Programming Fundamentals  
Tutorial 03 - Relational Operators, Boolean Variables, Scope and if() Selection.

## Introduction

In this tutorial you will be introduced to the concept of logical true and false using boolean variables and relational operators. Simple selection using the if() statement based on boolean true and false conditions is discussed, along with potential logical errors that can occur. Variable scope is considered and additionally visually explained. Finally, you will see how to use pseudocode and flowcharts to design a simple algorithm using selection that can then be translated into and implemented as a program.

## Relational Operators

In order to do anything useful in a program there needs to be a way of changing what happens based on a specific condition, or set of conditions, being **true** or **false**. **Relational operators** are used to compare values in order to determine if that comparison evaluates to **true** or **false**. The most basic example is the **equality operator '**==**'**, which consists of two equals signs. So, for example:

(5 == 5) is true

(5 == 6) is false

Of course, comparing two literals is fairly pointless, so generally the relational operators are used in combination with variables. For instance, if you had a variable score that equalled 200, and hiscore that equalled 300, then:

(score == 200) is true

(score == 999) is false

(score == hiscore) is false

There are several other relational operators, and these are summarised in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operator | Meaning | Examples | | |
| == | equal to | (5 == 5) is true | (5 == 6) is false |  |
| != | not equal to | (5 != 6) is true | (5 != 5) is false |  |
| > | greater than | (5 > 4) is true | (4 > 5) is false |  |
| < | less than | (2 < 7) is true | (7 < 2) is false |  |
| >= | greater than or equal | (4 >= 3) is true | (3 >= 4) is false | (5 >= 5) is true |
| <= | less than or equal | (1 <= 9) is true | (9 <= 1) is false | (6 <= 6) is true |

## Relational Operators and Boolean Variables

In the previous tutorial you were shown a new type of variable called a boolean that represents the values true and false. You can use boolean variables to store the result of a comparison using relational operators:

int score = 200;

int hiscore = 300;

bool is\_score\_200;

bool big\_score;

is\_score\_200 = (score == 200); // is\_score\_200 is set to **true**;

big\_score = (score > hiscore); // big\_score is set to **false**;

Notice the different use of the equals signs in the first assignment statement. The single '=' sign represents an **assignment** that sets a variable while the double '==' sign is the **equality** relational operator doing a comparison. Confusing these two is a common cause of errors in C/C++.

## Relational Operators and **if()** Statements

The last section demonstrated how to store the result of comparisons using relational operators. In order to do something useful with a comparison you need to be able to test the result of the comparison and carry out some actions, or not, based this result. The way this is done is by using an if() statement. The if() statement works as follows:

if(*expression* is true)

{

// do something in this statement block

}

The ***expression*** can be anything that gives a true or false value and if it is found to be true then any code in the statement block following the if() is executed, otherwise it is skipped. Here are some examples, first with the equality operator '==':

if(true)

{

printf("this ALWAYS executes\n");

}

if(false)

{

printf("this NEVER executes\n");

}

bool my\_bool = true;

if(my\_bool == true) // "if my\_bool **is equal** to true"

{

printf("my\_bool is true, so this code executes\n");

}

if(my\_bool == false) // "if my\_bool **is equal** to false"

{

printf("my\_bool is true, so this DOESN'T execute\n");

}

and here with other operators:

int score = 500;

int hi\_score = 400;

if(score > hi\_score) // "if score **is greater than** hi\_score"

{

printf("You've got the new high score!\n");

}

int lives = 3;

if(lives != 0) // "if lives **is not equal** to zero"

{

printf("You're still alive!\n");

}

int my\_height = 175, your\_height = 175;

if(my\_height <= your\_height) // "if my\_height **is less than or equal to** to your\_height"

{

printf("You are not taller than me.");

}

If you remember in the last tutorial it was noted that printf() does not automatically output the text "true" and "false" for boolean variables. Here is an example of how you might do this:

bool my\_bool = true;

if(my\_bool == true) // "if my\_bool **is equal** to true"

{

printf("true\n");

}

if(my\_bool == false) // "if my\_bool **is equal** to false"

{

printf("false\n");

}

There are a couple of potential problems you may come across when using if() statements, especially if you are new to C/C++. One difficult to spot error is a semi-colon added after the if() statement, for instance:

if(false); // DON'T DO THIS!

