**Nested Loops**

Consider the following task: Ask the user to enter a positive integer, n, and print out a n x n square of astericks. Thus, if the user enters 4, we want to print

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

If the user enters 6, we want to print:

\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*

Assuming that we use the variable numrows to store the size of one side of the square, our first attempt at solving the problem might look like this:

int i;

for (i=1; i<=numrows; i++) {

 printf("\*\*\*\*\n");

}

If we run this and enter 4, then our program will work, but if we run this and enter 6, our program will print the following:

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

The problem is that depending on what numrows is, we want the printf inside of our for loop to do different things. In reality, what we really want to do is print out numrows amount of stars, not 4 stars or 6 stars. The number of stars we want to print out on each row is *variable*, and not constant (the same thing) every time.

Thus, we need a loop to print out exactly numrows stars:

int j;

for (j=1; j<=numrows; j++) {

 printf("\*");

}

Thus, we might improve our solution as follows:

int i,j;

for (i=1; i<=numrows; i++) {

 for (j=1; j<=numrows; j++) {

 printf("\*");

 }

}

Once we run this, however, we find out that this, also doesn't work. The problem is that all the stars end up on one line. Upon analyzing the code again, we see why this happened: There are printfs that advance us to the next line. After careful consideration, we realize that once we FINISH printing all the stars on one line, we should advance to the next line:

int i,j;

for (i=1; i<=numrows; i++) {

 for (j=1; j<=numrows; j++) {

 printf("\*");

 }

 printf("\n");

}

The completed program is as follows:

#include <stdio.h>

int main() {

 int numrows;

 printf("How many rows do you want in your square?\n");

 scanf("%d", &numrows);

 int i,j;

 for (i=1; i<=numrows; i++) {

 for (j=1; j<=numrows; j++) {

 printf("\*");

 }

 printf("\n");

 }

 return 0;

}

This structure is known as a nested loop structure. It's a situation where inside of one loop lies another loop. Often times, when developing a nested loop, we first figure out the details of one loop and then realize that we need a second one inside of it. After enough practice, programmers will typically know when they need a set of nested loops. In this situation, the values of i and j advance as follows (using the example where numrows is 4) and each of the ordered pairs represents i followed by j:

(1, 1) (1, 2) (1, 3) (1, 4)

(2, 1) (2, 2) (2, 3) (2, 4)

(3, 1) (3, 2) (3, 3) (3, 4)

(4, 1) (4, 2) (4, 3) (4, 4)

Essentially, i first gets set to 1. Then, j goes through the values 1, 2, 3 and 4. After this occurs, i gets set to 2, and j goes through the values 1, 2, 3, and 4 again. The process continues until i gets to 4.

Now, let's try to print out a rectangle of stars. In a rectangle, the length and the width may be different, thus we must have the user enter two values. Let's read these into the variables numrows and numcols.

We will utilize almost the exact same structure, but our inner loop will run to a different value:

#include <stdio.h>

int main() {

 int numrows, numcols;

 printf("How many rows do you want in your rectangle?\n");

 scanf("%d", &numrows);

 printf("How many columns do you want in your rectangle?\n");

 scanf("%d", &numcols);

 int i,j;

 for (i=1; i<=numrows; i++) {

 for (j=1; j<=numcols; j++) {

 printf("\*");

 }

 printf("\n");

 }

 return 0;

}

Now, let's consider trying to print out a triangle:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

On each line, we print a different number of stars. Upon closer inspection, we see that the number of stars we want to print equals the number of the row we are on. Our general framework is as follows:

for (i=1; i<=numrows; i++) {

 // Print stars for row i.

 printf("\n");

}

Thus, what we really want to do is print i number of stars where the comment is. Well, that's easy, because a for loop lets us repeat something a particular number of times. This time, we want to print out a star exactly i times:

for (j=1; j<=i; i++)

 printf("\*");

Putting it all together, we get:

#include <stdio.h>

int main() {

 int numrows;

 printf("How many rows do you want in your triangle?\n");

 scanf("%d", &numrows);

 int i,j;

 for (i=1; i<=numrows; i++) {

 for (j=1; j<=i; j++) {

 printf("\*");

 }

 printf("\n");

 }

 return 0;

}

**Prime Number Example**

A prime number is a number that is only divisible by 1 and itself. Remember that we can test for divisibility with the mod operator. If a number n is divisible by a number p, then n%p will be equal to zero. The smallest prime number is 2.

Consider the problem of printing out all the prime numbers from 1 to 100.

First, let's consider an easier problem of determining if one number is prime. We will first write a program where we ask the user to enter n, and then determine if it's prime or not.

We will loop through each number from 2 to n-1 and see if any of them divides evenly into n. If any of them do, then we'll have to "save" the fact that we found a divisor. We can use a "boolean" variable to keep track of this. The term boolean means true or false. There is no boolean type in C. But, we can use an integer and store 1 for true and 0 for false. If we find a divisor, then the number we are testing is NOT prime. Before we find a divisor, we can assume a number is prime. Putting these steps together, we get the following program:

#include <stdio.h>

int main() {

 int n;

 printf("Enter an integer greater than 1.\n");

 scanf("%d", &n);

 int i, isprime = 1;

 for (i=2; i<n; i++) {

 if (n%i == 0) {

 isprime = 0;

 }

 }

 if (isprime == 1) {

 printf("%d is prime.\n", n);

 }

 else {

 printf("%d is NOT prime.\n", n);

 }

 return 0;

}

So, now that we've figured out how to determine if one number is prime, let's try to tackle the original problem of printing out all the primes from 1 to 100. We can just do a loop that goes through each number we are testing! The outer structure will loop like this:

int number;

for (number=2; number<=100; number++) {

 // Test to see if number is prime.

 // Print out the result for number.

}

Adapting our code, we get the following:

#include <stdio.h>

int main() {

 int number;

 printf("The primes from 1 to 100 are ");

 for (number=2; number<100; number++) {

 int i, isprime = 1;

 for (i=2; i<number; i++) {

 if (number%i == 0) {

 isprime = 0;

 }

 }

 if (isprime == 1) {

 printf("%d ", n);

 }

 }

 printf("\n");

 return 0;

}

Notice that we've built a nested loop stucture, and we arrived at it by designing one loop at a time, but realizing that after our first loop, we needed another one outside or inside of it.