Step 5: Review and Practice

This assigment is done in class on (monday)02-10-14 AND (friday)02-07-14.

This assignment is due for monday class 02-10-14 AND for the firday class 02-07-14.

Getting Started

By now you should be all logged on. You should still have a cse251 directory in your home directory from last week. Please change into that directory.

You should be able to do a pwd and see that you are in the cse251 directory at this time.

Remember, start gedit with this command line:

```
gedit &
```

Create a new program called fib.c with the following code:

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

/*
 * Program to compute Fibonacci numbers
 */

int main()
{
    int fn1 = 1;         /* F(n-1) during the loop */
    int fn2 = 0;         /* F(n-2) during the loop */
    int f;               /* Computed Fibonacci number */
    int i;               /* Loop counter */

    /* Print the first two Fibonacci numbers */
    printf("0\n");
    printf("1\n");

    /* Print 10 Fibonacci numbers */
    for(i=0;  i<10;  i++)
    {
        f = fn1 + fn2;  /* Compute the number */
        printf("%d\n", f);

        /* Update F(n-1) and F(n-2) */
        fn2 = fn1;
        fn1 = f;
    }
}
```

Be sure it compiles and runs. The commands to compile and run are, of course:

```
gcc -o fib -lm fib.c
./fib
```

1. A Quick Exercise

Modify the program fib.c so that it outputs like this:

```
cbowen@ubuntu:~/cse251$ ./fib
F(0) = 0
```

```
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55
F(11) = 89
```

This is telling me which Fibonacci number this is. This will take only a few changes in your code.

Playing Computer

First, create a new program playing.c and paste in this code. Be sure you can compile and run this program.

```
#include <stdio.h>

int main()
{
    printf("Playing Computer\n");
}
```

For each of the following exercises, I'll give you a short program. You are to "play computer". This means that you execute the program by hand and determine what the output is. Then, paste the code into playing.c, replacing the existing code, and see if you are correct. We will not count off if your computed answer is incorrect, provided you provide an explanation as to what you did wrong when you hand-executed the code.

You play computer by simply keeping track of which line you are at in the code. Keep track of your hand calculation, the actual calculation, and, if they are different, an explanation of why they are different (i.e, what you did wrong) in a text file that you will submit at the end of the exercise. Call the file worksheet.txt.

The += and -= operators are shortcuts for code like a = a + 7; and b = b - 3; Some examples:

i += 10;   equivalent to   i = i + 10;
j -= 20;   equivalent to   j = j - 20;
z += 0.5;   equivalent to z = z + 0.5;

The ++ and -- operators are shortcuts for code like a = a + 1; and b = b - 1; Some examples:

i++;   equivalent to i = i + 1;
j--;   equivalent to j = j - 1;

```
#include <stdio.h>

int main()
{
    int i = 1;
    int sum = 0;

    while(i < 5)
    {
        sum += i;
        i++;
    }

    printf("sum=%d\n", sum);
}
```

```c
#include <stdio.h>

int main()
{
    double q = 1;
    int i;

    for(i=1;  i<= 5;  i++)
    {
        q += i * i / 2.0;
    }

    printf("q=%f\n", q);
}
```

```c
#include <stdio.h>

int main()
{
    double q = 1;

    do
    {
        q = q * 2;
    } while(q < 0);

    printf("q=%f\n", q);
}
```

```c
#include <stdio.h>

int main()
{
    int i;
    int j;

    for(i=0;  i<11;  i++)
    {
        if(i < 5)
        {
            for(j=0;  j<=i;  j++)
                printf("+");
        }
        else
        {
            for(j=0;  j<=(10-i);  j++)
                printf("+");
        }

        printf("\n");
    }

    printf("---\n");
}
```

2. Moon Lander Game

We're going to write a simple game. It will simulate a lunar lander. The lander starts at an altitude of 100 meters and a velocity of 0 meters per second. For each second of flight you have to decide how much fuel to burn. One kilogram of fuel burned in one second will generate an acceleration of 1.5 meters per second per second. You also have gravity of 1.63 meters per second per second dragging you down to the surface. Your engine starts with 100 kilograms of fuel and can burn no more than 5 kilograms per second.

Your game will ask how much fuel to burn in one second. It will then update the velocity and altitude appropriately. You live if you manage to land with a speed less than or equal to 3 meter per second. Otherwise you die a horrible, gruesome death. You'll need to take off later, so you should try to save as much fuel as you can.

Create a new program called moon.c and paste this code into it:

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>

/*
 * Simple lunar lander program.
 * By:  <insert name here>
 */

int main()
{
    printf("Lunar Lander - (c) 2012, by <insert name here>\n");

}
```

Put your name at the two indicated spots in the program.

