COP 3223: C Programming
Spring 2009

Program Control Structures In C - Part 1

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

- Normally, statements in a program are executed one after the other in the order in which they appear in the code. This is called sequential execution.
- There are various C statements that we will soon see that enable the programmer to specify that the next statement to be executed may be other than the next on in sequence. This is called transfer of control.
- Historically speaking, it has been shown that all programs can be constructed in terms of only three types of *control structures*, namely the sequence structure, the selection structure, and the repetition structure.

Control Structures In C

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

1. The `if` selection statement either performs (i.e., selects) an action if a condition is true or skips the action if the condition is false. This is called a single-selection statement because it selects or ignores a single action.

2. The `if…else` selection statement performs an action if a condition is true and performs a different action if the condition is false. This is called a double-selection statement because it selects between two different actions.

3. The `switch` selection statement performs one of many different actions depending on the value of an expression. This is called a multiple-selection statement because it selects among many different actions.

Control Structures In C

- C provides three types of repetition structures in form of statements (we'll look at these in the next set of notes).

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.

Control Structures In C

- So that’s it, C provides you with a total of only seven control statements: (1) the structure, (3) selection statements, and (3) repetition statements.

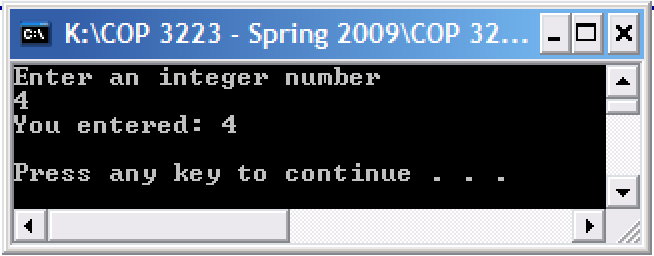
- Each C program is formed by combining as many of each type of control statement as is appropriate for the algorithm (the set of operations required to solve the problem at hand) the program implements.

- As we will see, these control structures are simply executed in the order they are placed in the program by the programmer. This is called control-statement stacking and with the added capability of control-statement nesting (we’ll see this later), all C programs are constructed using only these control structures combined in only these two ways.

The Sequence Structure

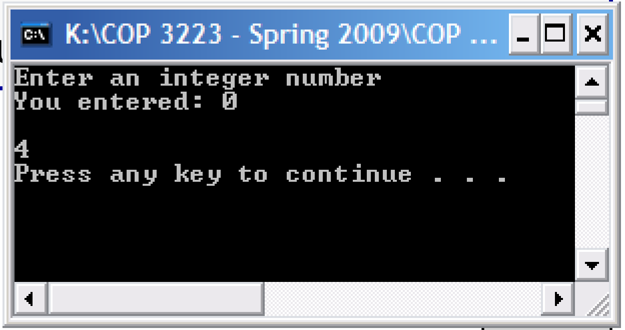
- The sequence structure of a C program is simply the order in which the various statements in the program appear.
- While the sequence structure of a program may not seem like a serious issue, it is in fact!
- Consider the two programs shown on the next page. These two program differ only in their sequence structure, however, one of the programs will execute successfully and print the number the user enters as input and the other will always print 0.
- Why? {The one on the left works correctly.}

```
// simple program to read and print a single
integer
// January 17, 2009   Written by: Mark
Llewellyn

#include <stdio.h>

//main function
int main()
{
    int integer1 = 0;  //user entered integer

    printf("Enter an integer number\n");
    scanf("%d", &integer1);
    printf("You entered: %d\n\n", integer1);
    system("PAUSE");

    return 0;
}  //end main function
```



```
Enter an integer number
4
You entered: 4

Press any key to continue . . .
```

```
// simple program to read and print a single
integer
// January 17, 2009   Written by: Mark
Llewellyn

#include <stdio.h>

//main function
int main()
{
    int integer1 = 0;  //user entered integer

    printf("Enter an integer number\n");
    printf("You entered: %d\n\n", integer1);
    scanf("%d", &integer1);
    system("PAUSE");

    return 0;
}  //end main fu
```



```
Enter an integer number
You entered: 0

4
Press any key to continue . . .
```

The `if` Selection Statement

- The format of the `if` selection statement is:

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

These statements are executed only if the condition evaluates to true, followed by statement x.

- When the `if` statement is executed, the condition is evaluated and if the condition evaluates to true, the statements in the body of the `if` statement are executed, followed by `statement x`. If the condition evaluates to false, the next statement to be executed will be `statement x`.

