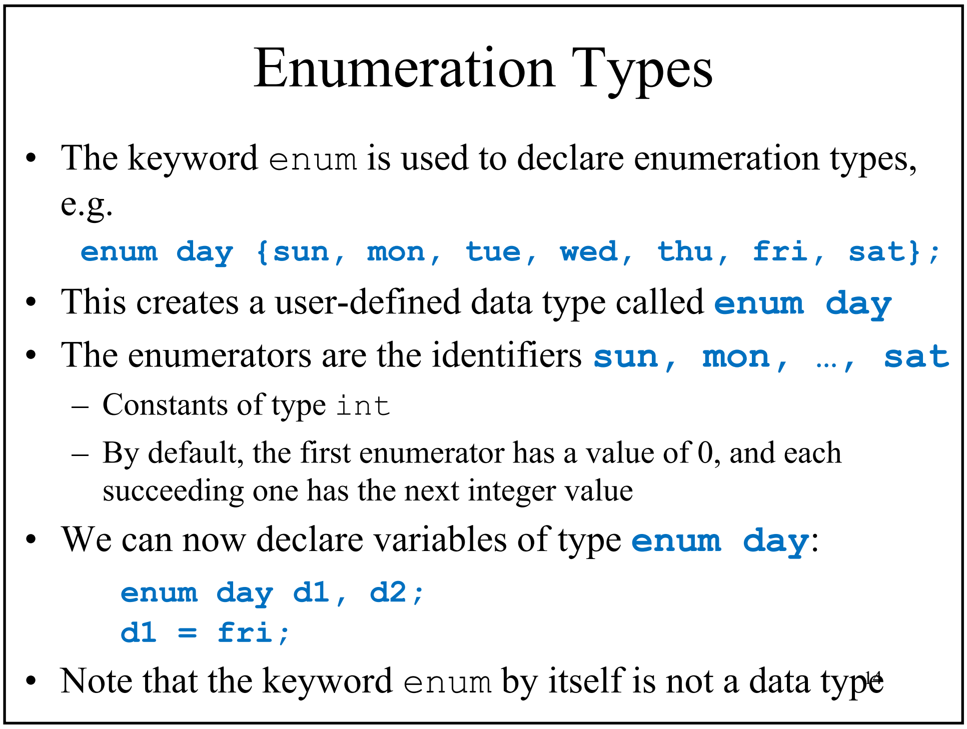


Enumeration Types

- An enumerated data type is defined by listing (enumerating) all possible values of the type
- Useful for representing non-numeric information as numeric (integral) values, e.g.
  - Days of the week
  - Months of the year
  - Directions
  - Categories of things

13

13



Enumeration Types

- The keyword `enum` is used to declare enumeration types, e.g.

```
enum day {sun, mon, tue, wed, thu, fri, sat};
```

