Step 6: Functions

This assigment is done in class on (monday)02-17-14 AND (friday)02-14-14.

This assignment is due for monday class 02-17-14 AND for the firday class 02-14-14.

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 combi.c with the following code:

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

/*
 * Name : <Insert name>
 * Program to determine combinations
 */

int main()
{
    int n;
    int k;
    int b;

    printf("Input n: ");
    scanf("%d", &n);
    if(n < 1)
    {
        printf("Must be greater than zero\n");
        exit(1);
    }

    printf("Input k: ");
    scanf("%d", &k);
    if(k < 0 || k > n)
    {
        printf("Must be between 0 and %d inclusive\n", n);
        exit(1);
    }

}
```

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

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

1. Combinations

If I give you a deck of playing cards, how many possible hands of five cards could we draw? It's a bunch, I know, but how would you compute that result? This type of problem is solved using the binomial coefficient:

$$\binom{n}{k}$$

What this equation means is: "for n unique items, how many subsets of size k can be selected?" One definition of the binomial coefficient is this:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

It should be pretty easy to write a computer program to compute this. After all, we've computed factorials before. But, it took a loop to do that computation. Do we want to create that loop three times? A better way is to create a factorial function. Add this code to the end of combi.c (after the closing curly brace for the main function):

```
/* Factorial Function */
int Factorial(int n)
{
    int f = 1;
    int i;
    for(i=1;  i<=n;  i++)
    {
        f *= i;
    }
    return f;
}
```

 Be sure this compiles and runs okay. It won't do anything, yet, though.

This is the *definition* of a factorial function. A function definition tells what the function does. This function has one *parameter* named n of type int. That means one integer can be passed to the function. The function is *named* Factorial. It *returns* an integer. Inside the curly braces, we have the *implementation* of the function and a return statement that returns the result to a calling program.

To use a function it also has to be declared. Add this line of code to the beginning of your program after the header files are included and before the main function:

```
int Factorial(int n);
```

 This tells code that follows what our function looks like. It's just the first line of the function without a body. I just copy and paste to make the declarations.

Now we'll test our function to see if it works. Add this code to the end of the main function:

```
    printf("n! = %d\n", Factorial(n));
```

 This is a function invocation. You should be able to run your program and have it compute factorials of the value you enter for n.

> This won't work for values in excess of 15 or so. The reason is that the numbers get too large for the integer type and it overflows, so the results are incorrect. We won't be able to use this result for our card problem, but we'll come up with another solution for that problem.

Now we can put in the code to compute the binomial coefficient. Delete the printf statement you just put in and replace it with:

```
    b = Factorial(n) / (Factorial(k) * Factorial(n - k));
    printf("%d items taken %d ways is %d\n", n, k, b);
```

 This should work and compute the binomial coefficient okay. As an example, 5 taken 2 ways is 10. If I have the letters a, b, c, d, and e, the possible combinations of two (without regard to order) are: ab, ac, ad, ae, bc, bd, be, cd, ce, de (10).

Here is our combi.c program up to this point with these three items clearly indicated by comments:

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

/*
 . 1. Function declaration.
 . This tells the compiler about our function.
 . It must match the first line of the function definition below.
 */

int Factorial(int n);

/*
 . This is a function definition for a function called main that
 . takes no arguments and returns an integer. You've been writing
 . this function all along.
 */

int main()
{
    int n;
    int k;
    int b;

    printf("Input n: ");
    scanf("%d", &n);
    if(n < 1)
    {
        printf("Must be greater than zero\n");
        exit(1);
    }

    printf("Input k: ");
    scanf("%d", &k);
    if(k < 0 || k > n)
    {
        printf("Must be between 0 and %d\n", n);
        exit(1);
    }

    /*
     . 3. Function invocations.
     . Here are function invocations for Factorial. There are
     . Three of them because the binomial coefficient formula
     . has three factorials in it.
     */

    b = Factorial(n) / (Factorial(k) * Factorial(n - k));
    printf("%d items taken %d ways is %d\n", n, k, b);
}

/*
 . 2. Function definition.
 . This is the definition of the Factorial function.
```

```
 . This is where you tell what the function does.
 */

int Factorial(int n)
{
    int f = 1;
    int i;
    for(i=1;  i<=n;  i++)
    {
        f *= i;
    }
    return f;
}
```

Now we know how to create a function. Create a new function to compute the binomial coefficient. It should have a declaration like this: (Note: Keep the Factorial function in your code, and use it to compute the Binomial Coefficient)

```
int Binomial(int n, int k);
```

Add the code for the Binomial function to your program and use this function to compute the Binomial Coefficient. Your invocation should look like this:

```
    b = Binomial(n, k);
    printf("%d items taken %d ways is %d\n", n, k, b);
```

> Notice how you have a function that you wrote that calls another function that you also wrote (well, entered). This allows us to break a program into building blocks that we can test and then put together to make more complex programs. This is one of the beauties of functions in a programming language.