The `if` Selection Statement

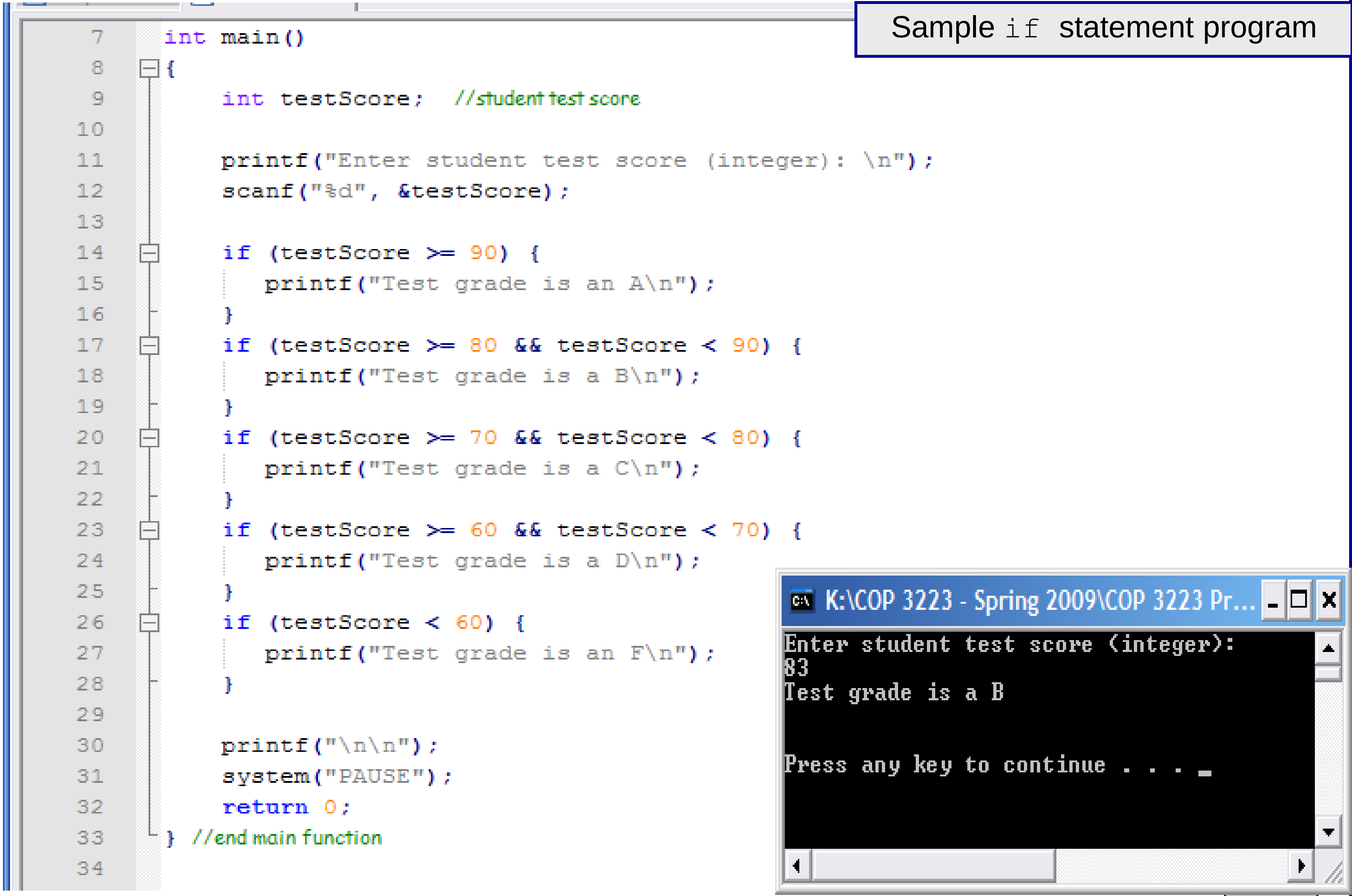
GOOD PROGRAMMING PRACTICE:

There are several common indentation styles that can be used with the `if` statement. See the style information on the course assignment page for specific options.

Whichever style you choose, be consistent throughout the source code to enhance the readability of your code.

- The program on the following page uses a series of if statement to determine which grade a student should be assigned based on their test score which is entered as input.

Sample `if` statement program



```c
int main()
{
    int testScore;   //student test score

    printf("Enter student test score (integer): \n");
    scanf("%d", &testScore);

    if (testScore >= 90) {
        printf("Test grade is an A\n");
    }
    if (testScore >= 80 && testScore < 90) {
        printf("Test grade is a B\n");
    }
    if (testScore >= 70 && testScore < 80) {
        printf("Test grade is a C\n");
    }
    if (testScore >= 60 && testScore < 70) {
        printf("Test grade is a D\n");
    }
    if (testScore < 60) {
        printf("Test grade is an F\n");
    }

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

```
Enter student test score (integer):
83
Test grade is a B


Press any key to continue . . .
```

The `if…else` Selection Statement

- The program on the previous page correctly prints the students letter grade based on the exam score that is input.

- However, the structure of this program is not very efficient from a processing point of view, because every `if` statement condition is evaluated even if the student's letter grade has already been determined.

- Consider, for example, a student who scored a 92 on the exam. Clearly, the first `if` statement's (line 14) condition evaluates to true and the letter grade of A is printed. The next statement to be executed is the `if` statement on line 17 and its condition will evaluate to false. The next statement to be executed is the `if` statement on line 20 and its condition will evaluate to false, and so on. They are all executed!

The if…else Selection Statement

- In cases such as this where once a condition is true and all others will be false, we need a more efficient way for the program to execute.
- The if…else selection statement is one possibility that can be used to solve our problem.
- Recall that the if…else statement is a double-selection statement in that it selects from two actions. The if statement is a single-selection statement.

The if…else Selection Statement

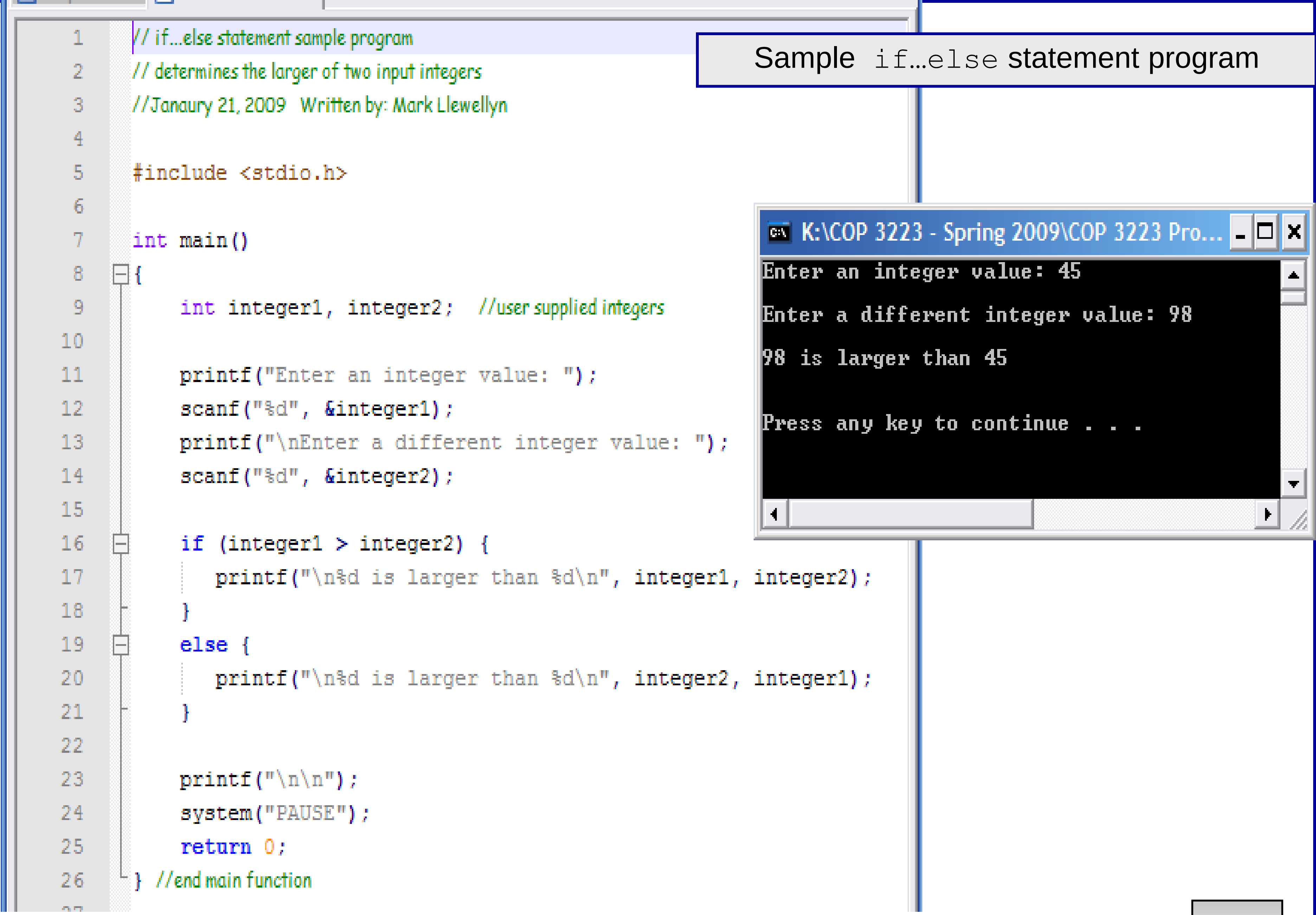
- The if…else selection statement has the following format:

```
if ( condition ) {
      statements
}
else {
      statements
}
 statement x;
```

These statements are executed only if the condition evaluates to true, followed by statement x.

These statements are executed only if the condition evaluates to false followed by statement x.

Sample if…else statement program



```c
// if...else statement sample program
// determines the larger of two input integers
//Janaury 21, 2009   Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int integer1, integer2;   //user supplied integers

    printf("Enter an integer value: ");
    scanf("%d", &integer1);
    printf("\nEnter a different integer value: ");
    scanf("%d", &integer2);

    if (integer1 > integer2) {
       printf("\n%d is larger than %d\n", integer1, integer2);
    }
    else {
       printf("\n%d is larger than %d\n", integer2, integer1);
    }

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

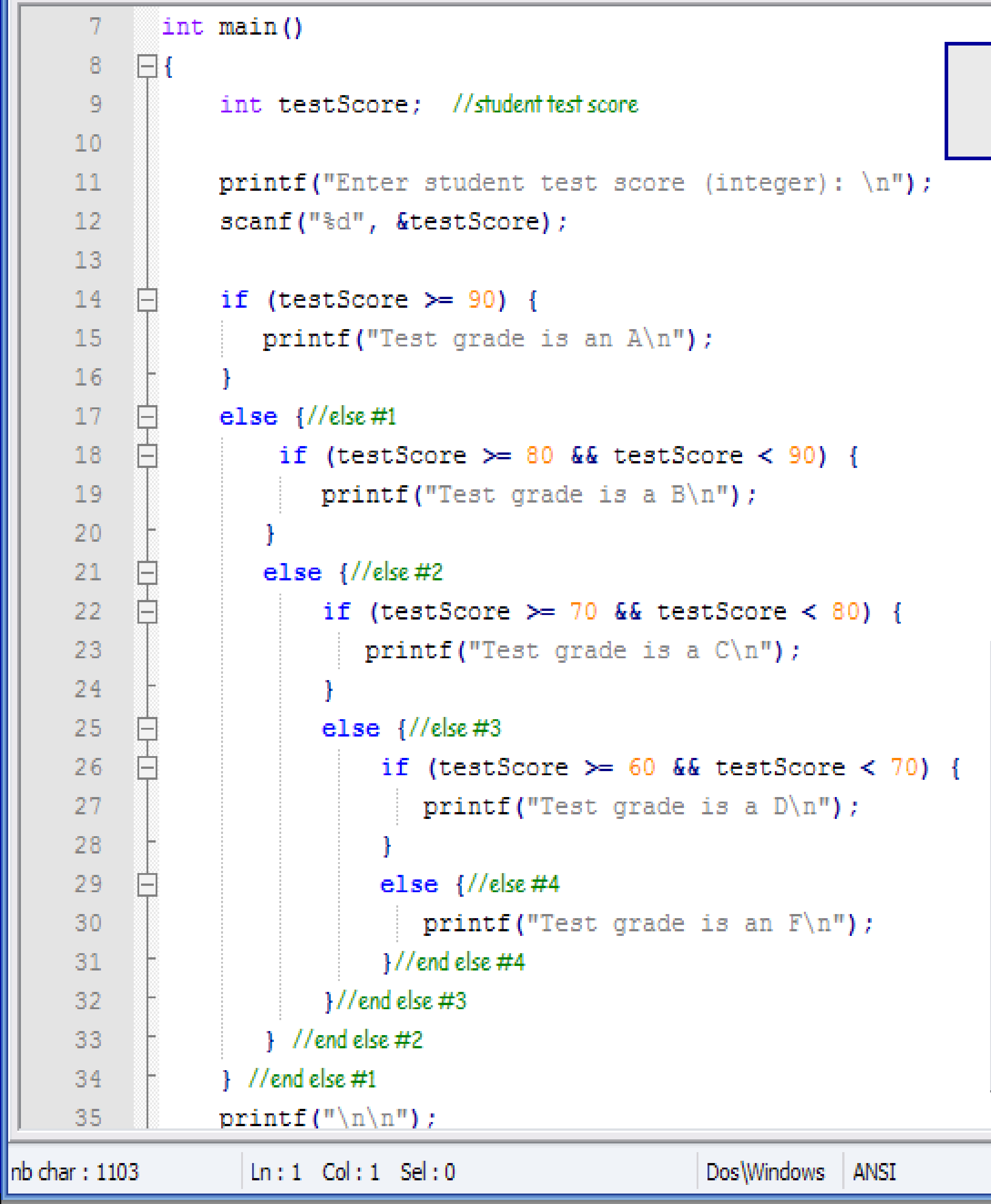
```
Enter an integer value: 45

Enter a different integer value: 98

98 is larger than 45


Press any key to continue . . .
```

