UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

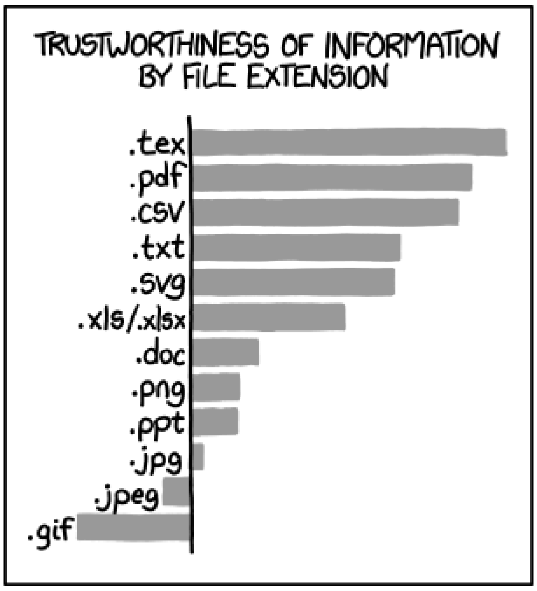
Computer Programming 143

Practical 4

2019

Aim of Practical 4:

- Complex Boolean expressions
- Functions



Instructions

1. Attendance is **compulsory** for all the practical sessions of your assigned group. See the study guide for more details.
2. The last section (usually the last 30 minutes) of the practical will be used for a test.
3. If more than two tests have been missed for what ever reason, you will receive an **incomplete** for the subject. See the study guide for more details.
4. You must do all assignments **on your own**. Students are encouraged to help each other **understand** the problems and solutions, but each should write his/her own code. By simply copying someone else's code or solutions, you will not build an understanding of the work.
5. You are responsible for your own progress. Ensure that you understand the practical work. Check your work against the memorandum that will be posted on Wednesday afternoons on learn.sun.ac.za.
6. Use H:\CP143 as your Code::Blocks workspace folder for all projects. But it is highly suggested that you also use a **flash drive to backup** all your work.
7. Create a new project **for each assignment**. See *Creating a Code::Blocks Project* in Practical 0 for instructions on how to do this.
8. Include a comment block at the top of each source file according to the format given. It must include the correct filename and date, your name and student number, the copying declaration, and the title of the source file.
9. **Indent your code correctly.** Making your code readable is not beautification, it is a time and life saving habit. Adhere to the standards (refer to the documents on SUNLearn).
10. Comment your code sufficiently well. It is required for you and others to understand what you have done.

Question A

Goal: *Write a program from a pseudocode description.*

Getting started

1. Create a project named `Assignment4A`. Make sure that this is the active project (the project name must be in bold, otherwise right click it and select activate) in the workspace before compiling the program.

2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.

3. Read the complete question before you start programming.

Program description

1. The program is a very simple electrical circuit simulation. The circuit is a DC voltage source connected to two resistors in series. Your mathematical model is Ohm's law: $V = IR$ (voltage = current × resistance).

2. Tell the user that all values to be entered must be greater than zero. Allow the user to enter the value for the two resistors in Ohm (Ω), as well as the potential of the voltage source in voltage (V). Choose an appropriate variable type (i.e. int, float, char) to store these values.

3. If any of these three values are entered as a negative value or zero, repeat the process until all the inputs are positive and non-zero. Once all the input values pass the test, use it to calculate and display the current of the circuit in ampere (A).

4. Print out the value in Ampere units (A) if the current is more that 0.1 A, else print it out in mA-units (1000 mA = 1A).

5. The above description of the program allows no input to be zero or less than zero. Change your program to allow EITHER one of the resistors to be zero. Both may not be zero at the same time, however at any one time one of them may be.

