Overview of Today’s Lecture

• Announcements
• Exam 1 Review

Announcements

• Exam 1: Thursday 2/25, in class
  – Open book, open notes, and calculator allowed
  – 75 minutes
  – No electronic devices, except calculator

Exam Format

• Focus
  – C programming, questions specific to the AVR ATmega128 processor
  – Memory Mapped I/O
  – Memory Layout
  – Interrupts

• Open notes, no electronic devices except calculators

• Covers material covered in lectures
  – No questions about UARTs
  – No questions about ADC

• 60 points total
  – 15% of your course grade

Exam Topics
Exam Preparation

Suggested preparation steps:
- Review the following slides. You should have a deep understanding of the content that appears on them
- If not, go back and review the lecture material on the given topic
- Run through the questions at the end of this PowerPoint

This set of slides are not comprehensive
- Review all lecture slides
- Review homework questions, try to re-do those questions

Keywords
- char
- short
- int
- long
- float
- double
- enum
- struct
- union
- break
- case
- continue
- default
- do
- else
- for
- goto
- if
- return
- switch
- while
- auto
- const
- extern
- register
- static
- signed
- unsigned
- volatile
- sizeof
- typedef
- void

Syntax

- Could you write the following statements by hand?
  - Loops (for, while, do)
    - Write a for loop to sum elements of an array or count characters in a string
    - Do you know the syntax of a do while loop, for loop, and while loop?
  - typedef
    - Could you write a typedef definition
  - Switch statements
    - Do you know where the semi-colon and colons go in a switch/case statement?
    - Do you understand how the control flow can fall through a case?
  - Control flow
    - Do you understand the keywords break and continue and their use?

Know your Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Suffix/postfix increment</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>-</td>
<td>Suffix/postfix decrement</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>(type)</td>
<td>Type cast</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>*</td>
<td>Indirection (dereference)</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>sizeof</td>
<td>Size-of</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical NOT</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>Bitwise OR</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>Equality</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>/=</td>
<td>Division assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td>=</td>
<td>Bitwise OR assign</td>
</tr>
<tr>
<td>^=</td>
<td>Bitwise XOR assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>&amp;=</td>
<td>Bitwise AND assign</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>? :</td>
<td>Ternary conditional</td>
<td>Right-to-left</td>
</tr>
</tbody>
</table>

Know how to use Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>3</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>3</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>3</td>
</tr>
<tr>
<td>++</td>
<td>Increment</td>
<td>4</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
<td>4</td>
</tr>
<tr>
<td>? :</td>
<td>Ternary conditional</td>
<td>5</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Logical NOT</td>
<td>6</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>7</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Bitwise OR</td>
<td>7</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
<td>8</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>8</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>9</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>9</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>9</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>9</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left shift</td>
<td>10</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right shift</td>
<td>10</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>11</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Bitwise OR</td>
<td>11</td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td>12</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assign</td>
<td>12</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assign</td>
<td>12</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assign</td>
<td>12</td>
</tr>
<tr>
<td>/=</td>
<td>Division assign</td>
<td>12</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assign</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>=</td>
<td>Bitwise OR assign</td>
</tr>
<tr>
<td>^=</td>
<td>Bitwise XOR assign</td>
<td>12</td>
</tr>
<tr>
<td>&amp;=</td>
<td>Bitwise AND assign</td>
<td>12</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>Left shift assign</td>
<td>12</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
<td>Right shift assign</td>
<td>12</td>
</tr>
<tr>
<td>sizeof</td>
<td>Size-of</td>
<td>12</td>
</tr>
<tr>
<td>new</td>
<td>New</td>
<td>12</td>
</tr>
<tr>
<td>delete</td>
<td>Delete</td>
<td>12</td>
</tr>
</tbody>
</table>

Know how to use Operator Precedence

• Can you use this table?
Know your Declarations

- Do these declarations make sense?

```cpp
void main() {
    char x = 5, y = 10;
    char z;
    char array1[10];
    char array2[3] = {1, 2, 3};
    char array3[5] = {1, 2, 3};
    char *str = "Hello!";
    int i = 7;
    int *ptr = &i;
    int **pp = &ptr;
    char *p;
}
```

Know your Declarations

- Do these declarations make sense?

```cpp
struct House {
    unsigned long value;
    unsigned char baths;
    unsigned char bedrooms;
    unsigned char stories;
    unsigned long footage;
};

void main() {
    struct House my_home;
    struct House *bob_home = &my_home;
    my_home.baths = 1;
    my_home.value = 115000;
    bob_home->baths = 3;
    bob_home->value = 230000;
}
```

Data Structures

- Array access
- Pointers
  - Dereference
  - Address operator
- Access members of structs and unions
- Bit-field defined struct
- Know the difference between a struct and union

Pointers

- What are pointers
- Relationship between
  - pointers
  - array names
- Pointer arithmetic
- Students: Be sure to review class examples and homework problems related to pointers

C-strings

- Review the concept of C-strings
- Relationship between
  - C-strings
  - arrays
- Understand the importance of the NULL byte of a C-string

Bitwise operations

- Setting bits to 1
- Clearing bits to 0
- Testing for bits set to 1
- Testing for bits cleared to 0
- Generic systematic checking example

```cpp
if ( (x & MASK_ALL1s) == MASK_ALL1s 
    && (~x & MASK_ALL0s) == MASK_ALL0s 
    && (x & MASK_ANY1s) 
    && (~x & MASK_ANY0s) )
```
Variable Scope

- Global variables
- Local variables
- Local static variables
- Global static variables
- volatile variables

Memory Mapped I/O

- Concept of Memory Mapped I/O
  - Device registers can be accessed from a program as if accessing memory (i.e. devices’ registers are “mapped” to memory addresses)
  - Allow sending commands and checking status of a devices by just reading and writing to memory mapped locations.
- Polling device status vs. device initiating an interrupt.

