Functions: Solutions

Using functions

I am not going to keep repeating the functions in the solutions. I will only show a complete main() function. The main function will call the functions, which are resident above the main.

Question 1

All I have to do here is get the user's input in a do-while loop, with the condition being that x is not a perfect square. Note: I'm using a do-while for a reason here. If I used a plain old while loop, I'd have to initialize x to be an integer that is not a perfect square. Typically, we don't want to write code where we have to pre-compute values, so to make life much easier, I will initialize to 0 (which in the particular function implementation would NOT be a perfect square anyway, but that's beside the point!) and get a number from the user before checking.

```
int main()
{
    int x = 0;
    do
    {
        cout << "Enter a perfect square: ";
        cin >> x;
    }while(!isPerfSquare(x));
    cout << x << "Is a perfect square!";
}
```

Question 2

Now, I have two (at least) ways to do this:

1. I could use cmath's hypot function and then determine if the output is an integer, but that's more difficult. However, since it is still technically a use of a function, that's an acceptable method of solving this problem. I'll demonstrate how to do so below.
2. I can compute the square of the hypoteneuse using my findHypotSquare function, then determine if it is a perfect square. If it is, then the hypoteneus is an integer, and I don't need to work with floating point values.

   int main() { int a = 0, b = 0;

   //strictly speaking, this should be two *positive* integers cout << "Enter two integers: "; cin >> a >>

b;

//THIS IS METHOD 2, not using cmath functions //so my goal is this: to first compute the hypothesis' square, //then to determine if it is, in fact, a perfect square.
//I don't need to use hypot or cmath to solve this problem. int c = findHypotSquare(a,b); if(isPerfSquare(c))

```
  cout << "Yes, the hypoteneuse is an integer.";
```

else

```
  cout << "No, not an integer.";
```

//THIS IS METHOD 1, using cmath (you'll need to include cmath to get this //to work). You may safely ignore this if you want. double h = hypot(a,b); //now, we need to extract the integer part of h, by a cast. int intPart = floor(h); //round down, not up //NOW, we need to remove the integer part from h, and determine if the //remaining number is close to zero, using tolerance double decPart = h - intPart; if(fabs(decPart) < 1e-6) //arbitrary tolerance

```
  cout << "Yes, the hypoteneuse is an integer";
```

else

```
  cout << "No, not an integer.";
```

}

Question 3

The simple way to do this is using the integer functions we gave you. You COULD decide to use cmath, as in example 2 above. Please don't :-)

int main() { //part 3 int s1 = 0, s2 = 0, hyp = 0; cout << "Enter s1, s2, and hypoteneuse, in that order: "; cin >> s1 >> s2 >> hyp;

```
  //first, compute what the square of the hypoteneuse *should* be
  int actualHyp = findHypotSquare(s1,s2);

  //next, compare it to the square of the entered hypoteneuse
  if(actualHyp == hyp*hyp)
      cout << "Yes! We have a right angled triangle!";
  else
```

```
        cout << "No, we do NOT have a right angled triangle.";
```

}

Writing Functions

Question 1

I like to approach this type of question by asking myself what the fundamental units of this code are. Fundamental units are things like getting input or doing a single computation. If I can figure out what they are, then I have my list of functions. So, here's what I would say this program has to be able to do:

1. It must be able to get user input
2. It must be able to print out a space
3. It must be able to print out *some variable number* of spaces
4. It must be able to print out a single character
5. It must be able to print out *some variable number* of that character
6. It must produce a line composed of some number of spaces and some number of characters
7. It must be able to print the pyramid

In this case, we are going to write a LOT of functions, and you may be wondering what the purpose of it all is. After all, when you did this in the assignment or practice, you probably just wrote it out in main, right? One of the biggest benefits of using functions is to organize your code into pieces that make sense to go together. Then, writing each step might be so simple as to be trivial (take a look at the functions printSpace and printSingleChar, below). You may not want to go into *quite* as much detail as I did, but the ideas are all there - the more functions you can break your code into, the simpler each function gets to write, and the more of a chance you have to write simple to understand code.

Writing so many functions is a good and a bad thing in terms of testing. For one thing, I can now test my individual functions without worrying about the rest of them. But the downside is that I have to test *every one of them* to make sure they work.

```
#include <iostream>
using namespace std;
int getNumberLines()
{
    int retVal = 0;
    while(retVal <= 0)
    {
        cout << "How many lines do you want to see?";
        cin >> retVal;
    }
    return retVal;
}
```

```
char getDesiredChar()
{
    char retVal = 'x'; //Arbitrary initialization
    cout << "Which character do you want to see?";
    cin >> retVal; //I don't need to validate this one
    return retVal;
}


/****
These next two functions will seem sort of useless.   They are here because they
allow us to talk about the algorithm in terms like "Print out spaces", rather
than "cout << " " ".

Notice that they are void because they do not return anything, they are purely
display functions
****/
void printSpace()
{
    cout << " ";
}
void printSingleChar(char c)
{
    cout << c;
}

void printSpaceRow(int nSpaces)
{
    for(int i = 0; i < nSpaces; i++)
    {
        printSpace(); //call the function if you have it, do NOT duplicate
        //functionality
    }
}

void printCharRow(char c, int nChars)
{
    for(int i = 0; i < nChars; i++)
    {
        printSingleChar(c); //again, call the function
    }
}

//primarily, we will be printing rows in the pyramid, so why not write this?
void printRow(char c, int nSpaces, int nChars)
{
    printSpaceRow(nSpaces);
    printCharRow(c,nChars);
    cout << endl;
}
```

```
void printPyramid(char c, int nRows)
{
    for(int i = 0; i < nRows; i++)
    {
        int nSpaces = nRows - i;
        int nChars = 2*i + 1;
        printRow(c,nSpaces,nChars);
    }
}

int main()
{
    int nRows = getNumberLines();
    char c = getDesiredChar();
    printPyramid(c,nRows);
}
```

