COP 3223: C Programming
Spring 2009

Nested Control Structures

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

An Aside On Boolean Values In C

- The C89 standard for the C programming language does not include the Boolean data type. (The C99 standard does, but not all C compilers have yet adopted the C99 standard).
- A common solution to this problem that has been adopted by many C programmers is to define your own definitions. This can be done in two different ways. I'll show you the most common way first.

  Define constants for both true and false as follows:

```
#define TRUE 1
#define FALSE 0
```

  Then to use these values do something like:

```
int flag = FALSE;    or    int flag = TRUE;
```

boolean test.c | [*] boolean representation.c

```c
//typical way to represent Boolean types in C
//January 26, 2009   Written by: Mark Llewellyn

#include <stdio.h>

#define TRUE 1     //nonzero == true in C
#define FALSE 0    //zero == false in C

int main()
{
    int control;  //a "Boolean" variable

    control = TRUE;  //comment this line to make control FALSE
    //control = FALSE;  //uncomment this line to make control FALSE
    if (control) {
      printf("The value of control was TRUE\n");
    }
    else {
       printf("The value of control was FALSE\n");
    }

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Programmin...

```
The value of control was TRUE


Press any key to continue . . .
```

boolean test.c | boolean representation.c

```c
//typical way to represent Boolean types in C
//January 26, 2009   Written by: Mark Llewellyn

#include <stdio.h>

#define TRUE 1     //nonzero == true in C
#define FALSE 0    //zero == false in C

int main()
{
    int control;  //a "Boolean" variable

    //control = TRUE;  //comment this line to make control FALSE
    control = FALSE;  //uncomment this line to make control FALSE
    if (control) {
      printf("The value of control was TRUE\n");
    }
    else {
       printf("The value of control was FALSE\n");
    }

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Programmi...

```
The value of control was FALSE


Press any key to continue . . .
```

An Aside On Boolean Values In C

- In the previous example notice that the conditional expression used in the `if` statement had the form:

```
if (control)
```

rather than

```
if (control == TRUE)
```

- The first form is the preferred form because (1) it is more concise and (2) the expression will still work correctly within the normal C environment even if control has a value other than 0 or 1.

An Aside On Boolean Values In C

- The other way of accomplishing this is to use the `typedef` statement to define a user defined type that can be used as a synonym for the built-in type it is based on:

```
typedef int Boolean;
```

then declare a variable to be of this newly defined type as in:

```
Boolean control;
```

- As the example program on the next page illustrates this technique, which is often combined with the first technique to define a complete definition of a Boolean type (i.e., the definitions for true and false are also used).
- We'll do more with the `typedef` statement later.

pictures.c | [*] boolean typedef.c

```c
//using a typedef to define a Boolean type in C
//January 26, 2009   Written by: Mark Llewellyn

#include <stdio.h>
#define TRUE 1     //nonzero == true in C
#define FALSE 0    //zero == false in C

typedef int Boolean;  //define a type named Boolean of the int type

int main()
{
    Boolean control;  //a "Boolean" variable

    //control = TRUE;  //uncomment this line to make control FALSE
    //control = FALSE;  //uncomment this line to make control FALSE
    printf("Enter 0 if you want FALSE and 1 if you want TRUE\n");
    scanf("%d",&control);

    if (control) {
      printf("The value of control was TRUE\n");
    }
    else {
       printf("The value of control was FALSE\n");
    }

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Programming\Spring 2009\C...

```
Enter 0 if you want FALSE and 1 if you want TRUE
1
The value of control was TRUE

Press any key to continue . . .
```

Nesting Control Structures

- We’ve seen the three selection statements (if, if…else, and switch) and the three repetition statements (while, do…while, and for) in isolation, but their real power comes from combining them together in sequence (the third control structure).
- The sequence in which the statements of a C program can are ordered is, of course, dependent upon the problem that the program is designed to solve.
- Recall that every selection and repetition statement has in its body a statement. There is no restriction on what that statement or statements might be. So far, we’ve basically just had simple arithmetic expressions or I/O expressions in the body of our control statements.

Nesting Control Structures

- Whenever a control structure statement includes, within its body, another control structure statement, the structures are said to be nested control structures or more commonly just nested statements.

- To illustrate the concept of nesting control statements, let's consider the following problem:
  - We want to print all the integer numbers between 1 and 30 and determine for each number if the number is odd or even and print this along with the number.

- Clearly, our solution will involve a loop that will allow us to operate on the first 30 integer numbers, but for each number, we also need to make a decision (i.e., a selection) about the number so we can print whether it is odd or even.

nested control structures 1.c

