CS2211b

Software Tools and Systems Programming

Week 8b

Conditional Statements and More on Operators

Announcements

- Assignment #3 Posted
  - Due March 14th
  - Will be covering what you need today and next week
  - Recommend getting a head start now.

- Midterm this Saturday @ 9:30AM in WSC 55

Warmup Activity

1. Write a C program that finds and prints the volume of a pyramid. Read in the height (h), width (w) and length (l) as floating point values. Use the equation $\mathrm{V} = \frac{l \times w \times h}{3}$

2. Write a C program that reads in a phone number in the format *(ddd) ddd-dddd* and outputs it in the format *dddddddddd*. For example, input *(519) 914-5555* and output *5199145555*.

Warmup Activity

1. Write a C program that finds and prints the volume of a pyramid. Read in the height (h), width (w) and length (l) as floating point values. Use the equation $V = \frac{l \times w \times h}{3}$

Warmup Activity

1. Write a C program that finds and prints the volume of a pyramid. Read in the height (h), width (w) and length (l) as floating point values. Use the equation $V = \frac{l \times w \times h}{3}$

```c
#include <stdio.h>

int main() {
        float h, w, l, area;

        printf("Input height, width, and length: ");
        scanf("%f%f%f", &h, &w, &l);

        area = (h * w * l) / 3;
        printf("Area is: %f\n", area);

        return 0;
}
```

Warmup Activity

2. Write a C program that reads in a phone number in the format *(ddd) ddd-dddd* and outputs it in the format *dddddddddd*. For example, input *(519) 914-5555* and output *5199145555*.

Warmup Activity

2. Write a C program that reads in a phone number in the format *(ddd) ddd-dddd* and outputs it in the format *dddddddddd*. For example, input *(519) 914-5555* and output *5199145555*.

```c
#include <stdio.h>

int main() {
        int x, y, z;

        printf("Input phone number: ");
        scanf("(%d) %d-%d", &x, &y, &z);
        printf("New format: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

A Few Loose Ends:

- More on Operators & Expressions
- Constants
- Basic Type Casting
- `getchar() & putchar`

Constants

- We have seen that we can assign a hard coded constant value to a variable before:

```
int i = 42;
float f = 3.1415f;
char c = 'a';
```

- We can use the `const` keyword to tell the compiler that this variable value can not change after this line:

```
const int i = 42;
const float f = 3.1415f;
const char c = 'a';
```

- Trying to change the value will cause an error when compiling:

```
error: assignment of read-only variable
```

Constants

- We can also use preprocessor directives to define constants using the `#define` directive.
- **Define Directive Syntax:**

```
#define NAME VALUE
```

Examples:

```
#define MYCONST 32
#define PI 3.14159
```

- This will cause the preprocessor to replace every instance of the word MYCONST in your .c file with the text 32 (with a few exceptions, will not replace in strings for example) and PI with 3.14159.
- This can be dangerous as it does not check if the replacement makes sense or is of the correct type.
- More like doing find/replace on text than code.

Constants

Example 5:

/cs2211/week8/ex5.c

```c
#include <stdio.h>
#define PI 3.14159

int main() {
        float r;

        printf("Input radius: ");
        scanf("%f", &r);

        printf("Area of circle is: %f\n", PI * r * r);
        return 0;
}
```

Constants

Example 5:

/cs2211/week8/ex5.c

After Preprocessor is Run

```
int main() {
 float r;

 printf("Input radius: ");
 scanf("%f", &r);

 printf("Area of circle is: %f\n", 3.14159 * r * r);
 return 0;
}
```

The text PI is replaced with the value 3.14159

You can see the output of the preprocessor by using -E option with GCC

Constants

Example 5:

/cs2211/week8/ex5.c

```c
#include <stdio.h>
#define PI 3.14159

int main() {
        float r;

        printf("Input radius: ");
        scanf("%f", &r);

        printf("Area of circle is: %f\n", PI * r * r);
        return 0;
}
```

This program reads in a floating point value (the radius) over standard input and outputs the area of a circle to standard output.

Input/Output:

```
[dservos5@cs2211b week8]$ ex5
Input radius: 2.5
Area of circle is: 19.634937
```

Operators

+= -= *= /= %=

Compound Assignment

- Compound assignment operators provide a shorthand for expressions that perform a simple arithmetic operation on a variable and store the result back in the same variable.
- **Examples:**

Normal Method:

```
i = i + 1;
k = k * i;
j = j - 10;
m = m / (k + 6);
n = n % 2;
```

Compound Assignment:

