PEN203

Structures and Bit Manipulations

C++ How to Program
Deitel & Deitel

Outline

- Structure Definitions
- Initializing Structures
- Accessing Members of Structures
- Using Structures with Functions
- Bitwise Operators

Structure Definitions

- Structures are collections of related variables under one name.
  - Unlike arrays, structures can contain variables of different data types.
- Structures are used to create linked lists, stacks, queues and trees.

Structure Definitions

- Example:

```
struct student {
        int id;
        float gpa;
};
```

- Keyword struct is used to define structure student
- student contains two members one integer id, the other is float gpa.

Structure Definitions

- Structures can not contain an instance of itself.
- However they can include a pointer to the same structure type.
  - Self referential structures
- Structure variable definitions:
  - student s1, s[50], *sptr; or

```
struct student {
        int id;
        float gpa;
} s1, s[50], *sptr;
```

Structure Definitions

- The following operations can be performed on structure variables:
  - Assigning a structure variable to another structure variable of the same type
  - Taking address of a structure
  - Accessing the members of a structure
  - Using sizeof operator to find the size of a structure

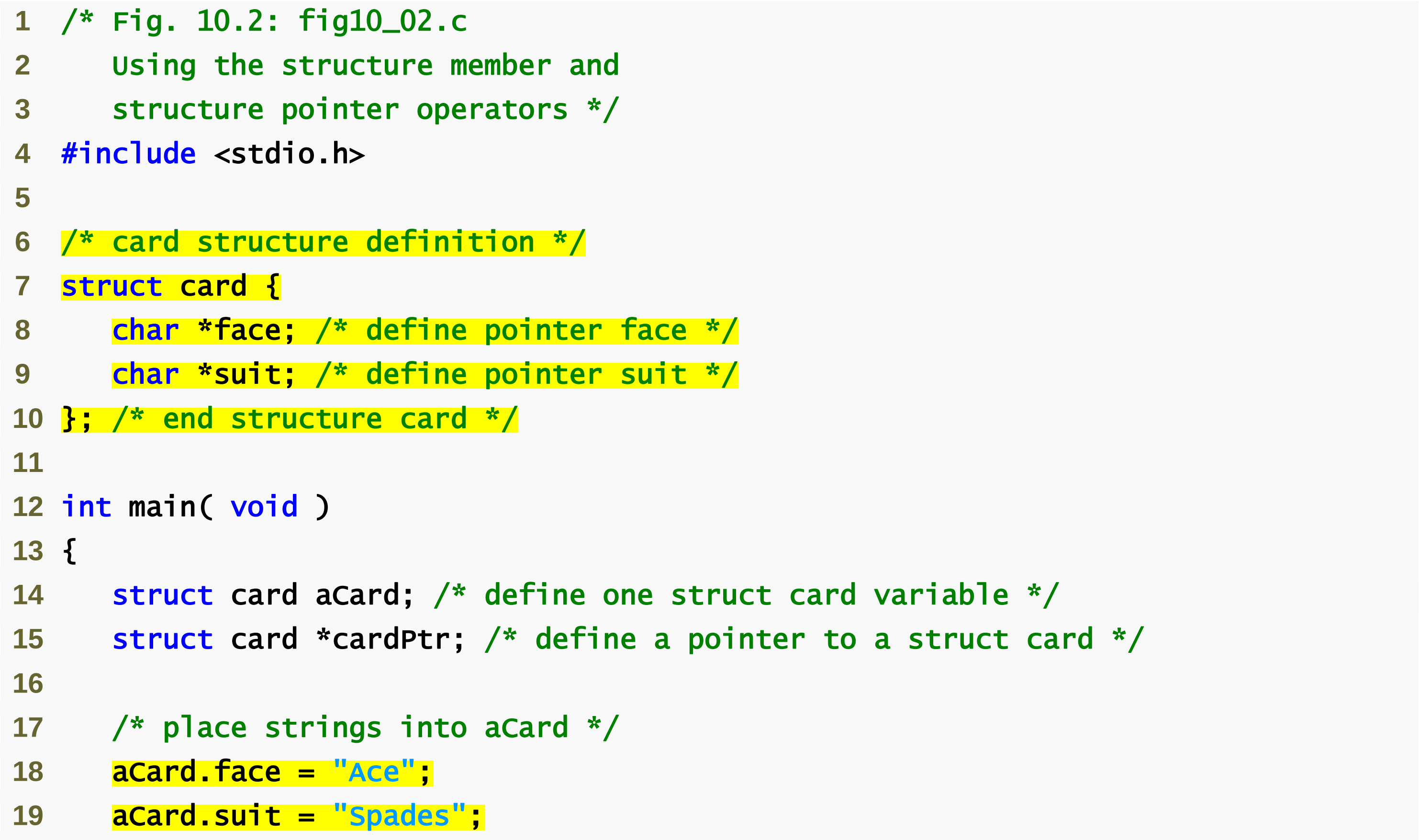
Initializing Structures

- Initializer List
  - student s1 = {123, 3.50};
- Assignment statements
  - Student s2=s1;
- student s3;
  - s3.id=435;
  - s3.gpa=2.50;

Accessing Members of Structures

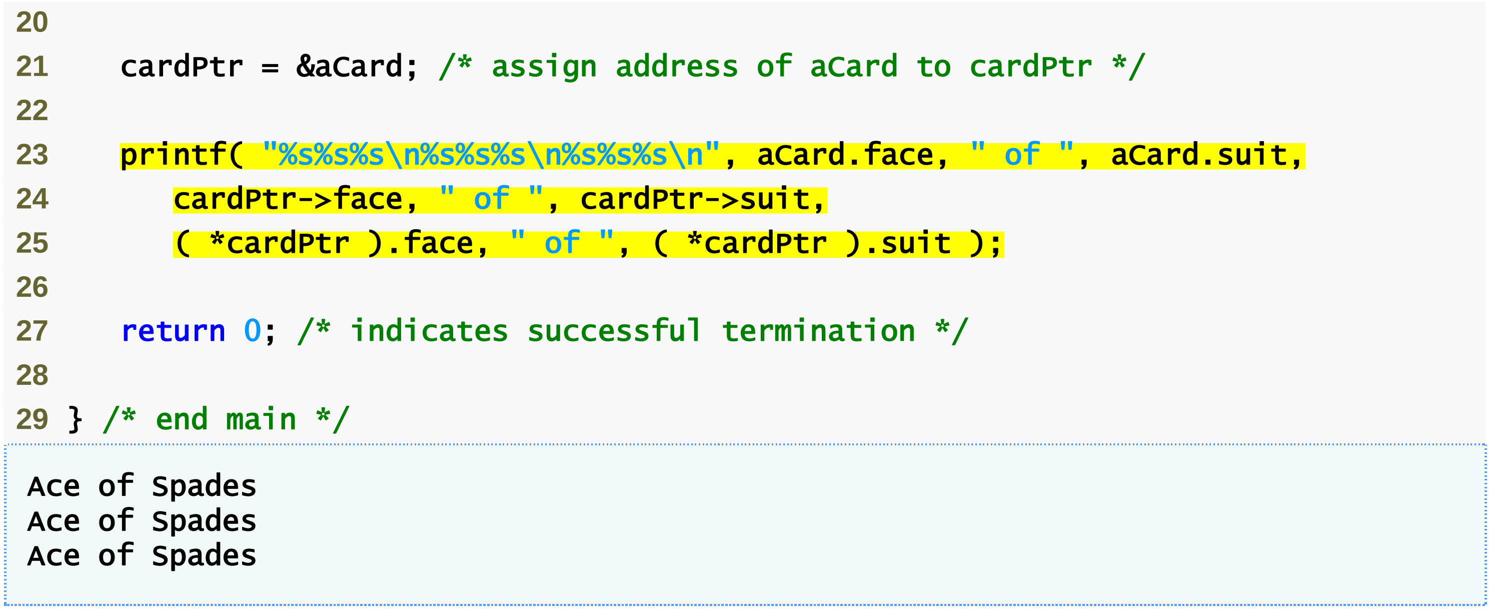
- (.) dot opeartor is used with structure variables
  - cout<<s1.id;
- (->) arrow operator is used with pointers to structure variables
  - cout<<sptr->id;
  - or cout<<(*sptr).id;