```c
//nested control structures example 1
//for the first 30 integer numbers determine i
//January 27, 2009   Written by: Mark Llewelly

#include <stdio.h>

int main()
{
    int counter;  //loop control and integer n

    printf("\n"); //just moves output down 1 line
    for (counter = 1; counter <= 30; counter++) {
        if (counter % 2 == 0) {
            printf("The integer value %2d is even\n", counter);
        }
        else {
            printf("The integer value %2d is odd\n", counter);
        }//end if...else stmt
    }//end for stmt
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Progr...

```
The integer value  1 is odd
The integer value  2 is even
The integer value  3 is odd
The integer value  4 is even
The integer value  5 is odd
The integer value  6 is even
The integer value  7 is odd
The integer value  8 is even
The integer value  9 is odd
The integer value 10 is even
The integer value 11 is odd
The integer value 12 is even
The integer value 13 is odd
The integer value 14 is even
The integer value 15 is odd
The integer value 16 is even
The integer value 17 is odd
The integer value 18 is even
The integer value 19 is odd
```

nested control structures 2.c

```c
//nested control structures example 2
//same as example 1 except different structure
//for the first 30 integer numbers determine if eac
//January 27, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int counter;  //loop control and integer number

    printf("\n"); //just moves output down 1 line
    for (counter = 1; counter <= 30; counter++) {
        if (counter % 2 == 0) {
            printf("The integer value %2d is even\n", counter);
        }//end if stmt
    }//end for stmt
    printf("\n");
    for (counter = 1; counter <= 30; counter++) {
        if (counter % 2 == 1) {
            printf("The integer value %2d is odd\n", counter);
        }//end if stmt
    }//end for stmt
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Progr...

```
The integer value  2 is even
The integer value  4 is even
The integer value  6 is even
The integer value  8 is even
The integer value 10 is even
The integer value 12 is even
The integer value 14 is even
The integer value 16 is even
The integer value 18 is even
The integer value 20 is even
The integer value 22 is even
The integer value 24 is even
The integer value 26 is even
The integer value 28 is even
The integer value 30 is even

The integer value  1 is odd
The integer value  3 is odd
The integer value  5 is odd
The integer value  7 is odd
The integer value  9 is odd
```

Same program as on page 10, but with a different structure (and slightly different output as well). Which is more efficient from an execution point of view?

Answer: the one on page 10, it has only 1 loop.

Nesting Control Structures

- In the section of notes that covered selection statements, we saw an example of nested `if…else` statements (see page 19 of Control Structures – Part 2).
- That example, was mainly to illustrate the preferred indentation style for nested `if…else` statements. However, we mentioned at the time that the C compiler uses a proximity rule when associating `else` clauses with `if` statements.
- More clearly stated this rule is:

  `An else clause belongs to the nearest if statement that has not already been paired with an else clause.`

Nesting Control Structures

- Notice that this is another reason to always use the braces (to make statement blocks) even if only one statement is contained inside the control statement.
- So, in this case we would have written:

```
if (y != 0) {
  if (x != 0) {
    result = x / y;
  }//end if stmt
}
else {
  printf("Error… y is 0\n");
}//end if…else stmt
```

Nesting Control Structures

- Failure to properly follow the nesting rules for `if…else` statements can get you into trouble. The problem is more commonly known as the dangling else problem. The problem below illustrates the dangling else problem.

- For each chunk of code assume `x = 9` and `y = 11`, and then repeat assuming `x = 11` and `y = 9`. What is the output in each case?

```
if (x < 10)
if (y > 10)
printf("****\n");
else
printf("####\n");
printf("$$$$\n");
```

(a)

```
if (x < 10) {
if (y > 10)
printf("****\n");
}
else {
printf("####\n");
printf("$$$$\n");
}
```

(b)

