COP 3223: C Programming
Spring 2009

Program Control Structures In C – Part 2

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

Control Structures In C

- C provides three types of repetition structures in form of statements.

1. The `while` repetition statement allows an action to be repeated as long as some condition remains true. This is a "top-tested" repetition statement, which means that the condition is evaluated before the action is executed the first time. If the condition is initially false, the action is not performed even once.
2. The `do…while` repetition statement allows an action to be repeated as long as some condition remains true. This is a "bottom-tested" repetition statement, which means that the condition is not evaluated until the action is performed the first time. Thus, the action is always performed at least once with this type of repetition statement.
3. The `for` repetition statement repeats an action a specific number of times based upon a counter value (an integer). This repetition statement is referred to as a "counted loop" statement.

The `while` Repetition Statement

- The format of the `while` repetition statement is:

```
while ( condition ) {
    statements;
}
statement x;
```

These statements are executed only if the condition evaluates to true. Statement x is the first statement to be executed when the condition is false.

- The condition is any expression which evaluates to true or false (i.e., a Boolean expression).
- When a `while` statement (also called `while loop`) is executed, the condition is evaluated first. If its value is nonzero (true) the statements in the body of the `while` loop are executed and the expression is evaluated again. Execution of the statements in the body of the loop continue as long as the condition remains true.

The `while` Repetition Statement

COMMON PROGRAMMING ERROR:

Assuming that a `while` statement's condition is initially true and execution enters the body of the `while` statement, there must be some statement within the body of statements that changes the value of the condition. Otherwise, the condition will always remain true and execution of the loop will never terminate. This is called an "infinite loop".

Always make sure that there is some statement inside the body of statements in a `while` loop that will eventually cause the condition to evaluate to false.

- The program on the following page uses a while statement to print the sum of the first 10 integers (one integer at a time).

template.html | commentform.html | calendar.html | slideshow.js | while stmt example 1.c

Sample `while` statement program

```c
//while repetition statement example 1
//prints sum of first 10 integer numbers
//January 20, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;   //current value
    int runningSum = 0;   //sum of first n integer values

    while (currentValue <= upperLimit) {
        runningSum += currentValue;
        printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
        currentValue++;
    }//end while stmt

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

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

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

This is the statement that modifies the conditional expression. Eventually, the `currentValue` will become greater than the `upperLimit`, thus terminating the statement.

An Aside On Increment and Decrement Operators

- Two very common variable operations that occur in programs, especially so in loop bodies, are incrementing (usually adding 1) and decrementing (usually subtracting 1).
- For example:

```
i = i + 1;   //increment operation
j = j - 1;   //decrement operation
```

- C provides the ++ (increment) and –– (decrement) operators as a way to shorten the expressions shown above into:

```
i++;   //increment operator
j--;   //decrement operator
```


An Aside On Increment and Decrement Operators

- While this may seem fairly simple, it is unfortunately more complicated than it seems.
- The first complication is that either operator can be used as either a prefix operator (`++i` and `--j`) or postfix operator (`i++` and `j--`).
- The prefix version increments the variable before its use (the reference to the variable) while the postfix version increments the variable after its use.
- To help you keep this straight think of `++i` as a "pre-increment operator" and `i++` as a "post-increment operator".

while stmt example 1.c

```c
//while repetition statement example 1
//prints sum of first 10 integer numbers
//January 20, 2009   Written by: Mark Llewel

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer t
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values

    while (currentValue <= upperLimit) {
       runningSum += currentValue;
       printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
       ++currentValue;
    }//end while stmt

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

C:\Courses\COP 3223 - C Programming\Spring 200...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

Notice that in this case the use of the prefix increment operator did not affect the output, i.e., the loop ran the same number of time.

while stmt example 1.c

```c
//while repetition statement example 1
//prints sum of first 10 integer numbers
//January 20, 2009   Written by: Mark Llewel

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values

    while (++currentValue <= upperLimit) {
       runningSum += currentValue;
       printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
    }//end while stmt

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

C:\Courses\COP 3223 - C Programming\Spri...