Accessing Members of Structures



```c
/* Fig. 10.2: fig10_02.c
   Using the structure member and
   structure pointer operators */
#include <stdio.h>

/* card structure definition */
struct card {
   char *face; /* define pointer face */
   char *suit; /* define pointer suit */
}; /* end structure card */

int main( void )
{
   struct card aCard; /* define one struct card variable */
   struct card *cardPtr; /* define a pointer to a struct card */

   /* place strings into aCard */
   aCard.face = "Ace";
   aCard.suit = "Spades";
```

Accessing Members of Structures



```
   cardPtr = &aCard; /* assign address of aCard to cardPtr */

   printf( "%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
      cardPtr->face, " of ", cardPtr->suit,
      ( *cardPtr ).face, " of ", ( *cardPtr ).suit );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Ace of Spades
Ace of Spades
Ace of Spades
```

Using Structures with Functions

- Passing entire structure or passing individiual members (call by value)
- To pass structures call by reference
  - Pass its address

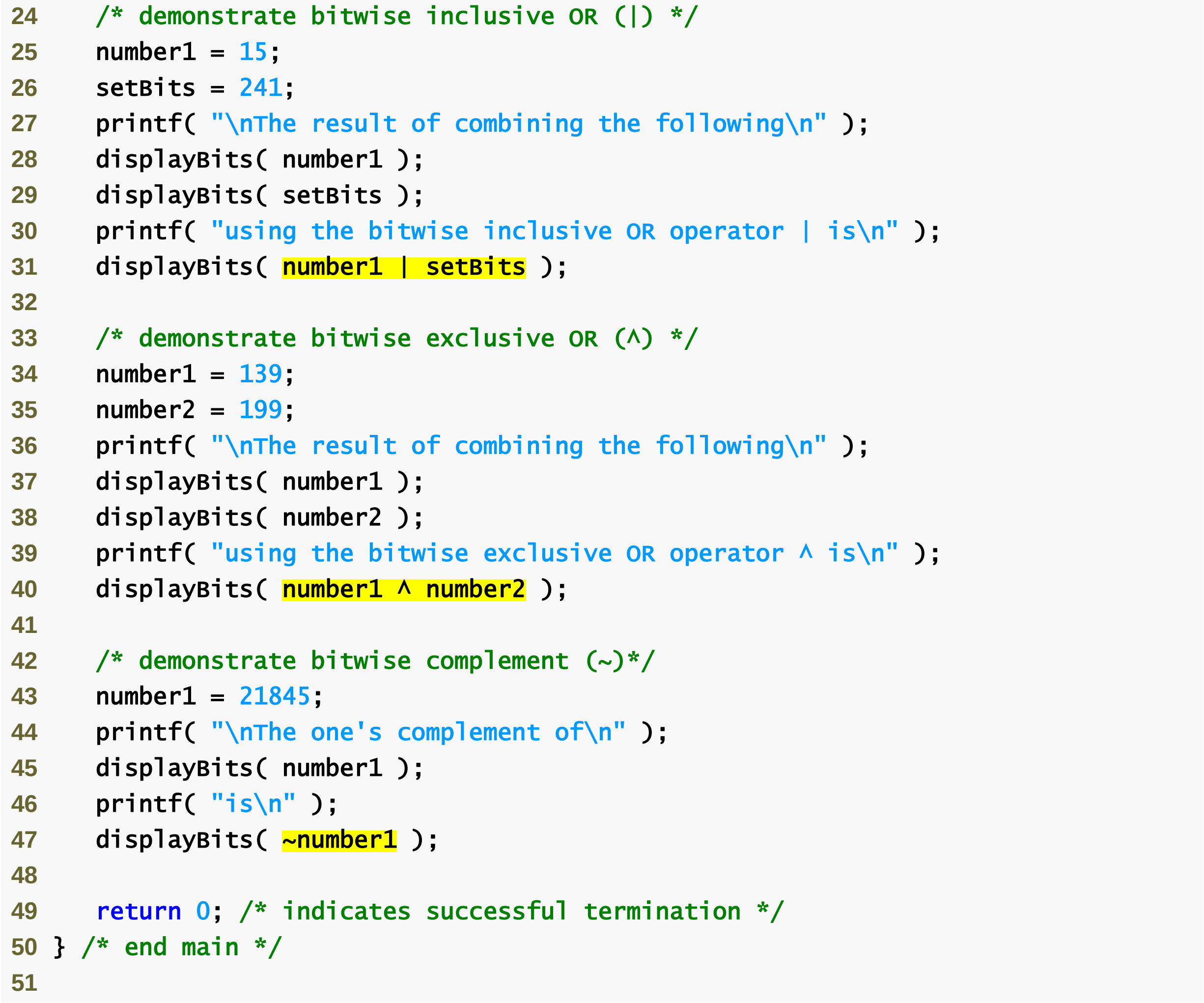
Bitwise Operators

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1 .
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	one’s complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise Operators

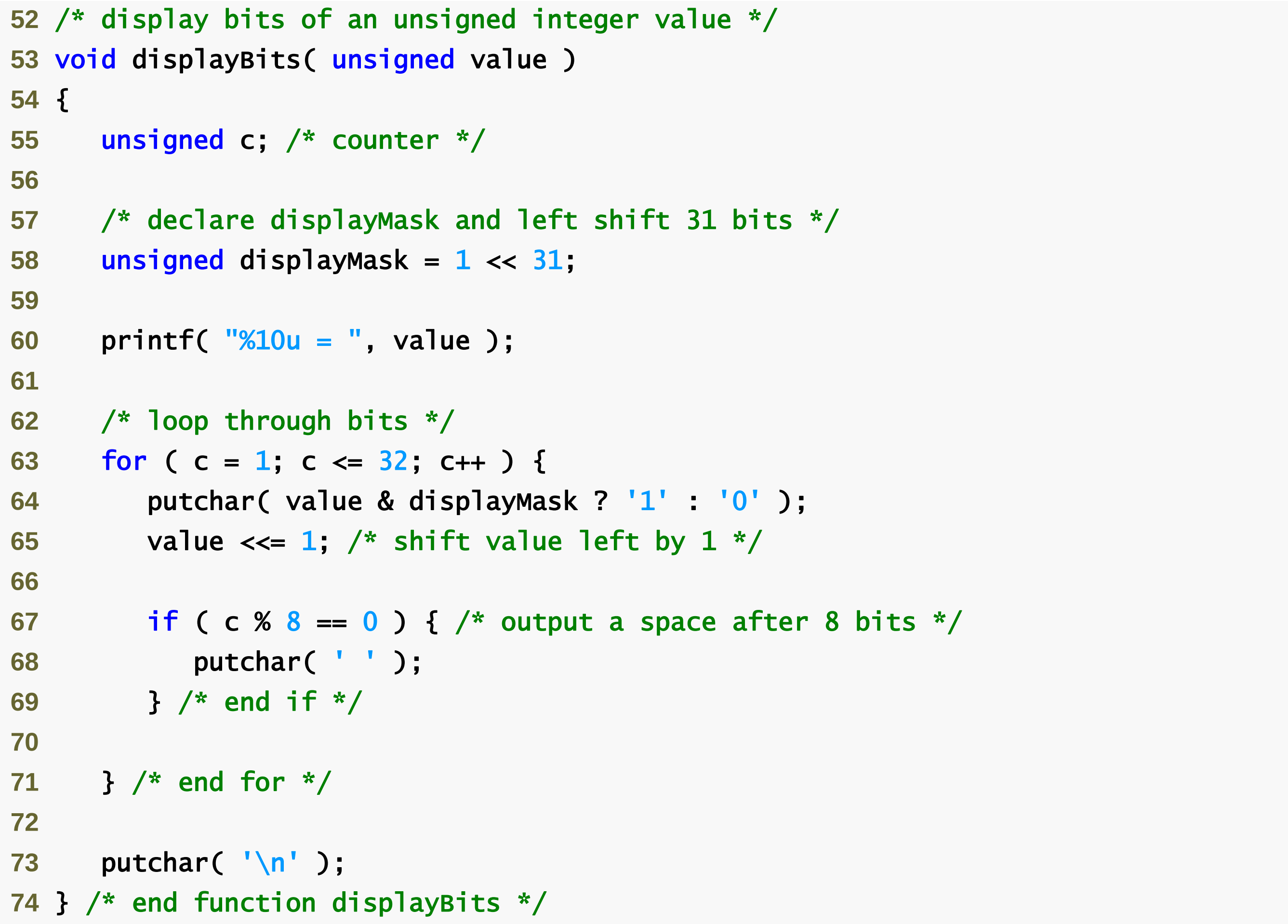
```
/* Fig. 10.9: fig10_09.c
   Using the bitwise AND, bitwise inclusive OR, bitwise
   exclusive OR and bitwise complement operators */
#include <stdio.h>

void displayBits( unsigned value ); /* prototype */

int main( void )
{
   unsigned number1;
   unsigned number2;
   unsigned mask;
   unsigned setBits;

   /* demonstrate bitwise AND (&) */
   number1 = 65535;
   mask = 1;
   printf( "The result of combining the following\n" );
   displayBits( number1 );
   displayBits( mask );
   printf( "using the bitwise AND operator & is\n" );
   displayBits( number1 & mask );

