Program Development Hints
(for CSC110 Computing A)

Alex Yakovlev
Computing Science Department
University of Newcastle upon Tyne

Electricity Bill Calculation

Problem Statement

An electricity bill is calculated by multiplying the **number of rooms** in a house by 15 and then charging 5 pence per unit for the resulting number of units. This is a fixed cost. If additional units are consumed on top of this, they are charged at 2 pence each.

Determining the number of units that have been consumed involves calculating the difference between the **old** and **new** meter readings.

Design and implement a program which will read the whole (that is, integer) numbers for the number of rooms, old and new meter readings, and which will output the **cost of the electricity consumed**.

Make your program so that it either produces the **total amount due in pence** or results are converted **to pounds and pence** if the **user so wishes**. (For example, *516 pence* would be output as *5 pounds and 16 pence*).

First thing to do: Read the Specification

Indeed, the best way to start with is to read the problem specification as carefully as you can. During this reading, try to identify the key objects in this specification, that is the objects that are supplied as input data to your program, and the objects that are supposed to be produced as the output results by your program.

Use a highlighter or some other means to bring these objects up to your attention. Note that in the above specification the input data and output results have been put in **bold face**.

The highlighting will help you further to identify the variables (and their types) that will be used in your program.

Second thing to do: Look at an Example

The next thing to do is still better not to rush to programming but rather make sure that you understand the problem correctly. For this, imitate the program's action on a simple example. What would your program do in a specific case, that is with concrete values for the number of rooms and the old and new meter readings?

It is like writing a script for a theatre play. Imagine that you have to write both parts, one for your future program and the other for its user.

For example:

```
Program: What is the number of rooms in your house?

User: 4

Program: What is the OLD Meter reading?

User: 356

Program: What is the NEW Meter reading?
```

```
User: 620

(Program now gets into calculations:

  (1) The number of units for fixed charge is: 15 * 4 = 60 units.
  (2) The fixed charge is: 60 * 5 = 300 pence.
  (3) The actual number of units is: 620 - 356 = 264 units.
  (4) Checking if the actual number of units (264) is greater than
      the fixed rate number (60). The answer is yes, it is greater.
  (5) The number of units above the fixed rate is: 264 - 60 = 204 units.
  (6) The cost of additions units is: 204 * 2 = 408 pence.
  (7) Total cost is: 300 + 408 = 708 pence.

Program is now ready to produce the result.)

Program: Do you want price in pounds and pence (answer Y or N) ?

User: Y

(Program now converts the result in pence into the one in pounds
and pence. It first divides 508 by 100 in an "integer" way - this gives 5.
Then it finds the integer remainder of division of 508 by 100, which gives 8.)

Program: 5 pounds and  8 pence.
         The end of program.
```

Third thing to do: Design the Program

Use, for example, the Design Structure Diagram notation to represent your design (see Figure 1).

Note that at the design stage you may decide that when calculating the total cost of the bill the following formula may be used:

```
Total cost = Number of units for fixed charge * 5 pence +
             Number of units above the fixed rate * 2 pence
```

Forth thing to do: Summarise Objects and Types

Now, before doing actual coding in C++, settle on the data types for the objects used in your program. The best way is to build a table of the objects:

Object	**Identifier**	**Type**	**Inp./Out./Aux.**
Number of rooms	`nosRooms`	`int`	Input
Old meter reading	`oldMeterReading`	`int`	Input
New meter reading	`newMeterReading`	`int`	Input
Number of units for fixed rate	`fixedUnits`	`int`	Auxiliary
Cost for fixed rate	`fixedCost`	`int`	Auxiliary
Actual number of units	`actualUnits`	`int`	Auxiliary
Number of units above fixed rate	`aboveUnits`	`int`	Auxiliary
Cost above fixed rate	`aboveCost`	`int`	Auxiliary
Total cost in pence	`totalCostPence`	`int`	Ouput
Yes or No Price in Pounds	`replyYorN`	`char`	Input
Total cost in pounds	`totalCostPound`	`int`	Output
Remainder in pence	`remainderPence`	`int`	Output

