Session 3: The Input Statement

3.1 Introduction

Before we introduce the Input Statement, it is necessary to go through the procedure of how we can solve programming problems. As such will first look to at the steps to follow when presented with a programming problem.

3.2 Programming Problem solving

A computer program also is a solution, but one that is implemented with a computer. To solve problems that require a computer solution, computer programmers should follow the following steps:

1. Understand the problem
2. Devise an algorithm
3. Implement the algorithm
4. Test and Debug
5. Maintain the program

3.2.1 Step 1 - Understand the Problem

You cannot solve a problem unless you *understand* it, and you cannot understand a problem unless you *analyze* it – in other words, unless you identify first what is required of you from the problem and also identify its important components. We will discuss two ways of how to understand the problem:

1. Rewrite the problem
2. Identify the important components of the problem

1. Rewrite the problem

Given a computer problem to solve, the first approach to solving that problem is make sure you understand the problem in terms of its needs. To do so, we to rewrite the problem such that we have a list of what is required of us to solve the problem.

We look at the problem statement, sentence by sentence, list the sentences. After which then sentence by sentence look at verbs in each sentence which guides on what the program will do; be sure to list them one by one. This way then we will know what is expected of us in terms solving the problem. To illustrate this step, we will use the following problem statement shown in Figure 3-1 below:

```
Mahir Athman wants a program that calculates and displays the amount he should tip a waiter
at a restaurant. The program should subtract any liquor charge from the total bill and then
calculate the tip (using a percentage) on the remainder.
```

Figure 3-1: Problem Specification for Mahir Athman

As shown in Figure 3-2, there are two sentences in this problem specification:

```
1. Mahir Athman wants a program that calculates and displays the amount he should tip a
   waiter at a restaurant.
2. The program should subtract any liquor charge from the total bill and then calculate the
   tip (using a percentage) on the remainder.
```

Figure 3-2: Extraction of Sentences from Mahir Athman problem

From these two sentences, we can now attempt to come up with the initial requirements for our problem. Remember this step of specifying help us to understand the problem by helping us know what is required of us to do say that we have solved the problem. The first attempt of the list of requirements are:

1. Calculate and display the amount of tip to be paid.
2. Subtract any liquor charge from the total bill and then calculate the tip.

You will realize that both requirements are compound statements in nature: like requirement 1 above talks of *calculate and display…* these are two requirements combined together. You cannot calculate and display at the same time, for simplicity sake separate the two.

1. Calculate the amount of tip to be paid.
2. Display the amount of tip to be paid.

Requirement 2 also is a compound statement in nature it talks of *subtract … and then calculate the tip* can be broken down as follows:

1. Subtract liquor charge from total bill to get the amount to be charged tip (call this amount as 'tip amount bill').
2. Calculate the tip as a percentage of tip amount bill.

How if we combine the two sets of requirements, we will have the following requirements as shown in Figure 3-3 below:

```
1. Calculate the amount of tip to be paid.
2. Display the amount of tip to be paid.
3. Subtract liquor charge from total bill to get the amount to be charged tip (call this
   amount as 'tip amount bill').
4. Calculate the tip amount as a percentage of tip amount bill.
```

Figure 3-3: Initial attempt on Requirements Specification for Mahir Athman problem

You realize that as we have restated the problem; it now enabled us to understand the problem much more easily, because it now easy to follow form the requirements specification given in Figure 3-3 above. In so doing, a solution can be easily found; however, without understanding what you want to solve, it will be a nightmare to solve to a problem you do not understand. Understanding the problem in your own perspective is key to problem – solving especially in programming.

2. Components of the Problem

The two important components of any problem are the problem's output and its input. The output is the goal of solving the problem, and the input is item or items needed to achieve the goal.

When analyzing a problem, you always search first for the output data items and then for the input data items. From our complete requirements list (see Figure 3 requirement 2), we easily identify the output by searching for an answer to the following question: *What does the user want to see displayed on the screen, printed on paper, or stored in a fil*e? The answer to this question typically is stated as nouns and adjectives in the requirements specification. For example, the requirements specification indicates that the user wants to see the amount of the waiter's tip displayed on the screen; therefore output data item is the tip. In this context, the word tip is a noun.

After determining the output, you then determine the input. From our requirements list in Figure 3, it is not easy to identify the input data items; reason, it assumed that the user has the total bill in his/her hand. From that assumption, we can now easily search for an answer to the following question: *What information will the computer need to know to display, print, or store the output items*? As with output, the input typically is stated as nouns and adjectives in the requirements specification. When determining the input, it helps to think about that information that you would need to solve the problem manually, because the computer will also need to know the same information. In this case, to determine the tip, both you and the computer need to know the total bill, liquor charge, and the tip percentage; these items, therefore, are the input. The liquor charge is specified in the total bill, whereas the tip percentage must specified in restaurant policy manuals. In this context, *total*, *liquor*, and *tip* are adjectives, while *bill*, *charge*, and *percentage* are nouns.

