Relational and Logical Operators in C

CMSC 104 Section 2
March 4, 2024

Administrative Notes

Classwork 4 due

Homework 4 released today

Classwork 5 on Wednesday - please do not fall behind

Quiz #2 will be next Wednesday - the last class before Spring Break

A review of Classwork 4 from last Wednesday

The program from that assignment

```
// What is the file we have to include for printf() and scanf()?

    int main() {
        char name[20];                  /* User's name (no spaces) */
        <data_type> heightInInches;     /* User's height in inches */
        <data_type> heightInCentimeters; /* User's height in centimeters */


        // Prompt for user's name


        // Prompt for user's height in inches


        // Calculate how many centimeters from number of inches


        // Greet the user and tell them how many centimeters tall they are


return 0; }
```

How to print out multiple variables in a single printf()

The last comment requires you to print out two values: the user's name; and the user's height in centimeters.

You can take the easy way out and use two separate printf statements; one for each variable.

A number of you tried:

```
printf("Hello, %20s ", nsme, "your height in cm is %f\n\n",heightInCentimeters);
```

And that would work in a lot of other languages, like Python. But the right way to do it in C:

```
printf("Hello, %20s, your height in cm is %f\n\n", name,heightInCentimeters);
```

Controlling field widths

- A number of students pointed out that the name is printed in 20 spaces, right-justified. If you use a five-letter name, you get 15 blank spaces in front and that doesn't look very good.
- That's the significance of "%20s". It tells C to use at least 20 spaces to print the string; right-justified and filled with blanks. If you don't want that, use a different number - e.g. "%10s" or don't use a number at all and take the default C behavior, which is to use exactly the number of spaces needed.
- It's the same principal with float and int. When you say "%1.5f" C will print at least 5 decimal spaces. If the value to print is "4", you're going to get 5 0's.
- What's the default for floats? printf("%f\n", 5) - how many decimal places get printed?

Now, new material - relational and logical operators

What are operators?

Symbols that cause actions to be taken to one or more values

- We already learned about arithmetic operators like -, +, *, / and %
- Now we'll talk about relational and logical operators
- Both types are used to produce ***Boolean*** values
- These values are used to control what parts of a program are executed

Boolean values

They have two values: `true`, or `false`

You can declare and use Boolean variables directly:

```
bool is_correct;

is_correct = true;
```

You can also calculate them, using the types of operators we'll talk about next

Relational operators

< - less than

> - greater than

<= - less than or equal to

>= - greater than or equal to

== - equal to

!= - not equal to

All of these result in boolean expressions - they evaluate to either true or false

NOTE: the difference between = and ==

= is an assignment statement; it puts the value on the right into the variable on the left. If you try to evaluate it as a boolean, it will always result in true

== is a test for equality; it returns true if the value of the quantity on the right is equal to the value of the quantity on the left. Otherwise it returns false. It does not assign any value to anything.

Logical Operators

&& - and . a && b is true if and only if both a and b are true. Otherwise it’s false

|| - or. a || b is false if and only if both a and b are false. Otherwise it’s true.

! - not. !a is the opposite of whatever a is.

A side note on arithmetic expressions

Any number that equals 0 is false

- The integer 0 is false
- The float 0.0 is false

Any other number, whether integer or float, is interpreted as true when you try to evaluate it as a boolean.

Conditional code

So why do we care so much about the value of all these operators? What's so important about returning true or false?

A lot of times, we want to write a program so that some code executes if and only if some expression is true - some "condition" holds

- We also may want code that executes as long as some condition is true, but that's Wednesday's lecture

In C, we implement this ***conditional code*** with `if` and `if-else` statements

Writing Programs

All programs can be written in terms of only three control structures:

1 Sequence: The statements are executed in the order in which they’re written.

2 Selection: Used to choose among alternative courses of action (also known as branching).

3 Repetition: Statements are repeated while some condition remains true.

`if` statements

The simplest form of conditional

Do something if a condition is true; do nothing if it's false

Syntax:

```
if (condition) {

 one_or_more_statements;

}
```

The condition must be something that evaluates to true or false; it's a boolean

These statements are called the body of the conditional

The curly braces {} can be omitted if there's only one statement in the body. But we'll be in the habit of always using them.

Example of an `if` statement

```
if (age >= 18) {

printf("Go vote!\n");

}
```

`if-else` statements

Slightly more complex: do one thing if a condition is true; do something else if that condition is false

Syntax:

```
if (condition) {
  one_or_more_statements;
} else {
  one_or_more_other_statements;
}
```

Example of if-else statement

```
if (value == 0) {
   printf("You cannot divide by zero.\n");
 } else {
   printf("The answer is %d\n", 100/value); }
```

Nesting `if-else` statements

What if there are more than two cases to consider? E.g. if you're 18 you can vote but you can't drink alcohol; if you're younger than 18 you can't do either; if you're above 21 you can do both.

This is handled by nesting if-else statements. In this case:

```
if (age<18){
    printf("Sorry, but you are still a minor \n")
} else if (age < 21){
    printf("You can vote but you can't drink \n")
} else {
    printf("Congratulations, you are now legally an adult. \n")
```

This can be as complex as you want

```
if ( condition1 ) {
     statement(s);
} else if (condition 2) {
     different statement(s);
} else if .... {
/* there can be any number of if-else
statements */
 } else {
statement(s); /* the default case */
}
```

```
if ( condition1 ) {
       statement(s);
} else if (condition 2) {
       different statement(s);
       if (condition 3) {
// conditions 2 and 3 are both true
       another statement();
       } else {
// condition 2 is true, but condition 3 is false
       yet another statement(); }
       } else { statement(s);
       /* the default case */ }
```