```
i += 1;
k *= i;
j -= 10;
m /= k + 6;
n %= 2;
```

Few cases where they are not equivalent (if used with operators that have side effects)

Operators

Side Effects & Assignment Operators

- We have already seen that the assignment operator = assigns a value to a variable, but this is considered a **side effect** of the operation.
- Normally, operators do not change the value of the variables they are used on. For example, x + y returns the result of adding the variables x and y and does not alter their value.
- When an operator alters the value it is considered a **side effect**.
- The assignment operator actually returns the value of the variable after assignment is performed.
- **For example:**

```
j = k = i = 5;
```

Operators

Side Effects & Assignment Operators

- We have already seen that the assignment operator = assigns a value to a variable, but this is considered a **side effect** of the operation.
- Normally operators do not change the value of the variables they are used on. For example, x + y returns the result of adding the variables x and y and does not alter their value.
- When an operator alters t
- The assignment operator
variable after assignment
- **For example:**

```
j = k = i = 5;
```

`i` is assigned 5 and the value of 5 is returned, k is assigned 5 and the value of 5 is returned, `j` is assigned 5 and the value of 5 is returned (and nothing is done with it).

Result is all variables set to 5.

Operators

Side Effects & Increment/Decrement Operator

- We have seen that ++ and -- can be used to increment or decrement the value of a variable by one. This is a **side effect** of the operator.
- Depending on if we use ++ or -- as a prefix or postfix, the value returned is either the value of the variable before or after incremented/decremented.
- **Examples:**

```
int x = 5;
printf("%d ", x++);
printf("%d", x);
```

Operators

Side Effects & Increment/Decrement Operator

- We have seen that ++ and -- can be used to increment or decrement the value of a variable by one. This is a **side effect** of the operator.
- Depending on if we use ++ or -- as a prefix or postfix, the value returned is either the value of the variable before or after incremented/decremented.
- **Examples:**

```
int x = 5;
printf("%d ", x++);
printf("%d", x);
```

Output: 5 6

Operators

Side Effects & Increment/Decrement Operator

- We have seen that ++ and -- can be used to increment or decrement the value of a variable by one. This is a **side effect** of the operator.
- Depending on if we use ++ or -- as a prefix or postfix, the value returned is either the value of the variable before or after incremented/decremented.
- **Examples:**

```
int x = 5;
printf("%d ", ++x);
printf("%d", x);
```

Operators

Side Effects & Increment/Decrement Operator

- We have seen that ++ and -- can be used to increment or decrement the value of a variable by one. This is a **side effect** of the operator.
- Depending on if we use ++ or -- as a prefix or postfix, the value returned is either the value of the variable before or after incremented/decremented.
- **Examples:**

```
int x = 5;
printf("%d ", ++x);
printf("%d", x);
```

Output: 6 6

Operators

Side Effects & Undefined Behavior

- Can lead to confusing and undefined behavior when operators with side effects are used in statements with the same variable or with Compound Assignment.

- **Example:**

```
int x = 5;
printf("x is %d\n", x + ++x);
```

What is the output?

Operators

Side Effects & Increment/Decrement Operator

- Can lead to confusing and undefined behavior when operators with side effects are used in statements with the same variable or with Compound Assignment.

- **Example:**

```
int x = 5;
printf("x is %d\n", x + ++x);
```

Undefined behavior

gcc outputs: 12

clang outputs: 11

In /cs2211/week8/

ex6.c	gccex6	clangex6
ex6b.c	gccex6b	clangex6b

Operators

Division Results

- The type returned by the division operator is dependent on the operands.
- If both operands are integers the result is an integer (decimal places are truncated/dropped).
- If one or more operands are floating point values, the result is a floating point number.

- **Examples:**

```
1 / 2     = 0          1.0 / 2      = 0.5
3 / 4     = 0          3   / 4.0    = 0.75
5 / 3     = 1          5.0 / 3.0    = 1.67
```

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit Conversion).

Example:

/cs2211/week8/ex7.c

```c
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit C

Example:

/cs2211/week8/ex7.c

Output:

```
x: 7   y: 5   z: 1
a: 8.000000   b: 1.990000   c: 5.000000
```

```c
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit C

Example:

/cs2211/week8/ex7.c

```
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Output:

```
x: 7  y: 5  z: 1
a: 8.000000  b: 1.990000  c: 5.000000
```

Value of constant 7.1234 is being converted to an integer by truncating the decmial places (dropping them).

Not the same as rounding, a value like 7.99 would become 7 and not 8.