The `if…else` Selection Statement

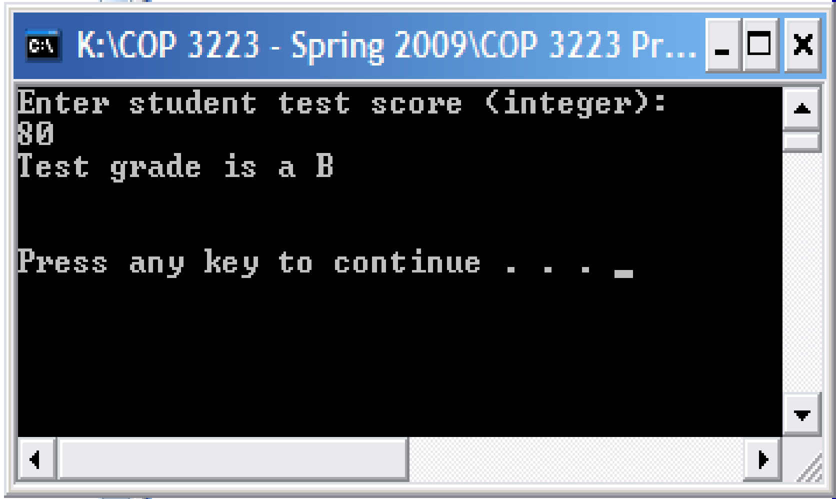
- Now let’s re-write the program on page 10 that determined a student’s letter grade using if-else statements to make the execution of the program more efficient.
- In this case notice that the statements that are included in most of the else clauses are if statements. There is no restriction on the type of statement that can belong inside these clauses. In fact in the program on page 10, we could have written the if statement on line 17 as:

```
    if (testScore >= 80) {
        if (testScore < 90) {
          printf(“Test grade is a B\n”);
        }
     }
```

Program from page 10 modified using `if…else` statements



```
int main()
{
    int testScore;  //student test score

    printf("Enter student test score (integer): \n");
    scanf("%d", &testScore);

    if (testScore >= 90) {
       printf("Test grade is an A\n");
    }
    else {//else #1
        if (testScore >= 80 && testScore < 90) {
           printf("Test grade is a B\n");
       }
       else {//else #2
           if (testScore >= 70 && testScore < 80) {
              printf("Test grade is a C\n");
           }
           else {//else #3
               if (testScore >= 60 && testScore < 70) {
                  printf("Test grade is a D\n");
               }
               else {//else #4
                  printf("Test grade is an F\n");
               }//end else #4
           }//end else #3
       } //end else #2
    } //end else #1
    printf("\n\n");
```



```
Enter student test score (integer):
80
Test grade is a B


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

The `if…else` Selection Statement

- In contrast to the program on page 10, the program on the previous page will execute more efficiently.
- Again, consider a student who scores a 92 on the exam. In the new program using the `if…else` statement, when the condition on line 14 is evaluated and it evaluates to true, the `printf` statement on line 15 is executed, but the next statement to be executed is now the `printf` statement on line 35. No other conditions are evaluated, making the execution of the new program more efficient.

The `if…else` Selection Statement

GOOD PROGRAMMING PRACTICE:

As with the `if` statement, there are several common indentation styles that can be used with the `if…else` statement. See the style information on the course assignment page for specific options.

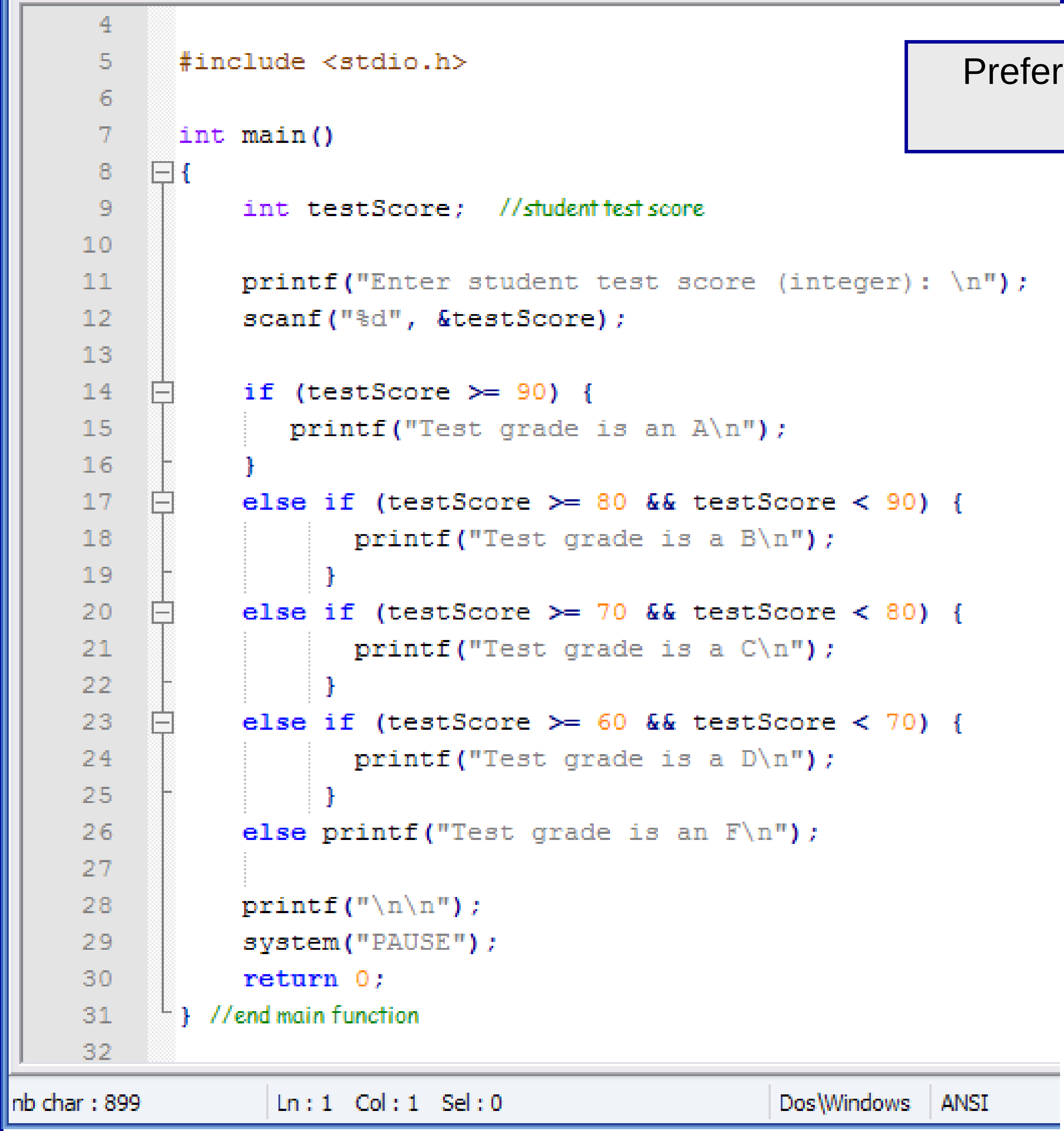
Whichever style you choose, be consistent throughout the source code to enhance the readability of your code.

GOOD PROGRAMMING PRACTICE:

The indentation style shown in the sample program on page 16 has certain disadvantages if (1) there are many nested if statements and/or (2) the statements in the body of the `if` or `else` clauses are fairly long. The problem is that the indentation becomes quite deep and causes statements to break across lines which decreases the readability of the program.

If many levels of nesting of `if…else` statements are used the style shown on the next page is actually the preferred style.

Preferred indentation style using nested if…else statements