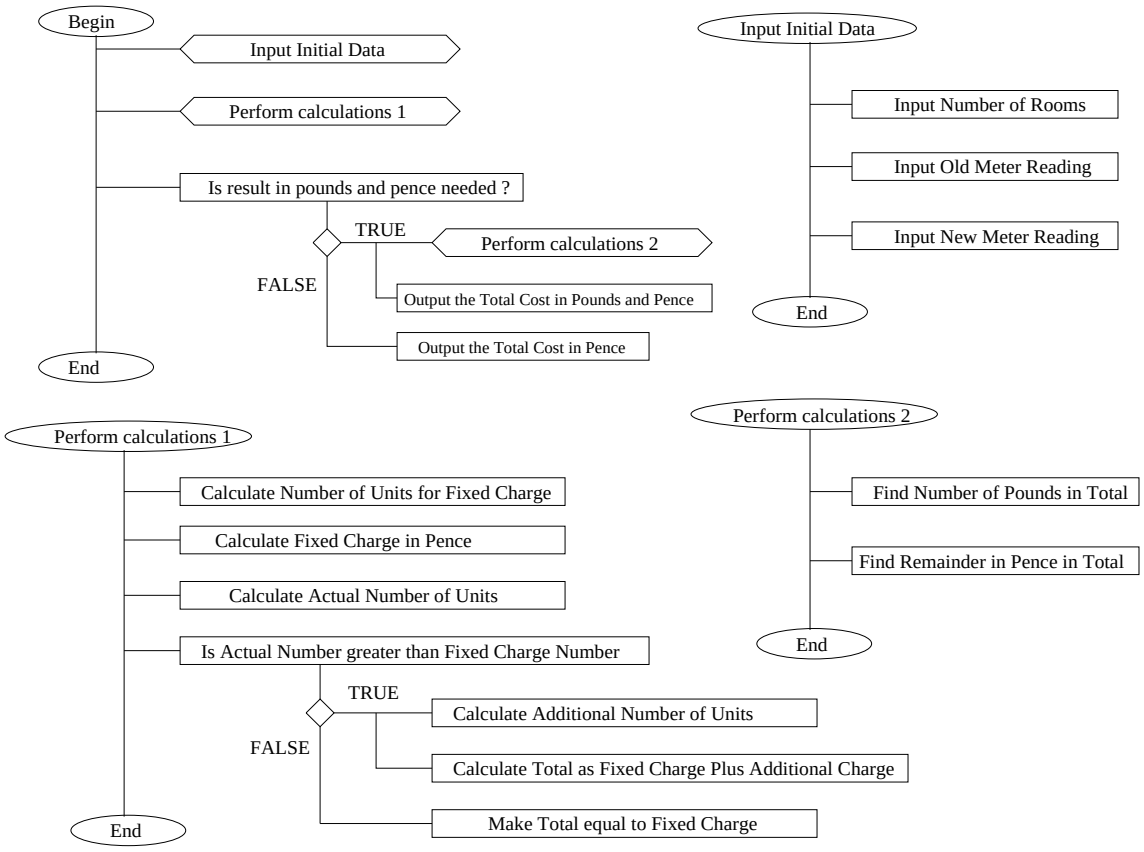

Figure 1: Design Structure Diagram for Electricity Bill

Fifth Thing to do: Implement the Program in C++

When writing you code, first include your statements from the Design Structure Diagram, as comments. They will guide you through the process of coding in C++

```
// This program does so and so ...
// Author: ....              Date: ...
// ...
#include<iostream.h>

// declare global constants
const int UnitsPerRoom  = 15;
const int FixedUnitCost =  5;
const int AddUnitCost   =  2;

void main()
{

// declare input variables
int nosRooms;
int oldMeterReading;
int newMeterReading;
char replyYorN;
int totalCostPence;
int totalCostPound;
int remainderPence;
```

```
    // Input Initial Data
    cout << "What is the number of rooms in your house? ";
    cin >> nosRooms; cout << endl;
    cout << "What is the OLD Meter reading? ";
    cin >> oldMeterReading;  cout << endl;
    cout << "What is the NEW Meter reading? ";
    cin >> newMeterReading;  cout << endl;

    // Perform Calculations 1
    // Calculate Number of Units for Fixed Charge
    int fixedUnits = UnitsPerRoom * nosRooms;
    // Calculate Fixed Charge in Pence
    int fixedCost = fixedUnits * FixedUnitCost;
    // Calculate Actual Number of Units
    int actualUnits =  newMeterReading - oldMeterReading;
    // Is Actual Number than Fixed Charge Number?
    if (actualUnits > fixedUnits)
    {   // TRUE Case
        // Calculate Additional Number of Units
        int aboveUnits = actualUnits -  fixedUnits;
        // Calculate Total as Fixed Charge plus Additional Charge
        totalCostPence = fixedCost + aboveUnits * AddUnitCost;
    }
    else
    {   // FALSE Case
        // Make Total equal to Fixed Charge
        totalCostPence = fixedCost;
    }
    // Is result in pounds and pence needed ?
    cout << "Do you want cost in pounds and pence (answer Y or N)? ";
    cin >> replyYorN; cout << endl;
    if (replyYorN == 'Y')
    {   // Case Yes
        // Perform Calculations 2
        // Find Number of Pounds in Total
        totalCostPound = totalCostPence / 100;
        // Find REmainder in Pence in Total
        remainderPence = totalCostPence % 100;
        // Output Total in Pounds and Pence
        cout << "Total Due: " << totalCostPound << " pounds and "
             << remainderPence << " pence." << endl;
     }
     else if (replyYorN == 'N')
     {   // Case No
         // Output the Total Cost in pence
         cout << "Total Due: " << totalCostPence << " pence." << endl;
     }
     else
     {   // Case neither Yes nor No - report Erroneous input
         cout << "Error in reply!" << endl;
     }
     cout << "the end of program" << endl;
}
```

As you can see in the above program code, some additions in you program design can be made in the process of implementing your design in C++. This happened, for example, when we added an option when the user types neither Y nor N to the program's question. This can be "fed back" to your original design