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit C

Example:

/cs2211/week8/ex7.c

```
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Output:

```
x: 7  y: 5  z: 1
a: 8.000000  b: 1.990000  c: 5.000000
```

Converting from a constant integer to a float works as you would expect. No difference to the number, but it can now have decimal places.

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit C

Example:

/cs2211/week8/ex7.c

```
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Output:

```
x: 7  y: 5  z: 1
a: 8.000000  b: 1.990000  c: 5.000000
```

Conversion also takes place when assigning values between variables of different types.

Here b is a float with the value 1.99 which is truncated when assigned to z. z is assigned the value 1.

Basic Type Casting

- In some cases C will convert the type of variables and constants for us (Implicit C

Example:

/cs2211/week8/ex7.c

Output:

```
x: 7   y: 5   z: 1
a: 8.000000   b: 1.990000   c: 5.000000
```

```c
#include <stdio.h>

int main(){
        int x = 7.1234f;
        int y = 5;
        int z;

        float a = 8;
        float b = 1.99f;
        float c;

        z = b;
        c = y;

        printf("x: %d  y: %d  z: %d\n", x, y, z);
        printf("a: %f  b: %f  c: %f\n", a, b, c);
}
```

Conversion also takes place when assigning values between variables of different types.

Here y is an integer with the value 5 which is assigned to c. c is assigned the value 5.

Basic Type Casting

- You can lose accuracy and run into problems if you convert back and forth between types.
- **Example:**

```
int x;
float a, b;
a = x = b = 3.1415f;
```

Basic Type Casting

- You can lose accuracy and run into problems if you convert back and forth between types.
- **Example:**

```
int x;
float a, b;
a = x = b = 3.1415f;
```

- **Result:**

  b = 3.1415

  x = 3

  a = 3.0000

Value of b (3.1415) was truncated when converted to an integer and sorted in x. Value of x (3) was then converted and stored in a as just 3.

Basic Type Casting

- We can also run into issues when converting to smaller types

- **Example:**

```
long l = 10000000000l;
int i = l;
printf("%d\n", i);
```

- **Output:**

```
1215752192
```

Basic Type Casting

- We can also run into issues when converting to smaller types

- **Example:**

```
long l = 10000000000l;
int i = l;
printf("%d\n", i);
```

Correct way to output would be:

```
printf("%ld\n", l);
```

- **Output:**

```
1215752192
```

Result is overflow

Basic Type Casting

- We can force C to convert a variable, constant or result of an expression using **type casting**.
- Uses **cast operator** with the following syntax:

```
(type_name) expression
```

- **Example:**

/cs2211/week8/ex8.c

```c
#include <stdio.h>

int main() {
        int sum = 32, count = 3;
        float mean;

        mean = (float) sum / count;
        printf("Value of mean : %f\n", mean );
        return 0;
}
```

Basic Type Casting

- We can force C to convert a variable, constant or result of an expression using **type casting**.
- Uses **cast operator** with the following syntax:

```
(type_name) expression
```

- **Example:**

/cs2211/week8/ex8.c

```c
#include <stdio.h>

int main() {
        int sum = 32, count = 3;
        float mean;

        mean = (float) sum / count;
        printf("Value of mean : %f\n", mean );
        return 0;
}
```

Output:

```
Value of mean : 10.666667
```

Basic Type Casting

Without casting, e.g. just:

Try /cs2211/week8/ex8bad.c

```
mean = sum / count;
```

The output would be:

```
Value of mean : 10.00000
```

- **Example:**

/cs2211/week8/ex8.c

```c
#include <stdio.h>

int main() {
        int sum = 32, count = 3;
        float mean;

        mean = (float) sum / count;
        printf("Value of mean : %f\n", mean );
        return 0;
}
```

Output:

```
Value of mean : 10.666667
```

Basic Type Casting

Example With Integers/Longs:

/cs2211/week8/ex9.c

```c
#include <stdio.h>

int main() {
        long l;
        int i = 100000;
        l = (i * i);
        printf("%ld\n", l);
}
```

Basic Type Casting

Example With Integers/Longs:

/cs2211/week8/ex9.c

```c
#include <stdio.h>

int main() {
        long l;
        int i = 100000;
        l = (i * i);
        printf("%ld\n", l);
}
```

Output:
1410065408

Result of 100,000 x 100,000 was to large to be integer. Overflows before converted to long.