Question 2

Part 1

I didn't specify what to DO with the result of the addition, so it's theoretically up to you. I suggest returning it, though.

Notice: I am choosing not to have an intermediate variable store the result, and instead am just returning it directly. For short functions like this one that's OK, but it's probably best to store things in variables for longer functions.

```
int addTwo(int a, int b)
{
  return a+b;
}
```

Part 2

```
int enterPositiveInt()
{
  int retVal = 0; //I am choosing 0 to be a non-positive number.
  //you may have decided otherwise.

  while(retVal <= 0)
  {
    cout << "Enter a positive integer: ";
    cin >> retVal;
  }
```

```
}
```

Part 3

I am choosing to assume that the user will enter an integer. I am going to choose the sign of the remainder (the output of modulo) to be equal to the sign of the numerator. So -5%7 will be **negative**.

```
int newMod(int num, int den)
{
  int sign = 1;
  if(num < 0)
  {
    num = -num; //make it positive, since we will have to below
    sign = -1;
  }

  //division by repeated subtraction is not elegant, but it is simple!
  while(num > den)
  {
    num -= den;
  }

  return sign*num;

}
```

Part 3: the string library

I'm going to post all of the functions at once, including a main that tests them. But before I do, I want to discuss how I approached writing the entire library. I noticed that there were a few useful functions that would simplify things. For instance, I used strlen extensively in many of my functions. I also noticed that some functions could be used to simplify others, even though they were only used once. In particular, I used atoi to simplify the implementation of atod, since a double can be thought of as two integers separated by a decimal point (see below), and I used findChar to simplify the implementation of findFirstOf.

When writing code that includes functions, think about how you can best use each function to simplify the others. If there are repeated or tricky calculations, break the code into functions and work on each piece separately.

One last thing to note: the function atod is a difficult one to write. It will take careful design. If you find yourself having trouble with it, you can safely ignore it and study my solution.

