Lecture 5

If statements

The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. The **if statement** allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the if statement is that it allows the program to select an action based upon the user's input. For example, by using an if statement to check a user-entered password, your program can decide whether a user is allowed access to the program

Without a conditional statement such as the if statement, programs would run almost the exact same way every time, always following the same sequence of function calls. **if statements** allow the flow of the program to be changed, this leads to more interesting code.

Before discussing the actual structure of the if statement, let us examine the meaning of TRUE and FALSE in computer terminology. A true statement is one that evaluates to a nonzero number. A false statement evaluates to zero. When you perform comparison with the relational operators, the operator will return 1 if the comparison is true, or 0 if the comparison is false. For example, the check 0 = = 2 evaluates to 0. The check 2 = = 2 evaluates to a 1. If this confuses you, try to use a printf statement to output the result of those various comparisons (for example printf("%d",2 = =1);). When programming, the aim of the program will often require the checking of one value stored by a variable against another value to determine whether one is larger, smaller, or equal to the other.

There are a number of operators that allow these checks.
Here are the relational operators, as they are known, along with examples:

```
>     greater than              5 > 4 is TRUE
<     less than                 4 < 5 is TRUE
>=    greater than or equal     4 >= 4 is TRUE
<=    less than or equal        3 <= 4 is TRUE
==    equal to                  5 == 5 is TRUE
!=    not equal to              5 != 4 is TRUE
```

It is highly probable that you have seen these before, probably with slightly different symbols. They should not present any hindrance to understanding. Now that you understand TRUE and FALSE well as the comparison operators, let us look at the actual structure of if statements.
The structure of an if statement is as follows:

```
if ( statement is TRUE )
     {Execute this block of code;}
```

To have more than one statement execute after an if statement that evaluates to true, use braces, like we did with the body of the main function. Anything inside braces is called a compound statement, or a block. When using if statements, the code that depends on the if statement is called the "body" of the if statement.

For example:

```
 if ( TRUE ) {
  /* between the braces is the body of the if statement */
  Execute all statements inside the body;
}
```

It is recommended always putting braces following if statements. If you do this, you never have to remember to put them in when you want more than one statement to be executed, and you make the body of the if statement more visually clear.

Else

Sometimes when the condition in an if statement evaluates to false, it would be nice to execute some code instead of the code executed when the statement evaluates to true. The "else" statement effectively says that whatever code after it (whether a single line or code between brackets) is executed if the if statement is FALSE. It can look like this:

```
if ( TRUE ) {
  /* Execute these statements if TRUE */
}
else {
  /* Execute these statements if FALSE */
}
```

Else if

Another use of else is when there are multiple conditional statements that may all evaluate to true, yet you want only one if statement's body to execute. You can use an "else if" statement following an if statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if statement is false, it will then check the condition for the else if statement. If the if statement was true the else statement will not be checked. It is possible to use numerous else if statements to ensure that only one block of code is executed.

Let's look at a simple program for you to try out on your own.

```
#include <stdio.h>

int main()                              /* Most important part of the program!
*/
{
    int age;                            /* Need a variable... */

    printf( "Please enter your age" );  /* Asks for age */
    scanf( "%d", &age );                 /* The input is put in age */
    if ( age < 100 ) {                  /* If the age is less than 100 */
     printf ("You are pretty young!\n" ); /* Just to show you it works... */
  }
  else if ( age == 100 ) {              /* I use else just to show an example */
     printf( "You are old\n" );
  }
  else {
    printf( "You are really old\n" );     /* Executed if no other statement is
    */
  }
  return 0;
}
```

Lecturer: Mihail Kulev