Basic Type Casting

Example With Integers/Longs:

/cs2211/week8/ex9b.c

Can fix with casting!

```c
#include <stdio.h>

int main() {
        long l;
        int i = 100000;
        l = ((long)i * (long)i);
        printf("%ld\n", l);
}
```

Output:

```
10000000000
```

getchar()

- In addition to `scanf` we can take input from the standard input buffer with the `getchar` function.
- The `getchar` function is from stdio.h just like `printf` and `scanf`.
- Takes no arguments and or format string and returns a single character. The next character in the input buffer.
- Can be useful if we wish to code our own input system (rather than using `scanf`) or remove a character from the input buffer (more on this later).

- **Syntax:**

```
getchar()
```

putchar()

- Like `getchar` but prints a single character to the screen.
- Also part of stdio.h.
- Takes a single character as an argument that will be displayed on the screen (sent to stdout).

- **Syntax:**

```
putchar(char)
```

getchar() & putchar()

Example:

/cs2211/week8/ex10.c

```c
#include <stdio.h>

int main () {
        char c;

        printf("Enter character: ");
        c = getchar();

        printf("Character entered: ");
        putchar(c);

        putchar('\n');
        return(0);
}
```

getchar() & putchar()

Example:

/cs2211/week8/ex10.c

```c
#include <stdio.h>

int main () {
        char c;

        printf("Enter character: ");
        c = getchar();

        printf("Character entered: ");
        putchar(c);

        putchar('\n');
        return(0);
}
```

Reads in a single character.

It can be any character including spaces and line breaks.

getchar() & putchar()

Example:

/cs2211/week8/ex10.c

```c
#include <stdio.h>

int main () {
        char c;

        printf("Enter character: ");
        c = getchar();

        printf("Character entered: ");
        putchar(c);

        putchar('\n');
        return(0);
}
```

Outputs a given character.

An integer will be converted to a character using the ASCII table.

getchar() & putchar()

Example:

/cs2211/week8/ex10.c

```c
#include <stdio.h>

int main () {
        char c;

        print
        c = getchar();

        printf("Character entered: ");
        putchar(c);

        putchar('\n');
        return(0);
}
```

Can be used with character constants to output a single character (e.g. a line break).

getchar() & putchar()

Example:

/cs2211/week8/ex10.c

```c
#include <stdio.h>

int main () {
        char c;

        printf("Enter character: ");
        c = getchar();

        printf("Character entered: ");
        putchar(c);

        putchar('\n');
        return(0);
}
```

Example Output:

```
[dservos5@cs2211b week8]$ ex10
Enter character: D
Character entered: D
```

Conditional Statements

Logical Expressions

- We have briefly discussed the **equality** and **relation** operators before, but have yet to see how they work or what they return.

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Logical Expressions

- We have briefly discussed the **equality** and **relation** operators before, but have yet to see how they work or what they return.

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Ch va
>	Ch rig
<	Ch operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

True result equals: 1

False result equals: 0

No Boolean data type by default (in C99 and up, one can be included from the library stdbool.h)

Logical Expressions

Example:

/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 1

Value of i (5) is equal to value of i (5) so result is 1.

The statement **5 is equal to 5** is **true**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 0

Value of i (5) is equal to value of i (5) so result is 0.

The statement **5 is not equal to 5** is **false**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 1

Value of i (5) is greater than 2 so result is 1.

The statement **5 is greater than 2** is **true**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 0

Value of i (5) is less than 4 so result is 0.

The statement **5 is less than or equal to 4** is **false**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 0

Value of i (5) is not equal to f (4.957) so result is 0.

We can compare values and variables of different types.

The statement **5 is equal to 4.957** is **false**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 1

Value of i (5) is greater than f (4.957) so result is 1.

We can compare values and variables of different types.

The statement **5 is greater than to 4.957** is **true**.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 1

Value of c ('a') is less than 'b' on the ASCII table (97 vs. 98), so the result is 1.

We can compare character types based on their ASCII values.

Logical Expressions

Example:
/cs2211/week8/ex11.c

```c
#include <stdio.h>

int main() {
        int i = 5;
        float f = 4.957;
        char c = 'a';

        printf("%d\n", i == i);
        printf("%d\n", i != i);
        printf("%d\n", i > 2);
        printf("%d\n", i <= 4);
        printf("%d\n", i == f);
        printf("%d\n", i > f);
        printf("%d\n", c < 'b');
        printf("%d\n", c == 97);

        return 0;
}
```

Output: 1

Value of c ('a') is equal to 97 (on the ASCII table), so the result is 1.

When characters are compared to numbers, the value of the character is its value on the ASCII table.