```
The sum of the first 2 integers is: 2
The sum of the first 3 integers is: 5
The sum of the first 4 integers is: 9
The sum of the first 5 integers is: 14
The sum of the first 6 integers is: 20
The sum of the first 7 integers is: 27
The sum of the first 8 integers is: 35
The sum of the first 9 integers is: 44
The sum of the first 10 integers is: 54


Press any key to continue . . .
```

I moved the increment operator to inside the condition. Notice that in this case the use of the prefix increment operator did affect the output, i.e., the loop “missed” the first value of 1 and thus, all of our running sum numbers are incorrect!

while stmt example 1.c

```c
//while repetition statement example 1
//prints sum of first 10 integer numbers
//January 20, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values

    while (currentValue++ <= upperLimit) {
        runningSum += currentValue;
        printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
    }//end while stmt

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

C:\Courses\COP 3223 - C Programming\Sprin...

```
The sum of the first 2 integers is: 2
The sum of the first 3 integers is: 5
The sum of the first 4 integers is: 9
The sum of the first 5 integers is: 14
The sum of the first 6 integers is: 20
The sum of the first 7 integers is: 27
The sum of the first 8 integers is: 35
The sum of the first 9 integers is: 44
The sum of the first 10 integers is: 54
The sum of the first 11 integers is: 65


Press any key to continue . . .
```

I moved the increment operator to inside the condition. Notice that in this case the use of the postfix increment operator did again affect the output, i.e., the loop "missed" the first value of 1 and thus, all of our running sum numbers are incorrect, plus the loop iterated through the 11th integer!

Another Aside On Compound Assignment Operators

- Another shorthand C operator is the compound assignment operator that is again helpful with typical increment and decrement operations. An example of this appears in the program on page 5 (line 14).
- The compound assignment operator converts:

  `i = i + 1;` and `j = j - 1;`

  to: `i += 1;` and `j -= 1;`
- Unlike with the increment and decrement operators, there are three more compound assignment operators, which are:

  `*=`, `/=`, and `%=`
- The program on the next page illustrates these operators.

while stmt example 1.c | increment and decrement.c

```c
//illustration of increment and decrement operators i
//January 20, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int counter = 1;

    printf("The value of counter is: %d\n", counter);
    printf("The value of counter is: %d\n", ++counter);
    printf("The value of counter is: %d\n", counter++);
    printf("The value of counter is: %d\n", counter);
    printf("The value of counter is: %d\n", counter += 2);
    printf("The value of counter is: %d\n", counter -= 1);

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

C:\Courses\COP 3223 - C Programming\S...

```
The value of counter is: 1
The value of counter is: 2
The value of counter is: 2
The value of counter is: 3
The value of counter is: 5
The value of counter is: 4


Press any key to continue . . .
```

Yet Another Aside On Arithmetic In C

- C allows the programmer a fair amount of flexibility when it comes to mixing types of operands inside arithmetic statements.
- For example, suppose we have:

```
int num1;  double num2;

     num1 + num2   or num2 + num1;

     num1 * num2   or num2 * num1;

     num1 - num2   or num2 - num1;

