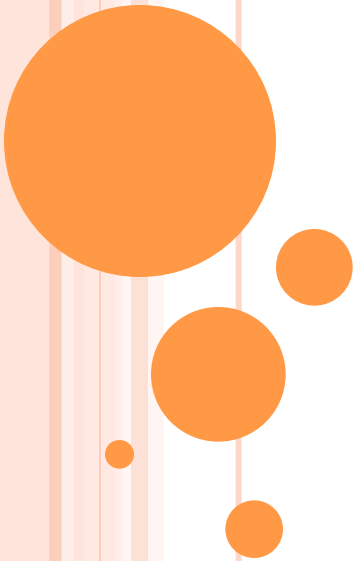
Chapter 4 Notes – **Program Control**

C How to Program, 6th Edition

Deitel & Deitel



SWE110 Lecture Note
Instructor: Zachi Baharav

(Slides credit: Thien An Nguyen (An))

Objectives

- In Lab2, we learned:
  - How to write/read a double number to/from standard output/input
  - scanf("%lf", &dvar), printf("%lf %.2lf", dvar1, dvar2)
  - To repeat a task:
    - Do not copy and paste
    - Use the while loop
- Today, we continue with other control statements:
  - for loop
  - do…while loop
  - switch statement
  - break and continue statements
  - Use function getchar()
  - Logical Operators: &&, ||, ==, and !=
- Require reading: Chapter 3 + 4

4.4 – The `for loop` Statement

while loops syntax:

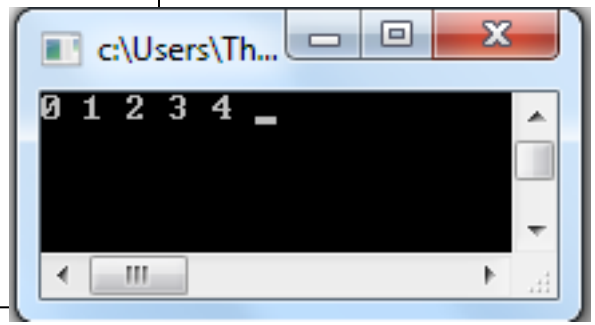
```
   Initialization;                  // begin value
   while (loop-condition){          // if the condition is true
      ...                           // execute the body statements
      condition-update;             // last, update the condition
   }
```

for loops syntax:

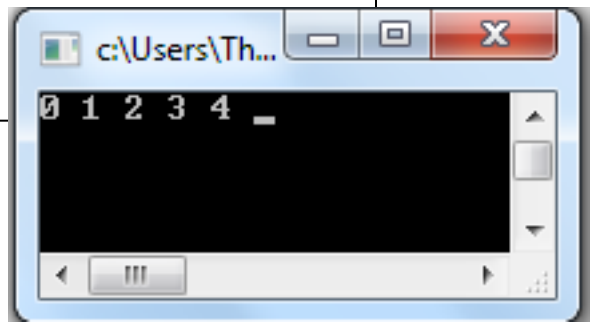
```
   for (initialization; loop-condition; condition-update){
      ...                           // body statements of the for loop
   }
```

4.5 – The `for loop` Statement cont.

```
int MAX_NUM = 5;
int counter;      // an int variable

counter = 0;
// print 0 to MAX_NUM - 1
while (counter < MAX_NUM){
   printf("%d ", counter);
   counter++;
}
```




```
int MAX_NUM = 5;
int counter;      // an int variable

// print 0 to MAX_NUM - 1
for (counter = 0; counter < MAX_NUM; counter++){
   printf("%d ", counter);
}
```