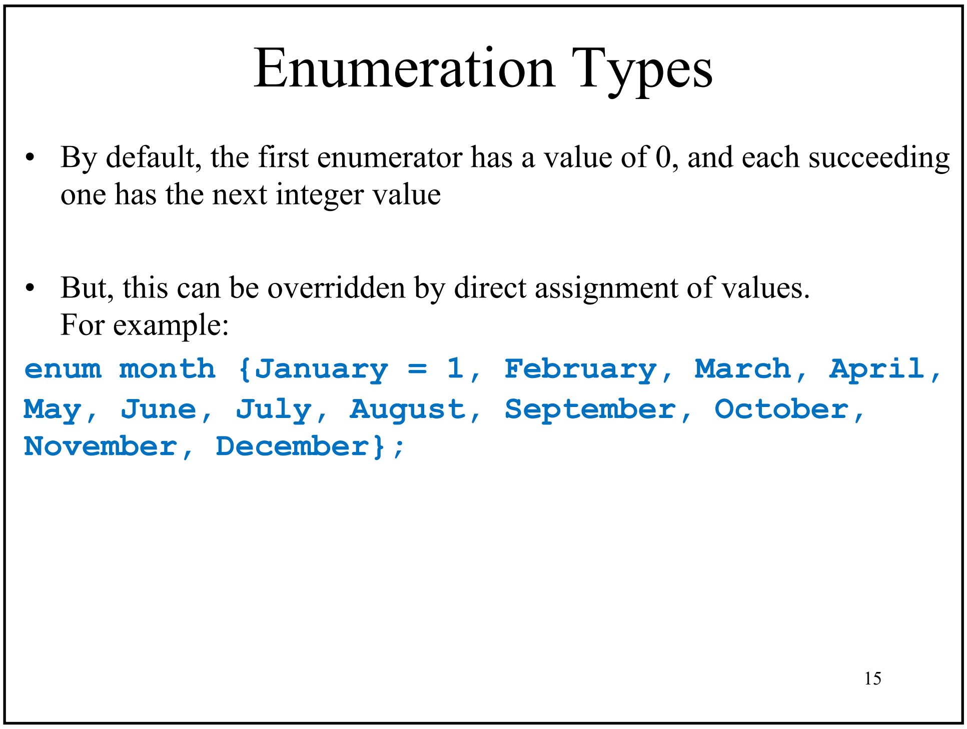
- This creates a user-defined data type called **`enum day`**
- The enumerators are the identifiers **`sun, mon, …, sat`**
  - Constants of type `int`
  - By default, the first enumerator has a value of 0, and each succeeding one has the next integer value
- We can now declare variables of type **`enum day`**:

```
enum day d1, d2;
d1 = fri;
```

- Note that the keyword `enum` by itself is not a data type

14

14



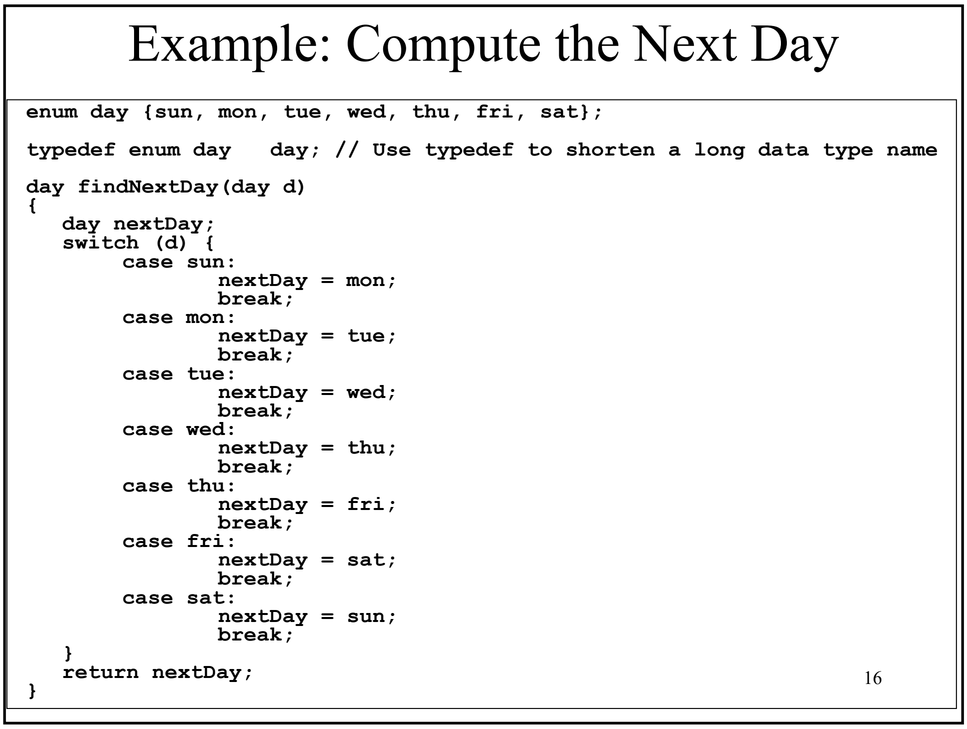
Enumeration Types

- By default, the first enumerator has a value of 0, and each succeeding one has the next integer value
- But, this can be overridden by direct assignment of values.
  For example:

```
enum month {January = 1, February, March, April,
May, June, July, August, September, October,
November, December};
```