Remember, there are three things you need for a function:

1. The function definition. This is the implementation of the function.
2. The function declaration. This tells C about your function so it can use it.
3. The function invocation. This is where you "call" your function.

When done turn in combi.c via http://secure.cse.msu.edu/handin/

2. Recursive Functions and Practice

First, copy your program combi.c as a new program combi1.c. This is so we can compare results to the first version.

A problem with this program is that it can't answer the question: how many 5 card hands are there in a deck of cards? The reason is that it uses integer math, which is limited to numbers from -2147483648 to 2147483647 by the precision of the representation in memory. But, 52! is much, much larger than that. I plugged it into my calculator and it gave me: 8.065817517094387857166063685640 4e+67. So, the way we have been computing the binomial coefficient will fail unless we use a much larger number representation.

Fortunately, there are other ways to compute the binomial coefficient that don't require doing factorials on large numbers. For example, there is a recursive definition of the binomial coefficient:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

This is something we can implement as a recursive function, since Binomial(n, k) = Binomial(n-1, k-1) + Binomial(n-1, k). But, if we just did this simple function it will fail. Paste the following code into your program, replacing your Binomial function:

```
int Binomial(int n, int k)
{
```

```
    return Binomial(n - 1, k - 1) + Binomial(n - 1, k);
}
```

 This should fail with a Segmentation fault. To see why, add this printf statement to the program:

```
int Binomial(int n, int k)
{
    printf("n=%d k=%d\n", n, k);
    return Binomial(n - 1, k - 1) + Binomial(n - 1, k);
}
```

 This should give you an idea of what is going on. You'll have to hit Ctrl-C to get it to stop. Then remove just the printf statement.

When we write a recursive function, any function that calls itself, we need to have a base case, some case that does not call itself. There are two base cases here where we know the answer. When we try to pick n items n ways, there is only one possible subset, right? So:

$$\binom{n}{n}=1$$

This means that if k == n, the result is one. We can do that with an if statement.

The other case is if we take any number of items 0 ways there is also only one possible subset, the empty set. So:

$$\binom{n}{0}=1$$

So, if k == 0, the result is one.

Remember: The logical OR operator in C is ||
That's shift and the "\" key.

In our program, that means if k == n or k == 0, we should return 1 from Binomial(n, k). Otherwise, return the recursive version.

 Add this test to your program and get it to work.

Turn in combi1.c via http://secure.cse.msu.edu/handin/

Cards.c

Create a new file called cards.c and put this code in it:

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

/*
 * Name : <Insert name here>
 * Program to draw playing cards
 */

int main()
{
    int suit;
    int card;

    /*
     . This seeds the random number
```

```
     . generator with the current time
     */
    srand(time(NULL));

    /* Create a random card and suit */
    /* This will compute a random number from 0 to 3 */
    suit = rand() % 4;

    /* This will compute a random number from 1 to 13 */
    card = rand() % 13 + 1;

    switch(card)
    {
    case 1:
        printf("Ace");
        break;

    case 11:
        printf("Jack");
        break;

    case 12:
        printf("Queen");
        break;

    case 13:
        printf("King");
        break;

    default:
        printf("%d", card);
        break;
    }

    printf(" of ");

    switch(suit)
    {
    case 0:
        printf("Hearts");
        break;

    case 1:
        printf("Diamonds");
        break;

    case 2:
        printf("Spades");
        break;

    case 3:
        printf("Clubs");
        break;
    }

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


When you compile and run this you should get a random card each time.

The function rand() computes a random number and returns it. The number is between 0 and RAND_MAX, a large number. A common trick is to modulus this with some number n to get numbers in the range 0 to n-1. The % operator means

modulus. For integers, it returns the remainder after integer division. If a divide

modulus. For integers, it returns the remainder after integer division. If we divide by 13, the remainder will be a number from 0 to 12.

The random number generator has to be *seeded*. It's not really random, it's just a function that computes a sequence of numbers based on a seed that is pseudorandom. If you give it the same seed, you'll get the same numbers. A seed is what will grow your random numbers. A common idea is to see with the current time. But, if you run your program fast enough, you'll run it twice in the same second and get the same cards. You only call srand() one time in your program!

We're going to draw and print more than one card. I sure would not want to have multiple copies of that big chunk of code that prints out the card. So, I'm going to move it to a function.

First question to answer: What will your function do? In this case it will print out the card and suit.

Second question to answer: What do we want to name our function? I'm going to give it the name PrintCard. Add this code to the end of your program, but don't try to compile, yet:

```
PrintCard()
{
}
```

You can use any valid identifier name in C for a function. The convention I like to use is to capitalize the first letter of a function and use "CamelCase". CamelCase means words are concatenated together with no space between them and the first letter of each word is capitalized.