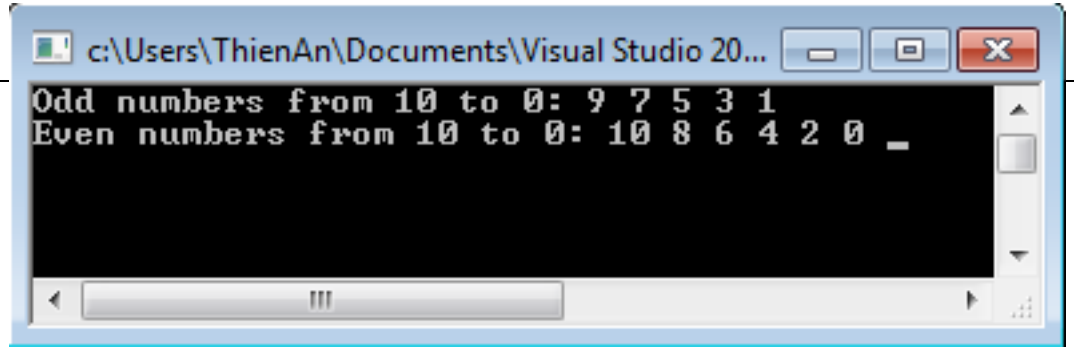


4.6 – The `for loop` Statement cont.

```
//*************************************************************
//  example4_4.c
//
//  print Odd and Even numbers from 10 to 0.
//*************************************************************
#include <stdio.h>

int main()
{
   int MAX_NUM = 10;
   int counter;                        // counter control

   // print odd numbers from MAX_NUM to 0
   printf("Odd numbers from 10 to 0: ");
   for (counter = MAX_NUM; counter >= 0; counter--){
      if ((counter % 2) > 0)
         printf("%d ", counter);
   }
```

4.6 – The `for loop` Statement cont.

```
    // print even numbers from MAX_NUM to 0
    printf("\nEven numbers from 10 to 0: ");
    for (counter = MAX_NUM; counter >= 0; counter--){
        if ((counter % 2) == 0)
            printf("%d ", counter);
    }

    return 0;                          // return code success
}
```



4.6 – The `for loop` Statement cont.

```
int MAX_NUM = 10;
int counter;                    // counter control

// other ways to print odd numbers from MAX_NUM to 0
for (counter = MAX_NUM - 1; counter >= 0; counter -= 2){
      printf("%d ", counter);
}

// other ways to print even numbers from MAX_NUM to 0
for (counter = MAX_NUM; counter >= 0; counter -= 2){
      printf("%d ", counter);
}
```

4.7 – Switch Statement

- The if statement gives you a single selection condition.
- The if…else statement gives the double selection conditions
- The if…else if… else gives more than triple selection conditions.
- Similar to the if…else if…else statement, the switch statement gives you the ability to handle multiple conditions.

4.7 – The `switch` Statement

The syntax:

```
switch (variable){
  case constant1:
     block of statements
     break;                // required to exit the switch
  case constant2:
     block of statements
     break;                // required to exit the switch
  .
  .
  .
  default:
     break;                // optional to the switch
}                          // end of switch statement
```

- As noted in the comments, the `break` statement is used at the end of a block of statements is required to skip the rest of the switch statement (exit the switch statement).

4.7 – The `switch` Statement

- In the next example, the program analyzes users input characters until the user press <Ctrl> z + <enter>
  - Give a report of how many A, B, C, D letters entered
  - The letters can be upper or lower cases
  - If the letters are not in the A – D range, ignore them
  - Also print the total letters entered
- Things we need to know before writing our program:
  - ASCCI code table – Appendix B (ASCII Character Set)
  - This time we will use function `getchar()` to read in user inputs
  - We will use switch statement to analyze user input
  - <Ctrl> z from the keyboard generates character <EOF>

4.6 – The `switch` Statement cont.

```
//*************************************************************
//  example4_7.c
//
//  Report frequently used of A, B, C, and D. <Ctrl> z + <enter> to exit
//*************************************************************
#include <stdio.h>

int main()
{
   // Declare and initializations
   int others = 0;                  // track the other letters entered
   int inChar;                      // current user input
   int aCount = 0,                  // track A letters
       bCount = 0,                  // track B letters
       cCount = 0,                  // track C letters
       dCount = 0;                  // track D letters
```

```
// begin reading and analyzing user inputs
printf("Enter the letter (EOF to end): ");
while ((inChar = getchar()) != EOF){
   // analyze the input character
   switch (inchar){
      case 'A':
      case 'a':
        aCount++;
        break;

      case 'B':
      case 98:
        bCount++;
        break;

      case 'C':
      case 'c':
        cCount++;
        break;

      default:
        others++;
   }
}     // end while loop
```

```
    // Display output
    int total = aCount+bCount+cCount+others;

    printf("Total input characters: %d\n", total);
    printf("Total a or A: %d\n", aCount);
    printf("Total b or B: %d\n", bCount);
    printf("Total c or C: %d\n", cCount);
    printf("Other characters: %d", others);

    return 0;                  // return code success

}   // end program
```

4.8 – The do…while Statement

The syntax:

```
do{
   statemen(s)
} while (loop-condition); // if the condition is true, repeat the loop
```

- It is very similar to the while loop, except that it guarantee the statement(s) inside its body are executed at least once.
- After executing the block statement once, if the loop condition is true, the loop repeats.