15

15

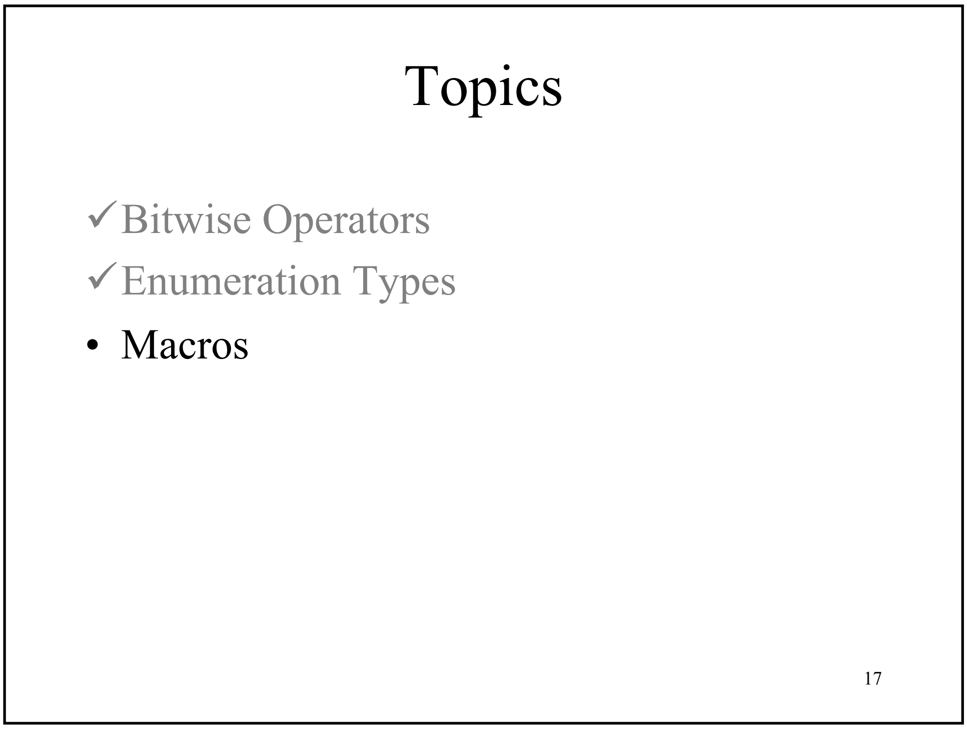


Example: Compute the Next Day

```
enum day {sun, mon, tue, wed, thu, fri, sat};

typedef enum day   day; // Use typedef to shorten a long data type name

day findNextDay(day d)
{
   day nextDay;
   switch (d) {
        case sun:
                nextDay = mon;
                break;
        case mon:
                nextDay = tue;
                break;
        case tue:
                nextDay = wed;
                break;
        case wed:
                nextDay = thu;
                break;
        case thu:
                nextDay = fri;
                break;
        case fri:
                nextDay = sat;
                break;
        case sat:
                nextDay = sun;
                break;
   }
   return nextDay;
}
```