We are going to build this program incrementally, meaning we create it a little bit at a time. This is usually the best way to write a computer program. We'll write some code, then we'll test that the code we just wrote compiles and works before we move to the next bit of code. I will put in the following icon:

When you see that icon, that means you should compile and run your program. Sometimes this is to ensure it compiles okay only. At other times you'll check to see that code you just entered is working.

So, start by compiling and running this program. It will only display the message for now.

Keeping Track of Information

So, what do we need to keep track of? I've taken my description above and highlighted the things we need to keep track of as our program runs:

> We're going to write a simple game. It will simulate a lunar lander. The lander starts at an **altitude** of 100 meters and a **velocity** of 0 meters per second. For each second of flight you have to decide how much **fuel to burn**. One kilogram of fuel burned in one second has the **power** to generate an acceleration of 1.5 meters per second per second. You also have **gravity** of 1.63 meters per second per second dragging you down to the surface. Your engine starts with 100 kilograms of **fuel** and can burn no more than 5 kilograms per second. If you land with a velocity of less than 3 meters per second, you live.

These are the obvious ones, though we'll actually pick a few more later. We keep track of information in a program using variables.

So, add these variables to your program for those items:

```
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */
```

We'll want the user to know some of these things as well. So, add these statements to print the current status and the introductory message:

```
    printf("Altitude: %f\n", altitude);
    printf("Velocity: %f\n", velocity);
    printf("You have %f kilograms of fuel\n", fuel);
```

This should compile and run okay.

When you run this, the output will look something like this:

```
cbowen@ubuntu:~/cse251$ ./moon
Lunar Lander - (c) 2012, by I. M. Genius
Altitude: 100.000000
Velocity: 0.000000
You have 100.000000 kilograms of fuel
```

This has way too many digits after the decimal points. Modify the code so we have two decimal places on altitude and velocity and one for fuel. It should look more like this:

```
cbowen@ubuntu:~/cse251$ ./moon
Lunar Lander - (c) 2012, by I. M. Genius
Altitude: 100.00
Velocity: 0.00
You have 100.0 kilograms of fuel
```

Of course, compile and run this to be sure it is working.

Burn Input

We want to ask the user how much fuel to burn. The simple version of this would be:

```
    printf("How much fuel would you like to burn: ");
    scanf("%lf", &burn);
```

Add that code to the end of the program and ensure it is working.

Of course, this has many flaws. It does not test to ensure the input is not negative. You can't burn negative fuel. It also does not test to ensure you are burning no more than the 5 kilograms or that you are burning more fuel than you have. We can add these tests easily enough. Add this code for these tests:

```
    if(burn < 0)
    {
        printf("You can't burn negative fuel\n");
    }
    else if(burn > fuel)
    {
        printf("You can't burn fuel you don't have\n");
    }
    else
    {
        printf("Burning %.1f kilograms of fuel\n", burn);
    }
```

I left out the test to ensure you are not burning more than 5 kilograms of fuel. I have to leave something for you to do.

First, add this code as it is above and test to be sure it is working.

When testing code like this, be sure you carefully test all of the cases. Enter a negative fuel to be sure that works. Then try a fuel greater than the 100. Watch out for boundary conditions. If you have 5 kilograms of fuel, will it allow you to burn all of it, or only 4.99 kilograms?

Add code to your program to also indicate if the user tries to burn more than 5 kilograms of fuel. Be sure you test that carefully. Try setting the amount of fuel available to 5 temporarily and ensuring you can burn 5 kilograms okay.

Putting Input in a Loop

If the user makes a mistake, we need to ask them to enter the burn amount again. So, we need to put this code into a loop. Here's the program so far:

```
int main()
{
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */

    printf("Lunar Lander - (c) 2012, by I. M. Genius\n");

    printf("Altitude: %.2f\n", altitude);
    printf("Velocity: %.2f\n", velocity);
    printf("You have %.1f kilograms of fuel\n", fuel);

    printf("How much fuel would you like to burn: ");
    scanf("%lf", &burn);

    if(burn < 0)
    {
        printf("You can't burn negative fuel\n");
    }
    else if(burn > 5)
    {
        printf("You can't burn more than 5 kilograms per second\n");
    }
    else if(burn > fuel)
    {
        printf("You can't burn fuel you don't have\n");
    }
    else
    {
        printf("Burning %.1f kilograms of fuel\n", burn);
    }

}
```