4.8 – The do…while Statement cont.

```
//*****************************************************************
//  example4_8.c
//
//  print all numbers from 0 to 10
//*****************************************************************
#include <stdio.h>

int main()
{
   int counter = 0;                  // Declare and initializations

   do {
      printf("%d ", counter);        // 0 1 2 3 4 5 6 7 8 9 10
      counter++;
   } while (counter <= 10);

   return 0;                         // return code success
}
```

4.8 – The do…while Statement

- In the next example, we will be comparing a while and a do… while loop

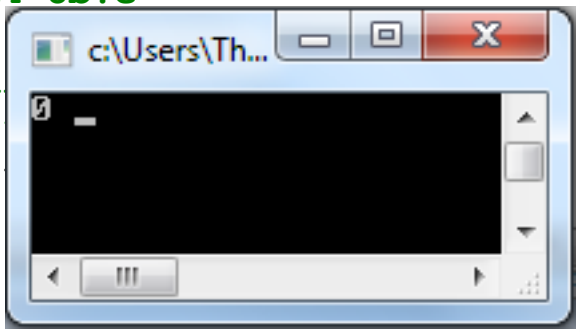
```
//*********************************
//  example4_8b.c
//
//  print all numbers from 0 to 10
//*********************************
#include <stdio.h>

int main()
{
   int counter = 0;

   do {
      printf("%d ", counter);
      counter++;
   } while (counter < 0);
   // at this point counter is 1

   return 0; // return code success
}
```

```
//*********************************
//  example4_8c.c
//
//  print all numbers from 0 to 10
//*********************************
#include <stdio.h>

int main()
{
   int counter = 0;

   while (counter < 0) {
      printf("%d ", counter);
      counter++;
   }
   // at this point counter is 0

   return 0; // return code success
}
```

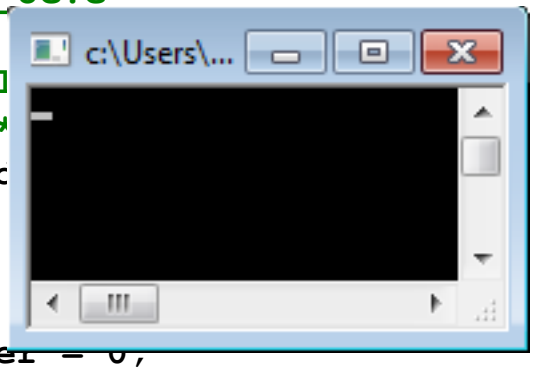
4.8 – The do…while Statement

- In the next example, we will be comparing a while and a do… while loop

```
//*********************************
//  example4_8b.c
//
//  print a
//*********
#include <s

int main()
{
   int counter = 0;

   do {
      printf("%d ", counter);
      counter++;
   } while (counter < 0);
   // at this point counter is 1

   return 0; // return code success
}
```




```
//*********************************
//  example4_8c.c
//
//  print all
//**********
#include <stc

int main()
{
   int counter = 0;

   while (counter < 0) {
      printf("%d ", counter);
      counter++;
   }
   // at this point counter is 0

   return 0; // return code success
}
```

