PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 9: Bitfields, Unions, and Enums Bitwise Operations

Prof. Jeff Turkstra

Announcements

- Homework 2 style grades went out this morning
- Homework 4 is a great chance to catch up if you've fallen behind
  - There probably won't be another opportunity like it

Grades

- Remember to monitor the gradebook and address issues within one week!

Reading

- Read Chapter 5
  - …and/or Chapter 7 in Beej's
  - Probably repeatedly

sizeof()

- The sizeof() operator can tell us the size (number of bytes) of any:
  - Variable definition
  - Type declaration

```
int array[100];
printf("Size of char = %d\n", sizeof(char));
printf("Size of Array = %d\n", sizeof(array));
```

struct revisited - bitfields

- You can create fields within structs that do not contain a *round* number of bits…

```
struct special {
   unsigned int sign: 1;
   unsigned int exp: 11;
   unsigned int frac_high: 20;
   unsigned int frac_mid: 16;
   unsigned int frac_low: 16;
};
```

- How big is it?

Why use bitfields?

- When you really want to store a lot of fields that contain small values
- When you finally decide that you too need to write a new operating system and you have to access special hardware devices
- When you want to do format conversion between different types of data
  - Previous example was a structural format for an IEEE double precision floating point value

Rule of thumb for bitfields…

- If you really need to use bitfields, you'll know it
- You probably won't feel the need to use them in this class

union

- A union declaration looks just like a struct...

```
union my_union {
   int i;
   float f;
   char c;
} my_var;
```

- All the internal elements overlap

Initialization

```
union my_union {
  int i;
  float f;
};
union my_union my_var = { 5.0 };
```

- Assumes you are initializing the first field!

- C99 has designated union initializers:

```
union my_union my_var = { .f = 5.0 };
```

Why?

- When you really need to save space in your program and you know that some datum will be one of two disparate types
- Deep operating system hacking
  - Peripheral I/O manipulation
- Format conversion
- If you need it, you'll know
  - Don't use it in this class

enum

- An enum declaration looks sort of like a struct declaration...

```
enum color {
   RED,
   GREEN,
   BLUE
};
enum color my_hue = GREEN;
```

- Use this when you want to attach a label to a value

enum example

```
#include <stdio.h>

enum color {RED, GREEN, BLUE};
int main() {
  enum color my_hue = GREEN;

  switch (my_hue) {
    case RED:
    case GREEN:
      printf("Red or Green.\n");
      break;
    case BLUE:
      printf("Blue.\n");
      break;
  }
  return 0;
}
```

enums can also have values

- You can assign exact values to the enum declaration's members...

```
enum british_transport {
   LAND=1,
   SEA=2,
   AIR=3,
   SUBMARINE=2,
   FLYING_SAUCER=400
};
```

- You can assign a value to an enum definition using an integer too

Use of that enum

```
#include <stdio.h>

int main() {
   enum british_transport craft = AIR;
   printf("Value of craft is %d\n", craft);
   return 0;
}
```

Purdue Trivia

- The phrase “one brick higher” comes from the destruction of Heavilon Hall in 1894 – four days after construction was completed
  - Contained a groundbreaking locomotive testing plant
- President Smart proclaimed “We are looking this morning to the future, not the past… I tell you, …, that tower shall go up one brick higher!”
  - Actually nine bricks higher
- Current Heavilon Hall was built in 1959
  - Bells are in the Bell Tower (built 1995)
  - Clock is in the ME Gatewood Wing Atrium (2011)

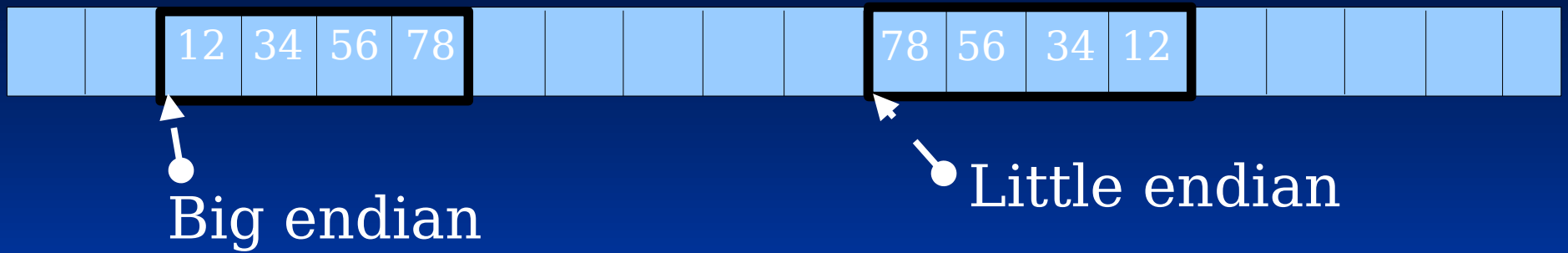
A note on endianness

- The order of bytes in a word or multi-byte value
  - Does not impact bit ordering for individual bytes!
- Two ways:
  - Big-endian: most significant byte first (lowest address)
  - Little-endian: least significant byte first

