Computer Science 3-4
Problem Set #6: Tic-Tac-Toe

This week we aren't going to introduce too much new syntax to work with, instead we are going to work on a more complicated program….in this case a simple two-player version of tic-tac-toe. But before we get to the problem, here are a couple of things that might come in handy:

A couple of new types that might be useful

There are times when we just want to make a variable that just holds *true* or *false*, we can do this by declaring a variable as the **bool** type (which is short for Boolean).

```
bool over21 = true;    // this initializes the variable and sets the default to true
```

One of the reasons that we use the bool type declaration is to make sure we don't accidentally modify our variable through arithmetic. Most of the time we are going to use bool variables as "flags" to keep track of things in our programs that are either "on" or "off" (like a player's turn or if something has been clicked).

Sometimes we want to make a variable a specific value and make sure that the value never changes…for instance if we want to set a specific value for pi. We can do this with the **const** type.

```
const pi = 3.14159;  // to change the value of pi you must change it here or you will get an error
```

Note that if you don't assign a value when you declare a const variable, you won't be able to assign it a value later on!

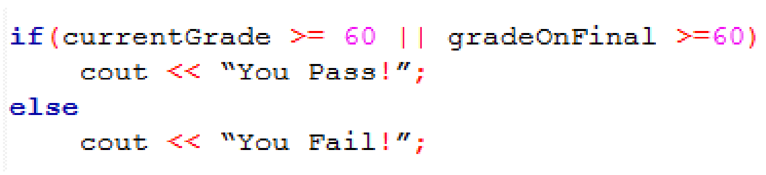
The last type for this week is the **char** type. This type actually allows us to do quite a lot (we'll explore this more as the term goes on), but for now the important aspect of this is that it allows us to store text in a variable (some of you have been using the *string* command for this, there are advantages to both). For instance:

```
char marksTheSpot = 'X'; // This will hold a single letter, note the single quotes
char imNumberOne = '1'; //  This holds the character of '1'
```

Note that for right now we can't use the char type to store a block of text (like your name), but we'll get there shortly…if you have a desperate need to store a line of text, use *string* instead.

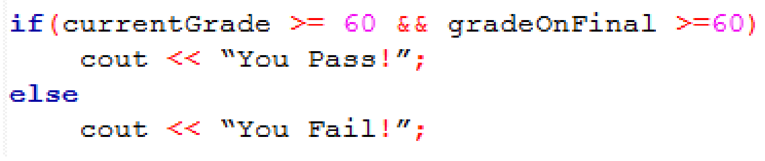
Comparing more than one thing

Sometimes we want to check to see if more than one thing is true in and if statement or we want to take action if "this" *or* "that" is true. We can do that with the **&&** and || commands ("and" and "or" respectively). Here is an example of how we use them:



```
if(currentGrade >= 60 || gradeOnFinal >=60)
     cout << "You Pass!";
else
     cout << "You Fail!";
```

This example says you pass this class if you either earned better than a 60 on all the current work in the class **or** you get at least a passing grade on the final.

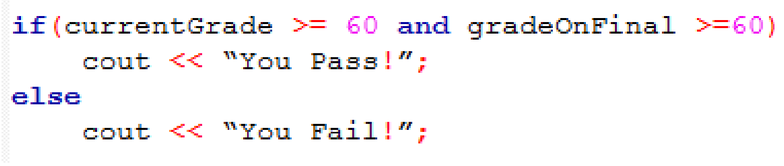


```
if(currentGrade >= 60 && gradeOnFinal >=60)
     cout << "You Pass!";
else
     cout << "You Fail!";
```

This example says you need both a 60 on the coursework **and** a 60 on the final to pass!

Note that you need to have the complete expression you are trying to test (grade > 60) then the && or || then the second expression….if you don't do this you'll get funny results or errors! Also note that you can chain as many and/or combos as you want, but be careful because they can be tricky to troubleshoot if you go too crazy!