Logical Expressions

- Logical Operators (!, && and ||) function like NOT, AND and OR do in other programming languages you may be familiar with.
- Also return 0 for FALSE and 1 for TRUE.
- Result will be based on following truth tables:

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

A	!A
0	1
1	0

Logical Expressions

Example:
/cs2211/week8/ex12.c

```c
#include <stdio.h>

int main() {
        printf("%d ", 0 || 0);
        printf("%d ", 1 || 0 );
        printf("%d\n", 1 || 1 );

        printf("%d ", 5 > 4 || 5 == 6);
        printf("%d\n", 5 >= 7 || 5 != 5);

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Logical Expressions

Example:
/cs2211/week8/ex12.c

```c
#include <stdio.h>

int main() {
        printf("%d ", 0 || 0);
        printf("%d ", 1 || 0 );
        printf("%d\n", 1 || 1 );

        printf("%d ", 5 > 4 || 5 == 6);
        printf("%d\n", 5 >= 7 || 5 != 5);

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Output:

```
0 1 1
```

Same as you would expect from the truth table for ||.

Logica

Example:

/cs2211/week

```
#include <

int main()
        pr
        pr
        pr

        printf("%d ", 5 > 4 || 5 == 6);
        printf("%d\n", 5 >= 7 || 5 != 5);

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Output:

```
1 0
```

Result of 5 > 4 is checked (it is true so the reuslt is 1). As this is true C does not need to check the statement 5 == 6 to determine that the result is 1 (true). This is important as if the statement had a side effect, it would not be run.

Logica

Example:

/cs2211/week

```c
#include <

int main()
        pr
        pr
        printf("%d\n", 1 || 1 );

        printf("%d ", 5 > 4 || 5 == 6);
        printf("%d\n", 5 >= 7 || 5 != 5);

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Output:

```
1 0
```

For example, if the statement was 5 > 4 || x++ the value of x would not be incremented as C would not need to evaluate the other part of the statement (as 5 > 4 is true).

Logica

Example:
/cs2211/week

```c
#include <

int main() {
        printf("%d ", 0 || 0);
        printf("%d ", 1 || 0 );
        printf("%d\n", 1 || 1 );

        printf("%d ", 5 > 4 || 5 == 6);
        printf("%d\n", 5 >= 7 || 5 != 5);

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Output:

```
1 0
```

The second output is 0 as both 5 >= 7 and 5 != 5 result in 0s (false).

Logical Expressions

Example:
/cs2211/week8/ex12.c

```c
#include <stdio.h>

int main() {
        printf("%d ", 0 || 0);
        printf("%d ", 1 || 0 );
        printf("%d\n", 1 || 1 )

        printf("%d ", 5 > 4 ||
        printf("%d\n", 5 >= 7 |

        printf("%d ", 0 && 1);
        printf("%d ", 1 && 1);
        printf("%d\n", 0 && 0);

        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
```

Output:

```
0 1 0
```

Same as you would expect from the truth table for &&.

/cs2211/week8/ex12.c

```c
#include <stdio.h>

in
```

Output:

```
0 1
```

5 > 4 returns 1 (true) but 5 < 4 returns 0 (false) so whole expression is false (returns 0).

Second output is 1 (true) as both expressions are true (5 == 5 and 5 > 1.23 both return 1).

```c
        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
        printf("%d\n", (5 == 5 || 5 != 5) && (5 > 4 || 5 < 10));

        return 0;
}
```

/cs2211/week8/ex12.c

```c
#include <stdio.h>

in
```

Output:

```
0 1
```

If the first expression is false, for example in the statement:
5 == 6 && 5 > 4

The second statement will not be evaluated (as this is not nessary to know that the whole statement is false).

```c
        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
        printf("%d\n", (5 == 5 || 5 != 5) && (5 > 4 || 5 < 10));

        return 0;
}
```

/cs2211/week8/ex12.c

```c
#include <stdio.h>

int main() {
```

Output:

```
0 1
```

This can be important if the statement has side effects as in:

5 == 6 && x++

As x will not be incremented.

```c
        printf("%d ", 5 > 4 && 5 < 4);
        printf("%d\n", 5 == 5 && 5 > 1.23);

        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
        printf("%d\n", (5 == 5 || 5 != 5) && (5 > 4 || 5 < 10));

        return 0;
}
```

Output:

```
0 0 1
```

The NOT opeartor (!) switches a 0 to 1 or a 1 to 0 (true to false and false to true).

We can use round brackets to enforce a particular order of operations. In the second output, the ! opeartor is applied after 5 > 4 && 5 < 10 is evaluated.

In the third output, the two OR (||) expressions are evaluated first and the results are ANDed (&&) together. In this case, both expressions in the round brackets must be true for the overall result to be true.