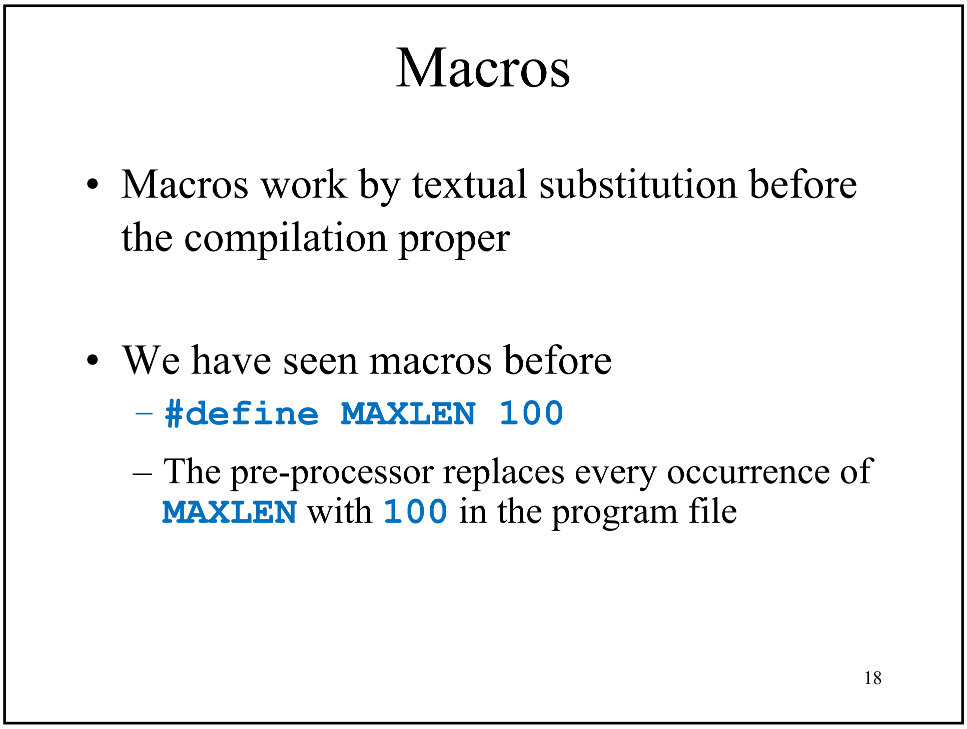
16

16



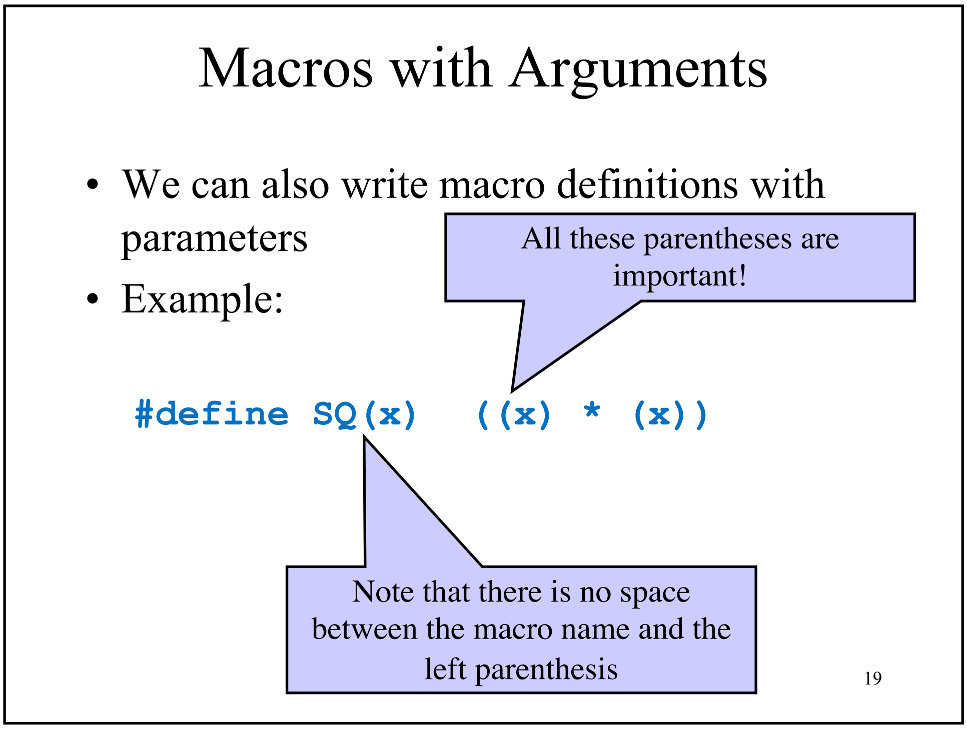
Topics

✓Bitwise Operators
✓Enumeration Types
- Macros



Macros

- Macros work by textual substitution before the compilation proper
- We have seen macros before
  - `#define MAXLEN 100`
  - The pre-processor replaces every occurrence of `MAXLEN` with `100` in the program file



Macros with Arguments

- We can also write macro definitions with parameters
- Example:

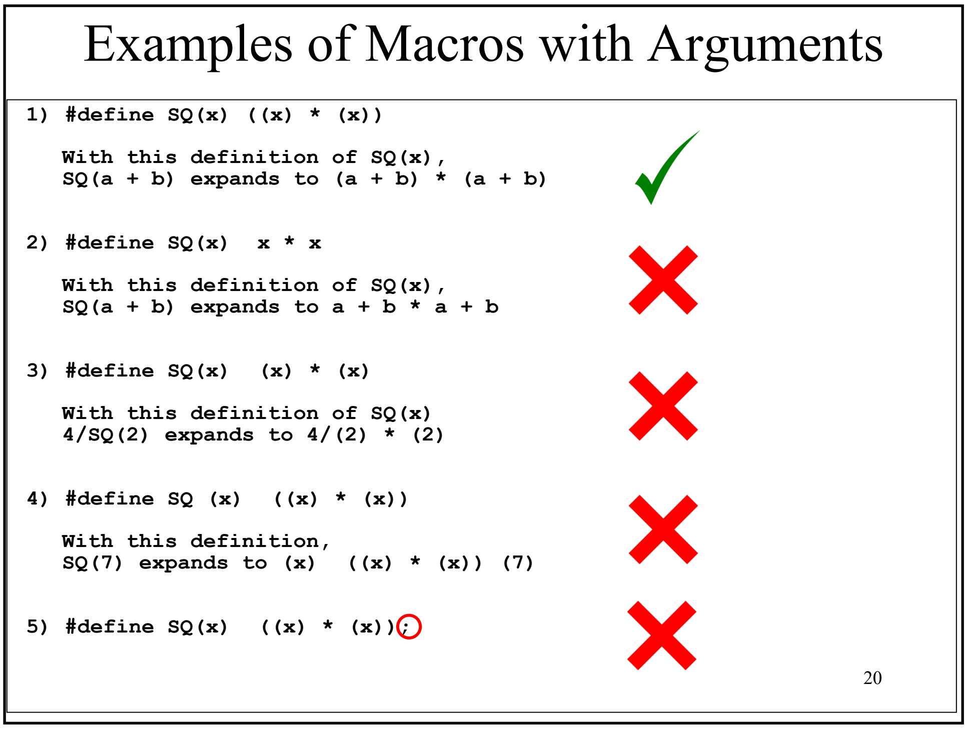
```
#define SQ(x)   ((x) * (x))
```

All these parentheses are important!

Note that there is no space between the macro name and the left parenthesis

19

19



Examples of Macros with Arguments

```
1) #define SQ(x) ((x) * (x))

   With this definition of SQ(x),
   SQ(a + b) expands to (a + b) * (a + b)
```

✓

```
2) #define SQ(x)  x * x

   With this definition of SQ(x),
   SQ(a + b) expands to a + b * a + b
```

✗

```
3) #define SQ(x)  (x) * (x)

   With this definition of SQ(x)
   4/SQ(2) expands to 4/(2) * (2)
```

✗

```
4) #define SQ (x)  ((x) * (x))

   With this definition,
   SQ(7) expands to (x)  ((x) * (x)) (7)
```

