Announcements

• If you want something regraded, please click on "Request Regrade" in Gradescope.

• Extra Credit 2 has been released. It does not ask you to write a program; it asks you to analyze one. Please be sure you use the template so we can grade it on Gradescope.
Really Common Errors

The following will give errors in Gradescope:

• Using "/="/ to start a comment
  // a comment going to the end of the line
  This causes an error

• Declaring a variable anywhere except at the beginning of a block
  for (int n = 3; n < 10; n++) printf("%d\n", n);
  This also causes an error

• If Gradescope doesn't compile your program, please check for these before asking us for help

May 21, 2024
ECS 36A, Spring Quarter 2024
About That Midterm

1. Midterm statistics: mean, 77.69; median, 71; max, 127; min, 41; standard deviation 23.17

2. Do not panic! Even though the midterm grades are not curved, the final course grade will be, and the curving method will be independent of your class standing; it will solely depend on your grade.
Midterm Question 2

What does the Linux/UNIX command “rm xyzzy” do when “xyzzy” is a directory?

a) Move the directory “xyzzy” to the directory “Trash” in the user’s home directory.
b) Copy the directory “xyzzy” to the user’s home directory.
c) Delete the directory “xyzzy”.
d) Delete the files in the directory “xyzzy”.
e) It gives an error message.

a) Moving the directory "xyzzy" to directory "Trash" in home directory: mv xyzzy $HOME/Trash
b) Copying the directory "xyzzy" to the user's home directory: cp -r xyzzy $HOME/Trash
c) Deleting the directory "xyzzy": rmdir xyzzy
d) Deleting files in the directory "xyzzy": rm -r xyzzy/*
Midterm Question 4

If \( a = 0 \), \( b = 5 \), and \( c = -1 \), what are the values of \( (a \mid\mid (b \&\& c++)) \) and \( c \)?

a) \( (a \mid\mid (b \&\& c++)) \) is 0 and \( c \) is \(-1\)

b) \( (a \mid\mid (b \&\& c++)) \) is 0 and \( c \) is 0

c) \( (a \mid\mid (b \&\& c++)) \) is 0 and \( c \) is 1

d) \( (a \mid\mid (b \&\& c++)) \) is 1 and \( c \) is \(-1\)

e) \( (a \mid\mid (b \&\& c++)) \) is 1 and \( c \) is 0

\( a = 0 \) (false), so result is result of \( (b \&\& c++) \)

\( b = 5 \) (true) so result is that of \( c++ \)

\( c = -1 \) (true), so result of \( (b \&\& c++) \) is 1 (true), meaning result of \( (a \mid\mid (b \&\& c++)) \) is 1 (true)

After, the value of \( c++ \) is 0
Midterm Question 10

Evaluate the expressions below, and give the values of the named variables after the expression has been evaluated. If the expression contains a syntax error, or if a value is undefined, say so. Treat each part as separate; that is, assume the following variable values for all parts, regardless of whether a previous part has changed them.

\[
\text{int a = 0, b = 4, c = 5, d = -2, x; double dx;}\]

a) \(x++ = a + b\); give values of \(x\)
   \(x++\) is an expression and you cannot assign a value to an expression. Syntax error.

b) \(x = (a \text{ || b++}) \&\& (c++ \text{ || a++});\) give values of \(x, a, b, c\)
   
   \begin{align*}
   a &= 0 \text{ (false), and } b = 4 \text{ (true), so } (a \text{ || b++}) \text{ is } 1 \text{ (true);} \\
   c &= 5 \text{ (true), so } (c++ \text{ || a++}) \text{ is } 1 \text{ (true); thus } x &= 1 \text{ (true)}
   \end{align*}

Initially \(a = 0\), and \(a++\) isn't evaluated, so \(a = 0\)
Initial \(b = 4\), and \(b++\) is evaluated, so \(b = 5\)
Initial \(c = 5\), and \(c++\) is evaluated, so \(c = 6\)
Midterm Question 10

Reminder: int a = 0, b = 4, c = 5, d = -2, x; double dx;

c) \( x = a + b / c \); give value of \( x \)
   Division has higher precedence than addition, so do \( b / c \) first
   As \( b \) and \( c \) are ints, this is integer division, so \( 4 / 5 = 0 \)
   Adding a (0) to that gives \( x = 0 \)

   d) \( dx = x = 3.2 \); give values of \( x \) and \( dx \)
   Assigning a double to an int truncates the double, so \( x = 3 \)
   Assigning an integer to a double does not restore the fractional part, so \( dx = 3 \)

   e) \( x = (c \mod d) + ((c / d) \times d) \); give value of \( x \)
   By the C standard, integer division and remainder are defined so that this expression
   evaluates to \( c \), so \( x = c = 5 \).
A Quick Review of Pointers