```
        printf("%d ", !1);
        printf("%d ", !(5 > 4 && 5 < 10));
        printf("%d\n", (5 == 5 || 5 != 5) && (5 > 4 || 5 < 10));

        return 0;
}
```

IF Statements

- IF statements allow our code to take different paths based on the result of a logical expression.
- **Basic Syntax:**

```
if ( expression )
   statement
```

- If **expression** evaluates to **1** (or any non zero value) the **statement** will be run.
- If **expression** evaluates to **0** the **statement** will not be run and execution will continue on the next line.
- In this format, only one **statement** can be used (not multiple lines/commands).

Compound Statements

- To give an IF statement multiple statements to run if true, we need to use compound statements.
- Compound statements allow us to group multiple statements together such that they will be treated as one statement.
- Denoted with braces, { and }.
- **Syntax:**

```
{
    statement1;
    statement2;
    …
    statementN;
}
```

All of these statements are grouped as a single statement.

Example with scanf

- In addition to setting the value of variables we give it, `scanf` also has a return value.
- `scanf` returns the number of format specifiers it has matched or a negative number if there is a read error.
- We can use this return with an if statement to help validate input.
- **Example:**

  Read in a phone number in the format `(DDD) DDD-DDDD` and convert it to the format DDDDDDDDDD. Print an error, if `scanf` failed to read all of the numbers.

Example with scanf

- **Example:**

  Read in a phone number in the format (DDD) DDD-DDDD and convert it to the format DDDDDDDDDD. Print an error, if scanf failed to read all of the numbers.

/cs2211/week8/ex13.c

```c
#include <stdio.h>

int main() {
        int x, y, z, r;

        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        }

        printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

Example with scanf

- **Example:**

  Read in a
  format D

/cs2211/we

Reads in a number formated like (D) D-D where Ds are integers and save them into the variables x, y and z.

scanf returns the number of successfully matched format specifiers and that return is stored in the variable r.

```
#include

int main(
        i

        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        }

        printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

Example with scanf

- The value of the variable r is checked to see if it is anything other than 3 (the number of integers we were trying to read in).

  The expression r != 3 will return 1 if r is not equal to 3.

  If the if statement receives a 1 (r is not equal to 3) the group of code surrounded by braces ({ and }) will be executed. Otherwise, the code will return to executing after the closing brace (}).

```
/cs
#i
in
        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        }

        printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

Example with scanf

- If r is not 3 the highlighted code will be run.

  In this case, the text “Error: Bad input\n” will be output and the main function will return an exit status of 1 (indicating failure).

  Calling return here causes the code to exit at this point (does not run any more lines in the main function).

```
/cs
#i
in
        int x, y, z, r;

        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        }

        printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

Example with scanf

- If r is equal to 3 the code will begin executing after the closing brace (}).

  In this case, the second printf statement is running outputting the phone number if the 10 digit format and an exit status of 0 (indicating success) is returned.

```
int main() {
        int x, y, z, r;

        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        }

        printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
        return 0;
}
```

Example with scanf

Example Input/Output:

```
[dservos5@cs2211b week8]$ ex13
(123) 456-7890
Phone Number: 1234567890

[dservos5@cs2211b week8]$ ex13
(001) 001-0001
Phone Number: 0010010001

[dservos5@cs2211b week8]$ ex13
123-456-7890
Error: Bad input!

[dservos5@cs2211b week8]$ ex13
cat
Error: Bad input!
```

```
}
```

Else and Else If

- The IF statement also supports Else and Else if syntax similar to the IF statement we saw in shell scripts.
- **Basic Syntax:**

```
if ( expression1 )
   statement1
else if ( expression2 )
  statement2
else if ( expression3 )
  statement3
…
else if ( expressionN )
   statementN
```

```
if ( expression1 )
   statement1
else
   statement2
```

Else and Else If

- The IF statement also supports Else and Else if syntax similar to the IF statement we saw in shell scripts.
- **Basic Syntax:**

```
if ( expression1 )
    statement1
else if ( expression2 )
   statement2
else if ( expression3 )
   statement3
…
else if ( expressionN )
    statementN
```

If the expression before it is false, the next else if statement will be tested. If its expression is true its statement will be executed and the subsequent else if statements will be ignored.

Else and Else If

- The IF statement also supports Else and Else if syntax similar to the IF statement we saw in shell scripts.
- **Basic Syntax:**

If the expression1 is true statement1 will be executed, if it is false statement2 will be executed.