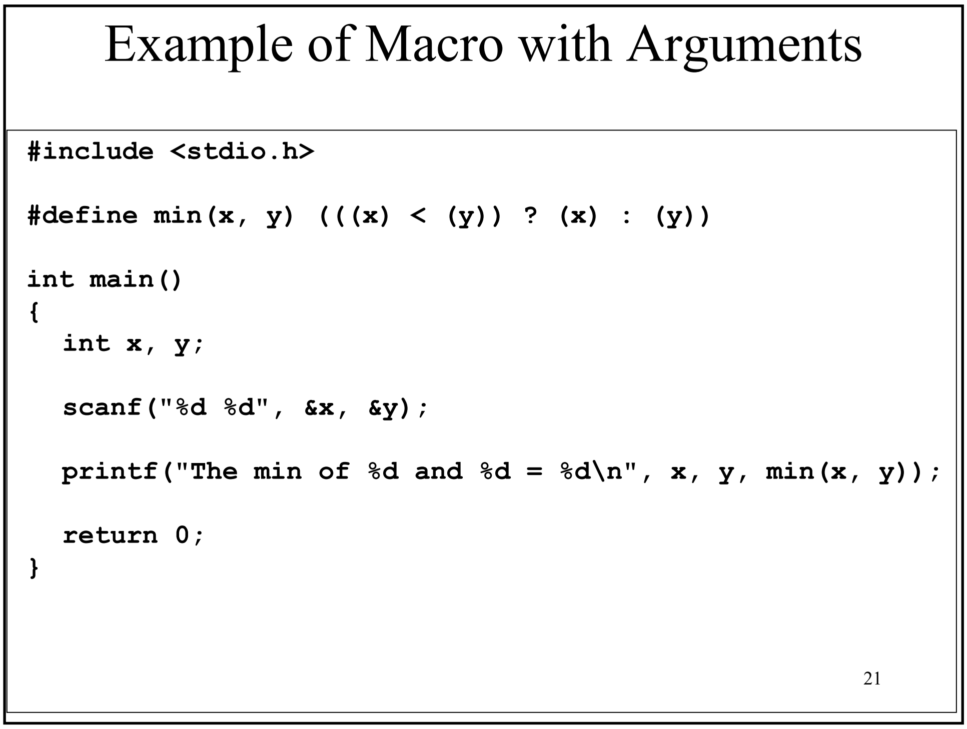
✗

```
5) #define SQ(x)  ((x) * (x));
```

✗

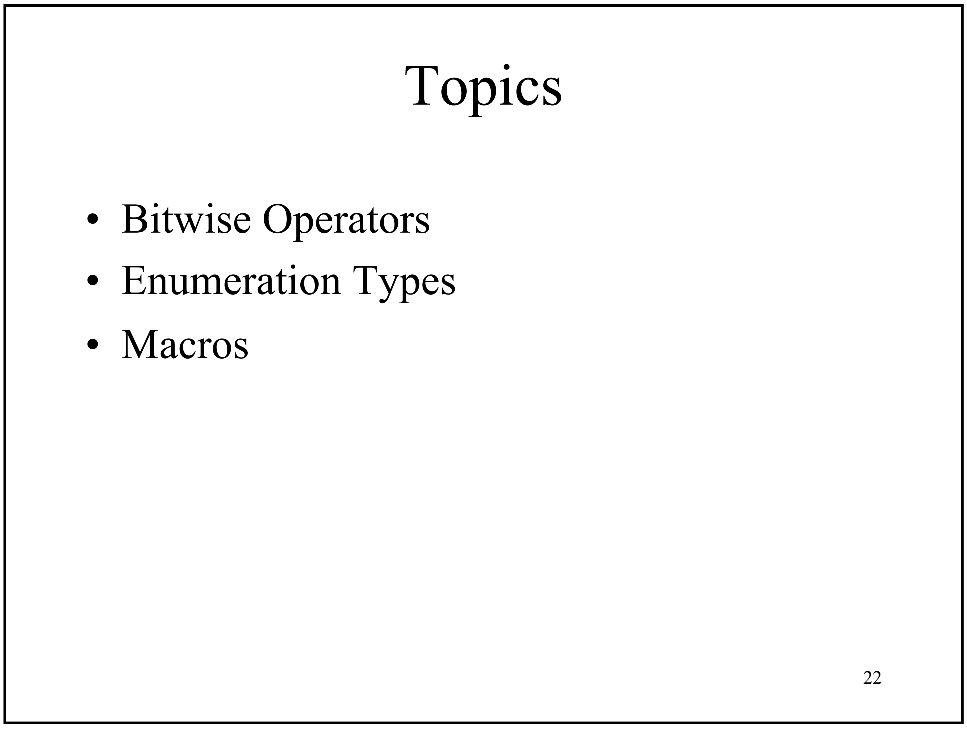
20

20



Example of Macro with Arguments

```
#include <stdio.h>

#define min(x, y) (((x) < (y)) ? (x) : (y))

int main()
{
  int x, y;

  scanf("%d %d", &x, &y);

  printf("The min of %d and %d = %d\n", x, y, min(x, y));

  return 0;
}
```



Topics

- Bitwise Operators
- Enumeration Types
- Macros