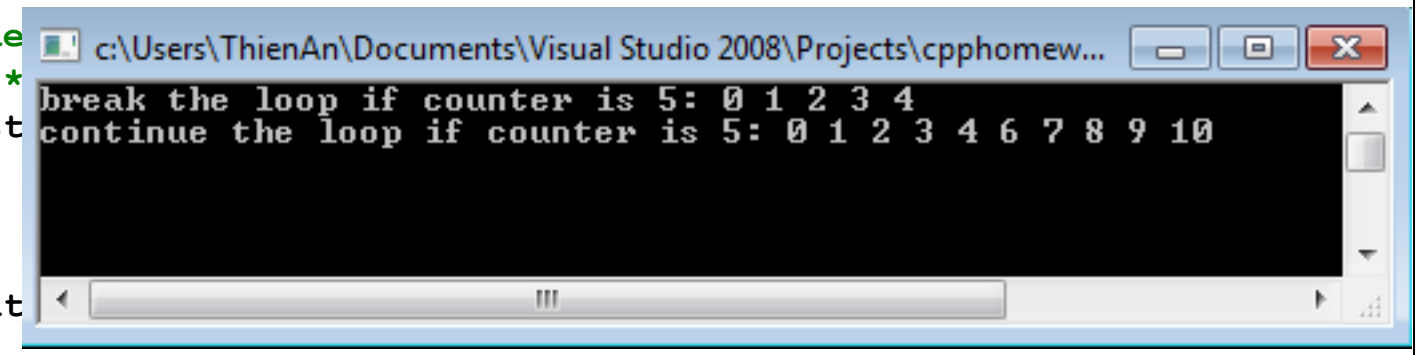


4.9 – The `break, continue` Statements

- The `break` statement is used to `terminate` a loop right away.
- The `continue` statement is used to `skip` the rest of the loop in the current iteration.
- The statements has to be inside a loop

```c
//*************************************************************
//  example4_9.c
//
//  Test the break and continue statements
//*************************************************************
#include <stdio.h>

int main()
{
   int counter;                        // Declaration

   // break the loop if counter == 5
   printf("break the loop if counter is 5: ");
   for (counter = 0; counter <= 10; counter++){
      if (counter == 5)
         break;                        // break
      printf("%d ", counter);
   }

   // skip printing number 5
   printf("\ncontinue the loop if counter is 5: ");
   for (counter = 0; counter <= 10; counter++){
      if (counter == 5)
         continue;                     // skip 1 iteration
      printf("%d ", counter);
   }

   return 0;                           // return code success
}
```

```
//*************************************************************************
//  example4_9.c
//
//  Test the
//**********
#include <st

int main()
{
   int count

   // break the loop if counter == 5
   printf("break the loop if counter is 5: ");
   for (counter = 0; counter <= 10; counter++){
      if (counter == 5)
         break;                    // break
      printf("%d ", counter);
   }

   // skip printing number 5
   printf("\ncontinue the loop if counter is 5: ");
   for (counter = 0; counter <= 10; counter++){
      if (counter == 5)
         continue;                 // skip 1 iteration
      printf("%d ", counter);
   }

   return 0;                       // return code success
}
```



4.10 – Logical Operators

- So far, we have been using single condition expressions such as: <, >, <=, >=, ==, and !=
- To combine multiple conditions, use the logical operators && (logical AND), || (logicall OR), or ! (logical NOT or logical negation).

4.10 – Logical Operators cont.

- With logical OR, only one of the condition expression needs to be true for the result to be true

expression 1	expression 2	expression1 || expression2
true	true	true
true	false	true
false	true	true
false	false	false

- C uses the so called "short cut circuit" to evaluate logical expressions:
  - if the first expression is true, the result is true
  - If the first expression is false, the result depends on the evaluation of the 2nd expression

4.10 – Logical Operators cont.

- The logical OR examples:

expression 1	expression 2	expression1 || expression2
true	true	true
true	false	true
false	true	true
false	false	false

```
int credits = 10;
int points = 12;

if (credits >= 10 || points < 0) // the 1st expression is true
  printf ("true.");        // will execute this statement

if (credits > 10 || 5)     // the 2nd expression is true
  printf("true");          // will execute this statement

if (0 || credits > 10 )    // both expressions are false
  printf("false");         // will not execute this statement
```

4.10 – Logical Operators cont.

- With logical AND, all the condition expressions have to be true in order to the result to be true

expression 1	expression 2	expression1 && expression2
true	true	true
true	false	false
false	true	false
false	false	false

- C uses the so called “short cut circuit” to evaluate logical AND expressions:
  - if the first expression is false, the result is false
  - If the first expression is true, the result depends on the evaluation of the 2nd expression

4.10 – Logical Operators cont.

- Logical AND examples

expression 1	expression 2	expression1 && expression2
true	true	true
true	false	false
false	true	false
false	false	false

```
int ivar = 3,
    ivar2 = 4;

if (ivar > 0 && ivar < 10)          // both expressions are true
  printf("will be executed");

if (0 && ivar)                      // the 1st expression is false
  printf("will not execute this statement");

if ((ivar >= 3) && (ivar2 == (ivar + 2))) // the 2nd is false
  printf("will not be executed.");
```

4.10 – Logical Operators cont.

- With logical NOT (logical negation), the result of an expression is reversed.

expression	!expression
true	false
false	true

```
int ivar = 0;
if (ivar)                    // false
  printf("false");           // the statement won't be executed

if (!ivar)                   // true
  printf("true");            // execute this statement

ivar = 1;
if (ivar)                    // true
  printf("true");            // execute this statement

if (!ivar)                   // false
  printf("false");           // the statement won't be executed
```

4.10 – Logical Operators cont.