• A pointer is simply an address
  • It’s just like a constant or variable

• A pointer constant cannot be changed
  • int pc [30]; /* here pc is a pointer constant and cannot be changed */

• A pointer variable can be changed
  • int *p; /* here p is a pointer variable and can be changed */
Midterm Question 11

• Setup:

```c
static int ar[] = { 10, 15, 4, 25, 3, -4 };
int *p;
p = &ar[2];
```

• In pictures:
Midterm Question 11(a)

• Value of:  
  * (p+1)

• In pictures:

p = &ar[2]; p+1 points to the next array element, so 
(p+1) = &ar[3], so *(p+1) = 25
Midterm Question 11(b)

• Value of:
  \( p[-1] \)

• In pictures:

```
0 1 2 3 4 5
10 15 4 25 3 -4
```

\( p[-1] = *(p-1), \quad p = &ar[2]; \quad p-1 \text{ points to the previous array element, so } *(p-1) = 15 \)
Midterm Question 11(c)

- Value of:
  \[ p - ar \]
- In pictures:

\[ \begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
10 & 15 & 4 & 25 & 3 & -4 \\
\end{array} \]

\[ p = &ar[2] = ar + 2, \text{so } p - ar = p + 2 - ar = 2; \]
Midterm Question 11(d)

• Value of:
  \( \text{ar}[\ast p++] \)

• In pictures:

\[
\begin{array}{c}
0 & 1 & 2 & 3 & 4 & 5 \\
\text{ar (100)} & 10 & 15 & 4 & 25 & 3 & -4 \\
\text{p (300)} & & & & & & \\
p = \&\text{ar}[2], \text{ so } *p = 4, \text{ and } \text{ar}[4] = 3
\end{array}
\]
Midterm Question 11(e)

• Value of:
  \((ar + ar[2])\)

• In pictures:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
10 & 15 & 4 & 25 & 3 & -4 \\
\end{array}
\]

\(ar[2] = 4, \text{ so } *(ar + ar[2]) = *(a + 4) = ar[4] = 3\)
Midterm Question 13

```c
int testandinc(int x)
{ return(x++); }

int p1testandinc(int *x)
{ return(*x++); }

int p2testandinc(int *x)
{ return((*x)++); }
```

```c
int a = 2;
int arr[3] = { 3, 4, 5 };
int *b = arr;
int *c = &arr[1];

int d = testandinc(a);
ed = p1testandinc(b);
f = p2testandinc(c);
```
Midterm Question 13

```c
int a = 2;
int arr[3] = { 3, 4, 5 };
int *b = arr;
int *c = &arr[1];
d = testandinc(a);
e = p1testandinc(b);
f = p2testandinc(c);
```
Approach

• Go through the program, and then get the values
Midterm Question 13

```c
int testandinc(int x) {
    return (x++);
}
...

d = testandinc(a)
```

May 21, 2024

ECS 36A, Spring Quarter 2024
Midterm Question 13

```c
int testandinc(int x)
{
    return(x++);
}

. . .
d = testandinc(a)

Return value of x
```
Midterm Question 13

```c
int testandinc(int x) {
    return(x++);
}

. . .
d = testandinc(a)
```

Add 1 to the value of x
Midterm Question 13

```c
int testandinc(int x) {
    return (x++);
}
```

```
Function ends
```

```c
d = testandinc(a)
```
Midterm Question 13

```c
int p1testandinc(int *x)
{
    return(*x++);
}
```

```c
e = p1testandinc(b)
```

![Diagram showing array and pointers]
Midterm Question 13

```c
int p1testandinc(int *x)
{
    return(*x++);
}

... 

e = p1testandinc(b)
```

May 21, 2024

ECS 36A, Spring Quarter 2024 24
Midterm Question 13

int pltestandinc(int *x)
{
    return(*x++);
}

. . .

e = pltestandinc(b)

May 21, 2024
Midterm Question 13

```c
int p1testandinc(int *x)
{
    return(*x++);
}

... 

e = p1testandinc(b)
```

Function ends
Midterm Question 13

int p2testandinc(int *x)
{
    return((x)++);
}

f = p2testandinc(c)
Midterm Question 13

```c
int p2testandinc(int *x)
{
    return((*x)++);
}

...  

f = p2testandinc(c)
```

