Exercise 4:
Write an if statement that assigns the value of x to the variable y if x is in between 1 and 20, otherwise y is unchanged.
QUIZ Lesson 4

Exercise 4:
Write an if statement that assigns the value of x to the variable y if x is in between 1 and 20, otherwise y is unchanged.

Exercise 5:
Now do the same using a conditional operator ... ? ... : ...
QUIZ Lesson 4

What is the difference between relational operators and logical operators?

Give 3 examples of each.
9. In the following list, which has higher precedence?
   a. == or <
   b. * or +
   c. != or ==
   d. >= or >
QUIZ Lesson 4

• Is $-42$ true or false?

• What is a more economical way to write $\text{if } (a \neq 0)$?
Lesson 5

Functions
#include <stdio.h>

long cube(long x);

long input, answer;

int main(void)
{
    printf("Enter an integer value: ");
    scanf("%ld", &input);
    answer = cube(input);
    /* Note: %ld is the conversion specifier for a long integer */
    printf("\nThe cube of %ld is %ld.\n", input, answer);
    return 0;
}

// Function: cube() - Calculates the cubed value of a variable
long cube(long x)
{
    long x_cubed;
    x_cubed = x * x * x;
    return x_cubed;
}
Anatomy of a function

Local variable (recognized only inside the function)

Function header

Function body

```c
long cube(long x)
{
    long x_cubed;
    x_cubed = x * x * x;
    return x_cubed;
}
```
QUIZ: Write a function that returns the absolute value of its float argument

```c
long cube(long x)
{
    long x_cubed;
    x_cubed = x * x * x;
    return x_cubed;
}
```

Hint: use an if statement or a conditional operator
A function may have multiple `return` statements

```c
int room_assign( char li) {
    // This if statement tests whether the first initial is A-M or N-Z
    // with the first group being assigned room 1045 and the rest 1055
    // The or part of the statement lets us check both lower and uppercase
    if ((li >= 'a' && li <= 'm') || (li >= 'A' && li <= 'M'))
        return 1045;
    else
        return 1055;
}
```
... but any call to a function always returns only one value!

//the little function who tried
unsigned multiple(unsigned){
    return 42;
    return 43;
    return 44;
}

When the first return is encountered during program execution, the function exits, and the value from that return is returned.
... actually a **void** function does not return any value!

```c
void no_return (int x){
    printf("x is %d\n", x);
}
```

Do not confuse a value **printed** by a function with a value **returned** by a function!
Parameters vs. arguments

Sometimes called *formal parameters*

Sometimes called *actual parameters*
int x = 1, y = 2;

void demo(void);

int main(void)
{
    printf("Before calling demo(), x = %d and y = %d.", x, y);
    demo();
    printf("After calling demo(), x = %d and y = %d\n", x, y);
    return 0;
}

void demo(void)
{
    /* Declare and initialize two local variables. */
    int x = 88, y = 99;

    printf("Within demo(), x = %d and y = %d.", x, y);
}
Before calling demo(), x = 1 and y = 2.
Within demo(), x = 88 and y = 99.
After calling demo(), x = 1 and y = 2.
Conclusion: local variables within a function are distinct from any global variables having the same name, and from any other local variables with the same name within other functions.
To do in notebook for next time:

• Read and take notes pp.92-108
• Answer end-of-chapter quizzes 1 – 6
• Answer end-of-chapter exercise 1
QUIZ

Write the declaration (prototype) for a function **ssquares** that **returns** the sum of the squares of its two **double** arguments.

Then write the **definition**.
QUIZ

Give an example of how the previous function would be used in the main program.
QUIZ

What is the difference between formal and actual parameters (arguments)?

Explain using the example of the previous function `ssquares`. 
QUIZ

What does this function return?

```c
void foo(int a){
    printf("%d\n", a*a);
}
```
Why functions?

FIGURE 5.1
When a program calls a function, execution passes to the function and then back to the calling program.

main()
{
    call func1
    ...
call func2
    ...
call func3
}

func1()
{
}

func2()
{
}

func3()
{
}

FIGURE 5.2
A structured program is organized hierarchically.
Two ways to write our functions

```c
#include <stdio.h>

/* recursive implementation of function factorial */
unsigned factorial(unsigned a);

void main(){
    printf( "Factorial is %u\n\n", factorial(13));
}

unsigned factorial(unsigned a){
    if (a == 1)  // base case
        return 1;
    else        // recursive step
        return a * factorial(a-1);
}
```

```c
#include <stdio.h>

/* recursive implementation of function factorial */
unsigned factorial(unsigned a){
    if (a == 1)  // base case
        return 1;
    else        // recursive step
        return a * factorial(a-1);
}

void main(){
    printf( "Factorial is %u\n\n", factorial(13));
}
```

If the function definition is placed before main, we don’t need a prototype!
Recursive functions

```c
#include <stdio.h>

/* recursive implementation of function factorial */
unsigned factorial(unsigned a){
    if (a == 1) // base case
        return 1;
    else // recursive step
        return a * factorial(a-1);
}

void main(){
    printf( "Factorial is \%u\n\n", factorial(13));
}
```