You want to think about this question: what needs to be in a loop? One way of looking at this question is: what do I repeat? For each time the user enters an invalid value, I'll want to prompt them again, enter a new value, and test it again. That's this part of the code:

```
int main()
{
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */

    printf("Lunar Lander - (c) 2012, by I. M. Genius\n");
    printf("Altitude: %.2f\n", altitude);
    printf("Velocity: %.2f\n", velocity);
    printf("You have %.1f kilograms of fuel\n", fuel);

    printf("How much fuel would you like to burn: ");
    scanf("%lf", &burn);

    if(burn < 0)
    {
        printf("You can't burn negative fuel\n");
    }
    else if(burn > 5)
    {
        printf("You can't burn more than 5 kilograms per second\n");
    }
    else if(burn > fuel)
    {
        printf("You can't burn fuel you don't have\n");
    }
    else
    {
        printf("Burning %.1f kilograms of fuel\n", burn);
    }

}
```

Highlight that code in gedit and hit tab to tab it in. That's a way to remember what you're going to put into the loop:

```
int main()
{
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */

    printf("Lunar Lander - (c) 2012, by I. M. Genius\n");

    printf("Altitude: %.2f\n", altitude);
    printf("Velocity: %.2f\n", velocity);
    printf("You have %.1f kilograms of fuel\n", fuel);

        printf("How much fuel would you like to burn: ");
        scanf("%lf", &burn);

        if(burn < 0)
        {
            printf("You can't burn negative fuel\n");
        }
        else if(burn > 5)
        {
            printf("You can't burn more than 5 kilograms per second\n");
        }
        else if(burn > fuel)
        {
            printf("You can't burn fuel you don't have\n");
        }
        else
        {
            printf("Burning %.1f kilograms of fuel\n", burn);
        }

}
```

Now, what type of loop do we want? We want to continue looping until the user enters a valid value. So, this looks like a good place to use a Boolean flag. Create this new variable:

```
    bool valid;             /* Valid data entry flag */
```

We can now create a loop around the code that will continue to loop as long as the data entered is not valid (!valid). Since we always want to execute the body of the loop once, this is a good place to use a do loop. I'm going to write the highlighted code with a do loop. We call this "putting the code into a loop."

```
    do
    {
        printf("How much fuel would you like to burn: ");
        scanf("%lf", &burn);

        if(burn < 0)
        {
            printf("You can't burn negative fuel\n");
        }
        else if(burn > 5)
        {
            printf("You can't burn more than 5 kilograms per second\n");
        }
        else if(burn > fuel)
        {
            printf("You can't burn fuel you don't have\n");
        }
        else
        {
            printf("Burning %.1f kilograms of fuel\n", burn);
        }
    } while(!valid);
```

This will compile, but don't try to run it yet.

The variable valid is a flag. It's raised to indicate the entry is valid and lowered to indicate it is not. We're not told it one way or the other, yet. One idea is to lower the flag at first and only raise it if the entered data is valid. When I say "lower it first", I mean add this line as the first line of the body of the do loop:

```
        valid = false;   /* Assume invalid until we know otherwise */
```

This goes here:

```
    do
    {
        valid = false;   /* Assume invalid until we know otherwise */

        printf("How much fuel would you like to burn: ");
        scanf("%lf", &burn);
```

When you compile and run this, it should keep asking for the burn over and over again, even if you entered a valid value. Do you see what that happens?

If we enter a valid burn value, we need to raise the flag to indicate our burn data is hunky dory. Add this line to the else where we indicate we are burning the fuel:

```
            valid = true;
```

This goes here:

```
        else
        {
            printf("Burning %f kilograms of fuel\n", burn);
            valid = true;
        }
```

This should work and you should be able to enter valid and invalid values with it working right in each case. Be sure you test thoroughly at this time.

Updating the Velocity and Altitude

Now we can compute how the velocity and altitude change after the burn. Each second we will add the force of gravity times the time period (one second, so that is easy) and add the power times the amount of fuel burned times the time period. You can compute that using this simple statement:

```
    velocity = velocity + g + power * burn;
```

We, of course, do this after we have successfully entered the amount of fuel to burn. This should compile okay.

The other two things we need to update are the altitude and amount of remaining fuel. I'm going to use those += and -= operators to update those:

```
    altitude += velocity;
    fuel -= burn;
```

Temporarily add these statements to the end of your program to test that these equations are working:

```
    printf("Altitude: %.2f\n", altitude);
    printf("Velocity: %.2f\n", velocity);
    printf("You have %.1f kilograms of fuel\n", fuel);
```

If you enter a 0, the velocity should increase to -1.63 (negative is falling) and the altitude drop to 98.37 (100 - 1.63). Here are a few other values I tried:

```
How much fuel would you like to burn: 1
Burning 1.0 kilograms of fuel
Altitude: 99.87
Velocity: -0.13
You have 99.0 kilograms of fuel
```

```
How much fuel would you like to burn: 3
Burning 3.0 kilograms of fuel
Altitude: 102.87
Velocity: 2.87
You have 97.0 kilograms of fuel
```

```
How much fuel would you like to burn: 5
Burning 5.0 kilograms of fuel
Altitude: 105.87
Velocity: 5.87
You have 95.0 kilograms of fuel
```

Delete the extra print statements you put at the end of your program to test the results.