- Logical NOT has higher precedence than the operator==

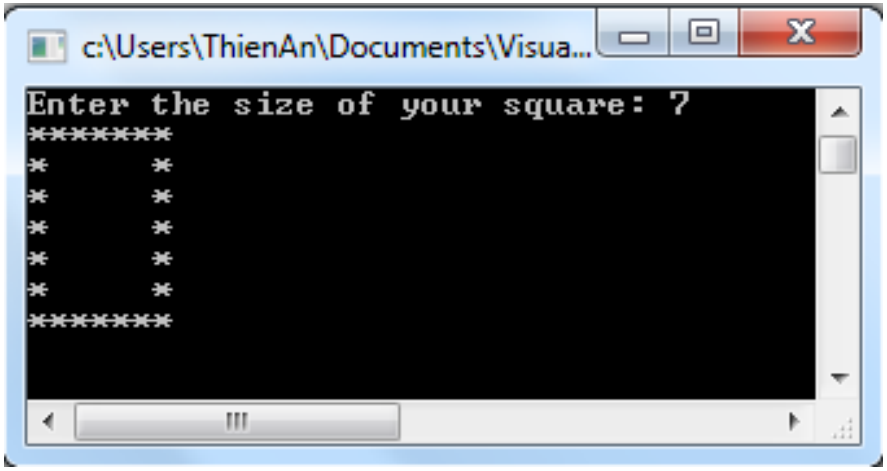
```
int credits = 10;
int points = 12;

if (!(credits == 10))      // false
  printf ("false.");       // this statement won't be printed

if (!(5 || credits > 10)) // false (negate true)
  printf("false");         // this statement won't be printed

if (!(0 || credits > 10)) // true (negate of both false)
  printf ("false");        // will be executed
```

4.10 – Logical Operators cont.

- Additional examples:
  - In the lab, we finished exercise 3.34, the hollow square of asterisks using the loop statements and single conditional expressions. The program was rather long with duplicate code fragments. Let's modify the program using logical OR.

4.10 – Logical Operators cont.

- Here is the algorithm using logical OR:
  - Loop through the rows
    - If it is the 1st OR last row
      - Print all columns with '*'
  - Otherwise (it is NOT the 1st OR last row)
    - loop through the columns
      - Print '*' only if it is the 1st OR the last column

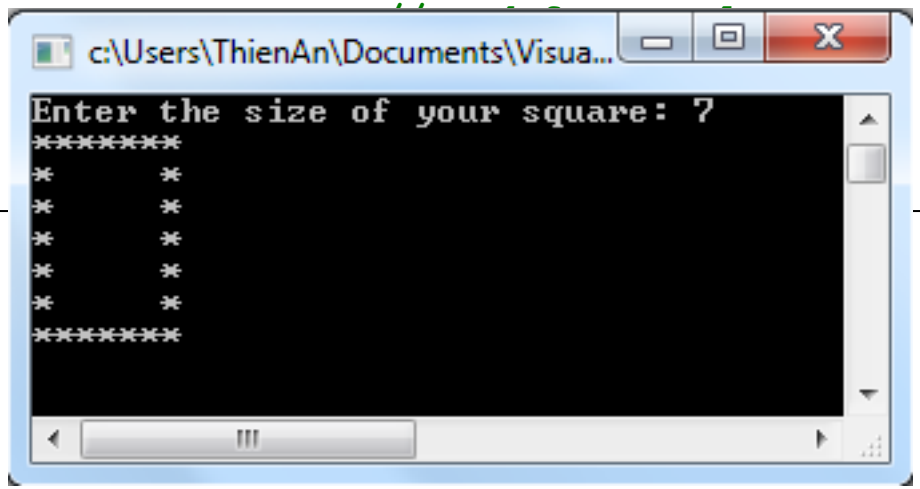


```
// ********************************************************
// Lab2.c
//
// Hollow Square of Asterisks with logical OR.
// ********************************************************
#include <stdio.h>

int main(void)
{
  int size;
  int row;
  int column;

  printf("Enter the size of your square: ");
  scanf("%d", &size);
```

```
  for (row = 1; row <= size; row++){ // for each row from 1st to last
    // check to see if it is the 1st row
    if (row == 1){
      for (column = 1; column <= size; column++){ // for each column
        printf ("*");              // print an asterisk (*)
      }
    } else if (row == size){       // or if it is the last row
      for (column = 1; column <= size; column++){ // for each column,
        printf ("*");              // print an asterisk - *
      }
    } else{                        // otherwise, it is not the 1st or last
      for (column = 1; column <= size; column++){ // for each column
         // check to see if it is the 1st column
        if (column == 1)
          printf ("*");            // print an asterisk
        else if (column == size)   // or if it is the last column
          printf ("*");            // print an asterisk
        else                       // otherwise,
          printf (" ");            // print a blank
       }                           // end for column loop
    }

    printf("\n");                  // new row
  }                                // end for row loop

  return 0;                        // return code success
}
```