A final point about and/or and the differences between compilers. We've talked a little bit about how C++ needs to be compiled to run and that is a language that evolved from C, but we haven't talked about all the differences in the actual compilers. A compiler is simply a piece of software that interprets the C++ language and converts it into low-level instructions the computer can understand. There have been many different C++ standards over the years (basically think of it as slight revisions to make things work better) and similarly there are many different styles of compilers (this is a common senior level assignment in college) but at this point it doesn't make sense to really talk about their subtle differences. Our Code::Blocks IDE has a nice feature in that you can select which compiler to use (remember all that setup we did to get the MinGW/GNU compiler on your home comp?) and because of that we don't always have to use the most traditional C++ code. For instance, instead of using && and ||, we can actually use the natural language and/or in our test conditions:



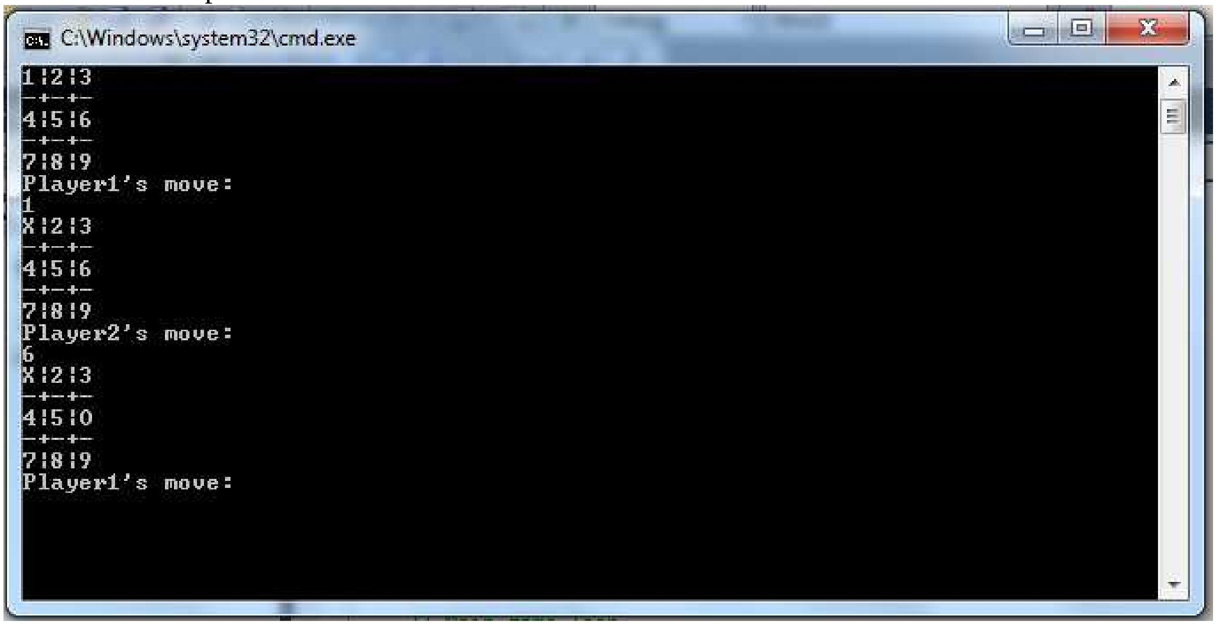
```
if(currentGrade >= 60 and gradeOnFinal >=60)
     cout << "You Pass!";
else
     cout << "You Fail!";
```

Most modern languages have made this switch, so a lot of the newer C++ compilers made this switch as well. If you want to be old-school stick with &&, but know that "and" works and is a lot easier to use while you are learning!

Tic-Tac-Toe-Task

You are going to make a program that lets two players (no computer AI…yet) play against each other in a game of tic-tac-toe. This is as much a test of your ability to plan out your program in advance as it is of your actual programming skill as there is quite a lot to keep track of!

Here is an example of what the screen will look like:



```
1|2|3
-+-+-
4|5|6
-+-+-
7|8|9
Player1's move:
1
X|2|3
-+-+-
4|5|6
-+-+-
7|8|9
Player2's move:
6
X|2|3
-+-+-
4|5|O
-+-+-
7|8|9
Player1's move:
```

1. **Plan your program! –** Type up an outline of what your program is going to look like without writing any code…use normal language and pseudocode. This will have three parts: The first is the flow of the game which will focus on how/when certain things occur (keeping track of turns, how to check to see if the game is over or not, etc.), the second will focus on how you plan to keep track of the board and other variables you might need, the third is how you plan to test your program to make sure everything works. Make sure to use the example above to help you! This should be no more than a page, but remember, this is an outline to help you with the logic of your game so be specific! Turn this in!

2. **Enact the plan! –** Program your version of the tic-tac-toe game, when you have finished call the instructor over and we'll do some testing to make sure it is ready for prime time!

3. **Challenge –** If you finish quickly, take some time to spruce up the game a bit to make it suit your own style!