```
if ( expression1 )
   statement1
else
   statement2
```

Else and Else If

- The IF statement also supports Else and Else if syntax similar to the IF statement we saw in shell scripts.
- **Basic Syntax:**

```
if ( expression1 )
   statement1
else if ( expression2 )
  statement2
…
else if ( expressionN )
   statementN
else
   statement_otherwise
```

We can use else if and else statements together in the same if statement if we wish.

statement_otherwise will only be run if all expressoins are false.

2nd Example with scanf

Example: Further improve the phone number example.

/cs2211/week8/ex14.c

```c
#include <stdio.h>
int main() {
        int x, y, z, r;
        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

2nd

Exam

/cs2211

Improved last example to check values of x, y and z to ensure they are between the correct ranges.

Will no longer allow inputs like (001) 001-1234 that are not valid (area and central office codes can not start with a zero).

Also will not allow inputs like (1) 2-3. Must have correct number of digits.

```
#inclu
int ma
                return 1;
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

2nd Example with scanf

Example: Further improve the phone number example.

/cs2211

Check that the value of x is less than 100 or greater than 999 (this would mean the area code is the wrong number of digits).

If it is, we print an error and return a non zero exit status.

```
#incl
int m

        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

2nd Example with scanf

Example: Further improve the phone number example.

/cs2211

Check that the value of y is less than 100 or greater than 999 (this would mean the central office code is the wrong number of digits).

If it is, we print an error and return a non zero exit status.

```
#incl
int m

        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

2nd Example with scanf

Ex

/cs

#i

in

Check that the value of z is less than 1000 or greater than 9999 (this would mean the line number is the wrong number of digits).

We are using 1000 and 9999 this time as the line number is four digits.

If it is, we print an error and return a non zero exit status.

```
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

2nd Example with scanf

Exan

/cs221

Otherwise (if none of the expressions were true), we print the phone number in the 10 digit format and return a 0 exit status.

```
#include <stdio.h>
int main() {
        int x, y, z, r;
        r = scanf("(%d) %d-%d", &x, &y, &z);
        if(r != 3) {
                printf("Error: Bad input!\n");
                return 1;
        } else if(x < 100 || x > 999) {
                printf("Error: Bad area code!\n");
                return 1;
        } else if(y < 100 || y > 999) {
                printf("Error: Bad central office code!\n");
                return 1;
        } else if(z < 1000 || z > 9999) {
                printf("Error: Bad line number!\n");
                return 1;
        } else {
                printf("Phone Number: %.3d%.3d%.4d\n", x, y, z);
                return 0;
        }
}
```

The `switch` Statement

- The switch statement is similar to the case statement in shell scripts but may only be used on characters and integers.
- It also does not support any patterns, just literal values.
- **Basic Syntax:**

```
switch ( expression ) {
   case value1: statement1
   case value2: statement2
   …
   case valueN: statementN
   default: default_statement
}
```

Expression is an expression that returns an integer value (characters are treated as integers in C).

Value is a constant expression. Either a literal constant (e.g. 'A' or 1) or a constant variable.

Statement is one or more statements to execute if the expression matches the value.

```
switch ( expression ) {
   case value1: statement1
   case value2: statement2
   …
   case valueN: statementN
   default: default_statement
}
```

switch Example

/cs2211/week8/ex15.c

```c
#include <stdio.h>

int main() {
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

/cs221

Ask for input and read in a character into the variable grade (we could have also used `getchar()` here).

```
#inc
int main() {
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

/cs2211/week8/ex15.c

Our switch statement will be testing he value of the variable grade.

```
#inclu

int ma
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

If the value of grade is equal to 'A' (i.e. 65) than the word “Excellent” will be printed.

If we do not want to run the code in the other cases, we need to include the break keyword after the statements we wish to include in this case.

```
/cs221
#incl
int m
        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

/cs2211/week8/ex15.c

```c
#include <stdio.h>

int main() {
        char grade;

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

Similarly, this prints “Good” if grade is B, “Average” if it is C, “Poor” if it is D and “Failing” if it is F.

switch Example

/cs2211/week8/ex15.c

```c
#include <stdio.h>

int main() {
        char grade;
                        break;
                case 'D': printf("Poor\n");
                        break;
                case 'F': printf("Failing\n");
                        break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

The default statement is run if none of the cases are matched. In this case “Illegal grade!” will be printed if grade is not A, B, C, D or F.