From the requirements stated in Figure 3- 3 above, requirement 3 and 4 specify the processing of the input data items to achieve the problems output. A **processing data item** represents an intermediate value (it is neither input nor output) used in the processing input data items into output. The processing typically involves performing one or more calculations using the input items. In this case, the tip is calculated by subtracting the liquor charge from the total bill and then multiplying the remainder by the tip percentage. For avoidance of confusion the first value that we obtain as the difference we will call it tip amount bill and second value which is then product of tip amount bill and tip percentage we will call tip. This completes the analysis step for Mahir Athman problem. Figure 3-4, below depicts that data items for Input, Processing and Output of the Mahir Athman problem.

```
Input items          Processing items          Output items
total bill           tip amount bill           tip
liquor charge        tip
tip percentage
```

Figure 3-4: Input, Processing and Output data items for the Mahir Athman problem

3.2.2 Step 2 - Devise an Algorithm

We define the term algorithm as an ordered set of unambiguous steps that produces a result and terminates in a finite time. In simple terms an algorithm is a well-defined, ordered set of instructions. After going through the all instructions in an algorithm we expect a result from the algorithm. It is also important to note that an algorithm must terminate.

We have a starting point to write the algorithm for the Mahir Athman problem. As we can see in Figure 3-3, in general there are two tasks defined in requirements 1 and 2 respectively; however, if we follow that list of requirements as an algorithm, it will not suffice – reason, we cannot display the value of the tip in requirement 2 while we have not computed the value in requirement 4. Therefore, there is need to rearrange the requirements, to make all the requirements logically flow as they are executed; this will result into our initial algorithm to solve the Mahir Athman problem as shown in Figure 3-5 below:

```
1. Calculate the amount of tip to be paid.
2. Subtract liquor charge from total bill to get the amount to be charged tip (call this
   amount as 'tip amount bill').
3. Calculate the tip amount as a percentage of tip amount bill.
4. Display the amount of tip to be paid.
```

Figure 3-5: Initial Algorithm for Mahir Athman problem

Remember, we had an assumption for the input data items; again from Figure 3-3, input to this program is not stated as a step in the algorithm; thus, we also need to add at the beginning an input step for the input data items. Again, from Figure 3-5, it is evident the way to achieve step 1 is actually defined in step 2 and 3; hence no need to have step 1. This modification realizes the algorithm in Figure 3-6 as follows:

```
1. Enter total bill, liquor charge and tip percentage.
2. Subtract liquor charge from total bill to get the amount to be charged tip (call this
   amount as 'tip amount bill').
3. Calculate the tip amount as a percentage of tip amount bill.
4. Display the amount of tip to be paid.
```

Figure 3-6: The Algorithm for Mahir Athman problem

We are ready to get to Step 3 of our problem solving procedure stipulated above.

3.2.3 Step 3 Implement the Algorithm

To implement the algorithm that is specified in Step 2 of the problem solving procedure, we need to have some basic knowledge of a programming language; as of now, we have only learnt about the structure of a C program and formally introduced the output statement. However, for us to accomplish the implementation of the algorithm specified in Figure 6, we need to learn more statements in C; which for the purposes of this session will be very simple, we will learn about the declaration statement, assignment statement and input statement in the subsections below.

1. Identifiers

One feature present in all programming languages is the identifier. An identifier is a user defined word; it is used to name data items and other objects in a program. Each piece of data in a computer is stored at a unique address. If there were no identifiers to symbolically represent data locations, you would have to know and use data addresses to manipulate them. Instead, you simply give data identifier names and let the compiler keep track of where they are physically located.

Rules to form identifiers:

1. The first character must be an alphabet.
2. The other characters may be alphanumeric.
3. The only special symbol allowed is the underscore (_).
4. A user defined identifier cannot be a reserved word or keyword.

2. Data Types

A data type defines a set of values and a set operations that can be applied to those values. The set of values for each type is known as the domain for the type.

C has six primitive data types namely `int`, `long`, `float`, `double`, `char` and `bool`.

Variables

Variables are names for memory locations. Variables are used to hold data that are required by the program for its operation. Each memory location (or word) in a computer has an address. Although, addresses are used by the computer internally, it is very inconvenient to for the programmer to use addresses because the programmer does not know where the program and data are stored in memory.

Names, as a substitute for addresses, free the programmer to think at the level at which the program is executed. A variable name is an alias to memory address.

C requires the programmer to define a type for the variable. The reason is that the type defines the range of the data that can be stored in the variable and defines the number of bytes that can be allocated for that variable.