```
#include <stdio.h>

int main()
{
    int testScore;   //student test score

    printf("Enter student test score (integer): \n");
    scanf("%d", &testScore);

    if (testScore >= 90) {
       printf("Test grade is an A\n");
    }
    else if (testScore >= 80 && testScore < 90) {
           printf("Test grade is a B\n");
         }
    else if (testScore >= 70 && testScore < 80) {
           printf("Test grade is a C\n");
         }
    else if (testScore >= 60 && testScore < 70) {
           printf("Test grade is a D\n");
         }
    else printf("Test grade is an F\n");

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

```

nb char : 899   Ln : 1   Col : 1   Sel : 0   Dos\Windows   ANSI

GOOD PROGRAMMING PRACTICE:

In selection statements (and also repetition statements that we'll see in the next set of notes), if the body of the selection clause is a single statement, e.g., the sample programs on pages 10, 16, and 19) then the braces are not necessary.

For example:

```
if (condition)
   statement;
```

and

```
if (condition) {
   statement;
}
```

are identical as far as the compiler is concerned. The code produced by the compiler is just as efficient with or without the braces.

You can decide whether or not to use the braces in single statement clauses. From a program maintainability and modifiability point of view, it is better to include them from the start. But as with other style considerations, be consistent in your choice.

However, anytime the clause contains more than one statement, the braces are required to form a block.

The `switch` Selection Statement

- The `switch` selection statement (a multiple-selection statement) is a versatile selection statement that is useful anytime a program needs to make a series of decisions in which a variable or an expression is tested separately for each of the constant integral values it might assume, and different actions are to be taken.
- The format of the `switch` selection statement is:

```
 switch ( expression ) {
 case labelx : statements;
              break;
. . .
 case labelz : statements;
              break;
 default : statements;
            break;
 } //end switch
```

Details Of The `switch` Selection Statement

- The `expression` of a `switch` selection statement must evaluate to the type `int` (integer).
- The `case` labels match the possible values to which the expression may evaluate. Each `case` label must be unique.
- After the expression is evaluated, control is passed to the appropriate case label (i.e., the one that matches the value of the expression).
- The case label `default`, which is optional, will be selected, if the expression evaluates to a value that does not match any of the other explicit case labels. There can be at most 1 default label in any `switch` statement.
- The C keywords `case` and `default` cannot occur outside of a `switch` statement in any C program.

Details Of The `switch` Selection Statement

COMMON PROGRAMMING ERROR:

Within a case of a `switch` statement, the `break` statement causes the execution control to pass to the next executable statement outside of the `switch` statement.

If the `break` statement is omitted within a case, the execution “falls through” to the next executable statement in the succeeding case.

Deliberately omitting a `break` statement can be a useful programming technique in certain cases, it is often simply overlooked, thus causing statements to be executed that the programmer did not intend to be executed, thus causing the program to produce incorrect results.

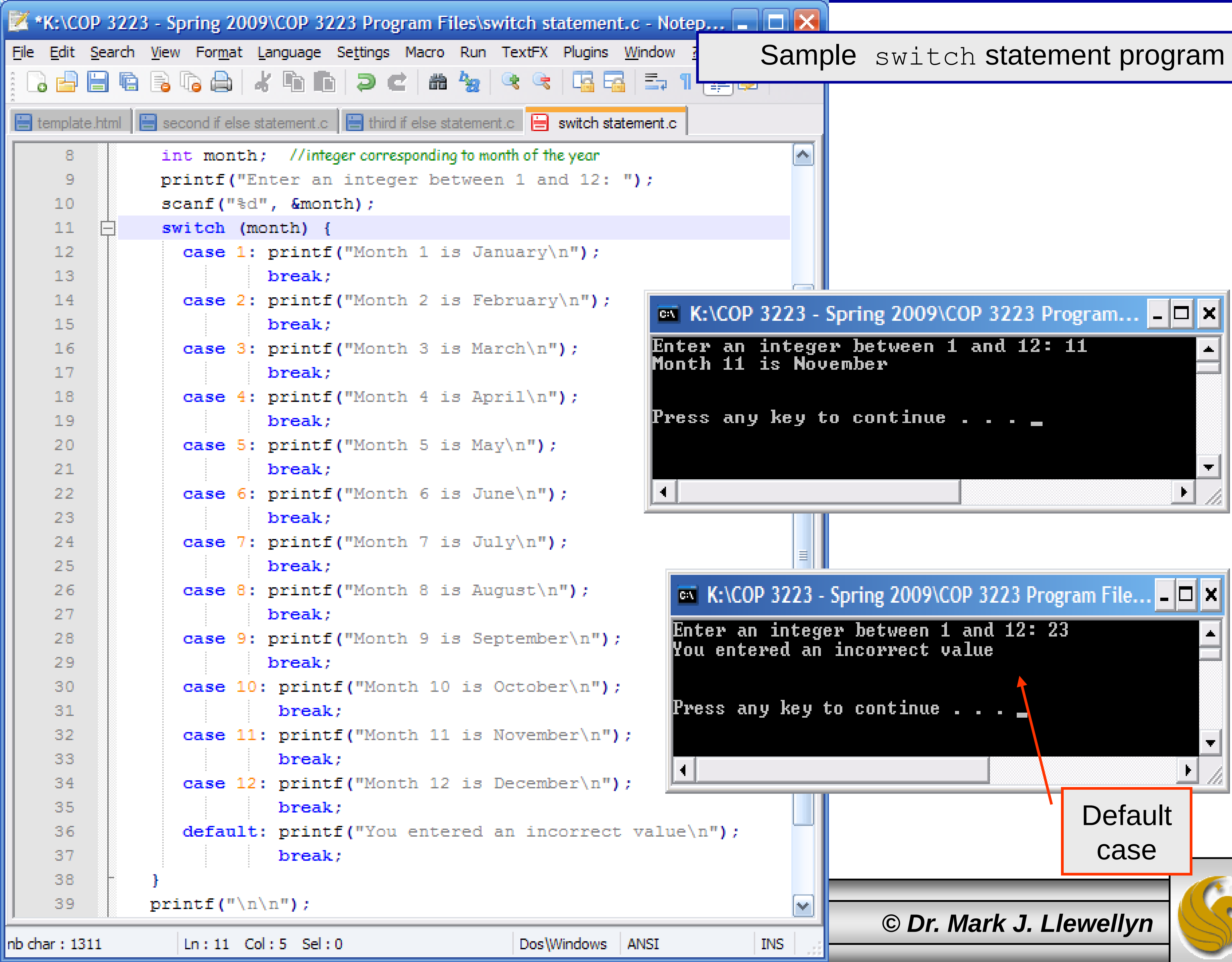
Missing `break` statements are NOT caught by the compiler!

Be very careful with `break` statements inside switch cases in C!

An Example Program With A `switch` Statement

- The following program will ask the user to enter an integer number between 1 and 12, which will correspond to the months of the year.
- Using a switch statement the program will print the correct month of the year.
- The default case will be used as an error trap in this sample program, which is often the case for how the default case is used.

Sample switch statement program

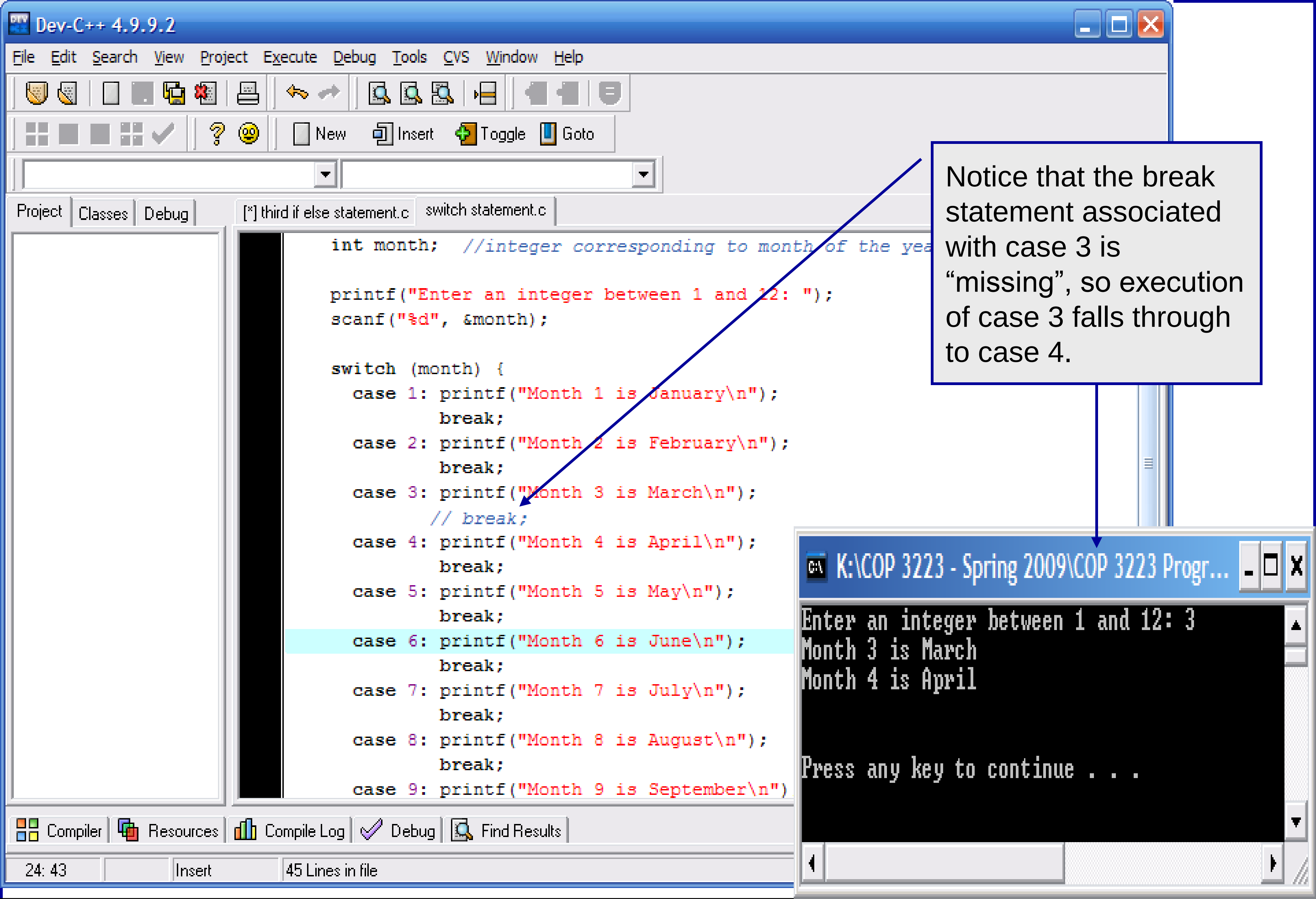

*K:\COP 3223 - Spring 2009\COP 3223 Program Files\switch statement.c - Notep...
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window
template.html
second if else statement.c
third if else statement.c
switch statement.c
nb char : 1311
Ln : 11 Col : 5 Sel : 0
Dos\Windows
ANSI
INS