May 21, 2024
Midterm Question 13

```c
int p2testandinc(int *x)
{
    return((*x)++);
}

...

f = p2testandinc(c)
```
Midterm Question 13

```c
int p2testandinc(int *x)
{
    return((*x)++);
}

... . .

f = p2testandinc(c)
```

May 21, 2024
Midterm Question 13

```c
int p2testandinc(int *x) {
    return((*x)++);
}
```

```
f = p2testandinc(c)
```

Function ends
## Midterm Question 13 Answers

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>arr or &amp;arr[0]</td>
</tr>
<tr>
<td>c</td>
<td>arr+1 or &amp;arr[1]</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>arr[0]</td>
<td>3</td>
</tr>
<tr>
<td>arr[1]</td>
<td>5</td>
</tr>
<tr>
<td>arr[2]</td>
<td>5</td>
</tr>
</tbody>
</table>

![Diagram of a 3-element array with values 2, 3, 5, and 5, with arrows indicating relationships between variables]
Rules for Pointers

• Treat a pointer like a constant or a variable
  • If it’s used as an array name, assume it’s a constant
  • Otherwise, assume it’s a variable
  • Note: in a function parameter list, it’s a variable, even if declared as an array

• A pointer p is an address
  • *p is the value stored at the address in p
  • &x is the address of the variable x
  • You can’t take the address of a constant, so this is illegal: char c[10]; d = &c;

• Draw pictures! They are very helpful
Recursion Speed-Up Technique

• When recursion recomputes a value, it adds time (and resources like memory use), which slows the program down

• Example: Fibonacci numbers, defined as \( f_0 = f_1 = 1, f_n = f_{n-1} + f_{n-2} \)

• To compute \( f_5 \):
  - \( f_5 = f_4 + f_3 \)
  - \( f_4 = f_3 + f_2 \)
  - \( f_3 = f_2 + f_1 \)
  - \( f_2 = f_1 + f_0 \)

• Notice the repetitions: \( f_3 \) is computed 2 times, and \( f_2 \) 3 times

• Now think of computing \( f_{100} \ldots \)
Memos

• Instead of recomputing, save intermediate values in an array
  • The array is like a memo book, hence the term "memo"
  ```
  int arr[5] = { -1, -1, -1, -1, -1 };
  ```
• When you compute $f_0, f_1, \ldots$ insert the computed values into
  ```
  arr[0], arr[1], \ldots
  ```
• Now the recursive call first checks $arr[n]$ to see if $f_n$ has been
  computed already
  • If yes, just return it; no recursion
  • If no, compute it, store the result in $arr[n]$, and return it
How Numbers and Letters Are Represented

• The computer stores these in binary representations

• Examples:
  • 345 in binary is 0000 0000 0000 0000 0001 0101 1001
  • –345 in binary is 1111 1111 1111 1111 1111 1110 1010 0111
    • This is two’s complement; flip the bits, add 1, and ignore overflow
  • If you add these, you get 0000 0000 0000 0000 0000 0000 0000 0000
  • ‘a’ is 97, which is 0110 0001
  • Floats use a different format:
    • 2.456 is 0100 0000 0001 1101 0010 1111 0001 1011

  sign bit  exponent  mantissa
Type Coercion

```c
int n;
float j = 2.456;

n = (int) j;
printf("float is %.6f, int is %d\n", j, n);
```

prints

```
float is 2.456000, int is 2
```
Representation of Data

• But if we want the bitwise representation of 2.456, we need to use a union
Unions

• Allows data to be viewed as multiple types
• Syntax is like a structure:

```c
union intfloat {
    int un;
    float uj;
} t;
```
Unions

• So to get the representation of 2.456 in hexadecimal:
  t.uj = 2.456
  printf("bit representation is 0x%x\n", t.un);
• And this prints
  bit representation is 0x401d2f1b
## Converting Hexadecimal to Binary

<table>
<thead>
<tr>
<th>bit pattern</th>
<th>hex digit</th>
<th>bit pattern</th>
<th>hex digit</th>
<th>bit pattern</th>
<th>hex digit</th>
<th>bit pattern</th>
<th>hex digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0100</td>
<td>4</td>
<td>1000</td>
<td>8</td>
<td>1100</td>
<td>c or C</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>0101</td>
<td>5</td>
<td>1001</td>
<td>9</td>
<td>1101</td>
<td>d or D</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>0110</td>
<td>6</td>
<td>1010</td>
<td>a or A</td>
<td>1110</td>
<td>e or E</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>0111</td>
<td>7</td>
<td>1011</td>
<td>b or B</td>
<td>1111</td>
<td>f or F</td>
</tr>
</tbody>
</table>