     num1 / num2   or num2 / num1;
```

- All of these are perfectly legal arithmetic expressions in C.

Yet Another Aside On Arithmetic In C

- However, the same is not quite true when it comes to the assignment statement, as many of you have discovered while working on the first assignment. So, I'll take a few pages here to give a quick overview of arithmetic assignments in C.
- C includes a number of automatic type conversions, known as implicit conversions. Because C has so many different arithmetic types, the implicit conversion rules are somewhat complex, so we'll introduce only a few right now and hold off on the others until later in the semester.
- C also provides the programmer the capability of performing explicit conversions using the cast operator. We'll also hold this discussion until later in the semester.

Yet Another Aside On Arithmetic In C

- Implicit conversions are performed in the following situations:
  - When the operands in an arithmetic or logical expression don't have the same type. In this case C performs what are referred to as the *usual arithmetic conversions*.
  - When the type of the expression on the right side of an assignment operator does not match the type of the variable on the left side.
  - When the type of an argument in a function call does not match the type of the corresponding parameter (we'll see this later too).
  - When the type of the expression in a return statement does not match the function's return type (we'll see more on this later as well).
- For now we'll look only at the first two cases.

Yet Another Aside On Arithmetic In C

- Most computer hardware only evaluates arithmetic expressions in which the operands are of the same type, the compiler must generate code that ensures the types of all operands are the same.
- To ensure this, the compiler performs an operation known as promotion (implicit conversion) on selected operands in the expression.
- For example, suppose we have:

```
int i;  double d;
```

then in `i+d` the types are not the same, so the compiler must promote the type of `i` to `double`.

Yet Another Aside On Arithmetic In C

- Why is `i` converted to a `double` and not `d` converted to an `int`?
- Answer: Loss of precision.
- Consider: `i = 3; d = 4.67;`
- If `i` is converted to a `double` then `i + d = 7.67`
- If `d` is converted to an `int` then `i + d = 7` and the fractional part of `d` is simply lost!
- In compiler lingo, a promotion is called a widening conversion, because it prevents a loss of precision.
- The typical compiler strategy is to convert operands to the "narrowest" type that will safely accommodate both operand values.

Yet Another Aside On Arithmetic In C

- To give you an even more concrete example from your homework assignment, consider problem 1A (the gas expense problem).
- You will have defined variables such as:

```
int milesPerGallon, milesDriven;
double pricePerGallon, cost;
```

- The expression you need to solve your problem looks like one of the following, but which one?

```
cost = milesDriven / milesPerGallon * pricePerGallon;
cost = pricePerGallon * milesDriven / milesPerGallon;
```

Yet Another Aside On Arithmetic In C

- In the absence of parentheses to override normal precedence rules, C uses normal arithmetic operator precedence evaluating expressions left to right.
- Using the first expression we would have:

```
cost = milesDriven / milesPerGallon * pricePerGallon;

        int     /       int

        1000            24

          = 41.667          *       3.00 (double)

          = 41   (int)

                    = 123.00 (double)
```


Yet Another Aside On Arithmetic In C

- Using the second expression we would have:

```
cost = pricePerGallon * milesDriven / milesPerGallon;

        double    *   int

        3.00          1000

            = 3000.00  (double)     /     24  (int)

                                  = 125.00  (double)
```

HINT: This is the correct version!!

The `do…while` Repetition Statement

- The `do…while` repetition statement is similar to the `while` statement except for one difference.
- That difference is that the while statement evaluates the condition at the beginning of the loop before the body of the loop is performed. Thus, if the condition evaluates the first time to false (zero) the body of the loop is never executed.
- The `do…while` statement evaluates the condition after the body of the statement (the loop) is executed. Therefore, the body of the loop will ALWAYS be executed at least once, even if the condition is initially false.

The `do...while` Repetition Statement

- The format of the `do...while` statement is:

```
do {
    statements;
} while (condition);
```

GOOD PROGRAMMING PRACTICE:
A with other selection and repetition statements we've encountered, it is technically not necessary to include the braces on the body of the loop if it contains a single statement. However, it will enhance the readability of the code if they are always present.

do while statement example 1.c

```c
//do...while repetition statement example 1
//prints sum of first 10 integer numbers
//January 21, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values

    do {
       runningSum += currentValue;
       printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
       currentValue++;
    } while (currentValue <= upperLimit);//end do...while stmt

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

C:\Courses\COP 3223 - C Programming\Sprin...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

Illustrating The Operational Differences Between The `while` And The `do…while` Statements