Third question: What does your function need to know? It needs to know the card and the suit. These are both integers in my program. What the function needs to know is what we will make the arguments to be function be. Change the code to add function arguments to PrintCard:

```
PrintCard(int card, int suit)
{
}
```

Fourth question: What does the function return? In this case our function is going to output only, so it won't return anything at all. When we have a function that returns nothing we call that a void function and give it the type void:

```
void PrintCard(int card, int suit)
{
}
```

 We now have a valid function definition for our function. It won't do anything yet, but you can compile it.

After making the function definition, even while it is empty, let's add the function declaration. Copy the first line of the function and put it at the top of the program after the #includes and put a semicolon on the end of it:

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

/* Function declaration */
void PrintCard(int card, int suit);
```

 Be sure this still compiles. It still won't do anything yet.

Now, we are going to move code from the body of our main function to this function. Highlight the following code and choose cut:

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

/* Function declaration */
void PrintCard(int card, int suit);
```

```
/*
 * Name : <Insert name here>
 * Program to draw playing cards
 */

int main()
{
    int suit;
    int card;

    /*
     . This seeds the random number
     . generator with the current time
     */
    srand(time(NULL));

    /* Create a random card and suit */
    /* This will compute a random number from 0 to 3 */
    suit = rand() % 4;

    /* This will compute a random number from 1 to 13 */
    card = rand() % 13 + 1;

    switch(card)
    {
    case 1:
        printf("Ace");
        break;

    case 11:
        printf("Jack");
        break;

    case 12:
        printf("Queen");
        break;

    case 13:
        printf("King");
        break;

    default:
        printf("%d", card);
        break;
    }

    printf(" of ");

    switch(suit)
    {
    case 0:
        printf("Hearts");
        break;

    case 1:
        printf("Diamonds");
        break;

    case 2:
        printf("Spades");
        break;

    case 3:
        printf("Clubs");
```

```
        break;
    }

    printf("\n");
}


void PrintCard(int card, int suit)
{
}
```

Then paste it into the PrintCard function:

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

/* Function declaration */
void PrintCard(int card, int suit);

/*
 * Name : <Insert name here>
 * Program to draw playing cards
 */

int main()
{
    int suit;
    int card;

    /*
     . This seeds the random number
     . generator with the current time
     */
    srand(time(NULL));

    /* Create a random card and suit */
    /* This will compute a random number from 0 to 3 */
    suit = rand() % 4;

    /* This will compute a random number from 1 to 13 */
    card = rand() % 13 + 1;


    printf("\n");
}


void PrintCard(int card, int suit)
{
    switch(card)
    {
    case 1:
        printf("Ace");
        break;

    case 11:
        printf("Jack");
        break;

    case 12:
        printf("Queen");
        break;
```

```
    case 13:
        printf("King");
        break;

    default:
        printf("%d", card);
        break;
    }

    printf(" of ");

    switch(suit)
    {
    case 0:
        printf("Hearts");
        break;

    case 1:
        printf("Diamonds");
        break;

    case 2:
        printf("Spades");
        break;

    case 3:
        printf("Clubs");
        break;
    }
}
```

Now, add a function invocation where that code used to be:

```
    /* Create a random card and suit */
    /* This will compute a random number from 0 to 3 */
    suit = rand() % 4;

    /* This will compute a random number from 1 to 13 */
    card = rand() % 13 + 1;

    PrintCard(card, suit);

    printf("\n");
```

 This should work just as it did before.

This is a common activity: moving code *to a function*. If we know the code is going to be used more than once, this is an easy way to put it in one place and use it more than once. But, keep these caveats in mind:

We can send as many parameters to the function as we want. We can only return one thing right now (or zero things in the case of a void function). *We will show ways to return more than one thing later one.*

The names you use in the function do not have to match the names where you invoke the function. Usually they do not.

Now, make your program choose two cards in a row and print the two cards. Don't worry about it picking the same card twice in a row. That's okay for now.

Your Final Task

Write a program that plays one hand of the game of war. The program should generate two cards, one each for players 1 and 2 and display their values. Then, the program should print one of three messages:

- Player 1 wins
- Player 2 wins

- There is a tie

You must have a function called PrintResult that accepts the two cards and prints the result message depending on who is the winner.

Rules: First compare the card. 2 is the lowest, followed by the rest of the numbers, then Jack, Queen, King, and Ace. So, an Ace beats a King, a Jack beats a 5. If the cards are the same value, compare the suits. The order from best to worst for the suits is:

- Hearts
- Diamonds
- Spades
- Clubs

Of course, in the program these values are all represented using integers. You'll have to figure out what integer from the code.

You must draw two unique cards. Random number generators can very well return the same numbers, since all numbers are equally probable. So, if the second player's cards are the same as the first, you should try again until you generate a unique second card. Note that this can be hard to test, since the odds of two identical cards in a row is 1 out of 52. You might want to consider making your program cheat temporarily to test this code. This does take a bit of thought!

Name your program: war.c and turn in via Handin.

CSE 251