The default statement should come last and does not require a break statement.

switch Example

Example Input/Output:

```
[dservos5@cs2211b week8]$ ex15
Input a letter grade: A
Excellent!

[dservos5@cs2211b week8]$ ex15
Input a letter grade: C
Average

[dservos5@cs2211b week8]$ ex15
Input a letter grade: Z
Illegal grade!

[dservos5@cs2211b week8]$ ex15
Input a letter grade: 9
Illegal grade!
```

switch Example

What if we remove the breaks?

```c
int main() {
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

What if we remove the breaks?

```c
int main() {
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                case 'B': printf("Good\n");
                case 'C': printf("Average\n");
                case 'D': printf("Poor\n");
                case 'F': printf("Failing\n");
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

Input/Ouput will be:

```
Input a letter grade: A
Excellent!
Good
Average
Poor
Failing
Illegal Grade!
```

Runs everything bellow the value that was matched!

```
        return 0;
}
```

switch Example

Input/Ouput will be:

```
Input a letter grade: C
Average
Poor
Failing
Illegal Grade!
```

Runs everything bellow the value that was matched!

```
                case 'F': printf("Failing\n");
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

switch Example

/cs

#i

How can we use this to ignore case?

```
int main() {
        char grade;

        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'A': printf("Excellent!\n");
                          break;
                case 'B': printf("Good\n");
                          break;
                case 'C': printf("Average\n");
                          break;
                case 'D': printf("Poor\n");
                          break;
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

```c
#include <stdio.h>

int main() {
        printf("Input a letter grade: ");
        scanf("%c", &grade);

        switch(grade) {
                case 'a':
                case 'A': printf("Excellent!\n");
                          break;
                case 'b':
                case 'B': printf("Good\n");
                          break;
                case 'c':
                case 'C': printf("Average\n");
                          break;
                case 'd':
                case 'D': printf("Poor\n");
                          break;
                case 'f':
                case 'F': printf("Failing\n");
                          break;
                default: printf("Illegal grade!\n");
        }

        return 0;
}
```

How can we use this to ignore case?

```c
#include <stdio.h>

int main() {
    char grade;

    printf("Input a letter grade: ");
    scanf("%c", &grade);

    switch(grade) {
        case 'a':
        case 'A': printf("Excellent!\n");
                  break;
        case 'b':
        case 'B': printf("Good\n");
                  break;
        case 'c':
        case 'C': printf("Average\n");
                  break;
        case 'd':
        case 'D': printf("Poor\n");
                  break;
        case 'f':
        case 'F': printf("Failing\n");
                  break;
        default: printf("Illegal grade!\n");
    }

    return 0;
}
```

If 'a' or 'A' is input, the text “Excellent!” is printed.

In-Class Activity

1. Write a C program reads in a whole number from the user and outputs “Odd” if it is odd and “Even” if it is even.

2. Write a C program that reads in a whole number representing a pH value and outputs “Neutral” if it is 7, “Weak Acid” if it is 6, 5 or 4, “Strong Acid” if it is 1 or 0, “Weak Base” if it is 8, 9, or 10 and “Strong Base” if it is 11, 12, 13 or 14.

Use a switch statement!

In-Class Activity

1. Write a C program reads in a whole number from the user and outputs “Odd” if it is odd and “Even” if it is even.

In-Class Activity

1. Write a C program reads in a whole number from the user and outputs “Odd” if it is odd and “Even” if it is even.

```
#include <stdio.h>
int main() {
        int n;
        printf("Input Number: ");
        scanf("%d", &n);

        if(n % 2 == 0) {
                printf("Even\n");
        } else {
                printf("Odd\n");
        }

        return 0;
}
```

In-Class Activity

2. Write a C program that reads in a whole number representing a pH value and outputs “Neutral” if it is 7, “Weak Acid” if it is 6, 5 or 4, “Strong Acid” if it is 1 or 0, “Weak Base” if it is 8, 9, or 10 and “Strong Base” if it is 11, 12, 13 or 14.

In-Class Activity

```
#include <stdio.h>
int main() {
        int ph;
        printf("Input pH: ");
        scanf("%d", &ph);

        switch(ph){
                case 7: printf("Neutral\n"); break;
                case 6: case 5: case 4:
                        printf("Weak Acid\n"); break;
                case 1: case 0:
                        printf("Strong Acid\n"); break;
                case 8: case 9: case 10:
                        printf("Weak Base\n"); break;
                case 11: case 12: case 13: case 14:
                printf("Strong Base\n"); break;
                 default: printf("Invalid pH!\n");
        }

    return 0;
}
```