- The programs on the next two pages illustrate the difference between how the `while` and the `do…while` statements execute.
- Again, this difference is that the `while` statement is a "top-tested" loop and the `do…while` statement is a "bottom-tested" loop.
- All I did was modify the original sum of the first 10 integers program from pages 5 and 15 to include the addition of a variable to control whether or not the loop is executed.
- Be sure you understand completely the operational differences between these two statements.

while stmt example 1.c

```c
//while repetition statement example 1
//prints sum of first 10 integer numbers
//January 20, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values
    int doLoop = 0; //0 means false, 1 means true

    while (currentValue <= upperLimit && doLoop) {
       runningSum += currentValue;
       printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
       currentValue++;
    }//end while stmt

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

C:\Courses\COP 3223 - C Programming\Spri...

```
Press any key to continue . . .
```

Since `doLoop` is false, the condition of the `while` loop is initially false, so no statements in the loop body are executed.

do while statement example 1.c

```c
//do...while repetition statement example 1
//prints sum of first 10 integer numbers
//January 21, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int currentValue = 1;  //current value
    int runningSum = 0;  //sum of first n integer values
    int doLoop = 0;  //0 means false, 1 means true

    do {
       runningSum += currentValue;
       printf("The sum of the first %d integers is: %d\n", currentValue, runningSum);
       currentValue++;
    } while (currentValue <= upperLimit && doLoop);//end do...while stmt

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

C:\Courses\COP 3223 - C Programming\Spri...

```
The sum of the first 1 integers is: 1

Press any key to continue . . .
```

Since `doLoop` is false, the condition of the `while` loop is initially false, but this is not tested until the statements in the loop body are executed the first time.

The `for` Repetition Statement

- The `for` repetition statement is considered a counted loop, but in actuality is very similar to the `while` statement and except for a few rare cases, a `for` statement can always be replaced by a `while` statement.
- The general format of a `for` statement is:

```
for ( expression1; expression2; expression3 ) {
      statement;
}
```

- As with other control statements, the braces are not required if the body consists of a single statement, but again, common programming practice is to always include the braces.

The `for` Repetition Statement

- In the general format of a `for` statement:

  `expression1` is an initialization expression that is performed only once, before any of the statements in the loop are executed.

  `expression2` controls the loop termination. The loop continues to execute as long as the value of `expression2` is true (nonzero).

  `expression3` is an operation that is performed at the end of each loop iteration. Typically, `expression3` is used to ensure that the value of `expression2` eventually becomes false (zero).

The `for` Repetition Statement

- The most common use of the expressions in the `for` statement are shown below:

Keyword `for`

Required semicolon

Required semicolon

```
for ( counter = 1; counter <= 10; counter++ )
```

Loop control variable with initial value

Loop continuation conditional expression

Increment (or decrement) of loop control variable

for statement 1.c

```c
//for repetition statement example 1
//prints sum of first 10 integer numbers
//January 22, 2009   Written by: Mark Llewell

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int runningSum = 0;  //sum of first n integer values
    int control;  //loop control variable

    for (control = 1; control <= upperLimit; control++) {
       runningSum += control;
       printf("The sum of the first %d integers is: %d\n", control, runningSum);
    }//end for stmt

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

C:\Courses\COP 3223 - C Programming\Spring 20...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

The `for` Repetition Statement

- It is possible in the `for` statement to omit any or all of the expressions; which would leave a format of:

```
for( ; ;  )
```

- The most common cases omit either or both of the first and third expressions leaving `for ( ; expression2;  ).` In cases such as these, it is assumed that (1) the initial value of the loop control variable is set before the loop is executed, and (2) some statement inside the loop is responsible for ensuring that the value of `expression2` eventually becomes false (zero).

- If `expression2` is omitted, it defaults to a true value, and the loop never terminates (an infinite loop) which will require some outside intervention to stop the program! Except in very unusual circumstances you don't want this to happen, so always include an `expression2` in your `for` statements.

for statement 2.c