structure charts, to reflect the change introduced at the coding stage.

Sixth Thing to do: Debugging, Testing and Enhancing the Program

Your program must be typed into a file, e.g. **bill.c**, compiled within a project environment. All the compilation errors must be rectified. Run your program several times with different initial data and user responses. Get rid of all run-time errors. Make notes of the way your program behaves for different sets of initial data and user's actions. This will give you ideas about possible modifications in your design.

This is what my implementation (on UNIX) has given (the result of copying and pasting the screen) for the previously considered example:

```
% CC el_bill.c
CC +p el_bill.c:
cc  -L/usr/local/CC/3.0.1/lib   el_bill.c -lC
% a.out
What is the number of rooms in your house? 4

What is the OLD Meter reading? 356

What is the NEW Meter reading? 620

Do you want cost in pounds and pence (answer Y or N)? Y

Total Due: 7 pounds and 8 pence.
the end of program
%
```

This is the right time to draw conclusions on how you may improve your program. E.g., make it more robust to the user's actions. For example, introduce the following option into the design - when the old and new Meter readings are inputted, check that the new reading is not less than the old reading. You may also make the user's life a bit more comfortable by allowing them to reply with small 'n' and 'y' (in addition to the already provided capital letter reply). This can be done by including an appropriate boolean connective (e.g., `((replyYorN == 'Y') || (replyYorN == 'y'))`) into the conditional expressions testing the user's reply.

Another enhancement can make your program deal with the cases when the Meter has a limited capacity (after all an electronic Meter is a digital Counter which has a finite bit word length), that is it operates until some MAX value, say 9999 and then returns back to 0000. In other words, it operates modulo 10000. Modify your design accordingly.

Your program may also be modified to allow it to work in a cyclic way, That is, once having done a calculation, it is ready for the next calculation and so it returns back to the initial state, in which it asks the user questions about the number of rooms etc. This modification allows your program to work with many users. Introducing it into your design will require using a repetition construct. A corresponding repetition statement (e.g., the **while**{ **...** } **loop**) must be added in your program code.

There is still a lot of potential to improve the robustness of user's input. That is, your program may "fall out" of the execution if the user types a number instead of a character or vice versa. To deal with this more robustly, you need to check if the user's input is of correct type. If the type is wrong, you may return the user back to the same question. Look for appropriate fragments of code in the set of the Course Lecture Notes (Sections 10-13).