{

printf("this should NEVER execute\n");

}

This code will compile and run, but unfortunately you will see "this should NEVER execute" displayed. This is because C/C++ allows you to use **empty statements**. This is a statement which does **nothing**, this may sound strange but there are uses for this in more advanced code. In C/C++, they are represented somewhat confusingly as a semi-colon on its own. For now all you need to know is that if you use the above code then the if() statement is associated with the empty statement represented by the semi-colon and it is the empty statement that doesn't get executed. Therefore it jumps to the next statement, which happens to be the printf() that should never be executed in the following statement block.

One of the biggest causes of logical errors in C/C++ programs is confusing '=' and '==':

int score = 400;

int hi\_score = 500;

if(score = hi\_score) // DON'T DO THIS! Use == in if() statements

{

printf("You've got the new high score!\n");

}

This code will also compile and run, but will not do as expected. Because the assignment operator has been used '=', instead of a comparison being made, the value of score is set to the value of hi\_score which is probably not what was intended.

Always check your if() and assignment statements are using the correct operator, especially if things aren't working as expected.

## Variable Scope

Now that you have been introduced to if() statements that have their own statement blocks it is a good time to introduce the idea of **variable scope**. Scope simply means where a variable can be seen or accessed by the program. There are two types of scope, **global** scope which is visible anywhere in a source file, or **local** scope which is limited to a particular statement block that is defined by a pair of braces. Here is a program that will show how scope works:

#include <stdio.h>

**int global\_integer = 10; // Visible throughout main.c**

int main()

{

**int local\_function\_integer = 3; // Visible throughout main() function statement block**

if(local\_function\_integer > 0)

{

**int local\_statementblock\_integer = 5; // Visible only in this block**

printf(“%d\n”,global\_integer);

printf(“%d\n”,local\_function\_integer);

printf(“%d\n”,local\_statementblock\_integer);

}

printf(“%d\n”,global\_integer);

printf(“%d\n”,local\_function\_integer);

//printf(“%d\n”,local\_statementblock\_integer); // Won't work, not in same scope

getchar();

return 0;

}

**// other functions - global\_integer still in scope**

Here you can see three different variables defined and used in separate parts of the program, each with a different scope. First is the global integer variable global\_integer which is accessible inside the main() function, inside the if() statement block, and if there were other functions it would also be visible there. Next is local\_function\_integer which is visible throughout the main() function including the if() statement block, but not elsewhere in the source file. Finally local\_statementblock\_integer is only visible inside the if() statement block and nowhere else. For instance, if you uncommented the line highlighted in yellow then the program would not compile because that variable is not accessible from there.

To better illustrate how scope works, here is the same program with colour coded scope - blue encompasses global scope, orange the local function statement block scope, and white local if() statement block scope:

#include <stdio.h>

**int global\_integer = 10; // Visible throughout main.c**

int main()

{

**int local\_function\_integer = 3; // Visible throughout main() statement block**

if(local\_function\_integer > 0)

printf(“%d\n”, global\_intege);

printf(“%d\n”, local\_function\_integer);

//printf(“%d\n”, local\_statementblock\_integer); // Won't work, not in same scope

\_getch();

return 0;

}

{

**int local\_statementblock\_integer = 5; // Visible only in this block**

printf(“%d\n”,global\_integer);

printf(“%d\n”,local\_function\_integer);

printf(“%d\n”,local\_statementblock\_integer);

}

**// other functions - global\_integer still in scope**

## **if()** Statements, Pseudocode and Flowcharts

Earlier tutorials introduced you to the ideas of inputs, operations and outputs, the concepts of pseudocode and flowcharts, and the processes of sequence and selection. In this section you will look at how you can go from a basic problem to a programmed solution using these techniques.

Let's design a program that fulfils a simple problem. "Ask the user to enter 2 numbers. Display both numbers. If both numbers are the same display a message saying so. Display the addition of both numbers."

First off let's consider the inputs. The problem starts with "Ask the user to enter 2 numbers". As it doesn't specify what sort of numbers, e.g. integer or float, it's down to you to choose an appropriate type based on the problem. As it doesn't seem to matter for this problem, integers are probably as good a choice as any. Let's call them integer1 and integer2. The outputs are the values of the two numbers input, a message saying they are equal, and the addition of the two numbers. The only other operation apart from this is the decision as to whether to display the message.