Factorial is 1932053504
How a recursive function works

factorial (5) = 5 * factorial (4)
  → 4 * factorial (3)
    → 3 * factorial (2)
      → 2 * factorial (1)
        → 1

On the way “down”, five instances of the function are left “pending”, because their return value is not yet known.
Because the results increase very fast, the most appropriate data type for **factorial** is unsigned long long.

```c
#include <stdio.h>

/* recursive implementation of function factorial using unsigned long long integers*/
unsigned long long factorial(unsigned long long a){
    if (a == 1) // base case
        return 1;
    else // recursive step
        return a * factorial(a-1);
}

void main(){
    printf( "Factorial is %llu\n\n", factorial((unsigned long long)20));
}
```

2.4 quintillion = $2.4 \cdot 10^{18}$
Of course, the range of any data type will be exceeded sooner or later ...

14.1 quintillion = 14.1 \cdot 10^{18}
Correct value is 51.0 \cdot 10^{18}
QUIZ

Write a function with two double arguments, that returns their product, as a float.
QUIZ

Define a recursive function.
QUIZ

What are the two parts of any recursive function?
Sum of numbers from 1 to n

```c
#include <stdio.h>

/* recursive implementation of function sum */
unsigned sum(unsigned a){
    if (a == 0) // base case
        return 0;
    else // recursive step
        return a + sum(a-1);
}

void main(){
    unsigned n = 100;
    printf( "sum from 1 to %u is %u\n\n", n, sum(n));
}
```

```
sum from 1 to 3 is 6
```
Sum of numbers from 1 to n

```c
#include <stdio.h>

/* recursive implementation of function sum */
unsigned sum(unsigned a){
    if (a == 0)     // base case
        return 0;
    else           // recursive step
        return a + sum(a-1);
}

void main(){
    unsigned n = 100;
    printf( "sum from 1 to %u is %u\n\n", n, sum(n));
}
```

sum from 1 to 3 is 6

3 + sum(2)
2 + sum(1)
1 + sum(0)
return 0;

\[ \text{sum(3)} = 3 + \text{sum(2)} = 3 + 2 + \text{sum(1)} = 3 + 2 + 1 + \text{sum(0)} = \ldots = 6 \]
More practice with recursive functions

```c
#include <stdio.h>

unsigned foo(unsigned a, unsigned b) {
    if (!a)
        return 0;
    else
        return b + foo(a-1, b);
}

void main(void) {
    printf("%u\n", foo(3, 4));
}
```

Draw the sequence of recursive calls, as we did in prev. examples. What is printed? What would be a better name for `foo`?
More practice with recursive functions

```c
int mystery(int a, int b) {
    if (b == 0)
        return 0;
    else if (b % 2 == 0)
        return mystery(a + a, b / 2);
    else
        return mystery(a + a, b / 2) + a;
}

void main(){
    printf("%d\n\n", mystery(2, 3));
}
```

What is printed?
More practice with recursive functions

```c
int mystery(int a, int b) {
    if (b == 0)
        return 0;
    else if (b % 2 == 0)
        return mystery(a + a, b / 2);
    else
        return mystery(a + a, b / 2) + a;
}

void main() {
    printf("%d\n\n", mystery(2, 4));
}
```

What is printed?
More practice with recursive functions

```c
#include <stdio.h>

int fib(int n) {
    if (n == 0 || n == 1) {
        return n;
    } else {
        return fib(n-1) + fib(n-2);
    }
}

void main(void) {
    printf("%i\n", fib(2));
}
```

What is printed?
More practice with recursive functions

```c
#include <stdio.h>

int fib(int n){
    if (n==0 || n==1)
        return n;
    else
        return fib(n-1)+fib(n-2);
}

void main(void) {
    printf("%i\n", fib(3));
}
```

What is printed?
Inline functions

```c
int fun(int a, int b) {
    return (a-b) + (a * b);
}

inline int fun(int a, int b) {
    return (a-b) + (a * b);
}
```

Generally, the compiler literally copies the code of the inline function into the body of any function that calls it (e.g. main)
Inline functions

```c
int fun(int a, int b) {
    return (a-b) + (a * b);
}

inline int fun(int a, int b) {
    return (a-b) + (a * b);
}
```

Trade-off between speed and memory!
MS Visual Studio 2012 only allows the keyword `inline` in C++ programs. In C, we have to use `_inline` instead.

```c
#include <stdio.h>

_inline fun(int a, int b);

void main()
{
    printf("Result: %d\n\n", fun(2, 3));
}

_inline fun(int a, int b){
    return (a-b) + a*b;
}
```
To do in notebook for next time:

• Read and take notes pp.109-115
• Answer end-of-chapter quizzes 7 – 10
• Answer end-of-chapter exercises 3, 4, 5
Homework for Chapter 5, due Wed, Feb.10:
• Exercises 7, 8, 11
  • Hint for 11: Multiplication is repeated addition!
• Not from text: Write a function that takes as arguments the radius and height of a cone and returns its volume. Test function with r = 2.5 and h = 3.5.

Every time a function is required, you have to also write the main function, and call your function at least once!
Capture a screenshot of both code and output.
Give instructor one printout with the entire homework.