```

Bitwise Operators



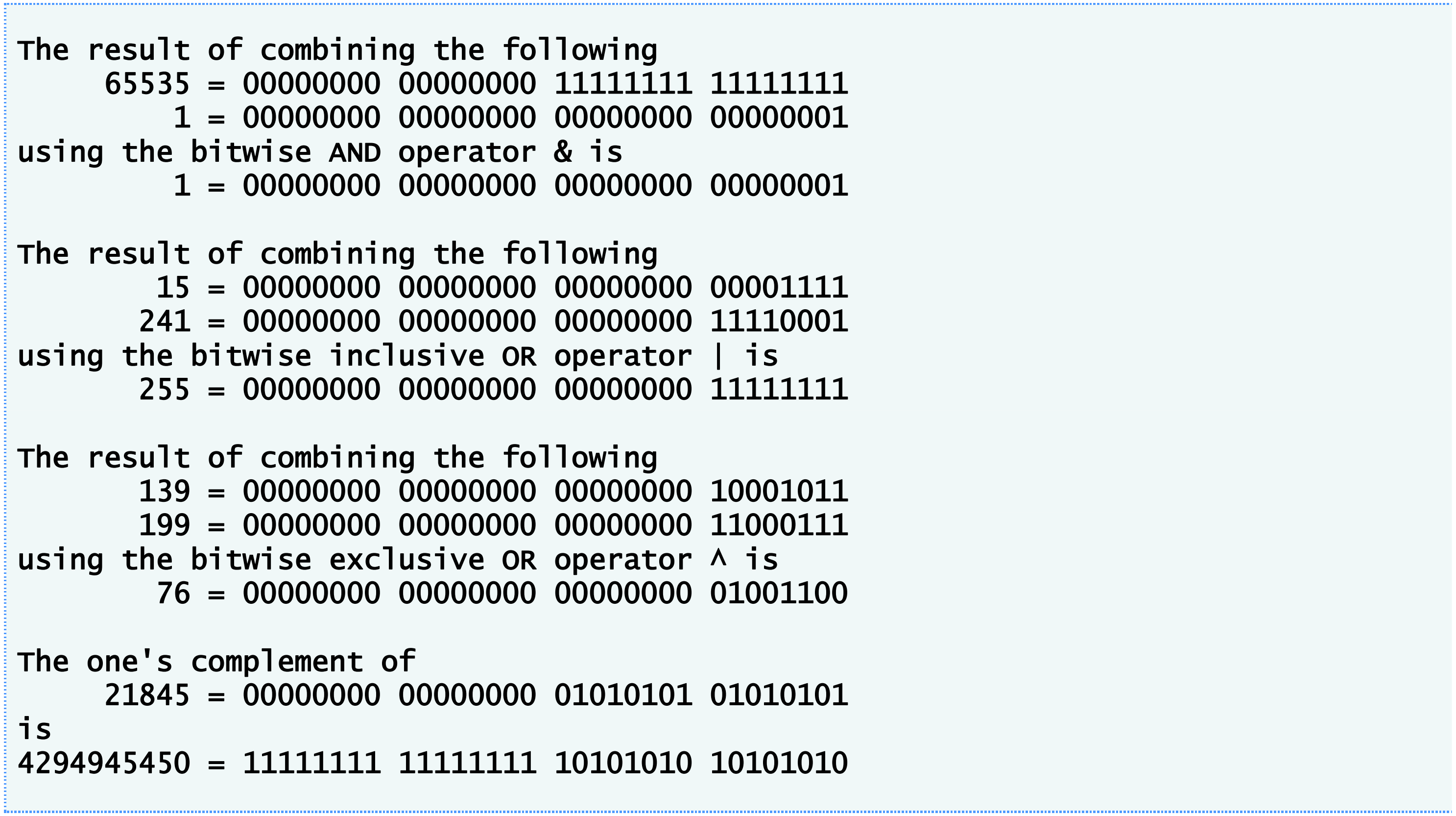
```
   /* demonstrate bitwise inclusive OR (|) */
   number1 = 15;
   setBits = 241;
   printf( "\nThe result of combining the following\n" );
   displayBits( number1 );
   displayBits( setBits );
   printf( "using the bitwise inclusive OR operator | is\n" );
   displayBits( number1 | setBits );

   /* demonstrate bitwise exclusive OR (^) */
   number1 = 139;
   number2 = 199;
   printf( "\nThe result of combining the following\n" );
   displayBits( number1 );
   displayBits( number2 );
   printf( "using the bitwise exclusive OR operator ^ is\n" );
   displayBits( number1 ^ number2 );

   /* demonstrate bitwise complement (~)*/
   number1 = 21845;
   printf( "\nThe one's complement of\n" );
   displayBits( number1 );
   printf( "is\n" );
   displayBits( ~number1 );

   return 0; /* indicates successful termination */
} /* end main */

```

Bitwise Operators



```c
/* display bits of an unsigned integer value */
void displayBits( unsigned value )
{
   unsigned c; /* counter */

   /* declare displayMask and left shift 31 bits */
   unsigned displayMask = 1 << 31;

   printf( "%10u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 32; c++ ) {
      putchar( value & displayMask ? '1' : '0' );
      value <<= 1; /* shift value left by 1 */

      if ( c % 8 == 0 ) { /* output a space after 8 bits */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

Bitwise Operators



```
The result of combining the following
     65535 = 00000000 00000000 11111111 11111111
         1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
         1 = 00000000 00000000 00000000 00000001

The result of combining the following
        15 = 00000000 00000000 00000000 00001111
       241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
       255 = 00000000 00000000 00000000 11111111

The result of combining the following
       139 = 00000000 00000000 00000000 10001011
       199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
        76 = 00000000 00000000 00000000 01001100

The one's complement of
     21845 = 00000000 00000000 01010101 01010101
is
4294945450 = 11111111 11111111 10101010 10101010
```