```
if (x < 10) {
if (y > 10) {
printf("****\n");
}
else {
printf("####\n");
printf("$$$$\n");
} }
```

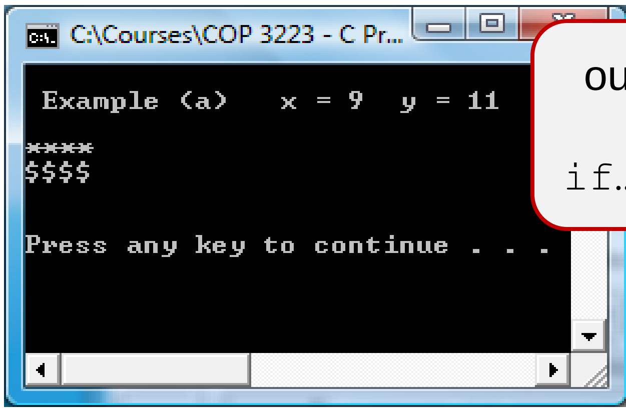
(c)

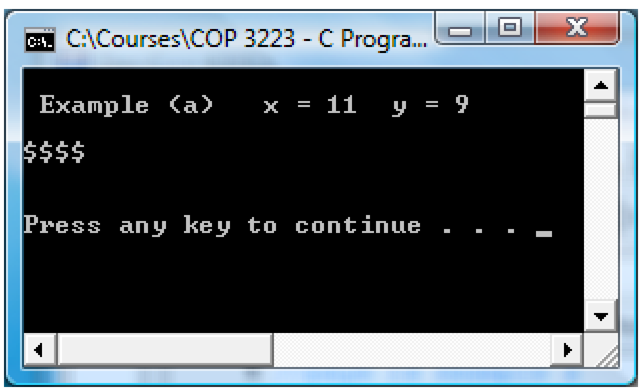
(a)

```
if (x < 10)
if (y > 10)
printf("****\n");
else
printf("####\n");
printf("$$$$\n");
```

outside of if…else

else belongs to inner if



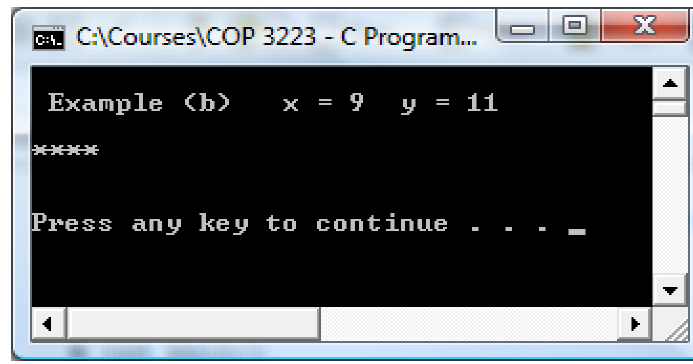


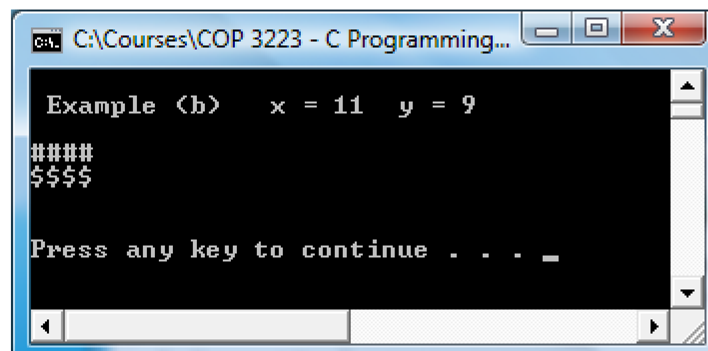
(b)

```
if (x < 10) {
if (y > 10)
printf("****\n");
}
else {
printf("####\n");
printf("$$$$\n");
}
```

inside else clause

else belongs to outer if



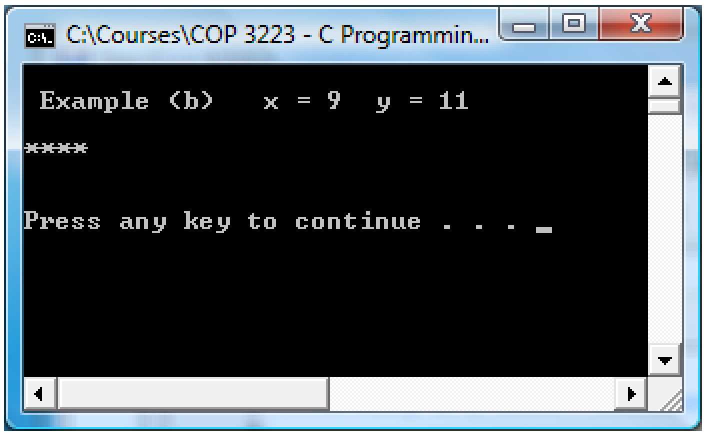


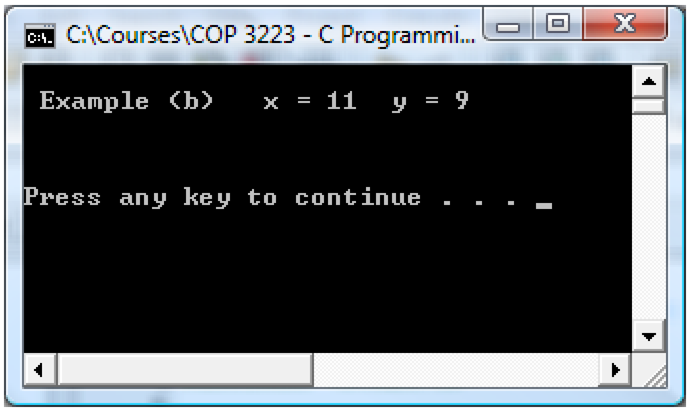
(c)