3. Variable Declaration and Definition

C requires that a variable be declared and defined before being used. A declaration statement makes known a variable to the compiler. Declaration statement is used to create the variable and to allocate a memory location for it.

When you create a variable, the declaration gives it a symbolic name, and the definition reserves memory for it. Once defined, variables are used to hold the data that are required by the program for its operations.

To define a variable C uses its declaration statement which has the following syntax:

```
Single variable declaration:
data_type variblename;

Multiple variable declaration:
data_type var1 [, var2, ...,varn];
```

Please note that any part of syntax, that is put in square brackets is said to be optional.

Example

```
float price;
```

4. Assignment statement

An assignment statement gives a value to a variable. The value can be a constant or an expression. An expression is a sub statement that logically combines data items and operators. The general syntax for an assignment statement is as follows:

General syntax for a declaration statement:

```
variable = value | expression;
```

Please remember all statements in a C program will end with a semicolon as shown above. The syntax uses the pipe '|' symbol to mean 'or' in syntax.

For this session will introduce the arithmetic operators used in C:

Operator	Description
*	Multiplication
/	Division
+	Addition
-	Subtraction
%	Modulus

Examples of an assignment statement

```
price = 90.75;
price = 0.8 * total;
```

5. Constants

Constants are data values that cannot be changed during the execution of a program. For example, a programmer may need to use the value of `pi (3.14)` several times. It is very convenient to define a constant at the top of the program and use it. On some occasions, the value can be changed, but not each time the program is run.

Constants are fall into three classes namely; literal constant, named constant, or symbolic constant.

Literal Constant

A literal constant is an unnamed value that is used in a program as is. For example, the following shows how a literal constant is used in an instruction in C. The value 2 is a literal constant. The variables are length, width, and circumference.

```
circumference = 2 * length * width;
```

Named Constant

A named constant is a value that you store in memory; you do not want the program to change it. The following shows how the C language creates a named constant:

```
const float pi = 3.14;
```

Symbolic Constant

A symbolic constant is a value that has only a symbolic name. The compiler can replace the name with the value. In most programming languages, the value is not stored in memory. The symbolic constant is defined at the beginning of the program to be conspicuous. It is used for a constant that changes its value occasionally. The following shows how the C language defines a symbolic constant:

```
#define taxRate 0.082
```

6. Input Statement

An input statement allow a user to put data into a specified variable. In C, the function used to define an input is called the scanf() function and it has the following general syntax:

Syntax:

```
scanf("%format_specifier", &variable);
```

There are two symbols used in the scanf() function

i. % used to define the format specifier for corresponding data type of the input variable.
ii. & - called the address indirection operator, used to get the address of the input variable.

The format specifier for each corresponding data type is shown below:

Format Specifier	Data Type
%d	int
%ld	long
%f	float
%lf	double
%c	char

An example of an input statement

```
int a;
float z;
...
scanf("%d", &a);
scanf("%f", &z);
```

7. Implementing the Algorithm

The algorithm as stated in the Figure 5 can now be implemented step by step. We are going to implement that algorithm with the statements that we have learnt so far.

1. **Write the basic C program structure:**
   For a complete C program, we use the program structure we discussed in Session 1 as follows:

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

```
int main()
{

      return 0;
}
```

2. **Declare all variables used in algorithm**

Before we implement the steps that we have in the algorithm we need to be sure of all variables used. Good luck, we already identified them in Figure 4 above as follows:

```
Input items
total bill
liquor charge
tip percentage

Processing items
tip amount bill
tip

Output items
tip
```

You realize the data items as given above some have more than one word, to conform to C's identifier naming rules; we devise a convection to use – any data item with more than one word, we name with all the words and first word will be in lower case, any other word will have its first letter capitalized, as follows:

```
Input items
totalBill
liquorCharge
tipPercentage

Processing items
tipAmountBill
tip

Output items
tip
```

The next thing we will do is get the data type of each data item. Now, looking at what we have, the tip is guide line for the other variable names because that is the computed value – tip is computed as percentage of tip amount bill, the percentage is a real number and therefore that computation will always realize a real number. Small value real numbers commonly used the data type float – and it is for this reason that we will use the float for all the variables.

So the declaration statements for all variables are as follows:

```
float totalBill, liquorCharge, tipPercentage;
float tipAmountBill, tip;
```

The complete program after this step should now look as follows:

```
#include <stdio.h>
int main()
{
	//variable declaration
	float totalBill, liqourCharge, tipPercentage;
	float tipAmountBill, tip;

	return 0;
}
```

From here on, all new code inserted to the existing program will be bold in colour to distinguish from the existing code. You realize that we have included another line of code which is new to you – we call new line of code which starts the two forward ("//") as a comment. A comment is a non-executable statement in C used to document a program. Comments are used in program help explain statements written in the program. It is good to note, we first write programs for ourselves as programmers and therefore there is need to have a program that can be easily read hence easily understood. There two types of comments in C, namely; single line comment and multiline comment.