```c
    int month;  //integer corresponding to month of the year
    printf("Enter an integer between 1 and 12: ");
    scanf("%d", &month);
    switch (month) {
      case 1: printf("Month 1 is January\n");
              break;
      case 2: printf("Month 2 is February\n");
              break;
      case 3: printf("Month 3 is March\n");
              break;
      case 4: printf("Month 4 is April\n");
              break;
      case 5: printf("Month 5 is May\n");
              break;
      case 6: printf("Month 6 is June\n");
              break;
      case 7: printf("Month 7 is July\n");
              break;
      case 8: printf("Month 8 is August\n");
              break;
      case 9: printf("Month 9 is September\n");
              break;
      case 10: printf("Month 10 is October\n");
               break;
      case 11: printf("Month 11 is November\n");
               break;
      case 12: printf("Month 12 is December\n");
               break;
      default: printf("You entered an incorrect value\n");
               break;
   }
   printf("\n\n");
```

```
Enter an integer between 1 and 12: 11
Month 11 is November


Press any key to continue . . .
```

```
Enter an integer between 1 and 12: 23
You entered an incorrect value


Press any key to continue . . .
```

Default case


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
[*] third if else statement.c
switch statement.c
Compiler Resources Compile Log Debug Find Results
24: 43
Insert
45 Lines in file
K:\COP 3223 - Spring 2009\COP 3223 Progr...

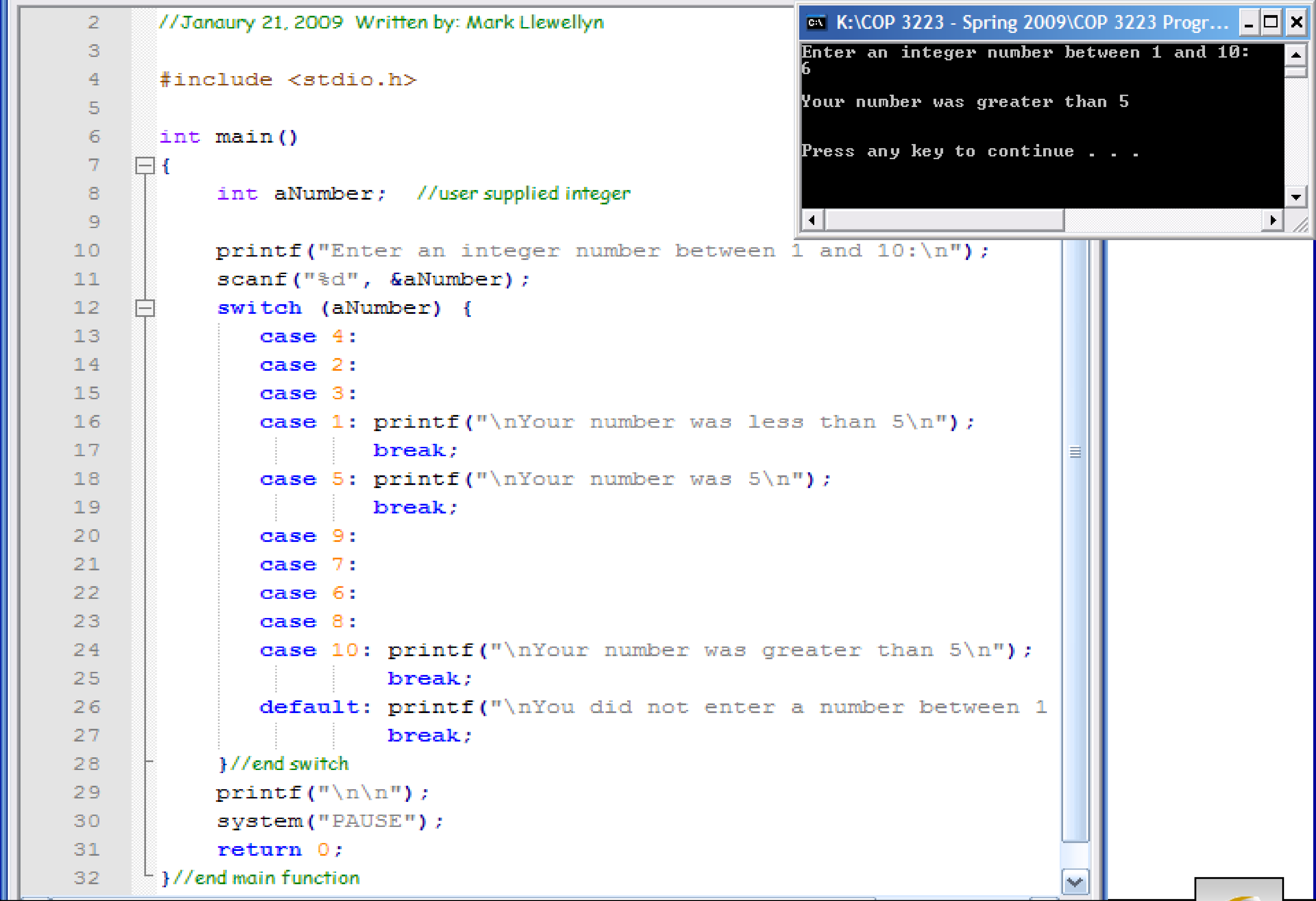
```
    int month;  //integer corresponding to month of the yea

    printf("Enter an integer between 1 and 12: ");
    scanf("%d", &month);

    switch (month) {
      case 1: printf("Month 1 is January\n");
              break;
      case 2: printf("Month 2 is February\n");
              break;
      case 3: printf("Month 3 is March\n");
            // break;
      case 4: printf("Month 4 is April\n");
              break;
      case 5: printf("Month 5 is May\n");
              break;
      case 6: printf("Month 6 is June\n");
              break;
      case 7: printf("Month 7 is July\n");
              break;
      case 8: printf("Month 8 is August\n");
              break;
      case 9: printf("Month 9 is September\n")
```