So 0x401d2f1b is 0100 0000 0001 1101 0010 1111 0001 1011
Similarly, 0000 0000 0000 0000 0000 0001 0101 1001 is 0x00000159
Easiest way to do this:

- **Binary to hexadecimal:** group binary digits in sets of 4, starting at the *end*; then use the table to translate
- **Hexadecimal to binary:** translate each hexadecimal digit to the 4 corresponding digits in the table above and merge them
Decimal to Binary

• Repeatedly divide by 2, then stop when you get 0
• Record the remainders; those are the binary digits
• Example: 345 in decimal:
  • 345 / 2 = 172 r 1
  • 172 / 2 = 86 r 0
  • 86 / 2 = 43 r 0
  • 43 / 2 = 21 r 1
  • 21 / 2 = 10 r 1
  • 10 / 2 = 5 r 0
  • 5 / 2 = 2 r 1
  • 2 / 2 = 1 r 0
  • 1 / 2 = 0 r 1
• So 345 in base 10 is 101011001 in base 2
Binary to Decimal

• Each digit is a power of 2, starting from the right (which is 2^0)

• So 101011001 in base 2 is:
  
  \[1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =\]

  \[256 + 0 + 64 + 0 + 16 + 8 + 0 + 0 + 1 = 345\]
# Dealing with Bits: Operation Tables

<table>
<thead>
<tr>
<th>AND (&amp;)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Examples:**
- \(10 \& 01 = 00\)
- \(11 \& 01 = 11\)

| OR (|) | 0 | 1 |
|------|---|---|
| 0    | 0 | 0 |
| 1    | 0 | 1 |

**Examples:**
- \(10 \mid 01 = 11\)
- \(11 \mid 01 = 11\)

<table>
<thead>
<tr>
<th>XOR (^)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Examples:**
- \(10 ^ 01 = 11\)
- \(11 ^ 01 = 10\)

<table>
<thead>
<tr>
<th>NOT (~)</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Examples:**
- \(\sim 10 = 01\)
- \(\sim 11 = 00\)
Dealing with Bits: Shift Operations

• $b << n$: shift $b$ left $n$ bits
  • Bits shifted beyond the end of the word are discarded
  • 0 bits are inserted at the right

• $b >> n$: shift $b$ right $n$ bits
  • If $b$ is signed, the high-order (most significant) bit is propagated
  • If $b$ is unsigned, 0 bits are inserted at the left
  • Bits shifted beyond the end of the word are discarded
How to Extract Bits

• Number the bits from 31 to 0
• To get the $i$-th bit of unsigned int $x$:
  \[ b = (x >> i) \& 01 \]

Example: 345 in binary:

0000 0000 0000 0000 0000 0000 0001 0101 1001

Extract bit 8:

\[ b = (345 >> 8) \& 01 = \]
\[ (0000 0000 0000 0000 0000 0000 0001 0101 1001 >> 8) \& 01 = \]
\[ (0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 \& 01 = 1 \]
How to Extract Groups of Bits

• Number the bits from 31 to 0
• To get the i-th through j-th bits of unsigned int x:
  \[ b = (x >> j) \& 0xZ \] where Z is the hex representation of i–j bits

Example: 345 in binary:

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
\end{array}
\]

Extract bits 8 to 5:

\[
b = (345 >> 5) \& 0xf = (345 >> 5) \& 0xf = \]

\[
(0000 0000 0000 0000 0000 0001 0101 1001 >> 5) \& 0xf = \\
(0000 0000 0000 0000 0000 0000 0000 0000 0101) \& 0xf = 0101 = 5
\]
Background

• System calls: interfaces to operating system functions
• Example: some Linux system calls
  • I/O: reading, writing, networking, etc.
  • Files: chown, chgrp, stat, etc.
  • Resource usage: ulimit, getrlimit, etc.
  • Timing: gettimeofday, time

• Library functions provide system-independent interface to them
  • Also provide other features
C Library Functions

• The C library provides many functions that do useful things
  • Standard I/O C library
  • Math library
• Character type
• String to integer or float/double types
• Handling options
• Time
• Random numbers
• String and memory manipulation