```c
//for repetition statement example 2
//prints sum of first 10 integer numbers
//January 22, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int runningSum = 0;  //sum of first n integer values
    int control = 1;  //loop control variable

    for ( ; control <= upperLimit; control++) {
       runningSum += control;
       printf("The sum of the first %d integers is: %d\n", control, runningSum);
    }//end for stmt

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

C:\Courses\COP 3223 - C Programming\Spri...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

`for` statement with `expression1` missing. Notice that the loop control variable is initialize before the loop begins.

for statement 3.c

```c
//for repetition statement example 2
//prints sum of first 10 integer numbers
//January 22, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int runningSum = 0;  //sum of first n integer values
    int control = 1;  //loop control variable

    for ( ; control <= upperLimit; ) {
       runningSum += control;
       printf("The sum of the first %d integers is: %d\n", control, runningSum);
       control++;
    }//end for stmt

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

C:\Courses\COP 3223 - C Programming\Spri...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

`for` statement with both `expression1` and `expression2` missing. Notice that the loop control variable is initialize before the loop begins and the statement inside the loop that modifies the value of the loop control variable.

Similarity Of The `for` And `while` Statements

- To illustrate how similar the for and while statement are, consider the general form of a for statement as shown below and how this would be represented using a while statement (plus additional statements).

  The `for` statement:

```
for( expression1; expression2; expression3  )
```

  An equivalent `while` statement:

```
expression1;
while ( expression2 ){
        statements;
        expression3;
}
```

- The program on the next page is a while statement version of the for statement example program on page 30, which illustrates this equivalence.

for while equivalence.c

```c
//while statement equivalence of a for statement
//prints sum of first 10 integer numbers
//January 22, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int upperLimit = 10; //largest integer to use
    int runningSum = 0;  //sum of first n integer values
    int control;  //loop control variable

    control = 1;
    while( control <= upperLimit) {
       runningSum += control;
       printf("The sum of the first %d integers is: %d\n", control, runningSum);
       control++;
    }//end while stmt
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

expression1

expression2

expression3

C:\Courses\COP 3223 - C Programming\Spri...

```
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55


Press any key to continue . . .
```

Practice Problems

1. Construct a C program that uses `while` statements to produce multiplication tables from 1 to 10 for the integer values from 1 to 10.

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

```
Multiplication Table for 1
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
Multiplication Table for 2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
Multiplication Table for 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

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

```
Multiplication Table for 4
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
Multiplication Table for 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
Multiplication Table for 6
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

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

```
Multiplication Table for 7
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
Multiplication Table for 8
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
Multiplication Table for 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
```

