ECE 131 – Programming Fundamentals

Module 3, Lecture 2: Program Flow Control – Conditional Statements

Dr. Daryl O. Lee

University of New Mexico

Conditional Statements

The three types of conditional statements in C are:

- if-else statement
- switch statement
- conditional operator (?:)

The if-else Statement

The C syntax for the if-else statement is:

if (*expr*)
    program block B_1
<else
    program block B_2>

- Where *expr* is any valid C expression.
  - If *expr* evaluates to any nonzero value (true), program block B_1 will be executed.
  - If *expr* evaluates to 0 (false), program block B_1 will *not* be executed; however, if the optional else portion of this statement is provided, program block B_2 will be executed.
- After execution of the if statement (which may result in no program blocks being executed), control proceeds to the statement immediately following the if-else statement.

THE UNIVERSITY of NEW MEXICO

The if-else Statement

Ex. The following program determines if a user-entered number is even or odd:

```
#include <stdio.h>
int main() {
   int a;
   printf("Input an integer number:  ");
   scanf("%d", &a); // accept input from keyboard
   if (a%2 == 0)
      printf("%d is an even number\n", a);
   else
      printf("%d is an odd number\n", a);
}
```

The if-else Statement

Let's add some error checking, this is a common use for the if-else construct:

```
#include <stdio.h>
int main() {
    int a;
    printf("Input an integer number:  ");
    if (scanf("%d", &a) == 0) {
        printf("Error:  not an integer\n");
        return 1; // an error
    }
    if (a%2 == 0)
        printf("%d is an even number\n", a);
    else
        printf("%d is an odd number\n", a);
    return 0; // no error
}
```

THE UNIVERSITY of NEW MEXICO

The if-else Statement

- Notice that the if statement in the previous example executes only the single program statement that follows it.
- Important: If you want more than one statement to be executed as part of the if statement, you must enclose them in brackets. I.e., you must create a block.
  Ex: What does the following print?

```
int main() {
   int a=2, b=3;
   if (a == b) {
      printf("a equals b\n");
      a = 5;
   }
   printf("a = %d\n", a);
}
```

  Does the output change if we add these (red) curly braces? Yes!

- I've been using the curly braces sparingly due to limited slide space. Best practice is to always use them. Why?

THE UNIVERSITY of NEW MEXICO

Nested if-else Statements

The following function computes the sign of a number (note that 0 is neither positive nor negative).

```
int sign(double num) {
   if (num < 0) // num is negative
      return -1;
   else // num is nonnegative
      if (num == 0)
         return 0;
      else
         return 1;
}
```

- Notice that the second if statement is actually the program block that goes with the else portion of the first if statement.

Nested if-else Statements

The previous function is often written as follows:

```
int sign(double num) {
   if (num < 0) // num is negative
      return -1;
   else if (num > 0) // num is positive
      return 1;
   return 0; // num is 0
}
```

- Unlike some programming languages, there is no "elseif" keyword in C. As we can see from the above, we don't really need it.
- Thus, the space between the "else" and the "if" is required in C programs.

THE UNIVERSITY of NEW MEXICO

The switch Statement

- We can chain many such "else if" constructs together in order to check a large number of possible values, and direct the flow of the program accordingly.
- However, C provides another conditional statement that you can use in some cases called the switch statement.
- This conditional statement allows you to direct program flow to any one of a number of cases, based upon the value of an expression evaluated at run-time.

The switch Statement

The syntax of the switch statement is as follows:

```
switch (expr) {
   case constant expr1:
        program statements
   case constant expr2:
        program statements
   ⋮
   case constant exprN:
        program statements
   default:
        program statements
}
```

- First, expr is evaluated, and then program control jumps to the case whose constant expression value equals that of expr. If no match is found, control jumps to the label default: if it exists, or to the end of the switch block if it does not.

THE UNIVERSITY of NEW MEXICO

The switch Statement

- Important: Once program control jumps to a given case, every statement that follows that case will be executed, including those that fall under the following cases. This is called "falling through".
- For this reason the break statement is commonly used with switch statements. The break statement causes the execution of the switch statement to immediately stop, and control is passed to the statement that immediately follows the switch statement.
- Because of the way the switch statement works, it's not necessary to enclose the program statements associated with a given case in curly braces. Using braces in switch cases is sometimes used for introducing temporary variables, following standard C scope rules.