It is very common when programming to temporarily add a printf statement that prints out what a value is at some point in a program. This is the most common means of simple debugging. It's a way to tell what is going on inside your program and test that the value at some point is what you expected or figure out if it is not.

Putting the Whole Process in a Loop

Right now you run the program and it does one second and exits. We want to repeat this process over and over again. That again means putting code into a loop. So, what goes in the loop this time? Every second we want to:

- Output the current altitude and velocity
- Output how much fuel we have
- Get the amount of fuel to burn from the user
- Compute a new altitude, velocity, and fuel amount

In my program right now, that's all of this highlighted code:

```
int main()
{
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */
    bool valid;             /* Valid data entry flag */

    printf("Lunar Lander - (c) 2012, by I. M. Genius\n");

    printf("Altitude: %.2f\n", altitude);
    printf("Velocity: %.2f\n", velocity);
    printf("You have %.1f kilograms of fuel\n", fuel);

    do
    {
        valid = false;   /* Assume invalid until we know otherwise */

        printf("How much fuel would you like to burn: ");
        scanf("%lf", &burn);

        if(burn < 0)
        {
            printf("You can't burn negative fuel\n");
        }
        else if(burn > 5)
        {
            printf("You can't burn more than 5 kilograms per second\n");
        }
        else if(burn > fuel)
        {
            printf("You can't burn fuel you don't have\n");
        }
        else
        {
            printf("Burning %.1f kilograms of fuel\n", burn);
            valid = true;
        }
    } while(!valid);

    velocity = velocity + g + power * burn;
    altitude += velocity;
    fuel -= burn;

}
```

Yep, nearly the entire program. It's common that nearly an entire program would be in a loop. Go ahead and tab this in like we did before.

So, what type of loop do we want? You might think to yourself: when do we keep entering burn data? The answer: while the altitude is greater than zero. That should tell you all you need to know. Just put this code in a loop that loops while(altitude > 0):

```
int main()
{
    double altitude = 100;  /* Meters */
    double velocity = 0;    /* Meters per second */
    double fuel = 100;      /* Kilograms */
    double power = 1.5;     /* Acceleration per pound of fuel */
    double g = -1.63;       /* Moon gravity in m/s^2 */
    double burn;            /* Amount of fuel to burn */
    bool valid;             /* Valid data entry flag */

    printf("Lunar Lander - (c) 2012, by I. M. Genius\n");

    while(altitude > 0)
    {
        printf("Altitude: %.2f\n", altitude);
        printf("Velocity: %.2f\n", velocity);
        printf("You have %.1f kilograms of fuel\n", fuel);

        do
        {
            valid = false;   /* Assume invalid until we know otherwise */

            printf("How much fuel would you like to burn: ");
            scanf("%lf", &burn);

            if(burn < 0)
            {
                printf("You can't burn negative fuel\n");
            }
            else if(burn > 5)
            {
                printf("You can't burn more than 5 kilograms per second\n");
            }
            else if(burn > fuel)
            {
                printf("You can't burn fuel you don't have\n");
            }
            else
            {
                printf("Burning %.1f kilograms of fuel\n", burn);
                valid = true;
            }
        } while(!valid);

        velocity = velocity + g + power * burn;
        altitude += velocity;
        fuel -= burn;
    }
}
```

This should work and you should be able to play the game.

The player needs to know if they won or not. I added this code to the end of the game. Note that this code is NOT in the loop, since it is only done after the player is finished landing:

```
    printf("You landed with a velocity of %.2f\n", velocity);
    if(fabs(velocity) > 3)
    {
        printf("Your next of kin have been notified\n");
    }
```

Again, this should work and you should be able to play the game.

A few remaining tasks I would like you to do:

- Add a way to keep track of time, so that when the player is done, they know they landed in how many seconds. Note that each iteration of the loop is one second, so this is just an integer counter.
- Output the amount of fuel remaining at the end of the game.

Try playing your game. In the comment block at the beginning of the program, indicate the best landing you managed to make in seconds and the amount of fuel remaining. The best landing is the one with the most fuel remaining. Your comment block should look something like this:

```
/*
 * Simple lunar lander program.
 * By:  I. M. Genius
 * Best landing: Time = 15 seconds, Fuel = 85.0, Velocity = -1.95
 */
```

And We Are Done!

Go to http://www.cse.msu.edu/handin. Select CSE 251 and Step 5 and hand in the files moon.c and worksheet.txt. Then we are done for today.

CSE 251