```
if (x < 10) {
if (y > 10) {
printf("****\n");
}
else {
printf("####\n");
printf("$$$$\n");
} }
```

`else` belongs to inner `if`

outer `if` ends here





Using The Math Library

- We've been using the standard input/output library since we wrote our very first C program. How the `printf` statement is defined is contained in the `stdio` library header file. Since all of our programs have made use in some fashion of the `scanf` and `printf` statements, we've included this library header file in all of our programs so far.
- So far, this is the only header file that we've included in any of our programs. That's about to change as we now introduce the standard math library in C.
- The standard library header file `math.h` contains the function prototypes for mathematical functions that fall into five different groups: trigonometric functions, hyperbolic functions, exponential and logarithmic function, power functions, and nearest integer, absolute value, and remainder functions.

Using The Math Library

Trigonometric Functions
`double acos(double x);`
`double asin(double x);`
`double atan(double x);`
`double atan2(double x, double y);`
`double cos(double x);   //argument in radians`
`double sin(double x);   //argument in radians`
`double tan(double x);   //argument in radians`

Hyperbolic Functions
`double cosh(double x);`
`double sinh(double x);`
`double tanh(double x);`

Using The Math Library

Exponential and Logarithmic Functions
`double exp(double x);  //returns e`^{x}
`double frexp(double value, int *exp);`
`double ldexp(double x, int exp);`
`double log(double x); //log base e`
`double log10(double x); //log base 10`
`double modf(double value, double *iptr);`

Power Functions
`double pow(double x, double y);  //returns x`^{y}
`double sqrt(double x); //returns square root of x`

Using The Math Library

Nearest Integer, Absolute Value, and Remainder Functions

`double ceil(double x);  //returns ceiling of x –`
`                         //smallest integer greater than`
`                         //or equal to x – i.e. rounds`
`                         //up.`

`double fabs(double x); //returns absolute value of x`

`double floor(double x); //returns floor of x – largest`
`                         //integer less than or equal to`
`                         //x – i.e. rounds down.`

`double fmod(double x, double y); //returns the`
`                                 //remainder when first`
`                                 //argument is divided`
`                                 //by the second.`

page 14c.c | math library.c

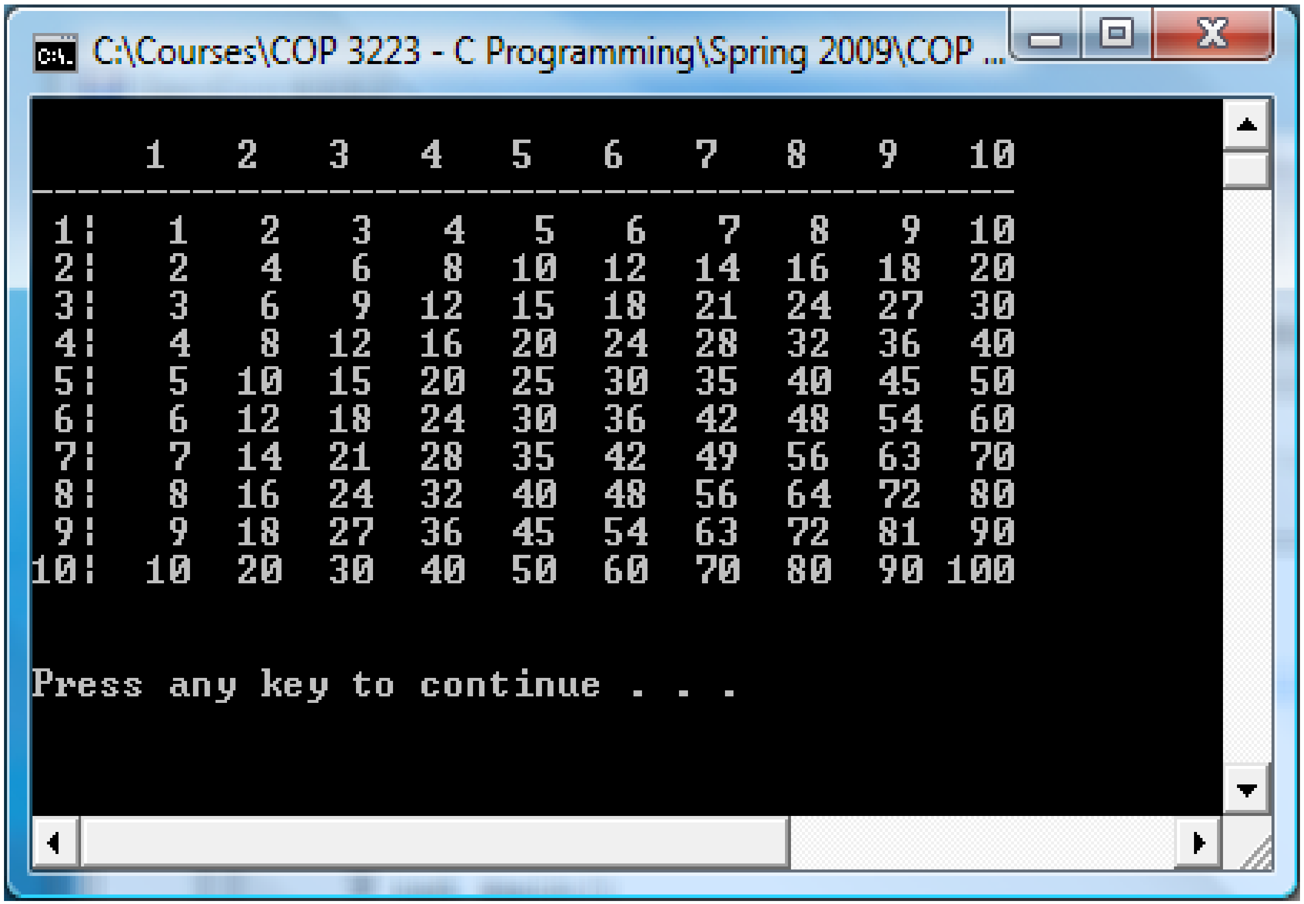
```c
//example using the math library
//Calculating compound interest
//Janaury 27, 2009    Written by: Mark