Here are the solutions. They are very long.

```
#include <iostream>
#include <cmath>
using namespace std;

//I need strlen, it's useful
int strlen(char str[])
{
int i = 0;
while(str[i] != 0)
i++;
//we can return i for the length of the string, or i-1 for the biggest
//index in the string.  I choose to return i itself.
return i;
}

//strcmp is actually fairly complicated.  let's consider each case
int strcmp(char str1[], char str2[])
{
//first, test length
if(strlen(str1) < strlen(str2))
return -1;

//Now, we need to test equality. This bit is tricky.  First, notice that
//we are assured that strlen(str1) >= strlen(str2), so a while loop can be
//used looking for the end of str2
int i = 0;
bool exactlyEqual = 1;
while(str2[i] != 0 && exactlyEqual == true)
{
if(str1[i] != str2[i])
{
  exactlyEqual = false;
}
i++;
}

if(exactlyEqual == false)
{
//They aren't equal, but which one comes first?
if(str1[i-1] > str2[i-1])
  return -1; //str1 comes first, alphabetically
else
  return 1; //str2 comes first
}

/****
OK, we aren't done yet!  Consider this: if str1 = "ABC" and str2 = "A",
then we'd be out of the while loop but exactlyEqual would still be true!

This case happens when strlen(str2) < strlen(str1), so we can just
check for that.  In this case, we decide that str2 comes before str1,
```

```
alphabetically.
****/
if(strlen(str2) < strlen(str1))
{
return 1;
}
else //they legitimately are equal!
return 0;
}

int atoi(char str[])
{
//it's up to us to deal with bad input, so we are dealing with it by
//ignoring it.  We assume that all input strings hold valid integer
//representations with one exception: we are going to allow the user to put
//in leading zeros, like "0001000".  We are going to remove those leading
//zeros, since it will help us with atod later

/****
It's actually unintuitive how to do this.  The simplest way is to find
the length of the string, then work backwards.
****/

//first, we might get a negative.
int startIndex = 0;
int sign = 1;
if(str[0] == '-')
{
sign = -1; //this will be multiplied in at the end
startIndex = 1; //this lets us skip that character
}

//now deal with leading zeroes
while(str[startIndex] == '0') //remember, the zero character, not the number
startIndex++;

int len = strlen(str);
int retVal = 0;
for(int i = startIndex; i < len; i++)
{
retVal = retVal*10 + (str[i] - 48);
/****
  huh? If the numbers 0 through 9 are represented by the numerical
  values 48 through 57, then if we subtract 48 from the character, we
  end up with the integer.  Watch:

  if '7' is represented by the number 55, then 55-48 = 7, the integer
  we want
****/
}
return sign*retVal; //remember, it may be negative!
}
```

```
//Ahh atod. This function is difficult if you try to do it before you write
//atoi, but it is far simpler if you have atoi first.  Still hard though...
double atod(char str[])
{
/****
we are going to do this in stages. First, extract the part before the
decimal.  This is an integer, so we will read it using atoi.

Next, extract the part after the decimal, if it exists.  If it does, then
this will ALSO be treated as an integer. We'll convert it.

Finally, take that last integer and divide it by 10 raised to the power
of however many digits we have after the decimal that are nonzero.
****/

//in order to extract, we will use a temporary character array.  Integers
//are at most 9 digits, so:
char tmp[10] = {0};
int startIndex = 0;
double sign = 1;
if(str[0] == '-')
{
sign = -1;
startIndex = 1;
}
//For this, we don't need to worry about leading zeros, atoi will deal with
//that
int intPart = 0;
int i = startIndex;
int j = 0;
while(str[i] != '.' && str[i] != 0 && i < 10)
{
tmp[j] = str[i];
i++;
j++; //need two, since we might start at 1
}
//convert tmp
intPart = atoi(tmp);

if(str[i] == '.')
{
i++; //increment to get away from the decimal
//hard case: extract the integers. We can re-use tmp
j = 0;
while(str[i] != 0 && j < 10)
{
  tmp[j] = str[i];
  j++;
  i++;
}
//now, how many digits is stored in decPart?  We need to search tmp
```

```
//backwards for the first non-zero character
int numDigits = strlen(tmp);
j = numDigits-1;
while(tmp[j] == '0' && j >= 0)
{
  numDigits--;
  tmp[j] = 0;
  j--;
}
//convert it again
double decPart = (double)atoi(tmp);


cout << decPart << endl;
decPart = decPart/pow(10.0,1.0*numDigits-1);
cout << decPart << endl;
//add them together and return
return sign*((double)intPart + decPart);
}
else
{
return (double)intPart; //easy case - no decimal point
}


}

//findchar is much simpler than atod.
int findChar(char str[], char c)
{
//initialize the return value to -1, because if we do not find the
//character, we return a negative value
int foundIndex = -1;

//We can either get the string's length using strlen, or find it ourselves.
//I'm going to use strlen for clarity.
int len = strlen(str);

//we need to indicate somehow that we found the character, so that we don't
//keep looping, so there are two conditions here: either we run out of
//string, or we find the character.  If we find the character, foundIndex
//will not be -1 anymore
for(int i = 0; i < len && foundIndex == -1; i++)
{
if(str[i] == c)
  foundIndex = i; //this will break us out of the loop
}

//just return it
return foundIndex;
}
```

```
/*
findFirstOf is hard if you do not use findChar, but it is much easier if you
do.  One thing to note: the way the specification is written, the following
must happen:

Let's assume we are searching the string "Hello" for characters in the
string "olleH".  We should see the output value 4, since 'o' was the first
character in the search string found.  That is, although the search string
contains the character 'H', which is the first character in the string we
are searching, we should still return 4, since 'o' is the first character
we will be looking for.

The above paragraph simplifies down to this: just search the first string
for all characters in the search string, starting at the first.  *As soon
as you find ANY character in the search string, just return.*  It makes
life a LOT easier.

*/
int findFirstOf(char str1[], char str2[])
{
//let's get the string length for str 2
int len = strlen(str2);

//a return index, as above, initialized to -1
int retIndex = -1;

//we are looping over str2, but also leaving the loop if we find a character
for(int i = 0; i < len && retIndex == -1; i++)
{
//if we find the character, retIndex will be anything but -1, otherwise,
//it will stay -1 and we'll keep looping
retIndex = findChar(str1, str2[i]);
}

//return t
return retIndex;
}


int main()
{
char str1[] = "ABC";
char str2[] = "ABC";
cout << strlen(str1) << endl;
cout << strlen(str2) << endl;
cout << strcmp(str1,str2) << endl;

//A particularly difficult value for atoi would be a negative number with
//trailing zeros, like -00010900.  This value is negative, includes the full
//range of characters, and has two trailing zeros and three leading zeros
char numVal[] = "-00010900";
cout << atoi(numVal) << endl;
```

```
char numVal2[] = "-10.0012500";
cout << atod(numVal2) << endl;

//let's search our existing strings for various characters, to test findChar
cout << findChar(numVal, 'A') << endl; //I should see -1
cout << findChar(numVal, '0') << endl; //I should see 1
cout << findChar(numVal, '-') << endl; // I should see 0;
cout << findChar(str1,'C') << endl; //I should see 2, good test of strlen

//now we test findFirstOf.  We'll need a new string
char str3[] = "Helloe";
char str4[] = "AbDEe";
cout << findFirstOf(numVal,str3) << endl; //I should see -1
cout << findFirstOf(str3, str4) << endl; //I should see 1
}
```