```
Enter an integer between 1 and 12: 3
Month 3 is March
Month 4 is April


Press any key to continue . . .
```

Notice that the break statement associated with case 3 is “missing”, so execution of case 3 falls through to case 4.

Combing Cases In A `switch` Statement

- Sometimes within a `switch` statement, you want the same action to occur for more than one case value. You do not need to copy the code more than once, rather you "combine" the cases by careful omissions of the `break` statement.
- The program shown on the next page illustrates this technique.
- This program also shows, that while the programmer may use some logical ordering to the case values, it makes no difference to the compiler the order in which the cases appear within the `switch` statement. However, from a readability point of view it is always best to apply some logical ordering to the case values. Ascending or descending order are the most common.



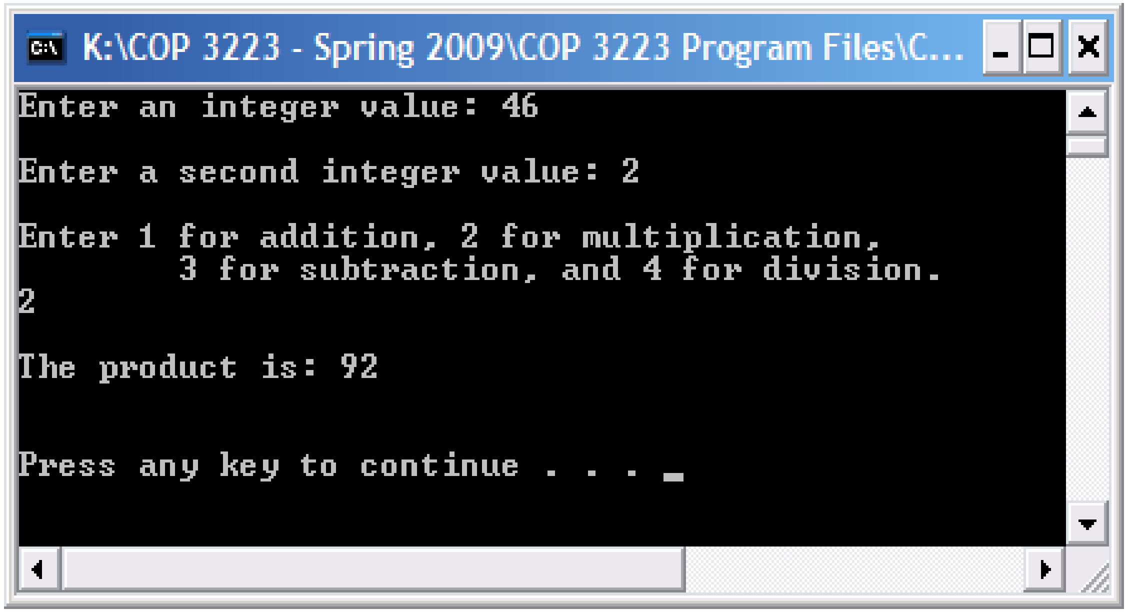
```c
//Janaury 21, 2009  Written by: Mark Llewellyn

#include <stdio.h>

int main()
{
    int aNumber;   //user supplied integer

    printf("Enter an integer number between 1 and 10:\n");
    scanf("%d", &aNumber);
    switch (aNumber) {
       case 4:
       case 2:
       case 3:
       case 1: printf("\nYour number was less than 5\n");
               break;
       case 5: printf("\nYour number was 5\n");
               break;
       case 9:
       case 7:
       case 6:
       case 8:
       case 10: printf("\nYour number was greater than 5\n");
                break;
       default: printf("\nYou did not enter a number between 1
                break;
    }//end switch
    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

K:\COP 3223 - Spring 2009\COP 3223 Progr...

```
Enter an integer number between 1 and 10:
6

Your number was greater than 5

Press any key to continue . . .
```

Practice Problems

1. Construct a C program that uses a `switch` statement in the following manner: ask the user to enter two integer numbers and then ask them to enter a third number where the third number is a 1, if they want to add the first two numbers together, a 2 if they want to multiply the first two numbers, a 3 if they want to subtract the first number from the second number, and a 4, if they want to divide the first number by the second number.

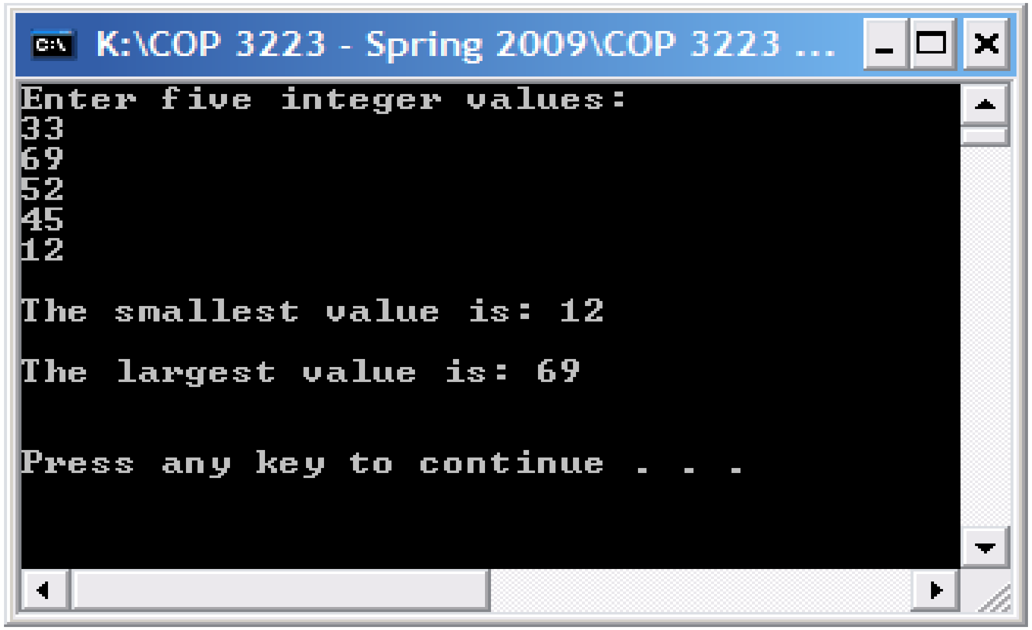


```
Enter an integer value: 46

Enter a second integer value: 2

Enter 1 for addition, 2 for multiplication,
        3 for subtraction, and 4 for division.
2

The product is: 92


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

Practice Problems

2. Construct a C program that uses `if` statements to determine which is the smallest and which is the largest of five integer values entered by the user.

3. Construct a C program that uses `if…else` statements to determine which is the smallest and which is the largest of five integer values entered by the user.



```
Enter five integer values:
33
69
52
45
12

The smallest value is: 12

The largest value is: 69


Press any key to continue . . .
```