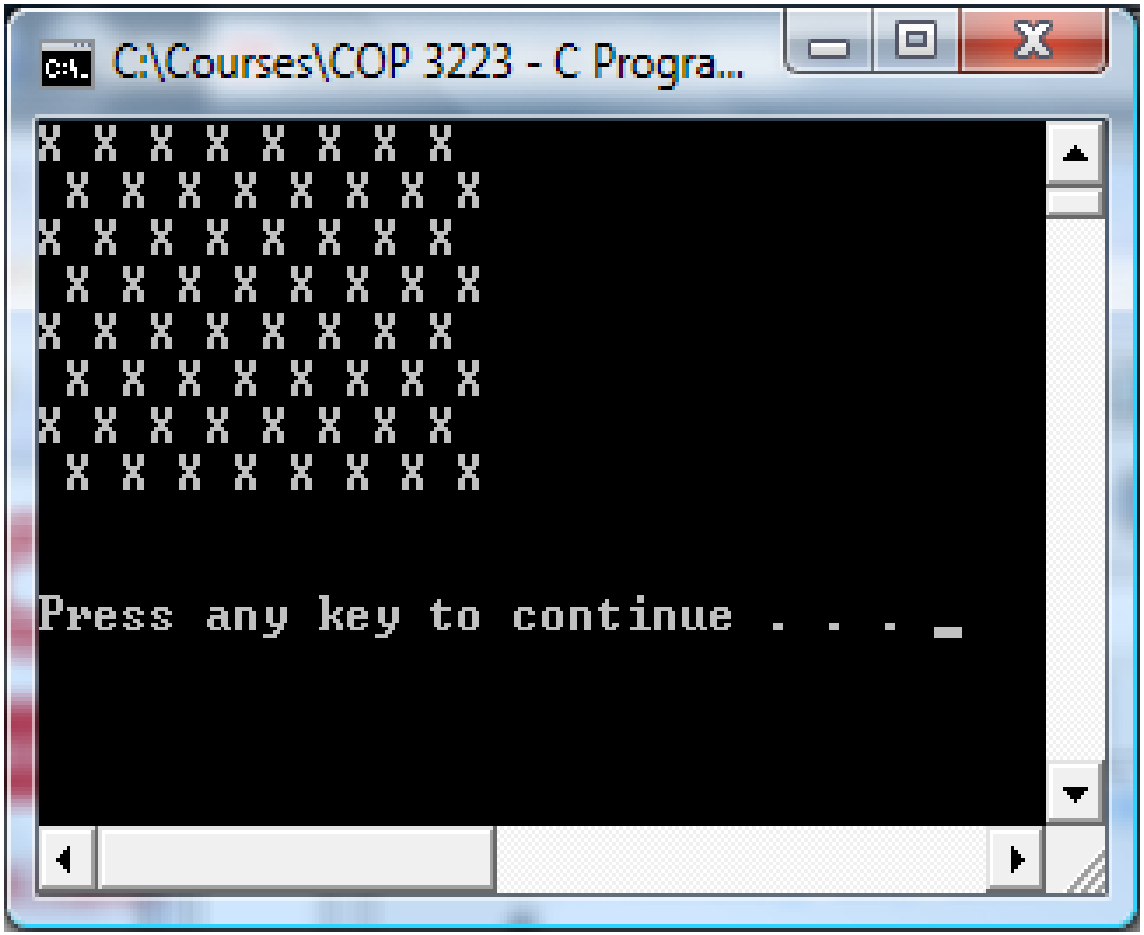
C:\Courses\COP 3223 - C Program...

```
Multiplication Table for 10
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100


Press any key to continue . . .
```

Practice Problems

2. Construct a C program that produces a chessboard pattern as shown.



Practice Problems

3. Construct a C program that prints both the sum and product of the first fifteen integer values.

C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Control Structures Pa...

```
SUMS
The sum of the first 1 integers is: 1
The sum of the first 2 integers is: 3
The sum of the first 3 integers is: 6
The sum of the first 4 integers is: 10
The sum of the first 5 integers is: 15
The sum of the first 6 integers is: 21
The sum of the first 7 integers is: 28
The sum of the first 8 integers is: 36
The sum of the first 9 integers is: 45
The sum of the first 10 integers is: 55
The sum of the first 11 integers is: 66
The sum of the first 12 integers is: 78
The sum of the first 13 integers is: 91
The sum of the first 14 integers is: 105
The sum of the first 15 integers is: 120


PRODUCTS
The product of the first 1 integers is: 1
The product of the first 2 integers is: 2
The product of the first 3 integers is: 6
The product of the first 4 integers is: 24
The product of the first 5 integers is: 120
The product of the first 6 integers is: 720
The product of the first 7 integers is: 5040
The product of the first 8 integers is: 40320
The product of the first 9 integers is: 362880
The product of the first 10 integers is: 3628800
The product of the first 11 integers is: 39916800
The product of the first 12 integers is: 479001600
The product of the first 13 integers is: 6227020800
The product of the first 14 integers is: 87178291200
The product of the first 15 integers is: 1307674368000
```