You could initially break the problem up into pseudocode like this:

1. **Input two numbers**
2. **Display both numbers**
3. **If both numbers are the same display message saying they are equal**
4. **Display addition of both numbers.**

You could then stepwise refine and add reference to specific inputs and outputs like so:

1. **Input two numbers**
   1. **Display message asking for first number.**
   2. **Input integer1**
   3. **Display message asking for first number.**
   4. **Input integer2**
2. **Display integer1 and integer2**
3. **If integer1 is equal to integer2 // (or "if integer1 == integer2 ")**
   1. **Display message saying they are equal**
4. **Display addition of integer1 and integer2 // (or "Display integer1+integer2")**

This could then be converted to a flowchart like this:

**START**

**Ask for first number**

**Input into**

**Integer1**

**Ask for second number**

**Input into**

**Integer2**

**Integer1 = = Integer2?**

**Display "EQUAL"**

**Display integer1 + integer2**

**END**

**FALSE**

**TRUE**

You might notice several new symbols in this flowchart. The first are the round boxes called **terminators**. There should be one **start** node with a single flow arrow out; this is obviously where the program begins. There should also be an **end** node with a single input flow arrow that shows where the program ends. Finally, the parallelogram represents any kind of **input** or **output** that happens in the program.

To finally convert this into a program, you need to look at each step of the pseudocode and/or flowchart and create lines of code that implement that step:

#include <sddio.h>

int main()

{

int integer1;

int integer2;

printf("Input first number\n");

scanf(“%d”, &integer1);

printf("Input second number\n");

scanf(“%d”, &integer2);

if(integer1 == integer2) // Selection

{

Printf("Both numbers are equal\n");

} // Note program flow rejoins here with connector

printf("Sum of both numbers is\n");

printf(“%d\n”, integer1 + integer2);

getchar();

return 0;

}

**end**

**Display "EQUAL"**

**Integer1 == Integer2?**

**TRUE**

**FALSE**

**start**

**Ask for first number**

**Input into**

**Integer1**

**Ask for second number**

**Input into**

**Integer2**

**Display integer1 + integer2**

As you can see, you have to add the 'scaffolding' that allows the compiler to understand the code such as the #includes and variable declarations, but generally the pseudocode represents the eventual code. It won't always be this straightforward, especially with much more complex programs, and you may need to rewrite or refine the pseudocode and flowchart several times before it can be easily converted into code, but, just as with programming, it gets easier with practice.

## Exercises

#### Exercise 01

Given the following variables:

int i = 16;

int j = 32;

int k = 8;

bool r;

Evaluate in your notebook the following expressions and state the value of r:

1. r = (i == 16);
2. r = (k != 16);
3. r = (k < 10);
4. r = (j > 33);
5. r = (j == (k \* 2));
6. r = (j == (i \* 2));
7. r = ((k + i) <= (j – i));
8. r = ((i / k) == 2);
9. r = ((( i + j + k ) / k ) >= 7 );

#### Exercise 02

1. Design using pseudocode and flowcharts a program that has a high score defined as 500, inputs a score for player 1 and player 2, outputs a message saying which player has the highest score, and if the highest player score is higher than the high score outputs another message saying so, as well as updating the high score.
2. Implement your design, and test that it works using the debugger.

#### Exercise 03

Write a program that has a global integer variable called test\_number that is initialised at declaration to 100. Inside main() add a local variable also called test\_number that is set to 200. Display the value of test\_number in your program - what is it? Can you give a reason why that value is shown instead of the other? Investigate why this is the case and explain in your notebook. Can you find a way of accessing the global variable rather than the local one?

#### Exercise 04

Write a program that **calculates** all of the expressions in Exercise 01 and displays them on screen, e.g.:

"r = (i == 16) is true"

As printf() doesn't display true or false automatically you will have to use if() statements to do it.

#### Exercise 05

Examine the following code:

if(false)

;

{

printf("This text won't be shown.\n");

}

What do you expect to happen here? Write down what you think before running the program. **COPY AND PASTE** the above code into main() and run the program. Does it match your expectations, if so write down why in your notebook, if not find out why and try to explain. Run through it using the debugger to get used to spotting this situation.

#### Exercise 06

Examine the following code:

int x = 5;

int y = 7;

if(x = y)

{

printf("x and y are equal\n");

}

What do you expect to happen here? Write down what you think before running the program. **COPY AND PASTE** the above code into main() and run the program. Does it match your expectations, if so write down why in your notebook, if not find out why and try to explain. Run through it using the debugger to get used to spotting this situation.

#### Exercise 07 and up

Design two short programs that have basic decisions in them, using pseudocode and flow diagrams, then convert them into actual code.