Memory Layout

- Stack
  - What type of information is placed in the stack?
  - Typically starts at the top of memory and grows downward
- Function’s/Procedure’s Stack-Frame: The information placed on the stack by a function when it is called (e.g. local variables, input parameters, return address)
- Heap
  - Stores dynamically allocated memory (i.e. allocated using malloc).
  - Typically starts at a low address and grows upward
- Static Data segment: Region of memory where Global variables and static local variables are stored.

Interrupts

- What is an interrupt?
  - An event that occurs from outside your program (e.g. Keyboard requesting service, Timer that times out, UART receiving a byte of information)
- What is an Interrupt Service Routine (ISR)?
  - Code associated with a given interrupt (i.e. a given event), that executes when that interrupt occurs.
- What actions are taken when an interrupt occurs?
  - Normal program execution is paused
  - ISR execution begins
  - Normal program execution resumes after ISR completes

Writing the Body of a function

- There will be 1 or 2 problems where you will be asked to write the body of a function to implement a defined computation.

Show your work

- Please show intermediate steps. This will help us give partial credit
- Write down assumptions. In general Profs/TAs will not answer questions during the test.
  - They will just tell you to write down your assumptions
  - The exception will be if a major typo is found by a student.
Question 1

A. How many bytes are each of the following types (on the ATmega128)?
   • char, short, int, long, float, double

B. What range of values can be stored in an unsigned char?

C. What range of values can be stored in a signed char?

D. What is the value stored in x after this code runs?
   ```
   int x, y, z;
   x = y = z = 10;
   ```

Question 1 (answer)

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes sizeof()</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>signed char</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>0 to 255</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int (on ATmega128)</td>
<td>2</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>(pointer on ATmega128)</td>
<td>2</td>
<td>Address Space</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>2147483648 to 2147483647</td>
</tr>
<tr>
<td>signed long</td>
<td>4</td>
<td>2147483648 to 2147483647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>0 to 4294967295</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>4294967295 to 4294967295</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
<tr>
<td>double (on ATmega128)</td>
<td>4</td>
<td>±1.175e-38 to ±3.402e38</td>
</tr>
</tbody>
</table>

Question 2

A. Analyze the following code:
   ```
   char r = 0, s = 1, t = 2;
   char *p1 = &s;
   char *p2 = &t;
   char **pp3 = &p1;
   *p1 = 10;
   **pp3 = 15;
   *p2 = 30;
   *pp3 = &r;
   **pp3 = 5;
   *p1 = 25;
   ```

Question 2 (answer)

<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>25</td>
<td>15</td>
<td>30</td>
</tr>
</tbody>
</table>

Question 3a

- When is the condition of the following if statement true?

```java
if ((x = 3) || (x & 1)) {
   // do something
}
```
Question 3a (answer)

- When is the condition of the following if statement true?

```java
if ((x = 3) || (x & 1)) {
    // do something
}
```

- **The statement is always true.** Know the difference between the assignment operator (=) and the equality operator (==).
  - The value on the left (x = 3) is always true, as the value of an assignment is the value that was assigned. This allows programmers to have compound assignments.

---

Question 3b

- When is the condition of the following if statement true?

```java
if ((x == 3) || (x & 1)) {
    // do something
}
```

- **The statement is true if x is either equal to 3 or bit 0 is set.**

---

Question 4a

- When is the condition of the following if statement true?

```java
if (x & 0x08 == 0x08) {
    // do something
}
```

- **The statement is true if bit 0 of x is 1.** Operator precedence evaluates the == operator before the bitwise AND (&). Assumes TRUE (i.e. 0x08 == 0x08) evaluates to 1.

---

Question 4b

- When is the condition of the following if statement true?

```java
if ((x & 0x08) == 0x08) {
    // do something
}
```
**Question 4b (answer)**

- When is the condition of the following if statement true?

```
if ((x & 0x08) == 0x08) {
    // do something
}
```

- The statement is true if bit3 of x is set.
  - x = 0b00001000; condition is TRUE
  - x = 0b01001110; condition is TRUE
  - x = 0b00010101; condition is TRUE
  - x = 0b00000000; condition is FALSE
  - x = 0b11100000; condition is FALSE

**Question 6a**

- What are the values of c1, c2, c3, and c4 after the following code executes?

```java
char myarray[3] = {1, 2, 3};
char *ptr = myarray;
char c1 = *ptr++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

- Postfix increment has higher association precedence than dereference operator. But the increment does not occur until the next line.

**Question 6a (answer)**

```java
char myarray[3] = {1, 2, 3};
char *ptr = myarray;
```

- c1 is 1
- c2 is 2
- c3 is 1
- c4 is 2

**Question 6b**

- What are the values of c1, c2, c3, and c4 after the following code executes?

```java
char myarray[3] = {1, 2, 3};
char *ptr = myarray;
char c1 = (*ptr)++;
char c2 = *ptr;
char c3 = myarray[0];
char c4 = myarray[1];
```

**Question 6b (answer)**

```java
char myarray[3] = {1, 2, 3};
char *ptr = myarray;
```

- c1 is 1
- c2 is 2
- c3 is 2
- c4 is 2

**Review Homework Problems and Answers**