Example

- Consider the integer value 305419896
- In hexadecimal: 0x12 34 56 78
- Each pair of hexadecimal values corresponds to 8 bits or 1 byte

Stored in memory…



- Each box is 1 byte. We can look at it in binary too:

  0x12 = 0b0001 0010
  0x34 = 0b0011 0100
  0x56 = 0b0101 0110
  0x78 = 0b0111 1000

- 0x12345678 =
  0b00010010001101000101011001111000

- 0x78563412 =
  0b01111000010101100110100000010010

Bitwise operators

- You regularly use logical operators: || && in compound if statements
- What does this mean?

```
if (x) printf("x = %d\n", x);
```

- And this?

```
if (x && y) printf("x = %d\n", x);
```

- There are also bitwise operators: | &
- What does this mean?

```
if (x & y) printf("x = %d\n", x);
```

The difference between logical and bitwise operators

- Logical operators check whether the quantities are zero or non-zero. E.g.:

```
if (x && y)
   printf("y = %d\n", y);
```

...really means:

```
if ( (x != 0) && (y != 0) != 0 )
   printf("y = %d\n", y);
```

- And the result of && is either 1 or 0
- Use logical operators to make a yes/no decision

The difference between logical and bitwise operators

- Bitwise operators work on all of the bits. E.g.:

```
char x = 5;  /* binary 00000101 */
char y = 6;  /* binary 00000110 */
char z = 0;  /* binary 00000000 */

z = x & y;  /* result 00000100 */
```

- There are also OR (|), XOR (^), and NOT (~) operators
- Use bitwise operations when you want to work on the bits of a quantity

Truth tables

AND

X	Y	O
0	0	0
0	1	0
1	0	0
1	1	1

OR

X	Y	O
0	0	0
0	1	1
1	0	1
1	1	1

XOR

X	Y	O
0	0	0
0	1	1
1	0	1
1	1	0

NOT

X	O
0	1
1	0

We also have shift operators << and >>

- You can take a bunch of bits and shift it one way or another...

```
char x = 10;  /* binary 00001010 */
char y = 0;

y = x << 3;   /* result 01010000 */
```

- Note that every shift left is equivalent to a multiplication by two. E.g.:

```
y = x << 3;
```

means

$y = x * 2^3$

Example: cut a range of bits…

- Suppose we want to write a function that accepts a 32-bit integer and pulls a range of bits from somewhere in the middle:

```
unsigned int bit_range(unsigned int num,
                       unsigned int bits,
                       unsigned int offset) {
  return ((num >> offset) & ((1 << bits) –
1));
}
```

- You'll have to stare at that for a while to understand it…

Bit setting/clearing

- Use operators to clear/set bits in numbers…

```
int color = 44;  /* binary 00101100 */
int blue = 7;    /* binary 00000111 */

printf("Color with all blue is %d\n",
       color | blue);  /* 00101111 */
printf("Color with no blue is %d\n",
       color & ~blue);  /* 00101000 */
new_color |= blue & color;
printf("new_color: %d\n", new_color);
```

Bit checking

- How can we determine if a specified bit is set (i.e., set to 1)?

```
char bits = 44;  /* binary 00101100 */
char mask = 8;   /* binary 00001000 */

if ((bits & mask) == mask) {
  printf("The bit is set\n");
}
else {
  printf("The bit is cleared\n");
}
```

For next lecture

- Read Chapter 5
  - ...and/or Chapter 7 in Beej's
  - Probably repeatedly
- Understand the operators & and *

Takehome Quiz 5

```
unsigned short endian2_conversion(unsigned short number);
```

1. Write the above C function that converts a two byte number from Little Endian to Big Endian.
   - For example, if the number 0x1234 is passed to the function, it should return 0x3412
   - Hint: declare a union of one unsigned short and two unsigned chars
   - Do not use bitwise operations!
2. (Optional) What is your favorite programming language?

Boiler Up!