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

int main()
{
    double amount; //amount on deposit
    double principal = 10000; //starting principle
    double apr = 0.05; //annual percentage rate
    int year;  //year counter

    //output table column headers
    printf("%4s%21s\n","YEAR","Amount On Deposit");
    //calculate amount on deposit for each of 10 years
    for (year = 1; year <= 10; year++) {
        //calculate a new amount for the specified year
        amount = principal * pow(1.0 + apr, year);
        //output table row
        printf("%4d%15.2f\n", year, amount);
    }//end for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

C:\Courses\COP 3223 - C Pr...

```
YEAR    Amount On Deposit
   1         10500.00
   2         11025.00
   3         11576.25
   4         12155.06
   5         12762.82
   6         13400.96
   7         14071.00
   8         14774.55
   9         15513.28
  10         16288.95


Press any key to continue . . .
```

Practice Problems

1. Construct a C program that uses nested control structures to produce the following multiplication table.



Practice Problems

2. Construct a C program that produces gear ratio charts for bicycles. The gear ration is determined by the expression:

```
(size of front chainring / size of rear cog ) * wheelsize
```

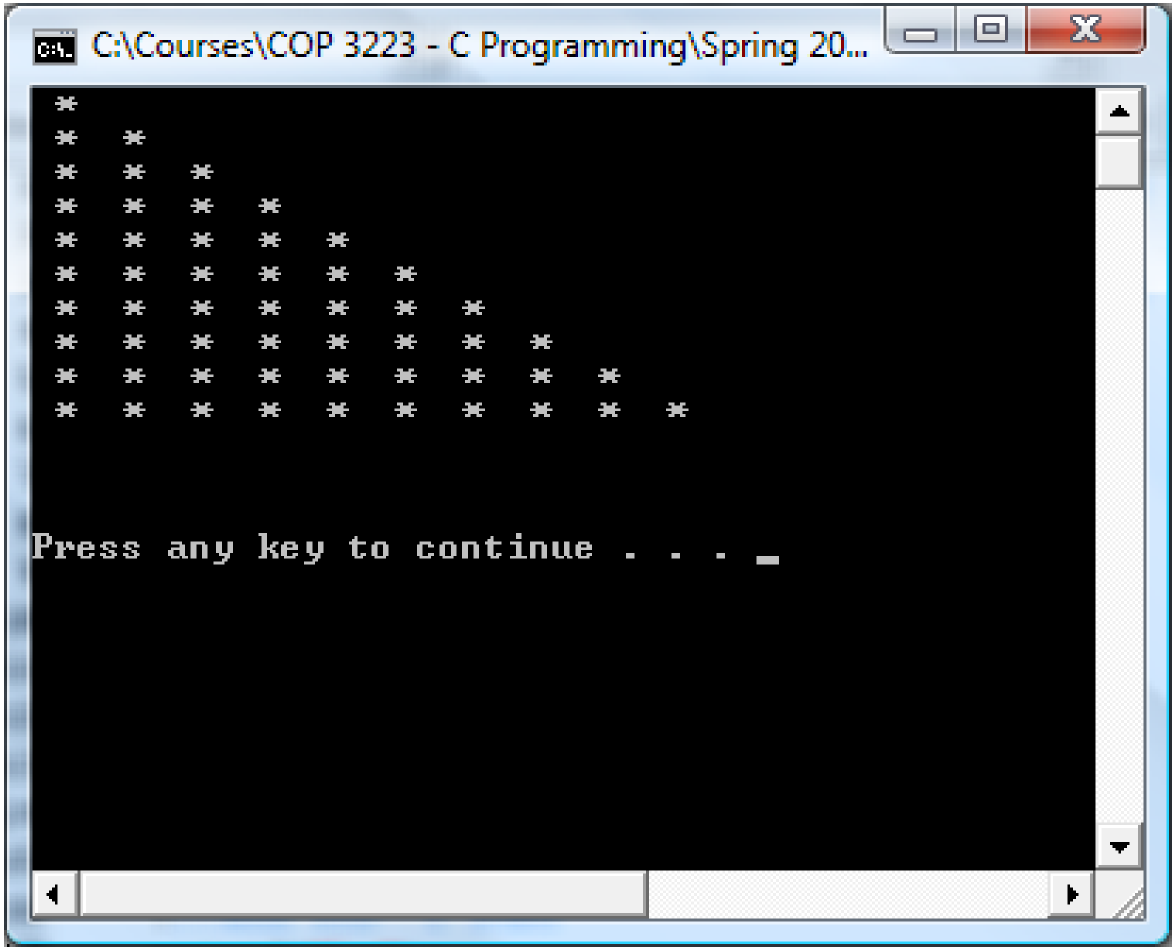