6. Sample output:
   Sample 1: one of the resistors is 0.0 (answer in mA)
   ```
   One resistor may be zero; neither may be negative;
   voltage must be greater than zero.
   Value for Resistor 1 (units: Ohm): 0.0
   Value for Resistor 2 (units: Ohm): 1000.0
   Value for Voltage Source (units: V): 12.0

   For a series circuit the current is 12.0 (units: mA)
   ```
   Sample 2: one of the resistors is 0.0 (answer in A)

```
One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): 33.0
Value for Resistor 2 (units: Ohm): 0.0
Value for Voltage Source (units: V): 24.0

For a series circuit the current is 0.7273 (units: A)
```

Sample 3: test that all invalid input is caught by the program

```
One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): 0.0
Value for Resistor 2 (units: Ohm): 0.0
Value for Voltage Source (units: V): 12.0

One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): -1000.0
Value for Resistor 2 (units: Ohm): 3300.0
Value for Voltage Source (units: V): 12.0

One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): 1000.0
Value for Resistor 2 (units: Ohm): -3300.0
Value for Voltage Source (units: V): 12.0

One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): 1000.0
Value for Resistor 2 (units: Ohm): 3300.0
Value for Voltage Source (units: V): -12.0

One resistor may be zero; neither may be negative;
voltage must be greater than zero.
Value for Resistor 1 (units: Ohm): 1000.0
Value for Resistor 2 (units: Ohm): 3300.0
Value for Voltage Source (units: V): 12.0

For a series circuit the current is 2.7907 (units: mA)
```

7. **Ensure that your code is indented correctly** and that the { } braces are on the correct lines. Use the prescribed textbook as guideline.

8. Ensure that you copy the **Assignment4A** project folder to a flash drive as a backup.

Question B

Goal: *Write a few functions to show how variables are passed and how the calling stack works.*

Getting started

1. Create a project named `Assignment4B`. Make sure that this is the active project in the workspace before compiling the program.

2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.

Program description

1. Retype the following code exactly. Add the missing function prototypes and bodies as specified in the next points, but do NOT change the code for the `main()` function.

```
#include <stdio.h>
#include <stdlib.h>

//function prototypes
void add(int x, int y); //one of the prototypes was written for you
// ***put the remaining function prototypes here***

int main(void)
{
        add(2,2);
        subtract(10,1);
        addSubtract(3,7,8,2);
        add(24,100);
        return 0;
}

//write the functions HERE!
```

2. Now write the functions called `add()`, `subtract()` and `addSubtract()`.

   - Write the functions so that all three functions return void, meaning they return no values.
   - The function `add()` accepts **two integers** as parameters, so does the `subtract()` function. The `addSubtract()` function accepts **four integers** as parameters.
   - Each function should do the following:
     – Display a message indicating that it is starting
     – Do the appropriate calculations with the values that the function has received.

        * In `add()` and `subtract()`, the two numbers should be added and subtracted respectively, using **normal arithmetic**.
        * In `addSubtract()` the calculations should be done by **calling the functions** `add()` and `subtract()` to add the first two numbers it received, and subtract the last two it has received.
    - Display a message indicating that is is ending

3. Do NOT add any other code inside the `main()` function other than that given in point 1.

4. Your program's output should look exactly like the following listing:

```
The function called add() is starting.
It adds the two integers that are sent to it: 2 + 2 = 4
The function called add() is ending.

The function called subtract() is starting.
It subtracts the two integers that are sent to it: 10 - 1 = 9
The function called subtract() is ending.

The function called addSubtract() is starting.
It adds the first two and subtracts the last two of the four
integers that are sent to it.
It subcontracts its work ;)

The function called add() is starting.
It adds the two integers that are sent to it: 3 + 7 = 10
The function called add() is ending.

The function called subtract() is starting.
It subtracts the two integers that are sent to it: 8 - 2 = 6
The function called subtract() is ending.

The function called addSubtract() is ending.

The function called add() is starting.
It adds the two integers that are sent to it: 24 + 100 = 124
The function called add() is ending.
```

5. Ensure that your code is indented correctly and that the {} braces are on the correct lines. Use the prescribed textbook as guideline.

6. Ensure that you copy the **Assignment4B** project folder to a flash drive as a backup.

Question C

Goal: *Develop functions that do unit conversions.*

1. Create a project named `Assignment4C`. Make sure that this is the active project in the workspace before compiling the program.
2. Include the **standard comment block** above your main function. Also, comment your whole program appropriately.