General syntax for a comment:

a. Single line comment

```
//comment is written as one single line of text
```

b. Multiline comment

```
/*
   This is a multiline comment
   you can have as many lines
   of text as possible
*/
```

3. Implement the first step in the algorithm

From Figure 5 above, the first step in the algorithm require the program enter the input data items. Therefore, we will use the input statement to input those data items, as follows:

```
//input the totalCharge, liqourCharge and tipPercentage
scanf("%f", &totalCharge);
scanf("%f", &liqourCharge);
scanf("%f", &tipPercentage);
```

The complete program after this step should now look as follows:

```
#include <stdio.h>
int main()
{
      //variable declaration
      float totalBill, liqourCharge, tipPercentage;
      float tipAmountBill, tip;

      //input the totalCharge, liqourCharge and tipPercentage
      scanf("%f", &totalCharge);
      scanf("%f", &liqourCharge);
      scanf("%f", &tipPercentage);

      return 0;
}
```

4. **Implement the second step in the algorithm**
   The step involves a computation of the difference between totalBill and liqourCharge, which can be written as follows

```
         //compute the tipAmountBill
         tipAmountBill = totalBill – liqourCharge;
```

   Below is the resultant program after adding the computation statement:

```
#include <stdio.h>
int main()
{
      //variable declaration
      float totalBill, liqourCharge, tipPercentage;
      float tipAmountBill, tip;

      //input the totalCharge, liqourCharge and tipPercentage
      scanf("%f", &totalCharge);
      scanf("%f", &liqourCharge);
      scanf("%f", &tipPercentage);

      //compute the tipAmountBill
      tipAmountBill = totalBill – liqourCharge;


      return 0;
}
```

5. **Implement the third step in the algorithm**
   This step also involves computation and therefore similar to the second step, hence it is also an assignment:

```
         //compute the tip
         tip = tipAmountBill * (tipPercentage/100);
```

   Hence, the complete program after adding this statement is as follows:

```
#include <stdio.h>
int main()
{
      //variable declaration
```

```
    float totalBill, liqourCharge, tipPercentage;
    float tipAmountBill, tip;

    //input the totalCharge, liqourCharge and tipPercentage
    scanf("%f", &totalCharge);
    scanf("%f", &liqourCharge);
    scanf("%f", &tipPercentage);

    //compute the tipAmountBill
    tipAmountBill = totalBill – liqourCharge;

    //compute the tip
    tip = tipAmountBill * (tipPercentage/100);


    return 0;
}
```

6. **Implement the fourth and final step in the algorithm**
   The final step in the algorithm talks about displaying the value of the output data item called tip. The output statement to confirm this is as follows:

```
    //display the tip amount
    printf("The tip amount is: %f.\n", tip);
```

   After adding this statement, the program should be as follows:

```
#include <stdio.h>
int main()
{
    //variable declaration
    float totalBill, liqourCharge, tipPercentage;
    float tipAmountBill, tip;

    //input the totalCharge, liqourCharge and tipPercentage
    scanf("%f", &totalCharge);
    scanf("%f", &liqourCharge);
    scanf("%f", &tipPercentage);

    //compute the tipAmountBill
    tipAmountBill = totalBill – liqourCharge;

    //compute the tip
    tip = tipAmountBill * (tipPercentage/100);

    //display the tip amount
    printf("The tip amount is: %f.\n", tip);

    return 0;
}
```

It is important to note, we as were implementing the algorithm, we did step by step; this enable us to stepwise complete the program and also help us to think about the program in bits. Again, the source code given after every step specified above, should be compiled to be free of syntax errors, this will help us as we add more statements to know where our errors are coming from. In terms of the solution this approach follows the Divide and conquer concept, where one divides a problem into solvable bits, solve each bit one after the other to achieve the overall solution. Dealing with the whole program at once, may not be advisable and is always going to be an error target.

You will realize that compiling some of the steps specifically Step 2 – declaring variables above will give you warnings. Do not worry about warnings, most warnings are giving information; they are not errors. So you can continue with the next step without fear of getting errors.

Exercises

a.) Write a program that accepts a product price (this is price is exclusive of VAT), computes the corresponding Value Added Tax (VAT) of the product, hence it should also compute the selling product price after tax and displays the product price and selling price after tax, and its corresponding VAT. Assume a 16% VAT rate.

b.) Write a program that accepts the number of hours a causal worker in a company has worked in a month. Compute the gross pay by multiplying the number of hours by an hourly rate of Kshs 450.50. Calculate the Withholding tax, which is 20% of the gross pay. Then calculate the net pay as difference of withholding tax from gross pay. The program should display the number of hours worked, rate of pay per hour, gross pay, withholding tax and net pay.