The switch Statement – Example

```
int main() {
   float operand1, operand2;
   char operator;
   printf("Type a binary expression:  ");
   scanf("%f %c %f", &operand1, &operator, &operand2);
   switch(operator) {
      case '+':   printf(" %.2f\n", operand1 + operand2);
                  break;
      case '-':   printf(" %.2f\n", operand1 - operand2);
                  break;
      case '*':   printf(" %.2f\n", operand1 * operand2);
                  break;
      case '/':   if (operand2 == 0)
                     return 0;
                  else
                     printf(" %.2f\n", operand1 / operand2);
                  break;
      default:    return 0;
   }
   return 0;
}
```

THE UNIVERSITY of NEW MEXICO

The Conditional Operator

The C syntax for the conditional operator is:

expr1 ? *expr2* : *expr3*

First, expression *expr1* is evaluated, if *expr1* evaluates to true, then *expr2* is evaluated, and this become the result of the entire operation; otherwise, *expr1* is false, *expr3* is evaluated, and this become the result of the entire operation.

- Notice that this is a ternary operator.
- Ex. The following function computes the absolute value of a number:

```
double abs(double num) {
   return num < 0 ?  -num :  num;
}
```

THE UNIVERSITY of NEW MEXICO

The Conditional Operator

The previous sign function could be rewritten as:

```
int sign(double num) {
   return num < 0 ?  -1 :  num > 0 ?  1 :  0;
}
```

- The ?: operator has lower precedence than the $<$ and $>$ operators, and it associates right-to-left.
- Thus, the order in which the operators in the above expression are evaluated is as follows:

```
(num < 0) ?  -1 :((num > 0) ?  1 :  0)
```

- This produces the correct result.

The break and continue Statements

The following two statements can modify the flow of control through any of the loop statements we have discussed:

- The break statement causes the immediate termination of a loop.
  - Upon encountering a break statement, the flow of control proceeds to the statement immediately following the loop.
  - If a break statement is executed within a set of nested loops, only the innermost looping containing the break statement is terminated.
- The continue statement causes the termination of the current iteration of a loop, control then proceeds to the next iteration of that same loop. I.e., it breaks the control flow out of the iteration, but not the loop.
- You can usually rework the logic of a program in order to avoid these statements— use them judiciously!

THE UNIVERSITY of NEW MEXICO

The break and continue Statements

Ex. In the following program, if a valid integer number is not entered, the control will skip the remainder of the loop, and move onto the next iteration:

```
#include <stdio.h>
int main() {
    char ch;
    int num;
    while (1) { // an infinte loop, break with ^C
        printf("Input an integer number -> ");
        if (scanf("%d", &num) == 0) {
            printf("Error:  not an integer\n");
            ch = getchar(); // flush the buffer
            continue;
        }
        printf("Thanks for entering an integer\n");
    }
}
```

Logical Operators

The logical operators are often used to create more complicated Boolean expressions used in the loop and if statements.

- && – logical AND operation. A binary operator that returns a 1 (true) if both of its operands are true.
- || – logical OR operation. A binary operator that returns a 1 (true) if either of its operands are true.
- ! – logical NOT operation. A unary operator that returns the negation of its operand. I.e. a 1 (true) if the operand is false, and a 0 (false) if the operand is true.

Ex: Let `a`, `b`, `c` and `d` be any valid C expressions.

- `(a && b) || (c && d)`
  true if either both `a` and `b` are true, or both `c` and `d` are true.
- `a && !a`
  true if both `a` and not `a` are true, i.e., it's always false.

THE UNIVERSITY of NEW MEXICO

Logical Operators

Ex. This loop will continue to execute as long as either a "Y" or "y" is entered:

```
char ans;
int num;
do {
   printf("Enter an integer number -> ");
   scanf("%d", &num);
   getchar(); // flush the buffer
   // program statements
   printf("Enter another number (Y or N) -> ");
   ans = getchar();
   getchar(); // flush the buffer
} while (ans == 'Y' || ans == 'y');
```