Program description

Something to think about: For engineering, indicating and converting units are of paramount importance. One of the more famous cases where unit conversion caused problems is NASA's Mars Climate Orbiter. In 1999 this 125 million dollar craft was lost after its journey of 286 days (roughly 9 and a half months). It went slowly off course and in doing so missed the angle of approach necessary to orbit Mars. Why did this happen? NASA works in metric units, but one of their contractors who was responsible for the data that control the thruster motors delivered this data in English (Imperial) units to them. The navigation team of NASA expected the data to be metric and for this reason no conversion of this data was done. This resulted in miscalculations and so the thrusters fired incorrectly sending the orbiter off course. Just think, if they used the simple function you are about to develop, a $125 million investment would not have been lost. Detail matters.

1. To prevent any future cases (see the extract above) where unit conversion may result in project failure you are contracted to develop an application which performs a number of different unit conversions.
2. A menu should appear indicating all the conversion options. The user will enter the letter of his option followed by the value in the current unit. The program will then display the equivalent amount in another unit. The original menu will appear for the next conversion. The program will continue until the escape character is entered. Refer to the sample output below:

   Example:

```
***Conversion Menu***
a.      Wh To Joule
b.      Joule To Wh
c.      Inch To Cm
d.      Cm To Inch
e.      Hp To Kw
f.      kW To Hp
g.      Sec to Hr:Min:Sec
x.      Exit
Select a valid option:
a
Enter the value (Wh):
```

```
2.56
Your answer is (J):
9216.000000 J

***Conversion Menu***
a.      Wh To Joule
b.      Joule To Wh
c.      Inch To Cm
d.      Cm To Inch
e.      Hp To Kw
f.      kW To Hp
g.      Sec to Hr:Min:Sec
x.      Exit
Select a valid option:
d
Enter the value (cm):
5.87
Your answer is (inch):
2.311024 inch

***Conversion Menu***
a.      Wh To Joule
b.      Joule To Wh
c.      Inch To Cm
d.      Cm To Inch
e.      Hp To Kw
f.      kW To Hp
g.      Sec to Hr:Min:Sec
x.      Exit
Select a valid option:
g
Enter the value (seconds):
5894
Your answer is (hh:mm:ss):
1:38:14

***Conversion Menu***
a.      Wh To Joule
b.      Joule To Wh
c.      Inch To Cm
d.      Cm To Inch
e.      Hp To Kw
f.      kW To Hp
g.      Sec to Hr:Min:Sec
x.      Exit
Select a valid option:
x
```

```
Goodbye!
```

3. The program can perform the conversions between Wh (Watt-hour) and J (Joule), inc (inches) and cm (centimetres), Hp (horsepower) and kW (kiloWatt) and finally from a total time in seconds into a hh:mm:ss (hours:minutes:seconds) format.

4. Ensure that all functions have **double** as arguments and return value. For the functions related with time ensure that the arguments are of type **int** and returns an **int**.

5. Implement each of the conversion cases with a separate function. The required function names are listed in the table below.

Function name	**Functionality**	**Conversion**
`wattHrToJoule`	converts watts hour to joule	1 J = 1 W × 1 s
`jouleToWattHr`	converts joule to watts hour	
`inchToCm`	converts inch to centimetre	1 inch = 2.54 cm;
`cmToInch`	converts centimetre to inch	
`hpToKW`	converts horsepower to kiloWatt	1 hp = 0.746 kW;
`kWToHp`	converts kilowatt to horsepower	
`timeToSec`	converts total time in seconds into seconds	Hint:
`timeToMin`	converts total time in seconds into minutes	Use integer division and modulus
`timeToHr`	converts total time in seconds into hours	

Table 1: Implement these nine functions.

6. Ensure that your code is indented correctly and that the { } braces are on the correct lines. Use the prescribed textbook as guideline.

7. Ensure that you copy the **Assignment4C** project folder to a flash drive as a backup.

Something to think about: Unix time is a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, not counting leap seconds. It is used in the majority of information systems to communicate the date and time in the form of a single large number.