where typical chainring sizes are between 28 and 55 teeth and typical cog sizes are between 11 and 25 teeth. The wheelsize is the diameter of the rear wheel in inches.

C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Nested Control Struct...

```
Enter number of teeth on big ring
53
Enter number of teeth on small ring
39
Enter wheel diameter in inches
27
Enter number of teeth on smallest rear cog
11
Enter number of teeth on largest rear cog
21


      11     12     13     14     15     16     17     18     19     20     21
39    95.7   87.8   81.0   75.2   70.2   65.8   61.9   58.5   55.4   52.7   50.1

53   130.1  119.3  110.1  102.2   95.4   89.4   84.2   79.5   75.3   71.6   68.1


Press any key to continue . . .
```

Practice Problems

3. Construct a C program that produces the following output.



Practice Problems

4. Construct a C program that produces the following output.

C:\Courses\COP 3223 - C Programming\Spring 2...

```
*
*   *
*   *   *
*   *   *   *
*   *   *   *   *
*   *   *   *   *   *
*   *   *   *   *   *   *
*   *   *   *   *   *   *   *
*   *   *   *   *   *   *   *   *
*   *   *   *   *   *   *   *   *   *
*   *   *   *   *   *   *   *   *   *
*   *   *   *   *   *   *   *   *
*   *   *   *   *   *   *   *
*   *   *   *   *   *   *
*   *   *   *   *   *
*   *   *   *   *
*   *   *   *
*   *   *
*   *
*



Press any key to continue . . .
```