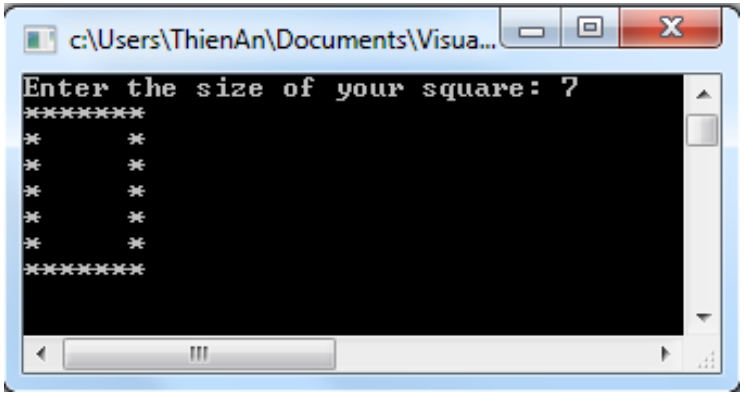
```
  for (row = 1; row <= size; row++){ // for each row from 1st to last
    // check to see if it is the 1st row or the last row
    if (row == 1 || row == size){
      for (column = 1; column <= size; column++){ // for each column
        printf ("*");              // print an asterisk (*)
      }
    } else if (row == size){       // or if it is the last row
      for (column = 1; column <= size; column++){ // for each column,
        printf ("*");              // print an asterisk - *
      }
    } else{                        // otherwise, it is not the 1st or last
      for (column = 1; column <= size; column++){ // for each column
        // check to see if it is the 1st OR the last column
        if (column == 1 || column == size)
          printf ("*");            // print an asterisk
        else if (column == size)   // or if it is the last column
          printf ("*");            // print an asterisk
        else                       // otherwise,
          printf (" ");            // print a blank
      }                            // end for column loop
    }

    printf("\n");                  // new row
  }                                // end for row loop

  return 0;                        // return code success
}
```

```
  for (row = 1; row <= size; row++){ // for each row from 1st to last
    // check to see if it is the 1st row or the last row
    if (row == 1 || row == size){
      for (column = 1; column <= size; column++){ // for each column
        printf ("*");              // print an asterisk (*)
      }
    } else {                       // otherwise, it is not the 1st or last
      for (column = 1; column <= size; column++){ // for each column
         // check to see if it is the 1st OR the last column
        if (column == 1 || column == size)
          printf ("*");            // print an asterisk
        else                       // otherwise,
          printf (" ");            // print a blank
       }                           // end for column loop
    }

    printf("\n");                  // new row
  }

  return 0;
}
```



4.10 – Logical Operators cont.

- Suppose, instead of the logical OR, we want to use a different logic to solve the problem, we use the logical AND.
- Here is the algorithm using the logical AND:
  - Loop through the rows
    - For each row, loop through the columns
      - If the row is between the 1st and the last, and the column is between the 1st and the last (exclusively), print a ' '
      - Otherwise, Print the '*'



4.10 – Logical Operators cont.

```
// ************************************************************
// Lab2.c
//
// Hollow Square of Asterisks with logical AND.
// ************************************************************
#include <stdio.h>

int main(void)
{
  int size;
  int row;
  int column;

  printf("Enter the size of your square: ");
  scanf("%d", &size);
```

```
  for (row = 1; row <= size; row++){ // for each row from 1 to side
    for (column == 1; column <= size; column++){
      // check to see if the row is in between the 1st and last
      if ((row > 1 && row < size) && (column > 1 && column < size)){
         printf (" ");                  // print an asterisk - *
      } else {                           // otherwise,
         printf ("*");                  // print an asterisk
      }
    }                                    // end for column loop

    printf("\n");                        // new row
  }                                      // end for row loop

  return 0;                              // return code success
}
```

End of slides.